# Proyecto IROBOT
## Visión por computador

**Octubre de 2019**

Sara Roos Hoefgeest Toribio
roossara@uniovi.es

# ÍNDICE

# 1 INTRODUCCIÓN

El objetivo del proyecto es el estudio de la viabilidad del desarrollo e implementación de un robot capaz de inspeccionar la totalidad de una chapa en busca de defectos para, posteriormente, ser capaz de subsanarlos.

Uno de los principales problemas a abordar dentro de este proyecto es conocer la posición del robot. Esto es un aspecto clave de cara a permitir la correcta navegación, tanto por la fábrica como por la chapa. Dadas las características del entorno, la posición y orientación del robot se estimará con la ayuda de un arco de cámaras y una serie de balizas dispuestas en posiciones conocidas. No se puede garantizar que la posición exacta de los marcadores con respecto a un sistema de coordenadas común sea conocida. Por ello, se propone una manera de localizar los marcadores y elaborar un mapa de los mismos que permita, posteriormente, ser utilizado para que el robot se desplace sobre la chapa con la mayor precisión posible.

También se desarrolló un método que permite localizar los defectos, marcados previamente por un operador humano, sobre la chapa para su posterior reparación. Para ello, se realizará una reconstrucción 2D de la chapa a partir de la información proporcionada por una cámara RGBD dispuesta sobre el robot. Este documento expone la reconstrucción 2D de la chapa a examinar. Se complementa con el documento que expone el proceso de inspección y control del robot para permitir el correcto saneamiento de los defectos marcados.

Para desarrollar este proyecto se utilizará el sistema operativo Ubuntu 16.04 LTS y ROS (*Robot Operating System*) [1], software que proporciona las librerías y herramientas necesarias para implementar aplicaciones de robótica. En concreto, se utilizará la versión ROS Kinetic.

En este documento se expone el trabajo desarrollado durante este proyecto. Se realizó un estudio de las diferentes alternativas existentes para solucionar el problema de la localización, expuesto en el capítulo *Estado del arte*. Además, se exponen las bases teóricas de los algoritmos desarrollados, así como los experimentos y resultados obtenidos en las numerosas pruebas desarrolladas con el fin de escoger la estrategia que mejor se ajustaba a este caso concreto. También se expone brevemente la implementación en ROS. Para ver más en detalle las implementaciones, ver el documento *Manual de usuario* que se adjunta junto a este informe.

## 1.1 RESUMEN MÉTODO IMPLEMENTADO

### 1.1.1 LOCALIZACIÓN

Previamente a la localización del robot, es preciso realizar un mapeo de los marcadores para obtener sus poses respecto a un sistema de referencia global. En ese sentido, se desarrolló un sistema de mapeado de marcadores a partir de una serie de imágenes de los mismos, tomadas con una sola cámara calibrada.

El sistema de referencia global, al que se llamará sistema del mundo, se fijará en el lugar del primer marcador detectado en las imágenes que cumpla unas determinadas

características. Una vez localizado el primer marcador y fijado el sistema del mundo, se irán estimando las poses del resto de los marcadores cuando se detecten en las imágenes junto a marcadores previamente almacenados en el mapa.

Para conseguir realizar el mapeo satisfactoriamente, es necesario que, cada vez que un nuevo marcador sea detectado, esté presente en la misma imagen otro cuya pose ya haya sido estimada previamente.

Una vez conocida la localización de los marcadores, se estima la posición y orientación del vehículo a partir de los marcadores encontrados mediante un arco de 8 cámaras calibradas dispuestas en el robot.

La estrategia de localización implementada comienza con una etapa de detección de los marcadores en las imágenes de cada una de las cámaras que conforman el arco. Después, cada cámara sigue un enfoque monocular calculando la pose absoluta del robot a partir de las correspondencias de las proyecciones 2D de las esquinas de los marcadores y sus correspondientes puntos 3D almacenados en el mapa de marcadores. Finalmente, se escoge la mejor estimación entre las 8 cámaras y se aplica un Filtro de Kalman para descartar malas estimaciones y proporcionar una localización más fiable.

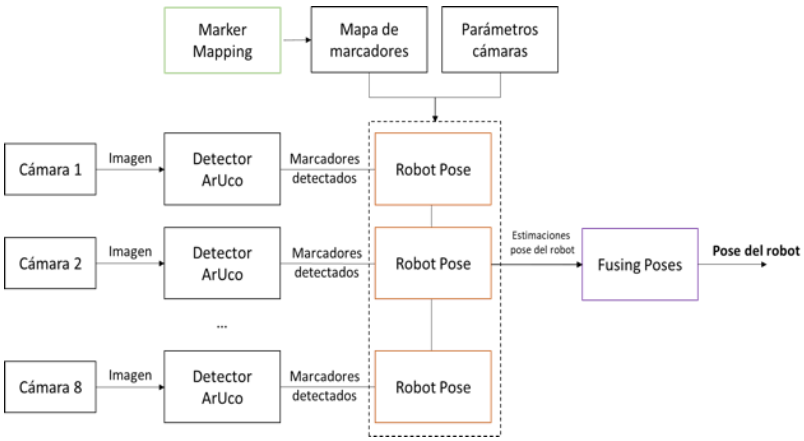


*Figura 1. Esquema del método propuesto. Robot Pose hace referencia al paquete de localización para cada una de las cámaras. Fusing Poses junta la información de las 8 cámaras para proporcionar la pose final del robot. Marker Mapping es el paquete de mapeo de marcadores.*

### 1.1.2 RECONSTRUCCIÓN 2D DE LA CHAPA

Se desarrolló un método para localizar los defectos marcados en la chapa de metal para su posterior reparación. Para ello, se realizará una panorámica de la hoja a partir de la información proporcionada por la cámara RGBD. Suponemos que las zonas defectuosas han sido marcadas previamente por un operador humano, cerrándolas con un contorno blanco, por ejemplo, utilizando tiza.

Para construir la panorámica, es necesario hacer un primer recorrido por la chapa para asegurarse de que las imágenes tomadas cubren toda la superficie. Los píxeles de la imagen correspondientes a la chapa se segmentan y se transforman en una vista cenital

mediante una homografía. Después tenemos un conjunto de imágenes que muestran diferentes regiones de la chapa en posición real. Un panorama de estas imágenes se construye a partir del cálculo de la traslación entre imágenes según un enfoque basado en *Template Matching*. Con el objetivo de cubrir toda la chapa, es necesario realizar varios recorridos con la cámara RGBD enfocada en diferentes sentidos. Así, se generan varios panoramas parciales que serán combinados en uno general.

Una vez obtenida la imagen panorámica, se buscan en ella los defectos marcados y se poligonalizan dichas marcas. Las dimensiones de la chapa son conocidas, por lo que las ubicaciones de los polígonos se pueden proporcionar en metros con respecto a un sistema de referencia situado en una esquina de la misma. Una vez hecho esto, el robot podrá dirigirse a estas posiciones para reparar la zona indicada.

Al realizar los trabajos de inspección y reparación, debe tenerse en cuenta que la chapa puede estar a una distancia considerable del suelo. Si el robot se sale de la superficie de trabajo, podría ser peligroso y causar daños graves al robot e incluso a los operadores humanos. Para solucionar este problema, la placa se divide en 4 zonas, cada una asociada a una posible orientación principal del robot. En cada zona, el robot podrá, manteniendo la orientación asociada, calcular trayectorias seguras para realizar su tarea.

Los defectos, recibidos en forma de polígono, se dividirán según las zonas de trabajo, determinadas en este caso por la posición y el tamaño de la herramienta respecto al robot. De esta manera, cada división se cubrirá con una única orientación. Para facilitar la reparación completa de los defectos, se realiza una descomposición trapezoidal, dando como resultado formas sencillas que facilitan la operación.

Tanto la división de la chapa en zonas de trabajo como la descomposición de los polígonos son tratadas en otro de los documentos adjuntos.

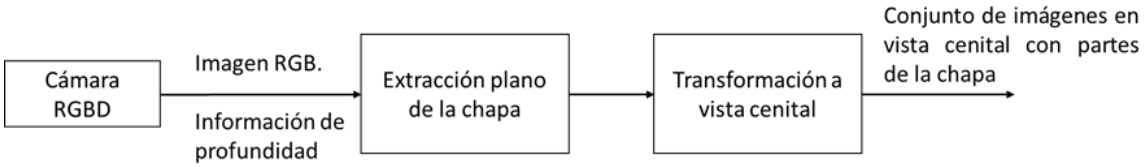


*Figura 2. Esquema que muestra el proceso de captura de imágenes (a), extracción del plano de la chapa (b) y transformación a vista cenital (c).*

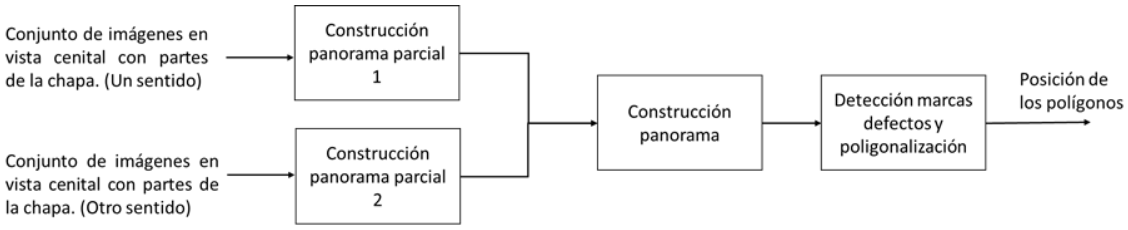


*Figura 3. Construcción de los panoramas parciales (c) a partir del conjunto de imágenes (a, b) para su posterior combinación en un panorama global (d). En la imagen (e) se muestran los polígonos que ajustan las marcas de los defectos.*

# 2 ESTADO DEL ARTE

## 2.1 LOCALIZACIÓN

En primer lugar, se estudiaron diferentes alternativas en el apartado de la localización del robot. 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 [2] o filtros de partículas [3], que permite combinar las estimaciones procedentes de distintas fuentes para lograr una pose más robusta. Por ejemplo, en [4], 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.

### 2.1.1 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).

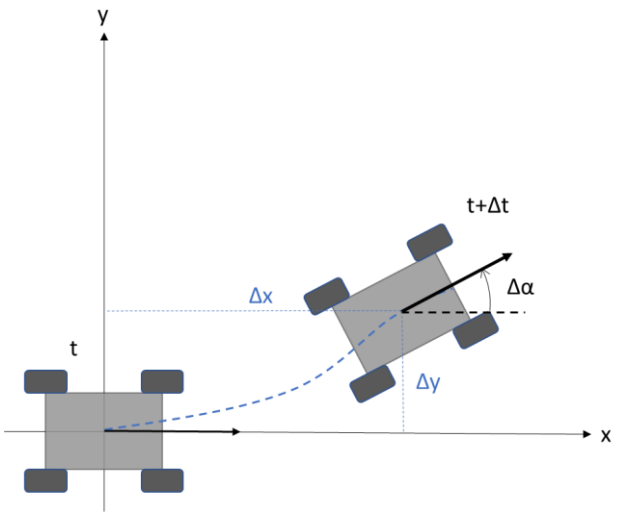


*Figura 4. Odometría. Estimación de la pose de un vehículo de manera incremental, recuperando el movimiento realizado en instantes consecutivos.*

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* [5].

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 [6]. 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 [7]. 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 [8]. 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 [9]. 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 [10].

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 [11]. 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 [12].

Además, en la última década, se han desarrollado algoritmos de odometría visual empleando cámaras RGB-D, como la Kinect de Microsoft [13]. Este sistema ha sido utilizado en diferentes aplicaciones, como en vehículos submarinos autónomos [14] o un micro vehículo aéreo autónomo (MAV) [15].

#### 2.1.1.1 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) [16]:** Este paquete estima la pose de una cámara RGB-D. La aproximación implementada en este método se describe en [17].

- **viso2_ros [18]:** 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 [19].

- **fovis_ros [20]:** 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 [19].

- **Módulo OpenCV RGB-Depth processing [21]:** Módulo incluido en librería OpenCV [22]. Este método está inspirado en el trabajo de Steinbrucker et al [23], en el que presentó un método global a partir de las imágenes RGB-D proporcionadas por una Kinect de Microsoft.

### 2.1.2 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 [24], científicos de la Politécnica de Torino implementaron un sistema basado en sensores infrarrojos [25] o el sistema utilizando láseres realizado por investigadores del departamento de robótica de la Universidad de Oxford [26].

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 [27]. En este método, el movimiento de la cámara y la estructura 3D del entorno son estimadas simultáneamente utilizando un filtro de Kalman extendido (EKF) [28]. 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*) [29], 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*) [30], que sigue una estrategia monocular basada en puntos característicos, según se describe en el artículo de los autores [31], u ORB-SLAM, librería para cámaras monocular, estéreo y RGB-D [32]. 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++ [33] o la de Tateno et al. [34].

### 2.1.3 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.

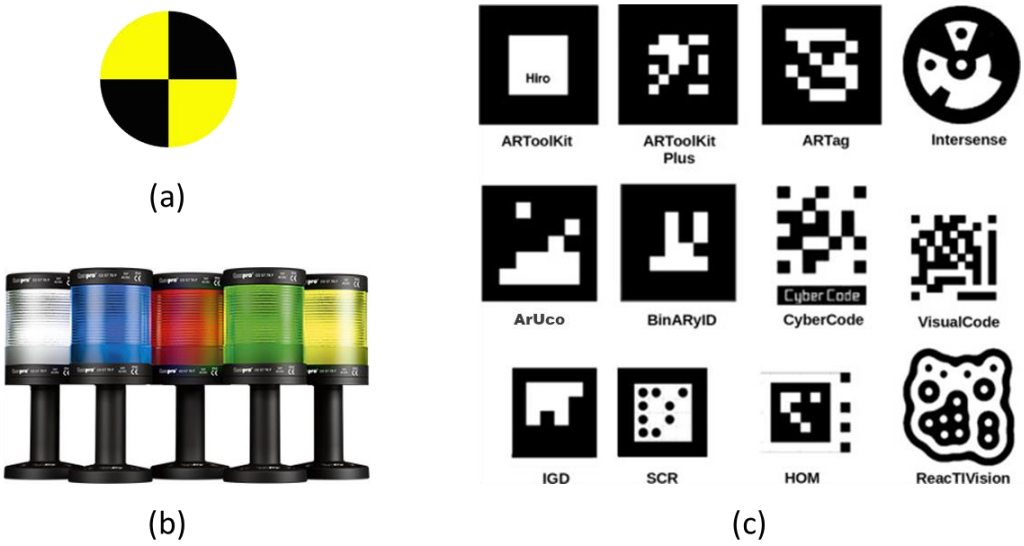


*Figura 5. Modelos de balizas visuales. (a) Marcador utilizado en los crash test. (b) Balizas luminosas de colores. (c) Sistemas basado en marcadores fiduciales.*

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 5, los basados en marcadores cuadrados son los que gozan de una mayor popularidad, especialmente en el área de la realidad aumentada [35], [36]. 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 [37]. Otros sistemas de marcadores cuadrados destacados son ARTag [38], ARToolKit Plus [39] o BinARyID [40].

Entre los marcadores no cuadrados, los más empleados son los circulares Intersense [41] y ReacTIVision [42], con forma de ameba.

## 2.2 RECONSTRUCCIÓN 2D DE LA CHAPA

Después de realizar una búsqueda, no se ha encontrado mucha bibliografía para esta aplicación específica. Sin embargo, la creación de panoramas es una técnica ampliamente

utilizada en diferentes aplicaciones. Existen diferentes estrategias que permiten la creación de un panorama, por ejemplo, en [43], [44] o [45] se describen diferentes métodos que utilizan características invariables de las imágenes. Otras estrategias serían el uso de métodos basados en la combinación de plantillas con características, como la propuesta de Shunping Ji y Dawen Yu [46].

# 3 LOCALIZACIÓN MEDIANTE UN ARCO DE 8 CÁMARAS

Uno de los principales problemas es la localización del robot dentro del entorno. A priori, no se puede garantizar que existan elementos externos que puedan ser utilizados como balizas. Para resolver este problema, se propone un sistema de localización basado en la instalación de marcadores ArUco y el uso de un anillo de 8 cámaras dispuesto sobre el robot móvil. Las cámaras estarían calibradas, por lo tanto, se conocerían sus parámetros intrínsecos. Además, sus parámetros extrínsecos serán también conocidos respecto del robot debido a que las cámaras se dispondrán sobre el mismo en posiciones y orientaciones conocidas.

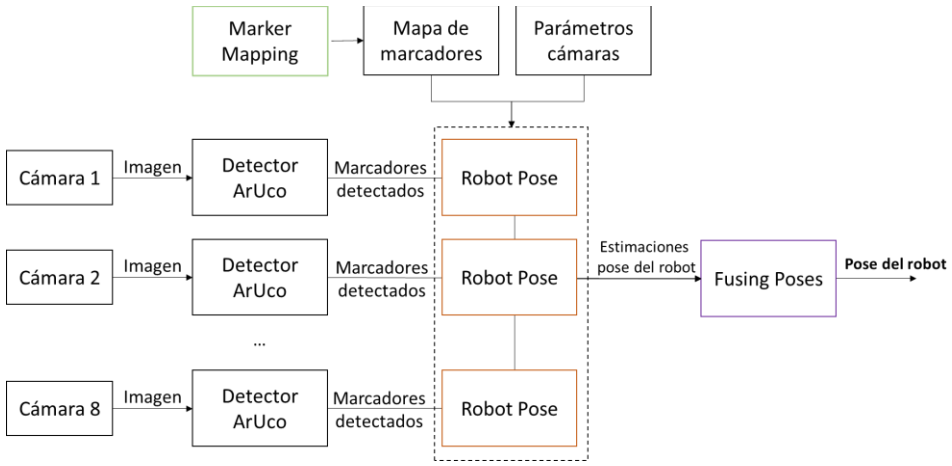


Figura 6. *Esquema del método propuesto.* Robot Pose *hace referencia al paquete de localización para cada una de las cámaras.* Fusing Poses *junta la información de las 8 cámaras para proporcionar la pose final del robot.* Marker Mapping *es el paquete de mapeo de marcadores.*

No se puede garantizar que la posición exacta de los marcadores con respecto a un sistema de coordenadas común sea conocida. En este artículo se propone una manera de localizar los marcadores y elaborar un mapa de los mismos utilizando una sola cámara, que permita, posteriormente, ser utilizado para que el robot se desplace sobre la chapa con la mayor precisión posible.

El sistema de mapeo utiliza una cámara calibrada para estimar la pose de los marcadores respecto a un sistema de referencia común, elegido en la posición de uno de los marcadores en el entorno. Una vez realizada una primera estimación de sus posiciones se aplica un ajuste *bundle* para minimización del error y optimizar sus localizaciones.

En cuanto a la localización, cada cámara que forma el arco sigue, individualmente, un esquema monocular, estimando la posición del robot resolviendo el problema de *Perspective-n-Point* (PnP) a partir de las poses 3D de marcadores y sus proyecciones en

sus imágenes. Posteriormente, se escoge la mejor estimación entre las 8 cámaras y se someten los datos a un filtro de Kalman para eliminar malas estimaciones y conseguir una localización más fiable.

## 3.1 DETECCIÓN DE LAS BALIZAS

Se han empleado los marcadores ArUco [47], ampliamente utilizados en aplicaciones de realidad aumentada y en algoritmos de navegación de robots.

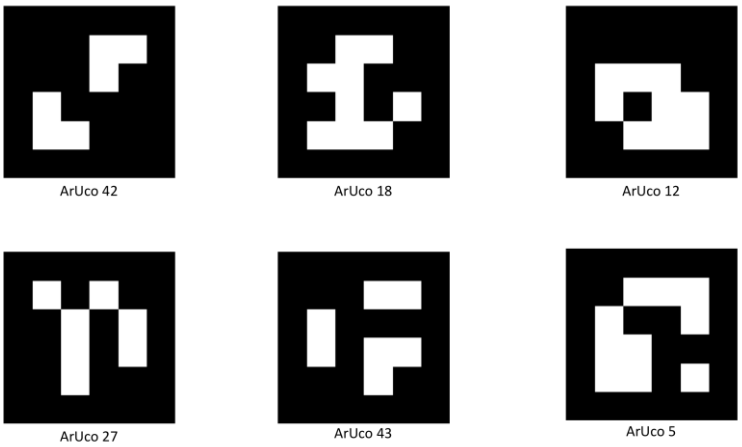


Figura 7. Marcadores *ArUco*

Un marcador ArUco, ver Figura 7, 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 se 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 [48, Cap. 6.1] para segmentarlos, después se extraen los contornos de la imagen umbralizada y se aplica el algoritmo de Suzuki and Abe [49] para descartar las formas que no se asemejen a un cuadrado.

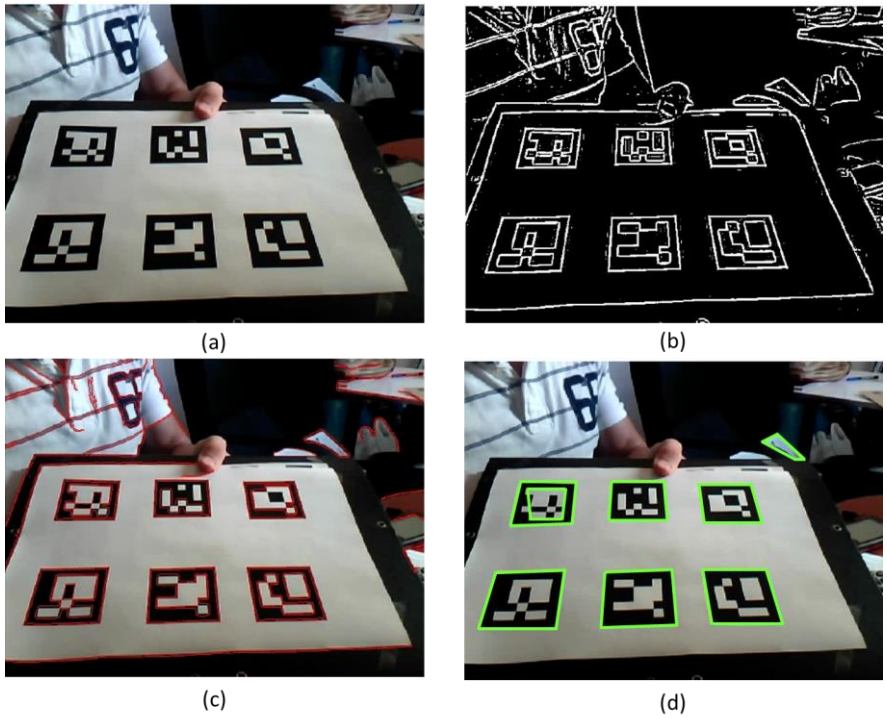


*Figura 8. Proceso de detección de marcadores. (a) Imagen de entrada. (b) Imagen tras el umbralizado adaptativo. (c) Resultado de la extracción de contornos. (d) Descarte de formas que no se asemejen a un cuadrado. Imagen obtenida de [50].*

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 [51], separando los bits blancos y los negros, como se muestra en la Figura 9.

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.

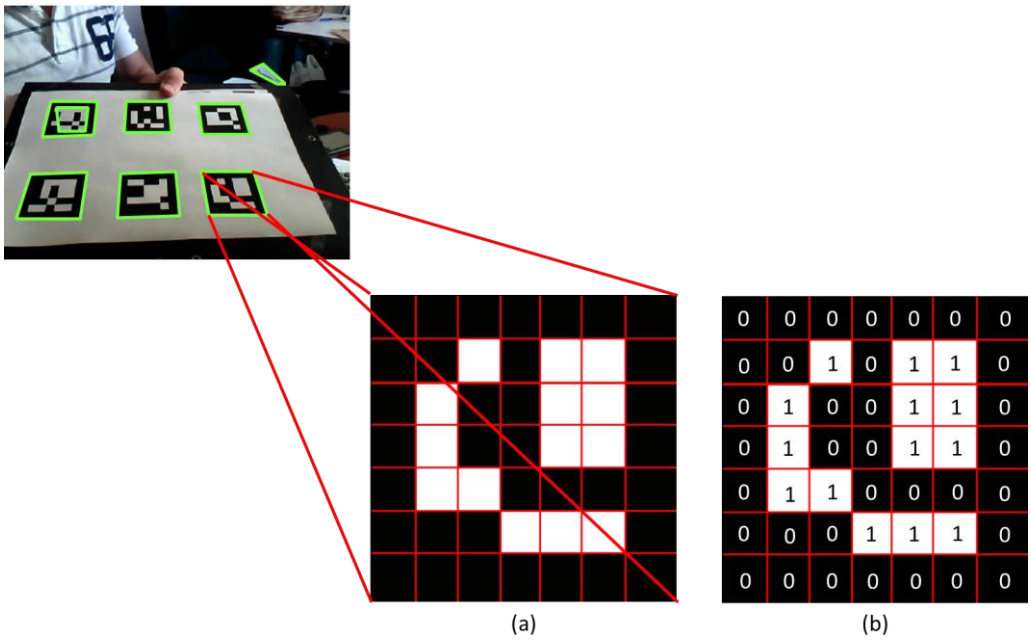


*Figura 9. Resultado del marcador tras aplicar una transformación de perspectiva y la división del ArUco en celdas (a). Después se analizan los bits de la matriz para identificar el marcador dentro de un diccionario específico (b). Imagen obtenida de [50]*

## 3.2 MAPEADO DE LOS MARCADORES

Para llevar a cabo la localización del robot utilizando marcadores dispuestos en el entorno es preciso conocer sus poses respecto a un sistema de referencia común. Para ello, se desarrolló un sistema de mapeo de marcadores a partir de una serie de imágenes de los mismos proporcionadas por una sola cámara calibrada.

Para que la cámara pueda capturar todos los marcadores el robot deberá moverse. En ese sentido, el robot realizará un movimiento controlado que asegure la visión de todos los marcadores. El movimiento del robot podrá alterarse en función del entorno para permitir la visión de todos los marcadores.

En el mapa se almacenan las poses de los marcadores referidas a un sistema de referencia común, al que se llamará sistema del mundo, que se fijará en uno de los marcadores ArUco.

El algoritmo de mapeo desarrollado se puede dividir en una serie de etapas. En primer lugar, se detectarán los marcadores en las imágenes y se escogerá uno de ellos para fijar el sistema de coordenadas del mundo. Posteriormente, se estimarán las poses del resto de marcadores respecto del sistema de referencia. Finalmente, una vez tenida una estimación de las poses, se aplicará un ajuste *bundle* para minimizar el error de la estimación y optimizar las poses de los marcadores.

El marcador que definirá la posición y orientación del sistema de coordenadas global será aquel que esté más cercano a la cámara, tenga una orientación más favorable y además tenga otros marcadores más cerca.

El sistema de coordenadas asociado a cada marcador se puede ver en la Figura 2 (b), en el que el eje Z será perpendicular al plano que contiene al marcador y el origen situado en el centro del mismo.

Una vez localizado el primer marcador y fijado el sistema del mundo, se irán estimando las poses del resto de los marcadores cuando se detecten en las imágenes junto a marcadores previamente almacenados en el mapa. Para ello, se recorren las imágenes que presenten más de un ArUco estimando la pose relativa entre ellos y con la cámara.

A priori, los únicos datos de los que se dispone son: el número de marcadores dispuestos en el entorno, sus identificadores y el tamaño de los mismos. A partir de estos datos y las imágenes tomadas por la cámara se detectan los marcadores en las imágenes utilizando un paquete de ROS ya desarrollado, *aruco_detect* [52]. Este paquete utiliza las librerías que proveen los autores del sistema de marcadores, explicado en el artículo original [50]. El algoritmo proporciona las 4 esquinas y el identificador de cada marcador presente en la imagen.

Para calcular la pose de los marcadores respecto del mundo, es preciso obtener las transformaciones entre los sistemas de la cámara para cada imagen tomada y los marcadores presentes en las mismas.

Para modelar la cámara se supone un modelo *pin-hole* [48, Sec. 11.3.1]. Resolviendo el problema de *Perspective-n-Point* (PnP) [53] 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. 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).

Resolviendo el problema PnP dadas las posiciones 3D de las esquinas en el sistema del propio marcador y sus correspondientes proyecciones 2D en la imagen, se obtiene la pose de la cámara calibrada respecto del sistema del propio marcador.

La figura 3 muestra este proceso. I1 e I2 representan las imágenes tomadas por la cámara que hace el mapeo para las posiciones 1 y 2 del robot. Supongamos que en la primera posición seleccionada se ven los marcadores 0 y 1, mientras que en la segunda se ven los marcadores 1 y 2. Usando los algoritmos descritos en los párrafos anteriores, obtendremos las transformaciones entre los sistemas de coordenadas de cada marcador y la posición de la cámara en cada imagen. En particular, para la imagen 2 obtendremos las transformaciones ${}^{C2}T_{M1}$ y ${}^{C2}T_{M2}$. Al ser común la posición de la cámara, se puede calcular la transformación entre ambos marcadores. En particular, la transformación que refiere la posición del marcador 2 respecto al marcador 1 será:

$$ {}^{M1}T_{M2} = {}^{M1}T_{C2} \cdot {}^{C2}T_{M2} \tag{1}$$

Así, si uno de los dos marcadores ya ha sido almacenado en el mapa, la transformación del marcador desconocido respecto del mundo se obtiene de una manera directa. En este ejemplo, se supone que el sistema asociado al marcador 0 se toma como sistema de coordenadas del mundo. Usando la ecuación (1), podemos calcular la transformación existente entre el marcador 1 y el sistema de coordenadas global, puesto que los

marcadores 0 y 1 son visibles en la misma imagen. Así, conoceremos ${}^{W}T_{M1} = {}^{M0}T_{M1}$ . Para calcular la pose del sistema de coordenadas asociado al marcador 2 respecto al sistema de mundo bastará con aplicar la ecuación (2) y así podremos añadirlo al mapa

$$ {}^{W}T_{M2} = {}^{W}T_{M1} \cdot {}^{M1}T_{M2} \tag{2}$$

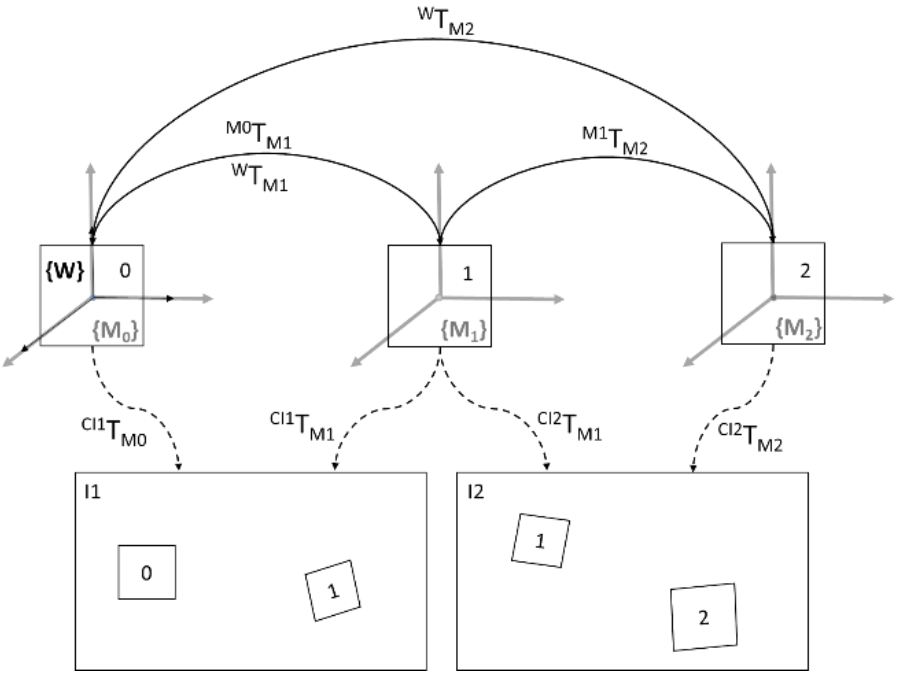


*Figura 10. Esquema transformaciones del sistema de mapeo. Siendo {M₀}, {M₁}, {M₂} los sistemas de coordenadas de los marcadores en el espacio e I1, I2 dos imágenes tomadas por la cámara en instantes diferentes.*

En el caso de haber más de un marcador almacenado en el mapa, se calcula la pose del marcador desconocido respecto al marcador conocido más cercano a la cámara, previsiblemente mejor detectado.

Una vez obtenida una primera estimación de la localización de todos los marcadores presentes en el entorno, se aplica un *ajuste bundle* [5] para optimizar las poses.

### *3.2.1* AJUSTE *BUNDLE*

La solución propuesta consiste en buscar una optimización global de la posición 3D de los marcadores mediante la minimización del error de reproyección de las esquinas de todos los marcadores presentes en las imágenes mediante un *ajuste bundle*.

Este método consiste en refinar conjuntamente un conjunto de estimaciones iniciales de los parámetros de la cámara para encontrar el conjunto de parámetros que predice con mayor precisión las ubicaciones de los puntos 3D observados en las imágenes disponibles buscando minimizar el error de reproyección de los mismos.

Dada una estimación inicial de las matrices de proyección de las diferentes vistas $P_i$, y un conjunto de puntos 3D reconstruidos con su proyección 2D en la imagen $\{X_j,\ x_j^i\}$ se busca minimizar el error de reproyección con respecto a todos los puntos 3D y los parámetros de la cámara:

$$\sum_{j} \sum_{i \in V(j)} \left| P_i(X_j) - x_j^i \right|^2 \tag{3}$$

Donde V(j) es la lista de índices de las vistas de la cámara en los que el punto $X_j$ es visible. $P_i(X^j)$ representa la reproyección 2D del punto $X_j$ en la cámara *i* utilizando la matriz de proyección $P_i$.

Un método eficiente para resolver este problema es la minimización de Levenberg-Marquart [54]. Se trata de un método iterativo utilizado para resolver problemas de ajuste de métodos por mínimos cuadrados no lineales.

## 3.3 LOCALIZACIÓN DEL ROBOT

En primer lugar, se detectan los marcadores presentes en las imágenes capturadas por las cámaras en cada instante. Como ya se ha explicado previamente, la detección de los marcadores en las imágenes se lleva a cabo utilizando el paquete *aruco_detect* [52]. Este paquete proporciona las 4 esquinas y el identificador de cada marcador presente en la imagen.

Una vez detectados los marcadores en las imágenes, se realiza una estimación de la pose del robot a partir los presentes en cada una de las cámaras.

Para modelar las cámaras se supone un modelo *pin-hole* [48, Sec. 11.3.1]. Resolviendo el problema de *Perspective-n-Point* (PnP) [53], ver apartado 3.3.1, 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. 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 3D de los marcadores correspondientes respecto del sistema del mundo, obtenidas del mapa de marcadores calculado previamente.

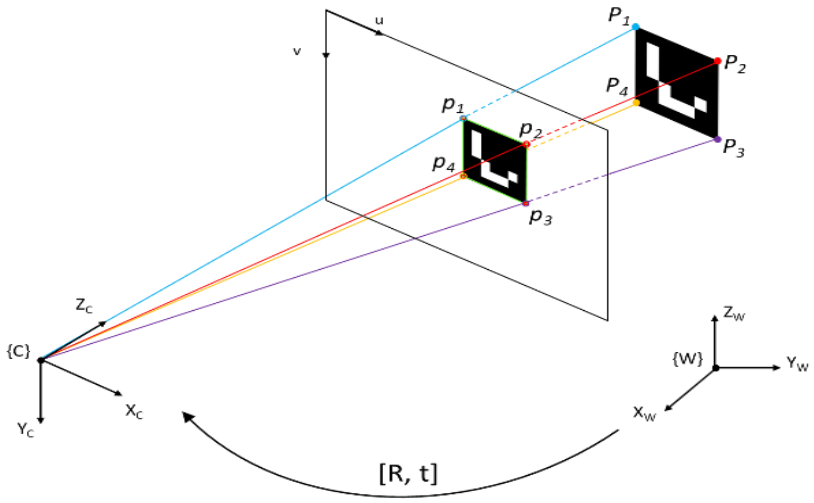


*Figura 11. Estimación de la pose de la cámara a partir de un marcador.* ${}^{C}T_{W} = [\,{}^{C}R_{W} \mid {}^{C}t_{W}]$

Tras resolver el problema, se obtiene la transformación entre el sistema de coordenadas de la cámara y el sistema del mundo, ${}^{C}T_{W}$, ver Figura 11. De esta manera, obtener la matriz de transformación del robot y el sistema del mundo ${}^{W}T_{R,}$ se puede calcular mediante de manera directa como:

$$ {}^{W}T_{R} = \; {}^{W}T_{C} \cdot {}^{C}T_{R} \tag{4}$$

Donde ${}^{W}T_{C}$ es la transformación inversa de ${}^{C}T_{W}$, y ${}^{C}T_{R}$, la transformación que relaciona los sistemas de coordenadas de la cámara y el robot, definido en la fase de montaje del arco de cámaras sobre el mismo.

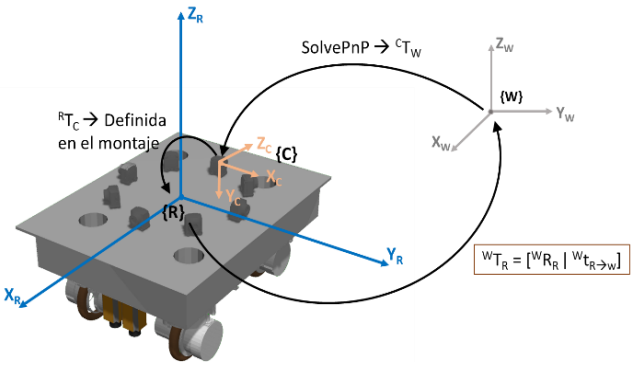


*Figura 12. Transformaciones entre sistemas de referencia.*

Para cada una de las cámaras se obtiene una estimación de la pose del robot en función de los marcadores presentes en cada una. Con el fin de escoger la mejor localización, se realiza una estimación de la precisión de la medida, basada en el error de reproyección y la distancia a la que el marcador se encuentra de la cámara.

Así, las estimaciones asociadas al menor error entre las ocho cámaras en cada instante de tiempo son sometidas a un filtrado de eliminación de *outliers*, descartando la medida si difiere notablemente de las anteriores o tiene un valor no coherente con el comando de movimiento ejecutado por el robot.

Tras pasar esa primera medida de eliminación de *outliers*, se incorpora la estimación a un Filtro de Kalman Lineal con el fin de hacer la localización final más robusta.

### 3.3.1 PERSPECTIVE-N-POINT (PNP)

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) [55].

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 [54], *P3P*, un método basado en el artículo de Gao et al [56] y *Efficient PnP*, introducido por Lepetit et al [57].

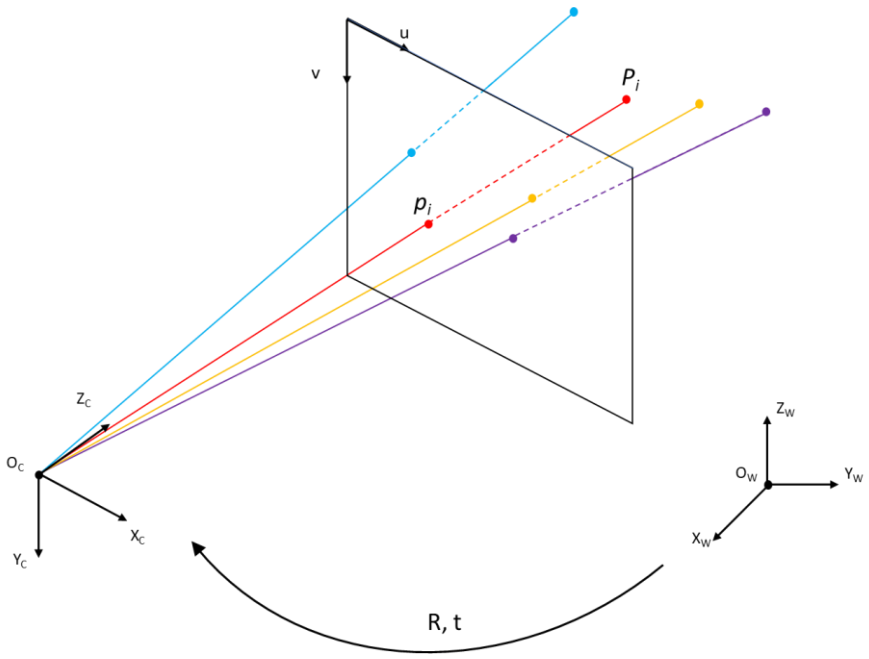


*Figura 13. Esquema del problema PnP. Dado un conjunto de puntos del espacio 3D $P_i$ (expresados en el sistema de referencia del mundo (Xw, Yw, Zw)) y sus correspondencias 2D $p_i$ en la imagen, se trata de estimar la pose de la cámara (rotación y traslación) respecto del mundo.*

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.

$$\begin{aligned} \tilde{p} &= M \cdot \tilde{P} \\ \tilde{p} &= K[R \mid t] \cdot \tilde{P} \end{aligned} \tag{5}$$

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.

$$K^{-1} \cdot \tilde{p} = [R|t] \cdot \tilde{P} \tag{6}$$

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 (7). Para extenderlo a todos los puntos, la función de coste será la suma de todos los errores de reproyección al cuadrado.

$$K^{-1} \cdot \tilde{p} - [R|t] \cdot \tilde{P} = E_{rep} \tag{7}$$

### 3.3.2 Filtro de Kalman Lineal

El filtro de Kalman [58, Sec. 3.5.1] es un filtro recursivo de predicción que se basa en el uso de técnicas de espacio de estados para estimar el estado de un sistema dinámico. En este proyecto, se ha implementado un filtro de Kalman lineal para eliminar malas estimaciones y conseguir una localización más fiable.

El filtro de Kalman se divide en dos etapas principales, predicción y corrección. En el primer paso, el modelo dinámico predice el estado del sistema. En el segundo, se corrige dicha predicción con el modelo de observación. Este procedimiento se repite para cada intervalo de tiempo.

Los estados sucesivos $s_t \in R^n$ de un proceso controlado se relacionan con un modelo dinámico que describe la transformación del vector de estado en el tiempo, descrito en la ecuación (8).

$$s_t = A s_{t-1} + w_t \tag{8}$$

Siendo *A* la matriz de estados y $w_t$ el ruido del proceso. El vector de estado que compone la matriz se definirá con un total de 18 estados (ecuación (9)): La posición (x,y,z), con su primer y segunda derivadas, es decir, velocidad y aceleración. La información de rotación en forma de ángulos de Euler (roll ($\psi$), pitch ($\theta$), yaw ($\phi$)), con su primer y segunda derivada, correspondientes a velocidad y aceleración angular.

$$X = (x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}, \psi, \theta, \phi, \dot{\psi}, \dot{\theta}, \dot{\phi}, \ddot{\psi}, \ddot{\theta}, \ddot{\phi}) \tag{9}$$

La medida de la posición del robot en cada instante t, se relaciona con el estado $s_t$ mediante el modelo de observación lineal de la ecuación (10), donde $v_t$ representa el ruido de la medida y C la matriz de observación.

$$z_t = Cs_t + v_t \tag{10}$$

El vector de medidas se compone de 6 medidas, correspondientes a la posición (x,y,z) y orientación, en forma de ángulos de Euler ($\psi, \theta, \phi$), del robot.

Como se mencionó previamente, la primera etapa del filtro de Kalman es la predicción. En cada instante de tiempo se realiza una primera estimación del estado actual, denominado estado a priori $s_t^-$, calculado al despreciar el ruido dinámico y resolviendo las ecuaciones que describen el modelo dinámico, ecuación (11). Su matriz de covarianzas $S_t^-$ es calculada durante esta etapa según la ecuación (12), donde $S_{t-1}$ es la covarianza del error de la estimación a posteriori en el instante anterior y $\Lambda_w$ es la covarianza del ruido del proceso que mide la calidad del modelo de movimiento respecto a la realidad

$$s_t^- = As_{t-1} \tag{11}$$

$$S_t^- = AS_{t-1}A^T + \Lambda_w \tag{12}$$

Posteriormente, sucede la etapa de corrección. En ella se mejora la predicción $s_t^-$ con las observaciones realizadas en el instante *t*, ($z_t$), obteniéndose la estimación a posteriori $s_t$ según la ecuación (13). Donde la diferencia $(z_t - z_t^-)$ es la medida residual que refleja la diferencia entre la medida prevista $z_t^- = Cs_t^-$ y la media real $z_t$.

En esta ecuación, el estado estimado y las mediciones son ponderados para calcular el estado corregido según $G_t$, matriz de ganancia del filtro. $G_t$ es calculada según la ecuación (15), siendo $\Lambda_v$ la matriz de covarianza de las medidas.

Su matriz de covarianza $S_t$ es calculada según la ley de propagación del error, ecuación (14).

$$s_t = s_t^- + G_t(z_t - z_t^-) \tag{13}$$

$$S_t = S_t^- - G_t C S_t^- \tag{14}$$

$$G_t = S_t^- C^T (CS_t^- C^T + \Lambda_v)^{-1} \tag{15}$$

## 3.4 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*) [59], que proporciona las librerías y

herramientas necesarias para implementar aplicaciones de robótica. En concreto, se utilizó la versión ROS Kinetic. Además, para el tratamiento de las imágenes se ha utilizado la librería de código abierto OpenCV [60], especializada en visión artificial.

Además, como se mencionó anteriormente, se utilizó el simulador Gazebo [61] con el fin de recrear un entorno industrial que pretende emular las instalaciones por las que se moverá el robot.

Algunos de 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* [62], una librería para *Python* especializada para manipulación y análisis de datos.

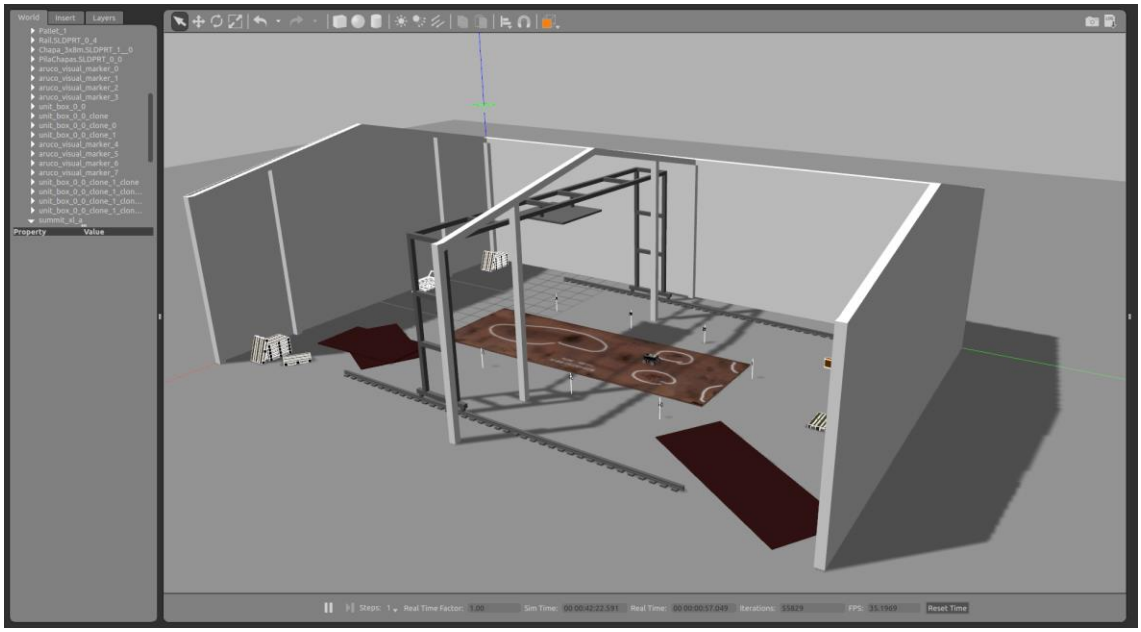

*Figura 14. Simulación en el entorno Gazebo de la industria y el robot dispuesto sobre la chapa a inspeccionar. La fábrica es de 30x20 metros y la chapa de acero de 14x5 metros.*

Para los experimentos se emplearon diferentes simulaciones en Gazebo, variando el número de marcadores y su posición, tomando como base la industria mostrada en la Figura 47, de dimensiones 30x20 metros. Esto se debió a que no pudimos disponer de un robot real. Las dimensiones de la chapa son de 14x5 metros.

### 3.4.1 CONFIGURACIÓN DEL ROBOT

Se utilizará un robot omnidireccional [63], por ello, para las simulaciones se empleó el modelo Summit-XL de Robotnik [64]. Sobre él se dispondrán un total de 8 cámaras formando un arco, según la configuración mostrada en la Figura 15.

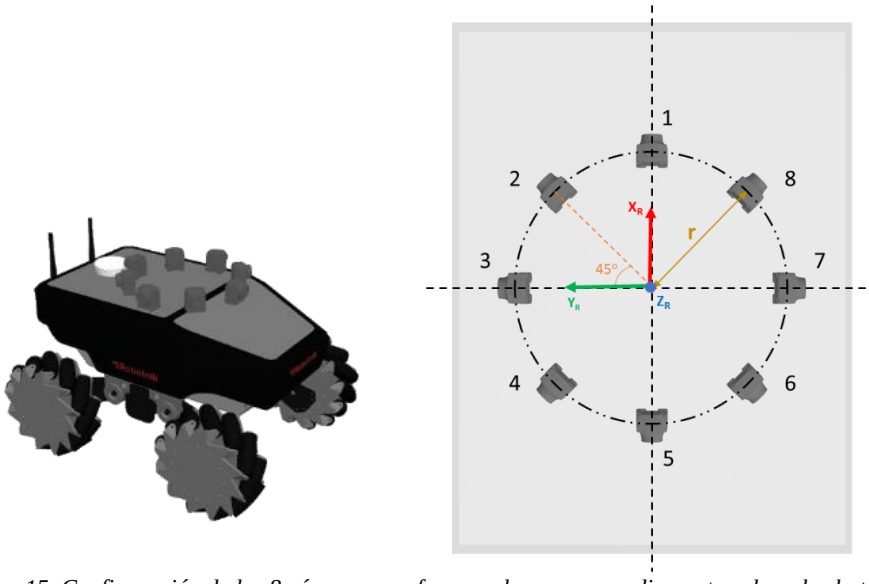


*Figura 15. Configuración de las 8 cámaras que forman el arco que va dispuesto sobre el robot.*

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 (16).

$$K = \begin{bmatrix} 1108.76 & 0 & 640.5 & 0 \\ 0 & 1108.76 & 360.5 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (16)$$

Para la tarea de localización se simuló un arco de 8 cámaras, mientras que para la reconstrucción de la chapa se simuló una cámara RGBD tipo RealSense [65].

### 3.4.2 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* [52]. Está desarrollado en lenguaje C++ basado en el módulo Aruco de OpenCV [66].

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.

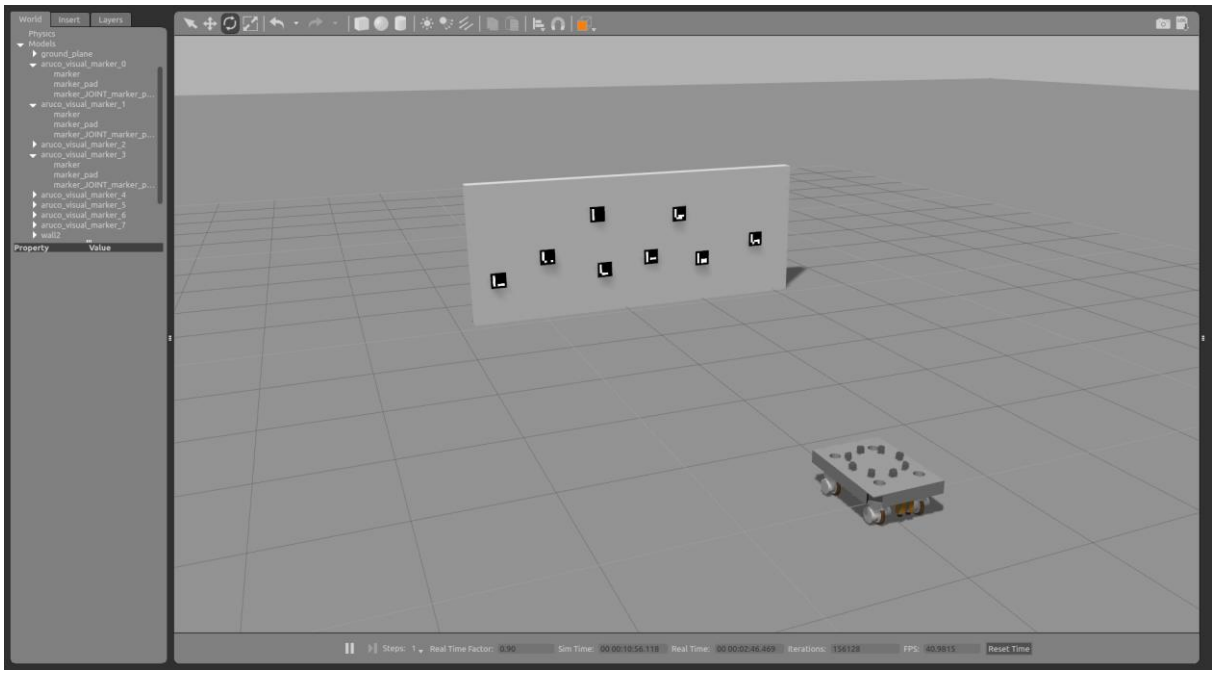

*Figura 16. Simulación en Gazebo para evaluar el funcionamiento del paquete aruco_detect.*

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 1. 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.

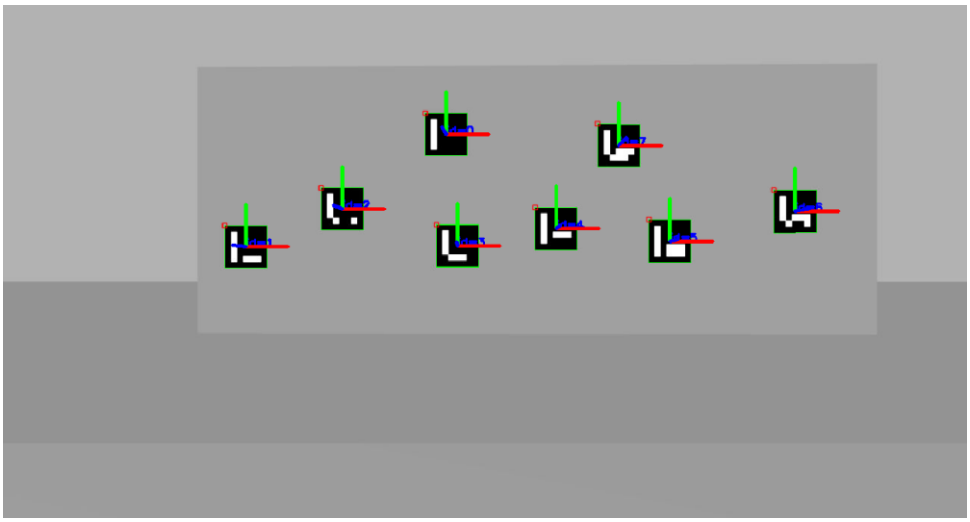

*Figura 17. Imagen publicada por el paquete aruco_detect. Se marca la esquina superior izquierda en rojo y se recuadra en verde el marcador completo. También se escribe el identificador de cada marcador según el diccionario escogido y se representa el sistema de coordenadas de cada marcador. Tamaño: 25cm. Distancia: 5m.*

### 3.4.3 MAPEO DE MARCADORES

Se realizaron pruebas en diferentes escenas para evaluar el sistema de mapeo de marcadores. En primer lugar, se evaluó en una escena con 8 marcadores, de 17 cm de lado, cercanos entre sí formando un círculo de diámetro 4 metros alrededor del robot, ver Figura 18. En este caso, el robot realizó un movimiento circular tomando un total de 9 imágenes.

Posteriormente, se evaluó el sistema en una escena industrial con una chapa en el centro para inspeccionar. En este caso, los marcadores se situaron alrededor de una chapa, de dimensiones 14x5m, con el objetivo de permitir al robot localizarse al inspeccionar y reparar la misma, Figura 19. Se tomaron un total de 33 imágenes.

A raíz de las figuras, se puede apreciar que los resultados obtenidos se ajustan a la disposición real. En el primer experimento los errores, recogidos en la Tabla 2, son menores, esto es debido a que los marcadores se encuentran más cerca de la cámara y están más cerca unos de otros, lo que facilita la detección y el cálculo de las transformaciones. Sin embargo, en el experimento de la figura 7, los errores aumentan, ver Tabla 3, debido a una mayor lejanía de los marcadores con la cámara y entre sí, lo

que obliga a que los marcadores se encuentren lo suficientemente lejos como para poder capturar un mínimo de dos en la misma imagen.

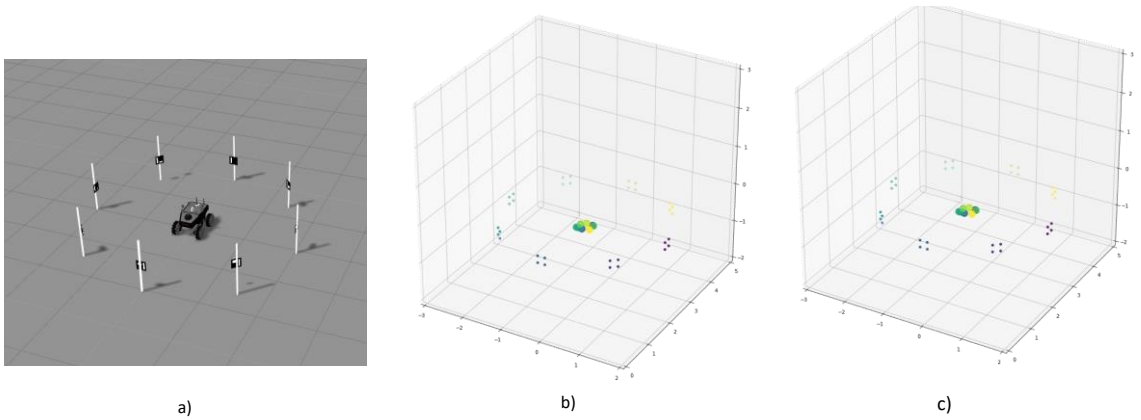


*Figura 18. Resultados mapeo marcadores. a) Escena simulada en Gazebo. b) Primera estimación de las posiciones de los marcadores. c) Resultado tras ajuste bundle. En b) y c) los puntos representan las esquinas de los marcadores, y los círculos las posiciones de las cámaras, codificados por color.*

*Tabla 2. Errores obtenidos en el experimento de la Figura 18. X,Y,Z representan las estimaciones previas al ajuste bundle e $X_{ba}$,$Y_{ba}$, $Z_{ba}$, las estimaciones tras el ajuste.*

| | X | Y | Z | $X_{ba}$ | $Y_{ba}$ | $Z_{ba}$ |
|---|---|---|---|---|---|---|
| Error medio (m) | 0.043 | 0.038 | 0.063 | 0.042 | 0.038 | 0.064 |
| Error máximo (m) | 0.158 | 0.104 | 0.212 | 0.158 | 0.106 | 0.210 |
| Error mínimo (m) | 0.001 | 0.001 | 0.004 | 0.001 | 0.004 | 0.002 |

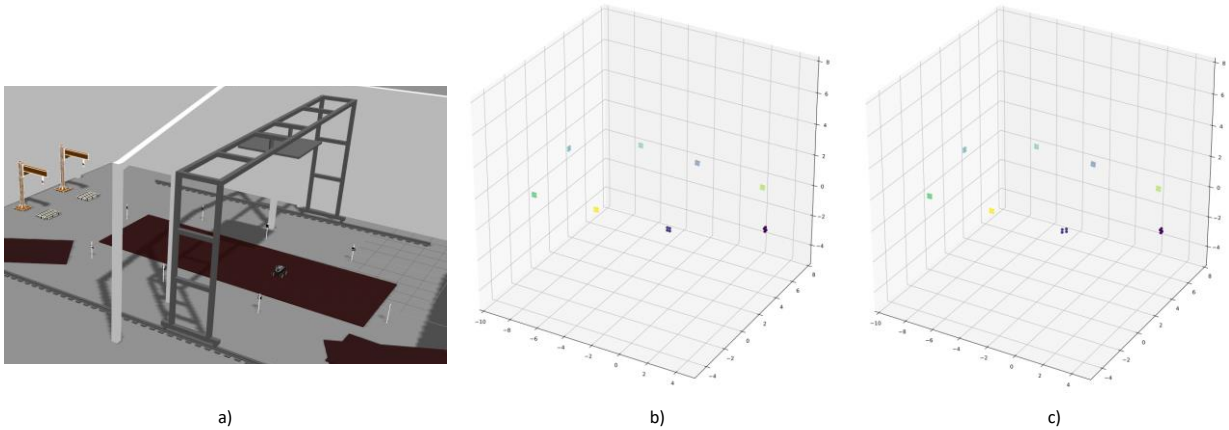


*Figura 19. Resultados mapeo marcadores. a) Escena simulada en Gazebo. b) Primera estimación de las posiciones de los marcadores. c) Resultado tras ajuste bundle. En b) y c) los puntos representan las esquinas de los marcadores, codificados por color.*

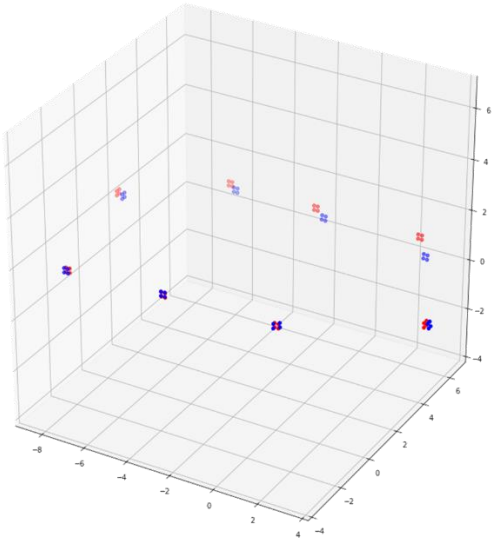

*Figura 20. Resultado final. Rojo: Real. Azul: Estimado.*

*Tabla 3. Errores obtenidos en el experimento de la Figura 19. X,Y,Z representan las estimaciones previas al ajuste bundle e $X_{ba}$,$Y_{ba}$, $Z_{ba}$, las estimaciones tras el ajuste.*

| | X | Y | Z | $X_{ba}$ | $Y_{ba}$ | $Z_{ba}$ |
|---|---|---|---|---|---|---|
| Error medio (m) | 0.110 | 0.245 | 0.180 | 0.111 | 0.247 | 0.192 |
| Error máximo (m) | 0.323 | 0.968 | 0.444 | 0.324 | 0.967 | 0.444 |
| Error mínimo (m) | 0.003 | 0.005 | 0.012 | 0.001 | 0.001 | 0.009 |

En estos experimentos, la optimización mediante el ajuste *bundle* no ha supuesto mejoría, siendo los errores prácticamente iguales que en la estimación previa al ajuste. Esto es debido a que estamos trabajando mediante simulación con cámaras calibradas en un caso ideal. Por ello, los errores de reproyección ya son pequeños en un inicio y es difícil minimizarlos aún más. En un futuro, cuando se trabaje con cámaras reales, se espera que el ajuste *bundle* sí optimice la posición de los marcadores. Los errores obtenidos en estos experimentos se pueden asociar a errores en la detección de los marcadores en las imágenes.

### 3.4.4 LOCALIZACIÓN DEL ROBOT

Con el fin de evaluar el método de localización propuesto, se desarrolló una simulación en la que el robot realiza una trayectoria sobre la chapa, de dimensiones 14x5m, simulando inspeccionarla. Se distribuyeron 8 ArUcos de 17 cm de lado alrededor de la misma.

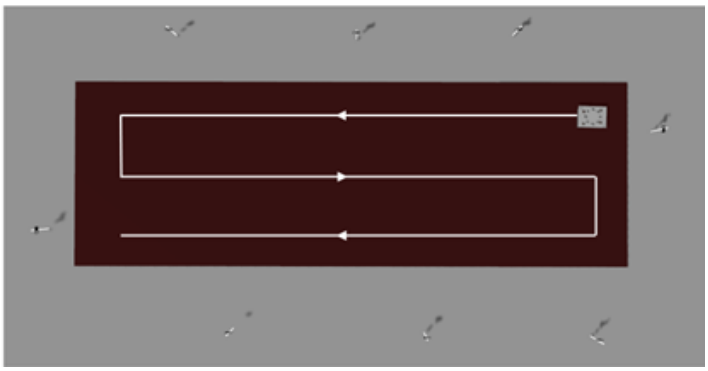

*Figura 21. Trayectoria seguida por el robot en la simulación.*

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 22.

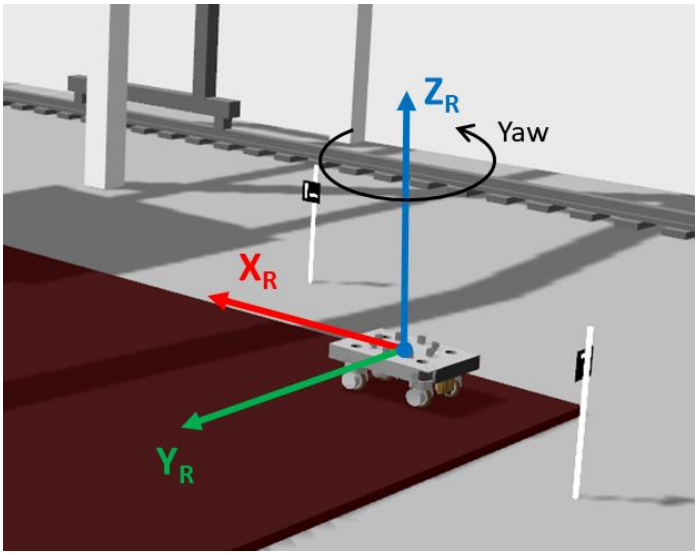


*Figura 22. Yaw: giro respecto del eje Z.*

A partir de la Figura 23 y los datos expuestos en la Tabla 4, se puede apreciar que la trayectoria estimada se ajusta bastante a la real. El máximo error alcanzado ha sido de 47 centímetros, sin embargo, los errores habituales son bastante más bajos, teniendo un error medio en la posición en X e Y de 14cm y 3cm respectivamente. Se puede observar que dónde mayor error hay en la trayectoria es en los extremos, precisamente donde hay un menor número de marcadores cercanos.

La orientación del robot también ha sido estimada de una manera bastante fiable, ver Figura 23. El error medio está en torno a 1.8º, siendo el máximo error igual a 6.2º.

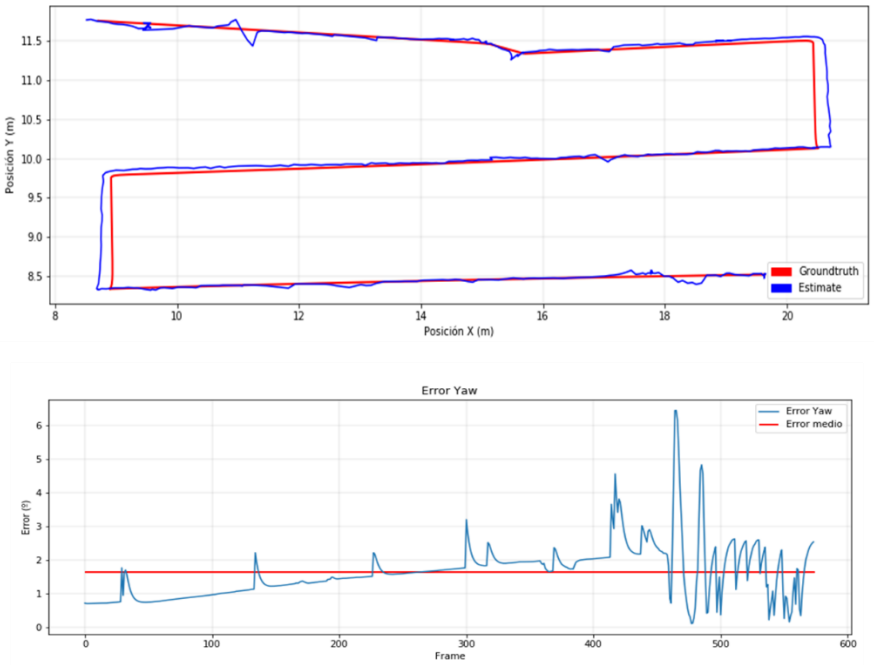


*Figura 23. Resultados. Arriba: Posición X-Y. Abajo: Error yaw.*

*Tabla 4. Resultados localización*

| | X | Y |
|---|---|---|
| Error medio (m) | 0.146 | 0.039 |
| Error máximo (m) | 0.479 | 0.270 |
| Error mínimo (m) | 0.0001 | 0.0001 |

Además, se realizaron otros experimentos en esta línea, alternado el número de marcadores presentes en el entorno. Se puede comprobar, según la Tabla 5, que a medida que aumenta el número de marcadores el error de posición disminuye.

*Tabla 5. Comparativa de errores medios según número de marcadores*

| | **4 ArUcos** | **6 ArUcos** | **8 ArUcos** |
|---|---|---|---|
| **Error medio de la posición (m)** | 0.109 | 0.095 | 0.083 |

## 3.5 ALTERNATIVAS DESCARTADAS

Previamente a la selección de la alternativa a implementar, se llevaron a cabo experimentos con dos librerías de odometría visual: *Fovis* [20] y *OpenCV RGB-Depth Processing* [21].

### 3.5.1 FOVIS

*Fovis* [67] 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 [15].

#### 3.5.1.1 Base teórica

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 [68], 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 [69], 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 [70].

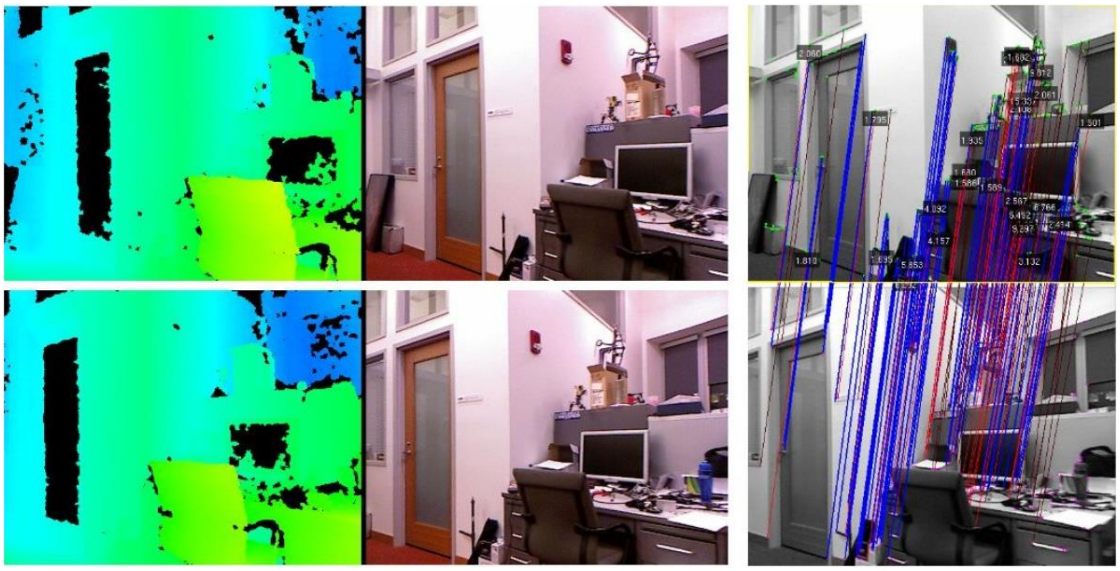

*Figura 24. La primera fila hace referencia al instante t y la inferior a t+Δt. La columna de la izquierda muestra las imágenes de profundidad y la del medio las imágenes RGBD. La columna de la derecha muestra las correspondencias entre puntos característicos, encontrados por el algoritmo FAST, emparejadas entre frames, donde los outliers eliminados son mostrados en color rojo. Imagen obtenida de [15].*

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 [71]. Después, la estimación es refinada minimizando el error de reproyección [72, Cap. 4.2.3] de los puntos característicos mediante un algoritmo de mínimos cuadrados no lineal [73]. 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.

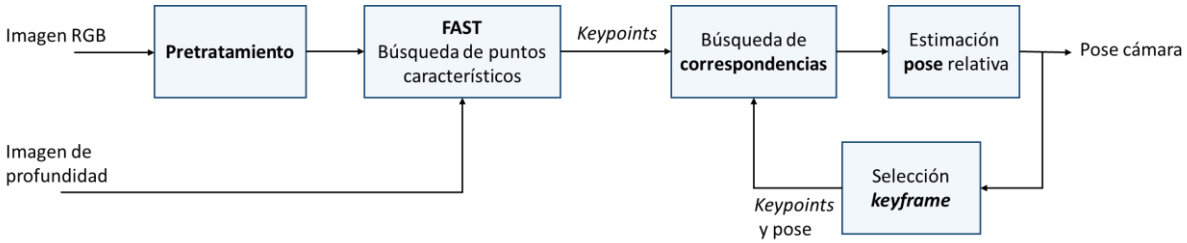


*Figura 25. Esquema interno del método seguido por la librería Fovis.*

### 3.5.1.2 Resultados

*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* [74], 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* [75].

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.

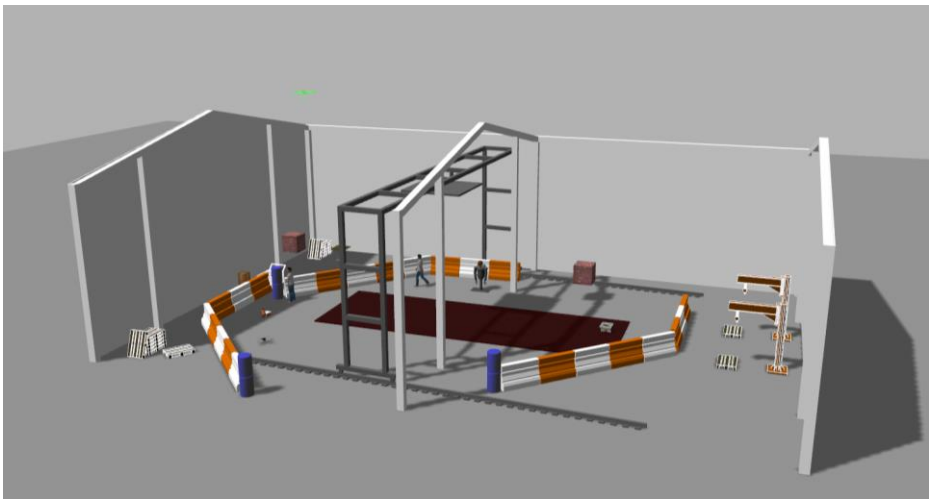

*Figura 26. Entorno sobre el que se realizaron los experimentos. El tamaño del recinto es, aproximadamente, de 30x20m.*

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 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.

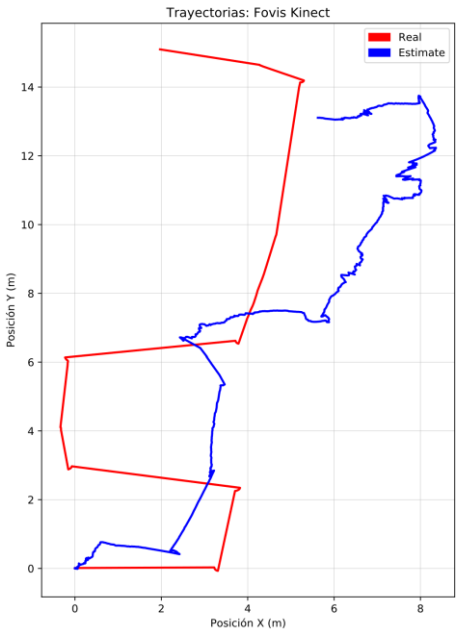


*Figura 27. Fovis: Trayectoria real (rojo) y estimada (azul) a partir de información RGB-D proporcionada por una Kinect situada sobre el robot. La trayectoria recorre la chapa incluida en el entorno mostrado en la Figura 26.*

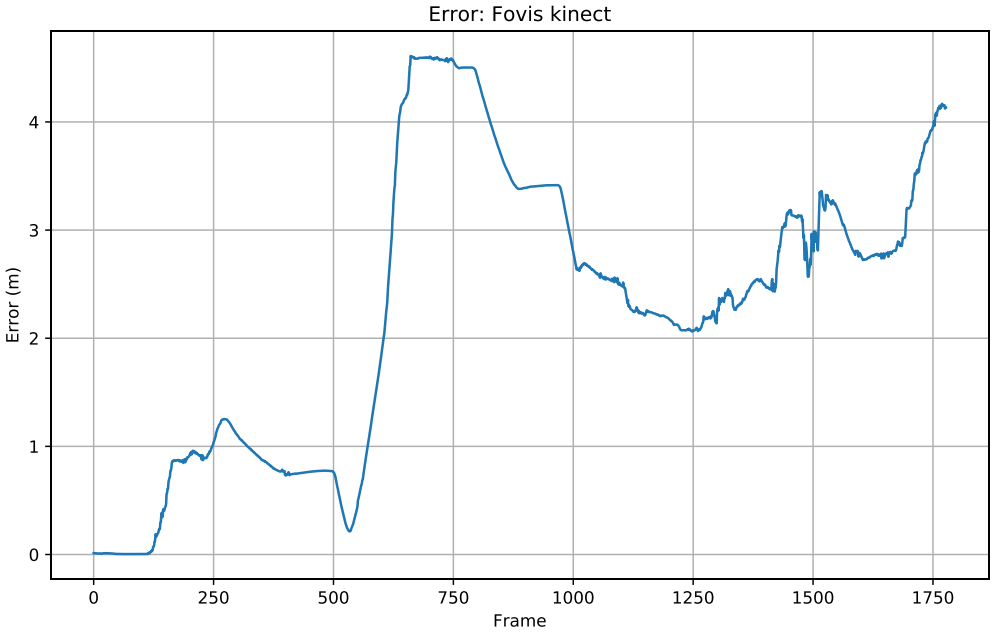


*Figura 28. Algoritmo Fovis para Kinect. Evolución del error de la posición a lo largo de los frames capturados.*

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 29se 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.

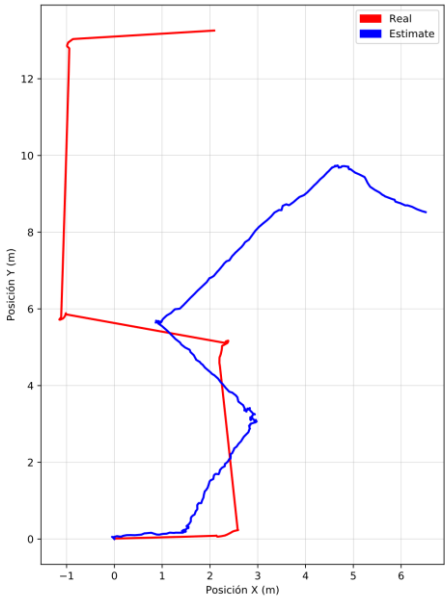


*Figura 29. Fovis: Trayectoria real (rojo) y estimada (azul) a partir de información proporcionada por un par estéreo situado sobre el robot. La trayectoria recorre la chapa incluida en el entorno mostrado en la Figura 26.*

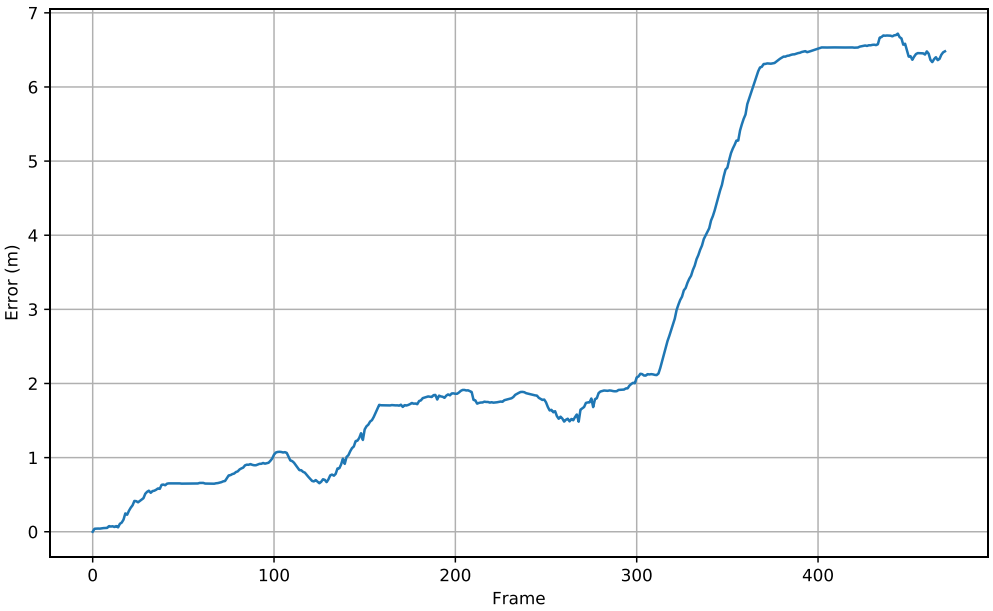


Figura 30. Algoritmo Fovis para un par estéreo. Evolución del error de la posición a lo largo de los *frames* capturados.

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 reducido de 10x10 metros.

Acorde con la Figura 32, 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.

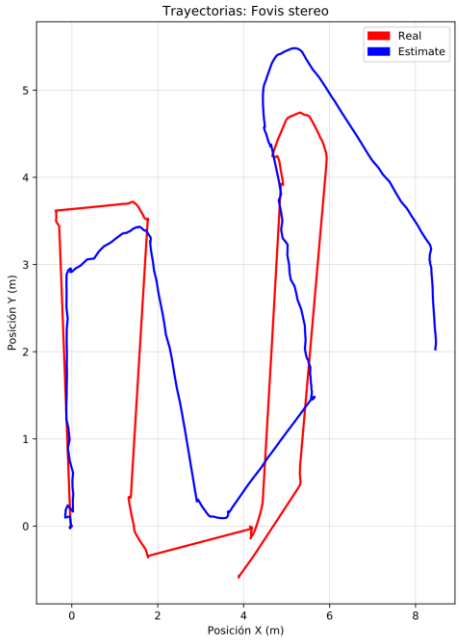


*Figura 32. Fovis: Trayectoria real (rojo) y estimada (azul) a partir de información proporcionada por un par estéreo situado sobre el robot. El robot hace un recorrido por el entorno reducido.*

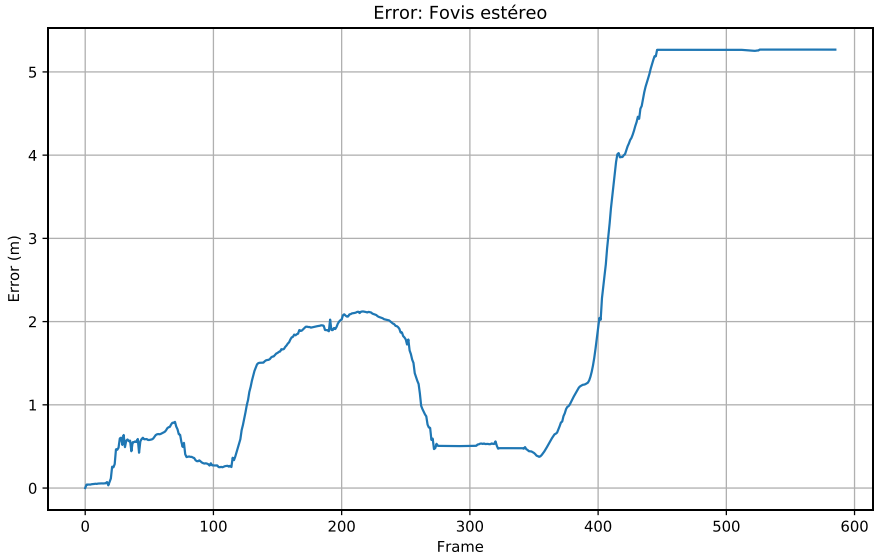


Figura 33. Algoritmo Fovis para un par estéreo en el entorno reducido. Evolución del error de la posición a lo largo de los *frames* capturados.

### 3.5.2 OpenCV RGB-Depth Processing

Maria Dimashova desarrolló el módulo RGB-Depth Processing [21] para la librería OpenCV [22]. 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 [23]. En la Figura 35 se ilustra el proceso seguido por el algoritmo implementado en el módulo de OpenCV [66].

#### 3.5.2.1 Base teórica

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 34.

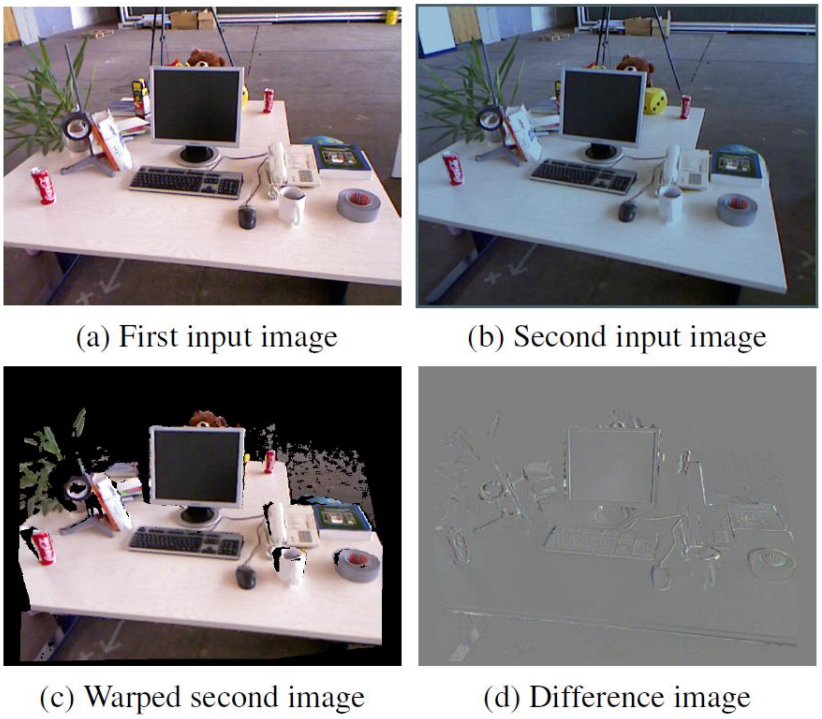


*Figura 34. Se propone una aproximación para estimar el movimiento de la cámara entre imágenes RGBD (a)+(b). La idea es encontrar una transformación rígida que transforme la segunda imagen en la primera (c), es decir, la imagen diferencia (d) calculada para localizaciones de profundidad confiable debe de ser cero (= gris). Imagen obtenida del artículo original del método [23].*

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 34.

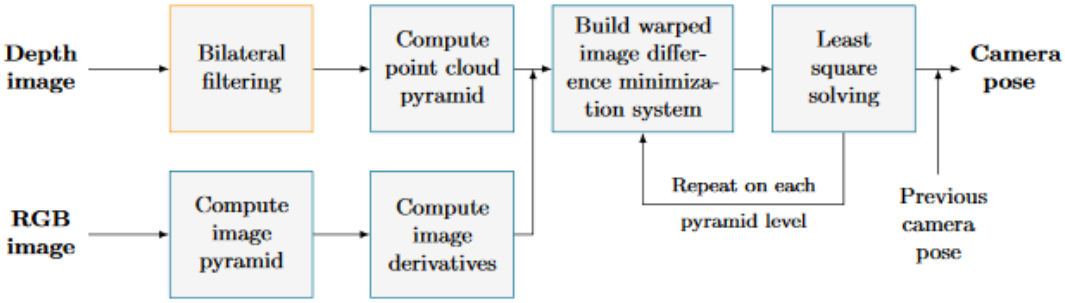


*Figura 35. Esquema interno del método seguido por el módulo OpenCV RGB-Depth processing. Imagen obtenida de [76].*

#### 3.5.2.2 Resultados

RGB-Depth Processing [21] es un módulo de la librería OpenCV [22] 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 [77], en concreto: *rgbd_dataset_freiburg2_pioneer_slam3* [78]. 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 36. Imágenes del dataset rdgd_dataset_freiburg2_pioneer_slam3. Izquierda: RGB. Derecha: Depth*

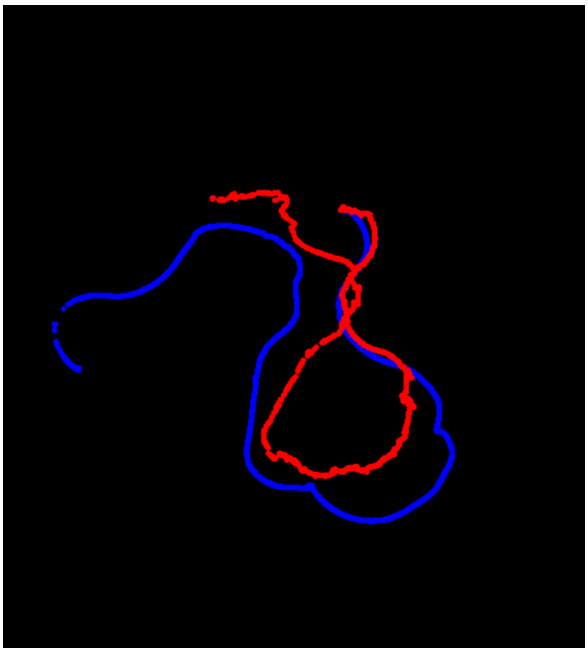

*Figura 37. Resultado obtenido mediante OpenCV a partir de las imágenes del dataset. Trayectoria Azul: Real, proporcionado por el dataset. Trayectoria roja: Estimada mediante el paquete RGB-Depth Processing de OpenCV. Dimensiones aproximadas del recorrido: 5x5 metros.*

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 37 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.

### 3.5.3 LOCALIZACIÓN DEL ROBOT: ESTRATEGIA ESTÉREO

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 utiliza un enfoque monocular estimando la pose a partir de una sola cámara y los marcadores detectados en la misma.

Además de esta estrategia, se realizaron pruebas utilizando una estrategia estéreo. Si se encuentra un mismo marcador en dos cámaras contiguas, se trabaja como si de un par estéreo se tratase.

#### 3.5.3.1 Base teórica

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.

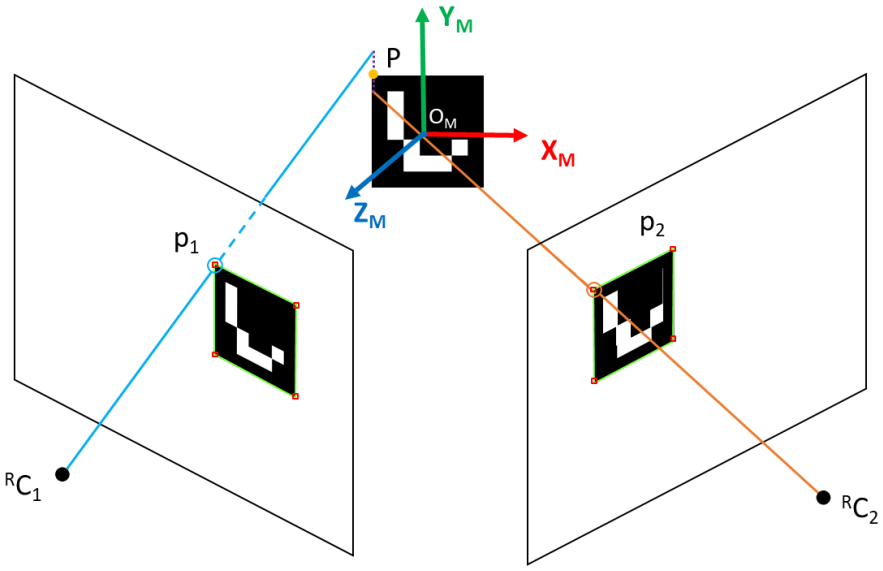


Figura 38. Triangulación a partir de la esquina superior izquierda de un mismo marcador presente en las imágenes de dos cámaras adyacentes.

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, ${}^RP_M$. 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, ${}^WP_R$. Para ello es necesario aplicar la transformación presente en la ecuación (17), realizando un cambio de sistema de la posición del robot respecto al sistema del marcador al sistema buscado.

$$^{W}P_R = {}^{W}T_M \cdot {}^{M}P_R \quad (17)$$

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.

$$^{M}P_R = -(^{M}R_R \cdot {}^{R}P_M) \quad (18)$$

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, $^{W}R_R$.

$$^{M}R_R = {}^{M}R_C \cdot {}^{C}R_R \quad (19)$$
$$^{W}R_R = {}^{W}R_M \cdot {}^{M}R_R \quad (20)$$

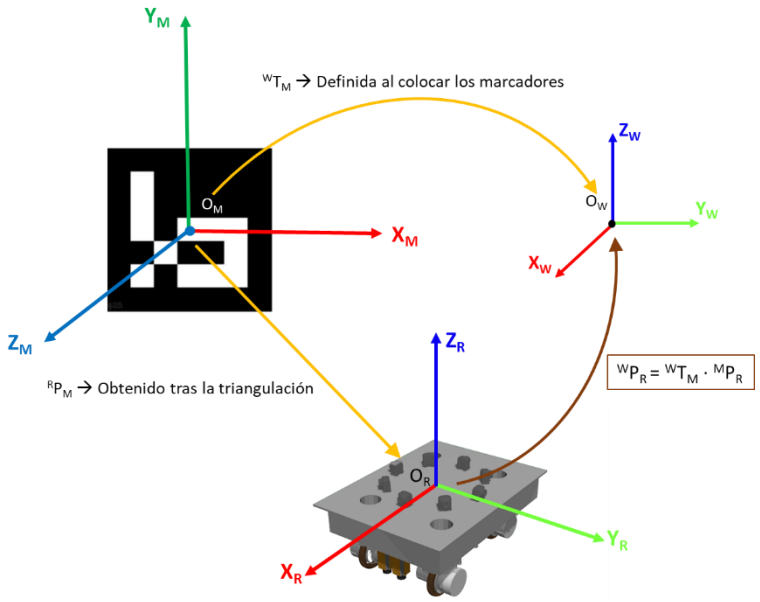


*Figura 39. Esquema del cálculo de la posición según la estrategia estéreo*

### *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 40.

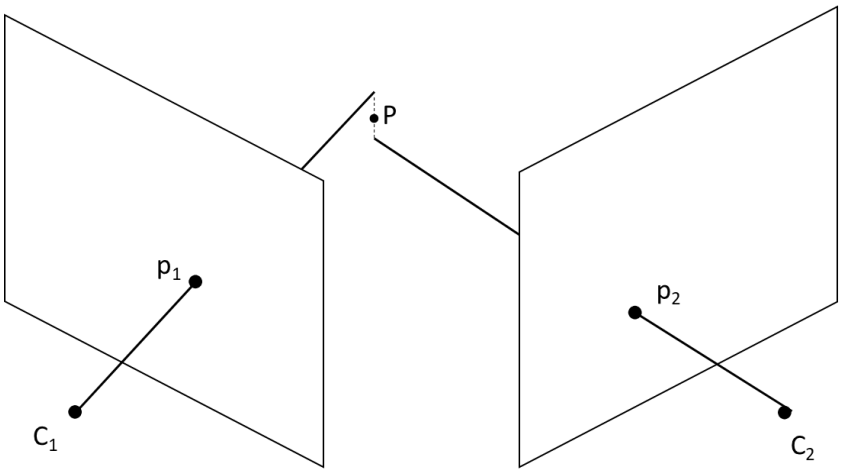


*Figura 40. En la realidad, los rayos reproyectados de $p_1$ y $p_2$ no llegan a cortarse en el espacio.*

Según el algoritmo de Hartley [79], 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 (21). 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.

$$\left.\begin{aligned}\widetilde{p_1} &= M_1 \cdot \tilde{P} \\ \widetilde{p_2} &= M_2 \cdot \tilde{P}\end{aligned}\right\} \tag{21}$$

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 (21) en un sistema de ecuaciones homogéneas de la forma Ax = 0 siguiendo los siguientes pasos.

Se parte del sistema de ecuaciones (21) y se generaliza para una cámara. Para plantear correctamente el nuevo sistema, es necesario deshacerse del factor de escala *n*.

$$\begin{bmatrix} nu \\ nv \\ n \end{bmatrix} = \begin{bmatrix} m_1^T \\ m_2^T \\ m_3^T \end{bmatrix} \cdot \tilde{P} \tag{22}$$

$$\left.\begin{aligned} nu &= m_1^T \cdot \tilde{P} \\ nv &= m_2^T \cdot \tilde{P} \\ n &= m_3^T \cdot \tilde{P} \end{aligned}\right\} \tag{23}$$

Sustituyendo el factor de escala y expresando las ecuaciones de manera matricial, se transforma el sistema (23) en el (25), obteniendo las dos ecuaciones que aporta cada cámara.

$$\left.\begin{matrix} u\, m_3^T\, \tilde{P} - m_1^T\, \tilde{P} = 0 \\ v\, m_3^T\, \tilde{P} - m_2^T\, \tilde{P} = 0 \end{matrix}\right\} \tag{24}$$

$$\begin{bmatrix} u\, m_3^T - m_1^T \\ v\, m_3^T - m_2^T \end{bmatrix} \tilde{P} = 0 \tag{25}$$

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 (26).

$$\begin{bmatrix} u_1\, {}^1m_3^T - {}^1m_1^T \\ v_1\, {}^1m_3^T - {}^1m_2^T \\ u_2\, {}^2m_3^T - {}^2m_1^T \\ v_2\, {}^2m_3^T - {}^2m_2^T \end{bmatrix} \tilde{P} = 0 \tag{26}$$

Se calcula $\tilde{P}$ mediante la aplicación de la descomposición SVD [80]. 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.

#### 3.5.3.2 Resultados

Previamente a la implementación del filtro de Kalman para eliminar valores atípicos, se realizaron pruebas con las dos estrategias de localización, monocular y estéreo, con el fin de escoger la mejor. Uno de los resultados se puede ver en la figura siguiente.

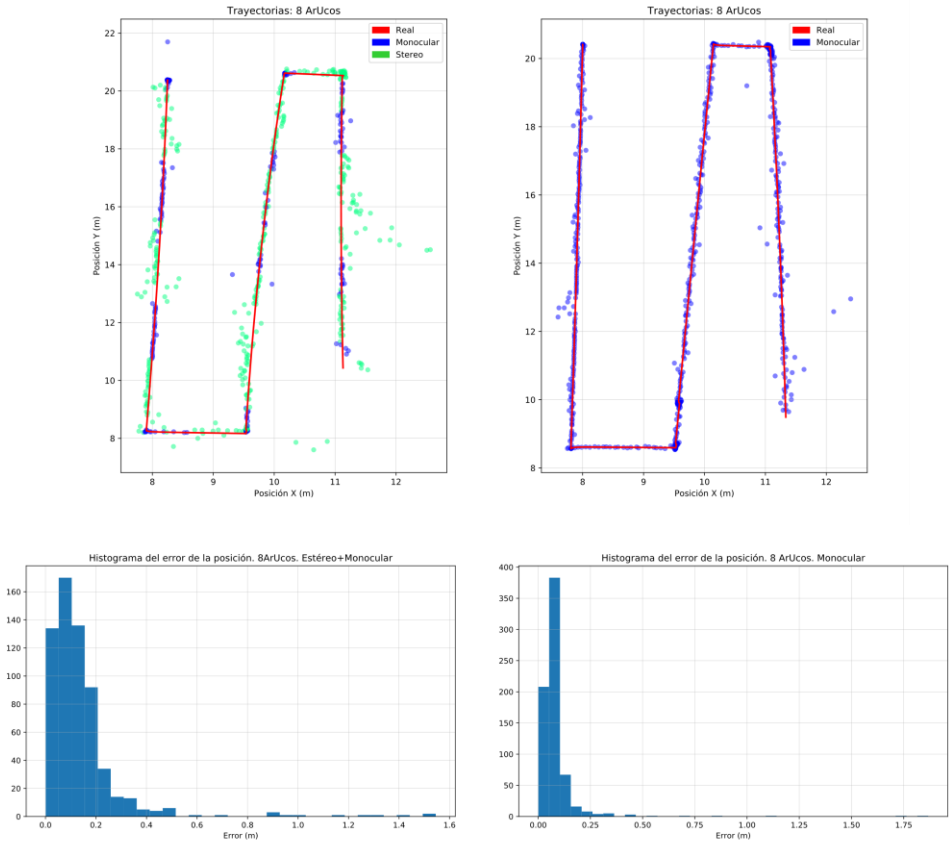


*Figura 41.* ***Arriba****: Trayectoria real (rojo) y estimada con 8 ArUcos de 17.78cm. Izquierda: En azul las posiciones estimadas según la estrategia estéreo y en verde, estrategia monocular. Derecha: Solo estrategia monocular. Los marcadores se encuentran aproximadamente a 2 metros de los bordes de la chapa.* ***Abajo****: Histogramas del error absoluto de la posición.*

Por un lado, se implementaron las dos estrategias conjuntamente, priorizando la estrategia estéreo cuando fuese posible. Por otro, únicamente la estrategia monocular. 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 menor que 10cm. En cuanto a la estrategia combinada, los errores se concentran en torno a los 20cm. Por este motivo, se escogió utilizar únicamente la estrategia monocular.

# 4 RECONSTRUCCIÓN 2D DE LA CHAPA Y EXTRACCIÓN DE DEFECTOS

Se propone un método que permite localizar los defectos marcados para su posterior reparación. Para ello, se realizará un panorama de la chapa a partir de la información proporcionada por la cámara RGBD dispuesta sobre el robot. En concreto se simuló una RealSense D435 [65].

Se supone que las zonas defectuosas habrán sido marcadas previamente, por ejemplo, con tiza, por un operador humano dotándoles de un contorno blanco en forma de polígono o circunferencia.

Para realizar el panorama, es necesario realizar un primer recorrido por la chapa que asegure que las imágenes tomadas cubren la totalidad de su área. Los píxeles recibidos deberán ser transformados para obtener su posición real y extraer los pertenecientes a la superficie a recorrer. Tras la aplicación de la homografía pertinente se dispone de un conjunto de imágenes que muestran distintas regiones de chapa en posición real. Se construirá un panorama de dichas imágenes a partir del cálculo de la traslación entre imágenes según un enfoque basado en *Template Matching*.

Obtenida ya la imagen panorámica, se buscan en ella los defectos marcados y se proporciona la localización de estos. Una vez hecho esto, el robot podrá dirigirse a dichas posiciones con el objetivo de realizar la subsanación del área indicada.

Para realizar el panorama, es necesario recorrer la chapa en diferentes sentidos para asegurar que las imágenes capturadas la cubren en su totalidad. Se empleará una cámara RGBD dispuesta en el robot. Esta cámara proporciona información RGB y de profundidad, clave para permitir la extracción del plano de la chapa.

## 4.1 EXTRACCIÓN PLANO DE LA CHAPA Y TRANSFORMACIÓN AL PLANO DE LA CÁMARA

Para localizar los defectos marcados, se realizará un panorama de la chapa a partir de la información proporcionada por la cámara RGBD.

Previamente a la realización del recorrido sobre la chapa, es preciso calcular la matriz de homografía que relaciona el plano del suelo y el de la imagen de la cámara con el fin de obtener las imágenes de la chapa en posición real. Para ello, se utilizará un marcador *ArUco* [81] dispuesto sobre el suelo.

Se captura una imagen en la que aparezca el *ArUco* y se buscan sus cuatro esquinas. Después, se calcula la homografía buscando la transformación de perspectiva que relaciona el plano en el que se encuentran las esquinas en dicha imagen con el plano imagen buscado. Se calcula resolviendo la ecuación (27) utilizando todos los pares de puntos y un esquema de minimización por mínimos cuadrados.

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{27}$$

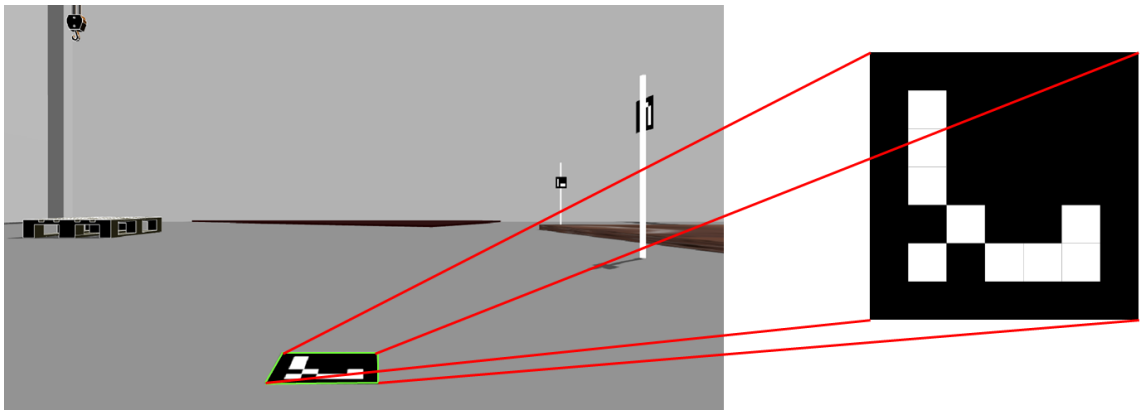

*Figura 42.Homografía buscada que transforma el plano del suelo al plano de la cámara.*

Una vez calculada la matriz de homografía, se realiza el recorrido que cubre la chapa. Durante el recorrido, se van capturando imágenes con la cámara. A partir de la información de profundidad, se extraen únicamente los píxeles que pertenecen al plano de la chapa y se les aplica la homografía calculada para obtener la parte de la chapa en posición real. Para identificar los puntos que pertenecen al suelo se utiliza la nube de puntos proporcionada por la cámara RGBD. Los que tienen una altura inferior a un valor determinado, dependiendo de la distancia a la que se encuentre la cámara desde el suelo, son seleccionados, descartando el resto de los puntos.

Después de la aplicación de la homografía hay un conjunto de imágenes que muestran áreas de la chapa sin distorsión de la perspectiva. Se construirá un panorama de estas imágenes a partir del cálculo de la traslación entre imágenes según un enfoque basado en el Template Matching [82].

## 4.2 CONSTRUCCIÓN PANORAMA

El *Template Matching* es un método para buscar la localización de una imagen modelo en otra imagen más grande. Para identificar la zona coincidente, la imagen modelo es comparada con la imagen de entrada deslizándola por la misma. Es decir, la imagen modelo se va moviendo un píxel cada vez dentro de la imagen general. En cada ubicación, se realiza una estimación de la fiabilidad de la localización según el grado de parecido entre el modelo y el área en cuestión.

Para estimar la fiabilidad del *matching*, se utilizó un método que emplea la diferencia al cuadrado normalizada. Dicho método está representado en la ecuación (28), donde $I$ es la imagen de entrada, $T$ la imagen modelo o *template* y $(x, y)$ las coordenadas de cada píxel en la imagen de entrada y $(x', y')$ las coordenadas de cada píxel en el *template*.

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}} \tag{28}$$

Una vez comparado el modelo en toda la imagen, las mejores correspondencias se traducen en máximos o mínimos según el método utilizado. En este caso, las mejores correspondencias serían mínimos, ya que la correspondencia ideal supondría el valor 0.

En este caso concreto, la técnica del *template matching* se utiliza para calcular los desplazamientos producidos entre instantes consecutivos. En la imagen obtenida en un instante *t* se busca la localización de una imagen modelo, constituida por un trozo de la imagen capturada en el instante *t+1*, ver Figura 43 y Figura 44.


*Figura 43. Imagen de entrada y template en un instante t*

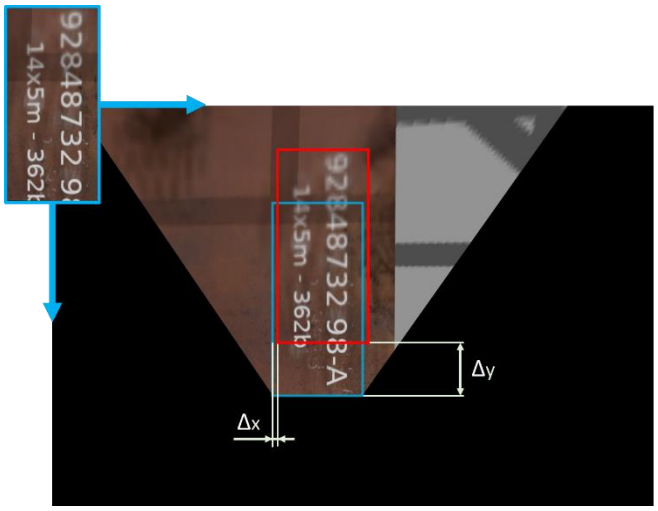


*Figura 44. Template Matching. El template en la esquina superior izquierda remarcado en azul. En rojo el template encontrado en la imagen de entrada*

El panorama se obtiene componiendo las imágenes de instantes consecutivos, de manera progresiva, gracias a los desplazamientos (Δx, Δy) obtenidos durante el *template matching.*

De esta manera se generan panoramas parciales según las trayectorias llevadas por el robot. Siguiendo el mismo esquema, se construye el panorama completo a partir de dichos panoramas parciales.

## 4.3 DETECCIÓN MARCAS DEFECTOS Y POLIGONALIZACIÓN

Una vez realizada la reconstrucción de la chapa se buscan los defectos marcados y se provee su localización, respecto a un sistema de referencia común localizado en una esquina de la chapa, al sistema de navegación.

Los defectos se encontrarían marcados con tiza blanca o similar, lo que proporciona un gran contraste entre las marcas y la chapa, facilitando su detección. Por ese motivo, se realizó un umbralizado [48, Cap. 5.1] para segmentar el color blanco correspondiente a las marcas.

Las marcas que delimitan los defectos pueden aparecer ligeramente abiertas, ya que son realizadas por un operario humano. En ese sentido, se aplica una operación morfológica de cierre [48, Sec. 11.3.4] para conseguir cerrar las formas blancas detectadas que se encuentren ligeramente abiertas.

Tras cerrar las formas que tienen pequeñas aperturas, se realiza una búsqueda de los contornos presentes en la imagen. Además, se hace una distinción entre contornos cerrados y abiertos. Estos últimos se suelen dar en los bordes de la chapa, considerando el propio borde la delimitación de la zona del defecto.

La búsqueda de los contornos se realiza aplicando el algoritmo de *Suzuki and Abe* [83]. Son divididos en contornos abiertos y cerrados.

Una vez detectados los contornos correspondientes a la señalización de los defectos, se aproxima cada uno de ellos a un polígono. Para ello, se utiliza el algoritmo de *Ramer-Douglas-Peucker* (RDP) [84]. Este algoritmo busca reducir el número de puntos utilizados en la aproximación de una curva.

Los contornos abiertos que se encuentran cerca de los bordes de la chapa son cerrados considerando el propio borde como parte del contorno. Ver Figura 45.

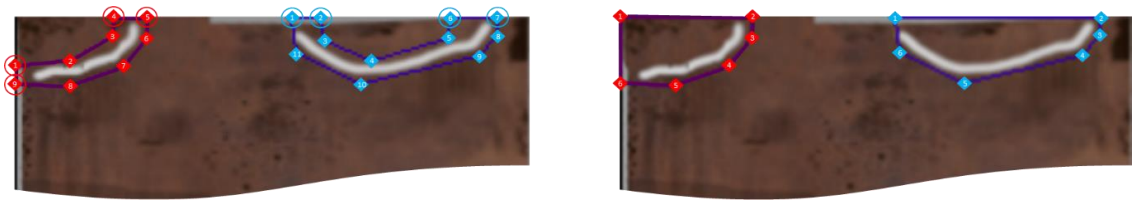

*Figura 45. Cierre de contornos abiertos en los bordes de la chapa*

Una vez calculados los polígonos, la localización de los puntos es convertida de píxeles a metros para facilitar la navegación. Conociendo el tamaño de la chapa en metros y en píxeles, a partir de la reconstrucción, el cálculo de la relación metros-píxeles es directa.

### 4.3.1 Algoritmo Ramer-Douglas-Peucker (RDP)

El objetivo del algoritmo es, dada una curva compuesta por segmentos, encontrar una curva similar con menos puntos. El algoritmo define una diferencia basada en la máxima distancia entre la curva original y la simplificada. La curva simplificada consiste en una reducción de los puntos que definían la original.

La curva original consiste en un conjunto de puntos ordenados. La aproximación se construye mediante un proceso recursivo. Inicialmente, se toma el segmento que une los dos puntos de los extremos de la curva y se busca el punto más alejado de dicho segmento, considerado el peor punto. Si ese punto está más cerca del segmento que un determinado umbral de distancia ε se termina el proceso y se descartan todos los puntos que están entre los dos extremos de la curva. En caso contrario, ese punto debe permanecer en la

simplificación de la curva. Se toman dos nuevas curvas, una con los puntos entre el primer y el peor punto y otra con los puntos entre el peor punto y el punto final de la curva y se repite el proceso anterior hasta finalizar. Cuando se completa la recursión la nueva curva puede ser generada a partir de los puntos que han permanecido tras haber aplicado el algoritmo. Un esquema del proceso se puede ver en la Figura 46.

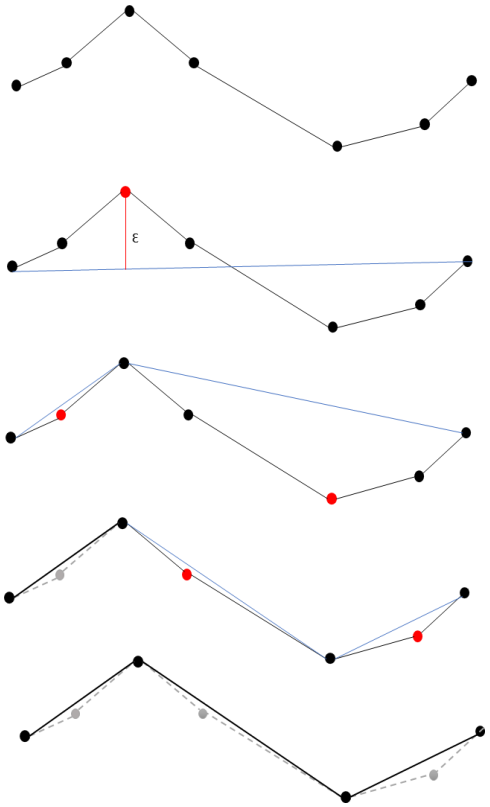


*Figura 46. Algoritmo Ramer-Douglas-Peucker (RDP)*

## 4.4 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*) [59], que proporciona las librerías y herramientas necesarias para implementar aplicaciones de robótica. En concreto, se utilizó la versión ROS Kinetic. Además, para el tratamiento de las imágenes se ha utilizado la librería de código abierto OpenCV [60], especializada en visión artificial.

Además, como se mencionó anteriormente, se utilizó el simulador Gazebo [61] con el fin de recrear un entorno industrial que pretende emular las instalaciones por las que se moverá el robot.

Algunos de 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* [62], una librería para *Python* especializada para manipulación y análisis de datos.

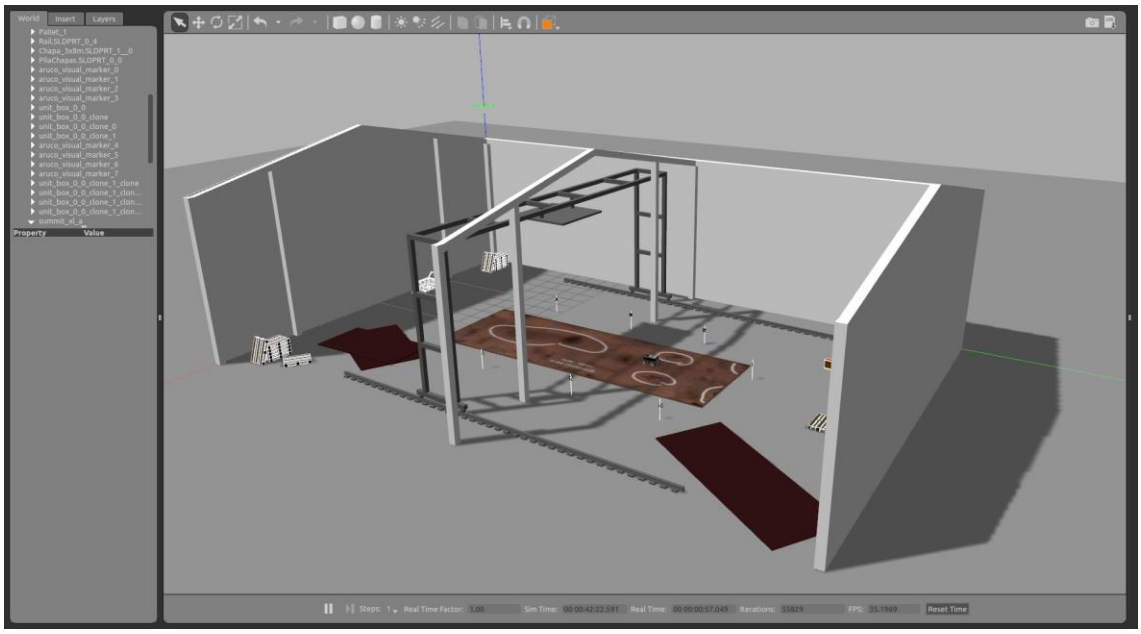

*Figura 47. Simulación en el entorno Gazebo de la industria y el robot dispuesto sobre la chapa a inspeccionar. La fábrica es de 30x20 metros y la chapa de acero de 14x5 metros.*

Para los experimentos se emplearon diferentes simulaciones en Gazebo, variando el número de marcadores y su posición, tomando como base la industria mostrada en la Figura 47, de dimensiones 30x20 metros. Esto se debió a que no pudimos disponer de un robot real. Las dimensiones de la chapa son de 14x5 metros.

En el entorno se incluye un robot móvil con una serie de sensores sobre el mismo. El modelo de robot utilizado fue el Summit-XL de Robotnik [64]. Para la tarea de localización se simuló un arco de 8 cámaras, mientras que para la reconstrucción de la chapa se simuló una cámara RGBD tipo RealSense [65].

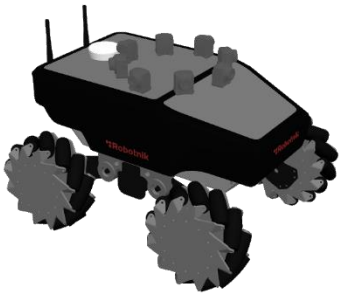

*Figura 48. Modelo de robot simulado en gazebo con diferentes sensores. Modelo Summit-XL*

A continuación, se exponen los resultados obtenidos durante la extracción de los polígonos que ajustan las marcas de los defectos señalados en la chapa.

En la Figura 49 se puede ver el proceso de captura de imágenes y transformación del plano del suelo al plano de la cámara. Primero se captura la imagen, después se filtran los puntos correspondientes al plano de la chapa y, finalmente, se aplica la homografía calculada para obtener la zona correspondiente en el plano de la cámara.

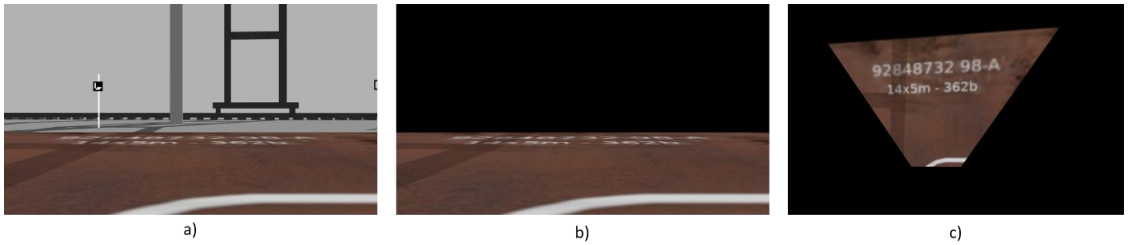


*Figura 49. proceso de captura de imágenes y transformación del plano del suelo al plano de la cámara. a) Imagen original. b) Segmentación del plano de la chapa. c) Plano de la chapa transformado al plano de la cámara.*

Una vez almacenas todas las imágenes transformadas, se construyen los panoramas parciales correspondientes a esas imágenes, obtenidas durante el trayecto del robot recorriendo la chapa en cada sentido. Posteriormente, a partir de dichos panoramas parciales, se construye el panorama completo. Los resultados se pueden ven en la Figura 50.

Se puede apreciar que el panorama reconstruido se ajusta bastante a la chapa original. Si bien se generaron ciertos desplazamientos al componer las imágenes, como se puede apreciar, sobre todo, en las marcas blancas que delimitan los defectos.

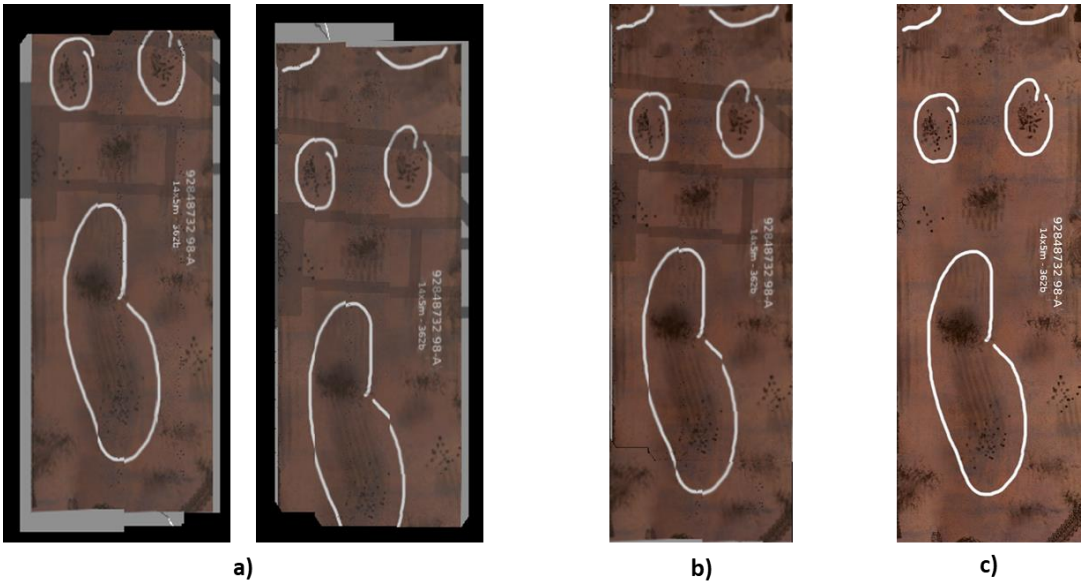


*Figura 50. Panorama completo (b) a partir de panoramas parciales (a) La imagen c) representa la chapa original, sin reconstruir.*

Una vez construido el panorama completo se buscan las marcas que delimitan los defectos y se calculan los polígonos que aproximan los contornos exteriores de dichas marcas. Los puntos de los polígonos se refieren a un sistema de coordenadas común con origen en una esquina de la chapa y en metros, y son proporcionados al módulo de navegación del robot.

Como se ve en la Figura 51, los polígonos se ajustan correctamente a los contornos blancos. Además, se corrigen correctamente las marcas que utilizan alguno de los bordes de la chapa como delimitación (b).

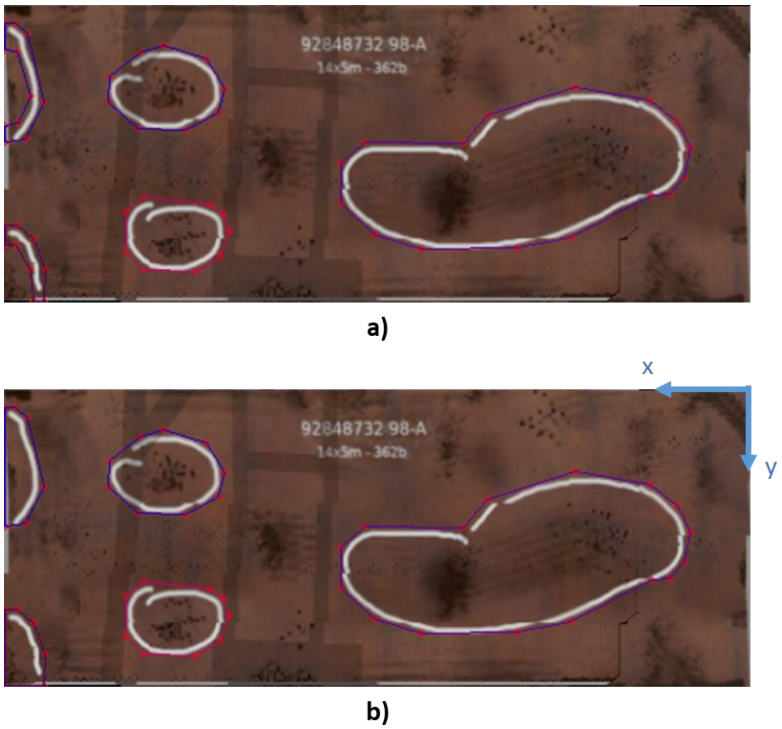


*Figura 51. Obtención de los polígonos que aproximan los defectos marcados. a) Previo a la consideración de los bordes de la chapa como limitación de las marcas. b) Resultado final.*

# 5 CALIBRACIÓN CÁMARA CON CHARUCO

El proceso de calibración de una cámara es necesario para la obtención de información 3D a partir de imágenes 2D de la escena. Existen diferentes técnicas para realizar una calibración. El objetivo de la misma es obtener los parámetros intrínsecos y extrínsecos de la cámara.

En este caso, para obtener los parámetros intrínsecos, se utilizó un *CharUco* [85], ver Figura 52.

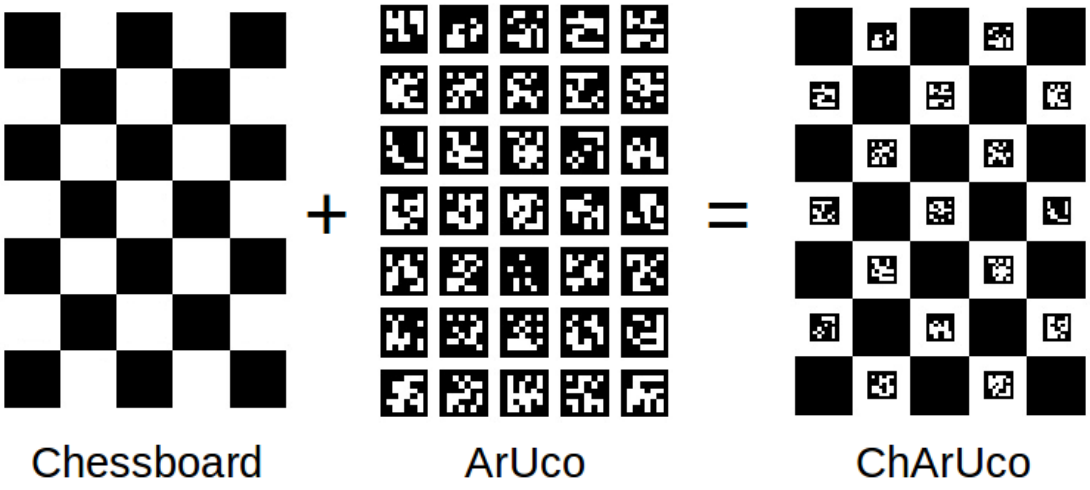


*Figura 52. Tabla CharUco.*

Los marcadores y tableros ArUco son muy útiles debido a su rápida detección y su versatilidad. Sin embargo, uno de los problemas de los marcadores ArUco es que la precisión de sus posiciones en las esquinas no es demasiado alta.

Por el contrario, las esquinas de los patrones del tablero de ajedrez pueden ser refinadas con mayor precisión ya que cada esquina está rodeada por dos cuadrados negros. Sin embargo, encontrar un patrón de tablero de ajedrez no es tan versátil como encontrar una tabla de ArUco: tiene que ser completamente visible y las oclusiones no están permitidas. Un tablero ChArUco trata de combinar los beneficios de estos dos enfoques.

Para calibrar usando un tablero de ChArUco, es necesario detectar el tablero desde diferentes puntos de vista, de la misma manera que la calibración estándar lo hace con el patrón tradicional del tablero de ajedrez. Sin embargo, debido a los beneficios del uso de ChArUco, se permiten oclusiones y vistas parciales, y no todas las esquinas deben ser visibles en todos los puntos de vista. En la Figura 53 se muestran ejemplos de imágenes utilizadas para la calibración.

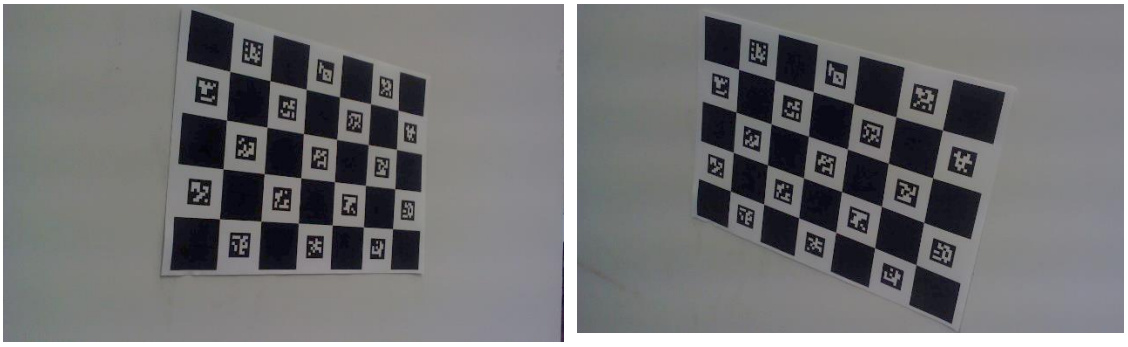

*Figura 53. Ejemplo de imágenes utilizadas para la calibración.*

Una vez capturadas las imágenes correspondientes, se realiza la calibración utilizando el módulo *aruco* [85] incluido en OpenCV.

Se ha desarrollado un código que permite realizar la calibración de una cámara a partir de las imágenes tomadas. Únicamente es necesario pasarle el número de imágenes que se emplearán para calibrar y la ruta en la que se encuentran.

# 6 SELECCIÓN DE HARDWARE

## 6.1 CÁMARAS MONOCULARES

Para formar el arco de cámaras se emplean cámaras monoculares. Hay múltiples alternativas en este campo.

*iDS Imaging* tiene un amplio catálogo de cámaras monoculares, los modelos uEye.
Hay modelos USB 2.0, USB 3.0, USB 3.1 y GigE. Se puede ver más información en [86].

*The Imaging Source* también tiene modelos de todos los tipos. USB 2.0, 3.0, 3.1 y GigE. Ver más en [87].

En este caso, se han simulado cámaras con las especificaciones del modelo *uEye XS* de *iDS Imaging*. Además, se ha realizado alguna prueba real con dicha cámara. En la

siguiente tabla se recoge alguna de sus especificaciones. Consultar sus especificaciones completas en [88].

*Tabla 6. Especificaciones cámara uEye XS.*

| | |
|---|---|
| **Modelo** | UI-1005XS-C Color |
| **Familia** | uEye XS |
| **Tamaño** | 23x26,5x21,5 mm |
| **Interfaz** | USB 2.0 |
| **Velocidad Máx.** | 30 – 99 fps |
| **Resolución** | 4-8 MP |
| **Tipo Sensor** | CMOS |

## 6.2 CÁMARAS RGBD

Para la reconstrucción 2D de la chapa y extracción de polígonos se emplean cámaras RGBD. Las mejores alternativas de cámaras RGBD son los modelos Astra de Orbbec y RealSense de Intel. Todos ellos compatibles con ROS Kinetic.

*Tabla 7. Cámaras RGBD*

| Dispositivo | Drivers ROS | Precio |
|---|---|---|
| Orbbec Astra [89] | Astra SDK, OpenNI 2 | 149 € |
| RealSense D435 [65] | Intel® RealSense™ SDK 2.0 | 179 € |
| RealSense D415 [90] | Intel® RealSense™ SDK 2.0 | 149 € |

| | |
|---|---|
| **N° Modelo** | ORBBEC |
| **Dimensiones** | 160 x 30 x 40 (mm) |
| **Peso** | 300 g |
| **Rango** | 0.4 -8 m |
| **Profundidad de Imagen** | 640*480 (VGA) 16bit @30 FPS |
| **Tamaño RGB** | 1280*960 @10FPS |
| **Campo de Visión** | 60° horiz. x 49.5 ° vert. (73° diagonal) |
| **Micrófonos** | 2 |
| **Sistema Operativo** | Windows, Linux, Android |

*Figura 54. Especificaciones Orbbec Astra.*

| | |
|---|---|
| **Use Environment** | Indoor/Outdoor |
| **Depth Technology** | Active IR Stereo (Global Shutter) |
| **Main Intel® RealSense™ component** | Intel® RealSense™ Vision Processor D4<br>Intel® RealSense™ module D430 |
| **Depth Field of View (FOV)—(Horizontal × Vertical × Diagonal)** | 86 x 57 x 94 (+/- 3°) |
| **Depth Stream Output Resolution** | Up to 1280 x 720 |
| **Depth Stream Output Frame Rate** | Up to 90 fps |
| **Minimum Depth Distance (Min-Z)** | 0.2m |
| **Sensor Shutter Type** | Global shutter |
| **Maximum Range** | Approx.10 meters; Varies depending on calibration, scene, and lighting condition |
| **RGB Sensor Resolution and Frame Rate** | 1920 x 1080 at 30 fps |
| **RGB Sensor FOV (Horizontal x Vertical x Diagonal)** | 69.4° x 42.5° x 77° (+/- 3°) |
| **Camera Dimension (Length x Depth x Height)** | 90 mm x 25 mm x 25 mm |
| **Connectors** | USB 3.0 Type - C |
| **Mounting Mechanism** | One 1/4-20 UNC thread mounting point<br>Two M3 thread mounting points |

*Figura 55. Especificaciones RealSense D435.*

| | |
|---|---|
| **Use Environment** | Indoor and Outdoor |
| **Depth Technology** | Active IR stereo |
| **Main Intel® RealSense™ component** | Intel® RealSense™ Vision Processor D4<br>Intel® RealSense™ module D410 |
| **Depth Field of View (FOV)—(Horizontal × Vertical × Diagonal)** | 69.4° x 42.5° x 77° (+/- 3°) |
| **Depth Stream Output Resolution** | Up to 1280 x 720 |
| **Depth Stream Output Frame Rate** | Up to 90 fps |
| **Minimum Depth Distance (Min-Z)** | 0.3m |
| **Maximum Range** | Approx. 10 meters; Varies depending on calibration, scene, and lighting condition |
| **RGB Sensor Resolution and Frame Rate** | 1920 x 1080 at 30 fps |
| **RGB Sensor FOV (Horizontal x Vertical x Diagonal)** | 69.4° x 42.5° x 77° (+/- 3°) |
| **Camera Dimension (Length x Depth x Height)** | 99 mm x 20 mm x 23 mm |
| **Connectors** | USB 3.0 Type - C |
| **Mounting Mechanism** | One 1/4-20 UNC thread mounting point<br>Two M3 thread mounting points |

*Figura 56. Especificaciones RealSense D415.*

# 7 REFERENCIAS


[1] «ROS.org | Powering the world's robots». .

[2] T. Söderström, «Using an Extended Kalman Filter for Rigid Body Pose Estimation», *Journal of Biomechanical Engineering*, vol. 127, n.º 3, p. 475, dic. 2004.

[3] F. Caron, M. Davy, E. Duflos, y P. Vanheeghe, «Particle Filtering for Multisensor Data Fusion With Switching Observation Models: Application to Land Vehicle Positioning», *IEEE Transactions on Signal Processing*, vol. 55, n.º 6, pp. 2703-2719, jun. 2007.

[4] M. Alatise y 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, n.º 10, p. 2164, sep. 2017.

[5] B. Triggs, P. F. McLauchlan, R. I. Hartley, y A. W. Fitzgibbon, «Bundle Adjustment — A Modern Synthesis», en *Vision Algorithms: Theory and Practice*, vol. 1883, B.

Triggs, A. Zisserman, y R. Szeliski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 298-372.
[6] K. Yousif, A. Bab-Hadiashar, y R. Hoseinnezhad, «An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics», *Intelligent Industrial Systems*, vol. 1, n.º 4, pp. 289-311, dic. 2015.
[7] H. P. Moravec, «Obstacle avoidance and navigation in the real world by a seeing robot rover.», STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1980.
[8] D. Nistér, O. Naroditsky, y J. Bergen, «Visual odometry», en *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, 2004, vol. 1, pp. I–I.
[9] Y. Cheng, M. Maimone, y L. Matthies, «Visual odometry on the Mars exploration rovers», en *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, 2005, vol. 1, pp. 903–910.
[10] «Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography», p. 42.
[11] D. Scaramuzza y R. Siegwart, «Monocular omnidirectional visual odometry for outdoor ground vehicles», en *International Conference on Computer Vision Systems*, 2008, pp. 206–215.
[12] A. I. Comport, E. Malis, y P. Rives, «Accurate Quadrifocal Tracking for Robust 3D Visual Odometry», en *Proceedings 2007 IEEE International Conference on Robotics and Automation*, Rome, Italy, 2007, pp. 40-45.
[13] «Kinect: desarrollo de aplicaciones de Windows». [En línea]. Disponible en: https://developer.microsoft.com/es-es/windows/kinect. [Accedido: 17-ene-2019].
[14] S. Wirth, P. L. Negre Carrasco, y G. O. Codina, «Visual odometry for autonomous underwater vehicles», en *2013 MTS/IEEE OCEANS - Bergen*, Bergen, 2013, pp. 1-6.
[15] A. S. Huang *et al.*, «Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera», en *Robotics Research*, vol. 100, H. I. Christensen y O. Khatib, Eds. Cham: Springer International Publishing, 2017, pp. 235-252.
[16] *Dense Visual Odometry. Contribute to tum-vision/dvo development by creating an account on GitHub*. TUM Computer Vision Group, 2019.
[17] C. Kerl, J. Sturm, y D. Cremers, «Robust odometry estimation for RGB-D cameras», en *2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, 2013, pp. 3748-3754.
[18] *A ROS wrapper for libviso2, a library for visual odometry: srv/viso2*. Systems, Robotics & Vision, University of the Balearic Islands, 2019.
[19] «Systems, Robotics & Vision Group | Universitat de les Illes Balears». .
[20] «fovis_ros - ROS Wiki». [En línea]. Disponible en: http://wiki.ros.org/fovis_ros?distro=hydro. [Accedido: 08-ene-2019].
[21] «OpenCV: RGB-Depth Processing». [En línea]. Disponible en: https://docs.opencv.org/3.4/d2/d3a/group__rgbd.html. [Accedido: 05-oct-2018].
[22] «OpenCV: OpenCV modules». [En línea]. Disponible en: https://docs.opencv.org/3.4/index.html. [Accedido: 05-oct-2018].
[23] F. Steinbrucker, J. Sturm, y D. Cremers, «Real-time visual odometry from dense RGB-D images», en *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, Barcelona, Spain, 2011, pp. 719-722.
[24] L. Kleeman, «Advanced sonar and odometry error modeling for simultaneous localisation and map building», en *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.

[25] F. Abrate, B. Bona, y M. Indri, «Experimental EKF-based SLAM for mini-rovers with IR sensors only», p. 6.
[26] D. M. Cole y P. M. Newman, «Using laser range data for 3D SLAM in outdoor environments», en *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, Orlando, FL, USA, 2006, pp. 1556-1563.
[27] A. J. Davison, «SLAM with a Single Camera».
[28] G. A. Terejanu, «Extended Kalman Filter Tutorial», p. 7.
[29] G. Klein y D. Murray, «Parallel Tracking and Mapping for Small AR Workspaces», en *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, Nara, Japan, 2007, pp. 1-10.
[30] «New package: (SVO) Semi-direct Monocular Visual Odometry - ROS robotics news». [En línea]. Disponible en: http://www.ros.org/news/2014/06/new-package-svo-semi-direct-monocular-visual-odometry.html. [Accedido: 31-ene-2019].
[31] C. Forster, M. Pizzoli, y D. Scaramuzza, «SVO: Fast semi-direct monocular visual odometry», en *2014 IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, 2014, pp. 15-22.
[32] R. Mur-Artal, J. M. M. Montiel, y J. D. Tardos, «ORB-SLAM: A Versatile and Accurate Monocular SLAM System», *IEEE Transactions on Robotics*, vol. 31, n.º 5, pp. 1147-1163, oct. 2015.
[33] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, y A. J. Davison, «SLAM++: Simultaneous Localisation and Mapping at the Level of Objects», en *2013 IEEE Conference on Computer Vision and Pattern Recognition*, Portland, OR, USA, 2013, pp. 1352-1359.
[34] K. Tateno, F. Tombari, y N. Navab, «When 2.5D is not enough: Simultaneous reconstruction, segmentation and recognition on dense SLAM», en *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 2295-2302.
[35] H. Kato y M. Billinghurst, «Marker tracking and HMD calibration for a video-based augmented reality conferencing system», en *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, San Francisco, CA, USA, 1999, pp. 85-94.
[36] M. Fiala, «Designing Highly Reliable Fiducial Markers», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, n.º 7, pp. 1317-1324, jul. 2010.
[37] A. Babinec, L. Jurišica, P. Hubinský, y F. Duchoň, «Visual Localization of Mobile Robot Using Artificial Markers», *Procedia Engineering*, vol. 96, pp. 1-9, 2014.
[38] M. Fiala, «ARTag, a Fiducial Marker System Using Digital Techniques», en *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA, 2005, vol. 2, pp. 590-596.
[39] D. Wagner y D. Schmalstieg, «ARToolKitPlus for Pose Tracking on Mobile Devices», p. 9.
[40] D. Flohr y J. Fischer, «A Lightweight ID-Based Extension for Marker Tracking Systems», p. 7.
[41] L. Naimark y E. Foxlin, «Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker», en *Proceedings. International Symposium on Mixed and Augmented Reality*, Darmstadt, Germany, 2002, pp. 27-36.
[42] «reacTIVision». [En línea]. Disponible en: http://reactivision.sourceforge.net/#usage. [Accedido: 01-feb-2019].
[43] M. Brown y D. G. Lowe, «Automatic Panoramic Image Stitching using Invariant Features», *Int J Comput Vision*, vol. 74, n.º 1, pp. 59-73, abr. 2007.

[44] M. Wang, S. Niu, y X. Yang, «A novel panoramic image stitching algorithm based on ORB», en *2017 International Conference on Applied System Innovation (ICASI)*, 2017, pp. 818-821.
[45] M. Alomran y D. Chai, «Feature-based panoramic image stitching», en *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2016, pp. 1-6.
[46] S. Ji, D. Yu, Y. Hong, y M. Lu, «Template Matching for Wide-Baseline Panoramic Images from a Vehicle-Borne Multi-Camera Rig», *IJGI*, vol. 7, n.º 7, p. 236, jun. 2018.
[47] «ArUco: a minimal library for Augmented Reality applications based on OpenCV | Aplicaciones de la Visión Artificial». [En línea]. Disponible en: https://www.uco.es/investiga/grupos/ava/node/26. [Accedido: 27-nov-2018].
[48] M. Sonka, R. Boyle, y V. Hlavac, *Image Processing, Analysis, and Machine Vision*, 4th ed. .
[49] Suzuki, Satoshi & be, KeiichiA., «Topological structural analysis of digitized binary images by border following», *Computer Vision, Graphics, and Image Processing*, 1985.
[50] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, y M. J. Marín-Jiménez, «Automatic generation and detection of highly reliable fiducial markers under occlusion», *Pattern Recognition*, vol. 47, n.º 6, pp. 2280-2292, jun. 2014.
[51] N. Otsu, «A Threshold Selection Method from Gray-Level Histograms», *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, n.º 1, pp. 62-66, ene. 1979.
[52] «aruco_detect - ROS Wiki». [En línea]. Disponible en: http://wiki.ros.org/aruco_detect. [Accedido: 27-nov-2018].
[53] X. X. Lu, «A Review of Solutions for Perspective-n-Point Problem in Camera Pose Estimation», *J. Phys.: Conf. Ser.*, vol. 1087, p. 052009, sep. 2018.
[54] A. Ranganathan, «The Levenberg-Marquardt Algorithm», p. 5.
[55] G. H. Seedahmed y T. Schenk, «DIRECT LINEAR TRANSFORMATION IN THE CONTEXT OF DIFFERENT SCALING CRITERIA», *Unpublished*, 2001.
[56] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, y Hang-Fei Cheng, «Complete solution classification for the perspective-three-point problem», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, n.º 8, pp. 930-943, ago. 2003.
[57] V. Lepetit, F. Moreno-Noguer, y P. Fua, «EPnP: An Accurate O(n) Solution to the PnP Problem», *International Journal of Computer Vision*, vol. 81, n.º 2, pp. 155-166, feb. 2009.
[58] E. R. Pi, «Implementation of a 3D pose estimation algorithm», p. 67.
[59] «ROS.org | Powering the world's robots». .
[60] «OpenCV». [En línea]. Disponible en: https://opencv.org/. [Accedido: 16-jul-2019].
[61] «Gazebo». [En línea]. Disponible en: http://gazebosim.org/. [Accedido: 10-dic-2018].
[62] «Python Data Analysis Library — pandas: Python Data Analysis Library». [En línea]. Disponible en: https://pandas.pydata.org/. [Accedido: 02-feb-2019].
[63] «Holonomic (robotics)», *Wikipedia*. 19-dic-2018.
[64] «SUMMIT-XL», *Robotnik*. .
[65] «Depth Camera D435 – Intel® RealSense™ Depth and Tracking Cameras». [En línea]. Disponible en: https://www.intelrealsense.com/depth-camera-d435/. [Accedido: 23-jul-2019].
[66] *Repository for OpenCV's extra modules. Contribute to opencv/opencv_contrib development by creating an account on GitHub*. OpenCV, 2019.

[67] «libfovis: FOVIS». [En línea]. Disponible en: https://fovis.github.io/. [Accedido: 09-oct-2018].
[68] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, y J. M. Ogden, «Pyramid methods in image processing», p. 9, 1984.
[69] «FAST Algorithm for Corner Detection — OpenCV 3.0.0-dev documentation». [En línea]. Disponible en: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html. [Accedido: 09-oct-2018].
[70] Howard, Andrew, «Real-Time Stereo Visual Odometry for Autonomous Ground Vehicles», 2008, pp. 3946-3952.
[71] B. K. P. Horn, «Closed-form solution of absolute orientation using unit quaternions», *Journal of the Optical Society of America A*, vol. 4, n.º 4, p. 629, abr. 1987.
[72] R. Hartley y A. Zisserman, *Multiple View Geometry in Computer Vision.* West Nyack: Cambridge University Press, 2004.
[73] E. Malis, «Improving vision-based control using efficient second-order minimization techniques», en *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, New Orleans, LA, USA, 2004, pp. 1843-1848 Vol.2.
[74] «Gazebo : Tutorial : ROS Depth Camera Integration». [En línea]. Disponible en: http://gazebosim.org/tutorials?tut=ros_depth_camera&cat=connect_ros. [Accedido: 12-feb-2019].
[75] «Gazebo : Tutorial : Gazebo plugins in ROS». [En línea]. Disponible en: http://gazebosim.org/tutorials?tut=ros_gzplugins#Multicamera. [Accedido: 12-feb-2019].
[76] V. Angladon *et al.*, «An evaluation of real-time RGB-D visual odometry algorithms on mobile devices», *Journal of Real-Time Image Processing*, feb. 2017.
[77] «The Entrepreneurial University - TUM». [En línea]. Disponible en: https://www.tum.de/. [Accedido: 31-ene-2019].
[78] «Computer Vision Group - Dataset Download». [En línea]. Disponible en: https://vision.in.tum.de/data/datasets/rgbd-dataset/download. [Accedido: 31-ene-2019].
[79] Richard I. Hartley, PeterSturm, «Triangulation». [En línea]. Disponible en: https://users.cecs.anu.edu.au/~hartley/Papers/triangulation/triangulation.pdf. [Accedido: 12-feb-2019].
[80] «Descomposición en Valores Singulares (SVD)», 2010.
[81] «ArUco: a minimal library for Augmented Reality applications based on OpenCV | Aplicaciones de la Visión Artificial». [En línea]. Disponible en: https://www.uco.es/investiga/grupos/ava/node/26. [Accedido: 15-jul-2019].
[82] N. Perveen, D. Kumar, y I. Bhardwaj, «An Overview on Template Matching Methodologies and its Applications», vol. 2, n.º 10, p. 8.
[83] S. Suzuki, «Topological Structural Analysis of Digitized Binary Images by Border Following», p. 15.
[84] T. K. P. DAVID H DOUGLAS, «ALGORITHMS FOR THE REDUCTION OF THE NUMBER OF POINTS REQUIRED TO REPRESENT A DIGITIZED LINE OR ITS CARICATURE», vol. 10, n.º Issue 2, pp. 112-122, dic. 1973.
[85] «OpenCV: Detection of ChArUco Corners». [En línea]. Disponible en: https://docs.opencv.org/3.4/df/d4a/tutorial_charuco_detection.html. [Accedido: 25-sep-2019].
[86] «Buscar cámara - encontrará cámaras industriales». [En línea]. Disponible en: https://es.ids-imaging.com/store/products/cameras.html. [Accedido: 26-sep-2019].

[87] «USB 3.1, USB 3.0, USB 3.0 Polarsens, USB 2.0, GigE, GigE Polarsens Industrial cameras». [En línea]. Disponible en: https://www.theimagingsource.com/products/industrial-cameras/. [Accedido: 26-sep-2019].
[88] «uEye 1005XS», *Infaimon*. .
[89] «Astra Series – Orbbec». [En línea]. Disponible en: https://orbbec3d.com/product-astra-pro/. [Accedido: 26-sep-2019].
[90] «Depth Camera D415», *Intel® RealSense™ Depth and Tracking Cameras*. [En línea]. Disponible en: https://www.intelrealsense.com/depth-camera-d415/. [Accedido: 26-sep-2019].