COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

NAVEGACIÓN AUTÓNOMA DE UN VEHÍCULO TERRESTRE USANDO MARCADORES FIDUCIALES

Autor: Alejandro Ucelay Jiménez

Director: Juan Luis Zamora Macho

Co-Director: Jaime Boal Martín-Larrauri

Madrid

Julio 2022

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Navegación autónoma de un vehículo terrestre usando marcadores fiduciales

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2021/2022 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada

de otros documentos está debidamente referenciada.

Firmado por UCELAY JIMENEZ, ALEJANDRO (AUTENTICACIÓN) el día 13/07/2022 con un certificado emitido por AC DNIE 006

Fdo.: Alejandro Ucelay Jiménez Fecha: 13 / 07 / 2022

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Juan Luis Zamora Macho Fecha: 13 / 07 / 2022

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Navegación autónoma de un vehículo terrestre usando marcadores fiduciales

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2021/2022 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada

de otros documentos está debidamente referenciada.

Firmado por
UCELAY JIMENEZ,
ALEJANDRO
(AUTENTICACIÓN)
el día 13/07/2022
con un
certificado
emitido por AC

Fdo.: Alejandro Ucelay Jiménez Fecha: 13 / 07 / 2022

Autorizada la entrega del proyecto
EL CO-DIRECTOR DEL PROYECTO

Firmado digitalmente por
BOAL MARTIN LARRAURI
JAIME - 05304600H
Fecha: 2022.07.13
10:09:07 +02'00'

Fdo.: Jaime Boal Martín-Larrauri Fecha: 13 / 07 / 2022

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

NAVEGACIÓN AUTÓNOMA DE UN VEHÍCULO TERRESTRE USANDO MARCADORES FIDUCIALES

Autor: Alejandro Ucelay Jiménez

Director: Juan Luis Zamora Macho

Co-Director: Jaime Boal Martín-Larrauri

Madrid

Julio 2022

DEDICATORIA

Este trabajo de fin de grado lo dedico con todo cariño a mis padres Fernando y Alejandra por su sacrificio y esfuerzo, por creer en mi capacidad, por darme la oportunidad de poder estudiar lo que realmente me apasiona y por su incondicional apoyo durante estos cuatro años de grado.

A mis hermanos Fernando y Gadea, por ser mi fuente de inspiración y motivación para poder superarme y dar la mejor versión de mí mismo cada día.

AGRADECIMIENTOS

Quiero agradecer a mis tutores Juan Luis Zamora Macho y Jaime Boal Martín-Larrauri por todo el esfuerzo y toda la ayuda durante la elaboración de este proyecto. Gracias por vuestro apoyo, comprensión, paciencia y por haber tomado vuestro tiempo en resolver cualquier duda que me inquietaba. Gracias por haberme dado la oportunidad de realizar este proyecto a vuestro lado y por haber compartido vuestro conocimiento conmigo.

Quiero agradecer a Antonio Martín Jiménez por ayudarme con el montaje del vehículo y por proporcionarme cualquier dispositivo electrónico necesario para la elaboración del proyecto.

Agradecer también a todo el profesorado de ICAI que ha colaborado en mi formación académica durante estos cuatro años de grado.

NAVEGACIÓN AUTÓNOMA DE UN VEHÍCULO TERRESTRE USANDO MARCADORES FIDUCIALES

Autor: Ucelay Jiménez, Alejandro.
Director: Zamora Macho, Juan Luis.
Co-Director: Boal Martín-Larrauri, Jaime.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

En este proyecto se ha desarrollado un sistema de navegación autónoma para robots móviles terrestres en entornos físicos cerrados. El sistema de navegación incluye el algoritmo de localización del vehículo a través de un sistema de detección de marcadores fiduciales mediante visión artificial.

Palabras clave: Marcadores fiduciales, sistema de cámaras Optitrack, robot móvil, navegación autónoma.

1. Introducción

El objetivo principal de este proyecto es el diseño de un sistema de localización basado en la detección de marcadores fiduciales. Este sistema de posicionamiento se ha integrado en un control de navegación para el vehículo terrestre utilizado por el departamento de Electrónica, Automática y Comunicaciones de la Escuela Superior de Ingeniería ICAI.

Se ha valorado la eficacia y precisión de este sistema de marcadores haciendo uso de un sistema de posicionamiento externo compuesto por un conjunto de cámaras infrarrojas. Además, el sistema de navegación consta con una arquitectura de comunicaciones entre los diferentes dispositivos *hardware* y *software* del proyecto.

2. Definición del proyecto

La electrónica del vehículo está compuesta por una Raspberry Pi 3B+. La Raspberry ejerce de ordenador y recibe las medidas de la IMU y del resto de sensores. Se encarga de las tareas de alto nivel como la gestión de las comunicaciones [1]. A su vez el vehículo cuenta con una Unidad de Medición Inercial (IMU) que se encuentra constituida por un acelerómetro y un giróscopo [2]. Entre los principales actuadores del vehículo se destacan los dos motores DC (EMG30) que se

controlan a través de señales PWM. En cada eje de cada motor se localiza un encoder magnético que se encarga de obtener la información de velocidad y sentido de giro de cada motor.

El algoritmo de detección de marcadores fiduciales se encargará de obtener la posición y orientación del vehículo en el entorno de trabajo. Este algoritmo se basa en un sistema de visión artificial. Es por ello que se ha empleado el sensor óptico *Raspberry Pi Camera Module V2* y la librería OpenCV para el procesamiento de las imágenes [3]. Además, se han empleado los marcadores de la familia ArUco como marcadores fiduciales.

Por último, el sistema de cámaras Optitrack se trata de un sistema capturador de movimiento que mediante un conjunto de cámaras infrarrojas permite obtener la posición y orientación del vehículo en el entorno de trabajo [4]. Este sistema externo de localización se ha empleado para caracterizar la precisión del algoritmo de detección de marcadores ArUco. En la *Figura 1,* se muestra un esquema de los diferentes dispositivos *hardware* del proyecto.

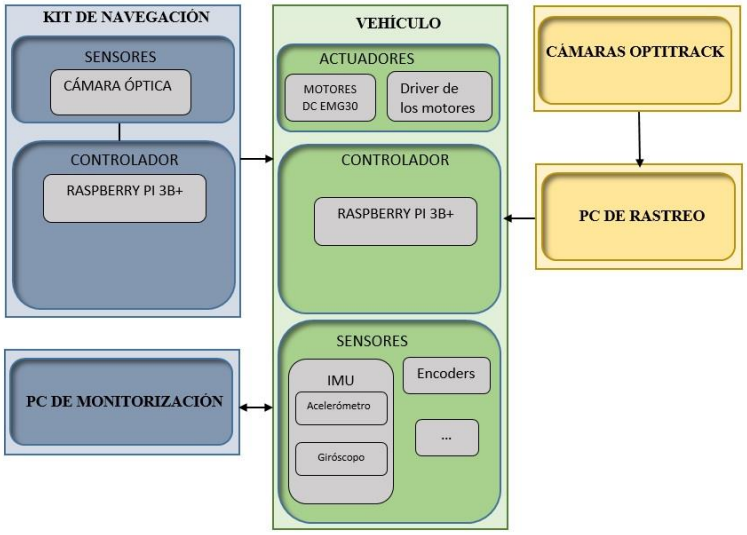


Figura 1: Arquitectura del hardware del proyecto.

3. Solución

El sistema de posicionamiento y orientación del vehículo basado en marcadores ArUco se trata de una aplicación de visión artificial que detecta marcadores ArUco en cada imagen capturada por la cámara. Para ello se ha empleado la librería OpenCV que cuenta con un módulo llamado `aruco` que permite la detección de marcadores, así como la obtención de la posición y orientación del

sensor óptico respecto al marcador detectado. El algoritmo de posicionamiento se ha programado en Python 3.10.

En segundo lugar, ha sido necesario establecer una arquitectura de comunicaciones entre los siguientes dispositivos *hardware* del proyecto: PC de monitorización, PC de rastreo de las cámaras Optitrack, Raspberry Pi 3B+ del vehículo y Raspberry Pi 3B+ del algoritmo de detección de marcadores. Para ello se han empleado diferentes protocolos de comunicación como UDP, TCP/IP o Bluetooth. A su vez, se ha empleado ROS2 (*Robot Operating System)* como sistema de comunicación entre ambas Raspberries del proyecto.

Para integrar el sistema de localización en un control de navegación, se han empleado tres controles de velocidad de avance, velocidad de giro, y ángulo de guiñada.

4. Resultados

Se ha comparado la posición y orientación del vehículo obtenida a través del algoritmo de detección de marcadores con la localización proporcionada por el sistema externo de cámaras Optitrack. A continuación, se muestran diferentes gráficas donde se caracteriza la precisión del algoritmo para el movimiento del vehículo a lo largo del eje *X*, *Y* y el ángulo del vehículo sobre su propio eje (*Z*). (*Figura 2).*

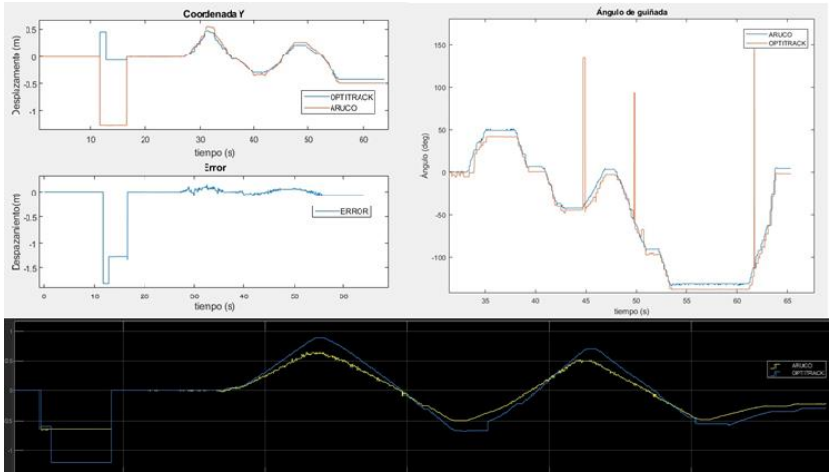


Figura 2: Caracterización de la precisión del sistema de marcadores.

A su vez, se muestran los resultados de la integración del sistema de posicionamiento en un control de navegación sobre el ángulo de guiñada del vehículo. Se han empleado controles tanto de tipo P como de tipo PID. (*Figura 3*).

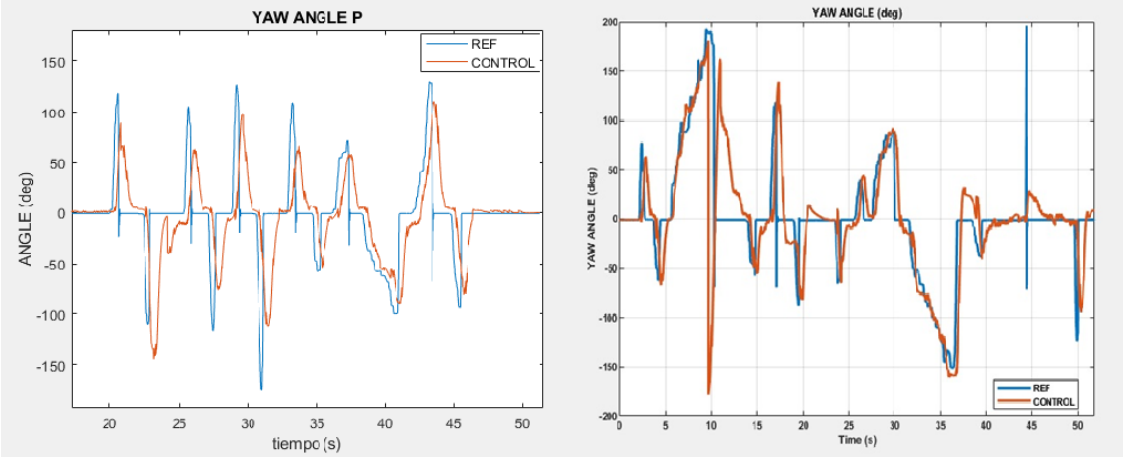


Figura 3: Controles P y PID del ángulo de guiñada del vehículo.

5. Conclusiones

Se puede concluir que esta tecnología de detección de marcadores es competente con otras tecnologías de posicionamiento, ya que como cualquier aplicación en ingeniería, es un compromiso entre prestaciones, precio y la casuística del proyecto en cuestión. La precisión de esta tecnología depende de la distancia entre la cámara óptica y el marcador fiducial. No obstante, esta limitación se puede resolver aumentando el tamaño del marcador o bien posicionando un mayor número de marcadores y distanciándolos entre sí para abarcar un mayor espacio de navegación.

A su vez, la precisión de este algoritmo de detección de marcadores se encuentra sujeta a factores como una correcta calibración de la cámara óptica o una favorable iluminación del entorno físico.

6. Referencias

[1] R. P. Ltd, «Buy a Raspberry Pi 3 Model B+», *Raspberry Pi*. https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/ (accedido 4 de julio de 2022).

[2] «Unidad de medición inercial», *Wikipedia, la enciclopedia libre*. 2 de julio de 2021. Accedido: 4 de julio de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Unidad_de_medici%C3%B3n_inercial&oldid=136743735

[3] «openCv», *OpenCV*. https://opencv.org/ (accedido 4 de julio de 2022).

[4] «Motion Capture Systems», *OptiTrack*. http://optitrack.com/index.html (accedido 4 de julio de 2022).

NAVEGACIÓN AUTÓNOMA DE UN VEHÍCULO TERRESTRE USANDO MARCADORES FIDUCIALES

Autor: Ucelay Jiménez, Alejandro.
Director: Zamora Macho, Juan Luis.
Co-Director: Boal Martín-Larrauri, Jaime.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

PROJECT SUMMARY

This project has developed an autonomous navigation system for terrestrial mobile robots in closed physical environments. The navigation system includes the vehicle localisation algorithm through a system of detection of fiducial markers by means of computer vision.

Key Words: Fiducial markers, Optitrack camera system, mobile robot, autonomous navigation.

1. Introduction

The main objective of this project is the design of a localisation system based on the detection of fiducial markers. This positioning system has been integrated into a navigation control for the ground vehicle used by the Electronics, Automation and Communications department of the ICAI School of Engineering.

The efficiency and accuracy of this marker system has been assessed by using an external positioning system composed of a set of infrared cameras. In addition, the navigation system has a communications architecture between the different hardware and software devices of the project.

2. Project definition

The vehicle electronics consists of a Raspberry Pi 3B+. The Raspberry acts as a computer and receives the measurements from the IMU and other sensors. It is responsible for high-level tasks such as communications management [1].

The vehicle also has an Inertial Measurement Unit (IMU) that consists of an accelerometer and a gyroscope [2]. Among the main actuators of the vehicle, the two DC motors (EMG30) are

controlled by PWM signals. A magnetic encoder is located on each axis of each motor, which is responsible for obtaining the speed and direction of rotation information of each motor.

The fiducial marker detection algorithm is responsible for obtaining the position and orientation of the vehicle in the working environment. This algorithm is based on an artificial vision system. For this reason, the Raspberry Pi Camera Module V2 optical sensor and the OpenCV library have been used for image processing [3]. In addition, the ArUco family markers have been used as fiducial markers.

Finally, the Optitrack camera system is a motion capture system that uses a set of infrared cameras to obtain the position and orientation of the vehicle in the working environment [4]. This external localisation system has been used to characterise the accuracy of the ArUco marker detection algorithm. A schematic of the different hardware devices in the project is shown in *Figura 4*.

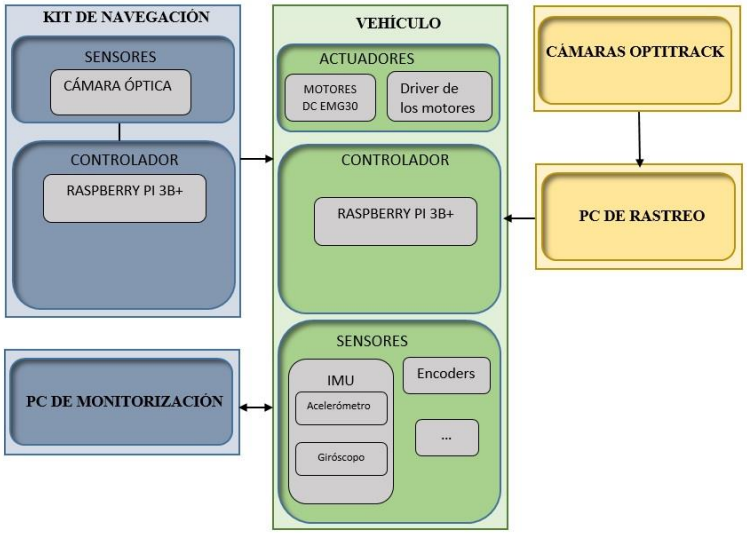


Figura 4: Hardware architecture of the project.

3. Solution

The vehicle positioning and orientation system based on ArUco markers is a computer vision application that detects ArUco markers in each image captured by the camera. For this purpose, the OpenCV library has been used, which has a module called `aruco` that allows the detection of markers, as well as obtaining the position and orientation of the optical sensor with respect to the detected marker. The positioning algorithm has been programmed in Python 3.10.

Secondly, it has been necessary to establish a communications architecture between the following hardware devices of the project: monitoring PC, Optitrack camera tracking PC,

Raspberry Pi 3B+ of the vehicle and Raspberry Pi 3B+ of the marker detection algorithm. For this purpose, different communication protocols such as UDP, TCP/IP or Bluetooth have been used. At the same time, ROS2 (Robot Operating System) has been used as the communication system between the two Raspberries in the project.

In order to integrate the localisation system into a navigation control, three controls for forward speed, turning speed and yaw angle have been used.

4. Results

The position and orientation of the vehicle obtained through the marker detection algorithm has been compared with the location provided by the external Optitrack camera system. Below, different graphs are shown where the accuracy of the algorithm is characterised for the vehicle movement along the X, Y axis and the angle of the vehicle on its own axis (Z). (*Figura 5*)

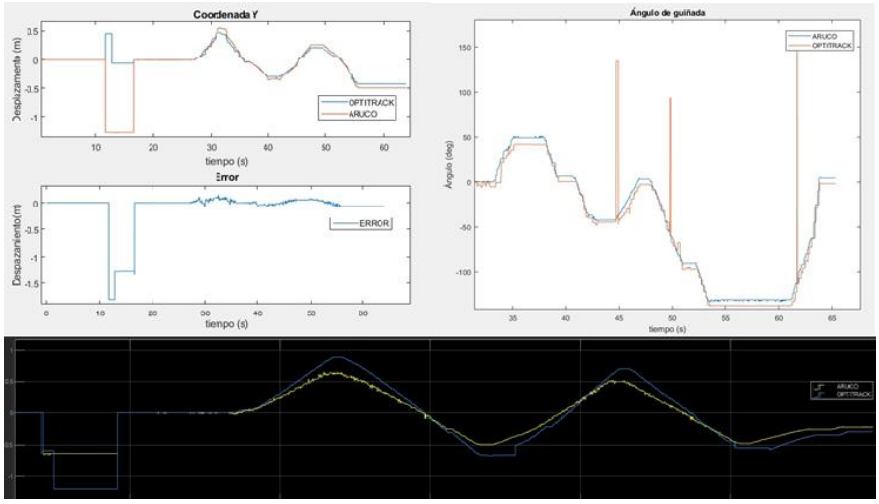


Figura 5: Characterisation of the accuracy of the marker system.

The results of the integration of the positioning system into a vehicle yaw angle navigation control are also shown. Both P-type and PID-type controls have been used. (*Figura 6*)

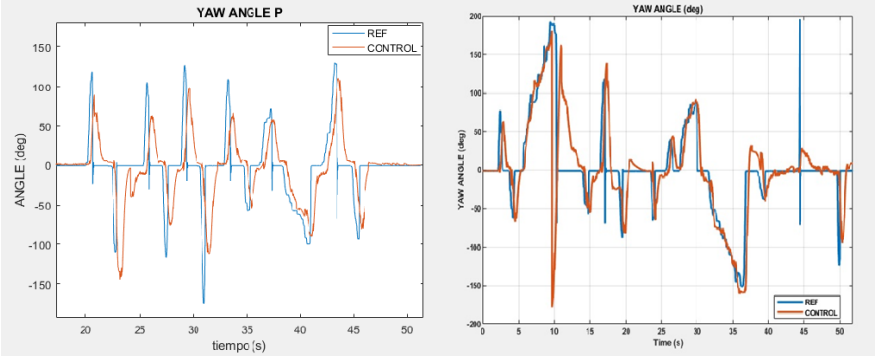


Figura 6: Vehicle yaw angle P and PID controls.

5. Conclusions

It can be concluded that this marker detection technology is competent with other positioning technologies, since like any engineering application, it is a compromise between performance, price and the casuistry of the project in question. The accuracy of this technology depends on the distance between the optical camera and the fiducial marker. However, this limitation can be solved by increasing the size of the marker or by positioning a larger number of markers and spacing them farther apart to cover a larger navigation space.

In turn, the accuracy of this marker detection algorithm is subject to factors such as proper calibration of the optical camera or favourable illumination of the physical environment.

6. References

[1] R. P. Ltd, «Buy a Raspberry Pi 3 Model B+», *Raspberry Pi*. https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/ (accedido 4 de julio de 2022).

[2] «Unidad de medición inercial», *Wikipedia, la enciclopedia libre*. 2 de julio de 2021. Accedido: 4 de julio de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Unidad_de_medici%C3%B3n_inercial&oldid=136743735

[3] «openCv», *OpenCV*. https://opencv.org/ (accedido 4 de julio de 2022).

[4] «Motion Capture Systems», *OptiTrack*. http://optitrack.com/index.html (accedido 4 de julio de 2022).

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

CAPÍTULO 1: INTRODUCCIÓN

En este primer capítulo se expone una visión general del proyecto. En la sección 1.3 *Estado del arte*, se explican diferentes proyectos o estudios acerca de la tecnología a emplear en este proyecto. A su vez, se establecen los objetivos del proyecto, así como la motivación del mismo.

1.1 MOTIVACIÓN

La sociedad evoluciona hacía una transformación digital y es por ello que crece la demanda de procesos industriales inteligentes con capacidad de planear, controlar y producir lo que genera mayor valor en una cadena de producción.

Los robots móviles desempeñan un papel fundamental en la optimización de los procesos industriales. No obstante, hacer un seguimiento y obtener la localización en cada instante de tiempo para cada robot, supone un reto para conseguir una eficaz navegación autónoma. Se puede afirmar que el posicionamiento y orientación de un objeto móvil es el corazón de la navegación autónoma del mismo.

Es por ello, que la navegación autónoma de un vehículo terrestre mediante la detección de marcadores fiduciales podría ser una forma innovadora de navegar y a su vez, un método para la localización de objetos móviles en entornos físicos cerrados. Además, este proyecto sería la antesala a la implantación de dicho sistema de navegación en vehículos aéreos no tripulados (VANT), comúnmente conocido como drones. Dichos drones podrían desplazarse de un punto a otro siguiendo unas determinadas rutas trazadas en el suelo de la nave en cuestión. A su vez, se podría elevar esta idea a gran escala disponiendo de una flota de drones interconectados entre sí, haciendo uso de una plataforma IoT.

1.2 OBJETIVOS

El presente proyecto tiene como objetivo el control de un vehículo terrestre, capaz de desplazarse de manera autónoma, en un entorno limitado, usando como referencia marcadores fiduciales. Dicho proyecto gira en torno a un microcontrolador *Raspberry Pi*, una cámara óptica y una serie de actuadores conectados a la *Raspberry Pi*. La cámara óptica se colocará en la parte superior del vehículo y podrá apuntar tanto al techo como a las paredes del entorno físico para detectar los marcadores. A su vez, el sistema de posicionamiento y localización del vehículo se monitoriza a partir de un sistema de cámaras externas *Optitrack* con el fin de valorar la precisión del algoritmo de detección de marcadores.

Por lo tanto, se ha dividido el proyecto en tres partes:

1. Desarrollar un sistema de localización basado en marcadores fiduciales.
2. Diseñar la estructura del sistema de comunicaciones del proyecto.
3. Caracterizar la precisión del sistema de localización mediante un sistema de cámaras externas.
4. Integrar el sistema de localización con un control de navegación para un vehículo terrestre.

1.3 ESTADO DEL ARTE

El auge de las nuevas tecnologías ha dado lugar a la utilización de robots móviles inteligentes para diversas aplicaciones. Con el paso de los años, la demanda de instalaciones de robots industriales ha ido aumentando y actualmente existe una necesidad de estudiar el potencial de estos robots para la automatización de determinadas tareas. Con el fin de conseguir dicha autonomía, es indispensable que los robots adquieran 2 capacidades esenciales: la localización (identificar su ubicación en un entorno físico) y la navegación (poder desplazarse a determinados puntos de interés).

Actualmente, el sistema de navegación más globalizado es el GPS (Global Positioning System). Éste es un sistema de radionavegación propiedad de los Estados Unidos que

proporciona servicios de posicionamiento, orientación y cronometría a cualquier usuario portador de un receptor GPS [5]. Sin embargo, el GPS presenta dificultades a la hora localizar objetos en espacios interiores al atenuarse la señal de radio entre el receptor y el satélite.

Es por ello que en los últimos años, se han abordado nuevos métodos que permiten conocer la posición absoluta de un objeto móvil en un entorno cerrado. Entre estos métodos destaca el posicionamiento a partir de balizas, bandas magnéticas, LIDAR etc. Estas nuevas tecnologías se utilizan hoy en día como sistemas de navegación, no obstante, aún se tienen que perfeccionar ya que presentan retos como el elevado coste de inversión, los elevados requerimientos computacionales (LIDAR) o la compleja instrumentación del entorno físico (bandas magnéticas) [6]. En el caso del sistema de balizas, la precisión en la localización del objeto, depende proporcionalmente del número de balizas empleadas y del tipo de señal utilizada para la comunicación baliza-vehículo (radiofrecuencia, Wi-Fi, infrarrojos…).

La tecnología de los marcadores fiduciales es competente con otras tecnologías de localización, ya que como todo en ingeniería, es un compromiso entre prestaciones, precio y la casuística del proyecto en cuestión. En una navegación SLAM basada en LIDAR, un entorno físico muy repetitivo como puede ser un aparcamiento con cientos de columnas sin referencias, puede suponer un verdadero problema a la hora de obtener con seguridad el posicionamiento del robot móvil. A su vez, la navegación LIDAR supone un problema en naves industriales con grandes zonas diáfanas en las que el LIDAR no tiene un alcance suficiente.

Con el objetivo de solucionar los problemas mencionados anteriormente, se creó una nueva tecnología basada en la implantación de marcadores fiduciales en el entorno físico. La facilidad en su instalación, el mínimo coste de inversión, los escasos requerimientos computacionales y la rápida obtención de la localización, constituyen las principales ventajas de este sistema de navegación [7]. De tal forma, estos marcadores se encuentran posicionados de forma fija y el robot móvil los detecta empleando un sistema de visión artificial. Al detectarlos no solo puede obtener su posición respecto a la base del marcador sino que también es conocedor de la transformación espacial que debe realizarse desde el marcador fijo hasta el punto del entorno al que desea desplazarse [8]. No obstante, el correcto funcionamiento de

este sistema de posicionamiento se encuentra sujeto a factores como la iluminación del entorno de navegación. A su vez, este algoritmo de detección de marcadores cuenta con limitaciones en entornos de navegación en el exterior ya que los marcadores se deterioran ante condiciones meteorológicas adversas.

Actualmente, numerosas empresas pioneras en el sector de la robótica están apostando por este sistema de navegación. La empresa española ASTI (Automatismos y Sistemas de Transporte Interno) creó el robot EBOT 350, *Figura 1*, una plataforma para el transporte de mercancías. Dicho robot hace uso de una tecnología de navegación de última generación a través de marcadores QR.

Figura 1.1: Platform Line ASTI [https://www.astimobilerobotics.com/]

Por otro lado, la unidad de investigación de la empresa Aldebaran Robotics, llevó a cabo un ensayo haciendo uso del robot humanoide NAO [9]. La empresa trató de implementar el control de navegación mediante la detección de marcadores. Cada marcador se identifica con un nº ID como se muestra en la *Figura 1.2*. El ensayo tuvo lugar en un entorno físico limitado (pasillo de 5m de longitud) y el resultado del mismo se muestra en [10].

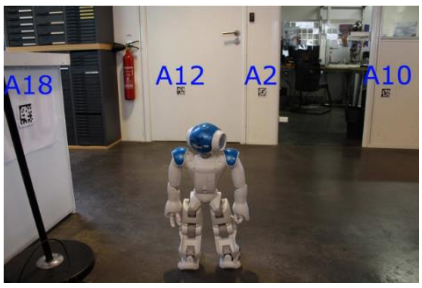


Figura 1.2: Robot NAO con los marcadores pegados en la pared. [https://www.softbankrobotics.com/emea/es/nao]

Un sistema de marcadores visuales está compuesto por una serie de marcadores 2D y un algoritmo de visión artificial para detectar y decodificar cada marcador. Hoy en día existen

numerosos marcadores fiduciales en el mercado (Vuforia, ArUco, AprilTag, QR-code, Aztec-code etc.).

El marcador más popular y más usado es el QR-code ya que contiene 4296 caractéres alfanuméricos y cuenta con un algoritmo de corrección de errores (Reed-Solomon) para detectar marcadores que se encuentren parcialmente ocultos o distorsionados [11]. No obstante, no se ajustan al objetivo de este proyecto ya que su uso no está orientado a conocer la posición de la cámara óptica respecto a ellos [12]. Por lo tanto, la búsqueda de los marcadores a emplear se restringe a aquellos con la capacidad de proporcionar información sobre la posición y orientación de la cámara. Se centrará el estudio en los marcadores ArUco, AprilTag, ARTag y STag.

Se han llevado a cabo numerosos estudios acerca de estos marcadores, sin embargo, cabe destacar un ensayo experimental llevado a cabo por *Unmanned Systems & Robotics LAB (USRL)* [11], con el objetivo de estimar la localización de la cámara (distancia y orientación) con respecto a cada marcador fiducial. Como cámara óptica se hizo uso de una Raspberry Pi Camera versión 2 (piCam). El objetivo de este ensayo era medir la eficacia de detección de cada marcador, así como su coste computacional. A continuación se muestran las conclusiones del estudio:

- Tanto AprilTag como STag y ArUco presentan altos porcentajes de detección (>95%) para cada uno de los escenarios del experimiento. Sin embargo, los marcadores ARTag presenta un porcentaje de detección en torno al 50%. Los marcadores STag obtuvieron la mejor precisión para estimar distancias y los marcadores AprilTag destacaron en la estimación de la orientación de la cámara. En ambos casos, los marcadores ArUco fueron los segundos mejores [13].

- Por último, en la *Figura 1.3*, se muestra el coste computacional de cada marcador en la Raspberry Pi. El uso de CPU se encuentra representado en porcentaje y está directamente relacionado con la cantidad de tiempo que emplea la Raspberry en

procesar las instrucciones. Se puede observar como los marcadores ArUco son los más eficientes computacionalmente, y los AprilTag los menos eficientes [13].

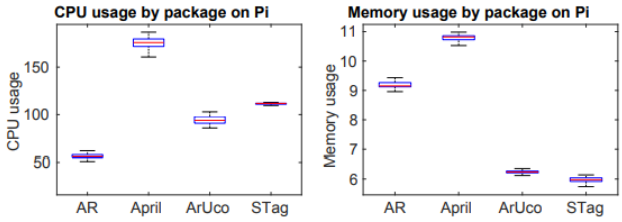


Figura 1.3: Coste de procesamiento para cada marcador fiducial.

El hecho que los marcadores ArUco demanden pocos recursos computacionales y por consiguiente, que presenten un tiempo medio de detección de 27 ms [11], supone una gran ventaja a la hora de poder localizar al vehículo en cada instante de tiempo. A su vez, estos presentan una gran eficacia a la hora de ser detectados. Es por ello, que los marcadores ArUco son idóneos para plataformas de baja potencia como la Raspberry Pi.

CAPÍTULO 2: ARQUITECTURA

En este capítulo se describen los diferentes dispositivos *hardware* que conforman este proyecto. Además se explica el funcionamiento y la labor que desempeña cada uno de ellos.

2.1 HARDWARE

Entre los diferentes dispositivos *hardware* del proyecto, se destacan los siguientes: vehículo terrestre, kit de navegación, sistema de cámaras externo Optitrack, PC de monitorización y PC de rastreo de las cámaras Optitrack. En la *Figura 2.1*, se muestra la jerarquía del *hardware* del proyecto.

En esta sección se describen los principales elementos que componen tanto el vehículo terrestre como el kit de navegación.

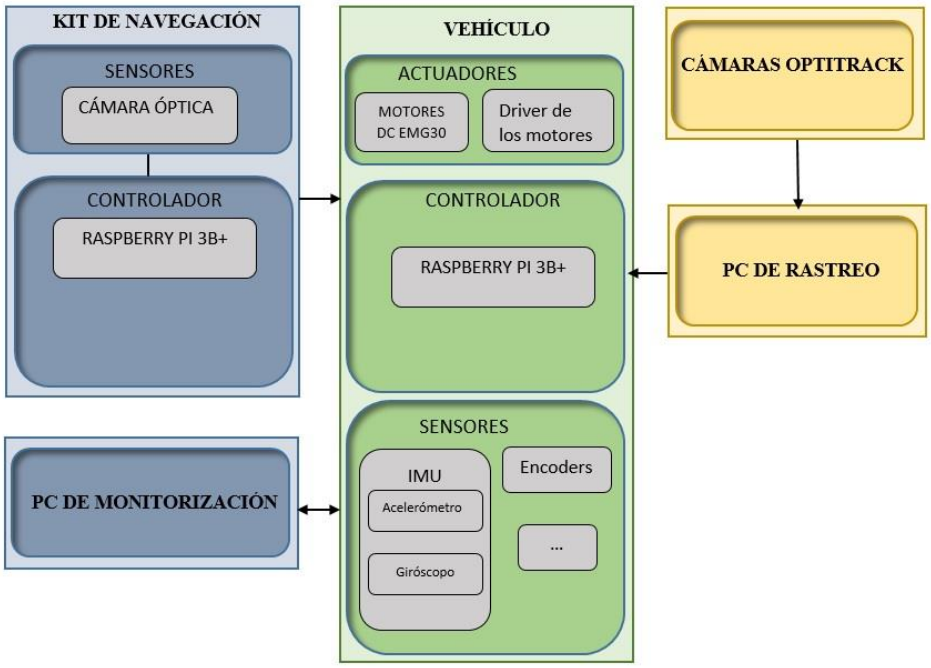


Figura 2.1: Jerarquía hardware del proyecto.

2.1.1 VEHÍCULO TERRESTRE

El vehículo de la *Figura 2.2* cuenta con una Unidad de Medición Inercial (IMU) y dos encoders magnéticos, estos sensores forman parte del hardware del vehículo.

En primer lugar, la IMU se encuentra constituida por un acelerómetro y un giróscopo. Las medidas de la aceleración se obtendrán gracias al acelerómetro, mientras que el giróscopo proporcionará las medidas de la velocidad angular [14] .

En segundo lugar, los encoders magnéticos se encuentran ubicados en los ejes de los motores DC. Estos proporcionarán información sobre la velocidad y sentido de giro de los motores. Además, el vehículo consta de 3 pulsadores para controlar el estado del vehículo. Los pulsadores se emplearán como señales digitales de entrada en la máquina de estados.

En cuanto a los diferentes actuadores del vehículo, se destaca los dos motores DC (EMG30) y un LED RGB para señalizar el estado del vehículo.

Por otro lado, un microcontrolador Raspberry Pi 3B+ obtiene las medidas de los sensores de la IMU y de los encoders magnéticos y monitoriza el estado de los 3 pulsadores enviando órdenes PWM al driver MD25 de los motores DC.

Para recibir la información sobre la localización y orientación del vehículo desde el ordenador, se usará una emisora de radiocontrol. A su vez, el vehículo cuenta con varios marcadores infrarrojos para que el vehículo sea detectado por el sistema de cámaras Optitrack.

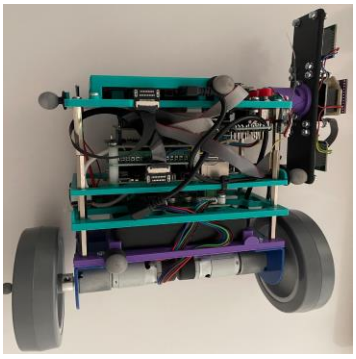


Figura 2.2: Vehículo terrestre.

2.1.2 KIT DE NAVEGACIÓN

El sistema de posicionamiento del vehículo mediante el uso de marcadores fiduciales está compuesto de varios componentes:

- Raspberry Pi 3B+: Es el microcontrolador encargado de la carga computacional requerida para la detección y obtención de la posición respecto de los marcadores ArUco. (*Figura 2.3*)

Figura 2.3: Raspberry Pi 3B+.

- UPS: *Uninterruptible Power Supply*. Ofrece una salida constante de 5,1 V con una intensidad de 3000mA durante 5 horas. Puede funcionar con una o con dos baterías de iones de litio 1650, con protección integrada contra sobrecorriente y protección contra sobretensión [15]. En la *Figura 2.4*, se muestra el UPS empleado.

Figura 2.4: UPS Hat. [https://geekworm.com/blogs/news/new-x728-max-5-1v-8a-18650-ups-power-management-board-with-ac-power-loss-detection-auto-on-safe-shutdown-function]

- Sensor óptico: El sensor óptico utilizado para la captura de imágenes es la cámara versión 2 de 8MP de la Raspberry Pi.

Para ensamblar todos los componentes ha sido necesario diseñar e imprimir el kit de navegación. Para el diseño de este se ha empleado el software de diseño 3D llamado SolidWorks. Cada pieza diseñada del kit de navegación se exportó a formato .stl para poder ser impresa en la HP MultiJet Fusion. Esta impresora emplea polvo de arena como material de impresión [16]. A continuación, en la *Figura 2.5*, se muestra el kit de navegación ensamblado junto al vehículo.

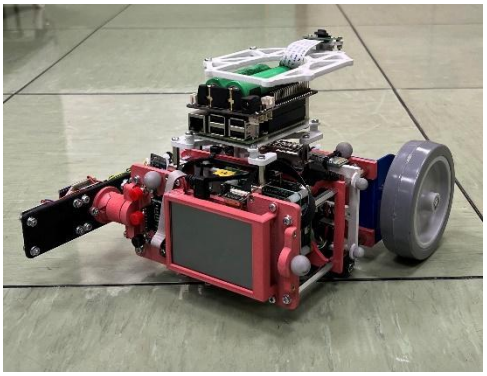


Figura 2.5: Vehículo terrestre y kit de navegación.

2.1.3 PC DE MONITOIZACIÓN Y RASTREO

El PC de monitorización contiene todos los ficheros de Simulink que albergan la lógica del vehículo. Además, es donde se monitorizan las variables de estado del vehículo y se configuran las variables de referencia.

Por otro lado, el PC de rastreo, es aquel que emplea el software *Motive* para obtener la posición y orientación del vehículo a partir de las cámaras Optitrack. El sistema de comunicaciones de ambos PCs con el vehículo se describe en el *Capítulo 6.*

A continuación, en la *Figura 2.6,* se muestra un esquema del hardware del proyecto, incluyendo las direcciones de comunicación entre los diferentes elementos.

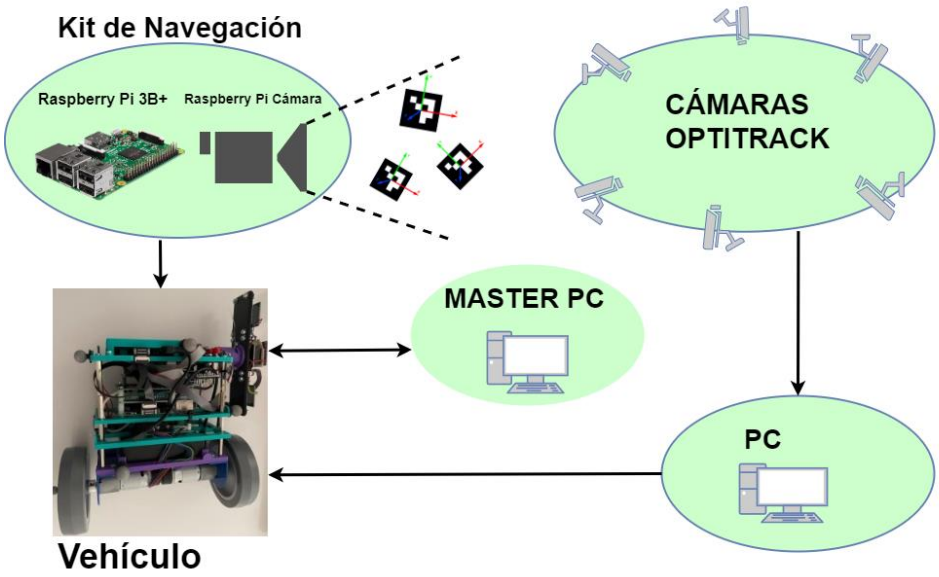


Figura 2.6: Esquema arquitectura hardware.

2.2 SOFTWARE

En esta sección se procede a explicar la labor que desempeña cada elemento *hardware* del proyecto.

En primer lugar, el kit de navegación se encarga de obtener la posición y orientación del vehículo a partir de un sistema de visión artificial. Para ello, se utiliza la librería OpenCV (*Open Computer Vision*) desarrollada por Intel. Cuenta con un módulo contributivo llamado `aruco`. El módulo `aruco` se basa en la biblioteca ArUco, una popular biblioteca para la detección de marcadores fiduciales cuadrados desarrollada por Rafael Muñoz y Sergio Garrido [17]. El proceso de detección de los marcadores se explica de forma detallada en el capítulo 3.3.1. La posición y orientación de la cámara es transmitida al vehículo a través del protocolo de comunicación TCP/IP o mediante el *software* ROS 2 (*Robot Operating System*).

Por otro lado, las cámaras Optitrack utilizan el *software Motive* para obtener la localización del vehículo en el entorno de trabajo. La posición y orientación de la cámara procedente de *Motive* es transmitida al vehículo a través del protocolo de comunicación TCP/IP.

Una vez conocida la posición y orientación del vehículo mediante ambos sistemas de posicionamiento, se utiliza Simulink para caracterizar la fiabilidad del algoritmo de detección de marcadores. A su vez, el control del vehículo se encuentra integrado en un fichero de Simulink, que se vuelca en la Raspberry Pi 3B+ del vehículo y permite monitorizar la navegación del robot móvil. En la *Figura 2.7* se muestra el esquema del *software* del proyecto.

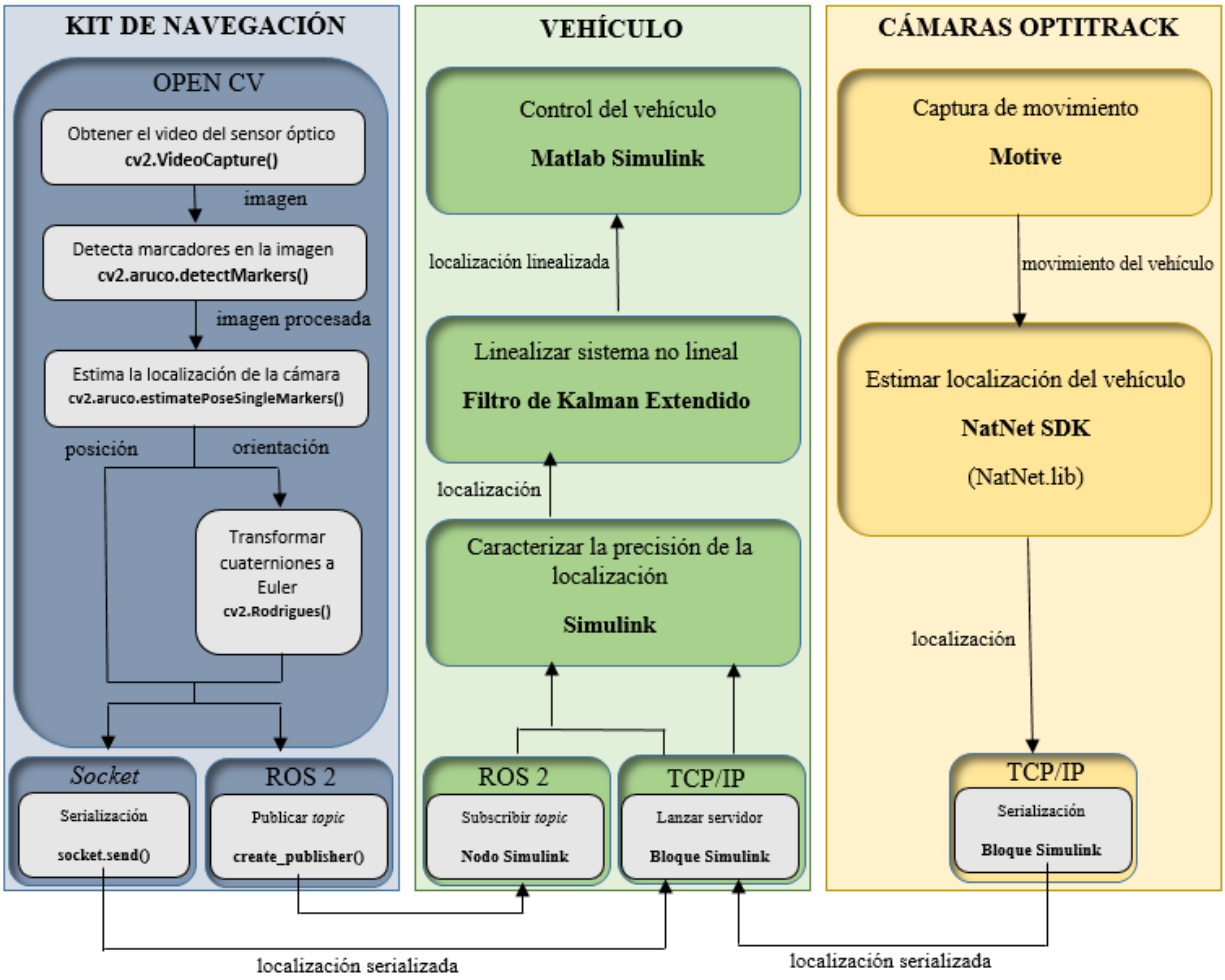


Figura 2.7: Esquema del software del proyecto.

CAPÍTULO 3: DETECCIÓN DE LOS MARCADORES ARUCO

El algoritmo de posicionamiento del vehículo se basa en un sistema de visión artificial. Dicho sistema incluye métodos para adquirir, procesar y analizar imágenes del entorno físico, con el fin de obtener información acerca de la posición y orientación del vehículo. En esta sección, se procede a explicar el método de implantación y configuración de dicho sistema de visión.

3.1 SENSOR ÓPTICO

Para conseguir una exitosa navegación autónoma del vehículo es imprescindible conocer en cada instante de tiempo su posición y orientación en el entorno de trabajo. Entre los diferentes sensores para conocer el estado del robot, se ha optado por utilizar un sensor óptico. A continuación, se destacan varias ventajas respecto al uso de este tipo de sensores:

- El correcto funcionamiento del sistema, únicamente depende de una óptima condición de iluminación del entorno.
- Mediante una adecuada calibración del sensor, se obtiene una gran precisión en la medición de la posición y orientación relativas al sensor óptico en cada uno de los tres ejes del espacio.
- El procesamiento de imágenes requiere de poca carga computacional, lo que favorece la obtención de medidas con poco tiempo de latencia.

En concreto, se ha hecho uso del sensor óptico *Raspberry Pi Camera Module V2*, sensor de imagen Sony IMX219 de 8MP y capaz de conseguir una resolución de 3280 x 2464 pixeles [18]. Dicho sensor se muestra en la *Figura 3.1*.

Se ha elegido este sensor debido a su elevada compatibilidad con la Raspberry Pi 3B+. Para su uso, únicamente es necesario habilitar el módulo de la cámara, añadiendo los siguientes comandos en el archivo *config.txt*:

```
$ sudo nano /boot/firmware/config.txt
gpu_mem=128
disable_camera_led=1
start_file=start4x.elf
fixup_file=fixup4x.dat
```

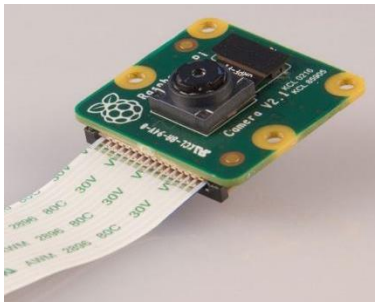


Figura 3.1: Raspberry Pi Camera v2.[https://tienda.bricogeek.com/accesorios-raspberry-pi/822-camara-raspberry-pi-v2-8-megapixels.html]

3.2 OPENCV

Para el ser humano es muy sencillo interpretar formas, colores y tamaño de una imagen. No obstante, este proceso "sencillo", esconde una enorme complejidad, tanto es así que los ordenadores de hoy en día empiezan a poder distinguir colores, formas etc. OpenCv nace en 1999 de la mano de Intel para abordar el problema de visión artificial.

OpenCV es una librería de software de código libre, para el procesamiento de imágenes y el desarrollo de tareas relacionadas con la visión artificial [19]. Actualmente está plenamente desarrollado en C++ aunque incluye conectores con diferentes lenguajes de programación (Python, Java y Javascript) . En este proyecto, para poder acceder a las funciones básicas del sensor óptico, se ha utilizado OpenCV *(Open Computer Vision).* A su vez, el algoritmo de posicionamiento se ha desarrollado en Python. A continuación, se muestran los comandos necesarios para instalar Python y OpenCV en la Raspberry Pi 3B+ con sistema operativo Ubuntu LTS 20.04:

```
$ sudo apt-get update
$ sudo apt-get install python3
$ sudo pip3 install opencv-contrib-python
```

Tras el proceso de instalación, se ha comprobado que las versiones de Python y OpenCV instaladas han sido, 3.8.10 y 4.5.5 respectivamente.

3.3 MARCADORES ARUCO

Los marcadores ArUco pertenecen a la familia de los marcadores fiduciales. Son marcadores de forma cuadrada, con fondo negro y una matriz binaria en su interior para indicar la singularidad del marcador. OpenCV cuenta con un módulo "contributivo" llamado `aruco` [17].

Es importante mencionar que estos marcadores se agrupan en distintas familias o "diccionarios". Un diccionario alberga un número determinado de marcadores, todos ellos del mismo tamaño (número de bytes) [20]. En este proyecto, se ha optado por elegir el diccionario DICT_4x4_50, es decir, un diccionario que contiene 50 marcadores diferentes con un tamaño de 16 bytes cada uno. En la *Figura 3.2*, se muestran los marcadores 1, 5 y 30 del diccionario DICT_4x4_50.

Figura 3.2: Marcadores ArUco del diccionario DICT_4x4_50.

A continuación se muestra el código fuente necesario para generar el marcador ArUco con número identificativo 1 del diccionario DICT_4x4_50:

```
import cv2
import numpy as np
aruco_id = 1
aruco_dict = cv2.aruco.Dictionary_get(cv2.aruco.DICT_4X4_50)
tag = np.zeros((400, 400), dtype=np.uint8)
markerImage = cv2.aruco.drawMarker(aruco_dict, aruco_id, 400, tag, 1)
```

El método aruco perteneciente a la librería OpenCV permite interpretar el marcador ArUco como un sistema de referencia, de forma que calibrando la cámara óptica, es posible conocer la posición y orientación del marcador ArUco respecto de la cámara. En la *Figura 3.3* se muestra conjuntamente tanto el sistema de referencia del sensor óptico como el sistema de referencia de un marcador ArUco.

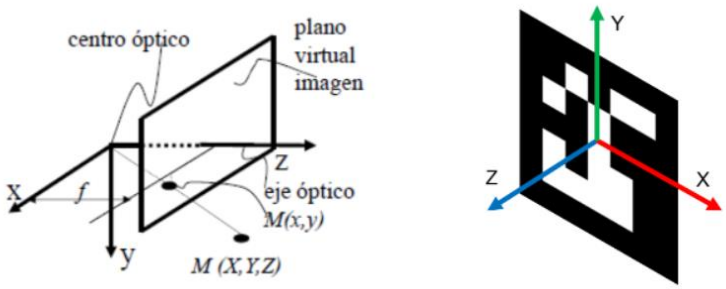


Figura 3.3: Sistema de referencia de la cámara (izq) y sistema de referencia del marcador (dcha).

Los marcadores se encuentran compuestos por un borde exterior negro y una región interior que alberga una matriz binaria. La matriz binaria es única e identificativa para cada marcador. Dependiendo del diccionario, existen marcadores con más o menos bits. Cuantos más bits, más marcadores dentro del diccionario y menor la posibilidad de confusión. No obstante, un mayor número de bits requiere una mayor resolución para una correcta detección. En la *Figura 3.4*, se muestra el sistema de referencia utilizado en la librería aruco. A su vez, la letra s hace referencia al tamaño del marcador.

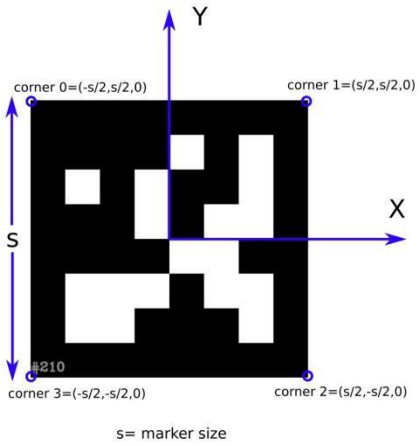


Figura 3.4: Sistema de referencia según las esquinas del marcador. [https://www.uco.es/investiga/grupos/ava/node/26]

3.3.1 DETECCIÓN DE LOS MARCADORES

Con el objetivo de poder llevar a cabo una correcta detección de los marcadores ArUco, es necesario seguir los siguientes pasos:

1. Cargar una imagen de entrada.
2. Cargar el diccionario ArUco correspondiente.
3. Definir los parámetros ArUco.
4. Realizar la detección del marcador ArUco en la imagen de entrada.

A continuación, se muestra el código fuente necesario para cada paso:

```
Paso 1:
video = cv2.VideoCapture(0)
ret, image = video.read()
Paso 2:
aruco_dict = cv2.aruco.Dictionary_get(cv2.aruco.DICT_4X4_50)
 Paso 3:
params = cv2.aruco.DetectorParameters_create()
Paso 4:
corners, ids, rejected = cv2.aruco.detectMarkers(image, aruco_dict,
parameters=aruco_params)
```

El proceso de detección de los marcadores se basa en obtener secuencialmente los bordes exteriores de los marcadores, los vértices donde se unen y la matriz binaria del interior [21]. Durante el proceso, se van descartando aquellos potenciales marcadores que fueron detectados, pero cuyo patrón interior no pudo ser analizado. Estos marcadores descartados se almacenan en el vector *rejected.*

Para cada marcador detectado se obtiene un vector *corners* donde se alberga la posición de las cuatro esquinas del marcador respecto a su origen de coordenadas (centro del marcador).

CAPÍTULO 4: ALGORITMO DE LOCALIZACIÓN

Un robot es un objeto rígido al que se le puede asignar un sistema de referencia relativo (móvil). La localización del vehículo en cada instante de tiempo estaría determinada por la relación existente entre el sistema de referencia global (*Rg*) en virtud del cual está definido todo el entorno físico y su sistema de referencia relativo (*Rr*). (*Figura 4.1*)

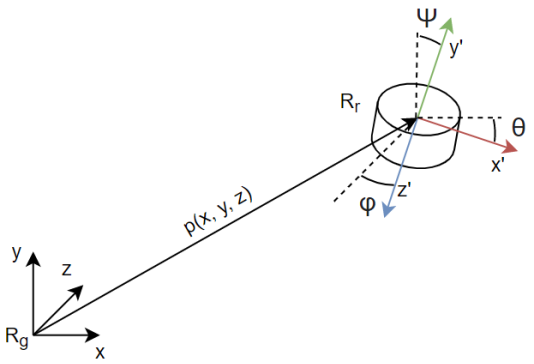


Figura 4.1: Sistema de referencias absoluto y relativo de un robot móvil.

Sea *l*, la localización del vehículo en un instante determinado de tiempo:

$$l = (p, \theta, \Psi, \varphi) = (x, y, z, \varphi, \theta, \Psi)$$

En esta sección, se procede a explicar el algoritmo desarrollado para obtener la posición y orientación del sensor óptico respecto a un sistema de referencia en cada uno de los tres ejes del espacio. A su vez, se explicará el método que hace uso del sistema de cámaras Optitrack para obtener la localización del robot móvil.

4.1 CALIBRACIÓN DEL SENSOR ÓPTICO

En aplicaciones de visión artificial es fundamental realizar una correcta calibración del sensor óptico. Esto es debido a que la cámara cuenta con unos parámetros intrínsecos que

pueden producir distorsión y adulterar el procesamiento de las imágenes. Los parámetros intrínsecos describen las propiedades internas de la cámara y se definen a través de la matriz de la cámara [22]. Por otro lado, los parámetros extrínsecos hacen referencia a la posición espacial de la cámara (traslación y rotación).

En este proyecto se ha empleado un patrón de tablero de ajedrez con cuadrados de dimensión conocida para obtener los parámetros intrínsecos y el vector de distorsión de la cámara. El proceso ha consistido en comparar la dimensión conocida de los cuadrados del tablero con las obtenidas en la proyección de cada una de las treinta imágenes empleadas. A continuación, en la *Figura 4.2* se muestra un ejemplo de una imagen con su homóloga corregida.

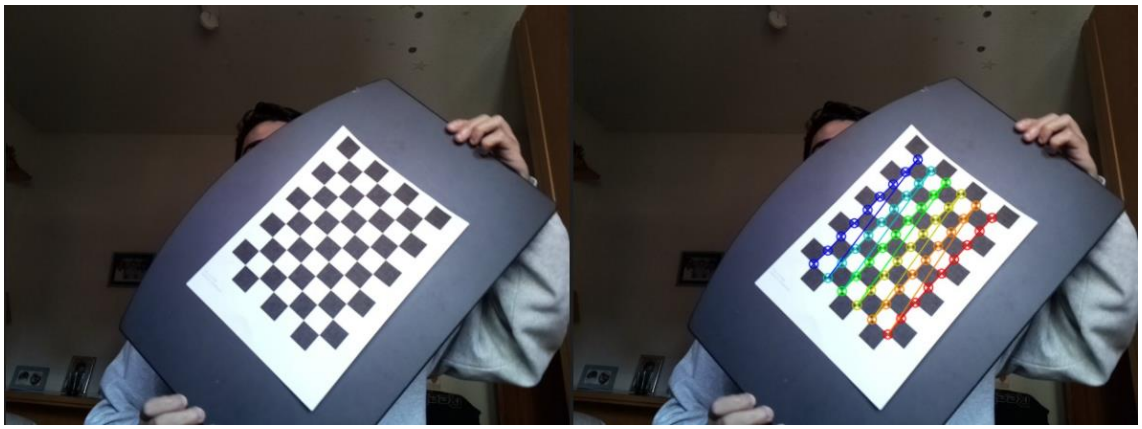


Figura 4.2: Imagen y su homóloga corregida.

Una vez terminado el proceso de calibración, se obtuvo la matriz de la cámara *M* y el *Vector Distorsión*, siendo *fx, fy* las distancias focales y *cx, cy* los desplazamientos del eje óptico respecto al chip de la cámara.

$$M = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 497.045 & 0 & 316.404 \\ 0 & 497.455 & 248.939 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Distorsión = [0.142 \quad -0.187 \quad 0.003 \quad -0.002 \quad -0.254]$$

4.2 ESTIMACIÓN DE LA POSICIÓN

Una vez los marcadores ArUco pueden ser detectados de forma correcta, se procede a la obtención de la localización de la cámara respecto a ellos. Previamente, se ha de realizar la calibración de la cámara expuesta en el apartado anterior.

Para conseguir la localización absoluta de la cámara, antes se debe obtener la posición y orientación de la cámara respecto a cada uno de los marcadores del entorno de trabajo [23].

Si la cámara empleada está calibrada, se puede utilizar aruco para estimar su pose (es decir, su posición y orientación en el espacio con respecto al marcador). La detección de las cuatro esquinas de un marcador, permite aplicar los algoritmos estimadores de la posición. Estos estiman la pose relativa de la cámara respecto al centro del marcador.

Es muy importante señalar que la estimación de la posición utilizando sólo 4 puntos coplanares está sujeta a ambigüedad. Como se muestra en la *Figura 4.3*, un marcador podría proyectarse en los mismos píxeles en dos ubicaciones diferentes de la cámara. En general, la ambigüedad puede resolverse si la cámara está cerca del marcador. Sin embargo, a medida que el marcador se hace pequeño, los errores en la estimación de las esquinas aumentan y la ambigüedad se convierte en un problema [24].

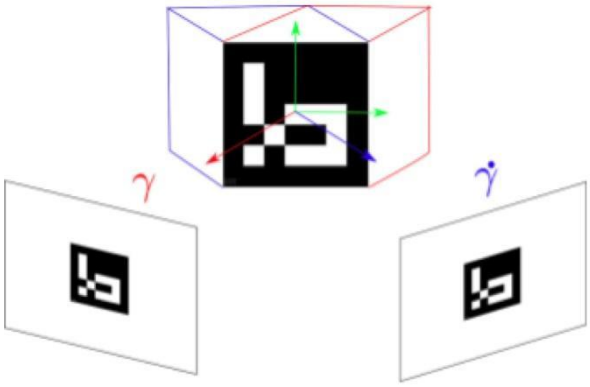


Figura 4.3: Problema de la ambigúedad. [https://ciis.lcsr.jhu.edu/doku.php?id=courses:456:2019:projects:456-2019-10:project-10]

4.2.1 POSICIÓN RELATIVA DE UN MARCADOR

La posición de la cámara respecto a un marcador es la transformación 3D del sistema de coordenadas del marcador al sistema de coordenadas de la cámara. Esta transformación 3D se expresa mediante un vector de traslación y un vector de rotación. El módulo `aruco` de la librería OpenCV proporciona una función para estimar la posición para cada uno de los marcadores detectados en la imagen. A continuación se muestra parte del código fuente.

```
rvecs, tvecs, obj_points = cv2.aruco.estimatePoseSingleMarkers(markerCorners,
aruco_marker_side_length, mtx, dst)
```

- El parámetro markerCorners hace referencia al vector de esquinas de los marcadores devuelto en la función *detectMarkers()*.
- El segundo parámetro es el tamaño de lado de los marcadores en la unidad que se desee obtener los vectores de traslación y rotación.
- El tercer parámetro (mtx) hace referencia a la matriz de distorsión de la cámara. A su vez, el cuarto parámetro (dst) hace referencia a los coeficientes del polinomio de distorsión de la cámara.
- Por último, las salidas son el vector de traslación (*tvecs*) y el vector de rotación (*rvecs*) de cada marcador en el vector *markerCorners*.

4.2.2 FUSIÓN DE LAS POSICIONES RELATIVAS

En este apartado se procede a explicar una medida para mejorar la precisión y robustez de la estimación final de la posición. En este proyecto se ha utilizado varios marcadores, es por ello, que en una imagen se pueden llegar a detectar más de un marcador simultáneamente. Debido a esto, se ha procedido a realizar una media ponderada de las posiciones estimadas para cada marcador de la imagen: cuanto menor es la distancia entre el marcador y la cámara, mayor es el peso asignado [25].

Esta fusión de las posiciones se realiza para cada imagen procesada, por lo que se obtiene una posición absoluta de la cámara para cada instante de tiempo. Para asignar un peso a cada marcador, se ha empleado la siguiente ecuación:

$$distancia_i = \sqrt{tvec[i][0][0]^2 + tvec[i][0][1]^2 + tvec[i][0][2]^2} \quad \forall\, i$$

$$ratio_i = \frac{1/distancia_i}{\sum_{j=0}^{n} distancia_j} \quad \forall$$

Para la obtención del vector de traslación (x, y, z) se ha empleado la siguiente ecuación:

$$[x, y, z]_{absoluta} = \sum_{i=0}^{n} ([x_i, y_i, z_i]) * ratio_i$$

Siendo n, el número de marcadores detectados en una imagen.

Para la obtención del vector de traslación (Ψ, φ, θ) se ha empleado la siguiente ecuación:

$$[\Psi, \varphi, \theta]_{absoluta} = atan\left(\frac{\sum_{i=0}^{n}(sen[\Psi_i, \varphi_i, \theta_i]) * ratio_i}{\sum_{i=0}^{n}(cos[\Psi_i, \varphi_i, \theta_i]) * ratio_i}\right)$$

Siendo n, el número de marcadores detectados en una imagen.

4.2.3 ESTIMACIÓN DE LA POSICIÓN ABSOLUTA

Una vez se conoce la posición ponderada de la cámara respecto a los marcadores detectados en cada imagen, se procede a obtener la posición absoluta de la cámara respecto a un sistema de referencia fijo. Para ello, se ha establecido el centro de un marcador como sistema de referencia. A continuación, es necesario conocer el posicionamiento y orientación del resto de marcadores del entorno respecto al sistema de referencia. Una vez conocidas las matrices de rotación y traslación de cada marcador respecto al marcador referente, se puede obtener la posición de la cámara en todo instante siempre y cuando al menos un marcador sea detectado. Conocida la posición del sensor óptico respecto a un marcador y la posición de dicho marcador respecto al sistema de referencia, se puede

obtener mediante transformaciones espaciales (triangulación) la posición absoluta de la cámara respecto al sistema de referencia global.

4.3 SISTEMA EXTERNO DE CÁMARAS OPTITRACK

En aplicaciones de robótica móvil en interiores, los sistemas de localización juegan un papel muy importante. En este proyecto, se ha decidido utilizar el sistema de cámaras Optitrack-Motive, como medida de fiabilidad para poder valorar de manera crítica la localización del vehículo obtenida a partir de los marcadores ArUco.

En este apartado se muestra una explicación tanto del sistema de cámaras empleado durante el desarrollo de este proyecto como del software que se ha usado para el procesamiento de los datos y su posterior transmisión.

4.3.1 HARDWARE: OPTITRACK

Se trata de un sistema capturador de movimiento que permite obtener la posición y la orientación de objetos móviles en el espacio tridimensional. Es un sistema de seguimiento 3D, líder en la industria debido al seguimiento óptico de alto rendimiento y entre sus aplicaciones destacan las siguientes: diseño de videojuegos, animación, realidad virtual, robótica y ciencias del movimiento [26], (*Figura 4.4*).

Figura 4.4: Optitrack en la industria de los videojuegos. [https://optitrack.com/]

El sistema de cámaras que se ha hecho uso durante el desarrollo de este proyecto, está compuesto por un conjunto de diez cámaras Optitrack Flex 13, *Figura 4.5*, dispuestas en forma de rectángulo. El modelo de cámaras Flex 13 presentan una resolución de 1,3 MP, cuatro veces superior a otros sistemas de captura de movimiento en la industria como es el modelo Optitrack Flex 3. A su vez, este sistema capturador de movimiento ofrece una elevada capacidad de rastreo ya que utiliza algoritmos de procesamiento en tiempo real propios de Optitrack.

Cada cámara del entorno de trabajo, es capaz de alcanzar una elevada área de captura, lo que permite obtener un mayor volumen de captura, con un menor número de cámaras. Tanto su volumen de captura como su elevada capacidad de rastreo, hacen de este sistema capturador, un sistema idóneo para el desarrollo de este proyecto.

Figura 4.5: Cámaras Optitrack. [https://optitrack.com/]

Además, es importante mencionar que el sistema de cámaras Optitrack 13 trabaja en escala de grises, lo cual mejora la capacidad del sistema para capturar movimientos más pequeños, como pueden ser ligeros cambios en la posición y orientación del vehículo. Esta escala de grises aumenta la velocidad de captura de movimiento hasta alcanzar los 120 fotogramas por segundo, una velocidad 5 veces mayor a la velocidad normal de reproducción de las películas. Esta última característica, nos ofrece la posibilidad de obtener una mayor precisión de la localización del vehículo en tiempo real.

4.3.2 SOFTWARE: MOTIVE

Motive es una plataforma de software diseñada para controlar sistemas de captura de movimiento para diversas aplicaciones de seguimiento. Motive no sólo permite al usuario

calibrar y configurar el sistema, sino que también proporciona interfaces para la captura y el procesamiento de datos 3D. Los datos capturados pueden grabarse o transmitirse en directo a otros canales. Motive obtiene la información 3D a través de la Reconstrucción, que es el proceso de compilación de múltiples imágenes 2D de los marcadores para obtener coordenadas 3D [27]. Utilizando las coordenadas 3D de los marcadores rastreados, Motive puede obtener datos de 6 grados de libertad (posición y orientación 3D) para múltiples cuerpos rígidos y esqueletos, y permitir el seguimiento de movimientos complejos en el espacio 3D.

Antes de proceder al seguimiento de la localización del vehículo, es preciso preparar el entorno de trabajo. Para ello, es necesario llevar a cabo tres pasos:

1. Calibración de las cámaras Optitrack.
2. Colocación de los marcadores infrarrojos en el vehículo.
3. Configuración de la sincronización de dispositivos externos.

Como en muchos otros sistemas de medición, la calibración también es esencial para los sistemas ópticos de captura de movimiento. Durante la calibración de la cámara, el sistema calcula la posición y la orientación de cada cámara y la cantidad de distorsiones en las imágenes capturadas, y se utilizan para construir un volumen de captura 3D en Motive. Esto se hace observando las imágenes 2D de múltiples cámaras sincronizadas y asociando la posición de los marcadores de calibración conocidos de cada cámara mediante triangulación [28].

Una vez obtenida la calibración del sistema de cámaras, se procede a la colocación de los marcadores infrarrojos en el vehículo. Una correcta configuración de los marcadores es esencial para tanto para la calidad del seguimiento del vehículo como para la fiabilidad de los datos capturados. Los marcadores deben estar asegurados a la superficie del vehículo y no deben sufrir alteraciones en su posicionamiento, ya que se pueden suceder pérdidas en los datos de seguimiento. En este proyecto, se utilizarán marcadores de forma esférica con superficies retro reflectantes para reflejar completamente la luz infrarroja hacía las cámaras.

A partir de la forma geométrica del posicionamiento de los marcadores, Motive calcula el centroide del vehículo. Debido a esto, es importante hacer una colocación correcta de los marcadores, ya que el centroide marcará el origen de coordenadas del sistema de referencia del vehículo.

CAPÍTULO 5: SISTEMA DE CONTROL

5.1 SISTEMA DE CONTROL DE NAVEGACIÓN

El control de navegación que se usa en este proyecto hace referencia a un control de velocidad y de giro que permite realizar al vehículo una trayectoria recorriendo una serie de balizas o *waypoints*. Es por ello, que se va a utilizar un control en cascada, con dos lazos de control. Dicho control en cascada se muestra en la *Figura 5.1*.

El lazo de control interno contiene los controles PID de velocidad y del ángulo de guiñada. Este lazo es controlado por las variables velocidad de avance v y velocidad de giro w que actúan sobre las tensiones común y diferencial de los motores. A partir de la velocidad angular w y mediante una acción integral, se puede obtener el ángulo de guiñada θ. Asimismo, a partir de la velocidad de avance v y mediante la función coseno y seno del ángulo de guiñada θ se obtienen las variables velocidad horizontal V_x y V_y respectivamente. Haciendo uso de una acción integral para cada variable de velocidad se obtiene las variables de posición x e y.

Por otro lado, el lazo de control externo contiene como variables del mando las referencias de la velocidad de avance v_{ref} y ángulo de giro θ_{ref}. Este lazo es controlado por las variables de posición X e Y, las variables de velocidad lineal V_x y V_y y las coordenadas de los *waypoints* definidas en los ejes X e Y [29].

En relación a la planta de velocidad de avance y de giro del vehículo, ya se habían realizado en previos estudios la identificación del modelo. La planta de velocidad de avance hace referencia a la función de transferencia entre la tensión común de los motores y la velocidad de avance. Por otro lado, la planta de velocidad de giro hace referencia a la función de transferencia entre la tensión diferencial de los motores y la velocidad de giro. A continuación, se muestran cada una de las plantas del vehículo:

$$P_v = \frac{0.087298}{(1 + 0.04s)(1 + 0.0531s)}$$

$$P_w = \frac{0.68781}{(1 + 0.03403s)(1 + 0.04s)}$$

Por otro lado, a continuación se muestran las función de transferencia entre el ángulo de guiñada del vehículo y la tensión diferencial de los motores:

$$P_{\mathrm{ud-\theta}} = \frac{0.68781}{s(1 + 0.03403s)(1 + 0.04s)}$$

Se puede observar como se trata de la misma función de transferencia P_w, con un polo nulo adicional.

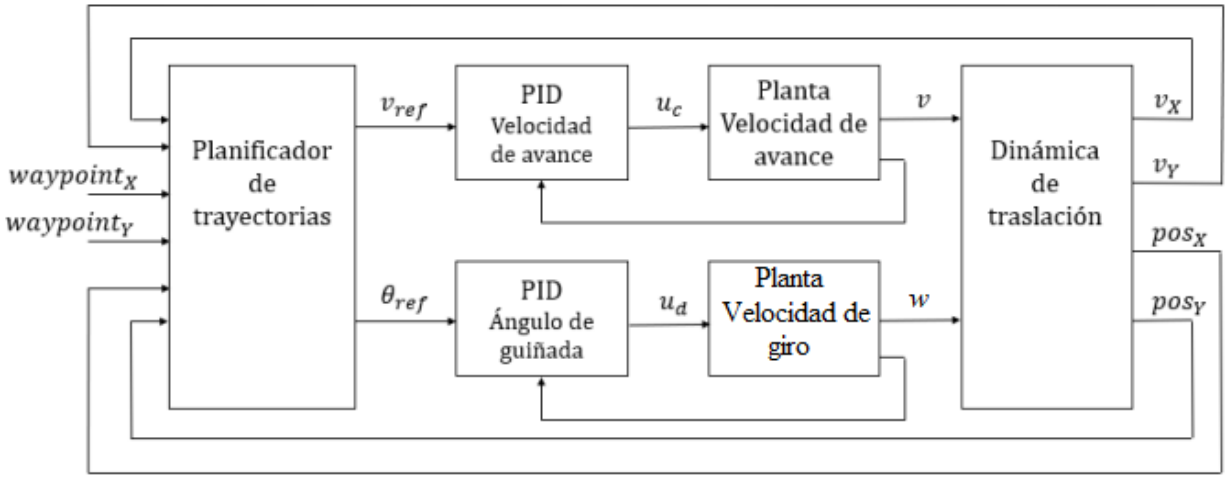


Figura 5.1: Estructura del control del vehículo.

5.2 LAZOS DE CONTROL DE AVANCE Y GIRO

En este proyecto se han empleado los controles de avance y de giro ya diseñados en proyectos anteriores. En total se han empleado tres controles: control de velocidad de avance, control de velocidad de giro y control del ángulo de giro. Para todos los controles se ha seleccionado el control de tipo PID.

5.2.1 CONTROLES DE VELOCIDAD

Tanto el control de velocidad de avance como el de velocidad de giro se diseñaron por margen de fase puesto que es más restrictivo que el margen de ganancia [29]. Los parámetros de los controles son los siguientes:

- Margen de fase ($\emptyset_m$) = 60º
- W_o/W_{o_p} = 1,4
- Factor de filtrado = 0,1
- Ponderación a la referencia (b) = 1
- Retraso de fase por acción integral = -10º

5.2.2 CONTROL DEL ÁNGULO DE GUIÑADA

Este control como los dos anteriores, se ha diseñó por margen de fase y sus parámetros son los siguientes:

- Margen de fase ($\emptyset_m$) = 55º
- W_o/W_{o_p} = 1,4
- Factor de filtrado = 0,1
- Ponderación a la referencia (b) = 0,75
- Retraso de fase por acción integral = -10º

5.3 SIMULINK PARA EL CONTROL DEL VEHÍCULO

Para poder llevar a cabo el desarrollo del control de velocidad de avance, el control de velocidad de giro y el control del ángulo de guiñada, se ha empleado el fichero de Simulink *CAR_CONTROL_SYSTEM.slx* de la *Figura 5.2*. En este fichero, la información del control del vehículo se almacena en los tres buses CONTROL, MSG Y MODEL. De esta forma, se tiene un mayor acceso y control de todas las variables contenidas en los buses.

- El bus CONTROL, almacena las variables propias al control del vehículo, donde se incluyen la información de los diferentes sensores, las medidas estimadas del Filtro de Kalman Extendido y los valores de las salidas y de los mandos del control.
- El bus MSG almacena toda la información relacionada con las comunicaciones entre el PC de monitorización y la Raspberry Pi del vehículo.
- El bus MODEL contiene los parámetros necesarios para el modelado de los sensores, actuadores y del vehículo.

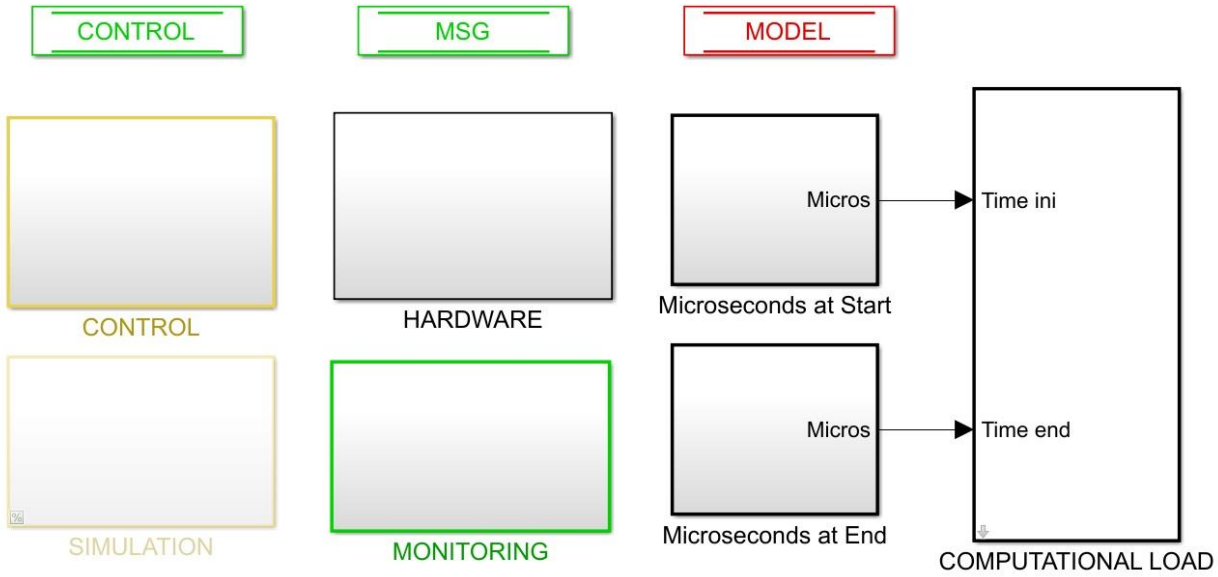


Figura 5.2: Diagrama Simulink CAR_CONTROL_SYSTEM.slx.

Por otro lado, el fichero está compuesto por un total de cuatro subsistemas: CONTROL, HARDWARE, SIMULATION y MONITORIZATION.

- El subsistema *Control*, *Figura 5.3*, contiene la información relacionada con la máquina de estados y el control del vehículo. A partir de esta información, se pueden calcular las estimaciones del Filtro de Kalman Extendido.

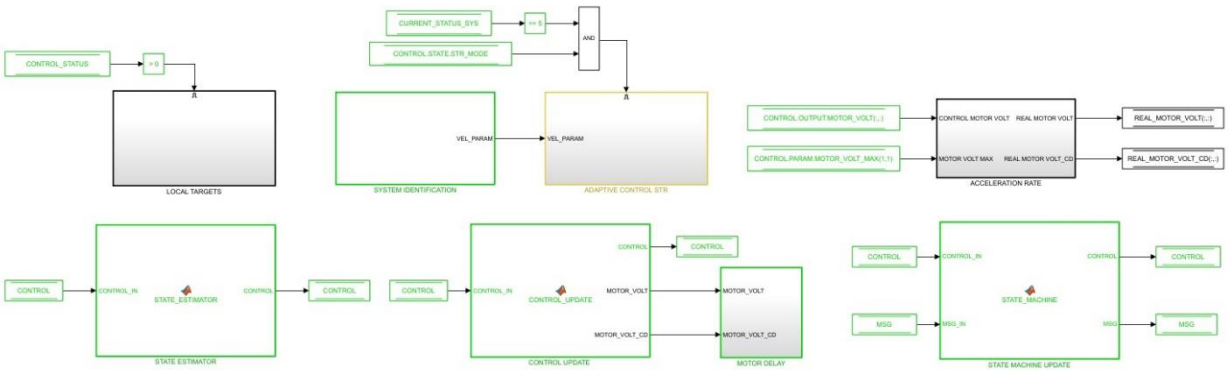


Figura 5.3: Subsistema Simulink CONTROL.

- El subsistema *Hardware*, *Figura 5.4*, alberga la información relevante a los sensores del vehículo, los actuadores, y las comunicaciones del mismo. Este subsistema se encuentra dividido en otros tres subsistemas: *Sensors*, *Actuators y Communications*.

Estos almacenan el código fuente necesario para controlar los elementos del hardware, así como las comunicaciones del vehículo.

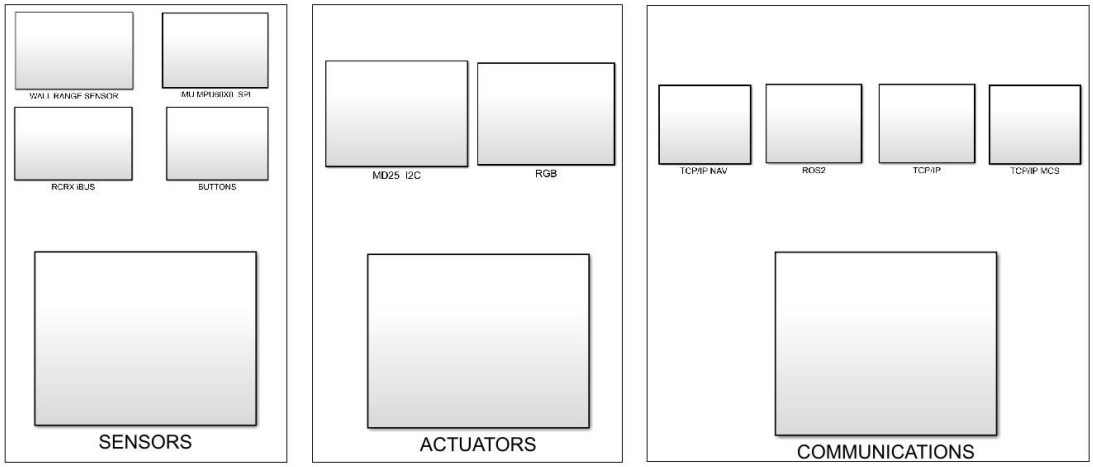


Figura 5.4: Subsistema Simulink HARDWARE.

- El subsistema *Simulation*, *Figura 5.5*, contiene la configuración del funcionamiento de los tres botones del vehículo, que se utilizan durante los ensayos para poner en marcha, pausar y finalizar el ensayo. A su vez, contiene funciones de Matlab donde se definen los modelos del vehículo y sus sensores.

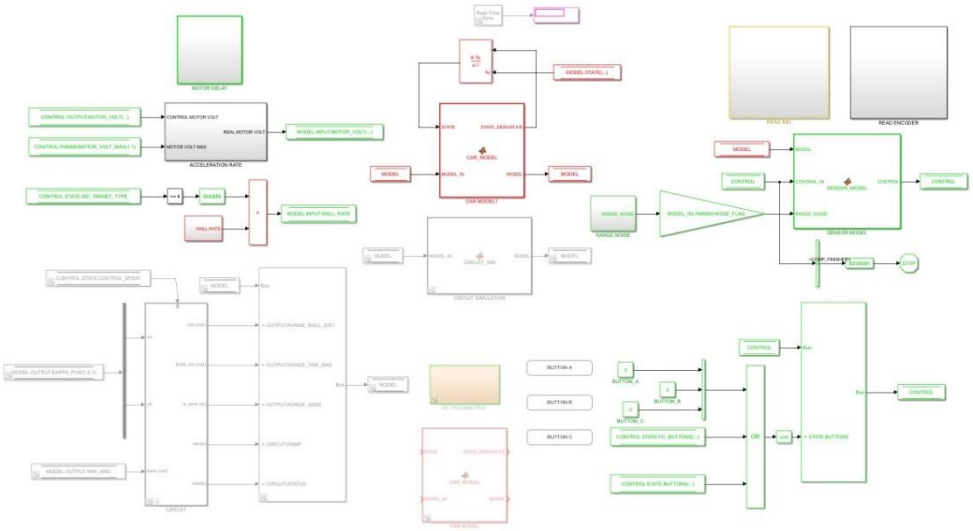


Figura 5.5: Subsistema Simulink SIMULATION.

- El subsistema *Monitorization* se utiliza para monitorizar y registrar las diferentes variables que sean interesantes para el desarrollo del proyecto. Este subsistema se encuentra compuesto por otros seis subsistemas que muestran a través de *displays* las variables de interés.

CAPÍTULO 6: COMUNICACIONES

En este apartado, se procede a describir y explicar el diseño de los diferentes métodos de comunicación del proyecto. Para el correcto funcionamiento del proyecto, ha sido necesario diseñar una estructura de comunicaciones que permita en todo momento establecer las comunicaciones entre el ordenador receptor de la información proveniente de las cámaras, el vehículo, el kit de navegación y el ordenador encargado de la monitorización de la navegación.

En primer lugar, se va a realizar una explicación de la transmisión de datos del sistema de localización de marcadores ArUco al vehículo. Esta transmisión de datos se llevará a cabo mediante dos protocolos diferentes: TCP/IP y ROS.
A continuación, se explica la comunicación entre el ordenador receptor de la información de las cámaras y el vehículo. Para dicha comunicación, es necesario el diseño de un sistema Bluetooth. Por último, se explica la transmisión de la información entre el vehículo y el ordenador de monitorización. Dicha comunicación se realizará vía Wi-Fi.
En la *Figura 6.1* se muestra un esquema de las comunicaciones entre el vehículo, el sistema de cámaras externo, el PC de monitorización y el PC de rastreo.

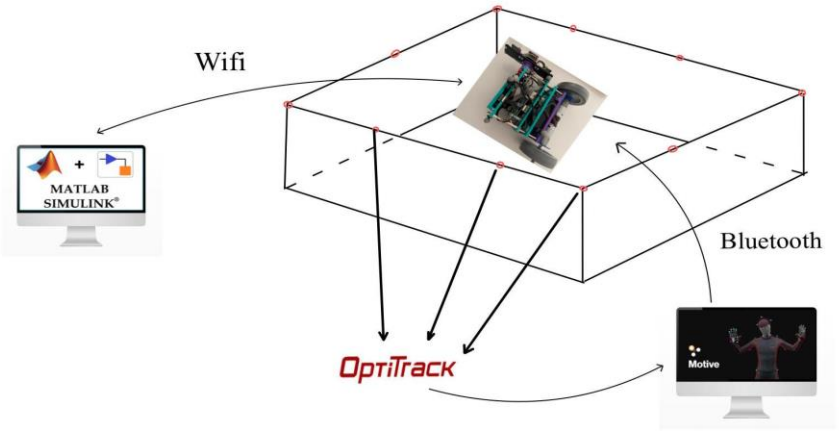


Figura 6.1: Esquema de comunicaciones.

6.1 ROS2 FOXY

ROS (Robot Operation System) es un conjunto de librerías y herramientas de software para construir aplicaciones robóticas. Además ROS consta de las herramientas de código abierto necesarias para el desarrollo de este proyecto. ROS se inició en 2007 y desde entonces se encuentra en continua evolución, por lo que el objetivo de ROS2 es adaptarse a los cambios que demanda la comunidad robótica mejorando las características del exitoso ROS1 [30]. Además ROS2 trata de ofrecer una plataforma estándar a los desarrolladores de cualquier rama de la industria que les ofrezca un mismo entorno desde las fases de investigación hasta desarrollo y producción. La distribución por la que se optó en este proyecto fue ROS2 Foxy.

ROS2 emplea el término nodos y servicios para explicar el funcionamiento de las comunicaciones. En primer lugar, las comunicaciones mediante ROS se dan entre diferentes nodos. Cada nodo es capaz de recibir y transmitir información a otros nodos a través de temas (*topics*).

En este proyecto, el esquema de comunicaciones ROS2 consta de los siguientes elementos:

- Nodo 1: Raspberry Pi 3B+ del kit de navegación.
- Nodo 2: Raspberry Pi 3B+ del vehículo.
- Tema *pilocation*: Ambos nodos se comunicarán a través del tema *pilocation*, este tema alberga la información acerca de la posición y orientación de la cámara. El nodo 1 será el que publica el tema (envía información) y el nodo 2 el que se subscribe al tema (recibe información).

Antes de todo, es necesario instalar ROS2 Foxy en ambas Raspberries. El proceso de instalación de ROS2 se encuentra explicado en el *ANEXO C: INSTALACIÓN ROS2 FOXY*.

6.2 COMUNICACIÓN ENTRE EL SISTEMA DE LOCALIZACIÓN Y EL VEHÍCULO

En este canal de comunicación, se transmite la información relevante a la posición y orientación del sensor óptico. El emisor de la comunicación es la Raspberry Pi del kit de navegación y el receptor la Raspberry Pi del vehículo. En primer lugar, se optó por realizar la comunicación mediante TCP IP, aunque también se ha desarrollado la comunicación mediante ROS 2 (Robot Operation System). Es por ello, que las dos tecnologías de comunicación son las siguientes:

1. Comunicación ROS.
2. Comunicación TCP IP mediante cable Ethernet.

En la comunicación TCP IP, el servidor empleará un tiempo de muestreo de diez milisegundos, es decir, cada diez milisegundos tratará de captar la información del *buffer* de recepción. No obstante, desde el cliente (Raspberry de localización) el tiempo de detección de los marcadores ArUco y el procesamiento de las imágenes es superior a diez milisegundos (27-40 ms) [11]. Es por ello, que cuando el servidor trate de captar la información y no haya datos en el *buffer*, se utilizará la última medida procesada. De manera similar, en la comunicación mediante ROS, la Raspberry de localización publicará la información cada diez milisegundos teniendo en cuenta que el tiempo de procesamiento de las imágenes es superior, por lo que se repetirán medidas en el envío. Desde la Raspberry del vehículo (nodo suscriptor) se obtendrá las medidas posicionales de la cámara cada diez milisegundos.

6.2.1 COMUNICACIÓN ROS

En primer lugar, para familiarizarse con ROS2, se llevaron a cabo pruebas entre la Raspberry del algoritmo de localización y un PC de manera local. Para ello, fue necesario instalar ROS tanto en la Raspberry (Linux) como en el PC (Windows). Una vez instalado ROS2 en ambos dispositivos, se procede a lanzar un entorno de comunicación de ROS. La Raspberry (*node Publisher*) manda la información a través de un *topic.* Por otro lado,

el PC (*node subscriber*) se subscribe al *topic* para acceder a dicha información. La información en cuestión se trata de un vector de 6 posiciones (*Twist*) que alberga la posición y orientación de la cámara. Para comprobar el correcto funcionamiento de la comunicación, se ha instalado en el PC Matlab 2022a, que dispone de ROS *Toolbox*, una librería con diferentes funciones para intercambiar datos con robots físicos compatibles con ROS2 [31].

A continuación, se conectan tanto la Raspberry como el PC bajo la misma red Wifi y se lanza desde la raspberry un *topic* llamado *pilocation* [32]. Desde Matlab ejecutamos el siguiente comando y podemos observar como el PC reconoce el topic *pilocation* y puede subscribirse a él.

```
>> ros2 topic list
/parameter_events
/pilocation
/rosout
```

Una vez terminadas las pruebas de manera local, se procede a probar la comunicación entre ambas Raspberries. Para ello, es necesario que ambas Raspberries tengan instalado ROS2.

El primer paso para llevar a cabo una comunicación a través del cable Ethernet es fijar a cada Raspbery una dirección IP estática. En el capítulo 6.2.2, se describe el proceso para asignar una dirección IP estática a cada Raspberry y se comprueba que ambas Raspberries se comunican correctamente.

Tras el proceso de asignar una IP estática para cada Raspberry, la Raspberry de localización se encuentra asociada a la dirección 192.168.0.124 y la Raspberry del vehículo a la dirección 192.168.0.124.

A continuación, se ejecuta ROS en la Raspberry de localización para publicar un mensaje de tipo *Twist* que albergue la posición y orientación de la cámara en el *topic* llamado "pilocation" mediante el siguiente comando:

```
ros2 run py_pubsub talker
```

Una vez construido el nodo de ROS en la Raspberry de localización, se procede a construir el nodo de ROS en la Raspberry del vehículo. Para ello, desde Simulink se usa la librería *ROS Toolbox* para construir un diagrama que trabaje como suscriptor. Simulink será el encargado de generar el código correspondiente en C++ a partir del diagrama de bloques y a continuación lo volcará en la Raspberry Pi. Previamente es necesario comunicarle a Simulink en qué robot se va a construir el nodo. En la *Figura 6.2,* se muestra el proceso de configuración del robot y una serie de mensajes en la terminal que aseguran la exitosa comunicación.

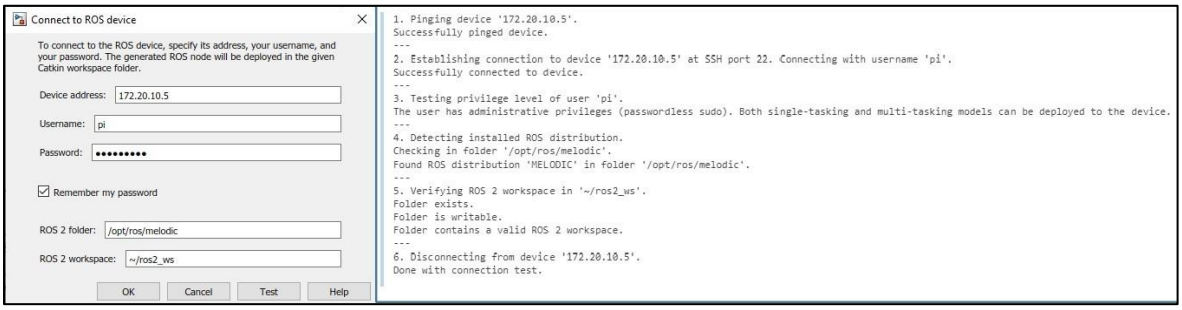


Figura 6.2: Configuración ROS2 en Simulink.

Por último, en la *Figura 6.3* se muestra una captura del diagrama de Simulink una vez construido el nodo suscriptor de ROS, donde se puede observar como el nodo se ha suscrito correctamente al *topic* "*pilocation*" y muestra los valores de la posición y orientación de la cámara.

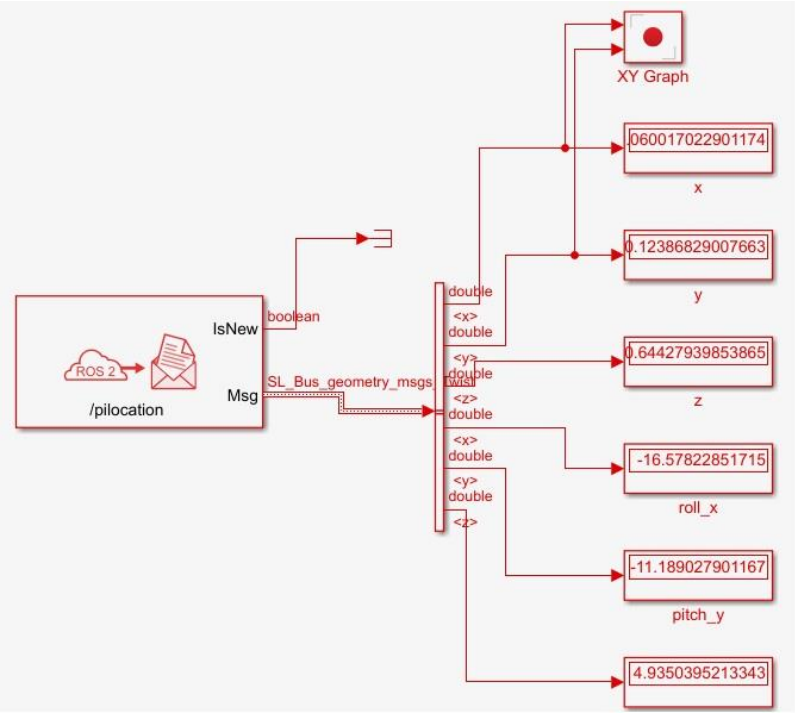


Figura 6.3: Diagrama ROS2 en SImulink.

6.2.2 COMUNICACIÓN TCP/IP

TCP/IP, *Protocolo de control de transmisión/Protocolo de Internet,* es un conjunto de reglas que permiten a los equipos comunicarse en una red como Internet. El protocolo TCP/IP se basa en la identificación del grupo de protocolos de red que hacen posible la transferencia de datos en redes entre equipos informáticos [33]. Dichos protocolos son los siguientes:

- TCP: Protocolo de Transmisión que permite establecer una conexión y el intercambio de datos entre dos anfitriones.
- IP: Protocolo de Internet que utiliza direcciones series de cuatro octetos con formato de punto decimal. Este protocolo transfiere los datos a otra máquina de la red.

Con el objetivo de conseguir un intercambio fiable de datos en una misma red, el protocolo TCP/IP emplea un sistema de jerarquías (se construye una capa a continuación de la anterior) que se comunican únicamente con su capa superior (envía datos) y su capa inferior (solicita datos).

Aunque ambas Raspberries se encuentren conectadas bajo la misma red WIFI, la comunicación se llevará a cabo mediante cable Ethernet para reducir el tiempo de latencia y así conseguir una mejor comunicación en tiempo real.

Cada vez que una Raspberry se conecta a la red WIFI, se le asigna de manera automática una dirección IP. Es por ello, que se le asignará a cada Raspberry una dirección IP estática, de manera que no cambie con cada conexión a la red. Al tener cada Raspberry un sistema operativo diferente, el proceso de asignar una IP fija se lleva a cabo de manera distinta.

En la Raspberry de localización (Ubuntu 20.04 LTS), se debe modificar el archivo */etc/netplan/50-cloud-init.yaml*, añadiendo las siguientes líneas de comandos:

```
ethernets:
    eth0:
       dhcp4: no
       addresses: [192.168.0.124/24]
       gateway4: 192.168.0.1
       nameservers:
         addresses: [192.168.0.1]
```

En la Raspberry del vehículo (Raspberry Pi OS Buster 32bits), es necesario modificar el archivo */etc/dhcpcd.conf*, añadiendo las siguientes líneas de comandos:

```
interface eth0
static ip_address=192.168.0.123/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1
```

Tras reiniciar ambas Raspberries, la Raspberry de localización queda asociada a la dirección IP 192.168.0.124 y la Raspberry del vehículo a la dirección 192.168.0.123. A continuación, se conectan ambas Raspberries a través del cable Ethernet y se valida la comunicación empleando el comando *ping*:

```
pi@ubuntu:~$ ping 192.168.0.123
PING 192.168.0.123 (192.168.0.123) 56(84) bytes of data.
64 bytes from 192.168.0.123: icmp_seq=1 ttl=64 time=0.667 ms
64 bytes from 192.168.0.123: icmp_seq=2 ttl=64 time=0.612 ms
64 bytes from 192.168.0.123: icmp_seq=3 ttl=64 time=0.572 ms
64 bytes from 192.168.0.123: icmp_seq=4 ttl=64 time=0.550 ms
^C
--- 192.168.0.123 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3078ms
rtt min/avg/max/mdev = 0.550/0.600/0.667/0.044 ms
```

Figura 6.4: Conexión Raspberry1-Raspberry2.

```
pi@raspberrypi-MuOHijZJct:~ $ ping 192.168.0.124
PING 192.168.0.124 (192.168.0.124) 56(84) bytes of data.
64 bytes from 192.168.0.124: icmp_seq=1 ttl=64 time=0.578 ms
64 bytes from 192.168.0.124: icmp_seq=2 ttl=64 time=0.581 ms
64 bytes from 192.168.0.124: icmp_seq=3 ttl=64 time=0.576 ms
64 bytes from 192.168.0.124: icmp_seq=4 ttl=64 time=0.577 ms
^C
--- 192.168.0.124 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 139ms
rtt min/avg/max/mdev = 0.576/0.578/0.581/0.002 ms
```

Figura 6.5: Conexión Raspberry2-Raspberry1.

Tanto en la *Figura 6.4* como en la *Figura 6.5* se imprimen sucesivos mensajes de conexión exitosos, lo que significa que ambas Raspberries se pueden comunicar entre sí.

Una vez ambas Raspberries pueden comunicarse entre sí, es necesario establecer el papel de cada Raspberry en la comunicación. La Raspberry del vehículo ejercerá de servidor, habilitando un canal de comunicación a través del cable Ethernet. El puerto local de comunicación es el 3000.

Por otro lado, conociendo la dirección IP del servidor y el puerto de comunicación, la Raspberry de localización ejerce de cliente.

Desde la Raspberry del vehículo se puede lanzar un servidor TCP/IP a través de un bloque de Simulink (TCP/IP Receiver), como se muestra en la *Figura 6.6.*

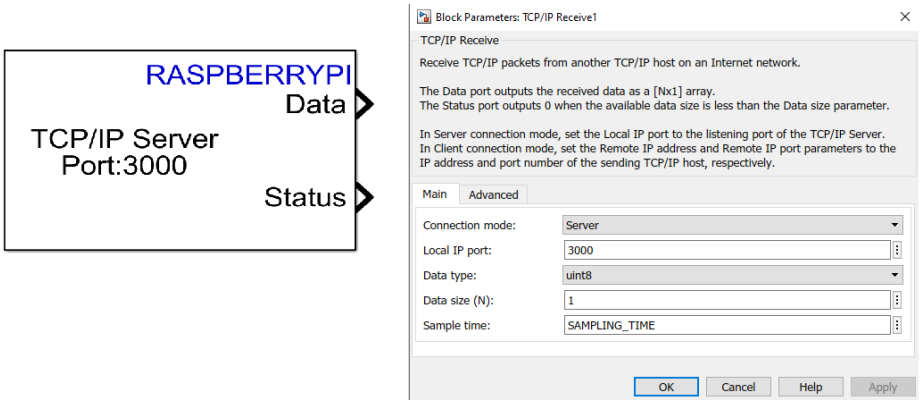


Figura 6.6: Configuración servidor TCP/IP.

A su vez, desde Python se lanza un cliente utilizando la librería *socket* a partir del siguiente comando:

```
s = socket.socket()
s.connect((192.168.0.123, 3000))
```

Una vez construidos el servidor y el cliente, se establece una conexión entre ambos y se procede a la transmisión del mensaje. El mensaje albergará la posición (x, y, z) y la orientación (*roll_x*, *pitch_y*, *yaw_z)* y tendrá un formato de vector de *bytes*. Cada elemento del mensaje es una variable de tipo *float* por lo que ocupa un tamaño de 4 *bytes* (32 *bits*). Es por ello, que el mensaje codificado estará compuesto por 24 *bytes*. La codificación del mensaje se realizará en el lado del cliente utilizando la librería *struct* partir del siguiente código fuente:

```
data = bytearray(struct.pack(">ffffff",camera_x, camera_y, camera_z, roll_x, pitch_y,
yaw_z)).hex()
data_cod = bytes.fromhex(data)
```

Antes de proceder a una comunicación dinámica entre el vehículo y el kit de navegación, se ha realizado una prueba inicial para comprobar la correcta comunicación entre ambos dispositivos. La prueba consiste en lanzar el servidor y que el cliente se conecte y le envíe el *byte* 127. En la *Figura 6.7*, se muestra como el servidor recibe el *byte* 127.

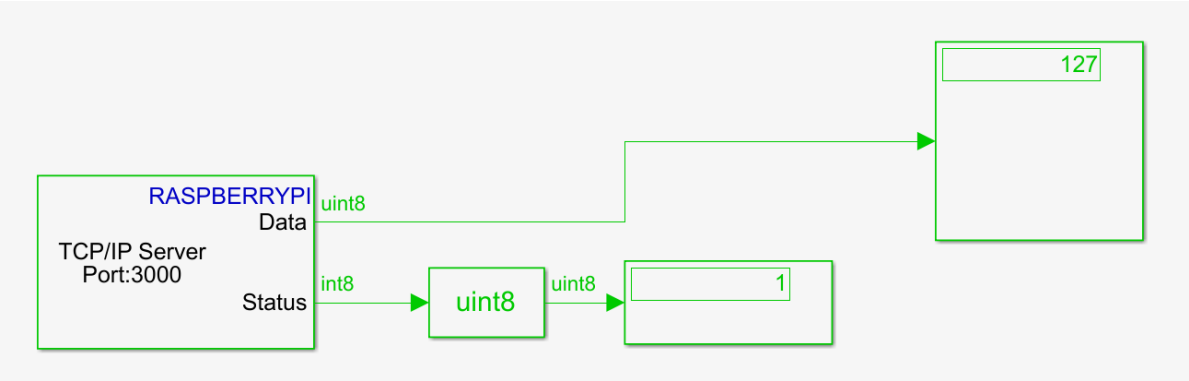


Figura 6.7: Prueba inicial comunicación TCP IP.

A continuación se procede a enviar el vector de 24 bytes con la información posicional de la cámara. En la *Figura 6.8*, se muestra una captura del diagrama de Simulink donde se observa que el mensaje ha sido recibido y decodificado.

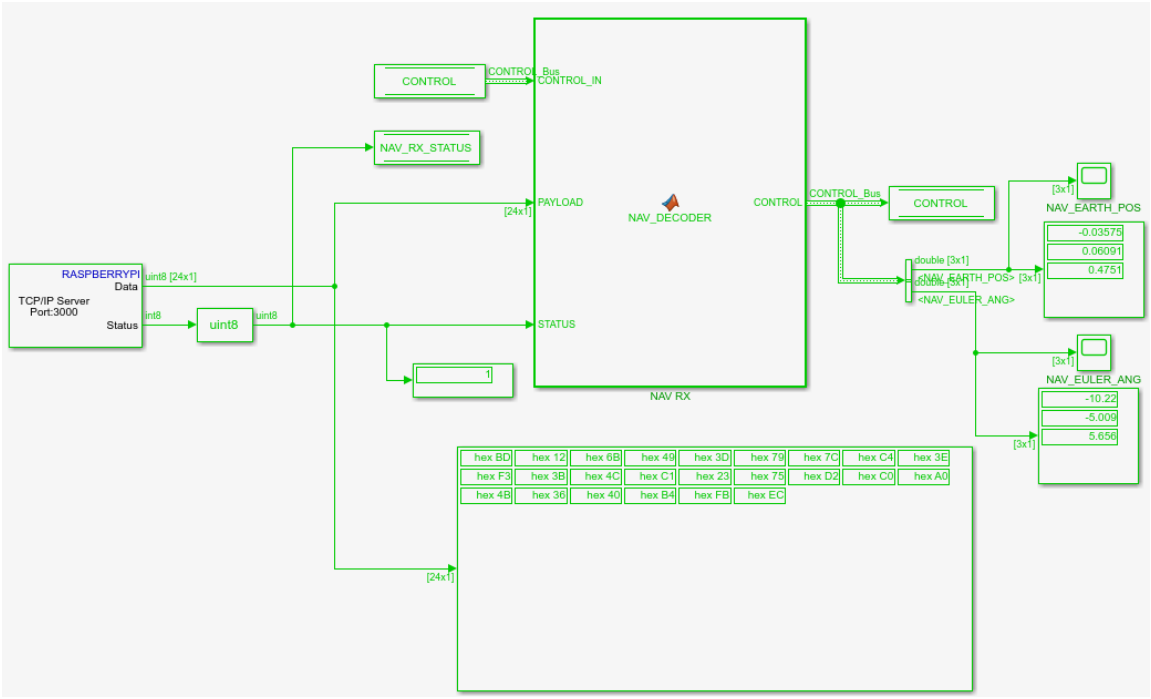


Figura 6.8: Prueba final comunicación TCP IP.

6.3 COMUNICACIÓN ENTRE MOTIVE Y SIMULINK

Como se ha explicado en el capítulo 4.3.2, Motive es una herramienta software que permite hacer un seguimiento del vehículo y obtener las coordenadas del mismo en el espacio tridimensional. Dichas coordenadas son la información requerida para poder caracterizar la precisión y fiabilidad del sistema de posicionamiento mediante marcadores ArUco. Estas coordenadas son enviadas al vehículo a través de Simulink desde el PC de rastreo. No obstante, previamente es necesario establecer el sistema de comunicación entre Motive y Simulink.

El sistema de comunicación se basa en un conjunto de herramientas cliente-servidor que emplean protocolo UDP (User Diagram Protocol). La gran ventaja de utilizar el protocolo UDP, es que este no necesita establecer una conexión entre el remitente y el receptor antes de que se envíen los datos [34]. De esta manera, es posible transmitir la información sin tener que esperar a una confirmación de la conexión. En este canal de comunicación, el software Motive actuará de servidor (envía datos) y el software Simulink de cliente (recibe datos).

6.3.1 DIAGRAMA DE SIMULINK PARA LA RECEPCIÓN Y ENVÍO DE DATOS DEL SISTEMA DE CÁMARAS OPTITRACK

Con el objetivo de recibir la información procedente del software Motive, procesar dicha información, y posteriormente transmitirla al vehículo, se ha empleado el diagrama de Simulink que se muestra en la *Figura 6.9*. Dicho diagrama de Simulink se encuentra en el fichero *MCS_CONTROL_STATION.slx.*

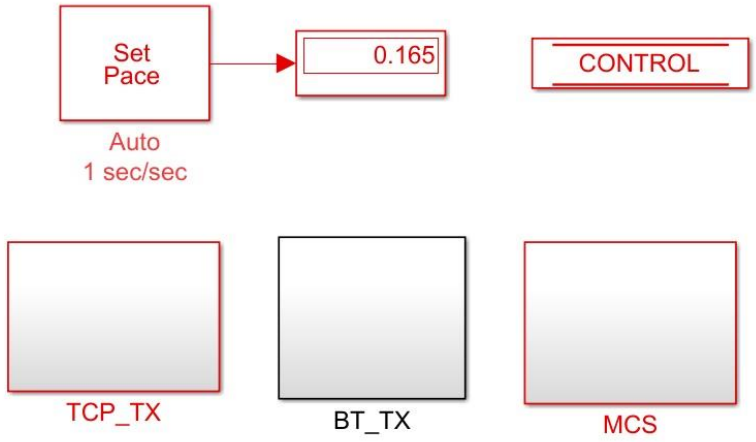


Figura 6.9: Diagrama de Simulink MCS_CONTROL_STATION.slx.

En este diagrama de Simulink, a través del Bus de Control, podemos tener controladas y almacenadas todas las variables y acceder a ellas para su tratamiento desde cualquier otra parte del programa. Este bus, se debe definir en los archivos de configuración, estableciendo sus dimensiones y el tipo de dato que va a almacenar.

Por otro lado, podemos observar tres subsistemas; el subsistema TCP_TX, el subsistema *BT_TX* y el subsistema *MCS*.

- El subsistema *TCP_TX*, se encarga de transmitir las coordenadas del vehículo en tiempo real a la Raspberry Pi 3B+ del vehículo. Esta información es enviada vía TCP/IP a través del puerto 15000.
- El subsistema *BT_TX*, se encarga de transmitir las coordenadas del vehículo en tiempo real a la Raspberry Pi 3B+ del vehículo. Esta información es enviada vía Bluetooth a través del puerto COM elegido.
- El subsistema *MCS*, *Figura 6.10*, es el encargado de recibir las coordenadas del vehículo en tiempo real. Esta recepción de datos se da gracias a la función *FuncMotive*, que conecta el software de Motive con Simulink.

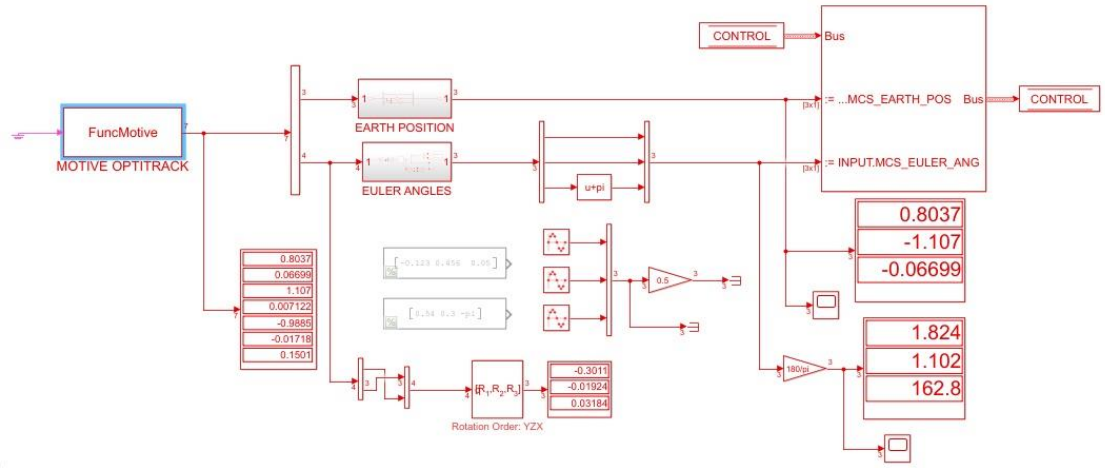


Figura 6.10: Subsistema MCS del diagrama MCS_CONTROL_STATION.slx.

La función *FuncMotive,* es un método perteneciente a la API facilitada por *Natnet,* que permite obtener las características del vehículo detectado por las cámaras. A partir de estas herramientas, se obtiene la posición tridimensional y los cuaterniones (orientación espacial). En la *Figura* 6.11, se puede observar el diagrama de Simulink que permite obtener los tres ángulos de Euler a partir de los cuaterniones.

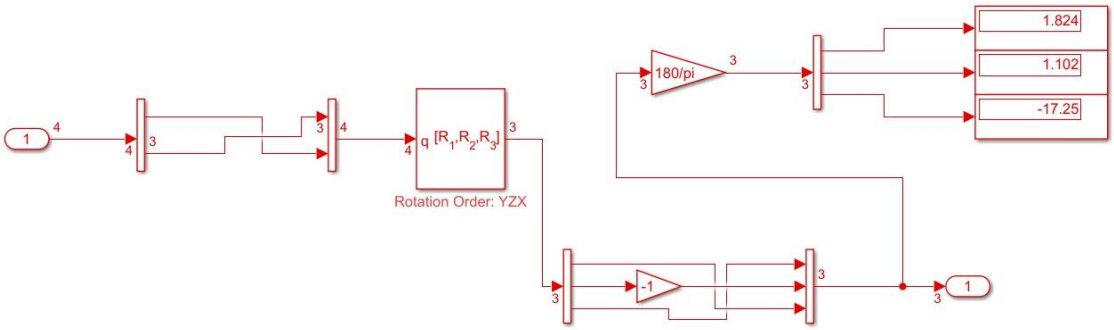


Figura 6.11: Bloque de Simulink "Euler Angles".

6.4 COMUNICACIÓN ENTRE EL PC DE RASTREO OPTITRACK Y EL VEHÍCULO

En este apartado se explica de forma detallada el sistema de comunicación empleado para transmitir los datos de las coordenadas de las cámaras a la Raspberry Pi del vehículo. Esta

comunicación se puede llevar a cabo vía Bluetooth o vía TCP/IP. La comunicación por Bluetooth se realiza a través de dos módulos Bluetooth HC05. En este proyecto se ha optado por realizar la comunicación vía Wi-Fi.

Para ello, se lanza el servidor en la Raspberry del vehículo que escucha a través del puerto 15000 y a continuación, se lanza el cliente desde el PC de rastreo para transmitir la información al puerto 15000 de la dirección IP del servidor (172.20.10.5). En la *Figura 6.12*, se muestran los diagramas de Simulink correspondientes a la comunicación TCP/IP y Bluetooth respectivamente.

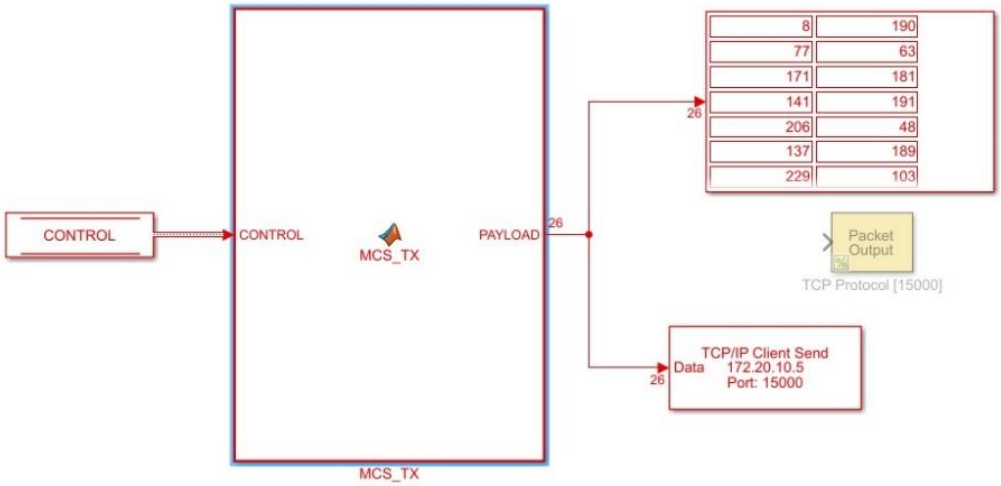


Figura 6.12: Diagrama de Simulink TCP/IP.

6.5 COMUNICACIÓN WI-FI ENTRE LA RASPBERRY DEL VEHÍCULO Y EL ORDENANDOR DE MONITORIZACIÓN

La comunicación entre el vehículo y el ordenador de monitorización se realiza vía Wi-Fi a través de un *router* inalámbrico. Tanto la Raspberry como el ordenador deben estar conectados al *router*. Es el *router* el encargado de asignar una dirección IP a cada uno de los dispositivos. Una vez, conocida la IP de cada dispositivo, se procede a realizar una conexión remota vía Wifi a través del protocolo SSH.

En el ordenador de monitorización se encuentran todos los ficheros de Simulink que contienen la lógica del vehículo, así como la monitorización de todas las variables de interés y la configuración de las referencias del vehículo. La comunicación entre ambos dispositivos se realiza mediante TCP IP a través del puerto 27000.

CAPÍTULO 7: ENSAYOS REALIZADOS Y RESULTADOS

En este capítulo se explican los diferentes ensayos realizados en el entorno de trabajo. Se pueden clasificar en dos tipos de ensayos:

1. Ensayos con los marcadores dispuestos en las paredes del entorno físico.
2. Ensayos con los marcadores dispuestos en el techo del entorno físico.

La primera disposición de los marcadores sobre la pared se ha utilizado para determinar el máximo alcance de detección de los marcadores y para caracterizar la precisión y fiabilidad del algoritmo de marcadores para el desplazamiento del vehículo en dirección horizontal y vertical. Para ello, se ha considerado la verdad absoluta las medidas de localización obtenidas a través del sistema capturador externo Optitrack.

Sin embargo, al posicionar los marcadores en las paredes, cuando el vehículo gira sobre su propio eje (Z), existe una mayor facilidad de que ningún marcador sea detectado y no se pueda estimar la localización y orientación del vehículo. Este inconveniente se puede remediar añadiendo tantos marcadores como sea necesario para cubrir el giro completo del vehículo sobre su propio eje.

Por otro lado, la configuración de los marcadores dispuestos en el techo se ha empleado para caracterizar la precisión del sistema de marcadores en la estimación del ángulo de guiñada del vehículo. En este caso, la distancia del vehículo al marcador permanece constante y tiene como valor la altura del techo del entorno de trabajo.

En todos los ensayos realizados se ha empleado el PC de monitorización para realizar una comparación entre las variables de localización de cada algoritmo de posicionamiento (ArUco y Optitrack). En el caso del algoritmo de los marcadores, estas medidas hacen referencia a la traslación y rotación del sensor óptico del vehículo. Por otro lado, las medidas del sistema de cámaras Optitrack, hacen referencia al centroide del vehículo. Es importante mencionar, que las primeras diez medidas que recibe el PC de monitorización se utilizan para realizar una media ponderada y establecer el posicionamiento inicial tanto de la cámara como del centroide del

vehículo. A su vez, ambos sistemas de referencia se trasladarán a la par ya que las matrices de traslación y rotación entre ellos son fijas.

7.1 ENSAYOS REALIZADOS CON LOS MARCADORES EN LAS PAREDES DEL ENTORNO DE TRABAJO

En primer lugar, se han dispuesto tres marcadores ArUco en las paredes del entorno de trabajo como se muestra en la *Figura 7.1*. Los números identificativos de estos marcadores son el 1, el 5 y el 30. Una vez colocados los marcadores, se ha establecido al marcador con número identificativo 30 como sistema de referencia.

Figura 7.1: Entorno de trabajo.

7.1.1 ALCANCE DE DETECCIÓN DE LOS MARCADORES

Este primer ensayo se ha realizado para conocer el alcance que tiene el sensor óptico para detectar un marcador a medida que se va alejando del él. En primer lugar, se coloca el vehículo a una distancia inicial de 1,67 metros del marcador. A continuación, se utiliza una radioemisora para ir

alejando el vehículo. Como se puede observar en la *Figura 7.2*, cuando el vehículo supera los tres metros de distancia, el ruido de la señal aumenta significantemente. Se puede concluir que la distancia óptima para que el sensor detecte un marcador (20 centímetros de lado) y estime correctamente su posición respecto de él, se encontraría en un rango de 0,5 a 2,5 metros de distancia.

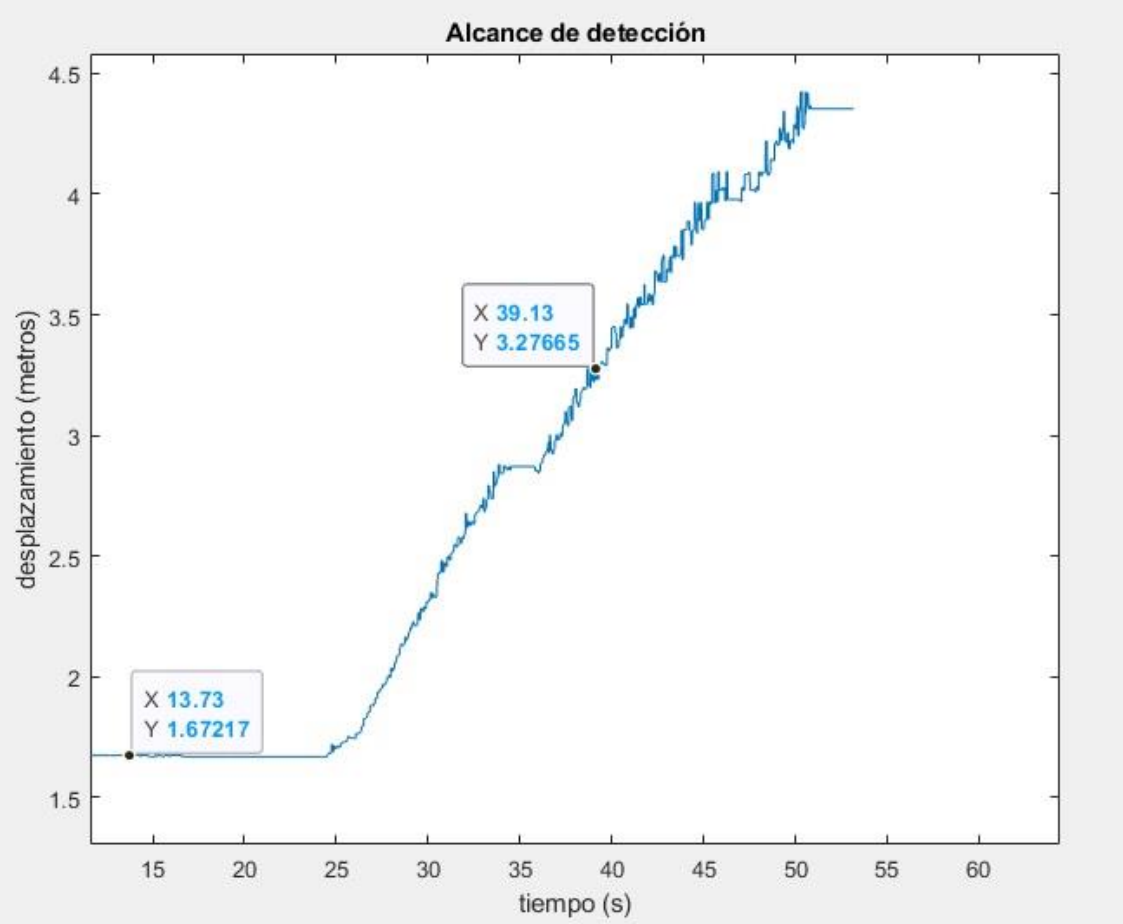


Figura 7.2: Alcance de detección de la cámara.

7.1.2 CARACTERIZACIÓN DE LA PRECISIÓN DEL ALGORITMO DE DETECCIÓN DE MARCADORES

En el primer ensayo se posiciona el vehículo a una distancia inicial de 1,5 metros del marcador con número identificativo 30. Una vez se termina de calcular en Simulink la posición inicial del vehículo, se procede a mover el vehículo a lo largo del eje de ordenadas del entorno de trabajo. En

la *Figura 7.3*, se pueden observar dos gráficas. La gráfica superior muestra una comparación entre la coordenada vertical del vehículo (y) para cada sistema de posicionamiento. La gráfica inferior muestra la diferencia entre ambas señales. Hasta el segundo 20, se calcula el *offset* de cada sistema de posicionamiento. Se puede observar como la diferencia en todo el ensayo se encuentra en torno a cero por lo que se puede afirmar que los marcadores ArUco ofrecen una gran precisión y fiabilidad en el desplazamiento vertical del robot móvil.

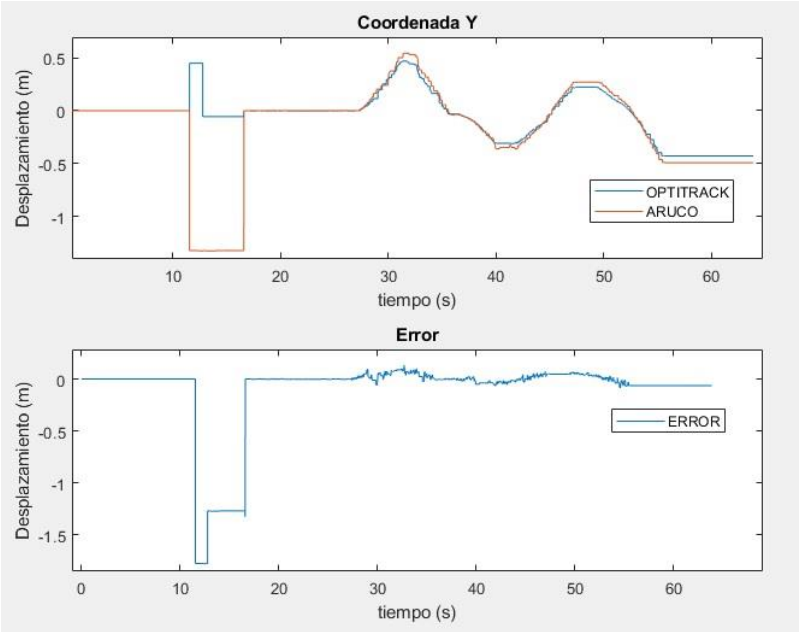


Figura 7.3: Caracterización coordenada vertical.

El siguiente ensayo se ha realizado para caracterizar la precisión del sistema de posicionamiento cuando el vehículo se desplaza de manera rectilínea a lo largo del eje de abscisas del entorno de trabajo. Para ello, se ha posicionado el vehículo a una distancia de 0,5 metros del marcador 5. En la *Figura 7.4*, se muestra el desplazamiento horizontal del vehículo (metros) a lo largo del tiempo (segundos). Se puede observar una gran similitud entre ambas gráficas ya que el error máximo entre ambas es de 20 centímetros.

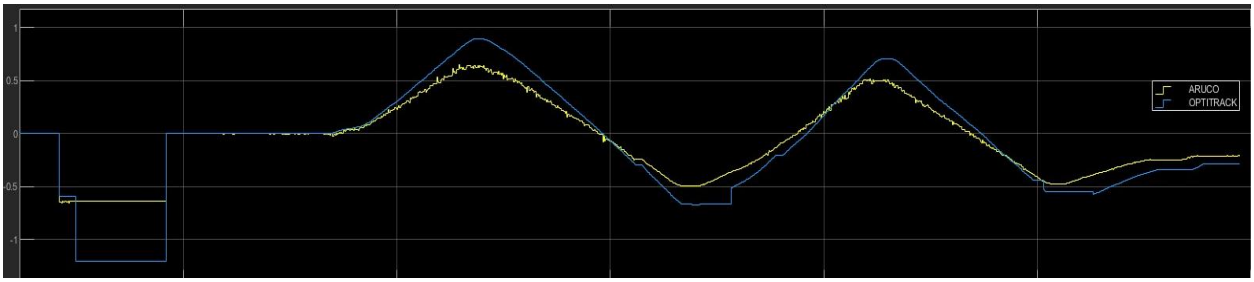


Figura 7.4: Caracterización coordenada horizontal.

El último ensayo consta en mover el vehículo de forma libre por el entorno de trabajo. Este ensayo es el más relevante de los cuatro ya que se le somete al vehículo a todo tipo de movimientos y de giros. Para ello se ha posicionado el vehículo en el centro del entorno de trabajo. En la *Figura 7.5* se pueden observar las gráficas correspondientes a la coordenada horizontal (gráfica superior) y a la coordenada vertical (gráfica inferior). Se puede encontrar grandes similitudes entre las medidas de las cámaras Optitrack y las medidas del algoritmo de marcadores, por lo que se puede afirmar que los marcadores ArUco ofrecen una gran precisión y fiabilidad en el posicionamiento. No obstante, cuando el vehículo se aleja de los marcadores la diferencia entre las gráficas aumenta. Esto se debe al problema de la ambigüedad explicado en el capítulo 4.2.

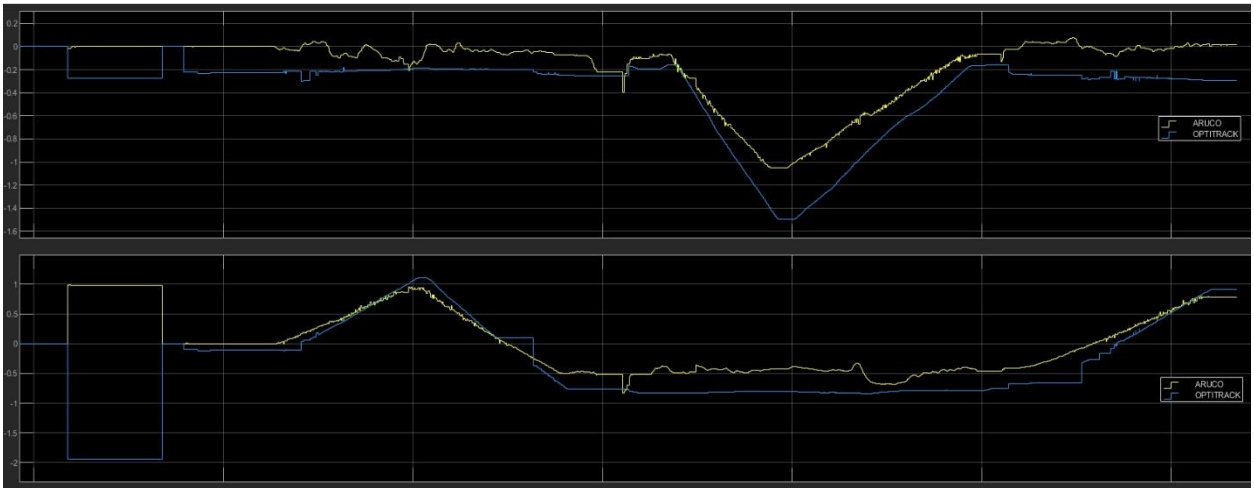


Figura 7.5: Caracterización ambas coordenadas.

7.2 ENSAYOS REALIZADOS CON LOS MARCADORES EN EL TECHO DEL ENTORNO DE TRABAJO

Esta disposición sería la más óptima para una navegación autónoma ya que el vehículo puede rotar sobre su propio eje giro y el sensor óptico siempre va a detectar al menos un marcador. Además, el techo del entorno tiene una altura de 2,40 metros por lo que no supone un problema en la detección del marcador. En la *Figura 7.6*, se muestra el marcador con número identificativo 5 dispuesto en el techo del entorno de trabajo.

Figura 7.6: Marcador número 5 dispuesto en el techo del entorno de trabajo.

7.2.1 CARACTERIZACIÓN ÁNGULO DE GUIÑADA DEL SISTEMA DE LOCALIZACIÓN

Para caracterizar la precisión del ángulo de guiñada del algoritmo de marcadores se ha empleado como referencia el sistema de cámaras Optitrack. A continuación, la gráfica de

la *Figura 7.7* hace referencia a la comparación de la medida del ángulo entre ambos algoritmos de localización.

Como se puede observar, ambas gráficas son muy parecidas, incluso la gráfica de los marcadores presenta menos ruido que la gráfica de las cámaras Optitrack.

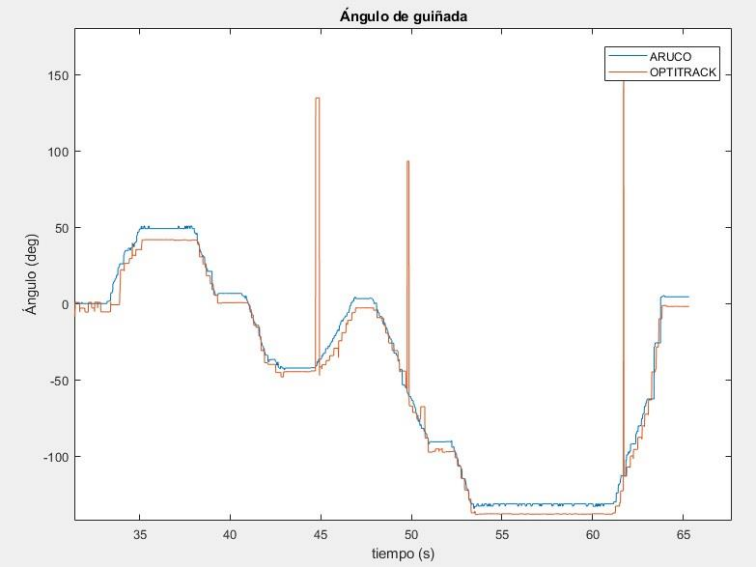


Figura 7.7: Caracterización de la precisión del ángulo de guiñada.

7.2.2 CONTROL DEL ÁNGULO DE GUIÑADA

Una vez se ha comprobado la precisión del algoritmo de marcadores, se procede a implantar en el vehículo el control sobre el ángulo de guiñada. Para ello se han diseñado dos controles: un control proporcional (P) y un control proporcional diferencial con acción integral (PID).

7.2.2.1 CONTROL PROPORCIONAL

A continuación, en la *Figura 7.8* se muestra el seguimiento del ángulo de guiñada dada una referencia. Se puede observar como el ángulo sigue a la referencia aunque es apreciable un cierto retraso debido al control.

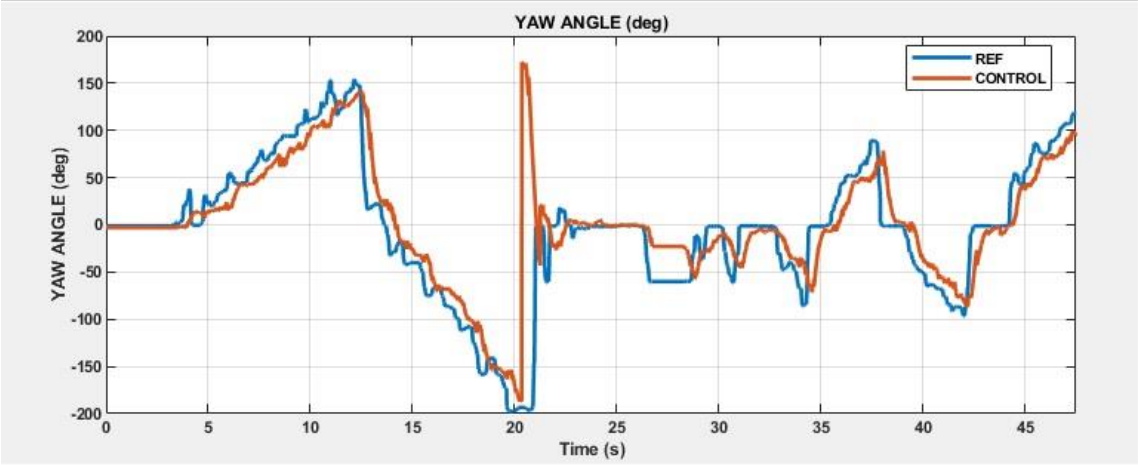


Figura 7.8: Control P ángulo de guiñada.

7.2.2.2 CONTROL PID

Con el objetivo de conseguir un menor tiempo de retraso entre el control y la referencia, se ha diseñado un control PID. Para el diseño del control se ha empleado el control explicado en el capítulo *5.2.2* **CONTROL DEL ÁNGULO DE GUIÑADA**

En la *Figura 7.9*, se muestra como se ha conseguido disminuir el retraso del control. No obstante, se puede observar como el control es más ruidoso.

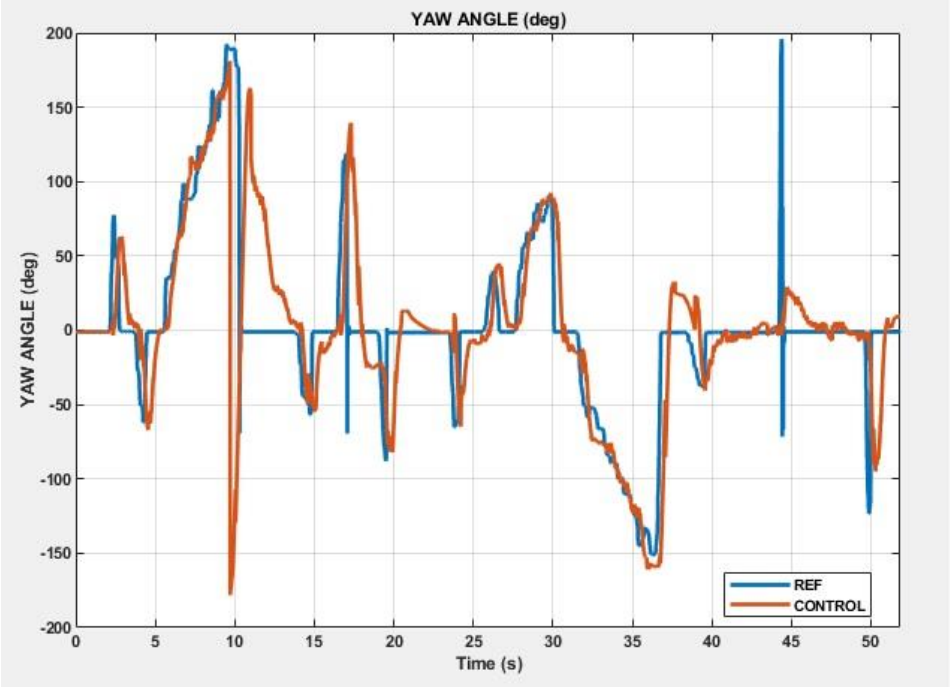


Figura 7.9: Control PID ángulo de guiñada.

REFERENCIAS

[1] R. P. Ltd, «Buy a Raspberry Pi 3 Model B+», *Raspberry Pi*. https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/ (accedido 4 de julio de 2022).

[2] «Unidad de medición inercial», *Wikipedia, la enciclopedia libre*. 2 de julio de 2021. Accedido: 4 de julio de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Unidad_de_medici%C3%B3n_inercial&oldid=136743735

[3] «openCv», *OpenCV*. https://opencv.org/ (accedido 4 de julio de 2022).

[4] «Motion Capture Systems», *OptiTrack*. http://optitrack.com/index.html (accedido 4 de julio de 2022).

[5] J. G. McNeff, «The global positioning system», *IEEE Transactions on Microwave Theory and Techniques*, vol. 50, n.º 3, pp. 645-652, mar. 2002, doi: 10.1109/22.989949.

[6] M. Cabezas Olivenza, «Desarrollo de algoritmos de localización y navegación de vehículos industriales en interiores», Escuela de Ingeniería de Bilbao, Trabajo Fin de Máster, 2020-2021.

[7] R. Muñoz-Salinas, M. J. Marín-Jimenez, E. Yeguas-Bolivar, y R. Medina-Carnicer, «Mapping and localization from planar markers», *Pattern Recognition*, vol. 73, pp. 158-171, 2018.

[8] A. Trujillo López, «Diseño, simulación y control de un astronauta robótico humanoide», Universidad de Alicante, Trabajo de Fin de Grado, Junio 2021.

[9] L. George y A. Mazel, «Humanoid robot indoor navigation based on 2D bar codes: Application to the NAO robot», en *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Atlanta Georgia USA, 2013, 15-17 Oct, pp. 329-335.

[10] «Humanoid Robot Indoor Navigation Based on 2D Bar Codes: (short corridor - high def) - YouTube». https://www.youtube.com/watch?v=p8m4wH2aO68&t=59s (accedido 6 de julio de 2022).

[11] G. C. La Delfa, S. Monteleone, V. Catania, J. F. De Paz, y J. Bajo, «Performance analysis of visualmarkers for indoor navigation systems», *Frontiers of Information Technology & Electronic Engineering*, vol. 17, n.º 8, pp. 730-740, 2016.

[12] M. Elgendy, T. Guzsvinecz, y C. Sik-Lanyi, «Identification of markers in challenging conditions for people with visual impairment using convolutional neural network», *Applied Sciences*, vol. 9, n.º 23, p. 5110, 2019.

[13] M. Kalaitzakis, S. Carroll, A. Ambrosi, C. Whitehead, y N. Vitzilaios, «Experimental comparison of fiducial markers for pose estimation», en *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE Robotics & Automation Society, Athens Greece, 2020, 1-4 Sept, pp. 781-789.

[14] V. Mazzari, «IMU and robotics: All you need to know», *Génération Robots - Blog*, 10 de febrero de 2020. https://www.generationrobots.com/blog/en/imu-and-robotics-all-you-need-to-know-2/ (accedido 6 de julio de 2022).

[15] «Geekworm Raspberry Pi UPS, X728 (MAX 5.1V 8A) 18650 UPS & Power Management Board with AC Power Loss Detection, Auto On & Safe Shutdown & RTC

Function for Raspberry Pi 4B/3B+/3B : Amazon.es: Informática». https://www.amazon.es/Geekworm-Raspberry-Management-Detection-Shutdown/dp/B087J7WTYM/ref=asc_df_B087J7WTYM/?tag=googshopes-21&linkCode=df0&hvadid=529543448548&hvpos=&hvnetw=g&hvrand=13218436716064233708&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9061040&hvtargid=pla-942743479094&psc=1 (accedido 6 de julio de 2022).

[16] «Multi Jet Fusion | Materialise - Innovators you can count on». https://www.materialise.com/es/manufacturing/tecnologia-de-impresion-3d/multi-jet-fusion (accedido 6 de julio de 2022).

[17] 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, doi: 10.1016/j.patcog.2014.01.005.

[18] «Raspberry Pi Camera Module V2 8MP : Amazon.es: Informática». https://www.amazon.es/Raspberry-Pi-Camera-Module-8MP/dp/B01ER2SKFS (accedido 6 de julio de 2022).

[19] «openCv», *OpenCV*. https://opencv.org/ (accedido 4 de julio de 2022).

[20] «OpenCV: opencv2/aruco/dictionary.hpp File Reference». https://docs.opencv.org/3.4/dc/df7/dictionary_8hpp.html (accedido 6 de julio de 2022).

[21] I. Lebedev, A. Erashov, y A. Shabanova, «Accurate autonomous uav landing using vision-based detection of aruco-marker», en *International Conference on Interactive Collaborative Robotics*, SPIIRAS and TUM, St. Petersburg Russia, 2020, 7-9 Oct, pp. 179-188.

[22] E. Sepúlveda Jorcano, «Localización automática de cámaras en sistemas de análisis de vídeo con múltiples vistas», B.S. Tesis de Licenciatura, Universidad Autónoma de Madrid, Escuela Politécnica Superior, Junio 2018.

[23] Z. Xu, M. Haroutunian, A. J. Murphy, J. Neasham, y R. Norman, «An underwater visual navigation method based on multiple ArUco markers», *Journal of Marine Science and Engineering*, vol. 9, n.º 12, p. 1432, 2021.

[24] A. Aalerud, J. Dybedal, y G. Hovland, «Automatic calibration of an industrial RGB-D camera network using retroreflective fiducial markers», *Sensors*, vol. 19, n.º 7, p. 1561, 2019.

[25] A. López-Cerón y J. M. Cañas, «Accuracy analysis of marker-based 3D visual localization», en *XXXVII Jornadas de Automática Jornadas de Automática*, Cómite Español de Automática, Madrid España, 2016, pp. 1124-1131.

[26] «OptiTrack Flex 13 Motion Capture Camera - Thinglab». https://tracklab.com.au/products/brands/optitrack/optitrack-flex-cameras/optitrack-flex-13/ (accedido 7 de julio de 2022).

[27] G. Nagymáté y R. M. Kiss, «Application of OptiTrack motion capture systems in human movement analysis: A systematic literature review», *Recent Innovations in Mechatronics*, vol. 5, n.º 1., pp. 1-9, 2018.

[28] A. Bilesan *et al.*, «Marker-based motion tracking using Microsoft Kinect», *IFAC-PapersOnLine*, vol. 51, n.º 22, pp. 399-404, 2018.

[29] A. Marroquín Rodríguez, «Navegación de robots móviles en entornos dotados de sistemas de localización externos», 2021.

[30] «ROS 2 Documentation — ROS 2 Documentation: Foxy documentation». https://docs.ros.org/en/foxy/index.html (accedido 7 de julio de 2022).
[31] «ROS Toolbox - MATLAB». https://es.mathworks.com/products/ros.html (accedido 7 de julio de 2022).
[32] «Writing a simple publisher and subscriber (Python) — ROS 2 Documentation: Foxy documentation». https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html (accedido 7 de julio de 2022).
[33] «IBM Docs», 19 de abril de 2022. https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/es/aix/7.2?topic=protocol-tcpip-protocols (accedido 7 de julio de 2022).
[34] «Protocolo de datagrama de usuario (UDP) (artículo)», *Khan Academy*. https://es.khanacademy.org/computing/ap-computer-science-principles/the-internet/x2d2f703b37b450a3:transporting-packets/a/user-datagram-protocol-udp (accedido 7 de julio de 2022).

ANEXO A: INSTALACIÓN UBUNTU 20.04 LTS

En este apartado se procede a explicar el procedimiento para instalar Ubuntu 20.04 LTS de 64 bits en la Raspberry Pi 3B+ que conforma el kit de navegación.

La decisión de utilizar este sistema operativo se basa en el alto grado de compatibilidad de Ubuntu con ROS. En primer lugar, se ha tenido que descargar desde la página oficial de Ubuntu la imagen del sistema operativo, para después escribirla en la tarjeta SD utilizando el programa Raspberry Pi Imager.

Una vez escrita la imagen, es necesaria la habilitación de la interfaz de SSH para conectarse de forma remota a la Raspberry Pi. SSH son las siglas de *Secure Shell,* y es un protocolo que requiere de autentificación, por lo que es necesario introducir el usuario y contraseña de la Raspberry Pi cada vez que se quiera acceder de forma remota a ella. A su vez, para poder realizar una conexión entre servidores, se necesitan tres factores fundamentales: usuario, puerto y servidor. El usuario es el usuario de la Raspberry, el puerto es el TCP 22 y el servidor es la dirección IP de la Raspberry.

En la página principal de *Raspberry Pi Imager*, hay que llevar a cabo los siguiente cuatro pasos:

1. Seleccionar Ubuntu Server 20.04 LTS en la pestaña del sistema operativo.(*Figura A.1*)

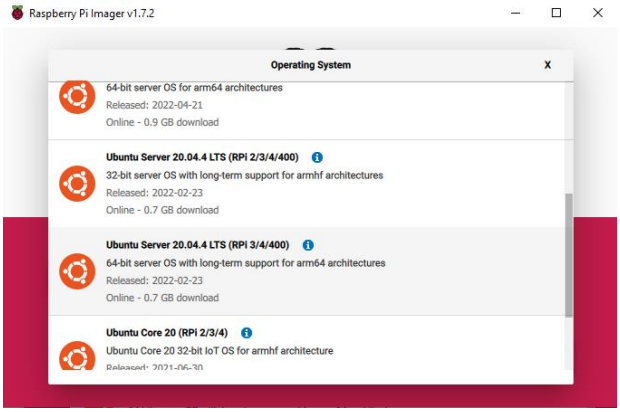


Figura A.1: Elegir Ubuntu 20.04 Server LTS.

2. Seleccionar el disco extraíble de la tarjeta SD en la pestaña de almacenamiento.

3. Habilitar el sistema de conexión de forma remota (SSH). (*Figura A.2*)

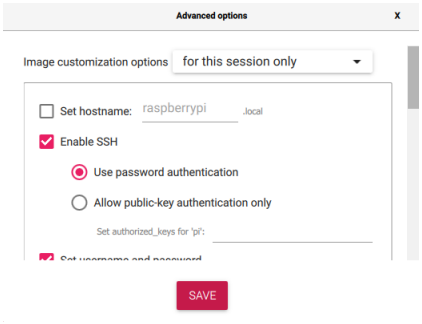


Figura A.2: Habilitar SSH.

4. Seleccionar el botón *Escribir* y esperar hasta que se complete la instalación de Ubuntu 20.04 en la tarjeta SD.

Una vez terminado el proceso de instalación, se puede extraer la SD del ordenador e introducirla en la Raspberry Pi 3B+. Se puede comprobar que el sistema operativo se ha instalado correctamente si al alimentar la Raspberry Pi, parpadea el led verde.

ANEXO B: INSTALACIÓN RASPBERRY PI OS BUSTER

Esta Raspberry es la que se encarga del control de navegación y por ello, habrá que implementarle desde Matlab el firmware del vehículo. Para ello, es necesario instalar una librería de Matlab llamada *Matlab Support Package for Raspberry Pi Hardware*, que permite la comunicación de la Raspberry de forma remota con un ordenador. Con esta librería se puede obtener información de los diferentes sensores conectados a la Raspberry mediante los pines GPIO, para después ser procesados en el entorno de Matlab. Para poder utilizar esta librería, es necesario instalar en la Raspberry la imagen customizada por *MathWorks*. Dicha imagen escribe Raspbian Buster de 32 bits. Una vez instalada la imagen de Raspbian customizada, se procede a la implementación del firmware. Para ello, desde Matlab se establece una conexión con la Raspberry Pi mediante el protocolo SSH, por lo que es necesario que tanto el PC como la Raspberry Pi se encuentren conectados en la misma red WIFI.

Para poder interactuar con la Raspberry desde Matlab y Simulink, es necesario instalar las siguientes librerías:

- MATLAB Support Package for Raspberry Pi Hardware
- Simulink Support Package for Raspberry Pi Hardware

Una vez instaladas, se procede a instalar el Sistema operativo en la tarjeta SD. Para ello, hay que seguir los siguientes pasos:

1. En la pestaña *Environment* de MATLAB, seleccionar *Add-Ons* y después *Manage Add-Ons.*
2. En la librería de *MATLAB Support for Raspberry Pi Hardware*, seleccionar el icono de Ajustes.
3. En la pestaña de la Figura X seleccionar el modelo de la Raspberry Pi. (*Figura B.1*)

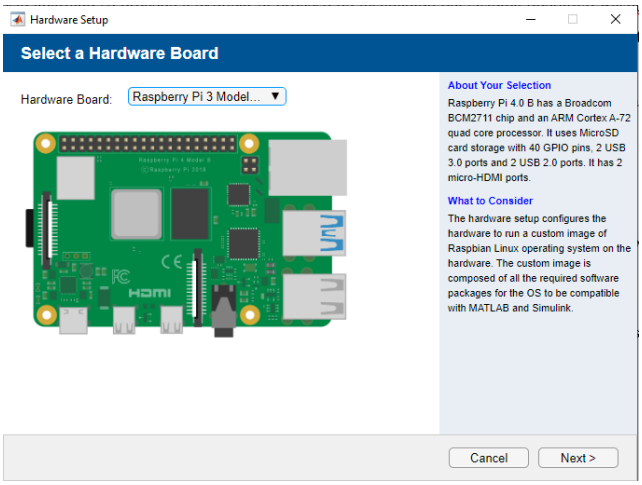


Figura B.1: Seleccionar el modelo de Raspberry Pi.

4. En la ventana siguiente se muestran dos opciones de descarga para la imagen de Raspbian. En este proyecto se ha escogido la opción que no contiene Deep Learning, puesto que no era necesaria. Tras descargarla, la imagen aparece en una carpeta .zip en la carpeta de "Descargas" de nuestro PC. En la siguiente ventana hay que seleccionar la imagen y darle a "Validate". Una vez validada la imagen, esta aparece con un tick. En la siguiente ventana, hay que escoger la opción de "*Connect* directly *to host Computer*" y elegir el puerto al que esté conectada la microSD. Una vez finalizado el proceso de carga de Raspbian en la microSD, aparece un mensaje indicando la finalización del proceso.

Una vez terminado el proceso de instalación se puede extraer la tarjeta SD del ordenador e introducirla en la Raspberry Pi 3B+ del vehículo.

ANEXO C: INSTALACIÓN ROS2 FOXY

En este apartado se procede a explicar el proceso de instalación de ROS2 Foxy en ambos dispositivos.

C.1 INSTALACIÓN EN RASPBERRY DEL KIT DE NAVEGACIÓN

Para este dispositivo, se ha seguido la guía de instalación oficial de ROS2 Foxy para Ubuntu (Debian). ROS2 Foxy Fitzroy se encuentra disponible para sistemas operativos con arquitecturas de 64 bits. Es por ello, que se ha escrito Ubuntu 20.04 LTS arm64 en la SD de la Raspberry Pi. Para instalar ROS2 Foxy es necesario acceder de forma remota a la Raspberry. A continuación se deben ejecutar los siguientes comandos.

```
$ sudo apt update
$ wget https://raw.githubusercontent.com/ROBOTIS-
GIT/robotis_tools/master/install_ros2_foxy_rpi.sh
$ chmod 755 .//install_ros2_foxy_rpi.sh
$ bash ./install_ros2_foxy_rpi.sh
$ sudo apt install python3-argcomplete python3-colcon-common-extensions libboost-
system-dev build-essential
```

C.2 INSTALACIÓN EN LA RASPBERRY DEL VEHÍCULO

Por otro lado, el proceso de instalación de ROS2 Foxy Fiztroy en la Raspberry Pi del vehículo ha sido más complicado. Esto es debido a la poca compatibilidad que existe entre la librería de Matlab (*Matlab Support for Raspberry Pi Hardware)* y ROS2. Para poder establecer una comunicación con la Raspberry y acceder a sus pines GPIO es necesario escribir en la SD la imagen customizada de *Mathworks.* Esta imagen escribe Raspberry Pi OS Buster 32 bits. A continuación, se procedió a construir ROS2 desde el código fuente (*building ROS2 from source)*. Para ello, previamente es necesario establecer una conexión remota entre la Raspberry Pi 3B+ y una máquina que corra Linux OS. En este proyecto, se hizo uso de *Windows Subsystem for Linux (WSL2)* como máquina virtual de Linux. Desde la terminal de WSL2, se debe clonar el siguiente repositorio de GitHub:

```
$ git clone https://github.com/poschl/ableros.git
```

A continuación se deben seguir los siguientes pasos:

1. Editar python-rosdep a python3-rosdep en */ableros/roles/install_ros2/tasks/install_packages.yml*
2. Editar eloquent a foxy en /ableros/roles/install_ros2/defaults/main.yml
3. Editar ssh_IP en /ableros/inventory.yml
4. Añadir una operación antes de la operación de *rosdep-update* en /ableros/roles/install_ros2/tasks/main.yml

```
- name: must
become: true
shell: 'export SSL_CERT_FILE=/usr/lib/ssl/certs/ca-certificates.crt'
```

A continuación, desde la terminal de WSL2 se debe ejecutar el siguiente comando:

```
$ ansible-playbook -i inventory.yml main.yml -k
```

Antes de la tarea "*Build ros*" cerrar el repositorio de WSL2 y conectar por SSH con Raspberry y dirigirse a *home/pi/ros2* para ejecutar el siguiente comando:

```
$ colcon build --symlink-install --cmake-args "-DCMAKE_SHARED_LINKER_FLAGS='-latomic'"
"-DCMAKE_EXE_LINKER_FLAGS='-latomic'" --packages-ignore qt_gui_cpp rqt_gui_cpp rqt
```

Una vez se hayan terminado de construir e instalar todos los paquetes del repositorio se debe ejecutar:

```
$ install/setup.bash
```

A continuación, ya se puede hacer uso de ROS2 Foxy.

ANEXO D: TRANSMISIÓN DEL VÍDEO CAPTURADO

El protocolo SSH no dispone de interfaz gráfica para poder visualizar el contenido capturado por el sensor óptico. Durante el desarrollo del proyecto, surgió la necesidad de monitorizar el contenido capturado por la cámara. Es por ello, que se ha decidido transmitir el vídeo de la cámara a un navegador web haciendo uso de Flask y Python.

Flask es un *framework* escrito en Python que permite desarrollar aplicaciones web de una forma ágil y rápida. Además, no necesita una infraestructura con un servidor web sino de una manera sencilla se puede correr un servidor web para visualizar el contenido. A su vez, Flask permite construir servicios web (como API Rest). Esto último será de gran utilidad ya que permite hacer peticiones HTTP.

En este proyecto se ha construido un servidor web capaz de transmitir el vídeo del sensor óptico, además de elaborar una tabla capaz de almacenar dinámicamente la posición y orientación de la cámara a partir de peticiones HTTP de tipo *GET*. Para ello, se han requerido conocimientos acerca de dos lenguajes de programación: HTML (para el diseño de la página web) y Python (para construir el servidor web).

A continuación, en la *Figura D.1* y en la *Figura D.2* se muestra una captura del vídeo donde se observa como el sensor óptico detecta varios marcadores del entorno de trabajo. Por otro lado, en la *Figura D.3* se muestra una captura de la página web desarrollada.

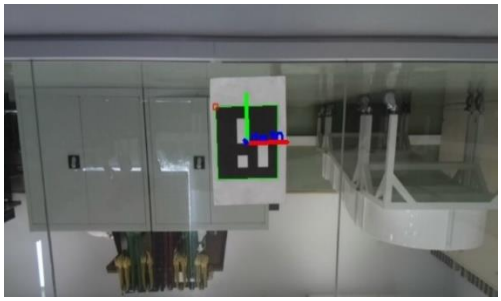
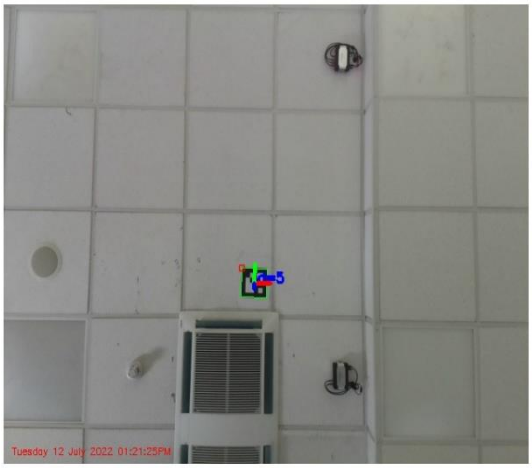


Figura D.1: ArUco con ID 30 detectado.

Figura D.2: ArUcos con IDs 1 y 30 detectados.

Raspberry Pi Camera



Get Camera Position

Posición de la cámara respecto a los marcadores

Traslación cámara (Metros)			Rotación cámara (Grados)		
X	Y	Z	Roll X	Patch Y	Yaw Z
0.016553013419490902	0.0505176587096067	2.12063511213486 54	0.2955846352286 3987	-0.025570969167 88955	0.065235585605 30837
0.08592007936834355	0.2195830704202692	2.149954700870024	-0.07763392819841297	-0.089755225330 55763	-0.730028617489 2614

Figura D.3: Página web para visualizar el contenido de la cámara.

ANEXO E: CONTRIBUCIÓN A LOS OBJETIVOS DE LA ONU PARA EL DESARROLLO SOSTENIBLE

En esta tesis también se explica cómo se relaciona este proyecto con los Objetivos de Desarrollo Objetivos de Desarrollo Sostenible (ODS) propuestos por la ONU para que la comunidad mundial trabaje para luchar contra la pobreza, mejorar el nivel de vida y proteger el medio ambiente.

Hay 17 objetivos en esta lista, pero este estudio puede ayudar especialmente en 2 de ellos, estos son:

- El objetivo número 9 aboga por una industria universal, sostenible e inclusiva que, junto con la innovación y la infraestructura, puedan dar rienda suelta a las fuerzas económicas dinámicas y competitivas que generan el empleo y los ingresos. Estas desempeñan un papel clave a la hora de introducir y promover nuevas tecnologías, facilitar el comercio internacional y permitir el uso eficiente de los recursos.

  En este proyecto se ha desarrollado un algoritmo de posicionamiento que se encuentra ligado a la innovación y creación de nuevas infraestructuras dentro de la industria. Algoritmos de posicionamiento como este permiten automatizar labores industriales como la recogida, transporte o gestión de mercancías en naves de logística.
- El Objetivo de Desarrollo Sostenible 11 trata de conseguir que las ciudades y asentamientos humanos sean lugares seguros, inclusivos, resilientes y sostenibles.

  En el contexto de este proyecto, la seguridad de las ciudades se encuentra en aumento debido a aplicaciones de visión artificial. En el ámbito de la seguridad vial, el uso de cámaras ópticas que sean capaces de detectar carreteras en mal estado, vehículos estacionados, o situaciones de tráfico potencialmente peligrosas, pueden suponer un gran avance en la seguridad de las ciudades. Por otro lado, el uso de vehículos autónomos se encuentra ligado a los motores eléctricos, lo que supone un compromiso con la sostenibilidad de las ciudades.