Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

Aser Crespo Rodríguez

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

Sistema de detección y estimación de pose de objetos basado en visión por computador para Planta Piloto Industria 4.0

JULIO 2019

MEMORIA

Índice

Índice de figuras

Índice de tablas

1 Introducción

El interés en productos basados en nuevas tecnologías como la Visión Artificial, la Realidad Aumentada o la Realidad Virtual ha crecido considerablemente en los últimos años. Particularmente, la Visión Artificial se utiliza hoy en día para dar solución a una gran variedad de problemas en diferentes ámbitos y con distintas aplicaciones (por ejemplo, seguridad, análisis de calidad, inteligencia de negocio, educación, salud, etc.), siendo además una de las tecnologías clave en la evolución de los entornos industriales hacia lo que se conoce como Industria 4.0.

El nuevo concepto de fábrica inteligente 4.0 integra una nueva manera de organizar los medios de producción con el objetivo de incrementar la adaptabilidad a las necesidades y procesos de fabricación, así como optimizar la asignación de recursos, abriendo la vía a lo que se considera la cuarta revolución industrial. Este objetivo se pretende conseguir integrando tecnologías digitales que permitan no sólo la automatización de los procesos industriales y la conectividad de todos los sistemas, si no la explotación de la información recogida. En este contexto, la Visión Artificial juega un papel clave tanto en los procesos de recogida e interpretación de la información como en la visualización de los datos, y resulta absolutamente indispensable para alcanzar la autonomía en robótica.

No obstante, el desarrollo y la aplicación de técnicas de Visión Artificial resultan costosos, especialmente a medida que los requisitos se vuelven más complejos, ya se trate del procesamiento de un gran volumen de imágenes, la urgencia de obtener resultados en tiempo real, o bien la exigencia de obtención de tasas de error mínimas; requisitos, por lo general, muy deseables en el ámbito de los procesos industriales.

Antes de comenzar con la descripción del proyecto es necesario hacer mención a La Fundación CTIC, ya que es el marco donde el alumno ha realizado sus prácticas de máster. Con este Trabajo Fin de Máster se pretende exponer los conocimientos adquiridos por el alumno en materia de detección de objetos 3D y, de forma más específica, del problema de *bin picking*. Para ello, se describirá el desarrollo de un sistema similar al que ha realizado en CTIC durante sus prácticas. No obstante, para evitar problemas de confidencialidad, se ha optado por adaptar el sistema a otro caso de uso, siendo este la detección de spinners dentro de la planta piloto de la Universidad de Oviedo.

1.1 La Fundación CTIC

La Fundación CTIC [1], Centro Tecnológico para el Desarrollo en Asturias de las Tecnologías de la Información y de la Comunicación, es una entidad privada, sin ánimo de lucro, dedicada a fines de interés general y de carácter permanente. El objetivo de CTIC es promover actividades relacionadas con el desarrollo de las Tecnologías de la Información y la Comunicación (TIC) en ámbitos económicos y sociales que puedan contribuir al desarrollo de la Sociedad. Las funciones que realiza CTIC engloban:

- Investigación
- Implantación
- Transferencia tecnológica
- Gestión y desarrollo de proyectos tecnológicos
- Difusión
- Dinamización
- Promoción
- Formación dirigidos a los ámbitos señalados

Se realiza por lo tanto una actividad encaminada al progreso social, empresarial, científico y tecnológico, todo ello en un entorno de trabajo de compromiso con la sociedad asturiana, basando sus actividades en complementar las actividades que realizan las empresas regionales.

CTIC cuenta con una serie de Líneas de especialización dedicadas a diferentes tecnologías, como Análisis de Datos, Internet de las Cosas o Tecnologías Emergentes. Durante las prácticas de empresa, el alumno ha sido asignado a la Línea de investigación en Tecnologías de Visión, en la cual se trabajan, principalmente, los campos de la Visión Artificial, la Realidad Virtual y la Realidad Aumentada.

Este departamento lo forman cinco personas con especializaciones en distintas materias: visión artificial, desarrollo de videojuegos, telecomunicaciones e informática. Así, la Línea de Visión está capacitada para llevar a cabo estudios, diseños e implantaciones de proyectos completos, utilizando tecnologías punteras puestas a su disposición por el centro.

1.1.1 Proyecto principal: sistema de *bin picking.*

En el marco de su colaboración en la Línea de Visión durante la realización de las prácticas de empresa en CTIC, el alumno participa en un proyecto orientado a la automatización del proceso de desbarbado y acabado superficial de piezas en una planta industrial. Se busca desarrollar e implantar un sistema de Visión Artificial para detectar piezas inicialmente distribuidas de forma aleatoria dentro de un contenedor. Un brazo robótico las cogería una a una y, en colaboración con otras herramientas, realizaría las operaciones ya citadas de desbarbado y acabado, para después colocarla con una posición y orientación previamente establecidas dentro de la línea de producción.

La automatización de este proceso conlleva la colaboración entre la empresa desarrolladora y la contratante, siendo la tarea de CTIC la de detectar la posición y orientación de las piezas dentro del contenedor mediante el sistema de Visión Artificial y poner esta información a disposición del brazo robótico.

Por lo tanto, el objetivo del proyecto ha sido el desarrollo de un sistema basado en Visión Artificial para la detección de objetos, de los que se dispone de su modelo 3D, y la estimación de su pose, es decir, su posición y orientación respecto a un sistema de coordenadas predeterminado. Este sistema ha sido implementado, proporcionando una interfaz para que se pueda interaccionar con el mismo desde otros módulos de una planta automatizada.

Los requisitos principales de este sistema son:

- Estimación de la pose de la pieza con un error lo suficientemente bajo para que el brazo robótico pueda actuar correctamente con las piezas detectadas.
- Funcionar en condiciones ambientales adversas, pues el brazo se encontrará en un ambiente cargado de partículas en suspensión y vapores.
- Fácilmente ampliable para detectar nuevas piezas, pudiendo ser introducidas por operarios que no tengan conocimiento extenso en el área de Visión Artificial.
- Funcionar con un único tipo de objeto por contenedor, nunca se encontrarán piezas distintas dentro del mismo.

1.2 Planta piloto de la Universidad de Oviedo

Desde la Universidad de Oviedo, y en concreto el Área de Ingeniería de Sistemas y Automática (ISA), se está llevando a cabo un proyecto para el diseño y construcción de una planta piloto para la fabricación de spinners con el objetivo de usarla como base para el desarrollo y la demostración de aplicaciones alrededor de los conceptos previamente mencionados de la industria 4.0. Esta

plataforma, desde la que es posible crear nuevos módulos de funcionamiento, genera una gran cantidad de posibilidades para el alumnado, que puede desarrollar sus trabajos alrededor de un caso práctico real.

Para el paso de ciertos módulos a otros dentro de la planta de fabricación, se ha planteado la opción de incluir un brazo robótico. Actualmente, la intención es colocar la pieza en un lugar preestablecido de forma que el movimiento del brazo sea siempre hacia el mismo punto. No obstante, en búsqueda de una mayor flexibilidad, se propone diseñar un módulo que, a partir de datos obtenidos mediante sensores 2D y/o 3D, sea capaz de detectar las piezas y proporcionar su posición y orientación al agente que tenga que interaccionar posteriormente con ellas. Este módulo permitiría la introducción de un brazo robótico en potencialmente cualquier punto de la línea de fabricación, gracias a la capacidad de reconocer las posición y orientación de los spinners en la escena de forma automática y sin ser necesaria una adaptación compleja del proceso productivo. Además, un módulo como este también podría ser útil para otras líneas de producción diferente a la planta piloto de spinners.

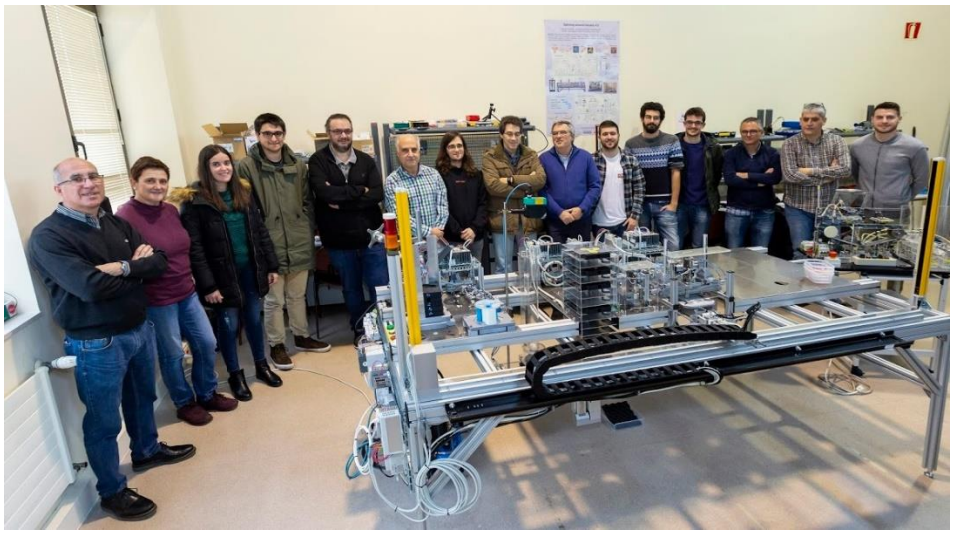


Figura 1. Planta Piloto Industria 4.0 de la Universidad de Oviedo [2]

1.3 Objetivos

El objetivo principal de este documento es realizar el diseño de un sistema basado en Visión Artificial para la detección de objetos de los que se dispone su modelo 3D y la estimación de su posición y orientación en la escena. Se estudiarán soluciones existentes en la actualidad y se realizará un análisis crítico de la funcionalidad y la eficacia del sistema propuesto en relación a estas. El marco teórico se acompañará de una serie de pruebas de análisis y validación prácticas con escenas reales, realizando una implementación del sistema a modo de prototipo, aunque se tendrán en cuenta técnicas eficientes de desarrollo software en el diseño de la aplicación.

No forma parte de los objetivos del proyecto controlar un brazo robótico para que realice la recogida de objetos en el sistema de *bin picking*. El sistema de detección está diseñado con la idea de proporcionar datos a un agente que interaccione con las piezas, pero en este proyecto sólo se desarrollará el módulo que proporcione dicha funcionalidad.

Además, y ya que resulta interesante poder reproducir este sistema en otros laboratorios, se diseñará la solución para que sea lo más modular posible, de forma que puedan emplearse con facilidad sensores y soportes físicos distintos a los empleados en este proyecto durante la fase de experimentación. Por este mismo motivo, además, se favorecerá el diseño de una solución que emplee equipos de bajo coste.

1.4 Condiciones de funcionamiento

Tendremos un número indeterminado de spinners distribuidos aleatoriamente en el interior de un contenedor de dimensiones y forma conocida. Como una de las restricciones o características del sistema, se asegura que en un contenedor sólo se tendrá un único tipo de pieza, por lo que realizar una clasificación de objetos no será necesario. Al encontrarse aleatoriamente distribuidos y apilados, los objetos estarán sujetos a mucha oclusión, factor que habrá que tener en cuenta a la hora de escoger un algoritmo de detección apropiado.

El diseño del sistema se realizará con la intención de que funcione lo más independientemente posible de las condiciones ambientales. La iluminación, aunque pueda ser ligeramente controlada por el usuario, no debe ser un factor determinante para el correcto funcionamiento de la solución. El sensor deberá ser calibrado para un entorno de trabajo concreto, pero no se empleará ninguna técnica de iluminación específica (como iluminación trasera o eliminación de la luz ambiental con una cámara oscura), teniendo la flexibilidad y simplicidad en mente.

Con el sistema propuesto en este proyecto se detectarán spinners como los que podemos ver en la Figura 2. Es importante destacar las características principales de estas piezas y lo que suponen en cuanto a su detección:

- Es una forma geométrica muy plana y delgada. Será necesario trabajar con un sensor que sea capaz de detectar la superficie del spinner y diferenciarlo del plano sobre el que esté situado. La resolución ofrecida por el sensor en la detección de la profundidad será clave, y se debe limitar la presencia de ruido en las medidas. No obstante, esta geometría propiciará que los spinners, generalmente, se encuentren apoyados en una de sus caras planas, ofreciendo la del lado contrario hacia el sensor. Esto facilitará la detección, pues esta es su área más característica.
- La superficie de los spinners no presenta ningún tipo de textura, son de un único color homogéneo. Esto implica un gradiente de color nulo en su superficie, por lo que no contaremos con puntos característicos que se puedan usar para establecer correspondencias. Esto tiene un gran impacto tanto en los sistemas de captura de datos (por ejemplo, un par estéreo tendrá problemas en reconstruir los puntos si no puede realizar correspondencias entre las dos imágenes) como en los de detección. Por este motivo, será necesario basarse en la información 3D y de superficie para definir los cuerpos; por ejemplo, mediante sus contornos.
- La alta simetría de los spinners respecto al plano transversal a su eje de giro causará problemas de ambigüedad a la hora de definir su pose, ya que vemos que un spinner resulta simétrico alrededor de este. Por otro lado, en el modelo representado en la Figura 3 se puede advertir que un volteo de 180º del spinner será muy difícil de identificar, siendo la única diferencia entre la cara superior y la inferior un pequeño apoyo para colocar el rodamiento, y posiblemente no podremos garantizar que el sistema vaya a poder discernir entre estas dos orientaciones.

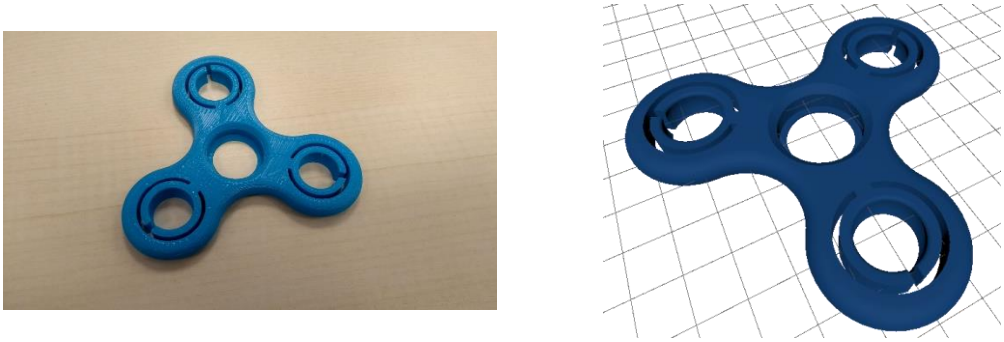


Figura 2. a) Spinner real. b) Modelo 3D de spinner.

1.5 Resumen solución propuesta

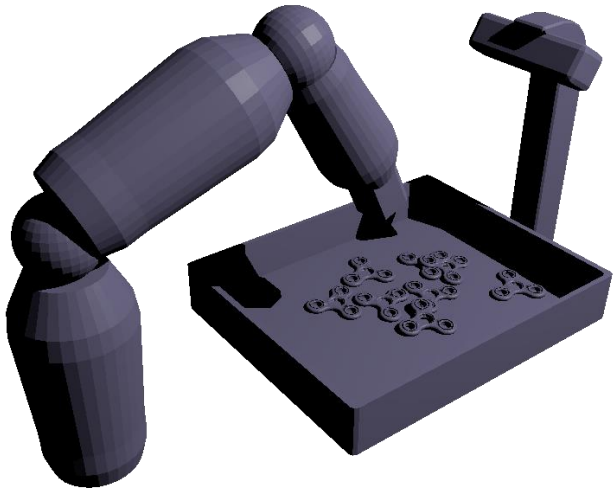


Figura 3. Representación física del sistema de *bin picking*

Para este proyecto se considerará una distribución de los componentes que conforman el sistema como se refleja en la Figura 3. En un espacio controlado, un contenedor de aproximadamente 200 x 300 cm, tendremos una serie de objetos a recoger por el brazo robótico, apilados y distribuidos aleatoriamente. El sistema de adquisición de datos se encontrará fijo a un soporte y permanecerá estático durante todo el proceso de vaciado, posición desde la que tendrá un campo de visión que abarque toda la escena y desde la que la escena no vaya a ser ocluida por otros elementos del sistema. Hay que tener en cuenta que el sensor también deberá estar colocado en un lugar donde resulte lo menos invasivo posible, pues trabajará en conjunción con un brazo robótico y no puede bloquear las rutas del movimiento de este. Por lo tanto, se debe estudiar una distribución espacial entre brazo/sensor de forma que ninguno de los dos subsistemas entorpezca el funcionamiento del otro.

Se propone un sistema para resolver el problema de *bin picking* con un proceso de ejecución similar al presentado en [3]. En este artículo se propone un sistema que analiza imágenes de color y profundidad tomadas desde distintos puntos de la escena, devolviendo la pose de los objetos detectados para su posterior manipulación. La parte más interesante de este proyecto parte de la separación clara entre algoritmos de detección de objetos y la estimación de su pose. Primero, el sistema analiza las imágenes RGB mediante una red convolutiva para detectar piezas, devolviendo una máscara sobre la escena por cada objeto detectado. Posteriormente, emplea esta información para aislar los puntos 3D pertenecientes a un objeto del fondo y de otros objetos. Cómo ya se tiene, entonces, los puntos 3D pertenecientes a una instancia del objeto, sólo es necesario realizar la estimación de la pose, pues la detección de este ya se ha realizado. El proceso está, por lo tanto, separado en dos etapas independientes que normalmente se encuentran integradas en otros algoritmos, como se verá en el apartado 2 de este documento.

La solución propuesta adopta esta filosofía para separar el proceso de ejecución en etapas claramente diferenciadas, como puede verse en el esquema general de la Figura 4.

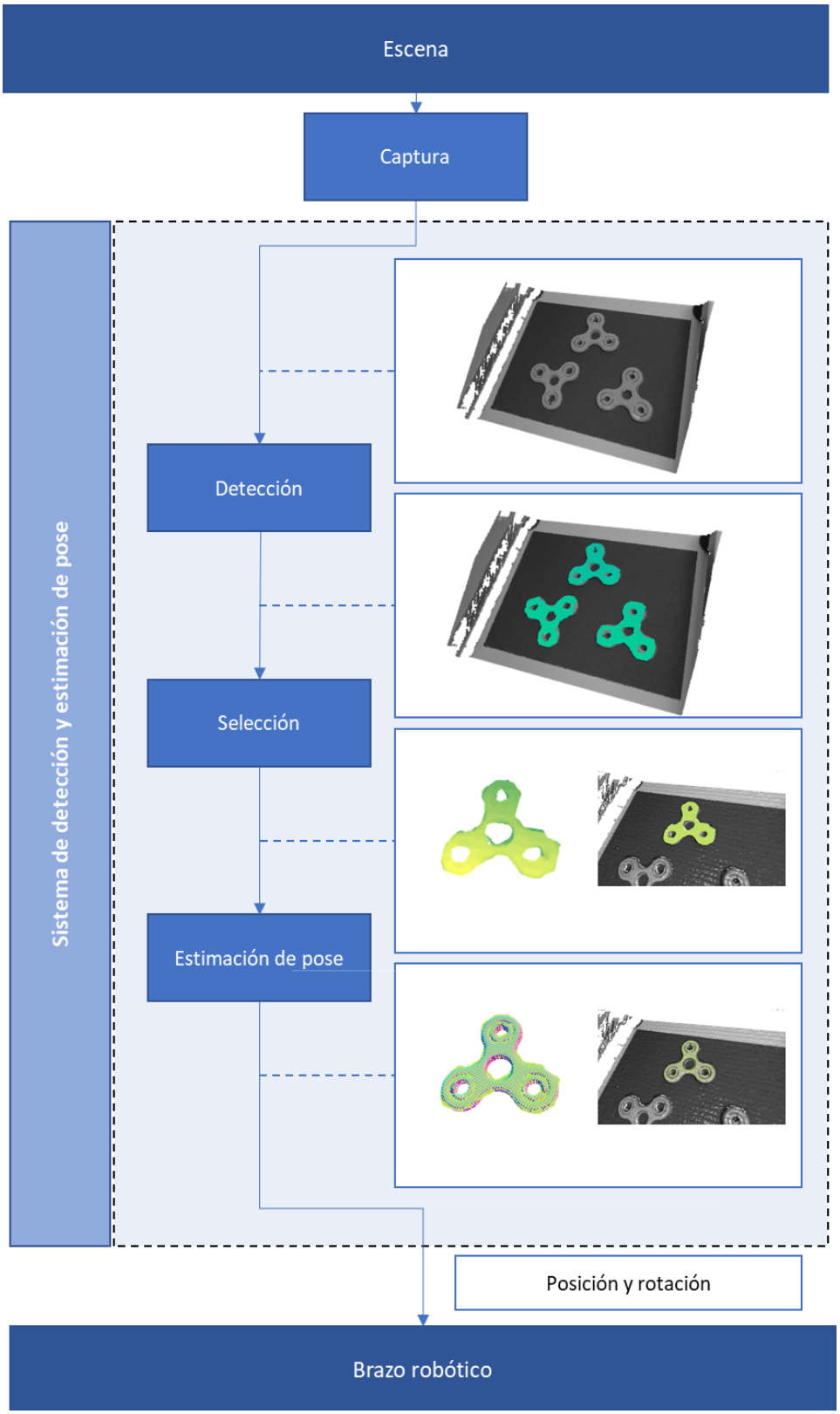


Figura 4. Esquema general sistema propuesto.

El funcionamiento básico del sistema parte de la obtención de imágenes RGB o monocromáticas de la escena al mismo tiempo que información 3D, representado como una nube de puntos que contiene información espacial y de nivel de brillo. La imagen monocromática o RGB será analizada mediante un detector de objetos, que detectará aquellas regiones en las que existan

puntos pertenecientes a un spinner. De este submódulo se obtendrán tantas máscaras como spinners haya en la nube de puntos. Sin embargo, el sistema debe escoger cuál es la pieza óptima para ser extraída por el brazo robótico, por lo que se realizará una selección buscando escoger aquella que se encuentre por encima de las demás. La máscara obtenida servirá para filtrar la nube de puntos de la escena y quedarse sólo con aquellos puntos que pertenezcan al spinner que vaya a ser extraído del contenedor. El siguiente paso es, consecuentemente, conocer la posición y la rotación de dicho spinner en la escena. Para ello, se pasa esta nube de puntos recortada por el siguiente submódulo del sistema, que realizará una estimación de la pose del objeto. Esta información será transmitida hacia el brazo robótico, que llevará su actuador final a la posición indicada y extraerá el spinner detectado. Una vez haya sido extraído, el sistema podrá comenzar nuevamente el ciclo para calcular la pose del siguiente spinner.

A lo largo de los siguientes apartados se profundizará en el funcionamiento de cada submódulo, escogiendo además entre distintas tecnologías disponibles para la realización de cada uno.

1.6 Submódulos del sistema

1.6.1 Captura

De la escena obtendremos, en primer lugar, la siguiente información:

- **Mapa de brillo**: imagen RGB (o en escala de grises, depende de la funcionalidad del sensor).
- **Mapa de puntos**: puntos 3D descritos en un sistema de referencia conocido.
- **Mapa de normales**: normales de la superficie en los puntos anteriores.

Es importante el concepto de mapa, remarcado en este proyecto voluntariamente, para este tipo de datos. En todo momento trabajaremos con datos obtenidos con un sensor que debe mantener una relación uno a uno de puntos en los tres campos: un punto de la escena tendrá un valor de brillo, unas coordenadas 3D y una normal asociada. Si no se dispone de algunos de esos datos (por ejemplo, no se ha podido calcular la posición 3D debido a oclusiones), se mantendrá el punto, aunque con un valor no válido en su lugar. Además, será necesario que la estructura sea bidimensional: que tenga alto y ancho. Este es el formato habitual de representación de imágenes, aunque no para las nubes de puntos. Sin embargo, se establece esta condición para asegurar siempre el mismo número de datos, pero, sobre todo, porque esta relación uno a uno será necesaria para segmentar todos los mapas en función de una máscara obtenida a partir de uno de ellos.

El sistema presentado en este proyecto funcionará independientemente de la técnica empleada para la captura de datos, siempre y cuando se mantengan estas convenciones. En este caso se empleará una cámara estéreo, que proporcionará información de profundidad y de valores de brillo en escala de grises.

1.6.2 Detección

A la imagen de la escena (mapa de brillo) se le aplicará un detector que segmentará la imagen aislando los objetos de interés del fondo, obteniendo regiones de puntos que pertenecientes a un spinner determinado. De este módulo se obtendrán entonces regiones que definen los puntos pertenecientes a cada instancia de spinner que sea detectado en la imagen, teniendo tantas máscaras como spinners se hayan encontrado. Ya que la relación de puntos es uno a uno, tendremos toda la información necesaria para posteriores etapas asociadas a cada instancia: valor de brillo, nubes de puntos y sus vectores normales.

La técnica empleada para la detección se basará en *Deep Learning*, aplicando una cascada de redes neuronales convolucionales basadas en regiones (Mask-RCNN). En el apartado 3.3 se realizará una comparativa de tecnologías y modelos disponibles.

1.6.3 Selección

Posteriormente, un módulo escogerá de entre todas estas regiones la correspondiente al spinner que se estime como óptimo para ser extraído. Esta decisión será tomada en base a unos índices que dependen del área de la superficie que es vista por el sensor y de la altura a la que se encuentre el spinner. Por lo tanto, para este paso será necesario también acceder a la información de profundidad recogida por el sistema de captura.

Una vez sea estimada la región en la que se encuentra la próxima pieza a extraer del contenedor, se recortan los mapas de puntos y sus normales para obtener una nube de puntos que contenga únicamente la información perteneciente a dicha región (por este motivo es necesario que todos los puntos tuviesen una correspondencia uno a uno). Esta nube de puntos que representa, en principio, la instancia de la pieza en la escena óptima para su extracción será sobre la que se aplicará el estimador de pose con el modelo 3D previamente cargado.

1.6.4 Estimación de pose

El último módulo del sistema recuperará la posición y orientación del spinner seleccionado en la escena mediante una técnica de búsqueda de correspondencias entre puntos del modelo CAD y de la nube de puntos recortada de la escena. La búsqueda de correspondencias se realizará empleando unos descriptores específicos llamados *Point Pair Features*, cuyo funcionamiento se explicará en profundidad en el apartado 3.5.

Por lo tanto, teniendo como entrada a este submódulo las coordenadas 3D y sus normales en los puntos de la región seleccionada previamente, obtendremos una estimación de la pose expresada como una translación y rotación que ubicaría el modelo 3D del spinner en la nube de puntos.

En este proyecto se realizará una visualización de este resultado mediante la transformación de los puntos que describen el modelo 3D con estos valores de translación y rotación superpuesta sobre la nube de puntos de la escena. Sin embargo, si el sensor y el brazo robótico están calibrados respecto al mismo sistema de coordenadas, esta matriz de transformación indicaría al brazo robótico dónde se encuentra la pieza. Por lo tanto, estos valores que definen la pose del objeto deberán ser los que se comuniquen al agente que vaya a interaccionar con las piezas.

1.7 Entrenamiento de los submódulos del sistema

Como se ha comentado en el anterior apartado, por cada tipo de pieza a detectar serán necesarios, por lo tanto, dos modelos: uno para realizar su detección en una imagen RGB o en escala de grises (dependerá del tipo de sensor que tengamos disponible), y otro con descriptores del objeto en el espacio 3D para poder buscar correspondencias en la nube de puntos obtenida del sensor de profundidad. Estos modelos serán generados de forma *offline*, y serán exclusivos para cada pieza. De ser posible, serán serializados y guardados en una base de datos a la que poder acceder desde el sistema de detección.

Para generar un modelo de red neuronal convolutiva que detecte spinners en imágenes RGB o monocromáticas sería necesario proporcionar muchas imágenes para el entrenamiento, acompañadas además de su correcto etiquetado. Esto supone una gran cantidad de tiempo para introducir nuevos modelos en el sistema en el caso de realizar el etiquetado de forma manual, ya que, además, el entrenamiento de modelos basados en *Deep Learning* requiere inmensas cantidades de datos para obtener resultados adecuados.

Para solventar el problema derivado del laborioso proceso de generación de datos y su etiquetado para el entrenamiento de la red, se entrenará el modelo de un nuevo objeto mediante imágenes generadas sintéticamente. Estas imágenes serán producidas mediante el simulador de datos

presentado en este proyecto, descrito en profundidad en el apartado 3.2. A partir del modelo 3D del spinner, el simulador reproducirá escenas similares a las que puedan obtenerse en el caso de uso real, automatizando el proceso de etiquetado y ofreciendo una gran flexibilidad para la generación de bases de datos.

En lugar de generar el modelo basándose únicamente en imágenes sintéticas, se empleará una técnica de transferencia de aprendizaje, entrenando las últimas capas de una red preseleccionada mediante las imágenes generadas de los spinners.

2 Estado de la técnica

Como se ha comentado en el apartado anterior, la automatización de los procesos de producción industriales está cada vez más presente en las empresas actuales. Los métodos de *bin picking*, que detectan y extraen automáticamente las piezas situadas aleatoriamente dentro de un contenedor, proporcionan una mejora en la cadena productiva reduciendo el tiempo del proceso, aumentando la productividad y mejorando la calidad de vida laboral de los operarios.

Aunque a priori parezca una metodología trivial, resulta complejo para un sistema informático robotizado distinguir y separar una pieza de otras dispuestas y apiladas de forma aleatoria, debido sobre todo a las oclusiones y a la arbitrariedad de su pose en la escena. Además, el cálculo de la distancia a la que se encuentran las piezas de la cámara para determinar correctamente las coordenadas 3D supone una dificultad añadida y requiere el uso de tecnologías de alto coste computacional.

En la última década, los métodos, algoritmos y sistemas de percepción basados en información tridimensional han experimentado un importante auge revolucionando el desarrollo de aplicaciones robóticas, tales como la navegación, guiado y localización de robots o como la detección y reconocimiento de objetos para la interacción de un robot con su entorno en tareas de agarre y manipulación inteligente.

Este avance ha estado impulsado, principalmente, por dos factores: la aparición de nuevos sensores capaces de generar información 3D en tiempo real a un bajo coste, y el crecimiento (tanto en capacidad computacional como espacial) que ha sufrido el hardware destinado al procesamiento de este tipo de información (GPUs, o simples CPUs de ordenadores personales). Como consecuencia, en estos últimos años se ha percibido un salto cualitativo de las técnicas aplicadas para llevar a cabo estas tareas haciendo uso de datos 3D. Estas mejoras están presentes en una gran variedad de algoritmos y métodos, tales como extracción de características, segmentación, reconstrucción de modelos 3D de objetos y escenas (en tiempo real), o descripción de superficies, incluyendo además la aparición de nuevas técnicas de clasificación.

Un sistema de visión 3D, sea cual sea su finalidad, requiere de la utilización de uno o más sensores (no necesariamente 3D) así como de un módulo de procesamiento capaz de generar un mapa (imagen) de profundidad a partir del cual se obtiene la información necesaria para reconocer un objeto, realizar una reconstrucción o guiar un robot; en definitiva, para dotar de cierta semántica a los datos de entrada.

Desde la aparición de los primeros sistemas de reconocimiento de objetos 3D, uno de los principales objetivos ha sido detectar tipos de objeto y recuperar su posición y orientación en el escenario en el que se encuentran. En cualquier caso, el principal problema subyace en que la información proporcionada por los métodos de descripción de superficies viene delimitada por los puntos de vista del sensor, que pueden estar sometidos a las oclusiones producidas por la propia geometría del objeto o por otros objetos presentes en la misma escena. Cabe mencionar que otro problema particularmente importante en los sistemas de reconocimiento es la búsqueda de características invariantes que no dependan de la posición y orientación del objeto, del escalado y del ruido. Además, hay que tener en cuenta que, en sistemas de reconocimiento enfocados a tareas robóticas, es una prioridad conseguir una solución capaz de funcionar en tiempo real.

Los sistemas existentes en el mercado son soluciones que incluyen tanto el sistema de Visión para la detección de las piezas, como el brazo robot y la estructura soporte. La mayoría de ellos se realizan a medida para extraer una pieza determinada y no ofrecen la posibilidad de añadir piezas nuevas de forma sencilla. Además, no están adaptados para trabajar en entornos contaminados y

todos se basan en el uso de cámaras 3D para el reconocimiento de los objetos. A continuación, en la Tabla 1 se recogen ejemplos de soluciones comerciales para *bin picking*.

EMPRESA	CARACTERÍSTICAS TECNOLÓGICAS	LOCALIZACIÓN
Liebherr	Cámara 3D, brazo robot con pinzas, interfaz de usuario para introducir nuevas piezas y contenedores.	Alemania
bcnvision	Cámara 3D, solución a medida para una determinada pieza.	España
FANUC America Corporation	Cámara 3D, solución a medida para una determinada pieza.	EEUU
Scape technologies	Cámara 3D, solución a medida para una determinada pieza.	Dinamarca
VisionSystems Design	Cámara 3D, solución a medida para una determinada pieza.	EEUU
SCHUNK	Cámara 3D, 32 puntos diferentes para el agarre.	Alemania

Tabla 1. Soluciones de *bin picking* existentes en el mercado.

En este documento se propone una solución enfocada únicamente a la detección y estimación de pose de objetos. Como se explica en la sección 1.5, se realiza una división del proceso en diferentes módulos, y en los siguientes apartados se realizará un estudio individualizado de opciones que nos sirvan para resolver cada uno de ellos.

La estimación de la pose en 3D de un objeto es un problema en la que una solución rápida y eficaz resulta muy interesante para diversas aplicaciones, de modo que se puede encontrar una gran cantidad de bibliografía al respecto. No obstante, las arquitecturas de los sistemas existentes difieren entre ellas, por lo que en esta sección se revisarán las más relevantes en el contexto del proyecto propuesto.

2.1 Correspondencias de puntos característicos basadas en descriptores locales

Una de las estrategias más comunes para estimación de pose es la de extracción de descriptores locales de puntos característicos de la escena y la búsqueda de correspondencias con los de un modelo teórico. Aunque aplicado a nubes de puntos, este método se basa en un procedimiento similar al de detección de objetos en imágenes.

En líneas generales, esta técnica consiste primeramente en la extracción de una serie de puntos característicos, es decir, repetibles y distintivos. La repetibilidad viene dada en aquellos puntos que se escojan siempre en varias iteraciones y desde distintos puntos de vista. Que sean distintivos implica que sean diferenciables del resto, siendo normalmente diferencias atribuidas a características topológicas de los datos. Por ejemplo, que sea un punto con gradientes en múltiples direcciones y que dicha característica se produzca a una determinada escala.

El siguiente paso es la extracción de un descriptor para cada punto. Un descriptor define a un punto con una serie de características que puedan ser después comparadas con las de otros puntos, e identificándolo de la manera más inequívoca del resto. Existen muchos tipos de descriptores (ejemplos en la Tabla 2), y la elección del tipo dependerá del caso de uso del sistema. Emplear un *pipeline* de este tipo requerirá un estudio sobre la escena y los objetos que se quieran detectar en ella.

Una vez calculados estos puntos característicos y sus descriptores se comparan para buscar correspondencias con los que se tienen en la base de datos del modelo del objeto, calculada previamente. Ya que existe la posibilidad de que se puedan establecer parejas de puntos de distintos objetos, sólo se agrupan aquellas correspondencias que sean geométricamente consistentes. Por ejemplo, si se encuentran dos puntos en la escena que corresponden con dos

puntos del modelo, pero, sin embargo, las distancias entre ambos pares son muy diferentes, se descartará la opción de que estos puntos pertenezcan al objeto (suponiendo la restricción de que este es rígido).

Por último, para recuperar la pose del objeto en la escena, habrá que buscar la rotación y translación que mejor ajuste las correspondencias, utilizando por ejemplo un método tipo RANSAC [4].

También es posible emplear descriptores globales que definan al objeto en su totalidad, en lugar de definirlo con una serie de puntos con sus descriptores locales. Para ello, se segmentará la escena en grupos de puntos que pertenezcan a un mismo objeto (*clusters*), de los que se extraerán descriptores globales que serán posteriormente comparados con los del objeto.

Cabe destacar la existencia de *The Point Cloud Library* (PCL)[5] para el tratamiento de nubes de puntos. Es una librería de código abierto con implementaciones de este tipo de algoritmos, y ofrece mucha documentación al respecto, así como ejemplos de uso.

3D descriptors

Name	Type	Size†	Custom PointType††
PFH (Point Feature Histogram)	Local	125	Yes
FPFH (Fast Point Feature Histogram)	Local	33	Yes
RSD (Radius-Based Surface Descriptor)	Local	289	Yes
3DSC (3D Shape Context)	Local	1980	Yes
USC (Unique Shape Context)	Local	1960	Yes
SHOT (Signatures of Histograms of Orientations)	Local	352	Yes
Spin image	Local	153*	No
RIFT (Rotation-Invariant Feature Transform)	Local	32*	No
NARF (Normal Aligned Radial Feature)	Local	36	Yes
RoPS (Rotational Projection Statistics)	Local	135*	No
VFH (Viewpoint Feature Histogram)	Global	308	Yes
CVFH (Clustered Viewpoint Feature Histogram)	Global	308	Yes
OUR-CVFH (Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram)	Global	308	Yes
ESF (Ensemble of Shape Functions)	Global	640	Yes
GFPFH (Global Fast Point Feature Histogram)	Global	16	Yes
GRSD (Global Radius-Based Surface Descriptor)	Global	21	Yes

† Values marked with an asterisk (*) indicate that the descriptor's size depends on some parameter(s), and the one given is for the default values.

†† Descriptors without their own custom PointType use the generic "pcl::Histogram<>" type. See Saving and loading.

Tabla 2. [6] Algunos descriptores para puntos de nubes de puntos implementados en la librería PCL [5].

Uno de los tipos de descriptores que más se referencia en la bibliografía es el *Point Pair Feature* (PPF) [6], y distintas versiones del mismo han sido propuestas en aras de aumentar la robustez y disminuir los tiempos de cómputo, como en [7]. Este tipo de descriptor es el seleccionado para emplear en el sistema, y será descrito con mayor detalle en el apartado 3.5 de este trabajo. Concretamente, para este proyecto se utilizará una versión implementada en OpenCV [8].

2.2 Template matching

Otro de los sistemas con gran aceptación en este ámbito es el llamando LINE-MOD [9], desarrollado por Stefan Hinterstoisser [10], [11]. Este sistema emplea técnicas de *template matching* en base a información conjunta de gradientes de color y de información de la superficie 3D del objeto. La idea principal es la de generar una base de datos con patrones del elemento a detectar, siendo un patrón información recabada desde un punto de vista específico y distinto al del resto. Recorriendo la imagen de la escena y comparando con una medida de semejanza basada

en el gradiente de color y de las normales del objeto contra todos los patrones de la base de datos, se obtiene aquel que obtenga mejor resultado (aquel más similar). Ya que el patrón depende del punto de vista desde dónde se ha calculado, se obtiene directamente una estimación sobre la pose del objeto. El cómputo de este sistema es costoso, pero optimizado se puede utilizar como un sistema de *tiempo real.*

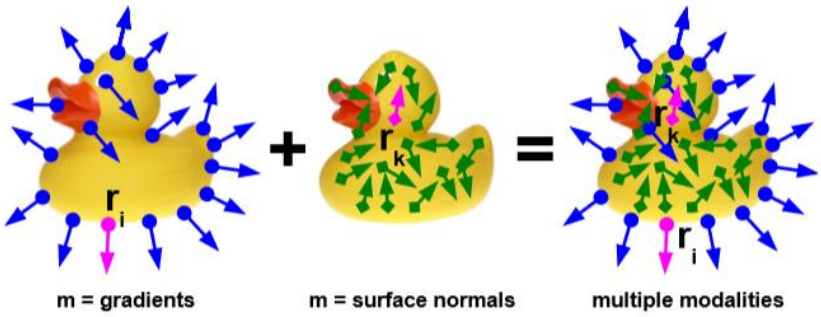


Figura 5. LINE-MOD: Gradientes de color y normales de superficie para definir un objeto [9]

2.3 Iterative Closest Point

Iterative Closest Point (ICP) [12] es un método extendidamente usado para minimizar la diferencia espacial entre dos nubes de puntos, buscando las transformación que hay que aplicar a una de ellas para que las dos se correspondan punto a punto con el menor error posible. Para ello, se aplican transformaciones sucesivas en una de ellas para minimizar una función de error métrico entre las dos, normalmente calculado a partir de las distancias relativas entre parejas de puntos. Estos emparejamientos de puntos suelen hacerse buscando intersecciones entre un punto de una nube con otro punto de la segunda en la dirección de su normal, aunque otros métodos como el uso de descriptores también pueden ser empleados.

Esta técnica es ampliamente usada por la rapidez con la que obtiene resultados y la precisión con la que los obtiene, aunque converge fácilmente a mínimos locales, pudiendo derivar en resultados incorrectos. Por ello, es muy común verlo usado como algoritmo de refinamiento una vez detectado el objeto y obtenida una estimación inicial de su pose. En este proyecto se empleará precisamente con este objetivo, el de proporcionar una estimación más exacta de la pose.

2.4 Deep Learning

Más recientemente, se han propuesto varias técnicas basadas en *Deep Learning* que pueden ser útiles para resolver el problema planteado. Por ejemplo, *PointNet* [13] es un segmentador/clasificador que consume directamente nubes de puntos desordenadas, siendo entrenado para detectar determinados objetos mediante su modelo CAD. Además, puede realizar una segmentación semántica de la escena; por ejemplo, no reconocer únicamente un tipo específico de silla, sino reconocer “sillas” habiendo entrenado con distintos modelos de estas.

PoseCNN [14] presenta una red neuronal convolucional que predice la pose 3D de un objeto mediante imágenes RGB de la escena, sin necesidad de emplear una cámara de profundidad (aunque es recomendable para refinar la pose estimada). Sin embargo, es necesario un amplio conjunto de imágenes y/o vídeos para poder entrenar esta red. Si bien hay métodos para hacer que el proceso de etiquetado de los datos esté más automatizado, tal y como se discute en el artículo, el proceso normalmente resulta muy laborioso y consume mucho tiempo. Como solución a este problema está la posibilidad de generar los datos de forma sintética [15]–[17]. En [18], por ejemplo, se presenta un detector y estimador de pose puramente basado en datos generados sintéticamente, empleando únicamente imágenes RGB para el proceso de detección. Este

proyecto hace uso de este concepto, empleando datos sintéticos para entrenar redes neuronales que sean posteriormente empleadas para detectar spinners en escenas reales.

En este proyecto no se emplearán técnicas basadas en *Deep Learning* para la estimación de la pose, no obstante, debido al nivel experimental en el que aún se encuentran. Sin embargo, se mencionan y son propuestas para ser valoradas en futuros trabajos.

2.5 Otros métodos

Otras soluciones están más relacionadas con la forma específica del propio objeto. En [19], por ejemplo, se descompone de la geometría de un objeto en figuras básicas, como cubos, esferas o planos. Estas aplicaciones pueden resultar muy robustas al contar con modelos matemáticos que definen de forma sencilla y paramétrica dichas figuras, y al existir muchas técnicas para detectarlas, siendo muy común usar algoritmos tipo RANSAC [4]. Son, sin embargo, sistemas poco flexibles para piezas con geometrías más complejas, además de que introducir nuevos objetos puede resultar costoso, por lo que se descarta su utilización en el marco de este proyecto.

Por otro lado, existen otras opciones como la presentada en [20] que busca zonas óptimas por dónde agarrar al siguiente objeto, sin importar la ubicación del objeto en sí. De esta forma, no se buscan instancias de objetos, sino que se buscan superficies o formas con las que el robot pueda interaccionar y se plantea un sistema que se base sólo en la búsqueda de estas. Es una aproximación muy interesante, pero el hecho de que no se reconozcan objetos implica que no se pueda, por ejemplo, realizar una clasificación para discernir el tipo de objeto con el que se va a interaccionar. También está el problema de los objetos que estén ocluidos por otros. Un objeto puede estar exponiendo una superficie candidata para la manipulación hacia el exterior, pero no ser realmente el mejor candidato para extraer por estar cubierto por otro en el resto de su superficie.

2.6 Sistema seleccionado

Teniendo en cuenta la revisión del estado de la técnica en los anteriores apartados se realiza un diseño de la solución tal y cómo se presenta en la introducción de este documento. A partir de datos de brillo, posición y orientación espacial de la escena, emplearemos un detector de piezas basado en una red convolutiva entrenada con imágenes sintéticas para detectar piezas, y un sistema basado en correspondencias de puntos característicos para la estimación de su pose. A lo largo de los apartados que componen esta sección se irá explicando, punto por punto, en qué consiste el funcionamiento de los módulos que componen la solución presentada.

3 Descripción del sistema

3.1 Sistema de captura

3.1.1 Tipos de sistemas

Como se ha señalado previamente, gracias al avance tecnológico producido en sensores 3D y en el aumento de la capacidad de procesamiento del hardware, se han incorporado al mercado una serie de sensores con un relativo bajo coste que facilita la labor de investigación y desarrollo en estas áreas. Por ejemplo, en [21] tenemos una lista de sensores con drivers disponibles para ROS, donde se comprueba la variedad de opciones disponibles y la existencia de una comunidad que trabaja con ellos y apoya su desarrollo. A continuación, se documenta un resumen de los sensores para la adquisición de información 3D más utilizados en aplicaciones robóticas, según la tecnología que empleen para ello.

3.1.1.1 Sensores de luz estructurada

El funcionamiento general de este tipo de sistema se basa en la proyección de un patrón de luz sobre una superficie 3D, el cual al ser observado desde otras perspectivas diferentes a la del proyector se ve deformado. Esta deformación está relacionada con la superficie de la escena, y puede ser analizada para realizar una reconstrucción geométrica.

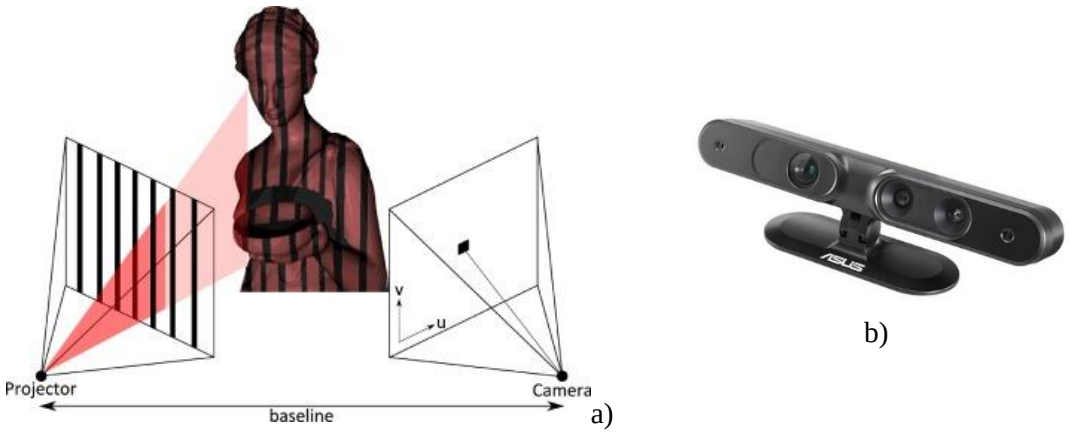


Figura 6. a) Funcionamiento de un sensor de luz estructurada. b) Ejemplo de senso: ASUS® XtionPro™ Live

3.1.1.2 Sensores de tiempo de vuelo (ToF)

La tecnología de *Time of Flight* (ToF) se usa para estimar las distancias de las superficies de los objetos que aparecen en la escena, calculando el tiempo transcurrido entre la emisión y la recepción de un haz de luz infrarroja. Estos sensores tienen incorporados uno o varios emisores de luz y un receptor. Los primeros emiten señales de luz que se reflejan en la superficie de objetos. La luz reflejada es captada por el receptor. Midiendo la diferencia temporal entre la emisión y la recepción de las señales, se calculan los valores de profundidad o distancia entre cámara y objeto. Uno de los principales beneficios de esta tecnología es la simplicidad para medir las distancias, evitando realizar cálculos posteriores necesarios en otras técnicas.

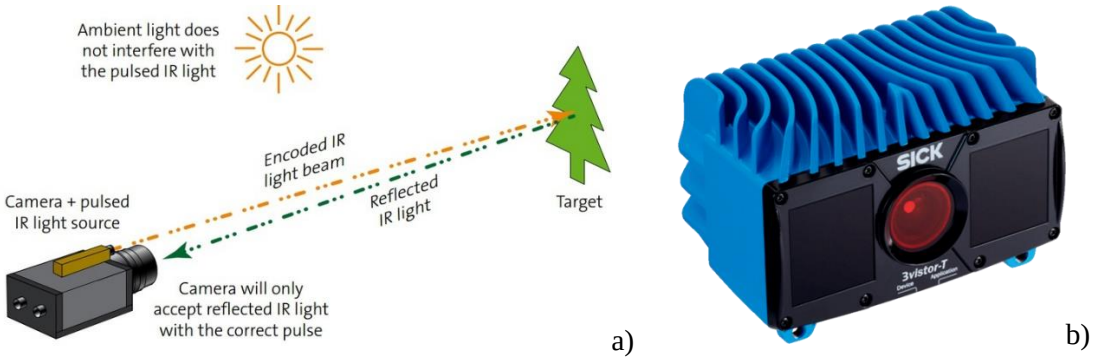


Figura 7. a) Esquema funcionamiento de un sensor tipo ToF [22]. b) Ejemplo de sensor: SICK® Visionary-T™

3.1.1.3 Laser Imaging Detection and Ranging (LiDAR)

Un sensor LiDAR (Figura 8) es similar a un radar, con la excepción de que el sensor envía y recibe pulsos de luz en vez de ondas de radio. Esta tecnología se utiliza principalmente para cartografiar entornos con sistemas de mapeado móvil (ya sean terrestres o aéreos). La mayoría de los sensores LiDAR están dotados de un único láser que se proyecta sobre un espejo giratorio y permiten determinar la distancia desde este emisor a un objeto o superficie utilizando un haz de pulsos láser. La distancia al objeto se estima midiendo el tiempo que tarda la señal desde la emisión del pulso y la detección de su reflejo, como se hacía en los sensores de Tiempo de Vuelo vistos anteriormente.

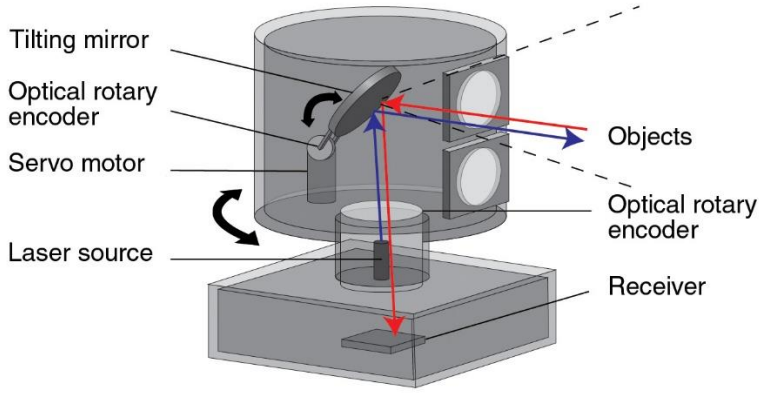


Figura 8 . Ejemplo de sensor LiDAR [23].

El LiDAR permite obtener una nube de puntos de superficies, aunque principalmente se suelen emplear no para obtener información de objetos sino de escenas completas, ya que trabajan a un mayor rango de distancia. Este rango oscila entre los 10 metros, con sensores relativamente baratos, hasta aquellos que llegan a los 50 o 70 metros de distancia, como son el caso del sensor LMS151 y el HDL-32E.

3.1.1.4 Par estéreo 3D

El par estéreo de cámaras (Figura 9) es una técnica que utiliza dos o más cámaras, generalmente de las mismas características, y no necesariamente calibradas (aunque suele ser lo más habitual para facilitar el procesamiento y obtención de información 3D). Se han desarrollado distintas técnicas para obtener información 3D desde múltiples vistas, pero la aproximación más sencilla es la que se basa en el uso de dos imágenes adquiridas desde dos cámaras calibradas y conocida su posición relativa. Las coordenadas de la proyección de un punto y de los dos centros ópticos de dos cámaras forman un triángulo; un hecho que puede ser visto como una restricción algebraica que involucra la pose de las cámaras y las coordenadas de la imagen, pero no la posición 3D de los puntos. Por lo tanto, dados suficientes puntos, esta restricción puede usarse para resolver la

pose de las cámaras, y una vez obtenidas estas poses, la posición 3D de los puntos en la escena se puede calcular fácilmente por medio de triangulación.

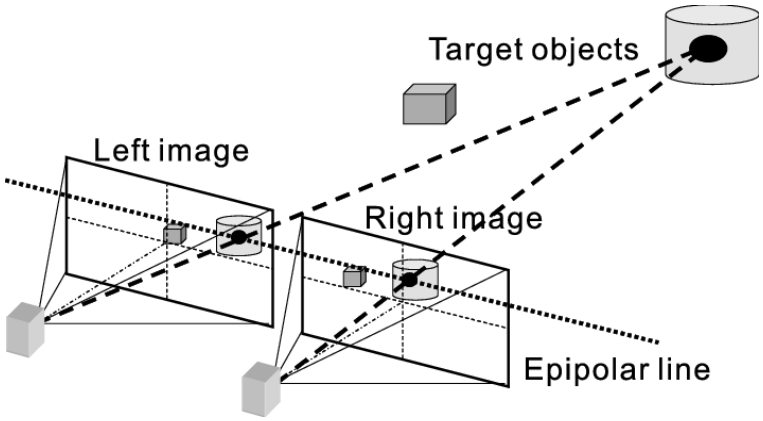


Figura 9. [24] Sensor par estéreo 3D y esquema de funcionamiento de la visión estereoscópica. Los rayos de proyección muestran los diferentes ángulos de visión de las cámaras sobre los objetos de la escena.

El sistema de emparejamiento de puntos entre las imágenes se basa en la gradación de contrastes y de brillo de los píxeles del sensor. Este tipo de sensores se usa extensamente en aplicaciones de control robótico debido a las ventajas que ofrece en cuanto a velocidad de captura, resolución y rango.

3.1.1.5 Consideraciones generales

De los dispositivos estudiados, muchos se basan en la proyección de haces de luz con un emisor y su posterior lectura con un receptor integrado. Para evitar trabajar en una longitud de onda que interfiera con la luz ambiental, se suele trabajar con luz infrarroja, por lo que es muy común verlo en las especificaciones de estos dispositivos.

Habitualmente, además de la información de profundidad, estos dispositivos suelen hacer uso de un sensor adicional CCD o CMOS para almacenar variaciones de intensidad de brillo de la escena (como una cámara RGB o monocromática). De esta forma, además de obtener información de profundidad, se puede capturar la información de brillo de cada punto detectado, aunque esto requiere una calibración entre los dos sistemas para buscar las correspondencias entre un sensor y otro. A este tipo de sensores se les suele denominar cámaras tipo RGB-D (RGB por el espacio de colores RGB para la información de brillo y D de *depth*, profundidad de los puntos respecto a la cámara).

Para este proyecto sería válida prácticamente cualquier tecnología que nos proporcione información de brillo y profundidad de la escena, siempre y cuando tuviese la resolución suficiente para poder recuperar información de la superficie del spinner.

3.1.2 Sensor empleado

El sensor 3D que se utilizará es la cámara Ensenso N35 (Figura 10a) [25]. Este sensor de profundidad es una cámara estéreo que emplea un sistema de proyección de textura. Integra dos sensores CMOS y un proyector LED de puntos aleatorios sobre el objeto cuya imagen se desea capturar, como se muestra en la Figura 10b. El patrón ofrece la capacidad de introducir textura en superficies que carecen de ella, pudiendo reconstruir objetos con superficies completamente planas y con colores uniformes, lo cual soluciona los problemas de falta de textura en los spinners.

El sensor cuenta con un proyector *FlexView* que opera con un actuador piezoeléctrico. El patrón proyectado se traduce en una dirección preestablecida por hasta 16 pasos de distancia. En cada paso, un par de imágenes con su correspondiente patrón de proyección escalonada se capturan y procesan y, a continuación, los datos individuales se combinan en una única nube de puntos 3D

con una precisión más alta. No obstante, la rapidez de la captura de datos se sacrifica en aras de la precisión, por lo que las configuraciones de mayor número de pasos sólo funcionan para escenas estáticas.

a) b)

Figura 10. a) Cámara Ensenso N35. b) Esquema de la cámara Ensenso N35 compuesta de dos sensores CMOS y un proyector LED (en el centro)

Otra ventaja del uso de esta cámara es que ofrece un API para los lenguajes HALCON, C ++, C# y .NET. Además, este modelo cuenta con el nivel de protección IP65/67, lo que la hace adecuada para su uso en entornos industriales.

Resumen de características de la cámara escogida:

- Sensores CMOS Global Shutter y proyector de patrones con LEDs azules integrados
- Resolución de imagen 3D de 1280 x 1024
- Velocidad. (3D): 10img/s (2x Binning: 30img/s) y 64 niveles de disparidad
- Concebida para distancias de trabajo de hasta 3000 mm y para campos visuales variables
- Sistema *Projected Texture Stereo Vision* para captar superficies sin textura
- Paquete de software incluido gratuitamente, con controlador y API para Windows y Linux
- Interfaz GigE
- Función integrada para la calibración de robot mediante placa de calibración
- Funciones de subsampling y binning para una mayor flexibilidad en las tasas de transferencia de datos y en las frecuencias de imagen

El software que acompaña a la cámara ofrece muchas funcionalidades de calibración, captura y procesado. Ofrece la posibilidad, por ejemplo, de calcular las direcciones normales a la superficie en los puntos capturados por esta. También incluye comandos para realizar post-procesado en las nubes de puntos, como el relleno de agujeros o la eliminación de *outliers*. Para la realización de pruebas se buscarán aquellas configuraciones que mejoren la calidad de las nubes de puntos, aprovechando al máximo la funcionalidad de la cámara.

3.1.2.1 Proceso de captura con la cámara empleada

Cómo se ha indicado previamente, se querrá obtener mapas de brillo e información 3D de la escena. Sin embargo, la cámara proporciona sólo la nube de puntos con la información 3D (coordenadas y orientación mediante las normales), pero no los valores de brillo para cada punto. Para solucionar esto, el sistema de captura se basará en el método que emplean las cámaras estéreo para calcular las nubes de puntos.

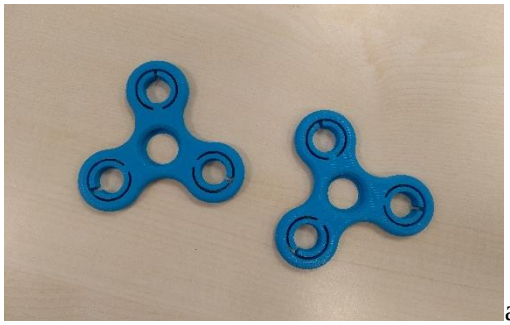
Resumidamente, un píxel determinado de la imagen izquierda le corresponde otro de la derecha, información con la que se extrae un correspondiente punto 3D. El emparejamiento de puntos que se realiza entre ambas imágenes para reconstruir la escena se hace, normalmente y cómo en

nuestro caso particular, respecto a la imagen izquierda rectificada. Por lo tanto, conociendo el valor de brillo de los píxeles en esta, se pueden asociar a los puntos 3D de forma directa.

Ya que esta cámara dispone de la posibilidad de exportar configuraciones a un archivo tipo JSON, crearemos dos perfiles de funcionamiento mediante una aplicación con interfaz gráfica que acompaña a su software (NxView). Estos perfiles serán cargados durante el proceso de captura y publicación de datos para cambiar entre la configuración de toma de imágenes monocromáticas y la de captura y cálculo de nubes de puntos con sus respectivas normales.

3.2 Simulador de datos

Para este proyecto se ha creado un simulador para la generación de datos con dos objetivos principales: entrenar los algoritmos de segmentación y realizar pruebas para depurar algoritmos. El hecho de generar las imágenes virtualmente proporciona una gran serie de ventajas, donde la de mayor relevancia es la de poder realizar un etiquetado automático de las imágenes. Incluir un módulo de etiquetado automático en sistemas que se basen en imágenes clasificadas para su entrenamiento supone la capacidad de automatizar todo el proceso de generación de modelos. Esto resulta muy interesante, por ejemplo, para líneas de producción en los que sea necesario incluir nuevos modelos sin la necesidad de invertir grandes cantidades de tiempo y recursos en etiquetar manualmente imágenes. Por otro lado, se puede conservar la información empleada para la generación de las imágenes y ser utilizada posteriormente en fases de depuración o validación de algoritmos.


a)

b)

Figura 11. a) Imagen real de los spinners. b) Imagen sintética.

El objetivo del simulador es, entonces, el de reproducir imágenes del contenedor con spinners en su interior con la mayor fidelidad posible al caso real, como la que se puede ver en la Figura 11a. Ya que se dispone del modelo 3D de los spinners, la tarea de este simulador consiste en el renderizado 3D de varias instancias distribuidas aleatoriamente en la escena. Por lo tanto, será necesario emplear un motor gráfico que renderice imágenes a partir de los modelos 3D de los spinners contenidos en un escenario virtual.

Sin embargo, ubicar los spinners arbitrariamente en el espacio sin más supone una serie de problemas, como cuerpos interesectando entre sí o poses desde las que no se pueden apreciar la geometría del spinner (Figura 12). Estos problemas se añaden al hecho de que en la imagen la pose de los spinners no es completamente fiel a cómo se encontrarían distribuidos en una situación real, ya que se verían “flotando” en el aire. Se propone entonces crear una simulación basada en la interacción física de los spinners con la escena y entre sí, de forma que los cuerpos se apilen y distribuyan de la misma forma que lo harían en el caso real. De esta manera, tendremos una aproximación más realista de la disposición de los spinners en la escena.

Figura 12. Problemas derivados de la colocación aleatoria en el espacio. a) Poses desde las que no se aprecia la geometría del spinner. b) Intersección entre cuerpos.

En este apartado se realizará una valoración y selección de las tecnologías existentes para implementar este sistema de simulación, explicando posteriormente su funcionamiento.

3.2.1 Plataformas de creación de aplicaciones para entornos virtuales

En la documentación revisada hay sistemas que hacen uso de imágenes sintéticas ya sea para entrenar, depurar o validar los algoritmos. En general, en la mayoría estos artículos se menciona el uso de OpenGL [26], que es una especificación estándar para la generación de gráficos que define una interfaz de programación de aplicaciones para distintos lenguajes y plataformas. Con este estándar se define la forma con la que se debe interaccionar sobre la tarjeta gráfica de un ordenador, y aunque es muy versátil, trabajar con ella resulta en una programación de muy bajo nivel. Por ello, en otros documentos se emplean librerías que abstraen al programador de todos los detalles del funcionamiento de los sistemas base como OpenGL u otras especificaciones como Direct3D. Ejemplos de ello son Ogre [27] o VTK [28].

Para aquellos sistemas en los que se desea, además, simular la interacción física de objetos, es necesario el uso de librerías adicionales. En varios documentos se menciona el empleo de Bullet [29], una librería que proporciona un motor de físicas para la simulación de las colisiones y la dinámica de cuerpos físicos.

No obstante, las anteriores opciones suponen una curva de aprendizaje muy alta, ya que para trabajar con ellas son necesarios amplios conocimientos tanto en la creación de gráficos por ordenador como en el uso de las librerías en sí. Esto presenta una gran dificultad para la realización de prototipos y la capacidad de iterar entre distintas soluciones, por lo que se han buscado otras posibles herramientas con las que generar las imágenes para este proyecto.

Se plantea entonces el uso de entornos normalmente empleados para el desarrollo de videojuegos. Este tipo de herramientas permiten situar los archivos gráficos en un mismo escenario, añadiendo interacciones, comportamientos e interfaces para construir aplicaciones de cualquier tipo. Este tipo de plataformas incluyen un motor gráfico para renderizar la escena, además de un motor de físicas para simular la interacción entre cuerpos, por lo que la funcionalidad que proporcionan resulta más que suficiente para el objetivo de este sistema. Algunos ejemplos son Unity [30], Unreal Engine [31] o CryEngine [32].

Por otro lado, también existe la posibilidad de emplear programas dedicados al modelado 3D que incluyan motores de físicas, como es el caso de Blender [33]. Blender es una plataforma de código abierto que incluye todas las funcionalidades necesarias en tecnologías 3D como, entre otras, el modelado, animación, simulación y renderizado de objetos.

Se decide emplear Unity como plataforma de desarrollo para este sistema. Los principales motivos para seleccionar a esta plataforma frente a las otras son la menor curva de aprendizaje, la gran cantidad de documentación disponible y la presencia de una comunidad muy extensa que la apoya. Además, la Línea de investigación en Tecnologías de Visión de CTIC tiene experiencia previa con esta herramienta, lo que supone una mayor facilidad para el alumno en el caso de necesitar apoyo durante el desarrollo.

3.2.2 Generación de datos

3.2.2.1 Simulación de físicas: generación de poses

Siendo la intención simular todas las potenciales condiciones en las que se puedan encontrar los spinners en una escena real, se aprovechará el motor de física de Unity. Se generarán instancias virtuales de spinners mediante su modelo 3D con una pose aleatoria y a una distancia determinada de un plano. Se dejarán caer sobre el mismo, simulando el efecto de la gravedad. Estos objetos, que responderán a las colisiones con el plano y entre ellos, se depositarán en la escena, apilándose según vayan cayendo más piezas hasta que se alcance el número máximo de elementos establecido. Una vez haya pasado un tiempo predeterminado (tiempo en el que se supone que la escena habrá alcanzado un estado estable), se guardará la pose de cada spinner y se volverá a iniciar la simulación, repitiendo el proceso hasta alcanzar un número de iteraciones establecido. De esta forma, al finalizar esta primera fase tendremos una lista de posiciones y rotaciones para cada spinner.

Cada spinner de la escena tendrá un identificador propio y único para diferenciarse del resto de objetos. Al comenzar la aplicación, crearemos un *object pool* de spinners (optimización para evitar instanciar y eliminar objetos en cada iteración), y en el momento de su creación cada uno obtendrá un identificador. En este caso, el identificador será un color del espacio de colores RGB, que se corresponderá con el que se vaya a dibujar al generar la imagen de máscaras, como se verá más adelante.

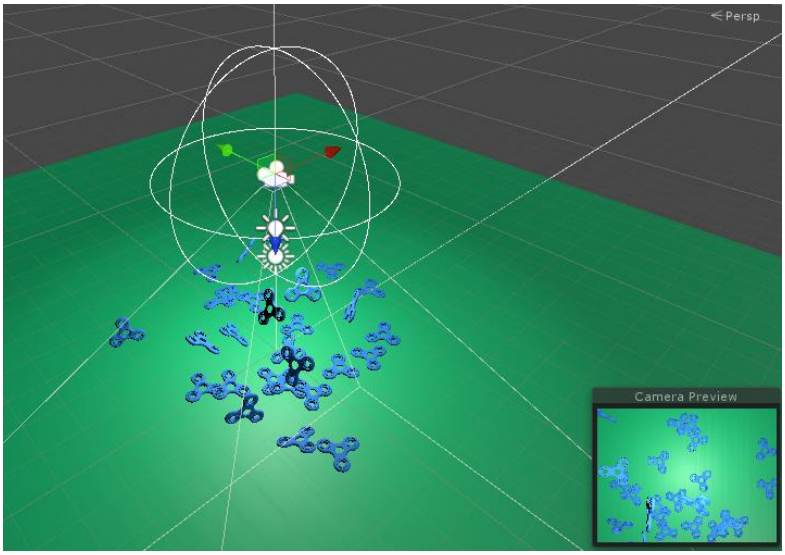


Figura 13. Ejemplo de escena del simulador Unity

3.2.2.2 Renderizado

Para cada entrada en la lista de poses generadas durante la fase de simulación de físicas, se procederá a renderizar la escena para ser posteriormente guardada en un archivo.

Se renderizará la escena desde una cámara virtual con una pose que se asemeje a la que vaya a tener en la configuración real. Sin embargo, en cada iteración se modificará la posición de la misma y la iluminación de la escena, produciendo imágenes diferentes entre sí para entrenar

correctamente al detector de piezas, que de otra forma solo funcionaría con imágenes iguales a las que ofrece el simulador.

Para ello se empleará el método de renderizado estándar que ofrece por defecto Unity. Describir en profundidad cómo funciona el renderizado 3D queda fuera del alcance de esta memoria, pero se puede consultar una buena introducción a cómo se realiza en [26]. En ese caso se describe el procedimiento para trabajar directamente sobre OpenGL, aunque los conceptos que se aplican son prácticamente análogos.

Un ejemplo del material empleado para visualizar cada spinner puede verse en la captura de pantalla de la Figura 14. Se basa en el *shader* empleado por defecto por Unity para renderizar los objetos, configurado para representar un tipo de material que se asemeje al plástico. Para ello, el valor metálico y la suavidad se mantienen en valores cercanos al cero.

Se podría añadir algún tipo de textura que simulase los defectos superficiales del spinner producidos durante la impresión 3D. En una primera iteración se buscó un mayor realismo mediante el empleo de este tipo texturas, pero se observaba que afectaba demasiado al detector de objetos, que sólo detectaba aquellas piezas que tuviesen exactamente los mismos defectos superficiales. Ante la opción de intentar generar estos defectos de forma procedural en su lugar, tarea que podría complicar bastante el sistema, se optó por utilizar el material sin más detalle que el que proporcione la geometría.

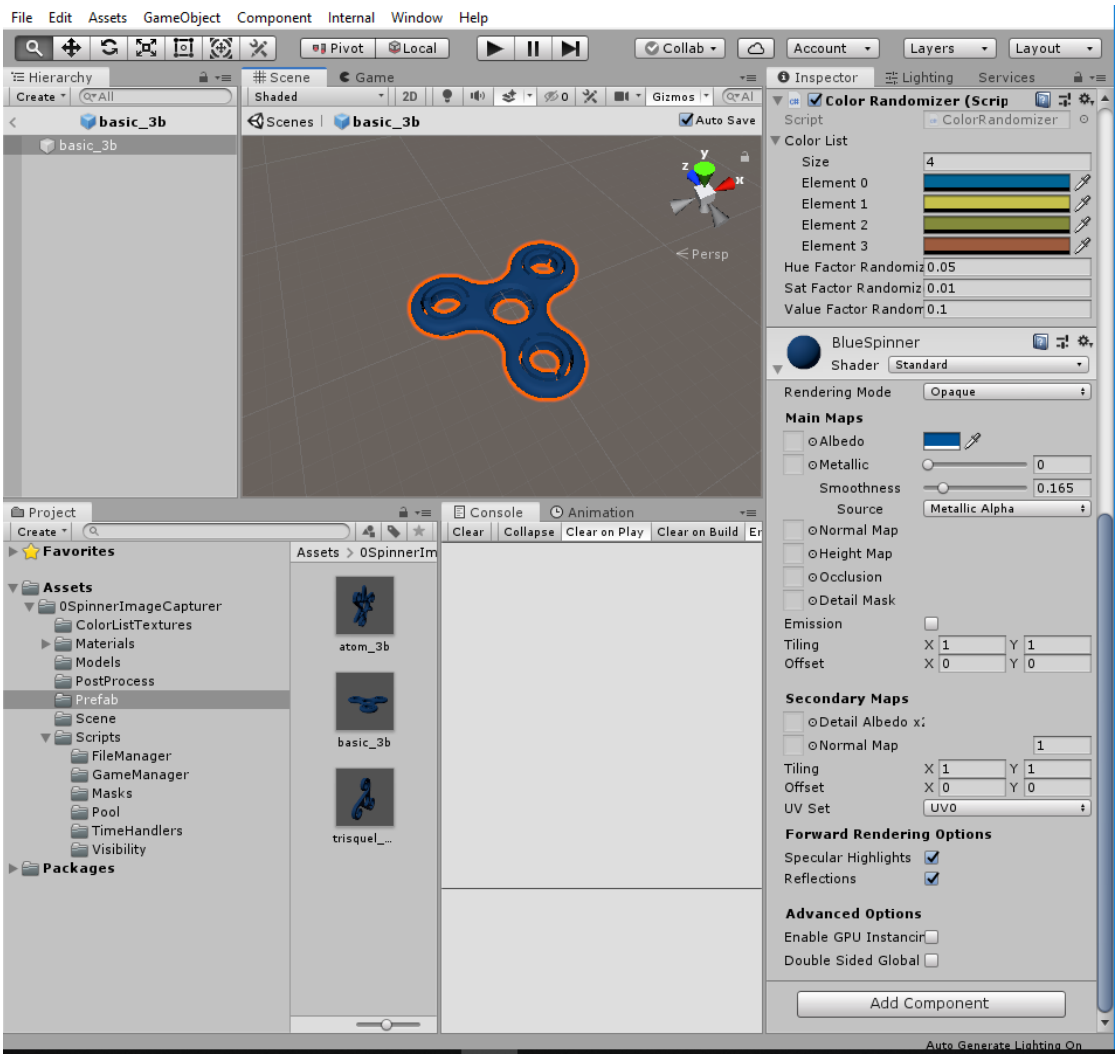


Figura 14. Captura de pantalla del *prefab* de spinner básico de 3 bolas.

Además, se da la opción de escoger el color de cada spinner entre los de una lista predefinida de forma aleatoria en el momento de su inicialización. De esta forma, cada spinner tendrá un color distinto de entre los que haya disponibles en la planta, introduciendo además desviaciones en torno al color base para que haya aún más variación en las muestras.

3.2.2.3 Etiquetado de las imágenes

Al mismo tiempo que se renderiza la escena, se generarán imágenes con máscaras que definan los píxeles que pertenecen a un objeto determinado. Estas imágenes contendrán regiones con colores, siendo cada instancia de objeto mapeado con su color identificativo. De esta forma, un spinner determinado en la imagen de la escena tendrá un único color asociado en la correspondiente imagen de máscaras, quedando convenientemente etiquetado para el entrenamiento de la red neuronal. Se ha escogido este formato porque servirá a la hora de gestionar las oclusiones entre objetos. Si dos spinners están superpuestos y ambos tienen máscaras del mismo color, no se podría saber dónde acaba uno y dónde comienza el otro. Otra opción sería escribir máscaras con menor número de canales y bits por cada pixel, y escribir un archivo por cada instancia de objeto; sin embargo, esto ocuparía mucho espacio en disco. No obstante, con el método elegido, se hace necesario poner especial atención a los formatos y la compresión de las imágenes generadas, pues la aplicación de las máscaras daría problemas en caso de perderse alguno de los valores de color definidos.

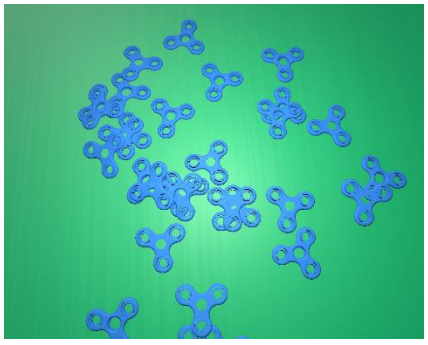
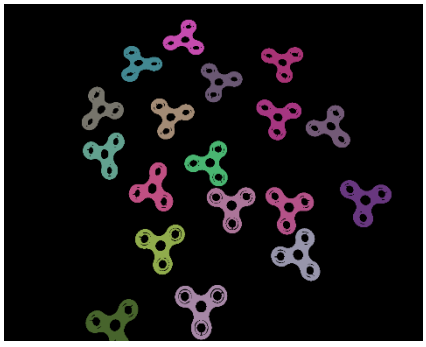
 a)  b)

Figura 15. a) Imagen generada b) Máscaras de las piezas para la misma imagen

Antes de generar la imagen de máscaras, se descartarán aquellos spinners que se encuentren muy ocluidos en la imagen (es decir, aquellos de los que la cámara pueda ver poca superficie). De esta forma, evitamos entrenar la red con spinners de los que casi no se tengan puntos en la imagen, ya que se ha comprobado de forma empírica que no realizar este paso previo de validación da lugar, posteriormente, a muchos falsos positivos.

Se empleará una técnica de traza de rayos (o *raycast*) para definir la visibilidad de un spinner en la cámara. Estos rayos son rectas que pasan por un punto en una dirección y sentido determinado, proporcionando información de todos los objetos a los que intersecte. Estos rayos se emitirán desde el centro de proyección de la cámara y pasarán por el centro de los píxeles en la imagen, de forma que a cada píxel le corresponda un rayo. Para ello, se creará una cuadrícula de puntos en el plano de clip cercano de la cámara con una resolución igual a la de la textura donde se va a renderizar la imagen, obteniendo una serie de puntos de muestreo en el espacio que se corresponderán con el centro de los píxeles en la textura. La colisión de un rayo con los objetos de la escena informará del objeto que está siendo renderizado en un píxel determinado, además de poder informar de los spinners que se encuentren detrás de este. De esta forma, se podrá llevar una cuenta interna de la cantidad de puntos visibles y los no visibles desde la perspectiva de la cámara para cada uno de los spinners. Si la proporción entre unos y otros no supera un umbral establecido, se descartará el objeto y su región no será dibujada en la imagen de máscaras, al determinarse que se encuentra sometido a demasiada oclusión.

3.3 Detección de spinners

Se debe generar un sistema que detecte las piezas en la escena a partir de imágenes 2D y segmente las regiones en las que se encuentran. Para ello, se construirá un modelo mediante *Deep Learning* o aprendizaje profundo, un método de aprendizaje automático que hace uso de una cascada de capas con unidades de procesamiento no lineal para extraer y transformar las imágenes de entrada.

El algoritmo para utilizar en este caso consiste en una cascada de redes neuronales convolucionales basadas en regiones (*Mask-RCNN*). Emplear esta técnica tiene la gran ventaja de eliminar la necesidad de extracción de características manualmente, por lo que no es necesario identificar a priori las características utilizadas para la clasificación de las imágenes, sino que el sistema aprende a detectarlas mediante decenas o cientos de capas ocultas entrenadas con bases de datos etiquetadas. Cada capa usa la salida de la capa anterior como entrada, de tal forma que el algoritmo va refinando su aprendizaje a lo largo de toda la red y va aumentando la complejidad de las características aprendidas de la imagen. Por ejemplo, la primera capa oculta podría aprender cómo detectar bordes, mientras la segunda aprende cómo detectar formas más complejas propias de la forma del objeto que se intenta reconocer.

Frente a otras tecnologías, el *Deep Learning* alcanza unos niveles de precisión de reconocimiento muy elevados, incluso en imágenes con baja calidad o en donde los objetos detectados aparecen parcialmente ocultos. A cambio, requiere el uso de gran cantidad de datos de entrenamiento previamente etiquetados. Además, este tipo de técnica requiere una potencia de cálculo mucho más elevada que los sistemas de aprendizaje automático convencionales.

Para solucionar estos problemas se hace uso de la técnica denominada *transfer learning* o transferencia de aprendizaje [34]. Este método consiste en utilizar un modelo ya entrenado para una tarea y usarlo como punto de inicio de otro modelo para una tarea distinta. Se trata de una técnica muy útil para generar nuevos modelos de *Deep Learning*, debido a que se reduce considerablemente el tiempo de procesamiento y la enorme cantidad de datos que harían falta para realizar un entrenamiento completo desde cero. Existen diferentes modelos pre-entrenados mediante *Deep Learning* utilizando conjuntos de entrenamiento de miles de imágenes. Empleando uno de ellos, se re-entrenarán las últimas capas con el conjunto de entrenamiento de la pieza deseada para este caso de uso particular. En la Figura 16 se muestra el esquema general de la generación del detector de spinners mediante *Deep Learning* utilizando la técnica de transferencia de aprendizaje.

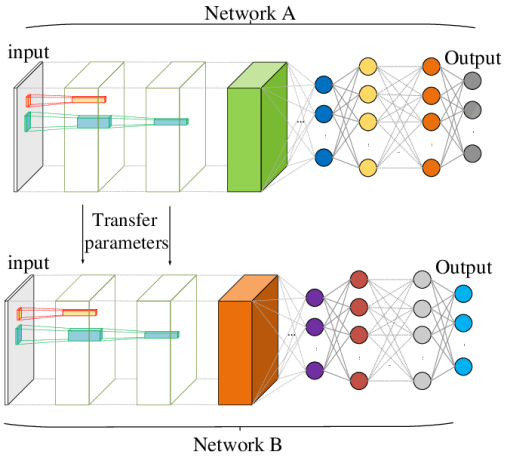


Figura 16. Esquema del sistema de transferencia de aprendizaje de *Deep Learning* para el algoritmo desarrollado.

El detector generado de esta forma ofrecerá como resultado, ante una imagen de entrada, la pieza o piezas que aparezcan en la imagen segmentadas mediante una máscara junto con el porcentaje de certeza de la detección.

3.3.1 Selección de la plataforma y del sistema de transferencia de conocimiento

Como se ha indicado previamente, se emplea una técnica basada en redes neuronales convolucionales para la detección de piezas en la escena a partir de imágenes. Se realizará un estudio de las diferentes plataformas existentes para el desarrollo de sistemas basados en *Deep Learning* para escoger aquella que mejor se adapte al proyecto. En la Tabla 3 se muestra una comparativa realizada por el equipo de Investigación y Desarrollo de Silicon Valley Data Science [35] de distintas plataformas de código abierto disponibles.

	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	

Tabla 3. Comparativa de diferentes plataformas existentes para el desarrollo de sistemas *Deep Learning*.

En el estudio realizado, se puntúan diferentes factores de cada plataforma basándose en experiencias propias de los investigadores, así como en estudios publicados con evaluaciones comparativas. Para cada sistema se valoran la cantidad de materiales disponibles (modelos entrenados y tutoriales), la capacidad de entrenamiento de redes neuronales convolucionales (CNN), la capacidad de entrenamiento de redes neuronales recurrentes (RNN), la facilidad de uso de la arquitectura, la velocidad, el soporte de múltiples GPUs y la compatibilidad con Keras, una biblioteca de código abierto de uso muy extendido para generar modelos de redes neuronales.

Además, también se aportan comparativas de las distintas arquitecturas según el interés de los usuarios en GitHub, el número de contribuciones en la misma, y el número de aportaciones (Figura 17).

Basándose en este estudio y analizando la aplicación necesaria para este proyecto se escoge la plataforma *TensorFlow* [36] ya que ofrece una amplia documentación, rendimiento alto y muy buenas herramientas para aplicaciones de reconocimiento de imágenes. Además, el hecho de que se pueda desarrollar en Python es valorado positivamente en esta elección, ya que es un lenguaje que permite hacer prototipado rápido durante la fase de desarrollo.

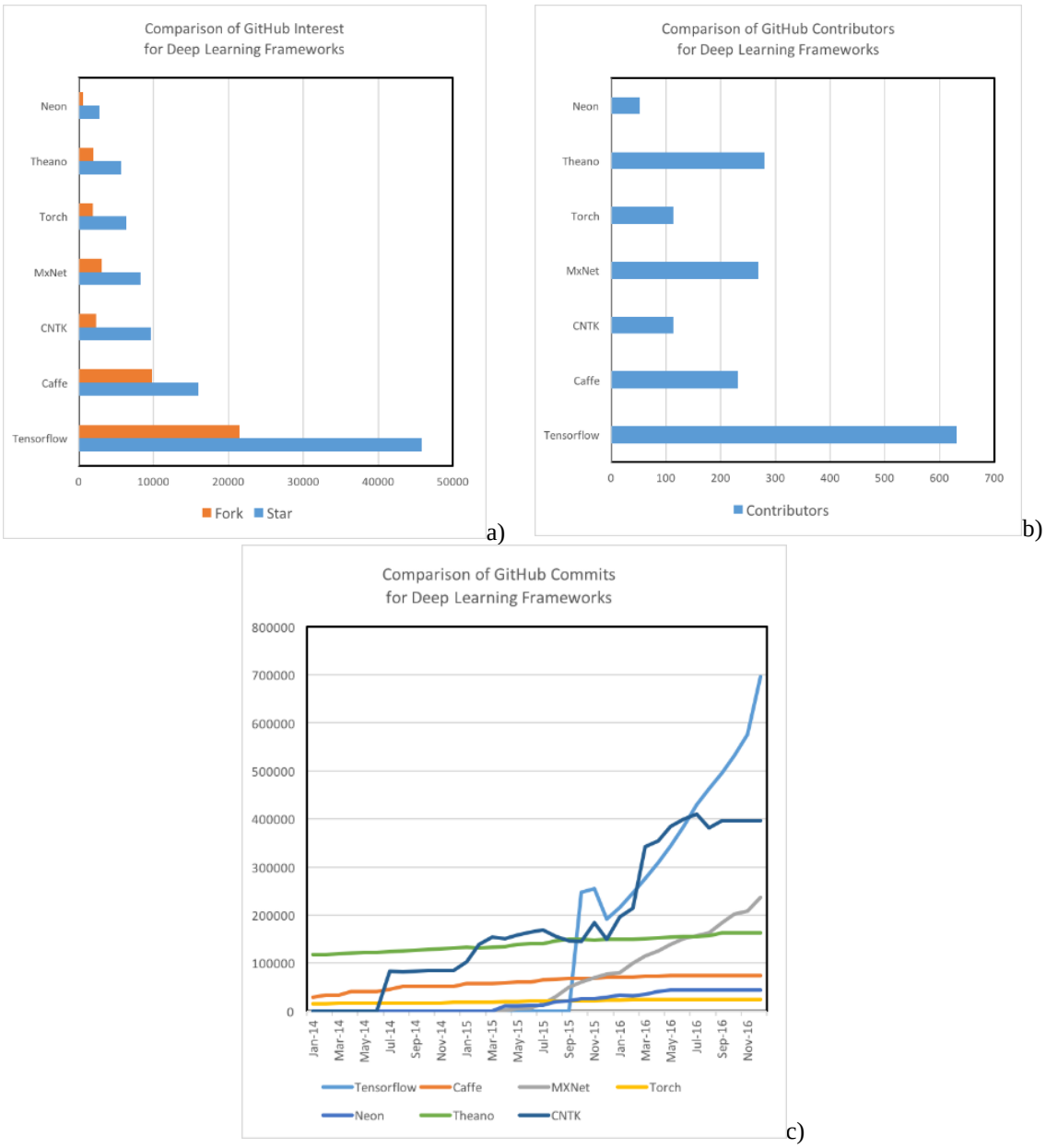


Figura 17. Comparativas de diferentes herramientas existentes para el desarrollo de sistemas *Deep Learning* según el interés (a), contribuciones (c) y aportaciones (b) en la plataforma GitHub.

En [37] tenemos, además, un *framework* implementado sobre *TensorFlow* que facilita la construcción, entrenamiento y despliegue de modelos para detectar objetos. En este mismo enlace se muestran los modelos disponibles pre-entrenados para *TensorFlow* utilizando el conjunto de datos COCO (*Common Objects in Context*) y sus características [38]. Entre los existentes, se ha elegido el modelo *mask_rcnn_inceptioon_v2_coco*. La elección del modelo se ha realizado basándose en el buen equilibrio que presenta este modelo entre velocidad y precisión, como puede observarse en la misma tabla.

Una vez que se ha creado la base datos y se ha preparado el sistema de transferencia de aprendizaje se re-entrenarán las últimas capas del modelo seleccionado utilizando las imágenes y etiquetas que se quieren detectar para el caso de uso particular. En este caso, será entrenado con las imágenes generadas con el simulador. Para ello es necesario convertir las imágenes y sus etiquetas al formato de datos que maneja la plataforma *TensorFlow* y lanzar el entrenamiento hasta obtener una determinada precisión o tras haber realizado un número suficiente de iteraciones.

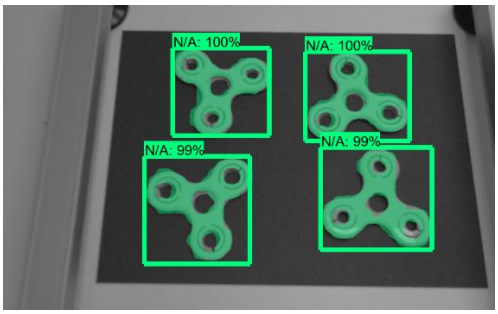


Figura 18. Resultado del proceso de detección de spinners.

Por lo tanto, teniendo como entrada una imagen de la escena, este módulo detectará y segmentará todos los spinners presentes en la misma. En la Figura 14 se puede ver un ejemplo del resultado devuelto por este módulo.

3.4 Selección de pieza y extracción de puntos 3D pertenecientes al objeto

Se realizará una selección de la mejor pieza a coger de entre todas las presentes, basándose en la información de profundidad (más arriba) y en su visibilidad (mayor área visible). Dicho de otra forma, el objeto óptimo para ser el siguiente a recoger por el brazo robótico será aquel que ha quedado por encima de los demás al volcarse en el contenedor. Para ello, se calculará el área en píxeles que ocupa la región detectada por el detector 2D y se extraerá la información de altura de esta misma región, realizando un promedio del valor de cada punto gracias a la información de profundidad correspondiente. A cada pieza se le dará una puntuación basada tanto en el área como en el valor medio de altura, dando mayor puntuación a aquellas que tengan valores más altos en los dos campos. Estas puntuaciones serán sumadas y la región que tenga una mayor puntuación total será escogida como próxima pieza a ser extraída.

En la práctica se ha observado que la máscara devuelta por el detector 2D está muy ajustada al objeto en sí, y que incluso puede llegar a no alcanzar los límites del objeto en ciertas partes de la región. Los bordes de las piezas son muy descriptivos de la geometría de estas y perder esta información resulta perjudicial para el desempeño de los algoritmos de estimación de pose. Por este motivo se va a realizar una pequeña dilatación a la máscara para intentar obtener todos los puntos pertenecientes a dicha pieza, aunque eso suponga añadir puntos del fondo no deseados.

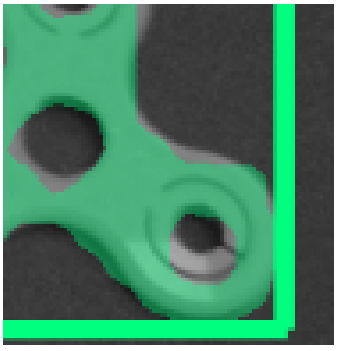


Figura 19. Ejemplo de máscara que no cubre toda la región del spinner.

Una vez definida la región de la próxima pieza a ser retirada del contenedor, se extraerá la nube de puntos correspondiente del mapa que contiene la información 3D de la escena. Aplicar esta máscara sobre el mapa 3D de la escena se hace de forma directa, pues un requisito del sensor es tener una relación uno a uno entre el mapa de valor de brillo y el de información 3D.

3.5 Estimación de la pose 3D

El siguiente paso, una vez seleccionada la pieza a coger por el brazo robótico y extraídos sus puntos 3D, consiste en el cálculo de su posición y orientación respecto al sistema de coordenadas establecido. Para ello, se utilizará la nube de puntos recortada, obtenida en el paso anterior, y el modelo 3D de la pieza.

Para detectar la pose de la pieza se emplea un algoritmo basado en *Point Pair Features* [6], implementado en el módulo de *surface matching* en OpenCV [8]. Este método consiste en describir un modelo del objeto mediante la extracción de descriptores basados en pares de puntos, que son posteriormente discretizados e indexados en una tabla. Estos mismos descriptores son extraídos de la escena en tiempo de ejecución y comparados con la tabla que define al objeto, utilizando un sistema de votos similar a la transformada de Hough generalizada [39] para realizar el emparejamiento de puntos con la escena real y la estimación inicial de la pose 3D. Por último, se refina después su posición y orientación final mediante el uso de un algoritmo *Iterative Closest Point* [12] optimizado para esta solución.

Suponemos que tanto la escena como el modelo son representados mediante nubes de puntos orientados (es decir, donde cada punto tiene una normal asociada). Los denotaremos como $s_i \in S$ para puntos en la escena y $m_i \in M$ para puntos pertenecientes al modelo.

3.5.1 Descriptor empleado

Como se ha comentado, el algoritmo se basa en la extracción e indexación de las características de pares de puntos (llamados en el documento *Point Pair Feature*), que se definen de la siguiente manera, donde m_1 y m_2 son dos puntos orientados del modelo (o escena), d es el vector de diferencia, $< (a_1, a_2)$ el ángulo que forman dos vectores a_1 y a_2, y n_1 y n_2 son las normales en m_1 y m_2, tal y como viene representado en la Figura 20:

$$F(m_1, m_2) = (\|d\|_2, < (n_1, d), < (n_2, d), < (n_1, n_2)) \quad (1)$$

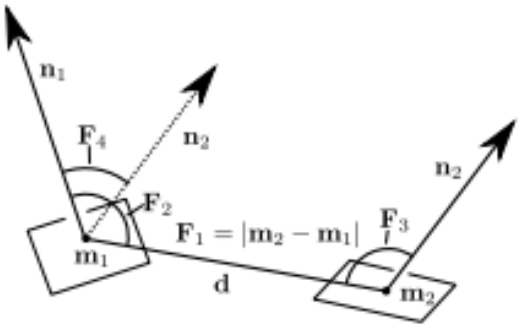


Figura 20. Característica del par de puntos F de dos puntos orientados. La componente F1 corresponde con la distancia entre los puntos, F2 y F3 con los ángulos entre las normales y el vector definido por estos dos puntos, y F4 es el ángulo entre las dos normales.

3.5.2 Descripción del modelo global

El modelo global de un objeto es representado por un conjunto de descriptores de pares de puntos donde los vectores descriptores que sean similares son agrupados juntos. Para ello, se calcula el vector descriptor comentado previamente para cada par de puntos perteneciente a la superficie del modelo. Los vectores resultantes se discretizan, y aquellos que tengan el mismo valor se agrupan.

El descriptor global del modelo es una tabla que mapea estos vectores descriptores a los puntos del modelo que han dado ese valor. Es decir, en la práctica, se genera una tabla *hash* indexada por

el descriptor que almacena los puntos del modelo correspondientes. En la Figura 21 podemos ver una representación de este modelo.

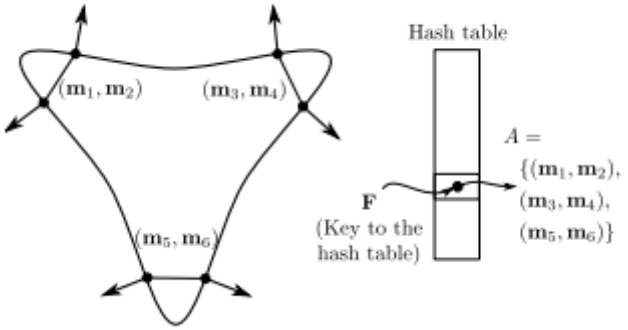


Figura 21. Descripción del modelo global. A la izquierda vemos una figura donde están marcados varios pares de puntos de la superficie del modelo con un vector descriptor F similar. Vemos, a la derecha, que estas parejas de puntos se almacenan en la misma entrada de la tabla hash.

De esta forma, todos los puntos con un descriptor del modelo que sean similares a un descriptor de la escena se pueden buscar de forma rápida a través de esta tabla.

3.5.3 Sistema de votos

Considerando que tenemos una correspondencia entre un punto del modelo y otro del objeto en la escena, si ambos están alineados (es decir, son coincidentes), así como sus normales, el objeto puede ser alineado con la escena mediante un giro alrededor de dicha normal. Por lo tanto, se puede describir la pose del objeto en la escena mediante un punto del modelo y un ángulo de rotación. A este par de valores (m_r, α) se los designará como **coordenadas locales del modelo** respecto a un punto de referencia de la escena s_r.

En el método propuesto, un par de puntos del modelo $(m_r, m_i) \in M^2$ es emparejado con un par de puntos de la escena $(s_r, s_i) \in S^2$ si ambos tienen un vector descriptor similar. La transformación de las coordenadas locales del modelo a las coordenadas de la escena se define con:

$$s_r = T_{s\to g}^{-1}\, R_x(\alpha) T_{m\to g}\, m_i \qquad (2)$$

La transformación $T_{m\to g}$ traslada m_r al origen y rota su normal para que coincida con el eje x. $T_{s\to g}$ hace lo mismo para pares de puntos de la escena. $R_x(\alpha)$ sirve para alinear, mediante una rotación alrededor de la normal del punto de referencia, los puntos m_i y s_i.

Dado un punto de referencia de la escena s_r, queremos encontrar las coordenadas locales óptimas para maximizar el número de puntos del modelo que están alineados con los de la escena. Para ello, se crea un acumulador de tantas filas como número de puntos del modelo y tantas columnas como intervalos de ángulo hayamos definido. Esta matriz representa un espacio discreto de coordenadas locales para un punto de referencia específico.

Un esquema del proceso de votación puede verse en la Figura 22. Los pasos son los siguientes:

1. Cada punto de referencia s_r se empareja con el resto de los puntos de la escena $s_i \in S$ y se calcula su vector descriptor $F_s(s_r, s_i)$.
2. Este vector puede usarse como índice para la tabla *hash* del modelo, que nos devuelve un conjunto de pares de puntos (m_r, m_i) con una orientación y distancia relativa similares a las de la escena, por lo que son susceptibles de ser los mismos.

3. Para cada posible resultado de cada pareja de puntos, se calcula el ángulo α que mapea el par de puntos (m_r, m_i) a (s_r, s_i) como se ha explicado anteriormente.
4. Se vota entonces para la coordenada local (m_r, α).

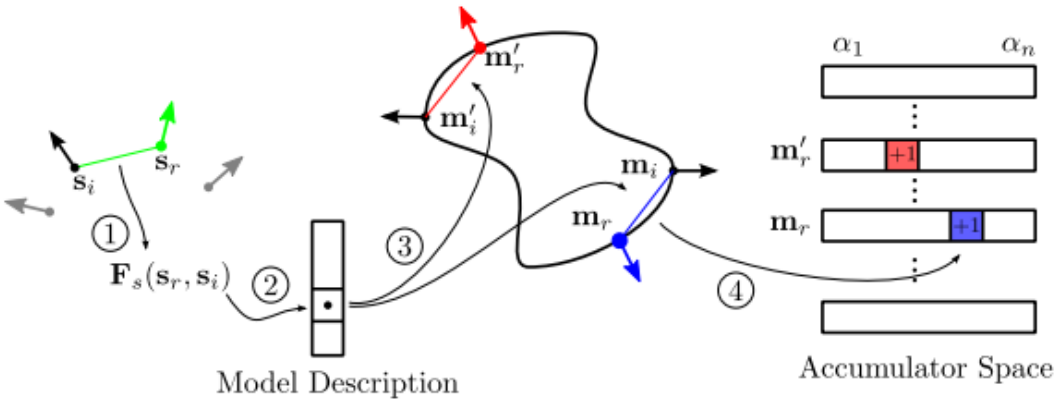


Figura 22. Esquema del proceso de votación.

Una vez se hayan procesado todos los puntos s_i, el máximo en el acumulador corresponde con la coordenada local óptima, con la que se puede obtener la transformación entre modelo y escena. En la implementación se incluye la optimización descrita en el documento de referencia [6], donde se divide la rotación alrededor del eje x para acelerar el proceso.

3.5.4 Estimación de la pose

El esquema descrito solo funcionará si el punto de referencia escogido pertenece al objeto, y, además, debido a la discretización realizada en las fases anteriores, la pose final será solo una aproximación. Para resolver estos dos problemas, se escogen varios candidatos de objeto dentro de la escena con la intención de asegurar que alguno pertenece a la instancia del objeto. Las transformaciones para cada posible punto de referencia se agrupan si son similares y se promedian, ordenándolas además según el número de votos para cada una.

Además, ya que debido a oclusiones o ruido en la captura de nubes de puntos la pose puede diferir de la real, se aplica el algoritmo de ICP para calcular la pose final de cada grupo de forma más robusta. Emplearemos este algoritmo tal y como viene implementado en [8], que incluye las siguientes optimizaciones:

- Extracción de un subconjunto de puntos para disminuir la carga de procesamiento mediante [40], que usa el espacio de vectores propios para seleccionar los puntos que afecten a la translación y la rotación.
- Para la búsqueda de correspondencias se emplea el algoritmo *Picky ICP* [41], que en el caso de haber más de una escoge la de la mínima distancia.
- Se rechazan las parejas que tienen una diferencia mayor a un umbral basado en la desviación estándar.
- Como se describe en [42], se utiliza una métrica del error basado en la estimación de planos.
- Aplicación de varias resoluciones distintas, comenzado las primeras iteraciones con menos puntos para ir aumentando la resolución según se va refinando la pose final.

Cada grupo contiene una posible instancia del objeto en una pose determinada, pero ya que nosotros sabemos que trabajamos con una única pieza, nos quedaremos con el resultado de mayor número de votos.

3.6 Comunicación entre módulos

La implementación del sistema propuesto supone un caso de combinación de diversas tecnologías, lenguajes y *frameworks*, sobre todo durante una etapa inicial de pruebas y comparación de estas. Por ejemplo, la API de la cámara con la que se trabaje puede estar disponible en un lenguaje como C++, mientras que la segmentación 2D se realice con un script de Python y querremos una interfaz desarrollada con el *framework* .NET en C#, o la propia herramienta Unity.

La interoperabilidad de los módulos se convierte entonces en un problema, pues durante las mismas pruebas se pueden escribir los datos a disco para compartir de un proceso a otro, pero no es una solución viable en el despliegue del sistema en producción. No sólo a causa del tiempo que eso añadiría en la ejecución de cada ciclo y el deterioro del hardware sobre el que se ejecutase (pues, en general, realizar muchas operaciones de lectura y escritura reduce la vida útil de un dispositivo de almacenamiento interno), sino por la dificultad añadida en tener que serializar todo dato que quiera ser compartido para poder almacenarlo en disco. Poner como condición que todo módulo implementado deba tener como nexo de unión mínimo el lenguaje sobre el que esté desarrollado, o poner a disposición *bindings* para este, condicionaría demasiado al desarrollador a la hora de poder intercambiar módulos, sobre todo pensando a la hora de querer escoger una cámara con una interfaz distinta.

3.6.1 Selección de tecnología

Se tiene en cuenta la posibilidad de desarrollar la aplicación en el *framework* ROS [43]. ROS es una plataforma de desarrollo para aplicaciones robóticas de uso muy extendido, uso que también se extiende a aplicaciones de Visión Artificial. El motivo por el que se valora esta plataforma es el sistema de comunicación entre módulos que proporciona, basado en el modelo publicador/suscriptor. De esta forma, se asegura que los módulos estén desacoplados entre sí, pudiendo intercambiar uno por otro simplemente respetando los mensajes que debe recibir y/o generar. Además, ofrece soporte para C++ y Python, por lo que un gran número de aplicaciones podría desarrollarse en cualquiera de estos lenguajes. No obstante, se descarta esta opción por la falta de conocimientos del entorno, y porque para el caso de uso especificado en este proyecto no serán necesarias ninguna de las otras funcionalidades que aporta este *framework*. Por otro lado, para la captura de datos de la escena se valora el uso de una cámara estéreo con una API disponible para .NET, y, además, el simulador en Unity también trabaja sobre este entorno. Esto último se tiene en cuenta ya que, si la tecnología lo permite, se podría crear una cámara virtual en este mismo entorno desde el que mandar datos generados sintéticamente. Se considera el uso entonces de un método más general que no impida poder desarrollar módulos en lenguajes distintos a Python o C++.

Pese a descartar ROS como *framework* de desarrollo, se adopta su filosofía del paradigma de publicador/suscriptor por las ventajas que ello ofrece, y se plantea un sistema similar al mismo. En este caso, se propone la creación de mensajes propios (un mensaje tipo JSON que contiene propiedades que describen el contenido y un campo para los propios datos) que sean enviados entre módulos con alguna herramienta que facilite la comunicación entre procesos.

En una primera fase se opta por MQTT [44], un protocolo de comunicación entre máquinas de uso bastante común en el ámbito del Internet de las Cosas. No obstante, tras implementarlo, se descarta por no estar diseñado para tamaños de datos tan grandes como los que se pretenden intercambiar entre módulos en este caso particular. En números, estamos hablando de aproximadamente 15MB para el caso de una nube de puntos antes de ser serializada, cuando en MQTT se suele trabajar con datos del orden de kilobytes.

En este desarrollo se escoge, finalmente, el uso de REDIS [45], que crea un almacenamiento de datos disponible para todos los procesos de una máquina en su memoria interna. Esto ofrece mucha rapidez en la comunicación, llegando a producirse de forma prácticamente inmediata.

3.6.2 Arquitectura del sistema

Se propone una arquitectura como la que se ve en la Figura 23. La cámara del sistema (que será controlada por el usuario con una aplicación de escritorio mediante su propia interfaz gráfica) publicará la información de la escena en tópicos establecidos para ello. Estos tópicos tendrán la información del mapa de brillo, de puntos 3D y de las normales correspondientes. A estos tópicos estará suscrito el módulo que realiza la detección y estimación de pose, que cuando tenga toda la información disponible iniciará la ejecución de los algoritmos. El resultado (la pose del próximo spinner a extraer) será publicada en un tópico específico para ello, al que se podrá suscribir cualquier agente que use la pose del objeto para realizar alguna acción. En el caso del prototipo realizado, se suscribirá a la interfaz gráfica para poder visualizar el resultado. A su vez, esta interfaz estará suscrita a los tópicos de información de la escena, de forma que podamos ver la nube de puntos de entrada y la pose de la pieza tras el procesado.

No obstante, se querrá visualizar el modelo 3D del spinner ubicado en la pose indicada como resultado. Ya que el modelo 3D tendrá que ser entonces un recurso compartido por el módulo de estimación de pose y el de la interfaz gráfica, se propone guardarlo en memoria compartida para poder ser accedido por los dos procesos.

Por último, cambios en los parámetros del módulo de detección y estimación de pose se podrían realizar a través de la suscripción a un nuevo tópico de configuración. En este tópico se podría, por ejemplo, publicar mensajes de configuración a través de la interfaz gráfica.

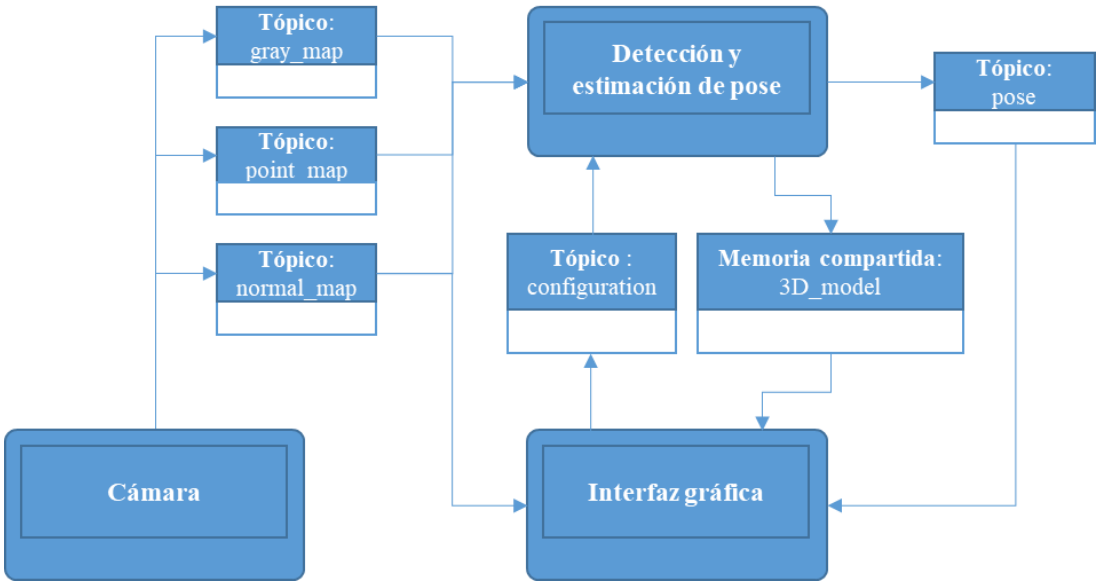


Figura 23. Esquema propuesto para la comunicación básica entre módulos.

En la Figura 24a se puede observar la facilidad que proporciona la arquitectura del sistema para intercambiar módulos. Por ejemplo, durante la fase de experimentación de este proyecto se han empleado imágenes de la escena almacenadas en archivos. Normalmente se tendría que reescribir los scripts que realizan el proceso de detección y estimación de pose para que lean los archivos desde disco en lugar de obtenerlos de una cámara. Sin embargo, con la arquitectura presentada no habría que modificar este módulo. Sólo es necesario generar un script que lea los archivos y los publique en los mismos tópicos en los que publica la cámara, un script sencillo de escribir y que no supone la modificación de ningún otro elemento del sistema.

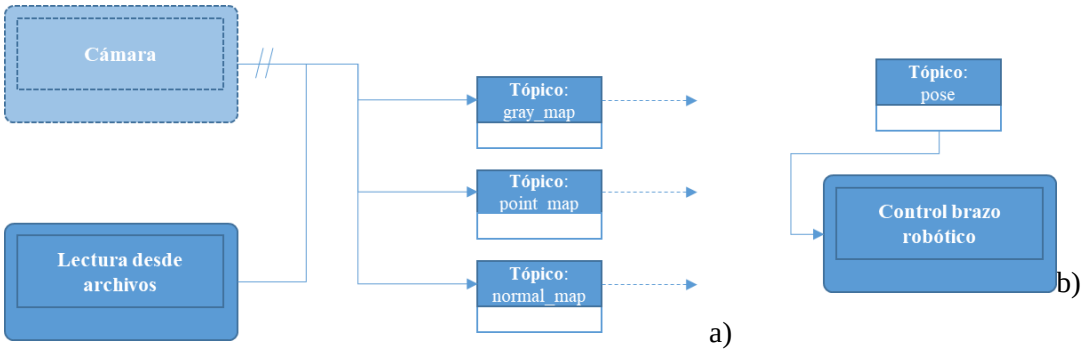


Figura 24. a) Sustitución del módulo de captura desde la cámara por lectura desde archivos. b) Suscripción de nuevos módulos.

De la misma forma, añadir nuevos módulos que se suscriban o publiquen en los tópicos existentes es sencillo (Figura 24b). Por ejemplo, además de visualizar la pose (mediante una interfaz suscrita a este tópico), también se querrá enviar al agente que vaya a interaccionar con los spinners. Para ello, simplemente suscribiremos al módulo que realice esta comunicación. En el caso de *bin picking*, seguramente se deba suscribir al controlador de un brazo robótico para que pueda leer las poses detectadas.

Es cierto que la complejidad del sistema aumenta debido a que es necesario establecer una serie de mensajes y su estructura. No obstante, una vez definidos, el sistema es mucho más flexible para intercambiar módulos, por lo que se considera que el tiempo invertido inicialmente tiene una recompensa muy alta a la hora de mantener el sistema o adaptarlo a nuevas líneas de.

Para el sistema actual se emplea un modelo de mensaje escrito en formato JSON. Salvo aquellos mensajes más básicos de configuración, todos los mensajes que querrán ser enviados entre los módulos del sistema serán, fundamentalmente, matrices de dos dimensiones (una imagen, una matriz de transformación…). La información contenida en la matriz será almacenada en el JSON como una cadena de caracteres, ya que REDIS sólo permite enviar o almacenar este tipo de datos. Para ello, las matrices se reducen a una lista de una única dimensión y se codifican en base 64, lo cual supondrá sólo un aumento del tamaño de los datos en un 33%. El resto de los campos del JSON definirán la resolución de dicha matriz y el tipo de dato de cada elemento (es decir, si es un entero de 8 bits como en el caso de las imágenes en escala de grises o un flotante de 32, por ejemplo). De esta forma, se es capaz de codificar y decodificar lo mensajes en cada módulo pudiendo hacer el envío a través de REDIS.

Figura 25. Ejemplo de mensaje.

4 Experimentación

Para la puesta a prueba del sistema desarrollado se propone la utilización de un sensor 3D del tipo par estéreo y una configuración como la que se ve en la Figura 26. El sensor se encuentra a aproximadamente 50 cm de la base del contenedor de piezas y con una inclinación de 20º respecto al eje vertical (Figura 26), posición desde la que se ve completamente su contenido y se maximiza la superficie de la escena que se expone al sensor. Se seleccionará una región de interés para recortar la imagen, de forma que se extraigan elementos del fondo que no aporten nada a la información de la escena (como los soportes).

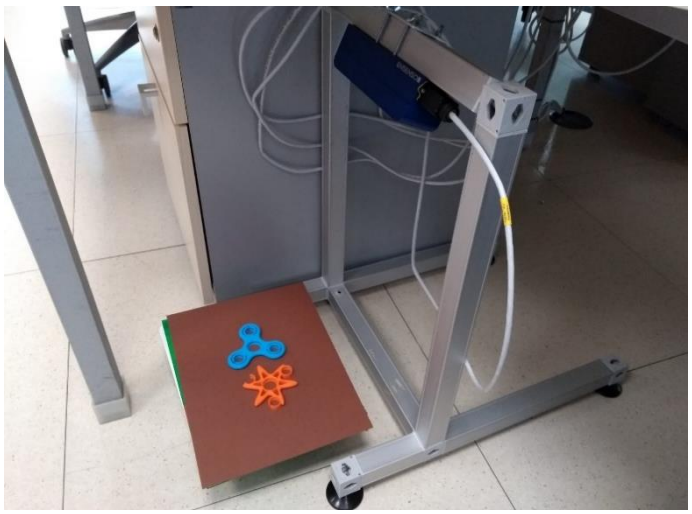


Figura 26. Sistema de captura de datos para la experimentación.

Las pruebas realizadas serán analizadas cualitativamente mediante la visualización de los resultados. El principal motivo para no realizar un estudio del error cometido de forma cuantitativa es la falta de instrumentos para conocer la pose de un objeto en la escena, ya que para poder dar una métrica del error es necesario conocer la pose real de los mismos y compararla con la que ha sido estimada.

Otra opción reseñable consistiría en crear un simulador con el que generar nubes de puntos de forma sintética, pero en este proyecto se decide realizar las pruebas del sistema con datos reales y la creación de un simulador de nubes de puntos no será cubierto. Se propone realizar una estimación numérica del error en posibles ampliaciones futuras.

4.1 Datos para la experimentación

Para el estudio experimental del sistema emplearemos, principalmente, el modelo de spinner básico de 3 bolas, por lo que por defecto se hablará de este modelo mientras no se indique lo contrario.

Con la configuración explicada en los anteriores apartados se capturará una serie de datos de la escena, buscando que recoja las posibles situaciones en las que se pueda encontrar el contenedor. Se tendrán ejemplos de casos en los que dentro del contenedor haya:

- Un único spinner.
- Varios spinners del mismo color.
- Varios spinners del mismo color apilados y ocluyéndose entre ellos.
- Modelos de spinners distintos.

Se han generado un total de 42 imágenes de escenas que contemplen estas situaciones.

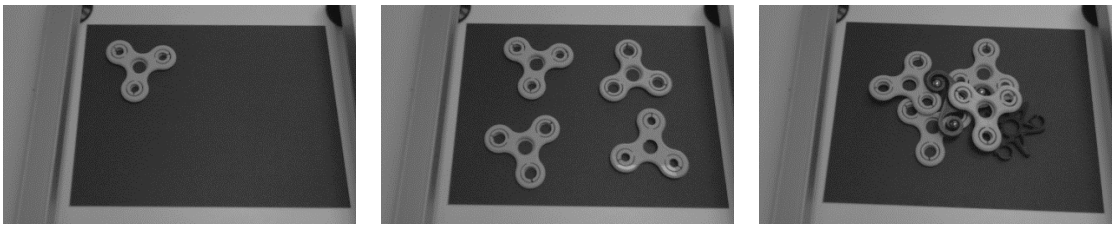
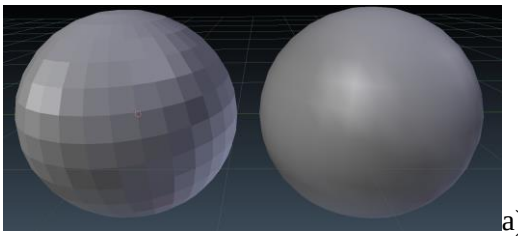


Figura 27. Ejemplos de datos para la comprobación del funcionamiento.

4.2 Adaptación previa del modelo

Los modelos 3D de los spinners han sido diseñados originalmente con el software de CAD SolidWorks [46] y exportados a archivos de tipo .STL, archivos con los que se trabaja en este proyecto. Ha sido necesario, sin embargo, realizar un proceso de adaptación de los modelos para solventar ciertos problemas derivados del tipo de topología que tienen las mallas de los spinners en el archivo de origen con el fin de visualizarlos correctamente.

4.2.1 Corrección de la malla para el renderizado.


a)

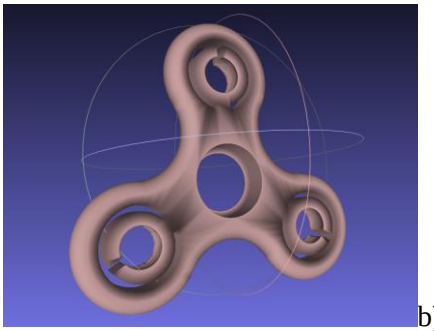

b)

Figura 28. a) Visualización de una esfera usando un sombreado por cara a la izquierda y suavizado a la derecha b) Visualización suavizada con problemas derivados de la triangulación del modelo.

Para la visualización de un modelo 3D se puede realizar un suavizado de las aristas durante el proceso de renderizado o visualizarlas en base a la orientación de las caras. Sin entrar en más detalles del proceso de renderizado, para el cálculo de iluminación en un punto determinado es necesario conocer la orientación del mismo. Esta orientación, determinada por un vector normal, puede calcularse de varias formas. Las más típicas consisten en emplear o bien la normal de la cara a la que pertenece, o bien una interpolación de las normales en los vértices que delimitan dicha cara. En la Figura 28a se puede observar la diferencia entre las dos técnicas. Para no generar posibles puntos característicos en las aristas debido al efecto de la visualización se empleará la segunda en este proyecto, que además es más realista.

En la Figura 28b se observa el modelo original de un spinner renderizado mediante una técnica de sombreado (*shading*) suavizado, donde existen unos artefactos que oscurecen el modelo en ciertas zonas. Esto no es debido a una mala definición de la orientación de los vértices en el modelo, si no a la interpolación que se realiza en puntos dónde no debería realizarse un suavizado del sombreado. Esto, sumado a la topología actual de la malla (con grandes triángulos distribuidos de forma aparentemente arbitraria, debido a la generación automática del mallado al realizar la exportación desde *SolidWorks*) genera una serie de artefactos en el proceso de visualización. Estos artefactos serán problemáticos a la hora de desarrollar el detector 2D, el cual será entrenado con renderizados de estos modelos.

Para solventar este problema sin tener que redefinir la topología de la malla, se utilizará un programa de modelado 3D en el que marcaremos aquellas aristas donde no deba realizarse un suavizado como “afiladas” (*sharp edges*). En la Figura 29a las vemos resaltadas con un color más

claro que el resto de las aristas de la malla, viendo el resultado de esta operación en la Figura 29b. Para realizar esta adaptación del modelo se ha empleado la herramienta Blender [33], software gratuito de modelado 3D.

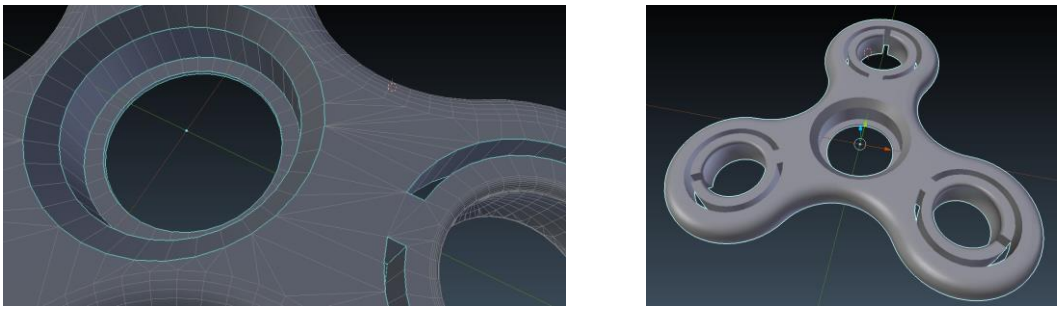


Figura 29. Arreglo de la malla para su visualización. a) Modelo con malla seleccionada, aristas marcadas como "afiladas" (*sharp edges*) en color más claro. b) Modelo con normales suavizadas de forma correcta.

4.2.2 Extracción de una nube de puntos.

Para el proceso de estimación de la pose se necesita una nube de puntos orientada del modelo. Una opción sería emplear el propio modelo original de la pieza, donde la nube de puntos estaría definida por los vértices y sus respectivos vectores normales. Sin embargo, en la Figura 30b se visualizan resaltados los vértices del modelo, y se observa a simple vista la poca uniformidad de la nube de puntos resultante. Los vértices están ubicados en aquellos puntos dónde la superficie del modelo sufre más cambios (por ejemplo, en las zonas curvas), encontrándose las zonas planas prácticamente despobladas.

Para tener una nube de puntos mejor distribuida se aplica una operación de muestreo uniforme a la malla que conserve las orientaciones de los nuevos vértices. Esta operación se realiza con Meshlab [47], una herramienta gratuita para la edición de mallas. La operación realizada consiste en la construcción de una representación volumétrica uniforme generada mediante la aplicación de un algoritmo de *marching cubes* ([48], [49]) sobre el volumen del modelo original. En la Figura 30d se puede ver el nuevo modelo con los vértices resaltados, dónde ya se visualiza un muestreo mucho más uniforme. Estos vértices constituirán la nube de puntos del modelo empleado para la estimación de pose.

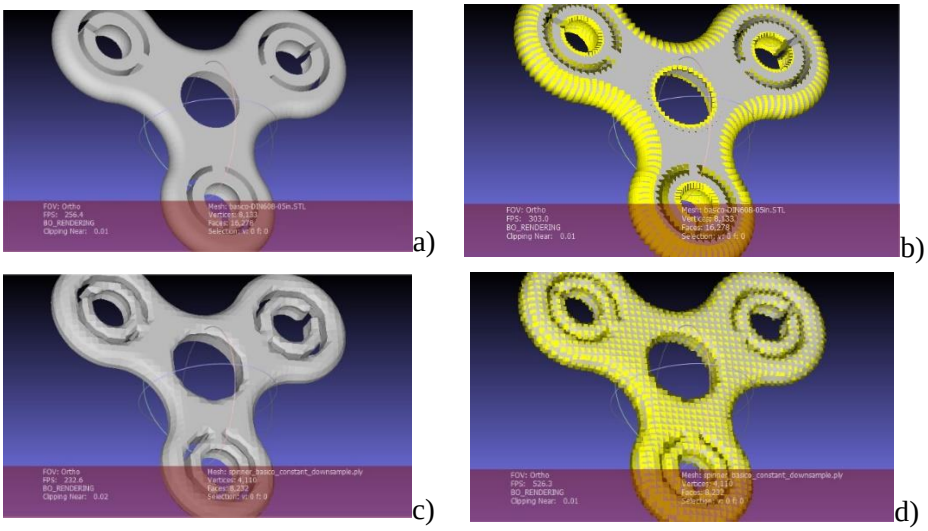


Figura 30. Modelo original: a) malla b) resaltado de vértices. Modelo muestreado uniformemente: c) malla d) resaltado de vértices.

4.3 Definición de pruebas para comprobar el funcionamiento de módulos

4.3.1 Detección de spinners

Se creará una serie de conjuntos de imágenes etiquetadas mediante el simulador desarrollado, realizando variaciones para cada conjunto. Los conjuntos creados se definen en la Tabla 4. Por cada conjunto de datos se seleccionará un único modelo de spinner o varios mezclados, aplicando en el caso que corresponda una modificación aleatoria en las condiciones de captura. Estas modificaciones afectarían a posición de la cámara, la posición de los puntos de iluminación o el color de los spinners.

	Aleatoriedad		
Modelo(s)	Cámara	Iluminación	Color
basico-DIN608-16mm	X		
basico-DIN608-16mm	X	X	
basico-DIN608-16mm	X	X	X
atomo-DIN608-16mm	X	X	X
trisquel-DIN608-16mm	X	X	X
mezcla	X	X	X

Tabla 4. Conjuntos de datos generados.

Se generarán un total de 150 imágenes para cada conjunto, con un número de spinners instanciados que pueden variar desde los 5 a 25 por escena (a no ser que no sean visibles por la cámara, entonces el número podría ser menor). Sólo se etiquetarán aquellos spinners de los que sea visible el 90% de su superficie. El plano sobre el que se dejarán caer los spinners es de un color uniforme, con una ligera textura rugosa para que no se vea completamente liso desde la cámara. El color, verde, ha sido seleccionado arbitrariamente, con la única intención de presentar contraste con los spinners.

Estas imágenes serán empleadas para entrenar al detector 2D que segmentará las regiones donde se encuentren spinners. Con cada conjunto se generará un modelo, y utilizando las imágenes de prueba descritas anteriormente se analizará el funcionamiento del detector. El análisis se realizará de forma cualitativa, observando las máscaras devueltas por el detector y comprobando que realmente segmenten las regiones de los spinners del fondo.

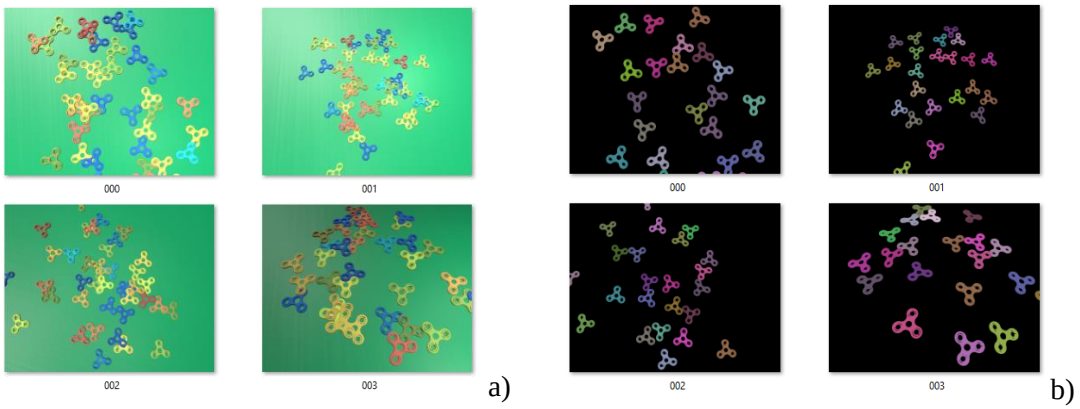


Figura 31. Ejemplo de imágenes para crear un modelo de detector. a) Imágenes generadas. b) Máscaras correspondientes.

4.3.2 Estimación de pose

Se aplicará el módulo de estimación de pose sobre las nubes de puntos obtenidas de la escena. La segmentación de dichas nubes de puntos, información que debería haber sido proporcionada por el anterior módulo, será realizada manualmente con el fin de testear este módulo de manera

individual, independientemente de la corrección de los resultados del módulo anterior. Mediante el editor de imágenes GIMP [50] se dibujarán las máscaras, utilizándolas posteriormente para recortar las nubes de puntos de la escena correspondiente.

Se obtendrán, por lo tanto, nubes de puntos que contienen información 3D de un único spinner. Se han capturado y recortado manualmente un total de 7 nubes de puntos, con las que se comprobará que el módulo realiza una estimación de la pose de los spinners válida. Esta comprobación se realizará de visualmente, comprobando que la nube de puntos del modelo tras aplicar la transformación calculada coincide con la pose que tiene el spinner en la escena.

4.3.3 Funcionamiento global del sistema

De los modelos de detección de spinner previamente entrenados, se escogerá aquel que haya presentado el mejor rendimiento. Se comprobará el funcionamiento global del sistema realizando una puesta en marcha de este, obteniendo imágenes y analizándolas posteriormente. Se realizarán una serie de iteraciones para comprobar el correcto funcionamiento del sistema.

4.4 Especificaciones hardware

A continuación, se definen las especificaciones hardware de los equipos con los que se ha realizado el proyecto:

Ordenador personal para desarrollo y pruebas de ejecución:

- Modelo: Lenovo Thinkpad W550s
- CPU: Intel Core i7-5600U @ 2.60GHz
- GPU: NVIDIA Quadro K620M
- RAM: DDR3 1600 12GB

Ordenador para entrenamiento de los modelos 2D:

- Construcción propia con componentes individuales.
- CPU: Intel Core i7-7700K @ 4.50GHz
- GPU: NVIDIA GTX 1080 8GB
- RAM: DDR4 2600 32GB

Tabla 5. Especificaciones hardware de los equipos.

No se tendrá en cuenta el tiempo de ejecución de los módulos para esta fase de experimentación. Sin embargo, se sabe que la duración de un ciclo completo con el ordenador de sobremesa se aproxima a los 5 segundos, y oscila en torno a los 25 en el portátil personal.

El uso de una tarjeta gráfica resulta muy interesante para módulos que puedan emplear aceleración hardware mediante CUDA [51], como es en el caso de *TensorFlow*.

5 Resultados

En los siguientes apartados se hará un resumen de los resultados obtenidos durante la fase de experimentación, haciendo hincapié en aquellos que describan el funcionamiento del sistema de forma más significativa.

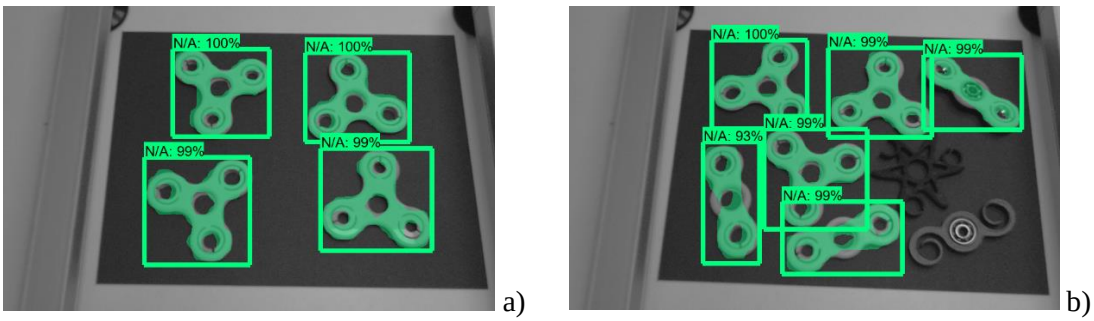
5.1 Detección de spinners

5.1.1 Detección de modelos básicos

Los modelos entrenados con imágenes en las que no se ha introducido variabilidad en las condiciones lumínicas, de perspectiva o de color, no son capaces de detectar piezas en las imágenes realizadas en condiciones reales.

Sin embargo, una vez se introduce esta variabilidad en el conjunto de imágenes de entrenamiento, los spinners del modelo basico-DIN608-16mm son detectados y segmentados correctamente. En la Figura 32a vemos un ejemplo de la segmentación realizada con el detector. Se observa, además de la correcta detección de las instancias de los spinners, como la máscara se adapta a la superficie del spinner. Se conservan, incluso, los cuatro agujeros presentes en la geometría.

El submódulo, sin embargo, detecta también piezas básicas de dos bolas. Un ejemplo de caso en el que se tienen estos falsos positivos puede verse en la Figura 32b. No obstante, spinners de otros modelos, como el atomo-DIN608-16mm o el trisquel-DIN608-16mm, no activan el detector, como puede verse en esa misma figura (y en el resto de las imágenes de prueba).



a) b)

Figura 32. Ejemplos de detección de spinners.

5.1.2 Detección de otros modelos

Las instancias del modelo de atomo-DIN608-16mm son detectadas, en general, correctamente. Se han detectado todas las instancias presentadas en el conjunto de datos de prueba, pese a que los spinners sea vean algo más oscuros a como se han generado en las imágenes sintéticas.

La segmentación, no obstante, no separa completamente la figura del fondo en la mayoría de los casos. En la Figura 33a se observa un ejemplo de funcionamiento en el que la pieza se ha detectado en la posición correcta, pero la máscara no se adapta perfectamente a la superficie del spinner. Este problema está aún más remarcado en el ejemplo de la Figura 33b, donde la parte superior del spinner queda fuera de la región devuelta por el detector.

Otra situación que plantea este modelo entrenado para las piezas de tipo atomo-DIN608-16mm es la detección de spinners del tipo basico-DIN608-16mm. Se detectan prácticamente todas las instancias de este último modelo en las imágenes de prueba, como puede comprobarse en las dos figuras mencionadas previamente (Figura 33a y b). La única diferencia con el detector entrenado para este modelo son las regiones que devuelve el sistema, que no se adaptan perfectamente a la superficie del spinner.

Además de detectar este tipo de piezas de forma común, la detección con este modelo presenta muchos más falsos positivos que su predecesor. Por ejemplo, detecta muchas regiones pequeñas en otras piezas que clasifica como instancias del modelo atomo-DIN608-16mm. En el ejemplo de la Figura 33c se observa como detecta como spinner uno de los brazos del modelo trisquel-DIN608-16mm.

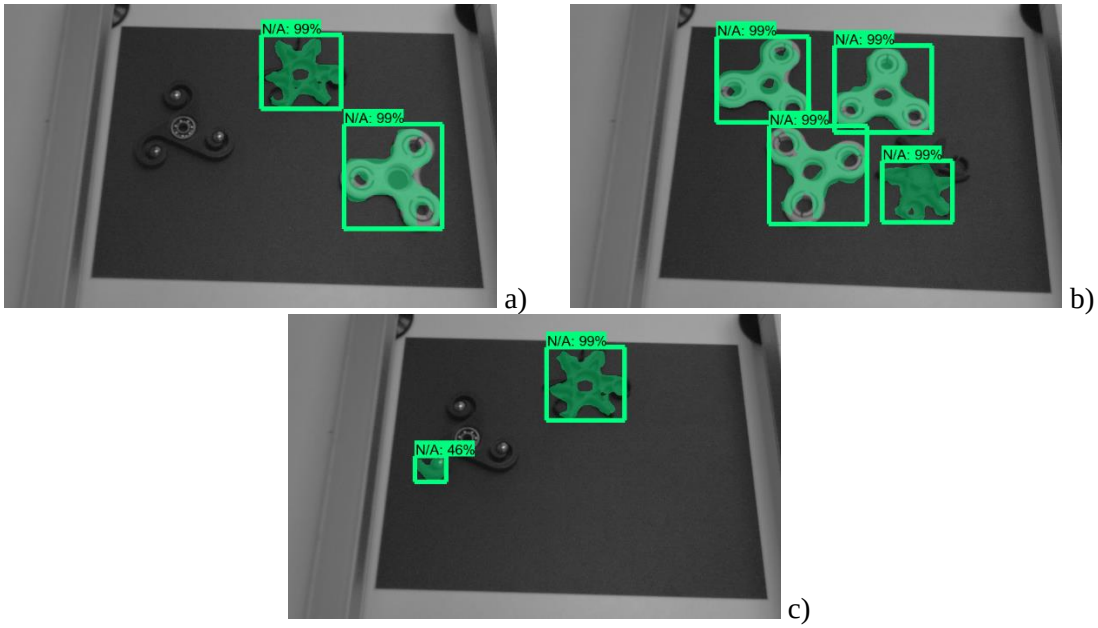


Figura 33. Ejemplos de detección con el modelo atomo-DIN608-16mm.

Por último, los resultados del modelo trisquel-DIN608-16mm y los obtenidos con la mezcla de todos ellos son negativos. El detector que emplea el modelo del trisquel-DIN608-16mm produce detecciones positivas en instancias de los demás tipos de spinners menos en el que corresponde. El modelo que se ha generado para este tipo de spinner se define, por lo tanto, como no funcional, ya que no ha sido capaz de realizar ninguna detección correcta en todo el conjunto de los datos de pruebas.

Por último, las pruebas realizadas entrenando el modelo con todos los tipos de spinners de forma simultánea también ha resultado negativa. Este detector clasifica la mayoría de spinners como tipo básico-DIN608-16mm, y algunos otros casos como atomo-DIN608-16mm. Sin embargo, esta clasificación resulta prácticamente arbitraria, además de presentar un índice de detecciones mucho más bajo que con el resto de las modelos. En la Figura 34a y b se comprueban estos hechos. En la primera sólo se detecta y clasifica correctamente el modelo básico-DIN608-16mm, aunque el resto de spinners de la escena no son detectados. Sin embargo, en la siguiente figura se demuestra el funcionamiento incorrecto del detector, ya que no detecta correctamente ninguno de los spinners de la escena.

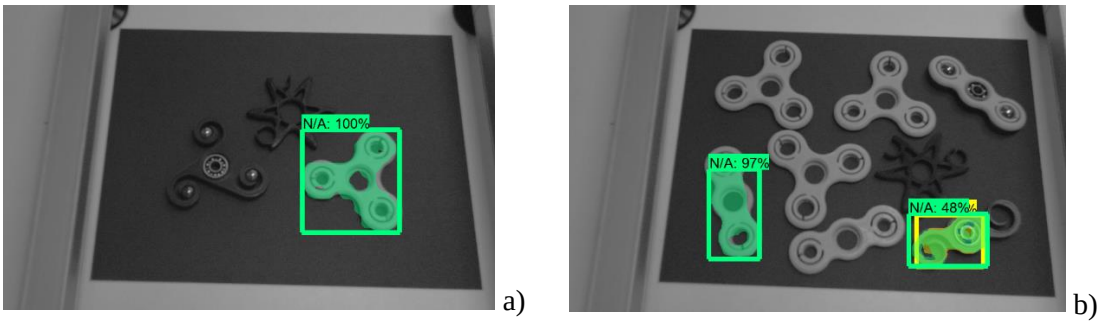


Figura 34. Ejemplos de detección con detector con el modelo trisquel-DIN608-16mm (a) y mezcla (b).

5.1.3 Resumen de resultados

Con los modelos entrenado con los conjuntos de datos en los que la iluminación o el color se han mantenidos fijos no se detectan spinners. Al introducir esta variabilidad sí que se detectan, aunque los resultados varían según el modelo. Estos resultados pueden verse descritos en la Tabla 6.

Modelo(s)	Comentarios
basico-DIN608-16mm	• Se detectan correctamente la mayor parte de los spinners del tipo correspondiente. • Los spinners ocluidos no son detectados. • La máscara presenta agujeros que coinciden con la geometría del spinner. • Falsos positivos detectando spinner básicos de dos bolas. • Falsos positivos detectando del tipo atomo-DIN608-16mm, aunque con baja puntuación de confianza.
atomo-DIN608-16mm	• Se detectan correctamente la mayor parte de los spinners del tipo correspondiente. • Los spinners ocluidos no son detectados. • Error al segmentar la geometría completa del spinner: la máscara no abarca toda su superficie. • Falsos positivos detectando del tipo basico-DIN608-16mm y el correspondiente de dos bolas.
trisquel-DIN608-16mm	• No detecta spinners del tipo correspondiente. • Falsos positivos detectando del tipo basico-DIN608-16mm.
mezcla	• En general, sólo detecta spinners del tipo basico-DIN608-16mm, y en menor número de ocasiones.

Tabla 6. Resultados del módulo de detección de spinners.

5.2 Estimación de pose

Tras analizar visualmente los resultados obtenidos en la fase de experimentación, se comprueba que la estimación de pose en todas las pruebas ha resultado correcta. El modelo se adapta completamente a la superficie de la nube de puntos capturada por la cámara tras aplicarle la transformación estimada.

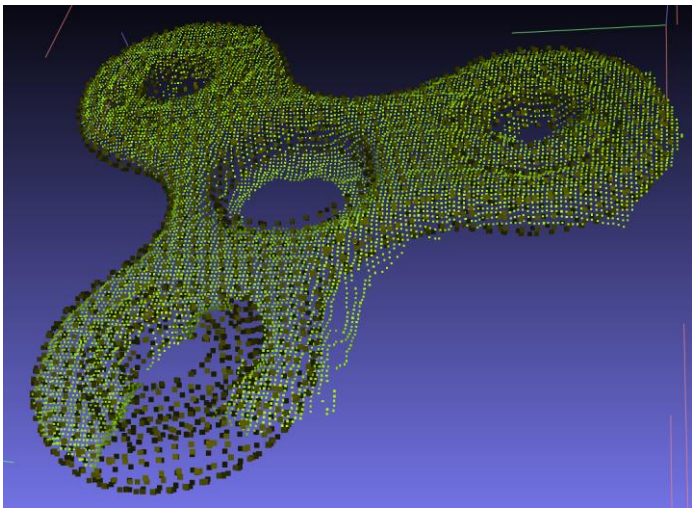


Figura 35. Visualización de una nube de puntos de la escena recortada (amarillo) y la nube de puntos del modelo alineada correctamente (gris).

Un ejemplo de resultado puede verse en la Figura 35, donde se visualiza la nube de puntos de la escena recortada (amarillo) y la nube de puntos del modelo transformada con la pose estimada (gris).

No obstante, durante la fase de desarrollo se han observado casos en los que la pose del spinner se detecta de forma errónea. Generalmente, estos fallos al estimar la pose siguen un patrón común. Dos de los agujeros de los brazos coinciden, quedando el último orientado en la dirección opuesta (Figura 36). Hay casos en los que, con la misma escena, la estimación de pose da un resultado correcto y en la siguiente iteración uno como el que se ve en la figura.

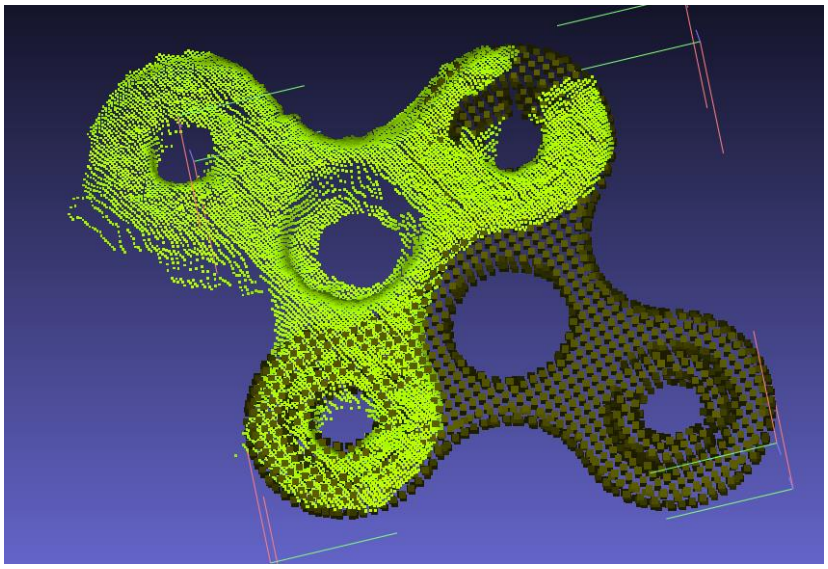


Figura 36. Visualización de una nube de puntos de la escena recortada (amarillo) y la nube de puntos del modelo alineada incorrectamente (gris).

Cabe destacar la importancia de la resolución con la que se toman muestras del modelo. El tiempo de cómputo crece exponencialmente a mayor número de puntos, y si la nube de puntos está muy poblada los tiempos de ejecución resultan demasiado altos como para plantearlos en un sistema de bin-picking.

5.3 Funcionamiento global del sistema

Una vez realizadas las pruebas por separado de los distintos submódulos del sistema y tras afinar sus parámetros de funcionamiento, se ha implementado la arquitectura completa tal y como se especifica en este documento.

El sistema cumple con los requisitos establecidos, ejecutándose correctamente durante varios ciclos sin presentar problemas de comunicación entre los distintos submódulos. Se ha probado además a lanzar y cerrar procesos de ejecución de forma aleatoria, comprobando que un submódulo funciona de forma completamente desacoplada del resto. También se ha comprobado que se puede sustituir un submódulo por otro en tiempo de ejecución sin afectar al funcionamiento del sistema, pudiendo lanzar un proceso que lea nubes de puntos desde archivos en lugar de capturarlos desde la cámara. La comunicación es correcta y prácticamente instantánea.

Además, y para el caso del modelo básico-DIN608-16mm, las detecciones han sido correctas en la mayoría de las situaciones, exceptuando algunos ejemplos como los descritos anteriormente en los que la estimación de la pose ha resultado errónea.

Se concluye, por lo tanto, que la arquitectura del sistema general funciona. En la Figura 37 se puede observar la detección y estimación de pose de un spinner en un ejemplo extraído del conjunto de pruebas.

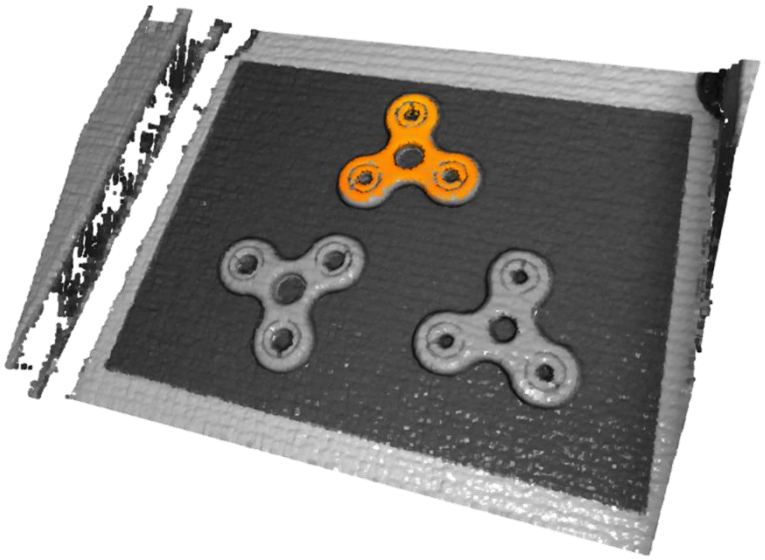


Figura 37. Ejemplo de resultado del sistema de bin-picking

6 Discusión

Si bien quedan problemas por resolver, tal y como puede verse en los resultados, el funcionamiento general del sistema resulta satisfactorio. El spinner basico-DIN608-16mm es detectado y su pose es estimada correctamente en la gran mayoría de los casos. Además, parece que el funcionamiento del sistema con el resto de los tipos de spinners se podría asegurar tras ciertas modificaciones en el módulo de detección de piezas.

En el caso del proyecto para el que este sistema fue creado originalmente, el brazo robótico interaccionaba con la mayoría de las piezas de forma correcta, incluso con piezas añadidas posteriormente por los operarios una vez puesta en marcha la célula. En este caso de uso no se ha podido comprobar el funcionamiento de un sistema de *bin-picking* completo, confirmando que otro agente pueda interaccionar con las piezas detectadas. Sin embargo, los resultados parecen indicar que es posible ya que ofrece, tras un análisis cualititatvo, unos resultados similares al del caso de uso del proyecto realizado durante las prácticas del alumno.

En los próximos apartados se hablará de cada submódulo de forma más específica, resaltando su funcionamiento actual y las posibles mejoras que podrían realizarse.

6.1 Funcionamiento de los submódulos

Un paso que debería incluirse en casi todos lo submódulos del sistema de forma común sería una validación de hipótesis. El sistema se basa en el funcionamiento consecutivo de una serie de módulos que proporcionan una funcionalidad específica y de cuyas salidas se alimentan los módulos posteriores. Si, por ejemplo, el detector de spinners detecta una región que incorrecta, el siguiente módulo de estimación de pose trabajará, innecesariamente, sobre una nube de puntos recortada que no se corresponde a una instancia de spinner. Es por ello necesario validar y escoger de entre todos los posibles candidatos de cada algoritmo aquel que supere un umbral de confianza, asegurando el resultado antes de ser enviado al siguiente módulo.

6.1.1 Captura de datos

Una posibilidad que explicaría el mal funcionamiento del sistema detectando spinners de tipo trisquel-DIN608-16mm es el nivel de brillo con el que se capturan dichas piezas.

Se ha observado que las figuras azules tienden a verse con valores de brillo muy altos mientras que el resto de los colores son capturados con mucha menos intensidad. A priori, esta posibilidad no había sido tenida en cuenta durante la elección del sistema de captura, pues se suponía que los sensores del par estéreo capturarían el brillo de la escena de una manera “estándar”. Sin embargo, podría darse el caso de que el sensor esté más expuesto a frecuencias correspondientes al color azul que al resto, mediante el uso de un filtro o un sensor específico para ello. Se realiza esta suposición en base a la tecnología de estéreo activo que la cámara emplea, ya que hace uso de un proyector que emite un patrón de luz azul. El fabricante puede haber amplificado la cantidad de luz detectada en esa banda de frecuencia frente al resto de colores buscando una mayor facilidad para distinguir el patrón del resto de la escena. Sin embargo, no se ha podido comprobar esta posibilidad mediante la documentación de la cámara, que no incluye información específica de su funcionamiento interno.

Debido a que el resto de los colores son capturados con menor intensidad en comparación con el azul, se puede observar cómo los spìnners atom-DIN608-16mm y trisquel-DIN608-16mm son capturados con valores de escala de gris muy bajos, mucho más bajos de aquellos con los que se han entrenado los modelos. Ya que el detector que se emplea se basa también en colores (o valores de brillo monocromáticos, en este caso), la detección de la pieza no se está realizando debido a la

diferencia entre los valores de entrenamiento y los de prueba. De todas formas, esta condición es sólo una hipótesis, y debería ser cotejada con más experimentaciones.

6.1.2 Simulador y detección de spinners

Se observa en los resultados que la detección y segmentación de los spinners es, en general, correcta. El modelo básico-DIN608-16mm se detecta en la gran mayoría de los casos, segmentando correctamente la superficie de cada instancia respecto al fondo. Existen esos falsos positivos para el caso del atomo-DIN608-16mm, pero debido a que una de las restricciones del sistema es que sólo exista un tipo de pieza en el contenedor al mismo tiempo, este hecho no resulta del todo preocupante. El problema se encuentra principalmente para el caso del modelo trisquel-DIN608-16mm, cuya detección no se ha podido realizar tal y como se ha visto en la experimentación. Aunque en el aparrado anterior se da un posible motivo para este problema, se requiere un estudio más exhaustivo para intentar darle solución.

No obstante, gracias al simulador se ha conseguido generar imágenes sintéticas suficientemente realistas como para que el modelo entrenado con ellas detecte spinners en escenas reales. Se ha comprobado que el concepto es funcional, y que otorga mucha flexibilidad al sistema. Por ejemplo, el proyecto original estaba diseñado originalmente para un tipo de piezas completamente distinto a los spinners, pero su adaptación ha resultado sencilla para otro caso de uso. Además, el hecho de haber implementado el simulador en Unity proporciona la posibilidad de continuar desarrollando el sistema de forma más rápida y sencilla que empleando otras tecnologías de más bajo nivel.

Por otro lado, en una siguiente fase sería interesante preparar el sistema de simulación para que genere también nubes de puntos. Actualmente, se ha probado el concepto aplicando una técnica de *raycast* como el que se emplea para asegurar la visibilidad de las piezas. No obstante, para acelerar procesos aprovechando la capacidad gráfica de la plataforma, existe la posibilidad de generarlas desde un *shader*. Se podría realizar una adaptación del código visto en [52] para obtener las coordenadas de los puntos en la escena en un sistema de coordenadas específico. En esa aplicación se calculan estas coordenadas 3D para realizar un efecto visual, pero podrían escribirse en una textura que representase la nube de puntos.

No obstante, respecto al módulo de detección en sí, habría que realizar un estudio y diseño más completo. Actualmente funciona para la mayoría de las situaciones, en función del tipo de imágenes generadas y del tipo de modelo en cuestión. Sin embargo, es posible que la problemática de las piezas que no pueden ser detectadas deba ser también analizada desde este otro punto del proceso y no sólo desde la generación de las imágenes. Diseñar la red, buscar otros modelos sobre los que basarse para hacer transferencia de aprendizaje, analizar el modelo para establecer una serie de restricciones o requisitos para el simulador… Hay una serie de necesidades no cubiertas en este proyecto para asegurar el funcionamiento completo de este módulo, que quedan propuestas para futuros desarrollos.

Está también la posibilidad de emplear la información de profundidad para realizar la detección de las piezas. Se podría emplear esta información buscando hacer hincapié en la geometría del objeto a la hora de realizar su detección, ya que actualmente se basa esencialmente en la textura de la imagen y los spinners carecen de ella. Además, generar datos de profundidad para el entrenamiento del sistema sería un cálculo fácilmente realizable con el simulador en el caso de ser necesario

6.1.3 Selección de pieza

Se ha creado un método de selección de pieza básico para poder proporcionar al módulo la capacidad de escoger la siguiente pieza a extraer sin hacerlo de forma completamente arbitraria. Sin embargo, actualmente, la selección de pieza se realiza de forma un poco “ingenua”,

seleccionando en función de la mayor área y el promedio de la altura de los puntos del spinner. No obstante, la solución no generalizaría bien para ciertas escenas. Por ejemplo, un spinner en la parte superior puede estar inclinado respecto a la cámara de forma que su área (debido al efecto de la perspectiva) sea mucho más pequeña que la de un spinner en la parte inferior del contenedor. Esto se intenta paliar mediante el uso de la información 3D, pero el hecho de emplear el promedio implica que, si ese mismo spinner está inclinado, el promedio puede quedar en dar lugar a valores más bajos de los que interesaría. Resultaría muy afectado, por ejemplo, por una segmentación incompleta (o que se exceda) de instancias en la escena, *outliers* producidos por el ruido o pequeñas superficies no reconstruidas por la cámara.

Este submódulo no ha dado problemas durante el desarrollo del proyecto, aunque habrá que tenerlo en cuenta con equipos de menor coste o en otras condiciones de la escena.

6.1.4 Estimación de pose

Se observa en los resultados que el módulo que realiza la estimación de la pose funciona correctamente. Serviría para el caso en el que se analizase la nube de puntos completa, pero gracias a la segmentación previa que se realiza se acotan los posibles resultados y se reducen los tiempos de cómputo. Además, esto hace que no sea necesario realizar un ajuste de parámetros muy estricto como se tendría que realizar en otro caso. Por ejemplo, para el caso de los spinners, con los parámetros por defecto los resultados son válidos. No obstante, la nube de puntos del modelo debe tener una resolución entre unos rangos específicos o la solución comienza a verse afectada, y es un ajuste que depende de cada modelo entrenado por el sistema. Eso obliga a adaptar el modelo 3D o el entrenamiento del estimador de pose, lo cual va en contra de la filosofía del sistema original: que entrenar nuevas piezas pueda ser llevado a cabo por cualquier operario sin conocimientos en visión por computador. Si bien es verdad que pueden darse una serie de guías para hacer que la adaptación a nuevas piezas esté más sistematizada, este sería uno de los puntos débiles del sistema en este aspecto.

Emplear ICP para refinar la pose da buenos resultados y la implementación realizada en OpenCV cumple con los requisitos exigidos. Una ventaja añadida es que, en el caso de ser necesario usar otras implementaciones, hay mucha bibliografía al respecto y está presente en cualquier librería que haga tratamiento de nubes de puntos, ya que es una técnica muy extendida.

El mayor problema puede derivar de la necesidad de acotar tiempos. Se está empleando una técnica usada para detectar instancias en una nube de puntos en que la ya se ha asegurado previamente que existe una y es única, además de encontrarse sólo los puntos que pertenezcan a esta. ¿Es, por lo tanto, necesario aplicar este tipo de algoritmo que requiere tanta potencia de cómputo? Es cierto que establece una relación entre puntos de modelo y escena, además de una pose inicial sobre la que ejecutar el algoritmo de ICP. Sin embargo, quizás se pueda emplear una forma más sencilla de buscar esta primera aproximación de pose. El problema estará en buscar que esta solución dependa lo menos posible del modelo 3D de la pieza, si es que esta es la intención.

6.2 Comunicación entre módulos

El método empleado para realizar la comunicación entre los módulos del sistema ha resultado ser muy útil durante la fase de desarrollo, destacando la facilidad con la que se pueden intercambiar unos módulos por otros para realizar las pruebas. Resulta, además, muy interesante mantener esta arquitectura para la posterior fase de producción.

El principal aspecto que debería ser revisado, en el caso de querer aumentar la flexibilidad y capacidad de ampliación del sistema, es el tipo de mensajes que se comunican entre los procesos.

Actualmente se envía una cadena de bytes codificada en base 64 e introducida en un JSON donde se define el contenido de esta (por ejemplo, indicando la resolución del mapa de brillo e indicando la longitud en bits de cada valor). Este formato de mensaje ha sido creado específicamente para esta aplicación, y en este contexto es una solución suficiente y válida. No es, no obstante, un formato de uso estandarizado, por lo que todo módulo que se pretenda añadir al sistema deberá conocer la topología de este tipo de mensajes. Aunque no supone un problema muy grave, también es cierto que la adopción de mensajes más estandarizados, como el envío de imágenes en formato PNG (por ejemplo), permitiría el uso de librerías que ya conocen ese formato y que recogen muchas más opciones de definición del contenido.

Para este sistema en concreto podría resultar interesante el tipo de archivo TIFF. TIFF permite valores negativos, de coma flotante y de varias capas. Podría ser útil para contener la información de la nube de puntos, ya que podría guardarse todo el contenido 3D en este mismo archivo.

Este tipo de codificaciones reducen, además, el tamaño de memoria que ocupan los datos, lo que resulta una ventaja a la hora de realizar la comunicación entre los módulos. Podría, incluso, valorarse el uso de otras tecnologías en el caso de que el mensaje a enviar fuese más pequeño. Una tecnología que sería interesante para poner a prueba es ZeroMQ [53] para la creación de un sistema de mensajería distribuido entre máquinas y/o procesos.

6.3 Otras soluciones y trabajo futuro

Si la solución va a ser específica para estimar la pose de spinners y la flexibilidad del sistema para adaptarse a objetos distintos no es un requisito, se pueden aprovechar las características comunes a todos los spinners para resolver el problema de forma más robusta. Dichas propiedades se extraen de la geometría específica de los spinners: la presencia de un número determinado de agujeros, superficies esencialmente planas en las caras principales del spinner, materiales de fabricación previamente conocidos… Por ejemplo, podría resultar interesante realizar una segmentación de la imagen basándose en regiones de color uniforme (como puede ser, la cara de un spinner), y posteriormente clasificar dichas regiones. Aplicar este tipo de características a la solución del sistema puede hacer que el sistema sea muy robusto detectando este tipo de figuras. De todas formas, habrá que tener en cuenta que se puedan introducir nuevos modelos de spinner, por lo que habría que buscar características comunes a todos ellos.

Por otro lado, en el estado del arte de este documento se han planteado otros sistemas usados para la detección y estimación de pose 3D de objetos, como LINEMOD [9] o PointNet [13]. Poner a prueba estos sistemas sería interesante, y se propone para futuros trabajos.

La validación del sistema se ha realizado cualitativamente, y si bien es verdad que los resultados son prometedores, un posterior análisis cuantitativo sería recomendable. Para ello, son necesarios datos y la definición de métricas para su evaluación. Sólo así se podrá conocer la fiabilidad del cada módulo y del sistema completo, además de poder ajustar los parámetros en busca de la optimización de todos los pasos del proceso. Se ha recabado información al respecto de cómo se podrían realizar estos análisis, aunque finalmente no se haya llevado a la práctica. Por ejemplo, en el documento [54] se presentan una métricas de evaluación para algoritmos que segmentan instancias de objetos de una imagen y que podrían ser empleadas para valorar el funcionamiento del detector de spinners,

Por último, también resultaría interesante realizar una ampliación sobre el simulador para mejorar el realismo de las imágenes obtenidas, comprobando si el incremento de realismo mejora el funcionamiento del detector de spinners. Podría conseguirse, por ejemplo, con técnicas como la creación de texturas procedurales para introducir relieve o detalles en las superficies, una mayor adaptación a posibles condiciones de contorno, emplear otro tipo de *shaders* o materiales…

7 Conclusiones

La investigación en el campo de Visión Artificial aplicada a información tridimensional ha ganado relevancia en los últimos años, debido en gran parte al avance y la disponibilidad de sistemas de percepción que proporcionen información 3D de la escena, y a la mejora en los tiempos de cómputo al procesar este tipo de información. En concreto, la detección de objetos 3D y la estimación de su pose en el espacio son dos de los campos de investigación más significativos de la robótica, ya que la capacidad de localizar objetos en su entorno dota a los robots de mucha autonomía.

El objetivo principal de este proyecto ha sido presentar el diseño de un sistema basado en Visión Artificial para realizar la detección de objetos 3D y la estimación de su pose en el espacio. Este sistema estaría adaptado al caso de un problema del tipo de *bin-picking*, donde objetos ya conocidos estarían distribuidos aleatoriamente en un contenedor. Para ello, se ha adaptado una solución diseñada durante otro proyecto en el que ha participado el alumno, pero para otro caso de uso, siendo este la detección de spinners dentro de la planta piloto de la Universidad de Oviedo. El sistema que se ha propuesto en este documento tenía dos objetivos fundamentales:

- **Extensibilidad**: detectar nuevas piezas distintas a las que se han usado durante el desarrollo, y que puedan ser introducidas por operarios que no tengan conocimiento extenso en el área de Visión Artificial.
- **Modularidad**: capacidad para integrar con facilidad nuevos componentes, ampliando la funcionalidad o reemplazando los módulos existentes del sistema. Por ejemplo, es posible reemplazar los sensores y soportes físicos empleados en este proyecto durante la fase de experimentación sin la necesidad de modificar el resto de componentes.

Se ha realizado un estudio de técnicas existentes que puedan aplicarse para un problema de este tipo, pasando por las soluciones completas al análisis de las tecnologías disponibles para cada funcionalidad concreta. En base a las opciones disponibles se ha desarrollado un sistema que cumpla con los objetivos propuestos, objetivos que tras realizar un análisis de funcionamiento del prototipo implementado se pueden considerar alcanzados. Aunque aún hay cierta incertidumbre respecto a la calidad de las mediciones del sistema, podemos concluir que podría ser viable para una puesta en funcionamiento en una línea de producción, y un futuro desarrollo está justificado.

Por último, se destaca la investigación y el estudio necesario de distintas metodologías por parte del alumno para resolver un problema específico, metodologías de las que no se tienen conocimientos previos. Esto incluye, además de la búsqueda, análisis y puesta a prueba de distintos métodos y algoritmos empleados para la detección de objetos 3D y la estimación de su pose en el espacio, la investigación de varias tecnologías para realizar una implementación del sistema finalmente propuesto. Ha sido necesario estudiar y emplear herramientas de visualización 3D, frameworks para la creación de interfaces gráficas, sistemas de comunicación entre procesos, distintos lenguajes de programación… Todo ello ha supuesto para el alumno la necesidad de adaptarse rápidamente a nuevos entornos de desarrollo de forma esencialmente autodidacta.

8 Referencias

[1] «Home | CTIC». [En línea]. Disponible en: https://www.fundacionctic.org/es/home. [Accedido: 03-may-2019].

[2] «Departamento de Ingeniería Eléctrica, Electrónica, de Computadores y Sistemas - Planta Piloto Industria 4.0 - Noticias». [En línea]. Disponible en: https://dieecs.uniovi.es/noticias/-/asset_publisher/0001/content/planta-piloto-industria-4-0?redirect=%2F. [Accedido: 26-may-2019].

[3] A. Zeng *et al.*, «Multi-view Self-supervised Deep Learning for 6D Pose Estimation in the Amazon Picking Challenge», *ArXiv160909475 Cs*, sep. 2016.

[4] M. A. Fischler y R. C. Bolles, «Random Sample Consensus: A Paradigm for Model Fitting with Applications To Image Analysis and Automated Cartography», *Commun. ACM*, vol. 24, pp. 381-395, 1981.

[5] «PCL - Point Cloud Library (PCL)». [En línea]. Disponible en: http://pointclouds.org/. [Accedido: 29-may-2019].

[6] B. Drost, M. Ulrich, N. Navab, y S. Ilic, «Model globally, match locally: Efficient and robust 3D object recognition», en *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 998-1005.

[7] T. Birdal y S. Ilic, «Point Pair Features Based Object Detection and Pose Estimation Revisited», en *2015 International Conference on 3D Vision*, Lyon, France, 2015, pp. 527-535.

[8] «surface_matching. Surface Matching — OpenCV 3.0.0-dev documentation». [En línea]. Disponible en: https://docs.opencv.org/3.0-beta/modules/surface_matching/doc/surface_matching.html#drost2010. [Accedido: 09-ene-2019].

[9] S. Hinterstoisser *et al.*, «Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes», en *2011 International Conference on Computer Vision*, Barcelona, Spain, 2011, pp. 858-865.

[10] «Stefan HINTERSTOISSER». [En línea]. Disponible en: http://www.stefan-hinterstoisser.com/. [Accedido: 23-may-2019].

[11] S. Hinterstoier, «Real-Time Detection and Pose Estimation of Low-Textured and Texture-Less Objects», p. 130.

[12] P. J. Besl y N. D. McKay, «A method for registration of 3-D shapes», *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, n.º 2, pp. 239-256, feb. 1992.

[13] R. Q. Charles, H. Su, M. Kaichun, y L. J. Guibas, «PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation», en *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 77-85.

[14] Y. Xiang, T. Schmidt, V. Narayanan, y D. Fox, «PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes», *ArXiv171100199 Cs*, nov. 2017.

[15] S. Hinterstoisser, V. Lepetit, P. Wohlhart, y K. Konolige, «On Pre-Trained Image Features and Synthetic Images for Deep Learning», *ArXiv171010710 Cs*, oct. 2017.

[16] W. Kehl, F. Milletari, F. Tombari, S. Ilic, y N. Navab, «Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation», en *Computer Vision – ECCV 2016*, vol. 9907, B. Leibe, J. Matas, N. Sebe, y M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 205-220.

[17] W. Abbeloos y T. Goedemé, «Point Pair Feature based Object Detection for Random Bin Picking», *ArXiv161201288 Cs*, dic. 2016.

[18] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, y S. Birchfield, «Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects», *ArXiv180910790 Cs*, sep. 2018.

[19] A. Berner, J. Li, D. Holz, J. Stuckler, S. Behnke, y R. Klein, «Combining contour and shape primitives for object detection and pose estimation of prefabricated parts», en *2013 IEEE International Conference on Image Processing*, Melbourne, Australia, 2013, pp. 3326-3330.

[20] A. Zeng *et al.*, «Robotic Pick-and-Place of Novel Objects in Clutter with Multi-Affordance Grasping and Cross-Domain Image Matching», *ArXiv171001330 Cs*, oct. 2017.
[21] «3D Camera Survey», *ROS-Industrial*. [En línea]. Disponible en: https://rosindustrial.org/3d-camera-survey. [Accedido: 30-may-2019].
[22] «3D time of flight cameras», *STEMMER IMAGING*. [En línea]. Disponible en: https://www.stemmer-imaging.com/fr-ch/donnees/cameras-3d-time-of-flight-cameras/. [Accedido: 31-may-2019].
[23] R. plc, «Renishaw: Application note: Optical encoders and LiDAR scanning», *Renishaw*. [En línea]. Disponible en: http://www.renishaw.com/en/optical-encoders-and-lidar-scanning--39244. [Accedido: 31-may-2019].
[24] «Fig. 1 Binocular stereo vision.», *ResearchGate*. [En línea]. Disponible en: https://www.researchgate.net/figure/Binocular-stereo-vision_fig1_220239577. [Accedido: 01-jun-2019].
[25] «Cámaras 3D Ensenso N35 - IDS Imaging Development Systems GmbH». [En línea]. Disponible en: https://es.ids-imaging.com/ensenso-n35.html. [Accedido: 01-jun-2019].
[26] «Learn OpenGL, extensive tutorial resource for learning Modern OpenGL». [En línea]. Disponible en: https://learnopengl.com/. [Accedido: 30-may-2019].
[27] «OGRE - Open Source 3D Graphics Engine | Home of a marvelous rendering engine». [En línea]. Disponible en: https://www.ogre3d.org/. [Accedido: 16-jun-2019].
[28] «VTK - The Visualization Toolkit». [En línea]. Disponible en: https://vtk.org/. [Accedido: 16-jun-2019].
[29] «Bullet Real-Time Physics Simulation | Home of Bullet and PyBullet: physics simulation for games, visual effects, robotics and reinforcement learning.» [En línea]. Disponible en: https://pybullet.org/wordpress/. [Accedido: 16-jun-2019].
[30] «Unity». [En línea]. Disponible en: https://unity.com/es. [Accedido: 30-may-2019].
[31] «What is Unreal Engine 4». [En línea]. Disponible en: https://www.unrealengine.com/en-US/. [Accedido: 16-jun-2019].
[32] «CRYENGINE | The complete solution for next generation game development by Crytek». [En línea]. Disponible en: https://www.cryengine.com/. [Accedido: 16-jun-2019].
[33] B. Foundation, «blender.org - Home of the Blender project - Free and Open 3D Creation Software», *blender.org*. .
[34] S. J. Pan y Q. Yang, «A Survey on Transfer Learning», *IEEE Trans. Knowl. Data Eng.*, vol. 22, n.º 10, pp. 1345-1359, oct. 2010.
[35] «Getting Started with Deep Learning», *Silicon Valley Data Science*, 15-feb-2017. [En línea]. Disponible en: https://www.svds.com/getting-started-deep-learning/. [Accedido: 01-jun-2019].
[36] «TensorFlow», *TensorFlow*. [En línea]. Disponible en: https://www.tensorflow.org/. [Accedido: 01-jun-2019].
[37] *Models and examples built with TensorFlow. Contribute to tensorflow/models development by creating an account on GitHub*. tensorflow, 2019.
[38] «models/detection_model_zoo.md at master · tensorflow/models · GitHub». [En línea]. Disponible en: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. [Accedido: 16-jun-2019].
[39] D. H. Ballard, «Generalizing the Hough transform to detect arbitrary shapes», *Pattern Recognit.*, vol. 13, n.º 2, pp. 111-122, ene. 1981.
[40] N. Gelfand, L. Ikemoto, S. Rusinkiewicz, y M. Levoy, «Geometrically stable sampling for the ICP algorithm», en *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, 2003, pp. 260-267.
[41] T. Zinsser, J. Schmidt, y H. Niemann, «A refined ICP algorithm for robust 3-D correspondence estimation», en *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*, 2003, vol. 2, pp. II-695.
[42] «(PDF) Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration». [En línea]. Disponible en: https://www.researchgate.net/publication/228571031_Linear_Least-

Squares_Optimization_for_Point-to-Plane_ICP_Surface_Registration. [Accedido: 14-ene-2019].
[43] «ROS.org | Powering the world's robots». [En línea]. Disponible en: https://www.ros.org/. [Accedido: 29-may-2019].
[44] «MQTT». [En línea]. Disponible en: http://mqtt.org/. [Accedido: 29-may-2019].
[45] «Redis». [En línea]. Disponible en: https://redis.io/. [Accedido: 29-may-2019].
[46] «SolidWorks». [En línea]. Disponible en: https://www.solidworks.com/es/home-page. [Accedido: 12-jun-2019].
[47] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, y G. Ranzuglia, «MeshLab: an Open-Source Mesh Processing Tool», *Eurographics Ital. Chapter Conf.*, p. 8 pages, 2008.
[48] W. E. Lorensen y H. E. Cline, «Marching Cubes: A High Resolution 3D Surface Construction Algorithm», en *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 1987, pp. 163–169.
[49] «Marching Cubes». [En línea]. Disponible en: http://www.cs.carleton.edu/cs_comps/0405/shape/marching_cubes.html#1. [Accedido: 12-jun-2019].
[50] «GIMP - GNU Image Manipulation Program». [En línea]. Disponible en: https://www.gimp.org/. [Accedido: 24-jun-2019].
[51] «CUDA Toolkit», *NVIDIA Developer*, 02-jul-2013. [En línea]. Disponible en: https://developer.nvidia.com/cuda-toolkit. [Accedido: 26-jun-2019].
[52] Broxxar, *A Depth-based Image Effect recreating the Topographic Scanner of No Man's Sky.: Broxxar/NoMansScanner*. 2019.
[53] «Distributed Messaging - zeromq». [En línea]. Disponible en: http://zeromq.org/. [Accedido: 27-jun-2019].
[54] C.-H. Hsieh y T.-L. Chia, «Analysis of Evaluation Metrics for Image Segmentation», p. 18.

ANEXO A: **PROGRAMACIÓN TEMPORAL**

		A1																							
		M1				M2				M3				M4				M5				M6			
FASE 1: DISEÑO																									
Hito 1	**Análisis del problema y definición de requisitos**	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
T1.1	Análisis de los procesos actuales y estado de la técnica																								
T1.2	Diseño conceptual del sistema																								
FASE 2: DESAROLLOS TECNOLÓGICOS																									
Hito 2	**Dispositivo de captura de datos**	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
T2.1	Selección de componentes hardware																								
T2.2	Diseño y construcción de la arquitectura hardware del sistema																								
T2.3	Implementación del sistema de control del dispositivo de captura																								
Hito 3	**Simulador de imágenes sintéticas etiquetadas**	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
T3.1	Diseño del componente																								
T3.2	Implementación del sistema de simulación de datos																								
Hito 4	**Algoritmos de visión artificial**	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
T4.1	Análisis y desarrollo de algoritmos para la detección de piezas																								
T4.2	Análisis y desarrollo de algoritmos para la selección de piezas																								
T4.3	Análisis y desarrollo de algoritmos para la estimación de pose																								
Hito 5	**Integración de componentes**	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
T5.1	Integración de módulos																								
T5.2	Pruebas de integración en entorno de laboratorio																								
FASE 3: VALIDACIÓN																									
Hito 6	**Plan de pruebas y validación**	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
T6.1	Diseño del plan de pruebas																								
T6.2	Ejecución del plan de pruebas y validación																								
FASE 4: DOCUMENTACIÓN																									
Hito 7	**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
T7.1	Redacción de la memoria																								
T7.2	Elaboración de anexos																								

ANEXO B: **PRESUPUESTO**

B.1. Costes de ejecución material

El coste de ejecución material incluye tres categorías: el coste de equipos, coste de software y coste de mano de obra por el tiempo empleado en el proyecto. Los costes serán calculados a partir de los tiempos estimados de la programación temporal (ANEXO A).

B.1.1 Costes de equipos

Coste asociado al uso de los equipos con los que se ha realizado el desarrollo del proyecto y que se calcula en concepto de amortización. Se considera un período de amortización de 3 años y un empleo de los equipos estimado de 2000 horas al año (40 horas semanales durante 50 semanas del año).

Concepto	Precio total
Lenovo Thinkpad W550s	849,99 €
Ordenador construcción propia	1759,72 €
Cámara Ensenso N35	9000,00 €

Tabla 7. Costes de compra de los equipos.

Concepto	Unidad	Precio unitario	Cantidad	Subtotal
Lenovo Thinkpad W550s	hora	0,14 €	960	136,00 €
Ordenador construcción propia	hora	0,29 €	480	140,78 €
Cámara Ensenso N35	hora	1,50 €	480	720,00 €
		Subtotal		996,78 €

Tabla 8. Coste de equipos.

B.1.2 Costes de software

Concepto	Unidad	Precio unitario	Cantidad	Subtotal
Unity 3D	mes	110,80 €	1	110,80 €
Librerías software	-	0,00 €	-	0,00 €
		Subtotal		110,80 €

Tabla 9. Costes de software.

B.1.3 Costes de mano de obra

El coste unitario de la mano de obra será de 30.00 €/h. La dedicación será de 40 horas semanales.

Concepto	Unidad	Precio unitario	Cantidad	Subtotal
Análisis del problema y definición de requisitos	hora	30,00 €	200	6.000,00 €
Dispositivo de captura de datos	hora	30,00 €	80	2.400,00 €
Simulador de imágenes sintéticas etiquetadas	hora	30,00 €	80	2.400,00 €
Algoritmos de visión artificial	hora	30,00 €	300	9.000,00 €
Integración de componentes	hora	30,00 €	100	3.000,00 €
Plan de pruebas y validación	hora	30,00 €	40	1.200,00 €
Realización de la documentación	hora	30,00 €	160	4.800,00 €
		Subtotal		28.800,00 €

Tabla 10. Costes de mano de obra

B.1.4 Coste total del presupuesto de ejecución material

Concepto	Subtotal
Costes de equipos	996,78 €
Costes de software	110,80 €
Costes de mano de obra	28.800,00 €
Subtotal	29.907,58 €

Tabla 11. Coste total del presupuesto de ejecución material.

B.2. Importe total

Los gastos generales y beneficio industrial son los gastos obligados que se derivan de la utilización de las instalaciones de trabajo más el beneficio industrial. Se estima un porcentaje del 10 % para los gastos generales y un 6% de beneficio.

Presupuesto de ejecución material			**29.907,58 €**
	10,00%	Gastos generales	2.990,76 €
	6,00%	Beneficio industrial	1.794,45 €
Total			**34.692,79 €**
	21,00%	IVA	7.285,49 €
Total, IVA incluido			**41.978,27 €**

El Importe Total del proyecto suma la cantidad de:

Cuarenta y Un Mil Novecientos Setenta y Ocho con Veintisiete Céntimos

ANEXO C: **MATERIAL ADJUNTO**

En este anexo se detalla el contenido de la carpeta con el código fuente desarrollado durante este proyecto. En la siguiente tabla (Tabla 12) se hace una descripción básica del contenido de cada una.

Carpeta		Contenido	Tecnologías
ensenso_camera_controller		Control de la cámara estéreo Ensenso N35.	C# EnsensoSDK REDIS Visual Studio Windows Forms
	nxSimple	Ejemplo básico de adquisición de nubes de puntos.	
	ensensoGUI	Interfaz gráfica para la captura de imágenes y nubes de puntos, guardando en archivos y/o publicando a través de REDIS.	
object_detection		Sistema de detección y estimación de pose	Python REDIS Tensor Flow OpenCV PyCharm
	train_models	Generación de modelos a partir de las imágenes generadas con el simulador.	
	detection_process	Módulos de detección 2D (detección de spinners en imágenes) y 3D (estimación de posición y orientación).	
	io_utils	Funciones para parsear mensajes y traducir tipos de datos.	
	file_examples	Ejecución de los módulos desde archivos guardados en disco.	
	mqtt_examples	Ejecución de los módulos usando MQTT como mecanismo de comunicación entre ellos.	
	redis_examples	Ejecución de los módulos usando REDIS como mecanismo de comunicación entre ellos.	
simulator		Proyecto de Unity (versión 2019.1.3f1) con el simulador de datos (imágenes sintéticas etiquetadas).	Unity C#
example_data		Datos de ejemplo para poder realizar la ejecución de los scripts.	
	camera_config	Configuración de la cámara para la toma de imágenes y para la adquisición de nubes de puntos.	
	models	Modelo PLY de spinner para la estimación de pose.	
	scene_example	Un ejemplo de mensajes generados con la cámara.	

Tabla 12. Contenido de las carpetas con el código fuente.

Cabe destacar que el código que se ha generado para este proyecto se ha realizado a modo de prototipo para poder realizar las experimentaciones y no se encuentra en su versión definitiva.