Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

Alejandro Garnung Menéndez

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

DESARROLLO DE UN SISTEMA PARA LA VALIDACIÓN DE LA POSE DE PIEZAS INDUSTRIALES EN CINTAS TRANSPORTADORAS CON ENFOQUE EN SIMULACIÓN Y EXPERIMENTACIÓN AVANZADA

AUTOR: D. Alejandro Garnung Menéndez

TUTOR ACADÉMICO: D. Rafael Corsino González de los Reyes

TUTORES DE EMPRESA: D. Daniel Pérez García, D. Jorge Marina Juárez

FECHA: febrero 2024

ÍNDICE GENERAL

MEMORIA

RESUMEN

Este proyecto se enfoca en desarrollar un sistema que utiliza imágenes de referencia y actuales de una cámara 2D para monitorizar y analizar piezas en una cinta transportadora en un entorno industrial. Se implementan interfaces gráficas, de comunicación y software para la integración de datos de los sensores, ejecutando algoritmos de preprocesamiento, detección de movimiento, extracción y comparación de características, así como reconocimiento de objetos mediante técnicas de visión artificial clásicas. Se propone una solución práctica desarrollada bajo el framework Qt respaldada por un simulador robusto que facilita la experimentación y el perfeccionamiento de algoritmos en entornos industriales simulados en 3D. El simulador se presenta como una herramienta versátil que puede apoyar la investigación de diversas técnicas de visión artificial y aprendizaje automático; desarrollado empleando el estándar OpenGL y el lenguaje C++ con enfoque orientado a objetos, constituye una herramienta potente y versátil para la emulación 3D de espacios industriales.

El algoritmo implementado, empleando la librería OpenCV, actúa como un clasificador analítico para determinar la validez de la pose de una pieza en una línea de producción. Se utilizan dos fuentes de datos: una colección estática de imágenes (piezas de referencia) y vectores de características precalculados y, por otro lado, frames provenientes de una cámara 2D (piezas a validar). El sistema emplea entidades algorítmicas cuasi-inteligentes para procesar imágenes y otras señales, realizar análisis numérico, reconocimiento de características y otras labores, generando una respuesta final basada en estimaciones cuantitativas, votos, ponderaciones y umbrales empíricos. La combinación de múltiples algoritmos demuestra efectividad ante variaciones y deformaciones complejas.

Se destaca una fase experimental tanto en simulación como en la realidad en la que se configuran diversos entornos y condiciones para medir la capacidad del sistema en clasificar piezas válidas y no válidas, así como su precisión en escenarios adversos. Finalmente, tras haber puesto a prueba el sistema mediante análisis estadísticos de repetitividad para probar su robustez ante diversas casuísticas, se analizan los resultados y se concluye que el sistema en conjunto y los algoritmos implementados demuestran ser efectivos, culminando con una solución integral que aborda con éxito el desafío de monitorizar y analizar la pose de piezas en una línea de producción.

Palabras clave: visión artificial, procesamiento de imágenes, detección de objetos, estimación de pose 2D, puntos de interés, registro de imágenes.

ABSTRACT

This project focuses on developing a system that uses reference and current images from a 2D camera to monitor and analyze parts on a conveyor belt in an industrial environment. Graphical, communication and software interfaces are implemented for sensor data integration, executing algorithms for preprocessing, motion detection, feature extraction and comparison, as well as object recognition using classical computer vision techniques. A practical solution developed under the Qt framework is proposed, supported by a robust simulator that facilitates experimentation and algorithm refinement in 3D simulated industrial environments. The simulator is presented as a versatile tool that can support the investigation of various machine vision and machine learning techniques; developed using the OpenGL standard and the C++ language with an object-oriented approach, it constitutes a powerful and versatile tool for 3D emulation of industrial spaces.

The algorithm implemented, using the OpenCV library, acts as an analytical classifier to determine the validity of the pose of a part in a production line. Two data sources are used: a static collection of images (reference parts) and precomputed feature vectors and, on the other hand, frames coming from a 2D camera (parts to be validated). The system employs quasi-intelligent algorithmic entities to process images and other signals, perform numerical analysis, feature recognition and other work, generating a final response based on quantitative estimates, votes, weights and empirical thresholds. The combination of multiple algorithms demonstrates effectiveness in the face of complex variations and deformations.

An experimental phase is highlighted, both in simulation and reality, in which different environments and conditions are configured to measure the system's ability to classify valid and invalid parts, as well as its accuracy in adverse scenarios. Finally, after having tested the system through statistical repeatability analysis to prove its robustness in different cases, the results are analyzed and it is concluded that the system as a whole and the implemented algorithms prove to be effective, culminating with a comprehensive solution that successfully addresses the challenge of monitoring and analyzing the pose of parts in a production line.

Keywords: machine vision, image processing, object detection, 2D pose estimation, points of interest, image registration.

ÍNDICE DE LA MEMORIA

GLOSARIO DE SIGLAS

ACC	*Exactitud/Accuracy*
AdaBoost	*Adaptive Boosting*
A-KAZE	*Accelerated KAZE*
ALU	*Arithmetic Logic Unit*
API	*Application Programming Interface*
BRISK	*Binary Robust Invariant Scalable Keypoints*
CAD	*Computer-Aided Design*
CCTV	*Closed-Circuit Television*
CLAHE	*Contrast Limited Adaptive Histogram Equalization*
CPU	*Central Processing Unit*
CSS	*Cascading Style Sheets*
CSV	*Comma-Separated Values*
CUDA	*Compute Unified Device Architecture*
CV	*Coeficiente de Variación*
DFT	*Discrete Fourier Transform*
DOG	*Difference of Gaussians*
DRN	*Dilated Residual Networks*
EBO	*Element Buffer Object*
ECC	*Enhanced Cross-Correlation*
FAST	*Features from Accelerated Segment Test*
FBO	*Framebuffer Object*
FED	*Fast Explicit Diffusion*
FFT	*Fast Fourier Transform*
FLANN	*Fast Library for Approximate Nearest Neighbors*
FN	*False Negative*
FNLM	*Fast Non-Local Means*
FOV	*Field of View*
FP	*False Positive*

FPGA	*Field-Programmable Gate Array*
FPS	*Frames Per Second*
FREAK	*Fast Retina Keypoint*
GBM	*Gradient Boosting Machine*
GFTT	*Good Features to Track*
GLAD	*GL Extension Wrangler*
GLEW	*OpenGL Extension Wrangler Library*
GLM	*OpenGL Mathematics*
GLOH	*Gradient Location and Orientation Histogram*
GLSL	*OpenGL Shading Language*
GLU	*OpenGL Utility Library*
GLUT	*OpenGL Utility Toolkit*
GNU	*GNU's Not Unix*
GPGPU	*General-Purpose computing on Graphics Processing Units*
GPU	*Graphics Processing Unit*
GUI	*Graphical User Interface*
HOG	*Histogram of Oriented Gradients*
IP	*Internet Protocol*
IQR	*Interquartile Range*
K-NN	*k-Nearest Neighbors*
LIFT	*Local Invariant Feature Transform*
LMEDS	*Least Median of Squares*
LOG	*Laplacian of Gaussian*
LTS	*Long Term Support*
MCC	*Matthews Correlation Coefficient*
ML	*Machine Learning*
MLDB	*Modified Local Difference Binary*
MLP	*Multilayer Perceptron*
MOC	*Meta-Object Compiler*
MOG/GMM	*Mixture of Gaussians/Gaussian Mixture Model*
MSE/ECM	*Mean Squared Error/Error Cuadrático Medio*

NPV	*Negative Predictive Value*
ORB	*Oriented FAST and Rotated BRIEF*
PCA	*Principal Component Analysis*
PNG	*Portable Network Graphics*
POO	*Object-Oriented Programming*
PPV	*Positive Predictive Value*
PROSAC	*PROgressive Sample Consensus*
PSNR	*Peak Signal-to-Noise Ratio*
QSS	*Qt Style Sheets*
R-CNN	*Region-based Convolutional Neural Network*
RAM	*Random Access Memory*
RANSAC	*Random Sample Consensus*
RGB-D	*Red Green Blue-Depth*
ROI	*Region of Interest*
SIC	*Simultaneous Inverse Compositional*
SIFT	*Scale-Invariant Feature Transform*
SOIL	*Simple OpenGL Image Library*
SSD	*Sum of Squared Differences*
SSIM	*Structural Similarity Index*
STB	*Sean T. Barrett*
SURF	*Speeded-Up Robust Features*
TILDE	*Temporally Invariant Learned DEtector*
TN	*True Negative*
TNR	*True Negative Rate*
TP	*True Positive*
TPR	*True Positive Rate*
VAO	*Vertex Array Object*
VBO	*Vertex Buffer Object*
VGC	*Visual Geometry Group*
XGBoost	*eXtreme Gradient Boosting*
XML	*eXtensible Markup Language*

LISTA DE FIGURAS

LISTA DE TABLAS

1.- INTRODUCCIÓN

En un proceso productivo cuya línea de producción está formada por muchos eslabones es importante controlar el flujo de piezas o productos en ciertas zonas críticas. Uno de estos controles consiste en determinar si las piezas que circulan por cintas transportadoras están bien o mal situadas en base a su pose (posición y orientación). La detección de piezas con poses incorrectas permitiría retirar dichas piezas de la cadena con suficiente antelación como para evitar problemas aguas abajo. Se plantea proponer una solución industrial viable a este problema y abordarlo colateralmente, extendiendo su complejidad empleando una herramienta de simulación potente que sirva de base sólida para el perfeccionamiento de algoritmos para este tipo de problemáticas y la experimentación con diferentes casuísticas.

El objetivo del presente trabajo es proponer una solución industrial viable a este problema. Adicionalmente, se plantea la necesidad de desarrollar una herramienta de simulación, lo más realista posible, de este tipo de problemáticas. Dicha herramienta servirá de base para el desarrollo y ajuste de los algoritmos propuestos sin la necesidad de interferir en la línea real hasta el momento de su implantación. También debe permitir la experimentación con diferentes casuísticas para poder valorar la robustez de las soluciones propuestas.

Este Trabajo Fin de Máster tiene como principales objetivos el diseño de los algoritmos de visión artificial y la arquitectura software sobre la que se implementarán. También se desarrollará una versión funcional que permita llevar a cabo la monitorización en tiempo real y la comunicación entre los diferentes sistemas virtuales y reales (entorno de simulación, sistema operativo, software de la cámara 2D industrial, servidores web…). El sistema de procesamiento de imágenes se centrará en el análisis de la pose 2D de piezas de automoción en una cinta transportadora (posición en el plano y rotación respecto al eje perpendicular al mismo). El simulador incluye el desarrollo de un sistema para manipulación dinámica de modelos tridimensionales de las piezas inspeccionadas, simulación de las imágenes capturadas por la cámara, selección y ajuste de parámetros ópticos e iluminación y, en general, la modelización y el renderizado de un entorno industrial análogo al caso real. El entorno de simulación permitirá desarrollar soluciones basadas en procesamiento mediante técnicas analíticas de visión artificial y también estudiar y experimentar con soluciones basadas en modelos de aprendizaje automático. La salida del sistema de inspección será una señal digital de si la pose de la pieza que se mueve sobre la cinta transportadora es válida o no.

En la siguiente figura se muestra, a grandes rasgos, un diagrama que presenta visualmente las tres partes fundamentales de que consta el trabajo realizado. Un conjunto de interfaces gráficas y de comunicación para establecer nexos entre el procesamiento y la adquisición de frames provenientes de distintos tipos de cámaras (virtuales, reales o

grabaciones), el algoritmo de visión artificial encargado de resolver la tarea mencionada desarrollado mediante las herramientas y librerías que posteriormente se detallarán (como el resto del sistema) y un simulador tridimensional para llevar a cabo parte de la fase experimental y probar o ayudar a reforzar el algoritmo; aparte, se emplea un dataset de imágenes de piezas de referencia (bien orientadas) y vectores de características que mejoran la eficiencia del sistema en tiempo de ejecución y sirven como su segunda entrada.

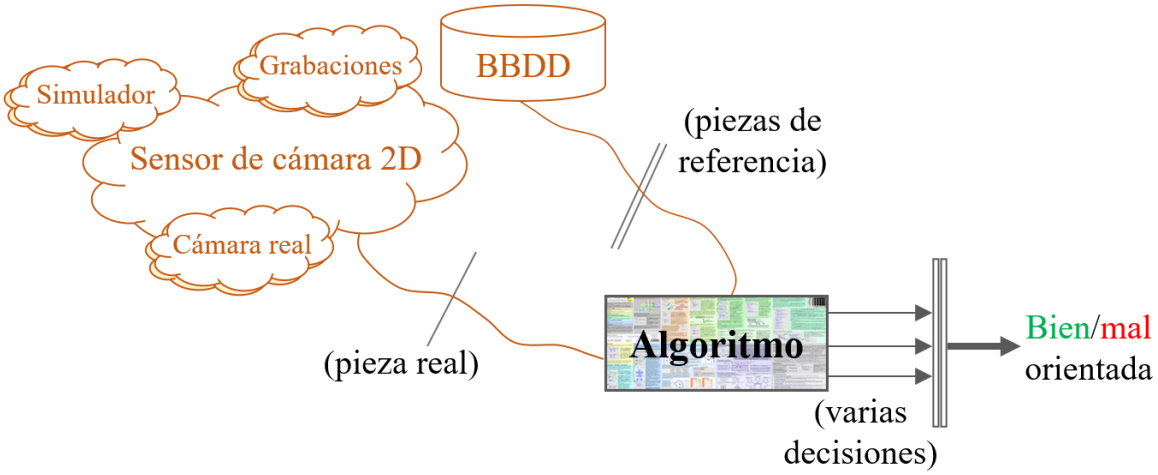


Fig. 1. A grandes rasgos, esquema que resume en conjunto el presente trabajo.

Habiéndose terminado de presentar al final este primer apartado de la memoria los antecedentes, objetivos del proyecto, el estado del arte y puesto el mismo en contexto, en "Sistema implementado" se expone y detalla la estructura y el funcionamiento de todas las partes del sistema (simulador, interfaces y algoritmia), en "Experimentos y resultados" se definen y prueban los métodos diseñados para probar la validez y el desempeño total del sistema y su algoritmo, en el apartado "Discusión" analizan los resultados mostrados en el apartado previo y "Conclusiones y trabajo futuro" se reserva para repasar brevemente el método seguido relacionándolo con los objetivos cumplidos en el trabajo y proponiendo algunas alternativas y trabajo futuro al que se da cabida. Finalmente, lo que resta del documento pertenece al presupuesto realizado para el proyecto y su planificación, además de un compendio de las referencias y bibliografía empleadas a lo largo de todo su desarrollo más varios anexos de interés que se complementan con esta memoria.

1.1.- Antecedentes

Este Trabajo Fin de Máster se ha realizado durante la estancia de prácticas curriculares y extracurriculares en la empresa CIN ADVANCED SYSTEMS GROUP (anteriormente Desarrollo de Soluciones Integrales Plus S.L. [DSI Plus]), un grupo especializado en la implantación de soluciones industriales de digitalización por visión artificial y automatización. Concretamente, las naves industriales y oficinas en que se desarrollaron las prácticas se corresponden con su centro de actividad principal, localizado en Leonardo Da Vinci, 20, 33211, Gijón (Polígono Industrial de Porceyo).

La empresa se ocupa principalmente de proporcionar soluciones de metrología a clientes del sector de la automoción, chapa y forja, principalmente. La empresa diseña y desarrolla máquinas a nivel eléctrico, electrónico, mecánico, robótico, de automatización, así como su software y sistemas de comunicación e información para realizar labores como el sensado, la medición, el análisis de piezas industriales para la clasificación y localización de defectos superficiales y la integración de técnicas y algoritmos de visión artificial para solventar problemáticas particulares de distinta índole que los clientes planteen. La implementación de técnicas de visión artificial resuelve desafíos que pueden llegar a ser muy específicos; este enfoque multidisciplinar refleja la complejidad y diversidad de proyectos donde esta rama del conocimiento desempeña un papel fundamental, proporcionando soluciones precisas y eficientes en entornos industriales.

El trabajo desarrollado se ha realizado mayoritariamente dentro del departamento de informática y visión artificial, en tanto en cuanto las principales actividades llevadas a cabo se supeditan al desarrollo de software, algoritmia y código fuente empleando cámaras 2D industriales. Paralelamente a la elaboración íntegra de este trabajo se han ido implementando y adaptando contribuciones desde múltiples algoritmos, entidades y código fuente a diversos proyectos en los que la empresa estaba involucrada durante los meses de desarrollo; contribuciones que son propiedad de la empresa y ajenas al proyecto que se va a explicar en esta memoria.

La visión artificial se revela como una herramienta esencial para la detección temprana de piezas y la mejora de los procesos industriales. La utilización de cámaras 2D industriales junto con algoritmos especializados posibilita realizar tareas críticas, como la medición, el análisis y la clasificación de componentes industriales a un bajo coste y conceden la posibilidad de explotar el imaginario investigativo que se traduce en la capacidad de desarrollar algoritmos avanzados y soluciones innovadoras que van más allá de la mera captura de imágenes.

A continuación se muestra un grafo visual que compendia los antecedentes, campos de interés y ramas del saber científico e ingenieril más importantes que se relacionan con conceptos relevantes investigados en algún momento del desarrollo y a los cuales el presente trabajo da cabida:

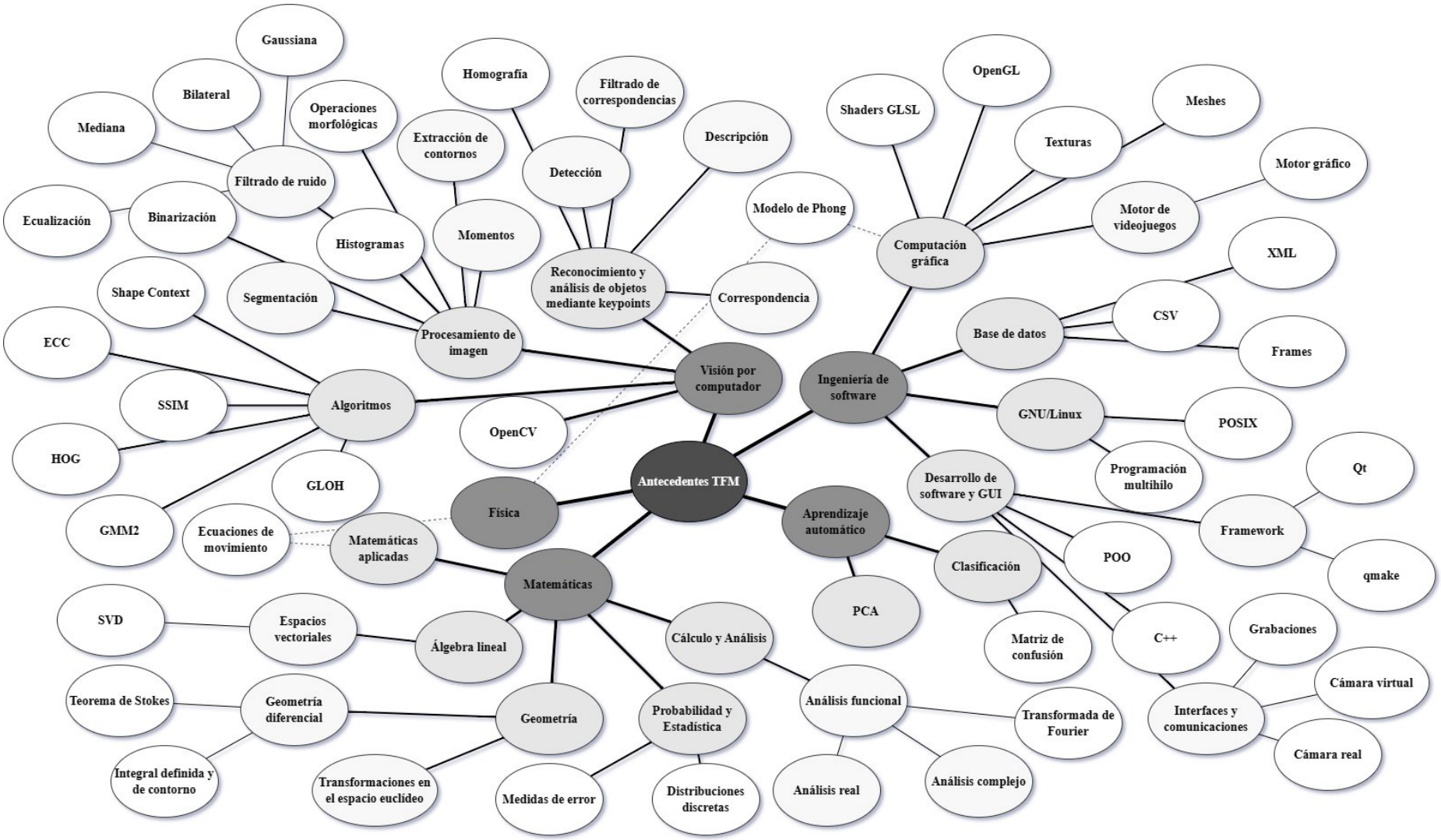


Fig. 2. Mapa conceptual de antecedentes del proyecto.

1.2.- Motivación y objetivos generales

Este Trabajo Fin de Máster tiene como principales objetivos el diseño de los algoritmos de visión artificial, la arquitectura software sobre la que se implementarán y el desarrollo del código fuente de una versión funcional para llevar a cabo la monitorización en tiempo real, la comunicación de diferentes sistemas virtuales y reales (entorno de simulación, sistema operativo, software de la cámara 2D industrial, servidores web…) y la puesta en marcha del sistema de procesamiento de imágenes para el análisis de la pose 2D (en el plano horizontal) de piezas de automoción en una cinta transportadora, incluyendo el desarrollo de un sistema auxiliar de simulación virtual para manipulación dinámica de modelos tridimensionales, extracción de frames, manipulación de parámetros ópticos e iluminación y, en general, la modelización y el renderizado de un entorno industrial análogo al caso real, a fin de demostrar que sirve como apoyo tanto al sistema de procesamiento mediante técnicas analíticas de visión artificial a desarrollar como al estudio y experimentación con modelos de aprendizaje automático para extender la solución, todo ello supeditado a resolver la tarea principal del proyecto, que es la validación de la pose de piezas conocidas *a priori* en una cinta transportadora a fin de determinar si están bien o mal situadas en el plano horizontal.

Un proyecto como este está respaldado por muchos campos del conocimiento, como se pudo ver en los antecedentes; entre muchos otros, el simulador se centra en la computación gráfica, el desarrollo de motores de videojuegos 3D mediante OpenGL y GLSL, la aplicación de modelos físicos de iluminación como el de Phong… los algoritmos desarrollados están circunvalados por numeras técnicas de procesamiento de imagen, detección, descripción y correspondencia de puntos de interés, matemáticas aplicadas y análisis funcional de señales 1D y 2D, etc. y la programación de interfaces compete al manejo de sistemas de base Linux, frameworks como Qt mediante C++ y la creación de una arquitectura software que apoye tiempo real al algoritmo investigado.

1.2.1.- Objetivos particulares

Para llevar a cabo este trabajo se han de desarrollar un conjunto de tareas que permiten perseguir objetivos parciales, culminando con la completitud del algoritmo así como la puesta en funcionamiento del sistema en su conjunto:

- Estudio detallado de la información de partida, incluyendo las herramientas de desarrollo de la empresa existentes para la integración de sistemas de visión artificial en procesos productivos reales y su familiarización con ellos, a fin de actualizar y definir las necesidades para el trabajo, para lo que se debe elegir la arquitectura del sistema y los algoritmos a desarrollar, los tipos de comunicación en tiempo real y offline a emplear y las interfaces encargadas de intercomunicar todas las partes de que se compone el software.

- Manejo de una cámara de procesamiento industrial bidimensional, diseño y programación de la interfaz, tanto en el apartado gráfico como en el de las comunicaciones, para realizar la adquisición y su parametrización, además del planteamiento y el desarrollo de técnicas y algoritmos de procesamiento de imagen y visión artificial para solventar la tarea principal.
- Diseño de una aplicación de escritorio con interfaz gráfica de usuario empleando lenguaje C++ y el entorno de desarrollo Qt Creator en sistemas de base Linux para la explotación y supervisión del proceso y la parametrización proactiva de parámetros del procesamiento. Análisis, estudio e investigación de técnicas y algoritmos de visión artificial y machine learning, su estado del arte, experimentación con diferentes algoritmos de procesamiento mediante el simulador y la GUI de la aplicación desarrollada e implementación de una solución analítica final del problema. Diseño e implementación de todo el software atendiendo a los paradigmas de la POO y (eminentemente) a la versión C++11 del lenguaje.
- Empleo de un motor de renderizado de escenas tridimensionales para proporcionar recursos explotables útiles como la emulación de una cámara y un entorno virtual que incluya la modificación de la distancia focal, el campo de visión, la iluminación, sombras y reflejos, modelos de mallas complejos, sus texturas y la manipulación de su pose, entre otros, a fin de servir como fuente de información y mejora para los algoritmos de procesamiento.
- Configuración de las interfaces de comunicación bidireccional entre la aplicación principal de procesamiento y las fuentes de frames, lo que incluye el sistema de archivos y el SO para establecer una comunicación local en una sola máquina y los servidores y clientes WebSocket para lograr una comunicación remota entre dos o más máquinas en tiempo real a fin de procesar frames local o remotamente.
- Puesta en marcha del sistema de adquisición y procesamiento de la pose de las piezas con demostración del funcionamiento integral del mismo mediante métodos analíticos y demostración de las capacidades del sistema real y virtual para el estudio de su viabilidad enfocado a realizar posibles ampliaciones y mejoras mediante técnicas de machine learning.

A lo sumo, el objetivo principal del proyecto es la búsqueda de una solución analítica funcional al problema de la validación de la pose 2D de piezas conocidas a priori en una cinta transportadora para determinar si están bien o mal situadas restringiéndolas a un rango de validez bidimensional prefijado. Para ello se emplean diversos algoritmos analíticos y técnicas de detección, descripción y emparejamiento de puntos característicos a la par que múltiples técnicas de procesamiento de imagen y análisis matemático. Además, se implementa un simulador, una interfaz gráfica de usuario y las debidas interfaces de comunicación para demostrar la capacidad de emular diferentes casuísticas tanto para la aplicación como para el estudio del desempeño de los algoritmos implementados y de la posibilidad de integrar otro tipo de modelos adecuados a problemas similares.

1.3.- Estado del arte

La visión artificial y el aprendizaje automático pueden ayudar a determinar la validez de los productos (bajo distintos supuestos) mediante su sometimiento a pruebas de seguridad y funcionalidad antes de que puedan transferirse entre dos etapas consecutivas. Como el error humano es el más natural de los errores, los sistemas artificiales se diseñan especial y específicamente para sobrepasar inconveniencias en tareas repetitivas y tediosas. Más aún, discerniendo puntillosamente los pliegues de las soluciones actuales pueden aparecer hiperparámetros que se antojan calibrar. La orientación de las piezas es uno de estos componentes que conforman el vector de incógnitas que muchas veces causa que ciertos defectos superficiales no puedan ser reconocidos u otras partes del sistema de automatización fallen espuria y extravagantemente. Por eso es recomendable que una pieza esté correctamente orientada antes de que pueda pasar un control de calidad fidedigno. Esta tarea constituye el objetivo de realización principal del presente proyecto.

Los controles de calidad en la industria, en especial en el campo que abarca la automatización y automoción, son sensibles a muchos tipos de perturbaciones que, ya sea desde fuentes humanas o artificiales, comprometen el cumplimiento de los estándares geométricos requeridos por los productos, el tiempo de ciclo de las etapas en línea de fabricación y otros aspectos. La progresiva robustez ante la detección de defectos superficiales, intrínsecos, de composición… de los productos sigue, tras décadas, mejorando los sistemas de metrología, facilitando así un control total y previsible de la producción. Hay otros problemas latentes que deben ser localizados y solventados para seguir mejorando, desde otros puntos de vista, sistemas que estén diseñados para mejorar holísticamente la eficiencia de la producción y disminuir la pérdida de material, tiempo y capital. El punto de vista desde el que aparece la necesidad de realizar este proyecto se corresponde con una situación en la que necesita verificar si la orientación de las piezas que atraviesan una cinta transportadora es correcta o no de acuerdo con unos requisitos, diseñando un sistema que emplea un sensor tan versátil y económico como una cámara bidimensional sin color.

Otros sistemas [1] complementan y fusionan diversos sensores, dispuestos radial o longitudinalmente a las piezas, de forma que “observen” desde diferentes perspectivas la pieza, a fin de extraer información de diferentes partes de la misma de manera concurrente. Se hace notar que la mezcla inteligente y fiel, *a posteriori*, de la información procedente de todos los sensores es más compleja en tanto en cuanto el número de sensores aumenta, dado que tanto la captura de información como el procesamiento de los datos se ven influenciados a una mayor cantidad de ruido, mas también por una mayor cantidad de información. Pero es loable y a veces preferible (sobre todo cuando la carga computacional es un impedimento, los retrasos temporales deben ser evitados o la eficacia de la aplicación es preponderante) explotar las capacidades de un solo sensor recurriendo a las herramientas necesarias y adicionales que suplementen su supuesta “falta de información”, lo que parece ser el caso de este proyecto.

La consecución de la germinación de conocimiento supradimensional partiendo de datos de menor dimensión es de especial interés porque puede desembocar en la adaptación de sistemas a soluciones menos costosas consiguiendo resultados y rendimientos similares, destilando la tecnología más directa y primaria para convertirla en ideas complejas y alternativas, como es el caso de la obtención de información sobre la profundidad de los objetos en un escena partiendo de imágenes bidimensionales, por ejemplo, mediante sistemas estéreo, multivista o técnicas de tipo "shape from" (*shape from shading*, *shape from motion*…). En [2] presentan un método para computar eficientemente momentos volumétricos de un objeto partiendo de los momentos bidimensionales de distintas proyecciones del objeto, consiguiendo reducir la complejidad del problema en una cantidad de raíz cúbica sin desmejorar el rendimiento.

En [3] proponen un algoritmo de aprendizaje automático muy relevante destinado a obtener una representación matemática (modelo neuronal) de la relación existente entre imágenes bidimensionales en poses arbitrarias y su orientación espacial tridimensional, consiguiendo un sistema que logra aprender la orientación de diferentes clases de objetos como una función de un sola imagen, lo cual representa un problema especialmente delicado debido a que muchas veces la localización en el espacio está subdeterminado debido a la pérdida de información en una imagen. También proponen una solución a la dubitativa que surge al decidir una orientación "correcta" de entre las múltiples disponibles para objetos simétricos. En este proyecto la orientación se puede suponer contenida en un espacio cartesiano (aunque internamente se pueda trabajar con representaciones tridimensionales de las piezas) y, aunque en el artículo mencionado resalta la dificultad que conlleva la representación de la orientación en espacios (esféricos) no euclídeos, la ambigüedad sigue aquí latente para objetos simétricos.

Al igual que la orientación de las piezas es un parámetro espacialmente relevante en procedimientos que involucran la manipulación mediante brazos robóticos, la orientación de la superficie y las deformaciones elásticas, también delimita la consecución de una manipulación más fina de la pieza. En [4] proponen un algoritmo encargado de la detección y modelización de deformaciones en superficies que debieran ser planas para evitar infortunios en ulteriores procedimientos de control visual, inspección y manipulación.

En lo relativo a extraer las características "más definitorias" de los objetos conocidos de cara a disponer de un registro que mantenga encapsulada la información que mejor explique la orientación de estos, la literatura contiene una cantidad enorme de métodos y algoritmos para detectar puntos y regiones de interés de distinta índole, mas considerando que, para este caso particular, lo más interesante sea quizá recoger las características contenidas en el borde o los contornos de los objetos (pues las piezas son prácticamente lisas, su superficie es homogénea y su textura similar, aunque presenten diferentes geometrías, mecanizados y agujeros) y establecer una buena transformación geométrica proyectiva entre las imágenes reales y las de referencia, se ha de realizar un estudio y una elección cuidadosa de los detectores y descriptores que más se adecuen a los intereses

propios, e incluso considerar métodos de aprendizaje profundo en vez de tradicionales para dichas tareas. En [5] se presenta y compendia una discusión cualitativa y cuantitativa para varias aplicaciones reales de las bondades y defectos de diferentes tipos de detectores (FAST, BRISK, ORB, SURF, SIFT, KAZE) y descriptores (BRISK, FREAK, BRIEF, SURF, ORB, SIFT, KAZE), así como otras alternativas de aprendizaje automático tanto para la detección (DetNet, TILDE, LIFT, detector multiescala, SuperPoint) como para la descripción preentrenada (Siamese, LIFT, red triplet, SuperPoint) de las características.

En [6] los autores tratan sobre la importancia de la estimación de pose tridimensional de objetos a partir de imágenes o videos en 2D o 3D, así como cualitativamente diversos métodos para ello, como los enfoques basados en modelos, en características, directos, híbridos y basados en aprendizaje profundo y las fortalezas y debilidades específicas de cada uno. Se destaca que la alta precisión que suele distinguir a los métodos puramente analíticos en la estimación de pose en el espacio 3D, así como su versatilidad en muy diversas aplicaciones y buen desempeño en el seguimiento en tiempo real y naturaleza no invasiva (en contraposición a otros métodos con sensores acelerómetros, por ejemplo), constituyen eminentes ventajas para estos. Sin embargo, desafíos como la complejidad computacional, precisión limitada en ciertas condiciones de iluminación adversa y oclusiones, la aplicabilidad limitada a objetos complejos y la sensibilidad al ruido en los datos de entrada son sus limitaciones típicas. Es mandatorio realizar un análisis consciente de la naturaleza y el contorno del problema para considerar el empleo y las (des)ventajas de todos los métodos existentes, como los basados en modelos, características, los directos, híbridos y basados en aprendizaje profundo y elegir aquellos que mejor se adapten a diferentes requisitos y escenarios. Por ejemplo, los métodos basados en modelos destacan cuando se tratan objetos geométricamente claros, mientras que los basados en características ofrecen versatilidad a este respeto pero pueden carecer de precisión para características menos distintivas. También, los métodos directos son adecuados para objetos no rígidos y los híbridos combinan fortalezas para lograr mejorar la precisión y robustez. Por otra parte, los métodos basados en aprendizaje profundo aprovechan el entrenamiento de agentes inteligentes para predecir directamente la pose, eliminando la necesidad de modelos geométricos explícitos. En resumidas cuentas, es claro que la elección de un método final depende de factores tales como los requisitos de precisión, los recursos computacionales y la adaptabilidad a datos y objetos diversos.

Similarmente, en el muy reciente artículo [7] se presenta una implementación de una plataforma de transporte de cargas equipados con sistemas de clasificación y brazos robóticos de apilamiento con los que se estima la pose 3D de sus ítems mediante una cámara de profundidad y se reajustan dinámicamente, superando las complejidades asociadas al aprendizaje profundo y el registro de datos. En el método propuesto para estimar la desviación angular de las cajas de carga a partir de la adquisición de imágenes se utiliza la detección de contornos de Canny en una región de interés que representa la ubicación de la carga. Luego, la transformada de Hough se aplica a la imagen binaria resultante para convertir las líneas cartesianas en ecuaciones paramétricas empleando

coordenadas polares. A partir de los segmentos de línea más largos identificados se calcula directamente la pendiente y el ángulo de inclinación y, luego, se utiliza un método iterativo para extraer la nube de puntos tridimensional de la caja de carga utilizando técnicas como el filtrado condicional y el algoritmo RANSAC para reconocer líneas de borde rectangulares. Posteriormente, se emplea el análisis de componentes principales (PCA) para calcular los vectores normales a los puntos del objeto y diseñar regiones delimitadoras tanto alineadas con los ejes de cartesianos (AABB) como orientadas a lo largo de los componentes principales (OBB) para la nube de puntos de la caja de carga; en concreto, se logra la alineación de la imagen con la nube de puntos y se utiliza la región delimitadora alineada con el eje de la nube de puntos (AABB) para discernir la posición de la caja en relación con el sistema de coordenadas de la cámara de profundidad, para después, con la transformada de Hough, extraer los segmentos de línea de la región de interés dentro de la imagen (OBB) y RANSAC para ajustar líneas en la nube de puntos. En comparación con las imágenes bidimensionales, los datos de nubes de puntos demuestran invarianza de forma durante las transformaciones rígidas, lo que implica la robustez de sus algoritmos de procesamiento contra rotaciones y traslaciones. Sin embargo, el manejo de grandes conjuntos de datos de profundidad presenta desafíos computacionales, mientras que si son pequeños pueden llevar a cálculos inexactos; por eso se aborda en el mencionado artículo el uso de RANSAC para ajustar líneas de contornos en la nube de puntos, permitiendo una extracción estable de bordes, superponiendo los datos de la imagen y la nube de puntos para superar las limitaciones inherentes a cada método y eliminando así la necesidad de aprendizaje profundo para este tipo de aplicaciones. Este enfoque proporciona información detallada sobre la orientación y la posición tridimensional de la caja de carga en el espacio; un experimento comparativo utilizando sensores acelerómetros para la adquisición de la pose reveló una desviación de menos de 0,7 º entre los dos procesos, lo que es un resultado prometedor.

Un interesante enfoque a la neurociencia se lleva a cabo en [8], donde se intenta mejorar las habilidades de los sistemas robóticos en su interacción con objetos cercanos mediante la estimación visual a través de la fusión de diferentes señales visuales de un mismo estímulo; se implementa un modelo computacional de estimadores de orientación estereoscópica y perspectiva basados en conceptos neuropsicológicos (se aborda la importancia de la integración de señales visuales en el córtex sensorial de los primates), fusionados según diferentes criterios, en un entorno robótico y se prueba en diversas condiciones. El trabajo incluye la utilización de la perspectiva y estereopsis para estimar la orientación de objetos y un estimador de distancia basado en datos propioceptivos de convergencia (referido a la información sensorial interna generada por el sistema propioceptivo del robot, es decir, la fijación del movimiento de los sensores visuales hacia la línea media del objeto para estimar la distancia). Para calcular la orientación del objeto el modelo emplea la diferencia entre los ángulos retinianos de los puntos en la imagen capturados por los «ojos» izquierdo y derecho, para estimar la distancia del centro óptico de visión al objeto se utiliza el mencionado ángulo de convergencia de los ojos y, finalmente, se fusionan las dos estimaciones mediante un enfoque de combinación

ponderada de indicios, esto es, la información de estereopsis (información 3D basada en la disparidad entre las imágenes capturadas por ambos ojos) con los datos de perspectiva y convergencia ocular. Los resultados experimentales sugieren que la integración de señales monoculares y binoculares puede hacer que los sistemas sensoriales de los robots sean más confiables y versátiles, además de que el estimador combinado (ponderando adecuadamente las diferentes señales) es robusto en diversas condiciones de trabajo.

También, en [9], los autores presentan un sistema de *Pick-And-Place* que identifica objetos de geometría simple, estima su orientación 2D y calcula los dos puntos óptimos sobre los que realizar el agarre con un *gripper* de dos dedos manipulador. Se utiliza un algoritmo que, tras el aislamiento de los objetos en la escena a través de su binarización, segmentación y la extracción de su geometría empleando operadores de gradiente en su región de interés delimitada y ordenando en un arreglo unidimensional los puntos del contorno según el ángulo con respecto al centroide, determina su orientación mediante e identifica del objeto clasificándolos en dos categorías (alargados y no alargados) usando un enfoque de ajuste de recta mediante el error cuadrático medio y valores propios de la matriz de la matriz de dispersión (*scatter matrix*, que es una herramienta intermedia utilizada para estimar la matriz de covarianza, pues esta última normaliza la primera dividiéndola por el número de observaciones) de las coordenadas de todos los puntos del contorno, identificando cada objeto mediante un perfil único en función de su radio y ángulo y, finalmente, selecciona los dos puntos más adecuados para el agarre basándose en la minimización de la distancia entre puntos opuestos, considerando la apertura mínima de la mano robótica. El trabajo resalta la efectividad del algoritmo para las condiciones de contorno consideradas inicialmente y su potencial adaptabilidad a diferentes tipos de pinzas más una sugerencia de incorporar un sensor de corriente para evaluar la fuerza de agarre.

Una integración compacta y concisa de redes neuronales en aplicaciones de inspección de calidad es presentada en [10], donde para el entrenamiento de la red (un perceptrón multicapa aprendiendo mediante “propagación hacia atrás”) utilizan un vector de características binario que recoge la presencia o falta de chavetas en piezas circulares en una escala de orientaciones discretizada cada 12 grados sexagesimales y, por otro lado, recurren a otro vector que recoge el histograma en niveles de gris las imágenes para identificar si un producto es defectuoso o no. Como se hace notar en el artículo mencionado, la mejora tras la elección de esa topología de red (perceptrón multicapa) y ciertas características específicas y manualmente seleccionadas de las imágenes, frente a otras redes de tipo convolucional que extraen características de manera ciertamente agnóstica de cara al exterior, destaca la importancia de estudiar minuciosamente los puntos débiles de un problema y tener en consideración particular y personal aspectos a nivel de procesamiento de imagen que proporcionen información inteligible sobre el caso de uso a resolver (en este caso, la extracción de la orientación de una pieza).

En [11] se presenta una aplicación virtualmente relacionada con algunos métodos considerados en este proyecto; los autores tratan de utilizar modelos de redes neuronales

para estimar y aplicar de manera conveniente las transformaciones geométricas subyacentes a dos imágenes con objetos, fondo y perspectivas diferentes. Primero proponen una red convolucional para el establecimiento de correspondencias entre dos imágenes imitando los pasos fundamentales típicos de las técnicas clásicas: la detección de los puntos de interés y extracción de sus características, el emparejamiento de los mismos desechando los valores atípicos y la estimación de los parámetros del modelo geométrico (afinidad, homografía, spline de plano fino…) que relaciona las imágenes. También proponen un moderno método de ajuste y entrenamiento de los parámetros de la red para mejorar su capacidad de generalización mediante imágenes sintéticas (sin necesidad de un etiquetado manual). La solución neuronal reemplaza las características locales por características "ocultas" propias del aprendizaje de la red y, con respecto a la tradicional (v. g. búsqueda de correspondencias locales con SIFT y RANSAC), pretende sobrepasar las situaciones perjudiciales en que los modelos contienen demasiados parámetros como para ajustarlos de manera robusta evitando los outliers, provocadas por las deformaciones "no rígidas" o los cambios espurios en la apariencia de los objetos de una misma clase. Esta idea se podría aprovechar en este proyecto para establecer mejores correspondencias entre frames cuyos objetos se ven sometidos a efectos de la perspectiva debido al movimiento excesivo en el campo de visión de la cámara, distorsiones en su lente o variaciones en la pose de la misma (de suerte, para tener un modelo neuronal más robusto).

Trabajos como [12-14] proponen varias CNN profundas que resuelven el problema de la clasificación (encontrar la categoría a la que pertenece) y localización de objetos (determinación de su posición relativa y orientación tridimensional [que se puede deducir de la pose relativa de la cámara]); el primero empleando imágenes únicas del objeto (minimizando una función de pérdida *triplet loss*), el segundo usando múltiples imágenes (extendiendo al anterior) (proporcionando directamente a la salida de la red la probabilidad de pertenencia de cada objeto a una clase y su orientación) y el tercero utilizando múltiples vistas (convirtiendo en variedades[1] 2D, para cada clase, las imágenes o vistas de entrada y conformando una base de datos constituida por las clases de los objetos, las orientaciones y de las cámaras asociadas y los descriptores) suponiendo que se conoce la posición de las cámaras en el sistema del mundo, a fin de comparar la estimación "en línea" de la red con la base de datos de entrenamiento para buscar los elementos (descriptores) más parecidos que cumplen la restricción de la orientación de la cámara. Estas propuestas buscan desarrollar sistemas de inteligencia artificial "end-to-end"[2] evolucionados no condicionados por características extraídas "manualmente" mediante técnicas tradicionales

[1] Variedad (*manifold*): en el contexto de la geometría y topología diferencial, es una noción generalizada de superficie que puede tener curvatura y forma más complicadas que en el espacio euclídeo convencional; localmente puede asemejarse a un espacio euclídeo (o se comporta como tal en la vecindad de cada punto) pero globalmente puede tener estructuras topológicas y geometrías más complejas.

[2] End-to-end (E2E): se refiere a la habilidad o capacidad de muchos sistemas o modelos de aprendizaje automático profundos de encapsular o albergar intrínsecamente la solución a un problema que es, por definición, complejo o extenso, de manera compacta en una sola representación matemática (que es el modelo), en contraposición con los sistemas tradicionales que habitúan a añadir capas intermedias específicas al flujo de trabajo desatendiendo al papel holístico como sistema de "caja negra".

como HOG, SURF… sino a través de las aprendidas estadísticamente por la estructura de la red. Se extraen dos conclusiones del último trabajo: las restricciones impuestas por el conocimiento de la posición relativa de las cámara y la interpolación de los datos de entrenamiento (orientaciones en la base de datos) mejoran la precisión de la estimación.

Aplicando un enfoque similar pero en otra aplicación, para teledetección, en [15] tratan de estimar regiones de interés de manera robusta ante orientaciones y superposiciones imprevisibles de edificios; se adaptan los métodos tradicionales de cajas delimitadores (ROI) alineadas con los ejes de la imagen para adoptar un uso de ROI orientadas arbitrariamente según la orientación de los edificios, empleando una CNN para la extracción de características y regresión de las ROI. El modelo se entrena para predecir simultáneamente la orientación, ubicación y extensión de las ROI, lo que resulta en un ajuste más fino y, por lo tanto, un aumento en la precisión promedio.

Existen multitud de trabajos (véase [16] y sus referencias) que utilizan datos de sensores más sofisticados, como nubes de puntos provenientes de sensores de triangulación láser o cámaras de profundidad RGB-D o tiempo de vuelo, para obtener estimaciones más precisas y manejar formas objetos más complejas, sin embargo, estos enfoques pueden ser costosos en términos de esfuerzo computacional y recursos. Como el presente proyecto, otras soluciones, algunas ya mencionadas, se basan en imágenes de canales RGB o monocanales como entrada, lo cual es una elección común debido a su disponibilidad y facilidad de adquisición, además ser más accesible y aplicable en una gran variedad de escenarios flexiblemente, pudiendo explotar inteligentemente las posibilidades que brinda en conjunción con otras técnicas.

Que la estimación de la pose a partir de imágenes RGB es una tarea notablemente desafiante (mucho más en objetos ambiguos de textura uniforme y fuertemente simétricos) se remarca en [17], en donde se trata de predecir la posición y orientación de objetos conocidos (representados mediante un cubo circunscrito orientado) mejorando el desempeño con respecto a métodos antiguos que empleaban *autoencoders* que codificaban la orientación implícita de los objetos (para no depender de una etapa explícita de etiquetado) [18-19]. Los autores proponen, primeramente, emplear algoritmos de extracción de contornos y combinarlos con las imágenes en color para proporcionar a la red características más discriminantes y que hagan de las imágenes sintéticas usadas para entrenar y las imágenes reales dos conjuntos de datos más parecidos; en segundo lugar, se mejora la regularidad en la codificación de la pose implícita aprendida gracias a un enfoque autosupervisado mediante la restricción de que para cualquier rotación de un objeto (dado un conjunto disperso de muestras) su codificación latente debería estar suficientemente cerca de la codificación de la rotación de referencia más cercana, considerando que las representaciones latentes de dos imágenes que presentan rotaciones cercanas deberían estar también cercanas en el espacio latente (se menciona que un problema frecuente de los autoencoders [aumentados] es la falta de regularización en el espacio latente aprendido, en el sentido de que cambios en la pose pueden no mapearse en cambios en la codificación latente). Se hace uso de una red convolucional para detectar

objetos, un autoencoder entrenado que codifica la orientación implícita de estos y se infiere la traslación comparando la escala de los cuadros delimitadores o regiones rectangulares del objeto de referencia y del real.

La observación e interpretación intensiva de imágenes RGB también se lleva a cabo en [20], donde los autores proponen una red neuronal end-to-end que estima la pose (sus 6 grados de libertad) de los objetos reconocidos en una escena. Primeramente emplean una red de segmentación (DRN) para extraer máscaras etiquetadas de los objetos reconocidos en la escena y posteriormente proporcionan cada máscara binarizada y clasificada a otra red (un extractor de características ResNet-18 seguido de un MLP) concatenada que las interpreta y estima la pose 6D de cada objeto. Cada red está entrenada con imágenes RGB sintéticas de objetos conocidos a priori; la dos redes conforman un sistema global que, en línea, se ejecuta en tiempo real a 20 Hz bajo imágenes RGB, sin utilizar información de profundidad ni ajustes ICP[3]. El MLP consiste en una capa totalmente conectada de 256 nodos seguida de dos ramas paralelas correspondientes a la posición y orientación, respectivamente; por adición, la red de interpretación de la pose se entrena para 5 objetos distintos, por lo que la rama de posición tiene 15 salidas (3 [x, y, z] para cada objeto) y la de orientación 20 (4 para cada objeto [cuatro componentes del cuaternión relativo al marco de coordenadas de la cámara], normalizadas a una magnitud unitaria por necesidad al interpretarlas). El trabajo introduce una nueva función de pérdida denominada *Point Cloud L1 Loss* para la estimación de poses y propone mejorar la robustez del sistema entrenado las red con objetos parcialmente ocluidos o máscaras deformadas u ocultas artificialmente, además los autores demuestran que eliminando la capa de agrupamiento promedio global del extractor de características preserva la información espacial de los mapas de características.

Es de hacer notar la importancia de realizar un entrenamiento adecuado y contemplativo de un modelo de inteligencia artificial. Como se comenta en [21], aunque las técnicas de extracción de características tradicionales son precisas y efectivas, ocasionalmente no son demasiado adecuadas para ciertas aplicaciones, como las que conllevan la manipulación con robots, pues es mandatorio disponer de un entorno perfectamente controlado y objetos rígidos con texturas muy detalladas. Si una tarea a realizar es, verbigracia, reposicionar una pieza arbitraria a una orientación requerida mediante un brazo robótico, las soluciones con redes convolucionales demostraron ser más capaces de proporcionar mejores resultados en entornos no controlados, aunque atadas al problema de tener que realizar y disponer de extensas bases de datos para el entrenamiento (y en parte por esto tales enfoques suelen restringirse a tareas generales de reconocimiento de objetos). En el artículo mencionado presentan un nuevo enfoque que combina la robustez de las CNN con la estimación “fina” de la pose 3D (6 grados de libertad) de objetos instanciados en la escena; se enfrentan al sub/sobreentrenamiento generando

[3] ICP (*Iterative Closest Point*): un tipo de algoritmo utilizada en visión por computador para hacer corresponder y alinear conjuntos de puntos en el espacio tridimensional (nubes de puntos), típicamente mediante métodos iterativos de minimización de la distancia entre los puntos más cercanos.

imágenes en entornos (y con objetos) puntualmente aleatorios y estudian experimentalmente el desempeño del modelo entrenado con imágenes (128 x 128 px., RGB) de objetos sintéticos aplicado a objetos reales. El modelo global se compone de una primera red convolucional (Faster R-CNN) que detecta y clasifica regiones en la imagen (devolviendo a su salida la "bounding box" de cada objetos con sus etiquetas correspondientes) y otra red (VGG-16 más capas totalmente conectadas), inspirada por [22]) especializada para cada objeto que estima sus poses. La segunda red está entrenada con los objetos conocidos bajo diferentes puntos de vista y resuelve, realmente, el problema de "encontrar la pose de la cámara" relativa a la bounding box del objeto de entrada, proporcionando como salida tres ángulos estimados (relacionados con el movimiento de la cámara) al estimador de la pose del objeto dentro de la bounding box. Se muestra que los modelos se pueden entrenar enteramente con imágenes sintéticas y aun así proporcionar resultados complacientes para aplicaciones reales de agarre de objetos conocidos con manipuladores robóticos, por ejemplo.

Por otra parte, en [23] se presenta un marco unificado para la detección y clasificación de objetos mediante nubes de puntos y aprendizaje profundo enfocado a la recogida y colocación de objetos mediante robots industriales. La solución tiene la particularidad de que para la clasificación la CNN utilizada (inspirada en VGG [VGG(2/3D)(Net)(-16…)… es una familia de redes basada en la red convolucional muy profunda propuesta por K. Simonyan y A. Zisserman —en *Visual Geometry Group*—]) combina los dos flujos de información existentes, correspondientes a las características extraídas de los datos RBG (imágenes escaladas a 64 x 64 y normalizadas) y las extraídas de los datos de profundidad. Así, la propia red bifurca dos flujos de características independientes (vectores de características de 512 dimensiones cada uno) y ulteriormente las combina (en forma de un vector de 1024 dimensiones) en las capas totalmente conectadas para la clasificación (51 categorías). Las dos corrientes de características se entrenan independientemente y luego se realiza un tercer entrenamiento en conjunto, mediante un optimizador de Descenso de Gradiente Estocástico (SGD) para minimizar la función de pérdida. El enfoque de "congelar" una de las corrientes durante el entrenamiento de la red mostró un mejor rendimiento en comparación con retropropagar los gradientes a través de toda la red durante el proceso de entrenamiento.

Un enfoque similar al de este proyecto se aborda en [24], donde se trata de realizar una clasificación binaria de imágenes de calor (radiación infrarroja) para monitorizar el estado de máquinas mecánicas, mejorando los sistemas basados en vibraciones y señales acústicas. Dotando de gran importancia a una etapa previa de preprocesamiento y segmentación de las máquinas en las imágenes (normalización de intensidad, ecualización adaptativa de histograma limitado por contraste [CLAHE], algoritmo de medios no locales rápidos [FNLM], detección de valores atípicos basándose en el rango intercuartílico [IQR]…), los autores recurren a una CNN para clasificar las máquinas sobrecalentadas (clases de interés) de otros objetos en los fondos complejos que pudieran ser ruido o máquinas en estado normal, realizando una clasificación binaria directa con las redes

neuronales. Es un ejemplo de aplicación en la que la mayor importancia recae en las etapas de procesamiento de imagen para adaptar los datos de entrada al modelo de inteligencia artificial para que este aprenda las características más discriminantes intrínsecas en la secuencia de entrenamiento, mas una selección más minuciosa y deliberada de las mismas podría resultar más fructuoso de cara a la precisión y capacidad de generalización de la red.

Un problema de índole similar al presente se trata en [25] para estimar la pose 6D (esto es, las tres rotaciones y traslaciones con respecto a un origen en el marco de coordenadas del mundo) de naves espaciales en órbita terrestre. Los métodos más típicos hubieran considerado bien la regresión directa de los parámetros buscados con imágenes en redes convolucionales o bien la comparación de los keypoints observados con otros keypoints predefinidos en una pose conocida para estimar la pose mediante la transformación proyectiva que lleva los "n" keypoints a sus relativos. Los autores, alternativamente, proponen emplear una red convolucional para predecir la localización de los keypoints que, posteriormente, es transferida a una segunda red para estimar o inferir la pose 6D incógnita. Se verifica una pérdida nula de precisión en comparación con métodos tradicionales y se garantiza un entrenamiento end-to-end, lo cual es beneficioso para la representación de un sistema compacto y un procesamiento en línea potencialmente más rápido.

Finalmente, merece la pena resaltar el trabajo abordado en [26], donde se presenta una CNN profunda denominada HomographyNet, que consta de 10 capas y toma como entrada dos imágenes en escala de grises apiladas, destinada a estimar la homografía relativa entre un par de imágenes. Se proponen dos arquitecturas de red: una red de regresión que estima directamente los parámetros reales de la homografía y una red de clasificación que produce una distribución sobre homografías cuantizadas (en esta última el modelo proporciona una distribución de probabilidad sobre ciertos valores discretos que representan distintas homografías posibles). Se discute la limitación de los métodos tradicionales analíticos que requieren etapas separadas de detección de características locales y estimación de transformación, proponiendo la posibilidad de que un algoritmo aprenda sus propios conjuntos de primitivas de manera end-to-end.

En [27] se presenta de manera introductoria la esquematización y el detalle de múltiples métodos de aprendizaje automático profundo para la detección de la pose de objetos empleando técnica monoculares, tanto aquellos cuya única entrada es la imagen (de profundidad o no) a analizar como los que proponen múltiples estimaciones, las combinan o se valen de modelos tridimensionales de referencia y estimaciones iniciales; demuestran la severa importancia que recientemente han demostrado la estimación de pose y el seguimiento de objetos en campos como la robótica o conducción autónoma y el papel más que prometedor que el deep learning está acuñando. Aunque en el presente proyecto no se contempla seguir la ruta de la inteligencia artificial, los trabajos mencionados proporcionan un punto de vista complementario cuyos objetivos se pretender cubrir holgadamente con técnicas puramente analíticas.

1.4.- Problema a resolver, solución propuesta y resumen del método

En este apartado se va a presentar ordenadamente la solución propuesta que resuelve el problema que define al presente proyecto. Lo dicho compete al enunciado del mismo (de modo similar a como el cliente lo plantearía), las distintas alternativas que, aunque en el proyecto finalmente presentado no se hubieran implementado, sí se consideraron en algún momento en etapas previas y, por último, el listado especificado de herramientas y útiles que sí se decidieron emplear y forman parte de la solución final comprendida y compendiada en este documento.

En el presente caso de estudio se trata de analizar la postura de piezas durante su paso por una cinta transportadora mediante cámaras 2D estáticas a fin de determinar, bajo cierto conocimiento a priori, si su colocación o disposición en una línea de producción o cinta de transporte es válida. Similarmente, se trata de saber si estas piezas están debida o correctamente orientadas o no mediante imágenes de referencia; así, se impone necesario un estudio de algoritmos de visión artificial para la determinación de estos parámetros en base a fotogramas de la pieza actual y varias piezas de referencia más su consolidación en un sistema software que abstraiga sus entradas, salidas y su funcionamiento interno del usuario externo e implantación de todo esto mediante un lenguaje de programación en una plataforma informática adecuada. Por adición, se da la coyuntura de que la existencia y disponibilidad de las debidas herramientas de simulación y experimentación proporcionan una manera sumamente flexible y favorable de atacar el problema; así pues, no sin la premisa principal de llegar a alcanzar un algoritmo viable y funcional en su conjunto, durante el desarrollo del trabajo se agregan diversos módulos software, como interfaces de comunicación o gráficas, aplicaciones de simulación 3D y monitorización, de configuración de las cámaras, de todo tipo de parámetros de los algoritmos de procesamiento… que impulsan el rendimiento y desempeño del proyecto para aumentar el acervo de casuísticas, posibilidades y su potencial usabilidad para crear y refinar más soluciones de esta índole.

A continuación se listan varias especificaciones del proyecto que sirven como condiciones de contorno que, desde el lado del cliente, impondrían las restricciones requeridas y enmarcan el trabajo en un abanico más específico y menos amplio sobre el que desenvolverse (punto muy importante para desarrollar una experimentación adecuada). El número de piezas es conocido, así como sus dimensiones; en principio siempre se conoce la referencia a inspeccionar y se dispone de imágenes de cada una en vista cenital. Se dispone de todas las piezas a inspeccionar, así como de todos sus modelos CAD o *mesh* (STEP, STL, DAE…), lo que facilita su integración en la herramienta de simulación. Además, puede ser accesible la información de bases de datos (o el sistema de archivos local) en tiempo de ejecución aunque el funcionamiento sea en línea, lo que puede facilitar el acceso a características precalculadas de las piezas. No es necesario detectar que una

pieza real se corresponda inequívocamente con otra del database (lo que sería necesario si, p. ej., el cliente quisiera, aun estando la pieza mal orientada, reconocerla para posteriormente recolocarla con un brazo robótico en vez de desecharla); basta con desecharla si se considera que tiene una colocación muy errónea, se haya asociado o no con alguna del database, aunque bien es cierto que desecharla significa reconocerla como una pieza inválida (esto se aprovecha para tratar al problema como un algoritmo de clasificación y no puramente de regresión; explicado en detalle en el apartado "Algoritmo"). Es decir, el sistema computa una orientación y posición, devuelve una salida digital (respuesta binaria) correspondiente a la validez de la pieza y en caso de que no se pueda calcular se devolverá un error, tratándose de una forma pertinente. Las piezas pueden aparecer dadas la vuelta (giradas en *pitch* y *roll*) y no solo rotadas en el plano horizontal alrededor del eje z (*yaw*), con lo que el emparejamiento con piezas de la database podría no dar resultado; en este caso, análogamente, se dará una señal de error. Se puede ver más de una pieza en cada fotograma, pues pueden no estar lo suficientemente separadas; esto depende de la apertura de la cámara y la distancia de trabajo pero es posible filtrar el número de piezas por fotograma en dicho caso. Las imágenes pueden ser en niveles de gris (lo cual se considera suficiente) o en canales de color. La posición de la cámara se puede considerar siempre cenital; está suficientemente bien fija mecánicamente y se pueden obviar las vibraciones. La iluminación será controlada (encima de la estructura hay luminarias fijas) pero puede haber ciertos reflejos en la superficie de las piezas. Además, al tener la cámara fija y no moverse la cinta, no se considera necesario realizar procesos de calibración cíclica de la cámara. Finalmente, la cámara estaría adquiriendo continuamente, mientras que el algoritmo se activaría en los momentos en que pasen piezas por la cinta (momentos que serían notificados; puede estar inactiva en momentos en los que no haya piezas), enviando un resultado cada vez que encuentre una pieza, comenzando en ese momento su procesamiento.

A lo largo del siguiente apartado se describen todas las partes y cada uno de los componentes de la solución propuesta para resolver el problema planteado, abarcando los aspectos relativos a las partes software y de visión artificial de que se compone holísticamente el proyecto y comentando explayadamente las necesarias, pero a continuación se introduce una explicación muy resumida del mismo. Como se ve en la siguiente figura, se ha decidido diseñar e implementar un sistema que actúa como caja negra y toma como entradas imágenes de referencia (piezas bien orientadas) e imágenes "actuales" (piezas monitorizadas). Dichas imágenes pueden venir de distintas fuentes, como un simulador, una cámara real o grabaciones albergadas en cualquier espacio de almacenamiento; todas las interfaces de comunicación e información y el software necesario para compatibilizar los frames con los algoritmos y sensores se han realizado debidamente. Tras leer las entradas, el sistema ejecuta distintos algoritmos de preprocesamiento, detección de movimiento, extracción y comparación de características, reconocimiento de objetos y otras técnicas de visión artificial (analítica) en general que sirven, en última instancia, para componer una decisión que el sistema lanza como

respuesta a la cuestión de si la pieza actual tiene una pose correcta o incorrecta, en función de las referencias proporcionadas.

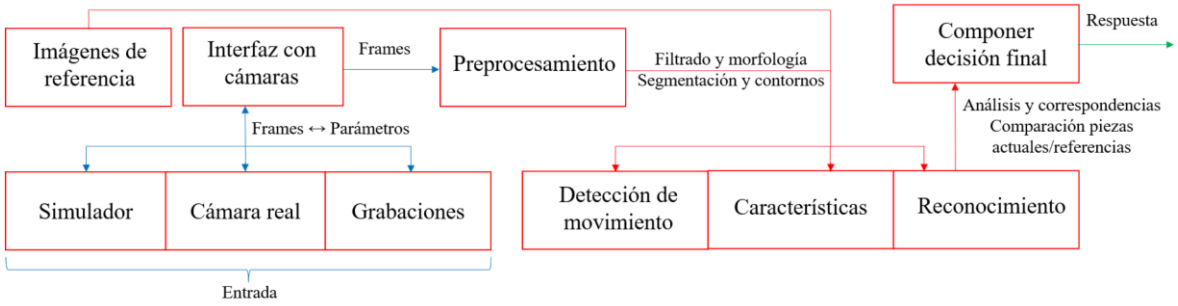


Fig. 3. Esquema general de la estrategia seguida para el diseño general del sistema.

En cuanto al algoritmo implementado, se puede realizar varias apreciaciones que lo sitúan un poco más en contexto (ver la siguiente figura). El algoritmo descrito se centra en el procesamiento de imágenes provenientes de sensores en un entorno industrial, específicamente en una cinta transportadora; se hace un uso eminente de C++ y la librería OpenCV para desarrollar el código fuente. El núcleo del algoritmo se encarga de adquirir frames de sensores de manera continua y transparente. Un debido preprocesamiento mejora la calidad de las imágenes al reducir ruido e impurezas, especialmente notado en simulaciones. La detección de una pieza inicia al identificar su entrada en la cinta. Un agente denominado “clasificador morfológico” evalúa la pieza mediante diversas métricas. Luego, se recurre a un segundo agente denominado “clasificador característico” para comparar la pieza con imágenes de referencia, empleando técnicas de detección, descripción y correspondencia de características basándose en puntos de interés en tiempo real y afrontando distintas invarianzas, buscando un equilibrio entre eficiencia y precisión. El algoritmo toma todos los resultados del análisis de ambos agentes y evalúa la similitud entre la pieza actual y las referencias, proporcionando una respuesta robusta. La información resultante se utiliza para tomar decisiones, informar al usuario y, en caso de detectar fallos imprevistos, dar una respuesta prematura. Se consigue un algoritmo sumamente modular, utilizando diferentes técnicas de procesamiento de imágenes y clasificación basadas en análisis funcional y de señales multidimensionales. El flujo de trabajo incluye la extracción de frames, preprocesamiento, detección de movimiento, reconocimiento de piezas, identificación de puntos clave, correspondencia de keypoints y cálculo de la pose final, empleando diversas métricas avanzadas y algoritmos para lograr una validación precisa de piezas en este tipo de entornos. Más adelante, en el apartado que detalla y explica en profundidad todo el algoritmo, se puede consultar la figura Fig. 57. Esquema general del sistema realizado y algoritmo implementado., que resume gráfica y prácticamente en su totalidad.

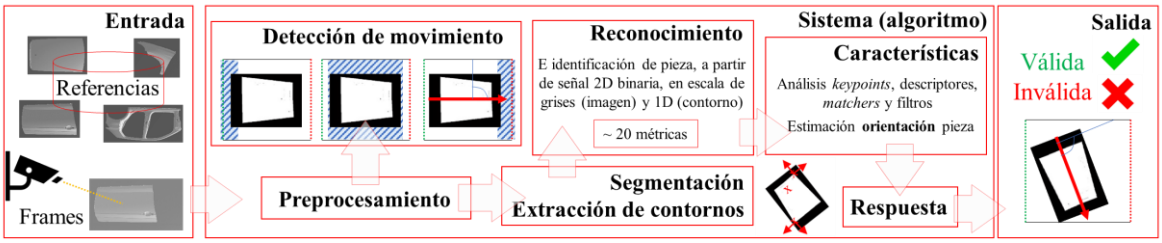


Fig. 4. Diagrama (muy) general que resume el algoritmo propuesto.

Son varias las diferentes librerías, herramientas… y los dispositivos, equipos… utilizados en la solución final propuesta. En el apartado "Sistema implementado" se introduce cada uno con mucho mayor detalle pero, entre otras, cabe mencionar a: Qt Creator 5.15 LTS, la especificación estándar OpenGL 3.3 en modo de compatibilidad (con esta se crea un interfaz para el simulador de modo que sirva de nexo entre la capa programática y los dispositivos de hardware gráfico; aunque la versión de OpenGL utilizada no es la más moderno, está ajustada a las capacidades de la máquina virtual empleada para la programación del simulador y cumple, convenientemente a las primeras pruebas que se fueron haciendo, con las funcionalidades que se requiere, además de la migración a un código más actualizado es sencilla y está abierta como trabajo futuro), la librería de solo encabezado GLM ("OpenGL Mathematics") ("glm.hpp") y varias de sus extensiones ("gtc/matrix_transform.hpp", "gtc/type_ptr.hpp") para el manejo de operaciones matemáticas en el contexto de OpenGL, la librería "STB Image" para la lectura y manipulación de texturas y la librería Assimp ("Open Asset Import Library") para la importación de modelos tridimensionales (con estas últimas se crea un motor de videojuegos gráfico para programar el sistema simulador mediante OpenGL en su paradigma de programación moderno).

En resumidas cuentas, parte del trabajo realizado se corresponde un proyecto de Qt en C++ que lanza un simulador de escenas tridimensionales personalizables mediante OpenGL con el que se pueden capturar frames y guardarlos a voluntad en un directorio. Asimismo, se realiza una librería para lanzar una interfaz con una cámara real para poder exportar los frames y procesarlos, además de otra librería relativa a una interfaz con la que se capturan y procesar frames provenientes del simulador, otra para vídeos locales y una última para aunar los tres tipos de fuente de frames en un solo módulo compacto con el que se pueda elegir de qué sensor capturarlos, procesarlos a posteriori y guardarlos (desde el simulador, la realidad o leerlos desde el sistema de archivos) (compuesta de un núcleo y un plugin[4] de escaneo dedicado a capturar cíclicamente los frames continuamente dentro del core). Luego, en la misma máquina (comunicación local) o en otra separada (comunicación

[4] Se denomina "plugin" a un bloque de software que añade una característica o servicio a un sistema más general o a varios, de forma que solo haya que añadirlo al programa principal para explotar las funcionalidades que ofrezca. Si el bloque tuviera que ser actualizado o cambiado en el futuro, solo habría que eliminar el antiguo bloque e introducir el nuevo y, siempre y cuando tuvieran las mismas entradas y salidas, el sistema desempeñaría sus funciones de manera exacta.

remota) se puede ejecutar la aplicación anterior para analizar mediante el algoritmo de visión artificial desarrollado los fotogramas transferidos de las piezas moviéndose por la cinta transportadora, real o virtual, dentro de lo que conforma el segundo proyecto de Qt realizado para este trabajo.

2.- SISTEMA IMPLEMENTADO

En este apartado se explican con detalle todas las partes de que se compone el sistema que conforma el proyecto realizado, categorizando los apartados desde un mayor a menor nivel de especificación y poniendo especial énfasis en mostrar el trabajo realizado de manera cualitativa aunque no poco literal sino mostrando muchas comparaciones con el código fuente implementado para seguir una línea común entre el entendimiento de la teoría empleada y los algoritmos aplicados en la práctica, además de tratando de dibujar meridianamente definida y en detalle la estructuración del software y el diseño del sistema en lo referente a la programación en C++ dedicada a cada parte integrante del proyecto.

En la siguiente figura se puede ver, en una panorámica muy distante, un sumario gráfico de mediano nivel de detalle que compendia las tres grandes partes que conforman el sistema implementado y todo el trabajo realizado. Todo lo relativo al simulador, a las interfaces, al algoritmo y a cada una de las figuras de que se compone la siguiente se detalla, explica y fragmenta con mucha más claridad a lo largo de todo este apartado.

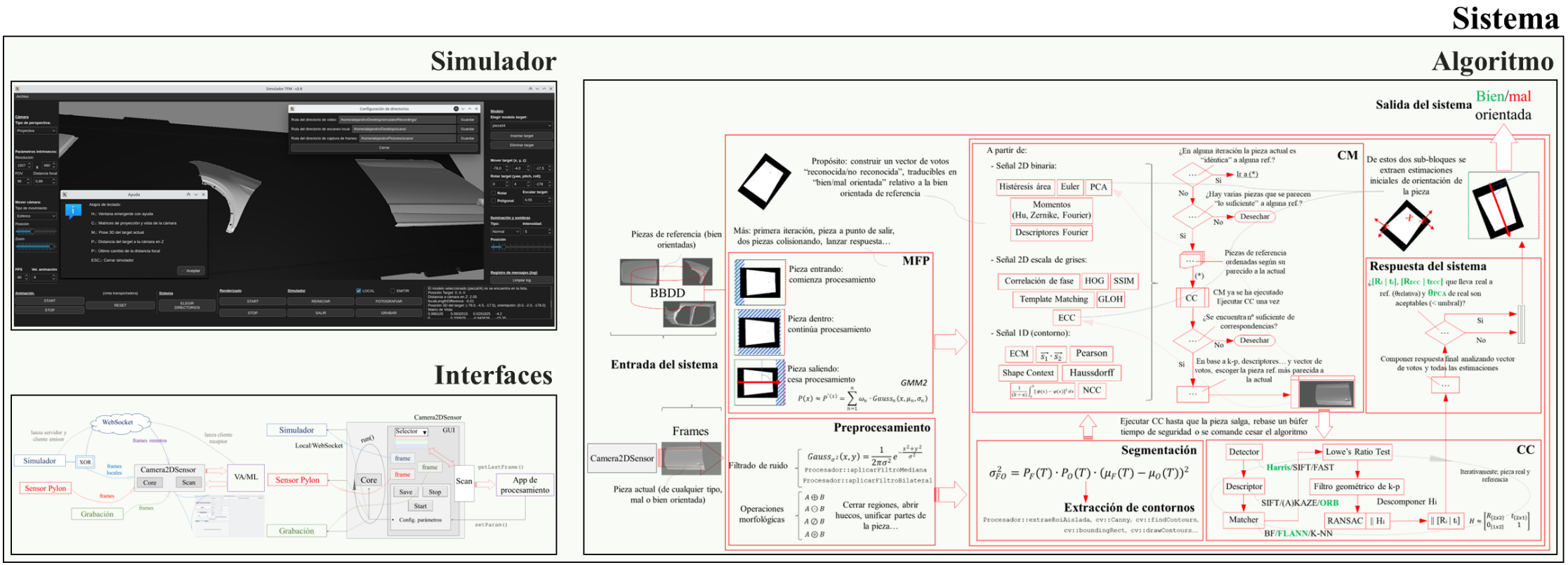


Fig. 5. Sistema diseñado e implementado en una panorámica general.

2.1.- Simulador

El simulador está realizado siguiendo el paradigma de programación orientada a objetos, enmascarándolo en un espacio de nombres global y subdividiéndolo en un conjunto de clases que brindan funcionalidades de manera modular, encapsulada y organizada en forma de interfaz, logrando implementar mediante una metodología estricta de código C++ mecanismos de construcción y destrucción de objetos, sobrecarga, herencia, polimorfismo, composición, virtualidad y otros paradigmas de la POO y la estructuración del código está realizada de forma que el sistema de simulación pueda ser utilizado a posteriori por diversos programas, distintas máquinas de manera local o remota o añadido a librerías existentes sin necesidad de agregar código fuente adicional.

El simulador desarrollado constituye una potente herramienta para la emulación (3D) de espacios industriales y sirve para experimentar y dar apoyo a los algoritmos de visión artificial que se desarrollan a lo largo del trabajo, o a futuras técnicas de aprendizaje automático las cuales fuera plausible estudiar. Se realiza a modo de motor gráfico de videojuegos para proporcionar todas las herramientas de renderizado, modelado de escenarios y dinamismo necesario para simular una línea de producción con múltiples piezas buscando una calidad meridiana tanto en fluidez de animación como en resolución; la programación se realiza empleando diferentes librerías de Qt, el estándar OpenGL bajo C++ y el lenguaje GLSL.

En la siguiente figura se puede ver en cierto nivel de detalle el simulador en pleno funcionamiento y las distintas zonas de que se compone, explicadas exhaustivamente a lo largo de este apartado:

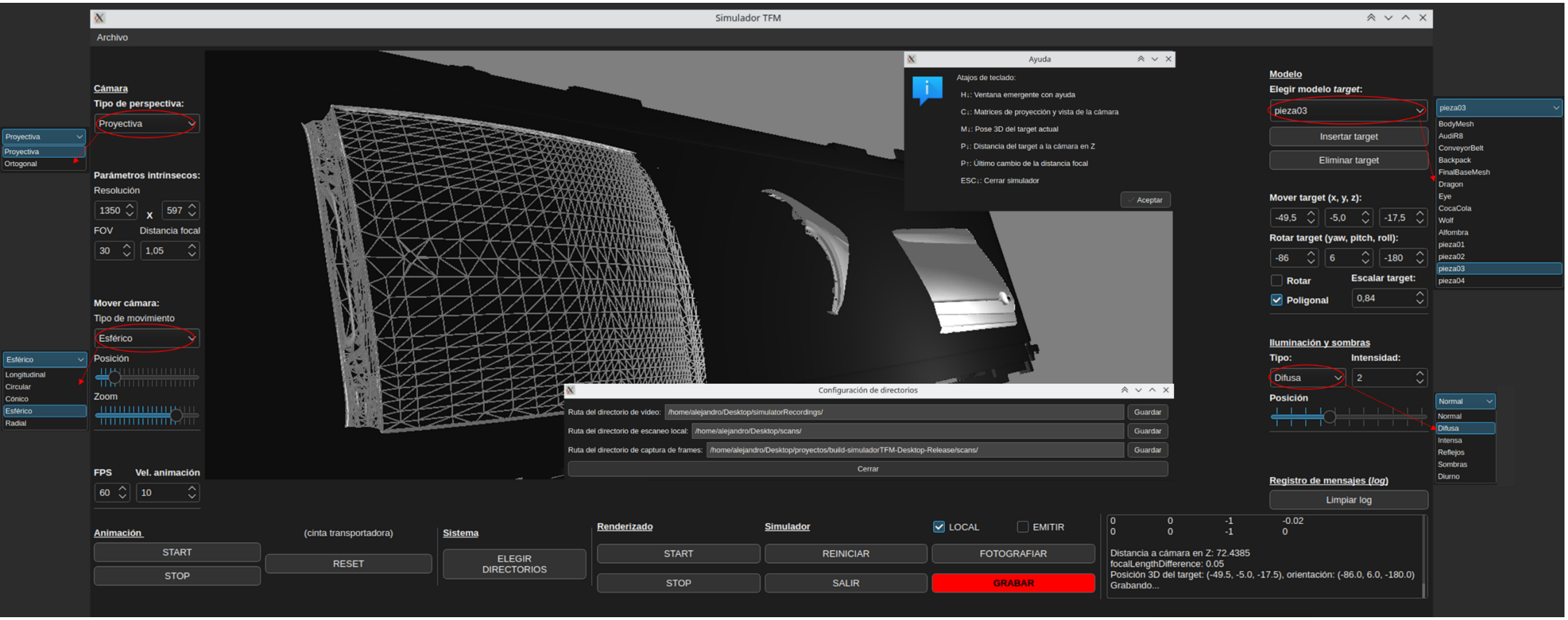


Fig. 6. Simulador implementado en funcionamiento.

2.1.1.- Metodología

A lo largo de la investigación y el desarrollo de diferentes proyectos de prueba se exploraron varias alternativas con el propósito de adquirir un sólido conocimiento en programación de simuladores gráficos y tecnologías afines. La experiencia acumulada en estos proyectos contribuyó a una toma de decisiones informada para la implementación del simulador final propuesto. A continuación se resumen varias alternativas otrora tomadas en consideración:

Se exploró la creación de simuladores utilizando Qt Quick y QML, además de la librería Qt 3D recientemente añadida a Qt 6, pero se abandonó debido a complicaciones en la fusión de las librerías de Qt Quick 3D bajo Qt 5, que es la versión que se debe utilizar. Qt Quick es la librería estándar de Qt (cualquier versión estable) para programar aplicaciones en QML, que es un lenguaje de Qt basado en JavaScript. Qt Quick 2 (una mejora) introduce un "scene graph" para un mejor renderizado, basado en OpenGL (ES) 2.0. Qt Quick 3D, lanzado en versiones 5 y 6, es una extensión de Qt Quick 2 para tres dimensiones; tiene la merced de evitar usar motores externos que causan problemas de sincronización con Qt y que añaden capas adicionales de abstracción. También se puede fusionar contenido 2D con 3D en escenas 3D uniendo Qt Quick 2 y Qt Quick 3D. A lo sumo, Qt 3D es un módulo (hay un poco en la versión 5 y está completo en la versión 6) que compendia todas las funcionalidades de renderizado 2D y 3D de Qt para simulaciones "casi" en tiempo real, tanto para C++ como para QML. Estrechamente relacionada con Qt 6, también se exploró la integración de Kuesa 3D, un motor de KDAB Group basado en Vulkan y C++, que tiene facilidades como la carga de meshes desde archivos de recurso .qrc. A pesar de no haber alcanzado una solución funcional (por razones similares a Qt 3D[5]), esta etapa proporcionó información valiosa sobre las capacidades de estas tecnologías [28-29].

Se programó un ejemplo simple a modo de introducción a Ogre3D en Qt 5. Se logró mostrar un modelo 3D en una ventana utilizando OpenGL y se implementaron eventos de interacción con el usuario. Concretamente, se investigó mediante el componente opcional de la librería "Bites", que está relativamente actualizada y compatible con la versión de Qt empleada. La librería proporciona la clase `OgreBites::ApplicationContextQt`, que usa `QWindow` para dar otorgar al programados varias facilidades, para usar (aunque en CMake, más general que qmake) las funciones de Ogre3D dentro el proyecto Qt de C++. También son relevantes clases como SdkQtCameraMan y otras librerías externas como QmlOgre, que integra el Ogre3D 1.8 en escenas Qt QML en versionas más antiguas del IDE. En

[5] Para aclarar: Qt 5 dispone de su librería Qt 3D con «algunas» funcionalidades 3D, que se pueden juntar con QML o C++; Qt 5.15 LTS tiene algunas funcionalidades de Qt Quick 3D que extiende QML a usar técnicas 3D más fácilmente. Por otro lado, Qt 6 dispone Qt Quick 3D de base, ya completamente desarrollado.

paralelo, se experimentó con diferentes herramientas de compilación, incluyendo CMake y qmake, con conclusiones similares [31-32].

Irrchlit es otro motor de videojuegos considerado; en su momento pareció la opción más desactualizada (la última versión estable es de 2016). Sus funcionalidades son más restringidas a videojuegos que Ogre (por tanto puede dar más problemas al importar formatos CAD y manejarlos) (lo que más importa en este proyecto es un renderizado realista a la par que eficiente y control total de la cámara) y no aporta muchas mejoras, además no tiene wiki oficial, lo que constituye más contras que pros.

OpenCascade es otra alternativa estudiada, que está más enfocada a las funcionalidades CAD avanzadas y al uso de UI con este propósito, al contrario de lo que se desea para el simulador, que es (más bien) un renderizado continuo de escenas dinámicas y la realización de animaciones, aunque ya hay vínculos con Qt 5 establecidos (incluso en las librerías de la propia empresa) y pudo haberse optado por este camino, que dispone de alternativas relacionadas de código libre [33-34].

Se consideró utilizar OpenGL, Assimp para trabajar con modelos 3D a través de la clase QOpenGLWindow. Se logró permitir al usuario cambiar la pose 6D de un objeto mediante controles en interfaz y el ratón. Esta clase se considera obsoleta según la documentación de Qt y, por lo tanto, se decidió abandonar la idea.

Se ha decidido, tras experimentar con todas las opciones comentadas y otros motores de renderizado, emplear C++, OpenGL y programas de sombreado GLSL (un estándar de programación a bajo nivel de aplicaciones gráficas interactuando con la GPU) para realizar desde cero el simulador, consiguiendo resultados muy complacientes en todos los sentidos (emulación de una cámara y un entorno virtual que incluya la modificación de la distancia focal, el campo de visión, la iluminación, sombras y reflejos, modelos de mallas complejos, sus texturas y la manipulación de su pose, entre otros). Las razones que llevaron a esta decisión son, entre otras, la falta de flexibilidad de los motores de renderizado comerciales a la hora de programar implementaciones a bajo nivel (como la emulación del círculo de confusión), la transparencia que proporcionan al programador (se considera deseable evitarla si el propósito es aprender lo máximo posible estas técnicas) y la pobreza en la documentación sobre la integración de sus librerías en el entorno de desarrollo Qt 5 (por lo menos en sus versiones no obsoletas más modernas).

Otros estándares de programación para GPU que se pudieron haber considerado son:

- DirectX: Formado por un conjunto de múltiples API (Direct3D para procesar y programar gráficos tridimensionales, Direct Graphics para renderizar imágenes…) desarrolladas por Microsoft para el desarrollo de aplicaciones multimedia y juegos en plataformas Windows. De la misma forma que OpenGL, es recomendable requerir de vastos conocimientos en lenguaje C++ para usarlo, y como por necesidad del proyecto el autor se formaría en dicho menester, se decidió decantar por una alternativa “del estilo”.

- MATLAB: El software propietario de cálculo numérico científico e ingenieril por excelencia ofrece un IDE para el desarrollo de aplicaciones con GUI empleando su lenguaje propio "M". Aunque contiene varias toolbox para la simulación y modelización de entornos físicos virtuales y brinda la posibilidad de establecer lazos programáticos con software externo escrito en lenguajes de programación como C/C++ o Java, se optó por recurrir a otras alternativas que permitan una integración más directa, dedicada y flexible para obtener un conocimiento más útil y profundo sobre lo trabajado.

Y algunas librerías auxiliares y herramientas otrora estudiadas y consideradas:

- GLU: OpenGL *Utility Library*, consiste en una serie de funciones que utilizan la biblioteca[6] OpenGL base (y normalmente se distribuye junto con dicho paquete, menos en OpenGL ES [para sistemas embebidos]) para proporcionar rutinas de dibujo de nivel superior (mapeado entre coordenadas de pantalla y del mundo, generación de mipmaps[7] de texturas, teselado de primitivas poligonales, interpretación de los códigos de error de OpenGL, una gama ampliada de rutinas de transformación para configurar volúmenes de visualización y el posicionamiento de la cámara…) a partir de las más primitivas que OpenGL proporciona, facilitando su uso.
- GLEW: OpenGL *Extension Wrangler*, es una biblioteca que maneja extensiones de OpenGL, proporcionando una interfaz sencilla para acceder a las funciones y capacidades extendidas de OpenGL. Facilita al programador el uso de extensiones y versiones más recientes de esta especificación.
- GLUT: OpenGL *Utility Toolkit*, es una biblioteca que simplifica la creación de ventanas, gestión de eventos del sistema e interacción con el usuario en aplicaciones de OpenGL; puede facilitar enormemente el desarrollo de programas que sean independientes del sistema operativo.
- FreeGLUT: Es una versión mejorada y de código abierto de GLUT; proporciona sus mismas funcionalidades y además agrega algunas características adicionales, como compatibilidad multiplataforma, soporte para ventanas a pantalla completa o el manejo mejorado de menús y la gestión de eventos, entre otros.
- GLAD: es una librería de encabezado único que gestiona punteros a funciones de OpenGL para facilitar la tarea al programador al utilizar una versión u otra de la especificación; se puede descargar desde fuentes online un comprimido que maneje la versión de interés y alojarla como librería estática en "/usr/include", por ejemplo.

[6] Biblioteca: entiéndase, ya sea estática (si se une directamente al ejecutable con el programa en tiempo de compilación) o dinámica (si se carga y enlaza por separado en tiempo de ejecución), como un conjunto de código predefinido (variables, clases, métodos, plantillas…) que proporciona una funcionalidad reutilizable (más documentación, ayuda auxiliar… opcionales). Se suele independizar semánticamente del término "librería" pues esta última se refiere más bien a la alternativa "estática", cuyo código no se comparte entre más de un ejecutable en tiempo de ejecución.

[7] Los mipmaps son secuencias precalculadas y optimizadas de imágenes de texturas, cada una de las cuales es una representación progresivamente de menor resolución que la anterior (típicamente una octava menor) utilizadas para mejorar el rendimiento la calidad de renderizado.

Como GLEW, las funcionalidades que ofrece esta librería son prácticamente las mismas que ofrecen librerías propias de Qt e incluirá redundantemente sería fútil si no incompatible.

- SOIL: *Simple* OpenGL *Image Library*, es una biblioteca que facilita la carga y manipulación de imágenes en aplicaciones de OpenGL; proporciona funciones para cargar texturas desde archivos de imagen y realizar operaciones básicas de manipulación de imágenes.
- STB: Con un nombre que hace honor a su creador (Sean T. Barrett), es una colección de librerías de una sola cabecera escritas en lenguaje C dedicadas a variados propósitos relativos al desarrollo de aplicaciones de OpenGL. La más comúnmente utilizada, y la que se decide emplear en este proyecto, es STB Image, que proporciona funciones para cargar imágenes en varios formato (se emplea para cargar las texturas de los modelos en la escena del simulador).

Se decidió no implementar el código bajo las anteriores librerías (salvo STB), aunque se estuvo varias semanas trabajando con ellas, a causa de que el esfuerzo en compatibilizar el marco de trabajo que estas y las propias clases de Qt más el manejo directo de las primitivas de bajo nivel de OpenGL ofrecían estaba escalando demasiado y, por practicidad, se vio más sensato utilizar una sola de las alternativas para realizar todo el simulador, de suerte que la solución de Qt es más acertada para proyectos creados dentro de su IDE. Además, la recomendación más moderna de programación con OpenGL desaconseja el uso de la mayoría de tales librerías, que quedaron muchas obsoletas con el lanzamiento de OpenGL 3.1 en 2009.

En otro orden de cosas, a continuación se van a tratar superficialmente tres conceptos de particular interés con el simulador realizado, que son las GPGPU, el estándar OpenGL y la librería GLM.

Las GPU (Unidades de Procesamiento Gráfico) tienen ventajas adicionales sobre las CPU (Unidades de Procesamiento Central); estas incluyen contar con más unidades de cálculo y un ancho de banda más alto para recuperar datos desde la memoria. Además, en muchas aplicaciones de aprendizaje profundo, procesamiento de imagen o renderizado (en las cuales es común hacer muchas operaciones matriciales en poco tiempo) que requieren un gran esfuerzo computacional, se pueden aprovechar las capacidades específicas de las GPU para acelerar aún más los cálculos.

La Computación en Unidades de Procesamiento Gráfico de Propósito General (GPGPU) reconoce la tendencia de emplear la tecnología de GPU para aplicaciones no gráficas. Hasta 2006, los estándares de API gráfica como OpenGL y DirectX eran las únicas formas de programar con GPU. Cualquier intento de realizar cálculos arbitrarios en la GPU estaba sujeto a las restricciones de programación de esas API [35, pág 384].

Las GPU estaban diseñadas para producir un color para cada píxel en la pantalla mediante unidades aritméticas programables llamadas “sombreadores” (*shaders*) de píxeles. Con el tiempo los programadores se dieron cuenta de que si las entradas eran datos

numéricos con un significado diferente al de los colores de los píxeles, podrían programar el sombreador de píxeles para realizar cálculos arbitrarios.

Sin embargo, existían limitaciones de memoria porque los programas solo podían recibir un puñado de unidades de color y textura como datos de entrada. Era casi imposible predecir cómo manejaría la GPU los datos en coma flotante (si es que podía procesarlos), por lo que muchas aplicaciones científicas no podían utilizar la GPU. Aquellos que deseaban resolver problemas numéricos tenían que aprender OpenGL o DirectX, las únicas formas de comunicarse con la GPU.

En 2006, NVIDIA presentó la GeForce 8800GTX, la primera GPU compatible con DirectX 10. Fue también la primera GPU en utilizar la arquitectura CUDA; esta arquitectura incluía varios componentes nuevos diseñados específicamente para la computación mediante GPU y pretendía eliminar las limitaciones que les impedían que las anteriores GPU se utilizaran para cálculos no gráficos. De hecho, las unidades de ejecución de la GPU podían leer y escribir memoria arbitraria, así como acceder a una caché mantenida por software denominada memoria compartida. Estas características arquitectónicas se añadieron para crear una GPU CUDA que también destacara tanto en cálculos de propósito general como en tareas gráficas tradicionales.

La división del espacio entre los distintos componentes es distinta en una GPU y una CPU; la primera dedica más transistores al procesamiento de datos, es un procesador altamente paralelo, multihilo y de muchos núcleos. En la GPU casi todo el espacio de semiconductores está dedicado a la ALU, lo que la hace idónea para cálculos repetitivos sobre grandes volúmenes de datos. La GPU accede a una memoria local y está conectada al sistema, es decir, a la CPU a través de un bus (actualmente, el *Peripheral Component Interconnect Express* [PCI Express]).

El nuevo, en aquel entonces, "modelo de programación de GPU" introdujo conceptos clave para comprender un nuevo paradigma de CUDA; la primera distinción es entre "host" (anfitrión) y "device" (dispositivo). El código ejecutado en el lado del anfitrión es la parte del código que se ejecuta en la CPU, incluyendo la RAM y el disco duro. En cambio, el código ejecutado en el dispositivo se carga automáticamente en la tarjeta gráfica y se ejecuta en esta. Otro concepto importante es el "kernel", que representa una función realizada en el dispositivo y lanzada desde el anfitrión. El código definido en el kernel se ejecutará en paralelo por una matriz de hilos.

En otro orden de ideas, la librería de gráficos libre (del acrónimo OpenGL) es una especificación estándar históricamente extendida que define una API (Interfaz de Programación de Aplicaciones) multilenguaje y multiplataforma para realizar aplicaciones que puedan renderizar gráficos 2D y 3D. OpenGL proporciona acceso directo a las capacidades gráficas de la máquina (v.g. su GPU) y ofrece un alto grado de control y personalización a través de las más de 250 funciones que proporciona para todo tipo de labores partiendo del manejo a un nivel fundamental mediante primitivas geométricas simples y control a bajo nivel de la aritmética del cómputo, permitiendo alcanzar

rendimientos considerablemente más óptimos que los que otras herramientas menos transparentes proporcionarían.

Al ser multiplataforma, OpenGL es compatible con gran variedad de sistemas operativos, incluyendo Windows, macOS y Linux; esto es beneficioso para el presente proyecto pues los computadores utilizados frecuentan servir bajo distribuciones de tipo Unix como Debian GNU/Linux (Ubuntu, Kubuntu, Pop!_OS…). Además, permite la realización de aplicaciones orientadas a diversas plataformas y exportables, ya sea para videojuegos en dispositivos móviles Android, iOS o computadores, escenarios de realidad virtual o aplicaciones médicas o de visualización de información, entre otras muchas posibilidades.

Cabe decir, basándose en la experiencia del autor de este trabajo, que existe un riesgo palpable para el profano que comience a programar con OpenGL y no investigue durante varios días sobre el quehacer típico y la historia de este gremio. En esta disposición se vio absorto el autor del proyecto que, no sin ser consciente de que lo que hacía durante las primeras iteraciones y etapas del trabajo era llevado a cabo en pos de la experiencia heurística y la adquisición de conocimientos de base sólida y necesarios, comenzó a notar llegado a cierto punto el sentido y significado más crudo de la palabra “obsoleto”, a decir, en los momentos en que los paquetes, las librerías y controladores que se antojaba necesario instalar ya no eran compatibles con el sistema utilizado.

Desde sus inicios y durante muchos años, los proyectos de OpenGL seguían una metodología de trabajo muy concreta y secuencial denominada *fixed pipeline* (flujo de trabajo fijo). A partir, aproximadamente de la revolución de las GPU a mediados de los 2000, una nueva versión “moderna” de utilizar OpenGL se abrió camino como *programmable pipeline*, la cual puede ser (de cierto modo) más compleja de programar pero otorga muchas más, y más poderosas funcionalidades a los programas.

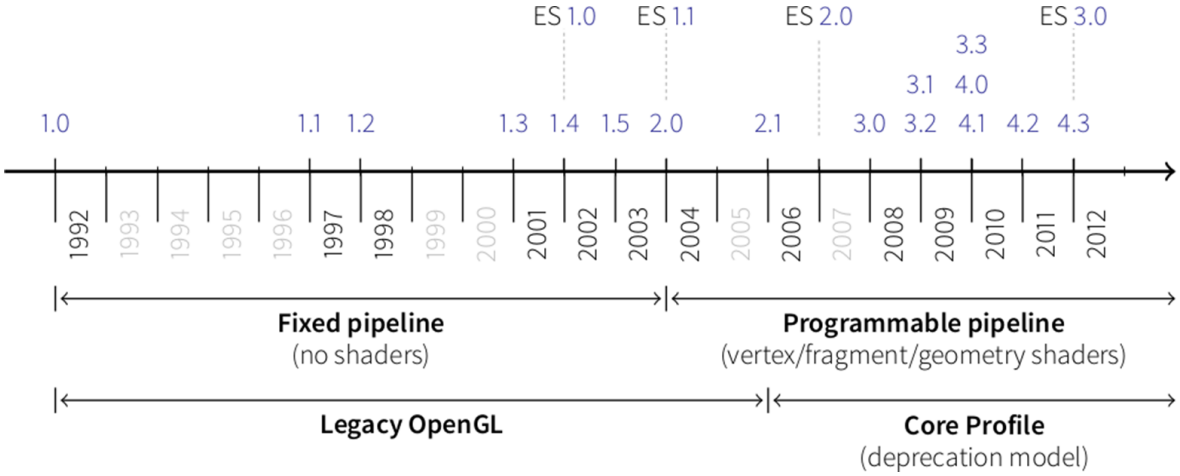


Fig. 7. Contraste entre los dos paradigmas en la programación con OpenGL. Recogido de la web: https://glumpy.github.io/modern-gl.html.

A lo largo de las primeras etapas del proyecto se trabajó con ambos enfoques, pero la alternativa preferida y finalmente implementada fue la moderna.

El flujo de trabajo fijo (o fixed pipeline) se refiere a la arquitectura clásica de procesamiento gráfico en OpenGL, en la que las etapas procesamiento gráfico están predeterminadas y tienen funcionalidades fijas. En este enfoque, la tarjeta gráfica sigue un conjunto predefinido de operaciones para renderizar gráficos en la pantalla. Las etapas típicas incluyen transformación e iluminación fijas y la programación del comportamiento gráfico se limita a configuraciones específicas proporcionadas por OpenGL. Era el enfoque común en versiones antiguas de OpenGL y actualmente no está recomendado para código nuevo, pues aunque pueda ser más fácil de aprender y sirva eficazmente para tareas básicas, no es adecuado para aplicaciones gráficas modernas y complejas y otorga menos flexibilidad, adaptabilidad o control al programador sobre las etapas de procesamiento.

El flujo de trabajo programable (o programmable pipeline) es una evolución moderna de OpenGL que proporciona mayor flexibilidad y control al programador sobre las etapas de procesamiento. En lugar de seguir un conjunto fijo de etapas o definiciones, este enfoque permite al desarrollador escribir sus propios programa, denominado “shaders” (de vértices, de fragmentos, texturas y otros, dependiendo de las necesidades de la aplicación) para realizar cálculos específicos en el hardware gráfico y, así, incluir transformaciones e iluminaciones con un grado de personalización notoriamente mayor. Por estos motivos se consideró mucho más conveniente proseguir con este enfoque para todo el trabajo realizado.

Dejando esto a un lado, OpenGL Mathematics (GLM) [60] es una biblioteca de matemáticas orientada a la computación gráfica (renderizado de escenas, raytracing[8] y rasterización[9], procesamiento de modelos 3D, simulaciones físicas…) y escrita en C++, independiente de plataformas y sin dependencias externas, que se utiliza exclusivamente a través de archivos de encabezado o cabeceras y está diseñada para aplicaciones gráficas basadas en las especificaciones del Lenguaje de Sombreado de OpenGL (GLSL).

GLM proporciona clases y funciones que siguen las mismas convenciones de nomenclatura y funcionalidad que GLSL, lo que significa que la misma filosofía e intuición de programación aplicada bajo el lenguaje GLSL puede utilizarse con GLM en C++ de manera semejable.

Un sistema de extensiones de GLSL ofrece a la biblioteca capacidades adicionales, como transformaciones de matrices, cuaterniones, empaquetamiento de datos, generación de números aleatorios o generación de ruido, entre otros. Funciona perfectamente con OpenGL y garantiza la interoperabilidad con otras bibliotecas y SDK de terceros.

[8] Raytracing: es una técnica de renderización que busca simular de manera realista los caminos de los rayos de luz a medida que interactúan con elementos tridimensionales, teniendo en cuenta fenómenos como reflejos, refracciones, sombras e iluminación indirecta, buscando una alta fidelidad visual con el mundo real.

[9] Rasterización: hace referencia al proceso de convertir una representación de imagen descrita en formato de gráficos vectoriales, compuesta por formas geométricas y coordenadas, en una imagen “ráster”, donde la información vectorial se convierte en una serie de píxeles, puntos o líneas que, al ser visualizados conjuntamente, recrean la imagen original; esto se traduce en un aumento de la eficiencia para traducir gráficos vectoriales a un formato adecuado para su representación visual inmediata.

GLM está escrita en C++98, pero puede aprovechar las características de C++11 (especificación de C++ utilizada en este proyecto) cuando son compatibles con el compilador, como es el caso porque se utiliza el compilador GNU C++ en su versión 11.4.0 para Kubuntu 22.04 LTS (`~$ g++ --version`).

2.1.2.- Implementación en C++

El simulador denominado "Simulador TFM" que forma parte de este proyecto es una aplicación de simulación tridimensional desarrollada en C++ utilizando la biblioteca Qt, especialmente diseñada para fines de investigación y desarrollo en el campo de la representación de escenarios virtuales. Este simulador se ha creado con el objetivo de proporcionar una herramienta versátil y personalizable para explorar y analizar la visualización de objetos tridimensionales en un entorno virtual. En este apartado se presentan aspectos clave de la implementación de este proyecto en C++ con Qt.

La configuración del proyecto es un componente primordial que establece las bases para la compilación y el enlace de la aplicación. El archivo de proyecto (.pro) contiene directivas que definen las dependencias del proyecto, el estándar de C++ a utilizar y la versión mínima de OpenGL requerida, entre otros. Además, se especifican los archivos fuente y de encabezado que componen el proyecto, así como otros recursos necesarios, como archivos de formularios y de sombreado (shaders). También se incluyen directivas para establecer dependencias con librerías externas a Qt, como Assimp, OpenCV o las de la propia empresa (a través de un archivo ".pri", que contiene configuraciones específicas para establecer variables de entorno, opciones de compilación, enlace y configuración de paquetes), lo que facilita la incorporación de funcionalidades adicionales.

Una de las características sobresalientes de este proyecto es su capacidad para gestionar el contexto OpenGL. Esto se logra mediante la configuración de opciones específicas, como la versión requerida de OpenGL, la creación de un formato de superficie (`QSurfaceFormat`) y la especificación de parámetros relacionados con el renderizado y la depuración. El uso de OpenGL es fundamental para lograr una representación tridimensional efectiva y una interacción fluida con los objetos en el simulador a través de la GUI.

También se incorpora el manejo personalizado de excepciones y mensajes de depuración (mediante el encabezado `<csignal>`). El manejador de excepciones captura y registra excepciones, lo que contribuye a mantener la estabilidad de la aplicación en situaciones imprevistas. Asimismo, el manejador de mensajes de depuración personalizados permite un seguimiento detallado de la ejecución del programa y facilita la identificación y corrección de errores (métodos `tfm::customMessageHandler` y `tfm::crashHandler`).

La función principal o el punto de entrada del programa se encuentra en el archivo "main.cpp". Antes de iniciar la aplicación Qt (`QApplication`), se registran (meta)tipos de datos personalizados (`qRegisterMetaType`) para que puedan ser utilizados en el sistema de señales y ranuras de Qt, como `cv::Mat` utilizado para matrices de OpenCV. Además, se carga una hoja de estilo personalizada para la interfaz de usuario a través del lenguaje QSS (*Qt Style Sheets*), que es un mecanismo de Qt similar a CSS (*Cascading Style Sheets*) (compuesto de selectores y declaraciones) destinado a personalizar la apariencia de

interfaces gráficas orientado a desarrollo web. La función "main" es el punto de entrada del programa y es responsable de iniciar la aplicación y gestionar situaciones excepcionales.

Cabe mencionar que en una aplicación desarrollada en Qt es común que se creen clases derivadas de la superclase `QObject` (u otras subclases suyas) para aprovechar distintas utilidades del entorno de desarrollo, como los elementos de interfaz gráfica o la comunicación entre objetos mediante señales y ranuras (*signals* y *slots*) (una útil y desacoplada manera de implementar gestión de eventos desde y hacia objetos en Qt, similar al patrón Observador[10], los punteros a funciones [callbacks] o incluso las interrupciones). Tales clases suelen definirse en archivos de encabezado (.h) con macros declarativas de Qt, como `Q_OBJECT`, `Q_PROPERTY`, `signals`, `slots`, etc. Pues bien, antes de compilar la aplicación Qt ejecuta el MOC (*Meta-Object Compiler*) de los archivos de encabezado que contienen estas clases o macros de interés; el MOC las analiza y genera código C++ adicional que se almacena en archivos temporales (que se añadirían como "adenda" a los binarios finales junto con el resto de código fuente) (si se emplean este tipo de macros en archivos de código fuente (.cpp) se deben generar los .moc [*$ moc -o moc_file.cpp main.cpp*] de estos e incluirlos al final de los mismos). Las funcionalidades propias de Qt que proporciona el MOC a cada clase quedan registradas en los .moc generados y, junto con el código fuente original, se puede compilar todo para conformar el ejecutable final, tras enlazar con las librerías necesarias.

Por último, en lo relativo al apartado gráfico de la aplicación, cabe remarcar que "ui" es un puntero a una instancia de la clase `Ui::MainWindow`, que se usa como variable privada para acceder (idealmente solo desde la clase de la que es miembro) a los elementos de la interfaz de usuario que se definen en el archivo de interfaz de usuario generado automáticamente por Qt Designer y se incluye en el código con una línea como #include "ui_mainwindow.h".

2.1.2.1.- Camera

La clase `tfm::Camera` representa una cámara virtual con diversas funcionalidades para controlar y personalizar la vista de una escena 3D. Se encarga de gestionar la vista de una escena 3D en la escena de renderizado mediante eventos de interrupción por movimiento y pulsación del ratón y elementos de la interfaz. Proporciona métodos para controlar varios aspectos de la cámara, como su posición, orientación, distancia focal y tipo de proyección. Esto permite a los usuarios interactuar con la vista de la escena a voluntad desde diferentes ángulos y configuraciones.

La clase Camera contiene métodos para realizar acciones como el zoom, la restauración de la configuración inicial de la cámara, la aplicación de transformaciones de

[10] *Observer pattern*: Es un patrón de diseño de software que define una dependencia del tipo uno a muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes. Se trata de un patrón de comportamiento, por lo que está relacionado con algoritmos de funcionamiento y asignación de responsabilidades a clases y objetos.

vista y proyección, la manipulación de parámetros de la lente y otros aspectos relacionados con la visualización de la escena. Además, permite configurar diferentes modos de movimiento de la cámara, como el movimiento longitudinal, circular, cónico, esférico y radial, lo que brinda flexibilidad en la configuración de la escena. Con el botón izquierdo del ratón se puede mover a voluntad la vista de la escena en el espacio a lo largo y ancho del plano perpendicular a la dirección a la que apunte la cámara en ese instante y con el botón izquierdo se puede rotar en el espacio 3D, para lo que se aproxima el movimiento bidimensional vertical y horizontal del ratón en un mapeo, además, sobre la dimensión de profundidad especialmente ajustado para proporcionar un manejo amigable al usuario.

Integrada con varias estructuras y funciones de OpenGL y GLM para gestionar matrices de vista y proyección, la clase también ofrece la posibilidad de cambiar entre modos de proyección perspectiva y ortográfica, lo que afecta a la forma de la que se proyecta la escena en la cámara. Los métodos proporcionados permiten configurar parámetros específicos, como el campo de visión (FOV), la relación de aspecto, los planos cercano y lejano de los modos de proyección o la distancia focal. Esto es esencial para lograr diferentes efectos visuales y perspectivas en la escena. También realiza cálculos necesarios para actualizar la dirección de la cámara en función de su orientación y la posición del objetivo.

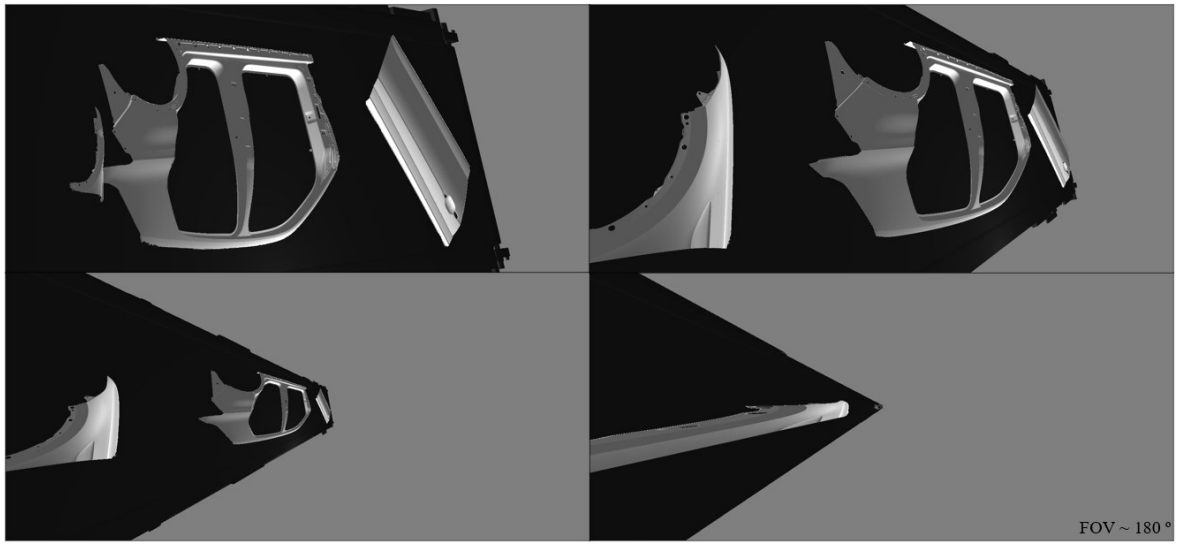


Fig. 8. Efecto de la variación del FOV en el simulador implementado. De izquierda a derecha y de arriba abajo el valor del ángulo de visión va creciendo, de 1 º a (casi) 180 º.

Para establecer la distancia focal de la cámara se emplea un sistema de unidades normalizado, adimensional, a fin de facilitar la programación. Se ajusta su valor dentro del rango [0,5, 1,5], donde un valor de 1,0 se corresponde con una distancia focal tal que la escena se corresponde el plano de enfoque y la cámara enfoca perfectamente todos los objetos; el usuario puede modificar este valor desde la interfaz variándolo de centésima en centésima.

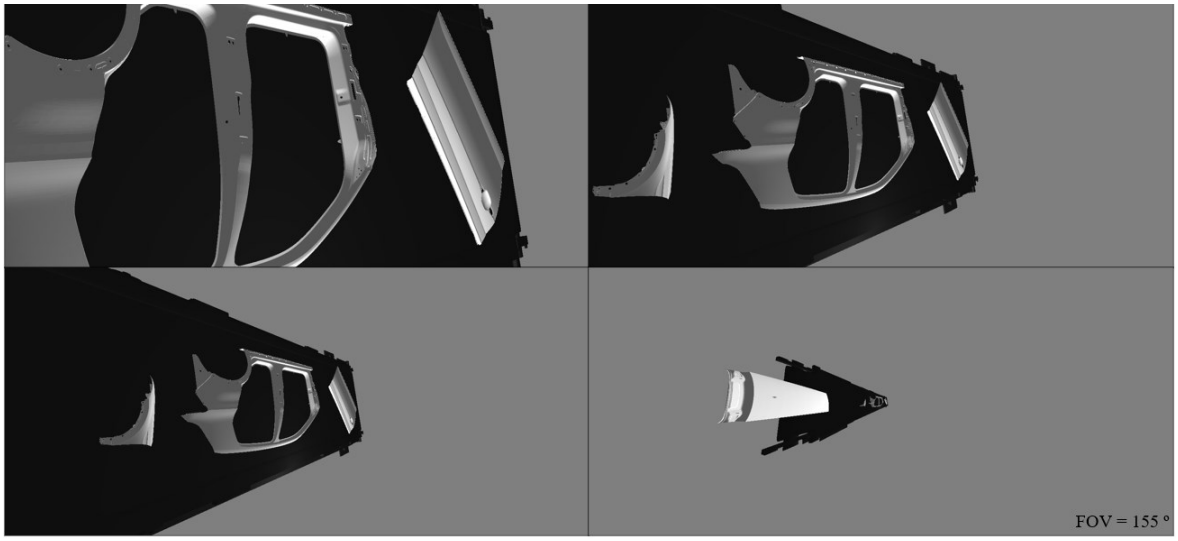


Fig. 9. Mismo caso que el de la figura anterior pero manteniendo la cámara estática durante la variación del FOV.

En el ámbito de la física óptica y fotografía, se tiene en cuenta que existe una relación inversamente proporcional entre la distancia focal y el ángulo de visión (también limitado, dentro del rango [1,0, 180]) dada mediante la siguiente expresión para una lente rectilínea delgada estándar (con poca o ninguna distorsión de barril):

$$FOV = 2\tan^{-1}\frac{x}{2f}, \tag{1}$$

donde x es la diagonal del tamaño del sensor de la cámara, FOV es el ángulo correspondiente al campo de visión en radianes y f es la distancia focal de la cámara, típicamente en mm. Se ve que medida que la distancia focal aumenta (manteniendo constante el tamaño del sensor), el FOV disminuirá, lo que dará como resultado un campo de visión más estrecho, una imagen con más "zoom" (manteniendo estática la cámara) y una deformación perspectiva más suave. Visto de otra forma, a distancias focales cercanas al infinito (cero grados de FOV), los rayos de luz son casi paralelos entre sí, por lo que el sujeto aparece "aplanado", mientras que a distancias focales pequeñas (mayor FOV), el sujeto aparece "escorzado". Como tamaño típico del sensor se escoge el del sensor APS-C, de alrededor de 22 mm (aunque esta implementación es meramente simbólica pues se podría haber emulado mediante código de manera más simplificada).

Programáticamente se mantiene un registro de los últimos cambios de esta valor para computar el gradiente (la tasa de cambio) de la distancia focal, a fin de que, aunque un valor superior a 1,0 que decrezca signifique una distancia focal "positiva" en coordenadas normalizadas, el método `Camera::setFocalLength` tenga en consideración que efectivamente el valor está decreciendo y se actualice el FOV correctamente. Más aún, se controla (`std::max` y `std::min`) que el FOV no varíe demasiado ni demasiado rápido (pues el rango de la función anterior está enormemente ampliado).

La simulación puede tener un modo de proyección perspectiva u ortogonal. La implementación de una deformación proyectiva de los objetos es muy relevante de cara a

ayudar a desarrollar un sistema real de calibración y corrección de perspectiva (con una matriz de calibración adecuada se pueden aplicar técnicas de corrección de perspectiva, como la rectificación de imágenes, para que las imágenes de la pieza en diferentes frames se muestren con la misma perspectiva o vista cenital), aunque este apartado no se va a tratar en este trabajo.

Mediante las funciones `glm::ortho` y `glm::perspective` se define en cada bucle de renderizado la matriz de proyección que se tratará en los programas de sombreado para transformar los vértices de los modelos iterativamente.

La siguiente ecuación matricial define una transformación ortográfica (o proyección ortogonal) aplicada a un punto del espacio en coordenadas homogéneas [36]:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ W' \end{bmatrix} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{-r-l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{-t-b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & \frac{-f-n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}, \tag{2}$$

donde los valores “r”, “l” (puntos más alejados hacia la derecha e izquierda en el sentido del eje x, respectivamente), “b”, “t” (puntos más alejados hacia abajo y arriba en el sentido del eje y, respectivamente), “n” y “f” (puntos a partir de los que “se ve” y “se deja de ver” en el sentido del eje z, respectivamente) definen un cuboide rectangular que especifica el área visible de la escena que es renderizada. Esta perspectiva mantiene los rayos de luz paralelos entre sí y a la normal del plano imagen pero pierde toda la información de profundidad; la distancia de la cámara virtual a un objeto no afecta al tamaño del objeto renderizado (en la vida real, los objetos que están más lejos de la cámara parecen más pequeños).

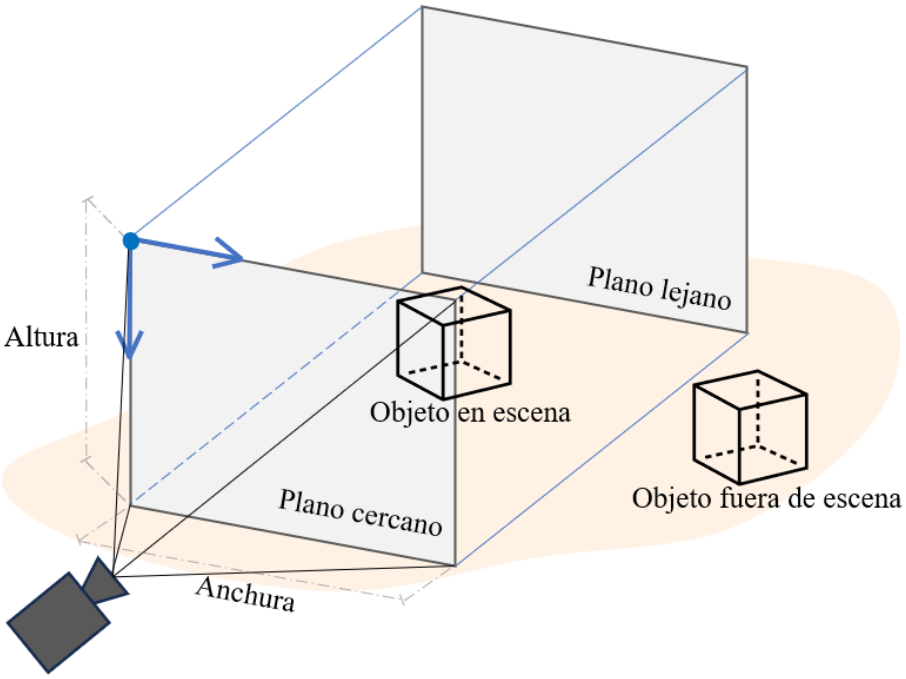


Fig. 10. Representación de la proyección ortográfica en OpenGL.

La siguiente ecuación matricial define una transformación proyectiva (o proyección perspectiva) aplicadas a un punto del espacio en coordenadas homogéneas [36]:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ W' \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & \frac{-n(r+l)}{r-l} \\ 0 & \frac{2n}{t-b} & 0 & \frac{-n(t+b)}{t-b} \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}, \tag{3}$$

donde los parámetros definen un "frustum", que es un objeto geométrico definido como la porción de un sólido situada entre dos planos paralelos que lo cortan (en el caso de una pirámide, las caras de la base son poligonales y las caras laterales son trapezoidales). "FOVy" es el ángulo entre los lados interior y superior del frustum, "a" es la relación de aspecto de la ventana de renderizado visible y "n" y "f" son las distancias al plano más cercano y lejano a partir de los que "se ve" y "no se ve", respectivamente. Esta transformación se asemeja más a la realidad porque mantiene el sentido de la profundidad, dado que los objetos alejados de la cámara parecen más pequeños y todas las líneas parecen proyectarse hacia puntos de fuga en los que se «cortan» las líneas paralelas.

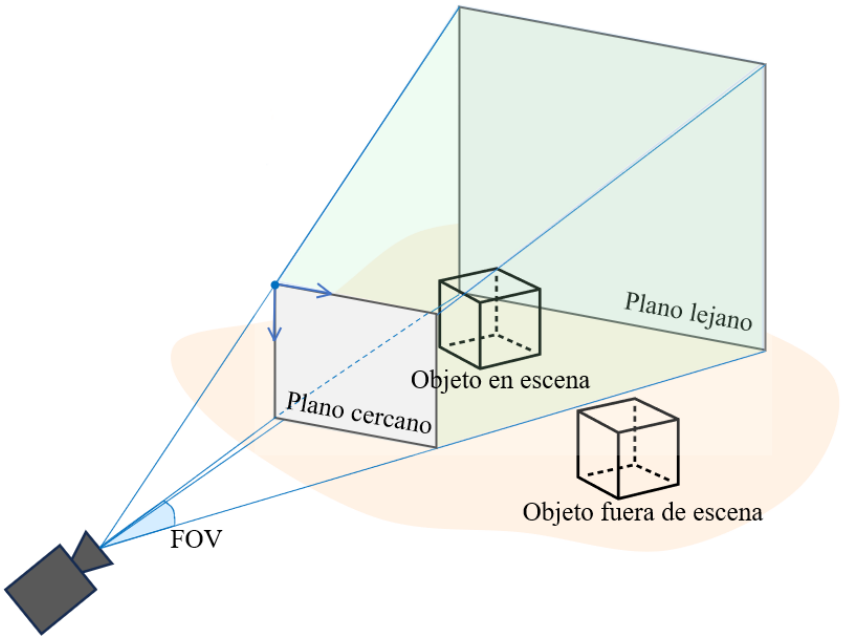


Fig. 11. Representación de la proyección perspectiva en OpenGL.

Se hace claro que un modelo de aprendizaje automático o de cualquier tipo debe ser robusto ante el efecto de la perspectiva y otras aberraciones que puedan ser causadas por la cámara; por esto se consideró tan interesante otorgar estas funcionalidades al simulador.

Las matrices de transformación codificadas en formato `glm::mat4` o `QMatrix4x4`, se transfieren de manera dinámica a los programas de sombreado, junto con otras variables como las posición de los vértices en coordenadas locales y del mundo, sus vectores normales, las coordenadas de la textura, la posición absoluta de la cámara y otros reales y booleanos (`useTexture`, `applyGaussianBlurFlag`, `focalLength`, `focalLengthDifference`…). La matriz de vista se utiliza para transformar las coordenadas en el espacio global del mundo a las coordenadas locales de la cámara (o vista); esta matriz se utiliza para que el mundo se desplace y rote de manera que la cámara esté en el origen del sistema de coordenadas en todo momento, resultando en que el observador visualice la escena desde el punto de vista de primera persona, consiguiendo que la cámara parezca estar en el centro de la ventana y mirando hacia la dirección negativa del eje z. Cuando se multiplica la posición del vértice por la matriz de vista en el shader de vértices, se coloca al objeto en una posición específica en relación con la cámara. Por otra parte, la matriz de proyección se utiliza para transformar las coordenadas de la cámara al “clipping space” o espacio de recorte (un sistema de coordenadas intermedio entre el mapeo de las coordenadas espaciales, las normalizadas en los programas de sombreado) y, finalmente, al espacio de pantalla (coordenadas de píxel finales en la ventana de renderizado); en el shader de vértices, cuando se multiplica la posición del vértice por la matriz de proyección, se lleva el objeto desde el espacio de cámara al espacio de recorte.

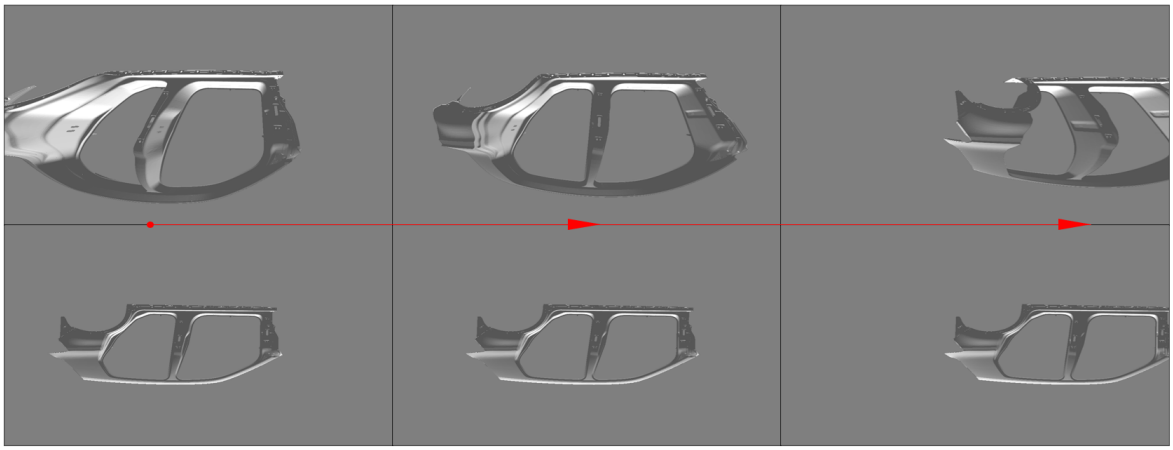


Fig. 12. Comparación entre una pieza en movimiento bajo proyección perspectiva (arriba) y ortográfica (abajo).

Luego de que el usuario modifique variables como la pose del modelo objetivo o la distancia de la cámara al mismo, su zoom, modelo de proyección… se deben actualizar las matrices de modelo, vista y proyección, que controlan de manera fundamental el sistema de coordenadas final que procesa el rasterizador y la posición de los vértices y el color de los píxeles que llegan a los programas de sombreado.

Por otra parte, se han implementado diferentes tipos de movimiento para la cámara con respecto a los modelos de la escena, los cuales el usuario puede seleccionar y controlar mediante la GUI. A continuación se explica brevemente la selección de movimientos escogida (movimiento longitudinal, circular, cónico, esférico y radial), que dota al simulador de una capacidad muy precisa de control sobre las diferentes casuísticas de experimentación con las piezas a analizar:

Fig. 13. Extracto de la GUI del simulador del selector y controlador del movimiento de la cámara.

A fin de unificar los movimientos de manera modular se implementa el método `Camera::configureMovement`, que devuelve directamente la traslación y rotación "adicional" que se necesita aplicar a la cámara, que apunta al centro del modelo sobre el

que se quiere mover, para que acabe en la posición deseada. De entre los argumentos que toma la función está el tipo de movimiento deseado (encapsulado en la enumeración `CameraMovementType`), las dos ternas de valores (x, y, z más yaw, pitch, roll) que contienen la información de la traslación y rotación elegida para mover la cámara a través de la GUI o programáticamente (`t` y `r`), la posición en coordenadas del mundo del modelo objetivo (`target`), la celeridad (`movementAmount`) y dirección (`direction`) de movimiento deseada (especificado mediante un `double` que proviene, p. ej., del gradiente de cambio de los `QSlider` de la interfaz para controlar la rapidez y dirección con que el usuario quiere que se mueva la cámara) y las sensibilidades a la traslación y rotación (`t_sensitivity` y `r_sensitivity`, respectivamente), que son dos parámetros empíricos (`float`) ajustados para lograr dinámicas más suaves y realistas.

En el movimiento longitudinal la cámara se mueve hacia adelante o atrás en línea recta apuntando al modelo objetivo. Se aplica la siguiente ecuación de movimiento para componer los vectores de retorno que contienen la orientación y posición adicional a aplicar al modelo:

$$\Delta x = \Delta y = \Delta\theta_x = \Delta\theta_y = \Delta\theta_z = 0 \tag{4}$$

$$distanceChange = movementAmount \cdot direction \cdot t_{sensitivity} \tag{5}$$

$$\Delta z = distanceChange, \tag{6}$$

donde “distanceChange” (o “angleChange” y otros similares, mostrados posteriormente) es la cantidad de movimiento o rotación que se debe aplicar a ciertos ejes cartesianos u otros. En este caso toda la cantidad de movimiento se aplica en la dirección del eje “z”, pues la cámara se mueve ese sentido, en una dirección u otra.

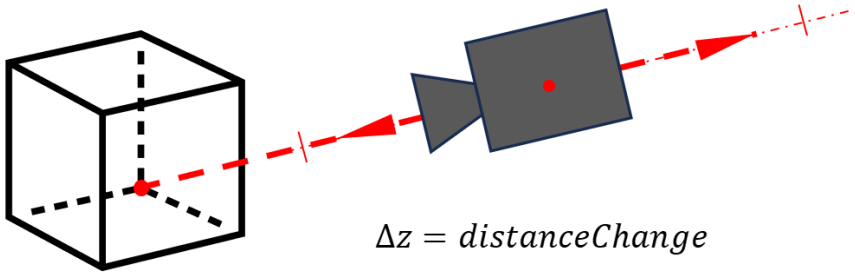


Fig. 14. Representación gráfica del movimiento longitudinal.

En el movimiento circular la cámara se mueve en una órbita circular paralela al plano horizontal apuntando al modelo objetivo. Se aplican las siguientes ecuaciones de movimiento para componer los vectores de retorno que contienen la orientación y posición adicional a aplicar al modelo:

$$\Delta x = \Delta y = \Delta z = \Delta\theta_x = \Delta\theta_y = \Delta\theta_z = 0 \tag{7}$$

$$angleChange = movementAmount \cdot direction \cdot r_{sensitivity} \quad (8)$$

$$yaw \equiv \Delta\theta_z = \tan^{-1}\left(\frac{t_z - target_z}{t_x - target_x}\right) \quad (9)$$

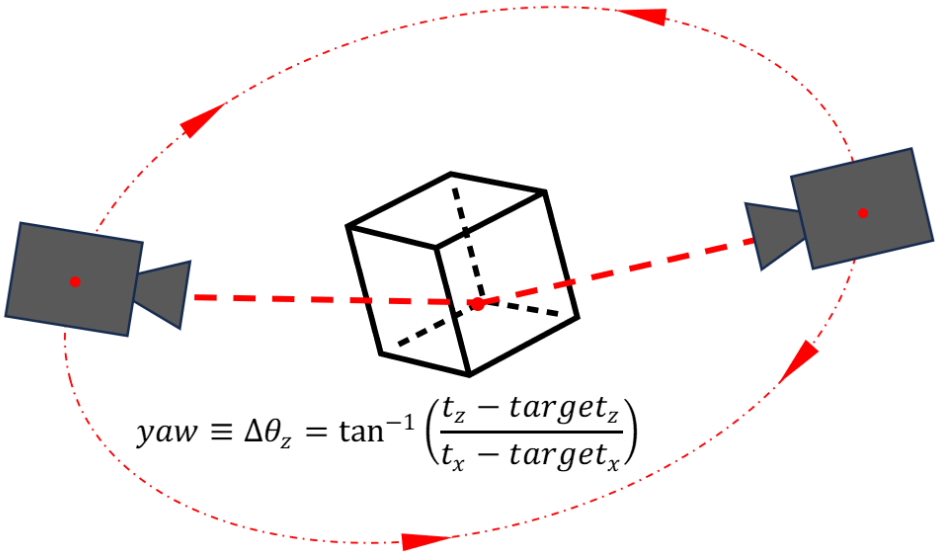


Fig. 15. Representación gráfica del movimiento circular.

En el movimiento cónico la cámara se mueve en una trayectoria cónica tangente a la dirección longitudinal apuntando al modelo objetivo. Se aplican las siguientes ecuaciones de movimiento para componer los vectores de retorno que contienen la orientación y posición adicional a aplicar:

$$radio = \frac{a}{t_{sensitivity}{}^{b}} \quad (10)$$

$$angleChange = \frac{movementAmount \cdot direction \cdot r_{sensitivity}}{a} \quad (11)$$

$$pitch \equiv \Delta\theta_x = \tan^{-1}\left(\frac{t_y - target_y}{t_x - target_x}\right) + angleChange \quad (12)$$

$$\text{selectedTranslation}_x = target_x + radio \cdot \cos(pitch) \quad (13)$$

$$\text{selectedTranslation}_y = target_y + radio \cdot \text{sen}(pitch)\,, \quad (14)$$

donde "radio" es la distancia entre el objetivo y la nueva posición de la cámara, "a" y "b" son dos parámetros empíricos (2,0 y 1,5, p. ej.) y la matriz de vista de la cámara se ajusta automáticamente al renderizar, acorde a la nueva posición de la misma y el objetivo al que apunta.

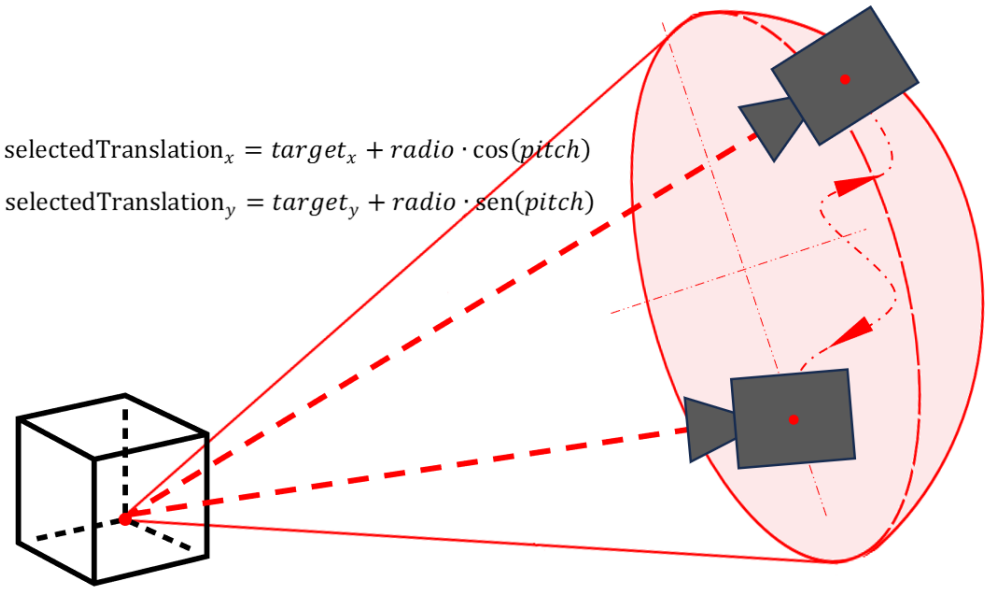


Fig. 16. Representación gráfica del movimiento cónico.

En el movimiento esférico la cámara se mueve en una trayectoria esférica alrededor del modelo objetivo. Se aplican las siguientes ecuaciones de movimiento para componer los vectores de retorno que contienen la orientación y posición adicional a aplicar:

$$radio = \frac{a}{t_{sensitivity}{}^{b}} \tag{15}$$

$$angleChange_x = angleChange_y = \frac{movementAmount \cdot direction \cdot r_{sensitivity}}{a} \tag{16}$$

$$pitch \equiv \Delta\theta_x = \tan^{-1}\left(\frac{t_y - target_y}{t_x - target_x}\right) + angleChange_x \tag{17}$$

$$yaw \equiv \Delta\theta_z = \tan^{-1}\left(\frac{t_z - target_z}{t_x - target_x}\right) + angleChange_y \tag{18}$$

$$\text{selectedTranslation}_x = target_x + radio \cdot \cos(yaw) \cdot \cos(pitch) \tag{19}$$

$$\text{selectedTranslation}_y = target_y + radio \cdot \text{sen}(yaw) \cdot \cos(yaw) \cdot \cos(pitch) \tag{20}$$

$$\text{selectedTranslation}_z = target_z + radio \cdot \text{sen}(pitch) \tag{21}$$

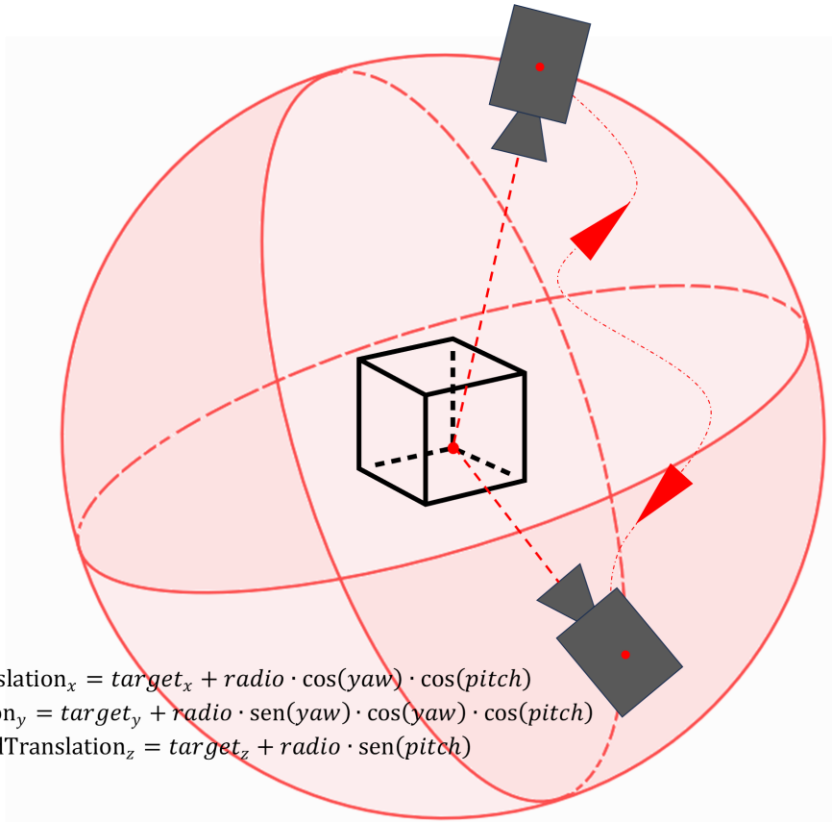


Fig. 17. Representación gráfica del movimiento esférico.

En el movimiento radial la cámara se mueve hacia la derecha o izquierda en línea recta apuntando al modelo objetivo. Se aplica la siguiente ecuación de movimiento para componer los vectores de retorno que contienen la orientación y posición adicional a aplicar:

$$\Delta y = \Delta z = \Delta\theta_x = \Delta\theta_y = \Delta\theta_z = 0 \tag{22}$$

$$distanceChange = movementAmount \cdot direction \cdot t_{sensitivity} \tag{23}$$

$$\Delta x = distanceChange \tag{24}$$

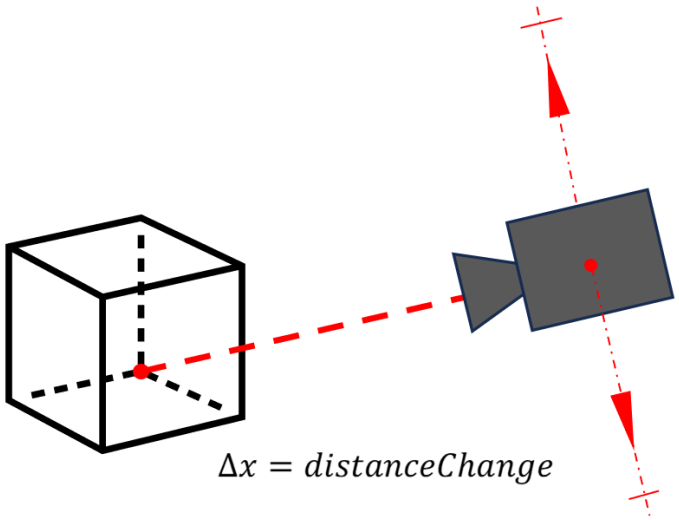


Fig. 18. Representación gráfica del movimiento radial.

El método puede retornar un `std::vector<QVector3D>` (vacío si hubo algún problema, para evitar resultados imprevistos) que contiene los siguientes vectores, adecuados al tipo de movimiento actual:

$$\Delta P = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix}, \Delta O = \begin{bmatrix} \Delta \theta_x \\ \Delta \theta_y \\ \Delta \theta_z \end{bmatrix}, \tag{25}$$

que representan dos vectores tridimensionales que se "adicionan a" o "substraen de" la posición y orientación actual de la cámara para configurar transparentemente su pose 6D.

Más aún, resultó conveniente (para el caso del movimiento cónico y esférico) que el método también pueda modificar directamente la pose de la cámara, por simplificar las operaciones, a través del argumento "selectedTranslation", un `QVector3D` que se introduce a la función por referencia no constante.

Se puede ver que esta es una forma de implementar una transformación rígida o euclídea 3D en la que se concatenan las operaciones de rotación y traslación directamente:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ W' \end{bmatrix} = \begin{bmatrix} i_1 & j_1 & k_1 & t_x \\ i_2 & j_2 & k_2 & t_y \\ i_3 & j_3 & k_3 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = \begin{bmatrix} R & \boldsymbol{t} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}, \Delta P = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix}, \Delta O = \begin{bmatrix} \Delta \theta_x \\ \Delta \theta_y \\ \Delta \theta_z \end{bmatrix}, \tag{26}$$

donde $\vec{\imath} = [i_1\ i_2\ i_3]^T$, $\vec{\jmath} = [j_1\ j_2\ j_3]^T$ y $\vec{k} = [k_1\ k_2\ k_3]^T$ representan los vectores unitarios del sistema de coordenadas (de la cámara) transformado y $\vec{t} = \begin{bmatrix} t_x\ t_y\ t_z \end{bmatrix}^T$ representa el vector que une el origen del sistema de coordenadas original y transformado, contribuyendo a que la transformación tenga seis grados de libertad (aunque la matriz de transformación esté definida hasta un factor de escala al ser un objeto homogéneo y pueda

ser escalada por un factor $\lambda > 0$ arbitrario). Por ejemplo, una rotación alrededor de θ grados del eje "y" sin traslación alguna viene representada por:

$$R = \begin{bmatrix} \cos\theta & 0 & \text{sen}\,\theta \\ 0 & 1 & 0 \\ -\text{sen}\,\theta & 0 & \cos\theta \end{bmatrix}, t = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (27)$$

2.1.2.2.- DirectoryConfigDialog

Esta clase es una ventana de diálogo diseñada para permitir a los usuarios configurar los directorios que utiliza el simulador. Esta ventana de diálogo hereda de `QDialog`, una clase de Qt que proporciona una interfaz gráfica para la interacción bidireccional con el usuario. La ventana de diálogo incluye varios campos de entrada de texto (`QLineEdit`) (representados por los miembros `mDirVideoLineEdit`, `mDirEscaneoLineEdit`, y `mDirFramesLineEdit`) donde el usuario puede especificar las rutas de directorios para diferentes propósitos, como directorios de video, escaneo y captura de fotogramas.

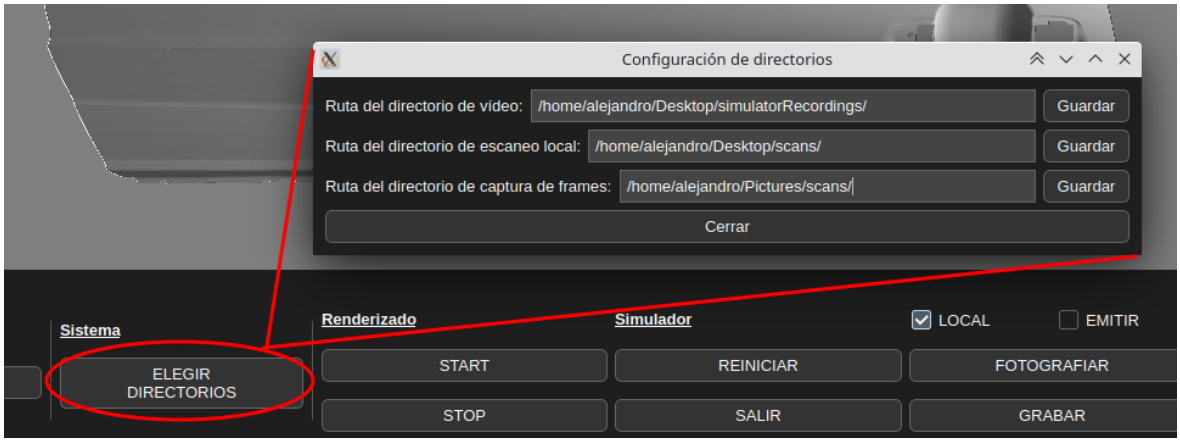


Fig. 19. Ventana emergente para la selección de directorios personalizada.

La clase ofrece tres funciones públicas, `guardarDirVideo`, `guardarDirEscaneoLocal` y `guardarDirCapturaFrames`, que permiten guardar las rutas de los directorios especificadas por el usuario. Estas funciones se conectan a elementos de la interfaz gráfica y se ejecutan cuando el usuario interactúa con la ventana de diálogo para guardar los directorios. Cuando se guardan los directorios, la clase emite señales, como `dirVideoGuardado`, `dirEscaneoLocalGuardado` y `dirCapturaFramesGuardado`, que notifican a otras partes de la aplicación sobre la actualización de las rutas de directorios (esencialmente a las clases `tfm::Renderer` y `tfm::FrameSaver`).

Además de las funciones públicas y los campos de entrada, la clase proporciona manejo de eventos mediante la reimplementación de las funciones virtuales heredadas como `resizeEvent` y `showEvent` para adaptar el tamaño de la ventana de diálogo y del widget convenientemente cuando se muestra al usuario.

2.1.2.3.- FrameSaver

Esta clase es la encargada de capturar y guardar fotogramas (frames) generados por la simulación OpenGL. Esta clase permite capturar los fotogramas generados por la simulación y guardarlos en varios formatos de imagen, incluyendo TIFF, PNG, JPEG y WebP, a niveles de compresión configurables. Está diseñada para ser ejecutada en un hilo secundario, lo que evita afectar al rendimiento de la simulación principal, al renderizado y a otros hilos.

La clase `tfm::FrameSaver` proporciona una ranura pública, `on_saveLocalFrame`, que permite guardar un fotograma localmente de acuerdo con la escena en pantalla durante el bucle de renderizado. Esta función toma como parámetros la imagen del fotograma, la ruta del directorio, el formato de imagen y opciones de compresión y calidad.

Además, la clase emite una señal, `sg_finished`, que notifica cuando la operación de captura y guardado de fotogramas ha finalizado. Por adición, en la siguiente figura se puede ver como al pulsar el botón de restablecer la animación, ya sea durante la grabación o fuera de esta, se reorganizan equidistantemente las piezas en la cinta de acuerdo con sus centros, a fin de preparar específicamente el entorno de experimentación de una manera mucho más rápida que si se tuvieran que colocar manualmente en cada iteración del procesamiento.

La clase `tfm::FrameSaver` se inicializa en su constructor con un puntero a un `SimuladorOpenGLWidget`, que proporciona funciones base de OpenGL utilizadas en todo el contexto OpenGL. También se utiliza un `QMutex` para garantizar la sincronización multihilo para el acceso al fotograma actual, lo que asegura una captura de fotogramas efectiva y evita problemas de concurrencia y condiciones de carrera.

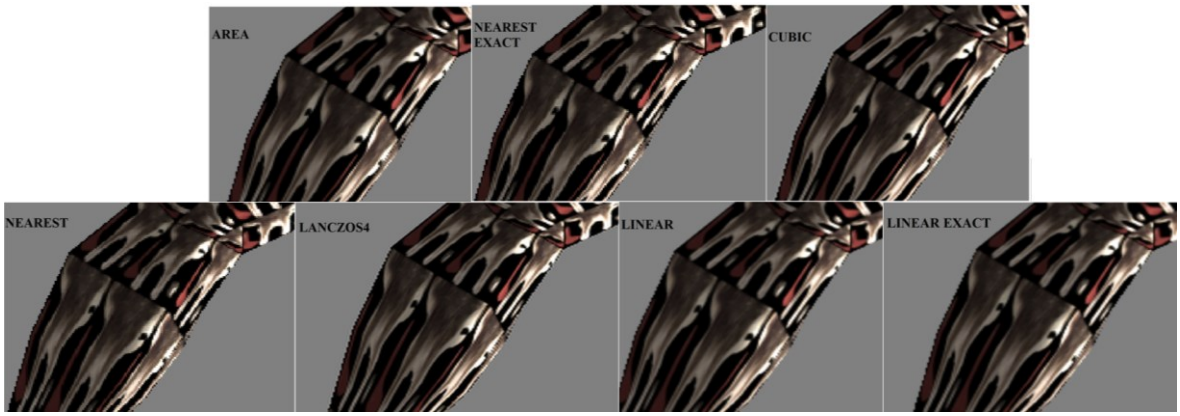


Fig. 20. Muestra del efecto en una textura de diferentes algoritmos de interpolación disponibles en OpenCV para la escritura de imágenes.

2.1.2.4.- MainWindow

Esta es la clase heredada de `QMainWindow` que, como la mayoría de las aplicaciones de esta índole en Qt, representa la ventana principal de la aplicación gráfica e incluye elementos como pulsadores (`QPushButton`), etiquetas (`QLabel`), cuadros de texto

(`QTextBrowser`) y diálogo (`tfm::SimuladorLogWindow`), selectores (`QComboBox`) y listas desplegables (`QListWidget`), espacios de diseño (`QGridLayout`), áreas de visualización (`QOpenGLWidget`), etc.

En el constructor de la clase se utiliza la lista de inicialización para definir el estado inicial de los elementos de la interfaz con los que interactúa el usuario y se define el puntero a esta interfaz, además, se establecen las conexiones entre `mSimuladorOpenGLWidget` y esta ventana (su padre), lo que permite la comunicación entre todos los elementos del apartado gráfico dentro del hilo principal. También se crea y configura un cuadro de diálogo para la configuración de directorios y, en el destructor, se eliminan los recursos asociados a la interfaz de usuario y se realizan acciones de limpieza, como la interrupción del hilo dedicado a la grabación de frames.

Todos los eventos de entrada comandados por acciones externas del usuario se transfieren al simulador a través de slots privados; en función de los valores capturados se realiza cierto preprocesamiento o tratamiento de los datos antes de transferirlos a las clases convenientes mediante señales de manera que la comunicación sea segura ante múltiples hilos (por ejemplo: en `on_perspectivaComboBox_activated` hacia `Camera`, en `on_posHorizontalSlider_sliderMoved` hacia la clase `Model`, en `on_resetAnimationPushButton_clicked` hacia `Renderer`… y así con las 50 ranuras con las que cuenta esta clase).

Además, se implementan tres ranuras privadas, `escribeEnLogWindow`, `actualizaResolucion`, y `mostrarOpenGLWidget` que llevan a cabo funciones directamente a través de la ventana de renderizado y sus capacidades, por lo que se cuida de que no se invoquen explícitamente desde fuera de la clase. Cabe comentar que, además de modificar el ancho y alto de la vista de la escena, los diales acoplados a la interfaz controlan el *offset* o desplazamiento de la cámara en las direcciones horizontal y vertical, de tal forma que, con una de las dimensiones fijas, la variación de la contraria controlará la parte de la escena que sirve como rectángulo delimitador de la vista, emulando el efecto de captura del mosaico de sensores en una cámara real convencional.

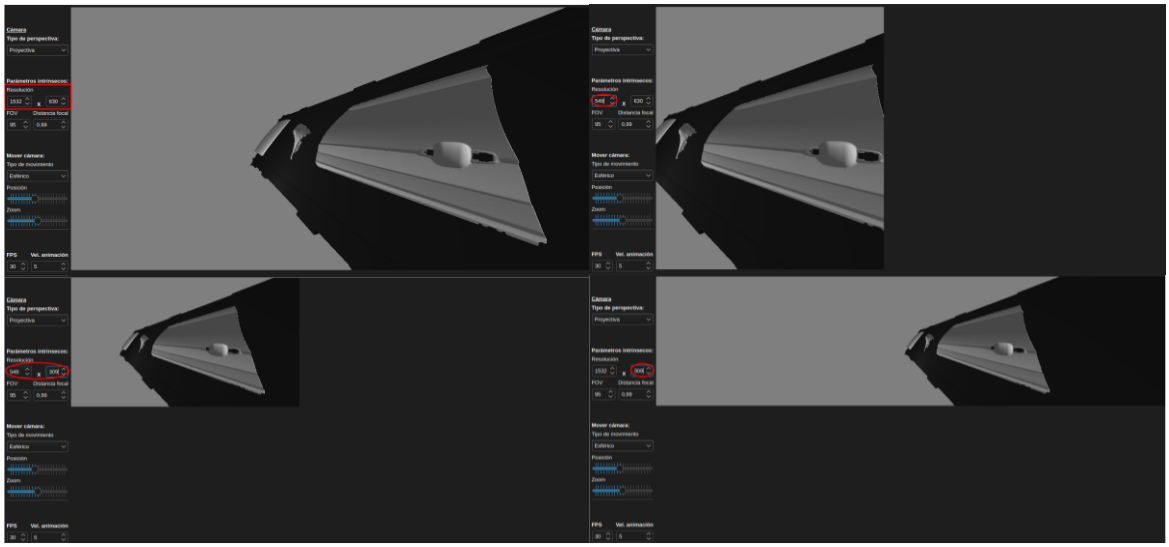


Fig. 21. Modificación de la altura y la anchura del plano imagen de la cámara. Se observa que, además de disminuir la resolución, se manipula el offset de la cámara proporcionalmente.

Por añadidura, la ventana principal es la encargada de comandar el inicio y la parada de la grabación en modo de captura de frames, mediante las señales `startRecording` y `stopRecording`, transmitidas hacia la clase `SimuladorOpenGLWidget`. En cada ranura se debe controlar la permisión y prohibición de la llamada a funciones OpenGL de manera adecuada para que el renderizado y las tareas relacionadas se realicen en el contexto activo (que también es el único creado, por comodidad) y se garantice una simulación sin fallos, mediante `QOpenGLWidget::makeCurrent` y `QOpenGLWidget::doneCurrent`.

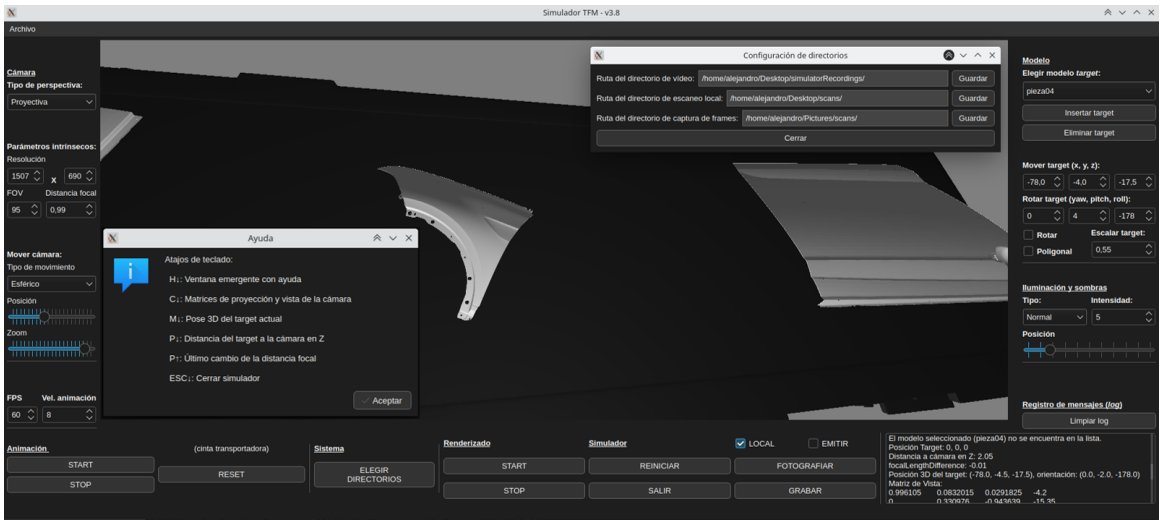


Fig. 22. Vista general de la ventana principal del simulador y dos de sus ventanas de diálogo emergente.

2.1.2.5.- Mesh

La clase `tfm::Mesh` representa el modelo tridimensional a nivel de malla. Esta clase está diseñada para trabajar con modelos 3D y facilita el acceso a información detallada sobre las mallas de estos modelos. En esencia, permite cargar y gestionar la información

relacionada con los vértices, normales, coordenadas de las texturas, índices, nodos, eslabones (huesos) y coordenadas absolutas de las mallas tridimensionales.

La funcionalidad principal de esta clase se encuentra en los métodos `processNode` y `processMesh`. El método `processNode` es especialmente importante, ya que se encarga de procesar todo los nodos de la escena proporcionada de manera recursiva. Dado que una escena puede contener múltiples nodos, cada uno con varios hijos y varias mallas, este método permite acceder a toda esta estructura y recuperar la información necesaria para su posterior uso en el simulador.

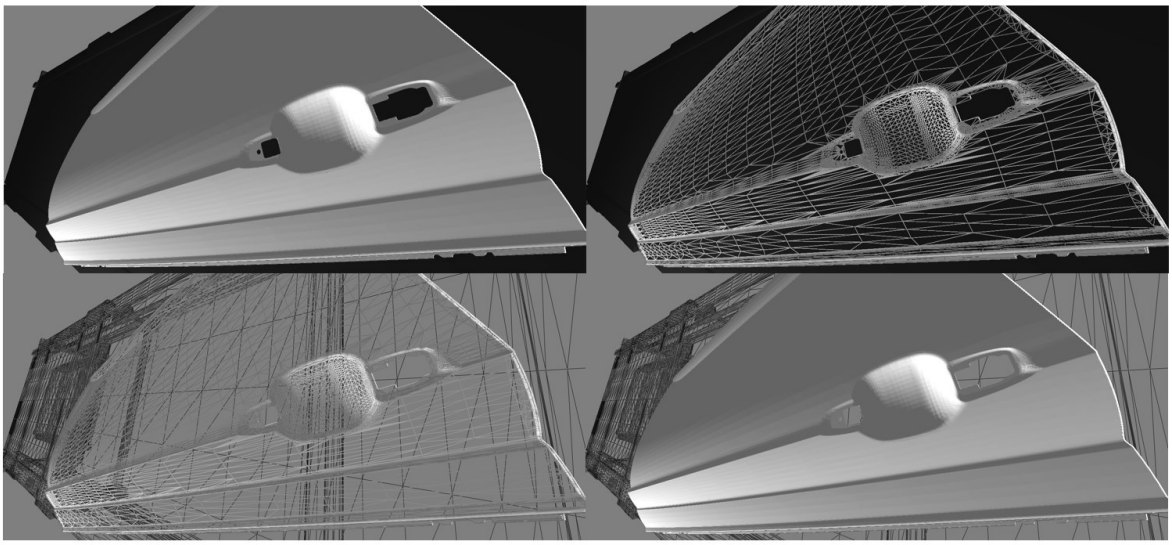


Fig. 23. Vista de las mallas de dos modelos tridimensionales, activando el modo "Poligonal" del simulador.

La clase `tfm::Mesh` también tiene atributos privados, como `mBoneIDs` y `mWeights`, que se utilizan para controlar con precisión los huesos y pesos asociados a cada eslabón de un modelo. Esto sería esencial en la representación de objetos u objetos que se deforman o cuya dinámica es diferente entre sí, por ejemplo.

Además, se utiliza una variable estática `mMeshData` que contiene toda la información de los vértices de un modelo 3D. Esta variable estático facilita el acceso a la información necesaria desde otras clases y se considera inmutable.

2.1.2.6.- Model

Esta clase está, diseñada para encapsular y gestionar modelos tridimensionales en la escena 3D. Esta clase ofrece una serie de atributos y métodos que facilitan la manipulación y representación de los modelos en el simulador.

El constructor de la clase `tfm::Model` permite crear instancias de esta clase y se puede inicializar con punteros a las funciones básicas de OpenGL (`myOpenGLFunctions`) y un programa de sombreado (`shaderProgram`) específico, que es constituido por un conjunto de shaders que cumplen una función específica y cuya ejecución se habilita en tiempo de ejecución (mediante el uso de funciones como `glCreateProgram`,

`glAttachShader` y `glLinkProgram`). Esto es esencial para la renderización de vértices, mallas, texturas y la posición específica de los modelos en la escena 3D.

Entre los métodos más importantes de la clase Model, se encuentran: `draw`, que se utiliza para dibujar las mallas de los modelos en la escena OpenGL (mediante funciones como `glBindVertexArray` y `glDrawElements`), `calculateCenter`, que calcula iterativamente y almacena las coordenadas espaciales del centro de cada modelo, que es la media aritmética tridimensional de las coordenadas de todos los vértices, `applyOrNotTexture`, que controla si se debe aplicar una textura al modelo, dependiendo de si se ha cargado una textura específica o si se opta por cargar el modelo con un color homogéneo por defecto, `applyTransform`, que aplica transformaciones al modelo, incluyendo rotación y traslación, ya sea desde la interfaz gráfica de usuario o debido a animaciones (también se puede activar una transformación poligonal para visualizar explícitamente las aristas del modelo) o `loadModelAndTexture`, que carga un modelo y su textura, con una lógica especial que busca nombres y directorios por defecto para cargar texturas automáticamente si el modelo dispone de ellas.

La clase `tfm::Model` también almacena atributos importantes, como el nombre del modelo, su color, la presencia de texturas, el factor de escala y la posición y orientación inicial y actual del modelo en el espacio 3D. Además, cuenta con punteros a mallas y texturas asociadas al modelo que se tienen en consideración en el renderizado del mismo.

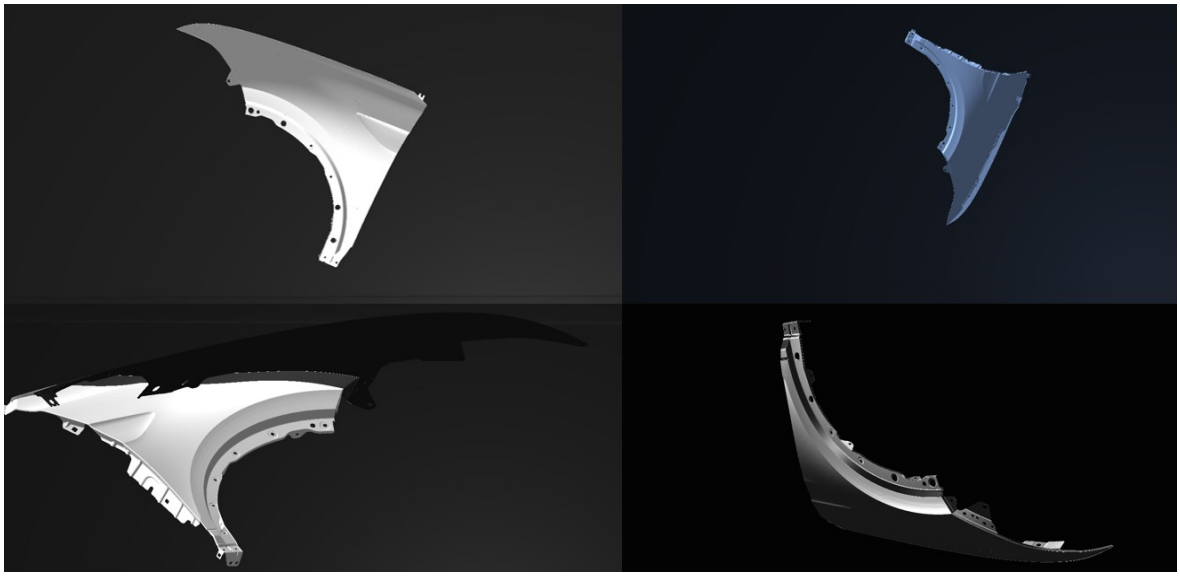


Fig. 24. Una pieza a distintas escalas y con distintas poses e iluminaciones.

El método `loadModelAndTexture` es uno de los más relevantes para el funcionamiento del simulador:

Este inicia su proceso mostrando un mensaje de depuración en la consola, informando que se está cargando un modelo 3D, lo que es útil para seguimiento y depuración del programa.

Luego, se emplea la biblioteca Assimp para importar el modelo 3D desde un archivo específico del sistema local. Durante este proceso, se aplican varias transformaciones al modelo, como la triangulación de caras, la inversión de coordenadas UV, el cálculo del espacio tangente y la generación de cajas delimitadoras; estas transformaciones son esenciales para asegurar la compatibilidad del modelo con el motor gráfico y su futura renderización.

Después de la importación, se realiza una verificación para asegurarse de que la operación fue exitosa. Se comprueba si la variable `scene` contiene datos válidos y si no se han producido errores durante la importación. En caso de error, se muestra un mensaje de error en la consola y la función devuelve false para indicar que la carga del modelo no fue exitosa.

El nombre del modelo, sin la extensión del archivo, se extrae y almacena en la variable `mName`. Lo que es útil para identificar al modelo durante el ciclo de vida del simulador.

Se verifica si la cadena `texturePath` está vacía; si es así, asume que la textura debe buscarse en el directorio "/textures" asociado al modelo. Si se encuentra, se carga. Si `texturePath` no es una cadena vacía, la función intenta cargar la textura desde el directorio especificado en el argumento homónimo.

Luego, se preparan los datos del modelo para su renderización. Los vértices, normales y coordenadas de textura se almacenan en la estructura `MeshData`, que se utiliza para crear los buffers de OpenGL necesarios, que se generan para almacenar los datos del modelo, incluyendo los vértices, normales, coordenadas de textura e índices y acceder de manera eficiente a dicha información. Los índices se cargan en un búfer especial de OpenGL (`GL_ELEMENT_ARRAY_BUFFER`) que se empleará para renderizar eficientemente el modelo.

A continuación, se configuran los atributos de posición, normales y coordenadas de textura en los búferes de OpenGL (cinco tipos de buffer para los cinco tipos de atributos a persistir), y se habilitan para su uso en el proceso de renderización.

Finalmente, se deshabilita la generación automática de coordenadas de textura en OpenGL, si es que fue habilitada previamente. La función devuelve true para indicar que la carga del modelo y la textura fueron exitosas.

La matriz de modelo define la traslación, rotación y escalado tridimensional a los que se somete un conjunto de vértices de un modelo, con respecto al sistema de coordenadas del mundo, antes de ser renderizado en pantalla. Esta matriz se utiliza para transformar las coordenadas de los vértices de un objeto desde su sistema de coordenadas local al sistema de coordenadas del mundo; esto significa que se multiplica por la posición de cada vértice del modelo que se desea transformar. En el "fragment" shader, la normal del fragmento se transforma usando la matriz de modelo para cálculos de iluminación.

Empleando el paradigma de OpenGL moderno se recurre a las funciones `glm::translate`, `glm::rotate` y `glm::scale` para definir las transformaciones que se aplican sucesivamente a los modelos, individual o colectivamente, según las órdenes del usuario a través de la interfaz o el código relativo a las animaciones o el movimiento de la cámara (zoom, perspectiva…). La aplicación de transformaciones a los modelos en tiempo real es muy delicada de implementar; se puede notar que no solamente es necesario elegir un modelo y aplicarle una transformación, sino también asegurar que el resto se mantengan tal como estén aunque gire o se acerque la cámara, conseguir que la deformación de los vértices de los objetos se corresponda al tipo de proyección y al movimiento libre al que estén sometidos, persistir la pose relativa de los vértices entre sí aunque la escena cambie, etc.

2.1.2.7.- Renderer

La clase `tfm::Renderer` se encarga de gestionar la renderización de la escena tridimensional y la debida administración de los modelos y la iluminación dentro de esta para que la simulación sea dinámicamente precisa. Su constructor permite crear instancias de esta clase con parámetros opcionales, como un puntero a las funciones básicas de OpenGL, el contexto de OpenGL, la superficie de renderizado, el shader utilizado y la cámara para la visualización.

Una vez inicializada, esta clase ofrece varios métodos significativos para interactuar con la escena tridimensional. El método `initRenderer` se utiliza para inicializar el renderizador y configurar los parámetros necesarios para la renderización (velocidad de la animación, modelos por defecto y su posición…). Por otra parte, `createModel` se usa para generar los modelos tridimensionales y agregarlos a un array dinámico almacenado en el *heap*[11] de modelos disponibles en el simulador, permitiendo cargar modelos desde archivos de malla y textura, además de configurar propiedades como su traslación, rotación y escala. Luego, `clearModels` implica eliminar todos los modelos de la escena, lo que es útil antes de reiniciar la simulación.. Por otra parte, `setVelAnim` permite ajustar la velocidad de animación, que influye en la rotación y traslación de los modelos en la escena. Además, se implementa la capacidad de que se realice un renderizado "de fondo" para que, por ejemplo, aunque la ventana se encuentre minimizada la emisión de frames al exterior pueda seguir realizándose cuando, a priori, la escena se dejaría de actualizar.

[11] El "heap" (montón) es una región de la memoria a utilizada para almacenar objetos y datos que se solicitan dinámicamente durante la ejecución de un programa. A diferencia del "stack" (pila), que se utiliza para almacenar variables locales y otras entidades cuyo tiempo de vida está vinculado al ámbito o alcance de las funciones o bloques de código, el montón se utiliza para datos que deben tener una duración más larga o cuya cantidad o tamaño no se conoce de antemano en tiempo de compilación.

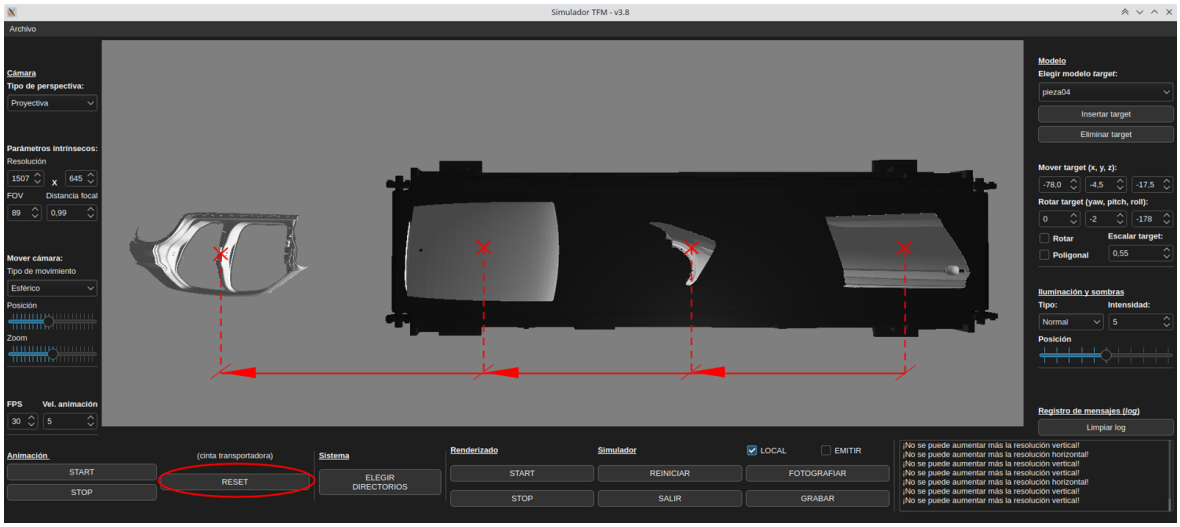


Fig. 25. Recreación del restablecimiento (reset) de la animación de avance de las piezas en la cinta.

Un aspecto destacado de esta clase es la capacidad de capturar imágenes de la escena. `captureFrame` permite tomar un fotograma de la escena y guardarlo como una imagen en el sistema de archivos, con opciones de formato, compresión, calidad y tamaño personalizables. La función `captureScreen` captura una imagen de la pantalla actual, mientras que `captureBlackFrame` obtiene un fotograma negro, que puede ser útil para ciertas aplicaciones. Además, se puede guardar un fotograma negro en el sistema de archivos utilizando `saveBlackFrame`.

La renderización de la escena tridimensional se lleva a cabo mediante el método `render`, que es el núcleo de esta clase y unos de los métodos más fundamentales del simulador. Se comienza comprobando si la renderización está pausada; si es así, se realiza una limpieza rápida y se establece un fondo blanco en la pantalla (limpiando el buffer de color y profundidad), lo que permite al usuario pausar y reanudar la visualización de la escena en cualquier momento. Luego, el método se encarga de inicializar los modelos en la escena; donde en la primera llamada a esta función se agregan varios modelos específicos (verbigracia, la escena con la cinta transportadora y las piezas preparada de antemano) con diferentes propiedades, como posición, rotación, escala y textura. Estos modelos se almacenan en el array `mModelList`. Después de esta inicialización, se configuran los aspectos iniciales de OpenGL y la cámara, incluyendo el modo de proyección y transformaciones de la cámara. A continuación, se establecen los parámetros de iluminación, se define el programa de sombreado a utilizar y se procede a renderizar cada modelo en la lista. Se aplican transformaciones en cada llamada al bucle de renderizado y se decide si aplicar texturas o usar colores sólidos en función de las propiedades de los modelos y del estado actual de la interfaz de usuario. Finalmente, se desactiva el efecto de desenfoque gaussiano y se desconecta el programa de sombreado, entre otros.

La clase también contiene una serie de atributos importantes que controlan el comportamiento del renderizador y la manipulación de los modelos. Estos atributos incluyen indicadores para detener la renderización, controlar la rotación de los modelos,

mostrar u ocultar aristas, iniciar o detener la animación, y elegir un modelo específico para operaciones particulares. La lista de modelos disponibles se almacena en un vector, y hay variables para trasladar modelos y controlar la interacción con el ratón.

De manera similar a como se transformaba la pose de la cámara con las distintas ecuaciones de movimiento, se pueden implementar para mapear los puntos de los vértices del modelo en el espacio cartesiano $\mathbb{R}^3$. Si además la transformación conlleva un cambio (uniforme, en este caso) en la escala del objeto, lo que puede ocurrir si se modifica el `QComboBox` denominado "Escalar target" que controla el tamaño del modelo objetivo desde la GUI, entonces se debe tener en consideración una transformación métrica o de semejanza, que preserva los ángulos y las líneas rectas pero añade un escalado uniforme (común para las coordenadas) en el espacio cartesiano:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ W' \end{bmatrix} = \begin{bmatrix} R & \boldsymbol{t} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} =^{*} \begin{bmatrix} sR & \boldsymbol{t} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}, \tag{28}$$

donde s_x, s_y y s_z son los factores de escala en las direcciones positivas de los ejes "x", "y" y "z", respectivamente y, si son iguales (*), la transformación tiene 7 grados de libertad.

Cabe comentar que para hacer más eficiente la simulación y el renderizado se hace uso de varios objetos típicos del enfoque moderno de OpenGL y de la programación gráfica en general. Todos son búferes de memoria; el primero es el VBO (*Vertex Buffer Object*), que almacena en la GPU datos de vértices de los modelos a fin de mejorar el rendimiento de la aplicación, el segundo es el VAO (*Vertex Array Object*), que almacena configuraciones de atributos de vértices, a fin de facilitar la administración de múltiples VBO y el tercero es el EBO (*Element Buffer Object*), que almacena índices de vértices para reutilizarlos y reducir el uso de memoria.

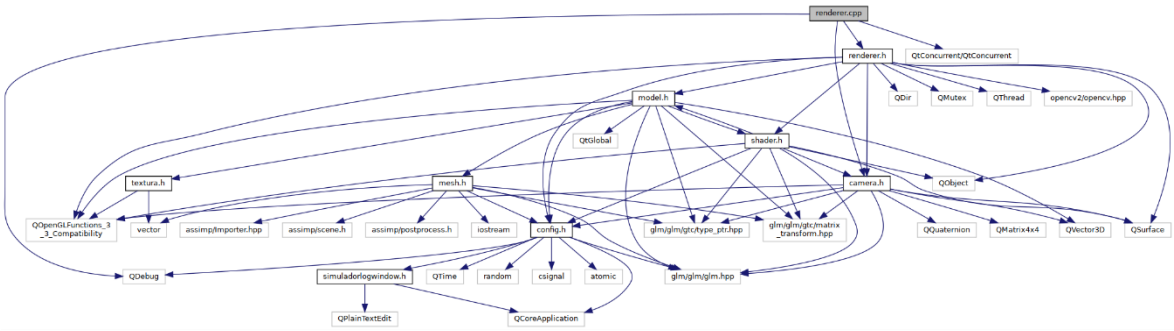


Fig. 26. Grafo de dependencias para "renderer.cpp", generado mediante Doxygen, disponible junto con la documentación completa en los anexos del trabajo.

2.1.2.8.- Shader

La clase `tfm::Shader` se encarga de administrar y controlar el programa de sombreado utilizado para la generación realista de sombras y efectos de iluminación en la escena tridimensional. Esta clase es fundamental para lograr representaciones visuales

realistas en el entorno tridimensional y, más importante aún, emular diferentes y variadas casuísticas con las piezas a fin de poder disponer de una base de datos surtida con la que experimentar tanto con métodos analíticos como de aprendizaje automático.

La inicialización de los shaders GLSL (de vértices [para el manejo de las posición espacial de las mallas] y de fragmentos [para el manejo de la dirección, reflexión, el mapeado… de la iluminación]) y el manejo de sombras son tareas importantes que se llevan a cabo en la función `initShaders`, que carga cargar los fragment shaders y "vertex" shaders desde archivo (mediante funciones de OpenGL como `glCreateShader`, `glShaderSource`, `glCompileShader`, `glGetShaderiv`, `glCreateProgram`, `glAttachShader`, `glLinkProgram`…). La función `initShadows`está implementada pero se mantiene para hacer notar que sería una manera de llevar a cabo lo que se hace en la anterior (establece las sombras en la escena) pero empleando la metodología fixed pipeline de OpenGL (obsoleta).

La función `configureMatrix` se encarga de configurar las matrices de vista y proyección, que son esenciales para la correcta representación de objetos en la escena. También configura los parámetros de iluminación en `configureIluminationParams`, considerando el tipo de iluminación deseada para especificar el número de focos, su posición y orientación, intensidad deseada, matrices de mapeado, modelo matemático de aproximación, etc., a fin de controlar flexiblemente y aplicar efectos de iluminación muy concretos a los objetos (iluminación difusa, intensa, diurna, con reflejos y con sombras, según lo definido en la variable `IluminationType`).

Además de estos aspectos, la clase `tfm::Shader` administra varias variables que influyen en la apariencia visual de la escena, como la habilitación de sombras, la intensidad de la reflexión especular, el tono de color de los focos de luz y el brillo de la superficie, entre otros.

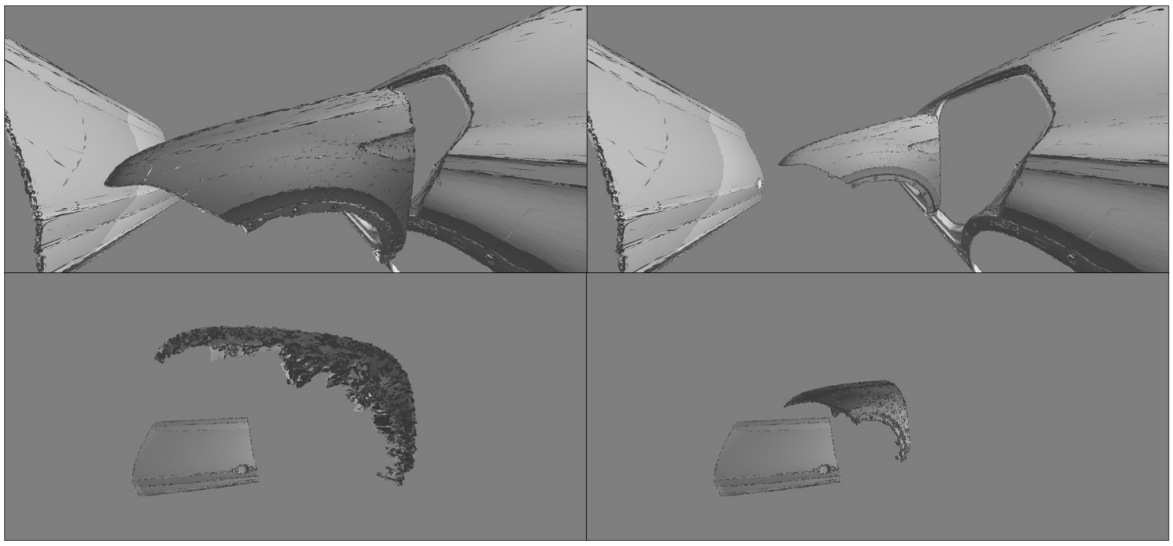


Fig. 27. Muestra notablemente exagerada del desenfoque artificial implementado. A medida que los objetos se alejan, con una distancia focal pequeña, el efecto es menos visible.

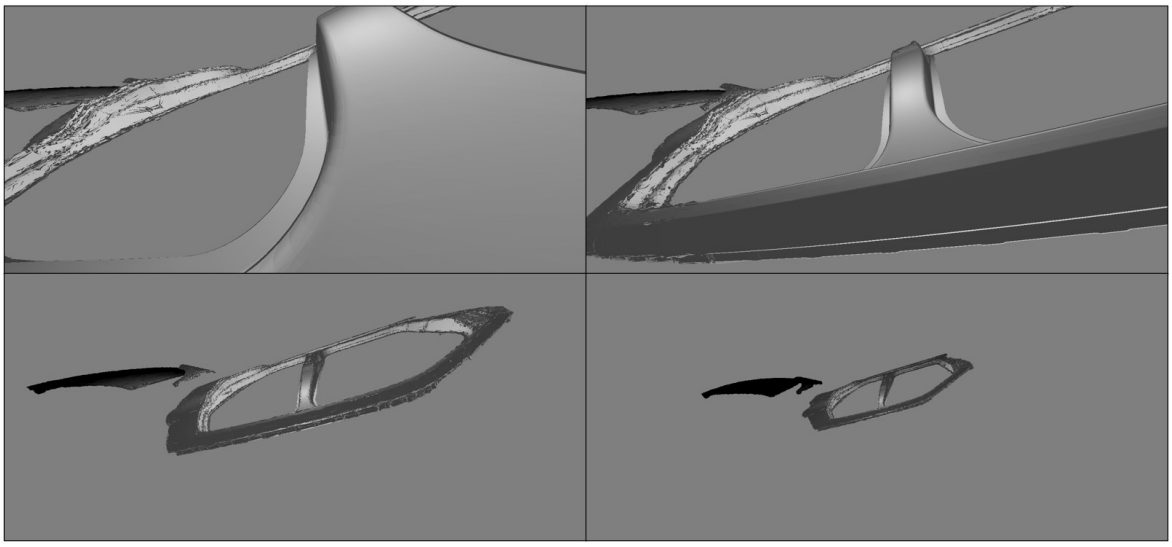


Fig. 28. Muestra notablemente exagerada del desenfoque artificial implementado. A medida que los objetos se alejan, con una distancia focal grande, el efecto es más visible.

Un shader es un programa altamente especializado que se ejecuta (típicamente) en la GPU y resulta ser el componente esencial del pipeline programable, responsable de transformar "entradas" (datos crudos de posición de vértices, colores de fragmentos) en "salidas" (tales datos transformados [deliberadamente] de «cierta» forma); además, operan de manera totalmente independiente con otros shaders, comunicándose solamente a través de sus entradas y salidas. Como ya se ha comentado, en el contexto de OpenGL, los shaders se escriben en un lenguaje llamado GLSL, que es un lenguaje similar a C (en cuanto a su especificidad para la manipulación de vectores y matrices, básicamente) diseñado para programación gráfica. Los shaders escritos en GLSL siempre comienzan con una declaración de versión ("330" en modo de compatibilidad, en este caso) seguida de la especificación de variables de entrada y salida o de tipo "uniform" y una función principal (main) donde se implementa la lógica del shader. En relación íntima con eso, vinculando varios shaders conjuntamente se puede crear un programa de shader en OpenGL; esto incluye, como mínimo, un shader de vértices (que procesa datos de los vértices de los modelos) (a nivel exclusivamente geométrico, en este trabajo) y otra de "fragmentos" o píxeles (que maneja el procesamiento a nivel de [intensidad de] píxel, usualmente pensado para agregar funcionalidades más avanzadas que las que suelen integrarse en el sombreador de vértices).

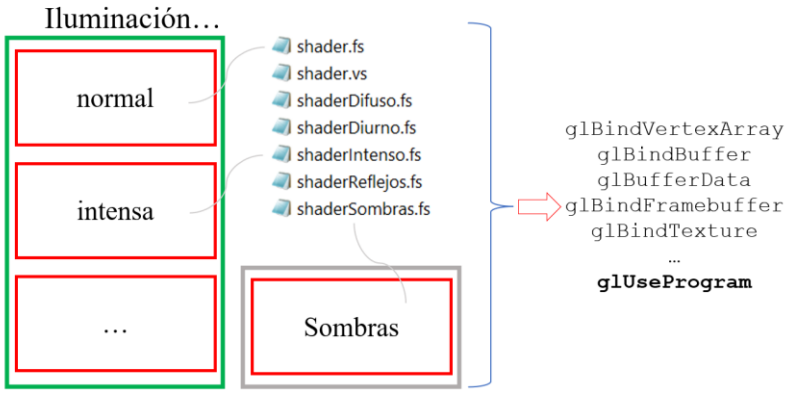


Fig. 29. Uso de los shaders en el simulador realizado.

Así, dentro del programa, los shaders quedan comunicados a través de sus entradas y salidas, que a su vez pueden ser variables (globales, locales, estáticas… pasadas desde la CPU al shader) de cualquier de las clases con las que trabaja el simulador. Los shaders se compilan, se vinculan en un programa y luego se utilizan durante la renderización para mostrar la escena diseñada. Una vez que un programa está vinculado al contexto de OpenGL, se puede utilizar para renderizar gráficos activándolo en el momento que se considere oportuno durante el proceso de renderización. Más aún, puede haber diferentes shaders asociados a diferentes programas (asociados a diferentes contextos) para llevar a cabo ciertas tareas u otras; esto es lo que se aprovecha para realizar desenfoque a la imagen, agregar iluminaciones especiales, componer una escena con sombras, procesar simultáneamente la posición de cada arista de los modelos, etcétera. Con piezas más pequeñas el efecto se ve más exagerado que con piezas mayor escala.

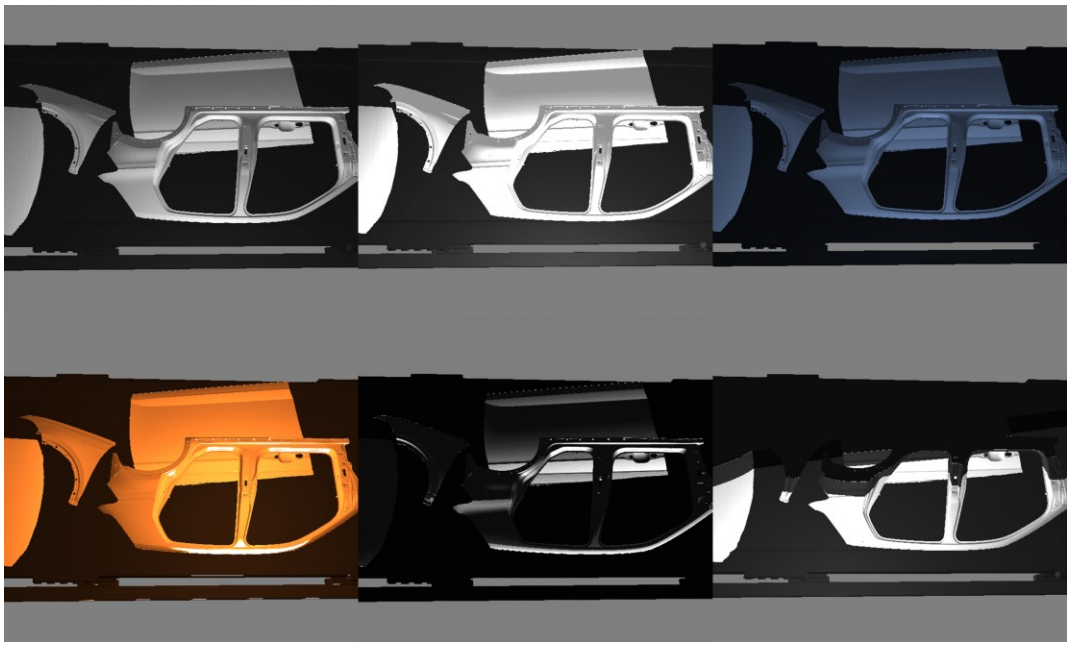


Fig. 30. Ejemplo de escena bajo el efecto de diferentes tipos de sombreadores; de izquierda a derecha y de arriba abajo: iluminación normal, difusa, diurna, intensas, con reflejos y con sombras (así se han denominado).

En el vertex shader se calcula la intensidad del filtro gaussiano en función de la diferencia en la distancia focal ideal (1,0) y la actual (`focalLengthDifference`), que se utiliza para aplicar un efecto de desenfoque de magnitud proporcional a su valor, con signo. Para emular el efecto de desenfoque de la cámara se aplica una deformación a cada uno de los vértices en función de la distancia entre la cámara y dichos puntos; la deformación se define como una transformación lineal (escala más offset) a cada vértice a lo largo de los tres ejes cartesianos (de manera alternada [uniendo la posición absoluta sobre el eje "x" con la del "y", etc.]), de forma que se establece una deformación aleatoria suficientemente mínima para provocar una emulación realista pero perceptible del círculo de confusión de la cámara. Junto con la modificación de la posición de cada uno de los vértices (más realista será el desenfoque cuanto más realista sea el modelo [más mallas tenga]), desde el fragment shader se aplica un efecto de iluminación gaussiana (ruido blanco) que provoca que los vértices del modelo desenfocado emanen un halo de luz oscura, contribuyendo sensiblemente a la emulación de la pérdida de nitidez de los objetos no pertenecientes al plano de enfoque. La mencionada deformación (pseudo)aleatoria de los vértices viene dada por la siguiente función que genera números aleatorios en el rango [0, 1] basada en una semilla:

$$customRand(\overrightarrow{seed}) \equiv f(x,y) = \text{fracc}\left(\text{sen}\left(\left(\overrightarrow{seed} \cdot \vec{v}\right) \cdot 43758{,}5453\right)\right), \qquad (29)$$

donde $\overrightarrow{seed} = [x\ y]^T$ es un vector bidimensional que representa la semilla de la función hash (coordenadas del vértice a deformar), $\vec{v} = [12{,}9898\ 78{,}233]^T$ es otro vector constante con el que se hace el producto escalar y "fracc" es un operador que devuelve la parte fraccional del resultado (el valor restante tras eliminar la parte entera). Esta función (que en el fondo es una senoide de altísima frecuencia) transforma coordenadas 2D en 1D con parámetros elegidos para que los números o se repitan frecuentemente, amplifica el error de la función mediante coma o punto flotante y una función sinusoidal y tomando la parte fraccional del resultado, lo que ayuda a aumentar su variabilidad. Es una de entre otras funciones de generación de ruido blanco similar al ruido Simplex, de Perlin, Voronoi, la emulación fracción del movimiento Browniano y otros métodos típicamente empleadas en el lenguaje GLSL [37].

Cuanto mayor es la distancia y mayor la diferencia entre las distancias focales, más desenfoque se aplica. Además, se tiene en consideración el valor de la distancia focal de forma que valores elevados causan que los objetos pertenecientes al primer plano se enfoquen y los del fondo de desenfoquen, al igual que valores muy pequeños de la distancia focal provocan que los objetos que se enfoquen más sean los del fondo; todo ello de manera suave y proporcional a dicho valor y a la distancia a la cámara (con lo que el efecto se logra aplica en tiempo real de manera dinámica).

Las sombras de los objetos también disponen de matrices de vista y proyección. La matriz de vista (es decir, la ubicación y dirección desde la cual se ilumina) se puede definir mediante la función `glm::lookAt` y viene especificada por tres vectores, la posición del

foco (o los focos) de luz, la posición del objetivo a ser iluminado (p. ej., el centro del modelo objetivo) (el punto en el espacio hacia el que la luz se dirige) y el vector que define la posición del "cielo" de la escena que, normalmente, apunta hacia arriba (`glm::vec3(0.0f, 1.0f, 0.0f)`). Por otra parte, la matriz de proyección de la iluminación funciona de manera similar a las explicadas anteriormente; define cómo se proyectan los vértices al espacio de recorte de la cámara de luz. Ambas se multiplican para resultar en la matriz del espacio lumínico de sombras, que se utiliza comúnmente en técnicas de sombras como sombras paralelas (*shadow mapping*) y sombras proyectadas desde una fuente de luz y su propósito principal es transformar las coordenadas de los vértices desde el espacio del objeto al espacio lumínico, lo que permite realizar cálculos de sombreado y determinar si un fragmento está en sombra o iluminado.

Utilizando las funciones `glUniform3fv` y `glUniformMatrix4fv` se envían las matrices al shader como variables uniformes (que deben definirse en el ámbito global del GLSL y pueden ser de cualquier tipo o agregación de tipos [dentro del shader son como constantes, aunque no como expresiones constantes]), donde se utilizan en el shader para realizar los cálculos de sombreado y determinar si un fragmento se encuentra en la sombra o bajo la influencia de la fuente de luz.

Para aplicar el modelo de luz a la escena de renderizado se sigue una secuencia de pasos bien definida: aplicar un factor de escalado singular a las componentes de color de la luz ambiental según las necesidades del simulador (da cierto color a los objetos), normalizar las normales de los fragmentos y la dirección de la luz para que tengan la misma dirección que los vectores los de entrada pero con longitud unitaria, calcular la reflexión o intensidad especular (reflexión y cambio de dirección de propagación de la luz al incidir sobre la superficie [efecto de espejo]) y difusa (reflexión de la luz en todas direcciones debido a la rugosidad de la superficie) en base a la matriz de vista, las normales a los fragmentos y la dirección de la luz y componer el resultado mediante la combinación lineal de todas las intensidades y el uso de un muestreador de textura 2D, utilizado para acceder a los valores de píxeles de una textura en las coordenadas de textura con el propósito de que el shaders pueda obtener información de color u otros datos de la textura en un punto o fragmento específico.

El modelo matemático utilizado en la simulación que aproxima la física óptica de la luz en la realidad es el modelo de Phong, el cual predice la reflexión basándose en el ángulo de incidencia y el ángulo entre la dirección especular y la dirección al plano del sensor (así, la máxima reflexión se obtiene cuando coincide el ángulo de reflexión con la dirección especular) y se implementa siguiendo las expresiones que se muestran a continuación:

$$spec = \left(\max\left(\vec{\alpha} \cdot \vec{\beta}, 0\right)\right)^{32} \cdot \vec{\gamma} \tag{30}$$

$$diff = \max(\vec{n} \cdot \vec{\omega}, 0) \cdot \vec{\gamma}, \tag{31}$$

donde "spec" es la intensidad especular resultante, "diff" es la intensidad difusa resultante, el producto escalar ente $\vec{\alpha}$ (vector de dirección desde el fragmento [el punto en el espacio 3D que se está procesando en el fragment shader] hacia la posición de la cámara) y $\vec{\beta}$ (la dirección del vector de reflexión, calculada mediante la función `reflect`, que refleja el vector incidente alrededor de una normal) representa la reflexión de la luz en la superficie, $\vec{n}$ es la normal al fragmento, $\vec{\omega}$ es la dirección de la luz, se evitan valores negativos mediante el operador "max" (porque el exponente podría ser impar), se utiliza un exponente de 32 empírico para controlar la concentración y el brillo del reflejo especular y "lightColor" es el color de la luz.

2.1.2.9.- SimuladorLogWindow

Esta clase es responsable de gestionar y mostrar registros de mensajes en una ventana de registro. Esta ventana es crucial para rastrear eventos, errores y mensajes importantes durante la ejecución del programa.

`tfm::SimuladorLogWindow` es una ventana de registro que muestra mensajes relevantes relacionados con la ejecución del programa. Puede utilizarse para mantener un registro de eventos, errores, y otros mensajes de importancia. Esta ventana se conecta a controles de la interfaz de usuario, como deslizadores y cuadros combinados, para interactuar con las funciones de la clase.

La ventana de registro está diseñada para ser un componente de la interfaz de usuario y hereda de `QPlainTextEdit`, lo que le permite mostrar texto con diferentes formatos. Además, tiene opciones de configuración, como hacer que el texto sea de solo lectura, habilitar el desplazamiento vertical ("scroll") para ver mensajes antiguos y evitar que las palabras se envuelvan u ocluyan automáticamente.

Cuando se inicializa la ventana de registro, se abre un archivo de registro llamado "logfile.txt" en el directorio de trabajo actual u otra ruta del sistema local o se crea uno nuevo en caso de que no exista. Luego, los mensajes que se agregan a la ventana de registro se registran en dicho archivo de registro, lo que permite mantener un historial de los mensajes que devuelve la aplicación.

La función `appendMessage` se utiliza para agregar mensajes a la ventana de registro. El mensaje se muestra en la ventana de registro y se registra en el archivo local. Como detalle, la ventana de registro se desplaza automáticamente hacia abajo para mostrar el mensaje más reciente. La función `clearLog` borra todos los mensajes en la ventana de registro y también elimina el contenido del archivo local, lo que proporciona una forma de limpiar el registro deliberadamente cuando el usuario considere necesario. Además, la función `readLogFile` se utiliza para leer todo el contenido del archivo de registro y devolverlo como una cadena de texto, lo que puede ser útil para revisar los registros antiguos.

Fig. 31. Captura del widget denominado "SimuladorLogWindow" en funcionamiento.

2.1.2.10.- SimuladorOpenGLWidget

La clase `tfm::SimuladorOpenGLWidget` representa un widget personalizado de OpenGL diseñado para servir como ventana de renderizado en la aplicación. Proporciona funciones para la inicialización de OpenGL, manipulación de eventos, y otras funcionalidades relacionadas con la simulación, como interacción con modelos, texturas, extracción de fotogramas mediante archivos locales y WebSocket, grabación, movimiento de la cámara y parametrización de la lente y tipo de perspectiva, iluminación y animaciones, entre muchos otros.

Además de como ventana de renderizado, gracias a heredar de `QOpenGLWidget` tiene la capacidad de servir como interfaz de comunicación mediante eventos con el usuario a través del ratón y teclado, para lo que se implementan los métodos virtuales que capturan dichos eventos y se procesa la información entrante debidamente.

En su constructor, `SimuladorOpenGLWidget` se encarga de inicializar el widget de OpenGL y, por añadidura, se puede utilizar en la interfaz de usuario de la aplicación pues es un miembro del puntero "ui" de la clase `tfm::MainWindow`. Es importante destacar que hereda funcionalidades de `QOpenGLWidget`, lo que le permite interactuar con eventos de mouse, teclado y otros aspectos de la interfaz.

La función `initializeGL` se encarga de inicializar el contexto OpenGL. Esto implica configurar aspectos esenciales del entorno de renderizado (instancias de `tfm::Shader`, `tfm::Camera`, `tfm::FrameSaver`...) y permitir que OpenGL se prepare para renderizar la escena. Este método se llama una vez antes de realizar cualquier renderizado OpenGL. La función `paintGL` se llama para realizar el renderizado OpenGL, lo que significa que se encarga de dibujar y actualizar la escena tridimensional en el widget; se llama al método propio de la instancia de `tfm::Renderer`, no al protegido de `QOpenGLWidget` (al cual Qt la llama automáticamente cuando es necesario un refresco del renderizado en el widget dedicado, pero otras veces es necesaria una invocación explícita). La función `resizeGL` maneja el evento de cambio de tamaño de la ventana OpenGL; ajusta la vista de acuerdo con las dimensiones de la ventana para garantizar que la escena se muestre correctamente dentro de unos límites establecidos, incluso cuando cambia el tamaño de la ventana en tiempo real mediante modificación de la GUI.

El método `initOpenGLScene` se encarga de activar una serie de flags mediante la función `glEnable` que mejoran la calidad visual y realizan acciones (unas mandatarias y

otras recomendables) que modifican el comportamiento del renderizado de los modelos en escena; entre otros: habilitar el modo sensación de profundidad con el cual se asegura que los objetos más cercanos el espectador se rendericen por encima de los más lejanos (`GL_DEPTH_TEST`), activar el *culling* (sacrificio) para evitar renderizar las caras traseras de los objetos (`GL_CULL_FACE`), habilitar un filtro antialiasing mediante una técnica de sobremuestreo para reducir el efecto de dientes de sierra en los bordes (`GL_MULTISAMPLE`), activar la influencia del material del objeto en el color final que se mostrará (`GL_COLOR_MATERIAL`), suavizar las líneas y los puntos dibujados (`GL_LINE_SMOOTH` y `GL_POINT_SMOOTH`), establecer el uso de texturas bidimensionales (`GL_TEXTURE_2D`) y activar la anisotropía de texturas (mantener su calidad visual aunque sea vista desde diferentes puntos de vista o direcciones) (`GL_TEXTURE_MAX_ANISOTROPY_EXT`), ajustar automáticamente a una magnitud unitaria los vectores normales después de transformaciones (`GL_NORMALIZE`), habilitar el uso de *sprites* de puntos (objetos bidimensionales muchas veces transparentes que aparentan ser objetos con volumen y distantes) (`GL_POINT_SPRITE`) y permitir la mezcla de colores durante el renderizado para mejorar algún efecto de iluminación combinando los colores ya renderizados con los presentes en los búferes (`GL_BLEND`).

La clase también maneja eventos de teclado, mouse y rueda del ratón a través de métodos virtuales como `keyPressEvent`, `keyReleaseEvent`, `mousePressEvent`, `mouseReleaseEvent`, `mouseMoveEvent`, y `wheelEvent`, lo que permite al usuario interactuar con la escena tridimensional moviendo la cámara a su voluntad, interactuando con las posición con modelos con el teclado y otras posibilidades. Los eventos de renderizado se emiten cíclicamente de manera controlada a través de un temporizador (`QTimer`) que controla los FPS definidos mediante la propia interfaz; con un ciclo de 16 ms se puede establecer una tasa de 1000 / 16 ≈ 60 FPS. A un nivel más bajo, Qt utiliza un patrón de diseño de observador para gestionar los eventos, donde cada evento está representado por un objeto de evento específico (derivado de `QEvent`); cuando se envía la señal asociada al evento que ocurre el bucle de eventos de Qt, que está ejecutándose continuamente (si no se bloquea) mientras la aplicación está en funcionamiento, lo captura, desencadenando toda la lógica de caminos y conexiones que hubiera quedado definida en el programa en tiempo de compilación.

Además de la funcionalidad de renderizado, la clase proporciona características de grabación de video, configuración de resolución, modos de simulación y comunicación a través de WebSocket mediante librerías de Qt. La aplicación puede grabar fotogramas a intervalos regulares de tiempo, lo que permite la creación de videos de la simulación. Además, se pueden configurar distintos modos de simulación y comunicarse con otra aplicación principal (p. ej., de procesamiento o captura de frames) mediante señales, ranuras y eventos personalizados.

En el contexto de OpenGL y la programación gráfica en general, el proceso de dibujo se suele realizar en dos buffers: un buffer “de dibujo” y otro “de presentación”. El primero

es un área de memoria donde se realiza el renderizado y dibujado todos los modelos y elementos de la escena, que es invisible para el usuario y no se muestra en la ventana principal y el segundo es el área de memoria que se muestra en la ventana en tiempo real y contiene la imagen que el usuario ve en la pantalla. La razón principal para utilizar dos buffers es evitar "parpadeos" y otros efectos visuales adversos mientras se actualiza la imagen en la ventana, pues de esta forma, en lugar de dibujar directamente en el búfer de presentación (lo que podría resultar en una visualización parcial, intermedia o entrecortada de la escena), permite realizar los cálculos en el buffer de dibujo y, una vez completada la escena, es viable intercambiar sinápticamente los dos búferes (mediante `QOpenGLContext::swapBuffers`). Esta idea se aprovecha para realizar un renderizando de fondo para que la escena siga calculándose, aunque no mostrándose, cuando la ventana principal está minimizada u ocluida, a fin de que no haya discordancias en ningún cálculo en el momento de reaparición de la escena; para lograrlo se realiza la llamada a la función `paintGL` de manera cíclica en un hilo secundario que comparte el mismo contexto de renderizado que la ventana principal. Otra manera de lograrlo podría ser la creación de un contexto OpenGL separado más la gestión de un FBO para realizar los cálculos en él y el uso de otro programa de sombreado para realizar el *blit* o *block transfer* (la copia de bits de una región de memoria a otra) del contenido del FBO a la pantalla; ambos enfoques abordan los problemas de rendimiento y captura de pantalla al renderizar contenido fuera de la vista, proporcionando una solución robusta para las necesidades específicas del usuario (por ejemplo, cuando realiza una grabación y minimiza la ventana principal mientras activa una animación específica).

En la siguiente figura se muestran algunos artefactos visuales que aparecían en el momento de guardar frames en una carpeta local o al enviarlos remotamente; gestionando la lectura/escritura de las imágenes mediante archivos temporales (primero se solicita guardar un frame y, tras crear un temporal homónimo, quien lo lea comprueba primero que exista este último y en caso positivo espera para acceder a la imagen, pues tras acabar de guardarse se elimina el archivo temporal y se habilita el acceso a su lectura, evitando colisiones en tiempo de guardado), empleando métodos de sincronismo entre hilos y congelando el buffer de renderizado en el estado actual antes de la actualización de nuevas líneas de escaneo se consiguieron superar estos problemas. El valor de los píxeles de la imagen de la escena se almacena en un búfer en un puntero inteligente de tipo `std::unique_ptr<GLubyte>` (de dimensión igual a tres veces su área); se usa la función `glReadPixels` para leerlos, `glCheckFramebufferStatus` para comprobar que el estado del búfer es adecuado para la lectura y `glPixelStorei(GL_PACK_ALIGNMENT, 1)` para evitar errores usando `GL_RGB` y asegurarse de que cada fila de píxeles en la memoria sea consecutiva, es decir, sin ningún espacio entre filas, pues el formato empleado es `GL_UNSIGNED_BYTE`.

Fig. 32. Problemas de colisión de escaneos al visionar y capturar antes de realizar el renderizado de fondo.

Durante la ejecución del hilo que guarda los frames en el sistema local mientras se está haciendo una grabación se observó que los FPS establecidos desde la interfaz influían muy abruptamente en el ritmo de guardado que acontecía. Para escalarlos de manera suavizada y sincronizarlos fluidamente con el ciclo de grabación se emplea la siguiente expresión que mapea el valor del tiempo de ciclo del `QTimer` dedicado a la captura de frames y su exportación al exterior calculado directa y proporcionalmente a los FPS actuales en una función que limita su tasa de crecimiento y lo suaviza cuando los frames por segundo comienzan a aumentar rápidamente:

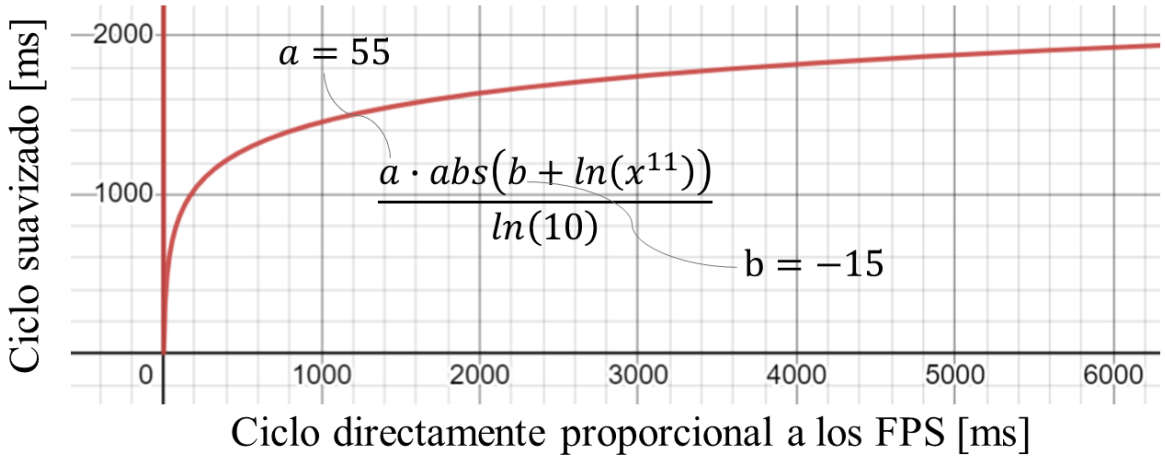


Fig. 33. Suavizado del mapeo entre FPS y tasa de emisión de frames al exterior.

Se ha observado, además, que la tasa de emisión de frames es demasiado elevada en modo local en comparación con el modo remoto, de manera que se exceden las capacidades actuales del sistema de comunicación WebSocket si se aumentan demasiado los FPS de la simulación; por ello, mediante la línea `mCycle = mLocal? 16 : qMax(50, qMin((int)mCycle, 1000));` se limita programáticamente desde el constructor de esta clase y otras partes la tasa de emisión de frames (es decir, la tasa de bits o *bitrate*) desde el simulador, de manera que se asegure que la información llegue a tiempo de que sea leída antes de volver a ser emitida y no haya colisiones de información.

2.1.2.11.- SimuladorWebSocketClient

La clase `tfm::SimuladorWebSocketClient` se encarga de establecer conexiones de WebSocket como cliente. Cuando se crea una instancia de esta clase, se le proporciona la dirección IP y el puerto del servidor WebSocket al que se conectará. La conexión se inicia automáticamente al crear un objeto `tfm::SimuladorWebSocketClient`, lo que significa que cuando el simulador se ejecuta, el cliente WebSocket se conecta al servidor WebSocket especificado.

Esta clase ofrece dos métodos fundamentales para la comunicación. El primero, `sendImage`, permite enviar imágenes codificadas en formato `cv::Mat`. Estas imágenes se empaquetan en un arreglo binario y se envían al servidor WebSocket para su procesamiento posterior, lo que es especialmente útil cuando se necesita transferir datos de imágenes entre el cliente y el servidor, lo que es común en aplicaciones de simulación. Esta trama (que se muestra a continuación) está formada por una cabecera compuesta por un entero (4 bytes) que representa la longitud del mensaje y un byte que representa el tipo de dato (puede ser una matriz de OpenCV, un `QImage`, `Pixmap`…) y, luego, los datos en sí mismos, compuestos por la altura, anchura de la imagen (8 bytes en total) y los datos crudos de la imagen (cuyo tamaño en bytes depende de la resolución de la ventana de renderizado) que representan la imagen en formato sin procesar.

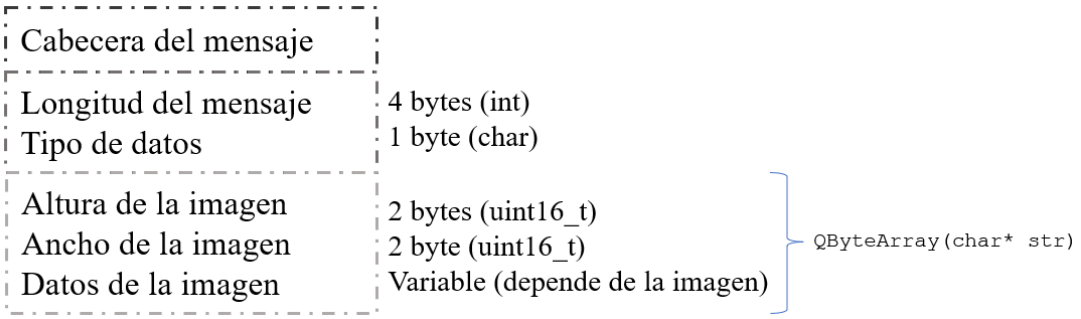


Fig. 34. Trama (array de bytes) transmitida vía WebSocket para comunicar remotamente el simulador y otro PC.

El segundo método, `sendText`, se utiliza para enviar mensajes de texto al servidor WebSocket. Esto proporciona una forma de comunicación basada en texto, lo que es valioso para enviar comandos, instrucciones u otra información textual entre el cliente y el servidor.

Como se usa una máquina virtual basada en Oracle VM Virtualbox, se establece su adaptador en modo “adaptador puente” para que su sistema tenga la dirección IP del PC que la alberga (192.168.0.21 en la máquina virtual frente a 192.168.0.12 en el PC) y esté conectada a su tarjeta de red para que sea accesible desde otros equipos conectados a la misma red.

2.1.2.12.- SimuladorWebSocketServer

La clase `tfm::SimuladorWebSocketServer` desempeña el papel opuesto al del cliente. En lugar de conectarse a servidores WebSocket, esta clase actúa como el servidor

que acepta conexiones entrantes. Al crear una instancia de `tfm::SimuladorWebSocketServer`, se especifica la dirección IP y el puerto en el que el servidor escuchará las conexiones entrantes.

Esta clase se encarga de gestionar las conexiones de los clientes WebSocket y recibir datos de ellos. Cuando se recibe un mensaje de texto o datos binarios de un cliente, la clase `tfm::SimuladorWebSocketServer` puede tomar medidas en consecuencia. Por ejemplo, si se recibe un mensaje de texto, el servidor podría reenviar ese mensaje a otros clientes o ejecutar acciones específicas basadas en el contenido del mensaje.

Cuando el servidor WebSocket recibe datos binarios, puede procesarlos y reenviarlos a otros clientes; esto se aprovecha para transferir (de vuelta) las imágenes de la escena de renderizado que le llegan desde el cliente hacia otro cliente que, desde una aplicación externa de procesamiento (por ejemplo), en una máquina distinta en la misma o incluso en otra LAN, esté escuchando conectada al propio servidor.

Especificar una dirección IP de `QHostAddress::Null` en la creación del servidor indica que el servidor no está asociado a ninguna dirección "127.0.0.1" o `QHostAddress::LocalHost` representa la dirección IP de "localhost" en IPv4 y sirve para establecer conexiones locales en una misma máquina y "0.0.0.0" o `QHostAddress::AnyIPv4` indica que se desea que esté disponible en todas las interfaces activas del servidor. Para la conexión del simulador, ejecutándose en una máquina, con otra máquina remota, se emplea la dirección de la red local disponible (configurable mediante la macro `WEB_SOCKET_SERVER_IP`) y un puerto arbitrario (configurable `WEB_SOCKET_SERVER_PORT`); al quedar registrado cada cliente en el establecimiento de la conexión se les asigna un puerto distinto y así quedan definidos los lazos de la comunicación.

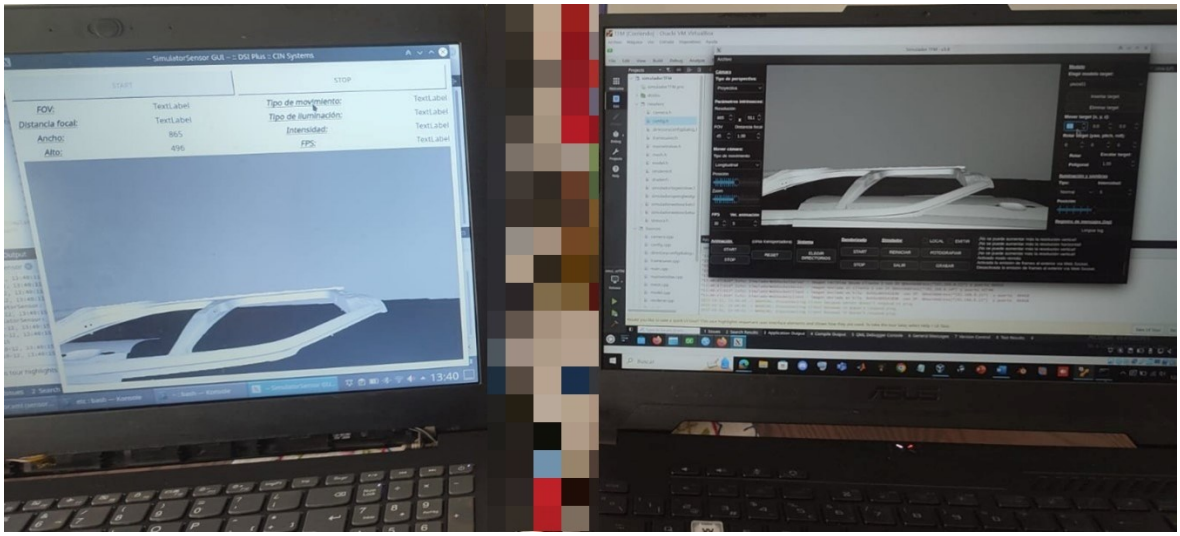


Fig. 35. Comunicación WebSocket en funcionamiento entre dos máquina distintas, una ejecutando el simulador y otra el sensor Camera2DSensor en modo de aplicación, bajo una misma red local.

2.1.2.13.- Textura

La clase `tfm::Textura` es la responsable de cargar (a bajo nivel) y gestionar texturas que se pueden aplicar a modelos dentro del simulador, que son imágenes o patrones (bidimensionales o tridimensionales) que se utilizan para dar realismo y detalles visuales a los objetos en la escena.

El constructor de la clase recibe dos parámetros esenciales: el primero es un puntero a las funciones base de OpenGL (`myOpenGLFunctions`) que se utilizan en todo el contexto de OpenGL y el segundo es el identificador del programa de sombreado (`shaderProgram`) que se utilizará para la renderización y lectura de las texturas y sus sombras.

El método `loadTexture` se utiliza para cargar una textura desde un archivo de imagen específico y asignarla a un índice determinado. Esto es importante porque un modelo puede tener múltiples texturas, y el índice se utiliza para identificar la textura correcta cuando se aplica a un modelo. Se utiliza la biblioteca STB para cargar imágenes desde archivos, se realiza un control de flujo para verificar si la carga de la textura fue exitosa y se manejan posibles errores. También se tienen en cuenta las unidades de textura máximas disponibles en el hardware bajo el que corre la aplicación (representadas por `maxTextureUnits`), lo que garantiza que las texturas se asignen a unidades válidas. Se configura el filtro de magnificación y reducción para las texturas, lo que afecta a cómo se verían las texturas cuando se aplicaran a los modelos. También se aplica un filtrado anisotrópico en caso de que sea compatible (lo que mejora la calidad de las texturas) y, finalmente, se asigna la textura al programa de sombreado para que se aplique a los modelos de la escena que corresponda.

El mapeo de texturas (*texture mapping*) en esta simulación implica asignar las coordenadas de una textura bidimensional (aunque existen variantes 1D y 3D utilizadas en técnicas como *sky-box* y *cube mapping*, que aquí no se usan) a los vértices de los modelos planos o tridimensional, permitiendo que las transformaciones se apliquen de manera coherente. Los “texels” o “píxeles de la textura” son los encargados de representar la información visual de las texturas, que en OpenGL tienen coordenadas normalizadas a un rango de 0 a 1. Los ejes X e Y de la textura se traducen a S y T, mientras que un tercer eje T aparecería en contextos tridimensionales.

En OpenGL, el proceso de texture mapping involucra tres entidades clave: el objeto de textura OpenGL, la unidad de textura y el shader GLSL. Los objetos de textura contienen las imágenes, las unidades albergan estos objetos, y el shader aplica el código GLSL a la unidad de textura indicada mediante un “sampler”, que actúa como una interfaz permitiendo al shader acceder a las texturas eficientemente. El shader se comunica con las texturas a través de un “sampler uniform”, determinando a qué unidad de textura se accede.

Cabe decir que los pasos abarcados para implementar texturas en OpenGL fueron desde cargar las imágenes hasta configurar filtros de minificación y magnificación (que afectan a la representación visual cuando los triángulos o polígonos de las mallas se alejan

o acercan a la cámara), anisotrópicos y activar unidades de textura; todo ello se contempla en los métodos de inicialización de estas clases y en partes eventuales en las que es necesario cambiar la unidad de textura asociada a unos objetos u otros (a través del puntero a las funciones base que se utilizan en todo el contexto OpenGL, `mMyOpenGLFunctions`).

2.2.- Interfaces

En este proyecto se trabaja con interfaces para buscar la mayor transparencia posible a la hora de recibir frames desde cualquier tipo de sensor hacia cualquier hilo o proceso de análisis o visualización. De esta manera, se considera más oportuno emplear una superclase que reúna las características y funcionalidades necesarias y suficientes que debe cumplir una clase que se ocupe de recabar y proporcionar frames de manera ordenada y segura desde distintos tipos de sensores. Y de esta forma no solo se agrupan cámaras, láseres, sensores de triangulación… sino también (como en el caso de este trabajo) el simulador implementado y el sistema de archivos para procesamiento offline.

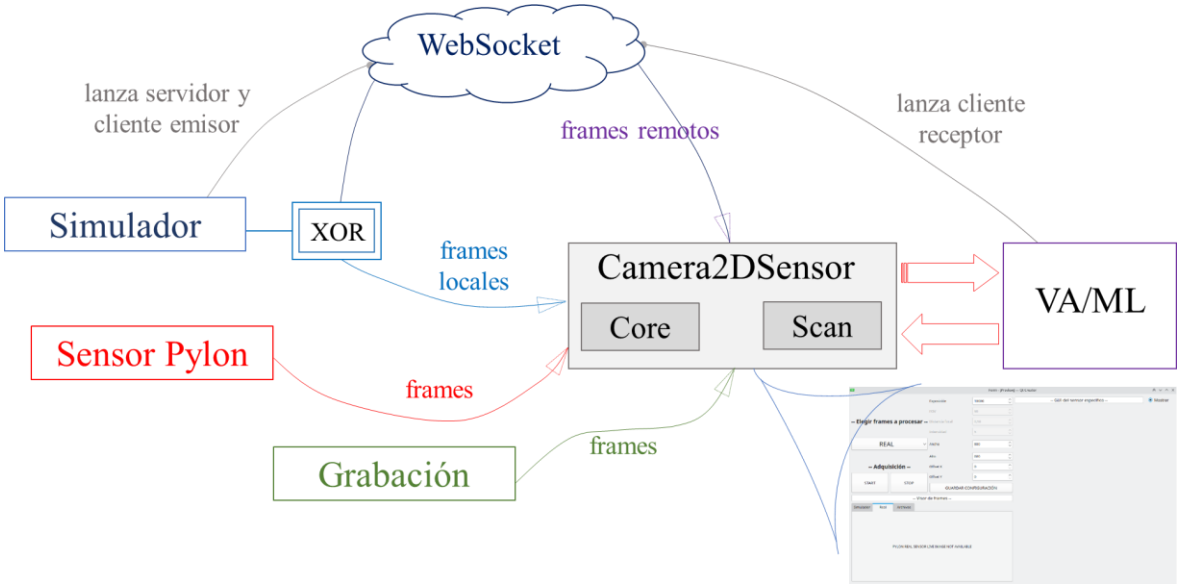


Fig. 36. Las clases implementadas sirven como una interface flexible para aplicaciones de visión artificial, machine learning… que necesiten suplirse de fotogramas de diversas fuentes de información y sensores.

Esta idea parece lógica se si piensa en un sistema informático como un conjunto holístico de sus partes; la parte dedicada a la captura y exportación solo debería preocuparse íntegramente de su labor, al igual que pasa con la parte dedicada al procesamiento de los datos, la parte destinada a la visualización de información, etc. Más aún, esta superclase (`SensorInterface`) (o más específicamente, su núcleo o core) se dedica a la captura de datos (denominados "scans") de cualquier tipo de fuente de scans de los que provengan, siendo transparente de dónde provenga; de esta manera, se puede notar también que de esta forma los flujos de entrada y la posición de la clase son arbitrarios, según el programador considere conveniente. La clase puede contar con una (ella misma) o más interfaces, manejar uno o más tipos de scan y con uno o más de un tipo de interfaz a la vez.

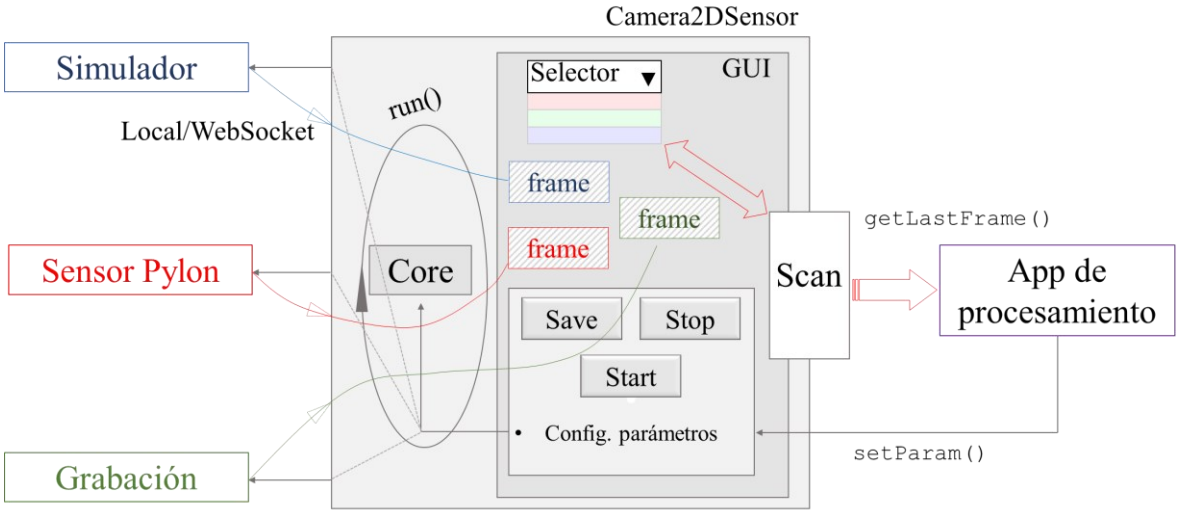


Fig. 37. Diagrama que resume los flujos de información y la logística del sistema de interfaces implementado.

Luego, cada clase derivada que herede de esta superclase puede contener, o no, su parte dedicada a la visualización e interacción con el usuario para que este pueda modificar o parámetros del sensor. Y esto se "adheriría" flexiblemente sin afectar al bucle de captura de scans por parte del núcleo de la clase, en un hilo separado. Así, este bloque que «proporciona» frames (quienes los «generan» son realmente la cámara, el simulador, los archivos locales…) ligada a cualquier otro programa de procesamiento (sea la aplicación `conveyorBeltProcessingTFM` de este proyecto) y así, en cualquier momento, se puede elegir usar el sensor que se quiera.

Esto también conlleva poder crear una superclase (`ScanInterface`) tan general como la mencionada pero que se dedique al manejo de todos los tipos de scans posibles y sus atributos, como pueden ser los datos "crudos" en sí mismos (formato, dimensiones, tamaño…), el nombre o tipo que los identifica, la ruta en donde se guardan, la fecha en que se crearon, etc.

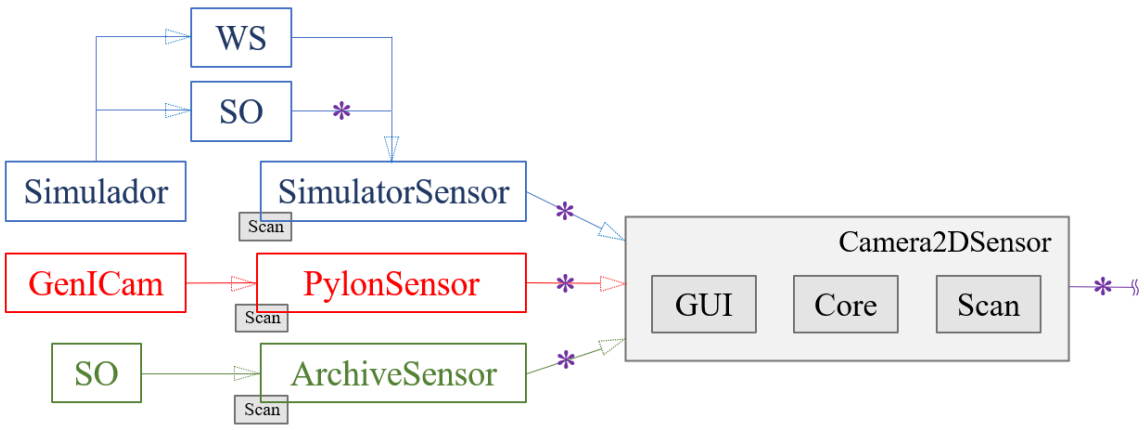


Fig. 38. Se hace evidente que con el sistema así estructurado se podría situar en cualquiera de las partes indicadas con un asterisco (*) un programa idéntico en funcionalidad y alcance que el desarrollado para al algoritmo de procesamiento, sin cambiar en absoluto el funcionamiento de cara al exterior; esta es la flexibilidad que otorgan las interfaces y sus clases.

Con estos fines, para este trabajo se han integrado las librerías `PylonSensor` (dedicada a la captura de frames de cámaras bajo el estándar GenICam, como la de Basler que se utiliza aquí), `SimulatorSensor` (dedicada a la captura de frames del simulador realizado), `ArchiveSensor` (dedicada a la captura de frames del sistema de archivos local de la máquina), `Camera2DSensor` (que agrupa modularmente las funcionalidades de todas las anteriores para acceder flexiblemente a cada tipo deseado) y `Camera2DScan` (que actúa a modo de plugin para las clases anteriores pues proporciona las funciones de manejo de scans especificados para frames provenientes de cámaras bidimensionales o similares).

2.2.1.- Sensor de cámara real

La librería `PylonSensor` está diseñada para interactuar con cámaras basadas en estándares GenICam a través de la biblioteca Pylon [38]. De manera general, utiliza OpenCV para procesar y manipular imágenes y presenta una estructura modular y una interfaz de usuario asociada para controlar el sensor y visualizar y exportar los resultados de la adquisición de imágenes.

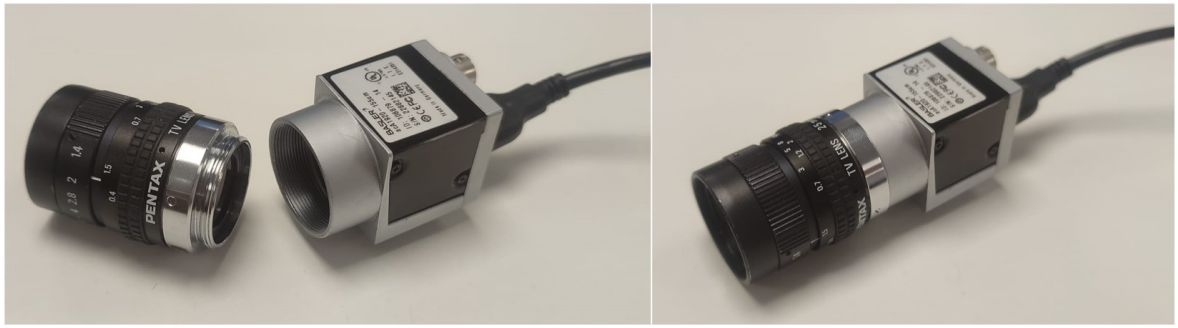


Fig. 39. Cámara y objetivo empleadas en este trabajo.

La cámara es un modelo acA1920-155um de la marca Basler, con una resolución máxima de 1920 x 1200 píxeles y una interfaz USB 3.0. Pertenece a la serie "ace" de cámaras Basler, conocidas por su velocidad ultrarrápida, rentabilidad y calidad de imagen superior en un diseño compacto. La cámara tiene una velocidad de captura de 164 FPS, una temperatura operativa entre 0 y 50 °C y dimensiones de 29.3 mm x 29 mm x 29 mm, con un peso de 80 g.

La lente u objetivo Pentax utilizada es de tipo CCTV, tiene una distancia focal de 25 mm, montura en C (relativo a la conexión mecánica y eléctrica de la lente con la cámara) y apertura máxima de f/1.4 ("1.4" es la relación entre la distancia focal y el diámetro efectivo de la apertura), con un ángulo de visión de 19,6 º y una distancia óptima (o típica) al objeto de 30 cm.

La clase `PylonCameraCore` es el núcleo de la interacción con la cámara, donde se utiliza la biblioteca de Pylon. Se emplean métodos para descubrir el número de cámaras disponibles (conectadas a la máquina) en el momento de inicializar el sensor accediendo a la capa de transporte de la aplicación, comparar el número de serie, nombre o ID de cada cámara con el indicado usando la factoría[12] `Pylon::CTlFactory`, parametrizar las instancias de cada dispositivo y acceder mediante punteros a estos, cargar una configuración por defecto de la cámara desde archivos locales de texto plano, iniciar, parar,

[12] Factoría: en el ámbito de la programación orientada a objetos, se refiere a un patrón de diseño creacional (que incrementa la flexibilidad y la reutilización del código existente) utilizado para proporcionar una interfaz para la instanciar clases sin necesidad de conocerla (sin llamar directamente al constructor), de tal forma que una clase base declara una interfaz que crea un objeto pero permite que las subclases lo alteren; de esta forma no es necesario transferir el puntero apuntando a una clase a otras, sino que con llamar a este método se obtiene acceso al control del objeto de interés.

finalizar o esperar por la adquisición de imágenes por parte del sensor (manejándola en un hilo distinto), conversión de los scans a un formato conveniente como `cv::Mat` o `QImage`, almacenar las imágenes en un búfer dinámico y enviar señales que indican el inicio y la finalización de la adquisición o el estado del sensor, proporcionar un número arbitrario de frames al exterior o el último fotograma que se haya capturado, limpiar el búfer o eliminar el último frame en el bucle de adquisición para liberar la RAM en cada iteración, etc.

La clase `PylonSensor` es la capa de abstracción más alta de la librería; utiliza a `PylonCameraCore` y le agrega funcionalidades adicionales, como la posibilidad de cargar configuraciones desde un archivo XML y gestionar escaneos mediante `ScanInterface`. Esta es la clase que el usuario o el programador de otra aplicación manejaría desde el exterior; es la que la `Camera2DScan` y la aplicación principal de procesamiento emplea para manejar libremente el sensor real.

Por último, la clase `PylonSensorUI` representa una interfaz de usuario para interactuar con este sensor. Utiliza un conjunto de elementos de GUI de Qt y funciones para conectarse y desconectarse del sensor, iniciar y detener la adquisición, mostrar información sobre el sensor, modificar sus parámetros, etc.

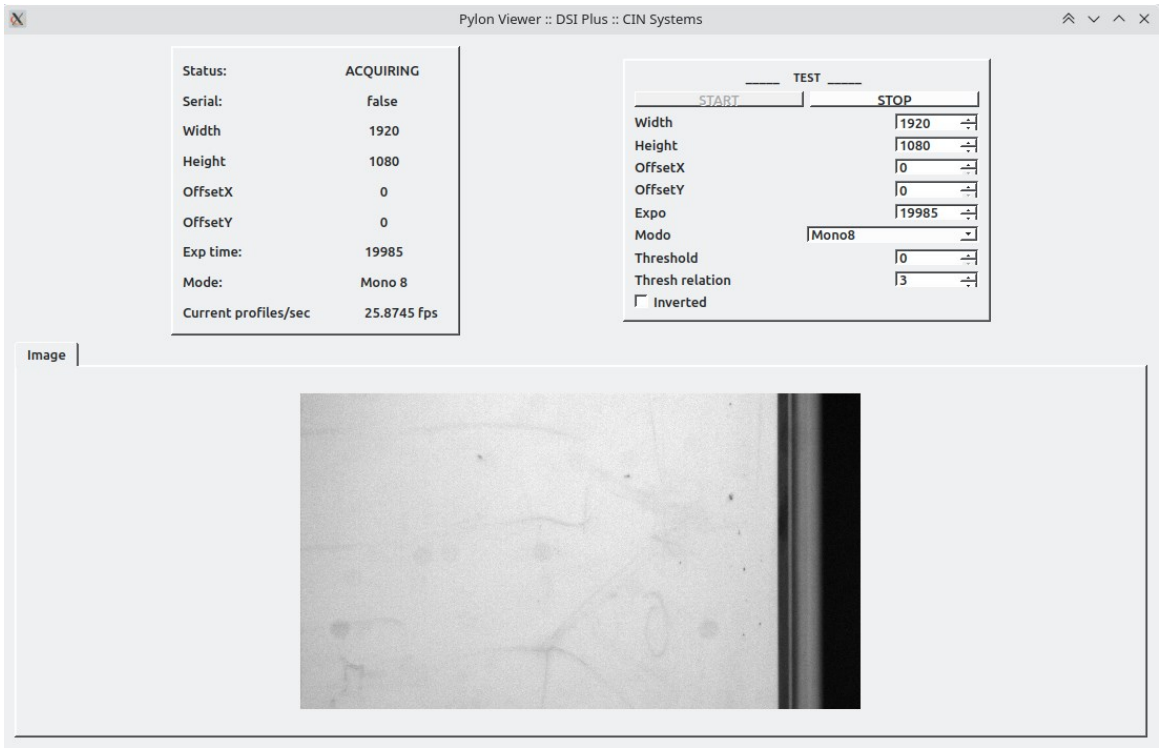


Fig. 40. GUI del sensor de cámara real.

El tipo, nombre e ID del sensor son útiles para diferenciarlo si se emplea en una aplicación (como la de `Camera2DSensor`) con más de un tipo de dispositivo. A través de la etiqueta "Serial" se indica el número de serie de la cámara utilizada, mandatorio para

localizar y manipular la cámara que debe reconocerse desde la capa de transporte[13] del sistema, en "SettingsFile" se especifica la ruta completa para parametrizar a voluntad el estado inicial de la cámara, a través de un archivo de texto plano que debe contener el nombre y valor de aquellos parámetros concretos que se desean definir (exposición[14], ancho y alto del plano imagen, formato de los píxeles, offset que agrega la FPGA de la cámara…). Por último, mediante la etiqueta "ScanPlugins" se indican las librerías precompiladas (disponibles en una ruta de inclusión que debe conocer el programa, típicamente definida a través del archivo .pro) que se utilizan para manejar los escaneos del sensor.

```
<?xml version="1.0" encoding="UTF-8"?>
<Sensor>
      <Type>PylonSensor</Type>
      <Name>Pylon Sensor</Name>
      <Id>0</Id>
      <Serial>22882145</Serial>
      <SettingsFile>/home/alejandro/Desktop/proyectos/dsi/sensor/cameras/pylon/config_example/periscope.pfs</SettingsFile>
      <ScanPlugins>
        <Scan>DsiCamera2DScan</Scan>
      </ScanPlugins>
</Sensor>
```

Fig. 41. Archivo de configuración de la cámara real.

Además, se usa un archivo de texto plano (ver la última figura de este apartado) a través del cual se especifican varios parámetros de interés de la cámara real, que indiferentemente se podrían modificar mediante su interfaz pero puede resultar eficiente definirlos fácilmente de esta forma al inicio de la aplicación. Estos parámetros están relacionados con el árbol de nodos específico del modelo de cámara Basler empleado y resultó necesario indagar someramente en la documentación de Pylon y en su API para consultar qué parámetros brindaban la posibilidad de personalizar las librerías del fabricante.

En la siguiente figura se puede ver la GUI de este sensor integrada en la del sensor de carácter general Camera2DSensor, monitorizando una pieza real en tiempo real:

[13] La capa de transporte es el cuarto nivel del modelo OSI [*Open Systems Interconnection*] de ISO [*International Organization for Standarization*]; es responsable de la transmisión de datos a través de la red, regulando el flujo de información desde el origen hasta el destino y garantiza que dos sistemas puedan comunicarse entre sí mediante una transmisión de datos segura, fluida y transparente de extremo a extremo.

[14] La exposición se refiere al control del tiempo durante el cual la luz incide sobre el sensor de la cámara penetrando en su sensor óptico, que determina la cantidad de luz capturada en una imagen o fotograma (cantidad de electrones en que se convierte la luz que incide en los "pozos de potencial" que son los píxeles). De este parámetro también depende la velocidad máxima de adquisición, pues a más tiempo de exposición, menos velocidad de adquisición máxima.

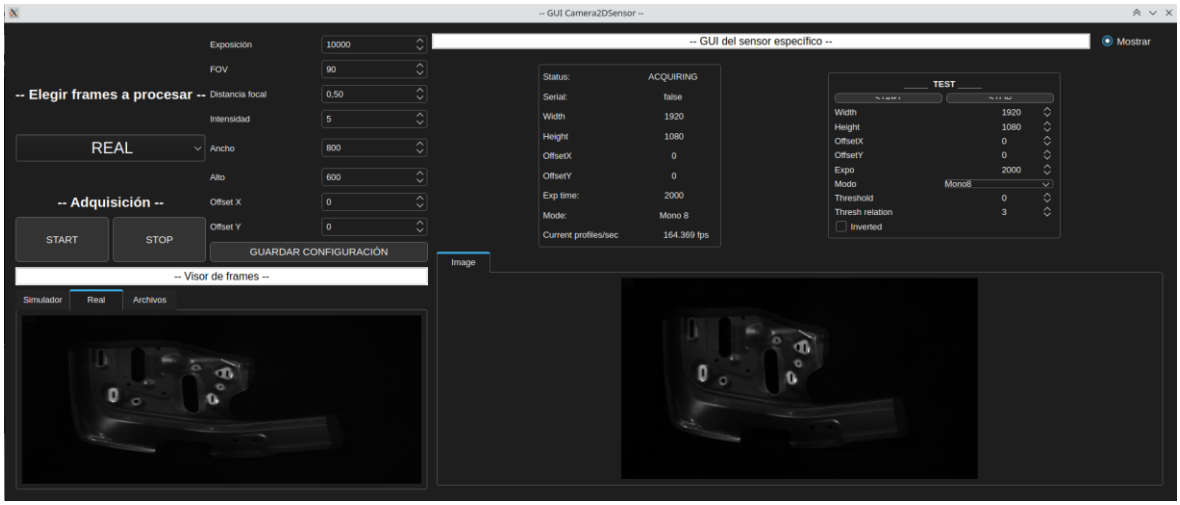


Fig. 42. GUI del sensor de cámara real (izda.) en funcionamiento junto con la GUI de Camera2DSensor (dcha.) (ver apartado "Sensor de cámara 2D general").

```
# {05D8C294-F295-4dfb-9D01-096BD04049F4}
# GenApi persistence file (version 3.1.0)
# Device = Basler::UsbCameraParams -- Basler USB3Vision camera interface --
# Device version = 1.0.0 -- Product GUID = 5a809e15-f447-480f-a21f-0fb7256b8400 --
# Product version GUID = B0C5F330-6DB8-4BB7-8CDD-82B5AEB2F8D6
ExposureAuto    Off
GainAuto        Off
Width   1920
Height  1080
OffsetX 0
OffsetY 0
CenterX 0
CenterY 0
BinningHorizontalMode   Sum
BinningHorizontal       1
BinningVerticalMode     Sum
BinningVertical 1
ReverseX        0
ReverseY        0
PixelFormat     Mono8
ShutterMode     Global
ExposureMode    Timed
ExposureTime    20000.0
```

Fig. 43. Archivo de parámetros de la cámara real.

2.2.2.- Sensor de cámara virtual

La librería `SimulatorSensor` está diseñada para interactuar con sensores simulados a través una comunicación local o remota. En términos generales, utiliza OpenCV para procesar y manipular imágenes, presentando una estructura modular y una interfaz de usuario asociada para controlar el sensor, así como para visualizar y exportar los resultados de la adquisición de imágenes.

La clase `SimulatorSensorCore` es el núcleo de la librería; deriva de `QThread` (por lo que se ejecuta enteramente en un hilo de ejecución secundario al hilo que invoque las funcionalidades del objeto) y gestiona a más bajo nivel la adquisición de datos del sensor, ya sea a través de un directorio local al detectar cambios en archivos de escaneo o conexión mediante mensajes WebSocket. La configuración del simulador se especifica mediante la estructura `SimulatorConfig`, el hilo principal de ejecución (`run`) monitoriza continuamente la adquisición de datos y también se implementa la clase `SimulatorSensorWebSocketClient` (mediante bibliotecas de Qt) para manejar la comunicación remota y la recepción de datos; otros métodos como `startCamera` y `stopCamera`, como en los otros sensores, controlan el inicio y detención de la adquisición de datos. La implementación detallada incluye la gestión de mutex y semáforos para garantizar la sincronización en la manipulación del buffer de frames; además, resulta interesante destacar funciones como `onDirectoryChanged` y `onWebSocketChanged`, que se ocupan de procesar los datos locales o remotos, respectivamente, por orden de las señales pertinentes.

La clase `SimlatorSensor` es la capa de abstracción más alta de la librería; se encarga de preparar y gestionar la comunicación básica con el sensor homónimo, cargando la configuración desde un archivo XML y proporcionando métodos para iniciar, detener y forzar la detención de la adquisición de imágenes. Además, implementa la lógica del bucle de adquisición que interactúa con el núcleo del simulador (`SimulatorSensorCore`) y crea instancias de escaneo (`ScanInterface`) para procesar y almacenar los fotogramas capturados. La configuración del sensor se carga desde el archivo XML y la adquisición se realiza cíclicamente, gestionando la captura de imágenes de simulación de manera segura. Esta es la clase que el usuario o el programador de otra aplicación manejaría desde el exterior; es la que la `Camera2DScan` y la aplicación principal de procesamiento emplea para manejar libremente el sensor real.

Por último, la clase `SimulatorSensorUI` representa una interfaz de usuario para interactuar con este sensor. Utiliza diversos elementos del módulo gráfico de Qt y funciones para conectarse y desconectarse del sensor, iniciar y detener la adquisición, mostrar información sobre el sensor, modificar sus parámetros, etc. Esta clase hereda de `QWidget` y se encarga de gestionar (mediante eventos) la visualización en tiempo real de los frames capturados por el sensor simulado. Incluye botones para iniciar y detener la adquisición, así como un objeto `QLabel` para mostrar los frames capturados más recientes mediante temporizadores. Entre otros, el método `mostrarFrame` convierte y muestra un

frame en la interfaz gráfica y los métodos `onConnect` y `onDisconnect` actualizan dinámicamente el estado de los botones y habilitan o deshabilitan la interfaz según el estado de la adquisición.

Fig. 44. GUI del sensor de cámara virtual.

El tipo, nombre e ID del sensor son útiles para diferenciarlo si se emplea en una aplicación (como la de `Camera2DSensor`) con más de un tipo de dispositivo. A través de la etiqueta "SettingsFile" se especifica la ruta completa para parametrizar a voluntad el estado inicial de la cámara, a través de un archivo de texto plano que debe contener el nombre y valor de aquellos parámetros concretos que se desean definir (exposición, ancho y alto del plano imagen, FOV, distancia focal…). Mediante la etiqueta "ScanPlugins" se indican las librerías precompiladas (disponibles en una ruta de inclusión que debe conocer el programa, típicamente definida a través del archivo .pro) que se utilizan para manejar los escaneos del sensor. "FolderName" especifica el directorio en el que, inicialmente, el sensor debería buscar los escaneos locales para leerlos, "Scans" es una etiqueta inutilizada que serviría para indicar cuántos frames de los disponibles en cada ciclo de lectura se quieren mantener (esto ya se controla mediante la tasa de escritura desde el simulador en función de los FPS elegidos), "Local" es un booleano que indica si la lectura de escaneos se quiere realizar a través de la senda local (vía sistema local de archivos) o remota (vía WebSocket) e "IP" y "Port" indican la dirección y el puerto del protocolo de Internet asociado al cliente WebSocket que se conectaría al servidor en que estarían disponibles los frames en tiempo real si el booleano previo no fuera cierto.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Sensor>
    <SimulatorSensor>
            <Type>DsiSimulatorSensor</Type>
            <Name>SimulatorSensor</Name>
            <Id>1</Id>>
            <SettingsFile>/home/alejandro/Desktop/proyectos/dsi/sensor/simulator/config_example/simulator.params</SettingsFile>
            <ScanPlugins>
              <Scan>DsiCamera2DScan</Scan>
            </ScanPlugins>
            <FolderName>/home/alejandro/Desktop/</FolderName>
            <Scans></Scans>
            <Local>true</Local>
            <IP>127.0.0.1</IP>
            <Port>8080</Port>
    </SimulatorSensor>
<Sensor>
```

Fig. 45. Archivo de configuración de la cámara virtual.

```
# Cambiar parámetros iniciales del simulador desde la interfaz
Width   1920
Height  1080
FOV 90
FocalLength 1.0
Exposure        10000
```

Fig. 46. Archivo de parámetros de la cámara virtual, como posible ampliación.

Se plantea el uso de un archivo de texto plano (figura anterior), análogo al caso real, a través del cual se especifican varios parámetros de interés de la cámara virtual del simulador, que indiferentemente se podrían modificar mediante su interfaz pero puede resultar eficiente definirlos fácilmente de esta forma al inicio de la aplicación.

2.2.3.- Sensor de archivo (cámara offline)

La librería `ArhiveSensor` está diseñada para interactuar con escaneos de cámaras en modo fuera de línea a través del sistema de archivos local. De manera general, utiliza OpenCV leer las imágenes y diferentes librerías de Qt proporcionando una interfaz para cargar y procesar frames desde fotogramas o vídeos locales.

La clase `ArchiveSensor` compendia el núcleo de la interacción con la cámara y la capa de abstracción más alta de la librería; da la posibilidad de cargar configuraciones desde un archivo XML y gestionar escaneos mediante `ScanInterface`, la elección de un directorio particular del que leer los datos y gestión del ciclo de vida del sensor, el registro de eventos, etc. Esta es la clase que el usuario o el programador de otra aplicación manejaría desde el exterior; es la que la `Camera2DScan` y la aplicación principal de procesamiento emplea para manejar libremente el sensor real.

La adquisición de frames se gestiona mediante el bucle run, que es la rutina que se ejecuta en el hilo afín a la instancia de esta clase. Cuando se inicia la adquisición con `startAcquisition`, se monitoriza un directorio especificado para leer archivos de imágenes, convirtiéndolos en frames. El bucle de adquisición continua mientras la adquisición está activa, y cada iteración ejecuta la función `onCoreDirectoryMonitoring`, que carga la imagen actual del directorio, la convierte en un objeto `CameraFrame`, la añade al buffer actual y actualiza la interfaz gráfica. La detención de la adquisición con `stopAcquisition`, que finaliza la adquisición, liberando los recursos dinámicos correspondientes. Como el resto de sensores, este ejecuta en un hilo separado, permitiendo la adquisición continua de frames desde archivos locales en respuesta a cambios en el directorio de imágenes provenientes del apartado gráfico.

Por último, la clase `ArchiveSensorUI` representa una interfaz de usuario para interactuar con este sensor. Utiliza un conjunto de elementos de GUI de Qt y funciones para conectarse y desconectarse del sensor, iniciar y detener la adquisición, mostrar información sobre el sensor, modificar sus parámetros, etc. La interfaz también incluye la capacidad de configurar directorios para la adquisición de frames y visualizar en tiempo real los frames capturados por el sensor. La clase `DirectoryConfigDialog` proporciona una ventana de diálogo para configurar el directorio de vídeo y ajustar otras opciones relacionadas con la adquisición de frames.

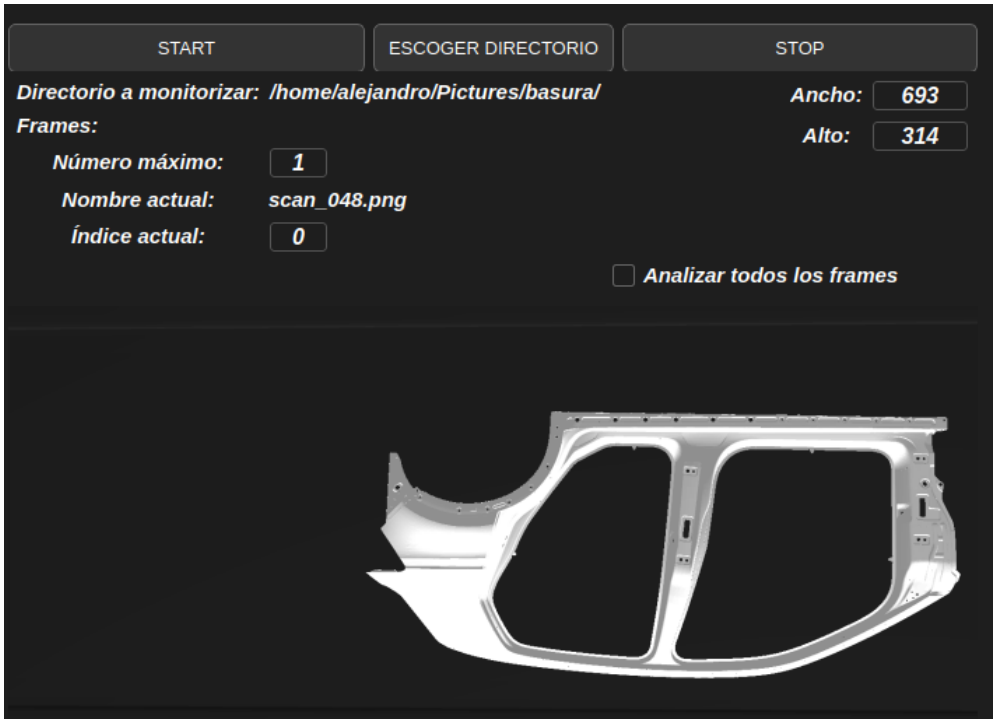


Fig. 47. GUI del sensor de archivo.

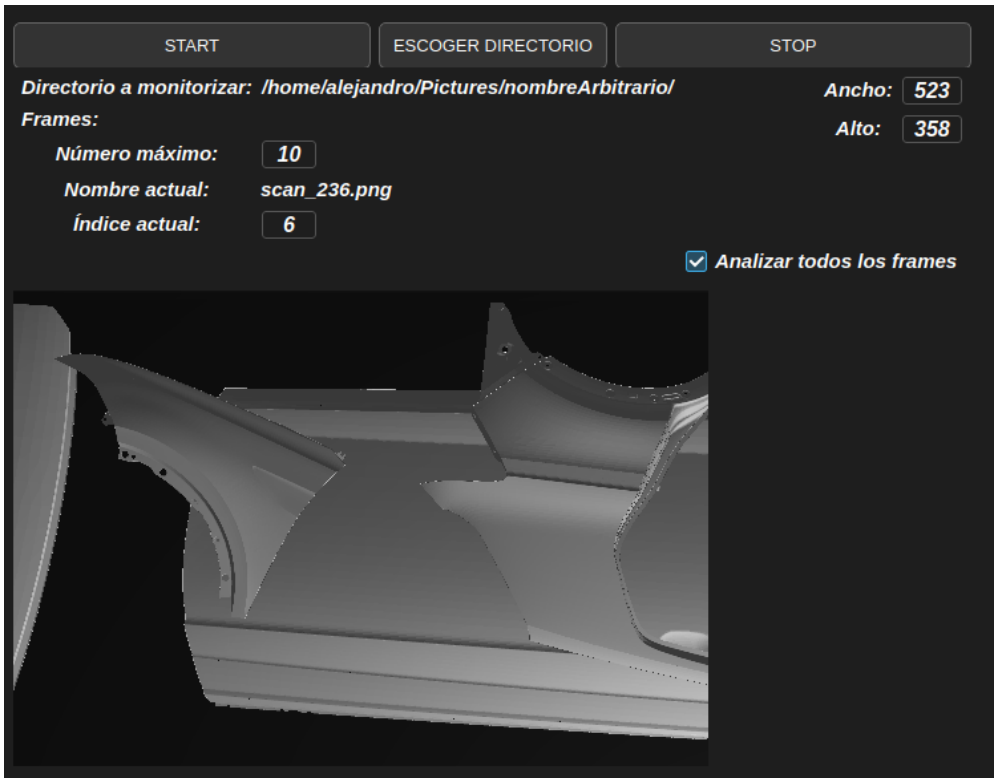


Fig. 48. Otra captura de la GUI del sensor de archivo, con distinta configuración XML que la anterior.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Sensor>
    <ArchiveSensor>
        <Type>DsiArchiveSensor</Type>
        <Name>ArchiveSensor</Name>
        <Id>2</Id>>
        <ScanPlugins>
          <Scan>DsiCamera2DScan</Scan>
        </ScanPlugins>
        <FolderName>/home/alejandro/Desktop/folderBasura</FolderName>
        <AllScans>true</AllScans>
        <ScansNumber>3</ScansNumber>
    </ArchiveSensor>
</Sensor>
```

Fig. 49. Archivo de configuración de archivo.

El tipo, nombre e ID del sensor son útiles para diferenciarlo si se emplea en una aplicación (como la de `Camera2DSensor`) con más de un tipo de dispositivo. Mediante la etiqueta "ScanPlugins" se indican las librerías precompiladas (disponibles en una ruta de inclusión que debe conocer el programa, típicamente definida a través del archivo .pro) que se utilizan para manejar los escaneos del sensor. "FolderName" especifica el directorio en el que, inicialmente, el sensor debe buscar los escaneos locales para leerlos, "AllScans" es un booleano mediante el que se indica si se desean capturar absolutamente todos los frames que haya disponibles en la ruta indicada y "ScansNumber" alberga un guarismo que representa un número entero mayor que cero correspondiente al número de escaneos concreto que se desean leer de la ruta, cíclicamente (en caso de que la etiqueta previa no sea cierta y el número indicado sea menor que el máximo de frames existentes en el directorio).

2.2.4.- Sensor de cámara 2D general

La librería `Camera2DSensor` está diseñada para interactuar con cámaras de cualquier índole como las explicadas anteriormente; es una agregación de sensores muy conveniente para desarrollar algoritmos dentro del marco establecido por el presente trabajo. La clase `Camera2DSensor` gestiona tareas cruciales como la inicialización y cierre del sensor elegido o la adquisición y configuración de sus parámetros. El hecho de que se decidiera que esta clase también heredara de `SensorInterface` constituye una ventaja de cara a su manejo desde el exterior porque mantiene exactamente las funcionalidades de sus clases o sensores componentes y, en cualquier momento, solo hay que redefinir el sensor activo que se desea utilizar para que su gestión sea absolutamente transparente desde fuera.

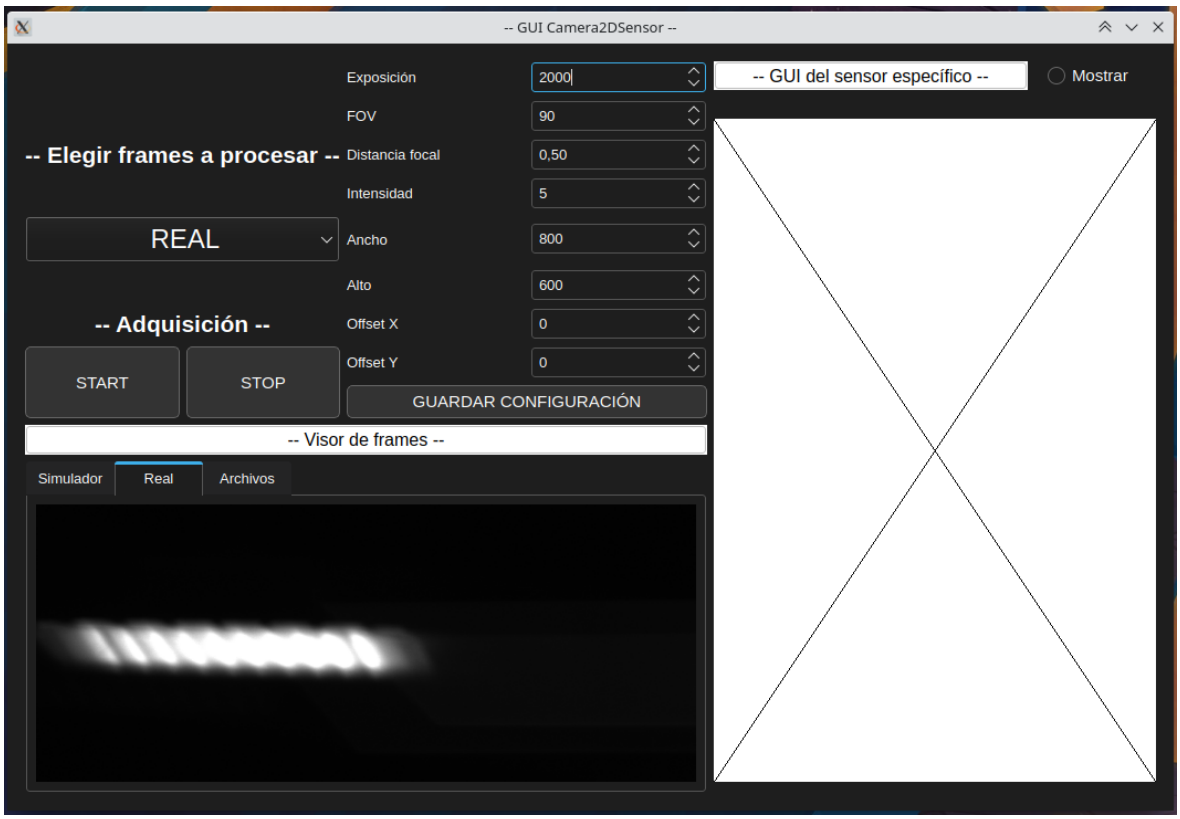


Fig. 50. GUI de Camera2DSensor simplificada (deshabilitada la opción de mostrar la GUI del sensor específico).

En el código fuente `Camera2DSensor` se implementa toda la funcionalidad del esta entidad; primeramente, con la función `NewSensor`, se crea una instancia del sensor y se inicializa con la fecha y hora actual (también registra un mensaje en el sistema de registro indicando la compilación correcta de la clase), luego se inicializa en constructor, donde se configuran diversos parámetros y se asigna el tipo de sensor activo por defecto y se implementan funciones para cargar la configuración desde un archivo XML, inicializar el sensor seleccionado, gestionar la adquisición de datos, cambiar el sensor activo y cargar configuraciones específicas a cada sensor. El método `run` representa el hilo de adquisición del sensor; en este bucle (que se mantiene mientras el sensor esté adquiriendo y no se haya solicitado detener la adquisición desde fuera) se obtiene el último escaneo del sensor activo

y se verifica su integridad. La función `changeActiveSensor` se encarga de cambiar el sensor activo, deteniendo previamente el sensor actual si estuviera en modo de adquisición. Asimismo, la función `loadSettingsToSensors` ajusta los parámetros del sensor activo, como la exposición, ancho, alto, desplazamientos verticales u horizontales de la vista… según corresponda.

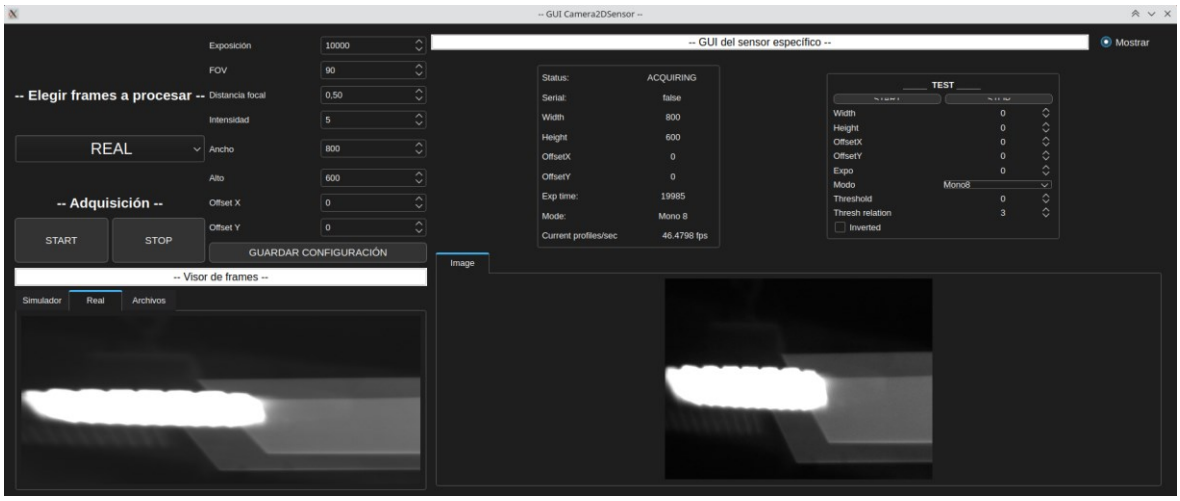


Fig. 51. GUI de Camera2DSensor en funcionamiento con la cámara real.

Por último, la clase `Camera2DSensorUI` representa una interfaz de usuario para interactuar con cualquiera de los tres tipos de sensor. Utiliza un conjunto de elementos del módulo gráfico de Qt y funciones para conectarse y desconectarse del sensor, iniciar y detener la adquisición, mostrar información sobre el sensor, modificar sus parámetros, etc. Además, mediante un elemento `QComboBox` permite elegir fácilmente, en cualquier momento (e incluso en tiempo de ejecución), el sensor que se desea monitorizar, para lo cual carga su GUI específica en una región apartada de la ventana dando la posibilidad de esconderla o mostrarla a voluntad y muestra la vista en tiempo real de la cámara de interés, con todos los parámetros de su interfaz actualizada concurrentemente.

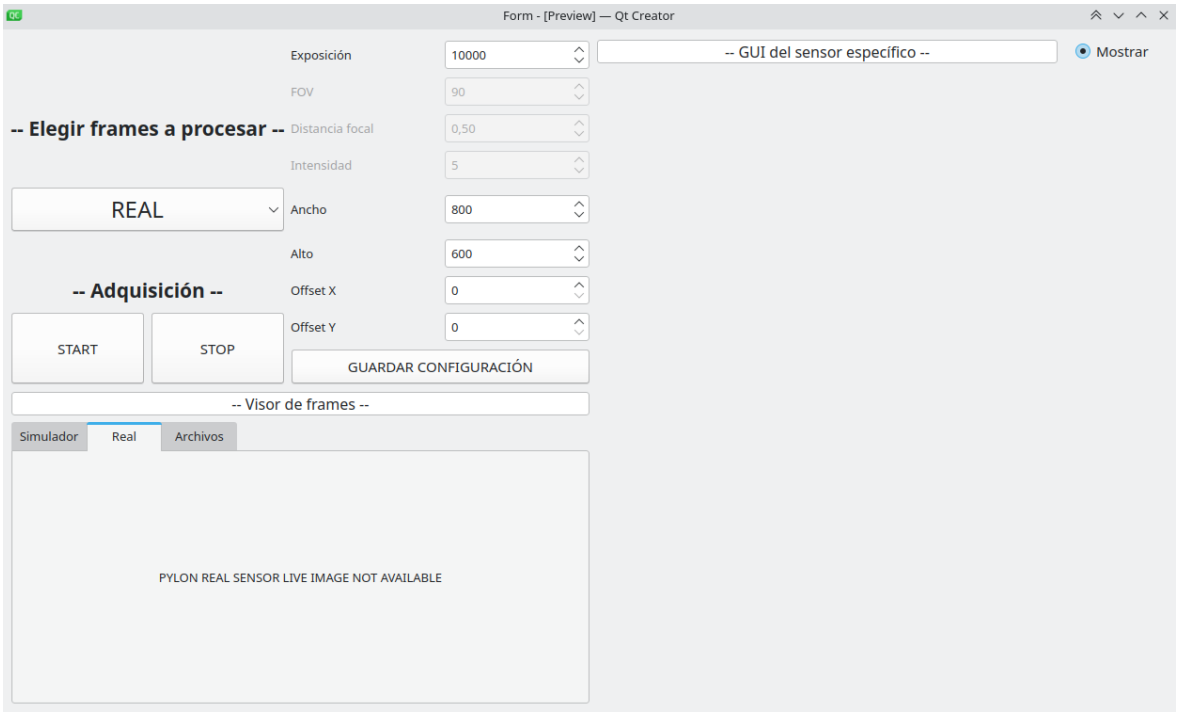


Fig. 52. GUI del sensor de cámara 2D general en modo de previsualización.

El tipo, nombre e ID del sensor son útiles para diferenciarlo si se emplea en una aplicación con más de un tipo de dispositivo. En la etiqueta “DefaultSensor” se especifica el tipo de sensor de cámara 2D, de entre los anteriormente explicados, que se desea manejar al inicio de la aplicación (posteriormente se podría cambiar en tiempo de ejecución en cualquier instante, así como pararlo o reiniciarlo). A través de “SettingsFile” se especifica la ruta completa para parametrizar a voluntad el estado inicial de la cámara, a través de un archivo de texto plano que debe contener el nombre y valor de aquellos parámetros concretos que se desean definir (exposición, ancho y alto del plano imagen, FOV, distancia focal, offsets…). Mediante la etiqueta “ScanPlugins” se indican las librerías precompiladas (disponibles en una ruta de inclusión que debe conocer el programa, típicamente definida a través del archivo .pro) que se utilizan para manejar los escaneos del sensor. El resto de etiquetas son exactamente las mismas que se explicaron para los anteriores sensores y cumplen su misma función, agrupadas de manera similar como estaban organizados los XML usados para el lanzamiento individual de cada uno, por comodidad.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Sensor> <!-- Se mantiene "Sensor" como elemento raíz == Pylon por defecto para no modificar la librería PylonSensor -->
    <DefaultSensor>Archive</DefaultSensor> <!-- Pylon, Simulator, Archive -->
    <Type>PylonSensor</Type>
    <Name>Pylon Sensor</Name>
    <Id>1</Id>
    <Serial>22565624</Serial>
    <SettingsFile>/home/alejandro/Desktop/proyectos/dsi/sensor/cameras/camera2d/config_example/camera2dsensor.params</SettingsFile>
    <ScanPlugins>
      <Scan>DsiCamera2DScan</Scan>
    </ScanPlugins>
    <Camera2dSensor>
            <Type>DsiCamera2DSensor</Type>
            <Name>Camera2DSensor</Name>
            <Id>0</Id>
            <SettingsFile></SettingsFile>
            <ScanPlugins>
              <Scan>DsiCamera2DScan</Scan>
            </ScanPlugins>
            <FolderName></FolderName>
            <Scans></Scans>
    </Camera2dSensor>
    <SimulatorSensor>
            <Type>DsiSimulatorSensor</Type>
            <Name>SimulatorSensor</Name>
            <Id>1</Id>>
            <SettingsFile>/home/alejandro/Desktop/proyectos/dsi/sensor/simulator/config_example/simulator.params</SettingsFile>
            <ScanPlugins>
              <Scan>DsiCamera2DScan</Scan>
            </ScanPlugins>
            <FolderName>/home/alejandro/Desktop/</FolderName>
            <Scans></Scans>
            <Local>true</Local> <!-- 0, 1, true, false, True, False, TRUE, FALSE... -->
            <IP>127.0.0.1</IP> <!-- Solo se usa si Local es false -->
            <Port>8080</Port> <!-- Solo se usa si Local es false -->
    </SimulatorSensor>
    <ArchiveSensor>
        <Type>DsiArchiveSensor</Type>
        <Name>ArchiveSensor</Name>
        <Id>2</Id>
        <ScanPlugins>
          <Scan>DsiCamera2DScan</Scan>
        </ScanPlugins>
        <FolderName>/home/alejandro/Pictures/nombreArbitrario/</FolderName>
        <AllScans>true</AllScans>
        <ScansNumber></ScansNumber> <!-- Solo se usa si AllScans es false -->
    </ArchiveSensor>
</Sensor>
```

Fig. 53. Archivo de configuración de la cámara 2D general.

El fichero posteriormente mostrado (“app_config.xml”) es un archivo de configuración en formato XML que contiene parámetros específicos para la aplicación (en general) que lanza cualquier tipo de sensor. Este contiene la ruta del archivo de configuración del mismo, la ubicación de los plugins utilizados (y algunas librerías específicas de la empresa de las que hace uso) y otro conjunto de especificaciones relativas al nivel de detalle de los mensajes de registro, su localización, resolución temporal, etc. de menor interés.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <General>
      <Sensor>/home/alejandro/Desktop/proyectos/dsi/sensor/cameras/camera2d/config_example/camera2dsensor.xml</Sensor>
      <Plugin_Path>/usr/local/lib/dsi/debug</Plugin_Path>
    </General>
    <Log level="7" lazy_size="1">
        <Log_Path>/home/alejandro/Desktop/proyectos/dsi/logs/</Log_Path>
        <File level="4" general_name="">
            <Daily active="1" older="7"></Daily>
        </File>
        <Console active="1" level="5"></Console>
        <Callback active="1" level="4"></Callback>
        <Socket active="0" level="2"></Socket>
        <Interface active="0" view_length="100"></Interface>
        <Time_Resolution>ms</Time_Resolution>
    </Log>
</Configuration>
```

Fig. 54. Archivo de configuración de la aplicación, en general.

```
# {05D8C294-F295-4dfb-9D01-096BD04049F4}
# GenApi persistence file (version 3.1.0)
# Device = Basler::UsbCameraParams -- Basler USB3Vision camera interface --
Device version = 1.0.0 -- Product GUID = 5a809e15-f447-480f-a21f-0fb7256b8400 --
Product version GUID = B0C5F330-6DB8-4BB7-8CDD-82B5AEB2F8D6
ExposureAuto    Off
GainAuto        Off
Width   800
Height  600
OffsetX 0
OffsetY 0
CenterX 0
CenterY 0
BinningHorizontalMode   Sum
BinningHorizontal       1
BinningVerticalMode     Sum
BinningVertical 1
ReverseX        0
ReverseY        0
PixelFormat     Mono8
ShutterMode     Global
ExposureMode    Timed
ExposureTime    20000.0
# Para poder cambiar parámetros del simulador desde la interfaz
# FOV 90
# FocalLength 1.0
# Exposure      10000
```

Fig. 55. Archivo de parámetros de la cámara 2D general, como posible ampliación.

Por último, se hace uso de un archivo de texto plano, análogo al caso real y virtual, a través del cual se parametrizan varios atributos de interés de las cámaras, pudiendo combinar los parámetros específicos de cada cual.

2.2.5.- Escaneo de cámara 2D general

Esta librería, denominada `Camera2DScan`, implementa métodos para manejar escaneos de carácter general para fotogramas de cámaras 2D. Se implementan métodos para guardar los datos en formato ".scan" (que internamente contienen la imagen y otros atributos de esta) y en archivos binarios ".raw" separados, cargarlos desde archivos XML, adquirir continuamente frames de la cámara en un bucle de adquisición continua, mantenerse a la espera de cesar la adquisición si se aborta el escaneo, llevar un registro de los últimos frames adquiridos para proporcionarlos cíclicamente,

Esta componente software se entiende como un plugin porque se puede agregar o dinámicamente a las librerías anteriores (o quitar de ellas) para extender su funcionalidad sin modificar el código fuente original y les aporta sus funcionalidades independientemente de núcleos y se cargan en tiempo de ejecución (no en tiempo de ejecución). Además, los plugins permiten actualizaciones "en caliente", sin necesidad de reiniciar la aplicación, lo que puede ser beneficioso para sistemas como este que requieren disponibilidad continua (si las condiciones de la cámara o la forma de la que genera los frames varían, por ejemplo). En cierto momento se podría querer manejar otro tipo de escaneo diferente, como una imagen de profundidad o un perfil de triangulación láser, y bastaría con agregar otro tipo de plugin para que las información que generan los sensores sea compatible con las aplicaciones de procesamiento.

```
52      public:
53          explicit Camera2DScan(QString name = "Undefined");
54          virtual ~Camera2DScan() override;
55
56          bool loadScan(QString path) override;
57          bool saveScan(QString path) override;
58
59          void run() override;
60
61          CameraFramePtr takeNextFrame();
62          CameraFramePtr getLastFrame();
63
64          bool isAborted() override{return false;};
65          bool isReady() override{return false;};
```

Fig. 56. Extracto de los métodos públicos de la clase que maneja todos los escaneos de Camera2DSensor.

De entre los aspectos particulares del código fuente de este plugin se pueden mencionar algunos interesantes: se hace uso de la directiva `#pragma once` para garantizar que el archivo de encabezado solo se incluya una vez durante la compilación, evitando problemas de duplicación (sirve al mismo propósito que la protección de macros usual pero con una menor cantidad de código y sin la posibilidad de que lleguen a coincidir las identificaciones), también se utiliza `extern "C"` para declarar una función `NewScan` que crea y devuelve un objeto de esta clase, lo que facilita la creación de instancias de la clase desde contextos en los que se requiere una interfaz de lenguaje puramente de C. Luego, el archivo se enmarca en un espacio de nombres y se declara la propia clase, que hereda de la interfaz `ScanCameraInterface`. En su constructor, toma un nombre como parámetro y

proporciona implementaciones para cargar y guardar escaneos, así como para iniciar la ejecución de la cámara (`run`). Finalmente, se incluye una sección `typedef` en la que se crea un alias conveniente (`Camera2DScanPtr`) para servir como puntero compartido a esta clase.

2.3.- Algoritmo

El algoritmo del sistema implementado funciona como un clasificador y trata, en última instancia, de determinar si la pose de una pieza dentro de una línea de producción es correcta o no, definiendo su pose como válida o inválida. El algoritmo toma como entradas dos fuentes de datos: una colección estática de imágenes y algunos vectores de características precalculados (base de datos) de las piezas en la línea de producción en su pose correcta y, en segundo lugar, los frames provenientes de una cámara 2D apuntando a la línea de producción por la que se mueven las piezas (actuales).

Una primera etapa del algoritmo consiste en el preprocesamiento de los fotogramas de entrada para reducir el ruido que pudiera haber presente mediante distintos tipos de filtros espaciales. Luego se realiza una umbralización, una segmentación y se aplican varias operaciones morfológicas para obtener una representación binaria fidedigna de la pieza actual. A partir del objeto segmentado, además, se extrae el contorno de la pieza que servirá, a posteriori, para un análisis numérico de su forma.

Paralelamente, una entidad denominada "monitorización del flujo de piezas" (MFP) examina el transcurso de objetos por la cinta transportadora para, en función de si la pieza está entrando, dentro del campo de visión, saliendo…, comandar a las distintas partes del sistema.

Las imágenes (de la pieza actual y de la referencia) en escala de grises, las piezas segmentadas y sus contornos pasan a otra etapa del algoritmo, compuesta por un bloque denominado "clasificador morfológico" (CM), encargado de proporcionar información aprovechable para reconocer la pieza actual, a fin de hacerla corresponder con alguna (de suerte, la correcta, si existe) de las de referencia. Se implementa una serie de algoritmos y métricas a cada una de las señales de entrada para construir y más tarde comparar los vectores de características de referencia y los actuales. El propósito de esta entidad es comprobar si hay alguna pieza de referencia "idéntica" a la actual, en cuyo caso se ejecutaría la siguiente etapa del algoritmo. En caso contrario, se entiende que el objeto que se está viendo no se corresponde con ninguna de las referencias o que, con todo, aunque sí se corresponda, su pose es demasiado obtusa y difiere sobremanera con la que debiera tener para considerarse como válida (p. ej., si el tipo de pieza fuese correcta pero estuviera girada en la espacio 3D en vez del plano de la cinta, lo que no se permite), en cuyo caso el sistema avisaría al exterior mediante una señal de error y se abortaría el algoritmo (se desecharía la pieza de la línea de transporte). Por otro lado, si no hay ninguna referencia idéntica a la actual, se ordenan todas las referencias que sean "suficientemente parecidas" (si no las hay se devuelve la señal de error) por grado de similitud y se ejecuta la siguiente entidad, que ayudará a reconocer definitivamente la pieza actual, es decir, a identificarla. Además de extraer múltiples características e identificar la pieza, el clasificador morfológico devuelve como salidas dos primeras estimaciones de la posición y orientación de la pieza: una mediante PCA y otra mediante ECC, que son algoritmos más adelante detallados y puestos en contexto junto con el resto del sistema.

Cada una de las métricas integradas en el clasificador morfológico funcionan o se pueden ver como "sub-bloques" que se pueden habilitar o deshabilitar a voluntad (del usuario, en etapas de experimentación) para que aporten, o no, su "punto de vista" a la respuesta final del sistema. Esta metodología proactiva de estudio del desempeño de los algoritmos ha resultado potenciar enormemente las fortalezas de cada métrica desarrollada, ayudando a construir poco a poco un sistema cada vez más robusto, estable y especializado en el problema particular a resolver (sin dejar de lado el interés en alcanzar un equilibrio entre esto y cierta capacidad de generalización en cuanto a las distintas casuísticas que pudieran caracterizar al entorno de las piezas monitorizadas).

Así pues, las imágenes en escala de grises pasan a otra etapa del algoritmo, constituida por otra entidad denominada "clasificador característico" (CC), encargada, por una parte, de reconocer definitoriamente la pieza actual y, por otra, de devolver una tercera estimación de su posición y orientación mediante el cálculo y la descomposición de la transformación homográfica que la relaciona con la pieza de referencia, a través de un análisis de puntos característicos mediante su detección, descripción y correspondencia con técnicas especialmente seleccionadas más su refinamiento con diferentes filtros y técnicas que son explicadas posteriormente. Este bloque ejecuta su rutina de procesamiento iterativamente bien hasta que la pieza salga por la derecha de la cinta, se rebase cierto tiempo o límite de memoria de un búfer o se dé la orden de cesar el algoritmo.

A partir de todas las iteraciones del clasificador característico se juntan las tres estimaciones de la pose de la pieza actual, computadas con respecto de la pieza de referencia y se genera una respuesta final basada en ponderaciones y umbrales empíricos que indica si, a ojos del sistema completo, la pose de la pieza válida o no.

Se ha puesto un énfasis austero en emplear e investigar en técnicas de visión artificial clásicas, procesamiento de imagen, matemáticas aplicadas y análisis funcional para el desarrollo de todas las labores relativas a la algoritmia del trabajo, pues se considera que las tres son varias de las ramas del conocimiento científico e ingenieril más fascinantes e interesantes; la indagación en profundidad en estas, adquisición de nuevos conocimientos sobre ellas y confianza en sus útiles ha probado no desmerecer en el transcurso del presente trabajo. A continuación se ve una figura que compendia diagramáticamente la totalidad del algoritmo realizado:

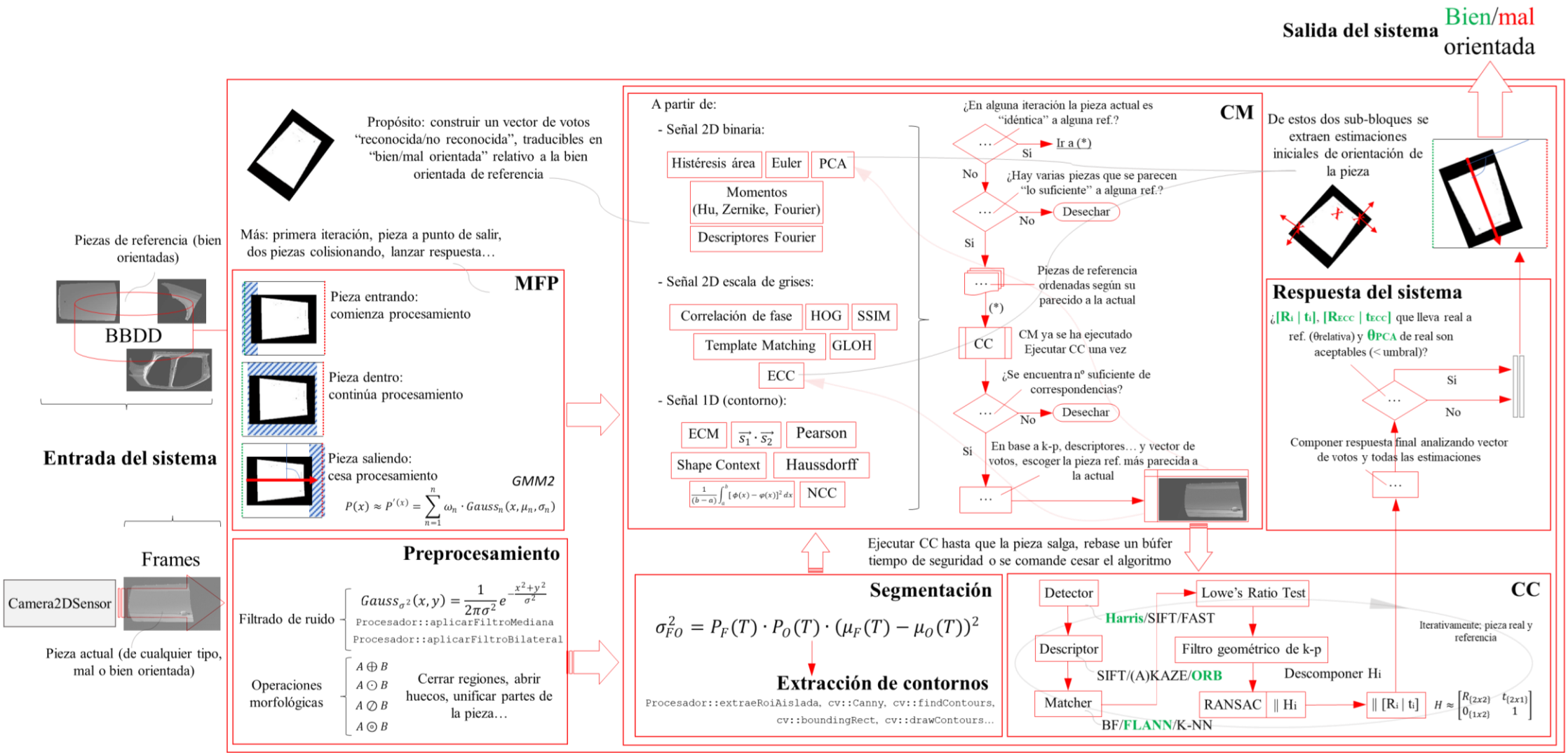


Fig. 57. Esquema general del sistema realizado y algoritmo implementado.

2.3.1.- Extracción de frames

La clase `Core` se encarga de adquirir continuamente frames (fotogramas, *scans*…) de las interfaces de los sensores disponibles, de manera totalmente transparente, para proporcionarlas de manera ininterrumpida y en tiempo a cualquier otra clase que las solicite.

La adquisición y disposición de frames se realiza en un hilo paralelo y de manera mutuamente excluyente. Cuando se ejecuta el programa, el control del sistema operativo se transfiere al inicio de la función `main`, en la que se realizan varias configuraciones primordiales antes de iniciar la interfaz gráfica. Se comienza registrando tipos de datos personalizados (metatipos) para facilitar la comunicación entre hilos, se utiliza `QApplication` para gestionar la aplicación y se carga un archivo de configuración XML que especifica rutas y configuraciones esenciales mediante una factoría estática. La aplicación comprueba la presencia del archivo de configuración en los argumentos de línea de comandos y muestra una advertencia si no se proporciona, luego se inicia un hilo dedicado para el núcleo de la aplicación (`tfm::Core`). Además, se crea una instancia de la ventana principal (`tfm::MainWindow`), se registra un manejador de señales para el manejo de fallos de segmentación (`SIGSEGV`), se realizan múltiples conexiones entre señales y ranuras que intercomunican y sincronizan los objetos y se inicia el bucle principal de la aplicación con el método `exec`. Se utiliza la clase `QSharedPointer` para gestionar eficientemente la vida útil de objetos como el núcleo del sistema (`tfm::Core`), el procesador (`tfm::Procesador`) y el controlador del flujo de piezas (`tfm::MonitorizadorFlujoPiezas`). Estos punteros compartidos facilitan la administración de la memoria y garantizan la eliminación segura de los objetos cuando ya no se necesitan.

La clase Core está diseñada para gestionar la lógica central de inicialización y manejo de un sensor de cámara 2D, ya sea real o virtual gracias a la librería `Camera2DSensor` implementada y explicada en el apartado anterior. En su método `initialize` se asegura de que la inicialización se realice solo una vez y además se carga la configuración del sensor desde un archivo XML. El método `isInitialized` verifica si el sensor está inicializado correctamente. La clase también proporciona métodos para acceder y manipular la última imagen capturada por el sensor, como `saveConfigToSimulator`, que guarda la configuración por defecto en el simulador y `getLastImage`, que obtiene la última imagen del escáner 2D. Más aún, se utiliza un mutex (gracias a la clase `QReadWriteLock`) para garantizar el acceso seguro a los datos compartidos entre hilos se emiten mensajes de depuración con `QDebug` para informar sobre posibles errores durante la ejecución relacionados con la mala configuración del sensor en el momento de extraer el frame actual o cargar el escaneo.

2.3.1.1.- Preprocesamiento

En un entorno industrial, y más en la superficie de una cinta transportadora de tono oscuro, es frecuente que aparezcan numerosas impurezas y ruido que afecte a la calidad de las imágenes tomadas, así como posibles reflejos tanto en la cinta como en las propias piezas debidos a iluminación adversa, ruido electrónico, vibraciones y otros factores, aunque hasta cierto punto las condiciones están suficientemente controladas.

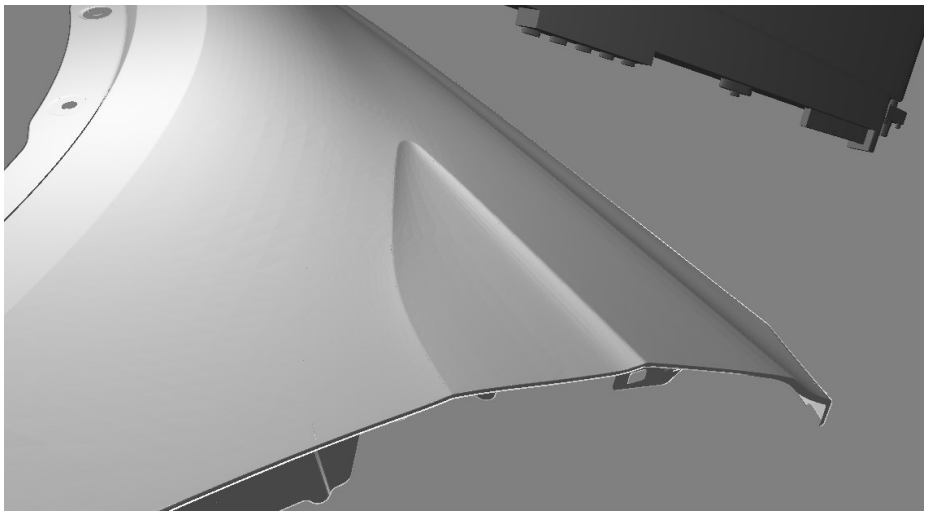


Fig. 58. Problema de reflejos especulares en puntos del borde del objeto debidos a su morfología. Se suprimen en la etapa de preprocesado mediante un filtro de mediana.

Así, una (aunque breve) etapa previa de preprocesamiento puede resultar útil para mejorar la detección precisa de las piezas y mejorar la calidad de los fotogramas. Esto se hizo más notorio en el caso de la simulación, donde las piezas poseían un reflejo muy intenso sobre todo su contorno a causa del propio modelado de las piezas y, mediante la aplicación de un filtro de mediana, se consiguió disminuir suficientemente ese efecto.

2.3.1.1.1.- Filtrado de ruido

Los filtros gaussianos utilizan un kernel (núcleo) que sigue una distribución gaussiana bivariante. Si la media y la varianza son las mismas para los dos ejes, los valores del kernel pueden obtenerse mediante la siguiente expresión:

$$Gauss_{\sigma^2}(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{\sigma^2}}, \tag{32}$$

que denota una gaussiana 2D isotrópica normalizada de tal forma que la integral doble en su dominio, es decir, su volumen, sea unitario. El valor máximo de la función (centro) se corresponde con el píxel de referencia y su cota disminuye en función de la varianza σ^2 establecida. Se nota que este resultado para 2D puede descomponerse en una convolución de dos kernels 1D (filtro separable).

En primera instancia, un filtro gaussiano puede ayudar a reducir el ruido blanco presente en las imagen, es decir, suavizar la pieza eliminando factores adversos causados

por ruido electrónico, variaciones aleatorias de en la intensidad de los píxeles o iluminación desigual mediante un promedio ponderado en un área local alrededor de cada píxel. Se cuida de no emplear un kernel demasiado grande para no perder componentes de muy alta frecuencia (detalles finos de la pieza) o pequeñas variaciones de intensidad que pueden ser beneficiosas para los posteriores algoritmos de análisis. Si se incrementase mucho el tamaño del kernel o el valor de su desviación estándar (en la dirección horizontal o vertical) los bordes se difuminarían en sobremedida y podría llegar a afectar negativamente a los detectores de puntos de interés cercanos a los contornos de la pieza, lo que se quiere evitar. Cabe comentar que el tamaño del kernel debe ser un número impar y positivo. Además, un valor σ sólo tendrá un efecto significativo en el resultado si el tamaño del núcleo también es suficientemente alto.

En el método `Procesador::aplicarFiltroGaussiano` se hace uso de la función `cv::GaussianBlur` para aplicar un filtro gaussiano a cada frame adquirido. El tamaño del kernel especifica el ancho y la altura de la ventana de convolución, mientras que las desviaciones estándar determinan la cantidad de suavizado aplicado en las direcciones horizontal y vertical.

El filtro de mediana, siendo un filtro que conserva mucho mejor los contornos y detalles abruptos que el gaussiano, sirve para eliminar el ruido impulsional (de sal y pimienta) presente en la imagen, que en un sentido real se puede corresponder polvo o suciedad, ya sea en la cinta como en la lente de la cámara. El filtro reemplaza el valor de cada píxel por la mediana de los valores de los píxeles en la ventana escogida, lo que causa que las zonas homogéneas sea aún más homogéneas y los contornos queden, aunque definidos, más claramente identificables. De esto resulta que sea capaz de corregir el efecto de los reflejos en los bordes de las piezas provenientes del simulador; en cierto sentido, esos reflejos eran valores atípicos o extremos entre el borde de la pieza y el fondo (la cinta), así que se lograba reducir su efecto pues la mediana no es sensible a ellos.

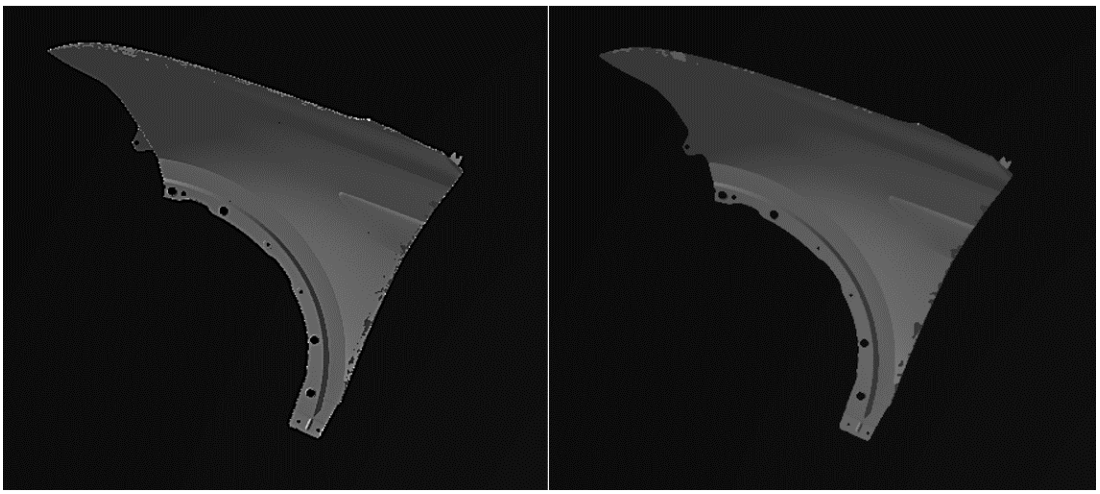


Fig. 59. Ejemplo de la reducción del brillo especular de las piezas mediante filtros espaciales.

En el método `Procesador::aplicarFiltroMediana` se hace uso de la función `cv::medianBlur` para aplicar un filtro gaussiano a cada frame adquirido. Se fuerza a que

el tamaño del kernel sea impar para garantizar la unicidad del centro de la ventana (evitar ambigüedades). Cabe comentar que es posible reducir la complejidad del algoritmo de $O(w^2 \log w)$ a $O(w)$ mediante un sencillo truco[15] [39, pág. 248] [40, pág. 131].

Otro filtro interesante es el bilateral [41], un filtro de suavizado que tiende a preservar los contornos. Una motivación típica para el suavizado gaussiano es que los píxeles de una imagen real en una imagen real deben variar lentamente en el espacio y, por tanto, estar correlacionados con sus vecinos, mientras que el ruido de fondo puede variar enormemente en el espacio (es decir, el ruido no está correlacionado espacialmente). En este sentido, el suavizado gaussiano reduce el ruido al mismo tiempo que preserva la señal. Desgraciadamente, este método falla cerca de los bordes, donde se espera que los píxeles no estén correlacionados con sus vecinos y consecuentemente se suavizan. A costa de un poco más de tiempo de procesamiento, el filtrado bilateral proporciona una manera alternativa de suavizar una imagen sin eliminar del todo los bordes.

Al igual que el suavizado gaussiano, el filtrado bilateral construye una media ponderada de cada píxel y sus componentes vecinos. La ponderación tiene dos componentes es la misma utilizada por el suavizado gaussiano. La segunda componente es también una ponderación gaussiana, pero no se basa en la distancia espacial entre el píxel central y los píxeles menos similares. El objetivo de este filtro es, de cierta forma, ponderar en mayor medida los píxeles más similares que los más diferentes, lo que es útil y ayuda para segmentar la imagen de manera más clara. El filtrado bilateral toma dos parámetros: el primero es la anchura del kernel (típicamente Gaussiano) utilizado en el dominio espacial (que es análogo a los parámetros sigma del filtro Gaussiano) y el segundo es la anchura del kernel en el dominio del color. Cuanto mayor sea este segundo parámetro, más amplia será la gama de intensidades (o colores) que se incluirán en el suavizado (y, por tanto, más extrema deberá ser una discontinuidad para que ser preservada.

En el método `Procesador::aplicarFiltroBilateral` se hace uso de la función `cv::bilateralFilter` para aplicar un filtro bilateral a los frames adquirido a la hora de segmentar dinámicamente la pieza en movimiento. La función toma como argumentos el tamaño de la vecindad de interés, la desviación estándar del filtro en el espacio de color (un valor mayor indica que los colores más distantes dentro la vecindad se mezclarán más, lo que resultará en áreas de color más uniforme) y la desviación estándar en el espacio de coordenadas de píxel (un valor mayor indica que los píxeles más alejados influirán en la imagen filtrada, en tanto en cuanto que sus colores sean lo suficientemente cercanos, correspondientemente al anterior parámetro).

2.3.1.1.2.- Operaciones morfológicas

La suma de Minkowski de dos conjuntos A y B se define como el conjunto de puntos resultantes de todas las posibles sumas vectoriales de los elementos de los dos conjuntos:

[15] El "truco" consiste en almacenar el histograma de grises de cada ventana junto con el valor de la mediana y el número de píxeles cuyo valor es menor que la mediana, pues estos dos últimos valores se pueden actualizar para cada píxel a un tiempo aproximadamente constante.

$$A \oplus B \equiv \{z : z = a + b, a \in A, b \in B\} = \bigcup_{b \in B} \{a + b : a \in A\} = \bigcup_{b \in B} A_b \,, \tag{33}$$

donde se ve que $A \oplus B$ es el conjunto $\{z\}$ tal que para cada punto en el conjunto hay algún punto a en A y b en B cuya suma es z, así como que $A \oplus B$ es la unión de los conjuntos resultantes de trasladar A por cada elemento de B..

La resta de Minkowski de dos conjuntos A y B se define de manera análoga:

$$A \ominus B \equiv \{z : z - b \in A, \forall b \in B\} = \bigcap_{b \in B} \{a + b : a \in A\} = \bigcup_{b \in B} A_b \,, \tag{34}$$

donde se ve que $A \ominus B$ es el conjunto $\{z\}$ tal que para cada punto z en el conjunto y para todos los puntos b en B, el punto z – b está en A, así como que $A \ominus B$ es la intersección de los conjuntos resultantes de trasladar A por cada elemento de B.

Las operaciones morfológicas de dilatación y erosión se derivan naturalmente de las anteriores, de tal forma que se corresponden con la suma y resta de Minkowski tras reflejar primeramente el segundo conjunto (la suma es idéntica a la dilatación, pues la salida de la primera no manifiesta reflejo alguno):

$$dilatación \equiv A \oplus B = \{z : z = a + b, a \in A, b \in B\} \tag{35}$$

$$erosión \equiv A \oslash B \equiv A \ominus \check{B} = \{z : z + b \in A, \forall b \in B\} \tag{36}$$

El cierre y la apertura se definen como una dilatación seguida de una erosión y viceversa, respectivamente. Tienen la ventaja de que conservan más parte de la información que se pierde al engrosar o reducir el tamaño de un conjunto (la masa de un objeto binario):

$$cierre \equiv A \odot B = (A \oplus B) \oslash B \tag{37}$$

$$apertura \equiv A \circledcirc B = (A \oslash B) \oplus B \tag{38}$$

La erosión y la apertura son útiles, de manera similar al filtro de mediana, para eliminar pequeños objetos, impurezas o detalles no deseados en las imágenes; pueden ayudar a separar objetos que están muy cerca unos de otros, como piezas superpuestas o agujeros mal binarizados. Aunque (la erosión más que la apertura) reducen la presencia de detalles finos en la imagen. La apertura es particularmente conveniente para suavizar las imágenes sin eliminar por completo los objetos de interés.

Alternativamente, la dilatación y el cierre pueden ayudar a unir zonas que están parcialmente separadas, como roturas imprevistas en una misma pieza, que suelen aparecer junto con los cambios abruptos de iluminación, o para rellenar huecos y normalizar la masa binaria de todas las piezas más uniformemente de cara a su comparación con distintas

métricas. El cierre es particularmente conveniente para mantener los objetos más grandes mientras se eliminan detalles no deseados.

Cada una de las anteriores operaciones, en una medida controlada empíricamente, sirven eficazmente para encontrar máscaras binarias de las piezas bien definidas y robustas antes cambios en la iluminación o movimientos de la cámara. Un elemento estructurante demasiado grande en una erosión o dilatación puede suponer una pérdida de información irrecuperable sobre el objeto.

Las operaciones morfológicas son especialmente útiles en la etapa de detección de movimiento posteriormente explicada para cerrar regiones, abrir huecos de las piezas y unificar el objeto como una masa binaria uniforme claramente definida, a fin de que la dinámica entre frames sea correspondiente y no haya ruido en la emisión de señales entre objetos.

Mediante la función `cv::getStructuringElement` se puede definir un elemento estructurante de la transformación morfológica y aplicarlo mediante `cv::erode`, `cv::dilate` o `cv::morphologyEx`, especificando su punto de anclaje (centro), el número de iteraciones a realizar y el tipo de tratamiento especial que se le da a los bordes de la imagen (valor constante, repetido, reflejado, etc.).

2.3.2.- Detección de movimiento

El primer paso consiste en detectar que un objeto está entrando; y aquí en vez de "entrando" se podría estar hablando de "saliendo", recorriendo el interior de la cinta, a punto de entrar/salir, inmediatamente después de haber sido considerada entrante… indistintamente. Pero no solo eso, sino que, más específicamente: que «una» pieza está entrando (no un objeto cualquiera), que «la» pieza está entrando (la pieza de interés de entre todas las piezas que se quieren identificar), que el área de movimiento útil detectado es suficientemente grande/pequeña para considerar que la pieza ha entrado/salido o que se es capaz (y cómo) de discernir entre la pieza moviéndose y el ruido de fondo u otros objetos de poco interés, entre otras cuestiones.

2.3.2.1.- Substracción de fondo

La problemática de la detección de piezas, circunscrita en el marco de este trabajo, consiste en un proceso de decisión entre dos alternativas: qué parte de la señal recibida corresponde a material de los objetos y qué otra parte es material de la cinta, a las que se llamarán pieza y fondo, respectivamente. En general, la información adquirida de las superficie a estudiar suele estar entremezclada, con lo que la parte de los datos recibidos que representa la pieza suele ser minoritaria respecto al volumen total de información y, además, es altamente ruidosa porque está en movimiento continuo. Pero el hecho de que esté en movimiento también puede ayudar a separar la información de ambos elementos y más aún teniendo en cuenta que la cinta es homogénea y su superficie presenta un contraste de color muy marcado con el objeto, lo que es un beneficio. En este apartado se presenta una técnica destinada a detectar movimiento en la escena y discernir si lo que está en movimiento se trata de una pieza, a fin de lanzar la orden de comenzar a analizarla.

2.3.2.1.1.- Modelo de mezcla de gaussianas

Se emplea un modelo de mezcla de gaussianas adaptativo [42-43] (*Gaussian Mixture Model* o *Mixture of Gaussians*) para segmentar las piezas en movimiento, cuya sofisticación y naturaleza le proporciona una mayor resiliencia antes los cambios poco interesantes en la escena (el ruido) en comparación con otros métodos que emplean la media o mediana, incluso adaptativamente. Segmentar una pieza en movimiento sobre un fondo estático es un problema que involucra registrar y construir un modelo discriminante que discierna píxeles pertenecientes al fondo o al primer plano basándose en la variación de intensidad de los mismos a lo largo del tiempo.

La idea es suponer que el fondo de la escena (tomando como entrada una secuencia de "n" fotogramas) se modela como una suma de varias distribuciones gaussianas con medias y covarianzas (o matrices de covarianza en espacio de color) diferentes, lo cual es asumible en la práctica por varias razones: la cinta transportadora (el fondo) presenta variaciones muy sutiles de intensidad en comparación con los píxeles en lo que cae la pieza en movimiento, el comportamiento de los píxeles (que son dimensiones o variables [aleatorias] independientes si se usa escala de grises) que pertenecen al mismo grupo

entendidos según el Teorema Central del Límite y, gracias a este método en particular, una gaussiana puede representar una característica muy concreta del fondo, como una región iluminada de manera particularmente diferente al resto o con sombras que proyectan las piezas sobre este.

Este método construye un histograma para cada píxel de cada secuencia y registra sus valores de intensidad; de suerte, suele resultar en que las intensidades más frecuentemente repetidas, que están agrupadas, se corresponden con el fondo estático, mientras que las variaciones más "espurias" se deben a objetos que caen eventualmente en el píxel y registran sus valores de intensidad que, si son suficientemente distintos al fondo, el modelo logra tener en consideración.

A la gaussiana unidimensional que modela al fondo en un histograma se le agrega una componente de ponderación ω, denominada escala o "evidencia", que varía la altura de esta concorde a la altura de la subregión o subpoblación de fondo u objeto del histograma:

$$P(x) \approx P'^{(x)} = \sum_{n=1}^{n} \omega_n \cdot Gauss_n(x, \mu_n, \sigma_n) \tag{39}$$

donde "x" es la intensidad del píxel en cuestión, μ es la media, σ la desviación estándar y ω la escala o evidencia de la gaussiana, que viene dada por la siguiente expresión:

$$Gauss(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{40}$$

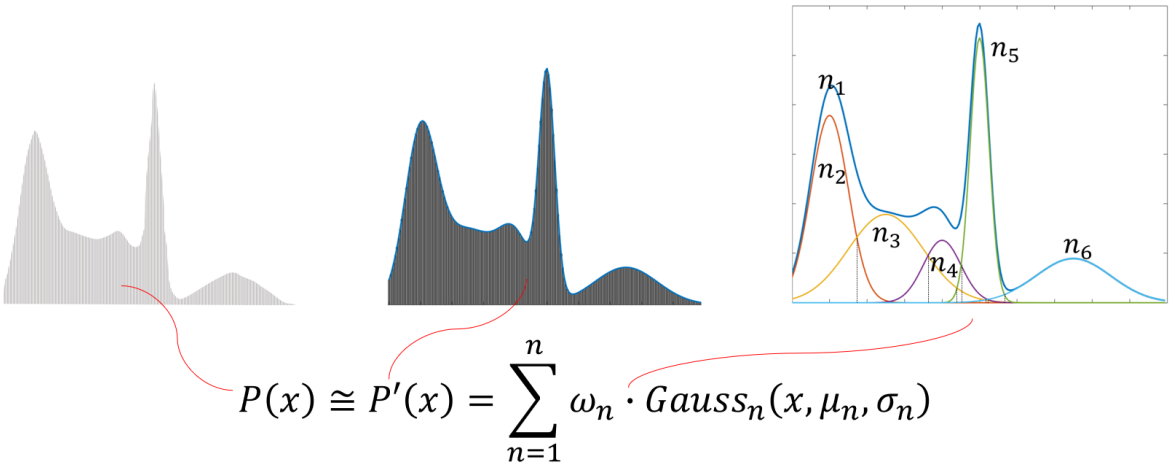


Fig. 60. Idea detrás del modelo de mezcla de gaussianas.

El método trata de estimar y parametrizar las "n" gaussianas que, adicionadas, se quiere modelen el histograma de intensidad de cada píxel. El modelo queda entonces determinado por el la suma ponderada del número de gaussianas a utilizar. Si este número es muy elevado los datos de entrada pueden resultar en respuestas inestables, pero si es muy bajo el modelo puede ser poco preciso al tener poca flexibilidad; en la práctica, la

parametrización de los modelos se tuvo que calibrar empíricamente porque el comportamiento y las condiciones en la realidad y simulación eran notablemente diferentes.

Dada la distribución del modelo, parece razonable pensar que los píxeles son fondo "la mayoría del tiempo" y que su variación de intensidad abarca un rango de valores estrecho, así que el caso contrario se da para el primer plano, con lo que las gaussianas con un ratio ω/σ elevado y reducido se corresponden a una y otra parte, respectivamente. De manera general, estos modelos asignan cada observación a un clúster maximizando la probabilidad a posteriori de que un punto de datos pertenezca al clúster asignado. El algoritmo adaptativo escogido resulta adecuado para eliminar las sombras de los objetos gracias al emplea de ecuaciones recursivas que no solo actualizan constantemente los parámetros de las gaussianas sino también para seleccionar simultáneamente el número apropiado de componentes para cada píxel, aprovechando el histograma computado en la iteración anterior para mejorar la eficiencia del cómputo.

La clase `MonitorizadorFlujoPiezas` se encarga de resolver estas tareas. Se emplea la función de OpenCV `createBackgroundSubtractorMOG2` para crear un objeto que herede los métodos necesarios para ejecutar el algoritmo descrito. En el bucle de monitorización se espera a que el tamaño del búfer de "últimas imágenes" provenientes del sensor (simulador, archivo, cámara real…) alcance un mínimo establecido antes de procesar el flujo de fotogramas (obtenidos en otro hilo secundario). Se llama al método `MonitorizadorFlujoPiezas::segmentateGMM` para obtener la máscara de primer plano acumulada de las imágenes en el búfer actual, que toma como entradas un vector de imágenes, parámetros del modelo de mezcla de gaussianas como la tasa de aprendizaje inicial (0,9), el umbral de varianza (si es demasiado alto la pieza se ha de mover muy rápido para ser detectada) (0,1 en simulación, 25 en la realidad) o el número de gaussianas (escogidos experimentalmente para la simulación y realidad) (32 en simulación, 124 en la realidad)… y devuelve un frame con la pieza en movimiento segmentada. También se aplican varias operaciones morfológicas (erosión, dilatación y cierre) relevantes para mejorar la calidad de la máscara resultante. Se usa la enumeración `ActiveSensor` para saber qué tipo de sensor se está utilizando y reajustar dinámicamente los parámetros del modelo según se trate de simulación o realidad; todos estos valores se tuvieron que ajustar empíricamente y son distintos entre sí.

En la siguiente figura se pueden ver dos casos de cómo unos parámetros pobres del modelo de detección de movimiento, como pocas mezclas, una tasa de aprendizaje inicial baja o una varianza excesiva, pueden afectar a dicha labor. Se ven dos tipos de pieza entrando y dentro de la cinta cuyo movimiento se detecta y representa mediante imágenes binarias que aproximan los píxeles cuya variación de intensidad es suficientemente dinámica como para considerar que se son objetos moviéndose; por dos partes se ven detecciones aceptables y malas, estando las últimas caracterizadas por que el objeto queda con un área mucho menor del ideal y agujereado por zonas que se supusieron fondo estático.

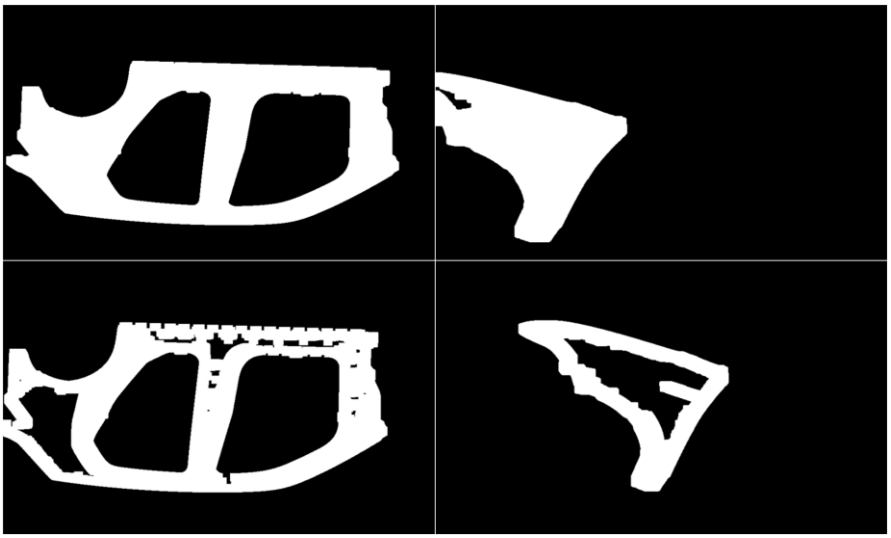


Fig. 61. Movimiento de piezas detectado bien (arriba) y mal (abajo) por el modelo de mezcla de gaussianas.

También se calcula el área de la pieza en movimiento para discernir si la pieza es suficientemente grande, demasiado pequeña o de tamaño permisible y se determina su posición en relación con ciertos umbrales y movimientos en los bordes izquierdo y derecho, a fin de emitir señales (`sg_entraPiezaIzda`, `sg_salePiezaDcha`…) según la pieza está entrando, a punto de entrar, moviéndose en interior de la cinta, saliendo (o incluso si hay dos piezas a la vez en la escena)…, que son emitidas a la clase `Procesador` para implementar una lógica de inicio y fin de procesamiento robusta.

2.3.3.- Reconocimiento de una pieza

El registro de imágenes (*image registration*) es la tarea de «superponer» dos o más imágenes de una misma escena tomadas en distintos momentos, desde diferentes puntos de vista o múltiples sensores con el propósito de encontrar una función que mapee los puntos de una imagen a los puntos (realmente) correspondientes de otra, consiguiendo así, geométricamente, un alineamiento entre estas. Es un proceso de transformación de diferentes conjuntos de datos que compensa las diferencias de contexto (movimiento de la escena, cámara, iluminación adversa…) de cada marco de referencia y traduce los datos de entrada a un mismo (o por lo menos más similar) sistema de coordenadas en donde su análisis sea más equiparable; en el campo de la visión artificial, se utiliza para el alineamiento, la composición de panoramas, el reconocimiento automático de objetos u otros tipos de análisis de imágenes y es necesario para poder comparar o integrar de manera precisa los datos obtenidos de diferentes fuentes de información. El presente problema es otro claro ejemplo en el que esta técnica puede ayudar a componer una respuesta útil, pues el principal objetivo del algoritmo a desarrollar es comparar la orientación de una pieza arbitrariamente orientada con respecto a esa misma pieza perfectamente situada, para lo que referenciar ambas imágenes dentro de un marco espacial y frecuencial común surge de provecho de cara a todo el análisis.

Los métodos basados en píxeles usan directamente los valores de intensidad de la imagen para registrarla, a diferencia de la registración basada en características, que identifican y describen rasgos distintivos en regiones de la imagen. Los métodos basados en características tienen algunas ventajas sobre los otros, sobre todo cuando bajo diferentes condiciones de iluminación o tiempos de exposición o cuando las imágenes se superponen solo parcialmente. Por otro lado, la principal ventaja de los métodos basados en píxeles en comparación con los otros es su mejor precisión para aquellas imágenes tomadas bajo condiciones de iluminación similares y que tienen una superposición significativa, debido a que se utiliza toda la información disponible en la imagen, lo que permite buscar (a veces) una precisión subpíxel.

Más aún, los métodos de registro de imágenes basados en píxeles y aquellos que utilizan características pueden complementarse entre sí de manera fructuosa; en esta aplicación se obtiene un registro preliminar de la pieza en la escena visible a reconocer de manera aproximada utilizando algoritmos basados en píxeles y luego, refinando la identificación, se prepara minuciosamente el procesamiento del extractor de su pose utilizando algoritmos basado en características dentro del área de interés que dejaría superpuestas la pieza actual y su referencia correspondiente.

Así pues, una vez que una pieza se encuentra totalmente dentro del campo de visión de la cámara se ha de identificar. Para esta labor se ha desarrollado una entidad denominada “clasificador morfológico” que emplea diversas técnicas de procesamiento a nivel de píxel para proporcionar una primera respuesta plausible sobre la posibilidad de que la pieza se corresponda con alguna de las disponibles en la base de datos (ya sea real o

simulada). El clasificador morfológico evalúa una pieza mediante múltiples métricas y construye un vector de votos específico a través de un enfoque de combinación ponderada de indicios, cuya norma discrimina unos candidatos de otros en función de las métricas que hayan sido consideradas como suficientemente buenas en base a ciertos umbrales empíricos.

Este enfoque de componer o ensamblar en una sola varias respuestas es lo que se conocería en el ámbito del machine learning como "boosting", que es una técnica en la que se combinan varios modelos de aprendizaje débiles para crear un modelo más fuerte y preciso. Algoritmos como AdaBoost (*Adaptive Boosting*), XGBoost (*eXtreme Gradient Boosting*), GBM (*Gradient Boosting Machine*)… asignan pesos a las instancias de datos (lo que serían las salidas de las distintas métricas) y ajustan iterativamente los modelos (según los datos de entrada etiquetados) para dar más énfasis a las instancias mal clasificadas en las iteraciones anteriores; así cada modelo débil se entrena para corregir los errores cometidos por los modelos anteriores, lo que lleva a un modelo final más robusto. De cierta forma, si algún método de entre los desarrollados tiende a ser particularmente menos preciso que el resto, a lo largo de la etapa de entrenamiento, el modelo lo ponderaría con menor fuerza que al resto y, así, se tendría "menos en consideración" su respuesta como aportadora a la respuesta final del sistema, pues su fiabilidad habría ido disminuyendo. Esto es lo que se emula en algunas partes del algoritmo en las que, por ejemplo (`Procesador::compareSimilarityData`), se da predilección a que una imagen tenga más puntos característicos en común con una referencia a que tenga mayor norma del vector de votos o cuando se "suma doblemente" si dicha norma es mayor que 10 o su longitud está completa, en, comparación con el resto de métricas, para aquellas que involucran el contorno de la pieza.

La salida de este primer bloque compendia todo un árbol de métodos analíticos del fotograma y es una señal binaria que indica si la pieza actual se da por válida o no a priori según la base de datos que contiene todas piezas consideradas como buenas. Esto puede servir para detectar si hay una deformación muy evidente en la pieza o, directamente, si viene girada en la tercera dimensión, con lo cual la pieza no debería ser considerada como válida y no tendría mayor sentido continuar con el resto del algoritmo. En caso de que la salida de este bloque sea cierta, el algoritmo proseguiría con la identificación de puntos característicos, correspondencias y el resto de tareas posteriormente explicadas. En este sentido, este bloque no busca encontrar la pieza de referencia que más se le parezca a la actual sino, más precisamente, en base a las referencias válidas determinar si la pieza actual pertenece a la categoría de "piezas suficientemente parecidas a las de referencia con una pose suficientemente similar a estas"; al sintonizar los parámetros del algoritmo se busca en este primer bloque emular una capacidad de discernimiento de piezas conocidas bien y mal orientadas a primera vista.

Como se ha comentado, el algoritmo en su conjunto funciona como un clasificador que analiza (eminentemente) la orientación de la pieza para determinar su validez; la traslación no es significativamente relevante en el análisis propuesto porque casi todo el

procesamiento se realiza dentro de la caja delimitadora de las piezas y siempre se puede elegir una referencia absoluta (un origen) en la imagen de manera trivial, pues la cámara es estática. También se puede notar que en la práctica la cámara estaría ajustada para que su campo de visión abarque justa y suficientemente lo que ocupen las piezas centradas a lo largo del ancho de la cinta transportadora, por lo que la traslación se considera indiferente salvo en el caso extremo de que se salga de los límites de la línea; además, este no es un problema de estimación de la pose, sino de su validación, mas sí sería interesante conocer (en otro proyecto) la traslación relativa a algún punto fijo del espacio de cara a que un brazo robótico, p. ej., se desplazase hacia la pieza inválida para retirarla de la cinta.

Como segunda salida, el clasificador morfológico proporciona la imagen de la base de datos que más se parece a la imagen actual; es decir, funciona como un modelo analítico completo que identifica una pieza tomando como referencia sencillas imágenes de las piezas que se desean reconocer (esto es una ventaja si las condiciones de contorno cambian y las piezas a identificar pasan a ser otras; no hay que recalcular ni reentrenar ningún tipo de modelo neuronal como ocurriría con soluciones de aprendizaje automático, salvando el *transfer learning*[16]).

De cierta forma, el clasificador morfológico es un agente inteligente que discierne de manera grosera pero robusta si la pieza actual está “muy mal” posicionada o si se considera válida. Dada la variabilidad en la forma de las piezas y las condiciones adversas de iluminación, exposición, ruido, precisión de la cámara y resolución… es ciertamente complicado construir un método de clasificación analítico invariante antes tales y otros cambios, pero con el clasificador morfológico se consiguen resultados resistentes pues se integran inteligentemente muchos tipos de métricas, los umbrales son escogidos debidamente de manera experimental y se construye un vector de respuestas que diferencia las mejores piezas de las peores (en lo relativo a su identificación).

A continuación se explican los métodos y algoritmos implementados en C++ mediante OpenCV y otras librerías para reconocer una pieza mediante técnicas analíticas.

2.3.3.1.- Segmentar piezas

Una imagen binaria digital es una matriz bidimensional que se ha obtenido a partir de una imagen en niveles de gris que se ha discretizado en dos niveles (0 y 1, 0 y 255…). Así, los píxeles que componen las regiones se etiquetan con el nivel alto, mientras que el fondo se etiqueta con el nivel bajo. Una imagen de este tipo se compone de todas las regiones planas conectadas que representan proyecciones de objetos percibidos sobre el plano discreto. De cierta forma, binarizar una imagen en escala de grises significa reducir su dimensionalidad perdiendo parte la información de la dimensión correspondiente a la

[16] Transfer Learning: técnica empleada en el ámbito del aprendizaje automático para traspasar el conocimiento aprendido por un modelo entrenado con un conjunto de datos de entrenamiento concreto a otro modelo que usa otro tipo de datos; por ejemplo, el hecho de transferir los pesos de las neuronas de una red entrenada dedicada a la tarea de detección de personas a otra dedicada a la clasificación de piezas, lo cual impulsa su eficiencia.

intensidad de los píxeles, de la misma forma de la que se puede proyectar un objeto tridimensional en una aproximación bidimensional proyectando todos sus puntos sobre uno de sus planos ortogonales. De esta manera, aunque se pierda información sobre la profundidad, iluminación, el contraste y otras características del objeto, se consigue tener una representación útil y manejable del objeto para realizar ciertas operaciones a nivel binario. Entonces, a partir de la región proyectada de cada objeto observado, pueden calcularse múltiples características geométricas y topológicas descriptivas.

El método `Procesador::extraePiezaFilled` segmenta los objetos presentes en el frame actual y devuelve el mayor contorno de la región (idealmente solo la pieza de interés). Es remarcable ver que una versión paleolíticamente simple del clasificador morfológico devolvería como pieza identificada esta región de área máxima segmentada; pero la complejidad de la identificación debe ser más completa.

2.3.3.1.1.- Otsu

Se escoge el algoritmo de binarización de Otsu como primer paso para la extracción de las piezas en las imágenes. El método de Otsu busca de manera automática un umbral óptimo para la binarización de la imagen que minimiza la varianza intra-clase (considerando dos clases: fondo y objeto), que se refiere a la variabilidad dentro de cada clase luego de aplicar el umbral.

Teóricamente, la probabilidad de que un píxel pertenezca a la clase “fondo” y “objeto”, respectivamente, se puede modelar por:

$$P_F = \sum_{i=0}^{T} P(i), P_O = \sum_{i=T+1}^{L-1} P(i)\,, \tag{41}$$

donde “L” es el número total de niveles de intensidad de la imagen (“celdas” de su histograma) y “T” el umbral escogido.

Por otro lado, las medias de intensidad en el nivel de gris de cada clase es:

$$\mu_F = \sum_{i=0}^{T} \frac{i \cdot P(i)}{P_F}, \quad \mu_O = \sum_{i=T+1}^{L-1} \frac{i \cdot P(i)}{P_O} \tag{42}$$

y la media global (promedio de la intensidad para todos los píxeles de la imagen) es:

$$\mu = \sum_{i=0}^{L-1} i \cdot P(i) \tag{43}$$

Disponiendo de esta información, la varianza intra-clase, que es una medida de dispersión dentro de clase tras haber aplicado el umbral, se define como la suma ponderada de las varianzas de cada clase:

$$\sigma_{FO}^2 = P_F \cdot (\mu_F - \mu)^2 + P_O \cdot (\mu_O - \mu)^2 \tag{44}$$

y el umbral de segmentación óptimo, que es el que se busca en el método de Otsu, se estima resolviendo (para T) la siguiente ecuación, optimizando la varianza intra-clase:

$$\sigma_{FO}^2 = P_F(T) \cdot P_O(T) \cdot (\mu_F(T) - \mu_O(T))^2 \quad (45)$$

Este es un algoritmo que ha servido suficientemente en la práctica incluso dadas las condiciones de iluminación más adversas en simulación y realidad; otras posibilidades habrían podido ser emplear algoritmos de umbralización adaptativa o local, como los de Niblack, Sauvola, Bernsen, Eikvil o Parker.

En la práctica, dado que se realiza experimentación en simulación y en la realidad, dependiendo del sensor del que provengan los frames se decide realizar una binarización ajustan manualmente el umbral o mediante el método de Otsu (ver el apartado "Discusión").

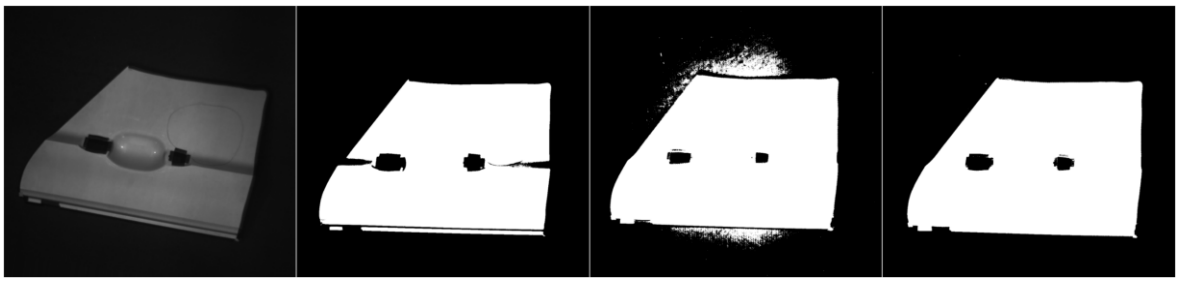


Fig. 62. De izda. a dcha.: pieza real original, binarización mediante Otsu, mediante un umbral manual mal escogido y mediante un umbral óptimo.

Se hace uso de la función `cv::threshold` para binarizar las imágenes; sabiendo que el fondo siempre será (mucho) más oscuro que los objetos, dado el material y tono de color de las piezas de que se dispone (aunque el fondo pueda tener algún reflejo eventual, que se filtraría por código), se eligen los valores 20 y 80 que diferencian, en el histograma considerado en el algoritmo, los píxeles de fondo y objeto. En la experimentación real, el abanico de valores que devolvían usualmente una segmentación correcta de las piezas rondaban entre los valores 5 y 25 en escala de grises, con 8 bits de precisión.

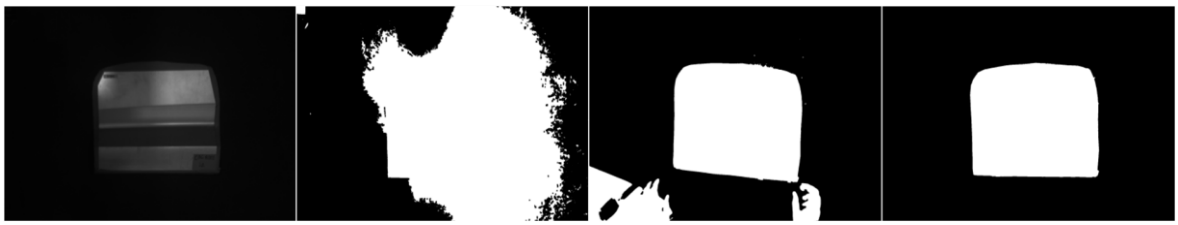


Fig. 63. Otro ejemplo en el que una elección mala, manual (segunda imagen) o automáticamente (Otsu, tercera imagen), del umbral afecta a la segmentación de la pieza (bien segmentada con un umbral preciso a la dcha.).

2.3.3.1.2.- Canny

El operador de detección de bordes de Canny es un método muy útil y eficaz con el cual se pueden encontrar los contornos de las piezas en las imágenes. Este algoritmo se basa en la idea de derivada para encontrar las zonas de la imagen cuyo gradiente de intensidad varía más bruscamente. Primeramente la imagen se suaviza con un filtro

gaussiano para reducir ruido blanco, luego se aplican operadores de derivada (como Sobel, o Prewitt, Roberts, Scharr, Frei-Chen, diferencia regresiva, adelantada y central…) para calcular sus gradientes, de los cuales se extrae su magnitud y orientación; luego, se realiza una supresión de no máximos para refinar los bordes detectados y se utilizan dos umbrales (uno inferior y otro superior) para clasificar los píxeles en tres categorías: bordes débiles, fuertes y no bordes. Finalmente se conectan los píxeles de bordes débiles a los de bordes fuertes en caso de que los primeros estén dentro de la vecindad de los segundos.

Tras la binarización con Otsu, se usa la función `cv::Canny` para este fin; que toma como argumentos de entrada los dos umbrales del algoritmo (valores razonables encontrados fueron el umbral de Otsu para el límite superior y su 80 % para el inferior) y devuelve una imagen de contornos en formato `cv::Mat`. Posteriormente se emplea `cv::findContours` para analizar de manera más flexible cada contorno de la imagen en formato `std::vector<std::vector<cv::Point>>`, a fin de acceder a la información espacial de cada uno de sus puntos. Estimando el área encerrada por cada contorno y la longitud de su perímetro se mantiene el mayor contorno del objeto para filtrar ruido y bordes indeseados.

El método `Procesador::extraeRoiAislada` extrae la ROI de la pieza anteriormente segmentada para servir de entrada a posteriores métricas. Utilizar la ROI en lugar del fotograma completo es una de las formas de las que se consigue invarianza a la traslación a la hora de identificar las piezas.

La extracción de la ROI no orientada se lleva a cabo en dicho método mediante las funciones de OpenCV `cv::boundingRect`, `cv::findContours` y `cv::drawContours`. Cabe notar que la salida de este método no es una imagen binaria, sino una imagen en todo el rango de la escala de grises (con 8 bits de resolución) en la cual la pieza se aísla con un fondo totalmente negro; de esta manera se mejora la respuesta de ciertas métricas que tienen en cuenta el histograma de intensidades de los píxeles y su nivel de gris, para enfrentarse a reflejos en la cinta, sombras u otros efectos adversos.

2.3.3.2.- Histéresis de área

Este sub-bloque conforma una primera forma de discriminar las piezas filtrándolas según el área que ocupan. Suponiendo que la cámara está en todo momento en una posición fija (es estática), los valores del área (número de píxeles en blanco) de la mayor y menor pieza disponible son conocidos. Sabiendo esto, es claro que un objeto con un área "mucho menor" al menor valor permitido o "mucho mayor" al mayor valor permitido se deberían considerar como piezas no válidas. Esto se hace más claro aún sabiendo que la rotación de las piezas solo puede ser bidimensional, con lo que (exceptuando los efectos desdeñables de la perspectiva de la pieza a lo largo del campo de visión de la cámara) el área no varía apreciablemente en tanto en cuanto la variación de la orientación sea pequeña; así pues, una variación excesiva del área detectada supone o bien una variación inadmisible en el ángulo de la pieza (con lo que, de primeras, la pieza pasaría a ser no válida, por parte de este sub-bloque) o bien que la pieza presente en ese momento no se

corresponde con ninguna disponible en la base de datos (por lo tanto, tampoco se reconoce y no es válida).

Para calcular el área de los objetos se emplea la función `cv::contourArea`, que utiliza la fórmula de Green para calcular el valor del área de la región encerrada por el contorno basándose en la posición de sus puntos, pudiendo devolver la orientación del mismo. En el ámbito de la teoría de campos vectoriales, la fórmula de Green (que en realidad no es más que consecuencia del teorema de Stokes) establece una relación entre una integral de línea a lo largo del contorno de una región plana y una integral doble sobre la región encerrada por ese contorno:

$$\oint_{C^+} (P\,dx + Q\,dy) = \iint_D \left(\frac{\partial Q}{\partial y} - \frac{\partial P}{\partial y}\right) dx\,dy\,, \tag{46}$$

donde D es un dominio de Stokes[17], C es su borde, que se supone que es una imagen de una sola curva cerrada simple (C^+ denota que está positivamente orientada) y P y Q son dos funciones de $\mathbb{R}^2$ diferenciables.

Los umbrales de área son, lógicamente, distintos para la simulación y realidad y se deben calibrar y definir empíricamente. La salida de esta métrica es el booleano `areaSimilarity`, que indica si la pieza es válida a ojos de este sub-bloque.

2.3.3.3.- Número de Euler

Dada la forma de las piezas de referencia de que se dispone, uno de los métodos más intuitivos para identificarlas es contar el número de agujeros que presentan. Anteriormente se comentaba que a partir de una imagen binaria pueden calcularse múltiples características geométricas y topológicas descriptivas del objeto; el número de Euler es una de estas características.

El número de Euler, para una imagen, se define mediante la siguiente expresión. En teoría de grafos, si tenemos un grafo plano conectado, su número de vértices, menos su número de aristas, más su número de caras (incluida la exterior) siempre será igual a dos.

$$E = N - H, \tag{47}$$

donde N es el número de componentes conectados de la imagen (regiones aisladas del fondo) y H es el número de agujeros en ellos. Así se puede comparar el número de agujeros dentro de cada componente conectado en la imagen actual y la de referencia, estableciendo un primer paso muy discriminante para diferenciar unas piezas de otras.

Aunque a simple vista un humano es capaz de computar casi instantáneamente el número de Euler de una imagen o de una pieza (suponiendo que el único componente

[17] Un dominio de Stokes es un "abierto" A (un conjunto $A \subset \mathbb{R}^n$ se llama abierto si cualquier punto $x \in A$ tiene una "bola" alrededor totalmente contenida en A) de $\mathbb{R}^2$ tal que su frontera ∂A es una unión finita de curvas cerradas simples.

conectado es la pieza entera y totalmente segmentada), su computación conlleva muchas iteraciones y se puede llevar a cabo mediante algoritmos muy variados. Como se detalla en [44] o [39, pág. 193], existen variedad de métodos para calcular el número de Euler de una imagen binaria, entra los que entran en juego la representación de la imagen mediante "quad-trees" o "bin-trees", el valor de una cierta función aditiva que pertenece a la familia de las integrales Quermass, su definición en términos de vértices, caras y aristas de un grafo de conectividad (equivaldría a considerar píxeles del primer plano como vértices, bordes como aristas y grupos de píxeles como caras) o el uso de los llamados operadores de Morse descomponiéndola en "k" formas conectadas mediante conectividades a 4 u 8, entre otros.

Viendo la siguiente imagen resulta patente que número de Euler depende críticamente de las capacidades del sistema para segmentar correctamente los objetos. La salida de este sub-bloque será muy ruidosa si no se aplican las operaciones morfológicas o transformaciones de intensidad adecuadas, o simplemente si no se controla debidamente la iluminación externa (lo cual no supone un problema en este caso).

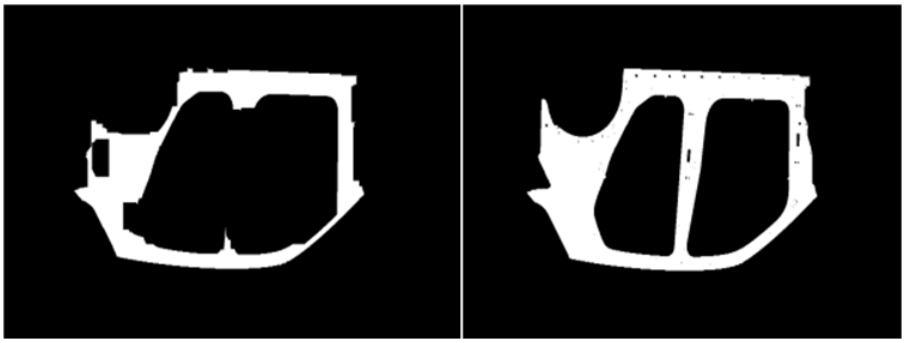


Fig. 64. Pieza segmentada perfectamente (dcha.) e incorrectamente (izda.) debido a una iluminación pobre.

Una forma sencilla de extraer el número de Euler es contar el número de componentes conectados (agujeros más objeto principal) y calcular el área de cada uno mediante las funciones de OpenCV `findContours` y `contourArea`; si es menor que cero significa el contorno actual es un agujero. Aunque OpenCV proporciona otros métodos para esta labor; entre ellos está la posibilidad de convertir el contorno a una región extrema (un conjunto de 4 píxeles conectados con todos sus valores de nivel de gris menores que los valores de su límite exterior) a través del constructor `cv::text::ERStat`.

2.3.3.4.- Momentos de Hu

En probabilidad, dados dos eventos aleatorios X e Y, la distribución o densidad (de probabilidad) conjunta de X e Y es la distribución de probabilidad de la intersección ambos, es decir, de los eventos X e Y ocurriendo de forma simultánea. Esta distribución se representa comúnmente mediante una función de masa de probabilidad $P(X = x, Y = y)$ para variables discretas o una función de densidad de probabilidad $f_{XY}(x, y)$ para variables continuas. El momento de orden (p, q) de la distribución conjunta (para variables aleatorias continuas) se define como:

$$M(p,q) = \mu_{pq} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} x^p \, y^q f_{XY}(x,y)\, dx\, dy\,, \tag{48}$$

que es el valor esperado del producto de las potencias (p, q) (para p, q = 0, 1, 2…) de ambas variables y proporciona información sobre la forma y las características de la distribución conjunta. Algunos casos particulares remarcables incluyen la media (o media marginal) (μ_{10}, μ_{01}), varianza y covarianza, coeficiente de correlación... Por ejemplo, μ_{10} representa el valor esperado (el promedio ponderado de los posibles valores que puede tomar una variable aleatoria) de la primera variable (X), μ_{01} representa el valor esperado de la segunda variable (Y), μ_{20} representa el segundo momento alrededor de la media para X, μ_{11} representa la covarianza entre X e Y, y así sucesivamente.

Más aún, el momento central de la distribución conjunta viene dado por:

$$U(p,q) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} (x-\mu_x)^p \left(y-\mu_y\right)^q f_{XY}(x,y)\, dx\, dy \tag{49}$$

Hu y Alt. [45] recogieron brillantemente estas relaciones de la teoría de la probabilidad y las aplicaron al análisis de la forma de imágenes binarias; basta con reemplazar la función de densidad de probabilidad conjunta con la imagen $F(x,y)$. Los momentos de imágenes son útiles herramientas para describir objetos posteriormente a su debida segmentación; propiedades de la imagen tales como centroide, área y orientación se pueden obtener a partir de estos momentos [46, pág. 600].

Pero no se puede calcular cualquier tipo de momento de dos imágenes y compararlos directamente esperando obtener una respuesta fidedigna sobre la correlación entre ambas, ya sea contando el número de coincidencias, lo que distan sus medias o medianas dentro de un rango admisible u otro método, pues aunque las piezas puedan ser las mismas hay factores como la perspectiva, iluminación, calidad de segmentación, rotación y traslación de la pieza, escala y ruido que hacen alto improbable que los momentos de una pieza de referencia y la del frame actual sean exactamente iguales, ni poco parecidos en muchas ocasiones. Por eso una primera medida de seguridad de emplear momentos (espaciales) especiales, que no sean sensibles a cambios de una forma u otra inevitables entre frames. Los momentos cartesianos no presentan invarianza a «nada», y son extremamente variantes a la escala (el ratio entre el momento de segundo orden y orden cero puede ser de varios órdenes de magnitud), por consiguiente usarlos y obtener buenos resultados conllevaría asumir que la cámara y las piezas están en una posición y en unas condiciones exactamente equivalentes en todos los frames que se analizan, lo que no se considera sensato. Aunque los momentos centrales buscan la invarianza a traslación (porque a cada punto del objeto se le resta su centroide, lo que sitúa el origen de coordenadas en el centro de masas del objeto) y los momentos centrales normalizados son, además, invariantes a cambios de escala (dividiendo los centrales por una potencia apropiada del momento de orden cero), se decide que los primeros momentos que se implementarán para extraer métricas fructíferas

de las piezas serán los 7 momentos de Hu, que presentan, además, invarianza a rotación y son un conjunto de siete combinaciones de los momentos centrales normalizados (η_{pq}) de hasta orden 3:

$$\phi_1 = \eta_{20} + \eta_{02} \tag{50}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{51}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{52}$$

$$\phi_4 = (\eta_{30} + 3\eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{53}$$

$$\begin{aligned}\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + \\ + (3\eta_{12} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]\end{aligned} \tag{54}$$

$$\phi_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} - \eta_{12})(\eta_{21} - \eta_{03}) \tag{55}$$

$$\begin{aligned}\phi_7 = (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + \\ + (3\eta_{12} - \eta_{30})(\eta_{32} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]\end{aligned} \tag{56}$$

El primer invariante de Hu (ϕ_1) es análogo al momento de inercia alrededor de centroide, donde las intensidades de los píxeles son análogas a la densidad física de un objeto o la densidad de probabilidad de una variable aleatoria. Por otra parte, el último invariante de Hu (ϕ_7) es, además, invariante "torcida", lo cual lo habilita para reconocer imágenes espejadas (valores de diferente signo pero la misma magnitud). Aunque son variantes antes cambios de perspectiva ni a rotaciones alrededor de ejes que no sean el eje perpendicular a la imagen, la proyección se puede suponer ortográfica situando debidamente la cámara y escogiendo una lente apropiada (o mismamente se pueden despreciar sin mayor error los cambios por perspectiva de la pieza) y por definición del problema y el presente trabajo dicha rotación provocará piezas no válidas, lo que aboga ciertamente el hecho de usar estos momentos.

Posteriormente al trabajo de Hu, J. Flusser [59] elaboró una teoría general sobre conjuntos completos e independientes de invariantes rotacionales derivados de momentos, en la que demostró que el conjunto invariantes de Hu no es ni completo ni independiente: por un lado ϕ_3 es redundante, pues es dependiente de otros y que, por otro, falta un invariante, que propuso y denominó "octavo invariante de Hu":

$$\phi_8 = \eta_{11}[(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2] - (\eta_{20} - \eta_{02})(\eta_{30} + \eta_{12})(\eta_{03} + \eta_{21}) \tag{57}$$

Dentro del método `Procesador::clasificadorMorfologico` se computan los momentos de Hu de los contornos de las piezas segmentadas gracias a las funciones `cv::moments` y `cv::HuMoments`. Los valores de los momentos devueltos son, típicamente, de un orden extremadamente bajo, por lo que para compararlos debidamente se aplica una transformación logarítmica decimal para que la escala no provoque comportamientos

erráticos debido a la comparación de números de punto flotante extremadamente pequeños y se normalizan al rango [0, 1] encontrando el valor del mayor momento en cada vector (relativo a la pieza de referencia y a la actual) y multiplicando su recíproco por uno de los momentos. Se ha observado en la fase experimental que el octavo momento de Hu es un muy buen indicativo de si la pieza está poco o muy rotada con respecto de la referencia.

Se consideraron e implementaron tres formas de comparar los vectores de momentos resultantes:

- Método 1: Si todos los momentos, uno a uno, se parecen lo suficiente, se da por válido. Esta es la alternativa más estricta (de cierta forma), pues se comparan los vectores elemento a elemento para todo su tamaño y, en base a un umbral preestablecido, se impone que todos los valores deben ser lo suficientemente parecidos entre sí para considerar que bajo (según) esta métrica la imagen de referencia y de entrada son similares. También puede requerirse que solamente un cierto porcentaje de los elementos han de ser lo suficientemente similares. Se emplea la siguiente ecuación para determinar la diferencia valor a valor:

$$\frac{2 \cdot |\eta_{Rpq} - \eta_{Apq}|}{|\eta_{Rpq} + \eta_{Apq}|}, \tag{58}$$

  donde η_{Rpq} representa el momento de orden (p, q) (o de Hu, o de Fourier) de la pieza de referencia, η_{Apq} representa lo mismo para la pieza actual y el resultado (coeficiente de discrepancia) ha de ser mayor que el umbral preestablecido para considerarse válido que los vectores son semejantes.
  Este método da la espalda a relaciones estadísticas entre los vectores pues y se considera más conveniente, para considerar la posibilidad de que el ruido (de las operaciones, p. ej.) afecte a los resultados, implementar los subsiguientes métodos que tienen en cuenta la media y la mediana de los vectores de momentos a fin de proporcionar una comparativa más flexible. Aunque también podrían implementarse otras variaciones de la media estadística como la media robusta o la media winsorizada, que eliminan un cierto porcentaje de los valores extremos antes de calcular el promedio, haciéndola más robusta a estos.
- Método 2: Si las medias de los momentos normalizados se parecen lo suficiente, se da por válido. La media se calcula como la suma de todos los elementos de un vector dividida por el tamaño del mismo. Es similar al anterior método, pues emplea la misma ecuación para calcular el coeficiente de discrepancia, pero algo más robusta porque tiene en cuenta la forma de la distribución de valores de los vectores y no los valores individualmente, lo que hace que la métrica sea más compacta; lo que es beneficioso en el caso de los momentos de Fourier, pues suelen resultar vectores de gran tamaño (y valores muy similares) y así proporciona un resultado más estable.
- Método 3: Si la moda de los valores positivos del método 1 es mayor que un umbral (es decir, si se repite más de un número preestablecido de veces que la métrica da la

respuesta positiva), se da por válido. De esta forma se es mucho menos sensible a los posibles valores atípicos que el primer método. También se puede implementar usando la mediana, pero considerando que los vectores tendrán el mismo tamaño en prácticamente cada muestreo y no habrá apenas pérdida de datos (pues se cuida de realizar una segmentación adecuada), no resulta significativo emplear la mediana, que sería sensiblemente menos sensible a los outliers.

Empíricamente (ver los dos siguientes apartados) se notó que los mejores resultados se obtenían llevando a cabo la comparación con el segundo método para los momentos de Fourier y el tercero para los restantes.

2.3.3.5.- Momentos de Zernike

Los momentos complejos de Zernike son otra forma de alcanzar invarianza a traslación, escala y rotación. Estos momentos usan un sistema de coordenadas con origen en el centroide del objeto y escalado de forma de forma que el objeto esté encerrado dentro del círculo unidad.

La transformación se basa en los polinomios complejos de Zernike, que son ortogonales dentro del círculo unidad[18]. En términos generales, los polinomios de Zernike se clasifican en pares e impares, donde los pares están definidos como productos de polinomios radiales y funciones coseno, mientras que los impares son productos de polinomios radiales y funciones seno. En la práctica presentan menos sensibilidad al ruido causado por la discretización que los momentos de Hu y pueden ser calculados para cualquier orden, por lo que permiten la reconstrucción del objeto con una precisión arbitraria.

De igual forma en el caso anterior, los valores de los momentos devueltos son, típicamente, de un orden extremadamente bajo, por lo que para compararlos debidamente se aplica una transformación logarítmica decimal para que la escala no provoque comportamientos erráticos debido a la comparación de números de punto flotante extremadamente pequeños y se normalizan al rango [0, 1] encontrando el valor del mayor momento en cada vector (relativo a la pieza de referencia y a la actual) y multiplicando su recíproco por uno de los momentos.

En la implementación del método `Procesador::calcularMomentosZernike` se computan los momentos de Zernike de hasta orden 5. Primeramente se calcula el radio R y centro del círculo unidad como el centro de la imagen y la mitad del mínimo entre el número de filas y columnas de la imagen. Luego, para cada par (m, n) (siendo m y n número no negativos con n ≥ m) de los momentos y para cada píxel dentro del círculo se calcular sus coordenadas polares (ρ, θ) (ρ se calcula mediante la distancia euclídea [norma vectorial] del punto al centro del círculo), se normaliza ρ dividiéndolo por R obteniendo un

[18] El disco abierto unidad alrededor de P (donde P es un punto del plano euclídeo), es el conjunto de puntos cuyas distancias desde P son menores que 1: $D_1(P) = \{Q : |P - Q| < 1\}$. El disco cerrado unidad tiene la misma definición pero con un "mayor o igual que".

valor dentro del rango [0, 1] y se computa y acumula el momento de Zernike actual según la fórmula:

$$z_{k_{mn}} = z_{k-1_{mn}} + F(x,y) \cdot R^n \cdot \cos(m \cdot \theta)\,, \tag{59}$$

donde, $z_{k_{mn}}$ denota el momento en el instante actual, $z_{k-1_{mn}}$ en el instante anterior (crece acumulativamente, partiendo de cero), $F(x,y)$ el valor de la intensidad del píxel actual normalizado al rango [0, 1] (dividido entre 255 si la precisión es de 8 bits con un formato sin signo) y θ es el ángulo polar de las coordenadas del punto (ángulo que encierra el corte entre la línea que une el punto y el centro del círculo y el eje horizontal de la imagen). Se puede notar que, por simplicidad, se implementan los términos pares de los polinomios de Zernike. Tras calcular todos los momentos para los pares (m, n) especificados, se devuelve un vector de momentos que se puede comparar a posteriori con otro semejante.

2.3.3.6.- Momentos de Fourier

Como resultados del cálculo diferencial, el perímetro de una curva arbitraria se puede representar por su curvatura instantánea en cada punto del perímetro. Considerando una curva continua cerrada (como es el contorno de una pieza segmentada) dibujada en el plano complejo, un punto del perímetro se puede representar mediante coordenadas polares como una función $z(s) = x(s) + iy(s)$ que es combinación de las partes real e imaginaria del punto de la curva y depende de la longitud del arco "s".

El ángulo tangente $\Phi(s)$ en términos de la función compleja $z(s)$ según la ecuación:

$$\Phi(s) = \tan^{-1} \frac{dy(s)/ds}{dx(s)/ds} \tag{60}$$

y la curvatura del contorno en el punto es una función real dada por la llamada "función de curvatura":

$$k(s) = \frac{d\Phi(s)}{ds}, \tag{61}$$

haciéndose notar que los puntos de coordenadas $x(s)$ e $y(s)$ pueden obtenerse a partir de la función de curvatura mediante la llamadas "fórmulas de reconstrucción":

$$x(s) = x(0) + \int_0^s k(\alpha) \cdot \cos\Phi(\alpha) d\alpha \tag{62}$$

$$y(s) = y(0) + \int_0^s k(\alpha) \cdot \text{sen}\,\Phi(\alpha) d\alpha\,, \tag{63}$$

que representan la acumulación de la posición del punto complejo a lo largo de la longitud del arco "s", donde $x(0)$ e $y(0)$ son las posiciones de partida de las coordenadas del punto.

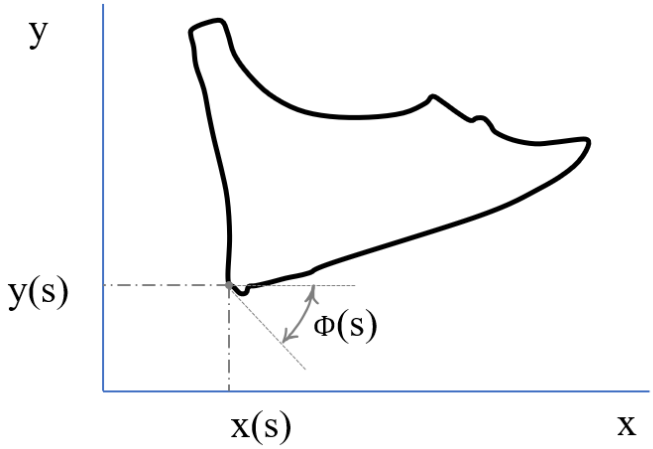


Fig. 65. Geometría de la pieza como una curva cerrada simple en el plano de Argand.

Aprovechando el hecho de que la función de curvatura es periódica a lo largo del perímetro P, se puede expresar en términos de una serie de Fourier como:

$$k(s) = \sum_{n=-\infty}^{\infty} \left(\frac{1}{P} \int_0^P k(s) \cdot e^{-\frac{2\pi \cdot i \cdot n}{P}} ds \right) \cdot e^{-\frac{2\pi \cdot i \cdot n \cdot s}{P}}, \tag{64}$$

expansión que se puede truncar por un número arbitrario de los primeros términos para obtener un conjunto de descriptores de Fourier que representan de manera aproximada al contorno. De esta manera se puede conseguir un análisis de los detalles (altas frecuencias) como de la forma general (bajas frecuencias) de un contorno, permitiendo una comparativa muy completa de la forma de dos piezas [46, pág. 613]. La discretización de la función de curvatura asume que una derivada de una función con respecto a "s" es la diferencia entre el valor de la función evaluada sobre el punto actual y anterior y viene dada por:

$$k(s_j) = \Phi(s_j) - \Phi(s_{j-1}), \tag{65}$$

donde s_j representa el paso "j" en la posición sobre el arco partiendo del punto de partida.

Cabe mencionar que si la curva presenta discontinuidades agudas (como esquinas rectangulares), la función de curvatura está definida en los vértices, aunque el empleo de una función acumulativa de forma (presentada a continuación) puede superar esta dificultad:

$$\theta(s) = \int_0^s k(\alpha) d\alpha - \frac{2\pi \cdot s}{P} \tag{66}$$

De igual forma en el caso anterior, los valores de los momentos devueltos son, típicamente, de un orden extremadamente bajo, por lo que para compararlos debidamente se aplica una transformación logarítmica decimal para que la escala no provoque comportamientos erráticos debido a la comparación de números de punto flotante extremadamente pequeños y se normalizan al rango [0, 1] encontrando el valor del mayor

momento en cada vector (relativo a la pieza de referencia y a la actual) y multiplicando su recíproco por uno de los momentos. Más aún, primeramente se tiene en cuenta que algunos momentos devueltos pueden ser exactamente cero, con lo que para evitar indeterminaciones al comparar cada vector de momentos se elimina cada cero (o aproximadamente cero) y el relativo del vector parejo que esté en la misma posición que el cero eliminado, suponiendo una relación inyectiva entre los vectores ya que el orden calculado y el tamaño de las imágenes son los mismos.

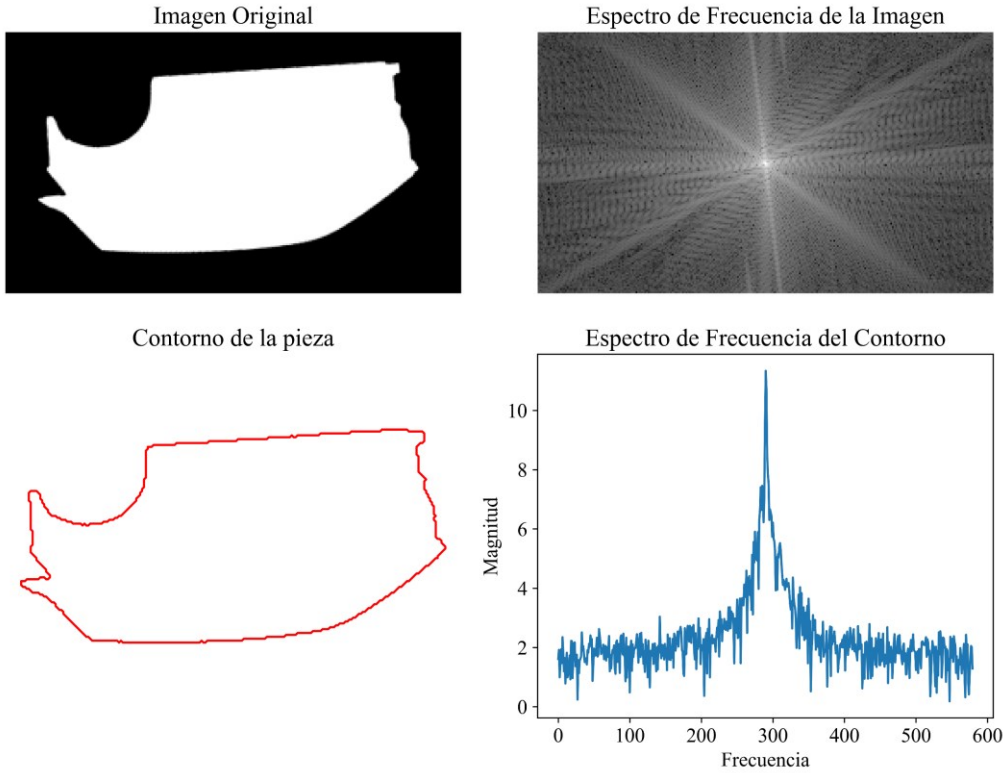


Fig. 66. De izda. a dcha. y de arriba a abajo: contorno segmentado de la cuarta pieza, representación de la magnitud frecuencial de la anterior imagen, contorno extraído de esta pieza (tratado como una señal 1D) y espectro de frecuencias de dicha señal.

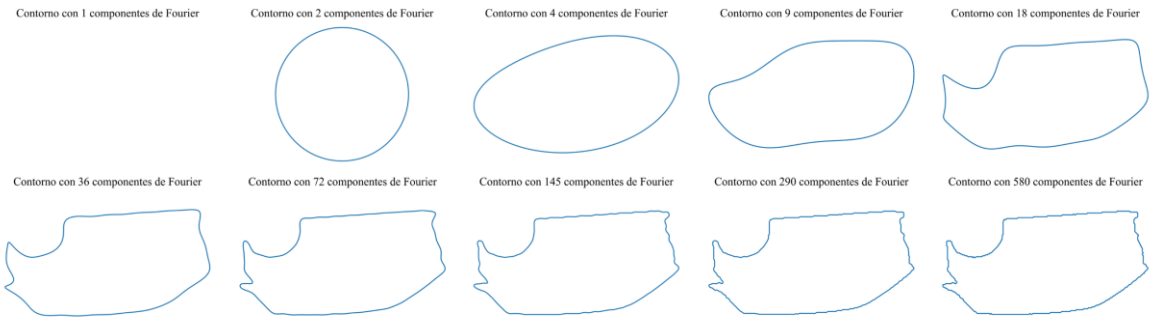


Fig. 67. Descriptores de Fourier para la cuarta pieza (transformada inversa usando las *n* mayores componentes); cada cual agrega más frecuencias y, por tanto, más detalle.

La fórmula general de la transformada de Fourier discreta es realmente la más relevante a la hora de implementar esta métrica, que viene dada por

$$F(k) = \sum_{n=0}^{N-1} f(n) \cdot e^{\frac{-2\pi i k n}{N}} = \sum_{n=0}^{N-1} f(n) \cdot \left(\cos\left(\frac{-2\pi k n}{N}\right) + i\,\mathrm{sen}\left(\frac{-2\pi k n}{N}\right)\right), \quad (67)$$

donde $F(k)$ es la transformada de Fourier en la frecuencia "k" (el orden del momento), $f(n)$ es la función de entrada en el dominio del tiempo (el valor normalizado de un píxel de la imagen) y N es el número total de muestras.

En la implementación del método `Procesador:: calcularMomentosFourier` se computan los momentos de Fourier de hasta orden 10. Primeramente se inicializan a cero las componentes compleja y real de los momentos. Para cada orden a calcular, se itera sobre cada píxel de la imagen y se calcula el ángulo tangente (la fase de la DFT bidimensional) en el punto basándose en la posición del píxel y el valor del orden del momento actual:

$$\Phi = -2\pi k \frac{n}{N} = -2\pi k \frac{x + y \cdot cols}{rows \cdot cols}, \quad (68)$$

donde "k" es el índice de la frecuencia específica (el orden del momento), "x" e "y" las coordenadas del píxel en la imagen y "cols" y "rows" el número de columnas y filas de la imagen, respectivamente. Esta manera de representar la imagen normaliza las coordenadas mediante el producto $rows \cdot cols = N$ (número total de muestras), donde $x + y \cdot cols = n$ transforma la posición de cada píxel muestreado en una representación unidimensional de la imagen.

Luego se normaliza el valor del píxel a un rango [0, 1] y se aplica la definición de la transformada de Fourier discreta para actualizar el valor de los momentos en cada iteración y, finalmente, se registra la parte real e imaginaria de todos los momentos de tipo `std::complex<double>` en el vector `std::vector<double> momentosFourier`:

$$f_{R_k} = f_{R_{k-1}} + F(x,y) \cdot \cos(\Phi) \quad (69)$$

$$f_{I_k} = f_{I_{k-1}} - i \cdot F(x,y) \cdot \mathrm{sen}(\Phi), \quad (70)$$

donde f_{R_k} y f_{I_k} denotan los momentos real e imaginario en el instante actual (respectivamente), $f_{R_{k-1}}$ y $f_{I_{k-1}}$ en el instante anterior (crecen acumulativamente, partiendo de cero) y $F(x,y)$ el valor de la intensidad del píxel actual normalizado al rango [0, 1] (dividido entre 255 si la precisión es de 8 bits con un formato sin signo). Se nota que lo que se hace es multiplicar los valores de intensidad por la parte real e imaginaria de la exponencial compleja asociada del cálculo de la DFT.

2.3.3.7.- Correlación de fase mediante espectro de potencia

En este apartado se realiza una implementa de la correlación cruzada de entre dos piezas en el dominio frecuencial. La correlación de fase es un método de comparación de similitud entre dos imágenes que lleva a cabo un análisis en el dominio de frecuencias de

estas. La fase de una señal o imagen describe la posición relativa de sus componentes en términos de ciclos completos de oscilación o periodos; en el caso de imágenes, la fase se refiere a la posición relativa de patrones repetitivos. En vez de comparar mediante correlación cruzada las amplitudes de las señales, es decir, los valores de intensidad de los píxeles, mediante este método se está comparando la fase de las señales, lo que puede ser especialmente beneficioso para buscar diferencias muy finas o detallistas entre dos objetos. A priori, puede ser que este método no sea el más indicado para identificar las piezas, donde la información más generalista tiene gran valor, pero desde luego puede ayudar a construir una respuesta final más fiable.

De esta manera, mediante la correlación de fase se puede estimar efectivamente el desplazamiento relativo y traslacional entre dos imágenes similares, y aproximadamente entre dos imágenes en general. El procedimiento se centra en aislar la información de fase de la representación en el espacio de Fourier del correlograma cruzado de las imágenes e implica aplicar una función de ventana a las imágenes (para mitigar la presencia de desplazamientos lineales en lugar de desplazamientos circulares), calcular la transformada discreta de Fourier 2D, obtener el espectro cruzado de potencia y determinar la correlación cruzada normalizada; a veces se emplean técnicas subpíxel para mejorar la precisión métodos de interpolación parabólica o basados en el centroide. Finalmente, la transformada inversa de Fourier da como resultado una delta de Kronecker, que representa un único pico. El método es resistente al ruido y a las oclusiones en las imágenes, además se puede extender para determinar diferencias de rotación y escala. Mediante el uso de las regiones de interés de las imágenes, en vez de los fotogramas enteros, se alivia el efecto de los desplazamiento lineales entre las piezas, que afectarían negativamente al algoritmo.

Las aplicaciones de la correlación de fase incluyen la conversión de estándares de televisión, donde se prefiere por su capacidad para minimizar artefactos. La técnica tiene un uso generalizado en diversos campos, ofreciendo ventajas en escenarios con desafíos inherentes como ruido o irregularidades en las imágenes.

Como se comentó en el apartado anterior, la transformada de Fourier descompone una imagen en sus componentes seno y coseno; en otras palabras, mapea una imagen de su dominio espacial a su dominio de frecuencias. Matemáticamente, es una forma de representar aproximadamente cualquier función en una suma de estos términos:

$$\mathcal{F}(x,y) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} f(x,y)e^{-2\pi i\left(\frac{mx}{N}+\frac{ny}{N}\right)} = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} f(x,y)\left(\cos\frac{2\pi imx}{N} + i\sin\frac{2\pi imx}{N}\right), \quad (71)$$

donde $f(x,y)$ es el valor de la imagen en su dominio espacial y $\mathcal{F}(x,y)$ en su dominio frecuencial; su visualización es posible mediante una imagen real y una imagen compleja o mediante una imagen de magnitud y una imagen de fase, mas en este caso solamente resulta interesante la imagen de magnitud, ya que contiene toda la información necesaria para analizar la estructura geométrica de las imágenes (si se transformara en el dominio

frecuencial deliberadamente y se quisiera regresar al espacial habría que emplear también la fase).

En el método `Procesador::calcularCorrelacionFase` primeramente se convierten las imágenes a una precisión de 32 bits con punto o coma flotante y en escala de grises para realizar las operaciones con números complejos adecuadamente. Luego se les aplica la transformada de Fourier discreta mediante la función `cv::dft` para obtener una representación de las imágenes en el dominio frecuencial. Posteriormente se multiplican elemento a elemento las imágenes complejas correspondientes a los espectros frecuenciales conjugando el segundo elemento para obtener un espectro de correlación cruzada, de acuerdo con la definición de producto interno o producto escalar para vectores complejos[19]. Luego se aplica la transformada inversa de Fourier a este resultado mediante `cv::idft` para obtener la imagen de correlación en el dominio espacial, se normaliza en el rango [0, 1] con `cv::normalize` y finalmente se busca el pico máximo de correlación para formar una primera idea de la similitud entre ambas imágenes bajo este criterio. Otra opción como medida de la similitud global sería quedarse con la primera frecuencia, que se corresponde (tal como lo devuelve la función) con la frecuencia cero (componente de continua) de la imagen, que es esencialmente el promedio de todos los valores de los píxeles en la imagen. En cambio, una manera mejor de analizar el resultado contenido en la imagen de correlación de fase en el dominio espacial es considerar no solo el pico de máxima amplitud, sino comprobar que la vecindad que lo rodea, en todas direcciones, es suficientemente grande, pues mayor será el parecido entre las piezas a nivel de detalle fino y grueso cuanto mayores valores presente la vecindad, en tanto en cuando la vecindad sea más amplia y todos los valores que contenga (o al menos la mayoría) superen los umbrales considerados.

Cabe mencionar que, por conveniencia, se modifica la representación frecuencial de los espectros de magnitud bidimensionales para situar la frecuencia cero en el centro de la imagen y las frecuencias más altas en las esquinas de la imagen. Se intercambian los cuadrantes de la imagen de tal forma que el cuadrante superior izquierdo se mueva al cuadrante inferior derecho, y viceversa, y lo mismo para los otros dos cuadrantes. Esto se hace para mayor claridad visual, aunque realmente las cuatro partes son simétricas y un solo cuadrante contiene toda la información frecuencial de la imagen, del mismo modo que el dominio frecuencial de una señal 1D es simétrico horizontalmente a partir de la frecuencia de muestreo.

La salida de esta métrica es el booleano `faseCorrSimilarity`, que indica si la pieza es válida a ojos de este sub-bloque.

[19] El mapeo $\mathbb{C}\ x\ \mathbb{C}\ \rightarrow \mathbb{C}$ definido por $(z, w) \longmapsto z \cdot \overline{w}$ es el producto interno estándar en el conjunto de los complejos $\mathbb{C}$.

2.3.3.8.- HOG

El descriptor HOG (*Histogram of Oriented Gradients*) es una técnica que captura la distribución de gradientes de intensidad en una imagen, lo que proporciona información sobre las texturas y formas presentes. La idea de HOG implica componer histogramas calculando gradientes de celdas (realizados utilizando convoluciones con operadores de derivadas) dentro de bloques que subdividen la imagen y luego normalizar estos bloques para obtener el descriptor final. La magnitud y fase de cada gradiente en la dirección horizontal y vertical se acumulan construyendo un histograma de orientaciones para cada celda que es ponderado por la magnitud del gradiente; finalmente las celdas se agrupan en bloques que son normalizados entre sí para refinar la invarianza a la ganancia de iluminación y al contraste.

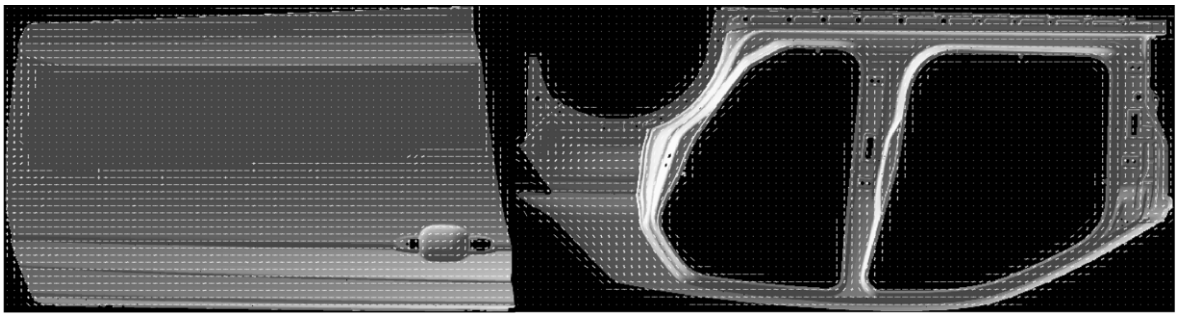


Fig. 68. Representación del descriptor HOG para dos de las piezas analizadas.

La clase `cv::HOGDescriptor` proporciona el método `compute` para computar este descriptor tomando como entrada una imagen `cv::Mat` y una variable de tipo `std::vector<float>` en el que se vuelca por referencia el resultado. De entre los parámetros que permite personalizar están `winSize` y `blockSize`, que definen el tamaño de la ventana de detección y el bloque utilizado para la normalización (64 x 64 y 8 x 8 píxeles, respectivamente), `blockStride`, que indica el paso o desplazamiento del bloque durante la normalización (4 x 4), `cellSize`, que define el tamaño de la celda unidad para la extracción de gradientes (4 x 4) y `nbins`, referido al número de "bins" o subregiones en el histograma de gradientes resultante. Para comparar los histogramas de la pieza actual y la de referencia se emplea el método `cv::compareHist` empleando una métrica basada en la distancia de Bhattacharyya (ver apartado "Perfil radial polar y GLOH").

El resultado final es un valor real cercano a 0 si las imágenes son poco similares e igual a 1 son exactamente iguales. La salida de esta métrica es el booleano `hogSimilarity`, que indica si la pieza es válida a ojos de este sub-bloque.

2.3.3.9.- Template matching

El "template matching" es una técnica utilizada para encontrar la ubicación de un objeto (plantilla o "template") dentro de una imagen usualmente más grande; la idea es comparar la plantilla superponiéndola a regiones del mismo tamaño de la imagen a analizar, desplazando o trasladando la plantilla a lo largo y ancho de toda la imagen para

finalmente quedarse con la mejor coincidencia, que determinaría la posición de la plantilla en la escena estudiada. En este contexto, la correlación cruzada normalizada es el método comúnmente más usado como medida de similitud entre la plantilla y las regiones de la imagen.

De esta manera, como se trata de comparar dos piezas aisladas en sus regiones de interés, se reescala el fotograma actual al tamaño de la de referencia y usa como plantilla para computar directamente la correlación cruzada normalizada (`cv::TM_CCOEFF_NORMED`) entre ambas piezas a través del método `cv::matchTemplate`. Aunque es una medida de la que no se deben extraer resultados decisorios (como se explica en apartados posteriores relativos a la correlación), pues la normalización, en estas condiciones, solo compete al nivel de brillo y, además, las imágenes se deforman deliberadamente para confrontarlas, en los casos más parecidos proporciona resultados que son beneficiosos para dirigir el clasificador morfológico hacia una respuesta positiva o que, por lo menos, lo conduzca hacia el camino dominante de entre los mayores verdaderos positivos proporcionados por los otros sub-bloques.

Por esto, los umbrales de esta métrica se escogen (experimentalmente) relativamente altos, para que el bloque no sea demasiado "permisivo" en su capacidad de decisión (pues es en la práctica "muy difícil" que dé por válida una pieza). Los parámetros que rigen la capacidad de validación de esta métrica son el valor máximo de la matriz de correlación entre las imágenes, el determinante de su valor absoluto (tras hacer "zero padding" a la matriz rectangular), su media aritmética total y en una vecindad centrada en dicho máximo y el recuento de coincidencias que superan el umbral preestablecido. La salida es el booleano `correlationSimilarity`, que indica si la pieza es válida a ojos de este sub-bloque.

2.3.3.10.- SSIM

La medida del índice de similitud estructural (SSIM) es un método para predecir la calidad percibida de imágenes digitales, incluyendo fotografías, televisión digital y cine digital, cuyo propósito es medir la similitud entre dos imágenes y proporcionar una métrica basada en la suposición de que una de las imágenes es una referencia absoluta carente de compresión o distorsión [47].

Así pues, SSIM se utiliza para evaluar la calidad de una imagen o de un volumen en escala de grises usando otra imagen como la referencia de máxima calidad. Fue desarrollada para abordar las limitaciones de otras métricas tradicionales, como el error cuadrático medio (MSE) o la relación pico señal/ruido (PSNR) (que se puede relacionar con el gado de suciedad de la cámara), que no siempre coinciden con la percepción humana de la calidad de una imagen. Por esto, se emplean tres parámetros que capturan la esencia de cómo el sistema visual humano percibe una imagen: la luminancia (que representa la intensidad global de la imagen y se relaciona con la percepción de brillo), el contraste (que mide las diferencias locales de intensidad en la imagen y está vinculado a la

percepción de la nitidez) y la estructura (que representa la correlación cruzada local entre las dos imágenes y está relacionada con la percepción de la forma).

La información estructural $s(X,Y)$ que captura SSIM se refiere a la idea de que los píxeles tienen fuertes interdependencias entre sí, especialmente en su vecindad (cuando están espacialmente próximos); la información por luminancia $l(X,Y)$ se refiere al hecho de que las distorsiones en una imagen tienden a ser menos visibles en regiones más luminosas y la información por contraste $c(X,Y)$ se refiere al fenómeno por el que las distorsiones se hacen menos visibles cuando hay una textura significativa en la imagen o en la superficie de los objetos presentes en esta.

La expresión general de esta medida para dos ventanas cualesquiera de una imagen viene dada por la combinación multiplicativa de estos tres términos:

$$l(X,Y) = \frac{2\mu_X\mu_Y + C_1}{\mu^2{}_X + \mu^2{}_Y + C_1}, c(X,Y) = \frac{2\sigma_X\sigma_Y + C_2}{\sigma^2{}_X + \sigma^2{}_Y + C_2}, s(X,Y) = \frac{\sigma_{XY} + C_3}{\sigma_X\sigma_Y + C_3} \quad (72)$$

$$SSIM(X,Y) = l(X,Y)^\alpha \cdot c(X,Y)^\beta \cdot s(X,Y)^\gamma, \quad (73)$$

donde μ_X y μ_Y son las medias de las imágenes, σ_X y σ_Y son sus desviaciones estándar (raíces cuadradas positivas de las varianzas $\sigma^2{}_X$ y $\sigma^2{}_Y$), σ_{XY} su covarianza, C_1, C_2 y C_3 son tres constantes de bajo orden de magnitud que se utilizan para evitar indeterminaciones en caso de divisiones por cero y α, β y γ son tres exponentes enteros mayores que cero arbitrarios. Las constantes mencionadas se pueden parametrizar según otras tres constantes relacionados con la imagen:

$$C_1 = (K_1L)^2, C_2 = (K_2L)^2, C_3 = (K_2L)^2/2 \text{ (típicamente)}, \quad (74)$$

donde K_1 = 0,01 y K_2 = 0,03 por defecto (valores empíricos) y L es el rango dinámico de la intensidad de los píxeles (256 para imágenes de 8 bits). Y si $\alpha = \beta = \gamma = 1$ y $C_3 = C_2/2$, el índice simplifica a:

$$SSIM(X,Y) = \frac{(2\mu_X\mu_Y + C_1)(2\sigma_{XY} + C_2)}{(\mu^2{}_X + \mu^2{}_Y + C_1)(\sigma^2{}_X + \sigma^2{}_Y + C_2)}, \quad (75)$$

que es la ecuación que se decide implementar en el método `Procesador::calculateSSIM`. En este método se convierten las imágenes a formato en coma flotante de simple precisión (32 bits) para afinar sensiblemente la resolución de los resultados (aunque para el cálculo de la varianza se pueden emplear enteros para no sufrir de error de cancelación[20]), se calculan los valores intermedios de las imágenes como el producto entre ambas (que refleja cómo coinciden los valores de intensidad de sus píxeles) y con ellas mismas elemento a elemento, se calculan las medias de las imágenes y las medias ponderadas de las matrices anteriores aplicando convoluciones 2D mediante filtros

[20] Véanse otros algoritmos para calcular la varianza [58].

gaussianos (que suavizan los valores de la imagen y dan más peso a los píxeles centrales de la ventana del filtro en el cálculo), se computan las varianzas de las imágenes (restando el cuadrado de las medias de los cuadrados de las imágenes originales) y de covarianza entre ellas (restando el producto de las medias ponderadas de las imágenes de la del producto entre ambas) y se registra una matriz de coeficientes SSIM para cada píxel aplicando la ecuación anteriormente presentada. Finalmente, se promedia este mapa de coeficientes para obtener un único valor escalar real que representa la similitud estructural entre las dos imágenes de entrada.

A continuación se muestran las ecuaciones matriciales específicas que se implementaron programáticamente en el método:

$$SSIM = \frac{1}{N}\sum_{i=0}^{N-1}\frac{(2\cdot\mu_F\cdot\mu_G + 6{,}25)(2\cdot\sigma_{FG} + 58{,}5225)}{(\mu^2{}_F + \mu^2{}_G + 6{,}25)(\sigma^2{}_F + \sigma^2{}_G + 58{,}5225)} \tag{76}$$

$$\mu_F\cdot\mu_G = (F * \aleph)(G * \aleph) \tag{77}$$

$$\sigma_{FG} = \mu_F\mu_G - (G\cdot F) * \aleph \tag{78}$$

$$\sigma_F + \sigma_G = \big(F - (F * \aleph)\big)^2 * \aleph + \big(G - (G * \aleph)\big)^2 * \aleph \tag{79}$$

$$\sigma^2{}_F + \sigma^2{}_G = \big((F^2 * \aleph) - (F * \aleph)^2\big) + \big((G^2 * \aleph) - (G * \aleph)^2\big), \tag{80}$$

donde $F = F(x, y)$ y $G = G(x, y)$ son las dos imágenes, N es el número de píxeles de la imagen (tamaño de la matriz de coeficientes SSIM final) (el sumatorio indica que se calcula el promedio de los valores de la matriz resultante), μ la media y σ^2 la varianza (entendida como la diferencia entre la media [esperanza]) del cuadrado de la imagen y el cuadrado de la media de la imagen) de la imagen, σ_{FG} la covarianza (entendida como el valor esperado del producto de las desviaciones al cuadrado de las media de cada imagen) y $\aleph$ es un filtro de promediado gaussiano bidimensional (se emplea un kernel de 11 x 11 y desviación estándar 1,5) con el que se convolucionan las imágenes para calcular sus medias[21].

Aunque no esté exactamente pensado para la identificación de objetos comparando dos muestras, proporciona resultados algo valiosos pues pueden aprovecharse para detectar si, por efecto de un valor de similitud muy bajo, el frame actual presenta algún problema perceptual (mucho ruido o compresión en los frames que se reciben, un fallo en la cámara…), lo que se puede investigar aguas arriba tras recibir una señal de error por parte de este sub-bloque. O, por ejemplo, dadas dos imágenes de la misma pieza que se diferencian principalmente en que las altas frecuencias están mucho más presentes en una que en la otra, un resultado de baja similitud sería útil para determinar si la resolución de la cámara hubiera disminuido de manera imprevista a causa del desenfoque u otros motivos.

[21] Sabiendo: $Cov(X, X) = Var(X) = E[(X - \mu_X)\cdot(X - \mu_X)] = E[(X - E[X])^2] = E[X^2] - E[X]^2$

Así pues, el resultado final es un valor real que, si está normalizado, es cercano a 1 si las imágenes son similares e igual a 0 si no presentan similitud alguna. La salida de esta métrica es el booleano `ssimSimilarity`, que indica si la pieza es válida a ojos de este sub-bloque.

A partir de aquí, en lo relativo a las siguientes métricas, se lleva a cabo un análisis distinto usando del contorno de la pieza, más que de su forma y masa en sí misma (aunque con los momentos de Fourier se intuyó esta idea). Mediante el método `Procesador::procesaOrientedROI` se extrae la ROI orientada de la pieza (binarizada y en escala de grises) y se estima la orientación de esta mediante un análisis bidimensional de sus componentes principales (PCA).

Esta estimación grosera inicial de la orientación de la pieza (que solo puede dar resultados totalmente confiables si la segmentación de la pieza es absolutamente perfecta) es una fuente de información muy útil para el resto del algoritmo, pues usada adecuadamente se puede conseguir que algunos métodos sean invariantes a la rotación. Para ello se calcula la orientación principal de las piezas de referencia y la pieza actual y se efectúa una transformación de rotación pura a todas las ROI según dicha orientación para que coincida con el eje horizontal (0 º). De esta manera, si la pieza actual fuese efectivamente la misma que una de referencia, se podría proceder a analizar su similitud para identificarla indiferentemente a que haya entrado rotada en la cinta transportadora. Podría pensarse que si se dispone de la orientación PCA ya se dispone de la orientación final de la pieza en la cinta y se podría saber si se valida o no, pero por varios motivos esta orientación no da la mejor respuesta y se debe complicar más el algoritmo: por una parte es muy influenciable por la calidad de la segmentación de la pieza, que obligadamente tiende a variar demasiado frecuentemente, por otra parte aún es necesario saber si la pieza se identifica antes de que calcular su orientación sea útil de cara al exterior y, por último, una vez se identifique la pieza existen otros métodos mucho más robustos a cambios en la pose, iluminación y escala que PCA para calcular la orientación (como se explicará en el apartado del clasificador característico). Aunque la información de esta orientación es útil, por lo que al final del algoritmo se tendrá en cuenta como factor de ponderación antes de proporcionar la respuesta final cuando la pieza salga de la cinta (o entre otra pieza antes de que salga la primera, o se solicite desde el exterior que se lance la respuesta).

2.3.3.11.- PCA

A la hora de estimar a primera vista la orientación en el plano de una pieza, el sistema visual humano parece no tener en consideración aspectos como el nivel de intensidad de los puntos o las sombras que proyecta, el contraste entre sus regiones o la distancia relativa entre sus partes; esto son en principio atributos que se muestran de interés para situar un objeto en el espacio 3D. Con una dimensión menos, en lo que parece lógico fijarse más analíticamente es en la propia masa del objeto proyectando todos sus puntos sobre un plano de referencia común y estableciendo un criterio para orientar en cierto

sentido la pieza; por ejemplo, hacia el lado que resulte del uso convencional del objeto (si se reconociese), hacia la zona con más o menos área, etc.

PCA es una técnica de reducción de dimensionalidad recurrentemente empleada para lidiar con la conocida "maldición de la dimensionalidad"[22]. La idea es que el espacio de características o datos se aproxima mediante una distribución gaussiana multivariable para luego, usando las varianzas de la gaussiana a lo largo de cada dimensión, determinar la importancia de cada una de estas, descartando aquellas con baja varianza. Esta gaussiana logra alinear un nuevo sistema de coordenadas perfectamente con los ejes principales de los datos de entrada (aquellos que mejor expliquen la información de partida).

En este sentido, se puede ver que el PCA puede extraer las características principales de un conjunto de datos y que; si los datos son un conjunto de tuplas (x, y) espacialmente repartidas, devolverá los ejes (dos en el caso bidimensional) sobre cuya direcciones se agrupan más puntos, pues cada dimensión (horizontal y vertical) se corresponde con una característica de interés. Si los puntos estuvieran agrupados prácticamente en una línea recta, uno de los ejes (vector propio o "eigenvector") sería muy superior en magnitud (valor propio o "eigenvalue") al otro, por lo que se podría reducir la dimensionalidad del espacio de características en una unidad. Para estimar mediante este método la orientación de una pieza, se supone que el mayor de los vectores propios resultantes del análisis de la masa del objeto segmentado se corresponde con la direccion en la que sus puntos más varían en cierta dimensión y su magnitud indica la importancia de esa dirección.

El inicio de los vectores propios cae en el centroide del objeto. Aplicar PCA a la matriz de covarianza de una imagen plana binarizada (píxeles de dos dimensiones) da lugar a dos vectores de dos dimensiones con origen el centro de masas del objeto y cierta magnitud y dirección cada uno. La matriz de covarianza se puede calcular a partir de los datos normalizados mediante la siguiente expresión:

$$\Sigma = \frac{1}{n-1}\sum_{i=0}^{n-1}\left(p_{xy_i} - \overline{p_{xy}}_i\right)\left(p_{xy_i} - \overline{p_{xy}}_i\right)^T = \frac{1}{n-1}\sum_{i=0}^{n-1} XX^T, \tag{81}$$

donde "n" sería el número total de observaciones (píxeles de la imagen) y p_{xy} es el vector de datos de entrada (las coordenadas "x" e "y" de los píxeles); se substrae la media de la imagen de sí misma.

Simplificadamente, PCA sigue este algoritmo: construir una matriz de datos normalizados donde cada fila representa una observación (píxeles de la imagen) y cada columna una característica (dirección "x" e "y"), calcular la matriz de covarianza de los datos normalizados (que muestran cómo las variables cambian conjuntamente), calcular los autovectores y autovalores de la matriz de covarianza mediante la definición de valor

[22] En el ámbito del aprendizaje automático, se conoce como "maldición de la dimensionalidad" al fenómeno que suele aparecer cuando en un espacio de características o datos hay muchas dimensiones y, por tanto, exceso de peso en memoria, computación, tiempo de entrenamiento y disminución del rendimiento.

propio $C\vec{v}_i = \lambda_i\vec{v}_i$ (donde $\vec{v}_i$ es el i-ésimo autovector y λ_i el i-ésimo autovalor) (un valor que mantiene al vector paralelo tras la transformación), ordenar los autovectores de acuerdo con sus autovalores en orden descendente, construir la matriz de proyección $P = [\vec{v}_1 \vec{v}_2 \dots \vec{v}_k]$ que transformará los datos originales reduciéndoles (si cabe) la dimensionalidad (donde "k" es el número de componentes principales seleccionadas) y proyectar el los datos del espacio de características en el nuevo espacio de componentes principales, según $Y = X \cdot P$. Adicionalmente, se podría analizar la varianza explicada por cada cierto principal dividiendo el autovalor en concreto por la suma total de los autovalores, lo que da una idea de cuánta información se retiene al seleccionar un número específico de componentes principales (útil en aplicaciones de compresión de imágenes).

En el método `Procesador::orientedROI` se extrae de la orientación de una pieza mediante el análisis de componentes principales (PCA) y se define un procedimiento para estimar y elegir la orientación natural de una pieza basándose en la distribución de la masa de la pieza segmentada, a fin de decidir, además de aproximar su valor, si la pieza está orientada "a derechas" o "a izquierdas". Tras determinar el centroide de la masa del objeto segmentado y el ángulo de orientación principal, se define un vector director $\vec{v} = (cos\theta, sen\theta)$ y los puntos `lineStart = centerROI - minRect.size.width × v` y `lineEnd = centerROI + minRect.size.width × v` que representan una línea media que pasa por el centroide y se extiende a lo largo del vector de dirección principal, incidiendo perpendicularmente en las bases del rectángulo delimitador orientado. Luego, para cada píxel en el contorno rellenado, se calcula el producto vectorial entre el vector que une el inicio de la línea media ($\vec{u}$) y esta, definido como $\vec{u} \times \vec{v} = u_x \cdot v_y - u_y \cdot v_x$, lo que determina si el punto está a la izquierda ($\vec{u} \times \vec{v} > 0$, vector $\vec{v}$ en sentido antihorario a $\vec{u}$) o a la derecha ($\vec{u} \times \vec{v} < 0$, vector $\vec{v}$ en sentido horario a $\vec{u}$) de la línea, pues su magnitud representa el área del paralelogramo formado por los dos vectores y su signo la orientación del segundo con respecto al primero (este resultado se aplica de manera general a tres puntos para determinar si, además, están alineados o no [producto vectorial nulo]), acumulando así píxel a píxel el área resultante de cada mitad del objeto… así la flecha se orientará a la zona donde más área haya.

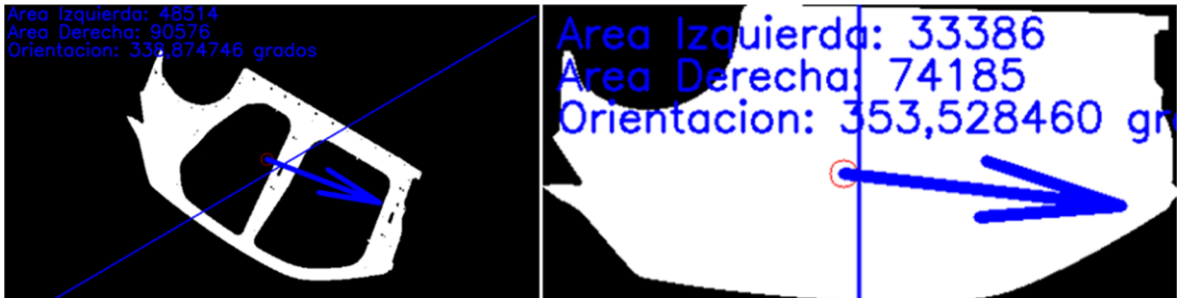


Fig. 69. Ejemplos del uso del producto vectorial para determinar el sentido de orientación natural de un objeto. La discordancia en la primera imagen se debe, además de al ruido, a que la flecha se estimó en base al área de la pieza rellenada y la línea media con el área útil de la pieza.

Primero se extrae el contorno externo de la pieza binaria enteramente segmentada mediante la función `cv::findContours` reduciendo los puntos redundantes y

comprimiéndolo, luego se asegura de tener el contorno externo que engloba toda la pieza manteniendo únicamente aquel más grande (el de mayor área) mediante `cv::countourArea`, luego se extraen todos los puntos del contorno en una variable de tipo `std::vector<cv::Point>`, se aplica una reducción de la dimensionalidad de los mismos usando PCA mediante la clase `cv::PCA` y dos componentes principales y se obtiene la media de los datos, los autovectores y autovalores para luego calcular la orientación del autovector más grande con respecto al eje horizontal mediante la inversa de la tangente del cociente entre las coordenada vertical y horizontal de dicho autovector. Además, para tener un marco de comparación común y algo más lógico para todas las piezas, se estima el cuadro delimitador orientado de la pieza empleando los vectores propios y la posición del centroide del objeto, se corta "a la mitad" el objeto según la perpendicular al autovector horizontal que pasa por el centroide y se computa área (número de píxeles en blanco) de la zona que queda a la derecha y a la izquierda dentro de esta región rectangular, antes de convertir la orientación a grados sexagesimales y ajustarla al primer y cuarto cuadrante en el rango (0, 90], [270, 360] si la masa del objeto está desequilibrada hacia la derecha o al primer y tercer cuadrante en el rango (90, 270) si lo está hacia a la izquierda, para tener un marco de comparación común para todas las piezas imaginando una orientación natural de las piezas. También se dibuja, con meros fines ilustrativos, el rectángulo que enmarca la pieza y las zonas en que la separa y una flecha que nace de su centroide y apunta a la dirección estimada. Aunque sabiendo que la línea en la dirección del segundo vector propio que pasa por el centroide ya divide la pieza aproximadamente a la mitad, igualitariamente, otra forma de ver la orientación natural podría ser aquella que quede bajo el (mayor) arco positivo a partir del eje horizontal.

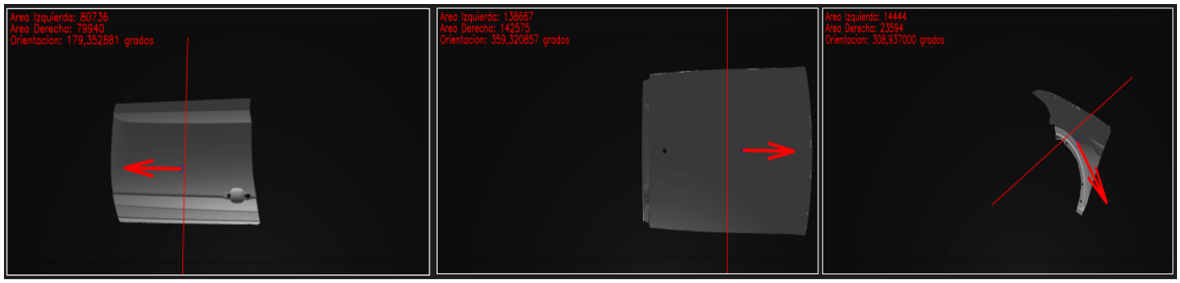


Fig. 70. Ejemplo de la estimación de la orientación de objetos mediante PCA. A priori la orientación natural se dirige a la parte de mayor masa, pero para compararla con las referencias se lleva al primer y segundo cuadrante y se mide con respecto a la orientación "cero" (la correcta).

Se ve que si se conoce la orientación de unas piezas de referencia "bien orientadas" es muy directo y sencillo validar la pose mediante este método de las piezas a analizar en tiempo real, pero no se considera una medida sumamente precisa porque necesita realizarse bajo unas condiciones ciertamente controladas y muy similares al caso de "entrenamiento". Pero sí proporciona una primera respuesta de la validez de la orientación de las piezas, que a posteriori se tendrá en cuenta para la respuesta que lanza el sistema en su conjunto al final del algoritmo. Si la situación fuera idílica y las piezas fuesen absolutamente siempre las mismas, y conocidas, siempre rotando estrictamente en el plano horizontal y no bruscamente, este método podría proporcionar la respuesta final, pero para complicar la

solución del problema (y porque se considera perjudicial estar sometido a unas restricciones tan poco laxas de cara a una implementación real) se usa como método auxiliar y no definitivo.

2.3.3.12.- Descriptores de Fourier

La transformada de Fourier responde naturalmente a un a manera de representar datos en la que las bajas frecuencias modelan su forma general y las componentes de alta frecuencia pueden corresponderse a datos que pueden no ser interés, como oclusiones o ruido que aparece debido a una resolución de muestreo muy grande y puede interferir con los detalles del objeto que realmente interesan. Con unas «pocas» componentes se puede llegar a tener una representación que captura la esencia del contorno de un objeto y compacta pero suficientemente significativa como para comparar distintas imágenes.

Como se comentó en el apartado de los momentos de Fourier, la secuencia de píxeles de un imagen puede ser tratada como una función vectorial de la longitud del arco. Y en el caso de un contorno cerrado la función es periódica, por lo que puede expandirse como una serie de Fourier mediante la DFT. La completitud y el detalle de representación crece con el número de componentes empleados; dependiendo de la complejidad del objeto puede bastar con más o menos coeficientes.

Se aplican varias técnicas de normalización para obtener invarianza a escala, rotación y traslación. Para invariancia a traslación es necesario encontrar el centroide de la forma y calcular el vector de cada punto del contorno que tenga el centroide como origen. Esto se hace, como se mencionó otras veces, restando las coordenadas “x” e “y” del centroide de las coordenadas de cada punto. Para la invariancia a escala de necesita encontrar la máxima magnitud de cada vector (ya normalizado para la invariancia a traslación) y dividir la magnitud de cada punto por ese valor de máxima magnitud. Una forma alternativa de conseguirlo es dividir cada coeficiente de Fourier por el coeficiente de frecuencia cero (primer coeficiente o componente de continua) ya que la información de escala está ahí representada. Por último, la invariancia a rotación se puede conseguir manteniendo sólo la magnitud de los coeficientes de Fourier [48]. Además de todo esto, hay que tener en cuenta que para aplicar la distancia euclídea a la comparación de descriptores, la longitud del descriptor para cada forma debe ser la misma. En la DFT, el número de coeficientes finales depende del número de puntos de contorno de la forma; la solución implementada consiste en interpolar entre puntos o submuestrear el contorno para alcanzar un número fijo de puntos para cada forma, a decir, 360.

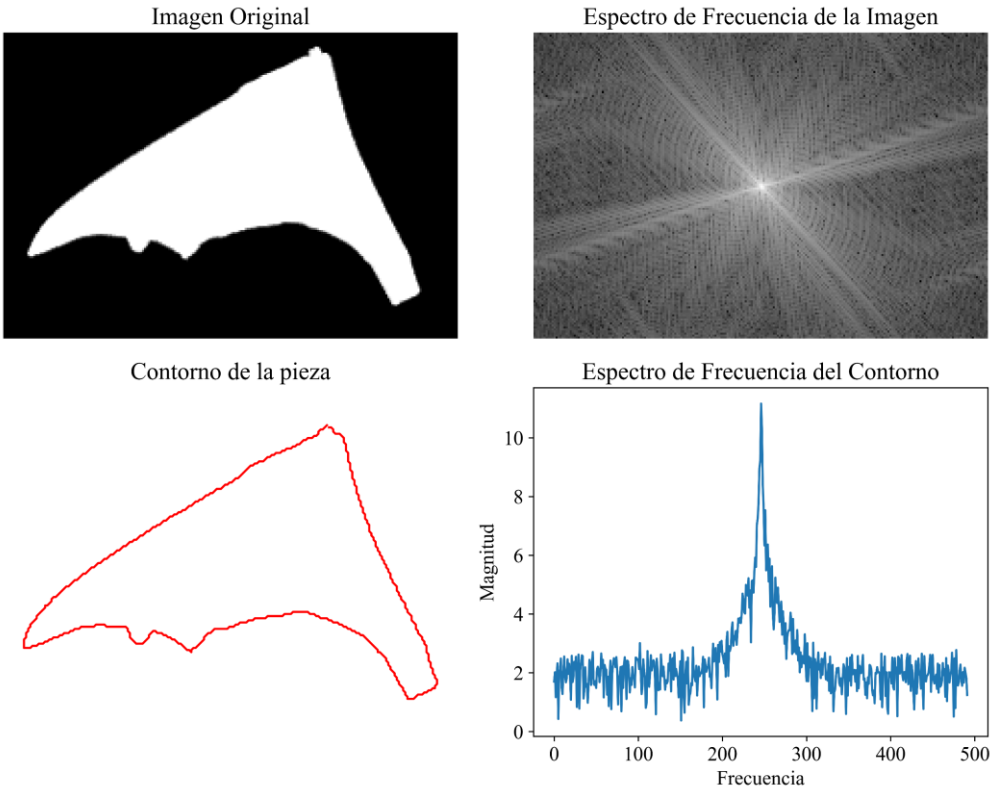


Fig. 71. De izda. a dcha. y de arriba a abajo: contorno segmentado de la segunda pieza, representación de la magnitud frecuencial de la anterior imagen, contorno extraído de esta pieza (tratado como una señal 1D) y espectro de frecuencias de dicha señal.

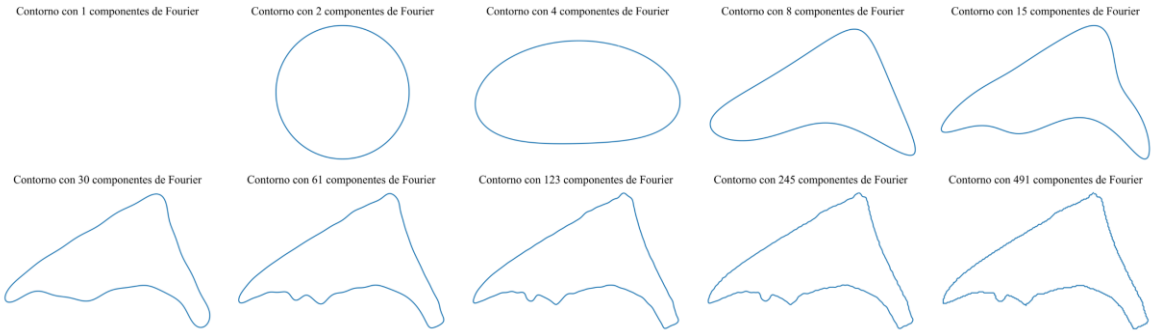


Fig. 72. Descriptores de Fourier para la segunda pieza (transformada inversa usando las *n* mayores componentes); cada cual agrega más frecuencias y, por tanto, más detalle.

Se emplea el útil método de los módulos adicionales de OpenCV `cv::ximgproc::ContourFitting::estimateTransformation` para realizar la comparación mediante los descriptores de Fourier del contorno de la pieza actual y referencia. El constructor de la clase acepta el número de descriptores de Fourier a emplear para el contorno objetivo (igual al número de puntos del contorno después de que lo remuestree) (1024, p. ej.) y el número de descriptores de Fourier que definen el segundo contorno (16, p. ej.), que tratará de ajustar al objetivo. Este método aproxima ambos contornos cerrados minimizando la distancia entre ellos, que viene definida mediante la siguiente fórmula:

$$d(C_A, C_B) = \sum_{n=0}^{N-1} \left(a_n - s b_n e^{j(n\alpha+\varphi)}\right)^2, \tag{82}$$

donde C_A y C_B son los contornos cerrados, a_n y b_n son sus descriptores de Fourier, "s" es el factor de escala, α el factor de ajuste del punto de inicio de la curva y φ el ángulo de rotación final entre los contornos, que resultan tras haber aproximado ambos entre sí.

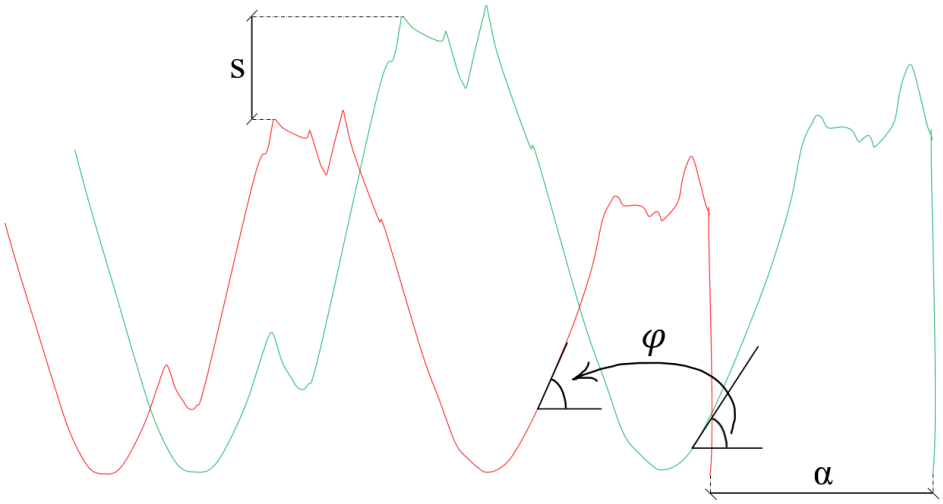


Fig. 73. Factores que trata de ajustar el método de los descriptores de Fourier basándose en dos contornos.

El método devuelve los valores finales tras la transformación que aproxima los contornos minimizando la anterior función de coste. Lo que resta es establecer unos valores umbrales empíricos que evalúen cada parámetro resultante con los que determinar si según está métrica los objetos son, o no, similares. La salida de esta métrica es el booleano `fourierContourSimilarity`, que indica si la pieza es válida a ojos de este sub-bloque.

2.3.3.13.- Perfil radial polar y GLOH

En este sub-bloque se transforman las imágenes de un marco de coordenadas cartesiano a polar y luego se computan el histograma de localización y orientación del gradiente (GLOH) de cada una para comparar los vectores de cada histograma mediante distancia de Bhattacharyya. El cambio a coordenadas polares supone que los cambios en rotación de las piezas en el plano de la cinta transportadora se convierten en cambios de traslación en las imágenes, lo que elimina la varianza a orientación y facilita el análisis de las piezas en imágenes a lo largo de una mismo eje; como las imágenes que sirven de entrada a este tipo de sub-bloques son la regiones de interés de las piezas completamente recortada de las originales y aislada del fondo, se consigue invarianza a traslación y, mediante otros métodos ya comentados (y teniendo en cuenta que realmente viene dada intrínsecamente, pues la cámara está fija y las piezas no varían en tamaño), a la escala.

En la siguiente figura se puede observar que los cambios de orientación en el espacio cartesiano se convierten en desplazamientos unidimensionales y deformaciones en el

espacio polar, lo que facilita el análisis del contorno de las piezas en el contexto del uso de técnicas de correlación mediante distancia como las que se están comentando.

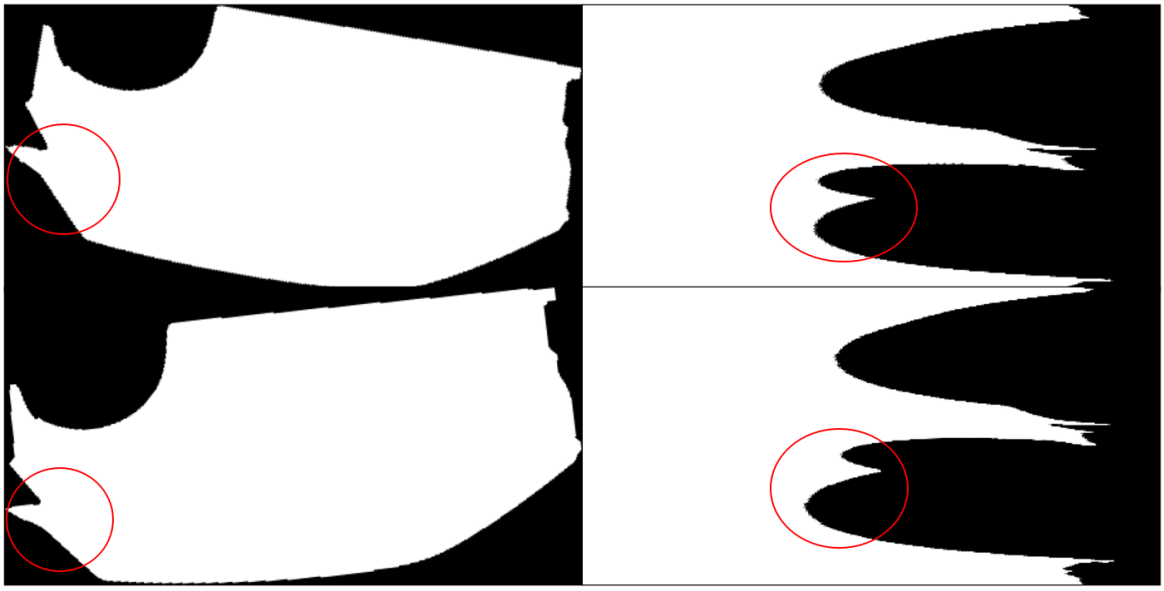


Fig. 74. Dos piezas segmentadas en su bounding box sin orientar y su perfil polar a la izquierda.

La distancia de Bhattacharyya o Hellinger es una medida estadística que cuantifica la similitud entre dos distribuciones de probabilidad. Esta métrica se deriva particularmente del coeficiente de Bhattacharyya, definido como:

$$BC(\mathrm{p},\mathrm{q}) = \sum_{x \epsilon X} \sqrt{p(x) \cdot q(x)}\,, \tag{83}$$

donde X es el conjunto de valores que puede tomar la variable aleatoria "x", "p" y "q" son dos distribuciones de probabilidad discretas y $p(x)$ y $q(x)$ son las probabilidades de ocurrencia en sendas distribuciones. En este contexto de procesamiento de imagen, "p" y "q" se pueden considerar como los histogramas de las imágenes, normalizados para obtener la probabilidad de ocurrencia de cada "bin" (cada nivel de intensidad de los píxeles). Así, la siguiente ecuación es la que define la distancia entre ambos, que sirve como métrica de similitud o semejanza entre sus distribuciones de intensidad:

$$D_B(\mathrm{p},\mathrm{q}) = -ln\big(BC(\mathrm{p},\mathrm{q})\big) \tag{84}$$

Para esta labor se emplea la función `cv::compareHist`, en cuyos argumentos se da la posibilidad de especificar un flag para implementar este método (y otros como la correlación lineal, la prueba chi-cuadrado o la divergencia de Kullback-Leibler). Aunque antes de esto se normalizan los histogramas dividiendo cada elemento entre el tamaño total. También se considera ecualizar los histogramas mediante la función `cv::equalizeHist` para hacer que la comparación sea más robusta frente a pequeñas variaciones en la iluminación, redistribuyendo las intensidades de los píxeles para que abarquen todo el rango dinámico disponible (mejorando el contraste y haciendo más

notables las diferencias de intensidad), es decir, logrando una distribución más uniforme o equitativa de las intensidades para todos los píxeles de la imagen, pero se prefiere no mantener esta técnica en la implementación final para no añadir información artificial a las imágenes originales, que se consideran bajo un entorno de experimentación suficientemente bien controlado. Cuanto menor es el valor (entre 0 y 1) devuelto por esta métrica, más parecidas son las piezas.

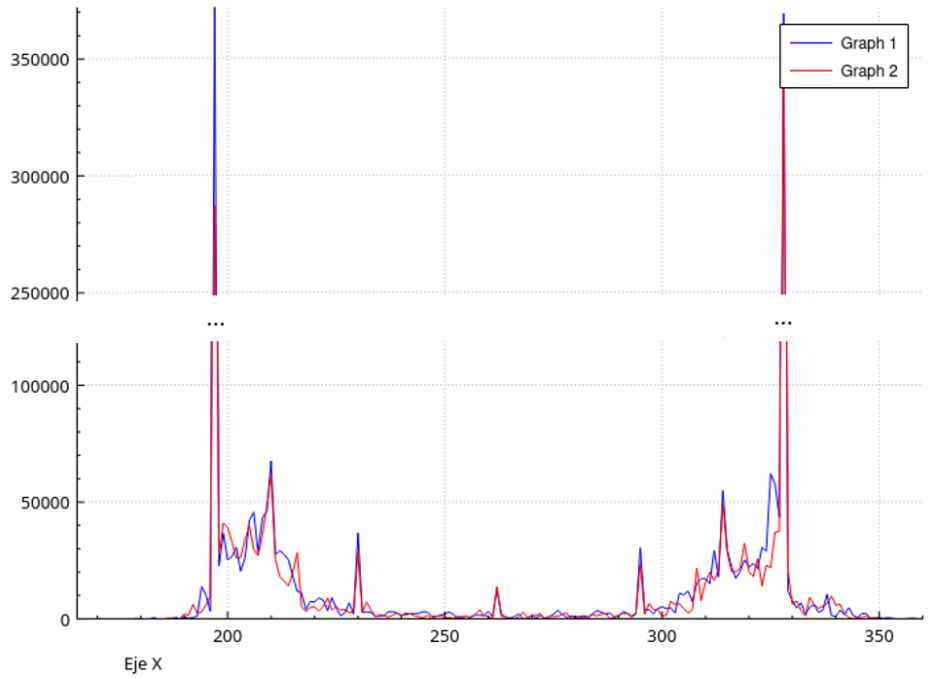


Fig. 75. Comparación del histograma de gradientes orientados de dos perfiles de piezas con diferente pose convertidos al mapa de coordenadas polar.

En la figura anterior se observa una forma, en general, similar entre ambas piezas, aunque un análisis numérico crudo puede complicarse debido a los órdenes de magnitud que se tratan; esto se maneja empleando las distancias logarítmicas comentadas e introduciendo otras métricas (ver siguientes apartados) que tienen en cuenta un factor de forma, más que de distancia.

2.3.3.14.- Señales 1D

Manteniendo la idea de los momentos de la región binaria y el PCA en mente, se computa un tipo de señal unidimensional (en forma de array dinámico `std::vector<double>`) empleando el centroide de la región segmentada y su contorno, tanto para la pieza actual como para cada referencia. Esta señal 1D es la distancia (subpíxel) del centroide a cada punto del contorno en dirección antihoraria. Se tiene cuidado de evitar zonas y contornos con dobles cruces (contornos internos) exigiendo que se cuenten los píxeles en blanco al computar la distancia, lo que es similar a exigir que se tenga en consideración solamente el contorno exterior.

Además, en el proceso de codificación de las señales se resta el centroide de cada punto del contorno para conseguir invarianza a la traslación, aunque ya estaba casi totalmente asegurada usando como imagen de entrada la ROI ajustada del objeto.

A estas señales 1D se les aplica una interpolación mediante el promediado de todos los puntos adyacentes, a fin de afrontar la pérdida de datos en su cómputo y para aumentar su resolución. El método `Procesador::interpolateMidpoints` se encarga de ello, tomando como argumento los datos a interpolar y el número de puntos intermedios a generar entre cada par de puntos dados, aunque en período de experimentación también se hicieron pruebas mediante splines cúbicos, B-splines y el filtro de Savitzky-Golay.

La interpolación mediante la fórmula de Lagrange [49, pág. 477] es otra alternativa apropiada de interpolar la señal para suavizarla (con cualquier número de puntos) basándose en su tasa de cambio inmediata, para muestrear fidedignamente valores que no fueron explícitamente capturados. En contraposición con otros métodos de interpolación, como el de Newton, en este caso cada término individual contiene uno y solo un valor de los dados por la función, lo que supone un beneficio por el carácter discreto de estas señales 1D. Además, la fórmula interpola la función explícitamente, sin necesidad de métodos recursivos; el propósito de esta idea sería arreglar la forma de algunos contornos, como los que se ve en la siguiente figura (extraída mediante la biblioteca "QCustomPlot"), que se pudieran no haber perfilado correctamente:

Fig. 76. Ejemplo de contornos superpuestos de dos piezas de un mismo tipo rotadas; la interpolación ayudaría a completar su forma para que no haya saltos bruscos y se puedan discretizar más precisamente.

Se define un polinomio de orden (n + 1):

$$\varphi(x) = (x - x_0)(x - x_1) \dots (x - x_n), \tag{85}$$

correspondiente a los puntos dados x_y. Diferenciando el polinomio mediante la regla del producto y sustituyendo x sucesivamente por sus raíces se llega a la expresión:

$$\frac{\varphi(x)}{(x - x_0)\varphi'(x_v)} = \frac{(x - x_0) \dots (x - x_{v-1})(x - x_{v+1}) \dots (x - x_n)}{(x_v - x_0) \dots (x_v - x_{v-1})(x_v - x_{v+1}) \dots (x_v - x_n)}, \tag{86}$$

que es un polinomio de grado n tal que en el punto $x = x_y$ toma el valor 1 y 0 en el resto de puntos x_i. Así pues, mediante la fórmula de interpolación de Newton [49, pág. 473] se puede obtener inmediatamente la siguiente expresión:

$$\phi(x) = \varphi(x)\left[\frac{f_0}{(x-x_0)\varphi'(x_0)} + \frac{f_1}{(x-x_1)\varphi'(x_1)} + \cdots + \frac{f_n}{(x-x_n)\varphi'(x_n)}\right], \quad (87)$$

que se corresponde con la fórmula de interpolación de Lagrange deseada, donde $\phi(x)$ es el la señal resultante, $\varphi(x)$ la original y $f_n = \varphi(x_n)$ (en contraposición a las diferencias divididas de Newton). Computando la señal de la distancia centroide-contorno como derivadas discretas del array dinámico original a interpolar (diferencias entre puntos adyacentes) y aplicando esta expresión se puede obtener una señal discreta que la suaviza, de una forma más acorde a los cambios de gradiente de la señal.

La ecuación implementada en el método `Procesador::lagrangeInterpolation` es la siguiente:

$$v_2 = \sum_{i=0}^{N-1} v_{1_{y_i}} \cdot P_i(v_{1_x}), = \sum_{i=0}^{N-1} v_{1_{y_i}} \cdot \prod_{j=0,\, j\neq i}^{N-1} \frac{v_{1_x} - v_{1_{x_j}}}{v_{1_{x_i}} - v_{1_{x_j}}}, \quad (88)$$

donde $P_i(v_{1_x})$ la base polinómica del polinomio interpolador (v_2) en la forma de Lagrange en la iteración "i" y v_1 es el vector interpolado (en formato `std::vector<double>`, como el interpolante). Primeramente se calcula el número de puntos N a partir del tamaño del vector de datos de entrada, luego se itera sobre cada punto "i" e inicializa el resultado a cero del polinomio, se itera sobre cada punto "j" excluyendo el punto "i" actual y se actualiza el valor de los miembros de la aproximación y el vector resultante en cada repetición del bucle. Este tipo de interpolación es muy útil en caso de que el muestreo levógiro de los puntos para computar la señal 1D u otra señal quiera hacerse en pasos más grandes, consiguiendo aproximar la forma por polinomios de orden arbitrario para volver a muestrearlos después a voluntad; se hacen pruebas con esta idea y, aunque pudiera ser aprovechable, se decide mantener el contorno muestreado de grado en grado en vez de utilizar la función interpolante pues no se encuentra una mejora significativa en eficiencia. En cambio, para hacer las veces de esta técnica cuando se vio necesario, en vez de utilizar el método que la implementa, se empleó la función de OpenCV `resize` para señales 1D, que realiza remuestreos más sencillos y de manera más directa.

Posteriormente y para suavizar, perfeccionar someramente las señales y evitar valores atípicos indeseables se consideró realizar un filtrado mediante un filtro de Hampel y otro de mediana mediante dos clases de las librerías de la empresa, `dsi::HampelFilter` y `dsi::MedianFilter1D`. El último es un filtro no lineal, pues suaviza una señal al reemplazar cada punto de datos por la mediana de los valores en una vecindad específica alrededor de ese punto, y el de Hampel es más flexible que el otro, pues para cada punto se calcula la mediana de un conjunto de puntos cercanos y, comparando el valor actual con

esta mediana, si la diferencia es mayor que un umbral predefinido (llamado "ventana de Hampel"), se considera que el punto es un valor atípico y se ajusta, adaptándose pues a fluctuaciones locales.

Como penúltimo paso antes de completar la generación de las señales 1D que servirán como base de análisis del resto de métricas, se submuestrean las señales al rango [0, 359] (en grados sexagesimales) para normalizar sus tamaños y poder compararlas debidamente. Para ello se calcula el espaciado entre los puntos originales para ajustarlos al número de elementos deseado, que se decidió fuera 360 º aunque el aumento de la precisión no debería resultar en un costo computacional significativo, ni ciertamente significante para los resultados.

Finalmente, para lograr invarianza a la escala, se normalizan las señales para que tengan una media de 0 (eliminar la componente de continua) y desviación estándar de $\sqrt{2}$; este ajuste de ganancia en las señales hace que los posibles errores introducidos en estas estén balanceados.

El método `Procesador::makeZeroMeanAndScale` calcula la media de las señales, la resta de cada uno de sus elementos y reescala proporcionalmente sus valores para que tengan la desviación estándar deseada. El hecho de que la localización de la cámara esté controlada a voluntad hace que el tamaño de las piezas (su escala) se pueda suponer prácticamente igual en todo momento y, por tanto, la normalización de escala de estas señales pueda no ser tan necesaria (pues el rango de las señales estaría comprendido entre [aprox.] los mismos límites). Pero para evitar diferentes fuentes de ruido, tanto desde la binarización como de los desajustes mecánicos, se decide mantener esta mejora.

Como medida adicional, se puede lograr cierta invarianza a la traslación entre las señales 1D corrigiendo el offset entre ellas. En los métodos `Procesador::normalizeOffset` y `Procesador::phaseAlign` se implementaron dos enfoques distintos para dar solución a este problema, con el objetivo principal de alinear ambas señales de manera que compartan una referencia temporal (de abscisas) común.

El primer método calcula la función de correlación cruzada entre las dos señales, que mide cuán similares son muestra a muestra, para diferentes desplazamientos o posiciones. Luego, identifica el desplazamiento que maximiza esta correlación cruzada, lo que indica el ajuste óptimo entre las señales. Finalmente, se procede a alinear las señales utilizando el desplazamiento hallado. Si el desplazamiento es positivo, la función desplaza la segunda señal hacia la izquierda; si es negativo, desplaza la primera.

El segundo método (aunque similar) busca corregir cualquier desfase que puedan tener las señales. En primer lugar, se selecciona una región de interés (ROI) en ambas señales. Luego, se interpola los datos en la ROI, lo que implica muestrear los puntos de datos a una cuadrícula de mayor resolución. A continuación, se resta la media de ambas señales en la ROI para eliminar componentes de continua y centrar las señales en cero. Luego, se calcula la función de correlación cruzada entre las señales en la ROI. Este cálculo revela el desplazamiento temporal que maximiza la correlación entre las señales, lo

que se corresponde con la alineación óptima. Si el desplazamiento resulta ser cero, se recalcula la correlación cruzada con un desplazamiento de 180 grados para garantizar que se capturen los cambios de fase. El método devuelve el desplazamiento de fase corregido como un valor decimal.

Aunque no se emplea en las pruebas finales pero sí se implementa en el método `Procesdor::phaseAlign`, una forma alternativa de ecualizar el desplazamiento entre las señales es alinearlas mediante la derivación de su desplazamiento relativo utilizando diferentes métodos de correlación cruzada [50].

Una manera de hacerlo es desplazar una señal mediante el remuestreo mediante interpolación lineal, ignorando los bordes de la señal (ya que el algoritmo se centra en alinear dos señales dentro de una región de interés), luego comparar la señal desplazada con la otra (que sirve como "plantilla") y se ejecuta una rutina de minimización del coeficiente χ^2 (chi-cuadrado[23], utilizado como medida estadística para evaluar la discrepancia entre un conjunto observado de datos y un conjunto de datos esperados) para estimar el desplazamiento óptimo entre ambas. Cada señal (plantilla y desplazada) se normaliza antes del desplazamiento de tal forma que sus medias sean igual a 1 para que el algoritmo se centre principalmente en comparar la forma de los datos, no su amplitud. Una advertencia sobre esta técnica es que es propensa a encontrar mínimos locales a menos que las restricciones previas estén fuertemente limitadas, ya que utiliza un método de optimización basado en gradientes.

La segunda rutina de alineación utiliza una correlación de fase en el dominio frecuencial, empleando la DFT como se explicó anteriormente pero en un caso unidimensional. Como se comenta en [50], este algoritmo se comporta de manera similar a la minimización de χ^2 pero típicamente con una precisión más baja. También se advierte que si se correlacionan datos a la resolución original solo puede lograr una precisión entera, por lo que para lograr una precisión subpíxel o decimal se debe (por ejemplo) aumentar la escala de los datos antes de la correlación cruzada, con lo que, aumentando la escala N veces, se puede lograr una precisión de N^{-1} en cuanto al valor de desplazamiento, lo que significa que si N = 100, el valor de desplazamiento más pequeño que puede obtener es 0,01.

Al realizar todos estos procedimientos, se garantiza que las señales sean coherentes en términos de tiempo y frecuencia, facilitando su análisis y comparación mediante las ulteriores métricas.

2.3.3.15.- Error cuadrático medio

El error cuadrático medio (ECM o MSE en inglés) es medida relativamente simple que cuantifica la diferencia promedio entre los valores de dos señales. Se decide

[23] $\chi^2 = \sum (O_i - E_i)^2 / E_i$, donde O_i son las observaciones reales (una imagen o su histograma), E_i son las observaciones esperadas (la imagen de referencia o su histograma) y el sumatorio se realiza sobre todas las categorías del conjunto de datos (todos los píxeles o valores de cada histograma).

implementar para estudiar si proporciona resultados factibles aun siendo un criterio de similitud muy sensible al desplazamiento relativo de las señales y al ruido.

La implementación la recoge el método `Procesador::calcularSimilitud`, en el que lleva a cabo una normalización de la métrica mediante la división de la diferencia acumulada por la longitud de las señales. Las señales también son escaladas al rango [0, 1] de acuerdo con la siguiente expresión:

$$\bar{s}(x) = \frac{s(x) - s_{mín}}{s_{máx} - s_{mín}}, \tag{89}$$

donde "s" representa la señal original y el miembro izquierdo de la igualdad la señal normalizada, $s_{máx}$ y $s_{mín}$ son los valores máximo y mínimo de la señal, respectivamente, y su diferencia es el rango de la señal original en el dominio definido por [0, 359] (en este caso).

El valor devuelto es un real que, si mayor (y si es mayor que un umbral experimentalmente definido), resulta en un indicio de que las señales son "más bien" parecidas; la ecuación codificada es la siguiente:

$$similitud\ SSD = 1 - \frac{1}{N}\sum_{i=0}^{N-1} |\bar{s}_1(x) - \bar{s}_2(x)| \tag{90}$$

La salida de esta métrica es el booleano `SSDSimilarity`, que indica si la pieza es válida a ojos de este sub-bloque. Cuanto menor es el valor (entre 0 y 1) devuelto por esta métrica, más parecidos son los contornos de las piezas.

2.3.3.16.- Correlación mediante producto interno real

En este apartado se realiza una implementa de la correlación cruzada de entre dos piezas en el dominio espacial. El producto escalar entre dos vectores da una idea de la cantidad de información explicada por uno que viene contenida en el otro. De manera análoga al proceso de normalización del offset entre las señales antes explicado, el método `Procesador::dotCorrelation` compara la similitud entre dos señales y devuelve un resultado entre 0 y 1, donde un valor de 1 indica una correlación perfecta, un valor de 0 indica ninguna correlación (valores negativos indicarían una correlación inversa).

Primeramente se calcula la suma acumulada de los productos de las muestras correspondientes en cada señal (del mismo tamaño) (producto punto o escalar de ambas señales):

$$\overrightarrow{s_1} \cdot \overrightarrow{s_2} = \sum_{i=0}^{N-1} s_{1i} \cdot s_{2i} = \|s_1\| \cdot \|s_2\| \cdot cos\theta, \tag{91}$$

donde s_{1i} y s_{2i} son los elementos i-ésimos de la primera y segunda señal, respectivamente. Se ve que la similitud se entiende como el coseno del ángulo entre los vectores normalizados. También se computa la longitud euclídea (magnitud) (l_1 y l_2) de las señales

como factor de escalado para normalizar a posteriori el resultado del producto escalar, manteniendo las dimensiones, mediante la raíz cuadrada de la suma de todos los elementos al cuadrado.

Evitando la división por cero se computa este intuitivo coeficiente de similitud mediante la expresión:

$$dot\ similarity = \frac{\overrightarrow{s_1} \cdot \overrightarrow{s_2}}{\sqrt{l_1.l_2}} = \left(\sum_{i=0}^{N-1} {s^2}_{1i}\right)^{-1/2} \cdot \left(\sum_{i=0}^{N-1} {s^2}_{2i}\right)^{-1/2} \cdot \sum_{i=0}^{N-1} s_{1i} \cdot s_{2i} \quad (92)$$

Si bien este método no sería indicado para señales que no han sido normalizadas ni acondicionadas ante grandes cambios en la traslación, rotación y escala de las piezas, bajo las asunciones consideradas y medidas llevadas a cabo para robustecer el algoritmo (y la normalización a distintos niveles de las señales) este sub-bloque proporciona respuestas muy razonables.

Un enfoque semejante se mantiene en el método `Procesador::cosineSimilarity`, que devuelve el valor del coseno del ángulo entre los vectores calculado mediante el producto escalar, sin normalizar, como cuantificación de la similitud direccional entre ambos. Un valor cercano a 1 significaría que son similares en dirección; cercano 0 significaría casi ortogonales.

2.3.3.17.- Contexto de formas

En este sub-bloque se hace uso del descriptor y algoritmo de búsqueda de coincidencias de formas *Shape Context*, propuesto en [51], para buscar la semejanza entre los contornos de las piezas segmentadas basándose en aspectos como la distancia entre los puntos característicos, la apariencia local de la forma y la energía de flexión necesaria para "transformar" morfológicamente uno de ellos para que capture la posición relativa entre los puntos del otro.

El establecimiento de correspondencias de este método se basa en calcular la posición y orientación entre pares de puntos relativos de las formas (muestreados de sus contornos internos o externos), agrupar la información en un histograma bidimensional y aproximar la distancia entre las formas (que mide cuánto se parecen entre sí) mediante la distancia entre sus respectivos descriptores. El descriptor captura la distribución de los puntos con respecto a un punto de referencia de la forma (por ejemplo, el centroide) y la similitud entre estas se calcula como la suma de los errores de coincidencia entre puntos correspondientes, junto con el término de distancia que indica la magnitud de transformación de la alineación. La búsqueda de una solución óptima para el problema de la correspondencia se trata de solventar maximizando similitudes y garantizando la unicidad de las parejas.

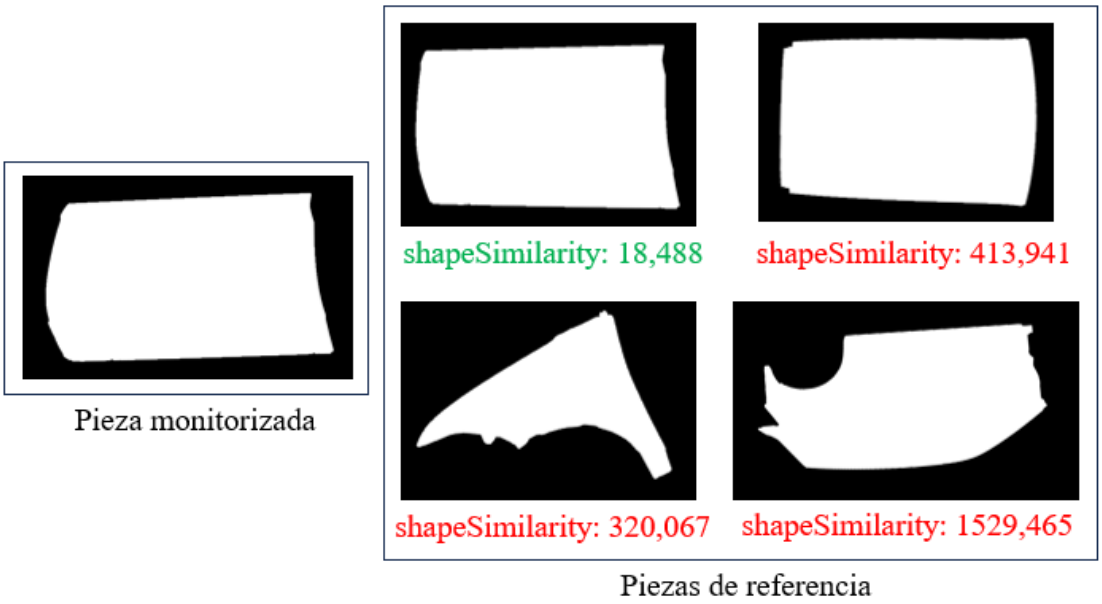


Fig. 77. Cuatro valores de la métrica que emplea el contexto de formas para medir la similitud con piezas válidas.

En el algoritmo implementado primeramente se configuran los parámetros del puntero `cv::Ptr<cv::ShapeContextDistanceExtractor>`, que integra el extractor de distancia de correspondencia por contexto de formas. Entre estos parámetros está el número de bins angulares y radiales para hacer más precisa la correspondencia, el radio interno y externo para tener en cuenta detalles más finos o groseros de los contornos, el número de iteraciones a llevar a cabo, el extractor de costos de histograma (se emplea el valor chi-cuadrado) y el empleo de un transformador de forma utilizando "splines de placa delgada". Se calcula la distancia de forma entre cada contorno con el método `cv::ShapeContextDistanceExtractor::computeDistance` y se filtra con un umbral empírico de 50 para verificar la validez de esta comparación mediante el booleano `ShapeSimilarity` (cualquier distancia mayor que este umbral indica que las formas de los contornos no son suficientemente similares para considerar que las piezas son las mismas, a ojos de este sub-bloque); un menor valor indica una mayor similitud entre los contornos de las piezas.

2.3.3.18.- Distancia de Hausdorff

La distancia de Hausdorff [52] es una medida relativamente simple que cuantifica la disparidad entre dos conjuntos y se puede extrapolar al procesamiento de imagen para comparar dos formas a través de los puntos de sus contornos. Matemáticamente, esta distancia define como sigue:

$$d_H(F,G) = \max\left(sup_{a\epsilon F} inf_{b\epsilon G} d(a,b), sup_{b\epsilon G} inf_{a\epsilon F} d(a,b)\right), \quad (93)$$

donde "sup" denota el supremo de una imagen (mayor límite superior), "inf" el ínfimo (menor límite superior) y $d(a,b)$ es la distancia entre los puntos "a" y "b". El valor $inf_{b\epsilon G} d(a,b)$ representa la distancia más corta entre un punto "a" de una imagen hasta

cualquier punto de la otra, y $sup_{a\epsilon F} inf_{b\epsilon G} d(a,b)$ toma la cota superior más baja de todas las distancias mínimas desde los puntos en F hasta G; similarmente, $sup_{b\epsilon G} inf_{a\epsilon F} d(a,b)$ es lo análogo al primer término pero considerando la distancia más corta desde un punto en G hasta F. El resultado no es el mismo en una dirección u otra porque esta métrica es asimétrica, pues cada uno enfoca la proximidad desde una perspectiva de forma; esto la dota de una capacidad propia para capturar características específicas de la relación entre los dos conjuntos sin requerir una correspondencia biunívoca entre los puntos de ambos.

Así entonces, la distancia de Hausdorff es la mayor de las distancias más cortas entre los puntos de un conjunto y los del contrario, en ambas direcciones. Como distancia se hace uso de la norma L2 (la magnitud del vector desde su origen); la clase mencionada da la opción de emplear alternativamente la norma L1 (la suma de los valores absolutos de todos los elementos del vector).

Además, como la comparación es punto a punto, se hace notar la importancia de dedicar algo de tiempo a normalizar la escala, rotación de las piezas y (particularmente) el desplazamiento entre las señales con métodos como la alineación mediante correlación de fase; de lo contrario el método sería de mucho menor provecho.

Se hace uso de la clase `cv::ShapeContextDistanceExtractor>`, que proporciona un acceso directo a este algoritmo. Se calcula la distancia de forma entre cada contorno con el método `cv:: HausdorffDistanceExtractor::computeDistance` y se filtra con un umbral empírico para verificar la validez de esta comparación mediante el booleano `hausdorffSimilarity` (cualquier distancia mayor que este umbral indica que las formas de los contornos no son suficientemente similares para considerar que las piezas son las mismas, a ojos de este sub-bloque). Cuanto mayor es el valor devuelto por esta métrica, más distintos son los contornos de las piezas.

2.3.3.19.- Similitud de Pearson

El coeficiente de correlación de Pearson es una medida de la dependencia lineal entre dos variables aleatorias cuantitativas. El coeficiente (aplicado a una muestra, en contraposición a una población) se calcula mediante la siguiente expresión (véase la equivalencia con la ecuación presentada anteriormente para el producto interno real):

$$r = \frac{\sum_{i=0}^{n-1}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=0}^{n-1}(x_i - \bar{x})^2}\sqrt{\sum_{i=0}^{n-1}(y_i - \bar{y})^2}}, \tag{94}$$

donde “n” es el tamaño de la muestra (número de puntos), x_i y y_i son los vectores de puntos de los contornos y $\bar{x}$ e $\bar{y}$ son sus medias muestrales. Un valor de 1 indica una correlación perfecta positiva, su opuesto indica una correlación perfecta negativa y un valor de 0 indica ausencia de correlación lineal. El numerador representa la covarianza entre las señales, mientas que el denominador representa el producto entre su varianzas.

En el método `Procesador::pearsonCorrelation` se implementa un algoritmo para extraer el coeficiente de correlación de Pearson entre las señales de los contornos, que es apto para mensurar si existe una relación lineal entre los contornos de las piezas, que si es suficientemente alto indicaría que, salvo por un factor de escala, las formas son semejantes.

2.3.3.20.- Correlación cruzada normalizada

De manera análoga al apartado en el que se comentó el template matching, en esta métrica se emplea la función `cv::matchTemplate` (indicando el flag `TM_CCORR_NORMED`) para calcular la correlación cruzada normaliza de las señales unidimensionales previamente alineadas y reescaladas debidamente (de no ser por dicho tratamiento, esta técnica se vería demasiado influenciada para ser fidedignamente interpretable, aun en su versión normalizada [que afronta mayoritariamente el cambio de escala], a los cambios por traslación y rotación de las piezas).

El proceso implica deslizar la plantilla a lo largo de la señal de entrada y calcular la similitud en cada posición; como ambas tienen el mismo tamaño el desplazamiento es nulo, pues encajan muestra a muestra. La salida de esta operación es un mapa de correlación que indica las ubicaciones donde la señal de entrada se asemeja más al patrón, lo que se podría usar también para ajustar el offset de señales que son suficientemente similares, exactamente a como se hizo en `Procesador::normalizeOffset` y `Procesador::phaseAlign`. La normalización es crucial para que la métrica no sea demasiado sensible a la amplitud o intensidad de las señales, permitiendo así comparaciones más robustas. La normalización es particularmente útil para hacer frente a cambios de magnitud pero, aunque aborda en mayor medida estos cambios, aún puede ser sensible a traslaciones y rotaciones. La métrica resultante proporciona, no obstante, información valiosa sobre la concordancia entre las señales, además de que es más invariante a estos cambios geométricos que la comentada anteriormente. Cuanto mayor es el valor devuelto por esta métrica, más parecidos son los contornos de las piezas.

2.3.3.21.- Integral definida

Una forma muy directa y efectiva de comparar los contornos de las piezas, si las señales están debidamente normalizadas, es evaluar el área que encierran bajo la curva (discreta) que los define. En la siguiente figura se muestra la sencilla idea tras esto, con la que se pretende filtrar de manera prematura la similitud de las piezas basándose en la relación de distancia entre cada punto contorno al centroide.

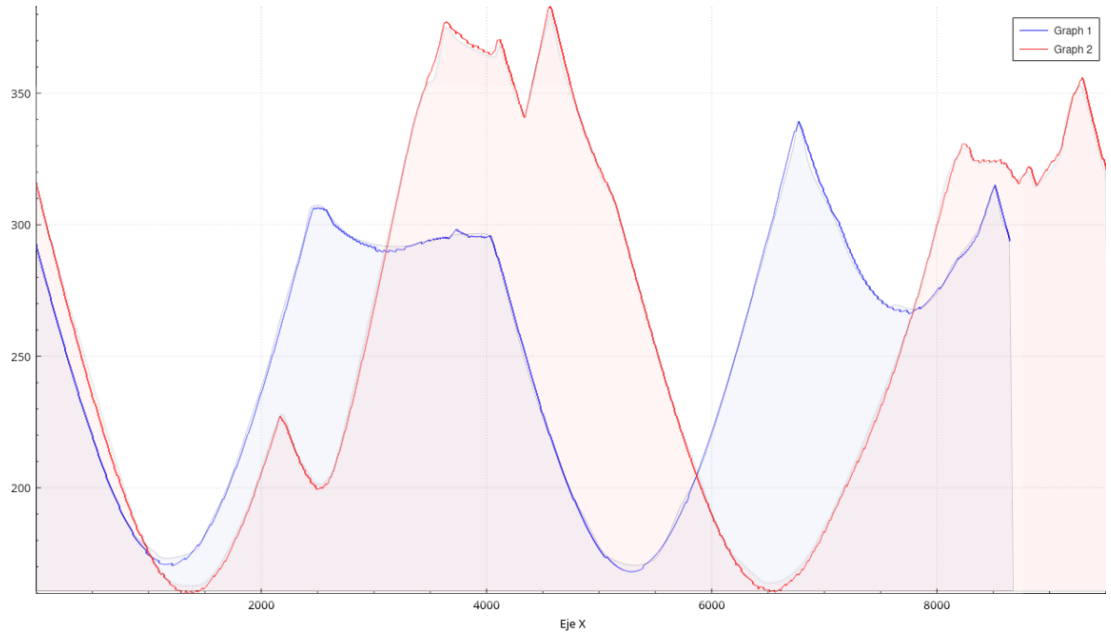


Fig. 78. Perfiles de dos piezas registrados como señales unidimensionales para su análisis, previa normalización.

En el método `Procesador::normalizedAreaDifference` se recorre los elementos de los vectores que discretizan las señales, se suma el valor de sus ordenadas para cada punto de abscisa para calcular sus áreas, se normalizan los valores resultantes por el número de elementos para que influya el muestreo de las señales y se computa su diferencia absoluta. Controlando la escala, amplitud y el offset de las señales de la manera en que se hizo, este método es muy útil y eficiente para medir cuán diferentes son en términos de su contenido de energía. Un valor cercano a 0 equivale a dos piezas que son similares en cuanto a forma.

2.3.3.22.- Error cuadrático medio integral

Cabe darse cuenta de que la aproximación de la integral, así como del uso del valor medio para interpolar una señal, no son ciertamente robustos ante valores atípicos. Por ejemplo, si se tiene una función $\varphi(x)$ definida en el intervalo $[a, b]$ (entre 0 y 360 grados) y se quiere evaluar el error cometido al sustituirla por otra función interpolante $\phi(x)$, se puede tomar como medida del error la desviación máxima entre las funciones (definida como el máximo de la diferencia absoluta entre las dos señales para cada punto), pero resulta mucho más conveniente tomar como medida del error la desviación media cuadrática δ^2 [53, pág. 397], definida como:

$$\delta^2 = \frac{1}{(b-a)} \int_a^b [\,\phi(x) - \varphi(x)]^2 \, dx \tag{95}$$

Aunque la desviación máxima entre una señal poco similar sin valores atípicos y un contorno de referencia pueda ser menor que la desviación máxima entre el último y una señal muy similar con un valor atípico (una diferencia considerable de las señales en un

intervalo muy pequeño), la desviación media cuadrática considera (más bien) la forma de la curva y, por tanto, es más apropiada para comparar los contornos de las piezas.

En la siguiente figura se puede ver un ejemplo de una situación (con dos piezas del mismo tipo rotadas un máximo suficiente para que ambas estén en un rango admisible de validez) que se beneficia de este análisis, pues exceptuando el tramo inicial de las señales, que se puede considerar como una región atípica debida a una mala segmentación a causa de una iluminación desigual, la forma de las señales es muy similar, salvo por un factor de escala, por lo que este sub-bloque consideraría, presumiblemente, que la pieza actual es válida.

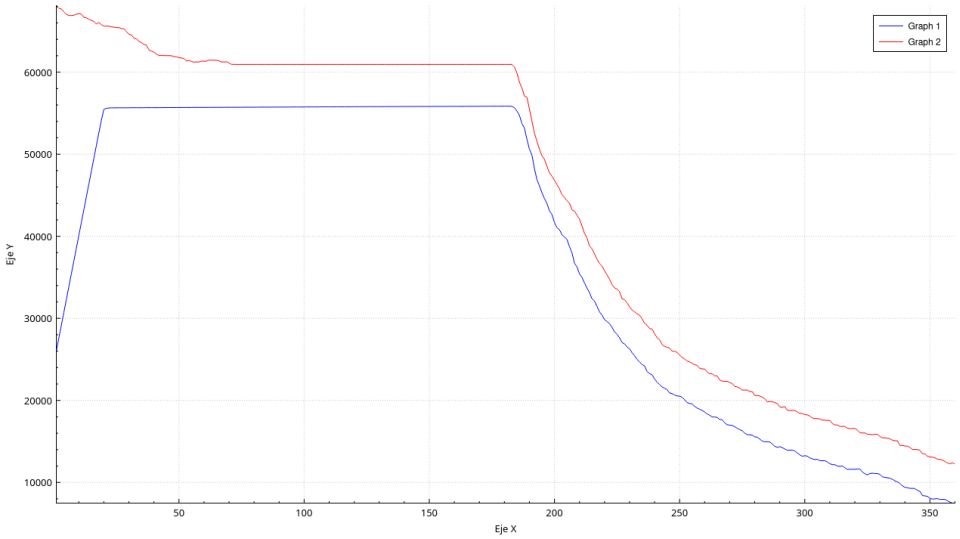


Fig. 79. El perfil radial extraído de dos piezas del mismo tipo con una región atípica inicial.

En el método `Procesador::MSDDifference` (la "D" proviene de *displacement*) se implementa esta técnica, que toma como entrada las señales normalizadas tanto en escala como en traslación. La salida de esta métrica es el booleano `MSDSimilarity`, que indica si la pieza es válida a ojos de este sub-bloque. Como en el caso anterior, un valor cercano a 0 equivale a dos piezas que son similares en cuanto a forma.

Las salidas de todos aquellos sub-bloques anteriormente presentados que se suplen de la información del contorno se agrupan en el booleano `ContoursSimilarity`, que indica si la pieza es válida a ojos de dichos sub-bloques.

2.3.3.23.- ECC

El coeficiente de correlación mejorado (ECC) es una medida de similitud entre imágenes que calcula la transformación geométrica óptima (matriz de mapeo) con respecto a un criterio que busca maximizar la correlación mejorada (una modificación del coeficiente de correlación tradicional) entre una imagen de entrada y una de referencia (o plantilla) después de aplicar la transformación especificada bajo cierto criterio [54]. El algoritmo ECC intenta encontrar los parámetros óptimos de la transformación (desplazamiento, rotación, escala, perspectiva...) para maximizar la correlación mejorada,

utilizando un enfoque iterativo lineal alternativo (dos, de hecho, uno basado en el enfoque aditivo directo de Lucas-Kanade y otro en el método composicional inverso simultáneo [SIC]) para resolver el problema de optimización no lineal (función de los parámetros de la deformación) que supone el método. La optimización continúa hasta que se alcanza un criterio de convergencia definido por unos parámetros preestablecidos; la salida del algoritmo es el coeficiente de correlación mejorada, que representa la calidad de la alineación entre las imágenes tras transformar la entrada con la matriz de transformación resultante. ECC es invariante a las distorsiones fotométricas de contraste y brillo (incluso a desalineaciones en los canales de color).

Este método ha dado resultados muy fructíferos en este proyecto, pues como las piezas no se ven (en funcionamiento normal) sometidas a variaciones bruscas en su orientación entre frames, este algoritmo es capaz, al unísono, de proporcionar una estimación de la pose relativa de la pieza y de su similitud con la pieza de referencia. De cierta forma ECC alberga una implementación reducida o comprimida pero sin enlazar de lo que sería el sistema de este proyecto formado por el clasificador morfológico y el característico, y a la vez forma parte del primero.

OpenCV proporciona a través de `cv::findTransformECC` varios modelos de movimiento para la alineación de imágenes, incluyendo la traslación, transformación euclídea, afín y homográfica. Para el proyecto se ha recogido parte del código fuente de la librería y se han reimplementado varias de sus partes para agregar componentes nuevos y hacerlo más eficiente computacionalmente, pues es un algoritmo que puede llegar a ser muy costoso en tiempo de cálculo si las imágenes crecen en tamaño. Los métodos implementados, que se explicarán a continuación junto con el algoritmo, son (dentro del espacio de nombres `Procesador`): `alineaImagenesECC`, `findTransformECC`, `computeECC`, `update_warping_matrix_ECC`, `project_onto_jacobian_ECC`, `image_jacobian_translation_ECC`, `image_jacobian_affine_ECC`, `image_jacobian_euclidean_ECC` y `image_jacobian_homo_ECC`.

En el método `Procesador::computeECC` se computa el coeficiente de correlación mejorada entre una imagen de entrada y una plantilla según el artículo original que lo describe. Se comienza verificando la validez de las imágenes, asegurándose de que ni la plantilla ni la imagen de entrada no estén vacías y ambas tengan el mismo tipo de datos. Luego, se calcula la media y la desviación estándar de la plantilla, se realiza un ajuste especial si el tipo de dato de las imágenes es entero sin signo para evitar problemas de cálculo (se convierte a enteros con signo, de mayor resolución, pues al normalizar puede pérdida de información si existen valores menores que la media). Después, se resta la media de la plantilla y se normalizan ambas imágenes, se calculan sus normas y devuelve el resultado del coeficiente, que es el producto punto (escalar) normalizado entre las imágenes preprocesadas (o sea, su correlación cruzada).

En el método `Procesador::findTransformECC` se estima la transformación geométrica que alinea la imagen de entrada con la plantilla. Como antes, se comienza

realizando verificaciones de validez para las imágenes y la matriz de transformación. Luego, se realiza un preprocesamiento que incluye la posibilidad de convertir las imágenes a una menor resolución mediante su submuestreo y la aplicación opcional de un filtro gaussiano a fin de aligerar la carga computacional y mejorar la velocidad del algoritmo.

El submuestreo, también conocido como *downsample* o decimación, es un proceso en el cual se reduce la cantidad de muestras en una imagen (u otro tipo de señal), lo que implica reducir su resolución al disminuir la cantidad de píxeles en su conjunto seleccionando un conjunto de píxeles de la imagen original y descartando el resto, persiguiendo conseguir una versión que mantenga información importante de la original pero de menor tamaño para que su posterior procesamiento sea menos costoso computacionalmente. Una forma de lidiar en cierta medida con la pérdida de información de escala al reducir el tamaño de una imagen y el efecto de aliasing[24] que pudiera aparecer por culpa de que remanezcan las altas frecuencias es suavizar previamente la imagen; así, se realiza submuestreo tras la convolución de la imagen con un filtro gaussiano (ver la siguiente expresión), tal como se hace para la construcción de una pirámide de gaussianas:

$$I^{(i+1)}(x,y) = (I^{(i)}(x,y) \circledast Gauss_{\sigma^2}(x,y)) \downarrow \sqrt[n]{2}, \tag{96}$$

donde i representa cada imagen sucesiva en la pirámide ($I^{(0)}$ es la original) y $\sqrt[n]{2} = 2^{n^{-1}}$ es el factor de escalado del submuestreo y se deja ver que hay n imágenes por octava25, cada cual una cantidad $2^{k/n}$ más pequeña que la imagen original ($k = \{1..n\}$). En el método `Procesador::alineaImagenesECC` se hace uso de la función `cv::pyrDown` para submuestrear un total de dos octavas las imágenes en caso de que su tamaño sea menor de (p. ej.) 100 x 100 píxeles, reduciendo la resolución de la imagen descartando cada fila y columna par.

[24] El aliasing es un efecto visual adverso producido, en el contexto de las imágenes, por una resolución insuficiente en una imagen cuya escena contiene frecuencias demasiado altas, debido a que la frecuencia de muestreo es demasiado baja para capturarlas de manera adecuada, lo que provoca efectos y artefactos como patrones *moiré*, de interferencia o bordes dentados (dientes de sierra).

[25] Se denomina octava a cada sucesiva reducción de una imagen, en resolución, por un factor de dos.

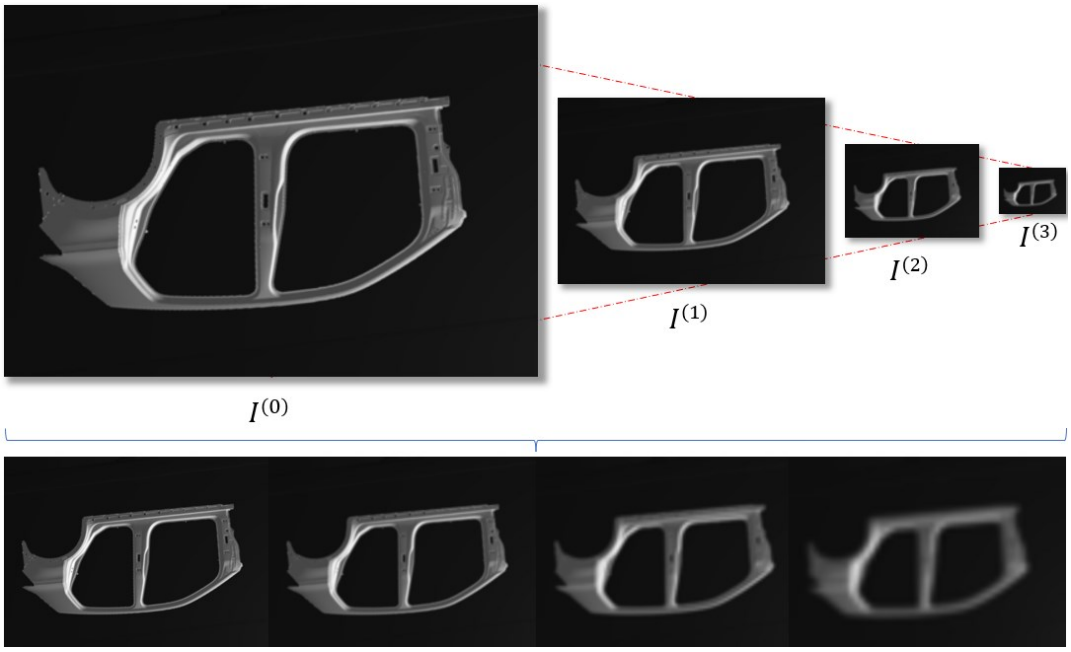


Fig. 80. Ejemplo del submuestreo y suavizado de imágenes en una pirámide de gaussianas de cuatro niveles.

Se ha comprobado que para un tamaño de imagen típico de 1920 x 1080 píxeles el método devuelve resultados aceptablemente similares si las imágenes entran en su tamaño completo como si se filtran con un kernel gaussiano de hasta 11 x 11 y se submuestrean hasta 3 octavas. Posteriormente se itera para encontrar la matriz de transformación que maximiza la correlación mejorada entre las imágenes; durante cada iteración se calcula el jacobiano y hessiano de la imagen alineada con respecto a los parámetros de la transformación y se utilizan estos cálculos para actualizar dicha matriz. Este proceso se repite hasta que se alcanza un número máximo de iteraciones o se logra la convergencia deseada, que son dos criterios establecidos previa y externamente y se emplean (típicamente) valores de 10^3 y 10^{-6}, respectivamente. Adicionalmente, se puede especificar un parámetro de "parada" adicional, que es el valor máximo de correlación a partir del cual se considera que las imágenes son suficientemente similares (en la iteración dada) (0,9, p. ej.).

El jacobiano (o la matriz jacobiana) es una matriz que representa las derivadas parciales de un sistema de ecuaciones con respecto a sus variables independientes. En este contexto, se utiliza para describir cómo cambian los píxeles de una imagen (en valor y posición) cuando se le aplica una cierta transformación geométrica o de nivel de gris y representa la relación entre los parámetros de una transformación actual y la diferencia entre la imagen [pieza] proyectada bajo esa transformación y la plantilla (imagen objetivo o [pieza] de referencia). Para cierta transformación $\boldsymbol{T} = (T_1(x,y), T_2(x,y))$ a los píxeles $\boldsymbol{p}(x,y)$ de una imagen, el jacobiano de esta sería:

$$J \equiv \nabla \boldsymbol{T} = \frac{\partial \boldsymbol{T}}{\partial \boldsymbol{p}} = \begin{bmatrix} \frac{\partial T_1}{\partial x} & \frac{\partial T_1}{\partial y} \\ \frac{\partial T_2}{\partial x} & \frac{\partial T_2}{\partial y} \end{bmatrix}, \tag{97}$$

donde T_1 y T_2 son las transformaciones que se aplican a las coordenadas de los píxeles. En el caso de una imagen, esto se extiende a un jacobiano por bloques de píxeles y, en la implementación realizada, en el caso de una homografía, se manejan bloques de 8 píxeles, pues es el caso más general, que más variaciones y flexibilidad en la geometría de la imagen permite e incluye un desplazamiento de los píxeles en las direcciones "x" e "y", una rotación que es función de un sesgo o cizallado y una escala para cada dimensión y dos deformaciones perspectivas.

Por otra parte, el hessiano (o la matriz hessiana) es una matriz que representa las derivadas parciales segundas de un sistema de ecuaciones con respecto a sus variables independientes; en este contexto se utiliza para describir cómo cambian a este respecto las derivadas del jacobiano, además ayuda a ajustar la tasa de convergencia el algoritmo; en la práctica se puede aproximar por $J^T J$, que con funciones aproximadamente convexas es una matriz normalmente definida positiva y ayudar a mejorar la estabilidad de la optimización:

$$H \equiv \nabla^2 \boldsymbol{T} = \frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{p}^2} = \begin{bmatrix} \frac{\partial^2 T_1}{\partial x^2} & \frac{\partial^2 T_1}{\partial x \partial y} \\ \frac{\partial^2 T_2}{\partial x \partial y} & \frac{\partial^2 T_2}{\partial y^2} \end{bmatrix} \tag{98}$$

La función `Procesador::alineaImagenesECC` utiliza el método `Procesador::findTransformECC` para alinear dos imágenes, pues es la que sirve como interfaz para acceder a este algoritmo desde el exterior, de una manera lo más transparente posible. En este se realiza convierten previamente las imágenes a escala de grises y tipo de datos de punto flotante. Si las imágenes son grandes, puede realizar un submuestreo para mejorar la velocidad. Por referencia se obtiene la matriz de transformación y el coeficiente de correlación mejorado resultantes. También proporciona la imagen de entrada alineada con la plantilla como tercera salida, utilizando la misma matriz de transformación y acepta como argumento el tipo de transformación que se desea utilizar mediante la enumeración `tfm::MotionType`.

En los métodos del tipo `Procesador::image_jacobian` se calcula el jacobiano del tipo de transformación elegida para computar el ECC. Para ello se proyectan las coordenadas de las imágenes con la matriz de transformación actual y se almacenan tantos bloques del jacobiano como la transformación requiera (según sus grados de libertad o variables independientes o el tamaño de la transformación dada su representación matricial o vectorial), dividiendo cada bloque de gradientes por un denominador común para todos los puntos.

En el método `Procesador::project_onto_jacobian_ECC` se proyectan las imágenes sobre el jacobiano durante el proceso iterativo del algoritmo. La proyección de una imagen sobre el jacobiano se hace (generalmente) para calcular la corrección de error con respecto a la actualización anterior en la transformación. En este contexto, se proyectan las diferencias entre las imágenes de origen y destino sobre el jacobiano y el resultado se utiliza para ajustar la transformación. Más concretamente, se define una función de coste o error (suma de la diferencia cuadrática, p. ej.) a minimizar que mide la discrepancia entre la proyección estimada y los puntos de destino reales y se calcula el gradiente (que indica la dirección en la que la función de error crece más rápidamente) de la función de coste con respecto a los parámetros de la transformación, que es el que se utiliza en cada iteración del algoritmo para el problema de optimización no lineal.

En el método `Procesador::update_warping_matrix_ECC` se actualiza la matriz de transformación basándose en el tipo de movimiento especificado y el "paso" que se añade en cada iteración a la suma o resta de orientación y traslación adicionales añadidas. La actualización de los parámetros de la transformación se estima utilizando el método de Gauss-Newton; en cada iteración se calcula una función residual r (que representa la diferencia entre las observaciones reales y las predicciones del modelo para el conjunto actual de parámetros), el jacobiano (la matriz de derivadas parciales de las funciones residuales con respecto a los parámetros) y la dirección de actualización $\delta\theta$ de los parámetros θ resolviendo el sistema lineal simplificado $J\delta\theta = -r(\theta) \leftrightarrow \delta\theta = -J^{+}r(\theta)$, para adicionarles una cantidad $\beta \cdot \delta\theta$, con β calculado según la regla de actualización del método iterativo (paso arbitrario, unitario, óptimo usando métodos como *line search*…), $\theta^{k+1} = \theta^{k} + J^{+}r$ (donde J^{+} es la pseudoinversa de la matriz jacobiana), en dirección del mínimo gradiente a una celeridad dependiente de la curvatura local de la función hasta cumplir alguno de los criterios de convergencia establecidos en búsqueda del mínimo local de la función de coste no lineal. En la iteración final del algoritmo, finalizado a causa del criterio de parada de convergencia que sea, la transformación final ajustada proyectaría los puntos de entrada ajustándolos de manera precisa a las ubicaciones más correlativas en la plantilla.

Finalmente, las salidas de todos aquellos sub-bloques anteriores se agrupan en la salida booleana del clasificador morfológico, que indica si la pieza es válida a ojos de este. Además, en cada iteración se almacena en la estructura `SimilarityData` un recuento del número de métricas que la pieza actual "supera" para cada pieza de referencia a fin de que otra entidad genere, a posteriori, la respuesta final de qué pieza es a la que más se parece.

2.3.4.- Identificación de la pieza

Ya encontrándose la pieza totalmente dentro del campo de visión de la cámara y tras haber reconocido el tipo de pieza de que se trata mediante el clasificador morfológico considerando como referencias válidas las de la base de datos, comparado la matriz de métricas de similitud para todas las imágenes y el frame actual construyendo una lista ordenada de los mayores parecidos según las métricas consideradas y averiguada la pieza de referencia más parecida, ahora toca computar los detectores, descriptores y "matchers" de las piezas en tiempo real, pues si la clasificación realizada es exitosa, lo indicado es estimar una homografía entre la pieza real y la de referencia y extraer sus parámetros de rotación, traslación y orientación. Aunque (como se hizo para otras variables que requerían un cómputo intensivo) se pueden calcular de antemano y cargar al inicio de la aplicación todos los keypoints y descriptores de las imágenes de referencia, se decide calcularlos junto con los frames actuales en cada ciclo porque en la fase de experimentación se elaboró una interfaz que permite modificar numerosos parámetros del algoritmo a voluntad, para buscar la mejor combinación posible de detectores, descriptores y match particularmente a las condiciones de la simulación y la realidad.

Lograr un balance ente la eficiencia computacional, la precisión en la localización y correspondencia de los keypoints y la robustez del método es un apartado primordial en el que recapacitar en cualquier problema de este tipo. Por una parte se podría pensar que conviene emplear detectores y descriptores invariantes a rotación porque el propósito principal del sistema es detectar piezas que pueden venir orientadas de diferentes formas en el plano de cinta, pero por otra parte puede considerarse más prudente forzar que los métodos sean notoriamente sensibles a la rotación para realizar una correspondencia de keypoints mucho más austera y (casi) obligando a la piezas a que estén bien orientadas para que el clasificador característico las analice. Como la última opción puede proporcionar demasiadas falsas correspondencias o keypoints que estén fuera de la propia pieza se decide que es más sensato buscar la máxima invarianza de los detectores y descriptores (no solo a rotación, sino a traslación, escala e intensidad por razones análogas, como se ha estado buscado a lo largo de todo el algoritmo previamente explicado).

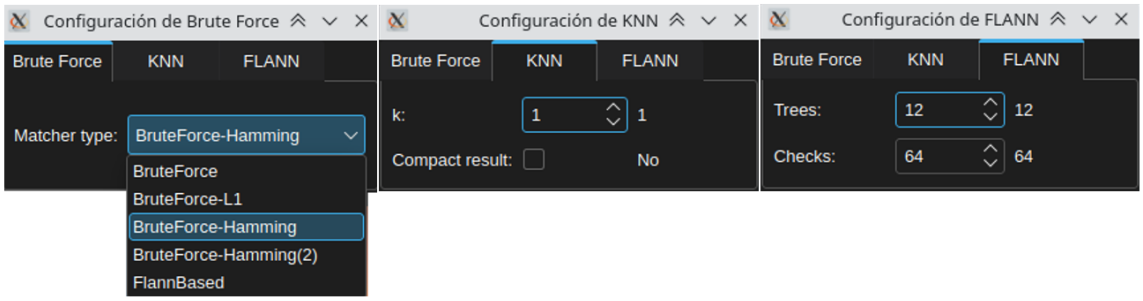


Fig. 81. Parámetros configurables desde las GUI de matchers para la fase de experimentación.

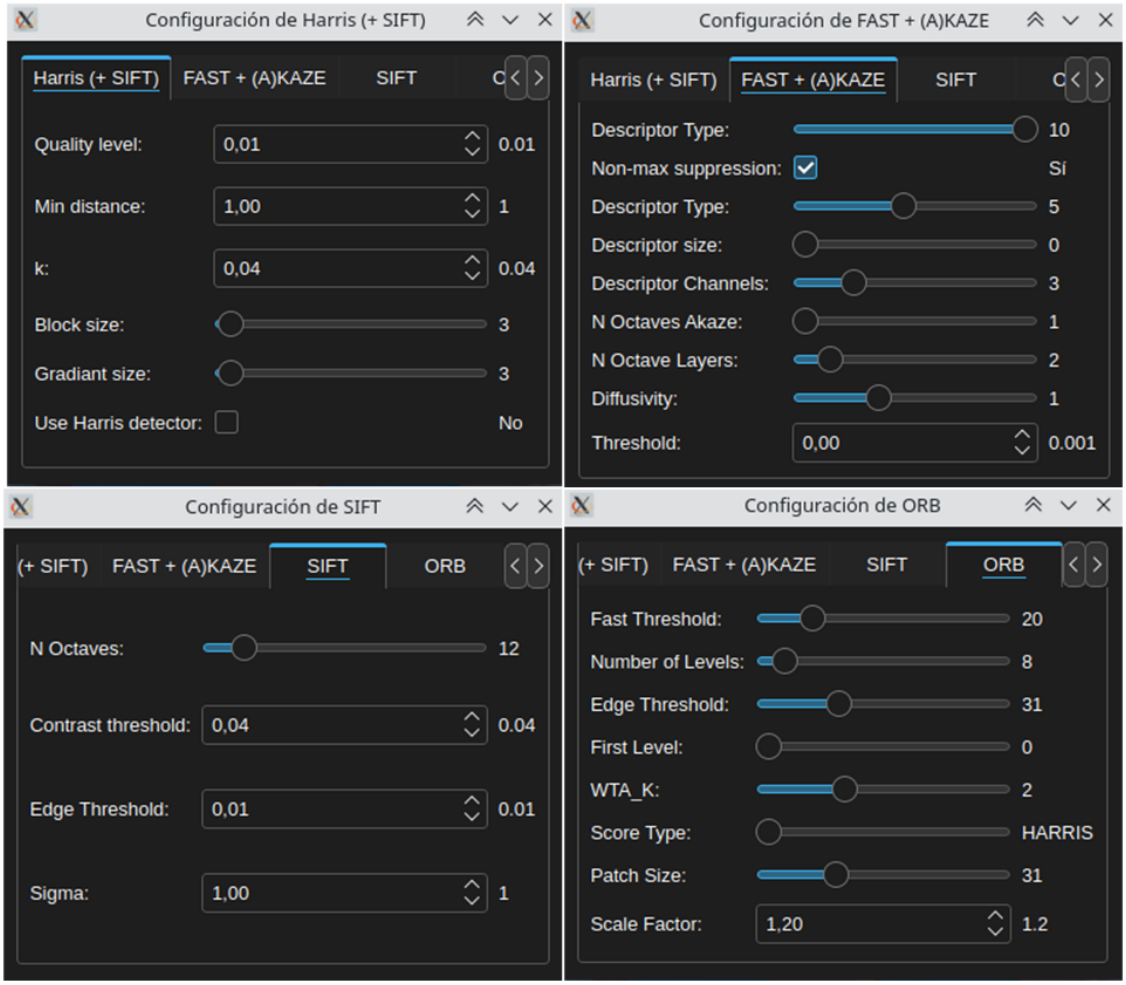


Fig. 82. Parámetros configurables desde las GUI de detectores y descriptores para la fase de experimentación.

Para esta labor se ha desarrollado una entidad denominada "clasificador característico" que emplea diversas técnicas visión artificial avanzada para proporcionar una respuesta fiable de la pieza de referencia más similar a la actual (ya sea real o simulada). El clasificador característico lleva a cabo pues, mediante los mismos algoritmos, dos tareas diferentes: una primera en la que se complementa con el clasificador morfológico para, comparando del número de correspondencias entre la imagen actual y todas las imágenes de referencia, proporcionar la respuesta final del "mayor parecido" de entre las piezas disponibles en la base de datos y, seguidamente, utilizar este "mayor parecido" (ya durante el resto del procesamiento) para almacenar iterativamente la pose relativa de la pieza actual con respecto de la (bien orientada) de referencia basándose en la descomposición dela matriz de transformación computada gracias a las características de las parejas de keypoints encontradas, hasta que la pieza actual abandone el campo de visión de la cámara (por la derecha) o hasta que se le ordene a fin de lanzar en ese momento una respuesta final óptima (pues tiene en cuenta diferentes "puntos de vista" [estadísticos basados en la respuesta del ECC, PCA, los descriptores…]) sobre la validez de la pose de la pieza.

Más aún, encontrar demasiadas pocas coincidencias en la primera ejecución del clasificador característico (esta primera iteración de este y el clasificador morfológico están controlados mediante flags manejados cuando se lanzan las señales correspondientes justo tras entrar la pieza en la cinta) sirve para detectar si hay una disparidad muy evidente en la pieza y cualquiera de las disponibles, con lo cual la pieza no debería ser considerada como válida y podría no tener mayor sentido continuar con el resto del algoritmo. En caso de que esta primera salida de este bloque sea un número de correspondencias suficientemente grande, el resultado se suplementaría al vector de votos del clasificador morfológico y el algoritmo general proseguiría iterativamente, ayudando este bloque a discernir el tipo de pieza concreta de que se trata y continuando seguidamente con la identificación de puntos característicos, correspondencias y el resto de tareas a continuación explicadas. En todo momento se pueden ajustar parámetros y umbrales en las entidades de clasificación para mejorar la sensibilidad y especificidad del sistema.

Para utilizar el clasificador morfológico y el característico para comparar una a una todas las piezas de referencia con la pieza actual y extraer la que más se parece se sigue este flujo de trabajo: se carga una lista de piezas de referencia desde el directorio de la base de datos, se utiliza un bucle `for` para iterar sobre cada pieza de referencia y se realiza una comparación con la pieza actual utilizando la entidad `clasificadorMorfologico`, se almacenan los resultados de la comparación en una estructura de datos llamada `similarityData` que contiene todos los valores de similitud de todos los sub-bloques y las métricas que se hayan decidido emplear, se imprimen los resultados en un log local y se ordenan según la similitud descendente (este bucle puede cortarse, para mayor velocidad y si se requiere, cuando se encuentre la primera coincidencia “perfecta” entre piezas), luego se utiliza otro bucle `for` para iterar sobre las imágenes ordenadas por similitud donde para cada pieza de referencia se realiza una comparación con la actual mediante la entidad `clasificadorCaracteristico`, se almacenan los resultados en una variable de tipo `std::vector<unsigned int>` denominada `mNumberMatches` que contiene el número de coincidencias entre las características de las piezas y posteriormente se selecciona la pieza con el máximo número de coincidencias (concretamente, se mantiene la que más matches tenga de las tres mejores piezas según los criterios de similitud del clasificador morfológico) y se persiste en una variable de tipo `cv::Mat` llamada `mMostSimilarImage`; además, el nombre de la pieza se extrae del nombre del archivo de la imagen relativa y se envía a través de la señal `sg_sendPieceName` hacia la GUI, junto con otra información de interés para monitorizar visualmente el estado del algoritmo. Adicionalmente, se realiza un preprocesamiento de `mMostSimilarImage` (filtrado de ruido, ecualización de histograma, operaciones morfológicas y otras técnicas de procesamiento de imagen) equivalente al realizado en el inicio del algoritmo para el fotograma actual.

El resultado de ambos bloques (`resultCM` y `resultCCMatches`) se utiliza para tomar decisiones durante el procesamiento posterior, informar al usuario de si la pieza se reconoce o no se identifica y, en caso negativo, abortar el procesamiento ulterior dando la respuesta definitiva (aunque prematura) de que la pose de la pieza no es válida, en cuyo

caso el sistema hubiera debido de detectar un error inaceptable durante su análisis (estas situaciones serían excepcionales durante el funcionamiento normal del sistema, pero se consideró muy importante y relevante considerarlas e implementarlas para ahorrar potenciales quebraderos de cabeza).

Fig. 83. Ejemplo de un objeto no reconocido y una pieza sí identificada por el sistema. En este caso se trata de un objeto extraño, pero tampoco reconocería una pieza que, aunque de forma conocida, tuviera una pose incorrecta.

A continuación se explican los algoritmos de visión artificial implementados en C++ en el método sobrecargado `Procesador::clasificadorCaracteristico` mediante OpenCV y otras librerías para identificar una pieza mediante técnicas analíticas. Cabe comentar que primeramente las imágenes se normalizan mediante `cv::compare`, `cv::minMaxLoc` y `cv::Mat::convertTo` para proporcionan un punto de entrada al bloque común a las imágenes porque en algunos métodos de más adelante se requiere que estén en otro formato (de tipo de dato) particular.

A través de la clase `NexoUIProceso`, que intercomunica el hilo (principal) de la GUI con los hilos de procesamiento, se obtienen (y escriben) de manera transparente los parámetros de proceso que se hayan configurado en la interfaz gráfica.

2.3.4.1.- Detección de puntos de interés

El método de detección de puntos de interés en este bloque utiliza diferentes algoritmos para identificar las regiones destacadas en las imágenes de las piezas. Los algoritmos implementados son los siguientes:

2.3.4.1.1.- Harris

Este algoritmo utiliza el detector Harris para identificar puntos de interés en forma de esquina y regiones de alta variación de intensidad en las imágenes. El algoritmo de Harris se basa en la matriz de autovalores de la derivada de la intensidad de la imagen, aprovechando el hecho de que esta derivada en una dirección particular se puede expresar como una matriz de convolución. En lugar de usar parches desplazados a ángulos de 45 º como en el detector de Moravec, Harris utiliza la derivada de la puntuación de esquina con respecto a la dirección directamente, lo que ha demostrado ser más preciso para distinguir discernir entre bordes y esquinas.

La detección de esquinas se realiza considerando una pequeña ventana "w" alrededor de cada píxel en la imagen (en el caso más simple la vecindad es 1). Simplificadamente, Harris define una función de energía E para cada píxel de la imagen y un umbral que se utiliza para identificar los que son de interés en base al valor de la función:

$$E(u,v) = \det(M) - k \cdot tr(M)^2, \tag{99}$$

donde "u" y "v" son las coordenadas de los píxeles, "det" indica el determinante de la matriz de autovalores, "tr" su traza y "k" es una constante definida típicamente entre 0,04 y 0,06 que se utiliza para ajustar la sensibilidad del detector. La matriz se calcula a partir de las derivadas de intensidad de la imagen en cada punto:

$$M = \sum_{(x,y)} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{100}$$

Primero, se aplica una máscara para normalizar la imagen y luego se utilizan parámetros configurables para ejecutar el algoritmo a través de la clase `cv::GFTTDetector`. Más específicamente, se utiliza el detector de puntos de interés GFTT (*Good Features to Track*) con opciones específicas para Harris; este método identifica esquinas y áreas de alta variación de intensidad.

Se configuran diversos parámetros para la detección de keypoints: `getMaxKp` representa el número máximo de keypoints a detectar (50000), `getQualityLevel_Harris` establece el umbral de calidad del detector (0,01), la distancia mínima euclidiana entre keypoints se define con `getMinDistance_Harris` detectar (1,0), el tamaño del bloque para el cálculo de derivadas se ajusta mediante `getBlockSize_Harris` detectar (3), `getGradiantSize_Harris` determina el tamaño de la vecindad para el cálculo del gradiente (3), la opción `getUseHarrisDetector_Harris` indica si se debe emplear

explícitamente el detector Harris detectar ("true"), y `getK_Harris` se corresponde al parámetro homónimo en la fórmula de Harris detectar (0,04).

En las dos siguientes figuras se ve la variación de la localización de parejas en función de nivel de calidad y la distancia mínima de Harris, respectivamente. Mayores valores en ambos parámetros provocan un menos número de keypoints detectados pero más distintivos y fiables (y viceversa); dependiendo de la aplicación, puede interesar disponer de muchas parejas probablemente precisas o pocas parejas aseguradamente correctas… ambas situaciones son viables, como se discutirá más adelante.

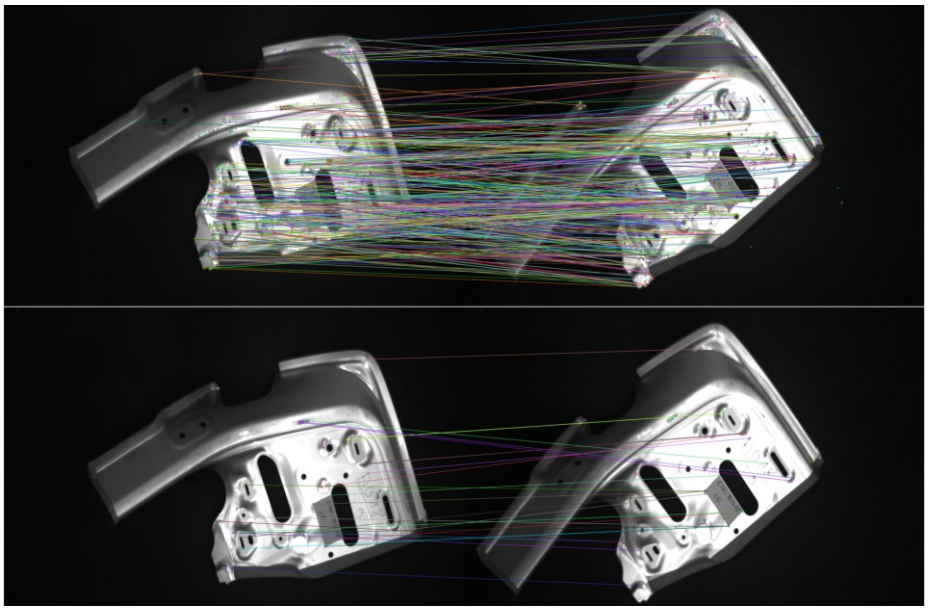


Fig. 84. Correspondencias con Harris (detector), SIFT (descriptor) y FLANN (matcher) empleando un nivel de calidad de 0,01 (arriba) y 0,15 (abajo).

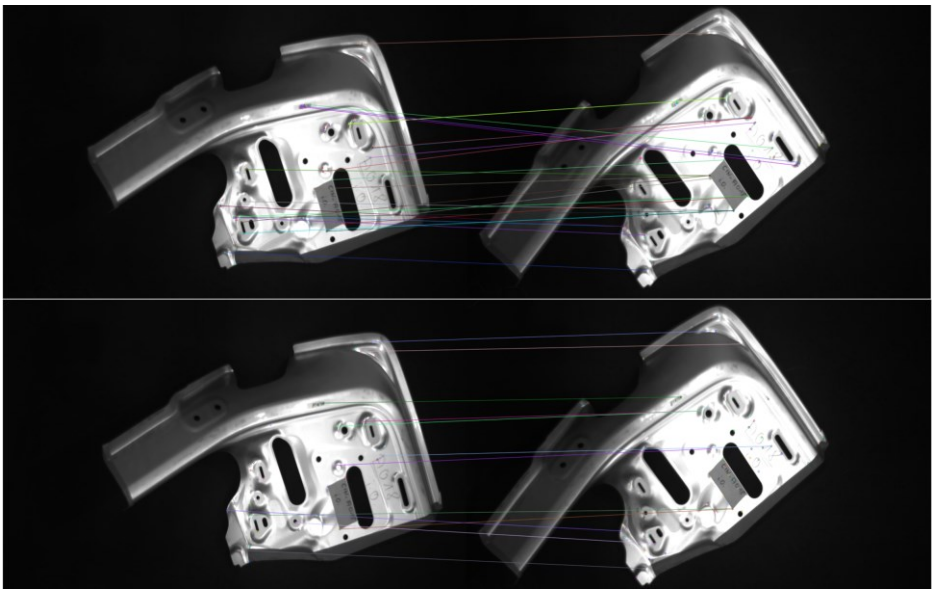


Fig. 85 Correspondencias con Harris (detector), SIFT (descriptor) y FLANN (matcher) empleando una distancia mínima de 1 (arriba) y 20 (abajo).

También se ha notado que cuanto menor es la sensibilidad "k" del detector más rápida y laxa es la detección y que a medida que se el kernel para el cálculo de derivadas se hace más grande más keypoints se detectan pero (lógicamente) más lento es el proceso, significando además que una aproximación más general de las derivadas provocan que se capturen frecuencias más bajas y se dejen de detectar puntos característicos cercanos a zonas como, por ejemplo, agujeros o esquinas de alto contraste (ver la siguiente figura); este último efecto es análogo (y aún más nítido o claro) a lo acaecido al variar el tamaño de la vecindad para el cálculo del gradiente.

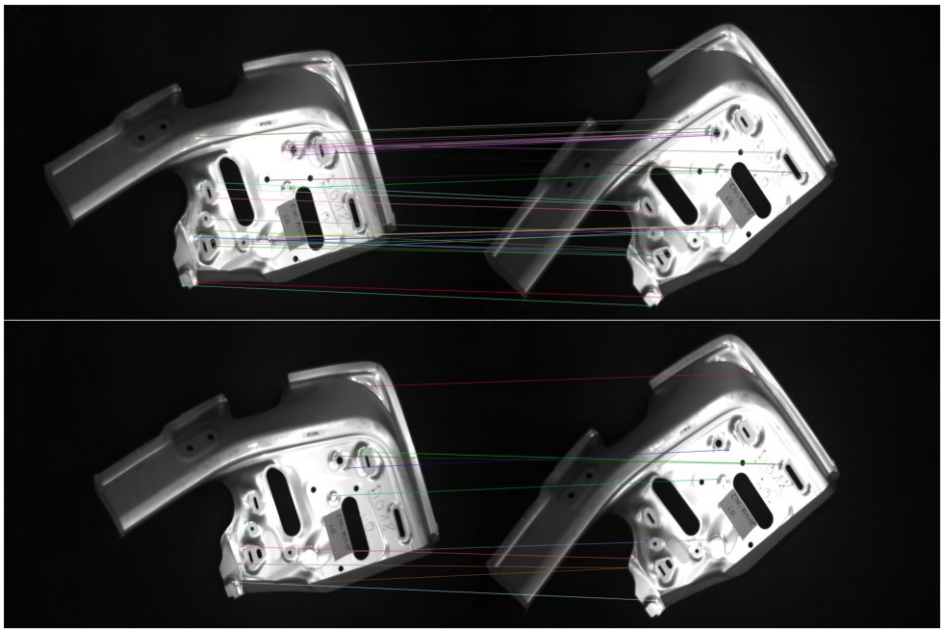


Fig. 86 Correspondencias con Harris (detector), SIFT (descriptor) y FLANN (matcher) empleando un tamaño del kernel para el cálculo de derivadas de 3 (arriba) y 15 (abajo).

En la siguiente figura se puede ver una caso de uso en el que durante varias iteraciones del clasificador característico se monitoriza el valor de rotación relativa, extraída de la homografía estimada, devuelto, usando a Harris como detector, SIFT como descriptor y FLANN como matcher (ver siguientes apartados) (donde la orientación real de la pieza es de 20 º). Sobre la iteración número 50 se aumenta drásticamente el umbral de calidad del detector, lo que causa que el sistema se vuelva inestable y produzca respuestas erróneas y muy poco fiables; esta situación puede ser típica en casos en los que se dejen de detectar keypoints, correspondencias o el número localizado sea muy pequeño por culpa de distintos factores adversos del entorno. Sobre la iteración 65 se devuelve el sistema a sus parámetros nominales (los mencionados en el párrafo de arriba) y su rendimiento vuelve a ser correcto, aunque en este caso la media de todas las estimaciones de rotación (línea azul) queda desplazada por un offset temporal, por culpa de dichos outliers:

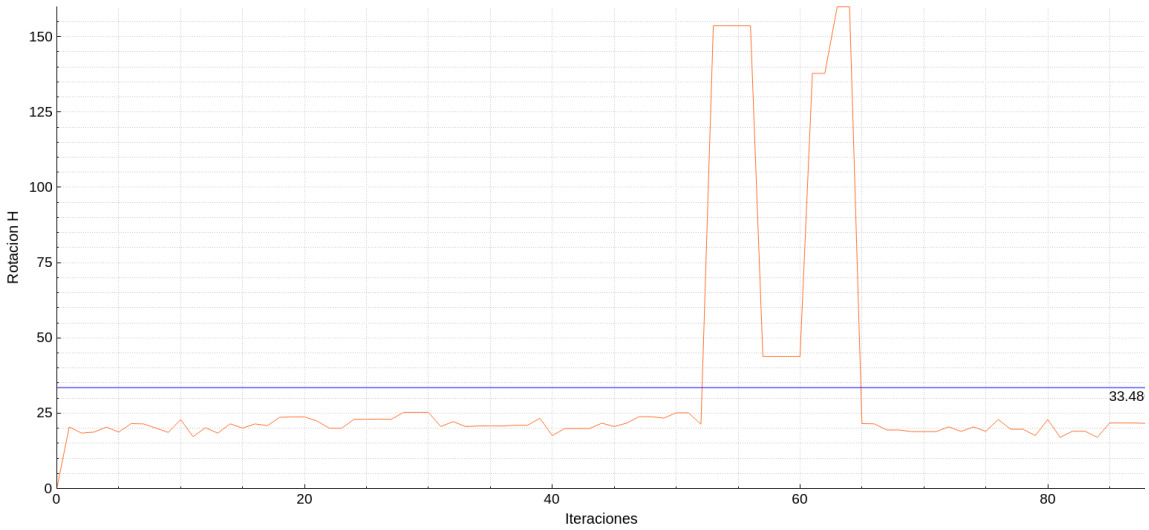


Fig. 87. Desestabilización y falla (causada artificialmente) en la estimación de la pose de mano del clasificador característico por falta de coincidencias (correctas).

2.3.4.1.2.- SIFT

El método SIFT (*Scale-Invariant Feature Transform*) se utiliza para la detección de puntos de interés en forma de "blob". Utiliza pirámides de imágenes para encontrar keypoints invariantes a la escala y la rotación, por lo que es un algoritmo muy robusto y eficaz. De manera general, SIFT crea en primer lugar un espacio de escala dividido por octavas suavizadas de la imagen, luego genera otra pirámide de imágenes mediante diferencias de gaussianas (DoG) como aproximación del laplaciano de la gaussiana (LoG) y busca máximos y mínimos locales en dicho espacio (para cada píxel y analizando también cada escala inferior y superior) para ubicar potenciales ubicaciones de keypoints, posteriormente realiza una interpolación con una cuádrica 3D mediante una expansión en serie de Taylor para mejorar la precisión de la posición de los keypoints (un píxel será punto de interés si es el máximo de entre sus 26 vecinos [nueve inferiores y superiores y ocho a su escala]) y ajustar su escala y, finalmente, se descartan puntos de bajo contraste o cercanos a contornos.

Se configuran varios parámetros para adaptarse a las características de las imágenes de las piezas y luego se utilizan para ejecutar el algoritmo a través de la clase `cv::SIFT`: `getMaxKp` define el número máximo de keypoints a detectar (50000), `getNOctaveLayers_SIFT` establece el número de capas por octava en la pirámide (12), el umbral de contraste para la detección de keypoints se define con `getContrastThreshold_SIFT` (0,04), `getEdgeThreshold_SIFT` representa el umbral para la detección de bordes (0,01) y la desviación estándar de la suavización gaussiana se controla mediante `getSigma_SIFT` (1,0).

Se ha comprobado que a cuantas más octavas se dedique SIFT, naturalmente, más tiempo tardará esta entidad en generar una respuesta en cada iteración, en una proporción de aprox. 1:1,25 con cada nueva octava, aunque para esta aplicación no es tan drástica la reducción de correspondencias o número de keypoints encontrados en tanto en cuanto se

aumentan las octavas, pues las piezas están prácticamente a la misma escala durante todo su transcurso. En la siguiente figura se muestra esto, donde primero se comparan dos piezas empleando 24 octavas (arriba) y luego empleando una sola (abajo); a partir de 30 octavas el algoritmo se vuelve impracticable por el alto coste computacional:

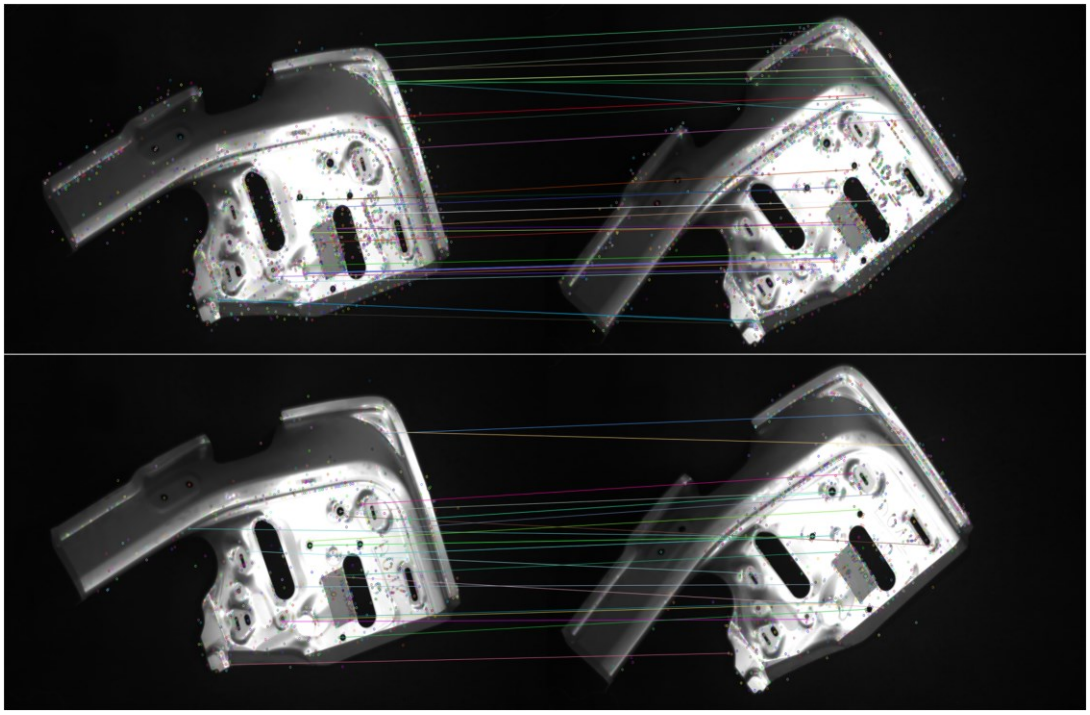


Fig. 88. Ejemplo de correspondencias con SIFT y FAST empleando distinto número de octavas.

Y en las dos siguientes figuras se ve la variación de la localización de parejas en función de los valores del umbral de contraste y el de la detección de bordes, respectivamente. Menores valores del primer parámetro provocan un menos número de keypoints detectados pero más distintivos y fiables (valores muy pequeños provocan incluso correspondencias muy lejos de la pieza). Mayores valores del segundo parámetro significan puntos de interés más precisos y ajustados al contorno del objeto, pero también un menor número de ellos, en general. No se han encontrado influencias significativas en el detector variando la desviación típica de la gaussiana en la primera octava, más allá de "comerse" algunos keypoints a causa del suavizado de la pieza con valores mayores; el sistema ya cuando con etapas de preprocesado que buscan este efecto.

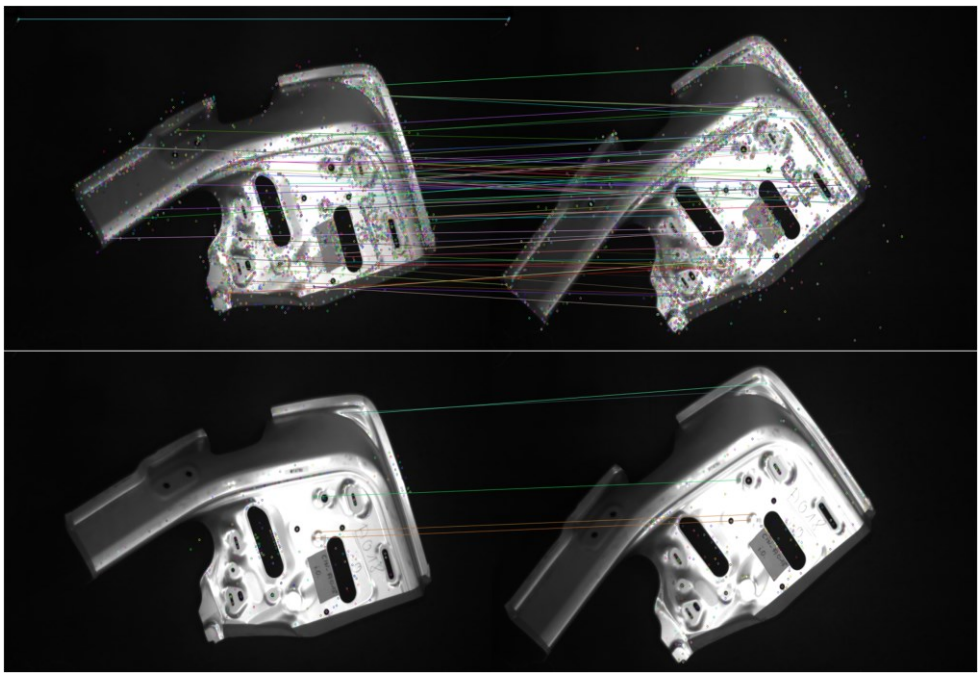


Fig. 89. Correspondencias con SIFT y FAST empleando umbrales de contraste de 0,01 (arriba) y 0,15 (abajo).

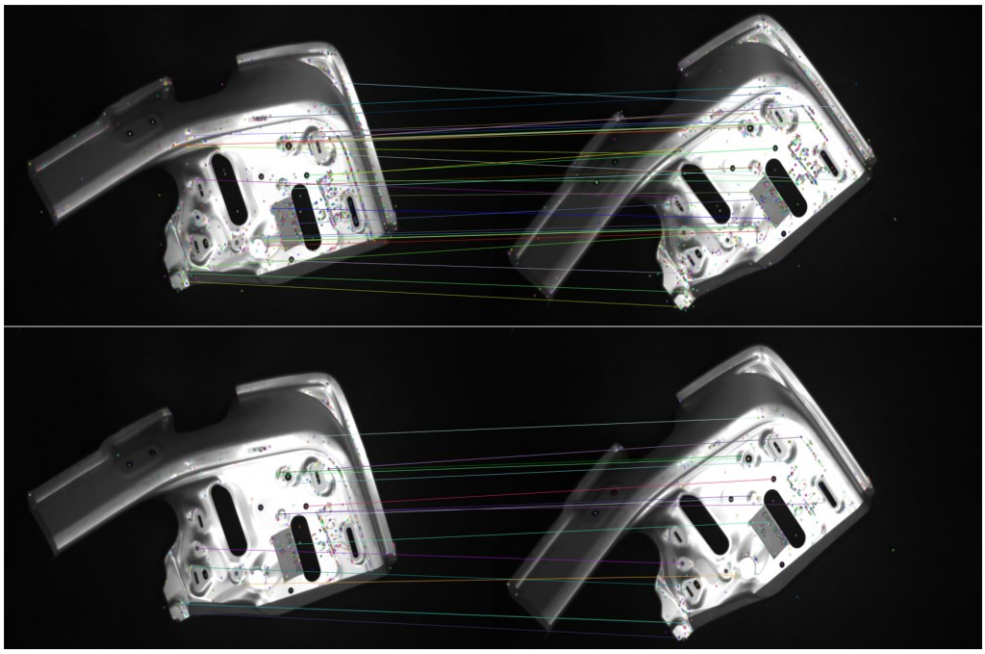


Fig. 90 Correspondencias con SIFT y FAST empleando umbrales de bordes de 0,25 (arriba) y 0,5 (abajo).

2.3.4.1.3.- FAST

FAST (*Features from Accelerated Segment Test*) es un algoritmo de detección de esquinas rápido y eficiente. De manera general, funciona analizando vecindarios circulares de 16 píxeles (empleando el algoritmo de Bresenham o del punto medio para circunferencias); marca cada píxel de una región como más brillante u oscuro que un umbral determinado (que se define en relación con la distancia al centro del círculo) y se

considera que esa vecindad se corresponde con un punto de esquina si contiene un número de píxeles contiguos marcados como más brillantes o más oscuros (también diferenciaría bordes y otras formas en fusión del número y la posición de los píxeles). FAST también tiene la cualidad de poder determinar bajo ciertas condiciones si una región es o no una esquina comprobando solamente 2 o 4 píxeles en lugar de 16, apresurando mucho más el cómputo.

Se configuran varios parámetros para adaptarse a las características de las imágenes de las piezas y luego se utilizan para ejecutar el algoritmo a través de la clase `cv::FAST`: `getThreshold_FAST` determina el umbral en la diferencia entre la intensidad del píxel central y los píxeles de un círculo alrededor de este píxel (10), y la opción `getNonMaxSuppression_FAST` indica si se debe realizar la supresión de no máximos ("true").

A continuación se ven dos ejemplos de localización de correspondencias usando FAST (como detector), AKAZE (como descriptor) y FLANN (como matcher). En el primer caso (piezas de arriba) se emplean parámetros nominales (los mencionados en el anterior párrafo) y en el segundo caso (piezas de abajo) no se hace la supresión de no máximos, conservando muchas más correspondencias aunque no tan fiables como en el primer caso o con un ratio de Lowe bajo.

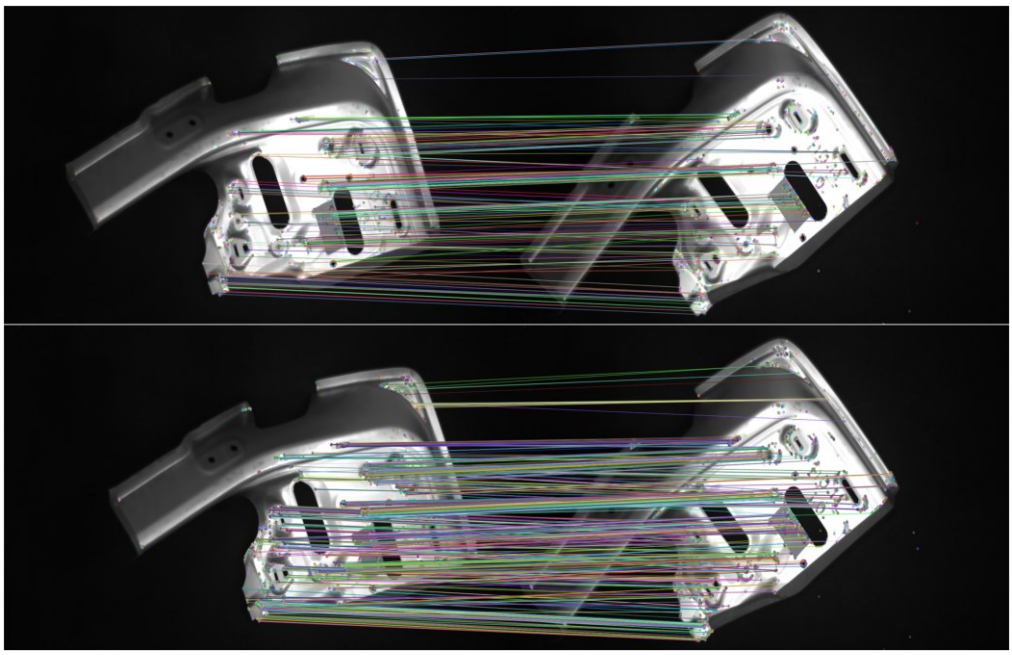


Fig. 91. Comparación entre correspondencias encontradas (inliers) con FAST, AKAZE y FLANN empleando (arriba) y no (abajo) supresión de no máximos.

2.3.4.2.- Descripción de puntos de interés

Una vez se han identificado los puntos característicos, se utilizan técnicas de descripción de su vecindad para representar cada uno mediante un descriptor. Los algoritmos de descripción implementados son los siguientes:

2.3.4.2.1.- SIFT

El mismo algoritmo SIFT utilizado para la detección de puntos de interés también se puede emplear para describirlos y extraer sus descriptores. Para ello SIFT calcula la orientación y magnitud del gradiente local en su vecindad en ocho direcciones (de 45 en 45 º); el vector de características que describe la información local de la región alrededor de cada keypoint está subdividida en 16 regiones cuyos gradientes también son calculados, además de que se rotan todos los gradientes relativos a aquel de mayor magnitud en cada área para lograr invarianza a la rotación (la invarianza a la escala se conseguía ponderando el histograma por la magnitud del gradiente a la escala considerada); esto resulta en un histograma de 128 elementos que se corresponde con el descriptor buscado.

A continuación se puede ver la correspondencia de puntos de interés empleando un umbral de contraste nulo (absolutamente permisivo) y otra caso en el que dicho umbral se aumenta solamente en una centésima; se nota que el ajuste de esta parámetro es crítico en el funcionamiento óptimo del sistema (y está relacionado principalmente por el tipo de fondo que rodea al objeto y las zonas homogéneas):

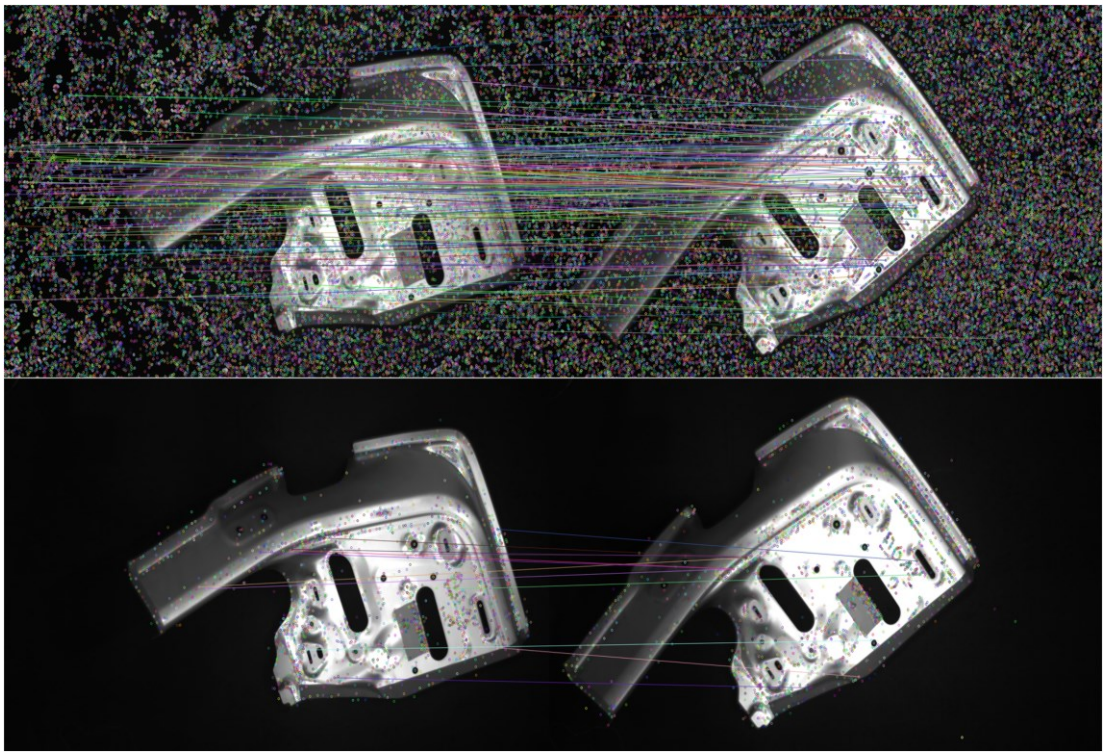


Fig. 92. Arriba las correspondencias, tras SIFT, sin umbral de contraste; todo el fondo es prácticamente una amalgama de keypoints. Abajo, el umbral aumentado en una centésima; el resultado mejora drásticamente.

Los parámetros del descriptor de SIFT son los mismos parámetros que toma de entrada el constructor de la clase que se empleó para computar los puntos de interés en el apartado relativo a SIFT anterior.

2.3.4.2.2.- (A)KAZE

También se propone utilizar el algoritmo KAZE (y opcionalmente AKAZE [*Accelerated* KAZE]) para describir los puntos característicos. KAZE utiliza un enfoque diferencial (pues se basa en el determinante normalizado de la matriz hessiana calculado a múltiples niveles de escala) para la detección de puntos clave, identificando regiones con cambios significativos en la intensidad de la imagen, para lo cual construye un espacio de escala no lineal a través de un filtro de difusión que permite una adaptación flexible del desenfoque local a las variaciones de tamaño en las características de las imágenes y genera descriptores de regiones invariantes a rotación, escala, y afinidad (limitada) para cada keypoint detectado (eligiéndolos como los máximos de las respuestas mediante una ventana móvil), capturando así información detallada sobre la textura y la estructura local de estos. AKAZE, por otra parte, utiliza un marco denominado Difusión Explícita Rápida (FED) para construir sus espacios de escala no lineales, lo que acelera de manera significativa el cómputo. Su detector se basa en el determinante de la matriz hessiana y la calidad de la invarianza a rotación se mejora empleando filtros de Scharr. También calcula un descriptor robusto basado en un algoritmo denominado Diferencia Binaria Local Modificada (MLDB), que aprovecha la información del gradiente del espacio de escala no lineal, consiguiendo será una versión más eficiente que KAZE (varias órdenes de magnitud más rápido) y resultados comparables para muchos conjuntos de datos.

Se configuran varios parámetros para adaptarse a las características de las imágenes de las piezas y luego se utilizan para ejecutar el algoritmo a través de la clase `cv::AKAZE`: `getDescriptor_type_AKAZE` especifica el tipo de descriptor utilizado por AKAZE (MLDB), `getDescriptor_size_AKAZE` determina el tamaño del descriptor (0, que indica el máximo tamaño posible, 128 bytes), `getDescriptor_channels_AKAZE` establece el número de canales para la representación visual (3), el umbral del algoritmo (que re relaciona con el valor del determinante de la Hessiana de la imagen, que mide la variación de intensidad de los píxeles en sus regiones) se controla mediante `getThreshold_AKAZE` (0,001), el número de octavas y capas por octava se ajusta con `getNOctaves_AKAZE` (4) y `getNOctaveLayers_AKAZE` (4), respectivamente y, además, `getDiffusivity_AKAZE` define el tipo de difusividad utilizado (difusión anisotrópico o Perona-Malik con una constante de difusividad de 0,75).

En la siguiente figura se muestra el efecto de varias el número de octavas en AKAZE y también la cantidad de capas en cada una. Como parecería razonable pensar a priori, aumentar cualquiera de los dos parámetros vuelve al sistema más lento pero también más robusto (en la tarea de descripción distintiva de keypoints para su posterior correcto emparejamiento); cuanto mayores son estos valores, detalles más finos del entorno de los keypoints se es capaz de describir.

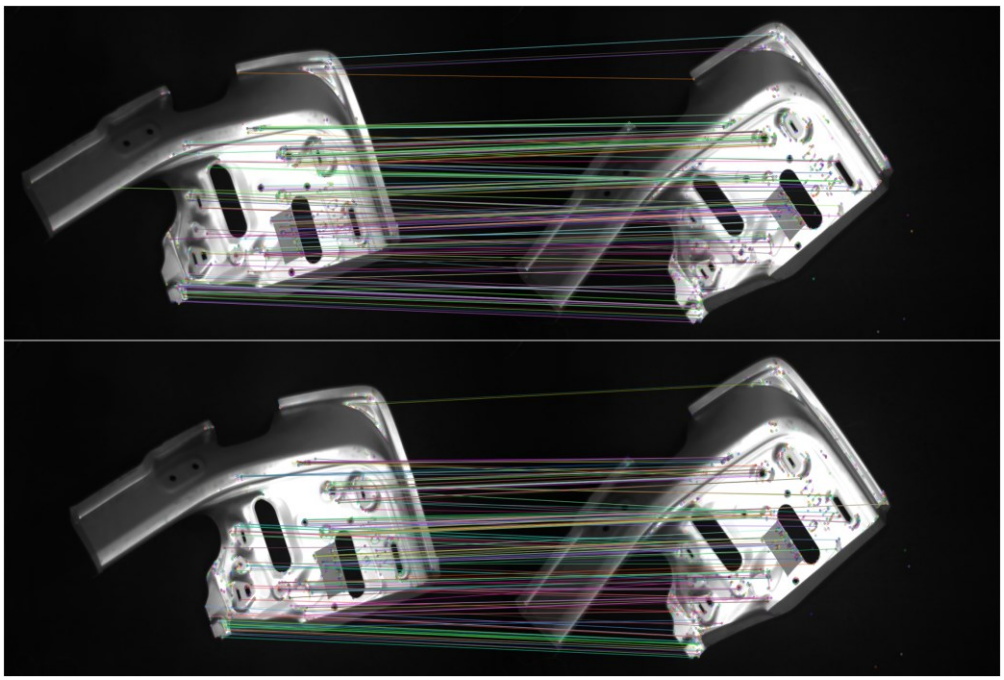


Fig. 93. Comparación entre correspondencias encontradas con FAST, AKAZE y FLANN empleando 10 y 10 (arriba) y 1 y 2 (abajo) capas por octava y octavas, respectivamente.

2.3.4.2.3.- ORB

El método ORB (*Oriented* FAST *and Rotated* BRIEF [*Binary Robust Independent Elementary Features*]) se emplea tanto para la detección como para la descripción de keypoints. Mejora los keypoints devueltos por FAST agregándoles una medida de su orientación involucrando el centroide de la vecindad ponderado por su intensidad y posteriormente genera descriptores binarios compactos mediante BRIEF: primero selecciona (pseudoaleatoriamente) un conjunto de pares de píxeles alrededor de cada keypoint, luego se comparan discerniendo qué punto tiene mayor valor y, con todas las comparaciones, se construye el descriptor binario (cada comparación contribuye con un bit del vector); además (pues BRIEF no es *per se* invariante a rotación), se aborda la invarianza a la rotación reorientando los descriptores basándose en la orientación calculada en la etapa de detección. Es un método sumamente rápido y especialmente adecuado, por la naturaleza y simplicidad de BRIEF, para la tarea de correspondencia de keypoints en tiempo real (típicamente mediante métricas de distancia como la de Hamming).

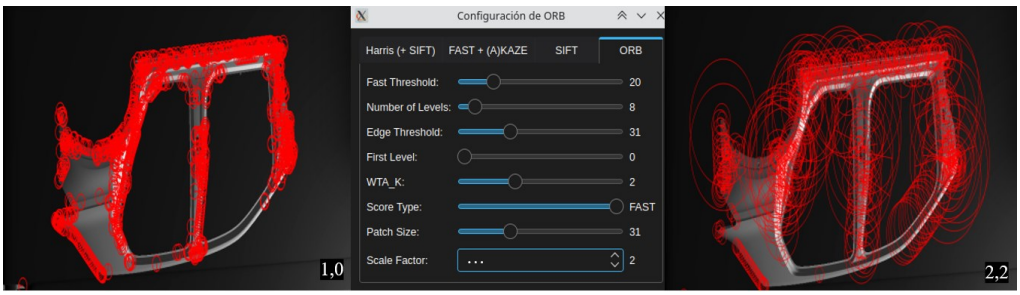


Fig. 94. Efecto del factor de escala del descriptor ORB.

Se configuran varios parámetros para adaptarse a las características de las imágenes de las piezas y luego se utilizan para ejecutar el algoritmo a través de la clase `cv::ORB`: `getScaleFactor_ORB` establece el factor de escala en la pirámide (1,2), `getNlevels_ORB` determina el número de niveles en la pirámide (8), `getEdgeThreshold_ORB` fija el umbral de intensidad de borde para la detección de keypoints (31), el primer nivel en la pirámide lo marca `getFirstLevel_ORB` (0), el número de puntos para el algoritmo de prueba `getWTA_K_ORB` (2), el tipo de puntuación o "score" se señala con `getScoreType_ORB` (se usa el algoritmo de Harris), `getPatchSize_ORB` indica el tamaño de la región local alrededor de cada keypoint (31) y el umbral de intensidad del algoritmo viene dado por `getFastThreshold_ORB` (20).

En la siguiente figura se pueden ver varias zonas en torno a las cuales ORB detecta y describe keypoints. Si el objeto presenta cierta asimetría, contornos angulosos, manchas distinticas no uniformes o iluminación desigual ORB puede ser adecuado, pues es un algoritmo muy versátil. Al subir el primer nivel en la pirámide los descriptores deberían provocar mejores correspondencias, pero no se ha notado que haya habido una mejoría substancial, además de que a partir de un valor de (aprox. 20) el peso del cómputo se vuelve excesivo.

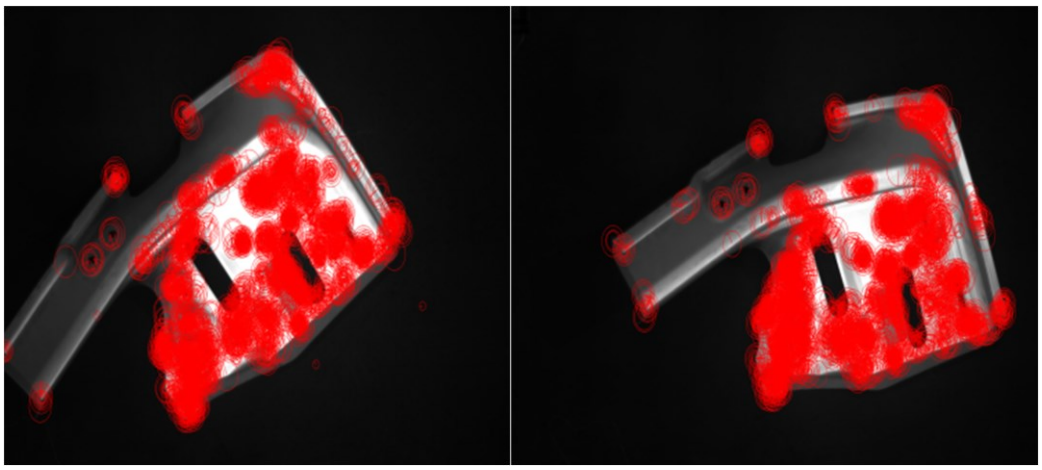


Fig. 95. Zonas de ejemplo en torno a las cuales ORB detecta keypoints.

En la siguiente figura se observa que, en esta aplicación, ORB provoca que la estimación de la homografía sea tanto más inestable (e imprecisa) cuanto mayor sea el "patch size"; en las zonas donde se observan "picos" que superan los 110 º se incrementó dinámicamente el tamaño de dicho parámetro para que tomara un valor mayor que 30. En las zonas donde se mantiene más estable (la orientación real de la pieza era de 20 º) se mantuvo en un rango de entre 3 y 15, logrando que capturara los detalles más finos de la pieza y, por tanto, consiguiendo que generara puntos característicos más distintivos y un mayor número de correspondencias correctas (una mejor estimación de la rotación relativa). En azul, el valor medio para todas las iteraciones:

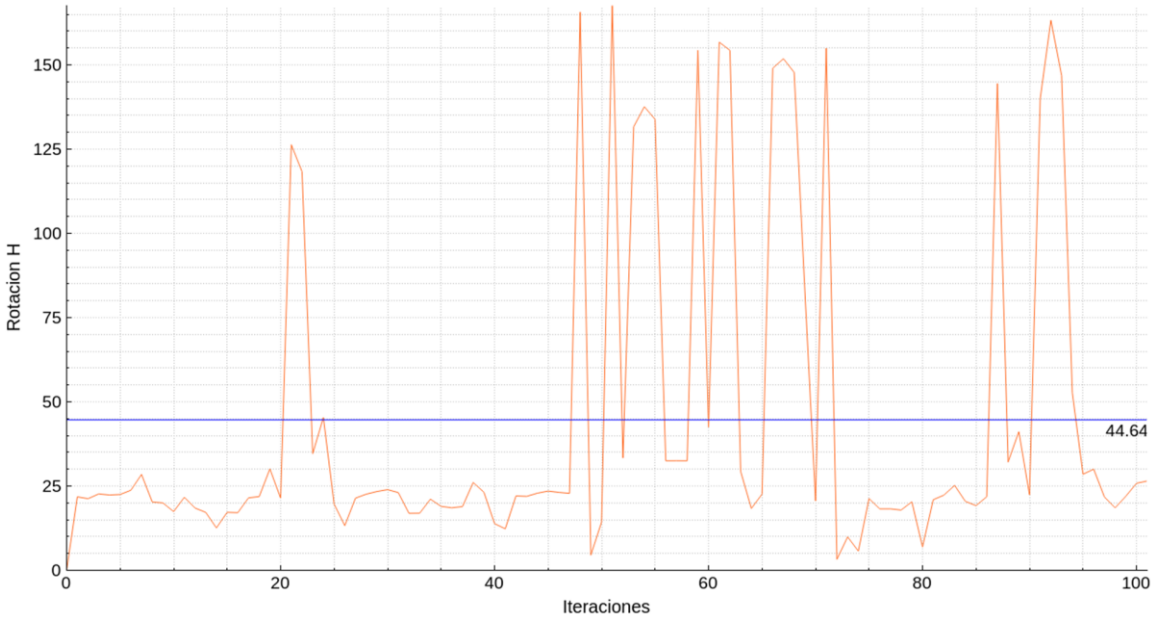


Fig. 96. Tendencia de ORB a volverse inestable cuando se aumenta el patch size y estable cuando se disminuye.

A continuación se ilustra en tres figuras, similarmente, cómo con «muchos» puntos de interés (sobre 2000) la estimación de la rotación relativa de la pieza es más estable (la orientación real es de 20 º), ajustada y precisa empleando como tipo de puntuación a FAST en lugar de a Harris, además de no usar el filtro geométrico (como se comenta en el apartado “Filtrado por ángulo y distancia”):

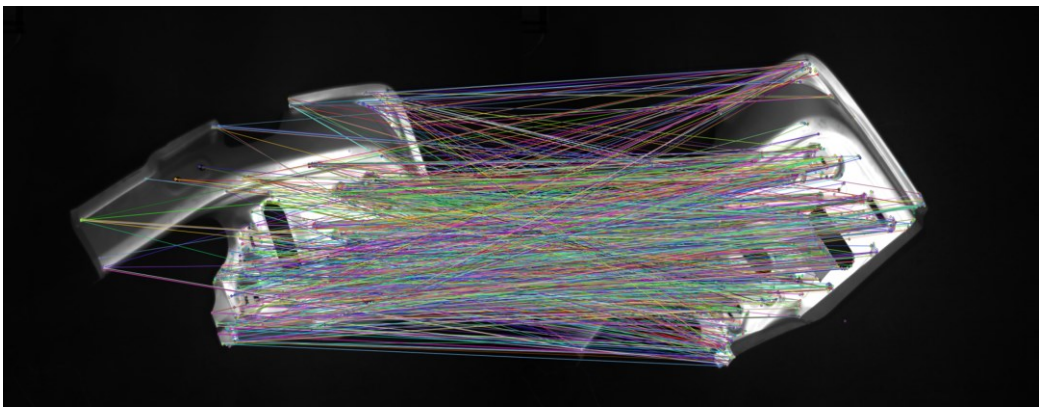


Fig. 97. Amasijo de correspondencias encontradas empleando ORB, con un ratio de Lowe unitario y sin filtro geométrico (sobre 2000 parejas).

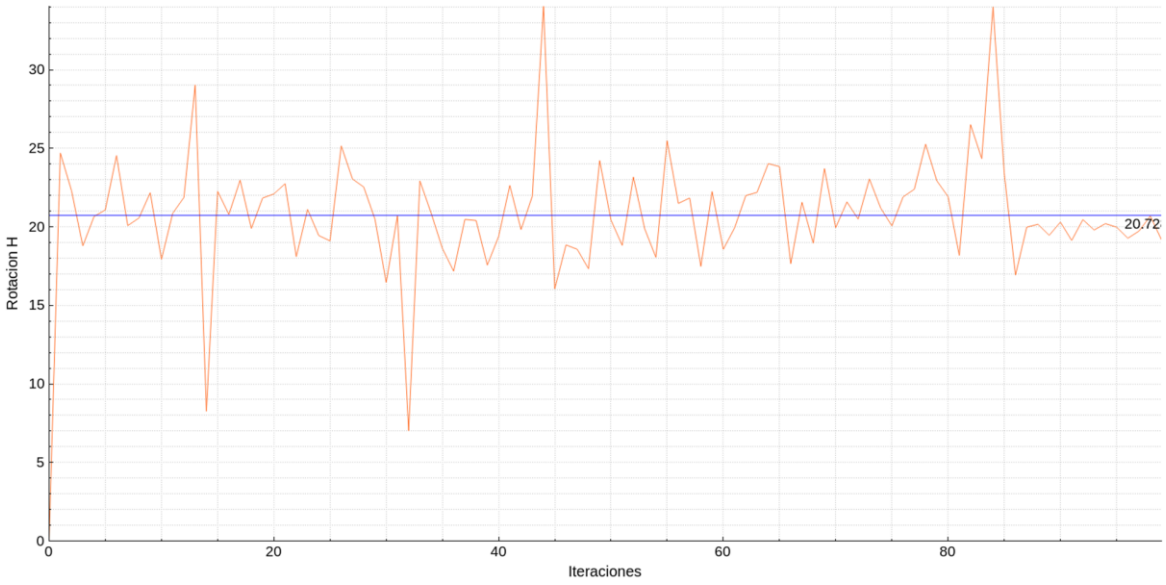


Fig. 98. ORB empleando como tipo de puntuación a Harris; sensiblemente menos estable y preciso que con FAST.

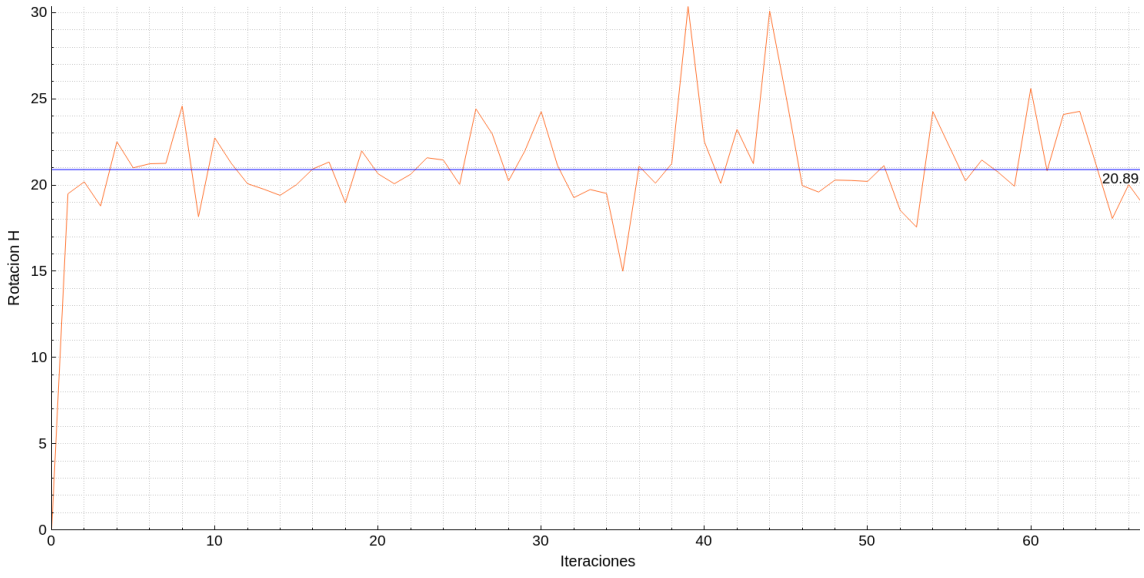


Fig. 99. ORB empleando como tipo de puntuación a FAST; sensiblemente más estable y preciso que con Harris.

2.3.4.3.- Correspondencia de puntos de interés

Una vez extraídos los descriptores de los puntos característicos de las imágenes de las piezas, se procede a la correspondencia de estos. Primeramente se da la opción de conservar únicamente los "n" keypoints de mejor (mayor) respuesta, donde el umbral puede ponerse en el número máximo de puntos permitidos establecido anteriormente o en uno algo menor para ser un poco más restrictivos con el filtrado de verdaderos y falsos positivos, para lo cual se ordenan los puntos de interés según su respuesta (atributo "response" de los objetos que son instancias de `cv::KeyPoint`) en orden descendente mediante una función lambda con `std::sort`. Los algoritmos de correspondencia implementados son los siguientes:

2.3.4.3.1.- Fuerza bruta

Un comparador por fuerza bruta (BF) (término que se asocia a un enfoque que da prioridad al agotamiento de todas las combinaciones posibles) requiere poca optimización; confronta dos conjuntos de descriptores de keypoints y genera un resultado que es una lista de coincidencias de tal forma que, para cada descriptor del primer conjunto, realiza comparaciones con cada descriptor del segundo conjunto, uno a uno. Cada comparación produce un valor de distancia y la mejor coincidencia puede elegirse en función de la menor distancia encontrada.

Un método que dé prioridad a ciertos criterios para mejorar inteligentemente la velocidad del cómputo puede tomar ciertos caminos que, eventualmente, la hagan obviar algunas correspondencias que no se rechazarían mediante fuerza bruta, pero puede no ser plausible para la aplicación particular emplear este matcher si el número de puntos de interés asciende, a decir, a más de 3 órdenes de magnitud. El equipo de que se dispone para los ensayos de este proyecto computa en décimas de segundos operaciones intensivas mediante C++ empleando fuerza bruta para más de 50000 parejas de puntos característicos, pero existen otros algoritmos, como los comentados posteriormente, que resultan en una solución mejor balanceada.

La ejecución del algoritmo es directa y se realiza a través de la clase `cv::DescriptorMatcher` (el tipo de matcher se determina mediante `getMatcherType` y los descriptores de ambas imágenes se comparan con el método `match`).

2.3.4.3.2.- FLANN

El algoritmo FLANN (*Fast Library for Approximate Nearest Neighbors*) se utiliza para realizar coincidencias de manera eficiente. Este método tiene la bondad de ahorrar tiempo a la hora de establecer coincidencias verdaderas entre puntos característicos, pues no espera a que terminen todas las iteraciones de análisis de posibles candidatos ni busca la coincidencia exacta, sino una aproximación que cumpla con ciertos criterios de similitud. Además, opera basándose en estructuras de datos como "índices" y árboles "kd" o "de búsqueda múltiple", que facilitan una búsqueda eficiente de correspondencias basándose en los vecinos más cercanos, permitiendo la recuperación de cierta información espacial.

Realmente, FLANN es una biblioteca para realizar búsquedas rápidas aproximadas del vecino más próximo en grandes conjuntos de datos y características de alta dimensión.

Se configuran varios parámetros para adaptarse a las características de las imágenes de las piezas y luego se utilizan para ejecutar el algoritmo a través de la clase `cv::DescriptorMatcher` (el tipo de matcher se determina mediante `getMatcherType` y los descriptores de ambas imágenes se comparan con el método `match`): el número de árboles se indica con `getTrees_FLANN` (12) y número de verificaciones realizadas durante la búsqueda del vecino más cercano en el espacio de descriptores mediante `getChecks_FLANN` (64).

2.3.4.3.3.- K-NN

Se implementa un enfoque de "k" vecinos más cercanos (K-NN) para buscar coincidencias. El método tiene como objetivo principal (en este ámbito) establecer correspondencias de puntos basándose en características específicas de otros puntos cercanos; estas características específicas son, efectivamente, los descriptores asociados. Presenta una alta similitud de enfoque con FLANN pero, mientras que K-NN realiza una búsqueda exhaustiva para encontrar los vecinos más cercanos, FLANN utiliza métodos más eficientes y aproximados para acelerar este proceso.

Se configuran varios parámetros para adaptarse a las características de las imágenes de las piezas y luego se utilizan para ejecutar el algoritmo a través de la clase `cv::FlannBasedMatcher` (los descriptores de ambas imágenes se comparan con el método `knnMatch`): entre otros específicos de la implementación propia de OpenCV, la cantidad de mejores coincidencias posibles encontradas por cada descriptor de consulta (si un descriptor tiene más de este valor) se indica con `getK_KNN` (1).

En las siguientes figuras se puede ver el efecto de la influencia de AKAZE en la tarea de proporcionar descripciones puntos característicos al matcher, relacionado con el desempeño de dos tipos de matchers diferentes… el método de fuerza bruta vuelve al sistema, en ocasiones, inestable (primera figura), mientras que K-NN (o FLANN) (segunda figura) (con un ratio de Lowe de 0,8 para hacer el efecto menos exagerado) el rendimiento del sistema aumenta drásticamente (la orientación real de la pieza es de 20 º en tales ejemplos):

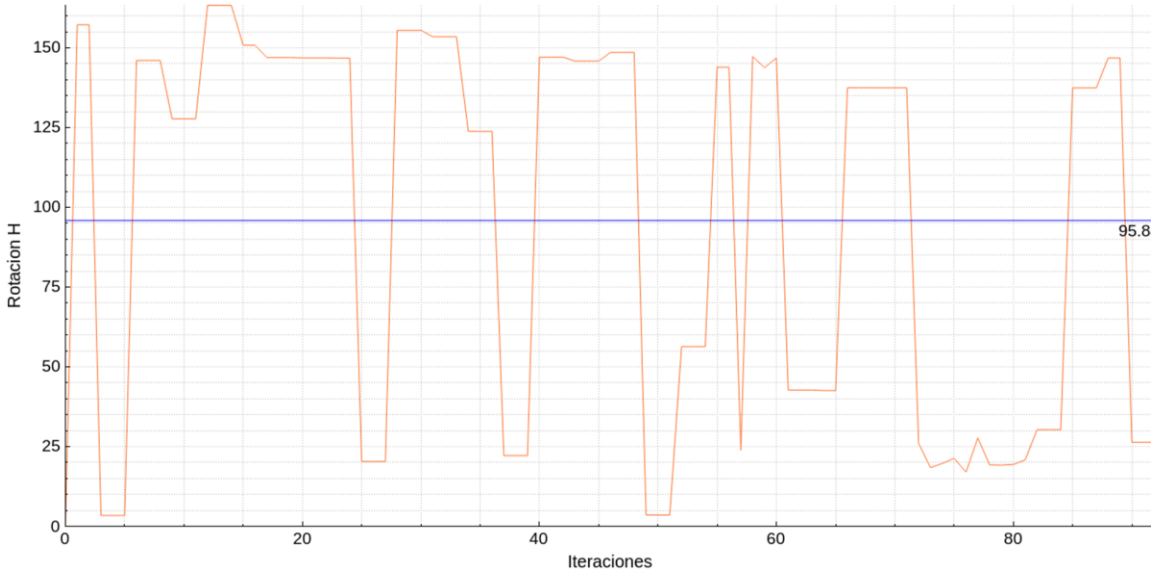


Fig. 100. Inestabilidad de AKAZE con fuerza bruta.

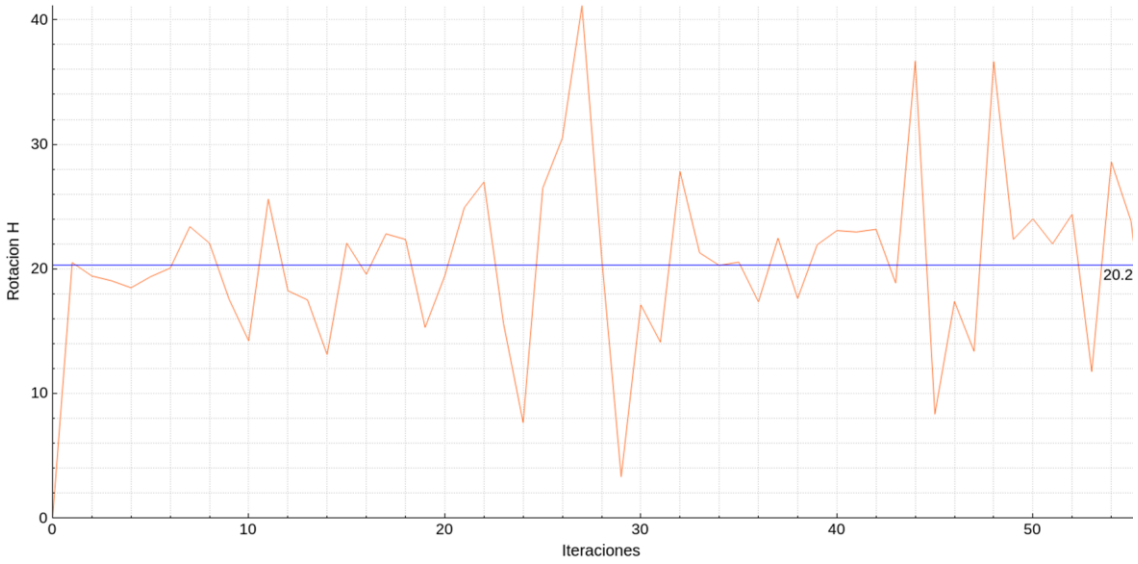


Fig. 101. Estabilidad de AKAZE con K-NN (similar a como sería con FLANN).

Cabe comentar que en todo momento se comprueba que ninguno de los vectores de keypoints, descriptores o correspondencias sea nulo, pues el mínimo de correspondencias ha de ser de 4 parejas para que se pueda calcular más tarde la homografía que las relaciona, en caso de que esto no se cumpla, no se "dibujarían" en la GUI los puntos ni las parejas encontradas y se devolvería un código de error para esa iteración en particular.

2.3.4.4.- Refinar las correspondencias

Se puede ver que el proceso completo de clasificación característica se basa en la combinación de distintos algoritmos para lograr una identificación robusta de la pieza en cuestión. Tras la realización de las correspondencias se aplican dos filtros (por lo menos), a continuación explicados, para mejorar y refinar aún más las coincidencias y eliminar aquellas que no cumplen con tolerancias o cualidades específicas. Este paso mejora sensiblemente la precisión del sistema al descartar coincidencias incorrectas o redundantes, que no contribuirían con más que ruido a la estimación de la pose de la pieza. A continuación se ve una figura que resume la idea básica de RANSAC, cuyo algoritmo completo se deja explicado en la literatura; se usa en el método que estima la homografía para aumentar el ratio entre inliers y outliers en las parejas de puntos característicos encontradas, siendo el primero de los tres métodos empleados específicamente para hacer más estable la inferencia de la rotación relativa (orientación) de cada pieza.

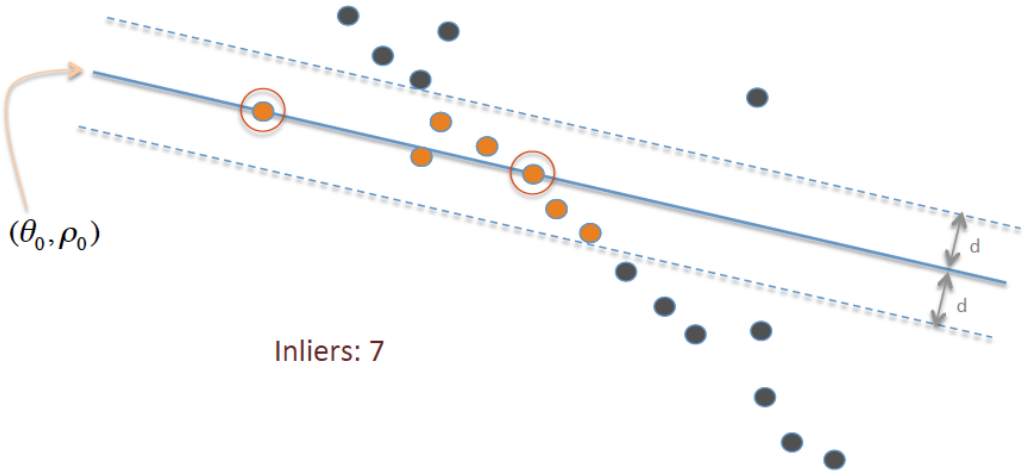


Fig. 102. Representación de RANSAC en una de sus iteraciones para ajustar una recta (parámetros θ_0 y ρ_0 [transformada de Hough]). Recogido de: https://en.wikipedia.org/wiki/Random_sample_consensus.

2.3.4.4.1.- "Ratio test" de Lowe

Una buena y recurrente práctica después de establecer una correspondencia entre puntos de interés es aplicar la prueba de proporción (*ratio test*) de Lowe (introducido por David G. Lowe, el mismo investigador responsable de SIFT) para filtrar las coincidencias de características basándose en la relación de similitud de sus pares más cercanos, constituyendo un método útil para hacer más robusta la definición de parejas firmes y rigurosas cuando el número de puntos es elevado, como es el caso del presente problema, en el que se busca sobredeterminar la solución y resultados más estables mediante muchas iteraciones.

Esto se hace en el método `Procesador::loweRatioTest`. Primero, gracias a los descriptores SIFT, ORB… y el matcher se buscan y almacenan las parejas más cercanas para las piezas en un vector de tipo `std::vector<cv::DMatch>`. Luego, para cada descriptor en la primera imagen, se consideran los dos descriptores más cercanos en la segunda imagen, se calcula la relación de las distancias entre el descriptor más cercano y el

segundo descriptor más cercano y se aplica un umbral entre 0 y 1 (de valor 0,7, típicamente [tanto mayor cuantas menos estricto se quiere que sea el filtrado]) al cociente de las distancias para mantener solamente los emparejamientos que cumplen con este umbral y descartar el resto. Es decir, se asume que todas las "segunda mejores" correspondencias para cada característica son malas pues se eliminan todas aquellas parejas que están suficientemente cerca de otra pareja la cual se mantiene si la distancia entre los descriptores correspondidos es suficiente pequeña y menor que las primeras parejas en una medida proporcional al umbral empleado; dicho de otra forma, "limpia" la vecindad de los puntos para asegurar que solo hay una pareja mantenida en cada vecindad y evita ambigüedades o dobles correspondencias, las cuales serían perjudiciales a la hora de establecer la transformación geométrica que relacione las imágenes entre sí.

En la siguiente figura se ve la idea de esta técnica. La probabilidad de que una correspondencia sea correcta se puede determinar tomando el ratio entre la distancia al primera vecino más cercano y al segundo; se dibuja la función de densidad de probabilidad (PDF); un valor de aprox. 0,75 para dicho ratio muestra un punto de corte o inflexión entre distribuciones de parejas correctas e incorrectas; a menor valor, mayor es la probabilidad de que la correspondencia sea correcta pero menor la de que aparezca (pues menos parejas supuestamente correctas hay usando ratios más pequeños).

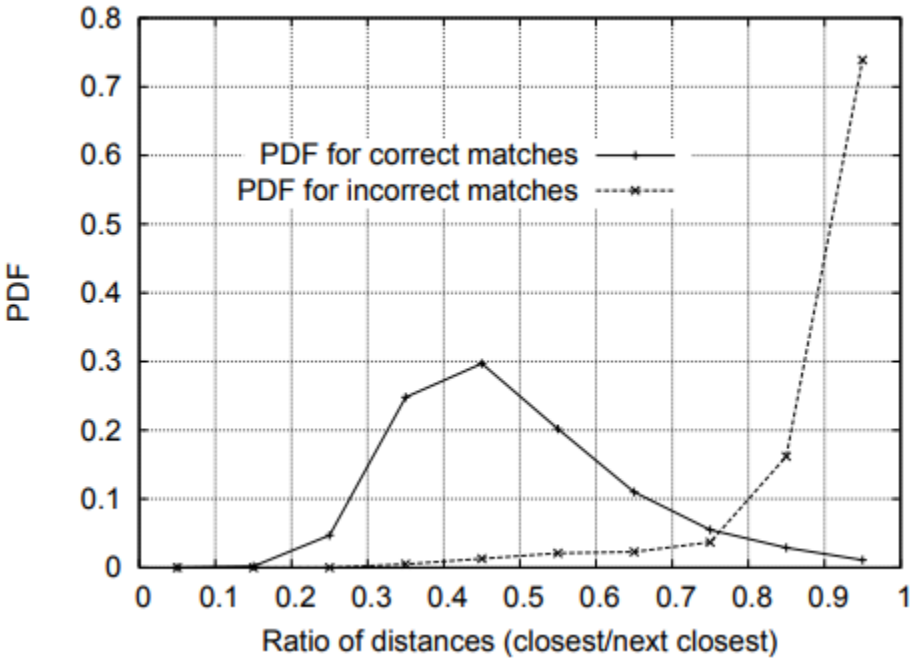


Fig. 103. Demostración original del ratio test de Lowe.

Esta técnica ayuda a mejorar la robustez del emparejamiento porque se asegura la unicidad de las parejas y la eliminación de falsos positivos. Los emparejamientos que pasan el "Lowe's Ratio Test" se consideran coincidencias válidas (de momento) y se utilizan para establecer las correspondencias definitivas en el siguiente paso.

2.3.4.4.2.- Filtrado por ángulo y distancia

Como no se considera suficientemente seguro suponer como completamente verdaderas las correspondencias de salida del paso anterior (pues el ruido en las imágenes no hace infalible el resultado de los matchers [que tampoco desmerecen] y los puntos detectados son una gran cantidad de ellos), se implementa en el método `Procesador::keypointsFilter` un algoritmo que realiza un filtrado de las parejas de acuerdo con los dos atributos que se consideran más importantes para validar la orientación de una pieza. El primero es la distancia que separa a un descriptor de otro (100 px., p. ej.); en concreto la distancia vertical, pues la pieza se estará moviendo horizontalmente y no habría manera de evitar esta variación salvo extrayendo la ROI de esta (que es lo que se hace, aunque además no es interesante ni aporta información sugerente a la distancia absoluta, pues el siguiente atributo ya contiene esta información). El segundo parámetro es el ángulo que forman los vectores de orientación principal o dominante de la vecindad de los descriptores de cada característica (p. ej., 15 º); este parámetro debe ser similar, sino idéntico, a la máxima orientación admisible de la pieza, pues valores muy dispares pondrán en riesgo la fiabilidad del sistema en este bloque, simplemente porque los keypoints que estén por debajo de este valor serán entendidos, por el clasificador característico, como falsos positivos. Este ángulo acotaría lo lejos que se le permitiría a una pareja de una imagen estar con respecto a la característica de la otra imagen, lo que permite un cálculo más ajustado de la variación en la orientación relativa de la pieza actual con respecto de la de referencia y se considera beneficioso porque si el número de correspondencias que sale de este paso es demasiado bajo, se puede entender (llegados a este punto) con poco lugar a equivocación que la pieza está mal colocada u ocurre algo extraño con ella. Si la pieza estuviese bien orientada (dentro de los rangos permitidos) o razonablemente fuera de los límites, el número de correspondencias finales sería suficientemente alto como para determinar la transformación geométrica de la que se deriva el giro y el desplazamiento que determinan si se puede o no dar por válida la pose de esta.

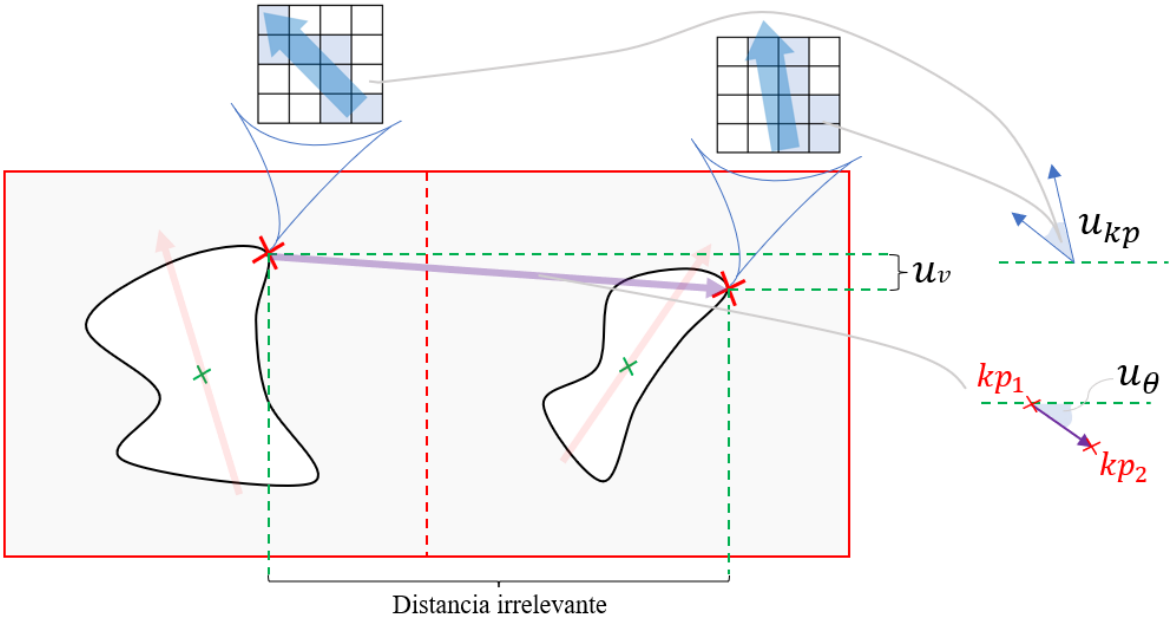


Fig. 104. Esquema que explica las (tres) restricciones geométricas aplicadas a los puntos característicos. Una por parte de la vecindad de cada punto característico de cada pareja encontrada y otras dos por el ángulo entre eje horizontal y el vector que los une y la distancia vertical entre ellos (componente vertical del vector).

En el método implementado se sigue el siguiente flujo de trabajo: calcular el ángulo relativo a la imagen, las diferencias angulares y la diferencia lineal en la posición vertical de los keypoints correspondientes de las dos imágenes, iterar sobre todas las coincidencias, filtrar según los criterios de los umbrales escogidos (ni la diferencia de ángulo ni posición debe superar los valores máximos permitidos) y actualizar los vectores de entrada con las correspondencias finales encontradas. Como medida adicional, si se desea que el filtrado sea aún más estricto y tenga en máxima consideración el avance de la pieza, se puede usar el umbral de la diferencia angular para filtrar una a una la orientación dominante de cada punto de interés, con lo que las únicas características que prevalecerían serían aquellas cuyo descriptor esté orientado en la dirección de avance de la cinta, lo en principio puede ayudar a reducir el ruido por tambaleos de la pieza pero en la práctica se asume insignificante.

La diferencia angular comienza en 0 º y se mide, sin importar el signo de las orientaciones, ángulo central subtendido por un arco cuya longitud es igual a la i-ésima parte de una circunferencia (con precisión simple) desde el punto de mayor angulación hasta el otro en sentido antihorario:

$$angleDifference = 180 - \big||\alpha_1 - \alpha_2| - 180\big|, \tag{101}$$

es decir, la diferencia angular se entiende como el camino angular más corto que lleva una orientación a la otra (habiendo normaliza previamente los ángulos entre 0 y 360).

La utilización de esta técnica tiene la ventaja de que con muy pocas correspondencias se puede obtener una estimación suficientemente precisa de la homografía como para que el sistema realice una clasificación correcta con muy alta probabilidad. Además, el tiempo

de cómputo se reduce drásticamente en comparación a lo que tardaría cada iteración si se empleasen todas las potenciales parejas que devolviesen los matchers. En la siguiente figura se puede ver una gráfica que muestra la rotación de una pieza a 20 º extraída de la homografía estimada a través de SIFT con un ratio de Lowe unitario (para que se detecten los máximos keypoints posibles) y con el resto de parámetros nominales (los especificados en el apartado "SIFT"); en poco tiempo el sistema devuelve, con estabilidad razonable, una estimación muy precisa de la orientación de la pieza:

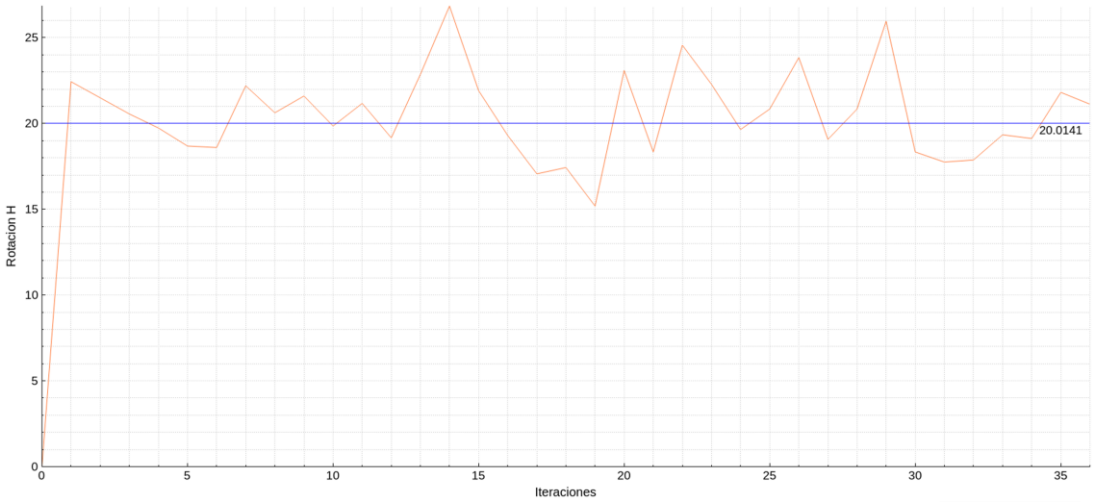


Fig. 105. Estabilidad de SIFT con filtro geométrico durante varias iteraciones.

Por otro lado, si se elimina el efecto del filtro geométrico (en las mismas condiciones nominales), el sistema tarda más en ejecutar cada iteración del clasificador morfológico pero la estabilidad y la precisión mejoran aún más, pues se nota que a medida que la pieza se desplaza hacia la derecha el número de keypoints del lado que se va ocluyendo desaparece pero queda compensado con otras buenas correspondencias que ahora tienen mayor probabilidad de aparecer en contraste con la situación en la que el filtro geométrico pudiera "censurarlas":

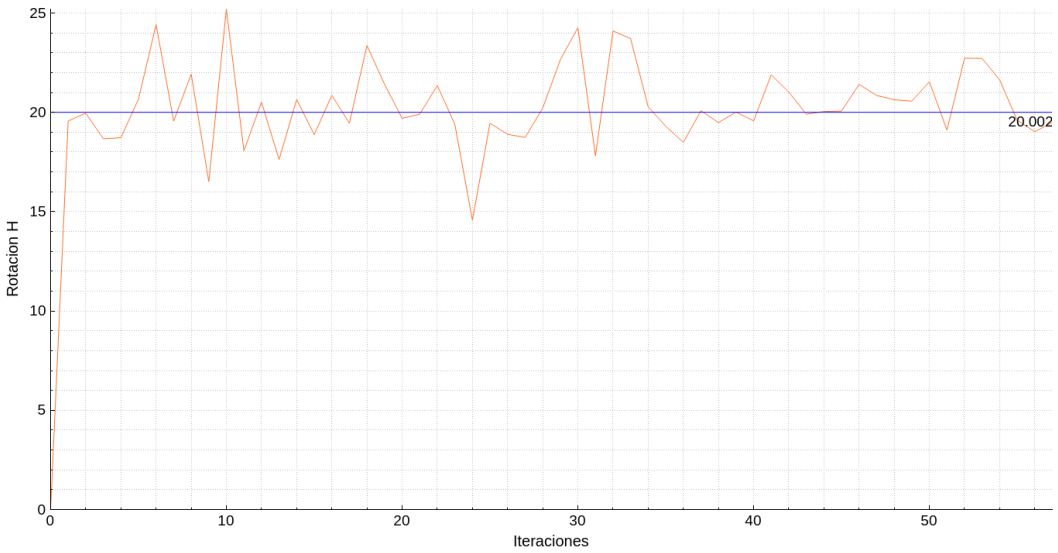


Fig. 106. Estabilidad de SIFT sin filtro geométrico durante varias iteraciones.

El sistema, respectivamente en los dos casos anteriores, pasa de detectar entre 100 y 200 keypoints, a localizar alrededor de 2000; la elección de optar por una solución conservadora con menos número de puntos característicos en vez de mantener cierta laxitud en su geometría depende de fines particulares… en esta experimentación, mantener unas restricciones similares a los límite de orientación admisibles (sobre 20 º y 100 píxeles en vertical para el filtro geométrico) fueron una elección suficientemente razonable.

2.3.5.- Cálculo de la pose final

Tras haber recogido el frame de la pieza en el instante actual, procesado la imagen en niveles de grises y filtrado el ruido que pudiera estar presente, binarizada la imagen y segmentado la pieza aislándola completamente del fondo y extraído su contorno lo más precisamente posible, analizado el tipo de pieza de que se trata mediante el clasificador morfológico considerando como referencias válidas las de la base de datos, comparado la matriz de métricas de similitud para todas las imágenes y el frame actual construyendo una lista ordenada de los mayores parecidos según las métricas consideradas y averiguada la pieza de referencia más parecida, computado los detectores, descriptores y comparadores para todas las imágenes mediante el clasificador característico e, iterativamente, para la pieza más parecida con la pieza actual y persistida en un vector la pose de las parejas de características para todas las iteraciones permitidas, lo que resta en este punto es descomponer de cierta forma la matriz de homografía estimada, que relaciona la pose de una pieza con otra, gracias a todas las correspondencias encontradas.

Cabe comentar que se construye un búfer de resultados desde la estructura `RespuestaSistema` en el que se almacenan distintos atributos en cada iteración: un booleano que indica si la pieza está bien o mal orientada y tres dobles que indican los grados sexagesimales y las traslación en píxeles extraídos de la homografía entre matches y la orientación absoluta (horizontal) de la pieza con respecto de la cinta estimada mediante PCA. Además, para validar la pose de las piezas se emplea únicamente la orientación de estas pues es el único parámetro que se considera relevante y de interés para el problema particular; debido a que la cámara está fija en todo momento, en la misma localización, la pieza está limitada por el plano horizontal de la cinta transportadora y el análisis se realiza en la región de interés de la pieza, la estimación de su posición (equivalentemente, de su centroide) es trivial y sería simbólica estimarla para esta solución, aunque lo único interesante sería calcular la traslación relativa de la pieza actual a la de la imagen de referencia y, lo que sí se lleva a cabo, es detectar si la pieza está fuera de los límites de la imagen o, análogamente, rebasando parcialmente los límites de la cinta, lo cual deriva en una pieza inválida o mal colocada y en la respuesta correspondiente. (realmente, ni siquiera se comenzaría a ejecutar el algoritmo de procesamiento en tal caso; la salida sería nula desde el principio de esta falsa detección).

Una matriz de homografía de nueve elementos representa una transformación geométrica proyectiva bidimensional en coordenadas homogéneas; en otras palabras, es una transformación perspectiva que mapea puntos entre dos planos, los cuales relaciona de tal forma que:

$$\lambda \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \cong H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \tag{102}$$

donde $\lambda \in \mathbb{N}$ es un escalar no nulo que no afecta a la homografía poque es un objeto proyectivo y, por definición, no tiene escala (se fija el último elemento de la matriz a 1 como elección arbitraria) (por eso se ve que los dos miembros de la expresión son congruentes).

Por cada pareja de puntos de entrada se puede establecer un sistema de ecuaciones matriciales homogéneo como este:

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ \vdots \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{103}$$

y para "n" parejas se extiende a $A_{\{2n\,x\,9\}} = h_{\{9\,x\,1\}}0_{\{n\,x\,1\}}$. De las incógnitas h_{ij} solo ocho son variables independientes (porque la homografía es un objeto homogéneo) (y por eso se requiere un mínimo de cuatro parejas, pues cada una proporciona dos ecuaciones linealmente independientes si los puntos no están alineados), así que el sistema pierde un grado de libertad, lo que se entiende como que existe un conjunto de vectores "h" cuya transformada con la matriz A resulta cero tiene dimensión unidad, es decir, que existe una dimensión que siempre se transforma en el elemento nulo sin que el resto de valores sea nulo. Haciendo uno el último elemento de "h" ya se puede resolver el sistema minimizando $\|Ah - 0\|^2$ (encontrar una solución que minimice la suma de los cuadrados de los errores entre las observaciones y las predicciones del modelo), pero otra opción más estable numéricamente es descomponer A en valores singulares (descomposición SVD), siendo la solución el autovector de A^TA asociado al menos autovalor.

Para estimar la homografía se hace uso del método `cv::findHomography`, que minimiza el error de reproyección definido por:

$$\sum_i \left(x_i' - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}}\right)^2 + \left(y_i' - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}}\right)^2 \tag{104}$$

Y que permite utilizar todos los pares de puntos para la estimación inicial de la matriz mediante un esquema de mínimos cuadrados o emplear uno de tres métodos robustos de establecimiento de correspondencias que se basan en distintas cualidades de los pares de puntos para estimar la bondad de la homografía: RANSAC (número de inliers), LMeDS (error de reproyección mediano mínimo) y RHO (basado en PROSAC [55]; número de inliers ordenados por probabilidad de que sean mejores correspondencias). Luego (independientemente de si se usa o no alguna de las técnicas anteriores), la matriz de homografía se refina aún más reduciendo el error de reproyección con el método de Levenberg-Marquardt, que es esencialmente un algoritmo de Gauss-Newton para sistemas no lineales que se convierte suavemente en un problema de descenso del gradiente cuando la actualización por Gauss-Newton comienza a fallar [56, pág. 618].

En la práctica se decide emplear RANSAC con un umbral de reproyección máximo permitido de 3 píxeles, 2000 iteraciones como máximo y un nivel de confianza de 0,995, aunque deshabilitarlo no proporciona resultados perceptiblemente diferentes en la salida del sistema pues las parejas ya quedan suficientemente filtradas tras los pasos previos como para que estos métodos no influyan en su permanencia.

También se tiene en cuenta que el número de puntos mínimo para estimar la homografía es de ocho; si las parejas de entrada son menos de cuatro (altamente improbable pues el clasificador morfológico habría impedido que este código se ejecutase pues habría saltado previamente al ver que las piezas son demasiado distintas) la homografía no se computa en esa iteración, sino que se computa (si el número de parejas necesario es correspondiente), pudiendo estimar alternativamente, en su lugar, la aproximación transformación afín, rígida (`cv::estimateRigidTransform`, similar pero que se decidió no escoger porque no implementa los filtros adicionales comentados) o la rotación pura pertinente.

2.3.5.1.- Descomponer la transformación

En este último paso se construye la respuesta final del sistema que, en función de la pose relativa de la pieza (actual), que la cámara estaría viendo durante cierto lapso de tiempo, con respecto a la pieza más parecida, bien orientada, de la base de datos disponible, sirva para lanzar una respuesta totalmente transparente al exterior y lo máximo verosímil posible que valide si la pieza está bien situada o no. Cabe darse cuenta de que la estimación de la orientación de la pieza se computa en valor absoluto, es decir, la dirección no viene especificada gracias al ajuste de la homografía mediante mínimos cuadrados de las parejas de puntos, por lo que aceptar tanto 5 º como -157 º constituiría un problema y una fuente de ruido, más si el método se trata de aproximar la respuesta final por la media de todas las respuestas. Similarmente, aunque la pieza correctamente orientada se tome como la pieza con una orientación de 0 º, no es raro que la descomposición de la transformación unas veces devuelva 1 º o 2 º como 359 º o 358 º, correspondiente; por ello en el código se trata de ajustar, por una parte, los cuadrantes en donde los métodos `Procesador::degreesFromR`, `Procesador::keypointsFilter` y otros manejan y retornan la orientación y, por otra, se deshace el offset de cada medida absoluta para convertirla en una medida positiva, en las mismas unidades, relativa a la correcta orientación de la pieza (aquella que tenga la de referencia, sean 0 º), para que todo el análisis sea en referencia a “lo que dista angularmente de una pieza correctamente orientada”.

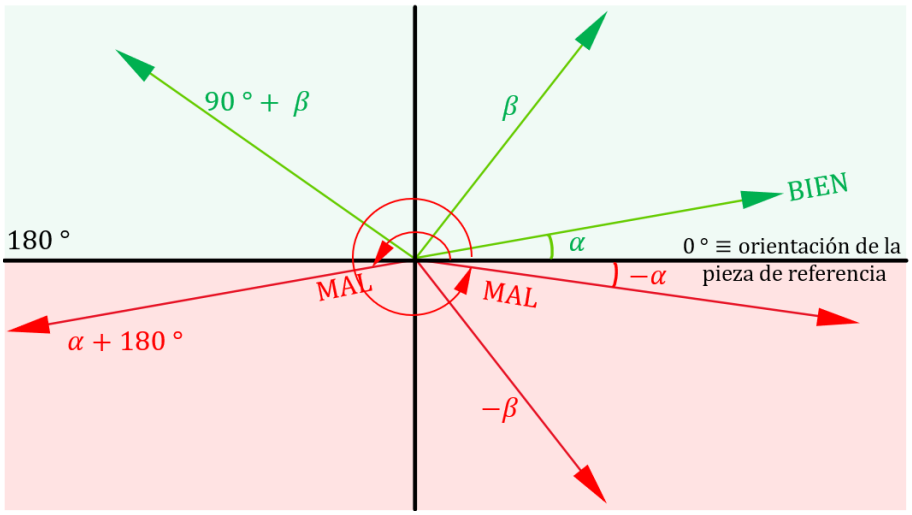


Fig. 107. Conversión de la orientación relativa a la posición angular correcta.

En el marco de este trabajo, se reconoce que aplicar una descomposición de R y t directamente a la homografía no es exacto debido a la deformación perspectiva introducida por H. Sin embargo, dado que se asume que las matrices de cámara actuales y de referencia son prácticamente equivalentes a una traslación más una pequeña rotación (idealmente), se considera aproximadamente válido extraer directamente R y t de H. Esto, y el hecho de que los puntos correspondientes de las imágenes son prácticamente coplanares, evita la necesidad de utilizar la matriz de cámara, la matriz fundamental y esencial, etc. y métodos como `cv::decomposeHomographyMat`. Además, se asume que la escala entre las imágenes es altamente similar, y cualquier discrepancia se reflejará en pasos previos del algoritmo, proporcionando una indicación previa de posibles problemas.

Ahora bien, cuando existe una situación como la de este proyecto en la que colocación de la cámara con respecto a la cinta está controlada a voluntad, se tiene la ventaja de poder establecer ciertas asunciones que simplifican inteligentemente la solución al problema sin perder la precisión necesaria para proporcionar respuestas confiables. Sabiendo que los elementos h_{31} y h_{32} de la homografía influyen en la manera de la que se incline el plano de la imagen tras la transformación, es decir, en el punto de fuga hacia el que convergen las líneas paralelas de la pieza (de ahí la apariencia de profundidad que provoca una transformación proyectiva 2D), se aprovecha el hecho de que la perspectiva debida a la lente de la cámara es despreciable y que el plano horizontal de la cinta mantiene constante los puntos de fuga de la pieza a lo largo de su recorrido por el plano imagen, con lo que la estimación de la homografía sería equivalente a la estimación de una transformación rígida y, por tanto, se puede aproximar por esta (*) debido a que los elementos proyectivos de la matriz son reducidos (los puntos de fuga son lejanos al origen y las líneas paralelas convergen con más lentitud) y la escala entre las imágenes es aproximadamente igual, pues las piezas se analizan recortando el cuadro delimitador que las aísla y se reescala su alto y ancho aun tamaño equivalente (como una transformación de similitud se convierte en una transformación rígida o euclídea al tener un escalado unitario):

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \approx^{*} \widehat{H} = \begin{bmatrix} \cos\alpha & -\sin\alpha & t_x \\ \sin\alpha & \cos\alpha & t_y \\ 0 & 0 & h_{22} \end{bmatrix} \cong \begin{bmatrix} R_{\{2x2\}} & t_{\{2x1\}} \\ \mathbf{0}_{\{1x2\}} & 1 \end{bmatrix} \quad (105)$$

Esta aproximación (alternativamente se podrían explorar directamente otras estimaciones aproximadas, como una transformación rígida o afinidad) conserva de manera suficiente los términos de rotación y traslación que interesan en este trabajo y brinda la flexibilidad de situar y tratar la cámara de manera personalizada y deliberada en la cinta transportadora.

Para extraer las componentes de rotación y traslación a partir de $\widehat{H}$ se normaliza la escala dividiendo toda la matriz por el elemento h_{22}; la orientación se puede calcular como el arcoseno o arcocoseno de una de las componentes de $R_{\{2x2\}}$ y la traslación se calcula como la norma euclídea de $t_{\{2x1\}}$. Para formar la respuesta final del sistema se descompone tanto la matriz de homografía resultante del análisis mediante keypoints, que a priori parece la que otorgará resultados más fiables y robustos, como la resultante del ECC y se combinan sus salidas.

Para conformar la salida definitiva del sistema se espera a que, por orden externa o porque haya comenzado a salir por la derecha del campo de visión de la cámara, se finalice el seguimiento de la pieza. Primero se emplea la estructura `RespuestaSistema`, que contiene seis vectores que indican la orientación y posición de la pieza en la cinta transportadora; son tres tuplas correspondientes a los tres métodos considerados más relevantes para este respecto: ECC, PCA y la homografía. Los campos incluyen si la pieza está bien orientada (`BM`), el ángulo de rotación (`rotation`), la traslación en el eje Y (`translationY`), y la orientación absoluta de la pieza con respecto a la cinta (`orientationPCA`); todo ello es suficiente para definir precisamente la pose absoluta y relativa con respecto de la cinta transportadora con la información de que se dispone y, más importante, si la pieza está bien o mal situada. El booleano que valida dicha situación es, en efecto, la respuesta final del sistema. Previamente quedarían almacenadas todas las respuestas del sistema en cada iteración que se haya dado (en el transcurso del camino de la pieza a lo largo de la cinta) en un vector denominado `mBufferRespuestas`, que se utiliza como un búfer para conformar la respuesta final calculando la moda y la media de los valores almacenados. Se suman los valores de rotación, traslación y orientación para calcular sus medias y, luego, se decide la respuesta final basándose en la moda de las respuestas de validación guardadas en `BM`. Así, los resultados finales se utilizan para actualizar la interfaz de usuario (`mGateawayUiProceso`) emitiendo varias señales (`sg_sendResultadoCC` y `sg_updateGUI`), además de que se actualizan las matrices promediadas para mostrarlas en la GUI, con fines ilustrativos.

2.3.6.- GUI

La clase `NexoUIProceso` actúa como un puente de enlace entre el hilo de principal (en el que se realizan las tareas gráficas y visuales y se interactúa con el usuario) y las clases de procesamiento (las dedicadas al algoritmo en sí). De esta manera todos los mensajes entre ambas partes quedan encapsuladas y organizadas de una manera homogénea y limpia. A través de ella se intercomunica el hilo (principal) de la GUI con los hilos de procesamiento paran obtener y escribir de manera transparente todos los parámetros de proceso configurables desde la interfaz gráfica y, en general, cualquier atributo que mantenga una relación entre cualquier hilo de la aplicación y el hilo principal dedicado a la interacción con el usuario.

En todo momento se pueden ajustar parámetros y umbrales en las entidades de clasificación para mejorar la sensibilidad y especificidad del sistema, además de activar y desactivar cada sub-bloque incluso en tiempo de ejecución para adaptarse a cada situación en fase experimental.

También se hace uso de QCustomPlot para generar y mostrar gráficas 1D, imágenes y otros elementos permitiendo una interacción básica con estos. El constructor de la clase, que deriva de `QObject` y sería afín a un hilo secundario que recibiría los datos a procesador mediante señales y ranuras, inicializa un puntero a un objeto `QCustomPlot`, la función `muestraGrafica1DQCustomPlot` crea una nueva instancia de dicha clase, configura un gráfico con datos proporcionados, muestra la ventana de la gráfica, guarda la gráfica como imagen PNG y realiza un breve retardo para garantizar que la ventana tenga tiempo de aparecer antes de guardar la imagen y la función `muestraImagenOpenCV` convierte una imagen de OpenCV a un formato compatible con `QImage`, la muestra en un `QLabel`, guarda la imagen como un archivo PNG y agrega un retardo similar permitiendo que la imagen se muestre antes de ser guardada.

También se usa dicha librería para realizar gráficos de tendencia en tiempo real; por ejemplo, para visualizar la rotación de las piezas con respecto a las referencias extraída de la homografía en cada iteración del clasificador morfológico. Basta con que desde `Procesador` se emita la señal `sg_updateRotationH` para que la reciba el hilo que ejecuta las funciones de la librería asíncronamente para que se actualice la gráfica debidamente (en el slot `Grafica1DQCustomPlot::updateRotationH`).

En la siguiente figura se puede ver una vista general de la interfaz creada para este proyecto, la cual se lanza desde la barra de herramientas superior de la GUI de Camera2DSensor o automáticamente cuando se pulsa el botón de “Start” de dicha interfaz. En definitiva, ha servido como un recurso harto valioso para la búsqueda de una óptima parametrización de todas las partes del algoritmo. Cabe comentar que se habilita el flag o atributo `Qt::WA_QuitOnClose` mediante `setAttribute` en la ventana principal de la interfaz para hacer que la aplicación finalice si el usuario la cierra.

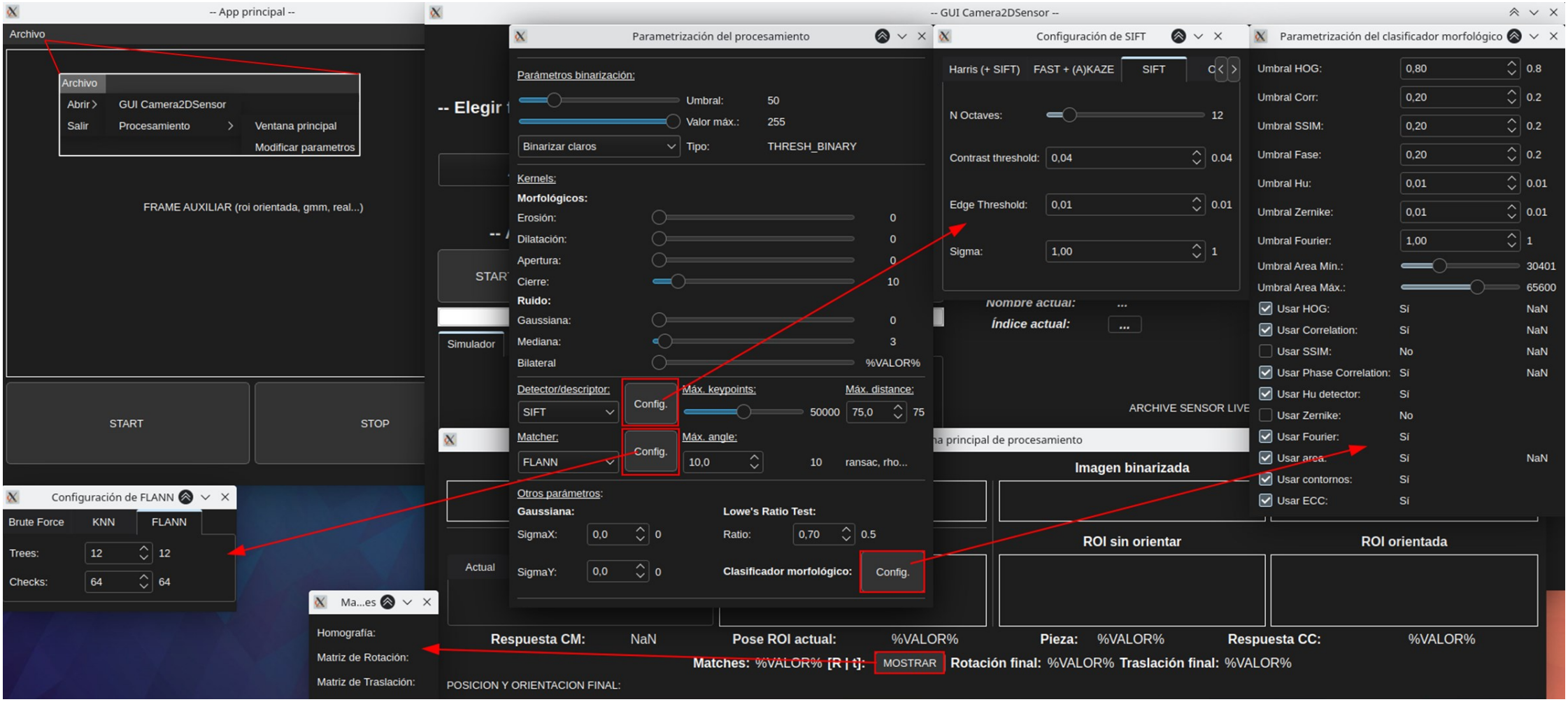
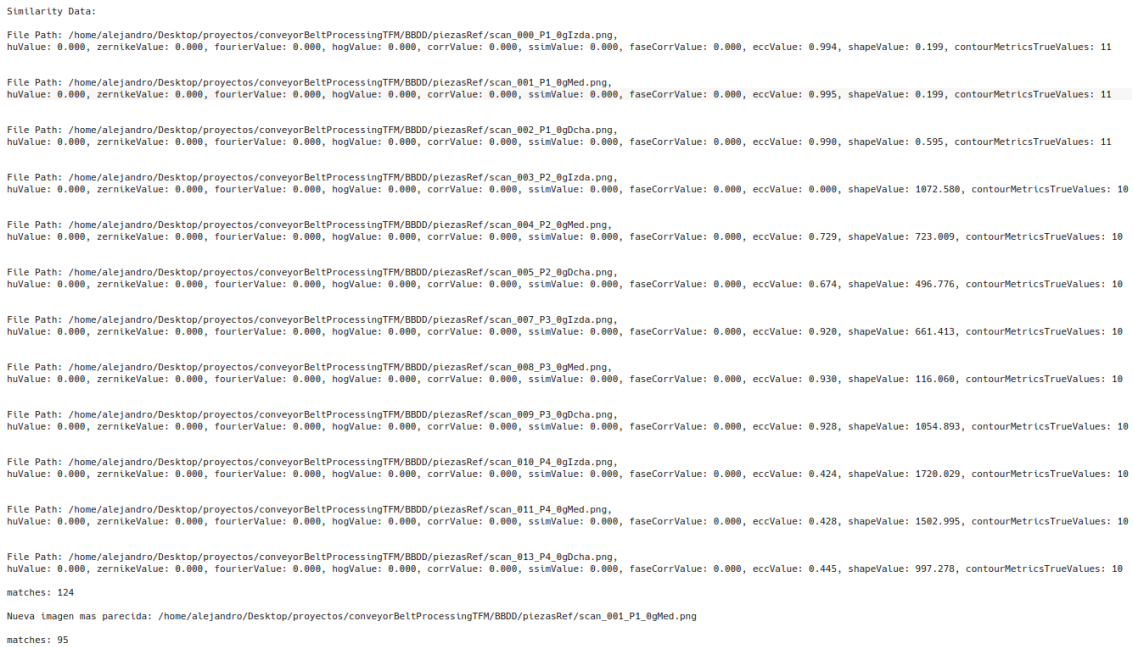


Fig. 108. Vista general de todas las ventanas de la interfaz creada para el proyecto.

La GUI de la aplicación desempeña un papel crucial para interactuar con las diferentes funcionalidades del sistema en fase experimental pero un papel meramente ilustrativo una vez parametrizados los algoritmos de la forma más "fetén" y fijados (y bloqueados) los valores de todas las variables. Todo interconectado por medio del puente de enlace `NexoUIProceso`, la clase `GUICamera2DSensor` representa, como ya se ha comentado, la ventana que se utiliza para comunicarse con la clase `Core`, responsable de gestionar el sistema de captura de imágenes desde una cámara 2D y mostrar la interfaz del sensor homónimo (permite al usuario iniciar y detener el procesamiento de imágenes, así como realizar varias configuraciones relacionadas con la cámara), la clase `FormParametrosProcesamiento` proporciona los controles para configurar los numerosos parámetros relacionados con el procesamiento de las piezas (como detectores, descriptores y algoritmos de los clasificadores) y otras clases adicionales como `FormConfigDetectorsDescriptors`, `FormConfigMatchers`, y `FormConfigClasificadorMorfologico` definen configuraciones específicas para la personalización de cada uno.

A través de la función global o libre `imprimeEnLog` se imprimen muchos resultados y mensajes de depuración e información hacia un archivo de texto plano denominado "LOG.txt", que sirve de gran ayuda para analizar el desempeño del sistema durante todos los experimentos y para encontrar los mejores parámetros que caracterizan a los algoritmos empleados.



```
Similarity Data:

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_000_P1_0gIzda.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.994, shapeValue: 0.199, contourMetricsTrueValues: 11

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_001_P1_0gMed.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.995, shapeValue: 0.199, contourMetricsTrueValues: 11

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_002_P1_0gDcha.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.990, shapeValue: 0.595, contourMetricsTrueValues: 11

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_003_P2_0gIzda.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.000, shapeValue: 1072.580, contourMetricsTrueValues: 10

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_004_P2_0gMed.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.729, shapeValue: 723.009, contourMetricsTrueValues: 10

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_005_P2_0gDcha.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.674, shapeValue: 496.776, contourMetricsTrueValues: 10

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_007_P3_0gIzda.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.920, shapeValue: 661.413, contourMetricsTrueValues: 10

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_008_P3_0gMed.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.930, shapeValue: 116.060, contourMetricsTrueValues: 10

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_009_P3_0gDcha.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.928, shapeValue: 1054.893, contourMetricsTrueValues: 10

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_010_P4_0gIzda.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.424, shapeValue: 1720.029, contourMetricsTrueValues: 10

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_011_P4_0gMed.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.428, shapeValue: 1502.995, contourMetricsTrueValues: 10

File Path: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_013_P4_0gDcha.png,
huValue: 0.000, zernikeValue: 0.000, fourierValue: 0.000, hogValue: 0.000, corrValue: 0.000, ssimValue: 0.000, faseCorrValue: 0.000, eccValue: 0.445, shapeValue: 997.278, contourMetricsTrueValues: 10

matches: 124

Nueva imagen mas parecida: /home/alejandro/Desktop/proyectos/conveyorBeltProcessingTFM/BBDD/piezasRef/scan_001_P1_0gMed.png

matches: 95
```

Fig. 109. Extracto ilustrativo del tipo de información de depuración del archivo de registro del sistema. En esta iteración se ve que "shapeValue" proporcionaba resultados buenos (un valor muy bajo para la primera pieza que era la actual en ese momento), con un contorno bien extraído, para localizar la pieza correcta.

2.3.7.- Base de datos

Para mejorar la eficiencia computacional del sistema se propone generar una pequeña base de datos o información para persistir variables e imágenes cuyo cálculo sea significativamente costoso en comparación a otras presentes en el código fuente y que sean de interés cargar el inicio del programa, en tiempo de ejecución, desde cualquier ruta del sistema de archivos local.

Tras analizar cuáles son los puntos del algoritmo en los que se dedica más tiempo a realizar cálculos repetitivos se decide elaborar un script de C++ para poder calcular de antemano ciertas variables y métricas para las imágenes de referencia y así ahorrarse su cálculo durante el funcionamiento en línea del sistema. Las variables que se calculan y persisten de antemano son: la matriz de imagen de cada pieza rotada según su orientación principal, la matriz de imagen de cada región de interés completamente aislada en un fondo binario a cero, los vectores que contienen todos los momentos de Hu, Zernike y Fourier, la señal unidimensional normal y normalizada del contorno de cada pieza, los histogramas de gradientes orientados (algunos cuyo tamaño supera los ocho millones de elementos de tipo `float`) y el vector de puntos que representa el contorno de cada referencia segmentada.

Al iniciar la aplicación se lee el conjunto de archivos CSV y las imágenes que contienen estas variables y se cargan en la memoria dinámica dentro del ámbito privado de la clase Procesador, para que puedan usarse durante todo el ciclo de vida de la aplicación, pues son variables que no cambian entre iteraciones. Se ha observado una triplicación en la velocidad de la aplicación con estos cambios (del orden de segundos), en lo que se refiere al análisis completo de una pieza en su transcurso por la cinta desde que entra hasta que sale del campo de visión de la cámara.

Las funciones `Procesador::cargarDatosDouble`, `cargarDatosFloat` y `cargarDatosCVPoint` realizan la lectura de datos desde un archivo CSV que contiene valores de tipo `correspondiente`. Cada fila del archivo representa un vector, estando el nombre de cada vector almacenado en la primera columna. Las funciones devuelven un vector de pares `std::vector<std::pair<QString, std::vector<typename T>>>`, donde cada par está compuesto por el nombre del vector y un vector de valores de tipo correspondiente. Por otro lado, las funciones `obtenerVectorDesdeDatosDouble`, `obtenerVectorDesdeDatosFloat` y `obtenerVectorDesdeDatosCVPoint` extraen un vector específico de la estructura de mapa previamente cargada en memoria que asocia nombres de vectores con vectores de pares de tipo (nombre, vector); dada una clave y subclave proporcionadas, devuelven un vector de valores del tipo correspondiente. Finalmente, las funciones `guardarDatosDouble`, `guardarDatosFloat` y `guardarDatosCVPoint` toman un vector de tipo `const std::vector<typename T>` y un nombre de archivo como entrada, convierten cada elemento en un formato de cadena de caracteres (`std::string`) y los agrega a una lista para luego, ayudándose de la biblioteca QtCSV [57], escribir los datos en un archivo CSV.

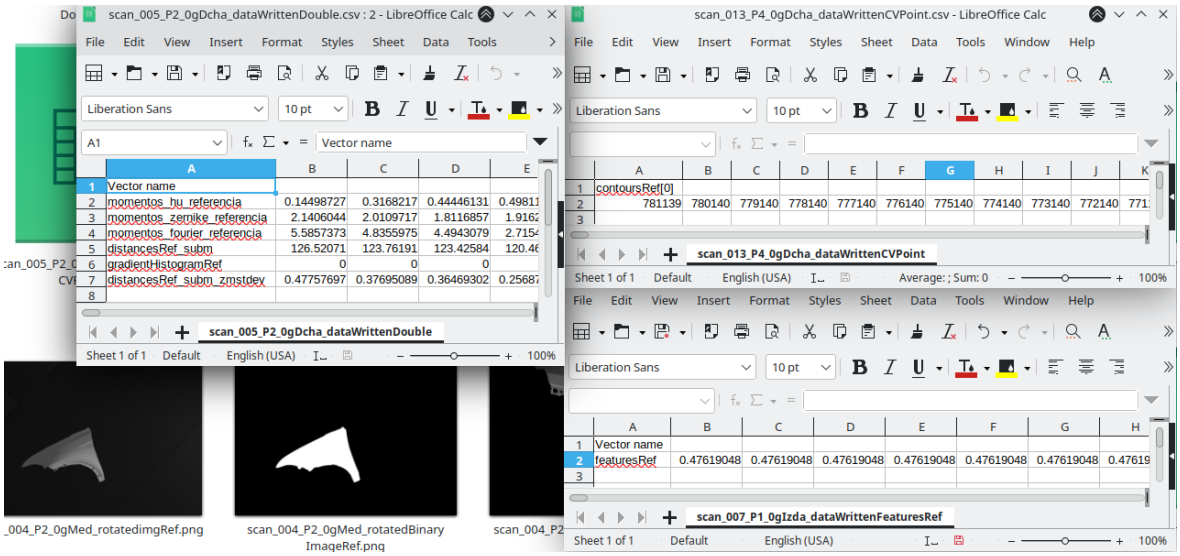


Fig. 110. Tres ejemplos de archivos CSV contenidos en la base de datos que se carga al inicio de la aplicación.

El guardado se puede ejecutar mediante un único script de C++ llamado "saveDataToBBDD.cpp", proporcionando en la ruta indicada las imágenes de referencia cuyas características se quieren extraer. De esta forma los métodos sirven para acceder mediante referencia directa a las variables relativas a las piezas de interés implícitamente, simplificando la cantidad de código a escribir y pudiendo integrarlo en el bucle de procesamiento de los clasificadores morfológico y característico realizados, facilitando la persistencia de información y posterior recuperación de datos en la aplicación.

3.- EXPERIMENTOS Y RESULTADOS

En este apartado se plantean los experimentos y ensayos a realizar para poner a prueba el sistema desarrollado. La evaluación del sistema se realizará esencialmente comparando mediante métricas experimentales y estadísticas su respuesta con las piezas disponibles en la base de datos de ensayos, que contiene imágenes de diferentes piezas bien y mal orientadas etiquetadas correspondientemente, junto con el valor de su pose codificado en ellas mismas. Para analizar el desempeño del sistema está, por una parte, la idea de verlo como una caja negra que ante la vista de una pieza en la línea de producción decide binariamente si está bien o mal situada; esta idea se mantiene en la primera fase de los experimentos y compete a la capacidad del sistema para identificar las piezas, en general y, en particular, identificar las piezas correctas. Por otra parte, una segunda manera de verlo, de "grano más fino", se trata de analizar los cálculos y las aproximaciones que el algoritmo lleva a cabo internamente, cuyo desglose es interesante realizar pues puede hacer saltar indicios de algoritmos que no están funcionando como debieran u otras cuestiones; este examen numérico compete a la comparativa de la capacidad del sistema para estimar precisamente la pose en sus tres posibles salidas, que son las suministradas por el algoritmo PCA, ECC y la homografía. Aunque la salida digital final del sistema sea correcta, las decisiones que le llevan a tomarla pueden estar influenciadas por valores poco deterministas, desequilibrando una balanza ambigua de precisión; es mandatorio comprobar si es este o no el caso para probar la validez del algoritmo.

En la presente investigación, se aborda exhaustivamente el análisis de resultados obtenidos en el estudio, centrándose eminentemente en la importancia de la evaluación de la capacidad clasificatoria del sistema. Si bien es innegable la relevancia de su capacidad para realizar clasificaciones precisas, se propone explorar más a fondo su funcionamiento, examinando las estimaciones numéricas que subyacen a sus respuestas digitales (análisis estadístico de regresión). Inicialmente, se plantean diversos escenarios mediante la ejecución de experimentos que abordan condiciones variables, tales como distintas situaciones de luz artificial, variaciones en la distancia a las piezas o cambios en la perspectiva de la cámara. Estos experimentos buscan poner a prueba la robustez del sistema ante diferentes situaciones. Posteriormente, se procede con el análisis de clasificación, elaborando matrices de confusión tanto para casos simulados como reales en cada experimento y evaluando métricas como la tasa de falsos y verdaderos positivos y negativos, la especificidad, el F1-Score… para obtener una visión integral del rendimiento del sistema en su interés primordial, que es el de clasificación. Finalmente, se lleva a cabo un análisis cuantitativo, donde se examinan las orientaciones "reales" y "estimadas" (dicho de manera general) en todos los experimentos, calculando métricas como desviación estándar, error cuadrático medio, NMSE y MAE… consolidando finalmente todos estos resultados en pos de proporcionar una visión holística del error del sistema desde diversas perspectivas, permitiendo extraer conclusiones fundamentadas sobre su desempeño en la discusión subsiguiente.

Definitivamente, los resultados del sistema delimitarán su capacidad para ser validado y empleado en una instalación real en línea. De las pruebas realizadas se deriva una serie de métricas de alto interés para discutir a posteriori las competencias del sistema. La forma elegida de representar los resultados de las pruebas es la matriz de confusión, tan frecuentemente utilizada en problemas de clasificación en el campo del aprendizaje automático pero que sirve de igual manera en este problema. Esta matriz se puede analizar para comprobar la precisión y sensibilidad del sistema ante las distintas casuísticas que se le planteen de manera directa e ilustrativa.

La matriz de confusión dispone de dos dimensiones, una dedicada a plasmar los resultados predichos por el sistema y otra para representar el valor real o la etiqueta de la muestra actual. Los valores que se disponen en esta son cualitativos y se corresponden a:

- Verdadero positivo (TP): Se corresponde con una pieza que en la realidad está bien orientada y el sistema ha clasificado como tal.
- Falso positivo (FP): Se corresponde con una pieza que en la realidad está mal orientada y el sistema ha clasificado como bien orientada.
- Verdadero negativo (TN): Se corresponde con una pieza que en la realidad está mal orientada y el sistema ha clasificado como tal.
- Falso negativo (FN): Se corresponde con una pieza que en la realidad está bien orientada y el sistema ha clasificado como mal orientada.

Tabla 1. Ejemplo de matriz de confusión.

		Predicción	
		Positivo	Negativo
Actual	Positivo	Verdadero positivo (TP)	Falso negativo (FN)
	Negativo	Falso positivo (FP)	Verdadero negativo (TN)

Las métricas que se extrapolan de la matriz de confusión, que compactan numéricamente el rendimiento del sistema, son las siguientes:

- Exactitud o *accuracy* (ACC): Indica el porcentaje de muestras bien clasificadas con respecto al total de muestras de la prueba. Hace referencia a lo cerca que esta el resultado del valor verdadero; es la cantidad de predicciones positivas que fueron correctas. Si hay un desequilibro notable entre la cantidad de muestras de una clase y otra esta métrica puede ser ciertamente engañosa, pero se cuida de disponer de un conjunto de datos de validación suficientemente equilibrado como para que esta métrica proporcione resultados verosímiles:

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + FP + TN + FN} \tag{106}$$

- Sensibilidad o *recall* (TPR): Es la tasa de verdaderos positivos que han sido correctamente identificados por el algoritmo; es una medida de la exhaustividad del modelo en este sentido:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{107}$$

- Especificidad o *selectivity* (TNR): Es la tasa de verdaderos negativos que han sido correctamente identificados por el algoritmo; mide la misma cualidad que el anterior parámetro pero en sentido opuesto:

$$TPN = \frac{TN}{N} = \frac{TN}{TN + FP} \tag{108}$$

- Precisión o valores predictivos positivos (PPV): Hace referencia a la dispersión del modelo sobre sus positivos; se calcula realizando el ratio entre los verdaderos positivos y los resultados de todos los positivos:

$$PPV = \frac{TP}{TP + FP} = \frac{TP}{TP + FP} \tag{109}$$

- Valores predictivos negativos (NPV): Hace referencia a la dispersión del modelo sobre sus negativos; se calcula realizando el ratio entre los verdaderos negativos y los resultados de todos los negativos:

$$NPV = \frac{TN}{TN + FN} = \frac{TN}{TN + FN} \tag{110}$$

- F1-Score: Es la media armónica entre la precisión y la sensibilidad; dicho de otra forma, es igual al cociente entre el doble del valor de la precisión en un producto con la sensibilidad y la suma de esos dos parámetros. Proporciona un equilibrio entre la capacidad del sistema para minimizar falsos positivos y falsos negativos, similar a métricas como el coeficiente de Sørensen-Dice:

$$F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{111}$$

  En función del valor de esta métrica se puede dar una de cuatro posibles situaciones en el sistema:

Tabla 2. Interpretación de las capacidades del sistema en función del F1-Score.

	Sensibilidad	
Precisión	Baja	Alta
Baja	Valor bajo, cercano a 0; el modelo presenta dificultades para clasificar correctamente las clases, siendo poco eficaz en su desempeño global	El valor puede ser bajo debido a una baja precisión, pero no necesariamente; el modelo identifica bien la clase, pero su detección incluye también muestras de otra clase, lo que hace que no sea completamente confiable
Alta	El valor puede ser alto, pero no necesariamente; aunque el modelo no detecta de manera óptima, cuando lo hace, la detección es confiable y precisa	Valor alto, cercano a 1; el modelo opera de manera eficaz, logrando un equilibrio óptimo entre precisión y sensibilidad

- Coeficiente de correlación de Matthews (MCC o coeficiente φ): Es el producto cruzado normalizado o ponderado entre los resultados verdaderos y falsos que tiene en cuenta su relación y combinación. Proporciona una medida de la calidad general de la clasificación, teniendo en cuenta los cuatro elementos de la matriz de confusión. Sus valores oscilan entre -1 y 1, donde 1 indica una predicción perfecta, 0 representa una predicción al azar, y -1 indica una discordancia total entre las predicciones del modelo y las observaciones reales. Es especialmente útil cuando existen desequilibrios en las clases:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \tag{112}$$

El primer experimento (Exp. I) involucra todas la piezas (puerta, “rueda”, capó trasero y lateral [para las pruebas en la realidad se usa otro conjunto de cuarto piezas, de otras zonas pero también de geometría compleja]) con una pose correcta (bien orientadas), una iluminación de referencia sin brillos, con la cámara en la misma posición en la que son tomadas las fotografías de referencia y, en general, en unas condiciones de funcionamiento normales o nominales. El segundo experimento (Exp. II) involucra todas las piezas en las mismas condiciones de iluminación que en el caso anterior pero mínima y arbitrariamente rotadas y trasladadas en varias configuraciones, dentro de la cinta, manteniéndose en unos rangos admisibles de orientación válida (aún bien orientadas, dentro de los límites). El tercer experimento (Exp. III) implica deformar la pose de todas estas piezas en un grado suficientemente alterado para que deba ser señalada como incorrecta (mal orientadas), en todos los casos; además se varía el tono, la intensidad y posición de la iluminación ínfimamente para comenzar a probar la robustez del sistema (condiciones levemente adversas). Algo parecido cabe decir del cuarto experimento (Exp. IV), en el que las piezas pueden estar bien o mal orientadas pero se varían sobremanera, y en tiempo real, los factores del entorno (condiciones gravemente adversas) y también se varía la orientación de algunos objetos en el espacio 3D (piezas que no han de reconocerse o, al menos, identificarse como inválidas), lo que igualmente debe hacer “saltar la alarma” del sistema.

Finalmente, el quinto y último experimento (Exp. V) involucra, en unas condiciones de entorno débilmente favorables (condiciones adversas), una variedad de objetos (se integran modelos desconocidos a la escena para comprobar si el sistema los identifica como inoportunos) y piezas bien y mal orientadas en todo tipo de localizaciones dentro y fuera de la cinta.

Las pruebas realizadas en el sistema real son homónimas a las anteriores pero se dividen en cuatro partes y se realizan en un entorno menos laxo o más controlado, emulando en el "banco de ensayos" la situación que se daría en una planta industrial real. De nuevo, se cuentan con cuatro piezas, diferentes a las anteriores; en el primer experimento las piezas están bien orientadas, en el segundo están ligeramente trasladadas y orientadas dentro de unos márgenes admisibles, en el tercero se salen de estos márgenes disponiéndose en poses incorrectas y en el último se giran en el espacio 3D para que se deban ver como objetos "intrusos".

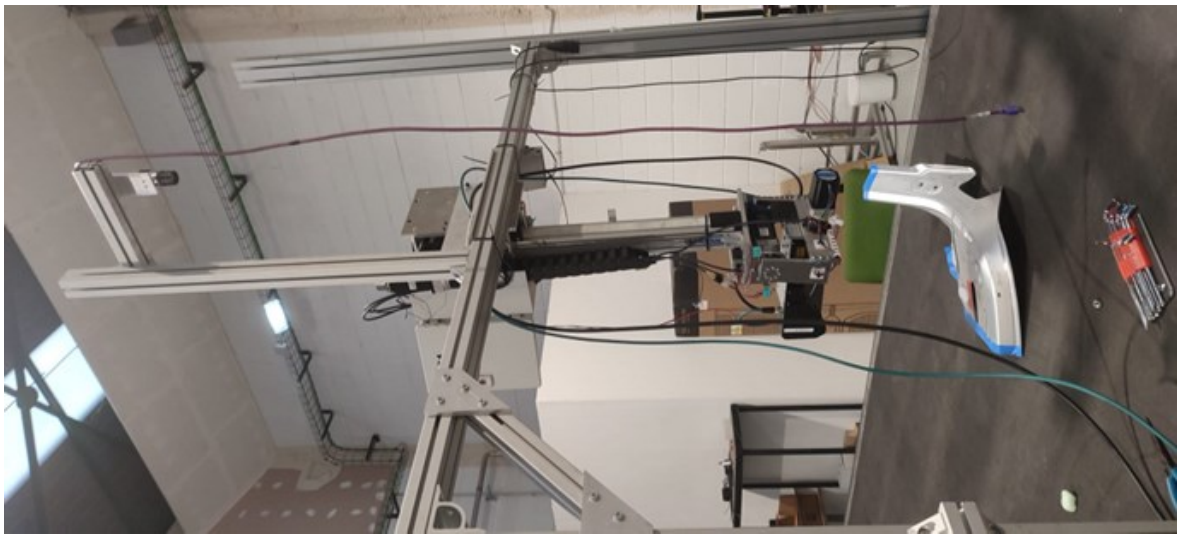


Fig. 111. Vista en alzado de parte de la cinta transportadora y el montaje de que se dispuso para realizar las pruebas en la realidad; arriba (o a la izquierda, en la figura), la cámara 2D apuntando cenitalmente a las piezas.

Como el objetivo de este proyecto no es estudiar la diferencia de rendimiento de los distintos tipos de algoritmos, detectores, descriptores… implementados sino utilizar un algoritmo definido con el que demostrar la validez del mismo, se van a comentar a continuación somera y directamente los parámetros y las técnicas escogidas, tras un debido tiempo de ensayo en segundo plano y estudio del desempeño de tales, para llevar a cabo con ellos los experimentos que validan el presente sistema. El sistema se ha diseñado para que facilite la explotación de los mejores algoritmos y parámetros para resolver el problema; así, en cuanto a las técnicas para la extracción de características de las piezas se han decidido utilizar para toda la fase de experimentación el detector SIFT, el descriptor ORB y el matcher FLANN con una parametrización similar a la que se mencionó anteriormente en cada uno de sus apartados, por varias razones que se comentan más adelante en el apartado "Discusión". En todas las pruebas se han mantenido constantes, a fin de establecer un marco común de comparación para todos los experimentos, aunque puede que modificando precisamente o cambiando el papel de alguno de estos se pudieran

obtener mejores resultados en ciertas situaciones. Cabe decir que en cada experimento se toman como referencia imágenes distintas con respecto al resto de experimentos de las piezas en una pose correcta (e igual a 0 º), pues se varía demasiado la posición de la cámara, su perspectiva y el tamaño de los objetos entre unos experimentos y otros como para utilizar siempre las mismas referencias y esperar un resultado razonable; sin embargo, este es un supuesto razonable dado que la cámara quedaría absolutamente estática en un escenario normal, además de que se agrega cierto factor de ruido en cada experimento y sí que se modifica someramente la posición de la cámara y su distancia focal para hacer las pruebas más austeras y evaluar el sistema de manera más robusta. Es importante señalar que no se usan como referencia fotogramas idénticos a los de las grabaciones que se procesan, pues esto sería similar a usar para la evaluación de una red neuronal algunos de sus datos de entrenamiento; por el contrario, se utilizan imágenes desde un punto de vista, con una iluminación, tamaño y posición de las piezas ligeramente distintos, dentro de un rango lo suficientemente admisible como para que no diverjan las respuestas del sistema por culpa de entradas inadecuadas.

Antes de presentar los resultados de las pruebas, se muestran dos tablas que contienen las piezas que, siendo de un tipo en concreto, fueron reconocidas por el sistema como ese tipo en particular. Los casos en que el sistema ha errado identificando el tipo de pieza y, aun así, se ha continuado con el análisis de su pose, se han incluido todos en los falsos positivos del experimento correspondiente; la situación contraria en que el sistema acierta es en la que identifica una pieza correctamente con la referencia adecuada, lo que es un verdadero positivo a priori y, solo si fuera válida la pose de ambas (real y estimada), también a posteriori, donde en el caso opuesto también se consideraría, finalmente, falso positivo o negativo, dependiendo de la entrada. Para aclarar: las piezas realmente válidas que son reconocidas correctamente se identifican como piezas válidas correctas, y suman valor al sistema, las realmente válidas e incorrectamente reconocidas se identifican como objetos aberrantes (desconocidos; el algoritmo cesa), lo que resta valor al sistema; las realmente inválidas y correctas se identifican como tales objetos desconocidos (resultado positivo para el rendimiento del sistema) y las realmente inválidas e incorrectas se identifican erróneamente como piezas válidas (segundo resultado negativo). Se muestra el número de piezas válidas (únicas entradas permitidas del sistema) e inválidas (objetos desconocidos o piezas rotadas en 3D) reconocidas correcta e incorrectamente por el sistema como piezas válidas, es decir, los resultados de clasificación en las pruebas simuladas y reales.

Tabla 3. Pruebas de clasificación en general, en simulación.

Piezas (total: 32)	válidas (total: 18)	correctas	16
		incorrectas	2
	inválidas (total: 14)	correctas	9
		incorrectas	5

Tabla 4. Pruebas de clasificación en general, en la realidad.

Piezas (total: 16)	válidas (total: 8)	correctas	7
		incorrectas	1
	inválidas (total: 8)	correctas	7
		incorrectas	1

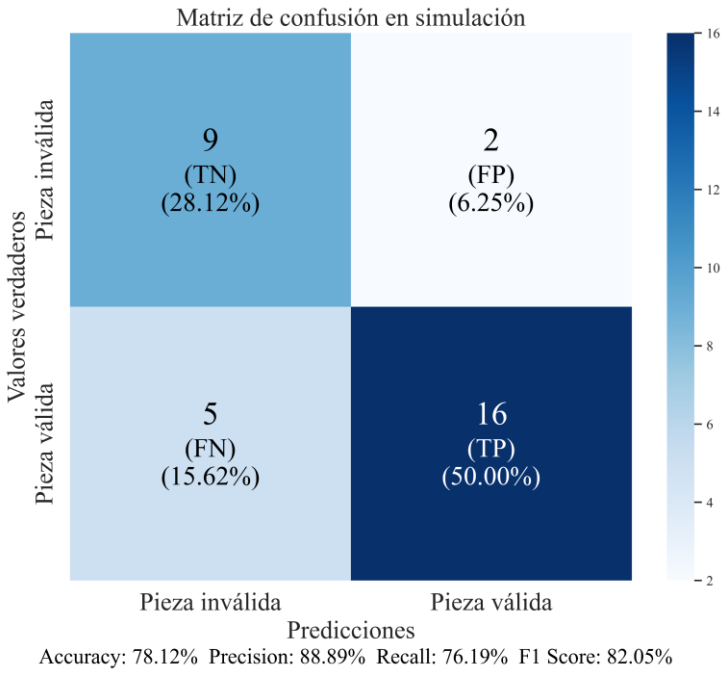


Fig. 112. Matriz de confusión que muestra los resultados finales de las pruebas de clasificación en simulación.

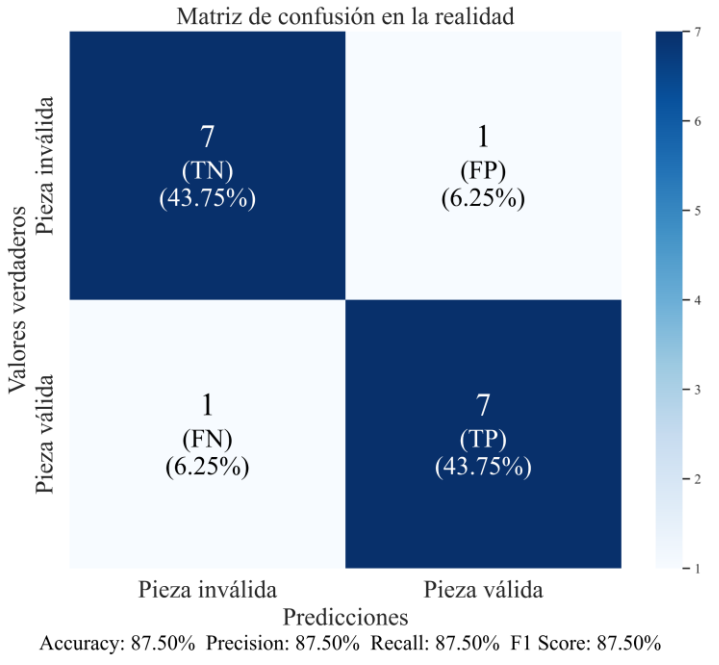


Fig. 113 Matriz de confusión que muestra los resultados finales de las pruebas de clasificación en la realidad.

Tabla 5 Métricas de rendimiento de los ensayos realizados en simulación.

Simulación	Exp. I	Exp. II	Exp. III	Exp. IV	Exp. V	Promedio
TP	7	4	0	2	3	-
FP	0	0	2	0	0	-
TN	0	0	6	2	1	-
FN	1	2	0	0	2	-
ACC	0,875	0,667	0,750	1,000	0,667	0,792
TPR	0,875	0,667	∄ TPR	1,000	0,600	0,785
TNR	∄ TNR	∄ TNR	0,750	1,000	1,000	0,917
PPV	1,000	1,000	0,000	1,000	1,000	0,800
NPV	0,000	0,000	1,000	1,000	0,333	0,467
F1-Score	0,933	0,800	∄ NPV	1,000	0,750	0,871
MCC	∄ MCC	∄ MCC	∄ MCC	1,000	0,447	0,724

Tabla 6 Métricas de rendimiento de los ensayos realizados en la realidad.

Realidad	Exp. I	Exp. II	Exp. III	Exp. IV	Promedio
TP	4	3	0	0	-
FP	0	0	1	0	-
TN	0	0	3	4	-
FN	0	1	0	0	-
ACC	1,000	0,750	0,750	1,000	0,738
TPR	1,000	0,750	∄ TPR	∄ TPR	0,802
TNR	∄ TNR	∄ TNR	0,750	1,000	0,583
PPV	1,000	1,000	0,000	∄ PPV	0,667
NPV	∄ TPR	0,000	1,000	1,000	1,000
F1-Score	1,000	0,857	∄ NPV	∄ NPV	0,826
MCC	∄ MCC	∄ MCC	∄ MCC	∄ MCC	∄ MCC

En la siguiente figura se comparan gráficamente los resultados de las métricas de clasificación obtenidas en simulación y realidad. Se observa una similitud remarcable de precisión general en ambos escenarios; las diferencias se deben mayoritariamente a una falta de población:

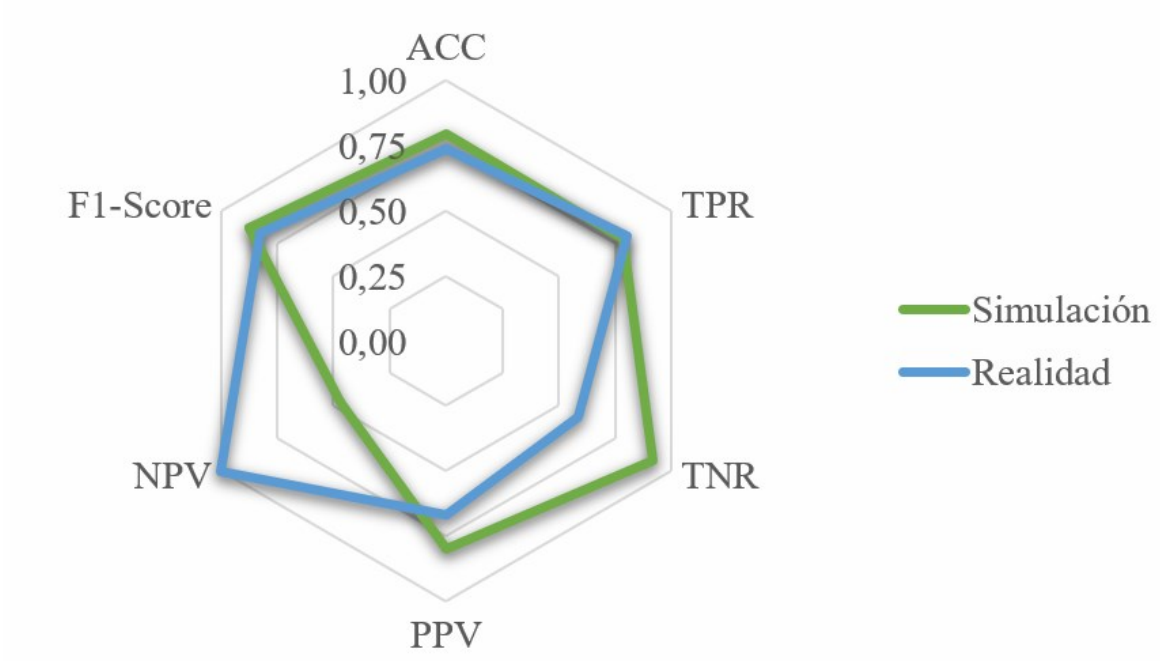


Fig. 114. Comparación de las métricas de clasificación obtenidas en simulación y realidad.

Finalmente, se proporcionan varias medidas de la validez del sistema analizando cuantitativamente los valores de orientación que proporciona (la pose es mucho menos interesante para el enunciado del problema) y los valores reales conocidos de antemano al etiquetas las piezas. Las medidas escogidas para evaluar la calidad de las estimaciones son el error cuadrático medio, que calcula el promedio de las diferencias al cuadrado entre las estimaciones y los valores reales (por lo que penaliza de manera más significativa las diferencias grandes) y la desviación estándar, que mide la dispersión de un conjunto de datos y ayuda entender cuán dispersas son las estimaciones de orientación alrededor de su valor real (un valor bajo de desviación estándar indica que las estimaciones están relativamente cerca del valor real si es que el valor medio estimado lo está [por esto la desviación estándar es más bien un indicativo indirecto del rendimiento del sistema], mientras que un valor alto indica una mayor dispersión). Más precisamente, la medida utilizada para obtener un indicativo de la dispersión de los distintos tipos de estimaciones entre sí es el coeficiente de variación (CV), que es la desviación estándar dividida por la media y se expresa como un valor porcentual adimensional.

Cabe decir que todas las piezas que son puestas a prueba en las tablas de resultados cuantitativos tienen orientación correcta, pues cuando el sistema las detecta como válidas es cuando realmente devuelve una estimación de su orientación, aunque también se podría rastrear artificiosamente, pero alteraría el sistema y se quiere mantener su código fuente inalterado durante todo el desarrollo de los experimentos. Las pruebas de clasificación ya se "ocupan" de contener los resultados discretos del sistema en cuanto a piezas mal orientadas e incógnitas, que como ya se remarcó son los resultados más interesantes en este problema. Además, como cada fila de las tablas se corresponde con una prueba (una medición o respuesta del sistema), se nota que no se cuentan con las suficientes muestras

como para describir y evaluar una distribución de error razonable (calcular la diferencia entre los resultados de todos los métodos y aproximarla mediante una distribución exponencial bilateral, una gaussiana si hay muchas estimaciones…), por lo que se considera suficiente, en este caso, comparar cada orientación estimada con la orientación real que se debería obtener, construir los errores de cada uno acumulando los de cada experimento y ver las diferencias y similitudes entre cada método computando la desviación estándar de estos con respecto de su media (la cual podría ser la salida final [otra sería una combinación lineal —ponderación— de las respuestas] del sistema si este realizara regresiones y no clasificaciones), así como analizar en base a las características y condiciones de cada experimento las fortalezas y debilidades de cada algoritmo.

Más aún, es reseñable que la desviación estándar no es realmente un indicativo directo del desempeño del sistema, así que se utilizan, además, otras medidas estadísticas aplicadas sobre el total de la población para discutirlo (que son funciones de la orientación real y la estimada; ver las ecuaciones posteriores): el NMSE (*Normalized Mean Squared Error*), que ayuda a mejorar la calidad de la evaluación del rendimiento del modelo en diferentes experimentos, ya que toma en cuenta la escala de los valores reales, el RMSE (*Root Mean Squared Error*) (cuya principal diferencia con el MSE es que el RMSE está en las mismas unidades que los datos originales, mientras que el primero está en unidades al cuadrado) y el MAE (*Mean Absolute Error*), que es menos sensible a valores extremos que el RMSE. Además, cabe notar que si se supone que las referencias están perfectamente orientadas y con orientación igual a cero, cosa que es razonable y de hecho el análisis parte de dicho planteamiento, se puede calcular la desviación estándar de todas las estimaciones indiferentemente del experimento de que se trate, lo que proporciona una medida, para los experimentos cuya pieza actual hubiera estado (y debido estar) bien orientada, del rendimiento del sistema para validar verazmente las piezas.

$$MSE = \frac{1}{N}\sum_{i}\left(\theta_{real_i} - \theta_{estim_i}\right)^2 \tag{113}$$

$$NMSE = \frac{1}{N}\sum_{i}\frac{\left(\theta_{real_i} - \theta_{estim_i}\right)^2}{\bar{\theta}_{real_i}\cdot\bar{\theta}_{estim_i}} = N\sum_{i}\frac{\left(\theta_{real_i} - \theta_{estim_i}\right)^2}{\sum_i \theta_{real_i}\cdot\sum_i \theta_{estim_i}} \tag{114}$$

$$RMSE = \sqrt{MSE} \tag{115}$$

$$MAE = \frac{1}{N}\sum_{i}\left|\theta_{real_i} - \theta_{estim_i}\right| \tag{116}$$

En las siguientes tablas de resultados se insertan “barras de progreso” para proporcionar una referencia visual a la disparidad entre los tres métodos que estiman la pose de la pieza y los errores que, en relativo a ellos, comete cada uno. En aquellas columnas referenciadas por θ se indica el valor de las orientaciones estimadas por cada

método, que se indica somo subíndice. θ_{real} es la orientación de la pieza de referencia correctamente orientada (verdadero positivo en cada experimento), θ_{media} es el valor medio de todas las estimaciones y las dos últimas columnas se corresponden con la desviación estándar de las estimaciones de cada método con respecto de su media y el coeficiente de variación para cada prueba. Luego, más abajo se indican, por filas, las métricas de error que cuantifican el rendimiento de cada método, en cada iteración, dentro de cada experimento, para cada pieza. Los subíndices "total" y "media" resumen el rendimiento general del modelo regresor para la pieza considerada. Tras cuatro tablas con cuatro piezas distintas se presenta una tabla de resultados finales, para simulación y en la realidad, que aglomera todas las pruebas llevadas a cabo en cada entorno; todos los resultados se analizan y racionalizan en los dos subsiguientes apartados.

Tabla 7. Tabla de resultados cuantitativos de los ensayos realizados en simulación para la primera pieza.

Pieza 1		Estimadas (simulación)					
	θ_{real}	θ_{ECC}	θ_{PCA}	θ_{H}	$\boldsymbol{\theta}_{media}$	σsestimaciones	CVestimaciones (%)
Exp. I	0,000	1,441	0,390	0,200	0,677	0,546	80,616
	0,000	1,440	0,269	0,124	0,611	0,589	96,428
Exp. II	5,000	6,196	5,612	5,623	5,811	0,273	4,694
	6,000	4,615	7,693	6,337	6,215	1,260	20,269
Exp. IV	0,000	1,025	4,839	2,180	2,681	1,597	59,559
						ECM_{total}	1,321
ECM_{ECC}	1,308	ECM_{PCA}	2,319	ECM_{H}	1,031	$NMSE_{total}$	0,248
$NMSE_{ECC}$	0,264	$NMSE_{PCA}$	0,650	$NMSE_{H}$	0,167	$RMSE_{total}$	1,149
$RMSE_{ECC}$	1,144	$RMSE_{PCA}$	1,523	$RMSE_{H}$	1,015	MAE_{total}	0,250
MAE_{ECC}	0,259	MAE_{PCA}	0,174	MAE_{H}	0,469	σsmedia	0,853
						CVmedio (%)	52,313

Tabla 8. Tabla de resultados cuantitativos de los ensayos realizados en simulación para la segunda pieza.

Pieza 2		Estimadas (simulación)					
	θ_{real}	θ_{ECC}	θ_{PCA}	θ_{H}	$\boldsymbol{\theta}_{media}$	σsestimaciones	CVestimaciones (%)
Exp. I	0,000	6,868	0,613	4,156	3,879	2,561	66,026
	0,000	6,549	0,380	3,912	3,614	2,527	69,940
Exp. II	2,000	12,714	2,440	1,938	5,697	4,966	87,167
	32,000	29,826	48,256	33,257	37,113	8,003	21,563
						ECM_{total}	4,121
ECM_{ECC}	7,239	ECM_{PCA}	8,139	ECM_{H}	2,922	$NMSE_{total}$	0,159
$NMSE_{ECC}$	0,441	$NMSE_{PCA}$	0,603	$NMSE_{H}$	0,093	$RMSE_{total}$	2,030
$RMSE_{ECC}$	2,690	$RMSE_{PCA}$	2,853	$RMSE_{H}$	1,709	MAE_{total}	1,019
MAE_{ECC}	1,644	MAE_{PCA}	1,411	MAE_{H}	2,015	σsmedia	4,514
						CVmedio (%)	61,174

Tabla 9. Tabla de resultados cuantitativos de los ensayos realizados en simulación para la tercera pieza.

Pieza 3		Estimadas (simulación)					
	θ_{real}	θ_{ECC}	θ_{PCA}	θ_{H}	$\boldsymbol{\theta}_{media}$	σestimaciones	CVestimaciones (%)
Exp. I	0,000	2,815	2,243	0,665	1,908	0,909	47,655
	0,000	2,618	2,331	0,939	1,963	0,733	37,361
Exp. II	17,000	7,230	14,770	19,534	13,845	5,066	36,588
	15,000	7,404	8,765	25,382	13,850	8,173	59,011
Exp. IV	0,000	7,120	0,649	3,465	3,745	2,649	70,747
						ECM_{total}	2,561
ECM_{ECC}	6,613	ECM_{PCA}	3,309	ECM_{H}	5,051	$NMSE_{total}$	0,145
$NMSE_{ECC}$	1,257	$NMSE_{PCA}$	0,297	$NMSE_{H}$	0,399	$RMSE_{total}$	1,600
$RMSE_{ECC}$	2,571	$RMSE_{PCA}$	1,819	$RMSE_{H}$	2,247	MAE_{total}	0,596
MAE_{ECC}	1,197	MAE_{PCA}	1,087	MAE_{H}	0,711	σsmedia	3,506
						CVmedio (%)	50,272

Tabla 10. Tabla de resultados cuantitativos de los ensayos realizados en simulación para la cuarta pieza.

Pieza 4		Estimadas (simulación)					
	θ_{real}	θ_{ECC}	θ_{PCA}	θ_{H}	$\boldsymbol{\theta}_{media}$	σestimaciones	CVestimaciones (%)
Exp. I	0,000	3,071	0,481	0,248	1,267	1,279	100,971
	0,000	2,400	0,953	1,343	1,565	0,611	39,046
Exp. II	5,000	2,814	0,286	5,984	3,028	2,331	76,979
	7,000	7,069	25,494	11,652	14,738	7,832	53,142
						ECM_{total}	4,118
ECM_{ECC}	2,235	ECM_{PCA}	9,558	ECM_{H}	2,474	$NMSE_{total}$	1,098
$NMSE_{ECC}$	0,434	$NMSE_{PCA}$	4,475	$NMSE_{H}$	0,424	$RMSE_{total}$	2,029
$RMSE_{ECC}$	1,495	$RMSE_{PCA}$	3,092	$RMSE_{H}$	1,573	MAE_{total}	0,784
MAE_{ECC}	0,483	MAE_{PCA}	1,260	MAE_{H}	0,536	σsmedia	3,013
						CVmedio (%)	67,534

Tabla 11. Resultados en simulación de la capacidad final del sistema para estimar cuantitativamente la orientación de las piezas.

Simulación						Sistema completo	
ECC		PCA		H		ECMtotal	2,424
ECMECC	3,479	ECMtotal	4,665	ECMtotal	2,295	NMSEtotal	0,330
NMSEECC	0,479	NMSEtotal	1,205	NMSEtotal	0,217	RMSEtotal	1,362
RMSEECC	1,580	RMSEtotal	1,857	RMSEtotal	1,309	MAEtotal	0,530
MAEECC	0,717	MAEtotal	0,786	MAEtotal	0,746	σsmedia	2,377
						CVmedio (%)	46,259

Tabla 12. Tabla de resultados cuantitativos de los ensayos realizados en la realidad para la quinta pieza.

Pieza 5		Estimadas (realidad)					
	θreal	θECC	θPCA	θH	**θ**media	σsestimaciones	CVestimaciones (%)
Exp. I	0,000	5,048	3,804	0,281	3,044	2,019	66,326
	5,000	6,348	3,346	4,234	4,643	1,259	27,124
Exp. II	10,000	7,910	16,024	11,432	11,789	3,322	28,181
	15,000	19,142	13,346	17,275	16,588	2,416	14,563
						ECMtotal	1,944
ECMECC	3,494	ECMPCA	3,750	ECMH	1,405	NMSEtotal	0,056
NMSEECC	0,169	NMSEPCA	0,205	NMSEH	0,032	RMSEtotal	1,394
RMSEECC	1,869	RMSEPCA	1,936	RMSEH	1,185	MAEtotal	0,424
MAEECC	0,789	MAEPCA	0,808	MAEH	1,690	σsmedia	2,254
						CVmedio (%)	34,049

Tabla 13. Tabla de resultados cuantitativos de los ensayos realizados en la realidad para la sexta pieza.

Pieza 6		Estimadas (realidad)					
	θreal	θECC	θPCA	θH	**θ**media	σsestimaciones	CVestimaciones (%)
Exp. I	0,000	0,493	2,929	1,803	1,742	0,995	57,153
	2,000	1,239	3,619	3,300	2,719	1,055	38,795
Exp. II	10,000	7,455	16,712	13,357	12,508	3,826	30,592
	12,000	9,777	12,864	9,397	10,679	1,553	14,541
						ECMtotal	1,702
ECMECC	1,750	ECMPCA	3,775	ECMH	2,397	NMSEtotal	0,070
NMSEECC	0,108	NMSEPCA	0,263	NMSEH	0,138	RMSEtotal	1,305
RMSEECC	1,323	RMSEPCA	1,943	RMSEH	1,548	MAEtotal	0,393
MAEECC	0,376	MAEPCA	0,517	MAEH	1,264	σsmedia	1,857
						CVmedio (%)	35,270

Tabla 14. Tabla de resultados cuantitativos de los ensayos realizados en la realidad para la séptima pieza.

Pieza 7		Estimadas (realidad)					
	θreal	θECC	θPCA	θH	θmedia	σsestimaciones	CVestimaciones (%)
Exp. I	0,000	4,536	2,858	0,235	2,543	1,770	69,613
	5,000	7,235	2,144	4,123	4,500	2,095	46,562
Exp. II	15,000	15,358	35,235	19,242	23,278	8,602	36,954
	17,000	13,236	11,230	14,210	12,892	1,241	9,623
						ECMtotal	4,799
ECMECC	3,157	ECMPCA	10,713	ECMH	2,579	NMSEtotal	0,230
NMSEECC	0,107	NMSEPCA	0,964	NMSEH	0,076	RMSEtotal	2,191
RMSEECC	1,777	RMSEPCA	3,273	RMSEH	1,606	MAEtotal	0,964
MAEECC	0,681	MAEPCA	1,474	MAEH	1,844	σsmedia	3,427
						CVmedio (%)	40,688

Tabla 15. Tabla de resultados cuantitativos de los ensayos realizados en la realidad para la octava pieza.

Pieza 8		Estimadas (realidad)					
	θreal	θECC	θPCA	θH	θmedia	σsestimaciones	CVestimaciones (%)
Exp. I	0,000	2,345	6,326	2,540	3,737	1,832	49,031
	5,000	7,125	7,652	5,875	6,884	0,745	10,825
Exp. II	13,000	11,642	10,560	13,540	11,914	1,232	10,338
	16,000	10,256	18,643	16,875	15,258	3,610	23,658
						ECMtotal	2,193
ECMECC	3,348	ECMPCA	3,873	ECMH	1,438	NMSEtotal	0,060
NMSEECC	0,168	NMSEPCA	0,163	NMSEH	0,025	RMSEtotal	1,481
RMSEECC	1,830	RMSEPCA	1,968	RMSEH	1,199	MAEtotal	0,466
MAEECC	0,723	MAEPCA	0,644	MAEH	1,898	σsmedia	1,855
						CVmedio (%)	23,463

Tabla 16. Resultados en la realidad de la capacidad final del sistema para estimar cuantitativamente la orientación de las piezas.

Realidad						Sistema completo	
ECC		PCA		H		ECMtotal	2,128
ECMECC	2,350	ECMtotal	4,422	ECMtotal	1,564	NMSEtotal	0,083
NMSEECC	0,110	NMSEtotal	0,319	NMSEtotal	0,054	RMSEtotal	1,274
RMSEECC	1,360	RMSEtotal	1,824	RMSEtotal	1,108	MAEtotal	0,449
MAEECC	0,514	MAEtotal	0,689	MAEtotal	1,339	σsmedia	1,879
						CVmedio (%)	26,694

En la siguiente figura se comparan gráficamente los resultados de las métricas de regresión obtenidas en simulación y realidad. Como en el caso anterior, se observa una similitud remarcable en la forma y tendencia de los errores en ambos escenarios; las diferencias se deben mayormente a una falta de población:

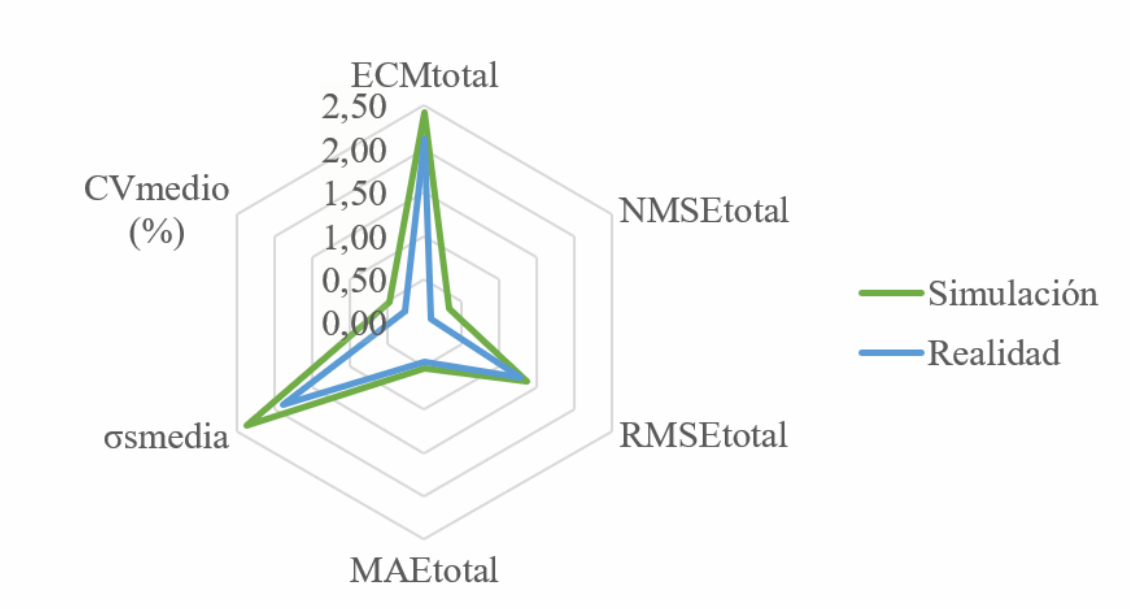


Fig. 115. Comparación de las métricas de regresión obtenidas en simulación y realidad.

Finalmente, en las dos figuras siguientes se muestran dos gráficos de barras en los que se comparan los errores cometidos en las estimaciones de la orientación de las piezas analizadas mediante los tres métodos principales que utiliza el algoritmo para esta tarea, en simulación y en la realidad (promediando todos los experimentos). Concuerda con lo que se explicará en la discusión; la estimación de la homografía es el método más estable y preciso, si bien todos tienen sus debilidades y fortalezas. En ningún caso se detectaron anomalías que no fueran outliers espurios por las que la desviación del valor estimado con respecto al valor real fuera preocupante y pudiera poner en riesgo la salida del sistema; aunque existan imprecisiones (no mayores de 5 º en ninguna de las pruebas reales o simuladas realizadas), la respuesta de clasificación, que es la de interés, se muestra efectiva.

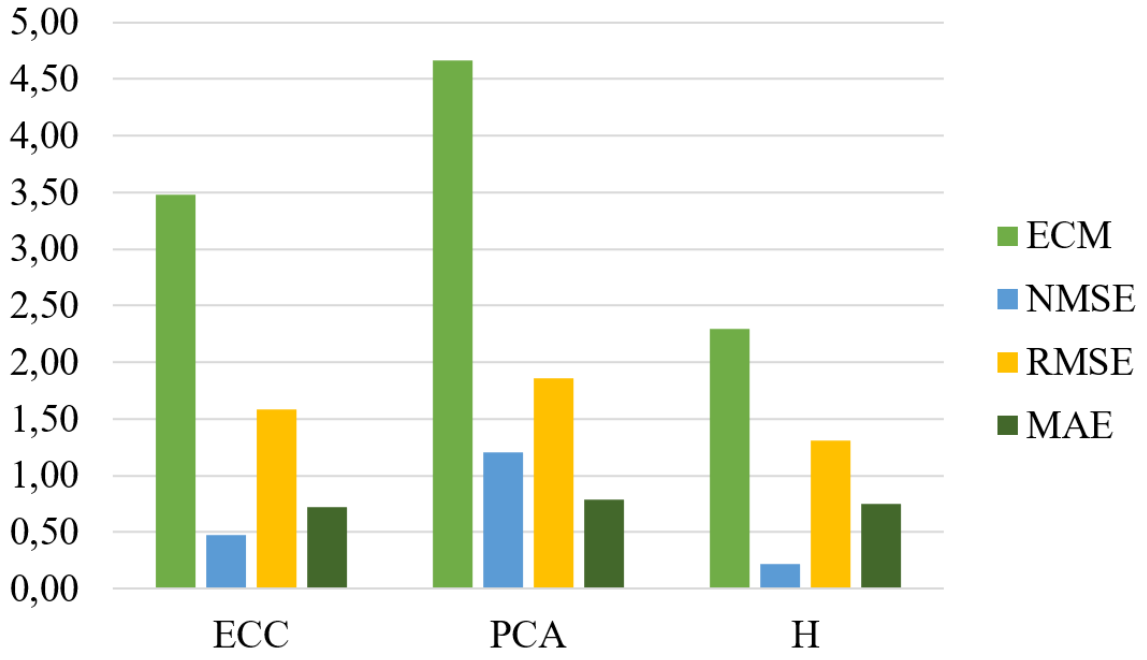


Fig. 116. Comparación de los errores de regresión obtenidos en simulación.

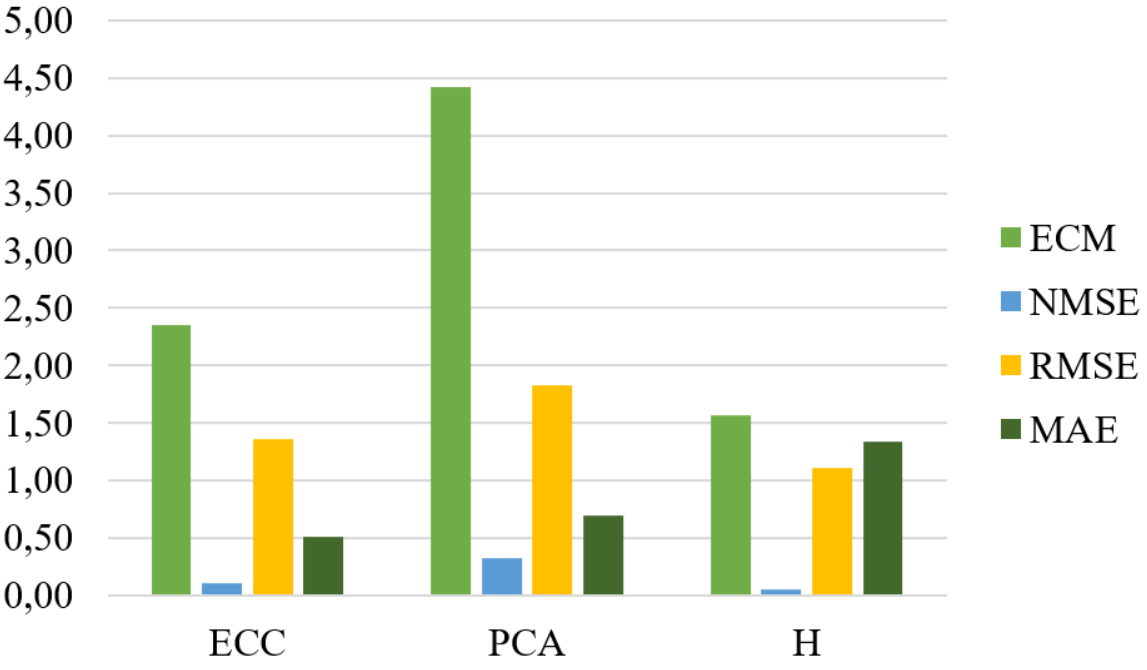


Fig. 117. Comparación de los errores de regresión obtenidos en la realidad.

4.- DISCUSIÓN

La idea inicial de emplear varias técnicas para estimar la pose de la pieza y luego complementarlas holísticamente ha demostrado superar varios escenarios en los que, con un solo método, la actuación del sistema se vería negativamente afectada. Utilizando diferentes técnicas, a decir, los algoritmos ECC, PCA y la estimación de la homografía, una puede suplir los fallos y puntos menos fuertes del resto, de manera que un apartado relevante de esta discusión compete a la búsqueda de bondades y limitaciones de cada cual. Se ha observado que el algoritmo ECC es especialmente útil para enfrentarse a variaciones de intensidad en las imágenes, pues el algoritmo está naturalmente diseñado para ello, aunque si la traslación entre referencia y pieza actual comienza a aumentar los resultados de su estimación convergen hacia mínimos locales (típicamente no es un problema, pues se emplean rectángulos delimitadores). Por otra parte, la estimación de la homografía es especialmente útil para afrontar cambios de perspectiva, desenfoque y otras aberraciones de la cámara, pues la calidad de identificación y emparejamiento de keypoints es en general suficiente para una estimación de transformación estable, el número de estos suele ser alto (aprox. 300 ÷ 500) y los detectores, descriptores, matchers y el resto de herramientas implementadas se pueden caracterizar para amoldarse de manera óptima al entorno en cuestión. Finalmente, el algoritmo PCA es útil para discernir piezas demasiado grandes, pequeñas, deformadas homogéneamente… con respecto a las referencias correctas o incluso para computar respuestas prematuras en etapas iniciales de cada ciclo de procesamiento que ayuden a mejorar la fiabilidad del sistema, a decir, detectar si hay más de una pieza en la cinta, si su forma (binarizada) difiere en sobremanera de cualquiera de las referencias o si su orientación inicial está exageradamente alterada.

La orientación estimada mediante PCA se ve notablemente más afectada, aproximadamente en un rango de ± 5 º en simulación para la mayor de las desviaciones angulares permitidas, a la perspectiva de la cámara. Así, mientras los otros dos métodos (homografía y ECC) pueden aproximarla, con respecto a la pieza de referencia, con una precisión de ± 0,5 º, el cuadro delimitador orientado se nota muy influenciado por el efecto de la distorsión por la traslación en la dirección horizontal de la cinta (sobre todo) y la calidad de la segmentación, que habría de ser más infalible (ante iluminación adversa y sombras) si solo se utilizada este método. Se podría decir que ECC podría ser, tras estudiar más en profundidad los métodos de optimización no lineal y mejorar su eficiencia computacional de diversas formas, el único método que reconozca las piezas y haga las veces de clasificador morfológico, pues es el que mejores resultados dio en este sentido, en general; pero también es cierto que merece mucho la pena estudiar más a fondo todas y cada una de las métricas usadas para ampliar sus capacidades, pues se nota que muchas de ellas son análogas y tienen una clara y estrecha relación matemática pero están enfocadas desde perspectivas sensiblemente distintas. En vista de los resultados de las tablas generales, el método más preciso es el de la transformación homográfica, aunque los otros

dos aportan también respuestas aceptables y no suponen valores atípicos en la composición de la respuesta final promediada.

Las condiciones reales se caracterizan por una iluminación naturalmente adversa y brillos espontáneos. Se tuvo que cambiar el método de binarización en la experimentación real; mientras ante simulación Otsu proporcionaba muy buenos resultados, la selección de un valor de intensidad umbral (25) para las piezas reales fue más precisa, tanto procesándolas offline como directamente con `PylonSensor`. Similarmente y como se ha comentado, múltiples parámetros del monitorizador del flujo de piezas se tuvieron que ajustar empíricamente según el flujo de frames que entraba al bloque de procesamiento provenían de la simulación, cámara en línea o las grabaciones mediante `ArchiveSensor` de una de las dos fuentes anteriores. Aunque el ajuste de estos factores fue concluyente y válido para todo el transcurso de las pruebas en todas las situaciones, debería haber cabida en trabajos futuros para el estudio de métodos de segmentación y monitorización del flujo de piezas automáticos y/o más robustos antes las distintas fuentes de luz y el bitrate de la fuente que emite las imágenes o los vídeos.

En general, la exactitud en las pruebas de clasificación es alta en la mayoría de los experimentos, indicando que el modelo acierta con frecuencia en la validación de las muestras; sin embargo, esta exactitud puede ser engañosa si hay variaciones dinámicas o muy marcadas de iluminación, especialmente en simulación. El cuarto experimento real destaca con un rendimiento perfecto, mientras que el cuarto y quinto en simulación muestra una precisión algo más baja, sugiriendo la presencia de falsos positivos por las causas mencionadas y el hecho de que el modelo no relaciona las medidas y las estimaciones exactamente igual en ambos entornos; un ajuste de algunos parámetros en algunos algoritmos en simulación haría que devolviese mejores resultados en estas pruebas con condiciones adversas pero no aportaría mucho valor a la finalidad del estudio. Los experimentos dan a inferir que el sistema logra tener una exhaustividad complaciente (la proporción de instancias positivas que se logran capturar). El F1-Score señala, por otra parte, a que hay un equilibrio óptimo entre la correcta clasificación de piezas válidas e inválidas, lo que es un buen indicativo. Finalmente, el MCC, algo reducido en el quinto experimento simulado, apunta a que conviene investigar en cómo hacer que el sistema sea más robusto en situaciones desfavorables; en el resto de pruebas en que pudo calcularse contrasta el buen funcionamiento del sistema, como lo hacen el PPV y NPV en el cuarto experimento simulado, indicando una predicción precisa tanto de casos positivos como negativos y el TPR en el cuarto, con acierto del 100%, indicando un rendimiento excelente en la identificación de casos positivos.

Aun habiendo pequeñas diferencias lógicas debidas a factores como la variabilidad el entorno, como los resultados en simulación y realidad son similares se sugiere que el modelo ha generalizado bien los datos desde el entorno de simulación al mundo real; en el contexto de la evaluación de modelos de clasificación esto implica que el rendimiento del modelo en condiciones simuladas refleja de manera efectiva su desempeño en situaciones del mundo real, lo que potencia las fortalezas del simulador y el algoritmo realizados.

Basándose en los resultados globales del sistema según el CV, el modelo tiene menos variabilidad en la realidad que en simulación con respecto a sus tres métodos principales, lo que concuerda con un razonamiento lógico pues las pruebas en simulación se realizaron bajo condicientes mucho más desfavorables (o, por lo menos, los experimentos reales no presentaban variaciones no constantes de iluminación, distancia a la pieza ni otros factores del estilo). Aunque en algunos casos la homografía se vuelva algo inestable y provoque desviaciones medias (calculadas mediante el MAE) de algo menos de 2 º (poco) con respecto al valor real, las métricas de error (que son ECM, NMSE Y RMSE) sugieren que su rendimiento es claramente superior a la de los otros dos métodos donde, además, por virtud especialmente del NMSE, ECC demuestra (por lo general) ser más fiable que PCA. Aunque la desviación estándar es vagamente inferior al valor del error cuadrático medio total del sistema, lo que puede parecer significativo, no es de mayor interés considerando que la respuesta del sistema promedia ponderadamente las inferencias logísticas de validez de la pose basándose en la moda de las respuestas individuales, además de que esta desviación, como se comentó, solo ayuda para comparar el desempeño de estos métodos. En esta línea, es remarcable que la acción conjunta de todos ellos hace parecer que se comporten igual y proporcionalmente peor en tanto en cuanto más adversas sean las condiciones (véanse los últimos experimentos), cuando en realidad el algoritmo que más afectado se ve, especialmente en la labor de estimar la orientación es, con mucho, ECC (por lo menos en simulación, que es donde se puso más a prueba) (véase el cuarto experimento).

En retrospectiva, los resultados obtenidos demuestran que el desempeño general fue positivo, destacando únicamente la falta de parametrización empírica en casos reales frente al simulador (varios sub-bloques del clasificador morfológico, estudio más detallado para una mejor extracción de keypoints…); es crucial subrayar la necesidad de adaptarse perfectamente al tamaño de la imagen, área de objetos o sombras, entre otros aspectos. Los resultados numéricos, tanto en simulación como en condiciones (más favorables) de experimentación real, revelaron errores medios de ECC y homografía de menos de 5 °, especialmente evidentes en el primer experimento. Sin embargo, se observó que para PCA, el algoritmo mostró menos falibilidad en condiciones reales, posiblemente debido a la dificultad para lograr una segmentación precisa en este escenario. Es también relevante señalar que ECC demostró una notable robustez frente a cambios de iluminación en las pruebas físicas, manteniendo un índice de correlación superior a 0,95 en todos los experimentos realizados con piezas del mismo tipo, dentro de los márgenes de orientación admisibles. Por otra parte, se nota que ECC da aún mejores resultados (aunque es sensiblemente más lento) en la realidad, debido a la mayor resolución del as piezas y calidad de la imagen de la cámara 2D en comparación con los frames provenientes de simulación. Todos los sub-bloques de reconocimiento, en su medida, devuelven resultados lógicos, más o menos precisamente según las condiciones cambian más o menos groseramente; la comparación de todos entre sí es un objeto de estudio aparte que merece otra discusión (como se comentó) y el intento de ajustar los parámetros específicos de

todas las métricas particularmente es una tarea ardua, como se pudo experimentar, pues por un lado ninguna funciona de la misma manera bajo unas condiciones (del entorno) concretas y por otro, bajo condiciones variables, sería como tratar de controlar decenas de variables dependientes al mismo tiempo. Pero, en resumen, la precisión de clasificación de piezas válidas, foco central de este análisis del sistema, ha sido satisfactoria.

La séptima pieza, usada en las pruebas reales, tenía muchos brillos, haciendo que ECC desempeñara algo peor (pero correctamente) que en el resto de experimentos; además, no solo su tamaño y circularidad (cuarto veces π por el área del contorno cerrado de la pieza partido por su perímetro al cuadrado) eran notablemente grandes en comparación con el resto, causando que PCA funcionase peor (ambigüedad del mayor vector propio), sino que también estaba extendido homogéneamente causando reflejos que crecían lo largo de todo su curso, provocando que la homografía fallase en alguna iteración (hecho también causado por la falta de contraste y esquinas en la pieza). Es también remarcable que, tras ajustar ciertos parámetros, los resultados de clasificación en simulación y realidad marcadamente similares, lo que aboga la utilidad y fiabilidad del simulador para desarrollar todo tipo de algoritmos.

A medida que la pieza es más pequeña la precisión del sistema parece disminuir; por eso la segunda pieza, que es la más pequeña de todas y la cual tiene una geometría medianamente compleja, provoca resultados ciertamente dispares (desviación estándar alta) entre los distintos métodos; en ECC se debe, seguramente, al submuestreo al que se ve atada la pieza que, de por sí, tiene una escala muy reducida (sin ello el resultado sería mucho más preciso, aunque más lento su cómputo). Sorprendentemente, en este caso PCA (que en el diseño del sistema parecía que sería el más impreciso) es el que se ajusta mejor a la orientación real de la pieza, pero no más que porque las condiciones de iluminación en Exp. I y Exp. II son exactas entre las piezas de referencia y actual; en el resto de pruebas se ve que los otros dos algoritmos mantienen, aunque con ciertas desviaciones razonables, también más estabilidad. También se observó que, a medida que se alabea la cámara y la pieza es más grande (y su superficie más homogénea), las correspondencias tienden a ser más equívocas, haciendo que sea plausible un ajuste de los parámetros de los detectores y descriptores de keypoints en tiempo de ejecución, para maximizar el rendimiento del sistema en este sentido; con este tipo de piezas la combinación FAST + AKAZE, con un número reducido de capas (2), octavas (1) y un factor de Lowe alto (1,0), funciona especialmente bien, detectando las características más externas y de más alta frecuencia de la pieza; sin una configuración perfeccionada para casos especiales como este el bloque que extrae la orientación basada en las correspondencias de puntos de interés tiende a dar resultados poco precisos. En este sentido, se hace visible que la estimación de la homografía proporciona mejores resultados que los otros dos métodos cuando la geometría de la pieza es muy compleja y tiene agujeros (por lo que la segmentación de su contorno es más complicada de realizar), como es el caso de la cuarta pieza; esto corrobora la idea de plantear ajustar dinámicamente el tipo de técnicas a utilizar en función de un análisis previo de la pieza que se está observando en tiempo real, para otorgar más flexibilidad al

sistema y no cohibir las fortalezas de unos algoritmos cuando en ciertas situaciones otros son mejores y viceversa. También se puede restringir la optimización no lineal del algoritmo ECC para que se busquen los parámetros de una transformación rígida, en vez de homográfica, a fin de que bajas resoluciones no afecten negativamente a la aproximación de la traslación y rotación en base a la homografía; cuestiones como estas quedarían estudiadas de cara a una implementación definitiva del sistema.

En el proceso de experimentación se ha explorado la capacidad de ajustar los umbrales de similitud de cada sub-bloque en el componente de reconocimiento (clasificador morfológico) para así adaptar su rigurosidad. Esta adaptación puede orientarse hacia la identificación estricta de tipos de piezas concretos con orientación válida o hacia una aproximación más flexible que reconoce solo los tipos de piezas en general, siendo tarea del clasificador características la de identificar finalmente la validez de su pose. La primera opción, enfocada hacia la identificación estricta, no requiere invarianza a traslación, rotación ni escala, debido a que las técnicas aplicadas, basadas en la correlación espacial o frecuencial, ya de por sí penalizan discrepancias en vecindades dispares; por otro lado, la segunda opción sí demanda estas invariantes. En este estudio se han llevado a cabo pruebas para implementar ambas perspectivas, individualmente y en complementación. La elección entre estas dos ideas influye en la personalidad de la etapa de reconocimiento, de tal forma que mientras la primera alternativa minimiza la importancia del problema de estimación de una pose cuantitativa, la segunda opción transforma el sistema hacia una naturaleza más “regresora” que “clasificadora”.

Como descriptor, SIFT demuestra su mejor rendimiento cuando el detector utilizado es también el suyo, aunque como segunda mejor opción, y más rápida, está el uso de FAST como detector. Por otra parte, ORB ha demostrado ser el descriptor más versátil, ayudando a proporcionar las correspondencias más estables (y mayor número de ellas) ante escenarios cambiantes, sobre todo con un tamaño característico de entre 1,5 y 2,2. Con todo, se ha notado que variaciones en la perspectiva denotan, no simplemente ruido, sino variaciones más o menos significativas en el rendimiento de los distintos detectores y descriptores, pero manteniendo constante el detector (p. ej., FAST), SIFT es el que (convincentemente) mantiene más estable su rendimiento ante este tipo de cambios y también frente a variaciones de escala rotación y afinidad en general y, además, como detector, muestra un buen desempeño (en comparación con el resto) frente a cambios en el grado de desenfoque (p. ej., contra el suavizado añadido por el filtro gaussiano o el de mediana), aunque no tanto como ORB o AKAZE en este sentido; contrariamente, se puede decir que la variabilidad de del rendimiento de los descriptores no es dependiente de los cambios en el desenfoque de la cámara, sino arbitraria, ya que estos no son generalmente los máximos responsables de la invarianza contra el desenfoque (como lo serían el detector y quizás la etapa de preprocesamiento). Se aprecia que AKAZE es el descriptor que peor mantiene la repetibilidad (capacidad de describir suficientemente bien los puntos como para generar la máxima correspondencias posibles, bajo las mismas condiciones) para piezas con reducciones de escala cada vez mayores, por lo que en una situación en la que

fuese necesario alejar más la cámara de lo común, esto sería un punto negativo; por el contrario SIFT parece mantener una repetibilidad constante. Se aprecia que ORB (gracias a BRIEF) es también el descriptor más rápido y además no tiene requisitos de orientación, lo que permite emparejarlo con cualquier detector; especialmente con FAST muestra un buen desempeño. El detector de Harris comienza a fallar cuando la textura de los objetos es muy marcada; el resto de detectores lo supera en la detección de regiones locales. También se ha notado que ORB es menos preciso que AKAZE ante cambios de escala y rotación, pero comparativamente similar frente a cambios de afinidad; por curiosidad, KAZE se ha notado también menos preciso que AKAZE para cambios de escala y afinidad, mientras que más frente a rotaciones.

El algoritmo de fuerza bruta funciona, en esta aplicación, notablemente peor que los otros dos métodos; en la siguiente figura se puede observar una evidencia de ello, en la que, con una pieza de referencia orientada a 20 º, se usó SIFT como detector y descriptor (sin filtro geométrico para mantener un comportamiento lo más estable posible) hasta la iteración 30 con BF como matcher, momento en que se cambió por FLANN (mejorando enormemente la precisión del sistema) y, mientras la iteración actual fuera menor que la número 60, se mantuvo en funcionamiento hasta finalmente cambiarla por K-NN, notándose que los dos últimos métodos presentan un funcionamiento prácticamente análogo:

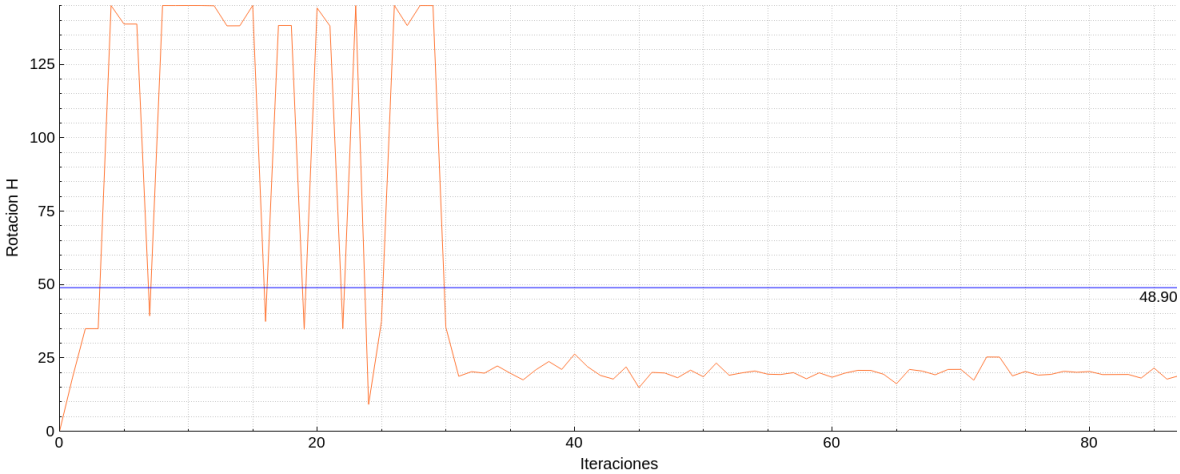


Fig. 118. Gráfica que muestra el mejor desempeño de FLANN y K-NN frente a BF, en esta aplicación.

Por otra parte, se puede concluir que, en base a las pruebas realizadas, si se requiere una respuesta "rápida" del sistema entonces emplear un filtro geométrico rígido (austero) es más indicado, pues otorga la capacidad de devolver respuestas fiables en pocas iteraciones (usando la media). Pero si se desea obtener respuestas aún más estables, sin importar demasiado el tiempo de ciclo, entonces sería más indicado ser lo más laxo posible en cuanto a la posición y orientación de los keypoints en el filtro geométrico (180 º y 500 px. o más en ambos casos) y "dejar" a los propios matchers y a RANSAC que "se ocupen" de todo el trabajo de eliminación de outliers; de esta manera aparecen miles (varios pares) de correspondencias, en proporciones de corrección arbitrarias, pero también se asegura

una mayor fiabilidad y la solución al problema de que los keypoints laterales desaparezcan aparece a medida que la pieza se aleja del centro del eje óptico de la cámara, provocado por la perspectiva y (sobre todo) los brillos… De suerte, al haber tal cantidad de puntos de interés, las correspondencias correctas (con un ratio de Lowe típicamente mayor que 0,75) compensan la pérdida de los que desaparecen, en comparación con el uso del filtro geométrico y un ratio de Lowe bajo; además, es en parte causa de que la homografía se resuelve a través de un sistema sobredeterminado en mucha mayor proporción.

En la siguiente figura se ve cómo la orientación estimada, aunque puede cambiar notablemente debido a la perspectiva y los keypoints de antes y después pueden tener muy pocas correspondencias, el algoritmo aprovecha los resultados de los bloques que usan las características morfológicas de la pieza para ser más insensible ante estas deformaciones geométricas. Por otra parte, es por esto por lo que la orientación "real" sería la que cayese cuando el objeto estuviera en la vertical de la cámara (en su verdadera magnitud); en esta posición (o dentro de un margen admisible centrado) se podrían analizar únicamente las piezas mediante el clasificador característico, si las situación fuera más estricta y se quisiera un funcionamiento menos permisivo, pero teniendo en cuenta que la elección de la óptica de la cámara se elige a voluntad, esto no representa ni siquiera un problema potencial para este trabajo. En caso contrario, otras posibilidades habrían podido ser emplear algoritmos de umbralización adaptativa o local, como los de Niblack, Sauvola, Bernsen, Eikvil o Parker.

Fig. 119. Pieza en distintas posiciones y perspectivas son su cuadro delimitador orientado estimado.

En cuanto a los sub-bloques que sirven como métricas del clasificador morfológico, aunque no sería labor de esta discusión hacer una comparativa de cada uno sino del desempeño del sistema en general, como algoritmo, se puede decir que las primeras técnicas que se ven negativamente afectadas en cuanto a rendimiento al mover las piezas a una pose incorrecta son el error cuadrático medio (aun con las medidas tomadas en cuenta para normalizar las señales todo lo posible hay iteraciones en las que no es fiable), Shape Context y Hausdorff. Aunque la última debería ser revisada y habría de mejorarse su rendimiento y la forma en que se le transfieren las señales de entrada para elegir la mejor métrica de distancia que utilizar, la situación de las dos primeras (y en cierto modo todas aquellas que utilizan el perfil unidimensional [las de `ContoursSimilarity`] en vez de las imágenes) indica que son las más restrictivas de todas (solo con 5 º que rote una pieza o se

traslade varias decenas de píxeles, afectándole la perspectiva, se detectará como mal orientada, según los umbrales actuales), por lo que (como se terminó por hacer en la práctica; con ocho sub-bloques que validen la pieza se da por válida según las métricas que analizan las señales 1D de contornos) se puede penalizar su decisión para inferir un cierto indicio de mala orientación de la pieza y así no influya concluyentemente en la salida del sistema. Una alternativa sería cambiar los umbrales y hacer que estos bloques sean mucho menos restrictivos, pero se perdería su capacidad de voto y decisión; se considera más provechoso interpretar de cierta forma una respuesta muy austera que ignorarla. Se ha notado que el uso del algoritmo ECC, presumiblemente a causa de las etapas de submuestreo añadidas para hacer que sea más eficiente computacionalmente, proporciona resultados algo inestables a la hora de estimar la matriz de homografía (del orden de ± 2 º en la repetición de un mismo experimento bajo condiciones análogas), mas no lo a la hora de estimar el coeficiente de correlación mejorado para identificar la pieza, pues la precisión en dicho respecto es altísima en todos los casos (coeficiente mayor que 0,85 bajo cualquier deformación razonable de perspectiva, desenfoque, iluminación o posición de las piezas); por otra parte, resulta comprensible que la extracción de las matrices de rotación y traslación a partir de la homografía proporcionada por ECC no sea un procedimiento estable, pues lo que trata el algoritmo es de agregar componentes perspectivas a la imagen de entrada para aproximarla a la plantilla, con lo que cuanto más distinta sea la pieza actual de la de referencia menos precisa será la aproximación de descomponer dichas matrices directamente de la homografía. Una idea sería privar al sistema del algoritmo ECC para la estimación de la pose de las piezas y solamente utilizarlo para identificarlas (y él solo podría ser suficiente para dicha tarea si se lograse hacer aún más veloz; habría que estudiarlo), o si introducir más de una cámara o algoritmos de geometría multivista y matrices de proyección de cámara para analizar más precisamente la matriz de homografía estimada. Aunque cabe decir que ECC es el algoritmo más fiable para este fin y para correlacionar las imágenes ante cambios groseros de iluminación, como se nota especialmente en Exp. IV para la primera pieza; además los valores menos ajustados de orientación van siempre ligados a los menores (peores) coeficientes de correlación y a las mayores traslaciones (de la transformación geométrica) en valor absoluto.

En la siguiente figura se muestra una observación notada en la fase de preprocesamiento de los frames de entrada al procesador. Denota la mejoría acaecida al omitir la ecualización (usual) del histograma de la imagen cuando la iluminación es muy intensa o está muy focalizada en la cinta transportadora, estableciendo una vecindad de mucho mayor contraste e intensidad que el resto y resaltando características no deseadas, lo que produce que se supongan como parte de la pieza píxeles que no forman parte de esta.

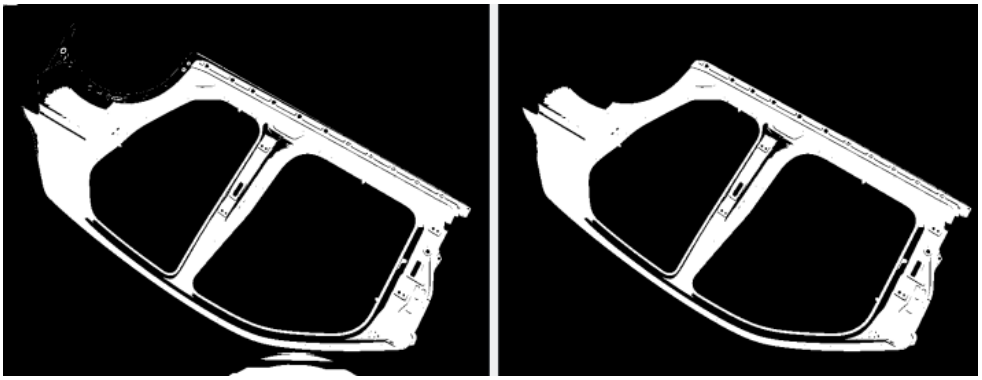


Fig. 120. A la izquierda la segmentación ecualizando el histograma del frame "puro" extraído. A la derecha la segmentación sin ecualización.

A continuación se muestra unos de los problemas típicos que se notaron en la fase de experimentación. El problema se trata de la disparidad en la fiabilidad de las respuestas de los sub-bloques que manejan la forma del contorno de la pieza en el clasificador morfológico (sobre todo el contexto de formas o la distancia de Hausdorff) ante cambios notables ya sea en la posición de la cámara o en la iluminación presente. La aparición de brillos y sombras puede mermar la calidad de la segmentación de la pieza y, con ello, estas métricas tenderían a proporcionar respuestas poco estables. Se hace notar, a futuro, un interés especial en hacer más robusta la calidad de la umbralización y posterior segmentación de la pieza.

Fig. 121. Efecto negativo de brillos en la segmentación de la pieza, que empeoran el rendimiento de métricas basadas en el análisis de la forma y distancia de los contornos.

El tiempo de cómputo del clasificador morfológico (especialmente) crece proporcionalmente al número de imágenes de referencia que haya en la base de datos; la identificación de la pieza puede alargarse demasiado si el tamaño de las imágenes es excesivo en algoritmos intensivos como puede ser ECC; se puede recurrir al submuestreo, como se hace, pero a medida que el problema crece se entra en un problema de desequilibrio entre precisión y peso computacional. Un modelo de aprendizaje automático entrenado previamente para reconocer las piezas (o más aún, las piezas con poses válidas) ayudaría clarísimamente a balancear este desequilibrio.

De todos modos, se puede decir que el sistema realiza una correcta interpretación de la pose de las piezas en tanto en cuanto su orientación no excede los límites admisibles;

además es totalmente invariante a la traslación, si el efecto de la perspectiva está mitigado, pues en todo el análisis y algoritmo se explota el rectángulo delimitador de la pieza. De cara al usuario (al exterior), el sistema se mantiene agnóstico a cuál es el valor concreto de la orientación y posición de la pieza, pues su objetivo es determinar si una pieza está bien o mal situada con respecto a una referencia válida, no proporcionar el valor exacto de su orientación y posición si es que es lógico hacer tal estimación; por ejemplo, mientras ante una entrada con la tercera pieza rotada aprox. 95 º (demasiado) el sistema lanza la respuesta prematura que indica que la pieza es inválida (no reconocida), donde el clasificador morfológico realmente sí que la identifica, pero ciertamente como una pieza mal orientada. Y la orientación estimada por PCA en esta prueba fue 99,5 º, donde la homografía dio 115,9 º, información que no saldría del algoritmo porque su precisión no sería de interés, al contrario que la decisión final con ella compuesta.

5.- CONCLUSIONES Y TRABAJO FUTURO

Para realizar un análisis experimental riguroso, es fundamental contar con una definición precisa de las muestras que serán objeto de estudio. Este requisito se extiende a la delimitación clara de las condiciones de contorno, abarcando aspectos cruciales como iluminación, posición, velocidad, tamaño de las piezas, entre otros. La importancia de esta precisión radica en la necesidad de garantizar una buena repetitividad en los resultados obtenidos. La repetitividad, en este contexto, se refiere a la capacidad de obtener mediciones consistentes y confiables cuando se replican las condiciones experimentales. Este enfoque riguroso contrasta con prácticas previas que podrían haber sido más laxas en cuanto a la definición detallada de las condiciones experimentales. La exigencia de ser meticuloso en este sentido se ha destacado como un elemento clave para lograr análisis reproducibles y consistentes, los cuales, aun con factores y agentes externos tan desfavorables en algunas configuraciones, el sistema ha demostrado superar las pruebas satisfactoriamente. Se han presentado dos análisis de los que se extrajeron numerosos corolarios sobre el desempeño del algoritmo implementado y sus partes; un análisis de clasificación y otro de regresión, ambos enfocados hacia un misma problemática y evaluados a través de varias métricas que reflejan puntos de vista del rendimiento del algoritmo desde diferentes perspectivas a través de la prueba con distintas casuísticas y experimentos que lo ponen a prueba en entornos tanto simulados como reales.

El análisis con la transformada de Fourier, que tan útil ha sido para desarrollar varias métricas de similitud, ha servido también para notar que merece un estudio aparte, junto con el algoritmo ECC para la tarea de identificación y un análisis más detenido en la matemática de la optimización no lineal para investigar la creación de algoritmos analíticos más complejos que permitan buscar la similitud entre dos piezas, “frecuencia a frecuencia”, a lo largo de todo el espectro, de manera similar a como en el córtex visual se procesa la forma más primaria de los objetos antes la agrupación de características y la identificación más precisa usando sus frecuencias altas. En otra consideración, este trabajo proporciona un estudio abstracto, aunque mediante un modelo analítico, de la relación entre los datos medidos, que son intensidades de píxeles a lo largo de la dimensión temporal y un dato categórico del mundo real, que es el tipo de pieza (válida o inválida); un paso más allá podría ser proporcionar una descripción más precisa entre las observaciones y la realidad a través de una relación estadística que, en definitiva, aproveche todo lo que se conoce sobre los datos medidos transformándolo en todo lo que se pueda conocer sobre la realidad, ya sea estimando la probabilidad a posteriori en un problema de clasificación o de manera reforzada mediante algún tipo de entrenamiento.

En lo relativo a una perspectiva futura y mejoras potenciales, con las que se busque mejorar la eficiencia y precisión del sistema (especialmente en el apartado de la segmentación de las piezas), una alternativa clarísima que se podría adoptar como forma de resolver el problema propuesto involucra plantear una regresión o clasificación con redes

neuronales. Un primer paso podría ser realizar la detección y segmentación de las piezas, en movimiento, gracias a arquitecturas como YOLO o Mask R-CNN; seguidamente, la extracción de características de manera automática mediante capas de convolución ayudándose de modelos como Inception ayudarían a identificar o (posteriormente) procesar las piezas. En este sentido, se puede recurrir al transfer learning para utilizar los pesos de redes preentrenadas y, por otra parte, "cortar" la arquitectura de alguna red convolucional profunda un paso antes de la etapa de clasificación llevada a cabo en las capas densas (totalmente conectadas) a fin de explotar de otra forma los vectores de características de interés, quizás mediante una SVM que separe mediante un hiperplano dos categorías: pieza válida y pieza inválida. Esto resulta más claro considerando que, en una implementación industrial definitiva, una propuesta razonable sería disponer de dos bloques: uno que usara técnicas de machine learning para detectar y clasificar la pieza de que se trata en el momento actual y otra que empleara técnicas analíticas y fuese diseñada para ser especialmente eficiente computacionalmente para resolver el problema de la validación de la pose de la pieza actual, tomando como entrada una imagen de referencia (bien orientada) que el bloque inteligente le enviase. En esta línea, con la información de la pose de la pieza estimada, un brazo robótico (p. ej.) podría ir a recogerla; por esto una idea planteable sería mejorar la precisión de esta estimación mediante marcadores de localización conocida en el espacio circundante de la cinta, para incorporar la pose con respecto de la cámara, no solo con respecto del plano imagen o el plano horizontal de la cinta transportadora. Se subraya el agnosticismo del sistema propuesto sobre el valor exacto de la pose de la pieza, centrándose en la determinación de si la pieza está correctamente situada o no con respecto a una referencia válida, pero se analizan con detenimiento sus decisiones cuantitativas internas y se logra extrapolar diversos puntos de mejora y hacer más precisas las métricas y técnicas del algoritmo.

En esta línea, podría ser viable explorar e implementar algún método de boosting (AdaBoost, XGBoost, GBM…) que involucre un modelo de aprendizaje profundo; definitivamente también sería interesante y muy fructífero apostar por arquitecturas de redes neuronales convolucionales como cualesquiera de las comentadas en el estado del arte. Adoptando estas ideas sucedería que, como se ha comentado sobre los detalles implementados en `Procesador::compareSimilarityData` y otras partes, si alguna métrica de validación de la pose de la pieza de entre las desarrolladas tendiera a ser particularmente menos precisa que el resto, a lo largo de la etapa de entrenamiento, el modelo lo ponderaría con menor fuerza que al resto y, así, se tendría "menos en consideración" su respuesta como "aportadora" a la salida final del sistema, pues su fiabilidad habría ido disminuyendo, proporcionando contrariamente mayor precisión al sistema inteligente en su conjunto. Por otro lado, también se podría estimar la forma tridimensional de las piezas y, por tanto, su orientación, utilizando sombreado fotométrico (shape from shading) y estereoscopía: en este sentido, la relación entre la irradiancia de la imagen, la posición de la cámara y la dirección conocida de fuentes de luz artificiales se utilizarían para calcular las normales de la superficie (recuperar la forma a partir de la

reflectancia de los objetos); el sombreado local proporcionaría información sobre la orientación local de la superficie sin necesidad de reconstruir explícitamente la profundidad, donde configuraciones típicas involucran una sola cámara 2D y múltiples fuentes de iluminación puntual, con una sola fuente activa en cada momento para utilizar la información de múltiples imágenes y la variación en la irradiancia permitiendo estimar la orientación del objeto. El empleo de otro tipo de sensores también abriría las puertas a una infinidad de posibles algoritmos; sin más, un dispositivo de triangulación láser (o un par estéreo de cámaras 2D) permitiría extraer la proyección de la forma 3D de la pieza a lo largo y ancho de, por lo menos, un plano (con el sensor estático) y se podrían extraer características de la nube de puntos (descriptores de superficie) como si de keypoints bidimensionales se trataran. O, en definitiva, apuntar a cualquier método de machine learning como los comentados en el estado del arte que sustituyera cualquier apartado analítico aquí realizado y someterlo a un análisis de rendimiento para valorar sus fortalezas y debilidades.

Una tarea que, por practicidad y tiempo, no se ha terminado de llevar a cabo en el momento de redacción de esta memoria sería la refactorización del código de la clase `Procesador` en varias subclases, de manera que las distintas entidades y bloques del algoritmo se vean más claras mirando directamente el código fuente y se permita además una interacción más fiable entre objetos. Esta sola clase cuenta con unas 5000 líneas de código, lo que no es muy escalable y podría ser organizado de una manera similar a como se reorganizó el código fuente del simulador realizado. También podría ser objeto de estudio futuro, pues en este no se le prestó demasiada atención ya que no era un requisito fundamental del proyecto, la comparación en profundidad del tiempo de cómputo de cada parte del algoritmo, además de su mejora en eficiencia, quizá "más" a través de *lvalues* o empleando el módulo CUDA de OpenCV para la GPU, por ejemplo.

Es claro, desde luego, que la precisión en las condiciones experimentales es clave para resultados repetibles y confiables, siendo mandatorio definir rigurosamente numerosas condiciones de contorno, previa experimentación (abarcando iluminación, posición, velocidad, tamaño de las piezas, etc.). En conclusión, se diseña e implementa fructíferamente un sistema analítico de alta complejidad en todas sus partes, logrando atacar integralmente a un problema industrial real. Este logro resalta una capacidad para abordar problemas desafiantes en el reconocimiento y posicionamiento de objetos de geometría compleja. Además, la interacción armoniosa de los distintas componentes del sistema indica una unión efectiva de algoritmos y herramientas; esta sinergia contribuye a la solidez y eficiencia global del sistema.

La combinación de diferentes tipos de algoritmos ha demostrado ser efectiva para estimar la pose de piezas en diversos escenarios. Aunque cada método tiene sus fortalezas y debilidades, la estrategia global ha logrado superar diversos desafíos, mostrando una precisión satisfactoria en la clasificación de piezas válidas en casos reales y virtuales, incluso en condiciones adversas de iluminación. Se destaca la utilidad de ECC para afrontar variaciones de intensidad, la eficacia de la homografía ante cambios de

perspectiva y la capacidad de PCA para discernir piezas de muy diferentes morfologías contando con una segmentación acertada. Se reconoce la importancia de ajustes empíricos en condiciones reales, especialmente en la binarización y parámetros del flujo de piezas. En la detección de keypoints, ORB muestra una especial versatilidad y SIFT demuestra la mayor estabilidad ante variaciones adversas de las características del entorno. Se sugiere explorar modelos de aprendizaje automático para mejorar el equilibrio entre precisión y tiempo de cómputo. A pesar de desafíos en métricas del clasificador morfológico, se concluye que el sistema logra una correcta interpretación de la pose de las piezas, siendo agnóstico al valor exacto de la orientación. Finalmente, quedan firmemente señaladas varias áreas de mejora, como la robustez en la umbralización y segmentación, para futuras investigaciones y ampliaciones.

A pesar de haber logrado resultados loables, este trabajo se configura como una versión embrionaria de un proyecto a futuro, revelándose a lo largo de este desarrollo distintas instancias propicias para su optimización. En torno a la aproximación al problema real, se divisaron diversos aspectos que demandan refinamiento. En esta perspectiva, se postula como paso fundamental ceñir las condiciones de contorno del problema de manera que las muestras, grabaciones, características del entorno… queden perfectamente definidas previamente a la indagación de resultados óptimos y, además, la agregación de agentes inteligentes para extender aún más la solución y analizar cómo desempeñan. Cabe decir que, aunque el sistema se ha sometido a unas pruebas poco benévolas, en el sentido de que se permitía la variación dinámica de la iluminación, la presencia de diferentes fuentes de ruido, la modificación de la perspectiva de manera notable, etc., en una situación práctica final y estable su desempeño sería aún más fiable. Y todavía más pudiendo especializar los algoritmos analíticos a un caso con unas condiciones de contorno completamente delimitadas a priori.

Resulta plausible señalar como aspecto altamente auspicioso el hecho de que, merced al trabajo emprendido hasta la fecha, se cuenta con un simulador viable, funcional, fácilmente moldeable y que, en este entorno, permite someter a prueba modelos algorítmicos mediante diversos sensores con el fin de discernir su pertinencia para el propósito concebido y otros similares. Adicionalmente, como se ha disertado a lo largo de este compendio, la operatividad real se encuentra en cierto grado vinculada al entorno simulado, de modo que el resto de adaptaciones adicionales requeridas para habilitar el código en un caso real no deberían conllevar un esfuerzo desmedido.

6.- PRESUPUESTO

6.1.- Coste de ejecución material

El coste de ejecución material incluye tres categorías: coste de equipos, coste de software y coste de mano de obra por el tiempo empleado en el proyecto.

6.1.1.- Coste de equipos

En este apartado se incluyen las herramientas hardware y software que se han utilizado a lo largo del proyecto y han permitido su avance y desarrollo. Se considera una amortización del 12,5 % (*), suponiendo una vida útil de los equipos de aprox. cuatro años y que fueron seis meses los que duró el proyecto.

Tabla 17. Coste de equipos.

CONCEPTO	PRECIO UNITARIO (€)	CANTIDAD	SUBTOTAL (€)
ASUS TUF Gaming F15 FX507ZC4 (*)	1249,00 · 0.125	1	156,13
Crucial RAM 32GB DDR5 4800MHz (*)	107,71 · 0.125	1	13,46
Microsoft Windows 10 Home (*)	110,82 · 0.125	1	13,85
Microsoft 365	11,70 €/mes · 2 meses	1	23,40
Monitor Asus VA24EHE 23.8"	99,00	2	198,00
Teclado Logitech K400 plus (*)	36,99 · 0.125	1	4,62
Ratón Logitech M705 (*)	58,99 · 0.125	1	7,37
Cámara acA1920-155um	1005,89	1	1005,89
Objetivo Pentax 25 mm F 1.4	44,10	1	44,10
Cable USB 3.0 - µB d2 m	9,99	1	9,99
Cable HDMI	15,04	2	30,08
Fuente alimentación 200 W (*)	54,99 · 0.125	1	6,87
	Subtotal (coste de equipos):		1513,77

Las licencias de Qt, OpenCV y Kubuntu 22.04 LTS están disponibles gratuitamente.

6.1.2.- Coste de mano de obra

En este apartado se incluyen los costes asociados a la mano de obra considerada para la ejecución del trabajo realizado. Para el cálculo del coste de mano de obra se parte de las horas dedicadas a cada parte del presente proyecto. La hora de ingeniería se establece con un coste de 40 € para aquellas tareas que requieran un conocimiento profundo de la materia ([1]), mientras que es de 20 € para las tareas rutinarias o repetitivas ([2]) y 15 € para la documentación ([3]), con un trabajo de ocho horas diarias y cinco días a la semana.

Tabla 18. Coste de mano de obra.

CONCEPTO	CANTIDAD (horas)	SUBTOTAL (€)
Estudios previos, formación y recopilación de información ([2])	120	2400,00
Instalación librerías y entorno de trabajo ([2])	10	200,00
Diseño del sistema y sus algoritmos ([1])	160	6400,00
Diseño y estructuración del código fuente ([1])	100	4000,00
Desarrollo de software ([1])	540	21600,00
Ensayos y experimentación ([2])	80	1600,00
Redacción documentos ([3])	380	5700,00
Total (horas):	1390	
	Subtotal (coste de mano de obra):	41900,00

6.2.- Coste total del presupuesto de ejecución material

Tabla 19. Coste total del presupuesto de ejecución material.

CONCEPTO	SUBTOTAL (€)
Coste de equipos	1513,77
Coste de mano de obra	41900,00
Subtotal (coste total del presupuesto de ejecución material):	43413,77

6.3.- Gastos generales y beneficio industrial

A los costes especificados anteriormente se les suma un 13 % de gastos generales (luz, uso de computadores, utilización de las instalaciones de trabajo…) y un 6 % de beneficio industrial. Al total de esto se le aplica un 21 % de IVA.

Tabla 20. Gastos generales y beneficio industrial.

CONCEPTO	SUBTOTAL (€)
Gastos generales (13 %)	5643,79
Beneficio industrial (6 %)	2604,83

6.4.- Importe total

Tabla 21. Importe total.

PRESUPUESTO DEL PROYECTO	
CONCEPTO	COSTE (€)
Coste de equipos	1513,77
Coste de mano de obra	41900,00
Coste total bruto	**43413,77**
Gastos generales (13 %)	5643,79
Beneficio industrial (6 %)	2604,83
Coste total sin impuestos	**51662,39**
IVA (21 %)	10849,10
TOTAL	**62511,49**

Por tanto, el presupuesto total del proyecto asciende a:

SESENTA Y DOS MIL QUINIENTOS ONCE EUROS CON CUARENTA Y NUEVE CÉNTIMOS.

Fecha: 02 de febrero de 2024

Firma del proyectante:

Alejandro Garnung Menéndez

7.- PLANIFICACIÓN

A continuación se muestre el diagrama de Gantt simplificado que compendia todas las etapas de que se conforma el proyecto y explica la planificación cronológica de este, en la que se indican las tareas y fases que lo constituyen y el tiempo dedicado a ellas.

	2023																2024							
	Septiembre				Octubre				Noviembre				Diciembre				Enero				Febrero			
FASE 1: DISEÑO CONCEPTUAL																								
Estudios previos y planteamiento del sistema	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
1.1 Análisis del estado del arte																								
1.2 Formación con las herramientas de trabajo																								
1.3 Diseño conceptual del sistema																								
1.4 Diseño teórico del algoritmo analítico																								
FASE 2: DISEÑO TECNOLÓGICO																								
Desarrollo de software	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
2.1 Elección del simulador a utilizar																								
2.2 Planteamiento de la estructura software del sistema																								
2.3 Programación del simulador																								
2.4 Programación de las interfaces																								
2.5 Fusionar simulador e interfaces (en Camera2DSensor)																								
2.6 Implementación del algoritmo analítico																								
2.7 Iteraciones y mejoras del algoritmo en sistema real y virtual																								
FASE 3: VALIDACIÓN																								
Pruebas experimentales y resultados	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
3.1 Diseño de las pruebas de verificación y validación																								
3.2 Ejecución de las pruebas																								
FASE 4: DOCUMENTACIÓN																								
Realización de la documentación	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
4.1 Elaboración del Doxygen del simulador																								
4.2 Redacción de la memoria																								
4.3 Elaboración de los anexos																								

Fig. 122. Cronograma del proyecto.

8.- REFERENCIAS Y BIBLIOGRAFÍA

[1] Klarák J, Kuric I, Zajačko I, Bulej V, Tlach V, Józwik J. Analysis of Laser Sensors and Camera Vision in the Shoe Position Inspection System. Sensors. 2021; 21(22):7531. https://doi.org/10.3390/s21227531

[2] Diggin, William & Diggin, Michael. (2020). Fast 3D Image Moments.

[3] A. Saxena, J. Driemeyer and A. Y. Ng, "Learning 3-D object orientation from images," 2009 IEEE International Conference on Robotics and Automation, Kobe, Japan, 2009, pp. 794-800, doi: 10.1109/ROBOT.2009.5152855.

[4] Agulló, M., Gil, C., Torres, P., & Fernando. (2015). Detección de deformaciones 3D calculando esqueletos de curvaturas. Actas de las XXXVI Jornadas de Automática, 2-4 de septiembre de 2015, Bilbao. Comité Español de Automática de la IFAC, pp. 66-72. http://hdl.handle.net/10045/58323.

[5] Liu, Cuiyin & Xu, Jishang & Wang, Feng. (2021). A Review of Keypoints' Detection and Feature Description in Image Registration. Scientific Programming. 2021. 1-25. 10.1155/2021/8509164.

[6] Sulong, Ghazali & Randles, M. (2023). Computer Vision Using Pose Estimation. Wasit Journal of Computer and Mathematics Science. 2. 85-92. 10.31185/wjcm.111.

[7] Zhou Z, Zhang H, Liu K, Ma F, Lu S, Zhou J, Ma L. Design of a Two-Dimensional Conveyor Platform with Cargo Pose Recognition and Adjustment Capabilities. Sensors. 2023; 23(21):8754. https://doi.org/10.3390/s23218754Hameed, Firas & Alwan, Hassan & Atiyah, Qasim. (2020). Pose Estimation of Objects Using Digital Image Processing for Pick-and-Place Applications of Robotic Arms. Engineering and Technology Journal. 38. 707-718. 10.30684/etj.v38i5A.518.

[8] Chinellato E, Grzyb BJ, del Pobil AP. Pose estimation through cue integration: a neuroscience-inspired approach. IEEE Trans Syst Man Cybern B Cybern. 2012 Apr;42(2):530-8. doi: 10.1109/TSMCB.2011.2168952. Epub 2011 Oct 20. PMID: 22027389.

[9] Hameed, Firas & Alwan, Hassan & Atiyah, Qasim. (2020). Pose Estimation of Objects Using Digital Image Processing for Pick-and-Place Applications of Robotic Arms. Engineering and Technology Journal. 38. 707-718. 10.30684/etj.v38i5A.518.

[10] Muñoz, Luis & González-López, Samuel & Robles, Jesús & Valencia, Gabriel. (2019). Diseño de interfaces de inspección de calidad utilizando redes neuronales y visión artificial para la industria de manufactura. Research in Computing Science. 148. 121-131. 10.13053/rcs-148-8-9.

[11] Rocco, Ignacio & Arandjelović, Relja & Sivic, Josef. (2017). Convolutional neural network architecture for geometric matching.

[12] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, “Learning fine-grained image similarity with deep ranking,” in Proc. of Computer Vision and Pattern Recognition, 2014, pp. 1386–1393.

[13] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, “Learning fine-grained image similarity with deep ranking,” in Proc. of Computer Vision and Pattern Recognition, 2014, pp. 1386–1393.

[14] K. Ogawara and K. Iseki, "Estimation of object class and orientation from multiple viewpoints and relative camera orientation constraints," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 2020, pp. 1-7, doi: 10.1109/IROS45743.2020.9340771.

[15] Y. Chen, W. Gong, C. Chen and W. Li, "Learning Orientation-Estimation Convolutional Neural Network for Building Detection in Optical Remote Sensing Image," 2018 Digital Image Computing: Techniques and Applications (DICTA), Canberra, ACT, Australia, 2018, pp. 1-8, doi: 10.1109/DICTA.2018.8615859.

[16] M. Awrangjeb, “Using point cloud data to identify, trace, and regularize the outlines of buildings,” International Journal of Remote Sensing, vol. 37, no. 3, pp. 551–579, Feb. 2016, http://www.tandfonline.com/doi/full/10.1080/01431161.2015.1131868.

[17] Y. Wen, H. Pan, L. Yang and W. Wang, "Edge Enhanced Implicit Orientation Learning with Geometric Prior for 6D Pose Estimation," in IEEE Robotics and Automation Letters, vol. 5, no. 3, pp. 4931-4938, July 2020, doi: 10.1109/LRA.2020.3005121.

[18] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel, “Implicit 3d orientation learning for 6d object detection from rgb images,” in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 699–715.

[19] M. Sundermeyer, Z.-C. Marton, M. Durner, and R. Triebel, “Augmented autoencoders: Implicit 3d orientation learning for 6d object detection,” International Journal of Computer Vision, pp. 1–16, 2019.

[20] J. Wu et al., "Real-Time Object Pose Estimation with Pose Interpreter Networks," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 2018, pp. 6798-6805, doi: 10.1109/IROS.2018.8593662.

[21] J. Josifovski, M. Kerzel, C. Pregizer, L. Posniak and S. Wermter, "Object Detection and Pose Estimation Based on Convolutional Neural Networks Trained with Synthetic Data," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 2018, pp. 6269-6276, doi: 10.1109/IROS.2018.8594379.

[22] H. Su, C. R. Qi, Y. Li, and L. J. Guibas, “Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3D model views,” Proceedings of the IEEE International Conference on Computer Vision, vol. 11-18-Dece, pp. 2686–2694, 2016.

[23] Soltan S, Oleinikov A, Demirci MF, Shintemirov A. Deep Learning-Based Object Classification and Position Estimation Pipeline for Potential Use in Robotized Pick-and-Place Operations. Robotics. 2020; 9(3):63. https://doi.org/10.3390/robotics9030063.

[24] Siami M, Barszcz T, Wodecki J, Zimroz R. Automated Identification of Overheated Belt Conveyor Idlers in Thermal Images with Complex Backgrounds Using Binary Classification with CNN. Sensors (Basel). 2022 Dec 19;22(24):10004. doi: 10.3390/s222410004. PMID: 36560373; PMCID: PMC9785391.

[25] Legrand, Antoine & Detry, Renaud & Vleeschouwer, Christophe. (2023). End-to-end Neural Estimation of Spacecraft Pose with Intermediate Detection of Keypoints. 10.1007/978-3-031-25056-9_11.

[26] DeTone, D., Malisiewicz, T., & Rabinovich, A. (2016). Deep Image Homography Estimation. ArXiv, abs/1606.03798.

[27] Fan, J., Zhu, Y., He, Y., Sun, Q., Liu, H., & He, J. (2021). Deep Learning on Monocular Object Pose Detection and Tracking: A Comprehensive Overview. ACM Computing Surveys, 55, 1 - 40.

[28] Qt Group. Qt Modules. [En línea]. Disponible en: https://doc.qt.io/qt-5/qtmodules.html. Accedido: ene. 28, 2024.

[29] Qt Documentation. Qt Quick 3D. [En línea]. Disponible en: https://doc.qt.io/qt-5/qtquick3d-index.html. Accedido: ene. 28, 2024.

[30] Ogre. ApplicationContextQt Class Reference. [En línea]. Disponible en: https://ogrecave.github.io/ogre/api/latest/class_ogre_bites_1_1_ application_context_qt. html. Accedido: ene. 28, 2024.

[31] Ogre3D Forums. Qt Ogre - A new approach. [En línea]. Disponible en: Enlace. Accedido: ene. 28, 2024.

[32] Ogre Wiki. Integrating Ogre into QT5. [En línea]. Disponible en: https://wiki.ogre3d.org/Integrating+Ogre+into+QT5. Accedido: ene. 28, 2024.

[33] Unlimited3D. QtQuick/QML in a Growing Project. [En línea]. Disponible en: https://unlimited3d.wordpress.com/qtquickqml-in-a-growing-project/. Accedido: ene. 28, 2024.

[34] GitHub - fougue/mayo. [En línea]. Disponible en: https://github.com/fougue/mayo. Accedido: ene. 28, 2024.

[35] Zaccone, Giancarlo & Karim, Rezaul. (2018). Deep Learning with TensorFlow - Second Edition.

[36] C. Wayne Brown, Learn WebGL (2015). Acknowledgements - References. [En línea]. Disponible en: https://learnwebgl.brown37.net/08_projections/projections_ introduction.html. Accedido: ene. 28, 2024.

[37] Gonzalez Vivo, P., & Lowe, J. (2015). The Book of Shaders. [En línea]. Disponible en: https://thebookofshaders.com/. Accedido: ene. 28, 2024.

[38] Basler AG. (2023). Software Downloads. [En línea]. Disponible en: https://www2.baslerweb.com/en/downloads/software-downloads/.

[39] Birchfield, S. (2016). Image Processing and Analysis. Cengage Learning. ISBN: 9781337515627.

[40] Sonka, M., Hlavac, V., & Boyle, R. (2014). Image Processing, Analysis, and Machine Vision. Cengage Learning. ISBN: 9781285981444.

[41] Tomasi, C., & Manduchi, R. (1998). Bilateral filtering for gray and color images. Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271), 839-846.

[42] Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on, volume 2, pages 28–31. IEEE, 2004.

[43] Zivkovic, Z., & van der Heijden, F. (2006). Efficient adaptive density estimation per image pixel for the task of background subtraction. Pattern Recognition Letters, 27(7), 773-780. https://doi.org/10.1016/j.patrec.2005.11.005.

[44] Sossa-Azuela, J.H., Santiago-Montero, R., Pérez-Cisneros, M., & Rubio-Espino, E. (2013). Computing the Euler Number of a Binary Image Based on a Vertex Codification. Journal of Applied Research and Technology, 11(3), 360-370. https://doi.org/10.1016/S1665-6423(13)71546-3.

[45] Alt, F. L. (1962). Digital pattern recognition by moments. Journal of the ACM (JACM), 9(2), 240-258.

[46] Rafael C. Gonzalez and Richard E. Woods. 2006. Digital Image Processing (3rd Edition). Prentice-Hall, Inc., USA.

[47] Zhou, W., A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. "Image Quality Assessment: From Error Visibility to Structural Similarity." IEEE Transactions on Image Processing. Vol. 13, Issue 4, April 2004, pp. 600–612.

[48] Dengsheng Zhang and Guojun Lu, "Generic Fourier descriptor for shape-based image retrieval," Proceedings. IEEE International Conference on Multimedia and Expo, Lausanne, Switzerland, 2002, pp. 425-428 vol.1, doi: 10.1109/ICME.2002.1035809.

[49] Courant, R., & John, F. (1999). Introduction to Calculus and Analysis I (1ª ed.). Classics in Mathematics. Springer Berlin, Heidelberg. ISBN: 9783540650584.

[50] Pearson, K. A., Griffith, C. A., Zellem, R. T., Koskinen, T. T., & Roudier, G. M. (2019). Ground-based Spectroscopy of the Exoplanet XO-2b Using a Systematic Wavelength Calibration. The Astronomical Journal, 157(1), 21. http://stacks.iop.org/1538-3881/157/i=1/a=21.

[51] Mori, Greg & Belongie, Serge & Malik, Jitendra. (2005). Efficient shape matching using shape contexts. IEEE transactions on pattern analysis and machine intelligence. 27. 1832-7. 10.1109/TPAMI.2005.220.

[52] D. P. Huttenlocher, G. A. Klanderman and W. J. Rucklidge, "Comparing images using the Hausdorff distance," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 15, no. 9, pp. 850-863, Sept. 1993, doi: 10.1109/34.232073.

[53] Bayón Arnau, L., Grau Ribas, J. M., & Suárez Rodríguez, P. M. (2008). Ampliación de Cálculo (1st ed.). Textos Universitarios. Servicio de Publicaciones. ISBN: 9788483179239.

[54] G. D. Evangelidis and E. Z. Psarakis, "Parametric Image Alignment Using Enhanced Correlation Coefficient Maximization," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 10, pp. 1858-1865, Oct. 2008, doi: 10.1109/TPAMI.2008.113.

[55] O. Chum and J. Matas, "Matching with PROSAC - progressive sample consensus," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 220-226 vol. 1, doi: 10.1109/CVPR.2005.221.

[56] Hartley, R., Zisserman, A. (2004). Multiple View Geometry in Computer Vision. United Kingdom: Cambridge University Press. ISBN: 9781139449144.

[57] Antony, I. (2023). qtcsv. GitHub. Accedido: ene. 28, 2024. [En línea]. Disponible en: https://github.com/iamantony/qtcsv.

[58] Chan, T. F., Golub, G. H., & LeVeque, R. J. (1983). Algorithms for computing the sample variance: Analysis and recommendations. The American Statistician, 37(3), 242-247. JSTOR: 2683386.

[59] Flusser, J. (2000). On the independence of rotation moment invariants. Pattern Recognit., 33, 1405-1410.

[60] GitHub - OpenGL Mathematics. [En línea]. Disponible en: GitHub - g-truc/glm: OpenGL Mathematics (GLM). Accedido: ene. 28, 2024.

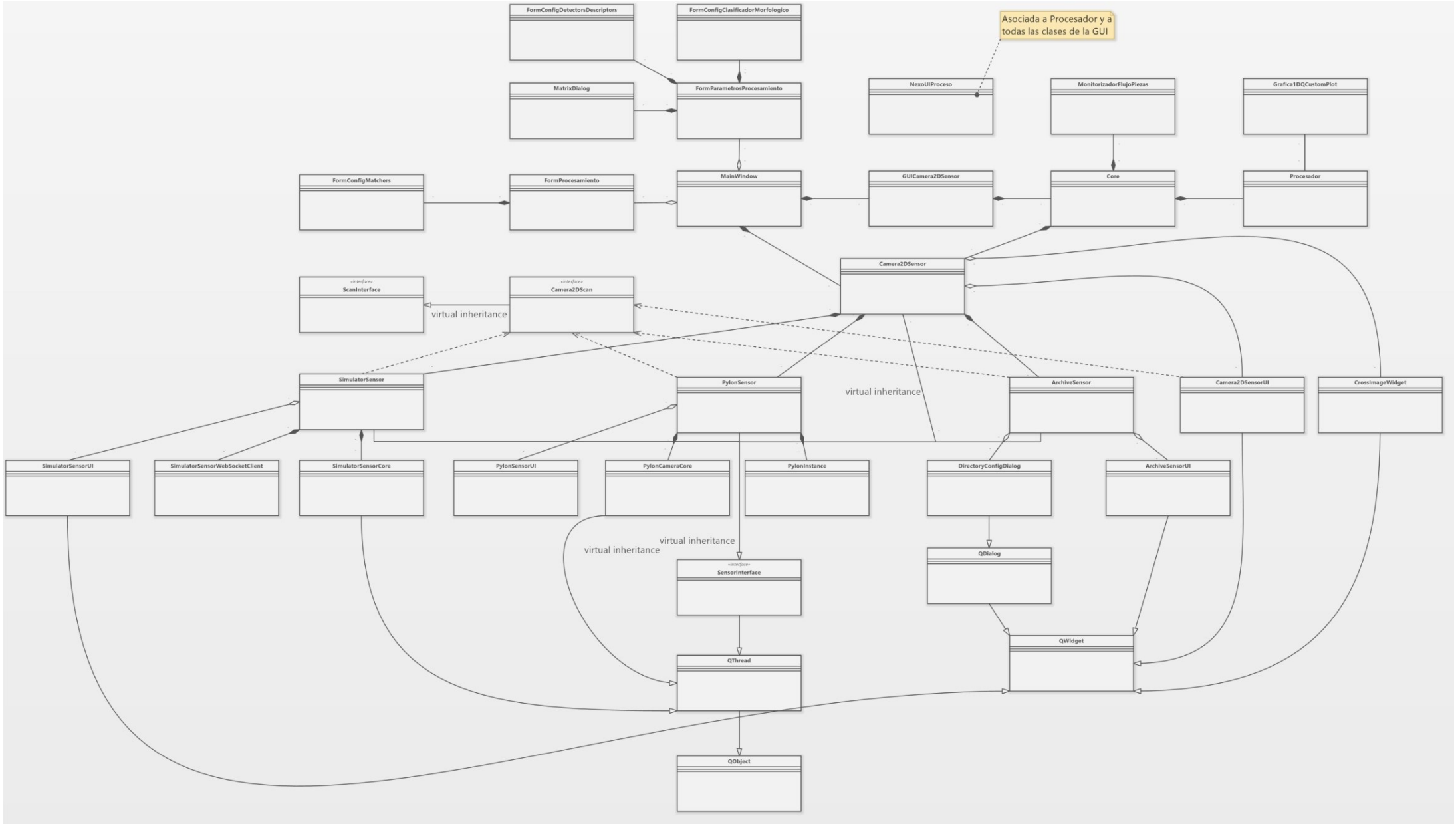
ANEXO I: Diagramas UML

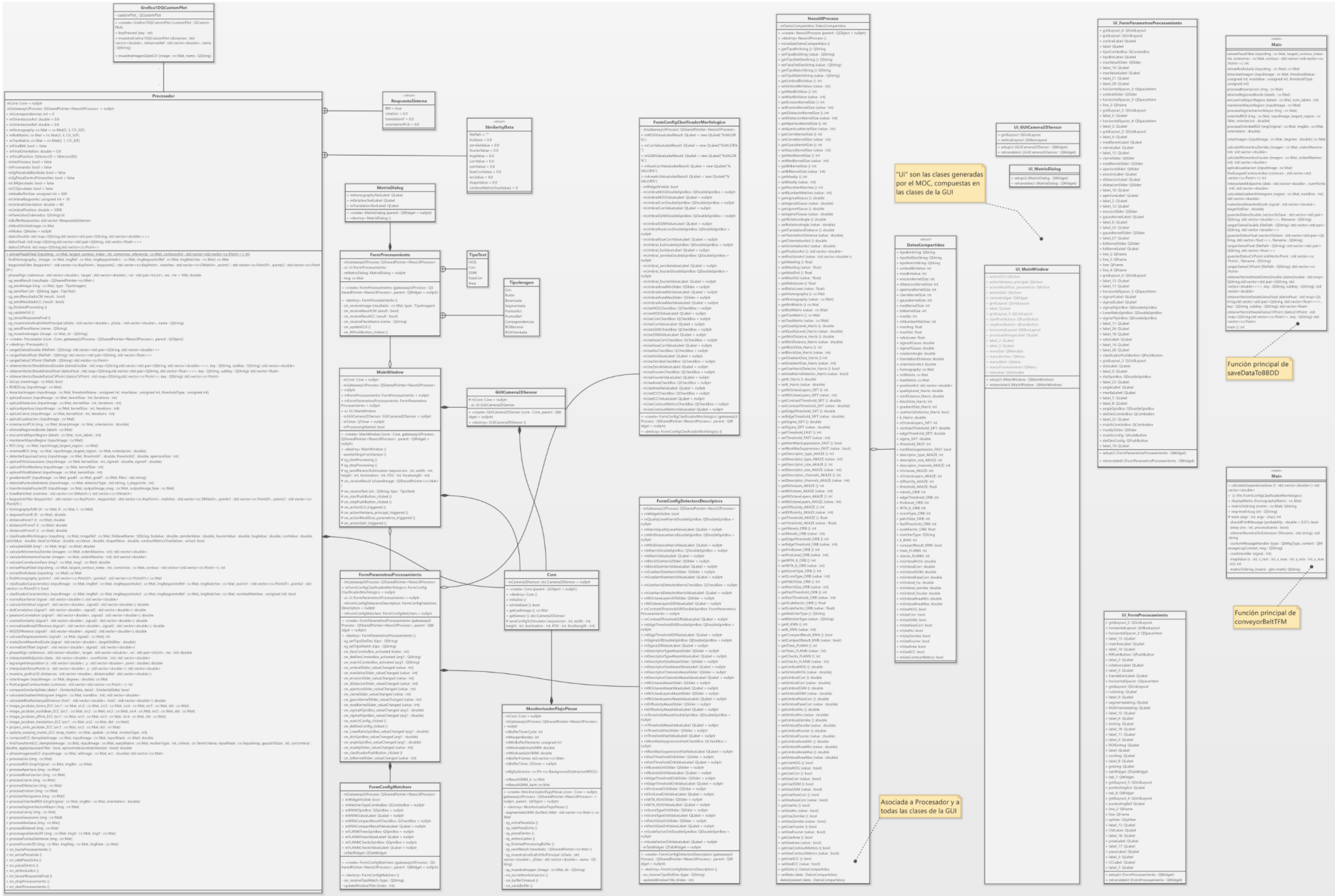
En este anexo se presenta un pequeño conjunto de diagramas UML (lenguaje unificado de modelado) ilustrativos que resumen los atributos y métodos de las clases más importantes del código fuente implementado en este trabajo, en todas sus partes, así como los distintos tipos de relación entre cada una de ellas, a fin de representar el concepto del software del sistema de manera más visual.

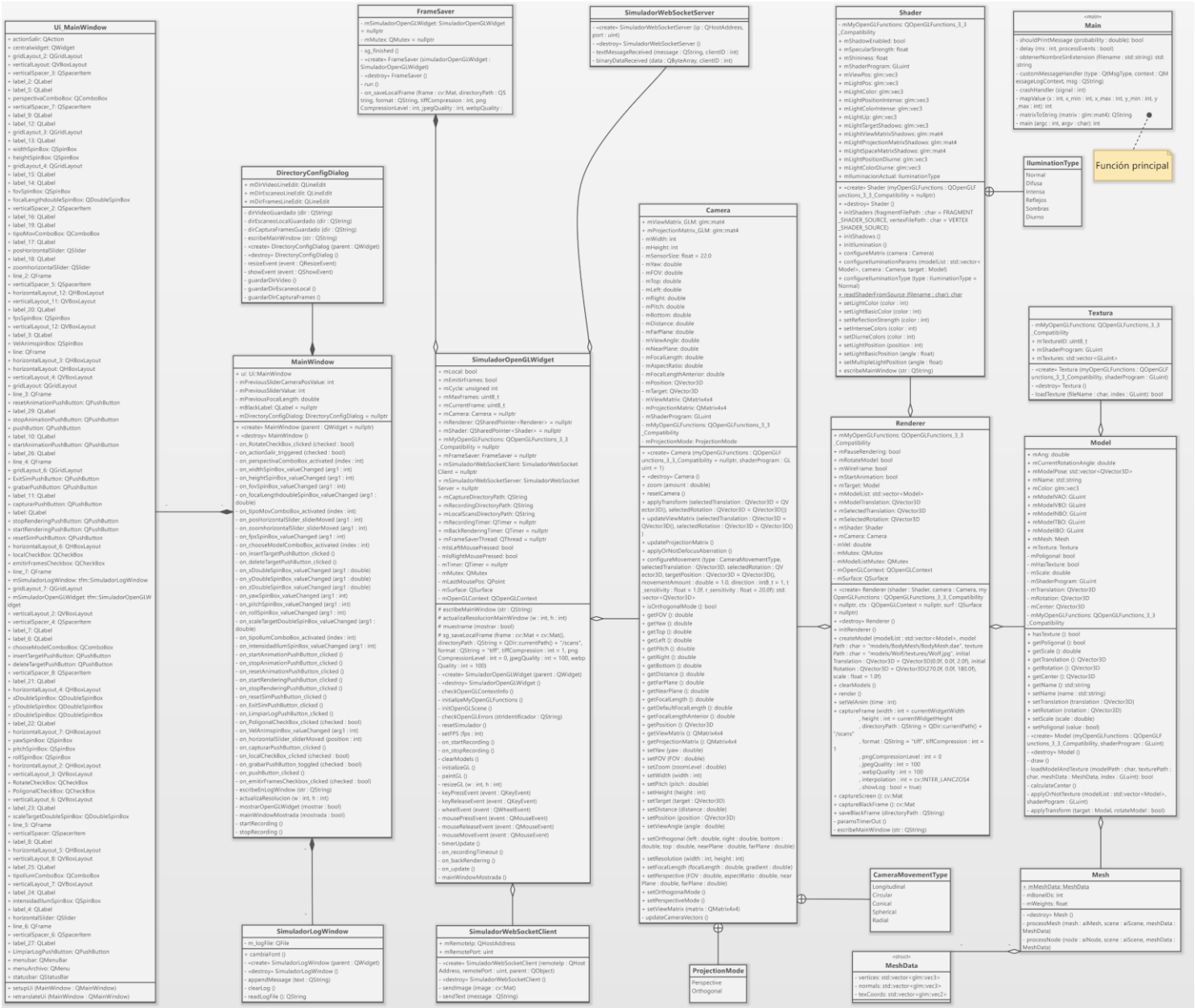
Se ha tratado de conservar la máxima resolución posible en las siguientes imágenes, pero a través del enlace mostrado a continuación se puede acceder al formato original de las mismas, para verlas en alta fidelidad:

- Enlace al primer anexo:

https://unioviedo-my.sharepoint.com/:f:/g/personal/uo269564_uniovi_es/Ev08FI2YH-pIk7_BJx-cXxsBQ5QTKpsy5eFyI5SsVFK3qA?e=TfkxAs


FormConfigDetectorsDescriptors
FormConfigClasificadorMorfologico
Asociada a Procesador y a todas las clases de la GUI
MatrixDialog
FormParametrosProcesamiento
NexoUIProceso
MonitorizadorFlujoPiezas
Grafica1DQCustomPlot
FormConfigMatchers
FormProcesamiento
MainWindow
GUICamera2DSensor
Core
Procesador
Camera2DSensor
«interface»
ScanInterface
«interface»
Camera2DScan
virtual inheritance
SimulatorSensor
PylonSensor
virtual inheritance
ArchiveSensor
Camera2DSensorUI
CrossImageWidget
SimulatorSensorUI
SimulatorSensorWebSocketClient
SimulatorSensorCore
PylonSensorUI
PylonCameraCore
PylonInstance
DirectoryConfigDialog
ArchiveSensorUI
virtual inheritance
virtual inheritance
QDialog
«interface»
SensorInterface
QWidget
QThread
QObject


"Ui" son las clases generadas por el MOC, compuestas en las clases de la GUI
Asociada a Procesador y a todas las clases de la GUI
Función principal de saveDataToBBDD
Función principal de conveyorBeltTFM


Ui_MainWindow
FrameSaver
SimuladorWebSocketServer
Shader
Main
DirectoryConfigDialog
IluminationType
Función principal
MainWindow
SimuladorOpenGLWidget
Camera
Textura
Renderer
Model
SimuladorLogWindow
SimuladorWebSocketClient
CameraMovementType
Mesh
ProjectionMode
MeshData

ANEXO II: Doxygen del simulador

Se ha documentado el código fuente del simulador realizado mediante la herramienta Doxygen, la cual se puede generar además, de manera automática, un árbol de directorios que contiene numerosos diagramas e hipervínculos en formato HTML para navegar cómodamente por la aplicación. La documentación está disponible a través del siguiente enlace y para acceder a ella basta con descargarla y abrir cualquier archivo en formato ".html" dentro suyo:

- Enlace al segundo anexo:

https://unioviedo-my.sharepoint.com/:f:/g/personal/uo269564_uniovi_es/ErJtfwjV7u1KqGienPXCvCIB1fDEn3NYZ7oVFbJfixFg1g?e=lgADuD

ANEXO III:
Vídeos demostrativos

Se han realizado varios vídeos breves que reúnen algunas partes del trabajo en funcionamiento; están disponibles a través del siguiente enlace:

- Enlace al tercer anexo:

https://unioviedo-my.sharepoint.com/:f:/g/personal/uo269564_uniovi_es/Eo8UPdYdOcNLocjVv-mlJFoBldL8um3cdOscaROQdrh-JQ?e=IYnj94

ANEXO IV:
Código fuente

El código fuente que se proporciona no es la totalidad de lo necesario (librerías auxiliares, dependencias, paquetes…) para compilar y ejecutar el sistema en su conjunto pero sí contiene la mayor parte de las clases y entidades de los tres bloques principales realizados: el simulador al completo, el programa de procesamiento (algoritmo) al completo y la interfaz para cámaras 2D en general explicada en la memoria. También se proporcionan varios scripts utilizados en el transcurso del proyecto y el archivo Excel que se empleó para realizar el análisis de los resultados presentado. Todo el código está disponible en un archivo comprimido a través del siguiente enlace:

- Enlace al cuarto anexo:

https://unioviedo-my.sharepoint.com/:f:/g/personal/uo269564_uniovi_es/Et0QLprB_whEjhKvoOH6LyoB0oO___x9Mt2OwEJDxpuwVg?e=k1PyNS