LOCALIZACIÓN DE UN ROBOT AUTÓNOMO PARA LA INSPECCIÓN Y SUBSANACIÓN DE IMPERFECCIONES EN CHAPA GRUESA

Memoria del Trabajo Fin de Máster realizado por

SARA ROOS HOEFGEEST TORIBIO

para la obtención del título de

MÁSTER EN INGENIERÍA DE AUTOMATIZACIÓN E INFORMÁTICA INDUSTRIAL

Tutor: Rafael Corsino González de los Reyes

Universidad de Oviedo

FEBRERO DE 2019

ÍNDICE

MEMORIA

ÍNDICE DE LA MEMORIA

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1 Objetivo

Este trabajo se enmarca en el proyecto de *Investigación y desarrollo de robots de inspección y subsanación de imperfecciones en chapa gruesa*.

En la actualidad, la inspección y saneo de chapas de acero es realizada por operarios humanos. En primer lugar, se inspecciona la chapa en busca de imperfecciones y, posteriormente, se realiza la subsanación de los defectos encontrados. Las tareas de subsanación, habitualmente, se traducen en largas jornadas en posiciones incómodas que pueden derivar en problemas físicos para el trabajador. Por ello, este proyecto busca liberar a los operarios, que se encargan del saneo de las chapas de acero, de grandes cargas de trabajo.

El objetivo general de dicho proyecto es el estudio de la viabilidad del desarrollo e implementación de un robot capaz de inspeccionar la totalidad de una chapa de acero en busca de defectos y, posteriormente, ser capaz de subsanarlos. Además, el vehículo deberá ser capaz de elaborar un mapa de los defectos encontrados en las chapas.

El robot albergará diferentes herramientas que le permitan llevar a cabo la navegación, inspección y subsanación de la chapa. Dispondrá de sensores inerciales, cámaras y láseres para permitir la localización del robot y mapeo del entorno. También contará con un sistema de inspección que permita encontrar los defectos en la chapa y un sistema con una herramienta de reparación para la subsanación de los mismos.

Uno de los principales problemas a los que hay que hacer frente es conocer la localización del robot sobre la chapa. El objetivo particular de este trabajo será, precisamente, el estudio de diferentes alternativas existentes para conocer la localización del robot en todo momento para conseguir escoger el método más adecuado. Además, se llevarán a cabo distintas simulaciones con el fin de evaluar su funcionamiento.

La estrategia escogida se desarrollará en forma de paquetes de ROS capaces de proporcionar el estado del robot a otros algoritmos encargados de tareas como la navegación y el mapeo del entorno. La navegación autónoma del robot corresponde a otro módulo del proyecto debido a su complejidad.

1.2 Estado del arte

La robótica es la rama de la ingeniería que se ocupa del diseño, construcción, operación y aplicación de los robots. En múltiples aplicaciones se encuentra ligada a otra rama de conocimiento, la visión artificial. La visión por computador o visión artificial tiene como finalidad la extracción de información del mundo físico a partir de imágenes por medio de un computador.

Estas dos especialidades, desde su creación, han sido áreas de investigación en auge que no han frenado su evolución, permitiendo el desarrollo de aplicaciones muy diversas en ámbitos dispares como la industria, la medicina o sofisticadas exploraciones espaciales.

En el ámbito industrial, el desarrollo de la robótica ha sido de suma importancia, permitiendo automatizar gran parte de los procesos, lo que se traduce en una mejora en la eficiencia de las tareas y en la seguridad de los operarios humanos, permitiendo desempeñar diferentes trabajos en entornos agresivos o que entrañan riesgo para las personas.

Cada vez resulta más habitual encontrar robots móviles en las industrias. Son dispositivos de transporte automático, es decir, una plataforma dotada de cierta autonomía para su desplazamiento en un determinado ambiente de trabajo. La localización en cada instante de tiempo es vital para estos robots. Este problema se ha tratado desde múltiples perspectivas, siendo común el uso de diferentes sensores que proporcionan información directa sobre la localización del robot en cada instante, o bien, sobre los cambios que se han producido en su entorno.

Se puede realizar una primera clasificación de estas técnicas en dos grupos según se utilicen medidas relativas, llamado odometría, o medidas absolutas, conocido simplemente como localización global.

Si la posición inicial del robot es conocida, los métodos odométricos estiman la posición y orientación del vehículo mediante información obtenida de diferentes sensores incluidos en el robot, como encoders, giroscopios o acelerómetros, que permiten la estimación incremental de la pose actual a partir de la anterior. Al tratarse de un método incremental se van acumulando errores tras cada iteración, por tanto es necesario aplicar continuos ajustes durante la navegación.

Los métodos globales obtienen la pose absoluta a partir de balizas y marcadores o señales satélite, como los sistemas GPS. Sin embargo, los sistemas basados en señales satélite no resultan adecuados en espacios confinados ya que, los GPS no reciben señal en recintos cerrados.

Además, es preciso contemplar otros métodos como el *Simultaneous Localization and Mapping* o SLAM, técnica que permite la construcción de un mapa del entorno, así como la localización del vehículo de manera simultánea mientras se desplaza en dicho entorno.

También es frecuente en robótica la fusión de sensores, generalmente mediante filtros de Kalman [1] o filtros de partículas [2], que permite combinar las estimaciones procedentes de distintas fuentes para lograr una pose más robusta. Por ejemplo, en [3], se fusiona la información procedente de sensores inerciales con un método de odometría visual.

A continuación, se hace un repaso de los diferentes estudios e implementaciones existentes en la literatura científica de los principales métodos comentados.

1.2.1 Odometría

La odometría es el estudio de la estimación de la posición de vehículos con ruedas durante la navegación respecto a su localización inicial. Para realizar esta estimación es común el uso de información sobre la rotación de las ruedas para estimar cambios en la posición a lo largo del tiempo. Sin embargo, este método puede no funcionar en terrenos desiguales o resbaladizos que provocan que las ruedas patinen y, por consiguiente, los sensores asociados a ellas no sean capaces de medir la rotación con exactitud.

Con el fin de solucionar esta problemática, es posible recurrir al uso de una o varias cámaras en pos de estimar la pose del robot, proceso conocido como odometría visual (OV).

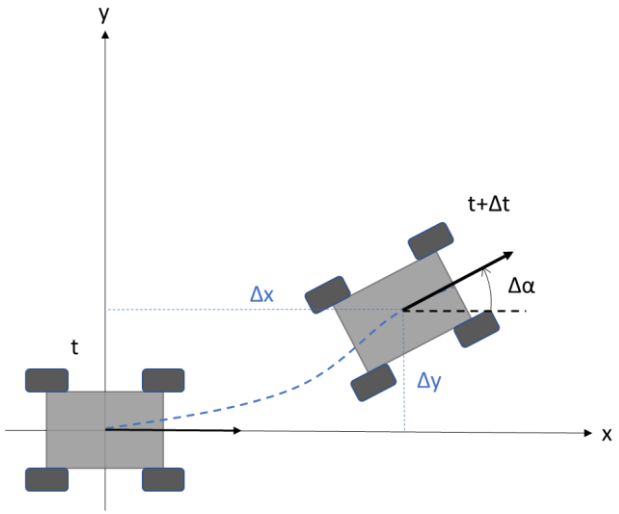

Figura 1. Odometría. Estimación de la pose de un vehículo de manera incremental, recuperando el movimiento realizado en instantes consecutivos.

La odometría visual opera con el fin de la estimación incremental de la pose del vehículo, a través del estudio de los cambios que su movimiento induce en las imágenes capturadas por cámaras dispuestas sobre el vehículo. Como en toda aplicación de visión artificial, el entorno es un factor clave, siendo necesaria una correcta iluminación y una escena estática con suficiente textura para permitir que se extraiga el movimiento aparente. Cabe la posibilidad de optimizar las medidas tras *n* estimaciones de la trayectoria realizando algún tipo de ajuste, por ejemplo, un ajuste *Bundle* [4].

Se puede realizar una primera clasificación de los sistemas de odometría visual según el tipo de cámaras utilizadas:

- **Sistemas monoculares**: Se trata de sistemas que utilizan una única cámara. En este tipo de sistemas es necesario implementar algún método para permitir realizar mediciones, pues solo permiten la reconstrucción hasta un cierto factor de escala, como puede ser la introducción de algún elemento en la imagen de tamaño conocido o restricciones de movimiento.

- **Sistemas estéreo**: Estos sistemas emplean más de una cámara. Permiten una fácil obtención de la estructura 3D del entorno y, por ello, la toma de medidas. El

problema de estos sistemas es que pueden degenerar en el caso monocular cuando la distancia a la escena es mucho mayor que la línea base del par estéreo.

- **Sistemas con cámaras RGB-D**: Las cámaras RGBD son un dispositivo de detección de profundidad que trabajan en asociación con una cámara RGB. Proveen imágenes a color y estimaciones de profundidad por píxel, es decir, son capaces de medir con un cierto error a qué distancia se encuentran los objetos capturados en los fotogramas.

Una segunda clasificación se puede realizar en cuanto al tratamiento de las imágenes con el fin de estimar la localización del robot:

- **Sistemas basados en puntos característicos:** Se basan en el rastreo *frame* a *frame* de unos puntos que son fácilmente distinguibles y repetibles en las imágenes

- **Sistemas globales:** Los sistemas globales utilizan la información de intensidad de todos los píxeles de la imagen o subregiones de la misma.

- **Sistemas híbridos:** Estos sistemas emplean una mezcla de los dos métodos anteriores.

Existen diferentes implementaciones de estos métodos en la literatura [5]. El problema de estimar la localización de un vehículo a partir de entradas visuales comenzó a principios de la década de 1980 y fue descrito por Moravec [6]. Desarrolló un sistema estéreo basado en puntos característicos, con una sola cámara deslizante sobre un raíl, en el que el vehículo se detenía para tomar imágenes con la cámara en nueve posiciones diferentes. Posteriormente, se buscaban correspondencias entre esas imágenes y se realizaba una reconstrucción 3D. Después, el movimiento de la cámara era obtenido alineando los puntos 3D reconstruidos observados desde diferentes localizaciones.

Sin embargo, no sería hasta el año 2004 cuando Níster utilizaría por primera vez el término de odometría visual [7]. Una de las aplicaciones más famosas fue en la misión espacial en Marte, comenzada en 2003, que incluía dos *rovers* para explorar la superficie del planeta [8]. Además, Níster propuso métodos pioneros para sistemas monoculares y estéreo,

centrándose en el problema de las falsas correspondencias entre puntos característicos o *outliers* y proponiendo un esquema de eliminación de los mismos mediante RANSAC [9].

Otras investigaciones son las realizadas por Scaramuzza y Siegwart enfocadas a vehículos de tierra en un ambiente al aire libre mediante una cámara omnidireccional monocular fusionando dos estrategias diferentes [10]. El primer enfoque, basado en puntos característicos, fue desarrollado utilizando SIFT para la extracción de estos y RANSAC para la eliminación de valores atípicos. El segundo enfoque utilizó un método global, originalmente propuesto por Comport, Malis y Rives en [11].

Además, en la última década, se han desarrollado algoritmos de odometría visual empleando cámaras RGB-D, como la Kinect de Microsoft [12]. Este sistema ha sido utilizado en diferentes aplicaciones, como en vehículos submarinos autónomos [13] o un micro vehículo aéreo autónomo (MAV) [14].

1.2.1.1 Implementaciones existentes en ROS

Ya que el objetivo de este trabajo es la localización de un robot móvil mediante ROS, resulta oportuno hacer un breve repaso de paquetes ya desarrollados. Entre ellos destacan los siguientes:

- **DVO (*Dense Visual Odometry*)** [15]**:** Este paquete estima la pose de una cámara RGB-D. La aproximación implementada en este método se describe en [16].

- **viso2_ros** [17]**:** Este paquete implementa una odometría visual utilizando la librería libviso2. Contiene dos nodos, uno que sigue un método monocular y otro con un par estéreo. Mantenido por el grupo de Sistemas, Robótica y Visión de la Universidad de las Islas Baleares [18].

- **fovis_ros** [19]: Proporciona una interfaz para *fovis*, una librería de odometría visual. Este paquete implementa dos métodos según el tipo de sensor, uno para una cámara RGB-D y otro para un par estéreo. Mantenido por el grupo de Sistemas, Robótica y Visión de la Universidad de las Islas Baleares [18].

- **Módulo OpenCV RGB-Depth processing** [20]**:** Módulo incluido en librería OpenCV [21]. Este método está inspirado en el trabajo de Steinbrucker et al [22],

en el que presentó un método global a partir de las imágenes RGB-D proporcionadas por una Kinect de Microsoft.

1.2.2 SLAM (*Simultaneous Localization and Mapping*)

El mapeo y localización simultáneas o SLAM, es una técnica utilizada en navegación autónoma que permite la construcción de un mapa del entorno, así como la localización del vehículo de manera simultánea mientras se desplaza por él.

La construcción del mapa del entorno es necesaria con el objetivo de descubrir si se ha vuelto a un área previamente visitada. A este procedimiento se le conoce bajo el nombre de *loop closure* y es utilizado para corregir el error en las estimaciones.

El mapeo del entorno se realiza de manera incremental con la ayuda de un conjunto de sensores. En las últimas décadas, se han realizado múltiples soluciones utilizando diferentes sensores. Por ejemplo, Kleeman desarrolló un sistema incluyendo sónares [23], científicos de la Politécnica de Torino implementaron un sistema basado en sensores infrarrojos [24] o el sistema utilizando láseres realizado por investigadores del departamento de robótica de la Universidad de Oxford [25].

Sin embargo, un aspecto muy importante dentro de la investigación de las técnicas de SLAM ha sido la introducción del uso de cámaras, debido a la gran información que provee del entorno. A esta estrategia se la conoce como *Visual* SLAM o vSLAM.

El método vSLAM consiste en el desarrollar el mapeo y localización simultáneas que caracteriza al SLAM por medio de cámaras.

El primer esquema monocular se presentó en 2003 en el trabajo de Andrew J. Davinson bajo el nombre de MonoSLAM [26]. En este método, el movimiento de la cámara y la estructura 3D del entorno son estimadas simultáneamente utilizando un filtro de Kalman extendido (EKF) [27]. Sin embargo, esta técnica resultó muy costosa desde el punto de vista computacional.

Posteriormente, MonoSLAM evolucionó en otros métodos, como PTAM (*Parallel Tracking and Mapping*) [28], que redujo el coste computacional de su antecesor mediante la separación del tracking y el mapeo en dos hilos diferentes en la CPU ejecutados en paralelo.

Otros algoritmos desarrollados son SVO (*Semi-direct Monocular Visual Odometry*) [29], que sigue una estrategia monocular basada en puntos característicos, según se describe en el artículo de los autores [30], u ORB-SLAM, librería para cámaras monocular, estéreo y RGB-D [31]. Todos ellos cuentan con paquetes específicos desarrollados para ROS.

Además, también se implementaron otras estrategias de vSLAM para cámaras RGBD, como la propuesta de Salas-Moreno, SLAM++ [32] o la de Tateno et al. [33].

1.2.3 Métodos de localización global

Los métodos globales, como ya se ha explicado con anterioridad, obtienen la pose absoluta a partir de balizas y marcadores o señales satélite, como los sistemas GPS.

Una estrategia para conocer la pose de un vehículo utilizando medidas absolutas es mediante la colocación de balizas en posiciones conocidas del entorno con el fin de facilitar la localización del robot.

Esta técnica es una de las más precisas en este ámbito. Si bien, su precisión está estrechamente relacionada con el tipo de señal utilizada, por ejemplo, radio, láser, infrarrojos o ultrasonidos, con las características del sensor y con el número de balizas empleadas. Además, se debe garantizar un entorno de trabajo propicio, permitiendo que las señales queden en todo momento libres de oclusiones. El ruido electromagnético también es un factor a tener en cuenta según el tipo de baliza utilizada. Por ello, esta técnica no se presenta adecuada para entornos muy dinámicos o no estructurados.

Existen diferentes implementaciones en la literatura científica. McGillem y Rappaport [34] desarrollaron un sistema de balizas infrarrojas con un dispositivo óptico giratorio para la navegación de vehículos autónomos. En esta propuesta, la posición del robot se estima mediante el cálculo de los ángulos entre balizas consecutivas a partir de la velocidad de giro del receptor.

Una estimación de la pose de un robot móvil mediante la fusión de información proporcionada por balizas de ultrasonidos y láseres es descrita en [35]. También se pueden encontrar ejemplos sobre el uso de señales WiFi [36].

También existen balizas visuales, como balizas luminosas de diferentes geometrías y colores, los marcadores empleados en los *crash test* [37] o marcadores fiduciales.

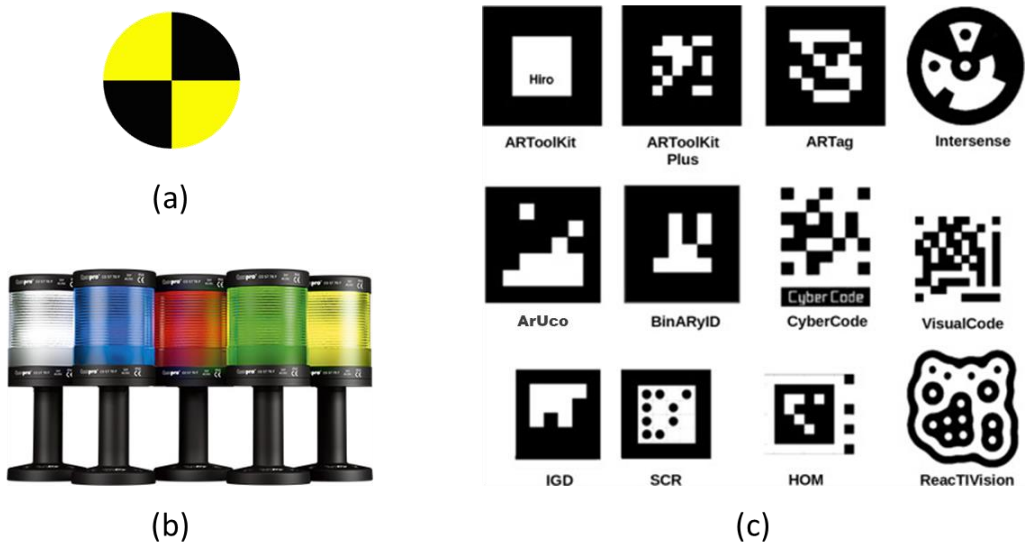

Figura 2. Modelos de balizas visuales. (a) Marcador utilizado en los crash test. (b) Balizas luminosas de colores. (c) Sistemas basado en marcadores fiduciales.

Los marcadores fiduciales son objetos artificiales especialmente diseñados para facilitar su detección, ampliamente utilizados en aplicaciones de realidad virtual, navegación de robots o de interacción de robots con personas.

Entre los muchos sistemas de marcadores fiduciales propuestos en la literatura, algunos ejemplos se pueden ver en la Figura 2, los basados en marcadores cuadrados son los que gozan de una mayor popularidad, especialmente en el área de la realidad aumentada [38], [39]. Principalmente, debido a que la estimación de la posición de una cámara calibrada es posible a partir de sus cuatro esquinas. Estos marcadores presentan un identificador único basado en un código binario, generalmente codificados dentro de un diccionario que almacena un conjunto de marcadores previamente desarrollados.

Los marcadores binarios han sido utilizados en múltiples aplicaciones de localización de vehículos. Por ejemplo, investigadores de la Universidad de Bratislava, desarrollaron un método para localizar robots móviles utilizando marcadores artificiales tipo ArUco

desplegados por el entorno y una sola cámara [40]. Otros sistemas de marcadores cuadrados destacados son ARTag [41], ARToolKit Plus [42] o BinARyID [43].

Entre los marcadores no cuadrados, los más empleados son los circulares Intersense [44] y ReacTIVision [45], con forma de ameba.

1.3 Resumen método implementado

Tras realizar un estudio de las diferentes alternativas existentes, se llevarán a cabo diferentes experimentos empleando distintas estrategias. Por un lado, se evaluarán dos librerías de odometría visual, *Fovis* [19] y *OpenCV RGB-Depth Processing* [20]. La primera librería se puede aplicar tanto en sistemas estéreo como en sistemas RGBD, mientras que la segunda, está desarrollada para trabajar únicamente con cámaras RGBD.

Por otro lado, se propondrá un enfoque de localización global que estima la pose del robot a partir de una serie de marcadores dispuestos en posiciones y orientaciones conocidas, guardando en un archivo sus localizaciones respecto a un sistema de referencia al que se llamará mapa de marcadores. Así, mediante un arco de 8 cámaras calibradas dispuestas en el robot, se estima la posición y orientación del vehículo a partir de los fiduciales encontrados en las cámaras.

Podemos dividir la estrategia de localización implementada en dos partes claramente diferenciadas. Primero, la detección de los marcadores y, después, el cálculo de la posición y orientación del robot a partir de las mismas.

La detección de los marcadores, en concreto se utilizarán marcadores ArUco, se llevará a cabo utilizando un paquete de ROS ya existente, *aruco_detect* [46], y se coordinará con el paquete implementado que estima la localización.

En cuanto a la estimación de la posición del vehículo, se desarrollará un paquete debido a que no se encontró ninguno ya implementado que se ajuste a los objetivos requeridos. Se seguirán dos estrategias diferentes. Por un lado, las 8 cámaras individualmente seguirán un enfoque monocular, calculando la localización según los marcadores encontrados y, por otro, cada par de cámaras adyacentes trabajarán según una configuración estéreo.

Al tratarse de un método global, el error no se acumulará tras cada iteración. Además, el vehículo tendrá que moverse por una chapa de acero situada en una nave industrial de gran tamaño, con una iluminación cuestionable y un entorno variable, por ello, resulta más conveniente colocar un número de marcadores en posiciones idóneas para facilitar la localización.

Además, se realizarán diferentes pruebas con el fin de encontrar el número y localización óptima de marcadores a disponer en el entorno.

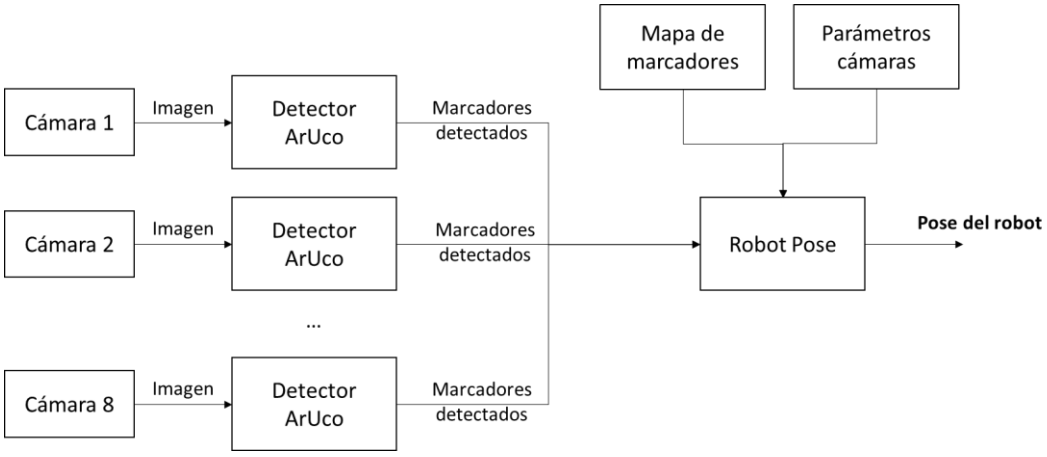

Figura 3. Esquema del método propuesto. *Robot Pose* hace referencia al paquete implementado.

En el capítulo siguiente, *Marco teórico,* se expondrán los fundamentos teóricos de los algoritmos utilizados. Posteriormente, en Implementación en ROS, se detallará la implementación del método propuesto en ROS. Después, un capítulo recogerá los diferentes experimentos llevados a cabo, así como los resultados obtenidos. Finalmente, se realizará un análisis de los resultados y posibles vías de mejora futuras en el capítulo *Discusión*.

2. MARCO TEÓRICO

2.1 Modelo de la cámara

Para modelar las cámaras se supondrá un modelo *pin-hole* [47, Sec. 11.3.1]. Este modelo describe la relación matemática existente entre las coordenadas de un punto tridimensional en el espacio y su correspondiente proyección en el plano imagen, siguiendo una línea recta que atraviesa el centro óptico, como se puede apreciar en la Figura 4. La proyección de todos los puntos del espacio forma el plano imagen, separado del centro óptico una distancia *f*, denominada distancia focal.

En el caso de una cámara, el sensor es discreto, por lo tanto, las proyecciones en el plano imagen están dadas en píxeles.

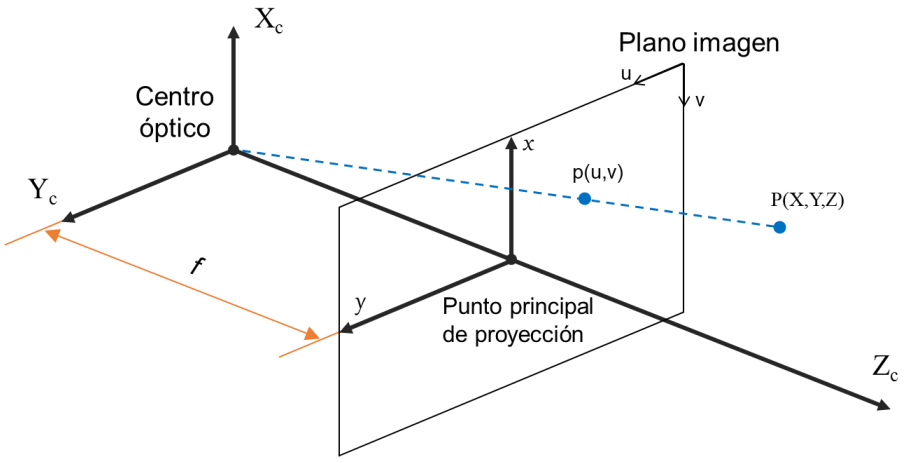

Figura 4. Geometría del modelo *Pin-hole*

El punto *p* en la imagen capturada por la cámara se trata de la proyección del punto P del entorno tridimensional, obtenido a partir de la intersección entre el plano imagen y la recta que une *P* con el centro óptico de la cámara. Siendo $P = (X, Y, Z)^T$ y $p = (u, v)^T$, ambos respecto a las coordenadas de la cámara, la relación entre ambos puntos se puede describir, mediante triángulos semejantes, como:

$$\begin{cases} \frac{u}{f} = \frac{X}{Z} \\ \frac{v}{f} = \frac{Y}{Z} \end{cases} \tag{1}$$

Representando los puntos en coordenadas homogéneas [48] y en términos de multiplicación de matrices se obtiene la ecuación (2).

$$\begin{bmatrix} nu \\ nv \\ n \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

El punto *P* se puede expresar en coordenadas del mundo, ${}^{W}P = (X_W, Y_W, Z_W)$, mediante la siguiente transformación.

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \tag{3}$$

Si se agrupan las ecuaciones (2) y (3) se puede expresar el modelo matemático que relaciona las coordenadas del punto tridimensional con el punto proyectado en la imagen.

En el modelo, las coordenadas en el plano imagen tienen como origen el punto principal de proyección, idealmente alineado con el centro óptico, mientras que las coordenadas, en píxeles, obtenidas de la cámara *p(u,v)*, presentan su origen en la esquina superior izquierda. Por ello, en la ecuación (2), hay que desplazar el centro hasta las coordenadas (u_0, v_0), que representan el punto principal de proyección (intersección entre el eje óptico y el plano imagen). Además, se tiene en cuenta la distorsión, con parámetros como el *skew (s)*, K_u y K_v.

$$\begin{bmatrix} nu \\ nv \\ n \end{bmatrix} = \begin{bmatrix} K_u f & s & u_0 & 0 \\ 0 & K_v f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \tag{4}$$

$$\tilde{p} = M \cdot \widetilde{{}^{W}P} \tag{5}$$

$$M = K[R|t] \tag{6}$$

Reescribiendo la ecuación (4) como (5), se ve que los puntos del mundo se relacionan con los de la imagen a través de la matriz M, de tamaño 3x4, conocida como la matriz de proyección que modela la cámara. Siendo la matriz K la matriz de parámetros intrínsecos de la cámara y [R|t] la matriz de parámetros extrínsecos que corresponde a la transformación euclídea del sistema de coordenadas del mundo al sistema de la cámara, donde *R* es la matriz de rotación y *t* el vector de traslación. También se puede referir a la matriz [R|t] como la pose de la cámara.

El término *s* de la matriz de parámetros intrínsecos es el parámetro de *skew*, que indica la falta de perpendicularidad entre los lados contiguos de cada pixel, habitualmente el valor es despreciable. K_u y K_v representan el total de píxeles por unidad en las direcciones *u* y *v* respectivamente. Los parámetros de esta matriz se pueden obtener sometiendo a la cámara a un proceso de calibración [49], [50].

2.2 Estimación de la pose de la cámara

En visión por computador, estimar la pose de la cámara a partir de *n* correspondencias de puntos 3D-2D es uno de los problemas fundamentales. La versión más general consiste en estimar los seis grados de libertad de la pose y los parámetros de calibración de la cámara. El problema se puede resolver con un mínimo de seis correspondencias, utilizando el método conocido como *Direct Linear Transform* (DLT) [51].

La simplificación más común de este problema consiste en trabajar con cámaras calibradas, conociendo, por ello, sus parámetros intrínsecos. Esta simplificación es conocida como *Perspective n Point problem (PnP).*

Se puede formular el problema *PnP* de manera sencilla. Se trata de la estimación de la pose de la cámara a partir de un conjunto de *n* correspondencias de puntos 3D del espacio y sus correspondientes proyecciones en el plano imagen y, los parámetros de calibración de esta.

Existen diferentes soluciones a este problema, por ejemplo, un método iterativo basado en la optimización de Levenberg-Marquart [52], *P3P*, un método basado en el artículo de Gao et al [53] y *Efficient PnP*, introducido por Lepetit et al [54].

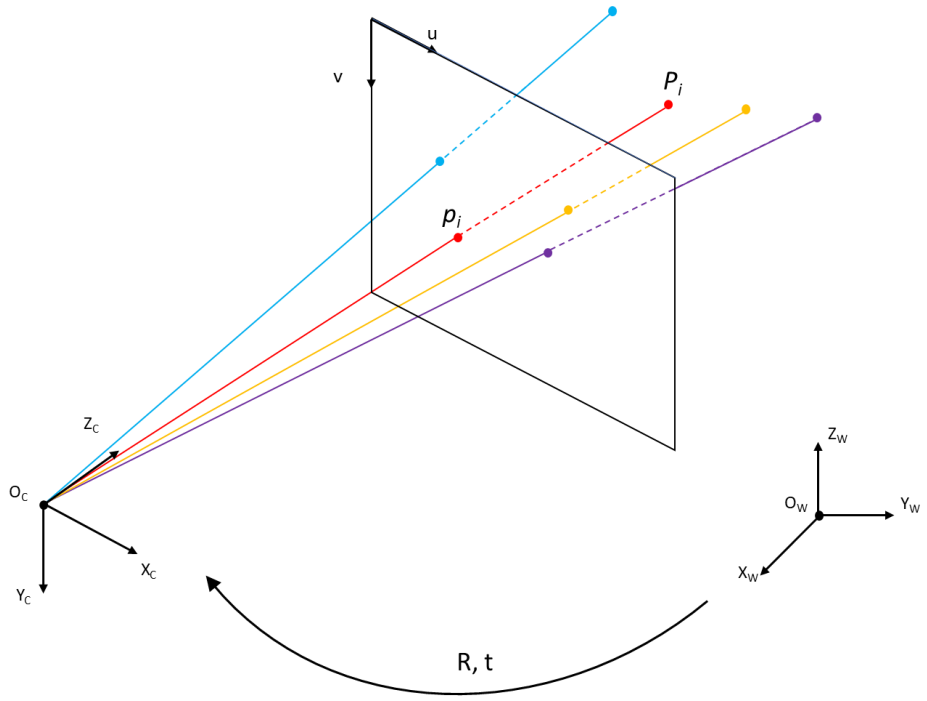

Figura 5. Esquema del problema PnP. Dado un conjunto de puntos del espacio 3D P_i (expresados en el sistema de referencia del mundo (Xw, Yw, Zw)) y sus correspondencias 2D p_i en la imagen, se trata de estimar la pose de la cámara (rotación y traslación) respecto del mundo.

Como ya se explicó en el apartado 2.1 un punto tridimensional del espacio se puede proyectar en un punto de dos dimensiones en la imagen de la cámara mediante su matriz de proyección M, compuesta por las matrices de parámetros intrínsecos y extrínsecos.

$$\tilde{p} = M \cdot \tilde{P}$$
$$\tilde{p} = K[R \mid t] \cdot \tilde{P} \tag{7}$$

El objetivo es obtener la pose de la cámara [R|t] a partir de un conjunto de correspondencias entre puntos 3D y sus proyecciones en la imagen. Además, la matriz de parámetros intrínsecos *K* es conocida.

$$K^{-1} \cdot \tilde{p} = [R|t] \cdot \tilde{P} \tag{8}$$

En este caso, se emplea el algoritmo iterativo basado en la optimización de Levenberg-Marquart. La función busca la pose que minimiza el error de reproyección, representado para

un punto en la ecuación (9). Para extenderlo a todos los puntos, la función de coste será la suma de todos los errores de reproyección al cuadrado.

$$K^{-1} \cdot \tilde{p} - [R|t] \cdot \tilde{P} = E_{rep} \tag{9}$$

2.3 Triangulación

En visión por computador, el término triangulación se refiere al proceso por el que se determina el punto 3D en el espacio dadas sus proyecciones en un mínimo de dos cámaras. Para resolver este problema es necesario conocer las matrices de proyección de las cámaras involucradas y que estén referidas a un sistema de referencia común.

Cada punto en la imagen se corresponde con una línea en el espacio tridimensional. En el caso ideal, si se conoce una correspondencia de puntos en dos imágenes, p_1 y p_2, se puede obtener el punto del espacio que los proyecta mediante el punto de corte de los rayos reproyectados. Sin embargo, debido al ruido y a otros posibles factores, ambos rayos se cruzarán. Ver Figura 6.

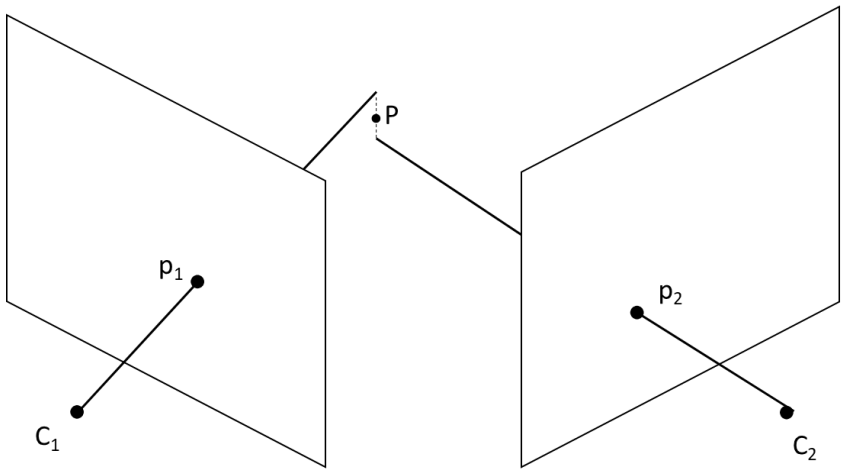

Figura 6. En la realidad, los rayos reproyectados de p_1 y p_2 no llegan a cortarse en el espacio.

Según el algoritmo de Hartley [55], se puede calcular el punto P más cercano a ambas líneas, siendo el punto medio de la recta de cruce. Sabiendo que la relación entre un punto 3D y su proyección en la imagen se relaciona mediante la matriz de proyección de la cámara *M*, se deduce el sistema de ecuaciones (10). Siendo $\widetilde{p_1}$ y $\widetilde{p_2}$ las proyecciones, en coordenadas

homogéneas, en la cámara 1 y en la cámara 2 respectivamente y $\tilde{P}$ el punto tridimensional, también en coordenadas homogéneas.

$$\left.\begin{matrix}\widetilde{p_1} = M_1 \cdot \tilde{P} \\ \widetilde{p_2} = M_2 \cdot \tilde{P}\end{matrix}\right\} \tag{10}$$

Una de las maneras más habituales de resolver el sistema es aplicando la descomposición SDV (*Singular Value Decomposition*). Para ello, se transforma el sistema de la ecuación (10) en un sistema de ecuaciones homogéneas de la forma Ax = 0 siguiendo los siguientes pasos.

Se parte del sistema de ecuaciones (10) y se generaliza para una cámara. Para plantear correctamente el nuevo sistema, es necesario deshacerse del factor de escala *n*.

$$\begin{bmatrix} nu \\ nv \\ n \end{bmatrix} = \begin{bmatrix} m_1^T \\ m_2^T \\ m_3^T \end{bmatrix} \cdot \tilde{P} \tag{11}$$

$$\left.\begin{matrix} nu = m_1^T \cdot \tilde{P} \\ nv = m_2^T \cdot \tilde{P} \\ n = m_3^T \cdot \tilde{P} \end{matrix}\right\} \tag{12}$$

Sustituyendo el factor de escala y expresando las ecuaciones de manera matricial, se transforma el sistema (12) en el (14), obteniendo las dos ecuaciones que aporta cada cámara.

$$\left.\begin{matrix} u\, m_3^T\, \tilde{P} - m_1^T\, \tilde{P} = 0 \\ v\, m_3^T\, \tilde{P} - m_2^T\, \tilde{P} = 0 \end{matrix}\right\} \tag{13}$$

$$\begin{bmatrix} u\, m_3^T - m_1^T \\ v\, m_3^T - m_2^T \end{bmatrix} \tilde{P} = 0 \tag{14}$$

Al ser $\tilde{P}$ un punto tridimensional, es necesario conocer las proyecciones de dicho punto en un mínimo de dos cámaras. Entonces, extendiendo la ecuación para las dos cámaras se obtiene el sistema de cuatro ecuaciones (15).

$$\begin{bmatrix} u_1\, {}^1m_3^T - {}^1m_1^T \\ v_1\, {}^1m_3^T - {}^1m_2^T \\ u_2\, {}^2m_3^T - {}^2m_1^T \\ v_2\, {}^2m_3^T - {}^2m_2^T \end{bmatrix} \tilde{P} = 0 \tag{15}$$

Se calcula $\tilde{P}$ mediante la aplicación de la descomposición SVD [56]. Cabe recordar que $\tilde{P}$ se encuentra en coordenadas homogéneas, por ello, para la obtención del punto P, es necesario realizar la conversión de coordenadas homogéneas a euclídeas.

2.4 Estrategias de localización del método propuesto

El método propuesto utiliza un esquema de localización basado en marcadores dispuestos en posiciones conocidas. Una vez el marcador se ha detectado en las imágenes proporcionadas por las cámaras se siguen dos estrategias diferentes. Por un lado, si se encuentra un mismo marcador en dos cámaras contiguas, se trabaja como si de un par estéreo se tratase. Por otro, se utiliza un enfoque monocular estimando la pose a partir de una sola cámara y los marcadores detectados en la misma. Se implementan estos dos métodos con el fin de abarcar la mayor cantidad de situaciones posibles y evaluar su funcionamiento.

En el caso estéreo, se estima la pose del robot en todos los pares de cámaras consecutivas que tengan, al menos, un marcador en común. De manera similar ocurre con el caso monocular, ya que se calcula la pose en todas las cámaras que detecten algún marcador. La estimación final se selecciona según el menor error de reproyección.

El error de reproyección es el error geométrico que corresponde a la distancia entre la proyección de un punto 3D y la reproyección del punto 3D estimado, en el plano imagen. Se utiliza para cuantificar la precisión con la que una estimación de un punto 3D P recrea la verdadera proyección de ese punto p. La distancia de reproyección d se calcula como la distancia euclídea entre p y p'.

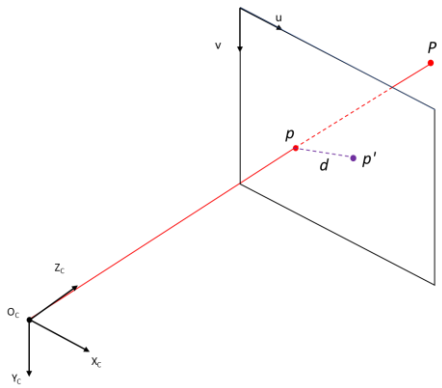

Figura 7. Error de reproyección.

2.4.1 Método monocular

Según esta estrategia, la pose del robot se estima a partir de uno o más marcadores detectados en una misma cámara calibrada.

Resolviendo el problema de *Perspective-n-Point* es posible estimar la pose de una cámara calibrada dados una serie de puntos 3D en el mundo y sus correspondientes proyecciones 2D en la imagen. Como ya se ha explicado, un solo marcador arroja las cuatro correspondencias necesarias, a partir de sus esquinas, para resolver el problema. La pose de la cámara consiste en 6 grados de libertad, orientación (roll, pitch, yaw) y traslación respecto del mundo (X, Y, Z).

Se resuelve el problema *PnP* con las proyecciones de las cuatro esquinas de los marcadores, proporcionadas por el paquete de detección de *ArUcos*, y las posiciones conocidas en el espacio de los marcadores correspondientes respecto del sistema del mundo, obtenidas del mapa de balizas definido previamente.

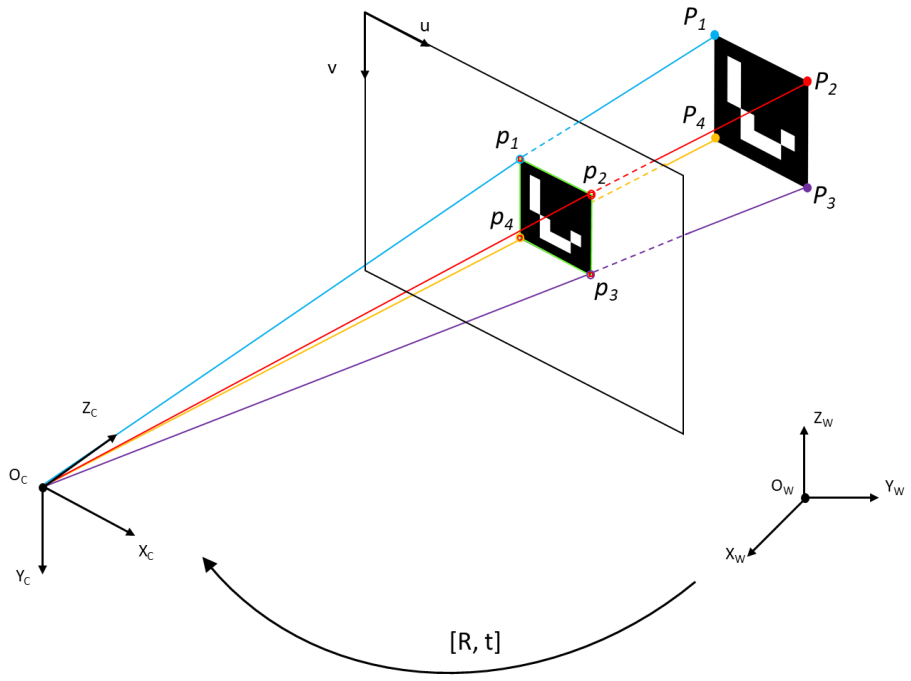

Figura 8. Estimación de la pose de la cámara [R|t] a partir de un marcador.

Tras resolver el problema, se obtienen los parámetros extrínsecos de la cámara, es decir, la rotación del sistema del mundo expresada en el sistema de la cámara, ${}^{C}R_{W,}$ y la traslación

del origen del mundo al origen del sistema de la cámara, expresada en coordenadas de la cámara, ${}^{C}t_{W}$.

Para conocer la orientación del robot ${}^{W}R_{R}$ en el sistema del mundo se aplica la rotación que relaciona los sistemas de la cámara y el robot ${}^{C}R_{R}$, definidos a priori durante la disposición del arco de cámaras sobre el robot. La rotación de la cámara respecto del sistema del mundo ${}^{W}R_{C}$ se calcula como la inversa de ${}^{C}R_{W,}$

$$ {}^{W}R_{R} = {}^{W}R_{C} \cdot {}^{C}R_{R} \tag{16} $$

En cuanto a la posición del robot, se transforma ${}^{C}t_{W}$ en ${}^{W}t_{C}$, posición de la cámara respecto del mundo, según la ecuación (17), y se suma la traslación entre el sistema de la cámara y el del robot, conocida a priori, en el sistema de coordenadas de la cámara ${}^{C}t_{C-R}$, ecuación (18)

$$ {}^{W}t_{C} = -({}^{W}R_{C} \cdot {}^{C}t_{W}) \tag{17} $$

$$ {}^{W}P_{R} = {}^{W}t_{C} + {}^{W}R_{C} \cdot {}^{C}t_{C-R} \tag{18} $$

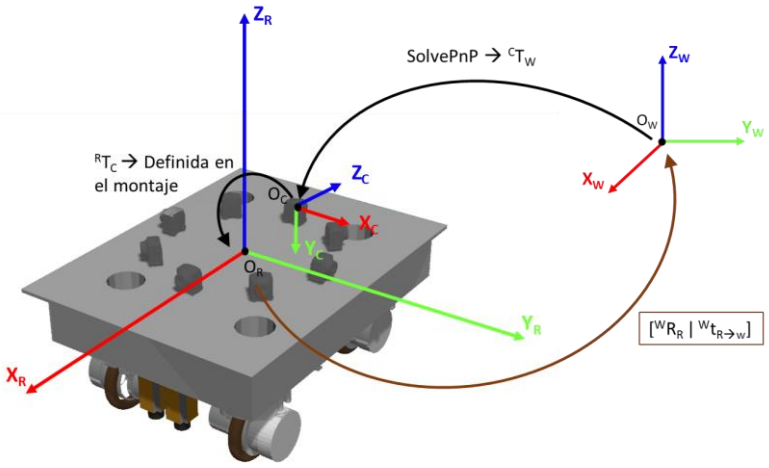

Figura 9. Transformaciones entre sistemas de referencia

2.4.2 Método estéreo

Si se encuentra un mismo marcador en dos cámaras próximas, se realiza una triangulación a partir de un punto del marcador detectado en las imágenes de ambas cámaras, obteniendo la posición de dicho punto en el espacio tridimensional.

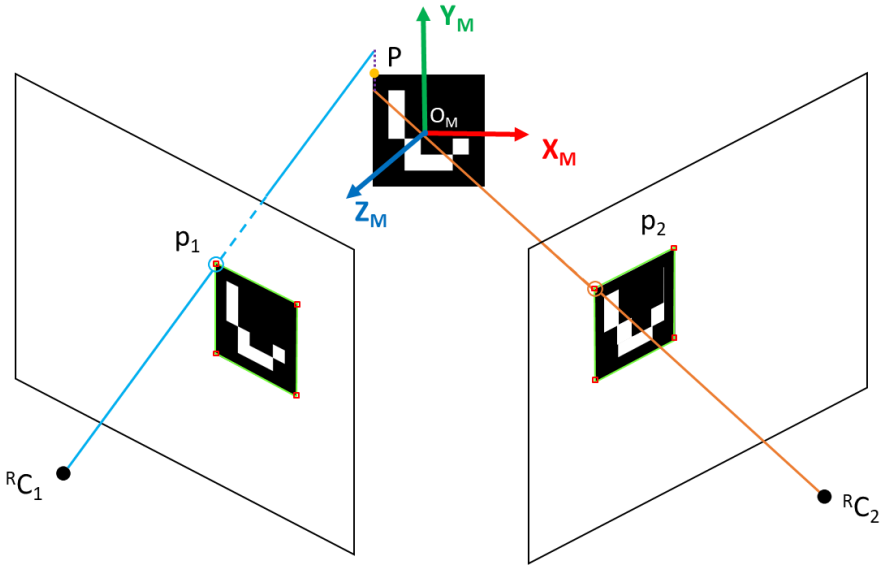

Figura 10. Triangulación a partir de la esquina superior izquierda de un mismo marcador presente en las imágenes de dos cámaras adyacentes.

El proceso de triangulación permite reconstruir la posición de un punto 3D a partir de sus proyecciones 2D en el plano imagen de dos cámaras, siendo conocidas las matrices de proyección de ambas. Ya que los parámetros extrínsecos de la cámara referidos al sistema del robot son conocidos, a partir de un punto bidimensional del marcador detectado en las imágenes de ambas cámaras, se realiza una triangulación, obteniendo la posición 3D del marcador respecto del sistema de coordenadas del robot, ${}^{R}P_{M}$. A partir de este punto y mediante la aplicación de las transformaciones correctas, es posible estimar la posición del robot respecto al sistema de coordenadas del mundo, ${}^{W}P_{R.}$ Para ello es necesario aplicar la transformación presente en la ecuación (19)**¡Error! No se encuentra el origen de la referencia.**, realizando un cambio de sistema de la posición del robot respecto al sistema del marcador al sistema buscado.

$$ {}^{W}P_{R} = {}^{W}T_{M} \cdot {}^{M}P_{R} \tag{19}$$

La matriz ${}^{W}T_{M}$ es la transformación del sistema de referencia del marcador al sistema del mundo. Es una matriz conocida, dado que la posición y orientación del marcador respecto del mundo son conocidas debido al mapa de balizas. ${}^{M}P_{R}$ es la posición del robot respecto al

sistema del marcador, sin embargo, tras realizar la triangulación se conoce la posición del marcador respecto del robot, ${}^{R}P_{M}$. Mediante la ecuación (20), se obtiene la posición del robot respecto al sistema del marcador.

$$ {}^{M}P_{R} = -({}^{M}R_{R} \cdot {}^{R}P_{M}) \tag{20} $$

La orientación se calcula a partir de los puntos 3D de las esquinas de los marcadores y sus proyecciones encontradas en la imagen. Este problema se ha descrito en el apartado anterior. En este caso se utilizan las cuatro esquinas del marcador respecto al sistema de coordenadas del propio marcador y sus correspondientes puntos en la imagen. Se obtiene la rotación de la cámara respecto al sistema del marcador ${}^{M}R_{C}$, tras transformaciones sucesivas, se obtiene la rotación del robot respecto del mundo, ${}^{W}R_{R}$.

$$ {}^{M}R_{R} = {}^{M}R_{C} \cdot {}^{C}R_{R} \tag{21} $$

$$ {}^{W}R_{R} = {}^{W}R_{M} \cdot {}^{M}R_{R} \tag{22} $$

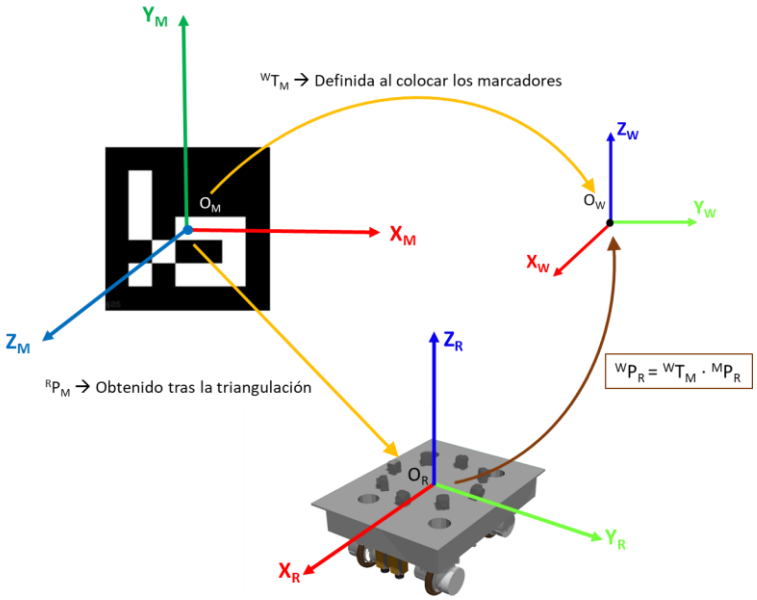

Figura 11. Esquema del cálculo de la posición según la estrategia estéreo

2.5 Detección marcadores *ArUco*

Se han empleado los marcadores ArUco [57], ampliamente utilizados en aplicaciones de realidad aumentada y en algoritmos de navegación de robots.

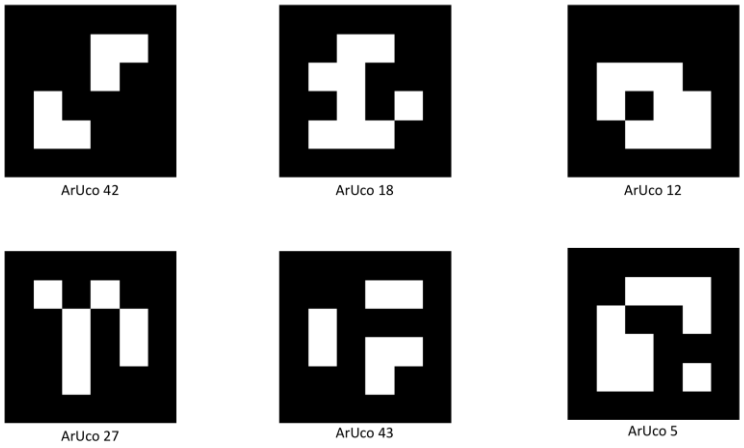

Figura 12. Marcadores *ArUco*

Un marcador ArUco, ver Figura 12, es un marcador sintético cuadrangular compuesto por un borde negro y una matriz binaria interna que determina un identificador. El tamaño del marcador determina el tamaño de la matriz interna.

Existen diferentes diccionarios ArUco que agrupan marcadores con distintos identificadores.

La detección de puede dividir en dos pasos, la detección de los candidatos y, posteriormente, la comprobación de si son realmente marcadores ArUco según la configuración de sus matrices internas.

En la primera etapa, la imagen de entrada es analizada con el fin de encontrar formas cuadradas que puedan ser marcadores. Primero, se aplica un umbralizado adaptativo [47, Ch. 6.1] para segmentarlos, después se extraen los contornos de la imagen umbralizada y se aplica el algoritmo de Suzuki and Abe [58] para descartar las formas que no se asemejen a un cuadrado.

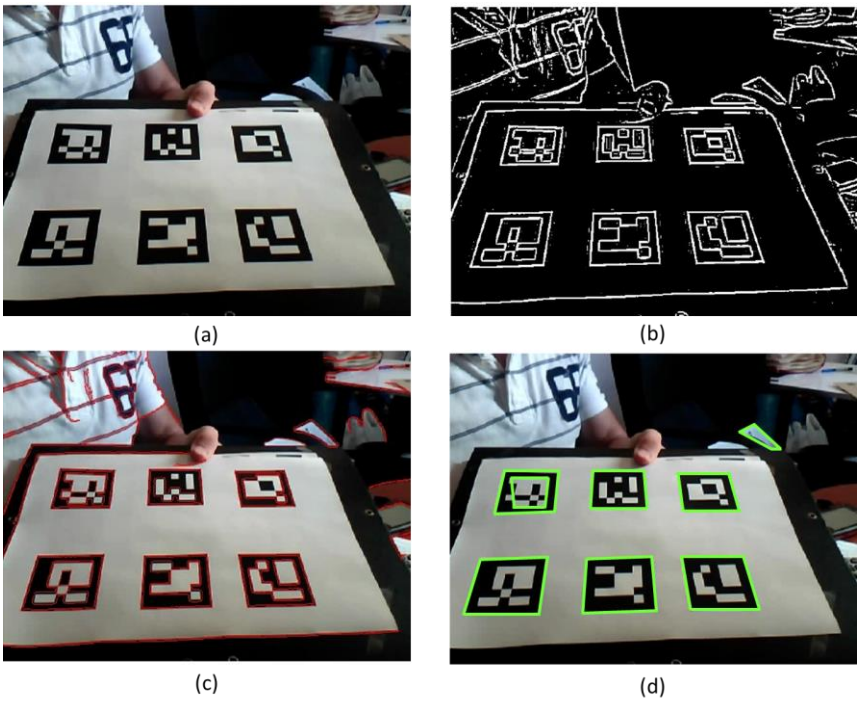

Figura 13. Proceso de detección de marcadores. (a) Imagen de entrada. (b) Imagen tras el umbralizado adaptativo. (c) Resultado de la extracción de contornos. (d) Descarte de formas que no se asemejen a un cuadrado. Imagen obtenida de [59].

Una vez se tienen los candidatos, se pasa a la segunda etapa. En primer lugar, se extraen los bits de cada marcador. Para ello, se aplica una transformación de perspectiva para obtener el marcador en su forma canónica y se umbraliza mediante el algoritmo de Otsu [60], separando los bits blancos y los negros, como se muestra en la Figura 14.

La imagen se divide en diferentes celdas según el tamaño del *ArUco* y el del borde negro, asignando cada celda con el valor de 0 o 1 según el valor de la mayoría de los píxeles que alberga. Una primera prueba de rechazo consiste en detectar la presencia del borde negro, si esto ocurre, se analizan los bits de la matriz interna para determinar finalmente si el marcador pertenece al diccionario específico.

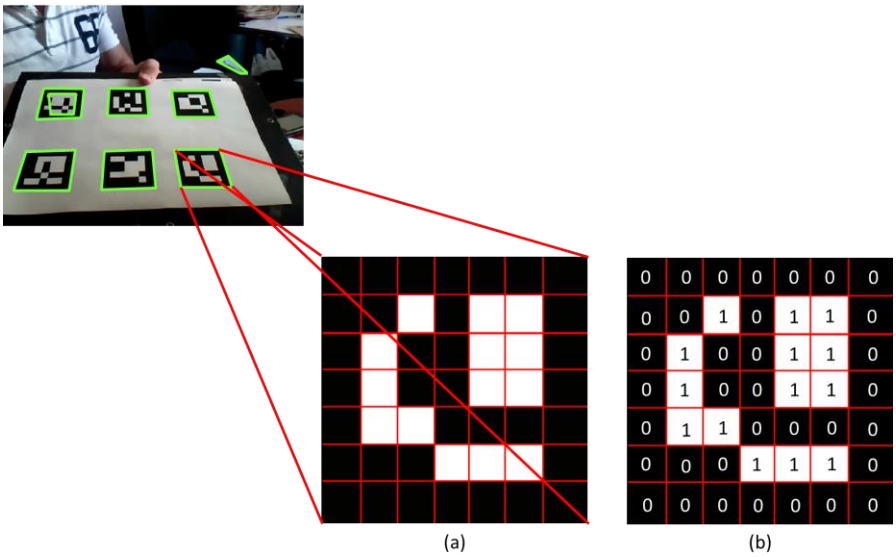

Figura 14. Resultado del marcador tras aplicar una transformación de perspectiva y la división del ArUco en celdas (a). Después se analizan los bits de la matriz para identificar el marcador dentro de un diccionario específico (b). Imagen obtenida de [59]

2.6 Localización basada en *Fovis*

Fovis [61] es una librería de odometría visual que estima el movimiento 3D de una cámara utilizando información de profundidad para cada píxel. Su primera implementación fue descrita por Huang et al para estimar el movimiento de un *Micro Air Vehicle* (MAV) a partir de la información RGB-D arrojada por una Kinect de Microsoft [14].

Se trata de un método basado en puntos característicos. Primero, se realiza un preprocesamiento de las imágenes de entrada. La imagen RGB es convertida a escala de grises y suavizada mediante filtros gaussianos, construyendo una pirámide gaussiana para permitir la detección de puntos característicos a diferentes escalas [62], lo que proporciona robustez frente a imágenes borrosas e invariancia ante cambios de escala.

Después, se detectan los puntos característicos de las imágenes de entrada utilizando el algoritmo FAST [63], y se extrae la profundidad correspondiente a cada uno de la imagen de profundidad, descartando las puntos que no tengan una profundidad asociada.

A cada punto característico se le asigna un descriptor que consiste en los valores de brillo de la región de 9x9 píxeles alrededor del mismo. Posteriormente, se buscan correspondencias de esos puntos entre *frames* mediante la comparación de los valores de sus descriptores.

Con el fin de seleccionar únicamente las correspondencias correctas, se procede a una etapa de eliminación de *outliers* o falsas correspondencias basada en la aproximación de Howard [64].

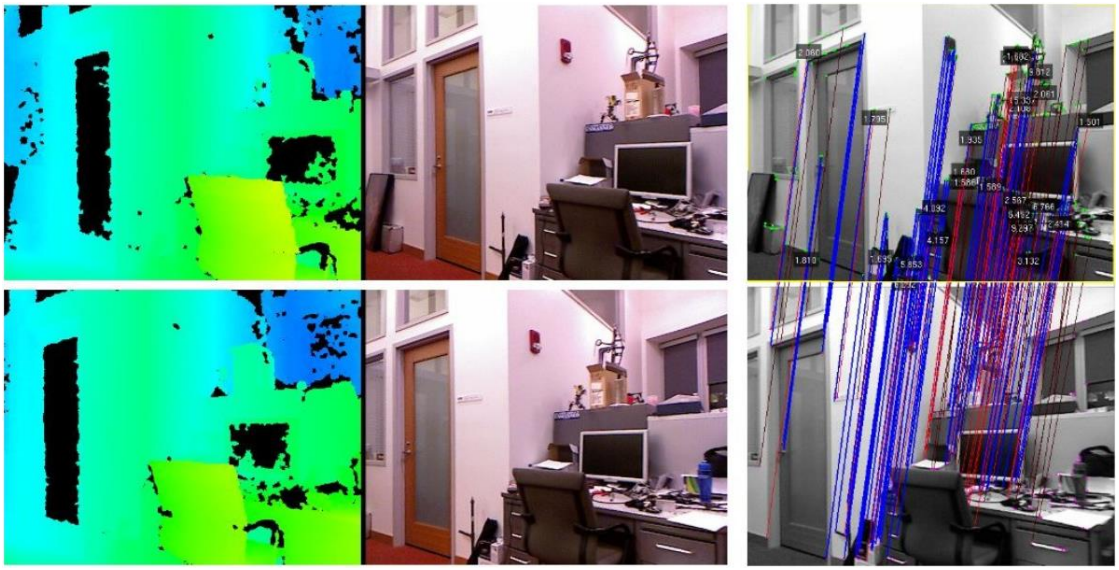

Figura 15. La primera fila hace referencia al instante t y la inferior a $t+\Delta t$. La columna de la izquierda muestra las imágenes de profundidad y la del medio las imágenes RGBD. La columna de la derecha muestra las correspondencias entre puntos característicos, encontrados por el algoritmo FAST, emparejadas entre *frames*, donde los *outliers* eliminados son mostrados en color rojo. Imagen obtenida de [14].

Finalmente, la estimación de movimiento es calculada a partir de las correspondencias en tres pasos. En primer lugar, se aplica el método de Horn para proporcionar una estimación inicial al minimizar las distancias euclídeas entre las correspondencias [65]. Después, la estimación es refinada minimizando el error de reproyección [66, Ch. 4.2.3] de los puntos característicos mediante un algoritmo de mínimos cuadrados no lineal [67]. Finalmente, las correspondencias que exceden un umbral de error de reproyección fijo son descartadas del conjunto de los *inliers* y la estimación de movimiento se refina una vez más.

Con el fin de reducir el error en situaciones donde el punto de vista de la cámara no varía significativamente, se utiliza una técnica de *key-frame*, estimando el movimiento mediante la comparación de una nueva imagen con un *frame* de referencia. Si el movimiento es calculado con el suficiente número de *inliers*, el *frame* de referencia no se cambia. Si se da el caso contrario, el nuevo *frame* pasa a ser el de referencia.

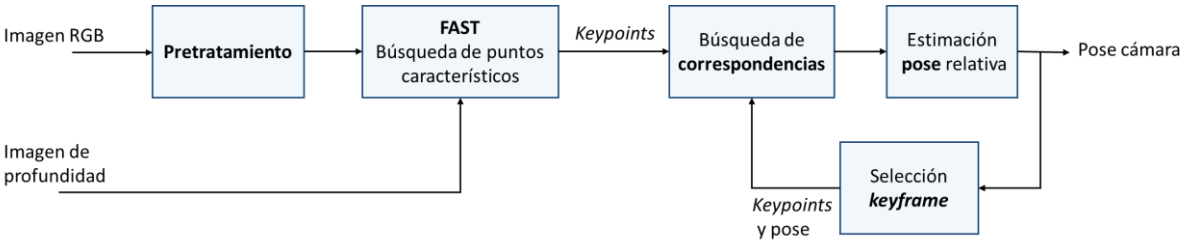

Figura 16. Esquema interno del método seguido por la librería Fovis.

2.1 Localización basada en *OpenCV RGB-Depth Processing*

Maria Dimashova desarrolló el módulo RGB-Depth Processing [20] para la librería OpenCV [21]. Este método está inspirado en el trabajo de Steinbrucker et al, en el que presentó un método global a partir de las imágenes RGB-D proporcionadas por una Kinect de Microsoft [22]. En la Figura 18 se ilustra el proceso seguido por el algoritmo implementado en el módulo de OpenCV [68].

Se trata de un método global, es decir, utiliza la intensidad de los píxeles, y utiliza una estrategia de búsqueda de correspondencias *frame* a *frame*.

En este método, se propone un enfoque de minimización de la intensidad de los píxeles para estimar el movimiento de la cámara entre *frames* a partir de las imágenes RGBD. La idea clave es abordar el problema inverso subyacente, minimizando el error de reproyección. El objetivo es encontrar una transformación rígida que represente el movimiento de la cámara de manera que la segunda imagen registrada coincida exactamente con la primera, imagen (c) de la Figura 17.

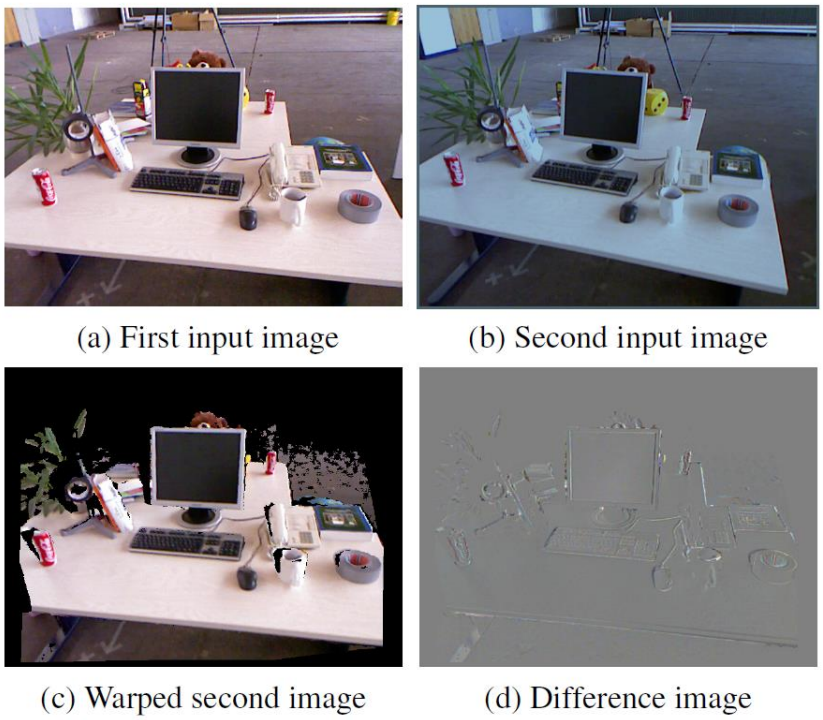

Figura 17. Se propone una aproximación para estimar el movimiento de la cámara entre imágenes RGBD (a)+(b). La idea es encontrar una transformación rígida que transforme la segunda imagen en la primera (c), es decir, la imagen diferencia (d) calculada para localizaciones de profundidad confiable debe de ser cero (= gris). Imagen obtenida del artículo original del método [22].

Para asegurar robustez frente a grandes cambios de movimiento, se aplica una aproximación de grueso a fino al trabajar en una pirámide de imágenes. El algoritmo calcula la transformación que relaciona dos *frames* minimizando la diferencia de intensidad entre el actual *warped RGB-D frame* y el *frame* anterior, imagen (d) de la Figura 17.

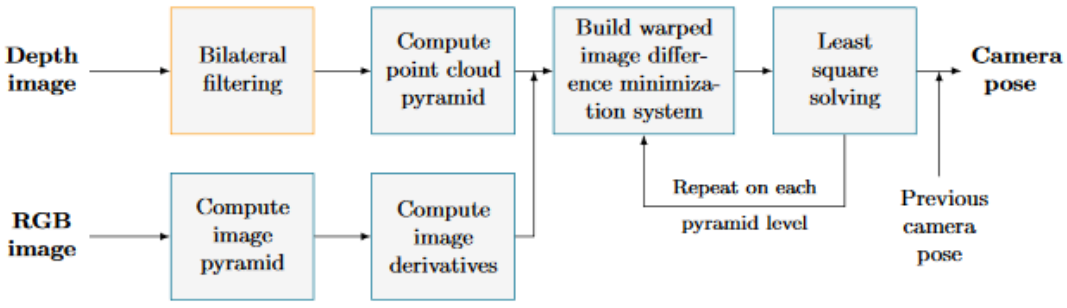

Figura 18. Esquema interno del método seguido por el módulo OpenCV RGB-Depth processing. Imagen obtenida de [69].

2.2 ROS (*Robot Operating System*)

El conjunto de los diferentes módulos que engloba el proyecto se enmarca en ROS (*Robot Operating System*). ROS es un *framework* para el desarrollo de software para robots que provee diferentes librerías y herramientas que permiten la creación de distintas aplicaciones [70].

ROS surge de la necesidad, por parte de la comunidad de investigación en robótica, de disponer de un *framework* de colaboración de código abierto. Sus orígenes datan a mediados de la década de los 2000, cuando investigadores de la Universidad de Stanford desarrollaron una serie de prototipos para dar soporte al proyecto de un robot con inteligencia artificial incorporada, *STAIR* [71].

Posteriormente, el instituto de investigación en robótica, Willow Garage [72], proporcionó los recursos necesarios para dar un impulso a este *framework* y desarrollar un mayor número de implementaciones desde entonces. En todo momento, el *software* se desarrolló utilizando la licencia de código abierto BSD [73] y, gradualmente, se ha convertido en una plataforma ampliamente utilizada por la comunidad de investigación científica.

Actualmente, la comunidad de ROS consta de decenas de miles de usuarios de todo el mundo, trabajando en aplicaciones para ámbitos muy diversos, desde proyectos de ocio hasta sistemas de automatización industrial.

ROS solo se puede incorporar a plataformas basadas en Unix. El software para ROS se utiliza principalmente en los sistemas Ubuntu y Mac OS X, aunque la comunidad ha estado contribuyendo con el soporte para Fedora, Gentoo, Arch Linux y otras plataformas Linux. Si bien es posible un puerto a Microsoft Windows para ROS, aún no lo han desarrollado completamente.

2.2.1 Conceptos básicos

En este apartado se van a repasar los conceptos más básicos de ROS para permitir entender el esquema del paquete de localización desarrollado.

A menudo resulta de gran utilidad dividir el software del robot en partes más pequeñas e independientes que cooperen para lograr el objetivo general. Estas partes independientes de software se conocen bajo el nombre de paquetes. Un paquete puede contener procesos de

tiempo de ejecución de ROS, llamados nodos, una biblioteca dependiente de ROS, conjuntos de datos, archivos de configuración o cualquier otra cosa que sea útil organizar conjuntamente. Además, se pueden distinguir los metapaquetes, paquetes que representan un grupo de otros paquetes relacionados.

Los nodos son procesos que llevan a cabo algún tipo de cálculo, escritos en algún lenguaje de programación, como C++ o Python. ROS está compuesto por un conjunto de nodos independientes, que se pueden comunicar entre sí mediante mensajes utilizando un modelo publicador/subscriptor o cliente/servidor. El ROS Master permite que los nodos se encuentren para poder comunicarse mediante mensajes. Un mensaje es una estructura de datos de un tipo determinado destinada a la comunicación entre nodos.

En el modelo publicador/subscriptor, los mensajes se envían publicando en un canal llamado *topic*, y los subscriptores pueden acceder a dicha información accediendo a ese canal. El nodo publicador no controla quién se subscribe, por lo tanto, la información es asíncrona. Los *topics* se crean con un nombre identificador y están asociados a un tipo de dato determinado que corresponde con el tipo de mensajes que pueden contener. La información publicada en los *topics* se puede grabar en archivos tipo *bag* para permitir su posterior reproducción cuando sea necesario, gestionado mediante la herramienta *rosbag* [74].

Si se busca una comunicación síncrona se deberá utilizar el modelo cliente/servidor. Este modelo se lleva a cabo a través de servicios, que se definen por dos estructuras de mensajes, uno para la solicitud y otro para la respuesta. El nodo servidor ofrece un servicio con un nombre y el cliente utiliza dicho servicio mediante el envío de un mensaje de solicitud y se queda en espera de la respuesta.

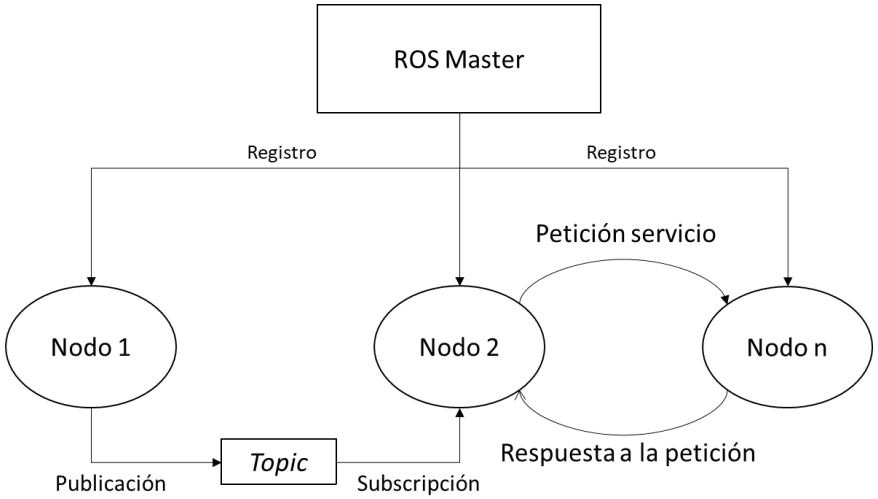

Figura 19. Esquema básico de ROS

3. IMPLEMENTACIÓN EN ROS

Volviendo a la Figura 3 expuesta anteriormente, la estrategia de localización implementada se puede dividir en dos partes claramente diferenciadas. Primero, la detección de los marcadores y, después, el cálculo de la posición y orientación del robot a partir de las mismas, nombrado como Robot Pose.

Como ya se ha comentado previamente, todos los algoritmos se encuentran implementados en forma de paquetes de ROS.

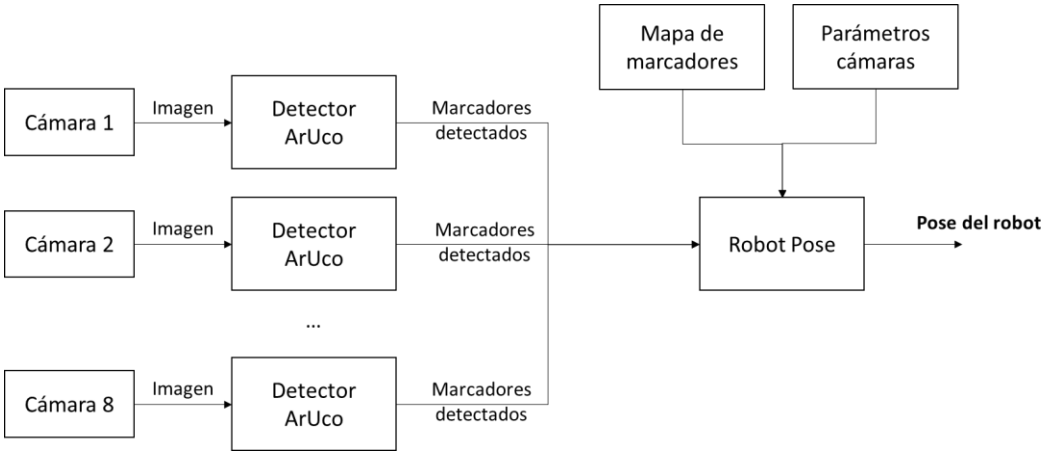

Figura 20. Esquema del método propuesto. *Robot Pose* hace referencia al paquete implementado.

3.1 Simulador Gazebo

Se utilizó el simulador Gazebo [75] con el fin de recrear un entorno industrial que pretende emular las instalaciones por las que se moverá el robot. Se trata de una herramienta para la creación, edición y simulación de entornos, utilizando modelos 3D, que permite la integración con ROS.

Gazebo cuenta con una amplia librería con modelos de diferentes sensores y robots reales. Emulan, tanto su apariencia física, como su comportamiento mediante *plugins* personalizados que imitan la comunicación con el dispositivo real. Esto facilita en gran medida el paso de la simulación al sistema real.

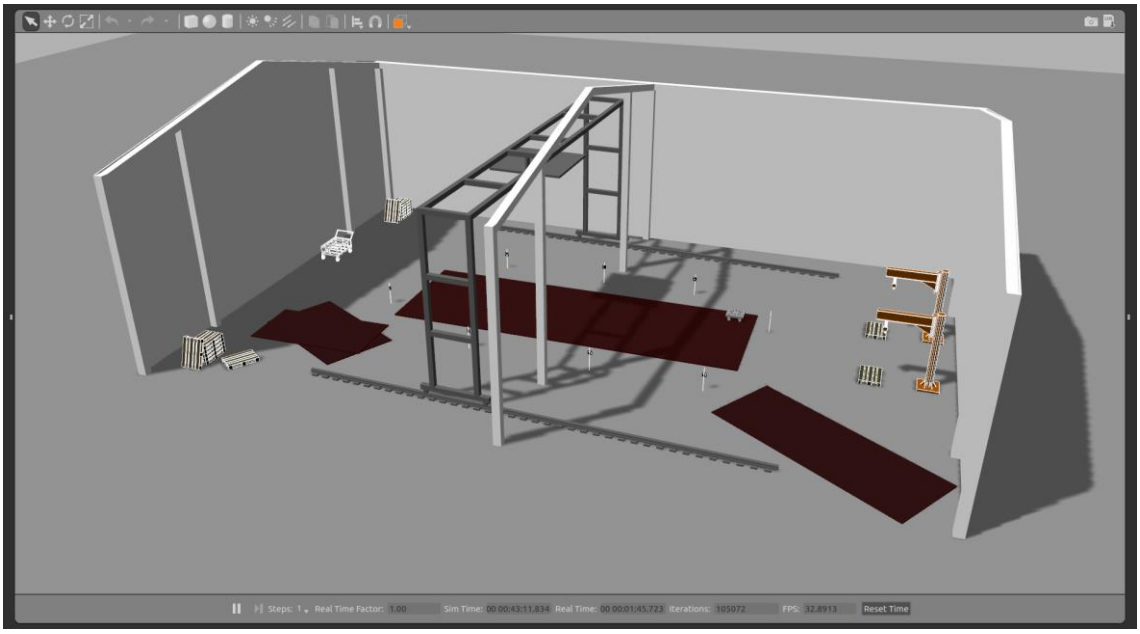

Figura 21. Interfaz de Gazebo mostrando el entorno que emula la industria desarrollado.

3.1 Configuración del robot

En el proyecto global, se utilizará un robot omnidireccional [76], por ello, para las simulaciones se empleó un modelo de un MPO-700 de Neobotix [77], ya que emplea las mismas ruedas que llevará el modelo real. Sobre él se dispusieron un total de 8 cámaras formando un arco, según la configuración mostrada en la Figura 22.

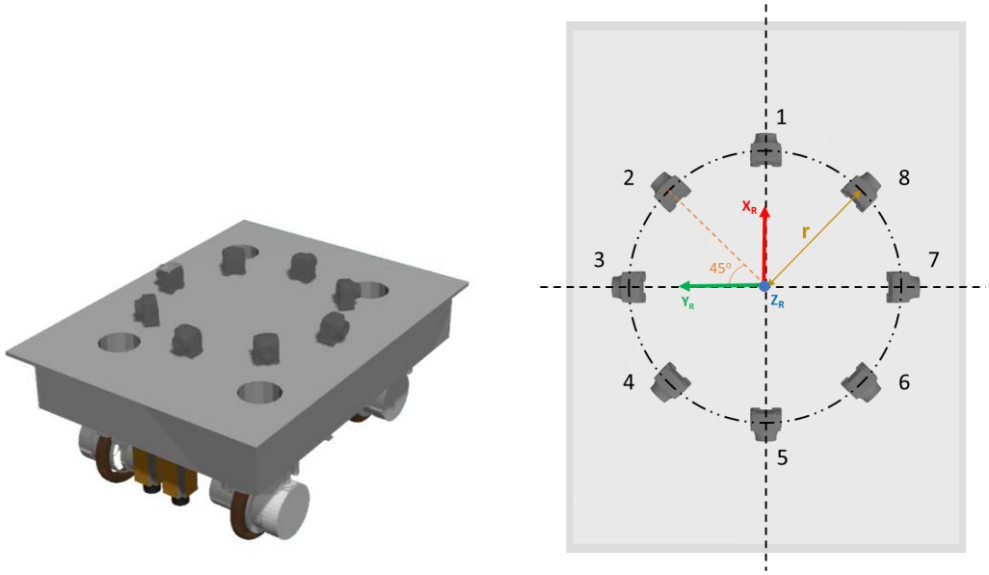

Figura 22. Configuración de las 8 cámaras que forman el arco que va dispuesto sobre el robot.

Los parámetros de las cámaras, tanto intrínsecos como extrínsecos, se almacenan en un archivo tipo YAML. Los parámetros extrínsecos se refieren al sistema del robot.

Los parámetros intrínsecos se obtienen tras realizar una calibración de la cámara. En este proyecto, al utilizar una simulación, los parámetros venían definidos directamente en el modelo de la cámara utilizada. Las 8 cámaras utilizaron el mismo modelo, presentando una resolución de 1280x720 píxeles y un campo de visión horizontal (FOV) de 60º. Su matriz de calibración se puede ver en la ecuación (23).

$$K = \begin{bmatrix} 1108.76 & 0 & 640.5 & 0 \\ 0 & 1108.76 & 360.5 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (23)$$

Para modelar el comportamiento de las cámaras se ha utilizado el *plugin libgazebo_ros_camera.so* [78], que provee una interfaz de ROS para simular cámaras publicando las imágenes capturadas y la información de la cámara en diferentes *topics* con el tipo de mensaje correspondiente. Las imágenes en mensajes de tipo *sensor_msgs/Image.msg* [79] y la información de los parámetros de la cámara, tipo *sensor_msgs/CameraInfo.msg* [80].

3.2 Sistemas de coordenadas

En cualquier aplicación de robótica resulta fundamental el empleo de diferentes sistemas de coordenadas. En ROS, el paquete más utilizado para la gestión de estos sistemas o *frames* y las operaciones que involucran es *tf (The Transform Library)* [81]. Esta librería fue presentada por Tully Foote en [82]. Posteriormente, se desarrolló *tf2* [83], una versión más eficiente que la original.

En el desarrollo del paquete de localización llevado a cabo en este trabajo se ha utilizado la librería *tf2*, definiendo cuatro sistemas de coordenadas diferentes. El sistema del mundo, definido en una esquina de la industria, será el sistema sobre el que se referirán las poses de los marcadores almacenados en la base de datos y la pose del robot estimada. Además, tanto las cámaras, como los marcadores y el robot tendrán su propio sistema de coordenadas. Ver Figura 23 y Figura 24.

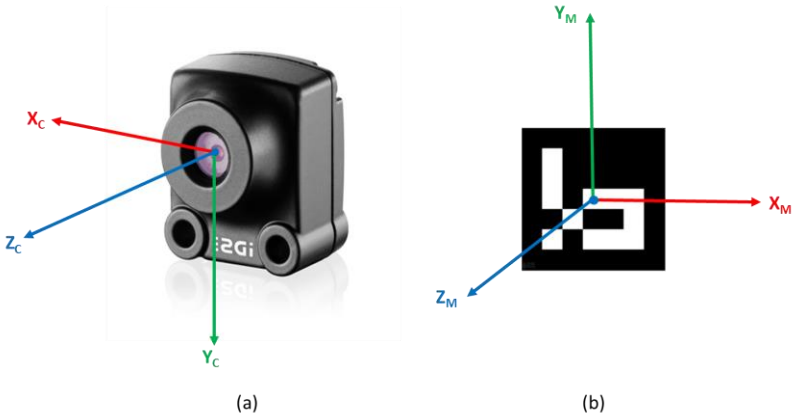

Figura 23. Sistemas de coordenadas de una cámara (a) y de un ArUco. Origen en el centro del marcador (b).

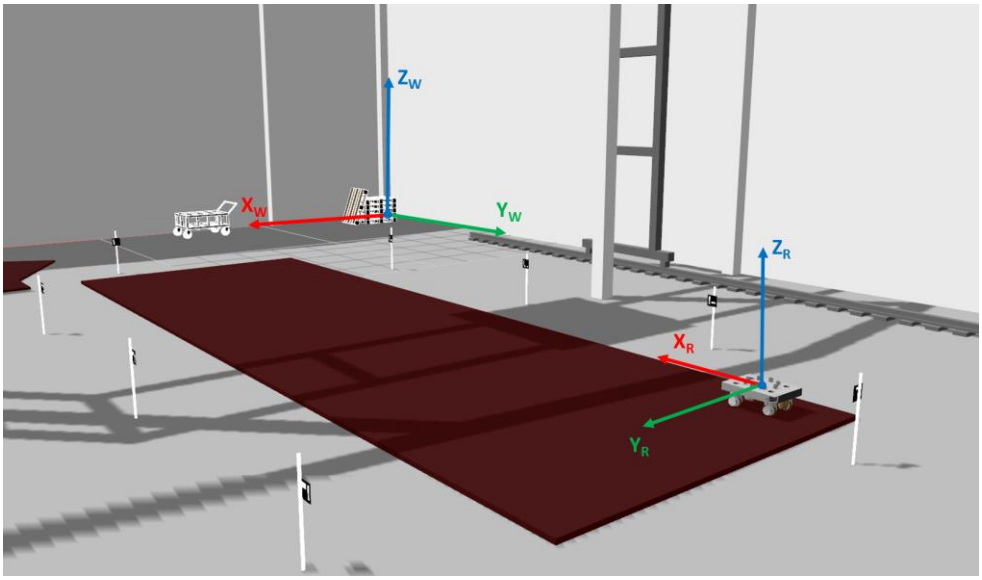

Figura 24. Entorno que representa la industria. Se pueden apreciar los sistemas de coordenadas del mundo situados en una esquina de la fábrica (Ejes X_W, Y_W y Z_W) y del robot (Ejes X_R, Y_R y Z_R).

3.1 Mapa de marcadores

Al tratarse de una simulación, la localización de los marcadores se obtiene directamente de Gazebo por mayor comodidad. Por ese motivo, se ha desarrollado un paquete de ROS que se subscribe al *topic* */gazebo/model_states*[84] con mensajes del tipo

gazebo_msgs/ModelStates.msg, que almacena la pose de todos los modelos presentes en la simulación.

La información sobre la pose de los marcadores se almacena en un archivo tipo YAML, al igual que los parámetros de las cámaras.

En este archivo se almacena el número de marcadores, el tamaño de su lado y sus posiciones y orientaciones respecto del sistema de coordenadas del mundo.

Es necesario ejecutar este paquete cada vez que se realicen modificaciones en la localización o en el tamaño de los marcadores.

3.2 Detección marcadores ArUco

La detección de los marcadores se ha realizado utilizando un paquete para ROS ya desarrollado, *aruco_detect* [46], basado en el módulo Aruco de OpenCV [68].

Este paquete detecta los marcadores presentes en las imágenes procedentes de una cámara. Una vez detectados, publica las posiciones 2D de los vértices de los marcadores detectados en la imagen, así como su identificador según el diccionario escogido.

Dada una imagen con varios ArUco presentes en ella, el proceso de detección devuelve una lista con los marcadores detectados. Cada uno incluye la posición de sus cuatro esquinas en la imagen y su identificador. El detector es capaz de encontrar los marcadores rotados en el entorno.

Con el fin de utilizar el paquete para llevar a cabo la detección de los marcadores, es necesario definir algunos parámetros clave, como son la medida del lado, en metros, y el diccionario al que pertenecen los ArUco que se pretende detectar.

La información sobre la localización de los marcadores detectados es publicada en el *topic* correspondiente en forma de mensajes de tipo *fiducial_msgs::FiducialArray*. Se explica más detalladamente este tipo de mensaje en las siguientes tablas.

Tabla 1. Parámetros detallados de los mensajes de tipo *fiducial_msgs::FiducialArray*

Parámetro	**Tipo**	**Descripción**
header	Header	Metadatos estándar. Utilizados para comunicar datos con marca de tiempo en un sistema de coordenadas en particular.
image_seq	int32	Identificador de la imagen en la que se han detectado los marcadores.
fiducials	Fiducial[]	Array con los marcadores detectados. Explicado en la Tabla 1.

Tabla 2. Parámetros detallados de los mensajes de tipo *fiducial_msgs::Fiducial*

Parámetro	**Tipo**	**Descripción**
Fiducial_id	int32	Identificador del marcador detectado
x0	float64	Posición x de la esquina 0
y0	float64	Posición y de la esquina 0
x1	float64	Posición x de la esquina 1
y1	float64	Posición y de la esquina 1
x2	float64	Posición x de la esquina 2
y2	float64	Posición y de la esquina 2
x3	float64	Posición x de la esquina 3
y3	float64	Posición y de la esquina 3

3.3 Paquete de localización

La localización del robot mediante los marcadores ArUco se ha implementado en un paquete para ROS, desarrollado en C++. Se ha desarrollado de manera que se subscriba a los *topics* que contienen la información publicada por el paquete *aruco_detect* ejecutándose sobre las imágenes de cada una de las 8 cámaras.

Como se ha explicado previamente, es necesario conocer la localización de los marcadores en el entorno referida al sistema del mundo. Por ello, dicha información se debe registrar en un archivo que será leído al comienzo de la ejecución del paquete de localización. Así como los parámetros de las cámaras, tanto intrínsecos como extrínsecos, dispuestas sobre el robot.

3.3.1 *Topics* a los que se suscribe

Para que el nodo se ejecute correctamente, es necesario la ejecución simultánea de los nodos ejecutados por el paquete *aruco_detect*, proporcionando la localización de los marcadores presentes en las imágenes de las cámaras en todo momento. El paquete se subscribe a los 8 *topics* generados por aruco_*detect* ejecutándose en cada cámara. Cada uno sigue una estructura similar a la siguiente:

- /fiducial_vertices (fiducial_msgs/FiducialArray)

El tipo de mensaje FiducialArray se detalló en la Tabla 1.

3.3.2 *Topics* en los que publica

La posición y orientación del robot estimada en este paquete es publicada en otro *topic*, en mensajes de tipo *geometry_msgs::Pose*, de manera que otros módulos del proyecto puedan acceder a esta información.

Tabla 3. Parámetros detallados de los mensajes de tipo *geometry_msgs::Pose*

Parámetro	**Tipo**	**Descripción**
position	point	Posición 3D (X,Y,Z)

orientation	quaternion	Orientación expresada en cuaterniones (x,y,z,w)

La posición se almacena en un punto 3D, mientras que la orientación se almacena en forma de cuaterniones [85].

3.3.3 Parámetros

Para el correcto funcionamiento del paquete es necesario definir correctamente dos parámetros:

- path_fiducials_map (string)

  Ruta al archivo de tipo YAML que almacena la localización de los marcadores, respecto al sistema de coordenadas de referencia, dispuestos sobre la escena.

- path_camera_params (string)

  Ruta al archivo de tipo YAML que almacena los parámetros de cada una de las cámaras.

3.4 Estructura en ROS

Como resumen del capítulo se incluye el esquema de la Figura 25, obtenida mediante la herramienta *rqt_graph* [86]. Se ha modificado para mostrar únicamente los nodos y *topics* involucrados en la estrategia de localización propuesta.

Se puede observar que, cada paquete de detección de marcadores ArUco, *aruco_detect*, se subscribe a los *topics* que proporcionan las imágenes y la información de cada cámara. Además, cada uno de ellos publica los marcadores encontrados en otro *topic*. Así, son publicados 8 *topics*, uno por cada cámara, a los que se subscribe el paquete de localización desarrollado, *robot_pose*. Este paquete se subscribe, además, al *topic* proporcionado por Gazebo, */gazebo/model_states* con el fin de obtener la pose real del robot para poder extraer información de error. Finalmente, la pose del robot estimada es publicada en */robot_pose*.

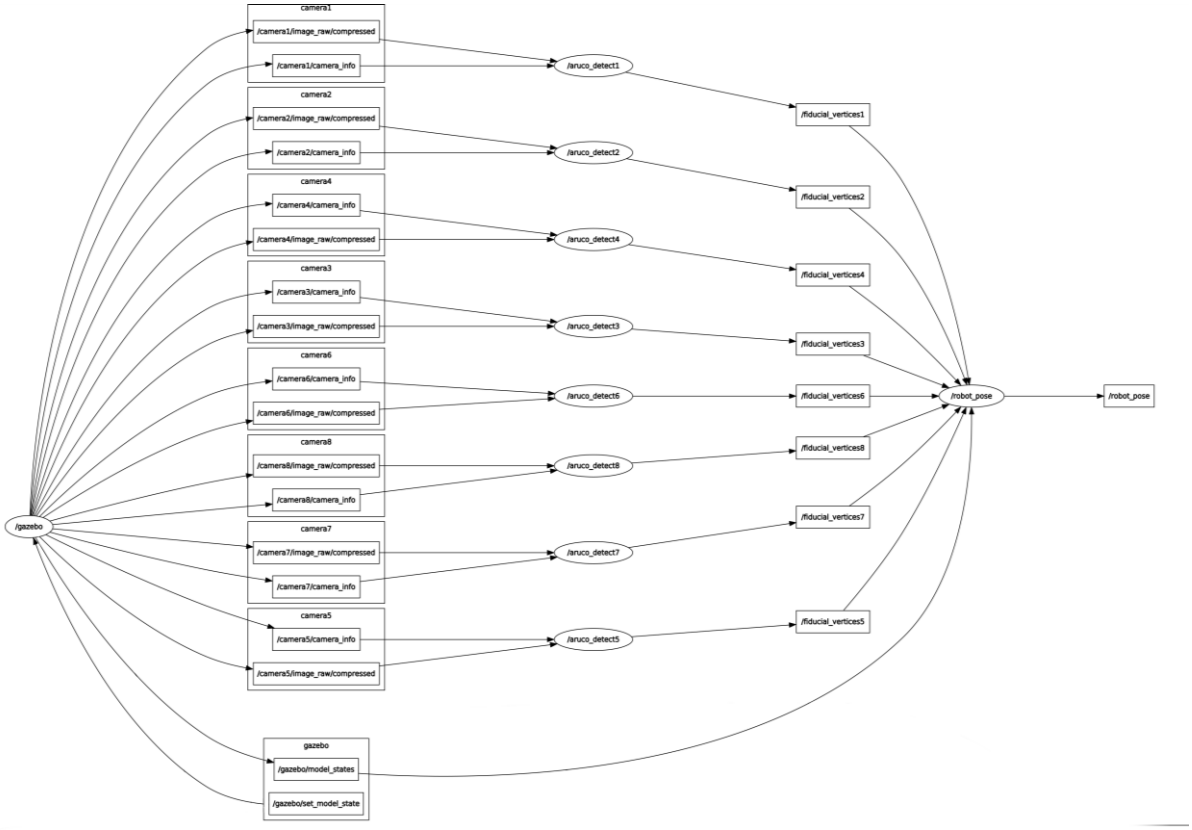

Figura 25. Representación gráfica de los principales nodos y *topics* involucrados en la estrategia de localización propuesta. Las elipses representan a los nodos y los rectángulos a los *topics*.

4. EXPERIMENTOS Y RESULTADOS

Se llevaron a cabo diferentes experimentos con el fin de evaluar alguno de los métodos expuestos anteriormente. Las pruebas se desarrollaron sobre el sistema operativo Ubuntu 16.04 LTS y ROS (*Robot Operating System*) [87], que proporciona las librerías y herramientas necesarias para implementar aplicaciones de robótica. En concreto, se utilizó la versión ROS Kinetic.

Además, como se mencionó anteriormente, se utilizó el simulador Gazebo [75] con el fin de recrear un entorno industrial que pretende emular las instalaciones por las que se moverá el robot.

En el entorno se incluye un robot móvil con una serie de sensores sobre el mismo, los cuales variarán según el experimento desarrollado.

El modelo de robot utilizado no es importante en este apartado del proyecto, ya que la localización es independiente al tipo de movimiento realizado. Además, tampoco la manera en la que se mueve el mismo, por ello, se realizaron trayectorias guiadas desde el teclado mediante paquetes de ROS existentes.

Los datos recabados durante los experimentos se fueron almacenando en archivos .csv, para su posterior análisis. La carga y el procesado de los datos se ha realizado mediante *pandas* [88], una librería para *Python* especializada para manipulación y análisis de datos.

En cuanto a métodos basados en odometría visual se han probado dos librerías diferentes, *Fovis* y *OpenCV RGB-Depth procesing*.

Se realizaron diferentes experimentos. En cuanto a la librería Fovis, se realizaron dos paquetes para permitir evaluar su funcionamiento, uno para cámaras RGBD y otro para un par estéreo. Además, se analizaron los resultados obtenidos en diferentes entornos.

También se realizó un paquete utilizando la librería de OpenCV RGB-Depth Processing. En este caso, su evaluación tuvo lugar utilizando un *dataset* ya existente, especialmente diseñado para algoritmos de odometría visual a partir de información RGBD.

La evaluación del sistema de localización mediante marcadores propuesto se ha llevado a cabo en dos partes diferenciadas. Por un lado, se evaluó la detección de los marcadores

llevada a cabo por el paquete *aruco_detect*, modificando la distancia de los marcadores a la cámara, así como sus tamaños. Por otro lado, se analizó la eficacia en el proceso de localización, variando el número de marcadores, su tamaño y su disposición en el entorno.

A continuación, se exponen los resultados obtenidos durante los diferentes experimentos mencionados.

4.1 Fovis

Fovis es una librería de odometría visual que permite estimar la posición 3D de una cámara. Está diseñada para trabajar con cámaras RGB-D o cámaras estéreo calibradas. El algoritmo diseñado para el primer tipo de cámaras necesita imágenes RGB y de profundidad para asociar un valor de profundidad a cada píxel en la imagen entrante. En el segundo modo, la información de profundidad es calculada a partir de las imágenes de un par estéreo calibrado.

Se han realizado diferentes pruebas con ambos tipos de sensores, en un caso un modelo que emula una Kinect de Microsoft y en otro un par estéreo. Para modelar el comportamiento de la Kinect se ha utilizado el *plugin libgazebo_ros_openni_kinect.so* [89], dotando al modelo de un funcionamiento análogo al de la cámara real. Para emular el comportamiento de un par estéreo, el *plugin* utilizado fue *libgazebo_ros_multicamera.so* [90].

En este caso, el entorno empleado tiene como base la industria desarrollada con la inclusión de objetos extra que puedan proporcionar más textura al entorno. Fovis, al tratarse de un algoritmo de odometría visual basado en puntos característicos, sus resultados son mejores en escenas con una mayor densidad de objetos y texturas.

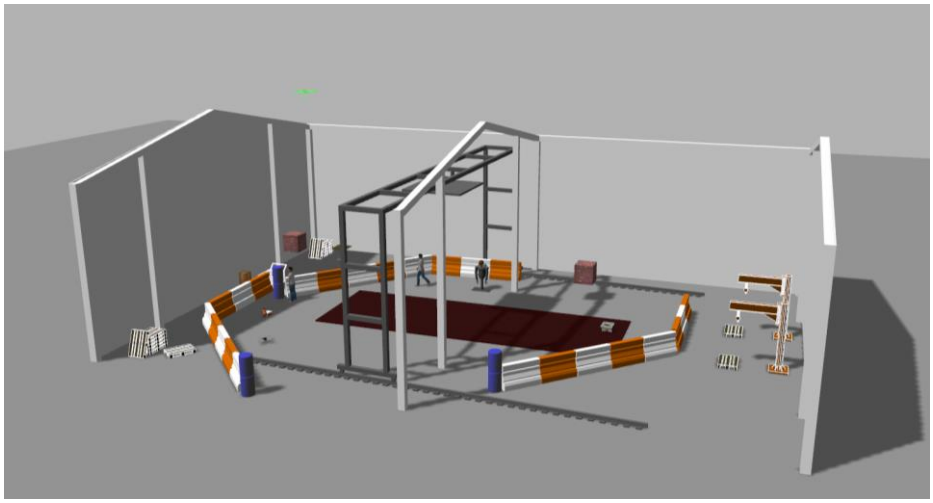

Figura 26. Entorno sobre el que se realizaron los experimentos. El tamaño del recinto es, aproximadamente, de 30x20m.

En la Figura 27, se pueden apreciar los resultados obtenidos mediante la aplicación del algoritmo para cámaras RGB-D. La trayectoria seguida fue sobre la chapa, dimensiones 10x5 metros, presente en el entorno de la Figura 26. A primera vista, se puede apreciar que los resultados no son especialmente satisfactorios, ya que la trayectoria estimada no se ajusta a la real desde el primer momento. Al ser un algoritmo incremental el error se va acumulando, sobre todo en grandes recorridos, como puede apreciarse en la Figura 28.

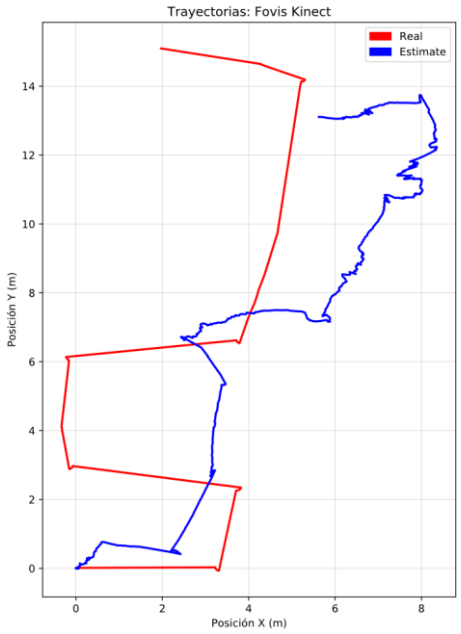

Figura 27. Fovis: Trayectoria real (rojo) y estimada (azul) a partir de información RGB-D proporcionada por una Kinect situada sobre el robot. La trayectoria recorre la chapa incluida en el entorno mostrado en la Figura 26.

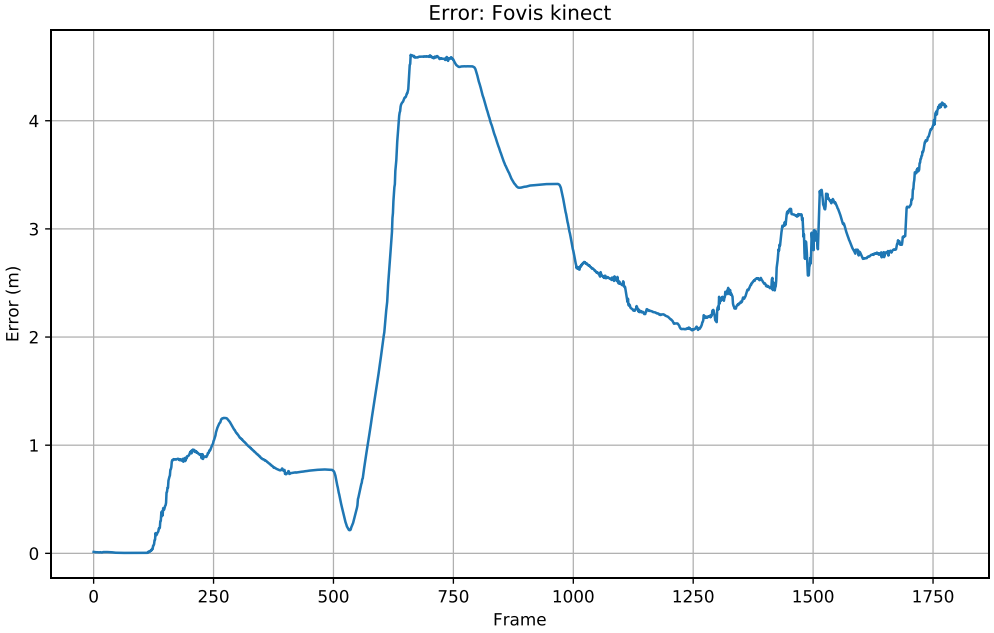

Figura 28. Algoritmo Fovis para Kinect. Evolución del error de la posición a lo largo de los *frames* capturados.

Fovis es una librería que implementa una odometría basada en puntos característicos. Además, el sensor de profundidad de la Kinect tiene un rango de 8 metros. Debido a estos motivos, junto a que la industria es de un tamaño de 30x20 metros y que sobre la chapa no se presenta ningún objeto, los resultados obtenidos no son buenos.

Sin embargo, los resultados mejoran en el caso de utilizar un par estéreo. Esto podría ser debido a que las cámaras presentan un rango de visión mayor. En la Figura 29, se puede apreciar un recorrido sobre la chapa similar al anterior. En este caso, aunque la trayectoria estimada tampoco se ajuste a la real, se puede apreciar que, al haber un error de estimación en la posición X de 1.5 metros, la estimación posterior degeneró al acumular el error, alcanzando un error hasta de 7 metros, ver Figura 30.

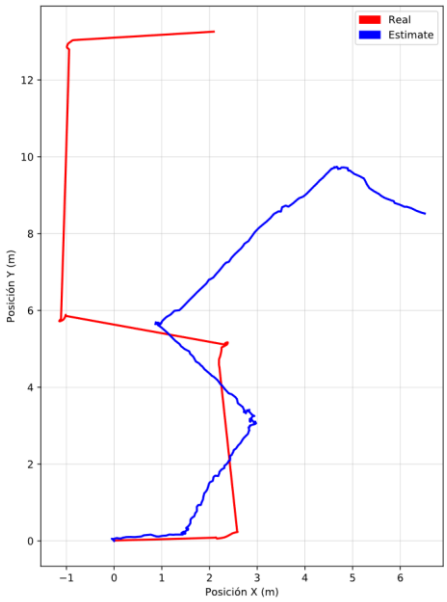

Figura 29. Fovis: Trayectoria real (rojo) y estimada (azul) a partir de información proporcionada por un par estéreo situado sobre el robot. La trayectoria recorre la chapa incluida en el entorno mostrado en la Figura 26.

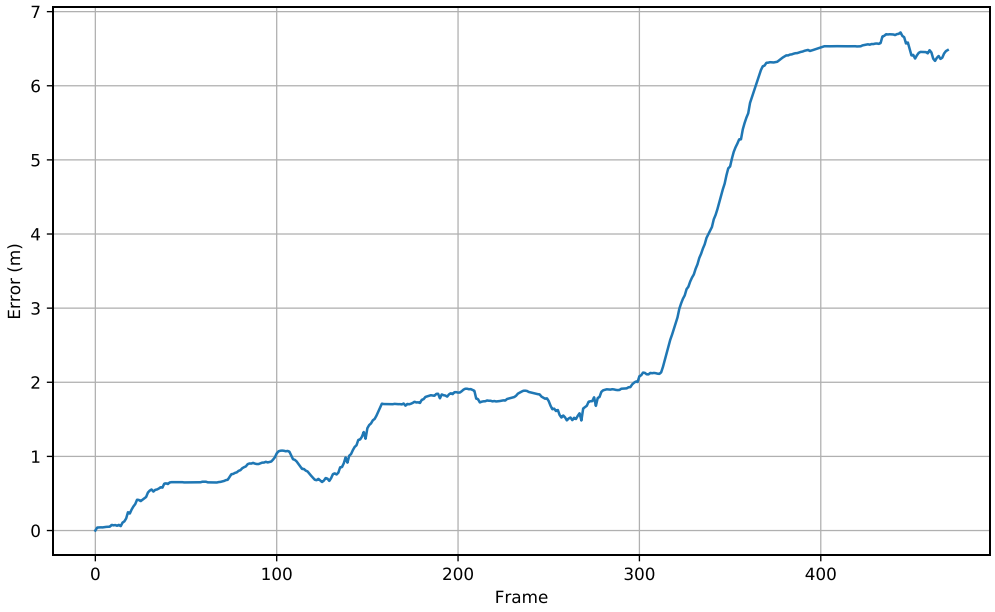

Figura 30. Algoritmo Fovis para un par estéreo. Evolución del error de la posición a lo largo de los *frames* capturados.

Se simuló un entorno más reducido y controlado, con mayor densidad de objetos, con el fin de comprobar si los resultados mejoraban. En este caso, las dimensiones del recinto son 10x10m y no dispone de una chapa a inspeccionar.

Figura 31. Entorno sobre el que se realizaron los experimentos mostrados en la Figura 32. El tamaño del recinto es de 10x10 metros.

Acorde con la Figura 32 y Figura 33, los resultados arrojados tampoco fueron satisfactorios. La trayectoria se asemeja durante los 3 primeros metros, pudiéndose comprobar que es en los giros cuando se pierde en mayor medida.

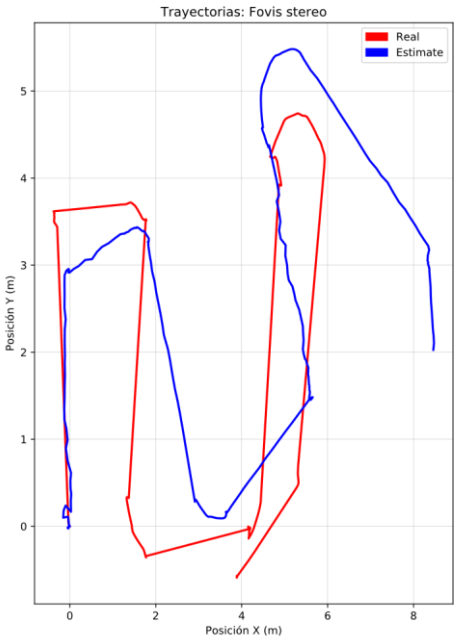

Figura 32. Fovis: Trayectoria real (rojo) y estimada (azul) a partir de información proporcionada por un par estéreo situado sobre el robot. El robot hace un recorrido por el entorno de la Figura 31.

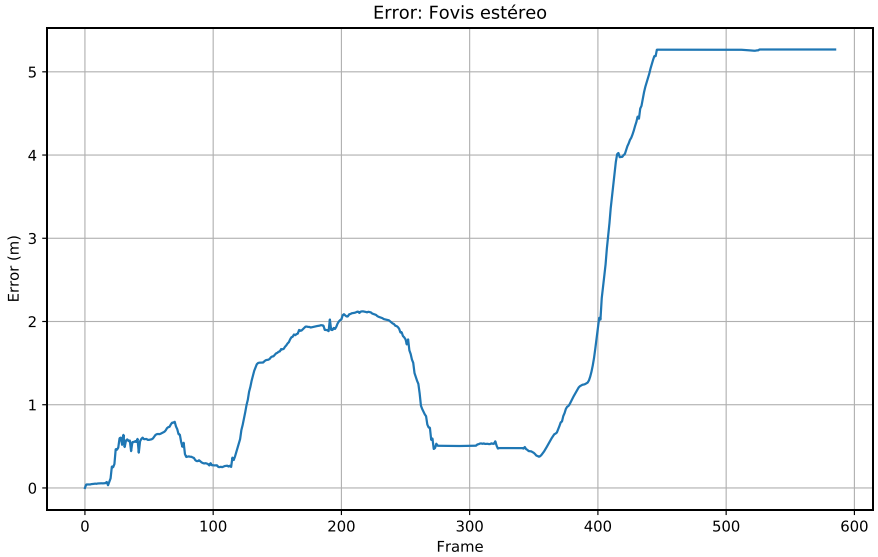

Figura 33. Algoritmo Fovis para un par estéreo en el entorno reducido de la Figura 31. Evolución del error de la posición a lo largo de los *frames* capturados.

4.2 OpenCV RGB-Depth Processing

RGB-Depth Processing [20] es un módulo de la librería OpenCV [21] para estimar la pose de una cámara a través de información RGB-D.

En este caso se ha utilizado uno de los *dataset* de la universidad de TUM [91], en concreto: *rgbd_dataset_freiburg2_pioneer_slam3* [92]. Posee imágenes RGB y de profundidad obtenidas durante un recorrido realizado por un robot Pioneer en un entorno controlado, de dimensiones aproximadas de 5x5 metros.

Se ha utilizado un dataset debido a que, en los experimentos anteriores con *Fovis* los resultados no han sido demasiado buenos. Esto podría deberse a que una simulación no es el método propicio para evaluar un algoritmo basado en el uso de cámaras, ya que las texturas simuladas no alcanzan el nivel de realismo requerido.

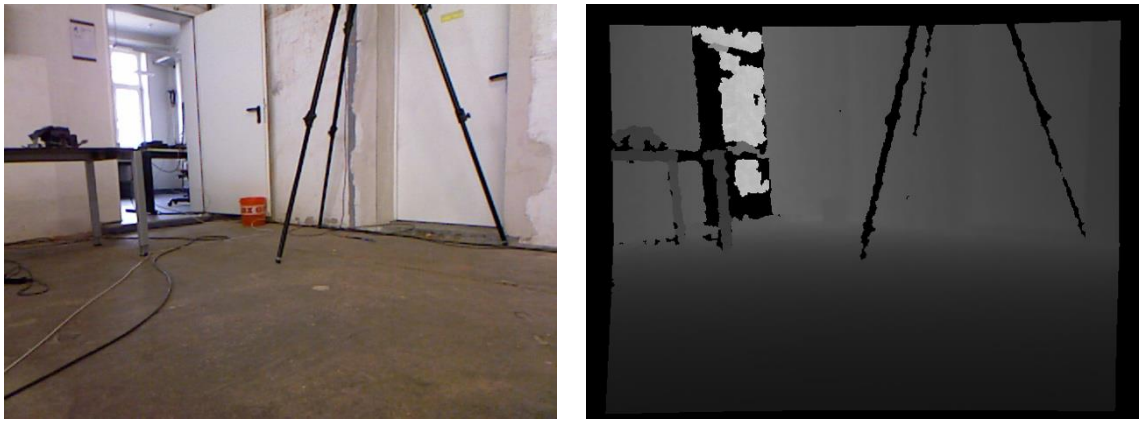

Figura 34. Imágenes del dataset *rdgd_dataset_freiburg2_pioneer_slam3*. Izquierda: RGB. Derecha: Depth

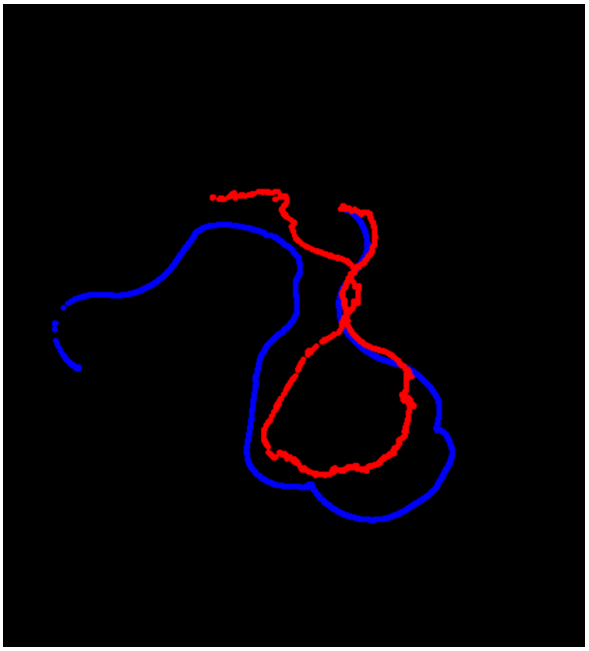

Figura 35. Resultado obtenido mediante OpenCV a partir de las imágenes del dataset. Trayectoria Azul: Real, proporcionado por el dataset. Trayectoria roja: Estimada mediante el paquete RGB-Depth Processing de OpenCV. Dimensiones aproximadas del recorrido: 5x5 metros.

Los resultados, al igual que en el experimento anterior, se expusieron dibujando la trayectoria estimada mediante sobre la real, proveída en este caso por el dataset. En la Figura 35 se ve el resultado final. Se puede apreciar que al comienzo del recorrido la trayectoria estimada se ajusta bastante a la real, pero, conforme va transcurriendo el tiempo, el error se

hace mayor. Esto es debido a que, al ser un algoritmo incremental, el error se va acumulando, al igual que ocurría en los algoritmos de Fovis.

Se puede concluir que, los algoritmos de odometría visual no resultan adecuados para la navegación por una industria de grandes dimensiones, sobre todo, en el proceso de inspección de las chapas de acero, ya que, durante la inspección el robot circulará sobre una chapa de grandes dimensiones, lo que implica la no presencia de objetos en un amplio espacio.

4.3 Detección de ArUcos: *aruco_detect*

Como ya se ha comentado con anterioridad, se ha empleado un paquete compatible con ROS *Kinetic* ya desarrollado, *aruco_detect* [46]. Está desarrollado en lenguaje C++ basado en el módulo Aruco de OpenCV [68].

Se han llevado a cabo diferentes experimentos para evaluar su funcionamiento según el tamaño de los marcadores y su distancia a la cámara. Para ello, se utilizó un entorno sencillo compuesto por una pared con los marcadores y el robot.

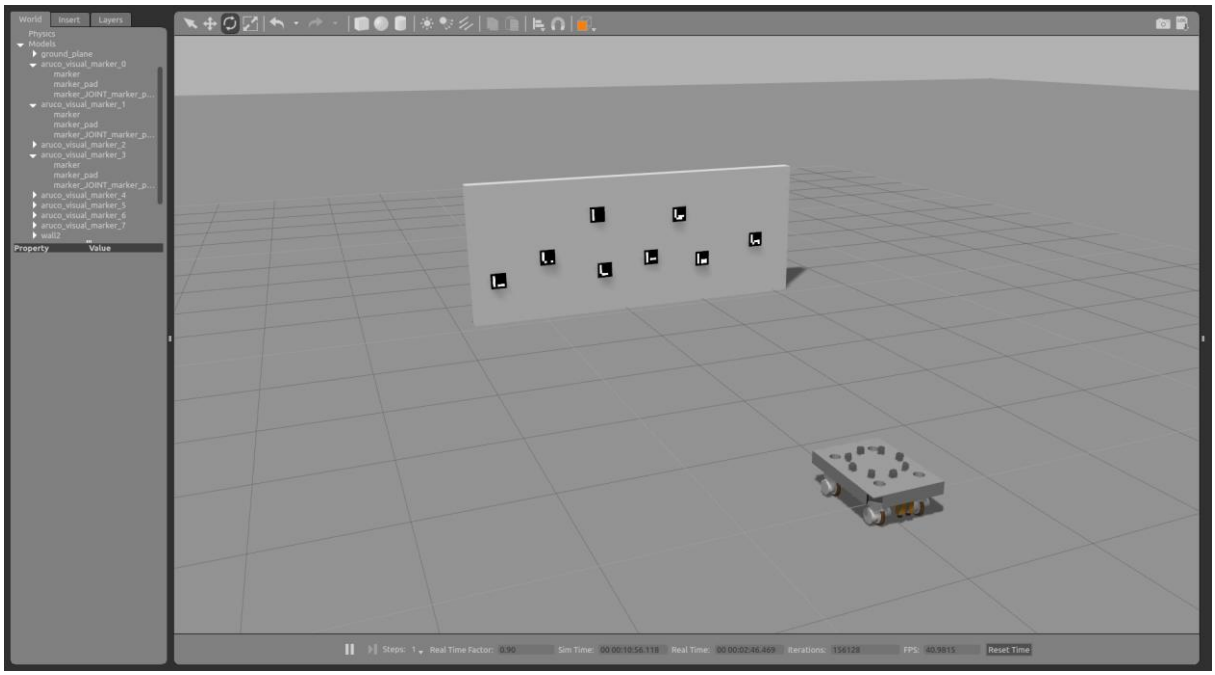

Figura 36. Simulación en Gazebo para evaluar el funcionamiento del paquete *aruco_detect*.

Se probaron tres tamaños de ArUco diferentes, 17.78, 25 y 50cm de lado. La primera medida se trata del tamaño asignado por defecto en el paquete. Además, se desplazó el robot según diferentes distancias de los marcadores.

Tabla 4. Número de marcadores detectados (de un total de 8) según su tamaño y distancia a la cámara

		Distancia de los marcadores a la cámara (m)				
		5	10	15	20	25
Tamaño del lado (cm)	17.78	8	8	6	3	0
	25	8	8	8	7	4
	50	8	8	8	8	7

Se puede comprobar que el tamaño de los marcadores y su distancia a la cámara son factores clave en cuanto a su detección. Cuando el marcador presenta un tamaño menor o se encuentra lejos de la cámara la tasa de detección disminuye. También se aprecian problemas originados a partir de la simulación de Gazebo, pues se producen brillos que dificultan la localización a medida que los marcadores se encuentras más alejados.

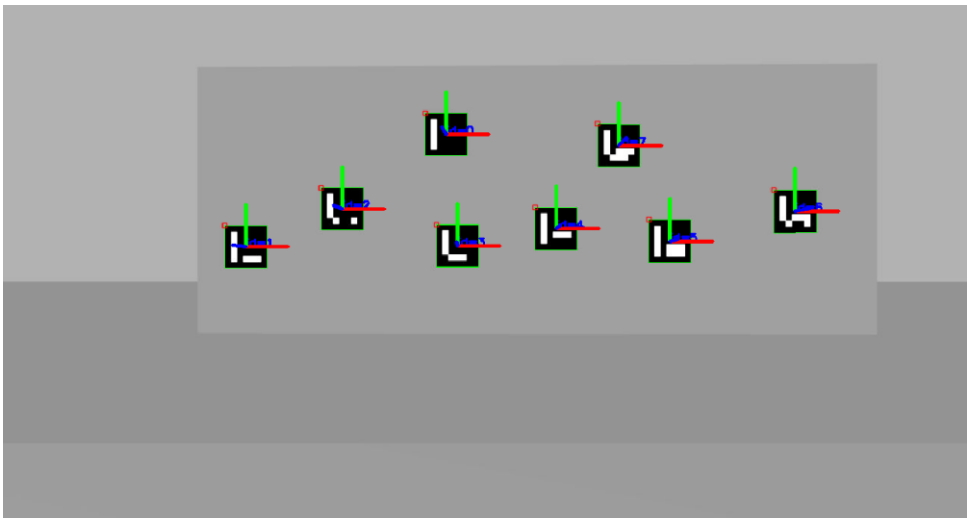

Figura 37. Imagen publicada por el paquete *aruco_detect*. Se marca la esquina superior izquierda en rojo y se recuadra en verde el marcador completo. También se escribe el identificador de cada marcador según el diccionario escogido y se representa el sistema de coordenadas de cada marcador. Tamaño: 25cm. Distancia: 5m.

4.4 Localización mediante marcadores

La localización del robot mediante los marcadores ArUco se ha implementado en un paquete para ROS desarrollado en C++. Fue necesario implementar este paquete, ya que no se pudo encontrar ninguna implementación ya desarrollada que coincidiese con los criterios buscados.

Cabe recordar que se ha desarrollado de manera que se subscriba a los *topics* que albergan la información publicada por el paquete *aruco_detect* ejecutándose sobre las imágenes de cada cámara.

En estos experimentos se emplearon diferentes simulaciones en Gazebo, variando el número de marcadores y su posición, tomando como base la industria mostrada en la Figura 38, de dimensiones 30x20 metros.

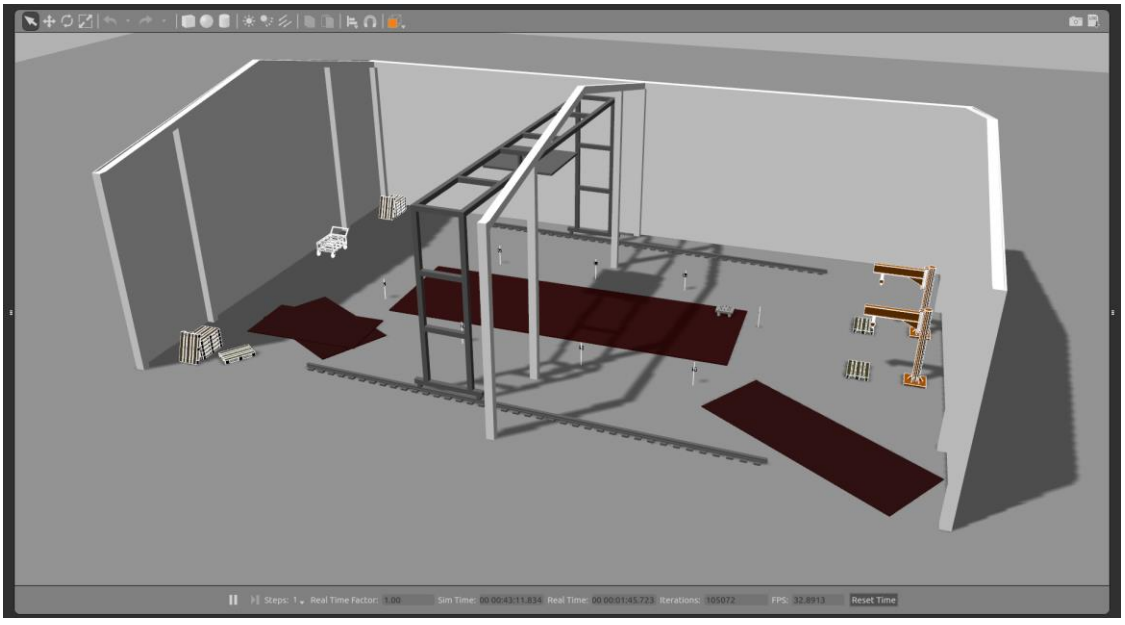

Figura 38. Simulación en el entorno Gazebo de la industria y el robot dispuesto sobre la chapa a inspeccionar. La fábrica es de 30x20 metros y la chapa de acero de 10x5 metros.

En todos los experimentos se ha seguido la misma ruta con el fin de poder contrastar los resultados entre sí de una manera más fiable. El robot se mueve únicamente dentro de la chapa, de dimensiones 10x5m, según el camino mostrado en la Figura 39. Para realizar la misma ruta en cada experimento, previamente se guardaron en un *bag* los comandos de velocidad dados al robot mediante el teclado y, posteriormente, solo era necesario reproducir

el *bag* guardado. Se pueden apreciar pequeñas diferencias en cada ruta, esto es debido a las limitaciones que presenta la grabación y reproducción de archivos *bag*. El camino seguido por el robot es muy sensible a pequeños cambios de tiempo en el sistema y la herramienta *rosbag* está limitada en su capacidad para duplicar exactamente el comportamiento de un sistema en ejecución cuando los mensajes son grabados y procesados y, posteriormente, cuando se producen los mensajes cuando se utiliza la reproducción.

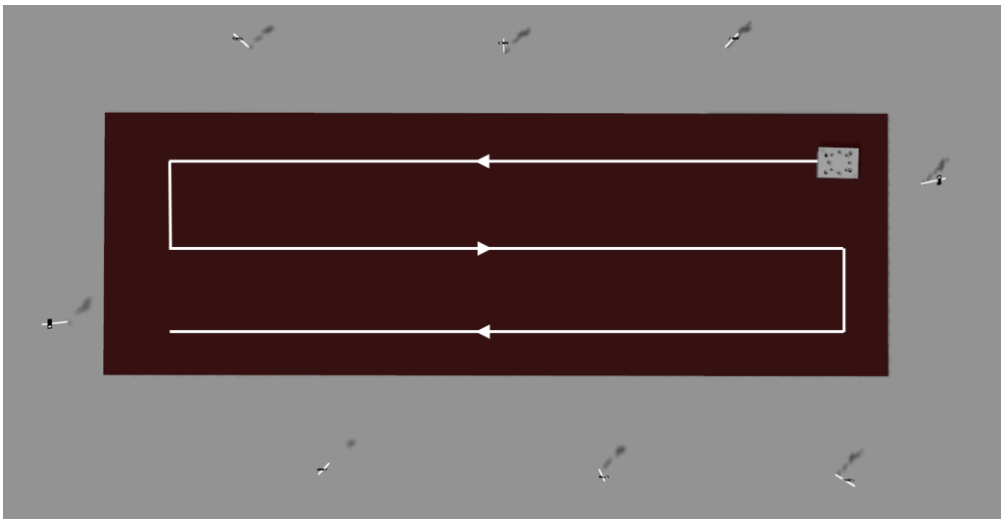

Figura 39. Ruta seguida por el robot durante las pruebas realizadas. Dimensiones de la chapa: 10x5 metros.

Las simulaciones comenzaron con 4 ArUcos y fueron creciendo de dos en dos hasta llegar a 8. El tamaño del lado de los ArUcos por defecto para el paquete de detección es de 17.78cm, por ese motivo fue el utilizado en un inicio. Posteriormente, se realizaron pruebas con marcadores de lado de 50cm.

Los resultados obtenidos durante la simulación se almacenan en un archivo .csv para permitir su posterior análisis. Se guarda la información relativa a la pose del robot respecto del sistema de coordenadas del mundo: posición (X,Y,Z) y orientación (roll, pitch, yaw), tanto real, obtenida directamente de Gazebo, y estimada. También se almacenan los errores medidos respecto a la pose real y determinados datos que permiten ayudar a analizar los resultados, como la estrategia utilizada (monocular o estéreo).

En particular, las componentes de la pose más importantes son, las posiciones en los ejes X e Y y el ángulo Yaw, giro respecto del eje Z, ya que el robot siempre se va a mover sobre el plano X-Y, ver Figura 40.

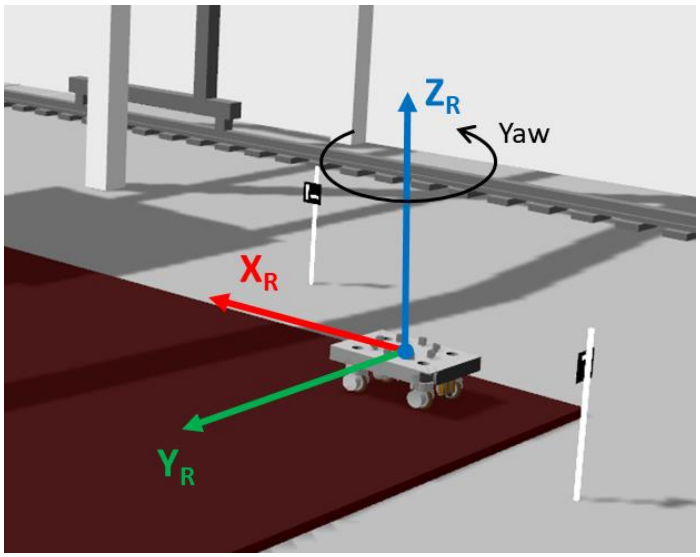

Figura 40. Yaw: giro respecto del eje Z.

A continuación, se exponen los resultados obtenidos durante distintas simulaciones, con 4, 6 y 8 marcadores de 17.78 cm de lado, colocados a dos metros de los bordes de la chapa. Los resultados se exponen de diversas maneras. Se representó de manera gráfica la trayectoria 2D seguida sobre la chapa. Además, se realizaron histogramas del error de la posición con el fin de ver qué valores de error son los más habituales. Estos histogramas están divididos en intervalos de 5cm. Además, Se analizaron las dos estrategias implementadas. En las imágenes de la izquierda se empleó la estrategia estéreo y, cuando no era posible aplicarla por falta de marcadores comunes, se completó con la estrategia monocular. En la derecha, únicamente la estrategia monocular.

En la Figura 41 se exponen los resultados obtenidos al aplicar el método de localización utilizando 4 marcadores de 17.78cm. Se puede apreciar que los resultados obtenidos son notablemente mejores al utilizar únicamente esta segunda estrategia. La mayor parte de estimaciones se concentran en un error en torno a los 10cm, a pesar de tener una estimación con un error mayor a 5m. Sin embargo, al tratarse de errores puntuales, se podrían eliminarán en un futuro. En cuanto a la estrategia combinada, los errores se concentran en torno a los 20cm.

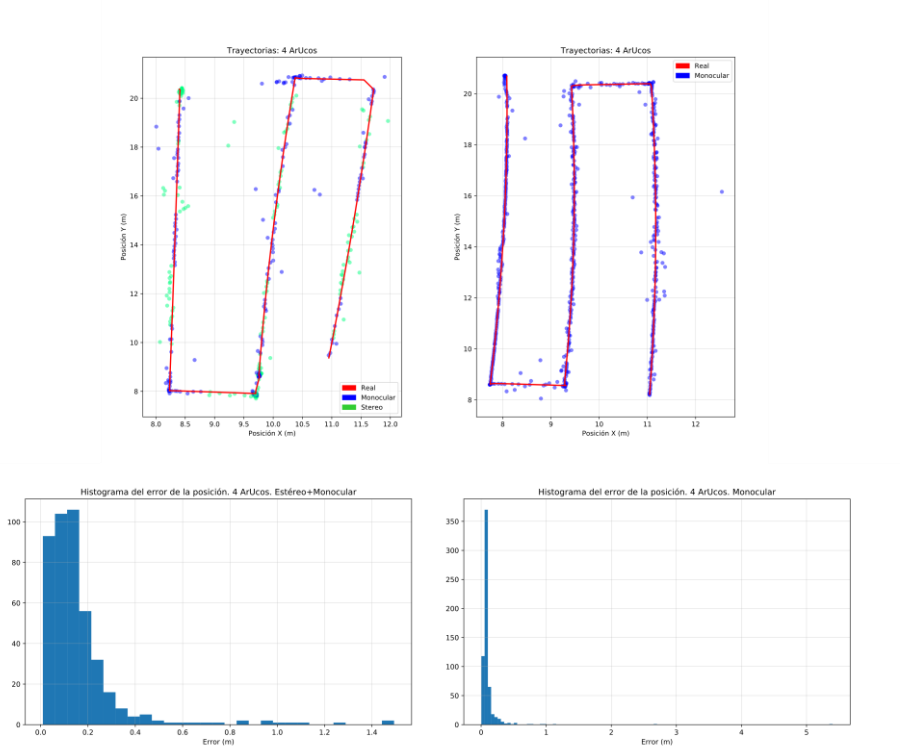

Figura 41. **Arriba:** Trayectoria real (rojo) y estimada con 4 ArUcos de 17.78cm. Izquierda: En azul las posiciones estimadas según la estrategia estéreo y en verde, estrategia monocular. Derecha: Solo estrategia monocular. Los marcadores se encuentran aproximadamente a 2 metros de los bordes de la chapa. **Abajo**: Histogramas del error absoluto de la posición, en intervalos de 5cm.

Los resultados obtenidos con 6 ArUcos se muestran en la Figura 42 y con 8 ArUcos en la Figura 43. Analizándolos, se puede comprobar que las conclusiones extraídas del experimento anterior se siguen cumpliendo. La estrategia monocular estima la posición del robot con un menor error. Agrupándose la mayor parte de las estimaciones en un error de posición menor a 10cm.

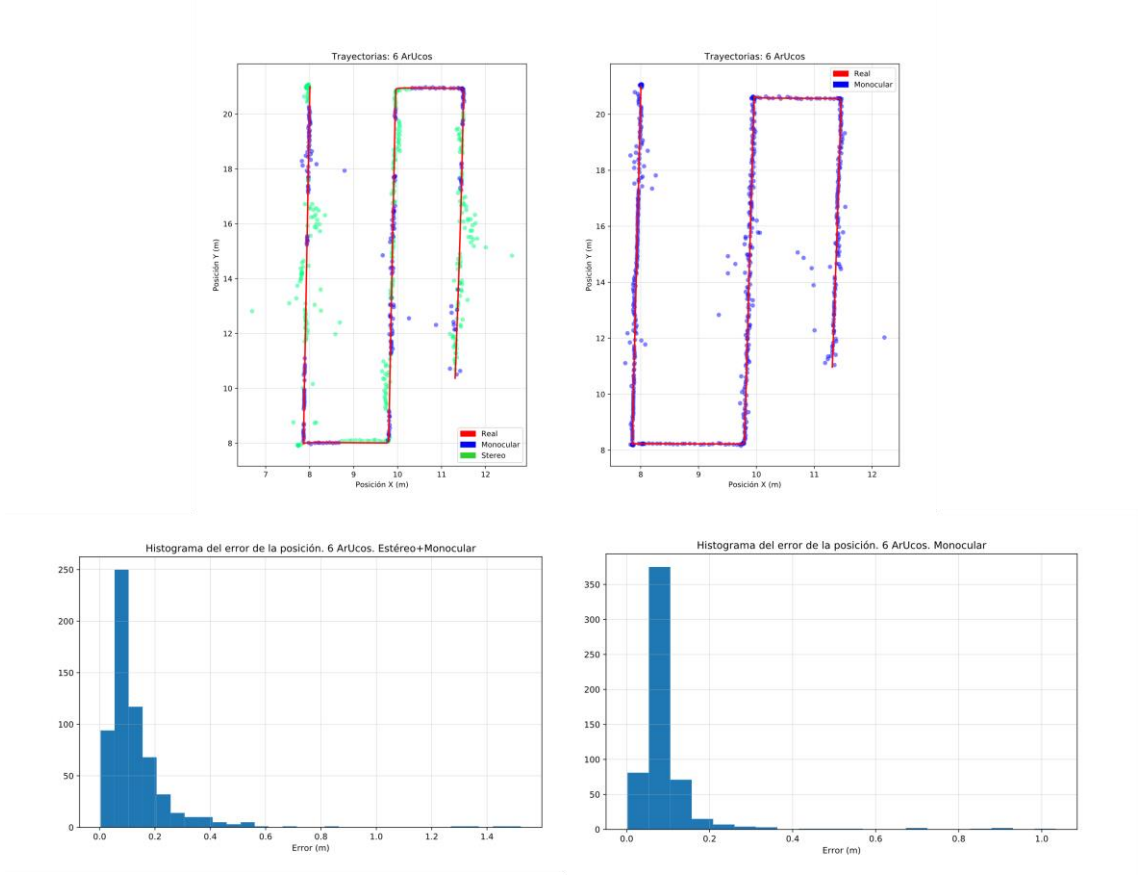

Figura 42. **Arriba:** Trayectoria real (rojo) y estimada con 6 ArUcos de 17.78cm. Izquierda: En azul las posiciones estimadas según la estrategia estéreo y en verde, estrategia monocular. Derecha: Solo estrategia monocular. Los marcadores se encuentran aproximadamente a 2 metros de los bordes de la chapa. **Abajo**: Histogramas del error absoluto de la posición.

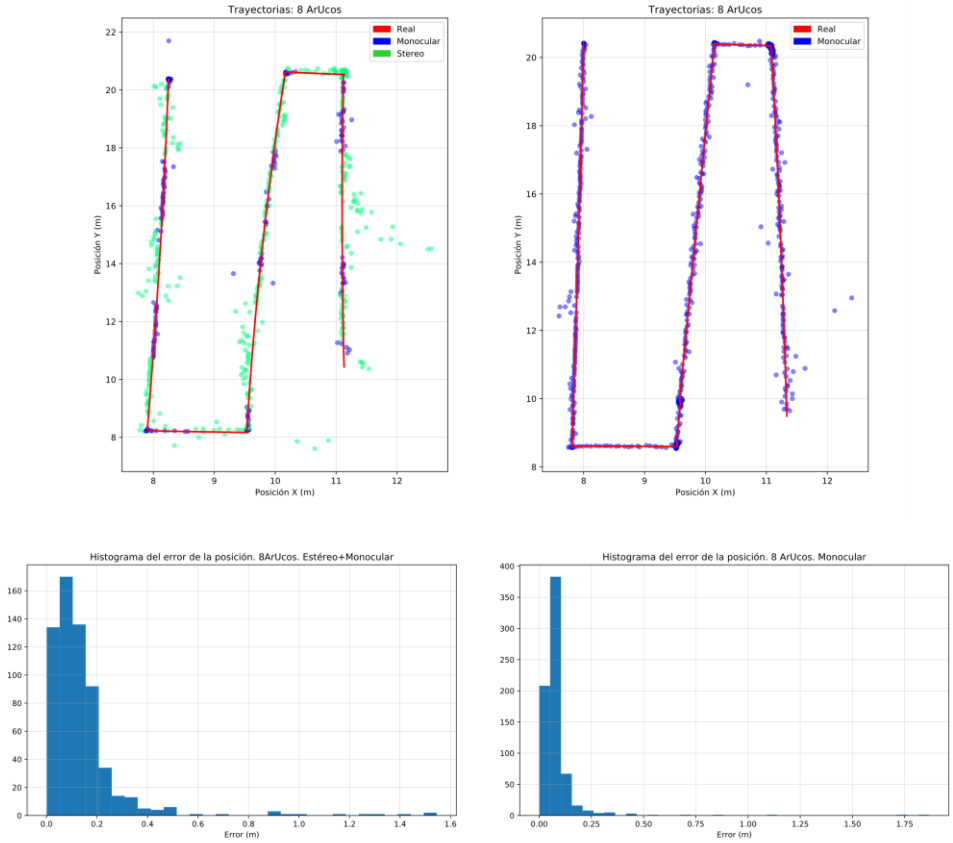

Figura 43. **Arriba**: Trayectoria real (rojo) y estimada con 8 ArUcos de 17.78cm. Izquierda: En azul las posiciones estimadas según la estrategia estéreo y en verde, estrategia monocular. Derecha: Solo estrategia monocular. Los marcadores se encuentran aproximadamente a 2 metros de los bordes de la chapa. **Abajo**: Histogramas del error absoluto de la posición.

Tabla 5. Comparativa de errores medios según número de marcadores

	4 ArUcos	**6 ArUcos**	**8 ArUcos**
Error medio de la posición (cm)	10.98	9.53	8.31
Error máximo (cm)	500.40	103.39	186.73
Error mínimo (cm)	3.94	2.056	0.59

Se puede comprobar, según la Tabla 5, que a medida que aumenta el número de marcadores el error de posición disminuye. La media no es especialmente representativa debido a estimaciones erróneas aisladas, pero puede proporcionar una idea, junto a los histogramas, de la fiabilidad de las medidas.

Además, se realizan pruebas con 8 ArUcos de 50cm de lado, tanto llevando la misma trayectoria sobre la chapa que los experimentos anteriores como realizando una ruta aleatoria por la fábrica.

En el primer caso, Figura 44, los marcadores se alejaron respecto a las pruebas anteriores. Se dispusieron, aproximadamente, a 7 metros de los bordes de la chapa. En este caso, se puede ver que las estimaciones suelen tener un error menor a 10cm, siendo 7.99cm el error medio.

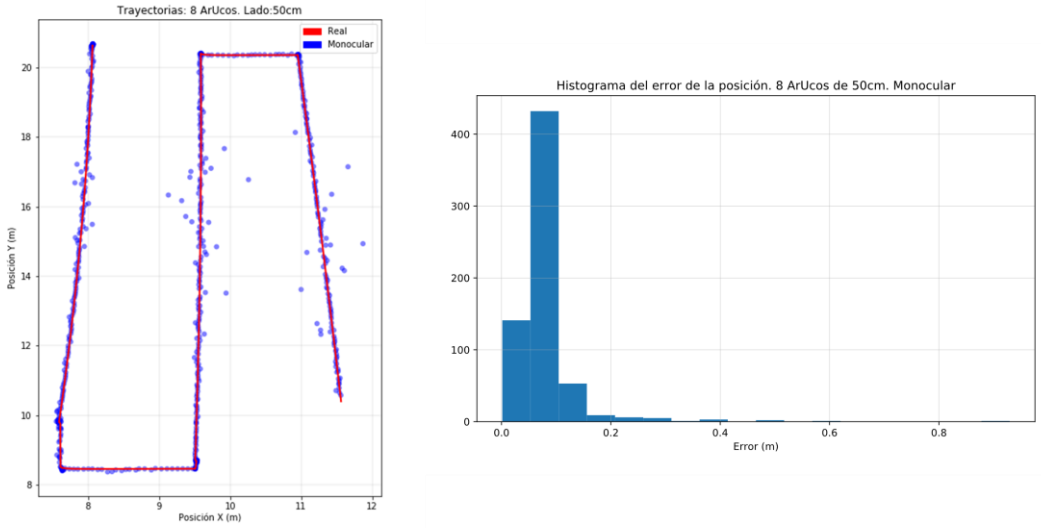

Figura 44. Izquierda: Trayectoria real (rojo) y estimada (azul) con 8 ArUcos de 50cm según la estrategia monocular. Derecha: Histograma del error de la posición. Los marcadores se encuentran aproximadamente a 7 metros de los bordes de la chapa.

Durante los experimentos anteriores, al tratarse de un robot omnidireccional y realizar una trayectoria tan recta, prácticamente no realizaba ningún giro. Debido a este motivo, se realizó una trayectoria aleatoria por la fábrica realizando diferentes giros. Los resultados pueden verse en la Figura 45 y Figura 46. En esta simulación, los marcadores se encontraban

en las paredes del recinto. Cabe recordar que las dimensiones de la industria son de 30x20 metros.

En cuanto a la posición, se puede ver que, al igual que en los experimentos anteriores, la trayectoria estimada se ajusta bastante a la real. En este caso, el error medio de posición es de 11.61cm, mayor que cuando circulaba únicamente por la chapa. Sin embargo, el área por el que se mueve ahora el robot es bastante mayor y pretende ser cubierto por el mismo número de ArUcos.

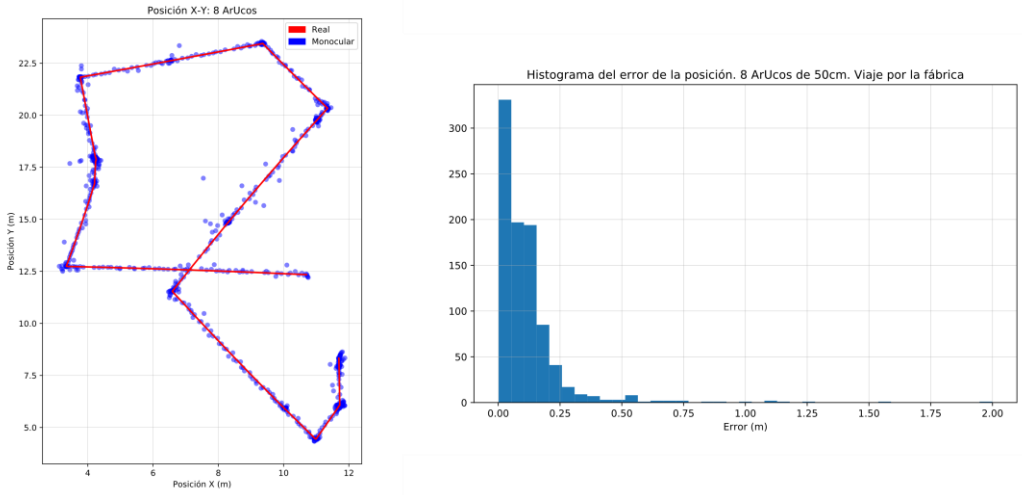

Figura 45. El robot se mueve por toda la industria de 30x20 metros. Los 8 marcadores están localizados en las paredes de la fábrica. Izquierda: Trayectoria real (rojo) y estimada (azul) con 8 marcadores de 50cm. Derecha: Histograma del error de posición.

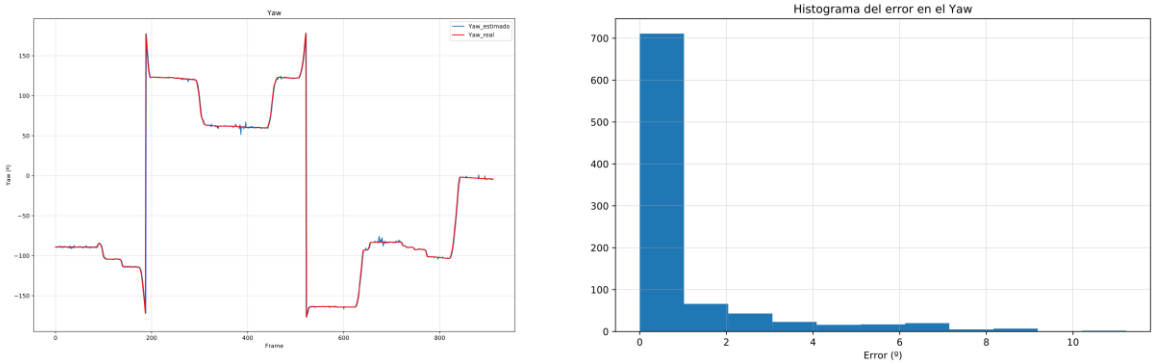

Figura 46. El robot se mueve por toda la industria de 30x20 metros. Los 8 marcadores están localizados en las paredes de la fábrica. Izquierda: Orientación real (rojo) y estimado (azul) con 8 marcadores de 50cm. Derecha: Histograma del error en la estimación de la orientación con intervalos de 1º.

La orientación del robot también ha sido estimada de una manera bastante fiable, ver Figura 46. La mayor parte de las estimaciones tienen un error menor a 1º, con errores máximos de hasta 11º.

La evolución del error a lo largo del tiempo se puede ver en la siguiente figura, comprobando que, al contrario que en los algoritmos basados en odometría, el error no se acumula a lo largo del tiempo, sino que es independiente del resto de estimaciones.

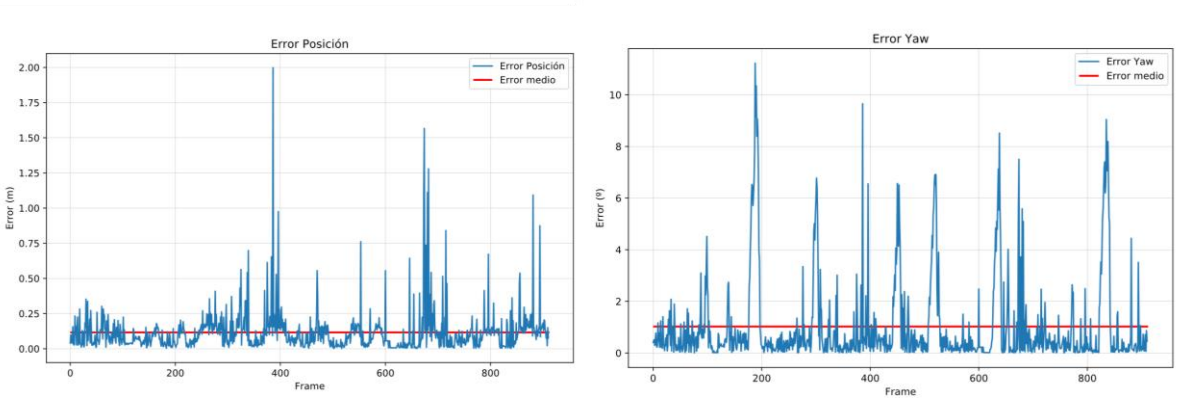

Figura 47. Evolución del error a lo largo del tiempo. Izquierda: Error posición. Derecha: Error Yaw.

5. DISCUSIÓN

Se ha realizado un estudio sobre las diferentes alternativas para localizar un robot móvil en un entorno cerrado. Además, se llevaron a cabo pruebas con métodos de odometría visual ya desarrollados y se implementó un método de localización mediante marcadores dispuestos en posiciones conocidas.

A la vista de los resultados, se descartó el uso de técnicas de odometría visual. Además, no parece la mejor opción para un entorno de grandes dimensiones que no presenta una gran cantidad de objetos. Por ello, se profundiza más en la técnica de localización mediante marcadores. Los resultados obtenidos mediante esta estrategia son bastante buenos, sin embargo, en ocasiones se producen estimaciones con un error bastante grande que se alejan de la tónica general. Este hecho podría contrarrestarse al fusionar dichas estimaciones con información procedente de otras fuentes, como las órdenes de movimiento dadas al robot o un IMU.

El método de localización mediante marcadores se ha desarrollado en forma de paquete para ROS. Este paquete recoge la información arrojada por un paquete ya desarrollado de detección de marcadores ArUco para estimar la pose del robot. Se realizaron dos estrategias de localización, pudiendo comprobar que el método monocular arroja mejores resultados.

La cantidad y tamaño de los ArUcos es especialmente importante en cuanto a la estimación. Un mayor número de marcadores favorece la localización en todo momento. Además, es necesario escoger el tamaño del ArUco según la lejanía de los marcadores a la cámara.

Al ser el método monocular más robusto que el estéreo, se podría intentar rebajar el número de cámaras, favoreciendo el tiempo de ejecución. Ya que se utilizaron 8 para favorecer la aparición de un mismo marcador en dos cámaras contiguas.

En un futuro, se estudiará si la localización mediante balizas se podría fusionar con información procedente de otras fuentes, como una unidad de medición inercial (IMU) o las órdenes de movimiento procedentes del sistema de navegación. Por ejemplo, mediante un filtro de Kalman extendido, que permite la fusión de información de diferentes fuentes con el fin de estimar la pose del robot con un menor error.

Más adelante, se estudiará la mejor manera de proporcionar una medida de la fiabilidad de la pose estimada. Además, conocer esta medida resulta especialmente interesante de cara a la fusión de información.

Además, se han realizado diferentes pruebas con el fin de encontrar el tamaño y número de marcadores a disponer en el entorno para que siempre se pueda localizar alguno. En el entorno desarrollado, de dimensiones de 30x20 metros, con 4 marcadores se han recabado buenos resultados.

También se podrían explorar otras alternativas en cuanto a la disposición de las cámaras, como utilizar un menor número dispuestas en *pan&tilts* que, a través de su movimiento, permitan la búsqueda de los marcadores.

Además, las pruebas se han realizado mediante una simulación en el entorno de Gazebo, sin embardo, la idea es trasladar la propuesta de localización al robot real. Para ello, la construcción del mapa de marcadores deberá realizarse de manera manual, así como realizar una calibración de las cámaras a utilizar.

También quedaría la elección de las cámaras, por lo que habría que estudiar cuáles serían más convenientes para este problema según sus parámetros.

Se podría estudiar la inclusión de cámaras en las columnas del entorno que garanticen una vista completa de las chapas de acero a inspeccionar y, por lo tanto, del robot moviéndose sobre las mismas. Con esta configuración, sería posible tener al vehículo localizado en todo momento si se le incluye algún elemento diferenciador, por ejemplo, una baliza luminosa. Este hecho permitiría completar la información de localización arrojada por los paquetes desarrollados anteriormente, con el fin de obtener datos más fiables.

Otra vía de estudio de cara a hacer más robusta la localización sería, el aprendizaje de determinados objetos que estuviesen siempre fijos en el entorno con el fin de tratarlos como un marcador extra. Es decir, el algoritmo sería capaz de encontrar objetos o puntos característicos que se encontrasen siempre en el mismo lugar, con ayuda de los ArUco dispuestos en el entorno, y tratarlos como un marcador más.

6. CONCLUSIONES

La localización de un vehículo móvil no es tarea fácil, en la literatura se han estudiado diferentes métodos con la intención de resolver esta problemática, pero ninguno funciona a la perfección. Por lo tanto, se puede concluir que los objetivos marcados al comienzo del proyecto se han cumplido con éxito, estudiando diferentes alternativas de localización, profundizando en alguno de los métodos y llevando a cabo experimentos empleando técnicas de odometría visual. Además, se ha implementado un método de detección con marcadores que permite la localización del robot sobre la chapa de acero en todo momento.

Por lo tanto, este proyecto ha servido para, además del desarrollo de un método de localización, dar una visión general de las múltiples alternativas existentes para localizar robots móviles mediante el uso de cámaras.

Se ha implementado un paquete para ROS que utiliza un método de localización a partir de una serie de marcadores situados en posiciones conocidas del entorno. Para ello, emplea un arco de cámaras calibradas y se coordina con un paquete de detección de ArUcos. Como se resumía en la figura inicial, recuperada a modo de resumen. Se han probado dos estrategias diferentes dentro de este método, una estrategia monocular y una estéreo.

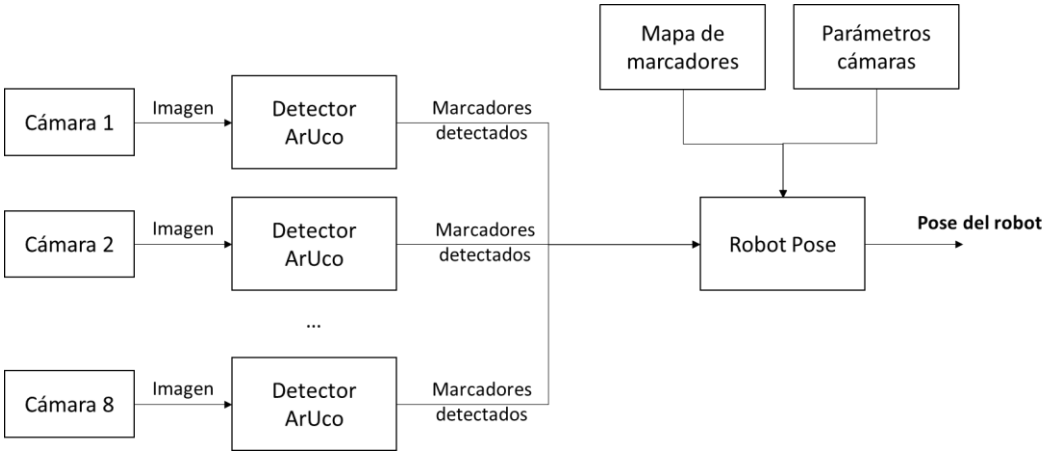

Figura 48. Esquema del método implementado. *Robot Pose* hace referencia al paquete implementado.

También cabe recordar que se han realizado programas con el fin de evaluar dos librerías de odometría diferentes, *fovis* y *RGB-Depth Processing*. Las alternativas de odometría visual

proporcionan peores resultados, por lo que se concluye que, para localizar un robot circulando sobre una chapa de acero en un entorno industrial de grandes dimensiones, resulta más eficaz la estrategia de utilizar marcadores en posiciones conocidas.

Es preciso resaltar los conocimientos adquiridos por el máster que resultaron útiles para llevar a cabo las tareas asignadas. Concretamente, de suma importancia han sido las asignaturas de Visión 3D y las de programación en C++, destacando la programación orientada a objetos, ya que se encuentran estrechamente ligadas a las actividades desarrolladas. Además, se puede destacar la programación en *Python*, utilizando librerías especializadas como *pandas*, para el análisis de datos.

Por último, cabe destacar el aprendizaje adquirido durante el desarrollo de este proyecto. Primeramente, la inmersión en el mundo de la localización de vehículos autónomos, estudiando las diferentes alternativas expuestas en la literatura científica. Se han probado diferentes métodos, tanto basados en odometría utilizando cámaras RGBD o estéreo, como métodos de localización absolutos mediante balizas dispuestas en posiciones conocidas.

En cuanto al software, cabe destacar el manejo del *framework* ROS, resaltando el aprendizaje en cuanto al desarrollo de nuevos paquetes y su coordinación con paquetes ya existentes. Además del desarrollo de diferentes simulaciones en el entorno Gazebo, utilizando diferentes modelos de sensores.

7. REFERENCIAS

[1]T. Söderström, "Using an Extended Kalman Filter for Rigid Body Pose Estimation," *J. Biomech. Eng.*, vol. 127, no. 3, p. 475, Dec. 2004.

[2]F. Caron, M. Davy, E. Duflos, and P. Vanheeghe, "Particle Filtering for Multisensor Data Fusion With Switching Observation Models: Application to Land Vehicle Positioning," *IEEE Trans. Signal Process.*, vol. 55, no. 6, pp. 2703–2719, Jun. 2007.

[3]M. Alatise and G. Hancke, "Pose Estimation of a Mobile Robot Based on Fusion of IMU Data and Vision Data Using an Extended Kalman Filter," *Sensors*, vol. 17, no. 10, p. 2164, Sep. 2017.

[4]B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle Adjustment — A Modern Synthesis," in *Vision Algorithms: Theory and Practice*, vol. 1883, B. Triggs, A. Zisserman, and R. Szeliski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 298–372.

[5]K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics," *Intell. Ind. Syst.*, vol. 1, no. 4, pp. 289–311, Dec. 2015.

[6]H. P. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover.," STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1980.

[7]D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, 2004, vol. 1, pp. I–I.

[8]Y. Cheng, M. Maimone, and L. Matthies, "Visual odometry on the Mars exploration rovers," in *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, 2005, vol. 1, pp. 903–910.

[9]"Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," p. 42.

[10] D. Scaramuzza and R. Siegwart, “Monocular omnidirectional visual odometry for outdoor ground vehicles,” in *International Conference on Computer Vision Systems*, 2008, pp. 206–215.

[11] A. I. Comport, E. Malis, and P. Rives, “Accurate Quadrifocal Tracking for Robust 3D Visual Odometry,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, Rome, Italy, 2007, pp. 40–45.

[12] “Kinect: desarrollo de aplicaciones de Windows.” [Online]. Available: https://developer.microsoft.com/es-es/windows/kinect. [Accessed: 17-Jan-2019].

[13] S. Wirth, P. L. Negre Carrasco, and G. O. Codina, “Visual odometry for autonomous underwater vehicles,” in *2013 MTS/IEEE OCEANS - Bergen*, Bergen, 2013, pp. 1–6.

[14] A. S. Huang *et al.*, “Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera,” in *Robotics Research*, vol. 100, H. I. Christensen and O. Khatib, Eds. Cham: Springer International Publishing, 2017, pp. 235–252.

[15] *Dense Visual Odometry. Contribute to tum-vision/dvo development by creating an account on GitHub*. TUM Computer Vision Group, 2019.

[16] C. Kerl, J. Sturm, and D. Cremers, “Robust odometry estimation for RGB-D cameras,” in *2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, 2013, pp. 3748–3754.

[17] *A ROS wrapper for libviso2, a library for visual odometry: srv/viso2*. Systems, Robotics & Vision, University of the Balearic Islands, 2019.

[18] “Systems, Robotics & Vision Group | Universitat de les Illes Balears.” .

[19] “fovis_ros - ROS Wiki.” [Online]. Available: http://wiki.ros.org/fovis_ros?distro=hydro. [Accessed: 08-Jan-2019].

[20] “OpenCV: RGB-Depth Processing.” [Online]. Available: https://docs.opencv.org/3.4/d2/d3a/group__rgbd.html. [Accessed: 05-Oct-2018].

[21] “OpenCV: OpenCV modules.” [Online]. Available: https://docs.opencv.org/3.4/index.html. [Accessed: 05-Oct-2018].

[22] F. Steinbrucker, J. Sturm, and D. Cremers, “Real-time visual odometry from dense RGB-D images,” in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, Barcelona, Spain, 2011, pp. 719–722.

[23] L. Kleeman, “Advanced sonar and odometry error modeling for simultaneous localisation and map building,” in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, Las Vegas, Nevada, USA, 2003, vol. 1, pp. 699–704.

[24] F. Abrate, B. Bona, and M. Indri, “Experimental EKF-based SLAM for mini-rovers with IR sensors only,” p. 6.

[25] D. M. Cole and P. M. Newman, “Using laser range data for 3D SLAM in outdoor environments,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, Orlando, FL, USA, 2006, pp. 1556–1563.

[26] A. J. Davison, “SLAM with a Single Camera.”

[27] G. A. Terejanu, “Extended Kalman Filter Tutorial,” p. 7.

[28] G. Klein and D. Murray, “Parallel Tracking and Mapping for Small AR Workspaces,” in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, Nara, Japan, 2007, pp. 1–10.

[29] “New package: (SVO) Semi-direct Monocular Visual Odometry - ROS robotics news.” [Online]. Available: http://www.ros.org/news/2014/06/new-package-svo-semi-direct-monocular-visual-odometry.html. [Accessed: 31-Jan-2019].

[30] C. Forster, M. Pizzoli, and D. Scaramuzza, “SVO: Fast semi-direct monocular visual odometry,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, 2014, pp. 15–22.

[31] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, “ORB-SLAM: A Versatile and Accurate Monocular SLAM System,” *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[32] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison, “SLAM++: Simultaneous Localisation and Mapping at the Level of Objects,” in

2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, 2013, pp. 1352–1359.

[33] K. Tateno, F. Tombari, and N. Navab, "When 2.5D is not enough: Simultaneous reconstruction, segmentation and recognition on dense SLAM," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 2295–2302.

[34] C. D. McGillem and T. S. Rappaport, "Infra-red location system for navigation of autonomous vehicles," in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, USA, 1988, pp. 1236–1238.

[35] N. Ko and T.-Y. Kuc, "Fusing Range Measurements from Ultrasonic Beacons and a Laser Range Finder for Localization of a Mobile Robot," *Sensors*, vol. 15, no. 5, pp. 11050–11075, May 2015.

[36] J. Biswas and M. Veloso, "WiFi localization and navigation for autonomous indoor mobile robots," in *2010 IEEE International Conference on Robotics and Automation*, Anchorage, AK, 2010, pp. 4379–4384.

[37] "¿Cómo se realiza un Crash Test?" [Online]. Available: https://www.quadis.es/articulos/-como-se-realiza-un-crash-test-/131649. [Accessed: 03-Feb-2019].

[38] H. Kato and M. Billinghurst, "Marker tracking and HMD calibration for a video-based augmented reality conferencing system," in *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, San Francisco, CA, USA, 1999, pp. 85–94.

[39] M. Fiala, "Designing Highly Reliable Fiducial Markers," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 7, pp. 1317–1324, Jul. 2010.

[40] A. Babinec, L. Jurišica, P. Hubinský, and F. Duchoň, "Visual Localization of Mobile Robot Using Artificial Markers," *Procedia Eng.*, vol. 96, pp. 1–9, 2014.

[41] M. Fiala, "ARTag, a Fiducial Marker System Using Digital Techniques," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA, 2005, vol. 2, pp. 590–596.

[42] D. Wagner and D. Schmalstieg, “ARToolKitPlus for Pose Tracking on Mobile Devices,” p. 9.

[43] D. Flohr and J. Fischer, “A Lightweight ID-Based Extension for Marker Tracking Systems,” p. 7.

[44] L. Naimark and E. Foxlin, “Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker,” in *Proceedings. International Symposium on Mixed and Augmented Reality*, Darmstadt, Germany, 2002, pp. 27–36.

[45] “reacTIVision.” [Online]. Available: http://reactivision.sourceforge.net/#usage. [Accessed: 01-Feb-2019].

[46] “aruco_detect - ROS Wiki.” [Online]. Available: http://wiki.ros.org/aruco_detect. [Accessed: 27-Nov-2018].

[47] M. Sonka, R. Boyle, and V. Hlavac, *Image Processing, Analysis, and Machine Vision*, 4th ed. .

[48] Rafael C.Gonzalez, Jose A.Cancelas, Ignacio Alvarez, Jose M.Enguita, “Geometria Proyectiva para Visión 3D.pdf.” .

[49] J. Heikkila and O. Silven, “A four-step camera calibration procedure with implicit image correction,” in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Juan, Puerto Rico, 1997, pp. 1106–1112.

[50] “OpenCV: Calibration with ArUco and ChArUco.” [Online]. Available: https://docs.opencv.org/3.4.3/da/d13/tutorial_aruco_calibration.html. [Accessed: 03-Feb-2019].

[51] G. H. Seedahmed and T. Schenk, “DIRECT LINEAR TRANSFORMATION IN THE CONTEXT OF DIFFERENT SCALING CRITERIA,” *Unpublished*, 2001.

[52] A. Ranganathan, “The Levenberg-Marquardt Algorithm,” p. 5.

[53] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng, “Complete solution classification for the perspective-three-point problem,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 8, pp. 930–943, Aug. 2003.

[54] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An Accurate O(n) Solution to the PnP Problem," *Int. J. Comput. Vis.*, vol. 81, no. 2, pp. 155–166, Feb. 2009.

[55] Richard I. Hartley, PeterSturm, "Triangulation." [Online]. Available: https://users.cecs.anu.edu.au/~hartley/Papers/triangulation/triangulation.pdf. [Accessed: 12-Feb-2019].

[56] "Descomposición en Valores Singulares (SVD)," 2010.

[57] "ArUco: a minimal library for Augmented Reality applications based on OpenCV | Aplicaciones de la Visión Artificial." [Online]. Available: https://www.uco.es/investiga/grupos/ava/node/26. [Accessed: 27-Nov-2018].

[58] Suzuki, Satoshi & be, KeiichiA., "Topological structural analysis of digitized binary images by border following," *Comput. Vis. Graph. Image Process.*, 1985.

[59] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognit.*, vol. 47, no. 6, pp. 2280–2292, Jun. 2014.

[60] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Trans. Syst. Man Cybern.*, vol. 9, no. 1, pp. 62–66, Jan. 1979.

[61] "libfovis: FOVIS." [Online]. Available: https://fovis.github.io/. [Accessed: 09-Oct-2018].

[62] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," p. 9, 1984.

[63] "FAST Algorithm for Corner Detection — OpenCV 3.0.0-dev documentation." [Online]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html. [Accessed: 09-Oct-2018].

[64] Howard, Andrew, "Real-Time Stereo Visual Odometry for Autonomous Ground Vehicles," 2008, pp. 3946–3952.

[65] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," *J. Opt. Soc. Am. A*, vol. 4, no. 4, p. 629, Apr. 1987.

[66] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. West Nyack: Cambridge University Press, 2004.

[67] E. Malis, "Improving vision-based control using efficient second-order minimization techniques," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, New Orleans, LA, USA, 2004, pp. 1843-1848 Vol.2.

[68] *Repository for OpenCV's extra modules. Contribute to opencv/opencv_contrib development by creating an account on GitHub*. OpenCV, 2019.

[69] V. Angladon *et al.*, "An evaluation of real-time RGB-D visual odometry algorithms on mobile devices," *J. Real-Time Image Process.*, Feb. 2017.

[70] "ROS.org | Powering the world's robots." .

[71] "STAIR." [Online]. Available: http://stair.stanford.edu/. [Accessed: 19-Dec-2018].

[72] "Willow Garage." [Online]. Available: http://www.willowgarage.com/. [Accessed: 10-Feb-2019].

[73] "The 3-Clause BSD License | Open Source Initiative." [Online]. Available: https://opensource.org/licenses/BSD-3-Clause. [Accessed: 19-Dec-2018].

[74] "rosbag - ROS Wiki." [Online]. Available: http://wiki.ros.org/rosbag. [Accessed: 03-Feb-2019].

[75] "Gazebo." [Online]. Available: http://gazebosim.org/. [Accessed: 10-Dec-2018].

[76] "Holonomic (robotics)," *Wikipedia*. 19-Dec-2018.

[77] "MPO-700 - Neobotix." [Online]. Available: https://www.neobotix-robots.com/omnidirectional-robot-mpo-700.html. [Accessed: 02-Feb-2019].

[78] "Gazebo : Tutorial : Gazebo plugins in ROS." [Online]. Available: http://gazebosim.org/tutorials?tut=ros_gzplugins. [Accessed: 12-Feb-2019].

[79] "sensor_msgs/Image Documentation." [Online]. Available: http://docs.ros.org/api/sensor_msgs/html/msg/Image.html. [Accessed: 12-Feb-2019].

[80] “sensor_msgs/CameraInfo Documentation.” [Online]. Available: http://docs.ros.org/api/sensor_msgs/html/msg/CameraInfo.html. [Accessed: 12-Feb-2019].

[81] “tf - ROS Wiki.” [Online]. Available: http://wiki.ros.org/tf. [Accessed: 10-Feb-2019].

[82] T. Foote, “tf: The transform library,” in *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, Woburn, MA, USA, 2013, pp. 1–6.

[83] “tf2 - ROS Wiki.” [Online]. Available: http://wiki.ros.org/tf2. [Accessed: 10-Feb-2019].

[84] “gazebo_msgs/ModelStates Documentation.” [Online]. Available: http://docs.ros.org/jade/api/gazebo_msgs/html/msg/ModelStates.html. [Accessed: 12-Feb-2019].

[85] J. Huerta, “Introducing The Quaternions,” p. 25.

[86] “rqt_graph - ROS Wiki.” [Online]. Available: http://wiki.ros.org/rqt_graph. [Accessed: 12-Feb-2019].

[87] “ROS.org | Powering the world’s robots.” .

[88] “Python Data Analysis Library — pandas: Python Data Analysis Library.” [Online]. Available: https://pandas.pydata.org/. [Accessed: 02-Feb-2019].

[89] “Gazebo : Tutorial : ROS Depth Camera Integration.” [Online]. Available: http://gazebosim.org/tutorials?tut=ros_depth_camera&cat=connect_ros. [Accessed: 12-Feb-2019].

[90] “Gazebo : Tutorial : Gazebo plugins in ROS.” [Online]. Available: http://gazebosim.org/tutorials?tut=ros_gzplugins#Multicamera. [Accessed: 12-Feb-2019].

[91] “The Entrepreneurial University - TUM.” [Online]. Available: https://www.tum.de/. [Accessed: 31-Jan-2019].

[92] “Computer Vision Group - Dataset Download.” [Online]. Available: https://vision.in.tum.de/data/datasets/rgbd-dataset/download. [Accessed: 31-Jan-2019].

ANEXO 1. PRESUPUESTO

ÍNDICE DEL PRESUPUESTO

1. COSTES DE EJECUCIÓN MATERIAL

El coste de ejecución material incluye tres categorías, coste de equipos, coste de software y coste de mano de obra por el tiempo empleado en el proyecto.

1.1 Costes de equipos

Se considera que el tiempo de duración de los equipos es de 5 años y la duración del proyecto es de, aproximadamente, 6 meses.

CONCEPTO	PRECIO UNITARIO	CANTIDAD	SUBTOTAL
Ordenador portátil Dell XPS 15	1928,74 €	5 %	96,40 €

Subtotal: ……. 96,4 €

1.2 Costes de software

CONCEPTO	PRECIO UNITARIO	CANTIDAD	SUBTOTAL
Sistema operativo Linux: Ubuntu 16.04	0 €	1	0 €
ROS Kinetic	0 €	1	0 €
Simulador Gazebo	0 €	1	0 €
Microsoft Office 365	4,20 €/mes	6	25,20 €

Subtotal: ……. 25,20 €

1.3 Costes de mano de obra

El coste unitario de la mano de obra será de 35 €/h. La dedicación será de 6 horas al día, 5 días a la semana.

CONCEPTO	CANTIDAD	SUBTOTAL
Estudio del problema y diferentes alternativas	20 días	4.200,00 €
Pruebas con algoritmos de odometría visual	25 días	5.250,00 €
Estudio y pruebas paquete detección ArUco: *aruco_detect*	10 días	2.100,00 €
Desarrollo paquete de localización mediante marcadores *ArUco: robot_pose*	30 días	6.300,00 €
Integración paquete *aruco_detect* y *robot_pose*	5 días	1.050,00 €
Desarrollo de varios entornos en Gazebo para realizar simulaciones	5 días	1.050,00 €
Pruebas localización mediante marcadores	20 días	4.200,00 €
Realización de la documentación	15 días	3.150,00 €

Subtotal: ……. 125 días ……. 27.300,00 €

La duración del proyecto es de 130 días laborables, aproximadamente 6 meses.

1.4 Coste total del presupuesto de ejecución material

CONCEPTO	SUBTOTAL
Coste de equipos	**96,40** €
Coste de software	**25,20** €
Coste de mano de obra	**27.300,00 €**

Subtotal: ……. 27.421,60 €

2. GASTOS GENERALES Y BENEFICIO INDUSTRIAL

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 16 % sobre el coste de ejecución material

CONCEPTO	SUBTOTAL
Gastos generales y beneficio industrial	4.387,46 €

3. IMPORTE TOTAL

CONCEPTO	SUBTOTAL
Coste total del presupuesto de ejecución material	**27.421,60 €**
Gastos generales y beneficio industrial	**4.387,46 €**

TOTAL: ……. **31.809,06 €**

IVA 21%: ……. **6.679,90 €**

TOTAL, IVA INCLUÍDO: ……. **38.488,96 €**

El Importe Total del proyecto suma la cantidad de:

Treinta y Ocho Mil Cuatrocientos Ochenta y Ocho Euros con Noventa y Seis Céntimos

ANEXO 2. PLANIFICACIÓN

En el siguiente cronograma se presentan los tiempos orientativos dedicados a las distintas tareas del trabajo.

Tareas	2018/2019																							
	Septiembre				Octubre				Noviembre				Diciembre				Enero				Febrero			
	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
Estudio del problema y diferentes alternativas																								
Pruebas con algoritmos de odometría visual																								
Estudio y pruebas paquete detección ArUco: *aruco_detect*																								
Desarrollo paquete de localización mediante marcadores ArUco: *robot_pose*																								
Integración paquete *aruco_detect* y *robot_pose*																								
Desarrollo de varios entornos en Gazebo para realizar simulaciones																								
Pruebas localización mediante marcadores																								
Realización de la documentación																								

Figura 1. Cronograma del proyecto

ANEXO 3. MATERIALES ADJUNTADOS

1. MATERIAL ADJUNTO

El material adjunto se puede descargar en el siguiente enlace:

https://drive.google.com/open?id=1gt0BBLnXLwU038a3I3uVtu6BvYpPPv3t

Se descargará un archivo comprimido MaterialAdjunto.zip. En él se incluyen los paquetes necesarios para realizar la localización a partir de marcadores. Además, se incluyen los programas utilizados para evaluar el funcionamiento de las librerías *Fovis* y *OpenCVRGB-Depth Processing*. También se incluyen los resultados obtenidos y un vídeo. Se detalla la estructura de los anexos a continuación:

- Código
  - Localización
    - cameraParams: Directorio que almacena los archivos con los parámetros de las cámaras en formato *.yml*.
    - map_aruco: Paquete de ROS que obtiene las localizaciones de los ArUcos de Gazebo y los escribe en un archivo en formato *.yml*.
    - robot_pose: Paquete de ROS que estima la localización de un robot a partir de 8 cámaras referidas al mismo y una serie de marcadores dispuestos en posiciones conocidas.
  - Fovis
    - Simulación_fovis: Paquete de ROS que estima la pose de una cámara RGBD mediante la librería *Fovis*.
    - Simulacion_estereo: Paquete de ROS que estima la pose de una cámara estéreo mediante la librería *Fovis*.
  - OpenCVRGBD
    - Simulacion_OCV: Paquete de ROS que estima la pose de una cámara RGBD mediante la librería *OpenCVRGB-Depth Processing*.

Cabe desatacar que, al tratarse de un proyecto en desarrollo, ciertos elementos del código no se encuentran en su versión definitiva.

- Resultados
    - Localización:
        - En este directorio se incluyen los resultados obtenidos durante los experimentos en un archivo *html*.
        - Además, se incluye un vídeo en el que se muestra una simulación con los procesos de mapeo navegación, realizado por otra persona en otro módulo de este trabajo, y la localización estimada tal y como se detalla en este documento.

          En el vídeo se muestran el simulador Gazebo, RViz [1], aplicación que permite la visualización de los mensajes enviados dentro del sistema, y FlexBe [2], herramienta mediante la cual se desarrolló la máquina de estados que coordina el proceso (también en otro módulo del proyecto).

    - Fovis
        - En este directorio se incluyen los resultados obtenidos durante los experimentos en un archivo *html*.
    - OpenCVRGBD
        - En este directorio se incluyen los resultados obtenidos durante los experimentos con el *dataset*.

2. REFERENCIAS

[1] «rviz - ROS Wiki». Accedido 13 de febrero de 2019. http://wiki.ros.org/rviz.

[2] «flexbe - ROS Wiki». Accedido 13 de febrero de 2019. http://wiki.ros.org/flexbe.