Trabajo Fin de Máster

Máster Universitario en Ingeniería Industrial

Guiado de robot móvil mediante sistema multicámara

Autor: Sergio Rosales Rey

Tutor: Manuel Vargas Villanueva

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025

Trabajo Fin de Máster
Máster Universitario en Ingeniería Industrial

Guiado de robot móvil mediante sistema multicámara

Autor:
Sergio Rosales Rey

Tutor:
Manuel Vargas Villanueva
Profesor Titular

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2025

Trabajo Fin de Máster: Guiado de robot móvil mediante sistema multicámara

Autor: Sergio Rosales Rey

Tutor: Manuel Vargas Villanueva

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2025

El Secretario del Tribunal

A mi familia

A mis maestros

A mis compañeros

Agradecimientos

Este trabajo debe agradecer al profesor Manuel Vargas por ser quien ideó la base a partir de la cual se ha podido desarrollar este proyecto además de sus consejos. También merecen mención todos los integrantes del laboratorio de Ingeniería de Sistemas y Automática ubicado en la planta baja L1 de la Escuela Técnica Superior de Ingeniería de Sevilla, por la facilidad y comprensión que han aportado.

Por otra parte, me gustaría agradecer a mi familia por la paciencia y la ayuda para que el proyecto pudiera ser finalizado lo antes posible.

Sergio Rosales Rey

Sevilla, 2025

Resumen

Los algoritmos de visión artificial son una herramienta cada vez más extendida en el día a día, desde facilitar diversas tareas cotidianas hasta la inspección de calidad o la navegación de vehículos autónomos. Este proyecto se centra en estudiar cómo, a partir de un sistema multicámara compuesto por cuatro cámaras orientadas hacia una plataforma de trabajo, es posible estimar la posición de un robot móvil y controlar su movimiento a partir de las fotografías tomadas por las cámaras y del uso de marcadores ArUco, los cuales facilitan la detección y el seguimiento en las imágenes. Estos marcadores están distribuidos sobre la plataforma de trabajo, incluido el que posee el propio robot en la parte superior.

Se desarrollará un sistema que, a partir de las transformaciones proyectivas de cada cámara (homografías), pueda realizar una correspondencia entre la imagen y el plano de la plataforma de trabajo, permitiendo estimar cualquier punto en la superficie o la posición del robot en coordenadas 2D. Con esta información, el objetivo es que el robot, además de poder ser controlado de forma manual, también pueda ir a cualquier punto deseado en la mesa de manera totalmente autónoma y en las condiciones impuestas por el usuario, además de tener un registro de la trayectoria seguida por el robot. Para que el robot pueda ir a cualquier punto de forma óptima, se tendrá que ajustar y evaluar la mejor configuración tanto del robot como de las cámaras.

Abstract

Computer vision algorithms are an increasingly widespread tool used in everyday life, from facilitating various daily tasks to quality inspection and autonomous vehicle navigation. This project focuses on exploring how, using a multi-camera system made up of four cameras oriented toward a working area, it is possible to estimate the position of a mobile robot and control its movement based on the images captured by the cameras and the use of ArUco markers, which simplify the detection and tracking within images. These markers are distributed across the working area, including one located on the top of the robot.

The system to be developed will use projective transformations (homographies) from each camera to establish a correspondence between the images and the plane of the working area, allowing the estimation of any point on the table or the robot's position in 2D coordinates. With this information, the robot will not only be able to be manually controlled but also be capable of navigating autonomously to any point on the table according to user-defined conditions, while also keeping a record of its trajectory. To ensure optimal movement, the robot and camera configurations will be adjusted and evaluated.

ÍNDICE

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice de Tablas **xix**

Índice de Figuras **xxi**

Notación **xxvii**

1 INTRODUCCIÓN **1**
1.1. Contexto y motivación *1*
1.2. Objetivos del proyecto *2*
1.3. Trabajos anteriores *2*
1.4. Estructura del documento *2*

2 ESTADO DEL ARTE **4**
2.1. Guiado, localización y percepción *4*
2.2. Robótica *7*
2.2.1 Origen de la robótica 7
2.2.2 Aplicaciones de la robótica 9

3 MARCO TEÓRICO **12**
3.1. Modelo de una cámara *12*
3.1.1 Modelo Pinhole 12
3.1.2 Parámetros intrínsecos 13
3.1.3 Parámetros extrínsecos 15
3.1.4 Distorsión de la lente 15
3.1.5 Método Zhang 16
3.2. Homografía *16*
3.2.1 Homografía en geometría 16
3.2.2 Homografía en imágenes 18
3.3. Control de robots con ROS *19*

4 RECURSOS EMPLEADOS **21**
4.1. Hardware *21*
4.2. Software *24*
4.3. Otros recursos *25*
4.4. Pruebas iniciales *25*

5 MONTAJE DE LA ESTACIÓN DE TRABAJO **28**
5.1. Distribución de cámaras *28*
5.2. Conexionado de cámaras *28*
5.3. Configuración y sincronización de cámaras *29*
5.4. Conexión a zona Wi-Fi del ordenador *31*
5.5. Ejecución de WSL *32*
5.6. ROS y puesta a punto del robot *33*

6 MÉTODOS DE CALIBRACIÓN PARA LAS CÁMARAS **36**
6.1. Calibración estéreo *36*
6.2. Homografía *41*
6.3. Comparación de los métodos de calibración *45*

7 CAPTURA DE IMÁGENES **46**
7.1. Imágenes necesarias *46*
7.2. Sincronización de cámaras *46*

7.3. Modos de captura de imágenes *48*
7.3.1 Modo 1: Imágenes para la homografía de la mesa 48
7.3.2 Modo 2: Imágenes para la homografía del plano robot 48
7.3.3 Modo 3: Detección de nuevos marcadores 48
7.3.4 Modo 4: Captura en tiempo real 49
7.4. Programa de captura de imágenes *49*
7.4.1 Inicio del programa 49
7.4.2 Selección de modo 50
7.4.3 Finalización del programa 51

8 TRATAMIENTO DE IMÁGENES Y CONTROL DEL ROBOT 52
8.1. Necesidades del programa. *52*
8.2. Carga de imágenes y detección de ArUco. *52*
8.3. Cálculo de homografías *54*
8.4. Inicialización de ROS *55*
8.5. Lectura de imágenes en tiempo real *55*
8.6. Control PID para el movimiento del robot *55*
8.7. Saturación y filtrado de velocidad *56*
8.8. Modos de funcionamiento *56*
8.8.1 Modo manual 57
8.8.2 Modo automático 57
8.9. Medidas de seguridad *58*
8.10. Gráficas *59*
8.11. Funcionamiento del programa *59*

9 INTERFAZ 62
9.1. Programa de captura de imágenes *62*
9.2. Programa de tratamiento de imágenes y control del robot *65*

10 EXPERIMENTACIÓN 74
10.1. Apertura de la lente *74*
10.2. Valores iniciales de los parámetros del menú de ajuste. *79*
10.3. Movimiento del robot en modo manual *82*
10.4. Sintonización del PID para el modo automático *87*
10.4.1 Obtención de la función de transferencia del movimiento lineal 87
10.4.2 Obtención de la función de transferencia del movimiento angular 94
10.4.3 Método 1: Control PI por cancelación de polos despreciando retrasos en bucle cerrado. 99
10.4.4 Método 2: Control por Ziegler-Nichols en bucle abierto. 103
10.4.5 Método 3: Control por Ziegler-Nichols con ajuste fino. 110
10.5. Guiado y movimiento del robot en modo automático modificando opciones. *115*
10.5.1 Modificación 1: Velocidades máximas 115
10.5.2 Modificación 2: Tolerancias 117
10.5.3 Modificación 3: Márgenes interno y externo. 120
10.5.4 Prueba con configuración final 121
10.6. Pérdida de imágenes *124*

11 PROYECTOS FUTUROS 126

12 CONCLUSIONES DEL PROYECTO 127

ANEXO 128
a. ANEXO 1: T_Nano-M1280-NIR_Default_GrandMaster_15ms_60Hz.ccf *128*
b. ANEXO 2: T_Nano-M1280-NIR_Default_GrandMaster_15ms_60Hz.ccf *132*
c. ANEXO 3: Actioncmd.cpp *136*
d. ANEXO 4: control_robot.py *165*

Referencias 203

Índice de Conceptos **207**

Glosario **209**

ÍNDICE DE TABLAS

Tabla 6.1 Obtención de parámetros intrínsecos de las cámaras. 38
Tabla 6.2 Pares de imágenes detectados. 40
Tabla 10.1 Tabla resumen de las funciones de transferencia lineales. 94
Tabla 10.2 Tabla resumen de las funciones de transferencia angulares. 99
Tabla 10.3 Tabla de valores para control por Z-N en bucle abierto. 103
Tabla 10.4 Valores de los controladores por Z-N bucle abierto. 106
Tabla 10.5 Resultados de la pérdida de imágenes en los experimentos. 125

ÍNDICE DE FIGURAS

Figura 2.1 Sensor de ultrasonidos. [35] 4
Figura 2.2 Esquema del funcionamiento del GPS. [36] 5
Figura 2.3 Ejemplo de sistema IPS. [37] 5
Figura 2.4 Marcadores AprilTags. [38] 6
Figura 2.5 Patrón Ajedrez. [39] 7
Figura 2.6 Automa Cavaliere. [40] 8
Figura 2.7 Portada del libro Yo Robot. [41] 8
Figura 2.8 Robot militar. [42] 9
Figura 2.9 Robot antropomórfico. [43] 10
Figura 2.10 Robot androide. [44] 10
Figura 2.11 Ejemplo de Cobot. [45] 11
Figura 3.1 Esquema de una cámara oscura. [46] 12
Figura 3.2 Esquema de una homografía. [49] 17
Figura 3.3 Homografía de la carátula de un libro. [50] 19
Figura 3.4 Esquema de funcionamiento de ROS. [52] 20
Figura 4.1 Cámara Genie Nano M1280-NIR. 21
Figura 4.2 Foco Komoto LV7-T. 21
Figura 4.3 Fuente de alimentación MEAN WELL SDR-75-12. 22
Figura 4.4 Ordenador del proyecto. 22
Figura 4.5 Monitor para el proyecto. 23
Figura 4.6 Switch Netgear MS510TXPP. 23
Figura 4.7 Adaptador Wi-Fi TP-LINK ARCHER T3U. [53] 23
Figura 4.8 Turtlebot 3 Burger. 24
Figura 4.9 Smartphone Samsung S21 FE. [54] 24
Figura 4.10 Estación de trabajo. 26
Figura 5.1 Distribución de las cámaras. 28
Figura 5.2 Fotografía de conexionado. 29
Figura 5.3 Información de la cámara en Network Configuration Tool. 29
Figura 5.4 Configuración de dirección IP en Network Configuration Tool. 30
Figura 5.5 Creación de la zona Wi-Fi. 31
Figura 5.6 Dispositivo conectado a zona Wi-Fi. 32
Figura 5.7 Datos de la conexión Wi-Fi. 32
Figura 5.8 Inicio de WSL con Ubuntu 20.04.6. 32
Figura 5.9 Introducción de direcciones IP. 33
Figura 5.10 Visualización de roscore. 33
Figura 5.11 Configuración Wi-Fi para la Raspberry. [74] 34

Figura 5.12 Bringup del Turtlebot 3. [74] 35
Figura 6.1 Sexta foto desde la cámara 1. 37
Figura 6.2 Sexta foto desde la cámara 2. 37
Figura 6.3 Sexta foto desde la cámara 3. 37
Figura 6.4 Sexta foto desde la cámara 4. 37
Figura 6.5 Sexta foto desde la cámara 1 segundo tablero. 41
Figura 6.6 Sexta foto desde la cámara 2 segundo tablero. 41
Figura 6.7 Sexta foto desde la cámara 3 segundo tablero. 41
Figura 6.8 Sexta foto desde la cámara 4 segundo tablero. 41
Figura 6.9 Fotografía de la mesa. 42
Figura 6.10 Origen de coordenadas y marcadores de la mesa. 43
Figura 6.11 Distribución 2D de la mesa. 43
Figura 8.1 Marcadores y ejes vistos desde la cámara 1. 53
Figura 8.2 Vista de la mesa tras realizar la homografía desde la cámara 1. 53
Figura 8.3 Dimensiones de la mesa. 54
Figura 8.4 Diagrama del PID lineal. 55
Figura 8.5 Diagrama PID angular. 56
Figura 9.1 Selección de red en la interfaz. 62
Figura 9.2 Selección de modo en la interfaz. 62
Figura 9.3 Inicio del modo 1 en la interfaz. 63
Figura 9.4 Fin del modo 1 en la interfaz. 63
Figura 9.5 Modo 2 captura sobre ID1. 63
Figura 9.6 Fin del modo 2 en la interfaz. 64
Figura 9.7 Modo 3 en la interfaz. 64
Figura 9.8 Inicio Modo 4 en la interfaz. 64
Figura 9.9 Previsualización desde la cámara 1. 65
Figura 9.10 Inicio de interfaz del programa de Python. 65
Figura 9.11 Menú de configuración. 65
Figura 9.12 Configuración PID. 66
Figura 9.13 Edición de los parámetros PID. 66
Figura 9.14 Edición de la configuración 2. 67
Figura 9.15 Conexión exitosa con ROS. 67
Figura 9.16 Conexión fallida con ROS Master. 67
Figura 9.17 Tópico no detectado a tiempo. 67
Figura 9.18 Comprobación de funcionamiento del programa de captura de imágenes. 67
Figura 9.19 No hay actualización de imágenes. 67
Figura 9.20 Posiciones calculadas de los marcadores. 68
Figura 9.21 Selección de modo de operación. 68
Figura 9.22 Interfaz del modo manual. 68

Figura 9.23 Interfaz del modo manual. 69
Figura 9.24 Resultados del modo manual. 69
Figura 9.25 Selección de marcador o punto. 69
Figura 9.26 Ejemplo de distribución del plano de la mesa en interfaz. 70
Figura 9.27 Ir a marcador. 70
Figura 9.28 Modo de selección del punto objetivo. 70
Figura 9.29 Introducción de punto objetivo por teclado. 70
Figura 9.30 Introducción de punto objetivo haciendo clic. 70
Figura 9.31 Ventana de selección por clic. 71
Figura 9.32 Resultados del movimiento automático tras llegar al objetivo. 71
Figura 9.33 Ejemplo de gráfica de errores. 72
Figura 9.34 Ejemplo de trayectoria sobre dibujo 2D. 72
Figura 9.35 Ejemplo de trayectoria desde la cámara 1. 72
Figura 9.36 Ejemplo de registro de velocidades. 72
Figura 9.37 Ejemplo de registro de velocidades. 73
Figura 10.1 Fotografía desde la cámara 1 con la apertura al máximo. 74
Figura 10.2 Fotografía desde la cámara 1 con la apertura al mínimo. 74
Figura 10.3 Fotografía desde la cámara 1 con la apertura a nivel intermedio. 75
Figura 10.4 Detección de marcadores con máxima apertura. 75
Figura 10.5 Detección de marcadores con mínima apertura. 76
Figura 10.6 Detección de marcadores con apertura intermedia. 76
Figura 10.7 Detección desde cámara 1 con nuevo marcador 50. 77
Figura 10.8 Detección de marcadores desde cámara 2. 78
Figura 10.9 Detección de marcadores desde cámara 3. 78
Figura 10.10 Detección de marcadores desde cámara 4. 79
Figura 10.11 Detección de marcador 120 desde cámara 3. 79
Figura 10.12 Prueba de velocidad lineal máxima. 80
Figura 10.13 Prueba de velocidad angular máxima. 80
Figura 10.14 Fotografía real de los nuevos marcadores. 81
Figura 10.15 Estimación de los nuevos marcadores sobre el plano 2D. 82
Figura 10.16 Experimento 1 del modo manual. 83
Figura 10.17 Resultados del experimento 1 del modo manual. 83
Figura 10.18 Trayectoria del experimento 1 del modo manual. 84
Figura 10.19 Comparación de marcadores ID 120. Marcador antiguo (izquierda) y marcador nuevo (derecha). 84
Figura 10.20 Experimento 2 del modo manual. 85
Figura 10.21 Trayectoria del experimento 2 del modo manual. 86
Figura 10.22 Registro de velocidades del experimento 2 del modo manual. 87
Figura 10.23 Escalón de 0.03 m/s de velocidad lineal. 88

Figura 10.24 Escalón de 0.06 m/s de velocidad lineal. 89
Figura 10.25 Escalón de 0.09 m/s de velocidad lineal. 90
Figura 10.26 Escalón de 0.12 m/s de velocidad lineal. 91
Figura 10.27 Escalón de 0.15 m/s de velocidad lineal. 92
Figura 10.28 Escalón de 0.18 m/s de velocidad lineal. 93
Figura 10.29 Escalón de 0.5 rad/s de velocidad angular. 94
Figura 10.30 Escalón de 1.0 rad/s de velocidad angular. 95
Figura 10.31 Escalón de 1.5 rad/s de velocidad angular. 96
Figura 10.32 Escalón de 2.0 rad/s de velocidad angular. 97
Figura 10.33 Escalón de 2.3 rad/s de velocidad angular. 98
Figura 10.34 Valores del PI por cancelación de polos introducidos. 102
Figura 10.35 Trayectoria seguida por el robot con PI por cancelación de polos. 102
Figura 10.36 Error con PI por cancelación de polos. 103
Figura 10.37 Valores para el control P por Z-N. 106
Figura 10.38 Trayectorias para el control P Z-N. 106
Figura 10.39 Registro de velocidades para el control P Z-N. 107
Figura 10.40 Errores entre marcador 6 y 9 para el control P Z-N. 107
Figura 10.41 Valores PI Z-N. 108
Figura 10.42 Trayectoria seguida PI Z-N. 108
Figura 10.43 Valores implementados PID Z-N. 109
Figura 10.44 Trayectoria PID Z-N. 109
Figura 10.45 Errores PID Z-N. 110
Figura 10.46 Valores para experimento 1 PID A-F. 110
Figura 10.47 Trayectoria seguida PID A-F 1. 111
Figura 10.48 Registro de velocidades PID A-F 1. 111
Figura 10.49 Errores PID A-F 1. 112
Figura 10.50 Valores para experimento 2 PID A-F. 112
Figura 10.51 Trayectorias seguidas PID A-F 2. 113
Figura 10.52 Valores para experimento 3 PID A-F. 113
Figura 10.53 Trayectorias seguidas PID A-F 3. 114
Figura 10.54 Registro de velocidades PID A-F 3. 114
Figura 10.55 Errores entre punto (1.50, 0.76) y marcador 9 para PID A-F 3. 115
Figura 10.56 Trayectorias de la modificación 1. 116
Figura 10.57 Registro de velocidades de la modificación 1. 116
Figura 10.58 Errores entre marcador 6 y 9 en la modificación 1. 117
Figura 10.59 Trayectoria de la modificación 2, tolerancia 0.01. 118
Figura 10.60 Errores en la modificación 2, tolerancia 0.01. 118
Figura 10.61 Trayectoria de la modificación 2, tolerancia 0.07. 119
Figura 10.62 Errores en la modificación 2, tolerancia 0.07. 119

Figura 10.63 Trayectoria entre ID 1 y 2 sin diferencia de márgenes interno y externo en la modificación 3. 120
Figura 10.64 Trayectoria entre ID 1 y 2 con margen interno de 0.25 m en la modificación 3. 121
Figura 10.65 Errores entre ID 1 y 2 con margen interno de 0.25 m en la modificación 3. 121
Figura 10.66 Configuración final PID. 122
Figura 10.67 Configuración final de los parámetros. 122
Figura 10.68 Trayectorias recorridas en la prueba final. 123
Figura 10.69 Registro de velocidades en la prueba final. 123
Figura 10.70 Errores 1º destino, prueba final. 124
Figura 10.71 Errores 2º destino, prueba final. 124
Figura 10.72 Errores 3º destino, prueba final. 124
Figura 10.73 Errores 4º destino, prueba final. 124

Notación

Kc o Kp	Constante proporcional
Kd	Constante derivativa
Ki	Constante integral
f	Distancia focal de la cámara
H	Matriz de homografía
R	Matriz de rotación
T	Vector de traslación
V	Voltaje
W	Vatio
Hz	Hercio
K	Ganancia
τ	Constante de tiempo
τ_d	Retardo

1 INTRODUCCIÓN

La estimación de posición mediante cámaras ha avanzado bastante en los últimos años, permitiendo conocer la posición y orientación de un objeto con una simple imagen y una referencia tanto en 3D como en 2D.

Gracias a estos avances podemos controlar en tiempo real el movimiento de vehículos, AGVs o robots.

1.1. Contexto y motivación

Actualmente el mundo se encuentra en constante cambio tecnológico debido al gran desarrollo que están teniendo ramas como la robótica, la inteligencia artificial o los sistemas de percepción.

Esto se puede observar en algunos trabajos profesionales como:

- En logística, el control de vehículos autónomos para hacer más rápido el transporte de material.
- En líneas de ensamblaje el uso de robots es ya una práctica habitual.
- Los automóviles tanto particulares como profesionales, incorporan sensores para el reconocimiento del entorno, como por ejemplo para la detección de peatones, otros vehículos o señales de tráfico.
- En cirugía, para poder llegar a partes del cuerpo que serían imposibles sin el uso de robots.
- Algoritmos de visión artificial e IA, para el reconocimiento de piezas defectuosas o sistemas de seguridad.

Además, con el paso del tiempo irán apareciendo más profesiones en las que aplicar estas tecnologías. Cada vez se necesita menos intervención humana y existe una mayor automatización.

En este proyecto nos centraremos en mezclar dos de las disciplinas anteriormente mencionadas, los sistemas de percepción y la robótica.

Como se ha dicho, la robótica es una disciplina que ha progresado bastante en los últimos años, pero alcanzar el control total que haga posible el ser capaz de reemplazar al ser humano completamente es un desafío muy complejo. Principalmente en lo que se conoce como robótica móvil, en la cual los robots se pueden desplazar en distintos entornos según sus capacidades. [1]

Para poder superar este reto, se necesita una percepción completa del entorno, esta se podría lograr con sistemas de visión los cuales a partir de las cámaras permitieran el conocimiento del entorno del robot. Puesto que el mundo es tridimensional sería interesante un desarrollo multicámara dado que permitiría una mayor precisión. Para el control de robots es clave la precisión dado que un error puede resultar fatal al no existir intervención humana, también habría que minimizar errores porque si no estos se podrían acumular.

Para desarrollar un guiado de robots móviles con sistemas multicámara, se podrían integrar algoritmos de visión artificial como la estimación de posición o la homografía a partir de las imágenes capturadas.

La motivación principal de este proyecto consiste en explorar e integrar métodos que permitan la navegación de robots móviles de manera óptima y eficiente. Por otra parte, esta deberá ser precisa y segura para no poner en riesgo ni a las personas ni al entorno ni al robot.

1.2. Objetivos del proyecto

El objetivo de este proyecto será la implementación del control y guiado de un robot móvil mediante un sistema de cuatro cámaras industriales el cual se encuentra en el laboratorio del Departamento de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla. Puesto que ya se hicieron experimentos con las cámaras en proyectos anteriores, en este trabajo se intentará añadir la capacidad de que con la información de las cámaras se pudiese mover un robot a cualquier punto de una mesa a la que apuntan las cuatro cámaras y obtener la trayectoria seguida sabiendo la posición del robot en todo momento sobre la mesa respecto a un origen de coordenadas bidimensional, con esta información se buscará la optimización del movimiento seguido por el robot.

Por otro lado, también se buscará la seguridad del robot, de manera que en ningún caso pueda caerse de la mesa o alcanzar velocidades que pudieran comprometer su correcto funcionamiento.

Para el correcto funcionamiento del sistema habrá que buscar tanto una buena obtención de las imágenes por parte de las cámaras como que la información obtenida de estas fotografías sea suficiente para el control del robot.

1.3. Trabajos anteriores

Es importante mencionar los trabajos que se realizaron anteriormente en los que se ha basado este proyecto.

En 2020 el trabajo "*Seguimiento de objetos móviles con sistema de posicionamiento multicámara*" de *Elisa Hidalgo Moreda* creó las bases para los proyectos posteriores. En este, se construyó un simulador utilizando Blender 2.82. Con los resultados del simulador se propuso una solución en C++. La finalidad del simulador era poder probar futuros algoritmos de estimación. [25]

Posteriormente, en el mismo año, el proyecto *"Experiencias en calibración de sistemas multicámara para seguimiento tridimensional"* de *Marcos Pérez Rus* inició el estudio de la sincronización y calibración de las cámaras del laboratorio. El objetivo era averiguar parámetros como la posición y la rotación relativa entre las cámaras para que en un futuro se pudiera realizar una localización tridimensional en interiores. [26]

En el año 2022, en el trabajo *"Puesta en marcha de sistemas multicámara para seguimiento de objetos en movimiento"* de *Francisco Javier Fernández Gómez* se tomaban fotos desde cuatro cámaras de la forma más sincronizada posible para estimar los parámetros intrínsecos y extrínsecos de estas cámaras. En el proyecto el cálculo de los parámetros se realiza con tres algoritmos distintos, patrón ajedrez, patrón aleatorio y patrón de punto. El objetivo era que estos procedimientos pudieran ser usados en el reconocimiento de objetos y su localización tridimensional. [27]

Por último, en 2023 *"Sistema multicámara para seguimiento a tiempo real de objetivos en movimiento. Adaptación a nuevo hardware y estimación tridimensional" de Alfonso Martínez Cuadros*. En este proyecto se capturaban imágenes simultáneamente y se almacenaban en el ordenador. Se calculaban los parámetros de las cámaras para realizar una reconstrucción 3D de la escena. Con la detección y seguimiento de marcadores ArUco se podía saber la posición y orientación del robot en tiempo real. Aunque en este trabajo el control del movimiento del robot no formaba parte del estudio. [34]

Cabe destacar que el presente trabajo utiliza una versión modificada del programa para la captura de imágenes de *"Sistema multicámara para seguimiento a tiempo real de objetivos en movimiento. Adaptación a nuevo hardware y estimación tridimensional"*, además del uso de ArUco para conocer la posición del robot. Por otra parte, el hardware utilizado en estos trabajos es muy parecido, con la novedad del robot móvil *Turtlebot 3*.

1.4. Estructura del documento

En este documento además de la breve introducción al proyecto ya realizada correspondiente al capítulo 1, en el capítulo 2 se explicarán el estado actual de las tecnologías relacionadas con este proyecto y un poco de su historia.

En el capítulo 3 se explicará la teoría necesaria para el entendimiento del trabajo.

El capítulo 4 hablará de los recursos tanto de hardware como de software necesarios para la realización del proyecto. También se detallarán las pruebas previas que se hicieron para conocer las herramientas que serían útiles.

Con el capítulo 5 se explica de forma detallada los pasos a seguir para la correcta puesta a punto de la estación de trabajo para su posterior experimentación.

En el capítulo 6 se expondrán los diferentes métodos de calibración de cámaras con sus ventajas e inconvenientes, además de la elección del método a utilizar en el desarrollo del proyecto.

El capítulo 7 explicará el programa desarrollado para la captura de imágenes por parte de las cámaras.

En el capítulo 8 se expondrá el programa el cual realiza el tratamiento de imágenes para que el robot se pueda mover de la forma deseada y su posterior desplazamiento, además de poder obtener los resultados.

El capítulo 9 contiene la explicación del funcionamiento de la interfaz de los programas realizados.

En el capítulo 10 se exponen los diferentes experimentos realizados y el análisis de los resultados dependiendo de las condiciones que se establezcan.

Para ir finalizando se propondrán diferentes ideas para proyectos futuros en el capítulo 11 y se terminará con las conclusiones que se han podido extraer de la realización de este trabajo en el capítulo 12.

Al final del documento, se muestran los anexos, las diferentes referencias utilizadas, el índice de conceptos y el glosario.

2 ESTADO DEL ARTE

2.1. Guiado, localización y percepción

En los últimos años, el guiado y la localización de robots se ha enfocado tanto en la mejora de la percepción de los sistemas empleados como en su completa autonomía. A continuación, se expondrá cada uno de los métodos más utilizados actualmente.

1. **Sensores de proximidad.**

Para la estimación de distancias se pueden usar:

- Sensores infrarrojos: se basan en un emisor y un receptor de infrarrojos con los que se calcula el tiempo que tarda la luz en llegar y su velocidad. Con estos parámetros se puede obtener la distancia. [2]
- Sensores de ultrasonidos: parten de la reflexión de las ondas de ultrasonidos, de manera que el sensor envía una onda hacia el objeto de medición. Cuando el pulso sónico incide sobre un objeto se refleja y regresa de nuevo al sensor de ultrasonidos. Con el tiempo entre el envío y la recepción de la onda se calcula la proximidad del objeto u obstáculo. [3]

Figura 2.1 Sensor de ultrasonidos. [35]

- LIDAR (Light Detection and Ranging): permite estimar la posición y la distancia de objetos usando un láser. Su funcionamiento consiste en la medición del intervalo de tiempo entre el envío de un haz de luz y su llegada a los objetos que detecta. En los últimos años se ha incrementado su uso en robótica. [4]

2. **Sistemas de visión por computadora.**

Los más usados son las cámaras monoculares y los sistemas estereoscópicos.

Las cámaras monoculares permiten hacer grandes aumentos de zoom. Esto hace que se puedan utilizar para reconocimiento de patrones o estimación de movimiento.

Los sistemas estereoscópicos dan la opción de estimar la profundidad de una imagen. Parten de la idea a partir de la cual los seres humanos pueden ver con sus ojos la profundidad, con cada uno de los dos ojos ven una imagen ligeramente diferente. Esto se puede reproducir artificialmente, mediante dos cámaras las cuales enfoquen a una misma zona u objeto, lo que permite hacer una reconstrucción tridimensional del entorno al igual que hace el cerebro. En el procesamiento artificial con las cámaras cuantas más imágenes y diferentes perspectivas se tienen mayor será la precisión. Las desventajas de estos sistemas es que están limitados al área común que barren ambas cámaras, además la luz o simples destellos pueden provocar grandes imprecisiones. [6]

3. GPS.

Permite a un dispositivo receptor localizar su posición sobre La Tierra con una precisión de incluso centímetros.

El GPS fue desarrollado por el Departamento de Defensa de Estados Unidos, pero en la actualidad pertenece a la Fuerza Espacial de los Estados Unidos y tiene 31 satélites operativos.

Para determinar la posición tridimensional del objetivo, el receptor debe localizar mínimo cuatro satélites que estén en la red. De estos satélites recibe señales que indican su identificación, hora del reloj e información sobre los satélites que forman parte del sistema. Con estas señales mencionadas, el objetivo sincroniza su reloj con el tiempo del GPS y calcula cuánto tiempo tardan en llegar las señales al equipo, con esto es capaz de saber la distancia al satélite. Usando trilateración inversa consigue conocer posición propia dado que ya se conocen las posiciones de los satélites y la distancia. Cada satélite genera una esfera cuyo centro es él mismo y una pseudodistancia como radio, haciendo la intersección de tres de las esferas eligiendo siempre el punto de corte que se encuentra en La Tierra, se tiene la posición del dispositivo objetivo. El cuarto satélite se suele usar para corrección de errores. [22] [23]

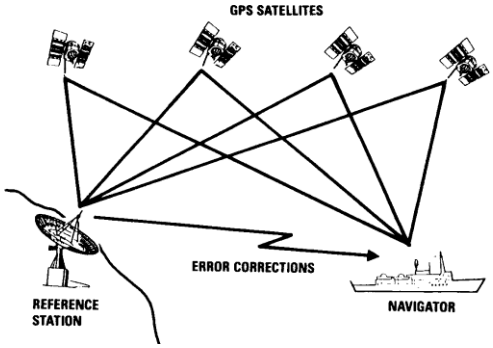

Figura 2.2 Esquema del funcionamiento del GPS. [36]

4. IPS.

Es parecido al GPS, pero se usa para la localización inalámbrica dentro de un edificio. En vez de utilizar los satélites usa nodos de los cuales se conoce su posición. Necesitamos un mínimo de 4 nodos para poder obtener la posición tridimensional sin errores grandes.

Los sistemas suelen estar basados en Wi-Fi, Bluetooth y Ultra Banda Ancha (UWB). [24]

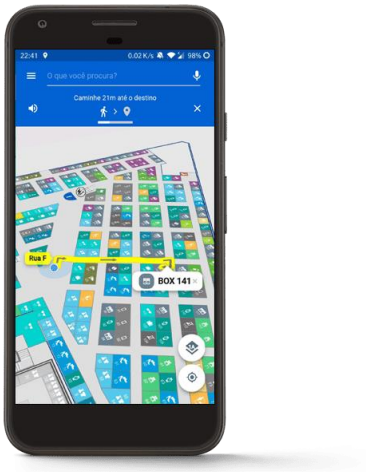

Figura 2.3 Ejemplo de sistema IPS. [37]

5. Sistemas multicámara.

Los sistemas multicámara constan de varias cámaras como su propio nombre indica, están dispuestas en

diferentes lugares transmitiendo simultáneamente una escena. Permiten un rango de visión más amplio y pueden detectar y triangular objetos mejorando su percepción.

A estos sistemas se les pueden incorporar algoritmos matemáticos como la homografía para una estimación en dos dimensiones o la estereoscopia para una vista tridimensional. [7]

6. Homografía.

La homografía describe una transformación proyectiva que relaciona los puntos, líneas o figuras que se encuentran en un mismo plano con sus correspondientes proyecciones en otro plano, siempre que ambos planos representen la misma escena observada desde perspectivas diferentes.[9]

Las homografías se utilizan actualmente en algoritmos de visión para tareas como la corrección de trayectorias, el seguimiento de objetos, sistemas multicámara, imágenes panorámicas o realidad aumentada, entre otras aplicaciones.

Para calcular una homografía se necesita un mínimo de cuatro puntos no colineales de un plano y sus puntos correspondientes en la otra imagen. Estos puntos se podrían obtener mediante el uso de marcadores de visión, lo que permitiría identificar de forma automática sus coordenadas y facilitar así el cálculo de la homografía.

Se profundizará más sobre la homografía en el capítulo 3.2.

7. Marcadores de visión.

Los marcadores son habitualmente usados para poder conocer puntos de referencia en el espacio. A continuación, se detallarán los más comunes.

- Marcadores ArUco: ArUco es una librería de código abierto para la estimación de posición usando marcadores cuadrados. Su desarrollo inicial fue en C++, pero actualmente se puede utilizar en otros lenguajes. Son especialmente utilizados en aplicaciones de visión por computadora y realidad aumentada. [11]
- AprilTags: Son parecidos a los ArUco anteriormente descritos, aunque con un diseño ligeramente diferente. Tienen gran robustez y velocidad de detección. [12]

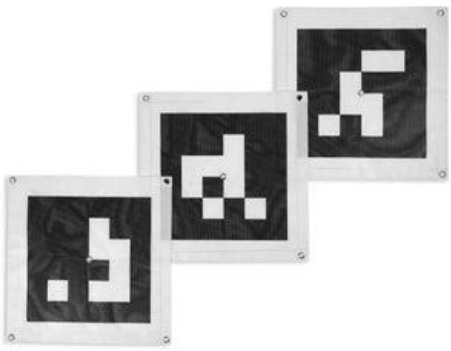

Figura 2.4 Marcadores AprilTags. [38]

- Códigos QR: Generalmente su aplicación principal es el almacenamiento de datos en un código de barras bidimensional, pero también se pueden usar para localización y posicionamiento. Lo positivo es que es una tecnología bastante desarrollada, pero son menos precisos dado que, su fin principal es el almacenamiento de datos como se ha dicho anteriormente.
- Marcadores fiduciarios: Consiste en colocar un objeto con unas características concretas en el campo de visión para ser usado como punto de referencia. A diferencia de los marcadores anteriormente expuestos estos marcadores no cuentan con diccionarios específicos ni patrones predefinidos.

- Tablero de ajedrez: Son usados principalmente para la calibración de cámaras al proporcionar a partir de los cuadrados que posee bastantes puntos de referencia bien definidos.

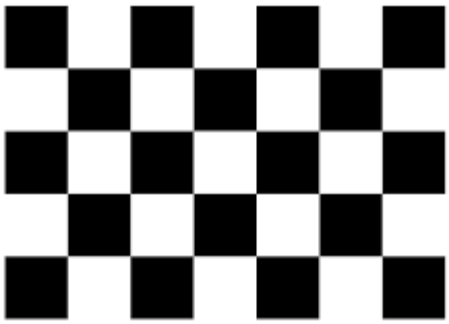

Figura 2.5 Patrón Ajedrez. [39]

- Códigos Data Matrix: Similares a los QR, pero de uso más industrial que estos dado que son capaces de tratar grandes cantidades de datos. [15]

2.2. Robótica

> *Técnica que aplica la informática al diseño y empleo de aparatos, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales.*
>
> Real Academia Española [16]

Etimológicamente la palabra robótica proviene de "robota", que en checo significa *trabajo forzado*.

Isaac Asimov definió la robótica como "*la ciencia que estudia los robots"*.

2.2.1 Origen de la robótica

En el Antiguo Egipto se desarrollaron modelos matemáticos y automatismos muy avanzados para la época. Siglos más tarde en la Antigua Grecia el matemático Aristóteles introdujo mecanismos como palancas y poleas capaces de actuar como brazos humanos.

Ya en la Edad Media fue cuando se empezaron a crear los primeros autómatas, su finalidad era la de imitar el movimiento de los seres vivos, principalmente el de los seres humanos. Uno de estos fue el "*Automa Cavaliere"*, diseñado por *Leonardo Da Vinci* a mediados del siglo XV, cuya finalidad era reproducir los movimientos de una persona.

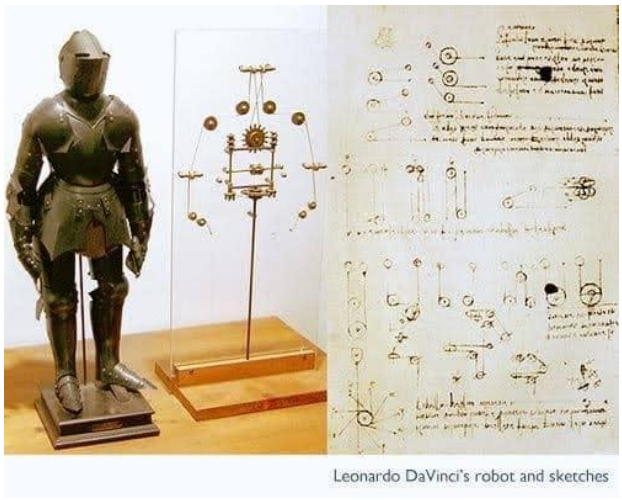

Figura 2.6 Automa Cavaliere. [40]

Cuando de verdad se empezaron a dar grandes avances en la robótica fue en el siglo XX. En 1937, *Bill Taylor*, un estudiante británico, diseñó el robot "*Gargantúa"*, capaz de recoger y colocar contenedores de mercancías.

En 1950, Isaac Asimov escribió *"Yo Robot",* este libro enumeró las *leyes de la robótica*. [18]

1. *Un robot no hará daño a un ser humano, ni por inacción permitirá que un ser humano sufra daño.*
2. *Un robot debe cumplir las órdenes dadas por los seres humanos, a excepción de aquellas que entren en conflicto con la primera ley.*
3. *Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la primera o con la segunda ley.*

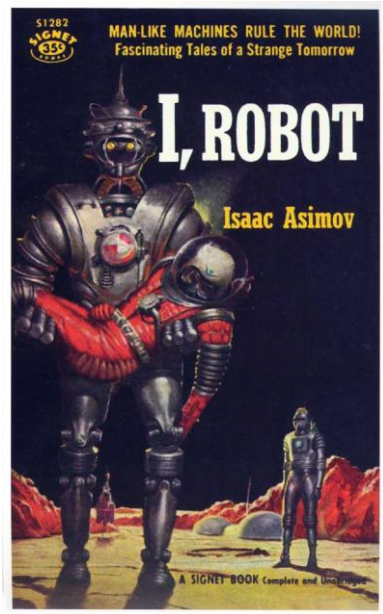

Figura 2.7 Portada del libro Yo Robot. [41]

Ya en la década de los 60, la compañía *Unimation* creó el primer robot de transferencia programable, una versión primitiva del robot industrial actual. Por otro lado, la empresa automovilística *General Motors* creó las primeras cadenas de montaje automatizadas para la fabricación de sus vehículos.

Antes de finalizar el siglo XX, ya empezaron a aparecer los robots en diversos entornos industriales.

Ya en el siglo XXI hay robots de toda índole, drones, vehículos, robots educativos, etc.

Con la aparición de la Inteligencia Artificial, el avance de la robótica podría ser enorme y permitiría sustituir con robots, procesos de la vida cotidiana que no se creía que se pudieran robotizar. [17]

2.2.2 Aplicaciones de la robótica

La robótica es utilizada actualmente en diversas disciplinas:

- Educación: Para promover entre los estudiantes las denominadas habilidades STEM (*Science, Technology, Engineering and Maths*).
- Industria: La robótica tiene un papel clave en el desarrollo de la industria reciente como puede ser el transporte, soldadura, montaje, reciclaje y control de calidad.
- Química: Los robots son utilizados en procesos como la disolución.
- Física: Es muy conocido su uso en exploración espacial, para poder llegar a lugares y realizar tareas que serían imposibles para los seres vivos.
- Medicina: Se pueden utilizar robots en cirugías que resultan difíciles para los humanos o para la distribución de medicamentos.
- Vida doméstica: Uno de los robots más conocidos son los robots aspiradora, los cuales pueden moverse de manera autónoma por el suelo, también hay otros como el robot de cocina.
- Militar: Los robots tienen como aporte la detección de bombas u operaciones de rescate. [19]

Figura 2.8 Robot militar. [42]

Los robots se pueden clasificar en función de la generación a la que pertenecen, el tipo de estructura que presenta o según su uso.

Clasificación según la generación del robot:

1. Primera generación: Engloba a los robots con sistemas sencillos de control, ya sea manual, de secuencia fija o variable. Su función principal es la manipulación de objetos.
2. Segunda generación: Son robots más complejos que los de la primera generación capaces de obtener información limitada del entorno. Presentan sistemas de control en lazo cerrado y sensores.
3. Tercera generación: Aparecen los robots reprogramables incorporando lenguajes complejos de programación. Utilizan sensores para la percepción del entorno y poder así, modificar la estrategia de control.

4. Cuarta generación: Añaden que la información captada por los sensores se envíe a la computadora, por lo que pueden interpretar el entorno y tomar decisiones.
5. Quinta generación: Es la más reciente, el control de los robots se basa en inteligencia artificial, de esta manera los robots tienen más autonomía.

Clasificación según la estructura:

1. Robots antropomórficos: Cuentan con dos articulaciones, una para el eje vertical y otra para elevar la articulación. Dependiendo del robot tendrá más o menos articulaciones y grados de libertad. Se suelen usar para tareas repetitivas.

Figura 2.9 Robot antropomórfico. [43]

2. Robots poliarticulados: Suelen mantenerse fijos, por lo que sus desplazamientos son limitados. Suelen tener bajos grados de libertad.
3. Robots zoomórficos: Su finalidad es imitar a seres vivos, generalmente animales que puedan andar.
4. Robots móviles: Tienen gran capacidad y facilidad para el desplazamiento. Suelen incluir ruedas, correas o patas para desplazarse. Su objetivo principal suele ser el transporte pudiendo evitar los obstáculos y llegar al objetivo analizando el entorno.
5. Robots androides: Buscan reproducir el movimiento de los humanos, ya sea de manera total o parcial.

Figura 2.10 Robot androide. [44]

6. Robots Híbridos: en esta clasificación entran los robots que no pertenecen a ninguno de los grupos anteriormente mencionados.

Clasificación según su uso:

1. Robots industriales: Tienen como objetivo el realizar tareas rutinarias y repetitivas, como puede ser el montaje de dispositivos.
2. Cobots: Están hechos para poder trabajar conjuntamente con los humanos en un espacio compartido, ayudando a los humanos a realizar tareas complejas.

Figura 2.11 Ejemplo de Cobot. [45]

3. Robots de servicio: Son robots desarrollados para facilitar la vida de las personas. Suelen ser dispositivos autónomos controlados por ordenador capaces de interactuar con las personas.
4. Robots médicos: Sirven de apoyo al sector sanitario, van desde ayudar a personas con movilidad reducida a participar en intervenciones quirúrgicas.
5. Robots domésticos: Se utilizan en tareas del hogar como limpieza o asistencia de personas.
6. Robots espaciales: Se usan bastante en misiones espaciales, para funciones de exploración. También pueden utilizarse como ayuda a astronautas.
7. Robots militares: Suelen tener como funcionalidades el transporte, el rastreo y el acompañamiento a escuadrones militares para inspección, como por ejemplo los drones.

[20] [21]

3 MARCO TEÓRICO

3.1. Modelo de una cámara

Una cámara es un dispositivo capaz de captar imágenes. Describe la forma en la que los puntos de un espacio tridimensional se proyectan en el plano bidimensional de una imagen.

Para empezar a estudiar el modelo de una cámara, hay que empezar hablando sobre la cámara oscura. Se podría decir que la cámara oscura es la pionera de las cámaras modernas, es un dispositivo óptico negro que permite obtener una proyección plana de una imagen externa sobre la zona interior de su superficie. La cámara oscura consiste en una caja cerrada con un pequeño agujero por el que entra una pequeña cantidad de luz que proyecta en la pared opuesta la imagen del exterior. [30]

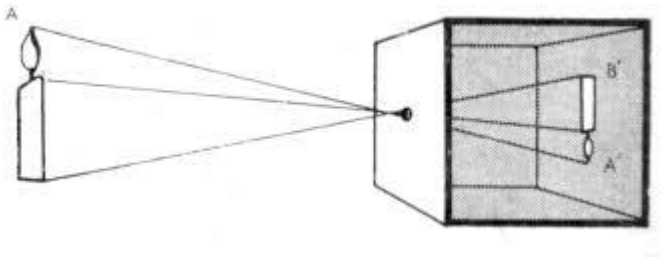

Figura 3.1 Esquema de una cámara oscura. [46]

A partir de la cámara oscura se desarrolló la cámara estenopeica, esta utiliza una caja pequeña con un orificio estenopeico muy pequeño. Para proyectar la imagen dentro de la cámara se utiliza un material sensible a la luz a diferencia de la cámara oscura que la proyectaba directamente en la pared opuesta.

Con la cámara estenopeica se consiguió ganar manejabilidad respecto a la oscura, al ser más compacta y ligera. Al tener un orificio muy pequeño, la imagen ganó nitidez.

En la cámara estenopeica, el obturador consiste en un material que tapa manualmente el orificio para controlar el tiempo de exposición. [31]

3.1.1 Modelo Pinhole

El modelo de Pinhole es la explicación geométrica de la cámara estenopeica, está basado en el principio de proyección de rayos de luz a través de un pequeño orificio. Aunque sea un modelo idealizado, es la base de funcionamiento de las cámaras reales.

Como se sabe, la cámara estenopeica no tiene lentes, la luz incide directamente en el orificio. Los puntos del espacio tridimensional se proyectan en un plano bidimensional gracias a una proyección perspectiva.

Suponiendo que se conoce el plano donde se forma la imagen proyectada a una distancia del centro óptico que se llamará distancia focal f y que el plano es perpendicular al eje óptico. Siendo el centro óptico el orificio por donde entra la luz.

En cuanto los sistemas de coordenadas, habrá un sistema Mundo (X,Y,Z) el cual está vinculado al espacio tridimensional y el plano de la imagen proyectada (x,y) que se referirá a las coordenadas en la imagen proyectada. [47]

La relación de proyección se puede expresar matemáticamente como:

$$x = \frac{fX}{Z} \tag{3.1}$$

$$y = \frac{fY}{Z} \tag{3.2}$$

Suponiendo un punto en el espacio tridimensional P=(X,Y,Z) y el asignado al plano imagen proyectada como p=(x,y).

$$p = (x, y) = \left(\frac{fX}{Z}, \frac{fY}{Z}\right) \tag{3.3}$$

Si se usan coordenadas homogéneas y aplicando un factor de escala w distinto de 0:

$$\tilde{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} xw \\ yw \\ w \end{bmatrix} \tag{3.4}$$

$$\tilde{P} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \sim \begin{bmatrix} Xw \\ Yw \\ Zw \\ 1 \end{bmatrix} \tag{3.5}$$

$$\tilde{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} xw \\ yw \\ w \end{bmatrix} \tag{3.6}$$

$$\tilde{P} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \sim \begin{bmatrix} Xw \\ Yw \\ Zw \\ 1 \end{bmatrix} \tag{3.7}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} fX \\ fY \\ fZ \end{bmatrix} \sim \begin{bmatrix} f\frac{X}{Z} \\ f\frac{Y}{Z} \\ 1 \end{bmatrix} \tag{3.8}$$

$$\tilde{p} = K * \tilde{P} \tag{3.9}$$

Donde K es la matriz de proyección. Esta matriz permite obtener las coordenadas del punto proyectado en el plano conociendo la distancia focal y las coordenadas del sistema tridimensional. Como se ha dicho, el modelo de Pinhole está idealizado, por lo que para las cámaras reales también hará falta conocer otros parámetros.

3.1.2 Parámetros intrínsecos

El modelo de Pinhole da un buen resultado, pero no llega a modelar una cámara real, por lo que habrá que añadir otros parámetros, como los intrínsecos.

Los parámetros intrínsecos de una cámara describen las características internas del sistema óptico y del sensor de imagen los cuales afectan a los puntos del sistema real que son proyectados al plano de la imagen bidimensional.

En las cámaras digitales que se usan actualmente la captura de imagen se realiza mediante sensores digitales formados por millones de píxeles. Los píxeles son rectángulos microscópicos que discretizan las coordenadas del modelo Pinhole. [47]

La relación entre el tamaño del sensor y el número de píxeles se obtiene mediante la dimensión efectiva del pixel en las direcciones horizontal y vertical (x e y). Se calculan como:

$$\rho_x = \frac{w}{N} \qquad \rho_y = \frac{h}{M} \tag{3.10}$$

(3.11)

En donde:

- ρ_x , ρ_y: Dimensiones efectivas del sensor x e y respectivamente.
- h: Altura del sensor.
- N: Número de píxeles por fila.
- M: Número de píxeles por columna.

En el modelo Pinhole el eje óptico pasa por el centro del sensor, esto provoca que el punto principal esté centrado en la imagen. Sin embargo, en la realidad suelen aparecer desalineaciones que hacen que este punto esté desplazado. Se definirá el punto principal como (u_0, v_0) teniendo en cuenta que puede estar desplazado del centro de la imagen.

En el sistema ideal, los ejes del sensor son ortogonales, pero en las cámaras reales puede ser que el ángulo entre los ejes sea diferente de 90º. Esto produce un coeficiente denominado skew. Se puede obtener como:

$$\gamma = \alpha_x * \tan(\alpha) \tag{3.12}$$

Donde:

- γ: Coeficiente skew.
- α_x: Distancia focal del eje x en píxeles.
- α: Ángulo entre los ejes x e y.

Para calcular el skew se ha utilizado la distancia focal del eje x en píxeles. Para calcular la distancia focal del eje x e y en píxeles:

$$\alpha_x = f * \frac{1}{\rho_x} \qquad \alpha_y = f * \frac{1}{\rho_y} \tag{3.13}$$

(3.14)

En la mayoría de las cámaras modernas este coeficiente es muy pequeño.

Todos estos parámetros que se han explicado se añaden a la matriz de transformación de manera que resulta:

$$K = \begin{bmatrix} \alpha_x & \gamma & u_0 & 0 \\ 0 & \alpha_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.15}$$

3.1.3 Parámetros extrínsecos

Los parámetros extrínsecos explican la relación espacial entre el sistema de coordenadas mundo y el sistema de coordenadas de la cámara. Definen como está posicionada y orientada la cámara en el mundo respecto a la imagen que se captura.

Los parámetros extrínsecos definen la traslación y la rotación. [48]

Definiendo P_w como un punto cualquiera en el sistema mundo y P_c como un punto cualquiera en el sistema cámara, se pueden relacionar como:

$$P_c = R * P_w + T \tag{3.16}$$

Donde :

- R: Matriz de rotación de 3x3.
- T: Matriz de traslación de 3x1.
- $P_c = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$: Punto del sistema mundo.
- $P_w = \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$: Punto del sistema cámara.

3.1.4 Distorsión de la lente

En la cámara estenopeica no había lente y en el modelo Pinhole tampoco se consideró, pero en las cámaras reales es importante considerar la distorsión que pueda introducir la lente en la cámara que se utilice. Estas distorsiones pueden ser producidas por la fabricación o ser geométricas, hay que tenerlas en cuenta para tener la mayor precisión.

A continuación, se explicarán dos tipos de distorsión, la distorsión radial y la distorsión cromática.

Distorsión radial.

La distorsión radial es a causa de la curvatura de las lentes, lo que hace que los rayos de la luz no se proyecten linealmente, esto ocasiona que los puntos de la imagen se desplacen. La corrección de esta distorsión se realiza con el siguiente modelo:

$$x' = x * (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.17}$$

$$y' = y * (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.18}$$

Donde:

- x e y: Puntos a corregir.
- k_1, k_2, k_3: Coeficientes de distorsión radial.
- $r = \sqrt{x^2 + y^2}$: Distancia radial.
- x´e y´: Coordenadas con la distorsión radial corregida.

Distorsión Tangencial.

La distorsión tangencial se produce durante la fabricación de la lente a causa de una desalineación entre la lente y el sensor, esto provoca un desplazamiento tangencial de los puntos en la imagen. Se puede modelar como:

$$x' = x + [2p_1xy + p_2(r^2 + 2x^2)] \tag{3.19}$$

$$y' = y + [p_1(r^2 + 2y^2) + 2p_2xy] \tag{3.20}$$

En donde:

- x e y: puntos a corregir.
- p_1, p_2: parámetros de distorsión tangencial.
- $r = \sqrt{x^2 + y^2}$: distancia radial.
- x´e y´: coordenadas con la distorsión radial corregida.

Combinando las ecuaciones tanto de la distorsión radial como de la tangencial se puede obtener la distorsión total.

$$x' = x * (1 + k_1r^2 + k_2r^4 + k_3r^6) + [2p_1xy + p_2(r^2 + 2x^2)] \tag{3.21}$$

$$y' = y * (1 + k_1r^2 + k_2r^4 + k_3r^6) + [p_1(r^2 + 2y^2) + 2p_2xy] \tag{3.22}$$

Siendo x´e y´ las coordenadas con la distorsión total corregida.

3.1.5 Método Zhang

El método Zhang, el cual lleva el nombre de su creador *Zhengyou Zhang*, permite obtener los parámetros intrínsecos y extrínsecos de una cámara a partir de las imágenes tomadas a un patrón plano en diferentes posiciones y ángulos. Este método ayuda a corregir la distorsión de la lente y mejora la relación entre las coordenadas de la imagen y las del mundo real. [55]

3.2. Homografía

Ya se hizo una breve introducción en el capítulo 2.1, pero ahora se profundizará más.

La homografía es una transformación que conserva la naturaleza de los elementos transformados. [32]

3.2.1 Homografía en geometría

Las homografías fueron introducidas para estudiar la perspectiva y proyecciones en la forma euclídea, permitiendo entender cómo se representan objetos tridimensionales en una superficie bidimensional.

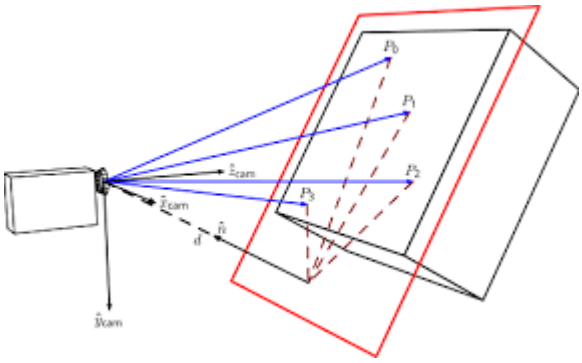

Figura 3.2 Esquema de una homografía. [49]

Las propiedades de la homografía son: [8]

- Biyectividad: Cada punto del espacio proyectivo tiene una imagen única y cada punto de la imagen una única preimagen.
- Razón doble: La razón doble de cuatro puntos que se encuentran en la misma línea se conserva en una homografía. La razón doble de cuatro puntos alineados (a,b,c,d), se puede definir como: [56]

$$R(a,b,c,d) = \frac{\overline{ca}}{\frac{\overline{cb}}{\frac{\overline{da}}{\overline{db}}}} \tag{3.23}$$

- Composición de homografías: si se componen 2 homografías se obtiene otra homografía.

La matriz de homografía representa la transformación lineal en el espacio proyectivo, al trabajar en este espacio habrá que utilizar coordenadas homogéneas para representar los puntos.

Suponiendo que hay dos planos en el espacio, para transformar los puntos de un plano a otro, la relación de homografía para un punto p con otro p' se puede definir cómo:

$$p = H * p' \tag{3.24}$$

Donde:

- $p = (x, y, z)^T$ Coordenadas homogéneas del punto en el plano original.
- $p' = (x', y', z)^T$ Coordenadas homogéneas del punto en el plano transformado.
- $H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$ Matriz de homografía de 3x3.

Desarrollando y suponiendo que z=1 al ser la imagen bidimensional.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.25}$$

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \tag{3.26}$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \tag{3.27}$$

Pasando los denominadores al otro lado:

$$x'(h_{31}x + h_{32}y + h_{33}) = h_{11}x + h_{12}y + h_{13} \tag{3.28}$$

$$y'(h_{31}x + h_{32}y + h_{33}) = h_{21}x + h_{22}y + h_{23} \tag{3.29}$$

$$xh_{11} + yh_{12} + h_{13} - x'xh_{31} - x'yh_{32} - x'h_{33} = 0 \tag{3.30}$$

$$xh_{21} + yh_{22} + h_{23} - y'xh_{31} - y'yh_{32} - y'h_{33} = 0 \tag{3.31}$$

Se puede considerar que $h_{33} = 1$ dado que cualquier múltiplo escalar de H representa la misma transformación. Con esto se obtienen 8 grados de libertad, por lo que se necesitan un mínimo de 4 puntos en el plano original de un espacio bidimensional y sus respectivas coordenadas en el plano transformado.

Resolviendo el sistema de ecuaciones usando el método que mejor convenga, mínimos cuadrados podría ser una opción, se obtienen los valores de la matriz de homografía.

Una vez conocida la matriz de homografía, se puede conseguir la transformación proyectiva de todos los puntos entre los dos planos. [10]

Las homografías se suelen utilizar para relacionar dos imágenes tomadas desde distintas perspectivas.

3.2.2 Homografía en imágenes

La homografía se usa en visión para encontrar la relación entre dos imágenes que cubran la misma escena, pero tomadas desde perspectivas diferentes. Esto es aprovechado en tecnologías actuales como la creación de imágenes panorámicas. [33]

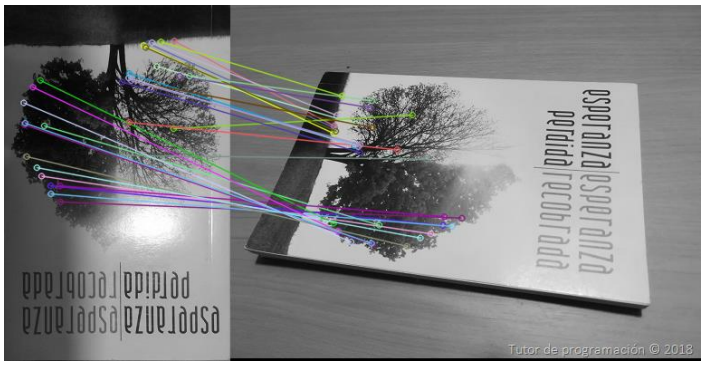

Figura 3.3 Homografía de la carátula de un libro. [50]

3.3. Control de robots con ROS

Robot Operating System (ROS) es un kit de desarrollo de código abierto cuya finalidad es el desarrollo de proyectos de robótica.

Surgió en el año 2007 con el nombre *switchyard* en el Laboratorio de Inteligencia Artificial de Stanford, con el fin de dar soporte a un proyecto de robótica con IA. A partir de 2008, el desarrollo del proyecto continúa en el instituto de investigación robótico *Willow Garage* en donde ROS empezó a ser un software referente en robótica. [51]

ROS está orientado para el sistema operativo Ubuntu (Linux) principalmente, pero también se está adaptando a otros como Debian o Windows, aunque aún son catalogados como experimentales.

ROS se divide en componentes llamados nodos que se pueden comunicar e intercambiar mensajes. Los procesos de ROS están agrupados en los nodos (*ROS nodes*) y están conectados por los *ROS topics* que funcionan como canales para la publicación o suscripción de la información. El proceso conocido como *ROS master* se encarga de registrar todos los nodos que se encuentran activos, facilita la conexión entre nodos y controla las actuaciones del servidor de parámetros. Los mensajes y llamadas no pasan por el *ROS master*, sino que este hace de guía para que puedan encontrarse entre ellos. Al ser los nodos independientes al *ROS master* se reduce la carga de trabajo de este.

Entre las principales herramientas de ROS, se encuentran:

- Rosbag: Permite grabar y reproducir datos del sistema dentro de ROS.
- Rviz: Es una herramienta de simulación y visualización 3D para robots, incluyendo la visualización de sensores como cámaras o LIDAR. También se usa para mapeo de robots, de manera que se muestre información de la posición, orientación y trayectoria de este.
- Gazebo: Simula robots en 3D sin necesidad de tener un robot en físico.
- Catkin: Se usa para la compilación de paquetes.
- Roslaunch: Es utilizado para poder ejecutar múltiples nodos de ROS de forma local o remota.

En cuanto a las aplicaciones de ROS:

- Robots Móviles: Para localización, mapeo y navegación de los robots.
- Brazos robóticos: Automatización de fábricas.
- Vehículos autónomos y drones: Detección y evasión de obstáculos y cálculo de trayectorias.

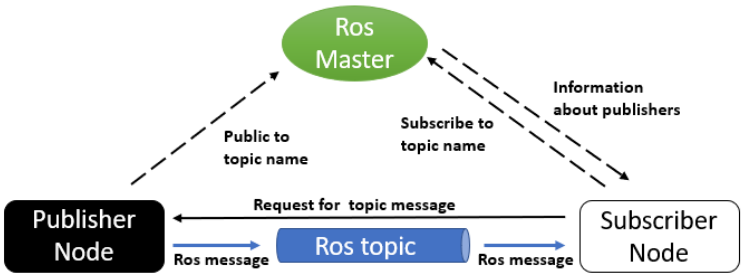

Figura 3.4 Esquema de funcionamiento de ROS. [52]

4 RECURSOS EMPLEADOS

4.1. Hardware

A continuación, se expone el hardware utilizado:

- **Cámaras**: Se usarán 4 cámaras *Genie Nano M1280-NIR*. Es un tipo de cámara industrial desarrollada por *Teledyne DALSA*, suelen utilizarse en aplicaciones que necesitan de espectroscopía del infrarrojo cercano. Equipa un sensor Python 1300 monochrome 1/2.3" CMOS, por otro lado, tiene una resolución de 1280 x 1024 píxeles y una velocidad de captura de hasta 92 fotogramas por segundo. La cámara soporta el SDK (Kit de desarrollo de software) Sapera LT creado por el propio fabricante para poder gestionar la adquisición de imágenes, la comunicación y el procesamiento. [28]

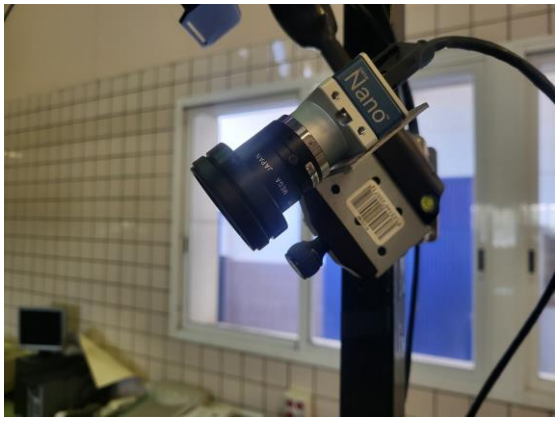

Figura 4.1 Cámara Genie Nano M1280-NIR.

- **Focos**: Al lado de cada una de las cámaras hay un foco infrarrojo de alta potencia *Komoto LV7-T*, en total hay 4 focos. Emite un espectro NIR cuya longitud de onda es de 850 nm. Usa tecnología LED. Alcanza hasta 40 metros. Cada foco posee dos unidades las cuales se pueden ajustar. Tiene un consumo de 16 W y necesita una alimentación de 12 V DC. [29]

Figura 4.2 Foco Komoto LV7-T.

- **Fuente de alimentación de los focos:** Cada foco necesita una alimentación de 12 V de corriente continua. Esta fuente de alimentación es capaz de proporcionar 6.3 Amperios. El modelo es MEAN WELL SDR-75-12. [57]

Figura 4.3 Fuente de alimentación MEAN WELL SDR-75-12.

- **Ordenador:** Habrá un ordenador principal para la realización de los experimentos, el cual posee entre otras características:
    - Procesador Intel Core i7-8700.
    - Memoria RAM de 16 GB.
    - Memoria ROM SSD de 512 GB.
    - Tarjeta gráfica *NVIDIA GeForce GTX 1660*.
    - Sistema Operativo *Windows 10 Home*
    - Tarjeta *PCIe-PoE550X*
    - Placa base *Gigabyte B360M DS3H*

Figura 4.4 Ordenador del proyecto.

- **Monitor:** Pantalla de la marca *Dell*.

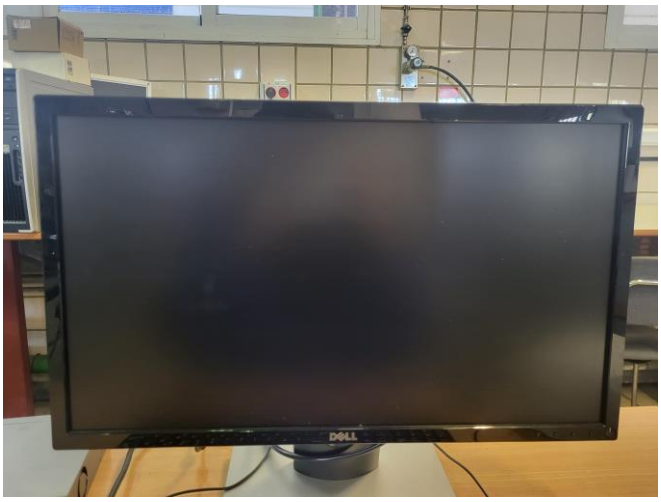

Figura 4.5 Monitor para el proyecto.

- **Switch:** El *Netgear MS510TXPP* posee 8 puertos POE+, con una potencia total de 180 W para alimentar cámaras IP o puntos de acceso inalámbrico. Es compatible con IPv4 e IPv6. [58]

Figura 4.6 Switch Netgear MS510TXPP.

- **Adaptador Wi-Fi:** *TP-LINK ARCHER T3U,* permite conectarse a redes Wi-Fi y ofrece velocidades de hasta 400 Mbps en la banda de 2.4 GHz y 867 Mbps en la de 5 GHz. [53]

Figura 4.7 Adaptador Wi-Fi TP-LINK ARCHER T3U. [53]

- ***Turtlebot 3 Burger*:** Es un pequeño robot móvil para el desarrollo de sistemas autónomos con el framework ROS creado por la empresa *ROBOTIS*, además, el robot está equipado con una *Raspberry Pi 4 model B*, un LIDAR de 360º, una unidad de control *OpenCR* y dos servomotores *Dynamixel* para controlar el movimiento de dos de las ruedas, además posee una tercera rueda la cual es loca. [59]

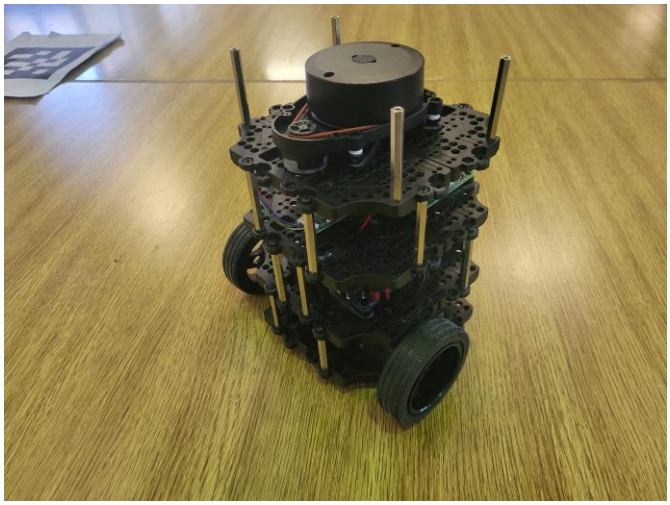

Figura 4.8 Turtlebot 3 Burger.

- **Smartphone:** El smartphone es necesario para crear una zona Wi-Fi, se usará un *Samsung Galaxy S21 FE*.

Figura 4.9 Smartphone Samsung S21 FE. [54]

- **Otros periféricos:** Además de lo anteriormente mencionado se usará teclado y ratón para usar el ordenador además de los cables necesarios.

4.2. Software

- ***Windows 10 home*:** Es un sistema operativo desarrollado por *Microsoft* diseñado para el uso personal y doméstico. [60]
- ***Sapera LT*:** Consiste en un kit de desarrollo de software de *Teledyne DALSA* para la adquisición y manipulación de imágenes de cámaras industriales, compatible con los dispositivos de la propia marca. Integra herramientas como *Sapera CamExpert*, el cual, permite ajustar los parámetros de las cámaras. Por otra parte, funciona en sistemas operativos *Windows* y soporta lenguajes de programación como C y C++. [61]

- ***Microsoft Visual Studio Community 2022*:** Es una edición gratuita de *Visual Studio* diseñado por Microsoft para estudiantes, desarrolladores y pequeñas empresas. Permite programar, depurar y compilar en diferentes lenguajes y plataformas. Entre los lenguajes que soporta se encuentran C, C++, Python y JavaScript. [62]
- ***WSL 1*:** *Windows Subsystem for Linux*, permite ejecutar un entorno Linux directamente en Windows sin necesitar arranque dual o una máquina virtual. [63]
- ***Ubuntu 20.04 LTS***: Es una versión del software libre *Ubuntu* desarrollado por Linux. Suele utilizarse para desarrollo y servidores. [64]
- ***Visual Code***: Editor de código abierto desarrollado por *Microsoft* que posee una gran cantidad de extensiones que se pueden incluir. [65]
- ***ROS Noetic***: Versión de ROS lanzada para *Ubuntu 20.04* la cual utiliza Python 3 e introdujo bastantes mejoras respecto a la versión anterior. [66]
- ***VcXsrv***: Servidor X para Windows que permite la ejecución de aplicaciones gráficas de *Linux* en *Windows*. [67]
- **Python:** Lenguaje de programación que posee una sintaxis clara y sencilla, contiene gran número de librerías. [68]
- **C++:** Lenguaje de programación compilado de alto rendimiento. [69]

4.3. Otros recursos

- **Plataforma de trabajo:** Estará constituida por dos mesas de superficie de madera y patas metálicas de 1 metro de alto, 2.125 metros de largo y 1.625 metros de ancho. Las dos mesas están unidas formando una cuyas dimensiones serán de 2.125 metros de largo y 1.625 metros de ancho. En los capítulos posteriores se hablará de esta plataforma de trabajo como mesa.
- **Marcadores ArUco**: Marcadores para la estimación de posición. Cada uno consta de una imagen monocroma cuadrada con un patrón binario. Al codificarse el patrón se obtiene un número que lo identifica. El diccionario que se usará en este proyecto es el de 6x6 bits existiendo hasta 250 identificadores posibles. En un principio, habrá 4 marcadores en las esquinas con sus identificadores (ID 1, 2, 3 y 4) y otro en el centro (ID 50).

4.4. Pruebas iniciales

Para este proyecto se partía de la estación de trabajo ubicada en el Laboratorio del Departamento de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla. Dicha estación, es una estructura cúbica con cuatro cámaras industriales en cada esquina que apuntan a una mesa. Además, cuenta con un ordenador, el conexionado de las cámaras y un switch. Por otro lado, se nos proporcionó un *Turtlebot 3 Burger* como robot móvil.

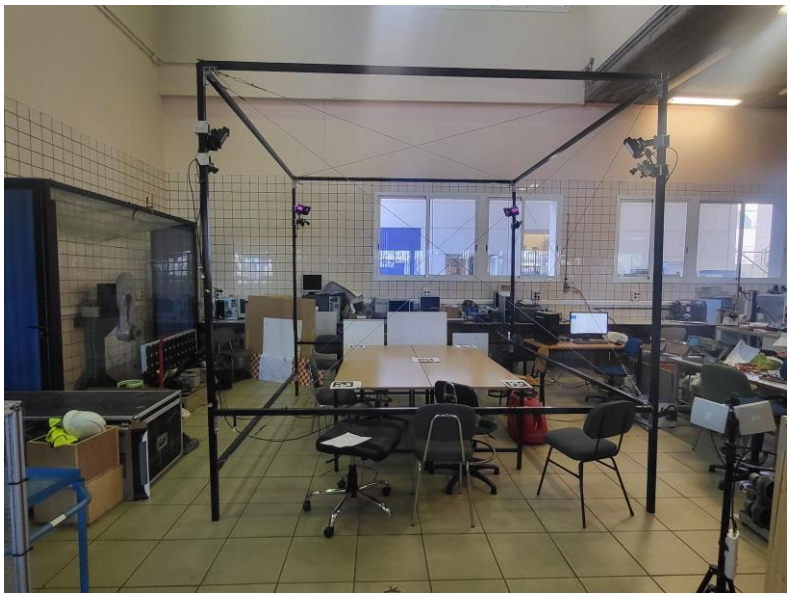

Figura 4.10 Estación de trabajo.

Desde el primer momento, uno de los mayores problemas fue que el software para el control de cámaras *Sapera LT* solo funcionaba en *Windows*, y ROS, para la conexión con el robot, solo funcionaba en *Ubuntu*, por lo cual, era un problema difícil de resolver al necesitar dos sistemas operativos independientes funcionando al mismo tiempo.

La primera idea fue intentar conectar las cámaras en *Ubuntu* y realizar todo el proyecto en este sistema operativo. Para realizar la configuración de las cámaras se utilizaría el *GigE-V framework* el cual proporciona soporte para cámaras de la marca Teledyne DALSA en *Linux*. Como el único sistema operativo instalado en el ordenador era *Windows 10 Home* se realizó una partición de disco para instalar también *Ubuntu 20.04.6 LTS*. Tras el sistema operativo, se instaló *Gige-V framework*, pero tras intentar conectar las cámaras, apareció el problema de que existía incompatibilidad con los controladores, por lo que la idea de usar solo *Ubuntu* tuvo que ser descartada.

La segunda idea era utilizar ROS directamente en *Windows*, pero aparecía el problema de que el *Turtlebot 3 Burger* no tiene soporte para esta versión de ROS. La opción que ofrecía ROBOTIS, el fabricante de *Turtlebot 3*, para Windows era que, en vez de utilizar una *Raspberry Pi model B* para la comunicación con el robot, usar un pequeño ordenador ensamblado al robot con *Windows* y conectarse a este desde el ordenador principal remotamente. La opción fue descartada dado que no se poseía del PC ensamblado y era una opción costosa y difícil.

La tercera idea fue utilizar *Matlab*. El software *Matlab* posee soporte tanto para ROS como para las cámaras *Genie Nano*, esta idea en principio funcionaba, pero tenía algunos problemas tanto en el tratamiento de las imágenes de las cámaras como en las conexiones con los nodos de ROS, pero el uso de *Matlab* sirvió como precursor de la solución final.

Con los problemas que daba *Matlab* y tras investigar alternativas se llegó a una, usar WSL (Windows Subsystem Linux). WSL permite acceder a *Linux* desde una máquina *Windows*, lo que hacía posible poder acceder a *Ubuntu 20.04* desde *Windows* usando esta aplicación. Además, WSL crea una carpeta a la cual se puede acceder directamente desde Windows con todos los archivos usados en *Ubuntu*. WSL posee dos versiones WSL 1 y WSL 2, aunque WSL 2 sea la más moderna y mejor, se tuvo que optar por WSL 1 dado que WSL 2 no permite el acceso a múltiples direcciones IPs al usar una IP única virtual, pero WSL 1 sí comparte directamente las IPs. Si no se usaran directamente las mismas direcciones IP reales en WSL podría haber problemas con ROS.

Ya en WSL 1 con *Ubuntu 20.04* se instalaron tanto ROS como las librerías necesarias para el control del *Turtlebot 3 Burger*. El resultado fue bastante positivo, lo único que hubo que añadir fue un servidor X para *Windows* con el fin de poder abrir imágenes directamente en WSL, al no tener este, soporte por sí mismo para aplicaciones gráficas.

A diferencia de con *Matlab*, con WSL no había problemas con ROS, pero tampoco había soporte para las cámaras. Aunque al poder acceder a carpetas de *Windows* desde WSL y viceversa ya se podría hacer la captura

de imágenes en Windows y después acceder a ellas desde WSL. Por lo cual, ya se podía hacer un programa de captura de imágenes en C++ basado en el de *"Sistema multicámara para seguimiento a tiempo real de objetivos en movimiento. Adaptación a nuevo hardware y estimación tridimensional"* [34] y después realizar el tratamiento de imágenes para el guiado del robot con un programa de Python ejecutado en WSL, teniendo el control del robot con ROS en este mismo programa.

5 MONTAJE DE LA ESTACIÓN DE TRABAJO

5.1. Distribución de cámaras

En la estación de trabajo, como se dijo en el capítulo 4.4, hay una estructura metálica cúbica con cuatro cámaras, cada una situada en una de las esquinas superiores de la estructura y apuntando a una plataforma de trabajo rectangular que se nombrará en adelante como mesa. Para mayor comodidad se denominará a cada cámara con un número. De esta manera las cámaras serán denominadas cámara 1, cámara 2, cámara 3 y cámara 4. En la Figura 5.1 se muestra un esquema de la distribución.

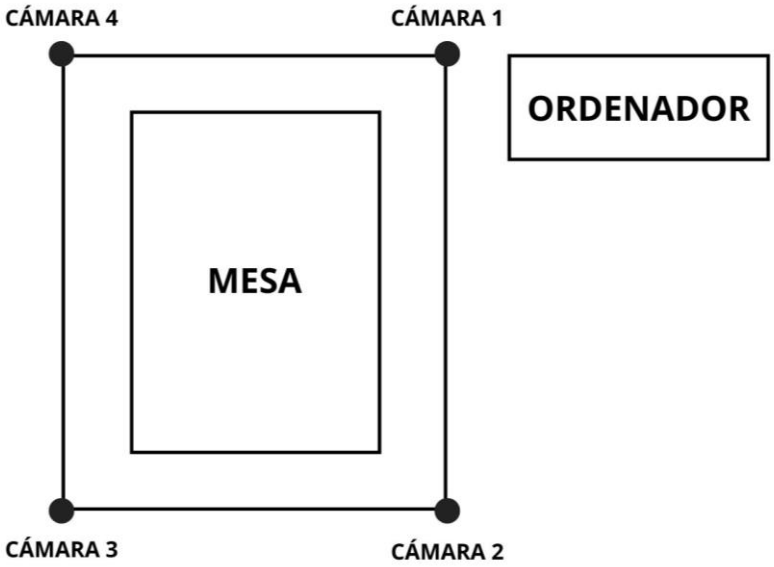

Figura 5.1 Distribución de las cámaras.

5.2. Conexionado de cámaras

Al lado de cada cámara se encuentran un foco infrarrojo. Los focos se alimentan con una fuente de alimentación de 12 V DC común. Cada cámara va conectada a uno de los puertos del switch mediante un cable de red, a su vez el switch va conectado también por cable de red a uno de los puertos Ethernet del ordenador.

Figura 5.2 Fotografía de conexionado.

5.3. Configuración y sincronización de cámaras

Para la configuración de las cámaras se dispone del software *Sapera Cam Expert* y del *Network Configuration Tool*. Desde este último se puede visualizar la dirección asignada a cada cámara y establecer una dirección fija para cada una.

El *Network Configuration Tool* permite la configuración de las direcciones IP de las cámaras conectadas al switch mediante el cable de red. Cada cámara conectada a un puerto Ethernet del switch tendrá una dirección IP propia, un número de serie y una dirección MAC.

Para cada una de las cámaras se decidió que el nombre que la define fuera el mismo que su número de serie.

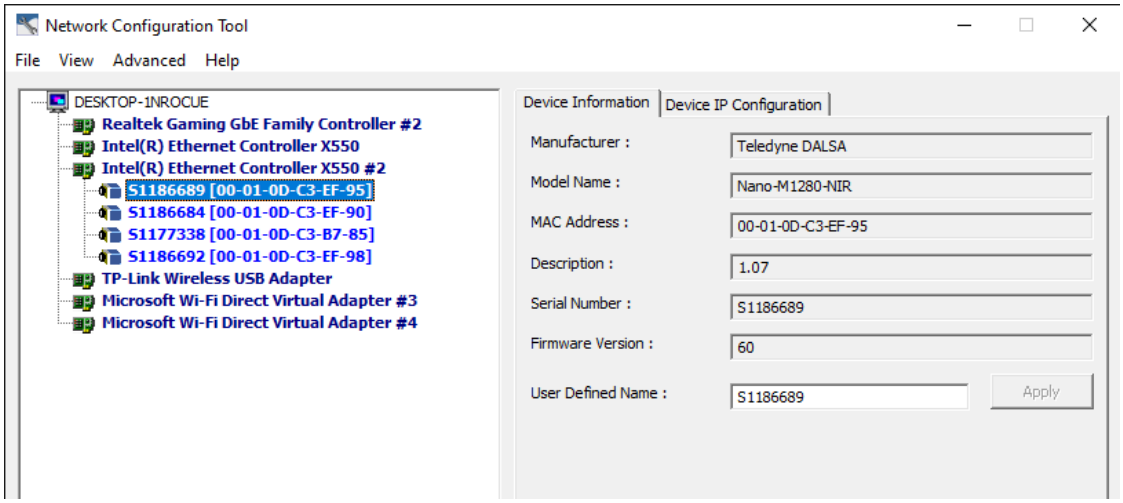

Figura 5.3 Información de la cámara en Network Configuration Tool.

A cada cámara se le asigna una dirección IP fija para que esta no varíe cada vez que se conecten.

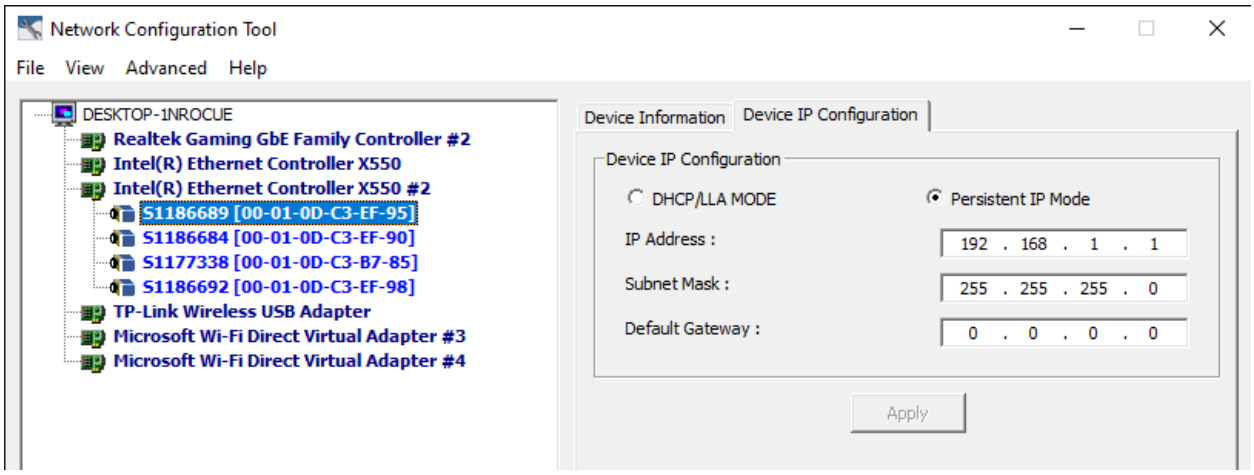

Figura 5.4 Configuración de dirección IP en Network Configuration Tool.

Con la realización de esta configuración, cada cámara queda definida con una dirección IP fija propia de la siguiente manera: (Todas las cámaras son del modelo Teledyne DALSA Nano-M1280-NIR)

- Cámara 1:
  - Número de serie: S1186692
  - Dirección MAC: 00-01-0D-C3-EF-98
  - Dirección IP: 192.168.1.3
  - Nombre que lo define: S1186692
- Cámara 2:
  - Número de serie: S1177338
  - Dirección MAC: 00-01-0D-C3-B7-85
  - Dirección IP: 192.168.1.4
  - Nombre que lo define: S1177338
- Cámara 3:
  - Número de serie: S1186684
  - Dirección MAC: 00-01-0D-C3-EF-90
  - Dirección IP: 192.168.1.6
  - Nombre que lo define: S1186684
- Cámara 4:
  - Número de serie: S1186689
  - Dirección MAC: 00-01-0D-C3-EF-95
  - Dirección IP: 192.168.1.1
  - Nombre que lo define: S1186689
- Switch:
  - Dirección MAC: 78-D0-04-2D-1F-AC
  - Dirección IP: 192.168.1.2

En cuanto al *Sapera Cam Expert* se decidió dejar la configuración implementada en el proyecto del *"Sistema multicámara para seguimiento a tiempo real de objetivos en movimiento. Adaptación a nuevo hardware y estimación tridimensional"* [34]. La cual usa un protocolo PTP. Las características se detallarán en el capítulo 7.2.

5.4. Conexión a zona Wi-Fi del ordenador

Para poder controlar el robot a distancia y usar ROS, se necesita que tanto el robot como el ordenador se encuentren en la misma red Wi-Fi, para ello es necesario que el ordenador posea un adaptador Wi-Fi y crear una red Wi-Fi. Para ello se usará un smartphone con la función "zona Wi-Fi", esta permite convertir el dispositivo en un punto de acceso inalámbrico para transferencia de archivos y compartir conexión a internet y dispositivos locales.

Los pasos por seguir son los siguientes:

1. Conexión de un adaptador Wi-Fi al ordenador.
2. Creación de una zona Wi-Fi desde el smartphone (También se podría usar la zona Wi-Fi de un router).

Figura 5.5 Creación de la zona Wi-Fi.

3. Conexión del ordenador a la zona Wi-Fi creada.
4. (Opcional) Desde el móvil se puede ver si la conexión es correcta y la dirección IP asignada al ordenador.

Figura 5.6 Dispositivo conectado a zona Wi-Fi.

Figura 5.7 Datos de la conexión Wi-Fi.

5.5. Ejecución de WSL

Como ya se explicó en el capítulo 4.4. El uso de WSL radica en que por un lado *Sapera LT* y la comunicación, calibración y adquisición de imágenes solo se puede realizar en el sistema operativo *Windows*, por otro lado, ROS, el framework necesario para el control del robot solo opera en *Ubuntu*. Por lo que para poder usar *Ubuntu* en *Windows* se hará uso de *Windows Subsystem for Linux* (WSL).

Se hará uso en WSL de *Ubuntu 20.04.6 LTS* dado que es la versión necesaria para utilizar *ROS Noetic*.

Para ejecutarlo basta con escribir WSL en la barra de búsqueda de Windows y se abre la versión seleccionada por defecto.

Para generar más pestañas se puede ejecutar otra vez WSL desde la barra de búsqueda de *Windows*.

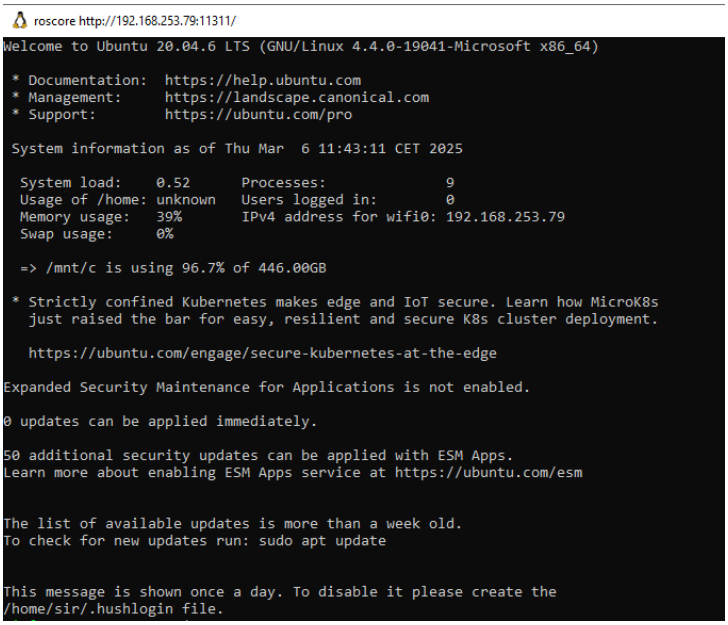

Figura 5.8 Inicio de WSL con Ubuntu 20.04.6.

5.6. ROS y puesta a punto del robot

Como se dijo en el capítulo 4.4, en este proyecto se hace uso del *Turtlebot 3 Burger* como robot móvil. Para conectarse a este robot es necesario el uso de ROS. El propio fabricante, *ROBOTIS* pone a disposición de los usuarios diferentes versiones de ROS y ROS2 en su página web además de guías de funcionamiento. [70]

Tras el testeo de varias versiones se decidió que la distribución más adecuada sería *ROS Noetic*.

El primer paso sería la instalación de *ROS Noetic* en el ordenador. *ROS Noetic* solo funciona con *Ubuntu 20.04*, por ello, se instaló esta versión para WSL. Dentro de un terminal de WSL hay que instalar *ROS Noetic* y las librerías correspondientes para el *Turtlebot3*.

Una vez instalado ROS Noetic en el ordenador se debe asignar la dirección IP Maestra (*ROS MASTER URI*), que es la dirección que actúa como servidor central para la comunicación entre los nodos. Por otro lado, el *ROS Master* también define la dirección IP que el resto de los nodos usa para conectarse con el nodo local (*ROS HOSTNAME*). Si se quiere que este nodo local sea el nodo maestro el *ROS Master* y el *ROS hostname* deben coincidir.

Para introducir la dirección de la IP maestra y del nodo local se debe escribir en el terminal:

$ nano ~/.bashrc

Editar los parámetros de la Figura 5.9:

```
export ROS_MASTER_URI=http://192.168.186.79:11311
export ROS_HOSTNAME=192.168.186.79
```

Figura 5.9 Introducción de direcciones IP.

Tras guardar el archivo, para aplicar los cambios hay que teclear: *$ source ~/.bashrc*

Una vez que el ordenador esté conectado a la zona Wi-Fi y esté definido el *ROS Master*, se debe ejecutar "*roscore"* en un terminal de WSL, con esta acción se inicializan los servicios de ROS para la comunicación entre nodos.

```
sir@DESKTOP-1NROCUE:~$ roscore
... logging to /home/sir/.ros/log/93e8345c-0e1b-11f0-a6cb-e0d55edc7d06/roslaunch-DESKTOP-1NROCUE-113.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.186.79:15289/
ros_comm version 1.16.0


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.16.0

NODES

auto-starting new master
process[master]: started with pid [121]
ROS_MASTER_URI=http://192.168.186.79:11311/

setting /run_id to 93e8345c-0e1b-11f0-a6cb-e0d55edc7d06
process[rosout-1]: started with pid [133]
started core service [/rosout]
```

Figura 5.10 Visualización de roscore.

Para la conexión del robot, lo primero es cargar la imagen de *Ubuntu 20.04.6* en una tarjeta microSD, para poder usarse en *la Raspberry 4* y posteriormente, instalar *ROS Noetic* y las librerías correspondientes. Para este proyecto se ha usado una imagen del sistema operativo especial para el robot dada por el propio fabricante *ROBOTIS*, la cual ya viene de serie con las librerías y *ROS Noetic*. Lo único que hay que configurar es el acceso a la zona Wi-Fi, esto se puede hacer conectando la tarjeta microSD a un ordenador o máquina virtual con *Ubuntu* y escribiendo lo siguiente en un terminal:

$ cd /media/$USER/writable/etc/netplan

$ sudo nano 50-cloud-init.yaml

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: yes
      dhcp6: yes
      optional: true
  wifis:
    wlan0:
      dhcp4: yes
      dhcp6: yes
      access-points:
        WIFI_SSID:
          password: WIFI_PASSWORD
```

Figura 5.11 Configuración Wi-Fi para la Raspberry. [74]

Habría que cambiar WIFI_SSID por el nombre de la red y escribir su contraseña donde pone WIFI_PASSWORD.

Tras esto, para configurar el sistema operativo del robot, se debe introducir la tarjeta microSD en la *Raspberry Pi* y conectarse con un monitor y teclado. Al acceder, al igual que en el ordenador se debe escribir la dirección del *ROS Master* y del *ROS hostname*, si el nodo del ordenador es el maestro se deberá introducir como ROS Master la dirección IP del ordenador, por otro lado, el *ROS Hostname* será la dirección IP del robot, la cual se puede saber usando el comando *"ifconfig"* o viéndolo directamente en el smartphone que ha creado la zona Wi-Fi. Ya hecho este proceso, basta con seguir la guía proporcionada por *ROBOTIS* para configurar el LIDAR. [71]

Para la configuración de la placa OPENCR que está conectada a la Raspberry, a los motores de las ruedas y a la batería del robot, además de actuar de enlace entre los sensores, actuadores, el microcontrolador y la Raspberry. Basta con seguir la guía de usuario desde el terminal de la Raspberry. [72]

Si hiciera falta el montaje del robot, el fabricante también aporta un enlace con un documento y vídeo explicativo. [73]

Una vez terminada tanto la configuración del ordenador como la del robot, hay que seguir los siguientes pasos cada vez que se quiera arrancar el robot. Es importante recordar que tanto el robot como el ordenador deben estar conectados a la misma red Wi-Fi.

1. Desde el ordenador y suponiendo que ya está conectado a la red. Ejecutar la orden "*roscore"* en un terminal de WSL, con esta acción se inicializan los servicios de ROS para la comunicación entre nodos.
2. Encender el robot y conectarse mediante escritorio remoto desde el ordenador. Para ello se debe ejecutar.

   $ ssh ubuntu@{IP_ADDRESS_OF_RASPBERRY_PI}

   Siendo *{IP_ADDRESS_OF_RASPBERRY_PI}*, la dirección IP de la Raspberry.

   Tras esto nos pedirá el usuario y contraseña o solo contraseña. Para la imagen de Ubuntu dada por el fabricante:

   Usuario: ubuntu

   Contraseña: turtlebot
3. Una vez se haya accedido remotamente al robot se debe seleccionar el modelo de *Turtlebot 3*, para este caso es el *Burger*, por lo cual hay que ejecutar:

$ export TURTLEBOT3_MODEL=Burger

Y a continuación, la orden para realizar la calibración del robot.

$ roslaunch turtlebot3_bringup turtlebot3_robot.launch

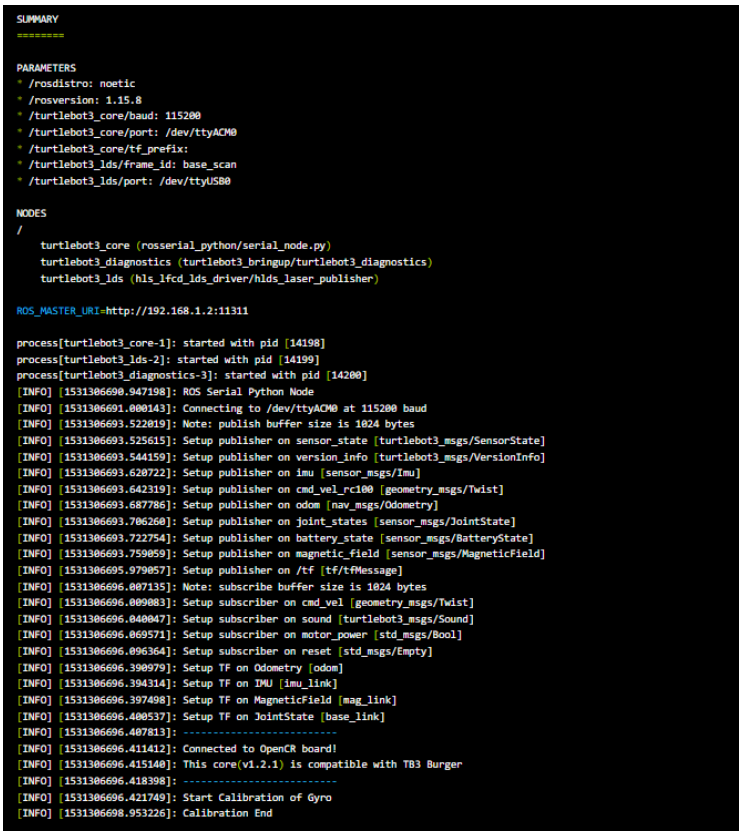

Figura 5.12 Bringup del Turtlebot 3. [74]

Esto creará la lista de *Topics* del robot y comprobará que tanto la placa OpenCR, como el LIDAR y los motores funcionan correctamente.

4. Tras la calibración del robot, se puede comprobar, como ayuda, que el robot se mueve ejecutando los siguientes comandos:

   $ export TURTLEBOT3_MODEL=Burger

   $ roslaunch teleop_twist_joy teleop.launch

   Esto carga un programa dado por el fabricante que permite modificar la velocidad lineal y angular del robot. También se puede comprobar la lista de *Topics* escribiendo:

   $ rostopic list

   [74] [75]

6 MÉTODOS DE CALIBRACIÓN PARA LAS CÁMARAS

6.1. Calibración estéreo

En proyectos anteriores, la calibración de las cámaras se realizaba calculando los parámetros intrínsecos y extrínsecos y la distorsión de la lente tomando bastantes fotos de un plano ajedrez, del cual se conocían las dimensiones, ubicándolo cada toma en una posición y orientación diferente a lo largo de la mesa. Con las fotos se hacía una calibración estéreo haciendo pares de cámaras adyacentes.

El objetivo con la calibración estéreo es realizar una reconstrucción 3D de la escena, en este caso de la mesa, a partir de un par de cámaras cada una situada en puntos diferentes, pero enfocando a la misma escena. Para ello habrá que tomar diferentes fotos de un patrón ajedrez cambiando su posición y orientación en cada toma. Antes de realizar la calibración estéreo se deben obtener los parámetros intrínsecos de las cámaras (matriz de parámetros intrínsecos y coeficientes de distorsión) usando el método Zhang explicado en el capítulo 3.1.5. Una vez con los parámetros intrínsecos obtenidos partiendo de las esquinas del patrón en las dos cámaras, se emparejan los puntos detectados de las dos imágenes. Con ello se puede obtener la matriz de rotación y el vector de traslación.

La matriz de rotación informa sobre cómo está orientada la segunda cámara respecto a la primera.

El vector de traslación indica el desplazamiento de la segunda cámara respecto a la primera.

Como prueba, se utilizó un set de 14 fotos tomadas simultáneamente por cada cámara, colocando una plantilla de tablero de ajedrez en diferentes posiciones, cuya distribución de cuadrados es de 20 columnas y 10 filas, con cuadrados de 5 cm de lado.

Figura 6.1 Sexta foto desde la cámara 1.

Figura 6.2 Sexta foto desde la cámara 2.

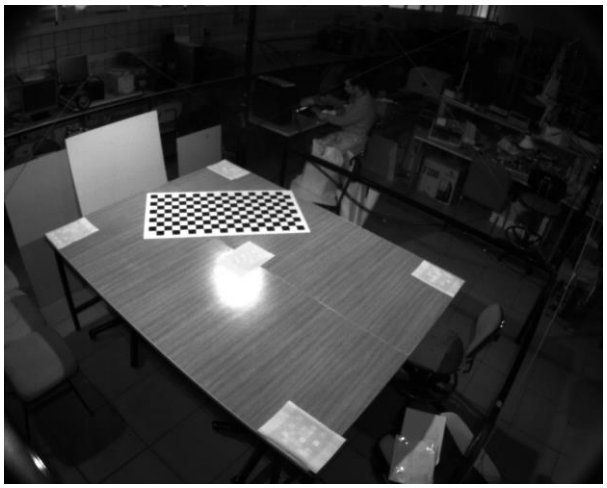

Figura 6.3 Sexta foto desde la cámara 3.

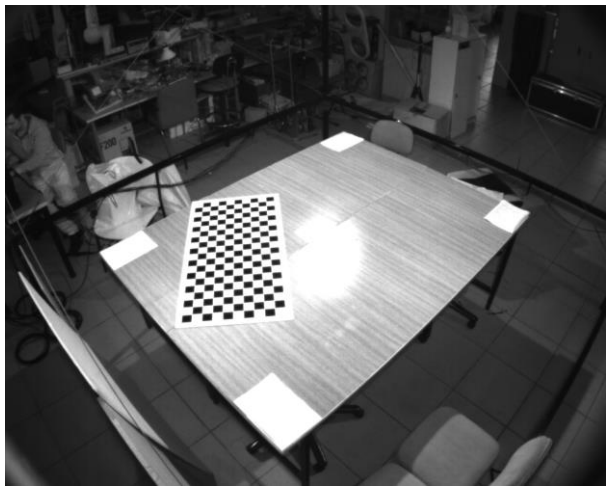

Figura 6.4 Sexta foto desde la cámara 4.

Se obtuvieron los parámetros intrínsecos para cada cámara con un simple programa de Python, pero se observó que en una gran cantidad de imágenes introducidas no se pudo detectar el patrón correctamente debido a los reflejos de luz que se producían principalmente en la zona central de la mesa.

A continuación, se presentan cuáles de las imágenes pudieron ser aprovechadas para la obtención de los parámetros:

¿Se pudo utilizar la imagen para obtener los parámetros?				
Nº Imagen	Cám. 1	Cám. 2	Cám. 3	Cám. 4
1	NO	SÍ	SÍ	NO
2	NO	SÍ	SÍ	SÍ
3	NO	SÍ	SÍ	NO
4	SÍ	SÍ	NO	NO
5	SÍ	SÍ	NO	SÍ

¿Se pudo utilizar la imagen para obtener los parámetros?				
Nº Imagen	Cám. 1	Cám. 2	Cám. 3	Cám. 4
6	SÍ	SÍ	NO	SÍ
7	SÍ	NO	NO	SÍ
8	SÍ	SÍ	NO	SÍ
9	SÍ	NO	NO	SÍ
10	SÍ	NO	SÍ	SÍ
11	SÍ	NO	SÍ	SÍ
12	NO	NO	SÍ	SÍ
13	NO	SÍ	NO	NO
14	SÍ	NO	NO	SÍ

Tabla 6.1 Obtención de parámetros intrínsecos de las cámaras.

Viendo que ninguna posición del patrón se ha podido capturar correctamente por parte de las cuatro cámaras a la vez ya hace intuir que los parámetros intrínsecos presentarán grandes errores.

Se define el error de reproyección como un indicador que permite saber si la calibración representa un modelo fiel a la realidad. Se obtiene como la diferencia entre la posición proyectada en 2D de los puntos y la posición real detectada, sabiendo las dimensiones del tablero ajedrez. Cuanto menos error, mayor será la precisión de la proyección de los puntos de 3D a 2D.

$$Error\ reproyección = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \widehat{x_i})^2} \tag{6.1}$$

Siendo:

- N: Numero de esquinas encontradas.
- x_i: Coordenada detectada en la imagen.
- $\widehat{x_i}$: Coordenada reproyectada.

Cámara 1:

- Número de imágenes utilizadas: 9
- Error de reproyección promedio: 3.0165
- Matriz intrínseca: $\begin{bmatrix} 577.8 & 0 & 629.2 \\ 0 & 414.7 & 524.37 \\ 0 & 0 & 1 \end{bmatrix}$
- Coeficientes de distorsión: $[-0.254, 0.304, 0.0266, 0.0049, -0.238]$

Cámara 2:

- Número de imágenes utilizadas: 8
- Error de reproyección promedio: 3.3207
- Matriz intrínseca: $\begin{bmatrix} 1279.55 & 0 & 577.863 \\ 0 & 692.22 & 360.82 \\ 0 & 0 & 1 \end{bmatrix}$
- Coeficientes de distorsión: $[2.433, -27.682, 0.3199, -0.0675, 93.751]$

Cámara 3:

- Número de imágenes utilizadas: 6
- Error de reproyección promedio: 2.6759
- Matriz intrínseca: $\begin{bmatrix} 1144.22 & 0 & 663.87 \\ 0 & 2781.62 & 526.02 \\ 0 & 0 & 1 \end{bmatrix}$
- Coeficientes de distorsión: $[-0.627, 2.24, 0.003586, -0.03812, -8.84]$

Cámara 4:

- Número de imágenes utilizadas: 10
- Error de reproyección promedio: 3.4156
- Matriz intrínseca: $\begin{bmatrix} 610.05 & 0 & 626.19 \\ 0 & 443 & 510 \\ 0 & 0 & 1 \end{bmatrix}$
- Coeficientes de distorsión: $[-0.214, 0.115, 0.0322, -0.00533, -0.135]$

Se obtiene un error de reproyección en torno a los 3 píxeles, lo que ya de por sí es un valor elevado.

Ya con los valores de los parámetros intrínsecos calculados, se realizó la calibración estéreo para 2 pares de cámaras. Para el par 1-2 y otra para el par 3-4.

¿Se pudo usar el par de imágenes para la calibración estéreo?		
Nº Imagen	Par Cám 1-2	Par Cám 3-4
1	NO	NO
2	NO	SÍ
3	NO	NO
4	SÍ	NO
5	SÍ	NO
6	SÍ	NO

¿Se pudo usar el par de imágenes para la calibración estéreo?		
Nº Imagen	Par Cám 1-2	Par Cám 3-4
7	NO	NO
8	SÍ	NO
9	NO	NO
10	NO	SÍ
11	NO	SÍ
12	NO	SÍ
13	NO	NO
14	NO	NO

Tabla 6.2 Pares de imágenes detectados.

Calibración estéreo Cámara 1- Cámara 2:

- Error de reproyección: 33.8172
- Matriz de rotación: $\begin{bmatrix} 0.507 & 0.411 & 0.757 \\ -0.321 & -0.726 & 0.609 \\ 0.80 & -0.551 & -0.236 \end{bmatrix}$
- Vector de traslación: $[-37.72, -36.314,\ 349.726]$

Calibración estéreo Cámara 3- Cámara 4:

- Error de reproyección: 37.9229
- Matriz de rotación: $\begin{bmatrix} 0.365 & -0.032 & -0.93 \\ -0.882 & 0.309 & -0.356 \\ 0.297 & 0.951 & -0.084 \end{bmatrix}$
- Vector de traslación: $[377.89, 68.268,\ 138.334]$

Para el par 1-2 se obtiene un error de reproyección promedio de 33.82 píxeles, valor demasiado elevado debido a que solo se pudieron aprovechar cuatro pares de imágenes. Para el segundo par, el error es de 37.92 píxeles, aún mayor que para el par anterior. Estos valores no son aceptables para poder hacer una estimación de posición.

Estos errores eran debidos a que en la estación de trabajo tanto la luz del sol como las luces de la estancia provocaban reflejos que al tomar las fotos hacían que el plano ajedrez no se viera bien, haciendo que el programa no detecte el patrón correctamente e invalidando la calibración.

Con el fin de subsanar este problema se probó con otro tablero más grande, para el cual se tomaron 18 fotos simultáneamente para cada cámara, cuya distribución de cuadrados era de 10 columnas y 7 filas, con cuadrados de 9 cm de lado.

Figura 6.5 Sexta foto desde la cámara 1 segundo tablero.

Figura 6.6 Sexta foto desde la cámara 2 segundo tablero.

Figura 6.7 Sexta foto desde la cámara 3 segundo tablero.

Figura 6.8 Sexta foto desde la cámara 4 segundo tablero.

Como se observa en las imágenes en casi ninguna foto se podía ver el tablero al completo, al ser este más grande que el anterior y tener un acabado menos mate que hacía que el tablero se viera menos aún con el reflejo de la luz. Se podría probar a realizar la calibración en horas en las que hubiera menos luz, con las luces del laboratorio apagadas o añadir una iluminación difusa, pero en este proyecto se quiere dar una solución para calibrar las cámaras a cualquier hora del día en condiciones estándar de luminosidad.

Puesto que en casi ninguna foto se podía obtener una vista clara del tablero ajedrez, la calibración estéreo no era un método válido.

6.2. Homografía

Como se ha visto, la calibración estéreo es muy imprecisa para este entorno, por lo que, como el objetivo del proyecto es realizar una estimación de posición y guiado de un robot, y se conocen las dimensiones de la mesa y del robot se puede omitir el realizar una calibración tridimensional y realizarla bidimensional, teniendo en cuenta que el robot trabaja sobre plano. Por lo cual, se podría realizar por un lado la homografía de la mesa y, por otro lado, la homografía del robot y hacer una relación de coordenadas mediante otra homografía entre el plano mesa y el plano robot.

Para realizar una homografía se necesitan mínimo cuatro puntos. Para que la homografía sea más precisa es recomendable que los puntos que se usen se encuentren en cada una de las esquinas. La homografía se podría realizar solo con una cámara, pero dado que se poseen cuatro colocadas en ubicaciones diferentes en la estación de trabajo se puede hacer la homografía del plano para cada una de las cámaras y después, hacer una ponderación de los puntos obtenidos de forma que, los errores disminuirán considerablemente, además serviría como método para corregir el error introducido por la distorsión de la imagen. Para poder hacer la relación se deben conocer las dimensiones reales de la mesa por lo que esta debía ser medida.

Las dimensiones de la mesa son 212.5 x 162.5 centímetros además de 1 metro de alto, aunque la altura de la mesa va a dar igual al trabajar en 2D, por lo que se tomará esta altura como coordenada z=0.

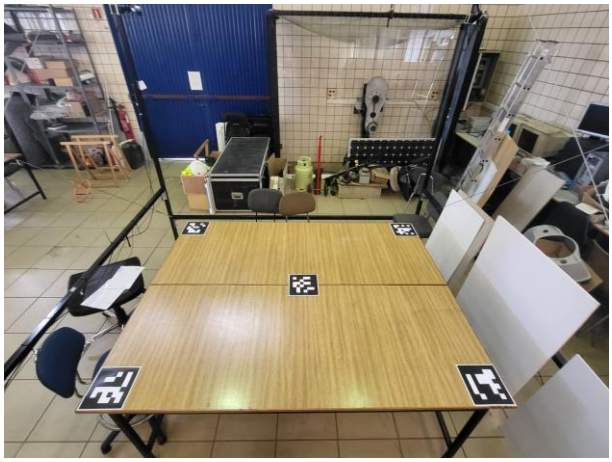

Figura 6.9 Fotografía de la mesa.

Para conocer la posición de las cuatro esquinas de la mesa y poder realizar la homografía se usarán cuatro marcadores ArUco con identificadores diferentes (1-4) colocados en cada una de estas. Por seguridad, no se usarán las esquinas reales de la mesa sino los centros de los ArUco de las esquinas para que no haya peligro de que el robot se pueda caer, por esto las dimensiones de la mesa que se consideran pasan a ser 1.901 x 1.403 metros. El origen de coordenadas se tomará en el centro del marcador (ID 2) situado en la esquina inferior izquierda vista desde la cámara 1. Siendo el eje X el largo de la mesa y el Y el ancho, ambos en sentido creciente hacia la esquina contigua de la propia mesa.

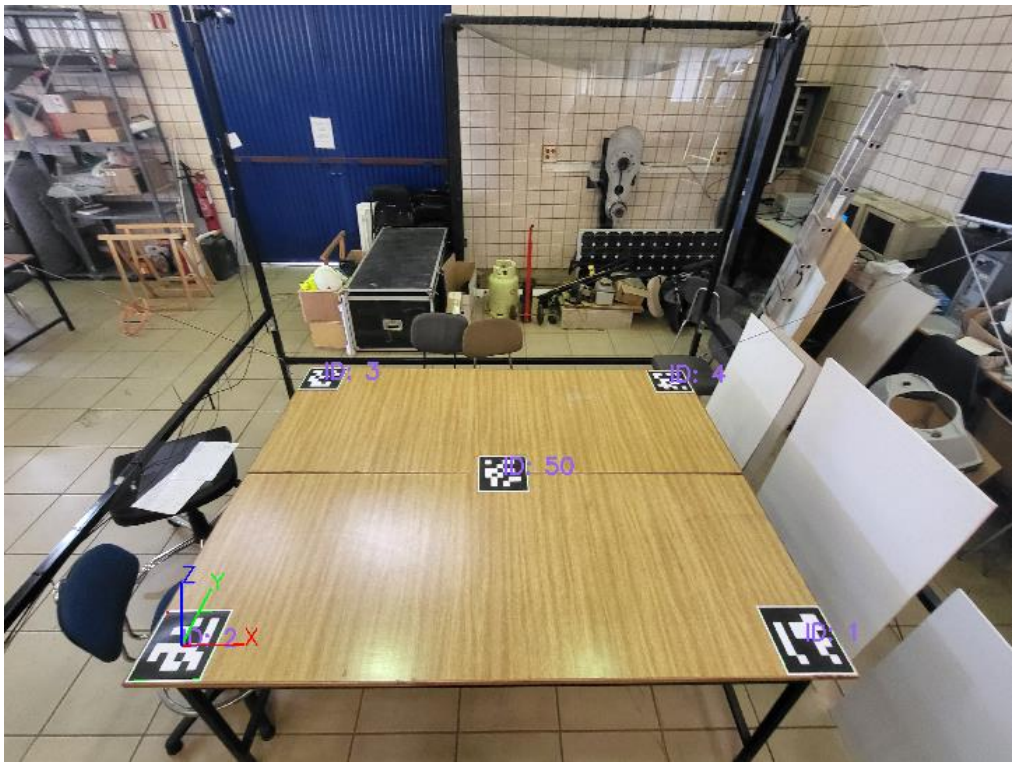

Figura 6.10 Origen de coordenadas y marcadores de la mesa.

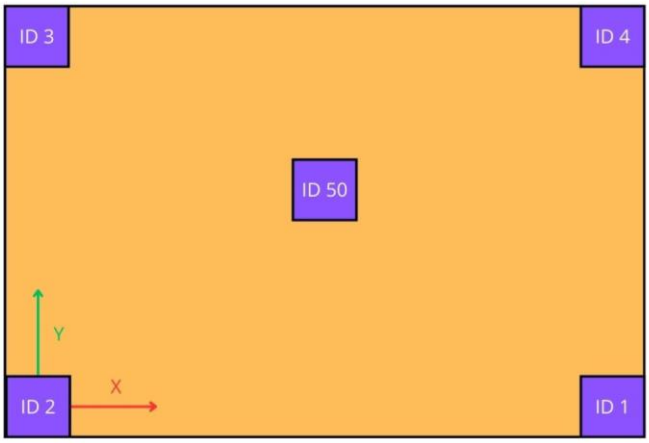

Figura 6.11 Distribución 2D de la mesa.

Ya conocidos los puntos esquina y sus coordenadas reales se debe calcular la matriz de homografía, resolviendo un sistema de 8 ecuaciones y 8 incógnitas (consideramos h_{33}=1). Si se hubieran introducido más puntos hubiera hecho falta usar métodos de optimización como mínimos cuadrados.

A continuación, se muestran los valores de las matrices de homografía para el plano de la mesa, calculadas en los experimentos del capítulo 10 de cada cámara:

Cámara 1:

$$\begin{bmatrix} -0.00553 & -0.00719 & 7.085 \\ -0.00483 & 0.00936 & -1.089 \\ -0.00001 & 0.00318 & 1 \end{bmatrix}$$

Cámara 2:

$$\begin{bmatrix} -0.00452 & 0.01046 & -1.370 \\ 0.00723 & 0.00882 & -8.548 \\ 0.00040 & 0.00361 & 1 \end{bmatrix}$$

Cámara 3:

$$\begin{bmatrix} 0.00591 & 0.00968 & -8.184 \\ 0.00637 & -0.00894 & 2.365 \\ -0.00034 & 0.00406 & 1 \end{bmatrix}$$

Cámara 4:

$$\begin{bmatrix} 0.00495 & -0.00757 & 0.439 \\ -0.00495 & -0.00669 & 7.349 \\ -0.00001 & 0.00300 & 1 \end{bmatrix}$$

Con la matriz de homografía de cada cámara calculada ya se puede hallar la posición de cualquier punto de la mesa que aparezca en la imagen.

Para la homografía del robot se debe tener en cuenta que su posición se calcula con un ArUco colocado en la cabeza del robot (ID 120). Este marcador está 20 cm por encima de la mesa, por lo que no se encuentra en el mismo plano que la mesa y la homografía calculada para esta no sirve. La parte positiva es que el marcador se encuentra en un plano paralelo a la mesa, pero el cual en vez de estar en z=0 está en z=0.2 m. La solución sería realizar otra vez la homografía a partir de las 4 esquinas, pero 20 cm más arriba, para ello una opción sería subir los marcadores ArUco a esta altura, pero habría que tener cuidado de que estuvieran a la misma altura que la cabeza del robot. Para no tener que subir los ArUco de las esquinas se llegó a la conclusión de que esto se podría hacer también colocando el robot con el ArUco en la cabeza en cada una de las esquinas y sacando un set de fotos cada vez. La cámara siempre va a estar en la misma posición, por lo cual como se conocen las coordenadas de las esquinas pudiéndose anotar el punto de la imagen donde se encuentra el centro del ArUco del robot en cada imagen, y una vez se tengan las coordenadas en la imagen correspondientes a cada una de las cuatro esquinas se calcula la matriz de homografía de este plano denominado como plano robot. Posteriormente, ya con el cálculo de la homografía del plano de la mesa y la del plano del robot habrá que realizar una relación entre los puntos de los dos planos.

Las matrices de homografía para el plano del robot son:

Cámara 1:

$$\begin{bmatrix} -0.00507 & -0.00704 & 6.332 \\ -0.00444 & 0.00860 & -0.975 \\ -0.00001 & 0.00336 & 1 \end{bmatrix}$$

Cámara 2:

$$\begin{bmatrix} -0.00408 & 0.01024 & -1.024 \\ 0.00661 & 0.00802 & -7.832 \\ 0.00043 & 0.00378 & 1 \end{bmatrix}$$

Cámara 3:

$$\begin{bmatrix} 0.00495 & 0.00883 & -6.771 \\ 0.00554 & -0.00885 & 1.574 \\ -0.00036 & 0.00384 & 1 \end{bmatrix}$$

Cámara 4:

$$\begin{bmatrix} 0.00442 & -0.00719 & 0.192 \\ -0.00440 & -0.00693 & 6.185 \\ -0.00003 & 0.00304 & 1 \end{bmatrix}$$

Para pasar las coordenadas del plano robot al plano mesa se necesitan algo que tengan en común, y es que en 2D las coordenadas de las esquinas usadas para ambas homografías tienen un homólogo para cada una de las 4 esquinas. Pues bien, la solución sería hacer una tercera homografía entre el plano de la mesa y el plano del robot, al tener 4 puntos con su correspondiente en los dos planos. Los puntos del plano robot se moverán 0.2 metros

en la coordenada Y para que no quede una matriz singular.

La matriz de homografía entre el plano de la mesa y el plano del robot calculada es:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -0.2 \\ 0 & 0 & 1 \end{bmatrix}$$

6.3. Comparación de los métodos de calibración

Tras analizar los métodos anteriores. Se ha visto que la calibración estéreo permite realizar una estimación tridimensional de la posición con un par de cámaras calculando los parámetros intrínsecos y extrínsecos, por el contrario, con la homografía solo se puede realizar la estimación bidimensional.

La calibración estéreo necesita una relación de puntos entre las dos imágenes, por lo que es común usar el plano ajedrez, del cual se precisa conocer sus dimensiones. Para la calibración estéreo se deben tomar bastantes pares de fotos con ambas cámaras colocando el plano ajedrez en diferentes posiciones y orientaciones.

La homografía precisa de conocer un mínimo de cuatro puntos, que para este caso han sido los marcadores ArUco de las esquinas de la mesa, además de que solo se puede operar en un plano al ser una estimación 2D. Por lo que para calcular la altura del robot hace falta crear otro plano y conocer la distancia al plano de la mesa, mientras que en la calibración estéreo al ser 3D la altura del robot se calcula directamente.

La calibración estéreo necesita de mucha precisión en el cálculo de los parámetros al tener que trabajar en un mundo tridimensional, por lo cual es muy posible que aparezcan errores, además una desventaja que impide la exactitud de los valores es que, al tomar las fotos, se producen reflejos en la mesa que a su vez hacen que el plano ajedrez no se vea bien en la imagen y estas fotos tengan que ser descartadas. Por lo cual, las matrices de rotación y el vector de traslación introducen errores. Por otro lado, si se usa la homografía, aunque sea un proceso más simplificado y que no estime la profundidad, sí da resultados más precisos. Pero presenta las desventajas de que se tendrá que recalibrar si se desplaza la mesa, además de que tiene que ser completamente lisa y plana la superficie.

Para este proyecto se ha optado por elegir la homografía. Dado que en los anteriores proyectos se optó por la calibración estéreo se quiere demostrar que también se pueden realizar otras técnicas como en este caso la homografía para la calibración. Otra de las razones es que en las esquinas de la mesa no suelen aparecer reflejos, además los marcadores ArUco se detectan bastante mejor que el tablero ajedrez, aunque también puedan aparecer los reflejos. Hay que añadir que no se quiere tener que apagar las luces del laboratorio o tener que incluir iluminación difusa para realizar la calibración, cosa que sería necesaria para la realización de la calibración estéreo al no poderse ver el tablero ajedrez correctamente. Por último, hay que decir que al ser la mesa lisa y el robot posee siempre la misma altura no es necesario realizar una estimación de profundidad, por lo que no hace falta trabajar en 3D, esto ahorra muchos cálculos.

7 CAPTURA DE IMÁGENES

7.1. Imágenes necesarias

Como se ha explicado en el capítulo 6.3, el método elegido para la calibración es la homografía, pero para poder realizarla, hace falta obtener primero una serie de imágenes, además de poder tomar las imágenes en tiempo real para el movimiento del robot:

1. **Imágenes para la homografía de la mesa.**

Será necesario una imagen de cada cámara en donde solo aparezca la mesa y los marcadores ArUco de las esquinas. Puede haber otros marcadores, pero en estas fotos solo importan los de las esquinas. En total debe haber cuatro imágenes, una captura desde cada cámara.

2. **Imágenes para la homografía del robot.**

Para la homografía del robot hacen falta más imágenes que para la homografía de la mesa. Se requerirá una foto de cada cámara en donde el robot con su marcador se encuentre sobre cada uno de los marcadores esquina, es decir, cuando el robot esté sobre un marcador ArUco de la esquina cada una de las 4 cámaras tomará una foto de manera simultánea. Esto hace un total de 16 imágenes.

3. **Detección de nuevo ArUco.**

Este modo de captura es parecido al de la homografía de la mesa, pero en vez de servir para la homografía, es para conocer la posición de nuevos marcadores ArUco que estén distribuidos por la mesa y poder hacer que el robot se dirija a ellos.

4. **Captura de imágenes en tiempo real.**

Por último, se necesitará un modo de captura en tiempo real para conocer la posición del robot en todo momento. Este modo deberá estar ejecutándose a la misma vez que el programa de tratamiento de imágenes, tomando imágenes simultáneamente de las 4 cámaras, almacenarlas y borrar las antiguas.

7.2. Sincronización de cámaras

A partir de la experimentación que se realizó con estas cámaras en proyectos anteriores se pudieron aprovechar para poder realizar la sincronización en este proyecto.

Para la cámara 1, la cual será designada como cámara maestra, se usará la configuración que se puede observar en el ANEXO 1. A continuación, se explicarán las principales características de manera que sea entendible:

Control del frame rate

- El control del frame rate se hará de forma manual.
- La frecuencia de adquisición es de 60 fps.
- El tiempo de exposición se establece en 15 ms.
- La exposición se alinea de manera síncrona.

Ganancia y nivel negro

- La ganancia analógica es de 3.2 y la digital de 1.
- El nivel de negro se coloca en 1.

Disparo

- El trigger estará deshabilitado para todos los modos, se usará un modelo de adquisición continua.

Líneas

- No se invierte la polaridad.

Contadores y temporizadores

- El contador estará desactivado.
- El temporizador permanecerá apagado.

LUT y corrección gamma

- La Look-Up Table estará deshabilitada.
- La corrección gamma será de 0.001.

Chunk mode

- Se incluyen metadatos en los fotogramas devueltos.

Configuración de red

- El protocolo PTP será automático.
- El ancho de banda de transmisión está limitado.
- El tamaño de paquete es de 9000.
- El intervalo entre paquetes es de 7.
- El tiempo de espera entre paquetes está asignado en 45000 ciclos.
- La dirección multicast es 239.10.10.10 por si hiciera falta.

Otros

- No se aplicarán múltiples regiones de interés.
- No habrá binning.
- El turbo transfer no se usará.

Para las cámaras 2,3 y 4 que serán usadas como "esclavas" los principales cambios respecto a la maestra son:

- La sincronización es externa.
- El trigger se activa para que espere un evento procedente del maestro.
- El siguiente disparo puede solaparse con el actual.

- Se introduce un retraso de exposición de 4 µs respecto al maestro.
- Hay un temporizador interno que se reinicia cada 17000000 ciclos.
- La frecuencia final estimada debería ser 58 Hz.

Como se ha visto en los datos técnicos la cámara 1 será la que marque el paso a las demás (método maestro-esclavo), de manera que esta será la que diga el momento de la captura, las otras 3 cámaras estarán a la espera. Esto se basa en el protocolo PTP (Precisión Time Protocol) el cual permite alinear los relojes de todas las cámaras con precisión de microsegundos.

Para el resto de las cámaras esclavas (2, 3 y 4) se utilizará la configuración detallada en el ANEXO 2. Entre los principales cambios respecto al programa para la cámara maestra:

- Se elimina la tasa de adquisición, al estar controlada por el maestro.
- Se introduce un retraso de 4 microsegundos con respecto al maestro.
- Se activa el trigger de manera que solo se produzca la captura cuando reciba la señal del maestro.

7.3. Modos de captura de imágenes

Como se explicó en el capítulo 7.1, se necesitan 4 modos. A continuación, se explicará cómo se ha implementado cada uno.

7.3.1 Modo 1: Imágenes para la homografía de la mesa

En este modo se toman 20 fotos con cada una de las cuatro cámaras. Realmente con una fotografía sería suficiente, pero es conveniente tomar más dado que, las imágenes para la homografía de la mesa son claves para el proyecto.

En las imágenes de cada cámara se deben poder ver los cuatro marcadores ArUco que se encuentran en las esquinas y otro en el centro de la mesa de manera que puedan ser analizados posteriormente.

7.3.2 Modo 2: Imágenes para la homografía del plano robot

Para este modo hay que colocar el robot con el marcador en la cabeza sobre cada marcador situado en cada esquina de la mesa en orden, empezando por el ArUco con ID1, y siguiendo secuencialmente con el ID2, ID3 e ID4.

Cuando el robot se encuentre en posición se debe realizar la captura de forma manual, cada cámara toma una fotografía de la escena, así con cada uno de los ArUco hasta haber pasado por los cuatro.

Se debe tener en cuenta que el robot y su marcador de cabeza deben estar alineados con el marcador de la mesa y encontrarse sobre la esquina adecuada para que la imagen sea aprovechable.

7.3.3 Modo 3: Detección de nuevos marcadores

La finalidad de este modo es poder conocer puntos específicos de la mesa gracias a nuevos marcadores ArUco repartidos por esta, no pueden tener un ID de los ya usados para la calibración. Estos nuevos marcadores se pueden situar en cualquier lugar de la mesa y una vez estén todos los requeridos se realiza el mismo proceso que en el modo 1 tomando 20 fotos por cada cámara.

7.3.4 Modo 4: Captura en tiempo real

Este modo es el que obtendrá las imágenes necesarias que serán usadas para el control del robot con el programa posterior.

Por un lado, se muestra una previsualización de una de las 4 cámaras pudiéndose elegir cual se quiere ver. Por otro lado, el programa tomará capturas en bucle continuo cada 50 ms, sobrescribiendo las imágenes.

Para evitar que el programa de tratamiento de imágenes, que será explicado posteriormente, acceda a las fotografías mientras se están escribiendo se hará uso de un triple búffer, de forma que se almacenan 3 capturas de cada cámara en una carpeta y se anota cual es la última, de manera continua se van sobrescribiendo las 3 imágenes siguiendo un ciclo. En un archivo de texto se apunta cual es la última imagen para que el programa de tratamiento de imágenes pueda saberlo.

7.4. Programa de captura de imágenes

En este apartado se abordará la explicación del programa final realizado para la obtención de las imágenes. El código completo se puede encontrar en el ANEXO 3. Este programa es una evolución del utilizado en el TFG *"Sistema multicámara para seguimiento a tiempo real de objetivos en movimiento. Adaptación a nuevo hardware y estimación tridimensional"* [34] y utiliza el lenguaje de programación C++.

7.4.1 Inicio del programa

Al compilar el código:

1. Se crean 4 directorios si estos no existiesen ya (uno para cada modo).
2. Se enumeran las interfaces de red Ethernet, asignándoles un nombre y mostrando su dirección IP. Dependiendo de la cantidad de direcciones mostradas se deberá pulsar el número asignado a esta (*1,2,3,* ...) y se almacena la elección, a continuación, se asocia el socket a esta dirección IP. De esta forma todos los mensajes que envíe el programa saldrán hacia la subred a la que están conectadas las cámaras. Se deberá seleccionar la dirección IP correspondiente con el switch.
3. Se abre el sistema Sapera.
4. Cada una de las cámaras es identificada por su número de serie, de manera que se le asocia un índice entre 1 y 4.

   Cámara 1: S1186692

   Cámara 2: S1177338

   Cámara 3: S1186684

   Cámara 4: S1186689

   Se crea el objeto para la adquisición de imágenes de cada cámara y se le asigna la configuración de cada una (serían los archivos del ANEXO 1 Y 2). De esta manera se realiza la sincronización mediante un protocolo PTP en donde la cámara 1 es la maestra y las otras 3 esclavas como bien se explicó en el capítulo 7.2.

   Se crea un buffer para almacenar las imágenes y un transfer para la transferencia del objeto de adquisición de imágenes hasta el buffer.
5. Se llega al menú de selección de modo.

7.4.2 Selección de modo

En la pantalla aparecerán cuatro opciones y dependiendo de la tecla que se seleccione entre 1 y 4 se activará uno de los modos.

Modo 1: Captura de imágenes para la homografía del plano mesa.

1. Se asigna la carpeta correspondiente para el almacenamiento de las imágenes. Dentro de esta carpeta habrá 4 subcarpetas, una para cada cámara (camara1, camara2, camara3, camara4).
2. Se activa el trigger para la captura.
3. Se esperan 2 segundos por seguridad.
4. Se inicia la adquisición de imágenes.
5. Se captura el número asignado de imágenes (N=20) y se almacena con un nombre "camaraX-fotoS.bmp." (Siendo X el número de cámara y S el número de imagen capturada).
6. Se termina el proceso y se puede elegir entre tomar otras N más imágenes pulsando *'ESPACIO'* o finalizar pulsando *"q"*.

Modo 2: Captura de imágenes para la homografía del plano robot.

1. Se asigna la carpeta correspondiente para el almacenamiento de las imágenes. Dentro de esta carpeta habrá 4 subcarpetas, una para cada cámara (camara1, camara2, camara3, camara4).
2. Se activa el trigger para la captura.
3. Se esperan 2 segundos por seguridad.
4. Se inicia la adquisición de imágenes.
5. Se define un vector con 4 IDs correspondientes a la posición del robot sobre el marcador (ID1, ID2, ID3, ID4).
6. Se imprime por pantalla un mensaje de que el robot esté en la posición sobre el marcador correspondiente y una vez se encuentre en posición se pulse *'ESPACIO'*.
7. Al pulsar *'ESPACIO'* la imagen se guarda como "camaraX-IDZ.bmp" (siendo X el número de cámara y Z el número de ID en el que se encuentra).
8. Se repite el proceso para cada una de las 4 posiciones.
9. Si la captura no sale bien se puede interrumpir pulsando *'q'*, en caso contrario termina cuando se realizan las capturas en las 4 posiciones.
10. Tras terminar se da la opción de repetir las capturas o salir.

Modo 3: Detectar nuevos marcadores ArUco.

1. Se asigna la carpeta correspondiente para el almacenamiento de las imágenes. Dentro de esta carpeta habrá 4 subcarpetas, una para cada cámara (camara1, camara2, camara3, camara4).
2. Se activa el trigger para la captura.
3. Se inicia la adquisición de imágenes.
4. Se captura el número asignado de imágenes (N=20) y se almacena con un nombre "camaraX-fotoS.bmp." (Siendo X el número de cámara y S el número de imagen capturada).
5. Se termina el proceso y se puede elegir entre tomar otras N más imágenes pulsando *'ESPACIO'* o finalizar pulsando *'q'*.

Modo 4: Captura de imágenes en tiempo real.

1. Se asigna la carpeta correspondiente para el almacenamiento de las imágenes. Dentro de esta carpeta habrá 4 subcarpetas, una para cada cámara (camara1, camara2, camara3, camara4).
2. Se activa el trigger para la captura.
3. Se ejecuta la función de llamada de cada cámara en orden. Para cada cámara, se captura primero la imagen almacenándose en un archivo temporal, a continuación, se renombra con el nombre de la imagen más antigua para evitar conflictos. Puesto que se usa un triple búffer se busca la imagen más antigua de las 3 existentes y se sobrescribe. En un archivo de texto se anota cual es la imagen más antigua.
4. Se selecciona la cámara desde la cual se quiere hacer una previsualización. Una pantalla mostrará lo que ve esta cámara en tiempo real.
5. Se crea un bucle infinito de captura, de manera que cada cámara toma una imagen cada 50 ms.
6. Para finalizar se pulsa la tecla *'q'*.

7.4.3 Finalización del programa

1. Se cierra la transferencia de datos de cada cámara.
2. Se destruye cada búffer.
3. Se cierra la adquisición de imágenes.
4. Se cierra el entorno Sapera.
5. Por último, se cierran los sockets.

8 TRATAMIENTO DE IMÁGENES Y CONTROL DEL ROBOT

Este capítulo se centrará en explicar el programa desarrollado en Python que, a partir de las imágenes obtenidas en el capítulo anterior, obtiene la información necesaria para conocer la posición del robot y poder hacer que vaya al lugar deseado.

8.1. Necesidades del programa.

Una vez que se ha completado el programa para la captura de las imágenes, se debe realizar un programa que, por un lado, haga el tratamiento de estas imágenes para conocer la localización en la mesa en todo momento y poder así, controlar el movimiento del robot desde el mismo programa.

La idea sería hacer un código en Python, dadas su gran cantidad de librerías.

Por un lado, hace falta una interfaz intuitiva para el usuario, de manera que, aunque este no haya usado el programa anteriormente con un mínimo de conocimientos sea capaz de realizar las operaciones con el robot. Por otro lado, se debe realizar la calibración mediante la homografía explicada en el capítulo 6.2. Para ello el programa primero deberá leer las imágenes capturadas. Una vez leídas detectar los ArUcos que aparecen en las imágenes tomadas por el modo 1 de captura de imágenes explicado en el capítulo 7.3.1. A continuación, realizar el mismo proceso para las imágenes del modo 2 y con ello conocer la posición del ArUco del robot para realizar la homografía de su plano. Una vez se conocieran las posiciones de estos ArUco, se obtendrían tanto las matrices de homografía del plano mesa como la del plano robot ya que se saben las dimensiones de mesa y robot. Se realizaría la transformación en todo momento de las coordenadas del plano robot al plano mesa. El programa debería tener 2 modos, uno manual en el que el robot se controlaría por teclado de forma que avance, retroceda o gire dependiendo de la tecla presionada. El otro modo sería el automático en el cual, el robot podrá ir a un punto de la mesa que se le indique o a uno de los ArUcos repartidos a lo largo de la mesa, estos ArUcos pueden ser los usados para la calibración o bien otros que detectaron con el modo 3 del programa de captura de imágenes. Para que el modo automático funcione sería necesario que el modo 4 del programa de captura de imágenes esté funcionando y sacando las fotos de cada cámara en tiempo real para conocer la posición del robot en todo momento y poder llegar a la posición destino. También hay que tener en cuenta que se tienen que ajustar las imágenes del triple búfer propuesto en el capítulo 7.3.4. Sería interesante dibujar la trayectoria recorrida por el robot tanto en modo manual como automático, para ello en el modo manual también tendría que estar en funcionamiento la adquisición de imágenes en tiempo real y calcular la posición del robot.

8.2. Carga de imágenes y detección de ArUco.

Lo primero que realiza el programa es la lectura de todas las imágenes que no se actualizan en tiempo real, es decir las obtenidas en los modos 1, 2 y 3 del programa de captura de imágenes en C++. Por lo que se cargarán las imágenes necesarias tanto para la homografía del plano mesa como del plano del robot además de las que contienen los nuevos ArUco detectados. Estas imágenes se cargan desde una carpeta a la cual tengan acceso tanto en programa ejecutado en Windows de C++ como este programa de Python ejecutado en WSL.

Se detectan los ArUcos de todas las imágenes para conocer los puntos a los cuales les corresponde una posición en el mundo real, tomando como punto el centro de cada ArUco.

Para saber las coordenadas y distancias en el mundo real de los marcadores esquina se deberán medir. Sabiendo que:

- Los identificadores ArUco estarán repartidos de forma que si se ve la mesa desde la estación donde se encuentra el ordenador. Considerando que el borde más cercano se considera el inferior y el más alejado el superior:

- El ID 1 se encuentra en la esquina inferior derecha.
- El ID 2 se encuentra en la esquina inferior izquierda.
- El ID 3 se encuentra en la esquina superior izquierda.
- El ID 4 se encuentra en la esquina superior derecha.

Todo esto visto desde la zona donde se encuentra el ordenador o visto desde la cámara 1.

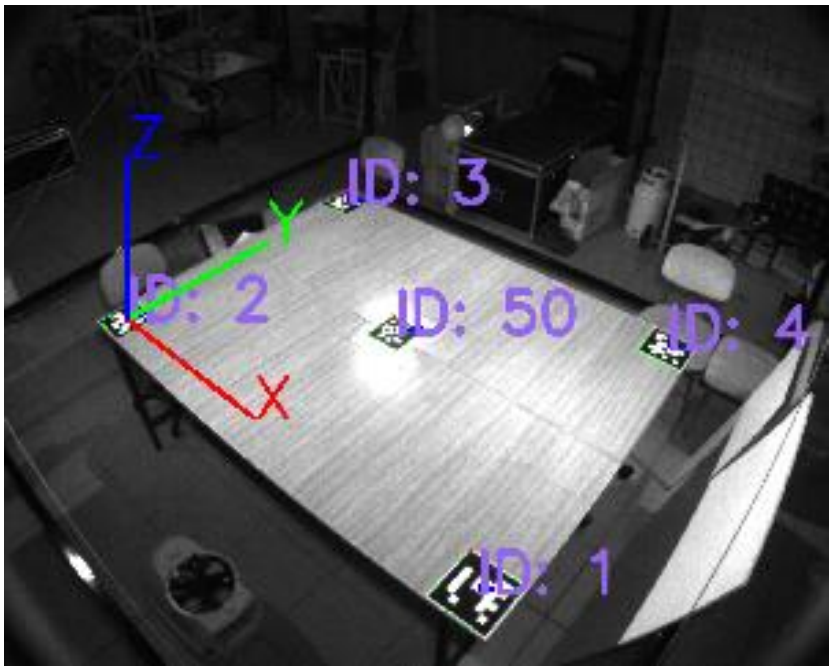

Figura 8.1 Marcadores y ejes vistos desde la cámara 1.

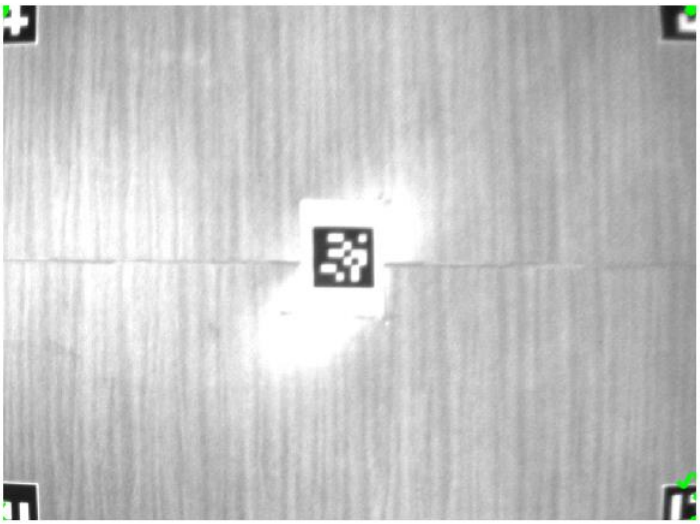

Figura 8.2 Vista de la mesa tras realizar la homografía desde la cámara 1.

La mesa mide 212.5 x 162.5 centímetros, pero como se explicó anteriormente se considerará como esquina el centro de los marcadores ArUco por seguridad. Por lo que las distancias quedan:

- 1.901 metros entre los marcadores 1 y 2 y entre el 3 y el 4.

- 1.403 metros entre los marcadores 2 y 3 y entre el 1 y el 4.

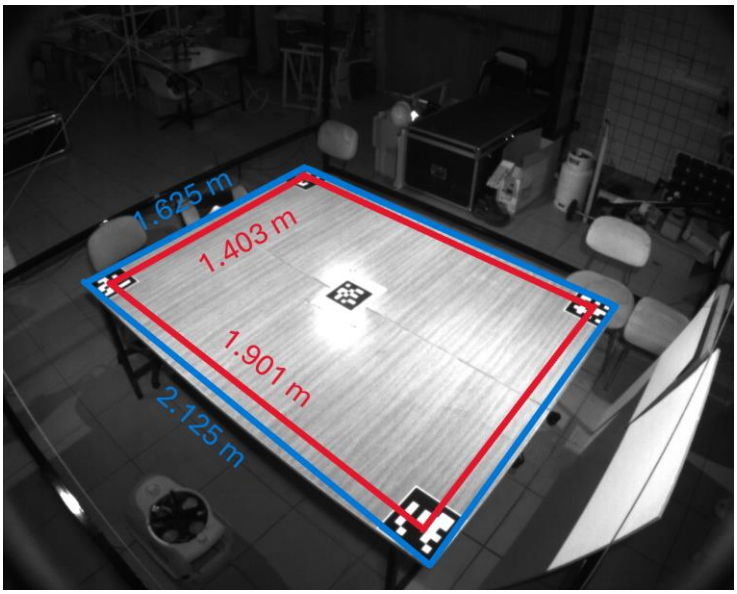

Figura 8.3 Dimensiones de la mesa.

Como se va a trabajar en 2D, solo se utilizan las coordenadas en el eje X e Y. El origen de coordenadas se considera el centro del ID 2 (0,0), de manera que el eje X coincide con la línea imaginaria que une el ID 2 con el 1 siendo creciente desde 2 hasta 1. El eje Y por su parte coincide con la línea imaginaria que une el ID 2 con el 3 siendo creciente desde 2 hasta 3.

La altura del robot es de 20 cm, se toma como referencia de este el ArUco posicionado en la cabeza del robot cuyo ID es el 120.

Además de los ArUco 1,2,3,4 en las esquinas de la mesa, habrá otro marcador ID 50 en el centro de la mesa cuya posición se calculará posteriormente. Su utilidad es servir de posición "home" del robot, se explicará con más detalle más adelante.

8.3. Cálculo de homografías

Ya con los puntos definidos tanto en las imágenes como en la realidad, los relacionamos mediante homografía. Primero para obtener las matrices de homografía de cada cámara para el plano mesa y después para la homografía del plano robot.

Ambas homografías se calculan de forma parecida, con la diferencia de que para el plano mesa solo hace falta una imagen por cámara y para el plano robot 4 imágenes por cámara al haber necesitado que el robot estuviera en cada una de las esquinas en cada imagen para conocer su posición.

Para saber con qué coordenada calculada sobre el plano robot se corresponde en el plano mesa hay que relacionar ambas homografías. Para ello se realiza una tercera homografía que relaciona los puntos de ambos planos como bien se explicó en el capítulo 6.2.

Ya con la tercera homografía hecha se podrá pasar las posiciones del marcador del robot al plano mesa.

Como es normal debido a la perspectiva de cada cámara se observa distorsión al ver la mesa. Pero como para obtener la homografía se hace un promedio la posición obtenida del objetivo para cada cámara deja un error bastante pequeño así que se considera que no es necesario hacer una corrección de la distorsión.

8.4. Inicialización de ROS

Se crea un nodo específico para poder utilizar ROS para el paso de mensajes con el robot.

Para confirmar la correcta conexión con el robot se verifica que se reciben mensajes del tópico /odom durante 5 segundos.

Se crea un publicador */cmd_vel* para poder enviar por mensaje la velocidad lineal y de rotación deseada para el robot.

Se subscribe al *topic /odom* para conocer la odometría del robot (posición y orientación).

Se devuelve el publicador.

Se supone que se ha realizado previamente la configuración del robot detallada en el capítulo 5.6.

8.5. Lectura de imágenes en tiempo real

El programa busca en la carpeta asignada para leer la imagen en tiempo real, en esta carpeta habrá 3 imágenes como ya se explicó en el capítulo 7.3.4. Se ordenarán las imágenes por fecha de modificación y se elegirá la segunda de las tres. Se lee la imagen y se busca la posición del marcador ID 120 situado en la cabeza del robot. Si no se pudiera leer la imagen porque se está sobrescribiendo, el programa leerá una de las otras dos que se encuentran en la carpeta y se anotará que ha habido una imagen defectuosa. Al final del recorrido del robot se muestra el acumulado del total de imágenes defectuosas.

8.6. Control PID para el movimiento del robot

Para controlar la respuesta del movimiento del robot, se hace un control PID en bucle cerrado (con realimentación) tanto para la velocidad lineal como para la angular.

Primero se calcula en valor absoluto tanto la distancia como el error en el ángulo.

- **Para el PID lineal:**

  El error introducido es en distancia hasta el objetivo, de manera que se utiliza un único bucle de control PID que regula la velocidad de avance dependiendo de la distancia restante hasta el objetivo, actualizándose con la tasa de frecuencia del robot.

  El bucle solo trabaja como entrada con el módulo de la distancia al objetivo, es decir, es independiente a los ejes, puesto que de la dirección se encarga el control PID angular.

  Tras pasar el error por el PID se calcula la velocidad y se aplica al robot. A continuación, se realimenta el ciclo con la nueva distancia del robot.

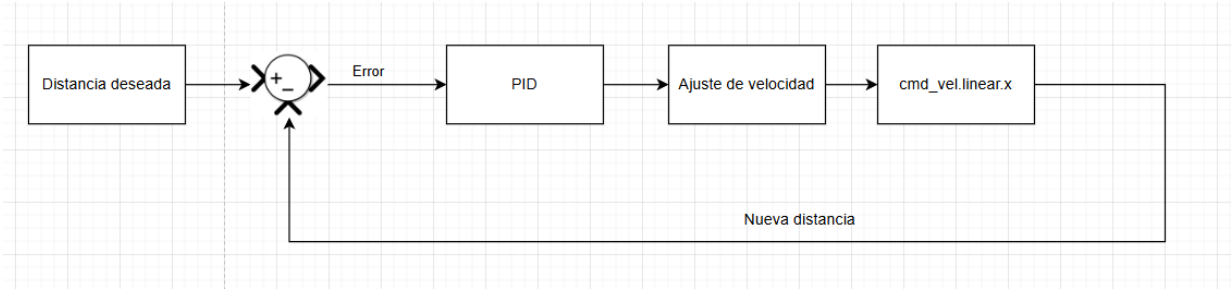

Figura 8.4 Diagrama del PID lineal.

- **Para el PID angular:**

Para este control la entrada es el error de orientación. El funcionamiento es parecido al del PID lineal, solo que el ajuste lo hace con el error del ángulo en vez del error en distancia.

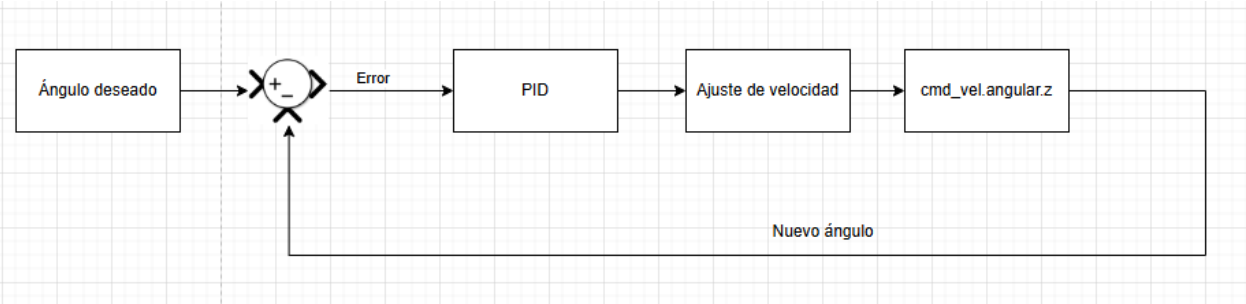

Figura 8.5 Diagrama PID angular.

Además de la posición del robot obtenida gracias a las cámaras, para saber el error angular también hace falta saber la dirección de avance del robot, es decir cuál es su parte delantera. Para ello se apunta en un archivo ".csv" pequeños movimientos del robot y se ve como se modifica la posición. Con ello se consigue hacer una estimación de la dirección del robot. Estos parámetros se cargan al iniciar el programa y se van actualizando con los diferentes experimentos del robot.

Si hubiera algún cambio en estos parámetros o no fueran acordes a la realidad. Como se ha dicho el robot haría pequeños movimientos para recalibrarlos y actualizarlos.

Los valores adecuados de los controladores PID se estudiarán en el capítulo 10.4.

Ambos controladores tendrán como límite de salida la velocidad impuesta como máxima del robot, tanto angular como lineal.

Cuando el PID satura se puede producir una sobreacumulación de la acción integral (windup) que provoca irregularidades cuando el controlador sale de la zona saturada. Para ello en el código se introduce un límite del término integral conocido como anti-windup, este hará que el error integral no se incremente más a partir de un cierto valor.

También se podrán meter valores límites mínimos y máximos de la salida del PID.

Por último, se podrá introducir una banda muerta (deadband), por debajo de la cual el error es 0.

8.7. Saturación y filtrado de velocidad

La velocidad máxima del robot lineal según el fabricante es de 0.22 m/s y por otra parte la angular se sitúa en 2.84 rad/s. Pero por experiencia (capítulo 10.2), se descubrió que, en condiciones reales de funcionamiento estas velocidades máximas se situaban en 0.18 m/s para el movimiento lineal y 2.3 rad/s para el movimiento angular. También en el capítulo 10 se verá que si se producen simultáneamente el movimiento lineal y el angular la velocidad máxima de ambos disminuye.

8.8. Modos de funcionamiento

El programa una vez haya comprobado la conexión con el robot y haya realizado la calibración mediante las homografías. Se podrá elegir entre el modo automático y el modo manual.

8.8.1 Modo manual

Este modo permite controlar el movimiento del robot solamente pulsando botones de teclado.

El funcionamiento es el siguiente:

- Si se deja pulsada la tecla *'w'* el robot avanza linealmente hacia delante.
- Si se deja pulsada la tecla *'s'* el robot avanza linealmente hacia atrás.
- Si se deja pulsada la tecla *'d'* el robot gira hacia la derecha.
- Si se deja pulsada la tecla *'a'* el robot gira hacia la izquierda.
- Si se pulsa la tecla *'e'* la velocidad lineal aumenta 0.02 m/s hasta un límite marcado por la velocidad máxima lineal del robot indicada.
- Si se pulsa la tecla *'r'* la velocidad lineal disminuye 0.02 m/s siendo el mínimo 0.00 m/s.
- Si se pulsa la tecla *'t'* la velocidad angular aumenta 0.05 rad/s hasta el máximo de velocidad angular impuesto.
- Si se pulsa la tecla *'y'* la velocidad angular disminuye 0.05 rad/s siendo el mínimo 0.00 rad/s.
- Si no se pulsa nada el robot permanece en reposo.
- Se pueden hacer movimientos de avance y giro simultáneos pulsando 2 botones.
- Para salir del modo se debe pulsar *'q'*.

Una vez terminado el modo se mostrará la trayectoria seguida por el robot durante el modo.

8.8.2 Modo automático

Este modo aprovecha el trabajo realizado anteriormente como son el cálculo de las homografías, la detección de los marcadores ArUco, el PID introducido, además de la comunicación con ROS. Permite que el robot se pueda mover independientemente a un punto de la mesa.

El punto objetivo que debe alcanzar el robot se puede introducir seleccionando uno de los marcadores previamente registrados o bien seleccionando un punto cualquiera de la mesa, el cual se puede elegir de dos maneras:

- Introduciendo manualmente las coordenadas, se podrá hacer escribiendo tanto la posición X e Y por teclado, teniendo en cuenta que estén dentro de los márgenes de la mesa.
- Haciendo clic en un plano 2D dibujado de la mesa. Si se hiciera clic fuera de la mesa o de los márgenes, se tomará el punto más cercano a este dentro de los márgenes delimitados para que no haya riesgo de caída del robot.

Una vez introducido el objetivo, comenzará el movimiento hacia él.

8.8.2.1 Movimiento

Como ya se dijo en el capítulo 8.5. Para conocer la orientación del robot, se aprovechan las imágenes, de manera que el robot realiza pequeños movimientos para calcular los incrementos tanto de posición lineal como angular. Tras varias iteraciones se podría decir que el robot "aprende" donde se encuentra colocado de manera que esta información se puede utilizar para la optimización de la trayectoria a seguir para alcanzar el punto objetivo. Estos datos se almacenan en un archivo para no tener que repetir este proceso en futuros usos, además se va actualizando con cada viaje hacia un objetivo del robot, para aumentar la precisión.

Si el robot se encuentra fuera de los márgenes externos de seguridad impuestos, el robot deberá dirigirse hacia

la posición *"home"*, esta posición se encuentra en el centro de la mesa marcado con el ArUco de ID 50. Al encontrarse el centro de la mesa, siempre habrá un momento en el que el robot vuelva a entrar dentro de los límites, una vez que esté dentro de estos recuperará el camino hacia el punto objetivo cuando entre dentro de los márgenes internos. Para que el robot no esté entrando y saliendo de los límites al estar en un valor muy límite se introduce una especie de histéresis que distingue entre márgenes externos e internos. Los márgenes externos son los que al rebasarse el robot deberá dirigirse a home a velocidad limitada, pero al entrar en los márgenes externos si sigue fuera de los internos el robot se seguirá dirigiendo a home hasta que entre dentro de los márgenes internos, aunque ya a velocidad normal, una vez que entre dentro de los interiores, recuperará el objetivo inicial y se dirigirá hacia él siempre y cuando no vuelva a rebasar un margen externo.

Para el movimiento se aplican los límites de velocidad.

Una vez llegado al objetivo el robot se detiene cuando su posición esté dentro de una tolerancia impuesta.

La repetición de cada ciclo se regula con la tasa *"rate"* de ROS en principio según datos del fabricante se pondrá en 30 Hz.

Si selecciona como objetivo uno de los marcadores de las esquinas, al ser peligroso que el robot los alcance se aplica un umbral especial para que se considere que ha llegado a él, aunque aún le falte un poco.

En cada iteración se aplica el PID tanto para la velocidad angular como para la lineal. Tomando en cuenta los errores de posición como se explicó en el capítulo 8.5.

8.8.2.2 Tolerancias

La tolerancia se aplica de forma que cuando el robot esté dentro del umbral impuesto se considere que ha llegado al objetivo.

Habrá tres tipos de tolerancias:

- Tolerancia para marcadores ArUco ya conocidos. Este umbral es más grande por lo general puesto que en este rango entran los marcadores situados en las esquinas y estas son posiciones relativamente peligrosas.
- Tolerancia para nuevos marcadores ArUco. Esta sería la tolerancia que se aplica para los nuevos marcadores detectados con las imágenes tomadas por el modo 3 del programa de captura de imágenes.
- Tolerancia para puntos introducidos. Es el umbral que se aplica para los puntos cuyas coordenadas han sido escritas o clicadas.

8.8.2.3 Fuera de márgenes

Si el robot está fuera del margen exterior su velocidad lineal y angular será reducida debido al peligro. Si el robot no consigue encontrar la posición home y se ve que se está alejando, dará un giro de 180º de manera que esto hace que la iteración de búsqueda de la posición home por lo general, converja.

8.9. Medidas de seguridad

Como el robot se mueve en una mesa que está 1 metro sobre el suelo se tienen que poner medidas que protejan al robot del riesgo de caída. Se imponen márgenes para que el robot si los superase (en modo automático) volviera dentro de estos, y si el punto objetivo estuviera fuera de estos márgenes se tomaría como meta el punto más cercano dentro de los márgenes.

8.10. Gráficas

Una vez se alcance el objetivo en el modo automático aparece una gráfica en la que se puede ver la evolución del error en distancia en la parte superior y del error del ángulo en la inferior.

Cuando se da por finalizado el modo automático o el modo manual se muestra primero una gráfica en 2D que muestra un dibujo de la mesa y la trayectoria seguida por el robot, para el modo automático también se muestran el punto inicial y los puntos objetivo, además si el punto objetivo fuera inalcanzable también se dibuja el punto que se ha tomado en su lugar.

Posteriormente, se muestra la trayectoria seguida vista desde la cámara 1, realmente la información es la misma que la de la gráfica anterior, pero proporciona una vista más espacial.

La tercera gráfica muestra velocidad lineal (m/s) y angular (rad/s) desarrollada, durante el movimiento. Una línea muestra la velocidad comandada y otra la real desarrollada por el robot.

8.11. Funcionamiento del programa

En este subapartado se hará una explicación conjunta de todo el código (ANEXO 4), es decir cómo funcionan conjuntamente todos los apartados anteriormente mencionados.

1. Al ejecutar el programa, este se conecta con ROS para tener una conexión fiable con el robot. Se comprueba que exista un nodo maestro verificando que se pueden recibir parámetros de odometría a través del ROS topic */odom*. Si la conexión es correcta el programa avanza, en caso contrario se cierra.
2. Se cargan los parámetros de configuración (si no existieran se crean y se toman unos ya definidos por defecto). Para cambiar un valor basta con escribir el número indicado por pantalla (entre paréntesis se indican los valores por defecto si se reinician):
    i. Parámetros PID
        i. Kp lineal (0.6)
        ii. Ki lineal (0.0)
        iii. Kd lineal (0.1)
        iv. Límite de integrador lineal (None)
        v. Output mínimo lineal (None)
        vi. Output máximo lineal (None)
        vii. Deadband lineal (0.0)
        viii. Kp angular (1.0)
        ix. Ki angular (0.0)
        x. Kp angular (0.2)
        xi. Límite de integrador angular (None)
        xii. Output mínimo angular (None)
        xiii. Output máximo angular (None)
        xiv. Deadband angular (0.0)
    ii. Velocidad lineal máxima (m/s) (0.15)
    iii. Velocidad angular máxima (rad/s) (1.0)
    iv. Velocidad lineal máxima fuera de los márgenes de seguridad (m/s) (0.05)
    v. Velocidad angular máxima fuera de los márgenes de seguridad (rad/s) (0.3)
    vi. Tolerancia de la distancia para alcanzar un marcador de los nuevos detectados (m) (0.05)

vii. Tolerancia de la distancia para alcanzar un marcador de los ya existentes (m) (0.2)
viii. Tolerancia de la distancia para alcanzar un punto objetivo (m) (0.05)
ix. Margen interno de seguridad (m) (0.15)
x. Margen externo de seguridad (m) (0.05)
xi. Restablecer valores por defecto
xii. Continuar con la ejecución

3. Una vez asignados los parámetros correspondientes, se crea un controlador PID para la velocidad lineal y otro para la velocidad angular con los parámetros correspondientes previamente establecido.
4. Como el programa de captura de imágenes es totalmente externo a este, el usuario deberá indicar de forma manual si este otro programa está en funcionamiento.
5. En este paso se realiza la calibración de las cámaras mediante homografías. Se cargan las imágenes con los ArUco necesarios para la homografía (4 ArUco en las 4 esquinas con ID de 1-4 y el ID 50 en el centro de la mesa que se usa como posición home). También se realiza la homografía para el denominado como plano robot que se encuentra en paralelo al plano mesa unos 20 cm por encima con sus correspondientes imágenes. Se obtienen la matriz de homografía entre los planos y las imágenes, también se hace otra homografía entre el plano mesa y el plano robot.
6. Los límites de la mesa vienen impuestos por las coordenadas reales de los ArUco introducidas y se le restan los márgenes para que el robot no pueda llegar a posiciones tan límites.
7. Tras esto el usuario podrá elegir entre el modo manual (el movimiento del robot se controla por teclado) o el modo automático (el robot se dirige a una coordenada alcanzable de la mesa de forma autónoma).
    i. Modo manual:
        i. Se utiliza la librería *Pygame* para poder mover el robot en tiempo real
        ii. Se puede ajustar la velocidad lineal y angular del robot manualmente dentro de un rango. Se realizan los movimientos requeridos por teclado.
        iii. Una vez se finalice el movimiento se mostrará primero los resultados, a continuación, la trayectoria seguida en un plano 2D, después la trayectoria vista desde la cámara 1 y finalmente las velocidades angulares y lineales registradas.
        iv. Finaliza el modo.
    ii. Modo automático:
        i. Se puede elegir entre ir a un marcador indicando si ID o a un punto introduciendo sus coordenadas o seleccionándolo en una gráfica con un clic.
        ii. Se comprueba que el robot se desplaza correctamente con el ajuste de su dirección.
        iii. Se calcula la distancia y el error angular entre la posición actual y el objetivo, además con los errores, los PID y los límites, se calculan las velocidades en cada iteración.
        iv. Durante el movimiento se tienen en cuenta los márgenes definidos, y si se superan los externos el robot se dirige a la posición home hasta entrar en el interno y recuperar la posición objetivo.
        v. Una vez se alcance el objetivo, se mostrarán los resultados del movimiento y se podrá ir a otra posición o finalizar.
        vi. Cuando se quiera finalizar, se mostrará primero las trayectorias seguidas en un plano 2D, después las trayectorias vistas desde la cámara 1 y finalmente las velocidades angulares y lineales registradas.
        vii. Finaliza el modo.
8. Una vez cerradas las gráficas el programa termina:
    i. Se ordena que el robot tenga velocidad 0 por seguridad.

ii. Liberación de recursos y cierre de ROS.

iii. Finalización del programa.

9 INTERFAZ

En este apartado se mostrará el funcionamiento de la interfaz tanto para el programa de captura de imágenes hecho en C++ como el del tratamiento de las imágenes y el control del robot realizado en Python. Cabe recordar que por lo general el programa de captura de imágenes se debe ejecutar previamente al otro.

9.1. Programa de captura de imágenes

1. Una vez se lanza el programa de captura de imágenes *ActionCmd.cpp* cuyo ejecutable se denomina *ActionCmd.exe*, lo primero que aparece es un texto introductorio.
2. Se nos dará a elegir una de las redes detectadas conectadas al ordenador escribiendo el número asignado. Aquí hay que buscar la dirección IP del switch al que van conectadas las cámaras. Para el caso del ejemplo de la Figura 9.1, sería la IP 192.168.1.2 que tiene el número 3 asignado por lo cual se debería teclear *'3'*, si hiciera falta se puede salir pulsando *'q'* de manera que el programa terminaría.

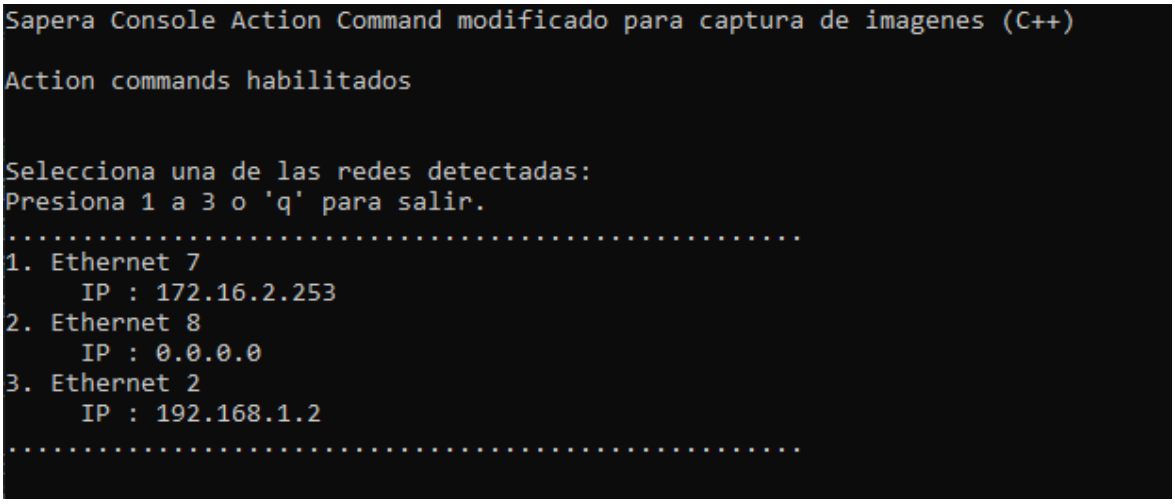

Figura 9.1 Selección de red en la interfaz.

3. Una vez escrito el número aparecerá un mensaje con la red correspondiente seleccionada y si todo es correcto se mostrará el menú de selección de modo.

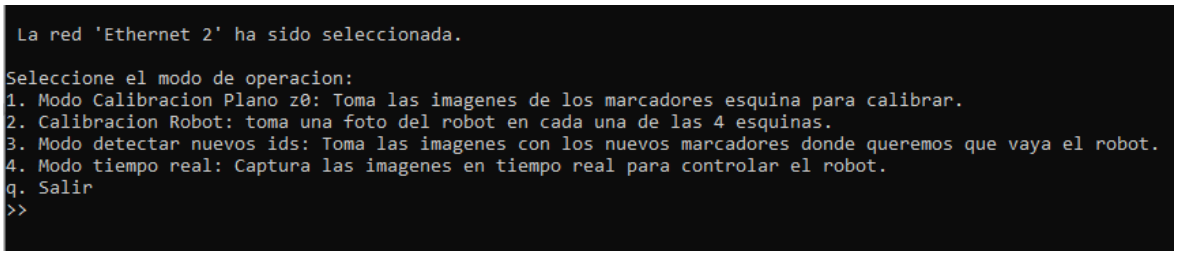

Figura 9.2 Selección de modo en la interfaz.

4. Como se muestra en la Figura 9.2 hay 4 modos para seleccionar del 1-4 y la opción de pulsar *'q'* para salir.
5. El primer modo al cual se accede pulsando *'1'* es la calibración del plano z0, se toman 20 fotos de cada una de las 4 cámaras. Tras acabar las capturas se da la opción de tomar otras 20 fotos o volver al menú de selección de modo tecleando *'q'*.

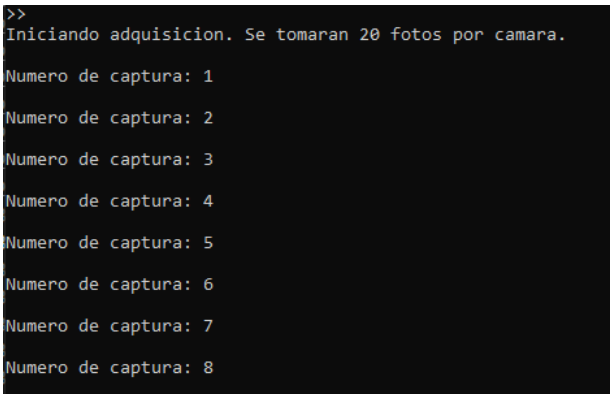

Figura 9.3 Inicio del modo 1 en la interfaz.

Figura 9.4 Fin del modo 1 en la interfaz.

6. El segundo modo es la toma de las fotos para la calibración del plano del ArUco que posee el robot en la cabeza, se activa tecleando *'2'*. Para este modo como hay que ir colocando el robot sobre los diferentes IDs aparecerá un mensaje acerca el ID sobre el que se debe colocar el robot e indica que se pulse *'ESPACIO'* cuando el robot esté en posición. En la Figura 9.5 se aprecia como es para la captura sobre el ID 1.

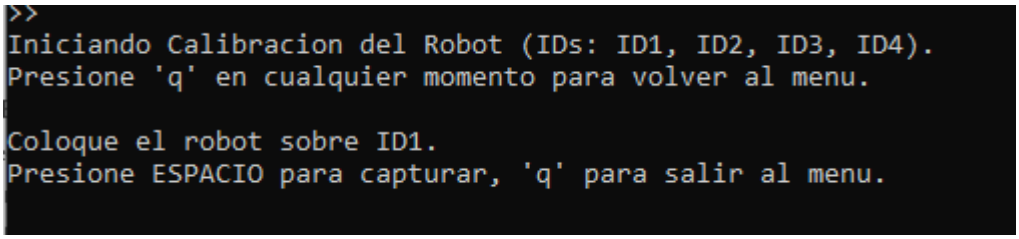

Figura 9.5 Modo 2 captura sobre ID1.

Se realiza el proceso para cada uno de los 4 IDs y cuando se termine se da la opción de repetir el proceso pulsando *'ESPACIO'* o salir del modo pulsando *'q'*.

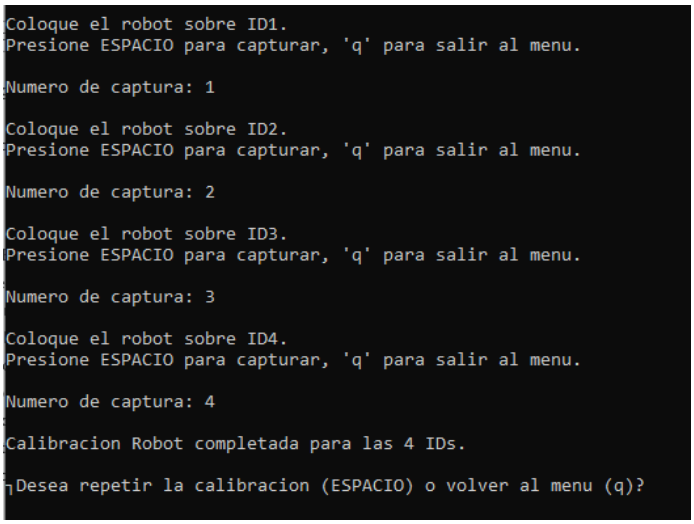

Figura 9.6 Fin del modo 2 en la interfaz.

7. Para el tercer modo el cual sirve para tomar capturas de los nuevos marcadores repartidos por la mesa, se accede tecleando *'3'*. Se toman 20 capturas de cada cámara y al final se da la opción de tomar otras 20 pulsando *'ESPACIO'* o terminar el proceso pulsando *'q'*.

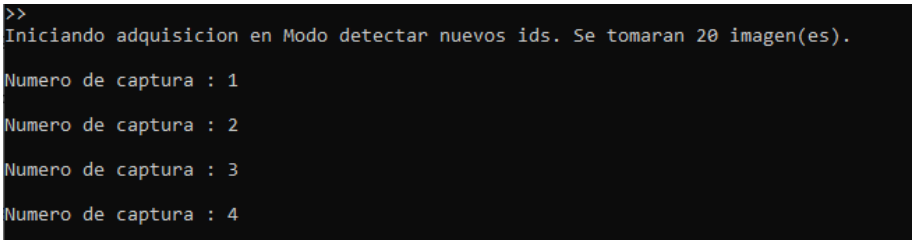

Figura 9.7 Modo 3 en la interfaz.

8. El último modo es el de captura de imágenes en tiempo real (el modo que debe estar en funcionamiento para conocer la posición del robot mediante las cámaras) se accede pulsando *'4'*. Tras esto se da la opción de elegir una cámara para poder tener una previsualización de lo que está viendo una de las 4 cámaras. Para detener el proceso basta con pulsar *'q'*.

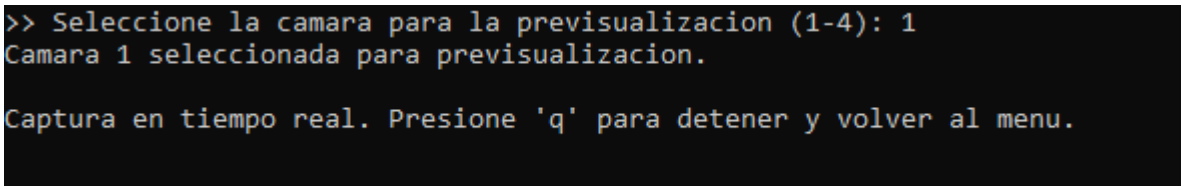

Figura 9.8 Inicio Modo 4 en la interfaz.

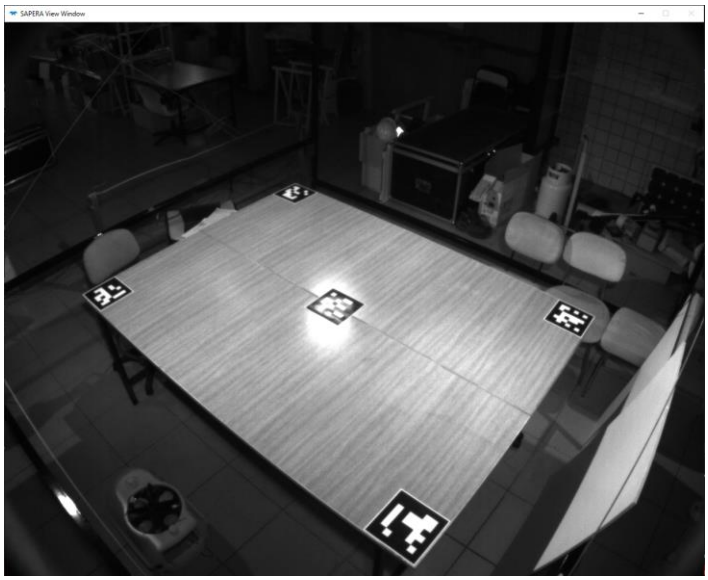

Figura 9.9 Previsualización desde la cámara 1.

9. Para terminar el programa una vez realizado todo lo necesario se pulsa *'q'* desde el menú de selección de modos. Apareciendo un mensaje que dice *"saliendo del programa"*.

9.2. Programa de tratamiento de imágenes y control del robot

Para ejecutar este programa, primero se deben realizar previamente los pasos explicados en el capítulo 5.5. Una vez realizados se puede lanzar el programa escribiendo en un terminal de WSL *"Python3 control_robot.py"*.

1. Lo primero que se mostrará será una breve una introducción del programa y que los archivos del PID están cargados.

```
Programa de control de robot mediante las imágenes tomadas por las cámaras iniciado
Parámetros PID cargados.
```

Figura 9.10 Inicio de interfaz del programa de Python.

2. Tras la introducción se muestra las opciones de. Las opciones se muestran en la Figura 9.11.

```
--- Menú de Configuración ---
1. Configurar parámetros PID
2. Configurar velocidad lineal máxima (actual: 0.07 m/s)
3. Configurar velocidad angular máxima (actual: 1.00 rad/s)
4. Configurar velocidad lineal máxima fuera de los márgenes (actual: 0.04 m/s)
5. Configurar velocidad angular máxima fuera de los márgenes (actual: 0.70 rad/s)
6. Configurar tolerancia de distancia para marcadores nuevos (actual: 0.07 m)
7. Configurar tolerancia de distancia para marcadores existentes (actual: 0.10 m)
8. Configurar tolerancia de distancia para puntos objetivo (actual: 0.07 m)
9. Configurar margen interno de seguridad (actual: 0.15 m)
10. Configurar margen externo de seguridad (actual: 0.15 m)
11. Restablecer valores por defecto
12. Continuar con la ejecución
Seleccione una opción (1-12):
```

Figura 9.11 Menú de configuración.

3. Si se pulsa *'l'* se accede al menú de configuración del PID en donde, primero se muestran los valores actuales tanto del controlador PID lineal como del angular y se pregunta si se quieren cambiar pulsando *'s'*, si se desean dejar iguales *'n'* o si se quiere restablecer a los definidos por defecto *'default'*.

```
--- Configuración de Parámetros PID ---

Controlador Lineal:
   Kp=0.6, Ki=0.0, Kd=0.1
   integrator_limit=0.5
   output_min=None, output_max=None
   deadband=0.0

Controlador Angular:
   Kp=0.809, Ki=2.048, Kd=0.0799
   integrator_limit=1.0
   output_min=None, output_max=None
   deadband=0.0

¿Desea cambiar los valores de los parámetros PID? (s/n/default):
```

Figura 9.12 Configuración PID.

4. Si se decide modificarlos, se deben ir introduciendo uno a uno, primero para el controlador lineal y después, para el angular. Los valores del proporcional, integrador y derivativo si se quieren dejar igual basta con escribir el valor actual, pero para el límite del integrador, los outputs y el deadband se pueden dejar vacíos y no se modifican. Una vez terminado se mostrará que se han guardado los parámetros del PID y la configuración.

```
¿Desea cambiar los valores de los parámetros PID? (s/n/default): s

Configurando controlador Lineal:
  Introduce Kp para lineal (actual=0.6): 0.6
  Introduce Ki para lineal (actual=0.0): 0
  Introduce Kd para lineal (actual=0.1): 0.1
  Introduce integrator_limit (actual=0.5, vacío para no cambiar):
  Introduce output_min (actual=None, vacío para no cambiar):
  Introduce output_max (actual=None, vacío para no cambiar):
  Introduce deadband (actual=0.0, vacío para no cambiar):

Configurando controlador Angular:
  Introduce Kp para angular (actual=0.809): 0.809
  Introduce Ki para angular (actual=2.048): 2.048
  Introduce Kd para angular (actual=0.0799): 0.0799
  Introduce integrator_limit (actual=1.0, vacío para no cambiar):
  Introduce output_min (actual=None, vacío para no cambiar):
  Introduce output_max (actual=None, vacío para no cambiar):
  Introduce deadband (actual=0.0, vacío para no cambiar):
Parámetros del PID guardados.
Configuración guardada exitosamente.
```

Figura 9.13 Edición de los parámetros PID.

Para salir, se indica que no se quieren modificar los parámetros, pulsando *'n'*.

5. Para la configuración del 2 al 10 el funcionamiento para cambiar los valores es bastante análogo. Se teclea el número correspondiente a la opción para cambiar el valor correspondiente. El programa pide el nuevo valor, se escribe y se pulsa *'ENTER'*. Aparecerá que se ha guardado correctamente y se vuelve a menú. Como ejemplo, en la Figura 9.14 se muestra la configuración para la opción 2 que permite cambiar la velocidad lineal máxima.

```
Seleccione una opción (1-12): 2
Introduce la velocidad lineal máxima (m/s): 0.07
Configuración guardada exitosamente.
```

Figura 9.14 Edición de la configuración 2.

6. La opción 11 si se selecciona, devuelve todos los valores de los parámetros de configuración a sus valores predefinidos por defecto.
7. La opción 12 se selecciona cuando no se quiera cambiar nada más de la configuración y se busque que el programa continúe.
8. Tras ejecutar la opción 12 se comprueba la conexión con el robot y con el *ROS Master*. Si es exitosa se indica el mensaje de la Figura 9.15.

```
Maestro de ROS encontrado.
Verificando conexión al maestro de ROS probando el recibimiento de mensajes
```

Figura 9.15 Conexión exitosa con ROS.

9. Si la conexión con el *ROS Master* es defectuosa se muestra la Figura 9.16 y se da la opción de seguir probando hasta que funcione tecleando *'s'* o *'n'* para cerrar el programa.

```
No se encontró el maestro de ROS. Detalle: [Errno 11] Resource temporarily unavailable
¿Deseas seguir intentando conectar al master? (s/n):
```

Figura 9.16 Conexión fallida con ROS Master.

Si lo que falla es la conexión con el robot y no se recibe el topic */odom* tras 5 segundos se cierra el programa automáticamente.

```
Verificando conexión al maestro de ROS probando el recibimiento de mensajes
No se detectó el tópico /odom en el tiempo asignado (5s). Cerrando el programa.
```

Figura 9.17 Tópico no detectado a tiempo.

10. Si la conexión con ROS y el robot es correcta aparecerá un mensaje preguntando si el programa de captura de imágenes está en funcionamiento, si está ya en marcha se deberá teclear *'s'* y si no se encuentra en funcionamiento *'n'* y se cierra el programa. Si se tecleara *'s'* pero el programa de captura de imágenes no estuviera en funcionamiento aparecería un error dado que las imágenes no se actualizan por lo que se impediría el movimiento del robot como se ve en la Figura 9.19.

```
Asegúrate de que el programa de captura de imágenes está funcionando. Selecciona (s/n): n
Programa finalizado. Por favor, ejecuta el programa de captura de imágenes.
```

Figura 9.18 Comprobación de funcionamiento del programa de captura de imágenes.

```
Asegúrate de que el programa de captura de imágenes está funcionando. Selecciona (s/n): s
[ERROR] [1741953221.784755]: La imagen no se ha actualizado (en el tiempo limite). Deteniendo robot por seguridad.
```

Figura 9.19 No hay actualización de imágenes.

11. Si el programa de captura de imágenes de las cámaras está funcionando correctamente. En la siguiente fase se mostrará la posición estimada mediante las homografías del ID 50, si no se puede estimar se supondrá que se encuentra a la mitad de distancia tanto en horizontal como en vertical del rectángulo que definen los 4 marcadores esquina, es decir, la posición (0.9505, 0.7015). Posteriormente, aparecerán los nuevos IDs detectados, se mostrará el número del ID correspondiente y su posición (x,y) estimada

en metros respecto al eje de coordenadas.

```
Posición promedio de ID 50 (posición home): [0.96626747 0.719327  ]
Posición promedio del nuevo marcador ID 6: [1.6436328 0.5840362]
Posición promedio del nuevo marcador ID 9: [0.953824   0.20382003]
Posición promedio del nuevo marcador ID 7: [1.3331156 1.1042452]
Posición promedio del nuevo marcador ID 8: [0.28911275 0.57853556]
Posición promedio del nuevo marcador ID 10: [0.49374822 1.113529  ]
```

Figura 9.20 Posiciones calculadas de los marcadores.

12. Tras ver las posiciones de los nuevos marcadores se da la opción de elegir entre el modo *"manual"* o el modo *"automático"*.

```
Seleccione el modo de operación: 'manual' o 'automatico':
```

Figura 9.21 Selección de modo de operación.

13. Si se selecciona el modo manual aparecerá un texto que explica brevemente el funcionamiento, el robot se puede controlar pulsando las teclas *'w'* (avance hacia delante), *'s'* (avance hacia atrás), *'d'* (giro a derechas), *'a'* (giro a izquierdas). También se puede ajustar la velocidad hasta los límites establecidos en la configuración. *'e'* aumenta la velocidad lineal 0.02 m/s. *'r'* reduce la velocidad lineal 0.02 m/s, *'t'* aumenta la velocidad angular 0.05 rad/s e *'y'* reduce la velocidad angular 0.05 rad/s. Para que funcione tiene que estar abierta la ventana creada por la librería *"pygame"*.
14. Para salir basta con pulsar *'q'* y se mostrará primero los resultados del modo manual:
    - Posición inicial real.
    - Posición final real.
    - Tiempo de duración del modo.
    - Imágenes no leídas del total.
15. Tras los resultados, se dibujan las gráficas de la trayectoria seguida y el registro de velocidades, posteriormente, finaliza el programa.

```
--- Modo Manual ---
Use las teclas:
  w/s => movimiento adelante/atrás
  a/d => giro izquierda/derecha
  q   => salir del modo manual
Ajuste de velocidad:
  [e/r] => +/- 0.02 vel. lineal base (inicial=0.12 m/s; max=0.07)
  [t/y] => +/- 0.05 vel. angular base (inicial=1.20 rad/s; max=1.00)
```

Figura 9.22 Interfaz del modo manual.

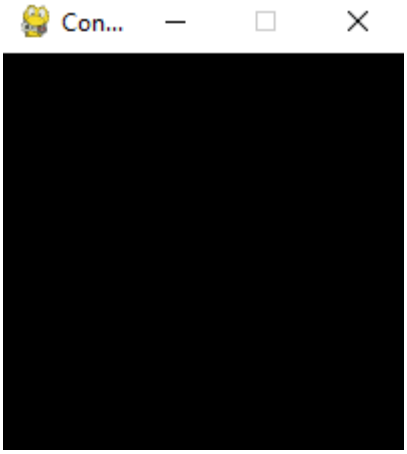

Figura 9.23 Interfaz del modo manual.

```
--- RESULTADOS MODO MANUAL ---
· Posición inicial real: (0.260, 0.453)
· Posición final real:   (1.158, 0.292)
· Tiempo transcurrido:   21.24 s
· Imágenes defectuosas / total: 5 / 720
-------------------------------------------------
```

Figura 9.24 Resultados del modo manual.

16. Si se selecciona el modo automático, lo primero que aparece es el estado del robot, si se encuentra dentro de los márgenes de la mesa pondrá *'normal'*, se mostrará un plano 2D con la distribución de la mesa y las posiciones de los marcadores, para continuar hay que cerrar esta imagen. Una vez cerrada se puede elegir entre ir a uno de los marcadores existentes o ir a un punto cualquiera dentro de la mesa.

```
Seleccione el modo de operación: 'manual' o 'automatico': automatico
Estado inicial del robot: normal

Modo automático.
¿Desea ir a un 'marcador' o a un 'punto'? (marcador/punto):
```

Figura 9.25 Selección de marcador o punto.

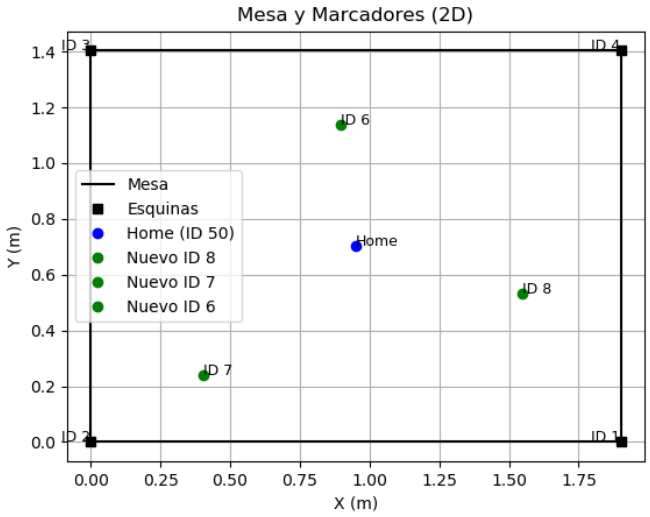

Figura 9.26 Ejemplo de distribución del plano de la mesa en interfaz.

17. Si se teclea *'marcador'*, aparecen los IDs de los marcadores disponibles y se pide que se escriba el número al que ir.

```
¿Desea ir a un 'marcador' o a un 'punto'? (marcador/punto): marcador

IDs disponibles: [1, 2, 3, 4, 50, 8, 7, 6]
Introduce el ID del marcador objetivo ([1, 2, 3, 4, 50, 8, 7, 6]): 7
Procesamiento en tiempo real. Presiona Ctrl+C para parar.
```

Figura 9.27 Ir a marcador.

18. Si se selecciona *'punto'* aparecen 2 opciones, introducir las coordenadas por teclado, 'manual' o seleccionarlas clicando sobre un plano 2D de la mesa 'clic'.

```
¿Desea ir a un 'marcador' o a un 'punto'? (marcador/punto): punto
¿Desea ingresar las coordenadas manualmente o por clic? (manual/clic):
```

Figura 9.28 Modo de selección del punto objetivo.

19. Si seleccionan las coordenadas manuales, primero habrá que introducir la posición X en metros y después la Y.

```
¿Desea ingresar las coordenadas manualmente o por clic? (manual/clic): manual
Introduce la coordenada X: 1.25
Introduce la coordenada Y: 0.25
Procesamiento en tiempo real. Presiona Ctrl+C para parar.
```

Figura 9.29 Introducción de punto objetivo por teclado.

20. Si se selecciona seleccionar el punto por *'clic'*, aparece un mapa con los marcadores en el que habrá que hacer clic con el ratón en la posición a donde se precisa que vaya el robot.

```
¿Desea ingresar las coordenadas manualmente o por clic? (manual/clic): clic
Punto de destino seleccionado: (0.39, 0.31)
Procesamiento en tiempo real. Presiona Ctrl+C para parar.
```

Figura 9.30 Introducción de punto objetivo haciendo clic.

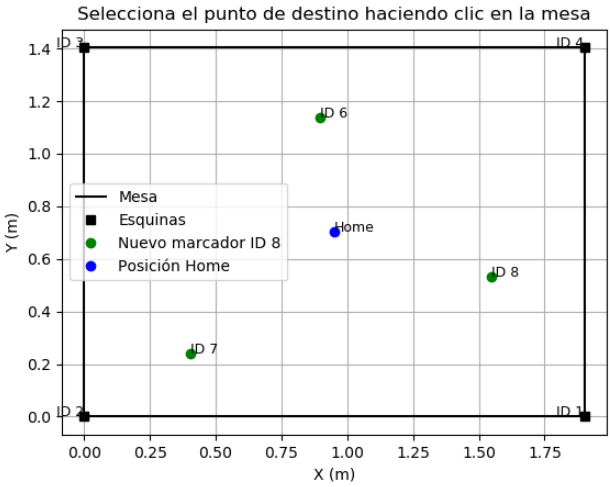

Figura 9.31 Ventana de selección por clic.

21. Una vez el robot llega al punto objetivo se muestran los resultados del movimiento y se pregunta si se quiere ir a otra posición *'s'* o *'n'* si se quiere finalizar. Los resultados del movimiento que se muestran son:

    - Posición inicial.
    - Posición final.
    - Tiempo que tarda el robot en llegar al objetivo.
    - Número de imágenes perdidas.

```
--- RESULTADOS DEL MODO AUTOMÁTICO AL LLEGAR AL OBJETIVO ---
· Posición inicial real: (1.158, 0.292)
· Posición final real:   (1.606, 0.530)
· Tiempo transcurrido:   8.57 s
· Imágenes defectuosas / total: 3 / 312
--------------------------------

¿Desea ir a otra posición? (s/n):
```

Figura 9.32 Resultados del movimiento automático tras llegar al objetivo.

22. Tras alcanzar la meta y mostrar los resultados aparece una gráfica con los errores en distancia (m) y ángulo (rad) durante el trayecto.

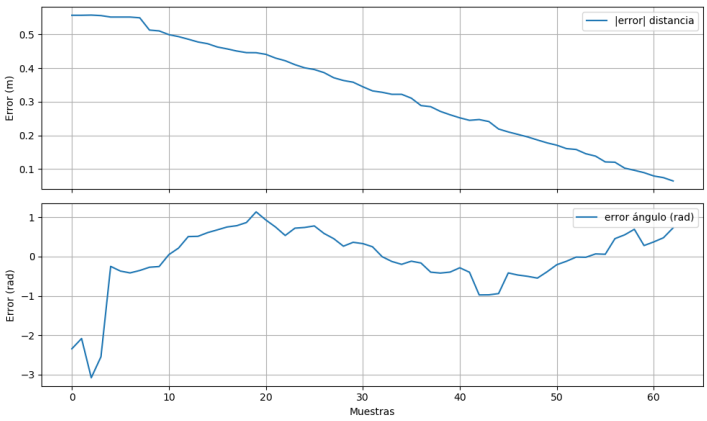

Figura 9.33 Ejemplo de gráfica de errores.

23. Una vez se haya finalizado el modo automático, se mostrará primero un dibujo 2D sobre un plano de la trayectoria seguida, después la trayectoria vista desde la cámara 1 y finalmente el registro de velocidades.

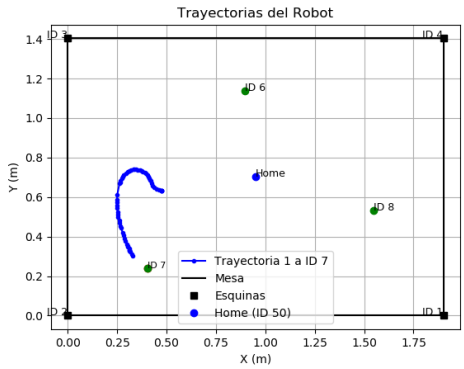

Figura 9.34 Ejemplo de trayectoria sobre dibujo 2D.

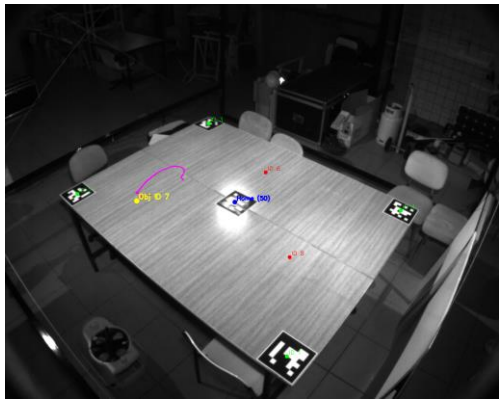

Figura 9.35 Ejemplo de trayectoria desde la cámara 1.

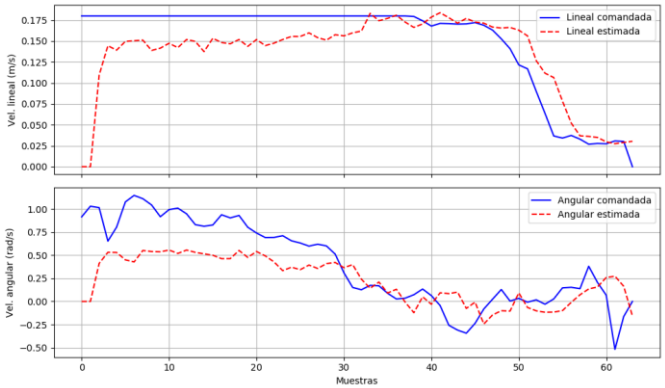

Figura 9.36 Ejemplo de registro de velocidades.

24. Después de las gráficas saldrá un mensaje indicando que el programa ha finalizado.

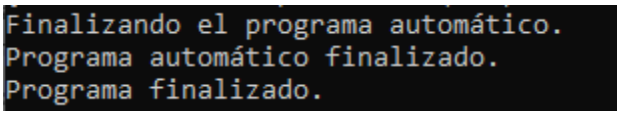

Figura 9.37 Ejemplo de registro de velocidades.

10 EXPERIMENTACIÓN

Una vez explicado todo el proyecto realizado se exponen las diferentes pruebas que se llevaron a cabo.

10.1. Apertura de la lente

A la hora de tomar las fotos, cada una de las cámaras Genie Nano M1280-NIR da la opción de aumentar o disminuir la entrada de luz en la lente girando un anillo anexo a este. Se han realizado varias fotos para evaluar su impacto.

Tomando imágenes desde la cámara 1, se han obtenido los siguientes resultados.

Apertura de la lente al máximo:

Figura 10.1 Fotografía desde la cámara 1 con la apertura al máximo.

Apertura de la lente al mínimo:

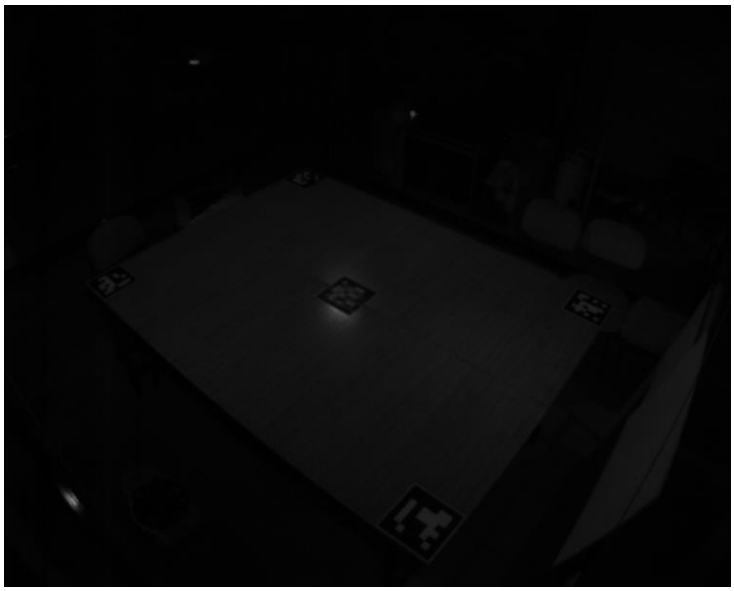

Figura 10.2 Fotografía desde la cámara 1 con la apertura al mínimo.

Apertura de la lente en posición aproximadamente intermedia:

Figura 10.3 Fotografía desde la cámara 1 con la apertura a nivel intermedio.

Se aprecia que como mejor se ve la mesa es con una apertura intermedia, dado que con la máxima apertura la mesa posee excesivo brillo y con la mínima se observa demasiado oscura.

Ahora se analizará la lectura de los ArUcos para cada una de las fotos anteriormente presentadas en las que se modificaba la entrada de luz.

Figura 10.4 Detección de marcadores con máxima apertura.

Con la apertura de lente al máximo solo detecta el ArUco ID 1 el cual es el más cercano a la cámara, se puede observar como con este nivel de apertura de lente es difícil el contraste de los elementos de la mesa, entre ellos los ArUco. Por ejemplo, en la imagen se puede apreciar que, debido a la gran luminosidad, el ID 50 no se llega ni a ver en la imagen.

Figura 10.5 Detección de marcadores con mínima apertura.

Por otro lado, como se observa en la Figura 10.5, si la apertura de la lente se abre al mínimo, hay muy poca luminosidad y bajo contraste, por lo que al haber un contraste tan bajo entre blancos y negros el programa es incapaz de detectar ningún ArUco.

Figura 10.6 Detección de marcadores con apertura intermedia.

Para un nivel intermedio de apertura sí que se logra la lectura de los marcadores esquina (1,2,3,4), pero el ArUco

central (50) no se lee debido a los reflejos y su alta luminosidad, así que la idea va a ser que para cada una de las cámaras va a haber que ir probando a ir cerrando la lente con un paso pequeño hasta alcanzar un punto donde se puedan leer los ArUcos esquina y a su vez el marcador central ID 50.

Tras varios experimentos, se observa que el problema reside en que no hay un punto intermedio en el que la apertura de la lente sea suficiente para detectar todos los marcadores de las esquinas y que a su vez exista contraste suficiente para que el ArUco del centro con ID 50, no le afecten los reflejos o fugas de luz para poder ser detectado.

Por ello se decide cambiar el material y tamaño del marcador 50, este pasa a ser ligeramente más pequeño y hecho de papel en vez de cartulina para que tenga un acabado más mate y no se refleje, además se incrementan los bordes del papel.

Figura 10.7 Detección desde cámara 1 con nuevo marcador 50.

Con esta modificación, ya sí que se puede encontrar la apertura de la lente adecuada para que se puedan ver todos los marcadores esquina además del marcador central como se aprecia en la Figura 10.7.

Ya conseguido esto, se ajusta la entrada de luz del resto de cámaras buscando el ajuste donde se detecten todos los marcadores.

Cámara 2:

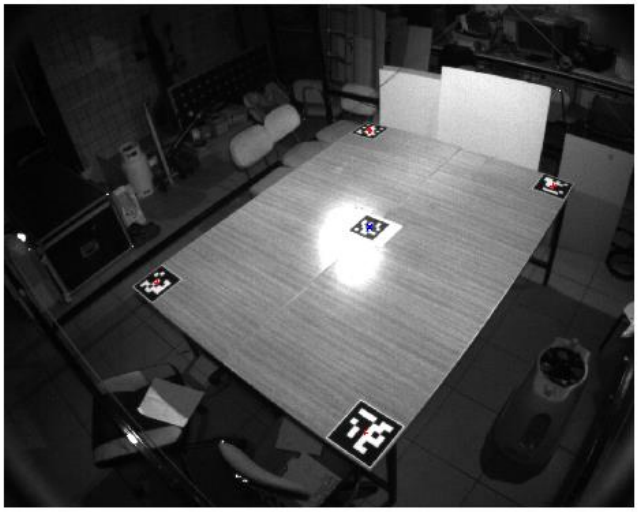

Figura 10.8 Detección de marcadores desde cámara 2.

Cámara 3:

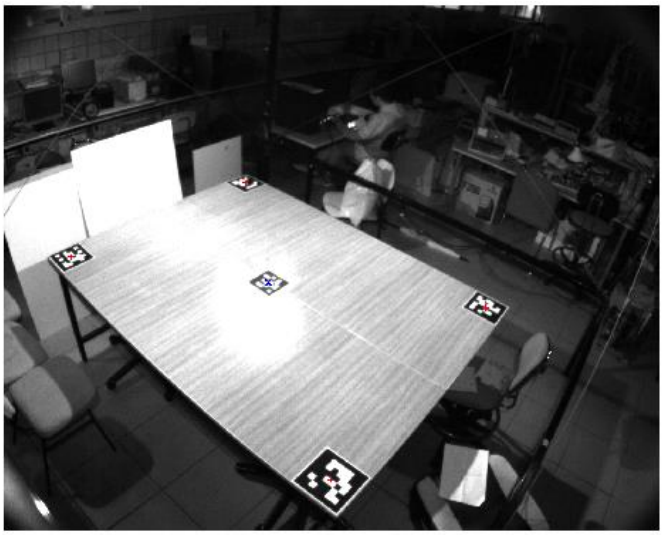

Figura 10.9 Detección de marcadores desde cámara 3.

Cámara 4:

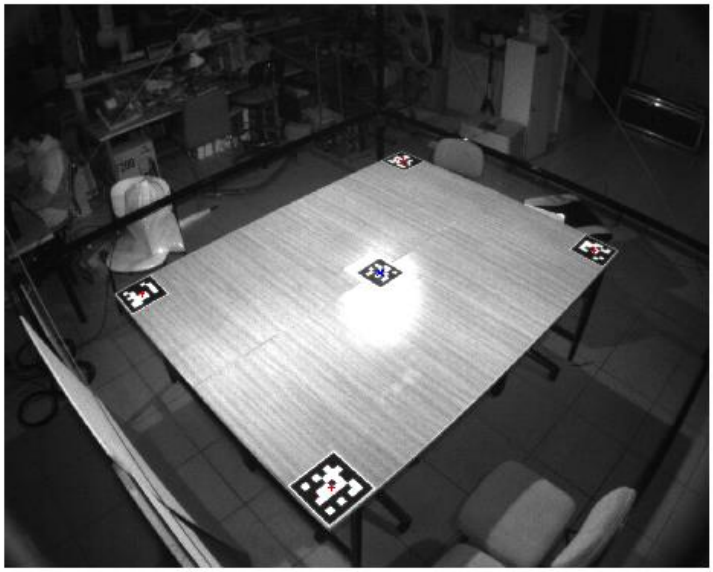

Figura 10.10 Detección de marcadores desde cámara 4.

Como se observa, no todas las cámaras tienen la misma entrada de luz dado que dependiendo de su posición y orientación puede necesitar más o menos luz para detectar los marcadores. Por ejemplo, se puede apreciar que las fotos realizadas desde la cámara 3 tienen mayor luminosidad que el resto debido a que necesita más luz para ver los marcadores.

Para la calibración del plano robot cuando el robot se encontraba sobre el ID 1 desde la cámara 3 no se detectaba el ID 120, como esta era la única imagen que daba problemas lo que se hizo fue trucarla sacando una foto en la que el robot se alejó unos centímetros del borde y se le dio un poco más de brillo. De esta manera el error introducido sería prácticamente despreciable.

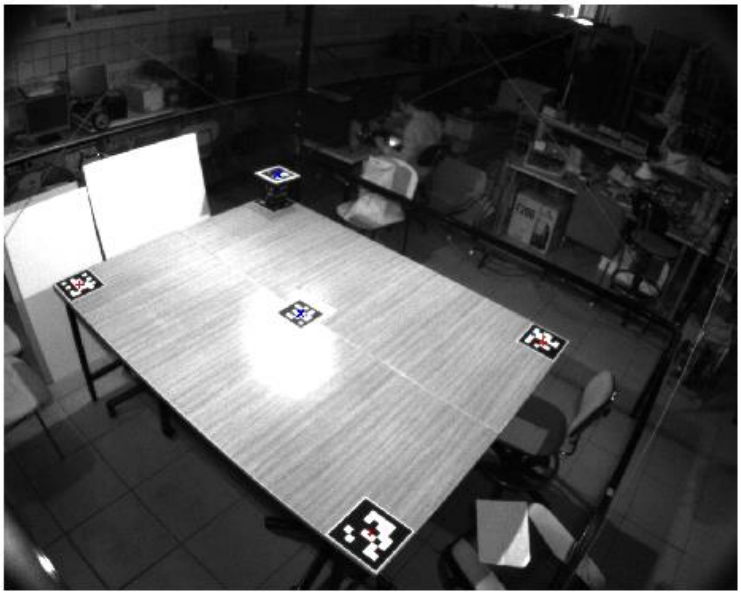

Figura 10.11 Detección de marcador 120 desde cámara 3.

10.2. Valores iniciales de los parámetros del menú de ajuste.

A continuación, se detallan los valores que tendrán los parámetros del menú de configuración para las primeras

pruebas.

Opción 1: Los valores adecuados para el PID se irán viendo en el capítulo 10.4.

Opción 2: Velocidad máxima lineal del robot. Para ver la velocidad máxima del robot se pondrá un valor elevado en modo manual y en la gráfica de velocidad desarrollada (Figura 10.12) se verá cuál es el máximo alcanzado.

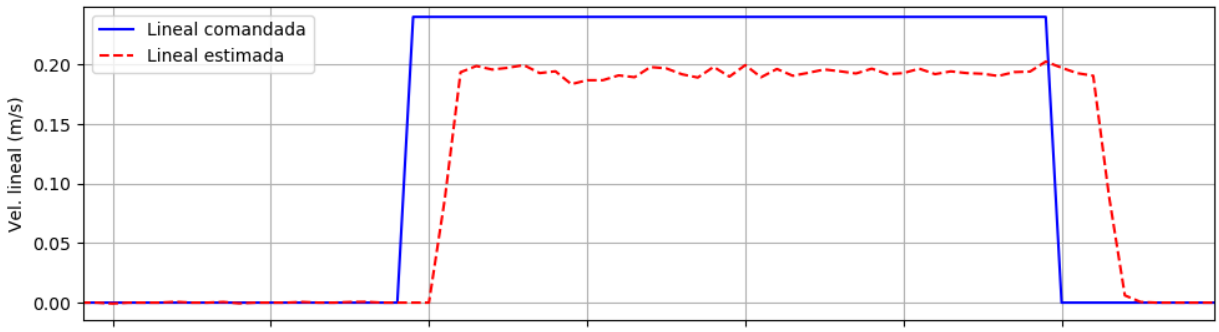

Figura 10.12 Prueba de velocidad lineal máxima.

En el ejemplo se le demandó 0.24 m/s, pero se observa que en ningún momento superaba los 0.20 m/s y los valores se encontraban entre 0.18 y 0.20 m/s de forma estable tras alcanzar la saturación. Por ello se tomará 0.18 m/s como velocidad lineal máxima.

Opción 3: Velocidad angular máxima del robot. Al igual que con la opción 2 se pondrá un valor elevado de velocidad angular máxima y se observará cuando la velocidad estimada satura.

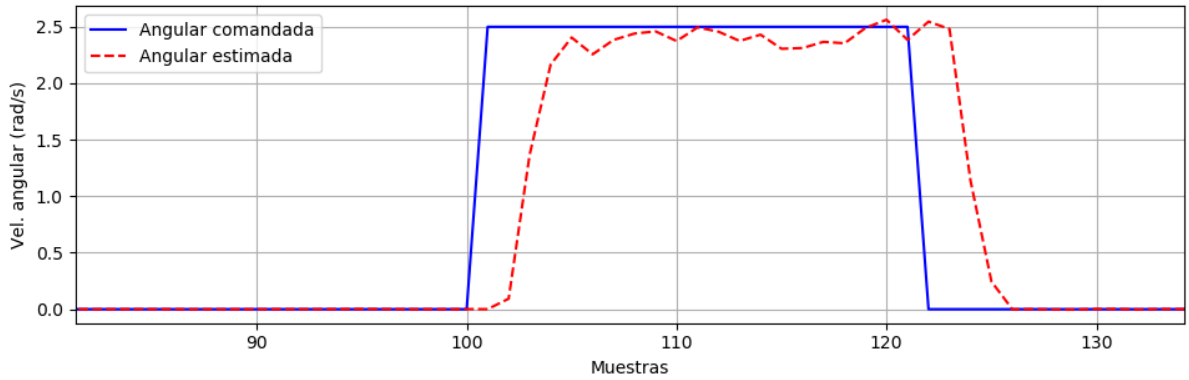

Figura 10.13 Prueba de velocidad angular máxima.

Se observa que algún pico consigue llegar a los 2.5 rad/s, pero de forma fugaz, la saturación varía entre 2.3 y 2.5 rad/s. Se supondrá que la velocidad angular máxima es de 2.3 rad/s, pero como valor inicial se establecerá en 2.25 rad/s.

Opción 4: Velocidad lineal máxima fuera de los márgenes. Esta velocidad tiene que ser muy pequeña porque si no podría provocar la caída del robot. También hay que tener en cuenta el desfase de tiempo hasta que se detecta que el robot está fuera de los márgenes. En principio se pondrá en 0.04 m/s que se consideran suficientemente seguros.

Opción 5: Velocidad angular fuera de los márgenes. Esta velocidad es menos peligrosa que la lineal fuera de los márgenes dado que para que el robot se caiga girando debe encontrarse muy cerca del límite de la mesa, además, no debe ser tampoco excesivamente baja porque si el robot se dirige hacia el borde de la mesa significa que tendrá que girar casi 180 grados para dirigirse a la posición home de seguridad. En principio estará en 1 rad/s.

Opción 6: Tolerancia para marcadores nuevos. Si la tolerancia es muy pequeña al robot le costará encontrar la posición objetivo y tendrá que dar más vueltas, por el contrario, si la tolerancia es demasiado grande podría quedarse lejos de la coordenada objetivo original. Como primer valor estará en 0.1 m.

Opción 7: Tolerancia para marcadores ya existentes. Aquí entran los marcadores esquina por lo que hay que tener cuidado de que el robot no se acerque mucho al estar en los límites de la mesa, de todas formas, como es bastante probable que se encuentren fuera de los márgenes externos se asignaría una posición dentro de los márgenes. Sería recomendable que la tolerancia fuera mayor que la de los marcadores nuevos. El valor inicial será 0.2 m.

Opción 8: Tolerancia a puntos objetivo. Esta sería la tolerancia para los puntos objetivos que o bien se han seleccionado en el mapa con el ratón o se han introducido las coordenadas manualmente. En principio se tomará en 0.1 m.

La opción 9 y 10 configura los márgenes, existe un margen interno y otro externo para que haya un poco de histéresis cuando el robot vuelve a entrar dentro de los márgenes. Si se hace que el robot vuelva a dirigirse a la posición objetivo en cuanto entre dentro de los márgenes al estar muy cerca del límite del margen es posible que vuelva a salir de este, para ello se diferencia entre margen interno y externo. El robot si sale del margen externo se considera que está fuera de los márgenes y se dirige a baja velocidad a la posición home, pero cuando entra dentro del margen externo se sigue dirigiendo a la posición home, pero ya a velocidad normal y una vez entra en el margen interno es cuando se vuelve a dirigir a la posición objetivo, pero si el robot sale del margen interno se considera que sigue dentro de los márgenes, para salir de los márgenes tiene que cruzar el externo. El margen externo debe tener siempre un valor menor o igual que el interno. Este representa los metros de distancia al borde de la mesa. Se empezará sin distinguir entre margen interno y externo (opción 8 y 9) se pondrá en 0.15 m.

Por otro lado, hay que añadir que los nuevos marcadores y sus posiciones (en metros) que se usarán serán:

- ID 50 (posición home): (0.966, 0.719).
- ID 6: (1.644, 0.584).
- ID 7: (1.333, 1.104).
- ID 8: (0.289, 0.578).
- ID 9: (0.954, 0.204).
- ID 10: (0.494, 1.113).

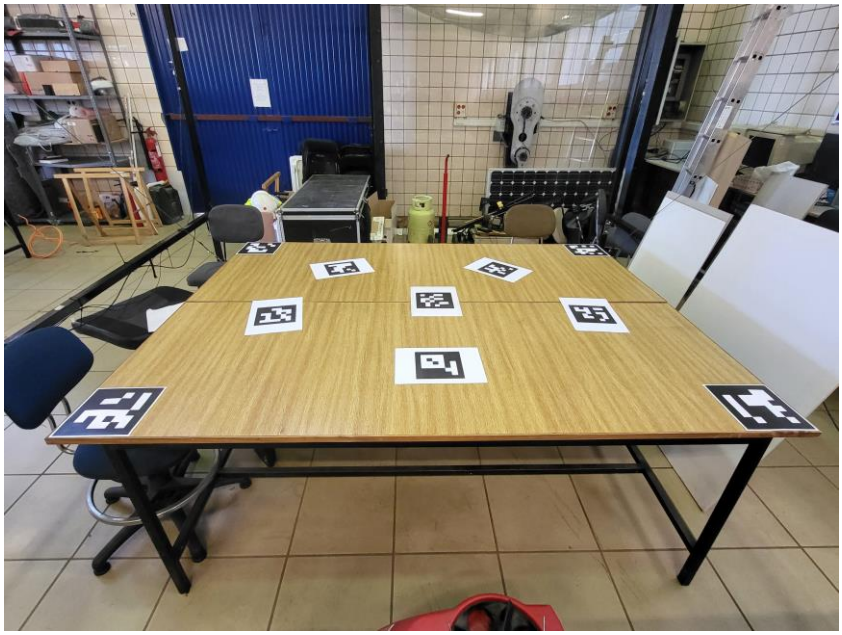

Figura 10.14 Fotografía real de los nuevos marcadores.

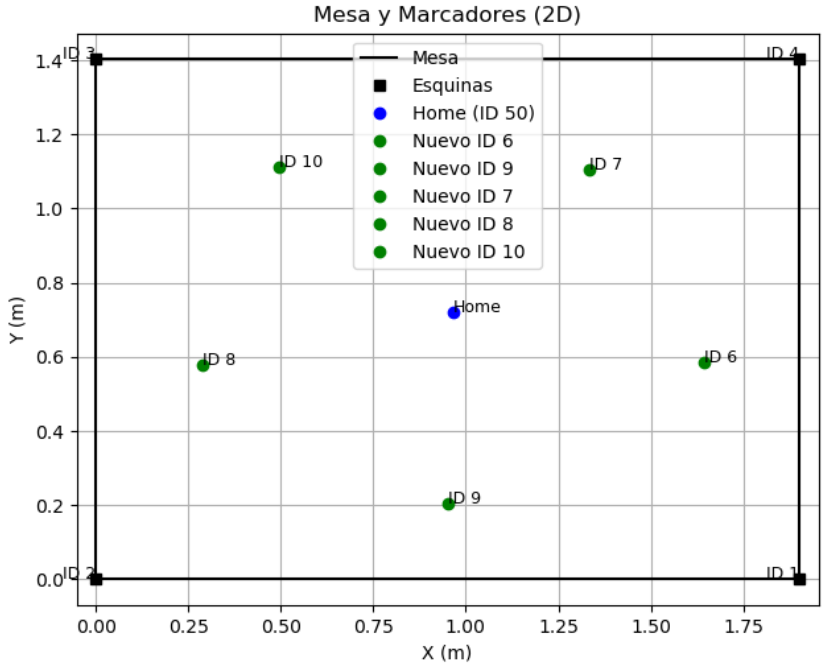

Figura 10.15 Estimación de los nuevos marcadores sobre el plano 2D.

10.3. Movimiento del robot en modo manual

Uno de los primeros experimentos será ver el funcionamiento del modo manual dado que no necesita una gran configuración, pero proporcionará información sobre si las cámaras estiman la posición del robot correctamente.

Para este experimento se coloca el robot en la mesa, en una posición que no es muy limítrofe y se pone a dar vueltas por todas las zonas de la mesa controlándolo por teclado.

```
--- Modo Manual ---
Use las teclas:
  w/s => movimiento adelante/atrás
  a/d => giro izquierda/derecha
  q   => salir del modo manual
Ajuste de velocidad:
  [e/r] => +/- 0.02 vel. lineal base (inicial=0.12 m/s; max=0.18)
  [t/y] => +/- 0.05 vel. angular base (inicial=1.20 rad/s; max=2.50)

eeVel. lineal base => 0.14 m/s
Vel. lineal base => 0.16 m/s
Vel. lineal base => 0.18 m/s
Vel. angular base => 1.25 rad/s
Vel. angular base => 1.30 rad/s
Vel. angular base => 1.35 rad/s
Vel. angular base => 1.40 rad/s
Vel. angular base => 1.45 rad/s
Vel. angular base => 1.50 rad/s
No se pudo estimar la posición actual del robot (no se detectó ID120).
No se pudo estimar la posición actual del robot (no se detectó ID120).
No se pudo estimar la posición actual del robot (no se detectó ID120).
No se pudo estimar la posición actual del robot (no se detectó ID120).
No se pudo estimar la posición actual del robot (no se detectó ID120).
No se pudo estimar la posición actual del robot (no se detectó ID120).
No se pudo estimar la posición actual del robot (no se detectó ID120).
No se pudo estimar la posición actual del robot (no se detectó ID120).
```

Figura 10.16 Experimento 1 del modo manual.

Como se observa en la Figura 10.16, se movió el robot y se probaron diferentes velocidades, pero cuando el robot se acercaba a la posición central de la mesa saltaba el aviso de que no se detectaba la posición, esta notificación no dejó de salir hasta que se alejó el robot de la posición central. En la Figura 10.17 se indica las posiciones iniciales y finales de la prueba y el tiempo de duración, el cual fue de 76.56 segundos, las imágenes defectuosas se analizarán en el capítulo 10.6.

```
--- RESULTADOS MODO MANUAL ---
· Posición inicial real: (0.598, 0.259)
· Posición final real:   (0.782, 0.207)
· Tiempo transcurrido:   76.56 s
· Imágenes defectuosas / total: 18 / 2516
-----------------------------------------------------
```

Figura 10.17 Resultados del experimento 1 del modo manual.

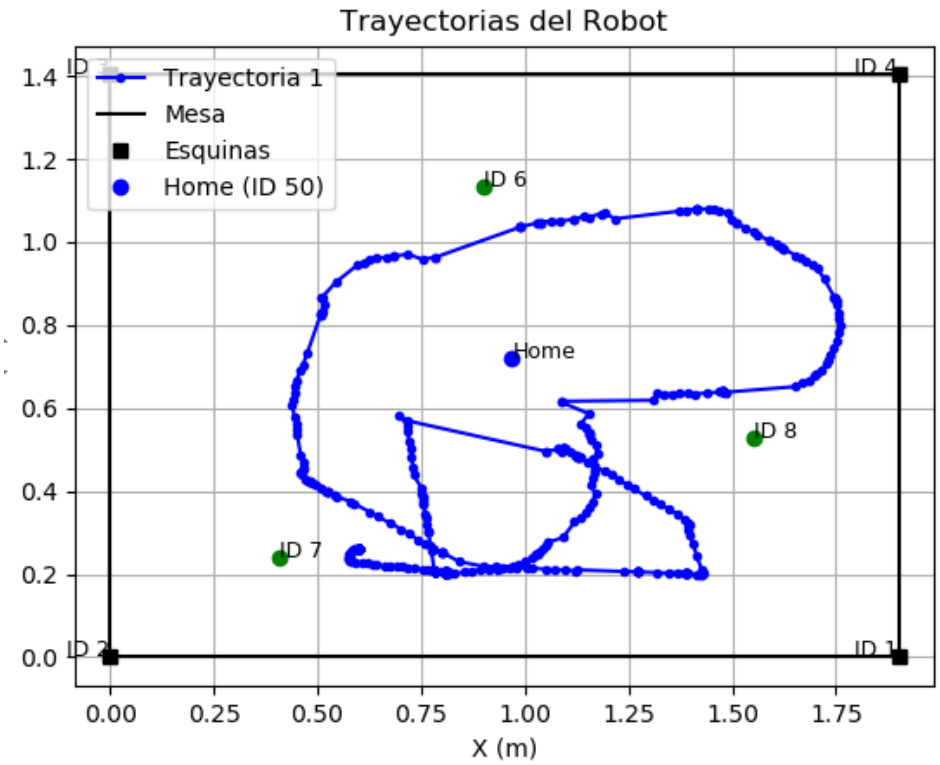

Figura 10.18 Trayectoria del experimento 1 del modo manual.

En la Figura 10.18 se aprecia cómo los puntos cercanos a la posición home han sido omitidos debido a que no se pudieron detectar.

Estos fallos en la detección del marcador 120 se debían al reflejo que sufría por las zonas cercanas al centro de la mesa. La solución para que el marcador sea detectado en el centro y en todos los puntos de la mesa va a ser parecida a la que ya se hizo en el capítulo 10.1 con el ID 50, se imprimirá un nuevo marcador ID 120 el cual es más pequeño y tiene un acabado mate para que refleje menos la luz. En la Figura 10.19 se puede comparar el marcador antiguo (izquierda) con el nuevo (derecha).

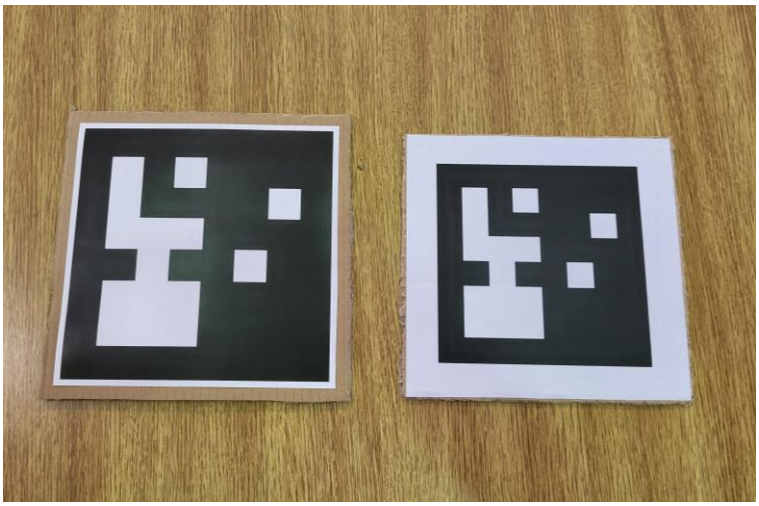

Figura 10.19 Comparación de marcadores ID 120. Marcador antiguo (izquierda) y marcador nuevo (derecha).

Tras cambiar el marcador del robot, lo primero es rehacer la calibración del plano robot, es decir, tomar otra vez las fotos del robot sobre cada uno de los marcadores esquina, una vez hecho esto se repitió el experimento.

```
--- Modo Manual ---
Use las teclas:
  w/s => movimiento adelante/atrás
  a/d => giro izquierda/derecha
  q   => salir del modo manual
Ajuste de velocidad:
  [e/r] => +/- 0.02 vel. lineal base (inicial=0.12 m/s; max=0.18)
  [t/y] => +/- 0.05 vel. angular base (inicial=1.20 rad/s; max=2.50)

Vel. lineal base => 0.14 m/s
Vel. lineal base => 0.16 m/s
Vel. lineal base => 0.18 m/s
Vel. angular base => 1.25 rad/s
Vel. angular base => 1.30 rad/s
Vel. angular base => 1.35 rad/s
Vel. angular base => 1.40 rad/s
Vel. angular base => 1.45 rad/s
Vel. angular base => 1.50 rad/s
Vel. angular base => 1.55 rad/s
Vel. angular base => 1.60 rad/s
Vel. angular base => 1.65 rad/s
Saliendo del modo manual por tecla 'q'.

Robot detenido.

--- RESULTADOS MODO MANUAL ---
· Posición inicial real: (0.308, 0.449)
· Posición final real:   (0.583, 0.227)
· Tiempo transcurrido:   77.51 s
· Imágenes defectuosas / total: 20 / 2592
------------------------------------------------------
```

Figura 10.20 Experimento 2 del modo manual.

Como se observa en la Figura 10.20 para el segundo experimento se partió de la coordenada (0.308, 0.449) y se finalizó en la (0.583, 0.227), las pruebas fueron parecidas a las del experimento 1, pero en este ya no saltaba el aviso de que la posición del robot no era detectada a pesar de que el robot pasó varias veces por el centro de la mesa.

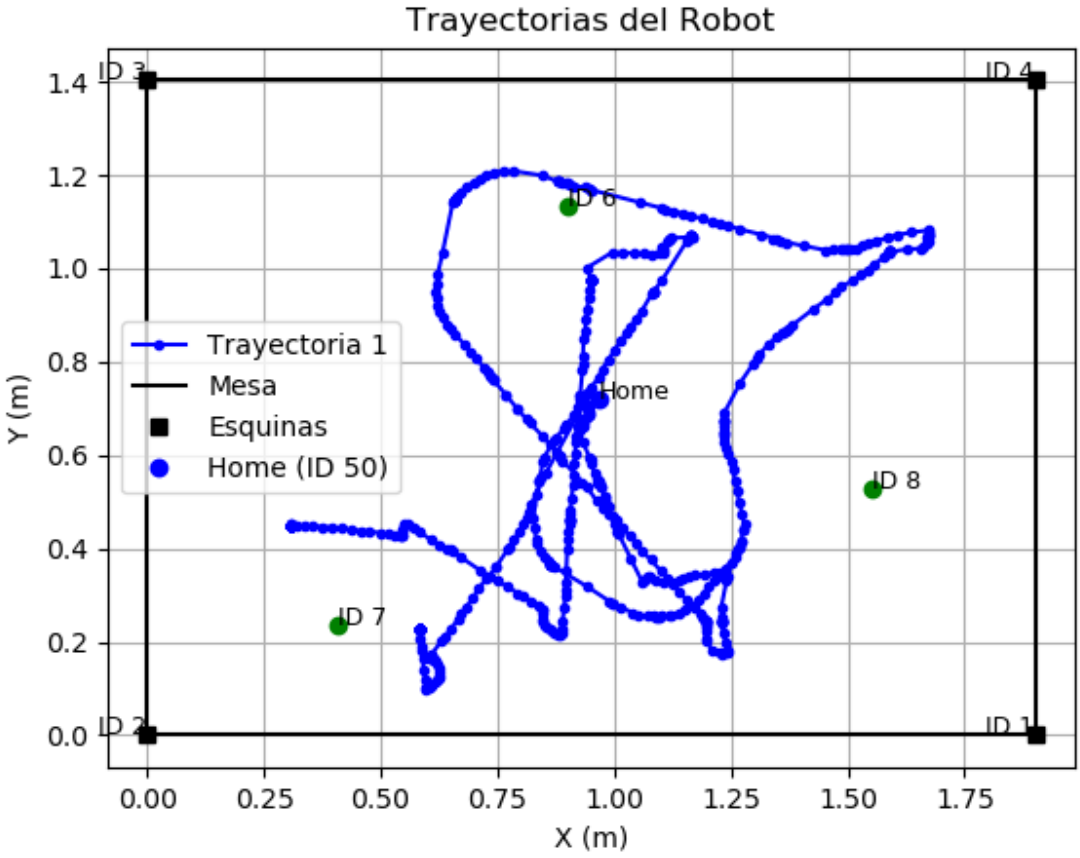

Figura 10.21 Trayectoria del experimento 2 del modo manual.

En la Figura 10.21 se visualiza que el robot ahora sí es detectado cuando se encuentra en zonas centrales, por lo que se puede concluir que con el nuevo marcador del robot el problema ha sido resuelto.

En cuanto al análisis del movimiento, los resultados son bastante buenos. El retraso entre el tiempo que se envía el comando por teclado y el robot acata la orden está aproximadamente en 0.4 segundos, valor bastante aceptable teniendo en cuenta que mínimo existirán 2 metros de distancia entre el adaptador Wi-Fi del ordenador y el robot. El ajuste de velocidad pulsando teclas funciona correctamente. También hay que decir que los movimientos simultáneos si se pulsan las teclas de avance y giro funciona y el robot describe un arco, aunque, como se muestra en la Figura 10.22, cuando se realiza este movimiento simultáneo tanto la velocidad lineal como angular que el robot puede desarrollar decae.

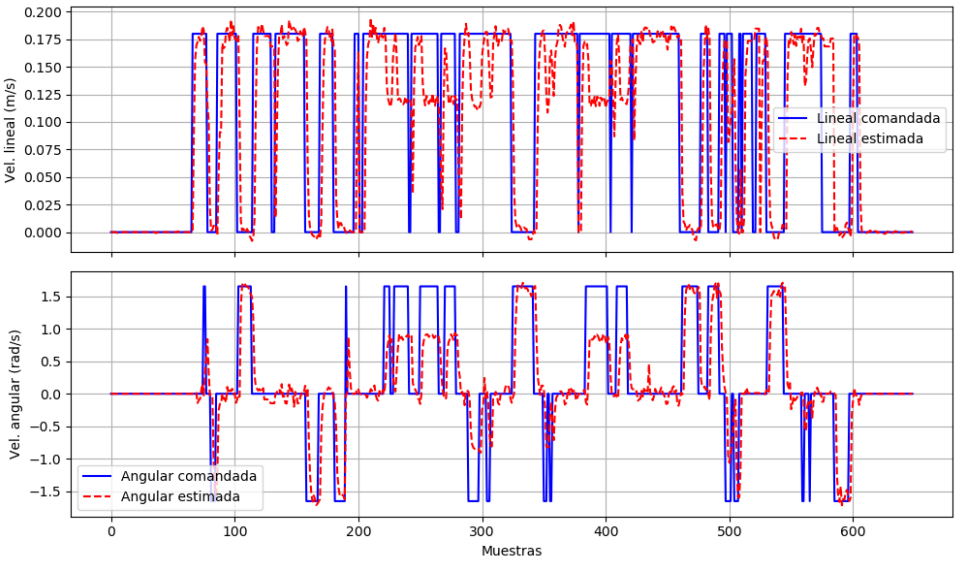

Figura 10.22 Registro de velocidades del experimento 2 del modo manual.

En cuanto a la detección de posición, es bastante preciso solo con errores de muy pocos centímetros debido al pequeño error que puede aparecer en la calibración, estos se pueden considerar aceptables.

10.4. Sintonización del PID para el modo automático

Para realizar la sintonización tanto del controlador PID para el movimiento lineal como para el controlador del movimiento angular se probarán tres métodos. Cancelación de polos en bucle cerrado ignorando los retrasos, Ziegler-Nichols en bucle abierto y Ziegler-Nichols con ajuste fino. En este último método el objetivo será que partiendo del controlador obtenido en el estudio del Ziegler-Nichols en bucle abierto se irán introduciendo pequeñas permutaciones de los términos proporcional, derivativo e integral y observar mediante experiencia si la respuesta mejora o empeora.

10.4.1 Obtención de la función de transferencia del movimiento lineal

Tras haber realizado diferentes pruebas, se ha tomado como velocidad máxima lineal del robot 0.18 m/s, por lo que para obtener la función de transferencia de primer orden que represente el movimiento se pondrá una entrada en escalón de velocidad en un instante determinado y se observará la respuesta de la velocidad lineal. Se hará este ensayo partiendo siempre del reposo y en cuanto a la amplitud del escalón se realizarán varias pruebas variando la amplitud desde 0.03 m/s hasta 0.18 m/s.

Para las pruebas se producirá el escalón en el instante 2 s, este se mostrará en rojo, en azul la velocidad real y en verde discontinuo un filtro de la velocidad realizado con medias móviles para que sea más sencillo la estimación del régimen permanente.

Se explicará la metodología seguida solo para la primera función de transferencia, dado que el método será análogo para el resto.

Escalón a 0.03 m/s:

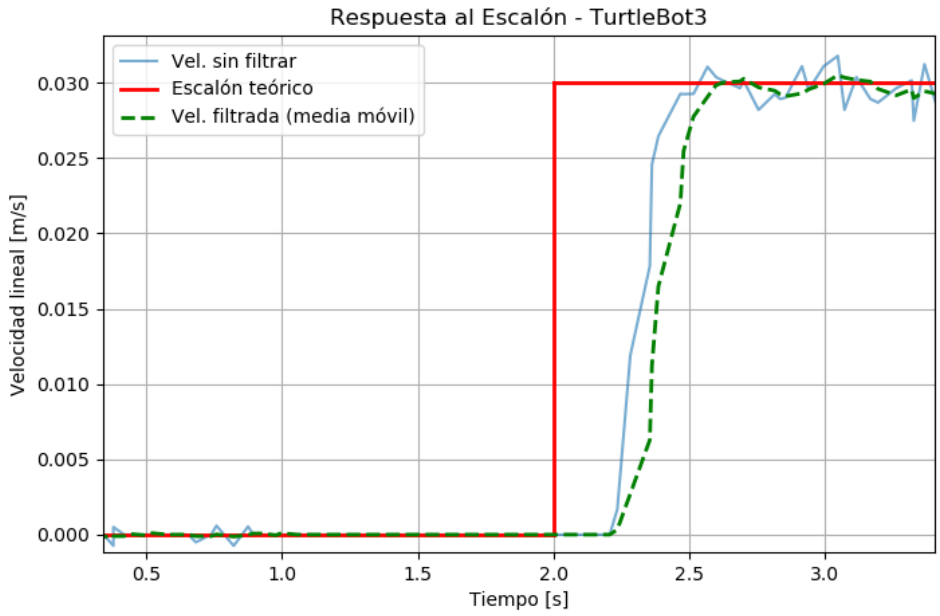

Figura 10.23 Escalón de 0.03 m/s de velocidad lineal.

De esta gráfica se obtiene que:

$$t_0 = 2$$
$$y_0 = 0$$
$$\tau_d + t_0 = 2.207$$
$$t_{rp} = 2.56$$
$$y_{rp} = 0.03$$

En donde:

- t_0 es el instante en que se produce el escalón.
- y_0 es la salida inicial.
- τ_d es el tiempo de retraso inicial.
- t_{rp} es el momento en el que se alcanza el régimen permanente.
- y_{rp} es el valor de la salida en régimen permanente ante la entrada en escalón.

Para sacar la función de primer orden se usará el conocido como "método del 63%" en el cual se toma como constante de tiempo τ el tiempo que se tarda en alcanzar el 63% del valor de la salida en régimen permanente (y_{rp}).

Si definimos la función de transferencia como:

$$G(s) = \frac{K}{\tau s + 1} e^{-\tau_d s} \tag{10.1}$$

La ganancia K, se puede calcular:

$$K = \frac{y_{rp} - y_0}{u_{rp} - u_0} \tag{10.2}$$

Siendo:

- u_{rp} es el valor final de la entrada tras el escalón.
- u_0 es el valor de la entrada previo al escalón.

Con el método del 63% la ecuación para calcular τ queda:

$$y_{0.63} = y_0 + 0.63 * y_{rp} \tag{10.3}$$

Con el $y_{0.63}$ que sería el 63% de la salida utilizando la gráfica se puede obtener en que instante se alcanza.

$$K = \frac{0.03 - 0}{0.03 - 0} = 1$$

$$y_{0.63} = 0 + 0.63 * 0.03 = 0.0189$$

$$t_0 + \tau = 2.35$$

$$\tau = 2.35 - 2 = 0.35$$

De esta forma ya se tiene la función de transferencia entera:

$$G(s)_{0.03} = \frac{1}{0.35s + 1} e^{-0.207s}$$

Escalón a 0.06 m/s:

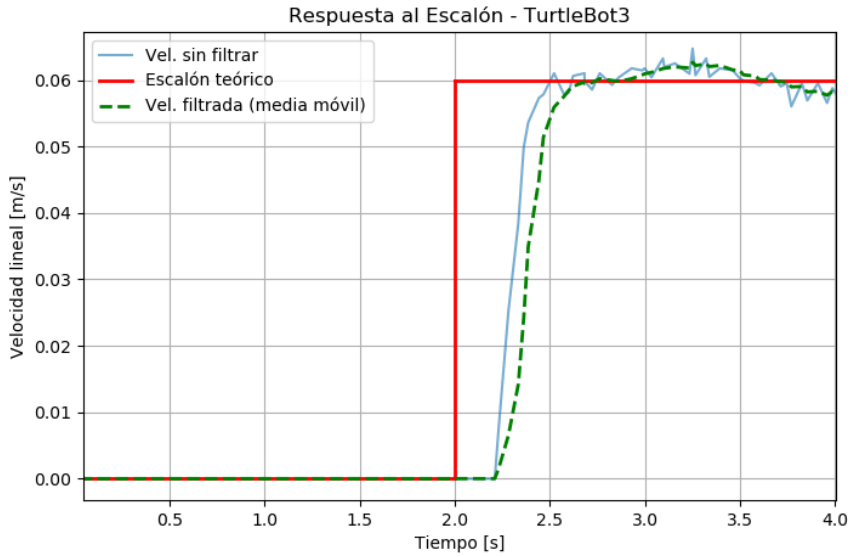

Figura 10.24 Escalón de 0.06 m/s de velocidad lineal.

De esta gráfica se obtiene que:

$$t_0 = 2$$
$$y_0 = 0$$
$$\tau_d + t_0 = 2.211$$
$$t_{rp} = 2.53$$
$$y_{rp} = 0.06$$

Realizando los cálculos de manera análoga al anterior.

$$K = \frac{0.06 - 0}{0.06 - 0} = 1$$
$$y_{0.63} = 0.0378$$
$$\tau = 2.332 - 2 = 0.332$$
$$G(s)_{0.06} = \frac{1}{0.332s + 1} e^{-0.211s}$$

Escalón a 0.09 m/s:

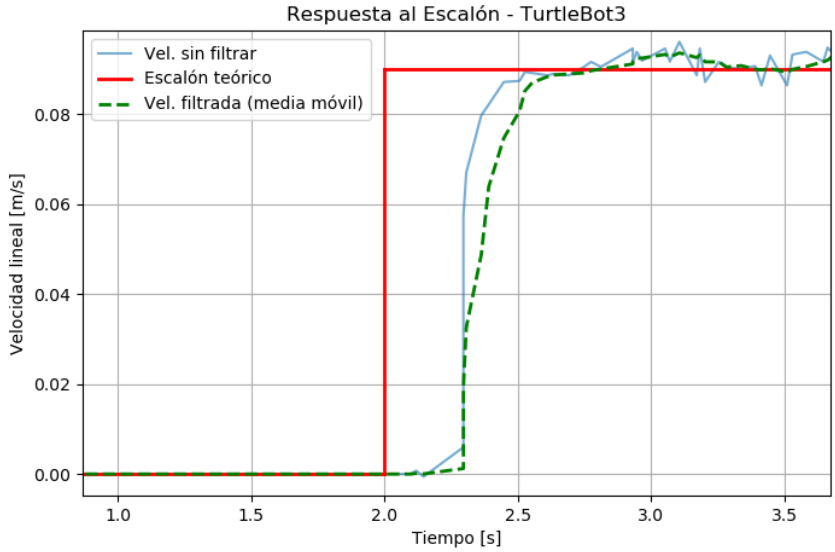

Figura 10.25 Escalón de 0.09 m/s de velocidad lineal.

De esta gráfica se obtiene que:

$$t_0 = 2$$
$$y_0 = 0$$
$$\tau_d + t_0 = 2.15$$
$$t_{rp} = 2.7$$
$$y_{rp} = 0.09$$

Realizando los cálculos de manera análoga.

$$K = \frac{0.09 - 0}{0.09 - 0} = 1$$

$$y_{0.63} = 0.0567$$

$$\tau = 2.296 - 2 = 0.296$$

$$G(s)_{0.09} = \frac{1}{0.296s + 1} e^{-0.15s}$$

Escalón a 0.12 m/s:

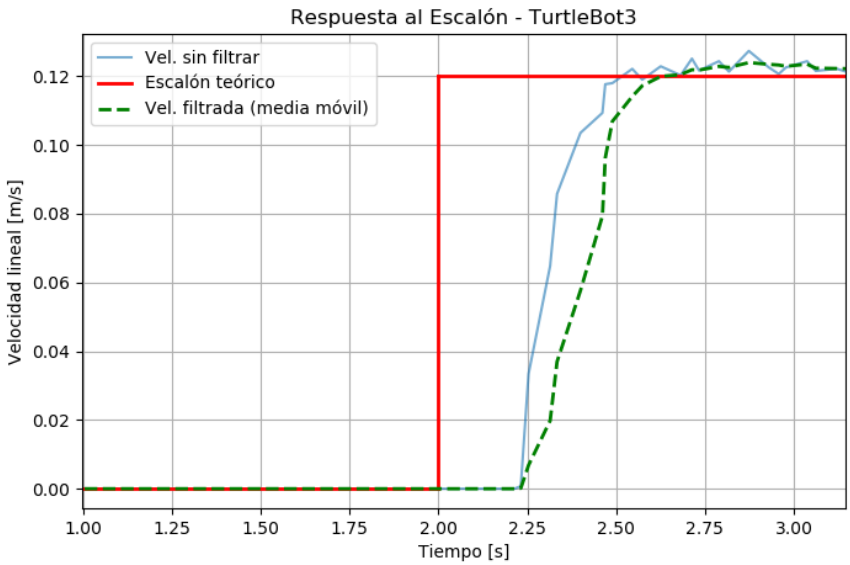

Figura 10.26 Escalón de 0.12 m/s de velocidad lineal.

De esta gráfica se obtiene que:

$$t_0 = 2$$

$$y_0 = 0$$

$$\tau_d + t_0 = 2.213$$

$$t_{rp} = 2.55$$

$$y_{rp} = 0.12$$

Realizando los cálculos de manera análoga.

$$K = \frac{0.12 - 0}{0.12 - 0} = 1$$

$$y_{0.63} = 0.0756$$

$$\tau = 2.324 - 2 = 0.324$$

$$G(s)_{0.12} = \frac{1}{0.324s + 1} e^{-0.213s}$$

Escalón a 0.15 m/s:

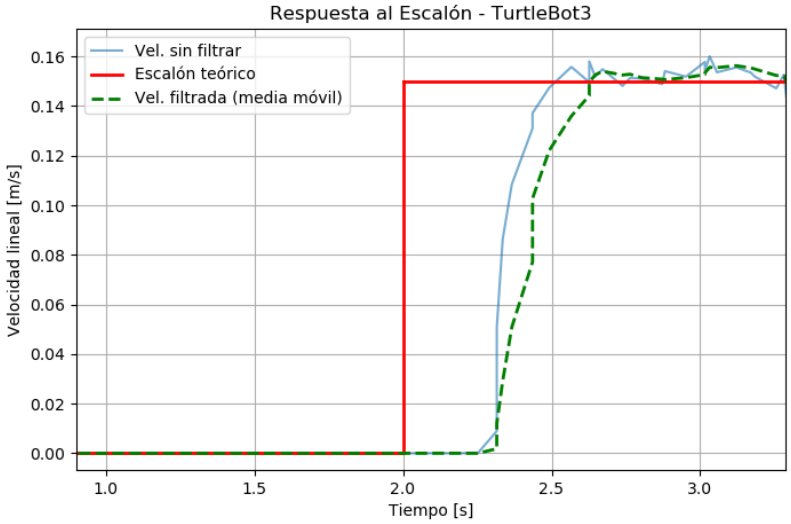

Figura 10.27 Escalón de 0.15 m/s de velocidad lineal.

De esta gráfica se obtiene que:

$$t_0 = 2$$

$$y_0 = 0$$

$$\tau_d + t_0 = 2.251$$

$$t_{rp} = 2.57$$

$$y_{rp} = 0.15$$

Realizando los cálculos de manera análoga.

$$K = \frac{0.15 - 0}{0.15 - 0} = 1$$

$$y_{0.63} = 0.0945$$

$$\tau = 2.347 - 2 = 0.347$$

$$G(s)_{0.15} = \frac{1}{0.347s + 1} e^{-0.251s}$$

Escalón a 0.18 m/s:

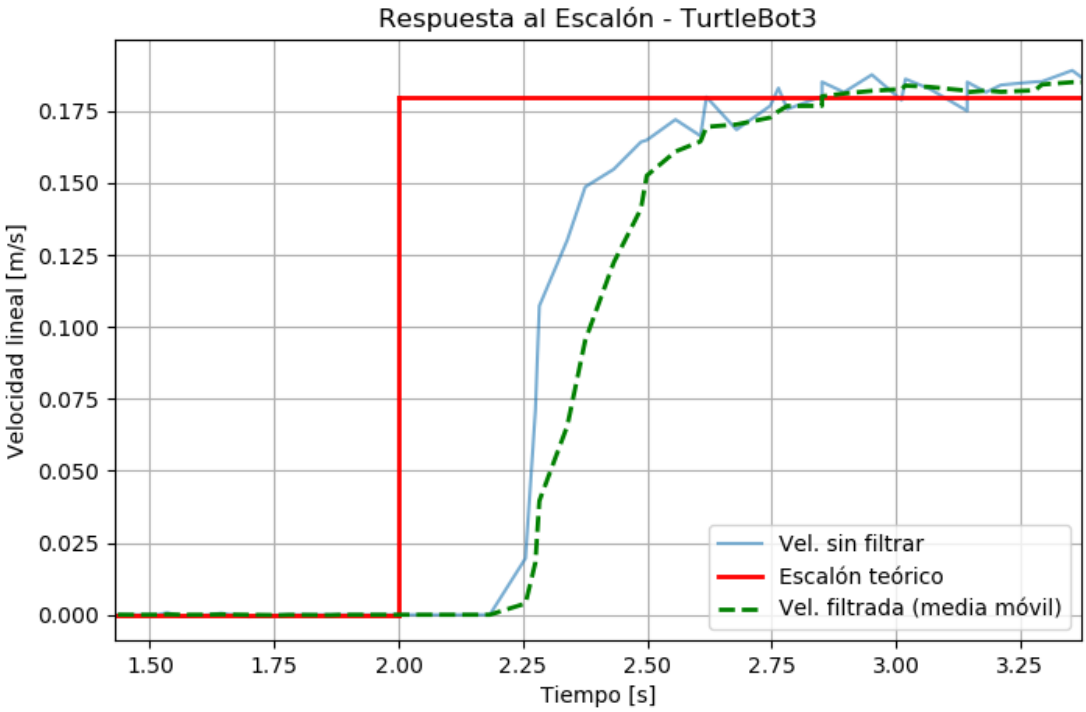

Figura 10.28 Escalón de 0.18 m/s de velocidad lineal.

De esta gráfica se obtiene que:

$$t_0 = 2$$
$$y_0 = 0$$
$$\tau_d + t_0 = 2.183$$
$$t_{rp} = 2.77$$
$$y_{rp} = 0.18$$

Realizando los cálculos de manera análoga.

$$K = \frac{0.18 - 0}{0.18 - 0} = 1$$
$$y_{0.63} = 0.1134$$
$$\tau = 2.298 - 2 = 0.298$$
$$G(s)_{0.18} = \frac{1}{0.298s + 1} e^{-0.183s}$$

Con esto ya se tienen todas las muestras para obtener una función de transferencia total. Esta función se calculará haciendo la media de la ganancia (K), constante de tiempo (τ) y retraso (τ_d) de todas las muestras anteriores. Los resultados se presentan en la siguiente tabla:

Valores función transferencia lineal			
Vel_lin (m/s)	K	tau	tau_d
0.03	1	0.35	0.207
0.06	1	0.332	0.211
0.09	1	0.296	0.15
0.12	1	0.324	0.213
0.15	1	0.347	0.251
0.18	1	0.298	0.183
MEDIA	1	0.3245	0.2025

Tabla 10.1 Tabla resumen de las funciones de transferencia lineales.

Con esto la función de transferencia que describe el movimiento lineal queda:

$$G(s)_{lin} = \frac{1}{0.3245s + 1} e^{-0.2025s}$$

10.4.2 Obtención de la función de transferencia del movimiento angular

El proceso para obtener la función de transferencia que define el movimiento angular será parecido al usado para el movimiento lineal. Tras diferentes experimentos la velocidad angular máxima que el robot puede desarrollar se puede estimar en 2.3 rad/s. Partiendo del reposo, se pondrá un escalón de amplitud 0.5, 1.0, 1.5, 2.0 y 2.3 rad/s a los 2 segundos y se observará la respuesta. Como en las gráficas del movimiento lineal la entrada será la línea roja, la azul la velocidad angular de la respuesta y la verde discontinua el filtro de medias móviles para facilitar la obtención del instante en que se alcanza el régimen permanente.

Como los cálculos son bastante parecidos a los del movimiento lineal utilizando el "método del 63%" para estimar la función de primer orden no se detallarán en exceso.

Escalón a 0.5 rad/s:

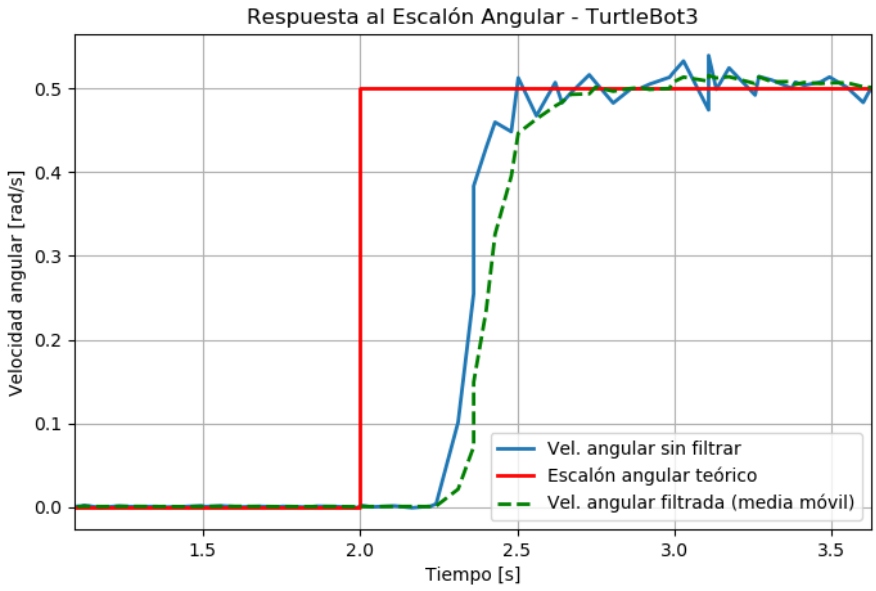

Figura 10.29 Escalón de 0.5 rad/s de velocidad angular.

De esta gráfica se obtiene que:

$$t_0 = 2$$
$$y_0 = 0$$
$$\tau_d + t_0 = 2.20$$
$$t_{rp} = 2.50$$
$$y_{rp} = 0.5$$

Realizando los cálculos de los parámetros de la función de transferencia:

$$K = \frac{0.5 - 0}{0.5 - 0} = 1$$
$$y_{0.63} = 0.315$$
$$\tau = 2.361 - 2 = 0.361$$
$$G(s)_{0.5} = \frac{1}{0.361s + 1} e^{-0.20s}$$

Escalón a 1.0 rad/s:

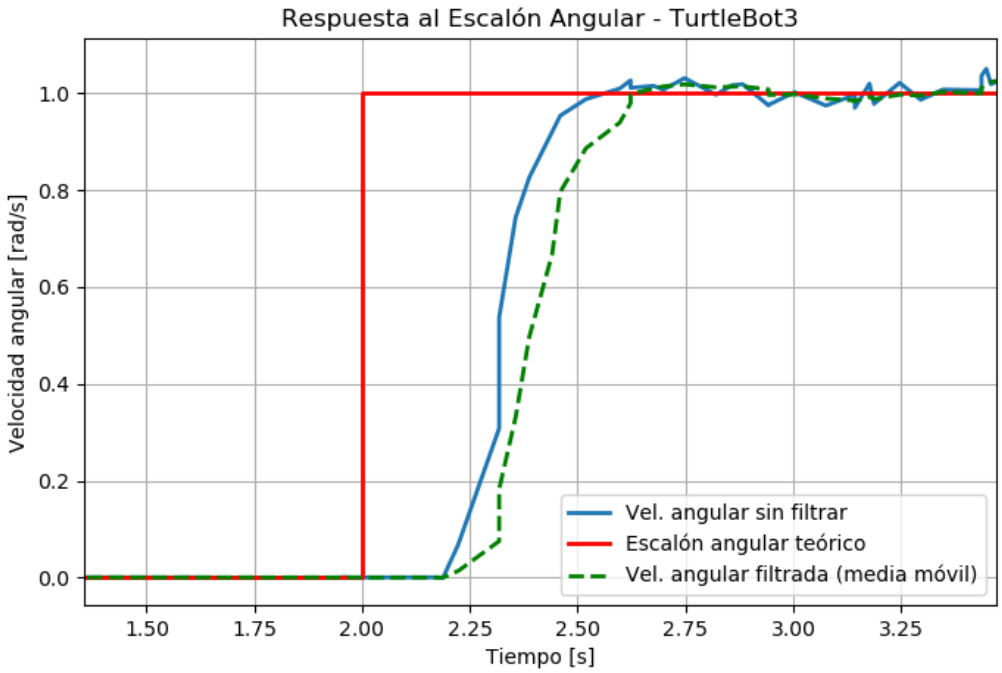

Figura 10.30 Escalón de 1.0 rad/s de velocidad angular.

De esta gráfica se obtiene que:

$$t_0 = 2$$
$$y_0 = 0$$
$$\tau_d + t_0 = 2.187$$
$$t_{rp} = 2.562$$

$$y_{rp} = 1.0$$

Realizando los cálculos de manera análoga al anterior:

$$K = \frac{1.0 - 0}{1.0 - 0} = 1$$
$$y_{0.63} = 0.63$$
$$\tau = 2.335 - 2 = 0.335$$
$$G(s)_{1.0} = \frac{1}{0.335s + 1} e^{-0.187s}$$

Escalón a 1.5 rad/s:

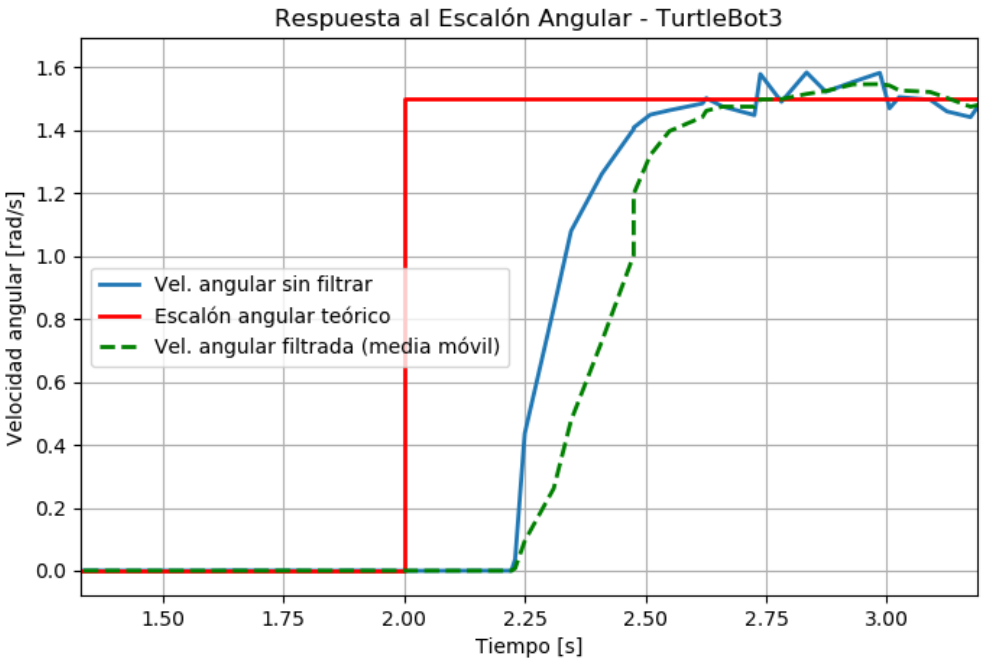

Figura 10.31 Escalón de 1.5 rad/s de velocidad angular.

De esta gráfica se obtiene que:

$$t_0 = 2$$
$$y_0 = 0$$
$$\tau_d + t_0 = 2.222$$
$$t_{rp} = 2.625$$
$$y_{rp} = 1.5$$

Realizando los cálculos de manera análoga:

$$K = \frac{1.5 - 0}{1.5 - 0} = 1$$

$$y_{0.63} = 0.945$$

$$\tau = 2.326 - 2 = 0.326$$

$$G(s)_{1.5} = \frac{1}{0.326s + 1} e^{-0.222s}$$

Escalón a 2.0 rad/s:

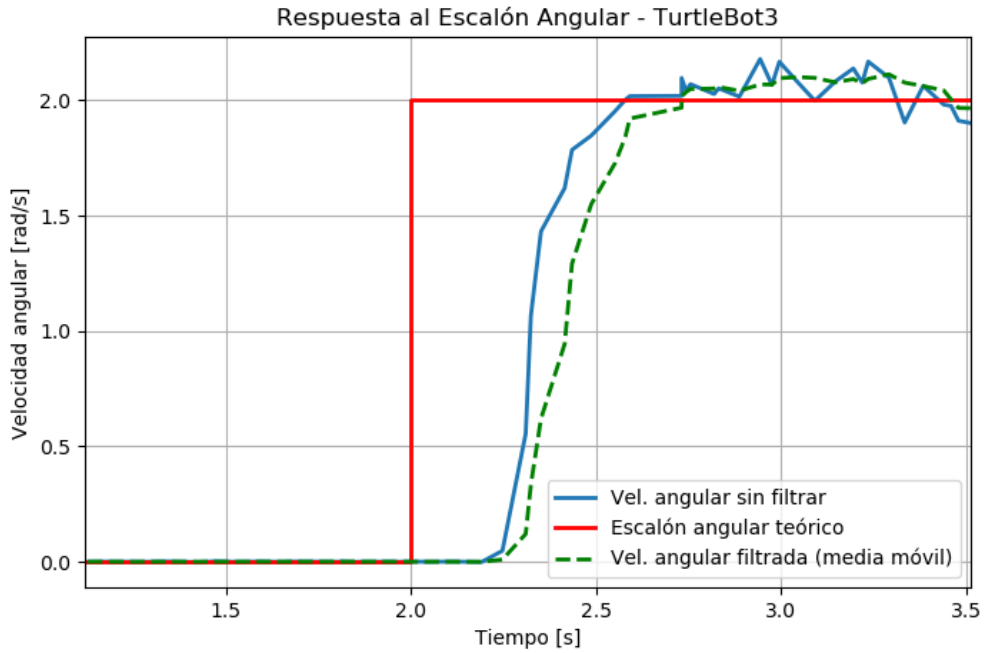

Figura 10.32 Escalón de 2.0 rad/s de velocidad angular.

De esta gráfica se obtiene que:

$$t_0 = 2$$

$$y_0 = 0$$

$$\tau_d + t_0 = 2.191$$

$$t_{rp} = 2.59$$

$$y_{rp} = 2.0$$

Realizando los cálculos de manera análoga al anterior:

$$K = \frac{2.0 - 0}{2.0 - 0} = 1$$

$$y_{0.63} = 1.26$$

$$\tau = 2.338 - 2 = 0.338$$

$$G(s)_{2.0} = \frac{1}{0.338s + 1} e^{-0.191s}$$

Escalón a 2.3 rad/s:

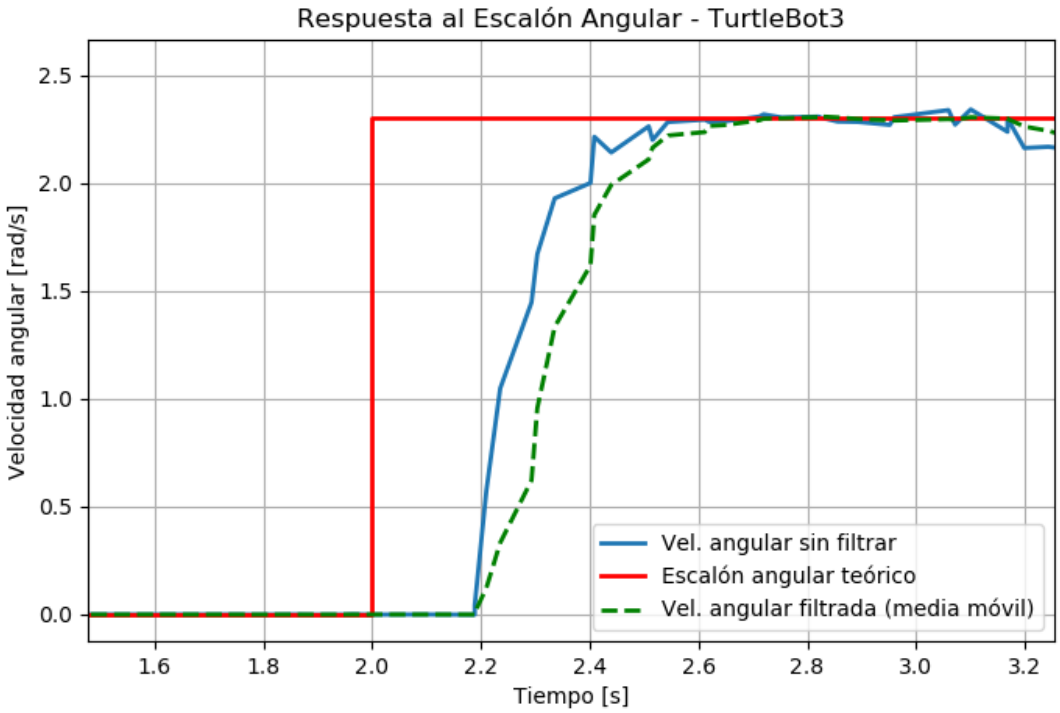

Figura 10.33 Escalón de 2.3 rad/s de velocidad angular.

De esta gráfica se obtiene que:

$$t_0 = 2$$
$$y_0 = 0$$
$$\tau_d + t_0 = 2.189$$
$$t_{rp} = 2.69$$
$$y_{rp} = 2.3$$

Realizando los cálculos de manera análoga al anterior:

$$K = \frac{2.3 - 0}{2.3 - 0} = 1$$
$$y_{0.63} = 1.449$$
$$\tau = 2.293 - 2 = 0.293$$
$$G(s)_{2.3} = \frac{1}{0.293s + 1} e^{-0.189s}$$

Ya con todas las muestras de diferentes amplitudes tomadas, se hace la media de K, τ y τ_d para obtener la función de transferencia final del movimiento angular.

Valores función de transferencia angular			
Vel_ang (rad/s)	K	tau	tau_d
0.5	1	0.361	0.2
1	1	0.335	0.187
1.5	1	0.326	0.222
2	1	0.338	0.191
2.3	1	0.293	0.189
MEDIA	1	0.3306	0.1978

Tabla 10.2 Tabla resumen de las funciones de transferencia angulares.

Finalmente, la función de transferencia que define el movimiento angular del robot queda:

$$G(s)_{ang} = \frac{1}{0.3306s + 1} e^{-0.1978s}$$

Ya con las funciones de transferencia tanto para el movimiento lineal como para el angular se puede diseñar el controlador.

10.4.3 Método 1: Control PI por cancelación de polos despreciando retrasos en bucle cerrado.

Para el primer método se usará un criterio bastante simple, la cancelación de polos, el objetivo es cancelar el polo del sistema, como su propio nombre indica, de manera que se pueda mejorar la respuesta. Su gran desventaja es la respuesta ante retrasos considerables, por ello se va a optar en omitirlos para poder probarlo. El despreciar los retrasos ya de por sí nos hace prever que la respuesta buscada no será del todo exacta.

Se va a realizar en bucle cerrado dado que la entrada de los controladores a diseñar son error en posición y ángulo, es decir, existe realimentación.

Controlador para el movimiento lineal:

Partiendo de la función de transferencia del movimiento lineal ya calculada y despreciando el retraso.

$$G(s)_{lin} = \frac{1}{0.3245s + 1} e^{-0.2025s} \rightarrow G'(s)_{lin} = \frac{1}{0.3245s + 1}$$

El controlador tendrá la forma:

$$C(s) = K_c * (1 + \frac{1}{T_i s}) \tag{10.4}$$

Se introduce un cero en: $s = -\frac{1}{\tau} = \frac{1}{0.3245} = -3.082$

Se impone que $T_i = \tau = 0.3245$

Se calcula la función en bucle abierto:

$$G(s)_{lin_{ab}} = C(s) * G'(s)_{lin} = Kc * \left(1 + \frac{1}{T_i s}\right) * \frac{1}{0.3245s + 1} = K_c * \left(1 + \frac{1}{0.3245s}\right) * \frac{1}{0.3245s + 1}$$

$$G(s)_{lin_{ab}} = K_c * \left(\frac{0.3245s + 1}{0.3245s}\right) * \frac{1}{0.3245s + 1}$$

$$G(s)_{lin_{ab}} = \frac{K_c}{0.3245s}$$

Ahora se obtiene la función en bucle cerrado a partir de la de bucle abierto.

$$G(s)_{lin_{bc}} = \frac{G(s)_{lin_{ab}}}{1 + G(s)_{lin_{ab}}} = \frac{\frac{K_c}{0.3245s}}{1 + \frac{K_c}{0.3245s}} = \frac{\frac{K_c}{0.3245s}}{\frac{0.3245s + K_c}{0.3245s}}$$

$$G(s)_{lin_{bc}} = \frac{K_c}{0.3245s + K_c} = \frac{1}{\frac{0.3245s}{Kc} + 1}$$

Con esto la constante de tiempo queda definida como: $\tau_s = \frac{0.3245}{Kc}$

Como condición se pondrá un valor aceptable como que el sistema sea el doble de rápido.

$$\tau_s = \frac{\tau}{2} \rightarrow \frac{0.3245}{K_c} = \frac{0.3245}{2}$$

$$K_c = 2$$

Con ello ya se tienen todos los parámetros del PI:

$$K_c = 2$$

$$T_i = 0.3245$$

$$K_i = \frac{K_c}{T_i} = \frac{2}{0.3245} = 6.613$$

Controlador para el movimiento angular:

Los cálculos son análogos a los del controlador lineal.

Partiendo de la función de transferencia que describe el movimiento angular ya obtenida y despreciando el retraso:

$$G(s)_{ang} = \frac{1}{0.3306s + 1} e^{-0.1975s} \rightarrow G'(s)_{ang} = \frac{1}{0.3306s + 1}$$

El controlador PI de la forma:

$$C(s) = K_c * (1 + \frac{1}{T_i s}) \tag{10.5}$$

Existe un cero en: $s = -\frac{1}{\tau} = \frac{1}{0.3306} = -3.025$

$T_i = \tau = 0.3306$

La función en bucle abierto sería:

$$G(s)_{ang_{ab}} = C(s) * G'(s)_{ang} = Kc * \left(1 + \frac{1}{T_i s}\right) * \frac{1}{0.3306s + 1} = K_c * \left(1 + \frac{1}{0.3306s}\right) * \frac{1}{0.3306s + 1}$$

$$G(s)_{ang_{ab}} = K_c * \left(\frac{0.3306s + 1}{0.3306s}\right) * \frac{1}{0.3306s + 1}$$

$$G(s)_{ang_{ab}} = \frac{K_c}{0.3306s}$$

Dinámica en bucle cerrado:

$$G(s)_{ang_{bc}} = \frac{G(s)_{ang_{ab}}}{1 + G(s)_{ang_{ab}}} = \frac{\frac{K_c}{0.3306s}}{1 + \frac{K_c}{0.3306s}} = \frac{\frac{K_c}{0.3306s}}{\frac{0.3306s + K_c}{0.3306s}}$$

$$G(s)_{ang_{bc}} = \frac{K_c}{0.3306s + K_c} = \frac{1}{\frac{0.3306s}{Kc} + 1}$$

La constante de tiempo es:

$$\tau_s = \frac{0.3306}{Kc}$$

Al igual que con el PI lineal la condición será que el sistema sea el doble de rápido:

$$\tau_s = \frac{\tau}{2} \rightarrow \frac{0.3306}{K_c} = \frac{0.3306}{2}$$

$$K_c = 2$$

Con ello ya se tienen todos los parámetros del PI:

$$K_c = 2$$

$$T_i = 0.3306$$

$$K_i = \frac{K_c}{T_i} = \frac{2}{0.3306} = 6.05$$

Una vez obtenidos los valores tanto para el PI lineal como del angular se procede a su experimentación. Como se observa en la Figura 10.34, se introdujeron los valores calculados al programa.

```
--- Configuración de Parámetros PID ---

Controlador Lineal:
   Kp=2.0, Ki=6.613, Kd=0.0
   integrator_limit=None
   output_min=None, output_max=None
   deadband=0.0

Controlador Angular:
   Kp=2.0, Ki=6.05, Kd=0.0
   integrator_limit=None
   output_min=None, output_max=None
   deadband=0.0
```

Figura 10.34 Valores del PI por cancelación de polos introducidos.

Se estableció como objetivo que el robot se dirigiera al marcador 9, pero cuando comenzó el movimiento, el robot no hacía más que dar vueltas siguiendo una trayectoria circular. Al cabo de un rato se abortó el experimento viendo que el robot no era capaz de encontrar el camino.

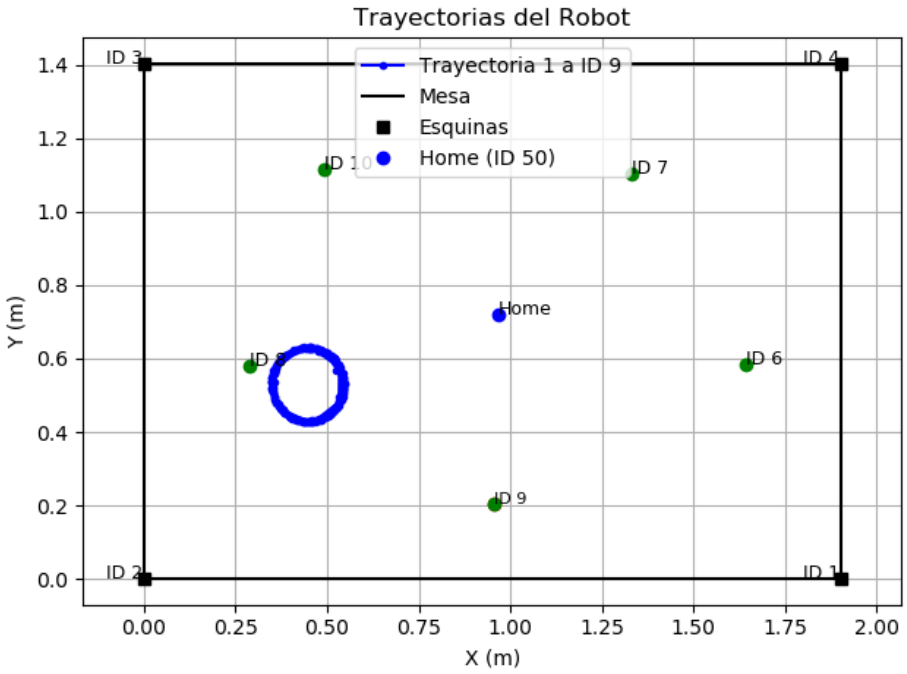

Figura 10.35 Trayectoria seguida por el robot con PI por cancelación de polos.

Observando los errores en la Figura 10.36, al estar describiendo un círculo, el error en distancia presenta una forma de onda senoidal al pasar siempre por los mismos puntos, mientras que el error en ángulo no es capaz de encontrar el punto donde se anula.

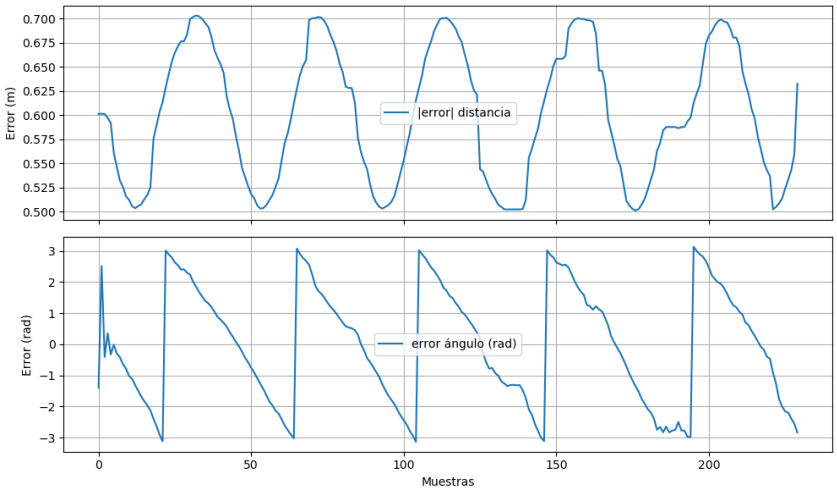

Figura 10.36 Error con PI por cancelación de polos.

Con estos resultados se puede decir que los controladores diseñados por cancelación de polos omitiendo los retrasos no han funcionado.

10.4.4 Método 2: Control por Ziegler-Nichols en bucle abierto.

La primera idea era realizar el Ziegler-Nichols en bucle cerrado, pero resultaba imposible el cálculo de la K crítica debido a las limitaciones del hardware del robot, dado que daba igual que constante proporcional se le pusiera, nunca alcanzaba las oscilaciones mantenidas al intentar llegar la posición objetivo, por ello se diseñará el control con Ziegler-Nichols en bucle abierto. Además del PID también se harán experimentos para los controladores P y PI.

Para el cálculo de los parámetros será necesaria la siguiente tabla:

Controlador	K_c	T_i	T_d
P	$\frac{\tau}{K * \tau_d}$	∞	0
PI	$0.9 * \frac{\tau}{K * \tau_d}$	$\frac{\tau_d}{0.3}$	0
PID	$1.2 * \frac{\tau}{K * \tau_d}$	$2 * \tau_d$	$0.5 * \tau_d$

Tabla 10.3 Tabla de valores para control por Z-N en bucle abierto.

Control para el movimiento lineal:

Como siempre, se parte de la función de transferencia.

$$G(s)_{lin} = \frac{1}{0.3245s + 1} e^{-0.2025s}$$

Para que el desarrollo del Z-N sea eficaz, hay que modificar la constante de tiempo de manera que:

$$\tau' = \tau - \tau_d = 0.3245 - 0.2025 = 0.122$$

Para facilitar las ecuaciones se escribirá τ' como τ.

$$G''(s)_{lin} = \frac{1}{0.122s + 1} e^{-0.2025s}$$

Con esto se tiene que:

$$K = 1$$
$$\tau = 0.122$$
$$\tau_d = 0.2025$$

Con esto ya se pueden diseñar los controladores:

Controlador P:

$$K_c = \frac{\tau}{K * \tau_d} = \frac{0.122}{1 * 0.2025} = 0.602$$
$$T_i = \infty \rightarrow \mathrm{K_i} \approx 0$$
$$T_d = K_d = 0$$

Controlador PI:

$$K_c = 0.9 * \frac{\tau}{K * \tau_d} = 0.9 * \frac{0.122}{1 * 0.2025} = 0.542$$
$$T_i = \frac{\tau_d}{0.3} = \frac{0.2025}{0.3} = 0.675 \rightarrow \mathrm{K_i} = \frac{K_c}{T_i} = \frac{0.542}{0.675} = 0.803$$

$$T_d = 0$$

Controlador PID:

$$K_c = 1.2 * \frac{\tau}{K * \tau_d} = 1.2 * \frac{0.122}{1 * 0.2025} = 0.723$$
$$T_i = 2 * \tau_d = 2 * 0.2025 = 0.405 \rightarrow \mathrm{K_i} = \frac{K_c}{T_i} = \frac{0.723}{0.405} = 1.785$$

$$T_d = 0.5 * \tau_d = 0.5 * 0.2025 = 0.10125 \rightarrow K_d = K_c * T_d = 0.723 * 0.10125 = 0.0732$$

Control para el movimiento angular:

Partiendo de la función de transferencia para el movimiento angular:

$$G(s)_{ang} = \frac{1}{0.3306s + 1} e^{-0.1975s}$$

Se modifica otra vez la constante de tiempo.

$$\tau' = \tau - \tau_d = 0.3306 - 0.1975 = 0.1331$$

Para tomará τ' como τ.

$$G''(s)_{ang} = \frac{1}{0.1331s + 1} e^{-0.1975s}$$

$$K = 1$$

$$\tau = 0.1331$$

$$\tau_d = 0.1975$$

Los controladores serán:

Controlador P:

$$K_c = \frac{\tau}{K * \tau_d} = \frac{0.1331}{1 * 0.1975} = 0.674$$

$$T_i = \infty \rightarrow \mathrm{K_i} \approx 0$$

$$T_d = K_d = 0$$

Controlador PI:

$$K_c = 0.9 * \frac{\tau}{K * \tau_d} = 0.9 * \frac{0.1331}{1 * 0.1975} = 0.6065$$

$$T_i = \frac{\tau_d}{0.3} = \frac{0.1975}{0.3} = 0.658 \rightarrow \mathrm{K_i} = \frac{K_c}{T_i} = \frac{0.6065}{0.658} = 0.923$$

$$T_d = 0$$

Controlador PID:

$$K_c = 1.2 * \frac{\tau}{K * \tau_d} = 1.2 * \frac{0.1331}{1 * 0.1975} = 0.809$$

$$T_i = 2 * \tau_d = 2 * 0.1975 = 0.395 \rightarrow \mathrm{K_i} = \frac{K_c}{T_i} = \frac{0.809}{0.395} = 2.048$$

$$T_d = 0.5 * \tau_d = 0.5 * 0.1975 = 0.09875 \rightarrow K_d = K_c * T_d = 0.809 * 0.09875 = 0.0799$$

Ya con todos los valores calculados se muestran todos los valores en la siguiente tabla a modo de resumen.

	Z-N bucle abierto					
	Lineal			Angular		
	Kc	Ki	Kd	Kc	Ki	Kd
P	0.602	0	0	0.674	0	0
PI	0.542	0.803	0	0.6075	0.923	0
PID	0.723	1.785	0.0732	0.809	2.048	0.0799

Tabla 10.4 Valores de los controladores por Z-N bucle abierto.

A continuación, se detallan los experimentos para los controles P, PI y PID.

Experimento del control proporcional (P):

```
Controlador Lineal:
   Kp=0.602, Ki=0.0, Kd=0.0
   integrator_limit=None
   output_min=None, output_max=None
   deadband=0.0

Controlador Angular:
   Kp=0.674, Ki=0.0, Kd=0.0
   integrator_limit=None
   output_min=None, output_max=None
   deadband=0.0
```

Figura 10.37 Valores para el control P por Z-N.

En este experimento, partiendo de la posición (0.289, 0.381), el robot se dirige a la posición del marcador 6, alcanzándola tras 7.75 segundos y situándose en la posición (1.551, 0.572). Tras este movimiento se decide hacer otro en el cual la meta es el marcador 9, partiendo de prácticamente la posición final del anterior (varía algún centímetro por el desfase minúsculo de cuando el robot comanda el detenimiento por llegar a la meta y se para realmente), en este movimiento se observa que el robot abandona los márgenes al principio lo que hace que se dirija a la posición home y tras entrar en los márgenes se encamina a la meta realizando un ligero arco, siendo la posición final (1.047, 0.208). Tras ello finaliza el movimiento.

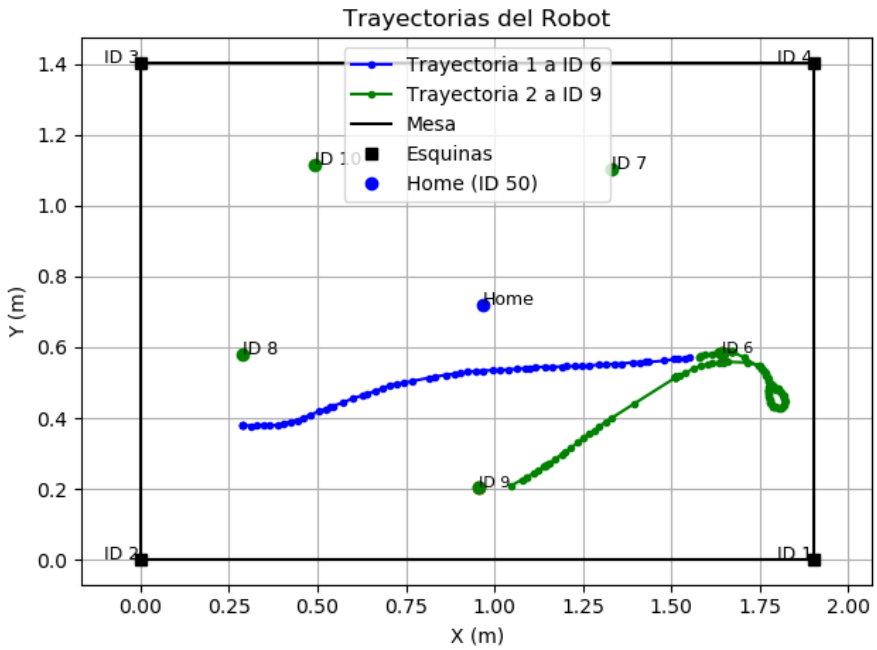

Figura 10.38 Trayectorias para el control P Z-N.

Mirando la Figura 10.38 se aprecia una trayectoria bastante lineal para el primer objetivo, para el segundo se ve como el robot da la vuelta y primero se acerca a home. Una vez está dentro de los márgenes se dirige al marcador

9.

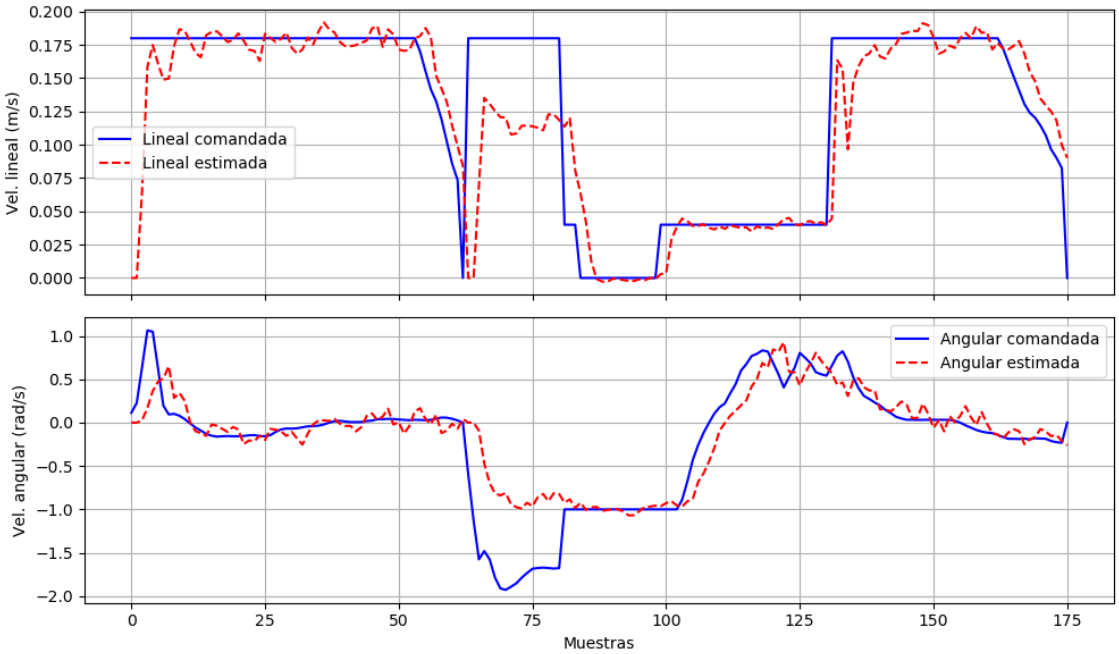

Figura 10.39 Registro de velocidades para el control P Z-N.

Observando los errores el movimiento entre el marcador 6 y el 9 en la Figura 10.40 se aprecia como la distancia hasta el objetivo disminuye ligeramente, aunque realmente se esté alejando de la meta, esto es debido a que el registro de errores toma como objetivo la posición home, una vez entra dentro de los márgenes y el robot se dirige hacia el destino final, el error en distancia disminuye hasta entrar dentro de la tolerancia.

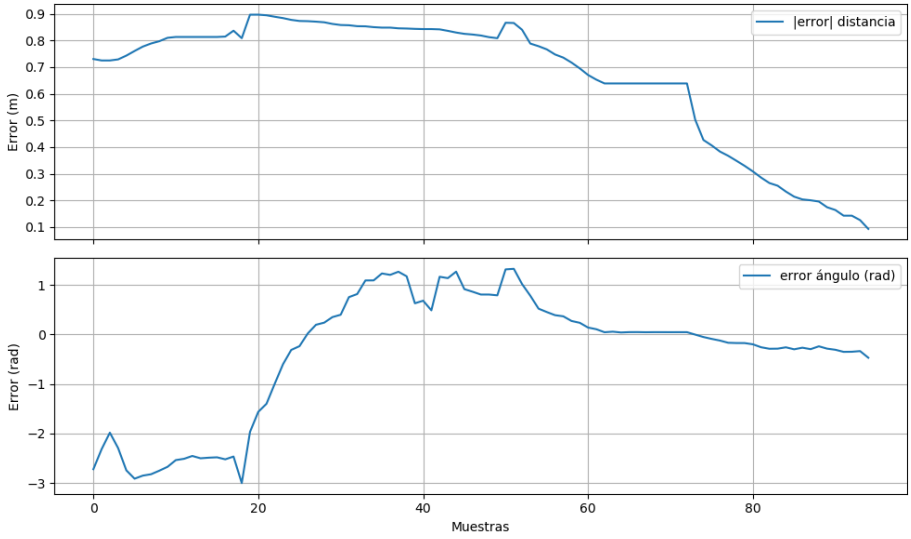

Figura 10.40 Errores entre marcador 6 y 9 para el control P Z-N.

Viendo las velocidades desarrolladas se observa que son bastante estables exceptuando que saturan tanto la velocidad lineal como angular cuando se mueven simultáneamente.

En resumen, el control P implementado funciona correctamente.

Experimento para el control PI:

```
--- Configuración de Parámetros PID ---

Controlador Lineal:
   Kp=0.542, Ki=0.803, Kd=0.0
   integrator_limit=None
   output_min=None, output_max=None
   deadband=0.0

Controlador Angular:
   Kp=0.6075, Ki=0.923, Kd=0.0
   integrator_limit=None
   output_min=None, output_max=None
   deadband=0.0
```

Figura 10.41 Valores PI Z-N.

Los valores de las constantes proporcionales tanto para el control lineal como para el angular son parecidos a las del control proporcional, pero se añade la acción integral.

Partiendo de una posición cercana al marcador 8 se indica como objetivo el marcador 9, pero el robot se pone a dar vuelvas en círculos en un bucle infinito como se muestra en la Figura 10.42, por lo que se tiene que interrumpir el movimiento.

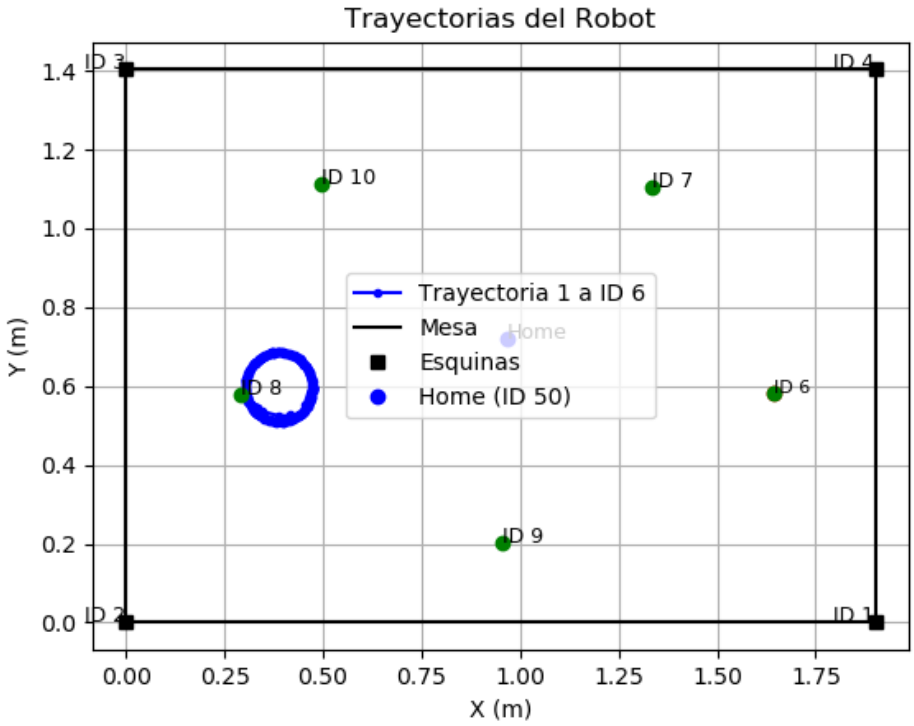

Figura 10.42 Trayectoria seguida PI Z-N.

Como esta situación no ocurría cuando los controladores solo tenían parte proporcional es bastante posible que sea debido a que la acción integral introducida es demasiado elevada. También cabe recordar que esto pasaba

con el control PI por cancelación de polos, el cual también tenía parte integral y que el error angular no disminuía.

Experimento con control PID:

```
--- Configuración de Parámetros PID ---

Controlador Lineal:
   Kp=0.723, Ki=1.785, Kd=0.0732
   integrator_limit=None
   output_min=None, output_max=None
   deadband=0.0

Controlador Angular:
   Kp=0.809, Ki=2.048, Kd=0.0799
   integrator_limit=None
   output_min=None, output_max=None
   deadband=0.0
```

Figura 10.43 Valores implementados PID Z-N.

Ahora se harán las pruebas para los controles PID diseñados con Z-N en bucle abierto. La parte integral crece considerablemente respecto al PI anteriormente analizado, por lo que es bastante posible que, aunque se introduzca el término derivativo, siga siendo excesivamente elevado.

Para el experimento se indica que el robot se dirija a la posición ID 6, pero como se podía prever el robot se queda dando vueltas en bucle infinito como en el control PI. En la Figura 10.44 se aprecia en rosa la trayectoria circular seguida.

Figura 10.44 Trayectoria PID Z-N.

Como se aprecia en la Figura 10.45 el error en distancia es periódico al pasar una y otra vez por los mismos puntos y el error en ángulo no se puede controlar, se observa una forma de onda de dientes de sierra debido a que gira sin parar en el mismo sentido.

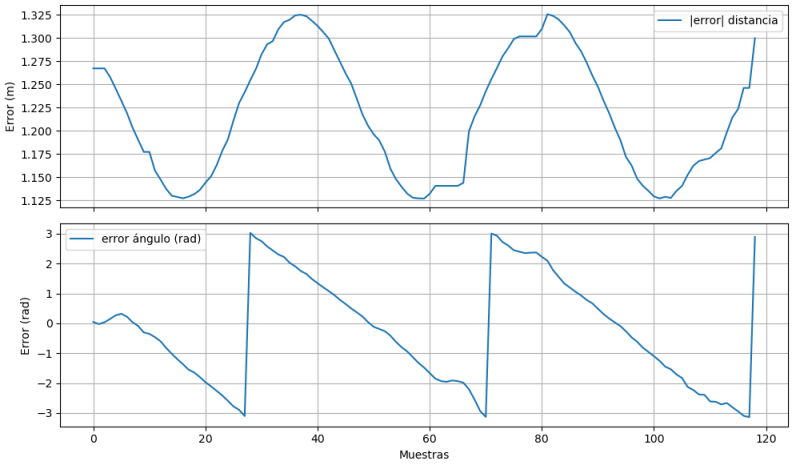

Figura 10.45 Errores PID Z-N.

10.4.5 Método 3: Control por Ziegler-Nichols con ajuste fino.

Para este método se partirá del PID calculado por Ziegler-Nichols en bucle abierto del apartado anterior y se irán modificando levemente los parámetros para que, por experimentación, conseguir que el sistema funcione.

Como se vio, el PID realizado por Z-N no funcionaba bien principalmente por el término integral, dado que el control proporcional si funcionaba, primero se va a introducir un valor límite al error del integrador (anti-wind up). Se pondrá 0.05 para el lineal y 0.5 para el término angular.

Experimento 1 PID con ajuste fino:

```
--- Configuración de Parámetros PID ---

Controlador Lineal:
   Kp=0.723, Ki=1.785, Kd=0.0732
   integrator_limit=0.05
   output_min=None, output_max=None
   deadband=0.0

Controlador Angular:
   Kp=0.809, Ki=2.048, Kd=0.0799
   integrator_limit=0.5
   output_min=None, output_max=None
   deadband=0.0
```

Figura 10.46 Valores para experimento 1 PID A-F.

Partiendo de la posición (0.383, 0.301) se introduce como objetivo el ID 6. El robot avanza describiendo una trayectoria de *"eses"* muy pronunciadas, pero se consigue alcanzar la meta en la posición (1.661, 0.504) tras 12.14 segundos.

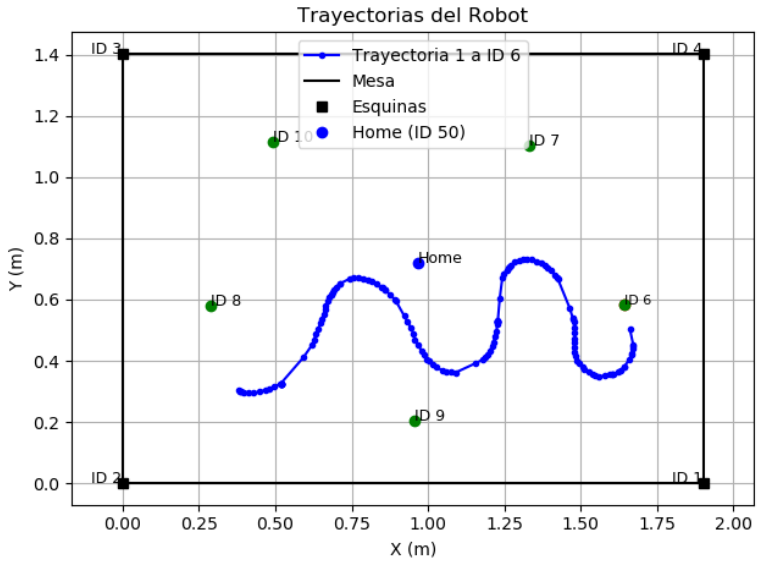

Figura 10.47 Trayectoria seguida PID A-F 1.

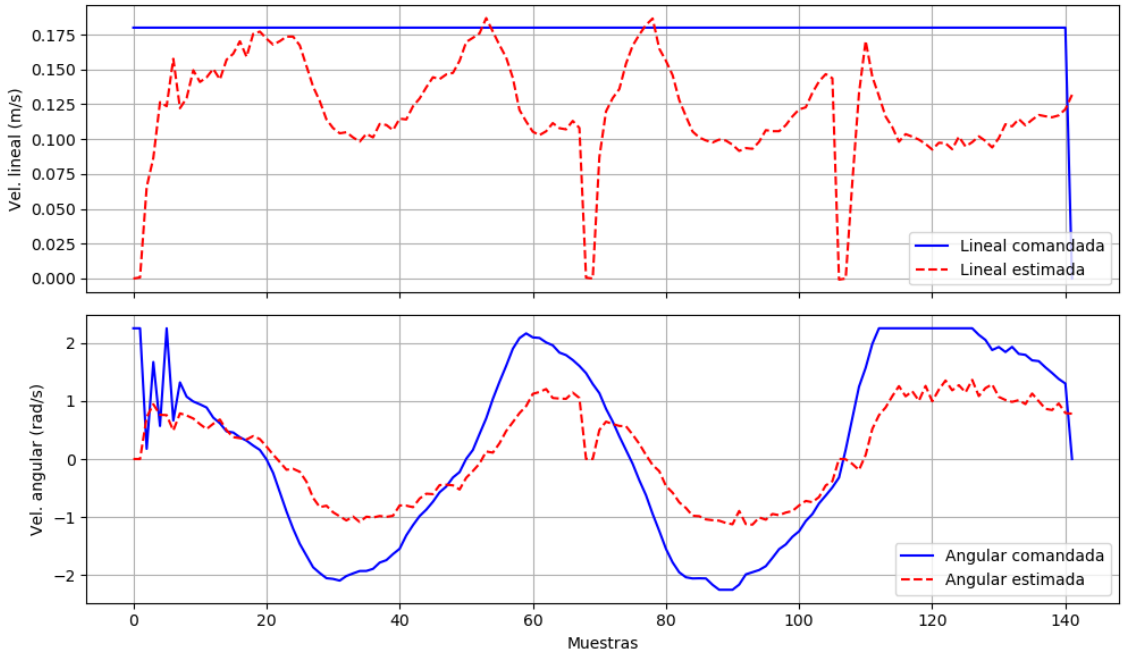

Figura 10.48 Registro de velocidades PID A-F 1.

En la Figura 10.48, se observa sobreoscilación en la velocidad angular debido a que el integrador es demasiado grande.

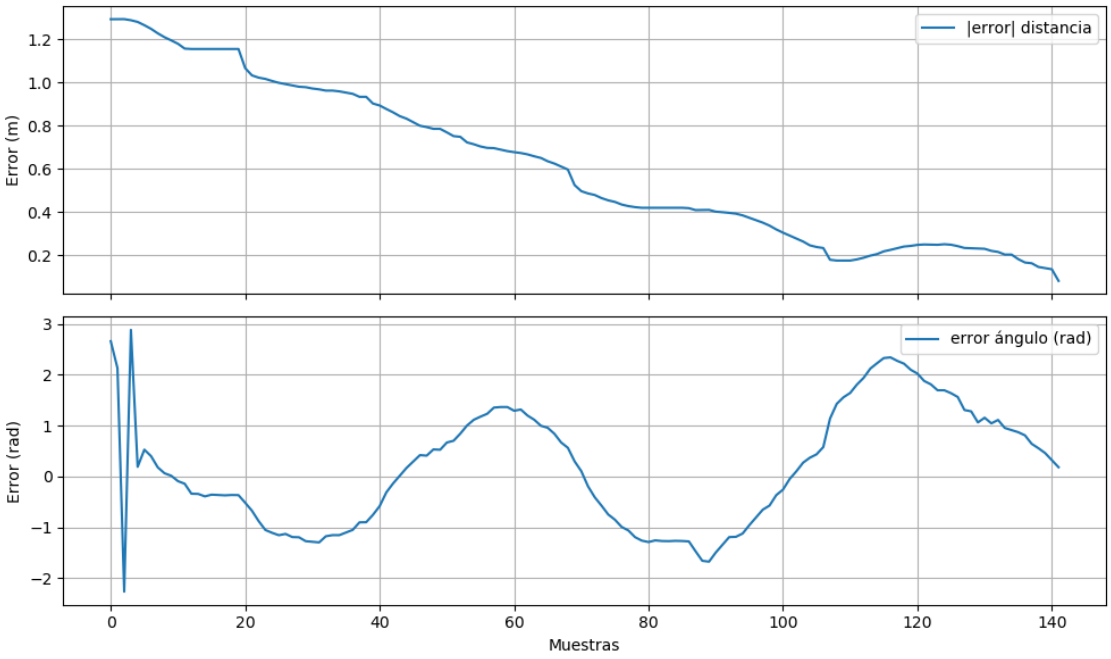

Figura 10.49 Errores PID A-F 1.

En cuanto al error en ángulo también se observa una onda senoidal debido a que el robot no es capaz de estabilizarse con error angular nulo.

La mejora del control es considerable, pero el robot sigue presentando una especie de zig-zag debido a que el termino integral sigue siendo alto incluso con los límites impuestos, por lo cual, para el siguiente experimento se reducirá el término integral. En cuanto al proporcional y derivativo parecen que están en unos valores adecuados.

Experimento 2 PID con ajuste fino:

Teniendo en cuenta lo visto en el experimento anterior se decidió probar a bajar el K_i lineal a 0.02 y el K_i angular a 0.03 para ver si la respuesta mejoraba.

```
--- Configuración de Parámetros PID ---

Controlador Lineal:
   Kp=0.723, Ki=0.02, Kd=0.0732
   integrator_limit=None
   output_min=None, output_max=None
   deadband=0.0

Controlador Angular:
   Kp=0.809, Ki=0.03, Kd=0.0799
   integrator_limit=None
   output_min=None, output_max=None
   deadband=0.0
```

Figura 10.50 Valores para experimento 2 PID A-F.

Partiendo de la posición (0.292, 0.519), se seleccionó como destino el punto (1.34, 0.68), el robot consiguió llegar a la meta en 6.89 segundos describiendo una trayectoria arqueada. Tras esto se hizo otra prueba en la que

el robot debía dirigirse al punto (0.90, 0.36) pero se observó que, al acercarse al punto, el robot empezó a describir una trayectoria en espiral acercándose poco a poco en cada vuelta hasta entrar dentro de la tolerancia del objetivo. En estos giros el robot perdió mucho tiempo, lo que hizo que tardara 32.04 segundos en alcanzar la meta.

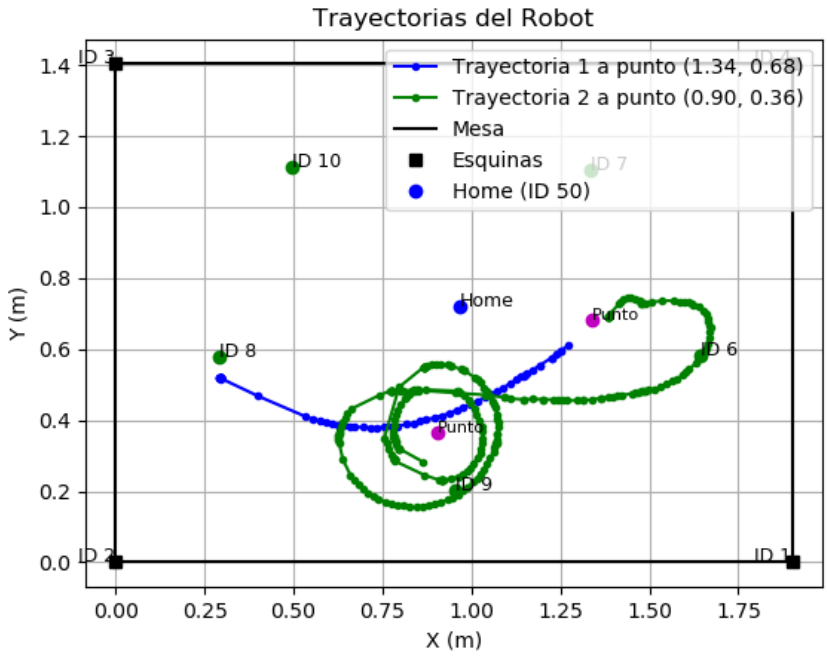

Figura 10.51 Trayectorias seguidas PID A-F 2.

Es posible que esta forma *"espiral"* se debiera a la acumulación del error integral cuando satura, por ello sería conveniente activar el anti-windup en el siguiente experimento.

Experimento 3 PID con ajuste fino:

Para esta prueba, los valores del PID son los mismos que en el experimento anterior, pero se puso un límite de integración de 0.02 para el controlador lineal y de 0.03 para el controlador angular.

```
--- Configuración de Parámetros PID ---

Controlador Lineal:
   Kp=0.723, Ki=0.02, Kd=0.0732
   integrator_limit=0.2
   output_min=None, output_max=None
   deadband=0.0

Controlador Angular:
   Kp=0.809, Ki=0.03, Kd=0.0799
   integrator_limit=0.3
   output_min=None, output_max=None
   deadband=0.0
```

Figura 10.52 Valores para experimento 3 PID A-F.

El robot parte de la coordenada (0.269, 0.440) y tiene como objetivo el punto (0.66, 0.94), tras hacer un arco, el robot alcanza la meta tras 4.65 segundos. Posteriormente, se introduce como objetivo la coordenada (1.50, 0.76), destino que se alcanza en un movimiento semi arqueado, tras 7.70 segundos. Finalmente se establece como meta el marcador 9, llegando el robot a este tras 6.99 segundos en una trayectoria arqueada.

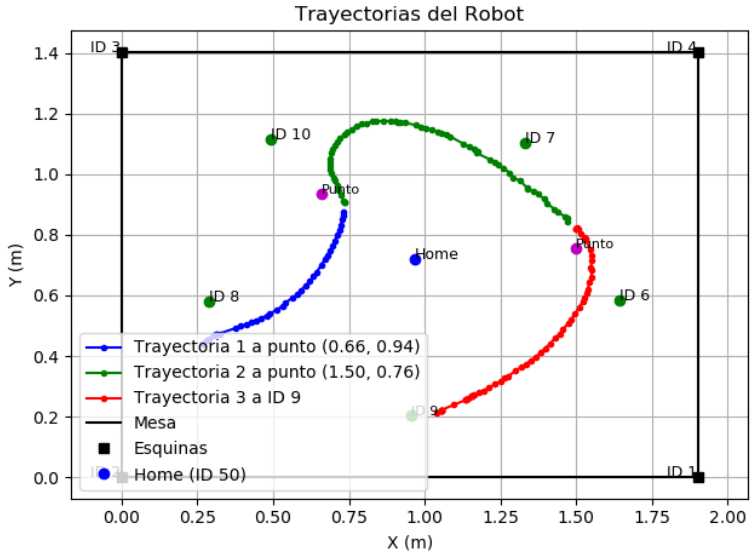

Figura 10.53 Trayectorias seguidas PID A-F 3.

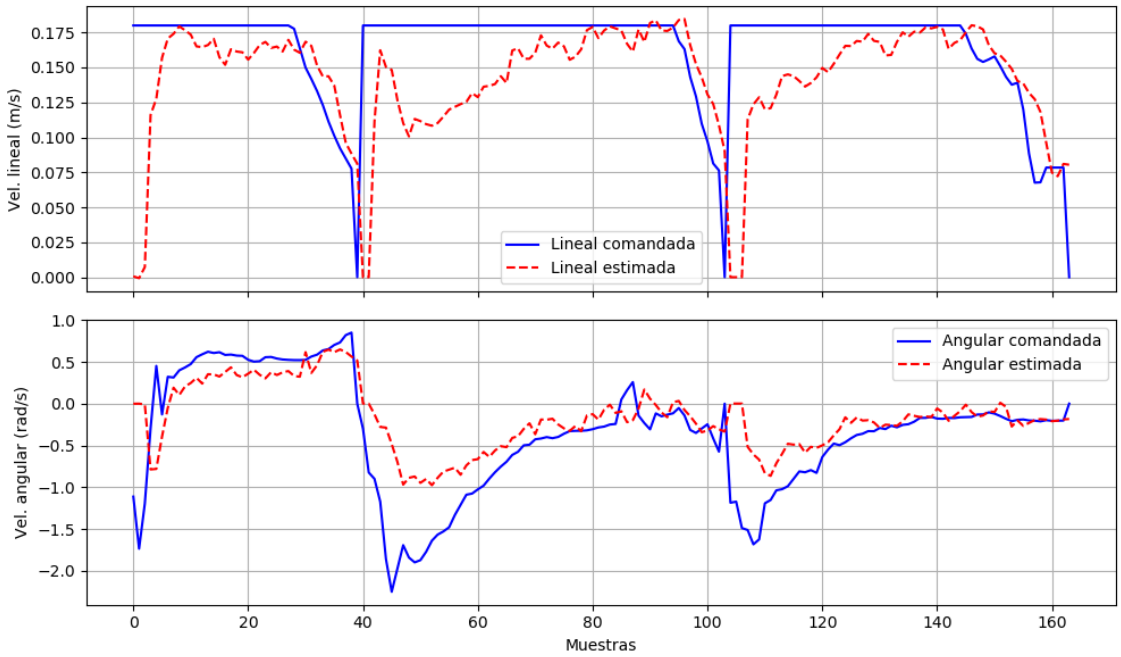

Figura 10.54 Registro de velocidades PID A-F 3.

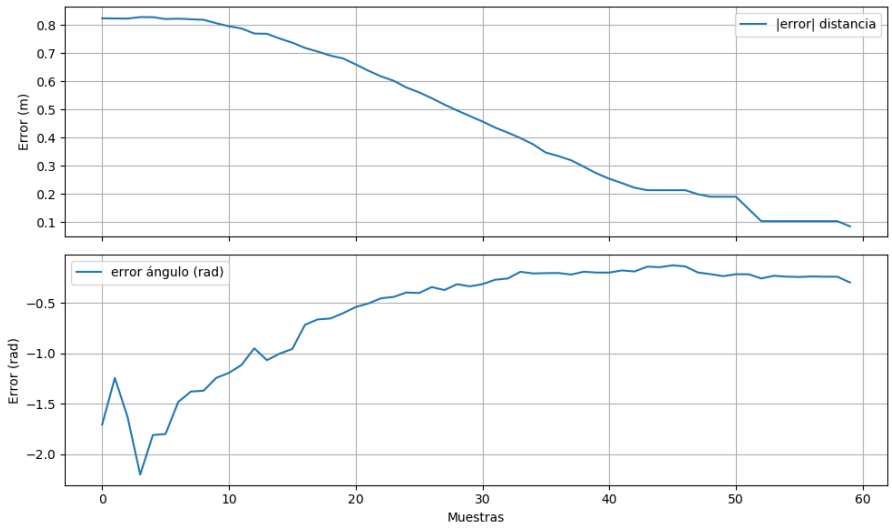

Figura 10.55 Errores entre punto (1.50, 0.76) y marcador 9 para PID A-F 3.

Con estos valores de PID se obtienen unas velocidades angulares suaves, es cierto que el comportamiento lineal sí es más brusco, pero de todas formas es bastante bueno. Las trayectorias son bastante óptimas, a lo mejor se podría hacer que el arco descrito fuera menos pronunciado, pero igualmente se consigue llegar al objetivo en poco tiempo. Se puede concluir por experiencia que la introducción del anti-windup ha permitido que las velocidades y trayectorias sean mucho más precisas.

Con esta experimentación se decidieron dar por válidos los valores del PID impuestos.

10.5. Guiado y movimiento del robot en modo automático modificando opciones.

Una vez con la sintonización del PID ya realizada, se harán experimentos modificando el resto de los parámetros de configuración.

10.5.1 Modificación 1: Velocidades máximas

En los experimentos del PID se observó que la velocidad del robot máxima de 0.18 m/s puede ser demasiado elevada al salir del margen hasta que se detecta que está fuera de estos, podría poner en peligro la integridad del robot. Además, se apreció que cuando se realizaban simultáneamente los movimientos de avance y giro, tanto la velocidad lineal como la angular saturaban a valores más bajos.

Viendo las gráficas de los experimentos anteriores se concluyó que las velocidades simultáneas máximas eran 0.075 m/s para el movimiento lineal y 1 rad/s para el angular. Estas se impusieron como velocidades máximas, además se rebajó el término integral del PID angular a 0.02 y el límite de integración angular del PID a 0.2 para que no aparecieran zigzagueos al haber reducido la velocidad máxima.

La velocidad angular máxima fuera de los márgenes también tuvo que ser reducida pasando de 1 rad/s a 0.7 rad/s.

En el experimento realizado con estas modificaciones, se partía de la coordenada (0.349, 0.355) y se quería que el robot fuera al marcador 6. Se observó una trayectoria bastante lineal con alguna que otra corrección, tras llegar al objetivo se ordenó que el robot fuera al marcador 9, el robot dio un giro bastante cerrado para cambiar de dirección y posteriormente, se apreció una trayectoria ligeramente zigzagueante, es cierto que podría ser debido a que los valores de la orientación del robot almacenados no estaban actualizados con los nuevos valores al ser este el primer experimento.

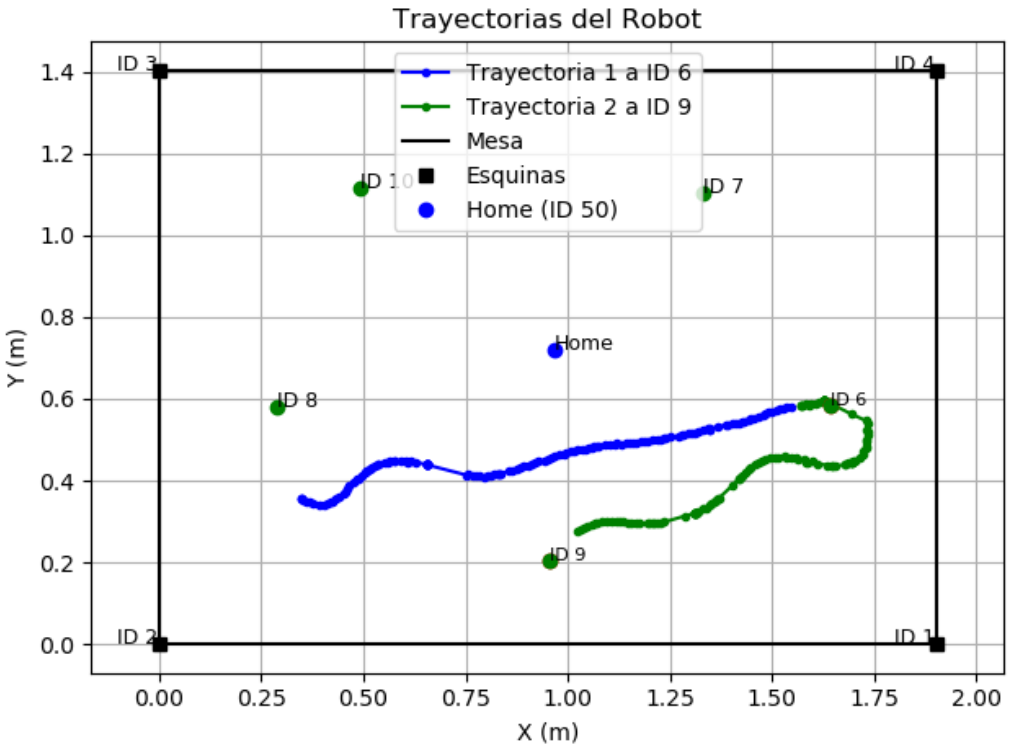

Figura 10.56 Trayectorias de la modificación 1.

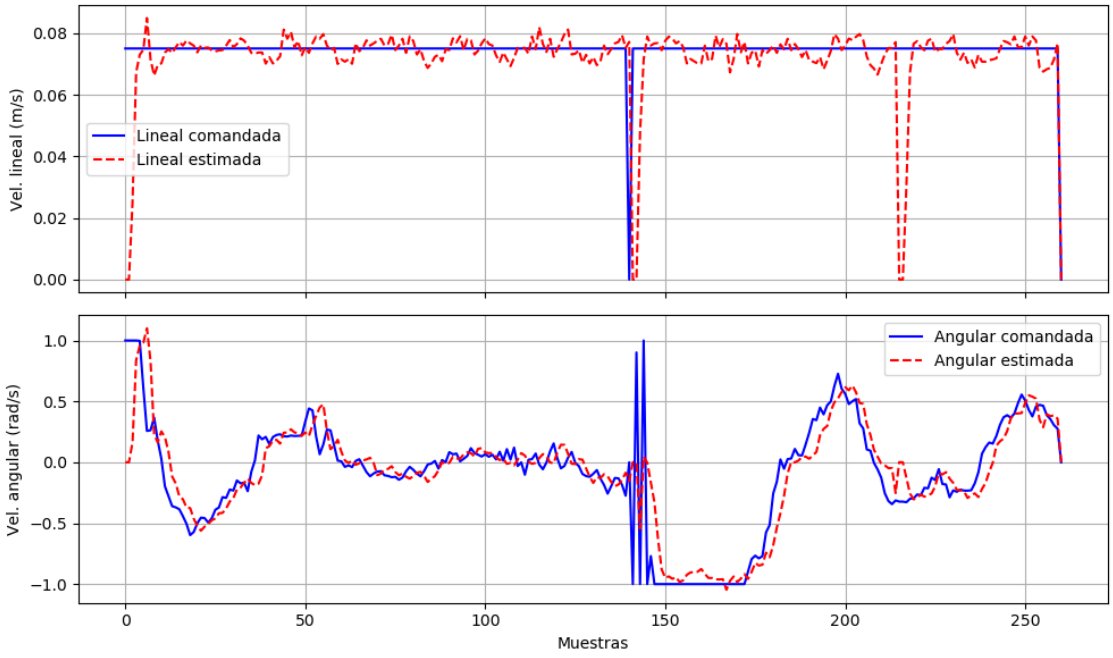

Figura 10.57 Registro de velocidades de la modificación 1.

Las velocidades se estiman bastante parecidas a las comandadas, teniendo en cuenta que el robot físicamente no puede alcanzar la velocidad comandada al momento y se produce un pequeño retraso. En la velocidad lineal se observan picos en los que el robot se para un instante, pero es debido a pequeños fallos de hardware.

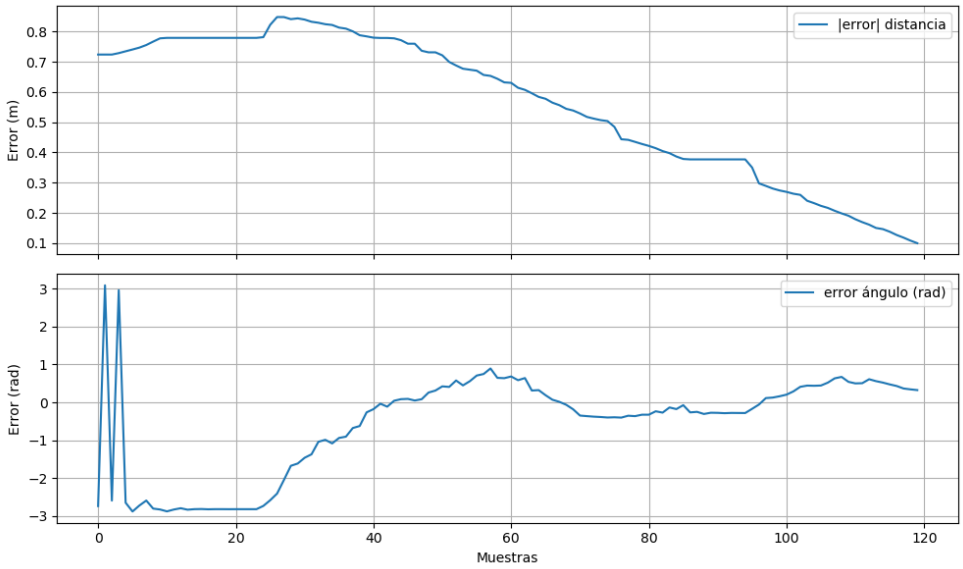

Figura 10.58 Errores entre marcador 6 y 9 en la modificación 1.

En la Figura 10.58, se observa como al principio, para dirigirse al marcador 9, el robot se aleja del objetivo debido a que está dando el giro para situarse en la dirección correcta.

10.5.2 Modificación 2: Tolerancias

En este apartado se estudiará como influye la modificación de las tolerancias de la distancia aceptable a la que se puede quedar el robot de la meta.

Para la primera prueba se pondrá que el robot se dirija a uno de los nuevos marcadores detectados, pero la tolerancia de este pasará de 0.1 m a 0.01 m.

Partiendo de una posición cercana al marcador 8 y teniendo como objetivo el marcador 6, se vio que cuando el robot estaba cerca de este, no era capaz de encontrarlo y se puso a dar vueltas entorno a él.

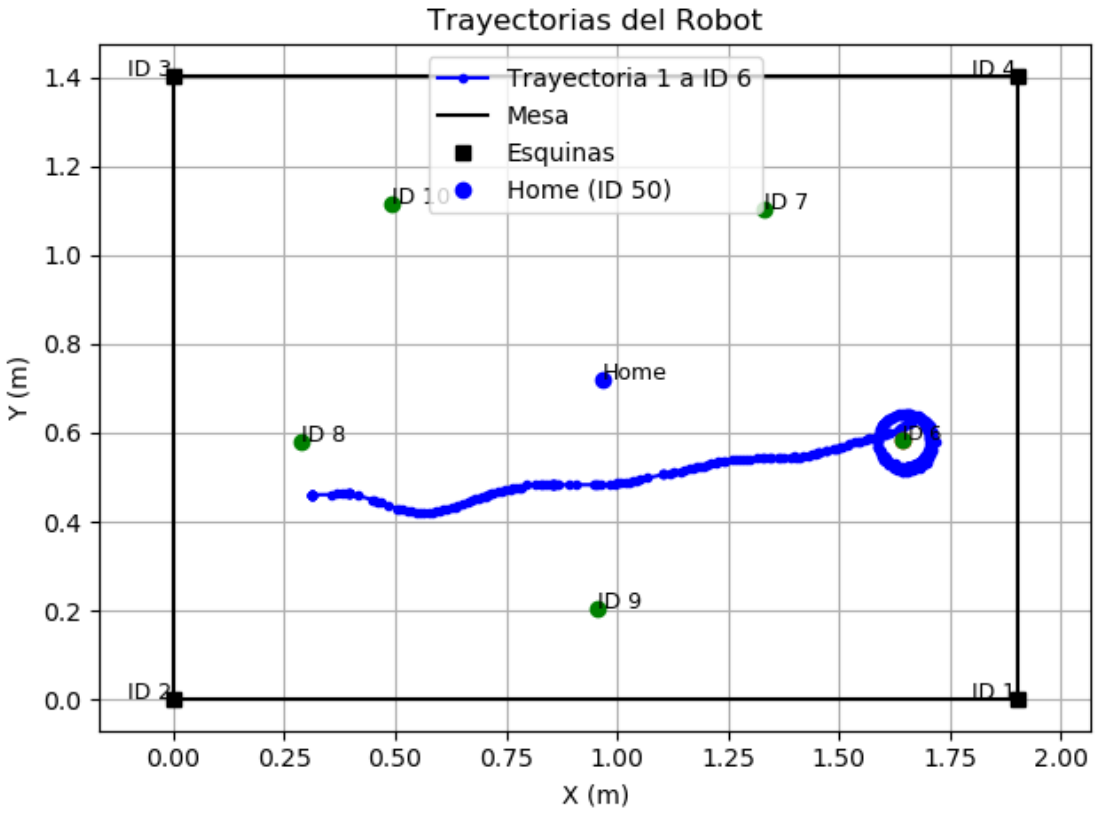

Figura 10.59 Trayectoria de la modificación 2, tolerancia 0.01.

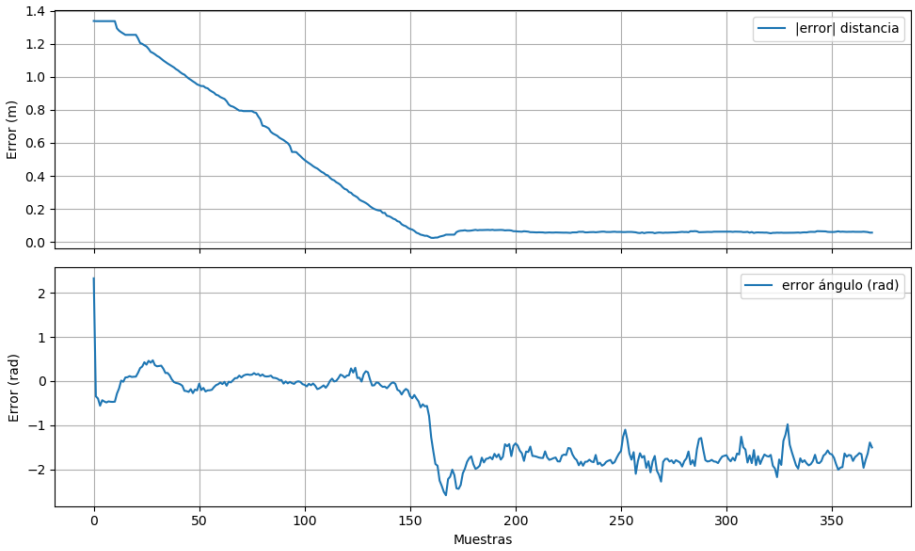

Figura 10.60 Errores en la modificación 2, tolerancia 0.01.

Se observa en la Figura 10.60 como cuando el robot se encuentra 0.05 m de distancia no disminuye más debido a que el robot se encuentra dando vueltas en círculos.

Habrá que subir la tolerancia. Tras diferentes observaciones se probará con una tolerancia de 0.07 m que parece el valor a partir del cual el programa es capaz de detectar que el robot ha llegado al punto objetivo.

Ya con la tolerancia de 0.07 metros introducida, se realiza una prueba partiendo de la posición (0.227, 0.380) y

el objetivo es el marcador 6. En la Figura 10.61 se observa que la posición se alcanza correctamente y en la Figura 10.62 que el error en distancia si es capaz de alcanzar este valor sin problema, por lo que esta tolerancia es bastante válida, además 7 centímetros no es una distancia muy grande respecto al marcador. La tolerancia para los marcadores existentes teniendo en cuenta que aquí se engloban las esquinas será la misma que la distancia del margen externo, 0.15 m.

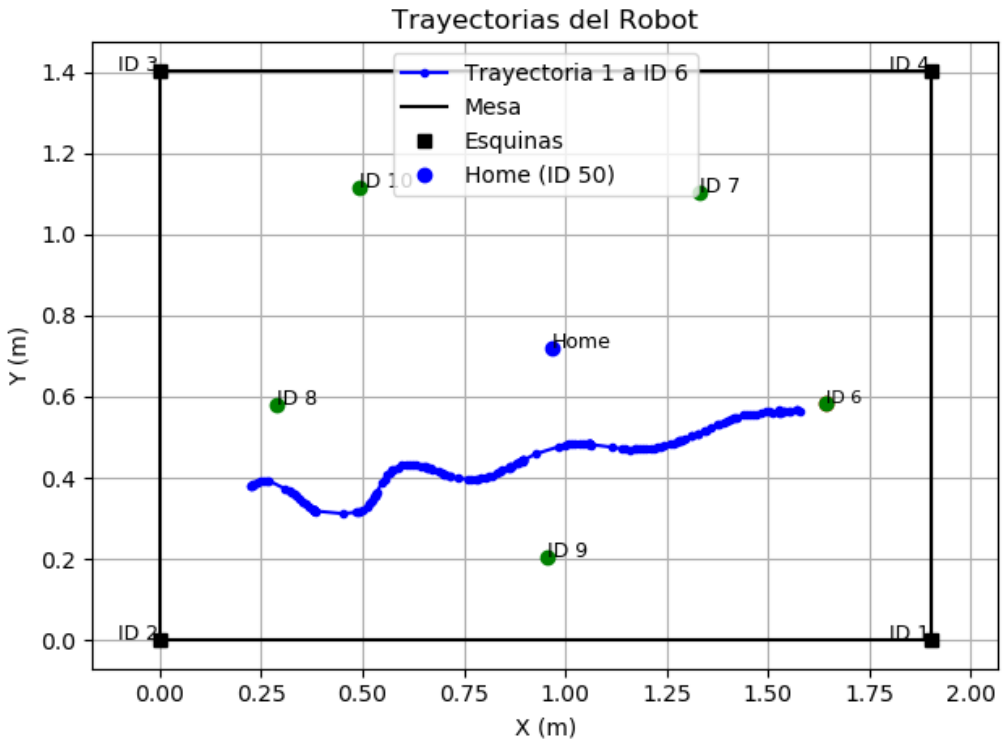

Figura 10.61 Trayectoria de la modificación 2, tolerancia 0.07.

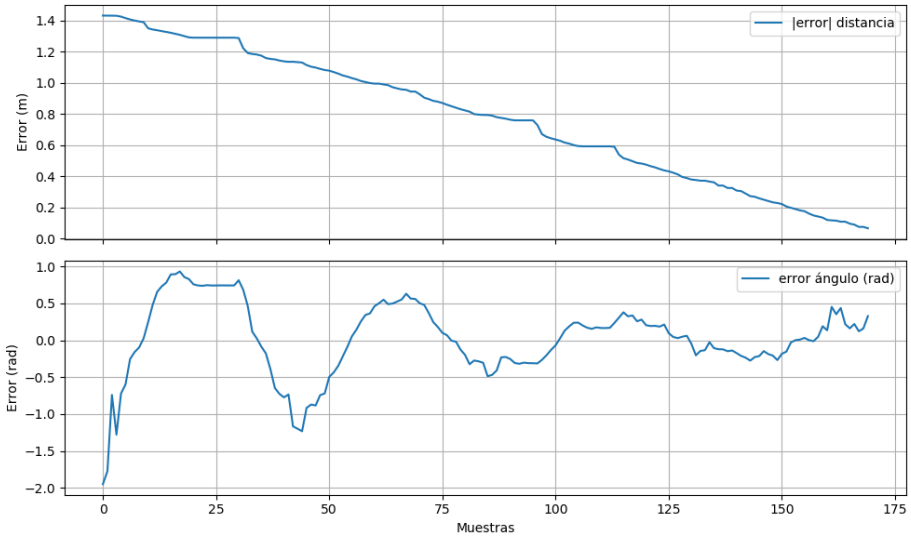

Figura 10.62 Errores en la modificación 2, tolerancia 0.07.

10.5.3 Modificación 3: Márgenes interno y externo.

15 centímetros de margen externo es un valor que, por experiencia, es bastante correcto, pero se observa que cuando el movimiento se realiza entre 2 marcadores esquina adyacentes, el robot presenta problemas, dado que la trayectoria ideal es bastante limítrofe y está todo el rato saliendo y entrando de los márgenes, como se puede apreciar en la Figura 10.63.

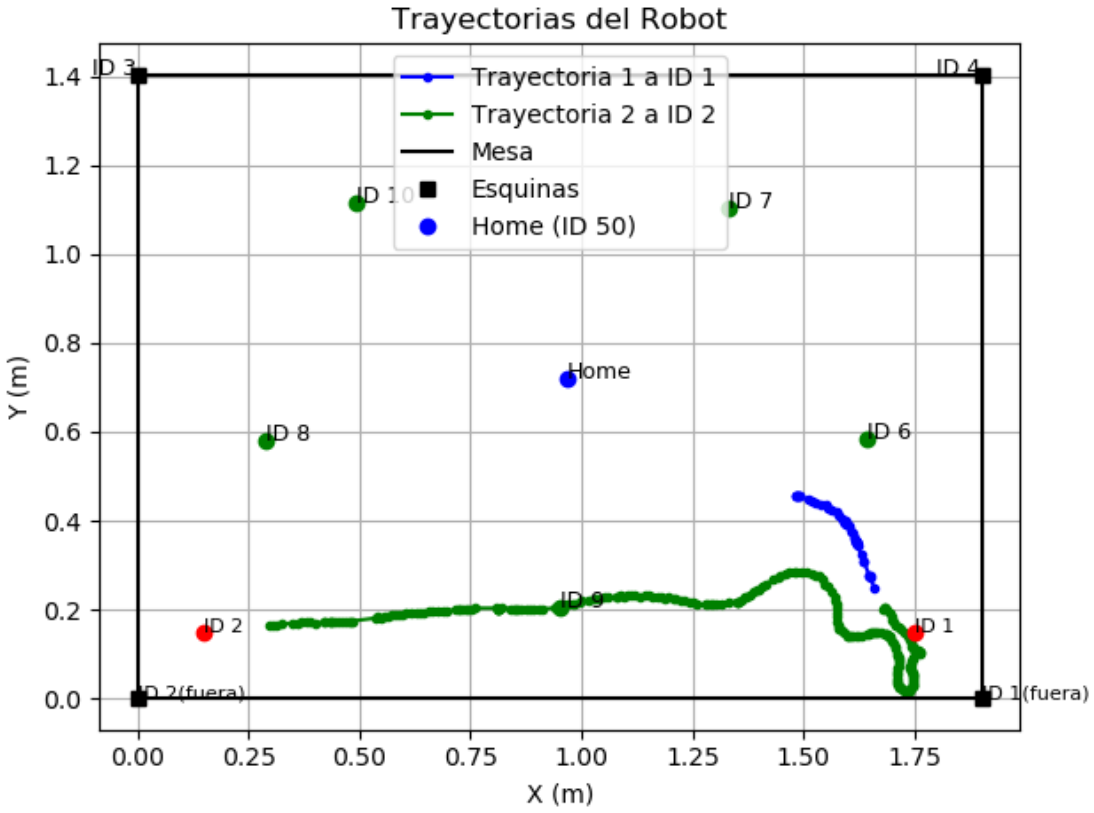

Figura 10.63 Trayectoria entre ID 1 y 2 sin diferencia de márgenes interno y externo en la modificación 3.

Por ello se pondrá un margen interno mayor para que el robot se acerque más a home y no esté tan cerca del margen interno.

Para ello se establece un margen interno de 0.25 m, de esta manera hay 10 cm de diferencia entre el margen externo y el interno.

Se observa en la Figura 10.64 que ahora la trayectoria es más interior y el robot no se encuentra tanto en el límite del margen externo.

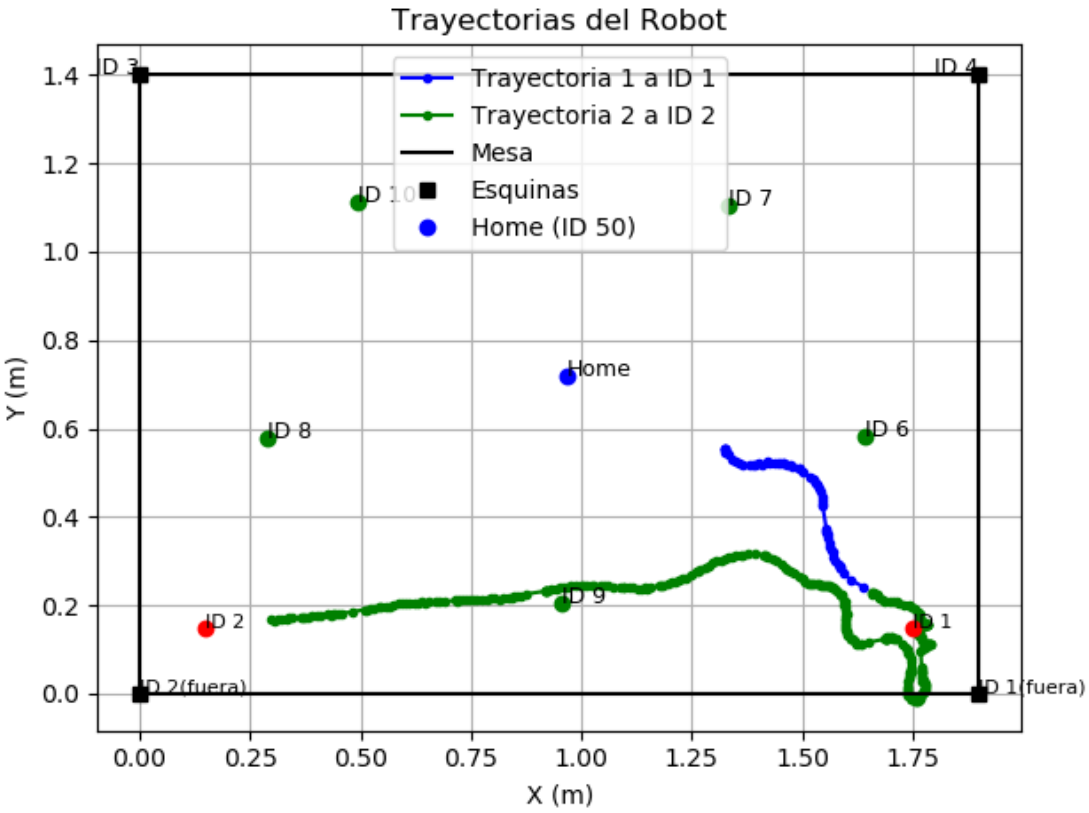

Figura 10.64 Trayectoria entre ID 1 y 2 con margen interno de 0.25 m en la modificación 3.

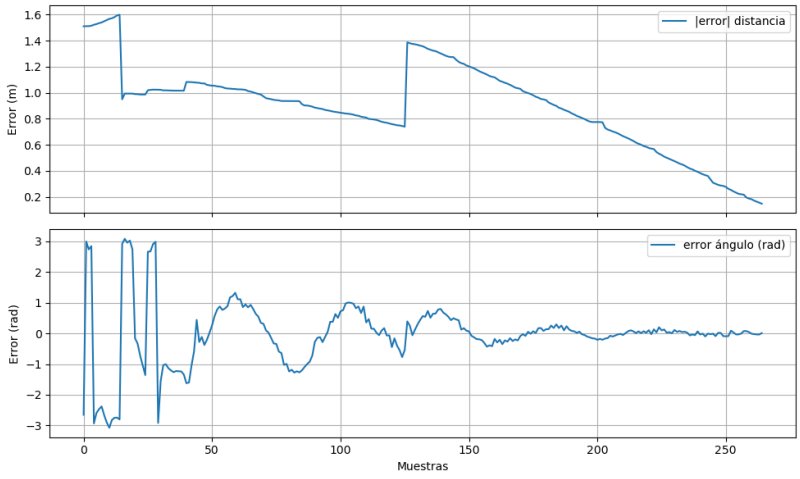

Figura 10.65 Errores entre ID 1 y 2 con margen interno de 0.25 m en la modificación 3.

En la Figura 10.65 se aprecia que el error en distancia al principio describe una especie de *"valle"*, esto es debido a que sale del margen exterior y pasa a considerar la posición home como objetivo, la cual se encuentra a una menor distancia, una vez entra en el margen interno recupera el ID 2 como objetivo.

10.5.4 Prueba con configuración final

Tras haber realizado las modificaciones oportunas, a continuación, se muestra la configuración final.

```
--- Configuración de Parámetros PID ---

Controlador Lineal:
   Kp=0.723, Ki=0.02, Kd=0.0732
   integrator_limit=0.2
   output_min=None, output_max=None
   deadband=0.0

Controlador Angular:
   Kp=0.809, Ki=0.02, Kd=0.0799
   integrator_limit=0.2
   output_min=None, output_max=None
   deadband=0.0
```

Figura 10.66 Configuración final PID.

```
--- Menú de Configuración ---
1. Configurar parámetros PID
2. Configurar velocidad lineal máxima (actual: 0.07 m/s)
3. Configurar velocidad angular máxima (actual: 1.00 rad/s)
4. Configurar velocidad lineal máxima fuera de los márgenes (actual: 0.04 m/s)
5. Configurar velocidad angular máxima fuera de los márgenes (actual: 0.70 rad/s)
6. Configurar tolerancia de distancia para marcadores nuevos (actual: 0.07 m)
7. Configurar tolerancia de distancia para marcadores existentes (actual: 0.15 m)
8. Configurar tolerancia de distancia para puntos objetivo (actual: 0.07 m)
9. Configurar margen interno de seguridad (actual: 0.25 m)
10. Configurar margen externo de seguridad (actual: 0.15 m)
11. Restablecer valores por defecto
12. Continuar con la ejecución
Seleccione una opción (1-12):
```

Figura 10.67 Configuración final de los parámetros.

Con estos valores se hará una prueba global para comprobar el correcto funcionamiento del programa.

Partiendo de la posición (1.064, 0.746), se selecciona como objetivo el punto (1.96, 1.09), se ha puesto queriendo una posición fuera de lo que se considera mesa para apreciar que el programa toma como objetivo el punto más cercano dentro de los márgenes internos, llega a la meta tras 12.86 segundos. Ahora los tiempos de viaje del robot son mayores que cuando se hicieron las pruebas del PID del capítulo 10.4., dado que se ha reducido la velocidad máxima. El siguiente objetivo es el punto (-0.07, 0.34) que también se encuentra fuera de la mesa virtual, por lo que se busca el punto más cercano dentro de los márgenes, tras 23.32 segundos se llega al segundo destino. El tercer viaje es al marcador 1, la trayectoria que elige el robot para llegar no le permite alcanzar la posición sin salir del margen externo, pero ahora con el margen interno de 0.25 metros se consigue que el robot intente entrar en una posición más diagonal tras haberse acercado a la posición home, de manera que logra alcanzar el punto dentro del margen tras 38.02 s. Por último, se indicará que el robot se dirija a la posición home, el robot al estar en una posición muy limítrofe abandona el margen externo, pero al ser la propia posición home el objetivo, el robot se va acercando a baja velocidad hasta que entra en el margen externo y recupera el ritmo normal, este viaje lo realiza en 19.76 s.

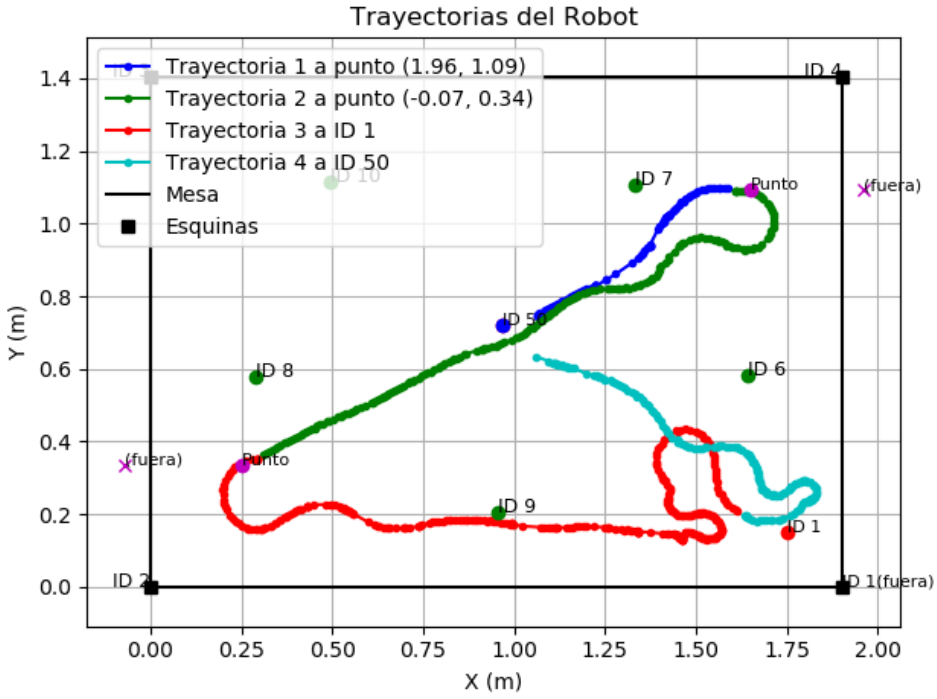

Figura 10.68 Trayectorias recorridas en la prueba final.

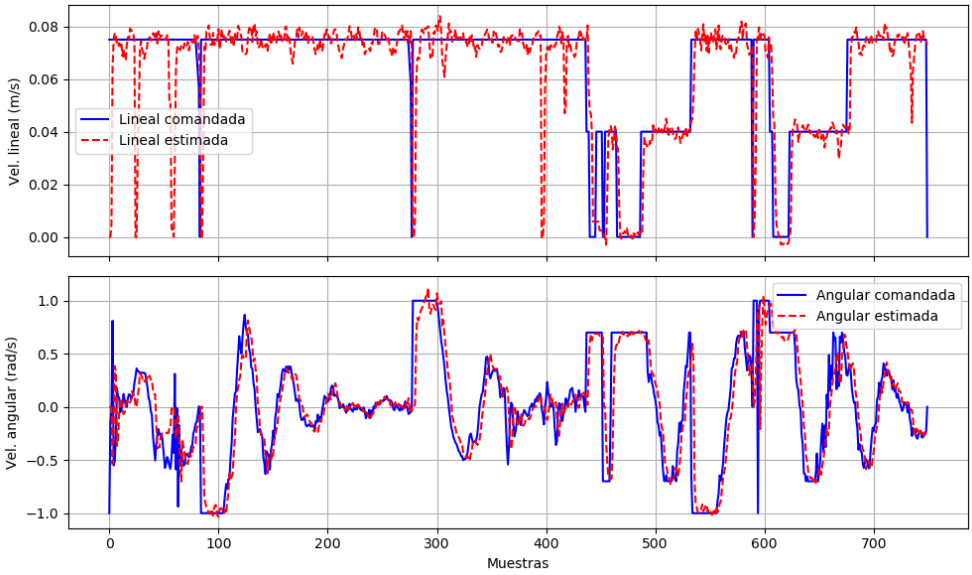

Figura 10.69 Registro de velocidades en la prueba final.

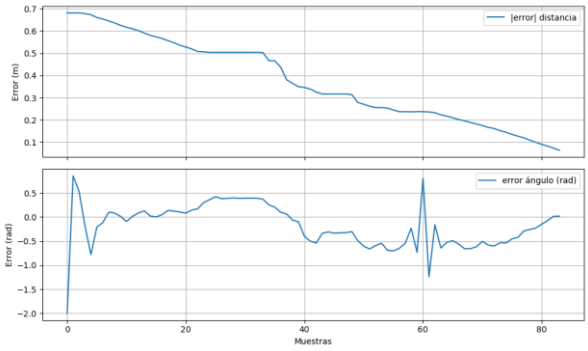

Figura 10.70 Errores 1º destino, prueba final.

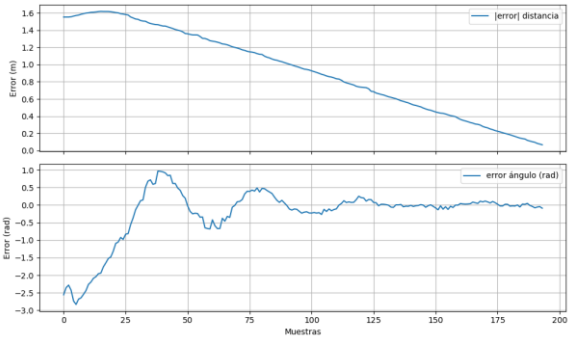

Figura 10.71 Errores 2º destino, prueba final.

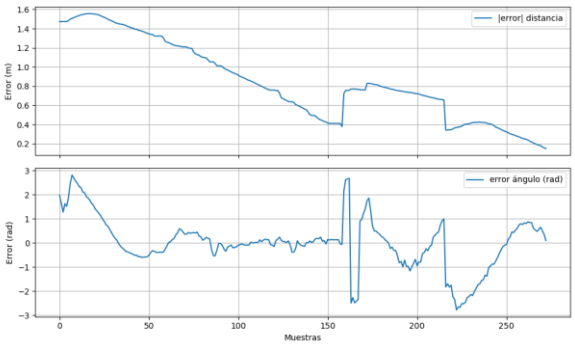

Figura 10.72 Errores 3º destino, prueba final.

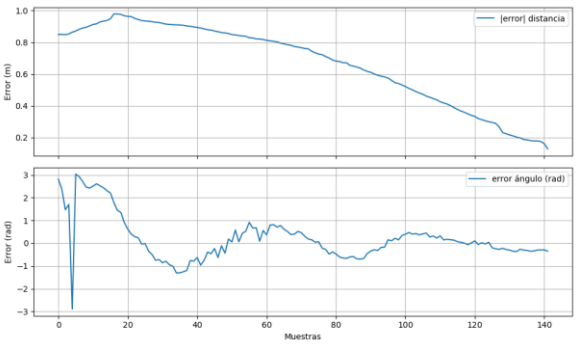

Figura 10.73 Errores 4º destino, prueba final.

Se puede decir que al haber bajado la velocidad el robot tarda más en llegar a los objetivos, pero ahora lo hace con trayectorias más óptimas y seguras, además tanto las velocidades lineales como las angulares no saturan en ningún momento, solo se observa el parón intermitente que se anunció anteriormente, pero este es debido a un problema en la energía que envía la batería, es decir, externo al proyecto. Por otro lado, los valores de tolerancias no han dado ningún tipo de problema y el quedarse a 7 centímetros del marcador no es un error muy grande. Por último, el haber aumentado el margen interno ha permitido que el robot no haga trayectorias que vayan tan cerca de los bordes, además como se ha visto en esta prueba en el camino hacia el ID 1, en posiciones objetivo muy limítrofes el margen interno permite entrar en un ángulo más seguro.

En cuanto a los errores, se observa como en la Figura 10.72 varía su tendencia cuando abandona los márgenes, pero esto no ocurre en la Figura 10.73 dado que, en este, el destino es la propia posición home.

10.6. Pérdida de imágenes

Aunque la implementación de un triple búffer ha permitido reducir considerablemente la pérdida de imágenes del programa sigue habiendo algunas que se pierden, como se ha visto anteriormente no tiene trascendencia en el programa, pero es conveniente realizar un análisis. Para ello se mostrarán los resultados obtenidos de la lectura de imágenes de todos los experimentos anteriores que finalizaron correctamente.

- Test 1: Experimento 1 del modo manual.
- Test 2: Experimento 2 del modo manual.
- Test 3: Prueba de control P por Z-N.
- Test 4: Experimento Z-N con ajuste fino 1.
- Test 5: Experimento Z-N con ajuste fino 2.

- Test 6: Experimento Z-N con ajuste fino 3.
- Test 7: Experimento con la modificación 1.
- Test 8: Experimento con la modificación 2 y tolerancia de 0.07 m.
- Test 9: Experimento de la modificación 3 con margen interno de 0.25 m.
- Test 10: Experimento final.

	PÉRDIDA DE IMÁGENES		
	IMÁGENES PERDIDAS	IMÁGENES TOTALES	PORCENTAJE
TEST 1	18	2516	0.715%
TEST 2	20	2592	0.772%
TEST 3	6	704	0.852%
TEST 4	6	568	1.056%
TEST 5	10	1292	0.774%
TEST 6	11	656	1.677%
TEST 7	16	1044	1.533%
TEST 8	8	680	1.176%
TEST 9	17	1480	1.149%
TEST 10	29	3000	0.967%
TOTAL	141	14532	0.970%

Tabla 10.5 Resultados de la pérdida de imágenes en los experimentos.

Se obtiene un total de pérdida de imágenes del 0.97%, es decir que el 99.03% se leen correctamente, además cuando se produce una pérdida se puede leer otra imagen de las tres del triple búffer.

Se puede decir que son resultados bastante buenos y que prácticamente la pérdida no se aprecia en la ejecución del programa.

11 PROYECTOS FUTUROS

Para proyectos futuros sería interesante que el robot aprovechara el LIDAR que posee para poder evitar obstáculos que se encuentren en la plataforma de trabajo, de manera que buscara una trayectoria óptima para bordear el obstáculo y seguir hacia el objetivo. También se podría investigar alguna forma de eliminar o hacer que afecten menos a las cámaras los reflejos en la mesa dado que los focos infrarrojos no son suficiente.

Se podría intentar usar ROS 2 en vez de ROS 1, dado que no haría falta un maestro y es una versión más actualizada de ROS.

Por otra parte, se podrían añadir técnicas de mapeo y localización para que el robot pudiera aprender del entorno.

En este proyecto se ha trabajado en 2D, pero en un futuro se podría intentar trabajar en tres dimensiones, esto ayudaría a que no se descalibrasen las cámaras si la mesa se mueve o se produce alguna permutación.

Se podría estudiar el aumentar la optimización de trayectorias o el uso de algoritmos de control más eficientes que el PID para que el robot tardara menos tiempo sin poner en riesgo la seguridad.

Las imágenes se podrían filtrar para eliminar los problemas de las sombras y reflejos, también se podrían modificar usando Inteligencia Artificial, este sería un método interesante sobre todo para las imágenes usadas en la calibración, pero sería más difícil su implementación en la lectura de imágenes en tiempo real, debido al tiempo que llevaría. Hay que decir que se podría poner un filtro que corrija las distorsiones.

En los experimentos se observó que cuando el robot no tenía mucha carga de batería, se producían pequeñas interrupciones del movimiento, se podría poner una batería nueva al robot para solventar estos problemas.

El uso de realidad aumentada podría permitir visualizar la trayectoria recorrida y la teórica a recorrer desde la previsualización de una de las cámaras, con esto el usuario tendría una idea del camino a recorrer.

Una idea que se podría implementar en el futuro es una mayor sincronización del PID lineal y del angular, dado que actualmente son bastante independientes.

Una de las ideas iniciales era que la captura de imágenes y que el control del robot estuviera en un mismo programa, en un futuro, se podría ver si esto sería posible, pero habría que tener en cuenta como hacer viable que las cámaras y el robot funcionasen en el mismo sistema operativo.

12 CONCLUSIONES DEL PROYECTO

En este proyecto se ha estudiado cómo, a partir de las imágenes tomadas por cuatro cámaras, se puede hacer una reconstrucción bidimensional de una escena usando una transformación proyectiva como es la homografía. La homografía, al solo permitir representar un plano, no permite tener en cuenta las alturas, algo que sí hacía la calibración estéreo. Sin embargo, las imágenes que se tomaban del tablero ajedrez para este último método, al presentar destellos principalmente en el centro de la plataforma de trabajo, impedían conseguir una cantidad aceptable de imágenes para realizar la calibración estéreo. Es cierto que hubo que hacer dos homografías para cada cámara y después una tercera para relacionar ambos planos. Por otro lado, se ha demostrado que con cuatro marcadores situados en las esquinas de la mesa es suficiente para conocer los puntos necesarios para realizar la homografía de manera precisa. Con las matrices de homografía y las medidas reales de la plataforma de trabajo es posible saber las coordenadas de cualquier punto de la mesa respecto a un origen de coordenadas. Esto sumado a que también se conoce la posición del marcador ArUco que posee el robot ha permitido saber dónde estaba el robot en todo momento y a partir de esto, el diseño de un control para que el robot se pudiera mover de manera manual o autónoma.

Se ha podido observar cómo, en condiciones reales, en proyectos que usan algoritmos de visión artificial, la iluminación y los destellos que se pueden producir hay que tenerlos en cuenta a la hora de realizar una calibración. Los cambios en el entorno deben considerarse, como podría ser un desplazamiento de la plataforma de trabajo en este caso.

El proyecto, usa a su vez, dos sistemas operativos en un mismo ordenador simultáneamente, dado que las cámaras solo funcionaban en Windows y ROS en Ubuntu, pero con el uso de WSL y carpetas compartidas se ha podido demostrar la viabilidad de poder poner en funcionamiento tanto las cámaras como ROS a la vez y ser capaces de transferirse información, en este caso, las imágenes.

En cuanto al robot, el *Turtlebot 3 Burger* se ha mostrado como un robot móvil capaz de alcanzar las diferentes metas que se le han impuesto durante los experimentos tras haber ajustado sus parámetros. Sin embargo, la batería del robot no supera las 2 horas de duración y para cargarlo, hay que extraerla del robot, lo que hace que sea una tarea complicada.

Al haber usado un sistema multicámara que cubre cada esquina de la plataforma de trabajo, se obtienen estimaciones precisas y, si la lectura de alguna cámara fallara, hay otras tres para poder hacer la estimación de posición. Además, en el hipotético caso de que todas las cámaras fallaran o no pudieran detectar la posición, el robot se detendría.

Se ha visto cómo se puede hacer un control de posición usando dos controladores PID, uno para el movimiento lineal cuya entrada era el error en posición, y otro para el movimiento angular cuya entrada era el error angular. Se observó que la sintonización por Ziegler-Nichols en bucle abierto daba un término integral demasiado grande, que tuvo que ser reducido por experimentación además de la implementación de un anti-windup para limitar el error integral.

Un robot móvil que se encuentra en una mesa de 1 metro de altura debe tener medidas de seguridad para no caerse, dado que es un elemento valioso y no es permisible una caída. Por ello, se han implementado márgenes de seguridad para que, si el robot se encuentra fuera de estos, vuelva con cuidado, limitando la velocidad, hasta que vuelva a la zona dentro de los márgenes.

Si se habla de aplicaciones prácticas en el día a día, este proyecto se podría utilizar en trabajos en los que necesite el transporte de objetos de forma autónoma en interiores. Haría falta un sistema multicámara en el cual, en todo momento, las cámaras vieran la ruta del trayecto. Además, habría que conocer las distancias reales entre los diferentes marcadores. La desventaja es que al trabajar en 2D, solo serviría para situaciones llanas en las que no exista ninguna pendiente.

Otra aplicación práctica sería en tareas de patrullaje de seguridad en interiores, de forma que el robot recorra pasillos u otras zonas de manera periódica y, con sensores de presencia, detecte si ocurre algo irregular.

ANEXO

a. ANEXO 1: T_Nano-M1280-NIR_Default_GrandMaster_15ms_60Hz.ccf

[Board]
Server Name=Nano-M1280-NIR
Device Name=S1177338
Device Index=0
[General]
Company Name=Teledyne DALSA
Camera Name=Default
Model Name=Nano-M1280-NIR
Vic Name=GrandMaster_15ms_60Hz
[Acquisition Device Features]
{05D8C294-F295-4dfb-9D01-096BD04049F4}
GenApi persistence file (version 2.4.0)
S1177338 = TeledyneDALSA::Nano -- Genie Nano Configuration XML -- Device version = 1.7.60 -- Product GUID = 6a3fe8b0-5fec-11e2-9520-ea771535200d -- Product version GUID = 713f235e-5fec-11e2-ba41-ea771535200d
acquisitionFrameRateControlMode Programmable
AcquisitionFrameRate 60
autoBrightnessMode Off
ExposureMode Timed
exposureAlignment Synchronous
ExposureTime 15000
GainSelector SensorAnalog
GainSelector SensorAnalog
Gain 3.2
GainSelector SensorDigital
Gain 1
GainSelector DigitalAll
Gain 1
GainSelector SensorAnalog
BlackLevelSelector AnalogAll
BlackLevelSelector AnalogAll
BlackLevelRaw 1
BlackLevelSelector AnalogAll
fastReadoutMode Active
sensorFpnCorrectionMode Active
TriggerSelector FrameStart
TriggerSelector FrameStart
TriggerMode Off
TriggerSelector FrameBurstStart
TriggerMode Off
TriggerSelector AcquisitionStart
TriggerMode Off
TriggerSelector FrameStart
LineSelector Line1
LineSelector Line1
LineInverter 0
LineSelector Line2
LineInverter 0
LineSelector Line1
LineSelector Line1
lineDebouncingPeriod 0
LineSelector Line2
lineDebouncingPeriod 0

LineSelector Line1
LineSelector Line3
outputLineSource Off
LineSelector Line4
outputLineSource Off
LineSelector Line1
counterSelector Counter1
counterSelector Counter1
counterMode Off
counterSelector Counter1
counterSelector Counter1
counterStartSource Line1
counterSelector Counter1
counterSelector Counter1
counterStartLineActivation RisingEdge
counterSelector Counter1
counterSelector Counter1
counterIncrementalSource InternalClock
counterSelector Counter1
counterSelector Counter1
counterDuration 1
counterSelector Counter1
counterSelector Counter1
counterResetSource Off
counterSelector Counter1
timerSelector Timer1
timerSelector Timer1
timerMode Off
timerSelector Timer1
timerSelector Timer1
timerStartSource Line1
timerSelector Timer1
timerSelector Timer1
timerStartLineActivation RisingEdge
timerSelector Timer1
timerSelector Timer1
timerDuration 1
timerSelector Timer1
defectivePixelReplacementMode Off
lutMode Off
lutType UserDefined
gammaCorrection 0.001
cyclingPresetMode Off
cyclingPresetCount 2
cyclingPresetRoiPositionSource FPGA
cyclingPresetIncrementalSource StartOfFrame
cyclingPresetResetSource EndOfAcquisition
cyclingPresetRepeater 1
cP_FeaturesActivationSelector ExposureTime
cP_FeaturesActivationSelector ExposureTime
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector ROI_Position
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector OutputLine3Control
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector OutputLine4Control
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector BinningHorizontal

cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector BinningVertical
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector SensorAnalogGain
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector ExposureTime
cP_PresetConfigurationSelector 1
cP_PresetConfigurationSelector 1
cP_GainSelector SensorAnalog
cP_PresetConfigurationSelector 2
cP_GainSelector SensorAnalog
cP_PresetConfigurationSelector 1
cP_PresetConfigurationSelector 1
cP_LineSelector Line3
cP_PresetConfigurationSelector 2
cP_LineSelector Line3
cP_PresetConfigurationSelector 1
ReverseY 0
Width 1280
OffsetX 0
Height 1024
OffsetY 0
PixelFormat Mono8
multipleROIModeOff
binningSelector InSensor
binningMode Average
binningSelector InDigitalDomain
binningMode Sum
binningSelector InDigitalDomain
binningSelector InSensor
BinningHorizontal 1
binningSelector InDigitalDomain
BinningHorizontal 1
binningSelector InDigitalDomain
binningSelector InSensor
BinningVertical 1
binningSelector InDigitalDomain
BinningVertical 1
binningSelector InDigitalDomain
ChunkModeActive 1
chunkCompatibilityMode SaperaLT
ptpMode Automatic
timestampModulo 0
timestampModuloStartTime 0
devicePacketResendBufferSize 6
DeviceLinkThroughputLimitMode On
DeviceLinkThroughputLimit 115000000
GevStreamChannelSelector 0
GevSCPSPacketSize 9000
GevStreamChannelSelector 0
GevStreamChannelSelector 0
GevSCPD 7
GevStreamChannelSelector 0
ImageTimeoutRaw 700000
InterPacketTimeoutRaw 45000
CommandAcknowledgeTimeout 250
CommandRetryCount 3
multiCastStreamingEnable 0
multiCastStreamingIPAddress 239.10.10.10

systemTransferStopDetectionMethod	Automatic
packetResendMax	100.0
turboTransferEnable	0

b. ANEXO 2: T_Nano-M1280-NIR_Default_GrandMaster_15ms_60Hz.ccf

[Board]
Server Name=Nano-M1280-NIR
Device Name=S1186692
Device Index=0
[General]
Company Name=Teledyne DALSA
Camera Name=Default
Model Name=Nano-M1280-NIR
Vic Name=Slave_15ms_58Hz
[Acquisition Device Features]
{05D8C294-F295-4dfb-9D01-096BD04049F4}
GenApi persistence file (version 2.4.0)
S1186692 = TeledyneDALSA::Nano -- Genie Nano Configuration XML -- Device version = 1.7.60 -- Product GUID = 6a3fe8b0-5fec-11e2-9520-ea771535200d -- Product version GUID = 713f235e-5fec-11e2-ba41-ea771535200d
autoBrightnessMode Off
ExposureMode Timed
exposureAlignment Synchronous
exposureDelay 4
ExposureTime 15000
GainSelector SensorAnalog
GainSelector SensorAnalog
Gain 3.2
GainSelector SensorDigital
Gain 1
GainSelector DigitalAll
Gain 1
GainSelector SensorAnalog
BlackLevelSelector AnalogAll
BlackLevelSelector AnalogAll
BlackLevelRaw 1
BlackLevelSelector AnalogAll
fastReadoutMode Active
sensorFpnCorrectionMode Active
TriggerSelector FrameStart
TriggerSelector FrameStart
TriggerMode On
TriggerSelector FrameBurstStart
TriggerMode Off
TriggerSelector AcquisitionStart
TriggerMode Off
TriggerSelector FrameStart
TriggerSelector FrameStart
TriggerSource timestampModuloEvent
TriggerSelector FrameStart
TriggerSelector FrameStart
TriggerDelay 0
TriggerSelector FrameBurstStart
TriggerDelay 0
TriggerSelector FrameStart
TriggerSelector FrameStart
TriggerOverlap ReadOut
TriggerSelector FrameBurstStart
TriggerOverlap ReadOut
TriggerSelector FrameStart
LineSelector Line1
LineSelector Line1
LineInverter 0

LineSelector Line2
LineInverter 0
LineSelector Line1
LineSelector Line1
lineDebouncingPeriod 0
LineSelector Line2
lineDebouncingPeriod 0
LineSelector Line1
LineSelector Line3
outputLineSource Off
LineSelector Line4
outputLineSource Off
LineSelector Line1
counterSelector Counter1
counterSelector Counter1
counterMode Off
counterSelector Counter1
counterSelector Counter1
counterStartSource Line1
counterSelector Counter1
counterSelector Counter1
counterStartLineActivation RisingEdge
counterSelector Counter1
counterSelector Counter1
counterIncrementalSource InternalClock
counterSelector Counter1
counterSelector Counter1
counterDuration 1
counterSelector Counter1
counterSelector Counter1
counterResetSource Off
counterSelector Counter1
timerSelector Timer1
timerSelector Timer1
timerMode Off
timerSelector Timer1
timerSelector Timer1
timerStartSource Line1
timerSelector Timer1
timerSelector Timer1
timerStartLineActivation RisingEdge
timerSelector Timer1
timerSelector Timer1
timerDuration 1
timerSelector Timer1
defectivePixelReplacementMode Off
lutMode Off
lutType UserDefined
gammaCorrection 0.001
cyclingPresetMode Off
cyclingPresetCount 2
cyclingPresetRoiPositionSource FPGA
cyclingPresetIncrementalSource StartOfFrame
cyclingPresetResetSource EndOfAcquisition
cyclingPresetRepeater 1
cP_FeaturesActivationSelector ExposureTime
cP_FeaturesActivationSelector ExposureTime
cP_FeaturesActivationMode Off

```
cP_FeaturesActivationSelector ExposureDelay
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector ROI_Position
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector OutputLine3Control
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector OutputLine4Control
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector BinningHorizontal
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector BinningVertical
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector SensorAnalogGain
cP_FeaturesActivationMode Off
cP_FeaturesActivationSelector ExposureTime
cP_PresetConfigurationSelector 1
cP_PresetConfigurationSelector 1
cP_GainSelector SensorAnalog
cP_PresetConfigurationSelector 2
cP_GainSelector SensorAnalog
cP_PresetConfigurationSelector 1
cP_PresetConfigurationSelector 1
cP_LineSelector Line3
cP_PresetConfigurationSelector 2
cP_LineSelector Line3
cP_PresetConfigurationSelector 1
ReverseY 0
Width 1280
OffsetX 0
Height 1024
OffsetY 0
PixelFormat Mono8
multipleROIModeOff
binningSelector InSensor
binningMode Average
binningSelector InDigitalDomain
binningMode Sum
binningSelector InDigitalDomain
binningSelector InSensor
BinningHorizontal 1
binningSelector InDigitalDomain
BinningHorizontal 1
binningSelector InDigitalDomain
binningSelector InSensor
BinningVertical 1
binningSelector InDigitalDomain
BinningVertical 1
binningSelector InDigitalDomain
ChunkModeActive 1
chunkCompatibilityMode SaperaLT
ptpMode Automatic
timestampModulo 17000000
timestampModuloStartTime 0
devicePacketResendBufferSize 6
DeviceLinkThroughputLimitMode On
DeviceLinkThroughputLimit 115000000
GevStreamChannelSelector 0
GevSCPSPacketSize 9000
GevStreamChannelSelector 0
```

GevStreamChannelSelector 0
GevSCPD 7
GevStreamChannelSelector 0
ImageTimeoutRaw 700000
InterPacketTimeoutRaw 45000
CommandAcknowledgeTimeout 250
CommandRetryCount 3
multiCastStreamingEnable 0
multiCastStreamingIPAddress 239.10.10.10
systemTransferStopDetectionMethod Automatic
packetResendMax 100.0
turboTransferEnable 0

c. ANEXO 3: Actioncmd.cpp

```
/*****************************************************************************************
ActionCmd.cpp

Description:
Broadcast an action command on a specific network using GVCP protocol.
Now includes a new mode for continuous real-time image acquisition with a preview of a selected camera.

*****************************************************************************************/
// version 2025
// MODIFICADO

//=============================================================
// INCLUDE FILES
//=============================================================
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <winsock2.h>
#include <IPHlpApi.h>
#include <ActionCmd.h>
#include <iostream>
#include <fstream>
#include <chrono>
#include <string.h>
using namespace std;
//_____________________________________________________________________________________________
_____________________________________________________
// Add Sapera LT include files here
#include "SapClassBasic.h"
// Sapera LT (end)

#pragma comment(lib, "ws2_32.lib")
#pragma comment(lib, "IPHLPAPI.lib")

// Commented out when using action commands which are executed immediately
#define USE_SCHEDULED_ACTION_COMMAND

#ifdef USE_SCHEDULED_ACTION_COMMAND
#pragma message ("Scheduled action commands are enabled")
#else
#pragma message ("Scheduled action commands are disabled")
#endif

//=============================================================
// Member variables
//=============================================================

// Definir el número de imágenes a capturar en cada “tanda”
const int NUM_IMAGENES = 1;

// Contadores para las fotos de cada cámara (archivos bmp)
int num_foto1 = 1;
int num_foto2 = 1;
int num_foto3 = 1;
int num_foto4 = 1;
```

```
static int GVCP_PACKET_SIZE = 0;

/********************* HEADER (hard coded) **********************/

static UINT8 key[GVCP_KEY_LENGTH] = { 0x42 };                    // Key identifying GVCP packets

// {0x01} (0b00000001) -> Scheduled Action Commands is disabled and an ACTION_ACK is requested
// {0x80} (0b10000000) -> Scheduled Action Commands is enabled and no ACTION_ACK is requested
// {0x81} (0b10000001) -> Scheduled Action Commands is enabled and an ACTION_ACK is requested
#ifdef USE_SCHEDULED_ACTION_COMMAND
static UINT8 flag[GVCP_FLAG_LENGTH] = { 0x81 };
#else
static UINT8 flag[GVCP_FLAG_LENGTH] = { 0x01 };
#endif

static UINT8 cmd[GVCP_COMMAND_LENGTH] = { 0x01, 0x00 };          // ACTION_CMD = 0x0100

// Length of the payload (in bytes)
// {0x00, 0x0C} -> the payload is 12 bytes (no action_time field)
// {0x00, 0x14} -> the payload is 20 bytes (action_time field included)
#ifdef USE_SCHEDULED_ACTION_COMMAND
static UINT8 length[GVCP_LENGTH_LENGTH] = { 0x00, 0x14 };
#else
static UINT8 length[GVCP_LENGTH_LENGTH] = { 0x00, 0x0C };
#endif

// action_time field, to use only if Scheduled Action Commands is set in the flag and if the value of "length"
// is set to 20 bytes {0x00, 0x14}.
static UINT8 action_time_H[ACTION_TIME_LENGTH] = { 0x00, 0x00, 0x00, 0x00 };
static UINT8 action_time_L[ACTION_TIME_LENGTH] = { 0x01, 0x00, 0x00, 0x00 };

static UINT8 req_id[GVCP_ID_LENGTH] = { 0x00, 0x00 };            // Number of request
// req_id will be incremented before to send the first command because req_id can never be 0.

/********************* PAYLOAD (hard coded) **********************/

// GROUP_MASK is set in FillCommandPayload because it depends on the user
static UINT8 device_key[DEVICE_KEY_LENGTH] = { 0x00, 0x00, 0x00, 0x01 };
static UINT8 group_key[GROUP_KEY_LENGTH] = { 0x00, 0x00, 0x00, 0x00 };

// SOCKET
static UINT8 datagram_out[BUFLEN];                    // Actual datagram to send -> command
static UINT8 datagram_in[BUFLEN];              // Actual datagram received -> ack
static SOCKET sock = INVALID_SOCKET;      // Socket to access
static struct sockaddr_in sock_source;
static struct sockaddr_in sock_destination;
static int nb_ack = 0;                                        // Number of ACK received
static int nb_non_ack = 0;                                    // Number of NON-ACK messages received

// To save the infos of the network chosen by the user.
struct _tNIC
{
        PWCHAR network_name;
        char* NICIP;
};

typedef struct _tNIC tNIC;
```

```
// Static Functions
static void MyAcquisitionCallback1(SapAcqDeviceCallbackInfo* pInfo);
static void MyAcquisitionCallback2(SapAcqDeviceCallbackInfo* pInfo);
static void MyAcquisitionCallback3(SapAcqDeviceCallbackInfo* pInfo);
static void MyAcquisitionCallback4(SapAcqDeviceCallbackInfo* pInfo);

void SetCameraStartTime(SapAcqDevice* m_AcqDevice, UINT64 starttime);
void ResetCameraTimestamp(SapAcqDevice* m_AcqDevice);

// Define directorios para cada modo
const char* calibracion_plano0_dir = "C:\\Users\\user\\Documents\\Captura Imagenes Camaras\\CalibracionPlano0\\";
const char* calibracion_robot_dir = "C:\\Users\\user\\Documents\\Captura Imagenes Camaras\\CalibracionRobot\\";
const char* calibracion_nuevo_marc = "C:\\Users\\user\\Documents\\Captura Imagenes Camaras\\Calibracion_Nuevo_Marc\\";
const char* modo_real_dir = "C:\\Users\\user\\Documents\\Captura Imagenes Camaras\\ModoReal\\";

// Iniciar los directorios para las imagenes
char camara1_dir[300];
char camara2_dir[300];
char camara3_dir[300];
char camara4_dir[300];

fstream file1_aux, file1_host, file2_aux, file2_host, file3_aux, file3_host, file4_aux, file4_host;

char auxtimestamps1_dir[300];
char auxtimestamps2_dir[300];
char auxtimestamps3_dir[300];
char auxtimestamps4_dir[300];
char hosttimestamps1_dir[300];
char hosttimestamps2_dir[300];
char hosttimestamps3_dir[300];
char hosttimestamps4_dir[300];

bool dirExists(const std::string& dirName_in)
{
	DWORD ftyp = GetFileAttributesA(dirName_in.c_str());
	if (ftyp == INVALID_FILE_ATTRIBUTES)
		return false;  //something is wrong with your path!

	if (ftyp & FILE_ATTRIBUTE_DIRECTORY)
		return true;   // this is a directory!

	return false;    // this is not a directory!
}

// Variable para seleccionar camara
int selectedCameraIndex = 1;

// SapView ventana de previsualización
SapView* view = nullptr;

//seleccionar camara para visualizar
void SelectCamera()
{
	printf("Seleccione la camara para la previsualizacion (1-4): ");
	int index = 0;
	char input = getch();
	if (input >= '1' && input <= '4') {
```

```
                index = input - '0';
                selectedCameraIndex = index;
                printf("%c\n", input);
                printf("Camara %d seleccionada para previsualizacion.\n", selectedCameraIndex);
        }
        else {
                printf("%c\n", input);
                printf("Seleccion invalida. Se mantiene la camara %d.\n", selectedCameraIndex);
        }
}

// Function to initialize tNIC
tNIC init_NICinfos();

//=============================================================
// Global Variables
//=============================================================
int currentMode = 0; // Tracks the current mode (1, 2, 3, or 4)

// -- CAMBIO: Variables estáticas acumulativas para cada modo:
static int totalCapturasModo1 = 0;
static int totalCapturasModo2 = 0;
static int totalCapturasModo3 = 0;

//=============================================================
// MAIN
//=============================================================
int main(VOID)
{
        // Crear directorios para cada modo
        if (!dirExists(calibracion_plano0_dir)) { CreateDirectory(calibracion_plano0_dir, NULL); }
        if (!dirExists(calibracion_robot_dir)) { CreateDirectory(calibracion_robot_dir, NULL); }
        if (!dirExists(calibracion_nuevo_marc)) { CreateDirectory(calibracion_nuevo_marc, NULL); }
        if (!dirExists(modo_real_dir)) { CreateDirectory(modo_real_dir, NULL); }

        printf("Sapera Console Action Command modificado para captura de imagenes (C++)\n\n");

#ifdef USE_SCHEDULED_ACTION_COMMAND
        printf("Action commands habilitados\n");
#else
        printf("Scheduled action commands are disabled\n");
#endif

        tNIC NICinfos = init_NICinfos();
        BOOL sock_open = FALSE;
        char KEYval = 0;
        BOOL valid_selection = TRUE;
        int mode = 0;

        // Llamar a get_Adapters_Info() para inicializar NICinfos
        if (!get_Adapters_Info(&NICinfos))
        {
                printf("No se selecciono ninguna interfaz de red o ocurrio un error.\n");
                return -1;
        }

        //-------------------------------------------------------------------
        //  CONFIGURACIÓN SAPERA LT para las cuatro cámaras
```

```
	//-----------------------------------------------------------------
	// *** CÁMARA 1 ***
	char* serialNumber = "S1186692";
	char* configFilename = "C:\\Archivos de programa\\Teledyne DALSA\\Sapera\\CamFiles\\User\\Alfonso\\T_Nano-M1280-NIR_Default_GrandMaster_15ms_60Hz.ccf";
	char serverName[MAX_PATH];
	int serverIndex = SapManager::GetServerIndex(serialNumber, SapManager::ResourceAcqDevice);
	if (serverIndex < 0) {
		printf("Error: Camera with serial number %s not found.\n", serialNumber);
		return -1;
	}
	SapManager::GetServerName(serverIndex, serverName, sizeof(serverName));
	SapManager::Open();
	SapAcqDevice* acqDevice = new SapAcqDevice(SapLocation(serverName, 0));
	acqDevice->SetConfigFile(configFilename);
	acqDevice->Create();
	SapBuffer* buffer = new SapBufferWithTrash(NUM_IMAGENES, acqDevice);
	buffer->Create();
	SapTransfer* transfer = new SapAcqDeviceToBuf(acqDevice, buffer);
	transfer->Create();

	// *** CÁMARA 2 ***
	char* serialNumber2 = "S1177338";
	char* configFilename2 = "C:\\Archivos de programa\\Teledyne DALSA\\Sapera\\CamFiles\\User\\Alfonso\\T_Nano-M1280-NIR_Default_Slave_15ms_58Hz.ccf";
	char serverName2[MAX_PATH];
	int serverIndex2 = SapManager::GetServerIndex(serialNumber2, SapManager::ResourceAcqDevice);
	if (serverIndex2 < 0) {
		printf("Error: Camera with serial number %s not found.\n", serialNumber2);
		return -1;
	}
	SapManager::GetServerName(serverIndex2, serverName2, sizeof(serverName2));
	SapAcqDevice* acqDevice2 = new SapAcqDevice(SapLocation(serverName2, 0));
	acqDevice2->SetConfigFile(configFilename2);
	acqDevice2->Create();
	SapBuffer* buffer2 = new SapBufferWithTrash(NUM_IMAGENES, acqDevice2);
	buffer2->Create();
	SapTransfer* transfer2 = new SapAcqDeviceToBuf(acqDevice2, buffer2);
	transfer2->Create();

	// *** CÁMARA 3 ***
	char* serialNumber3 = "S1186684";
	char* configFilename3 = "C:\\Archivos de programa\\Teledyne DALSA\\Sapera\\CamFiles\\User\\Alfonso\\T_Nano-M1280-NIR_Default_Slave_15ms_58Hz.ccf";
	char serverName3[MAX_PATH];
	int serverIndex3 = SapManager::GetServerIndex(serialNumber3, SapManager::ResourceAcqDevice);
	if (serverIndex3 < 0) {
		printf("Error: Camera with serial number %s not found.\n", serialNumber3);
		return -1;
	}
	SapManager::GetServerName(serverIndex3, serverName3, sizeof(serverName3));
	SapAcqDevice* acqDevice3 = new SapAcqDevice(SapLocation(serverName3, 0));
	acqDevice3->SetConfigFile(configFilename3);
	acqDevice3->Create();
	SapBuffer* buffer3 = new SapBufferWithTrash(NUM_IMAGENES, acqDevice3);
	buffer3->Create();
	SapTransfer* transfer3 = new SapAcqDeviceToBuf(acqDevice3, buffer3);
	transfer3->Create();
```

```
	// *** CÁMARA 4 ***
	char* serialNumber4 = "S1186689";
	char* configFilename4 = "C:\\Archivos de programa\\Teledyne DALSA\\Sapera\\CamFiles\\User\\Alfonso\\T_Nano-M1280-NIR_Default_Slave_15ms_58Hz.ccf";
	char serverName4[MAX_PATH];
	int serverIndex4 = SapManager::GetServerIndex(serialNumber4, SapManager::ResourceAcqDevice);
	if (serverIndex4 < 0) {
		printf("Error: Camera with serial number %s not found.\n", serialNumber4);
		return -1;
	}
	SapManager::GetServerName(serverIndex4, serverName4, sizeof(serverName4));
	SapAcqDevice* acqDevice4 = new SapAcqDevice(SapLocation(serverName4, 0));
	acqDevice4->SetConfigFile(configFilename4);
	acqDevice4->Create();
	SapBuffer* buffer4 = new SapBufferWithTrash(NUM_IMAGENES, acqDevice4);
	buffer4->Create();
	SapTransfer* transfer4 = new SapAcqDeviceToBuf(acqDevice4, buffer4);
	transfer4->Create();

	// Bucle principal: Menú de selección de modo
	while (1)
	{
		printf("\nSeleccione el modo de operacion:\n");
		printf("1. Modo Calibracion Plano z0: Toma las imagenes de los marcadores esquina para calibrar.\n");
		printf("2. Calibracion Robot: toma una foto del robot en cada una de las 4 esquinas.\n");
		printf("3. Modo detectar nuevos ids: Toma las imagenes con los nuevos marcadores donde queremos que vaya el robot.\n");
		printf("4. Modo tiempo real: Captura las imagenes en tiempo real para controlar el robot.\n");
		printf("q. Salir\n");
		printf(">> ");

		KEYval = getch();
		if (KEYval == 'q' || KEYval == 'Q')
		{
			printf("\nSaliendo del programa.\n");
			break; // Rompe el while(1) principal: fin del programa
		}
		else if (KEYval == '1') mode = 1;
		else if (KEYval == '2') mode = 2;
		else if (KEYval == '3') mode = 3;
		else if (KEYval == '4') mode = 4;
		else {
			printf("\nSeleccion invalida. Intente de nuevo.\n");
			continue;
		}

		//-------------------------------------------------------------
		// MODO 1
		//-------------------------------------------------------------
		if (mode == 1)
		{
			currentMode = 1; // Set current mode

			// Modo Calibración Plano z0
			strcpy_s(camara1_dir, calibracion_plano0_dir);
			strcat_s(camara1_dir, "camara1\\");
			strcpy_s(camara2_dir, calibracion_plano0_dir);
			strcat_s(camara2_dir, "camara2\\");
```

```
                        strcpy_s(camara3_dir, calibracion_plano0_dir);
                        strcat_s(camara3_dir, "camara3\\");
                        strcpy_s(camara4_dir, calibracion_plano0_dir);
                        strcat_s(camara4_dir, "camara4\\");

                        if (!dirExists(camara1_dir)) { CreateDirectory(camara1_dir, NULL); }
                        if (!dirExists(camara2_dir)) { CreateDirectory(camara2_dir, NULL); }
                        if (!dirExists(camara3_dir)) { CreateDirectory(camara3_dir, NULL); }
                        if (!dirExists(camara4_dir)) { CreateDirectory(camara4_dir, NULL); }

                        acqDevice->SetFeatureValue("TriggerMode", "On");
                        acqDevice->SetFeatureValue("TriggerSource", "Action1");
                        acqDevice2->SetFeatureValue("TriggerMode", "On");
                        acqDevice2->SetFeatureValue("TriggerSource", "Action1");
                        acqDevice3->SetFeatureValue("TriggerMode", "On");
                        acqDevice3->SetFeatureValue("TriggerSource", "Action1");
                        acqDevice4->SetFeatureValue("TriggerMode", "On");
                        acqDevice4->SetFeatureValue("TriggerSource", "Action1");

#ifdef USE_SCHEDULED_ACTION_COMMAND
                        // ...
#endif

                        while (true)
                        {
                                printf("\nIniciando adquisicion. Se tomaran %d fotos por camara.\n",
NUM_IMAGENES);

                                // Archivos de timestamps
                                strcpy_s(auxtimestamps1_dir, camara1_dir);
                                strcat_s(auxtimestamps1_dir, "Camara1_AuxTimestamps.txt");
                                strcpy_s(auxtimestamps2_dir, camara2_dir);
                                strcat_s(auxtimestamps2_dir, "Camara2_AuxTimestamps.txt");
                                strcpy_s(auxtimestamps3_dir, camara3_dir);
                                strcat_s(auxtimestamps3_dir, "Camara3_AuxTimestamps.txt");
                                strcpy_s(auxtimestamps4_dir, camara4_dir);
                                strcat_s(auxtimestamps4_dir, "Camara4_AuxTimestamps.txt");

                                strcpy_s(hosttimestamps1_dir, camara1_dir);
                                strcat_s(hosttimestamps1_dir, "Camara1_HostTimestamps.txt");
                                strcpy_s(hosttimestamps2_dir, camara2_dir);
                                strcat_s(hosttimestamps2_dir, "Camara2_HostTimestamps.txt");
                                strcpy_s(hosttimestamps3_dir, camara3_dir);
                                strcat_s(hosttimestamps3_dir, "Camara3_HostTimestamps.txt");
                                strcpy_s(hosttimestamps4_dir, camara4_dir);
                                strcat_s(hosttimestamps4_dir, "Camara4_HostTimestamps.txt");

                                file1_aux.open(auxtimestamps1_dir, ios::out);
                                file2_aux.open(auxtimestamps2_dir, ios::out);
                                file3_aux.open(auxtimestamps3_dir, ios::out);
                                file4_aux.open(auxtimestamps4_dir, ios::out);
                                file1_host.open(hosttimestamps1_dir, ios::out);
                                file2_host.open(hosttimestamps2_dir, ios::out);
                                file3_host.open(hosttimestamps3_dir, ios::out);
                                file4_host.open(hosttimestamps4_dir, ios::out);

                                transfer->Grab();
                                transfer2->Grab();
                                transfer3->Grab();
```

```
transfer4->Grab();

// -- CAMBIO: usaremos totalCapturasModo1 para que sea acumulativo
for (int i = 0; i < NUM_IMAGENES; i++)
{
	totalCapturasModo1++;
	printf("\nNumero de captura: %d\n", totalCapturasModo1);

	char action_cmd = '1'; // ACTION1
	sock_open = send_Action_cmd(NICinfos, action_cmd, sock_open);
}

transfer->Freeze();
transfer2->Freeze();
transfer3->Freeze();
transfer4->Freeze();

file1_aux.close();
file2_aux.close();
file3_aux.close();
file4_aux.close();
file1_host.close();
file2_host.close();
file3_host.close();
file4_host.close();

// Guardar imágenes en disco
int lastIndex = buffer->GetIndex();
for (int i = 0; i <= lastIndex; i++) {
	// CÁMARA 1
	{
		char numero[5];
		sprintf_s(numero, "%d", num_foto1);
		char cadena_total[300];
		strcpy_s(cadena_total, camara1_dir);
		strcat_s(cadena_total, "camara1-foto");
		strcat_s(cadena_total, numero);
		strcat_s(cadena_total, ".bmp");
		buffer->Save(cadena_total, "-format bmp", i);
		num_foto1++;
	}
	// CÁMARA 2
	{
		char numero2[5];
		sprintf_s(numero2, "%d", num_foto2);
		char cadena_total2[300];
		strcpy_s(cadena_total2, camara2_dir);
		strcat_s(cadena_total2, "camara2-foto");
		strcat_s(cadena_total2, numero2);
		strcat_s(cadena_total2, ".bmp");
		buffer2->Save(cadena_total2, "-format bmp", i);
		num_foto2++;
	}
	// CÁMARA 3
	{
		char numero3[5];
		sprintf_s(numero3, "%d", num_foto3);
		char cadena_total3[300];
```

```
                                        strcpy_s(cadena_total3, camara3_dir);
                                        strcat_s(cadena_total3, "camara3-foto");
                                        strcat_s(cadena_total3, numero3);
                                        strcat_s(cadena_total3, ".bmp");
                                        buffer3->Save(cadena_total3, "-format bmp", i);
                                        num_foto3++;
                                }
                                // CÁMARA 4
                                {
                                        char numero4[5];
                                        sprintf_s(numero4, "%d", num_foto4);
                                        char cadena_total4[300];
                                        strcpy_s(cadena_total4, camara4_dir);
                                        strcat_s(cadena_total4, "camara4-foto");
                                        strcat_s(cadena_total4, numero4);
                                        strcat_s(cadena_total4, ".bmp");
                                        buffer4->Save(cadena_total4, "-format bmp", i);
                                        num_foto4++;
                                }
                        }

                        printf("\nCaptura finalizada. ¿Desea tomar otras %d fotos (ESPACIO) o regresar al
menu (q)?\n", NUM_IMAGENES);
                        char op = getch();
                        if (op == 'q' || op == 'Q') {
                                break;
                        }
                        else if (op == 32) {
                                continue;
                        }
                        else {
                                printf("\nOpcion invalida, se regresa al menú principal.\n");
                                break;
                        }
                } // fin while(true) modo 1
        }
        //-----------------------------------------------------------------
        // MODO 2
        //-----------------------------------------------------------------
        else if (mode == 2)
        {
                currentMode = 2; // Set current mode

                strcpy_s(camara1_dir, calibracion_robot_dir);
                strcat_s(camara1_dir, "camara1\\");
                strcpy_s(camara2_dir, calibracion_robot_dir);
                strcat_s(camara2_dir, "camara2\\");
                strcpy_s(camara3_dir, calibracion_robot_dir);
                strcat_s(camara3_dir, "camara3\\");
                strcpy_s(camara4_dir, calibracion_robot_dir);
                strcat_s(camara4_dir, "camara4\\");

                if (!dirExists(camara1_dir)) { CreateDirectory(camara1_dir, NULL); }
                if (!dirExists(camara2_dir)) { CreateDirectory(camara2_dir, NULL); }
                if (!dirExists(camara3_dir)) { CreateDirectory(camara3_dir, NULL); }
                if (!dirExists(camara4_dir)) { CreateDirectory(camara4_dir, NULL); }

                acqDevice->SetFeatureValue("TriggerMode", "On");
                acqDevice->SetFeatureValue("TriggerSource", "Action1");
```

```
                        acqDevice2->SetFeatureValue("TriggerMode", "On");
                        acqDevice2->SetFeatureValue("TriggerSource", "Action1");
                        acqDevice3->SetFeatureValue("TriggerMode", "On");
                        acqDevice3->SetFeatureValue("TriggerSource", "Action1");
                        acqDevice4->SetFeatureValue("TriggerMode", "On");
                        acqDevice4->SetFeatureValue("TriggerSource", "Action1");

#ifdef USE_SCHEDULED_ACTION_COMMAND
                        // ...
#endif

                        while (true)
                        {
                                // Definir IDs
                                const int NUM_IDS = 4;
                                const char* calibration_ids[NUM_IDS] = { "ID1", "ID2", "ID3", "ID4" };
                                int current_id = 0;

                                printf("\nIniciando Calibracion del Robot (IDs: ID1, ID2, ID3, ID4).\n");
                                printf("Presione 'q' en cualquier momento para volver al menu.\n");

                                strcpy_s(auxtimestamps1_dir, camara1_dir);
                                strcat_s(auxtimestamps1_dir, "Camara1_AuxTimestamps.txt");
                                strcpy_s(auxtimestamps2_dir, camara2_dir);
                                strcat_s(auxtimestamps2_dir, "Camara2_AuxTimestamps.txt");
                                strcpy_s(auxtimestamps3_dir, camara3_dir);
                                strcat_s(auxtimestamps3_dir, "Camara3_AuxTimestamps.txt");
                                strcpy_s(auxtimestamps4_dir, camara4_dir);
                                strcat_s(auxtimestamps4_dir, "Camara4_AuxTimestamps.txt");

                                strcpy_s(hosttimestamps1_dir, camara1_dir);
                                strcat_s(hosttimestamps1_dir, "Camara1_HostTimestamps.txt");
                                strcpy_s(hosttimestamps2_dir, camara2_dir);
                                strcat_s(hosttimestamps2_dir, "Camara2_HostTimestamps.txt");
                                strcpy_s(hosttimestamps3_dir, camara3_dir);
                                strcat_s(hosttimestamps3_dir, "Camara3_HostTimestamps.txt");
                                strcpy_s(hosttimestamps4_dir, camara4_dir);
                                strcat_s(hosttimestamps4_dir, "Camara4_HostTimestamps.txt");

                                file1_aux.open(auxtimestamps1_dir, ios::out);
                                file2_aux.open(auxtimestamps2_dir, ios::out);
                                file3_aux.open(auxtimestamps3_dir, ios::out);
                                file4_aux.open(auxtimestamps4_dir, ios::out);
                                file1_host.open(hosttimestamps1_dir, ios::out);
                                file2_host.open(hosttimestamps2_dir, ios::out);
                                file3_host.open(hosttimestamps3_dir, ios::out);
                                file4_host.open(hosttimestamps4_dir, ios::out);

                                transfer->Grab();
                                transfer2->Grab();
                                transfer3->Grab();
                                transfer4->Grab();

                                // Bucle para cada ID
                                while (current_id < NUM_IDS)
                                {
                                        printf("\nColoque el robot sobre %s.\n", calibration_ids[current_id]);
                                        printf("Presione ESPACIO para capturar, 'q' para salir al menu.\n");
```

```
                char c = getch();
                if (c == 'q' || c == 'Q')
                {
                        printf("\nSaliendo de la Calibracion Robot...\n");
                        break;
                }
                else if (c == 32) // ESPACIO
                {
                        // -- CAMBIO: usaremos totalCapturasModo2 para la numeración
                        totalCapturasModo2++;
                        printf("\nNumero de captura: %d\n", totalCapturasModo2);

                        char action_cmd = '1';
                        sock_open = send_Action_cmd(NICinfos, action_cmd, sock_open);
                        current_id++;
                }
                else {
                        printf("Opcion invalida.\n");
                }
        }

        transfer->Freeze();
        transfer2->Freeze();
        transfer3->Freeze();
        transfer4->Freeze();

        file1_aux.close();
        file2_aux.close();
        file3_aux.close();
        file4_aux.close();
        file1_host.close();
        file2_host.close();
        file3_host.close();
        file4_host.close();

        if (current_id < NUM_IDS)
        {
                printf("\nNo se completo la captura de las 4 IDs.\n");
        }
        else
        {
                printf("\nCalibracion Robot completada para las 4 IDs.\n");
        }

        // Guardar imágenes
        int lastIndex = buffer->GetIndex();
        const char* calibration_ids_save[4] = { "ID1", "ID2", "ID3", "ID4" };
        int current_id_save = 0;

        for (int i = 0; i <= lastIndex; i++) {
                if (current_id_save >= 4) current_id_save = 3;

                char camara1_filename[300];
                sprintf_s(camara1_filename,          "%scamara1-%s.bmp",          camara1_dir,
calibration_ids_save[current_id_save]);
                buffer->Save(camara1_filename, "-format bmp", i);

                char camara2_filename[300];
```

```
                                        sprintf_s(camara2_filename, "%scamara2-%s.bmp", camara2_dir,
calibration_ids_save[current_id_save]);
                                        buffer2->Save(camara2_filename, "-format bmp", i);

                                        char camara3_filename[300];
                                        sprintf_s(camara3_filename, "%scamara3-%s.bmp", camara3_dir,
calibration_ids_save[current_id_save]);
                                        buffer3->Save(camara3_filename, "-format bmp", i);

                                        char camara4_filename[300];
                                        sprintf_s(camara4_filename, "%scamara4-%s.bmp", camara4_dir,
calibration_ids_save[current_id_save]);
                                        buffer4->Save(camara4_filename, "-format bmp", i);

                                        current_id_save++;
                                        if (current_id_save >= 4) current_id_save = 3;
                                }

                                printf("\n¿Desea repetir la calibracion (ESPACIO) o volver al menu (q)?\n");
                                char op2 = getch();
                                if (op2 == 'q' || op2 == 'Q') {
                                        break;
                                }
                                else if (op2 == 32) {
                                        continue;
                                }
                                else {
                                        printf("\nOpcion invalida, regresando al menú.\n");
                                        break;
                                }
                        } // fin while(true) modo 2
                }
                //-----------------------------------------------------------------
                // MODO 3
                //-----------------------------------------------------------------
                else if (mode == 3)
                {
                        currentMode = 3; // Set current mode

                        strcpy_s(camara1_dir, calibracion_nuevo_marc);
                        strcat_s(camara1_dir, "camara1\\");
                        strcpy_s(camara2_dir, calibracion_nuevo_marc);
                        strcat_s(camara2_dir, "camara2\\");
                        strcpy_s(camara3_dir, calibracion_nuevo_marc);
                        strcat_s(camara3_dir, "camara3\\");
                        strcpy_s(camara4_dir, calibracion_nuevo_marc);
                        strcat_s(camara4_dir, "camara4\\");

                        if (!dirExists(camara1_dir)) { CreateDirectory(camara1_dir, NULL); }
                        if (!dirExists(camara2_dir)) { CreateDirectory(camara2_dir, NULL); }
                        if (!dirExists(camara3_dir)) { CreateDirectory(camara3_dir, NULL); }
                        if (!dirExists(camara4_dir)) { CreateDirectory(camara4_dir, NULL); }

                        acqDevice->SetFeatureValue("TriggerMode", "On");
                        acqDevice->SetFeatureValue("TriggerSource", "Action1");
                        acqDevice2->SetFeatureValue("TriggerMode", "On");
                        acqDevice2->SetFeatureValue("TriggerSource", "Action1");
                        acqDevice3->SetFeatureValue("TriggerMode", "On");
```

```
                        acqDevice3->SetFeatureValue("TriggerSource", "Action1");
                        acqDevice4->SetFeatureValue("TriggerMode", "On");
                        acqDevice4->SetFeatureValue("TriggerSource", "Action1");

#ifdef USE_SCHEDULED_ACTION_COMMAND
                        // ...
#endif

                        while (true)
                        {
                                printf("\nIniciando adquisicion en Modo detectar nuevos ids. Se tomaran %d imagen(es).\n", NUM_IMAGENES);

                                strcpy_s(auxtimestamps1_dir, camara1_dir);
                                strcat_s(auxtimestamps1_dir, "Camara1_AuxTimestamps.txt");
                                strcpy_s(auxtimestamps2_dir, camara2_dir);
                                strcat_s(auxtimestamps2_dir, "Camara2_AuxTimestamps.txt");
                                strcpy_s(auxtimestamps3_dir, camara3_dir);
                                strcat_s(auxtimestamps3_dir, "Camara3_AuxTimestamps.txt");
                                strcpy_s(auxtimestamps4_dir, camara4_dir);
                                strcat_s(auxtimestamps4_dir, "Camara4_AuxTimestamps.txt");

                                strcpy_s(hosttimestamps1_dir, camara1_dir);
                                strcat_s(hosttimestamps1_dir, "Camara1_HostTimestamps.txt");
                                strcpy_s(hosttimestamps2_dir, camara2_dir);
                                strcat_s(hosttimestamps2_dir, "Camara2_HostTimestamps.txt");
                                strcpy_s(hosttimestamps3_dir, camara3_dir);
                                strcat_s(hosttimestamps3_dir, "Camara3_HostTimestamps.txt");
                                strcpy_s(hosttimestamps4_dir, camara4_dir);
                                strcat_s(hosttimestamps4_dir, "Camara4_HostTimestamps.txt");

                                file1_aux.open(auxtimestamps1_dir, ios::out);
                                file2_aux.open(auxtimestamps2_dir, ios::out);
                                file3_aux.open(auxtimestamps3_dir, ios::out);
                                file4_aux.open(auxtimestamps4_dir, ios::out);
                                file1_host.open(hosttimestamps1_dir, ios::out);
                                file2_host.open(hosttimestamps2_dir, ios::out);
                                file3_host.open(hosttimestamps3_dir, ios::out);
                                file4_host.open(hosttimestamps4_dir, ios::out);

                                transfer->Grab();
                                transfer2->Grab();
                                transfer3->Grab();
                                transfer4->Grab();

                                for (int i = 0; i < NUM_IMAGENES; i++)
                                {
                                        totalCapturasModo3++;
                                        printf("\nNumero de captura : %d\n", totalCapturasModo3);

                                        char action_cmd = '1';
                                        sock_open = send_Action_cmd(NICinfos, action_cmd, sock_open);
                                }

                                transfer->Freeze();
                                transfer2->Freeze();
                                transfer3->Freeze();
                                transfer4->Freeze();
```

```
file1_aux.close();
file2_aux.close();
file3_aux.close();
file4_aux.close();
file1_host.close();
file2_host.close();
file3_host.close();
file4_host.close();

int lastIndex = buffer->GetIndex();
for (int i = 0; i <= lastIndex; i++) {
	// CÁMARA 1
	{
		char numero[5];
		sprintf_s(numero, "%d", num_foto1);
		char cadena_total[300];
		strcpy_s(cadena_total, camara1_dir);
		strcat_s(cadena_total, "camara1-foto");
		strcat_s(cadena_total, numero);
		strcat_s(cadena_total, ".bmp");
		buffer->Save(cadena_total, "-format bmp", i);
		num_foto1++;
	}
	// CÁMARA 2
	{
		char numero2[5];
		sprintf_s(numero2, "%d", num_foto2);
		char cadena_total2[300];
		strcpy_s(cadena_total2, camara2_dir);
		strcat_s(cadena_total2, "camara2-foto");
		strcat_s(cadena_total2, numero2);
		strcat_s(cadena_total2, ".bmp");
		buffer2->Save(cadena_total2, "-format bmp", i);
		num_foto2++;
	}
	// CÁMARA 3
	{
		char numero3[5];
		sprintf_s(numero3, "%d", num_foto3);
		char cadena_total3[300];
		strcpy_s(cadena_total3, camara3_dir);
		strcat_s(cadena_total3, "camara3-foto");
		strcat_s(cadena_total3, numero3);
		strcat_s(cadena_total3, ".bmp");
		buffer3->Save(cadena_total3, "-format bmp", i);
		num_foto3++;
	}
	// CÁMARA 4
	{
		char numero4[5];
		sprintf_s(numero4, "%d", num_foto4);
		char cadena_total4[300];
		strcpy_s(cadena_total4, camara4_dir);
		strcat_s(cadena_total4, "camara4-foto");
		strcat_s(cadena_total4, numero4);
		strcat_s(cadena_total4, ".bmp");
		buffer4->Save(cadena_total4, "-format bmp", i);
		num_foto4++;
```

```
                                        }
                                }

                                printf("\n\nModo detectar nuevos ids: ¿Tomar %d cuadros de nuevo (ESPACIO) o regresar al menu (q)?\n", NUM_IMAGENES);
                                char op3 = getch();
                                if (op3 == 'q' || op3 == 'Q') {
                                        break;
                                }
                                else if (op3 == 32) {
                                        continue;
                                }
                                else {
                                        printf("\nSeleccion invalida. Regresando al menú principal.\n");
                                        break;
                                }
                        } // fin while(true) modo 3
                }
                //---------------------------------------------------------------
                // MODO 4
                //---------------------------------------------------------------
                else if (mode == 4)
                {
                        currentMode = 4; // Set current mode

                        strcpy_s(camara1_dir, modo_real_dir);
                        strcat_s(camara1_dir, "camara1\\");
                        strcpy_s(camara2_dir, modo_real_dir);
                        strcat_s(camara2_dir, "camara2\\");
                        strcpy_s(camara3_dir, modo_real_dir);
                        strcat_s(camara3_dir, "camara3\\");
                        strcpy_s(camara4_dir, modo_real_dir);
                        strcat_s(camara4_dir, "camara4\\");

                        if (!dirExists(camara1_dir)) { CreateDirectory(camara1_dir, NULL); }
                        if (!dirExists(camara2_dir)) { CreateDirectory(camara2_dir, NULL); }
                        if (!dirExists(camara3_dir)) { CreateDirectory(camara3_dir, NULL); }
                        if (!dirExists(camara4_dir)) { CreateDirectory(camara4_dir, NULL); }

                        acqDevice->SetFeatureValue("TriggerMode", "On");
                        acqDevice->SetFeatureValue("TriggerSource", "Software");
                        acqDevice2->SetFeatureValue("TriggerMode", "On");
                        acqDevice2->SetFeatureValue("TriggerSource", "Software");
                        acqDevice3->SetFeatureValue("TriggerMode", "On");
                        acqDevice3->SetFeatureValue("TriggerSource", "Software");
                        acqDevice4->SetFeatureValue("TriggerMode", "On");
                        acqDevice4->SetFeatureValue("TriggerSource", "Software");

                        acqDevice->RegisterCallback("FrameStart", MyAcquisitionCallback1, &buffer);
                        acqDevice2->RegisterCallback("FrameStart", MyAcquisitionCallback2, &buffer2);
                        acqDevice3->RegisterCallback("FrameStart", MyAcquisitionCallback3, &buffer3);
                        acqDevice4->RegisterCallback("FrameStart", MyAcquisitionCallback4, &buffer4);

                        SelectCamera();
                        switch (selectedCameraIndex) {
                        case 1: view = new SapView(buffer, SapHwndAutomatic);   break;
                        case 2: view = new SapView(buffer2, SapHwndAutomatic);   break;
                        case 3: view = new SapView(buffer3, SapHwndAutomatic);   break;
                        case 4: view = new SapView(buffer4, SapHwndAutomatic);   break;
```

```
                    default:view = new SapView(buffer, SapHwndAutomatic);   break;
                    }
                    view->Create();

                    transfer->Grab();
                    transfer2->Grab();
                    transfer3->Grab();
                    transfer4->Grab();

                    printf("\nCaptura en tiempo real. Presione 'q' para detener y volver al menu.\n");

                    while (true)
                    {
                            // Disparo por software
                            acqDevice->SetFeatureValue("TriggerSoftware", 1);
                            acqDevice2->SetFeatureValue("TriggerSoftware", 1);
                            acqDevice3->SetFeatureValue("TriggerSoftware", 1);
                            acqDevice4->SetFeatureValue("TriggerSoftware", 1);

                            Sleep(50);

                            if (_kbhit())
                            {
                                    char c = getch();
                                    if (c == 'q' || c == 'Q')
                                    {
                                            printf("\nDeteniendo captura tiempo real.\n");
                                            break;
                                    }
                            }
                    }

                    transfer->Freeze();
                    transfer2->Freeze();
                    transfer3->Freeze();
                    transfer4->Freeze();

                    if (view) {
                            view->Destroy();
                            delete view;
                            view = nullptr;
                    }
            }

} // fin while(1) principal

// Cleanup Sapera resources
transfer->Destroy();
delete transfer;
buffer->Destroy();
delete buffer;
acqDevice->Destroy();
delete acqDevice;

transfer2->Destroy();
delete transfer2;
buffer2->Destroy();
delete buffer2;
```

```
	acqDevice2->Destroy();
	delete acqDevice2;

	transfer3->Destroy();
	delete transfer3;
	buffer3->Destroy();
	delete buffer3;
	acqDevice3->Destroy();
	delete acqDevice3;

	transfer4->Destroy();
	delete transfer4;
	buffer4->Destroy();
	delete buffer4;
	acqDevice4->Destroy();
	delete acqDevice4;
	SapManager::Close();

	return 0;
}


//=============================================================
// FUNCTIONS
//=============================================================
static void MyAcquisitionCallback1(SapAcqDeviceCallbackInfo* pInfo)
{
	if (currentMode != 4) return; // Solo en modo 4

	static int bufferIndex = 0;
	SapBuffer* buf = *(SapBuffer**)pInfo->GetContext();

	char filename[260];
	switch (bufferIndex)
	{
	case 0: sprintf_s(filename, sizeof(filename), "camara1_image1.bmp"); break;
	case 1: sprintf_s(filename, sizeof(filename), "camara1_image2.bmp"); break;
	case 2: sprintf_s(filename, sizeof(filename), "camara1_image3.bmp"); break;
	default:sprintf_s(filename, sizeof(filename), "camara1_image1.bmp"); break;
	}

	char image_file_full_path[300];
	sprintf_s(image_file_full_path, sizeof(image_file_full_path), "%s%s", camara1_dir, filename);

	char temp_image_file[300];
	sprintf_s(temp_image_file, sizeof(temp_image_file), "%stemp.bmp", camara1_dir);
	char status_file[300];
	sprintf_s(status_file, sizeof(status_file), "%scamara1_latest.txt", camara1_dir);
	char temp_status_file[300];
	sprintf_s(temp_status_file, sizeof(temp_status_file), "%stemp_status.txt", camara1_dir);

	if (!buf->Save(temp_image_file, "-format bmp"))
	{
		fprintf(stderr, "Error al guardar imagen temporal\n");
		return;
	}

	if (remove(image_file_full_path) != 0)
	{
```

```
                if (errno != ENOENT)
                {
                        perror("Error al eliminar archivo destino existente");
                        return;
                }
        }
        if (rename(temp_image_file, image_file_full_path) != 0)
        {
                perror("Error al renombrar archivo temporal");
                return;
        }

        FILE* f = fopen(temp_status_file, "w");
        if (f)
        {
                fprintf(f, "%s", filename);
                fclose(f);

                if (remove(status_file) != 0)
                {
                        if (errno != ENOENT)
                        {
                                perror("Error al eliminar status file existente");
                                return;
                        }
                }
                if (rename(temp_status_file, status_file) != 0)
                {
                        perror("Error al renombrar status file temporal");
                        return;
                }
        }
        else
        {
                perror("Error al abrir archivo de estado temporal");
                return;
        }

        bufferIndex = (bufferIndex + 1) % 3;
        if (selectedCameraIndex == 1 && view) {
                view->Show();
        }
}

static void MyAcquisitionCallback2(SapAcqDeviceCallbackInfo* pInfo)
{
        if (currentMode != 4) return;

        static int bufferIndex = 0;
        SapBuffer* buf = *(SapBuffer**)pInfo->GetContext();

        char filename[260];
        switch (bufferIndex)
        {
        case 0: sprintf_s(filename, sizeof(filename), "camara2_image1.bmp"); break;
        case 1: sprintf_s(filename, sizeof(filename), "camara2_image2.bmp"); break;
        case 2: sprintf_s(filename, sizeof(filename), "camara2_image3.bmp"); break;
        default:sprintf_s(filename, sizeof(filename), "camara2_image1.bmp"); break;
```

```
}

char image_file_full_path[300];
sprintf_s(image_file_full_path, sizeof(image_file_full_path), "%s%s", camara2_dir, filename);

char temp_image_file[300];
sprintf_s(temp_image_file, sizeof(temp_image_file), "%stemp.bmp", camara2_dir);
char status_file[300];
sprintf_s(status_file, sizeof(status_file), "%scamara2_latest.txt", camara2_dir);
char temp_status_file[300];
sprintf_s(temp_status_file, sizeof(temp_status_file), "%stemp_status.txt", camara2_dir);

if (!buf->Save(temp_image_file, "-format bmp"))
{
	fprintf(stderr, "Error al guardar imagen temporal\n");
	return;
}

if (remove(image_file_full_path) != 0)
{
	if (errno != ENOENT)
	{
		perror("Error al eliminar archivo destino existente");
		return;
	}
}
if (!MoveFileEx(temp_image_file, image_file_full_path, MOVEFILE_REPLACE_EXISTING))
{
	DWORD error = GetLastError();
	fprintf(stderr, "Error al mover archivo temporal: %lu\n", error);
	return;
}

FILE* f = fopen(temp_status_file, "w");
if (f)
{
	fprintf(f, "%s", filename);
	fclose(f);

	if (remove(status_file) != 0)
	{
		if (errno != ENOENT)
		{
			perror("Error al eliminar status file existente");
			return;
		}
	}
	if (!MoveFileEx(temp_status_file, status_file, MOVEFILE_REPLACE_EXISTING))
	{
		DWORD error = GetLastError();
		fprintf(stderr, "Error al renombrar status file temporal: %lu\n", error);
		return;
	}
}
else
{
	perror("Error al abrir el archivo de estado temporal");
	return;
}
```

```
	bufferIndex = (bufferIndex + 1) % 3;
	if (selectedCameraIndex == 2 && view) {
		view->Show();
	}
}

static void MyAcquisitionCallback3(SapAcqDeviceCallbackInfo* pInfo)
{
	if (currentMode != 4) return;

	static int bufferIndex = 0;
	SapBuffer* buf = *(SapBuffer**)pInfo->GetContext();

	char filename[260];
	switch (bufferIndex)
	{
	case 0: sprintf_s(filename, sizeof(filename), "camara3_image1.bmp"); break;
	case 1: sprintf_s(filename, sizeof(filename), "camara3_image2.bmp"); break;
	case 2: sprintf_s(filename, sizeof(filename), "camara3_image3.bmp"); break;
	default:sprintf_s(filename, sizeof(filename), "camara3_image1.bmp"); break;
	}

	char image_file_full_path[300];
	sprintf_s(image_file_full_path, sizeof(image_file_full_path), "%s%s", camara3_dir, filename);

	char temp_image_file[300];
	sprintf_s(temp_image_file, sizeof(temp_image_file), "%stemp.bmp", camara3_dir);
	char status_file[300];
	sprintf_s(status_file, sizeof(status_file), "%scamara3_latest.txt", camara3_dir);
	char temp_status_file[300];
	sprintf_s(temp_status_file, sizeof(temp_status_file), "%stemp_status.txt", camara3_dir);

	if (!buf->Save(temp_image_file, "-format bmp"))
	{
		fprintf(stderr, "Error al guardar imagen temporal\n");
		return;
	}

	if (remove(image_file_full_path) != 0)
	{
		if (errno != ENOENT)
		{
			perror("Error al eliminar archivo destino existente");
			return;
		}
	}
	if (!MoveFileEx(temp_image_file, image_file_full_path, MOVEFILE_REPLACE_EXISTING))
	{
		DWORD error = GetLastError();
		fprintf(stderr, "Error al mover archivo temporal: %lu\n", error);
		return;
	}

	FILE* f = fopen(temp_status_file, "w");
	if (f)
	{
		fprintf(f, "%s", filename);
```

```
                fclose(f);

                if (remove(status_file) != 0)
                {
                        if (errno != ENOENT)
                        {
                                perror("Error al eliminar status file existente");
                                return;
                        }
                }
                if (!MoveFileEx(temp_status_file, status_file, MOVEFILE_REPLACE_EXISTING))
                {
                        DWORD error = GetLastError();
                        fprintf(stderr, "Error al renombrar archivo de estado temporal: %lu\n", error);
                        return;
                }
        }
        else
        {
                perror("Error al abrir archivo de estado temporal");
                return;
        }

        bufferIndex = (bufferIndex + 1) % 3;
        if (selectedCameraIndex == 3 && view) {
                view->Show();
        }
}

static void MyAcquisitionCallback4(SapAcqDeviceCallbackInfo* pInfo)
{
        if (currentMode != 4) return;

        static int bufferIndex = 0;
        SapBuffer* buf = *(SapBuffer**)pInfo->GetContext();

        char filename[260];
        switch (bufferIndex)
        {
        case 0: sprintf_s(filename, sizeof(filename), "camara4_image1.bmp"); break;
        case 1: sprintf_s(filename, sizeof(filename), "camara4_image2.bmp"); break;
        case 2: sprintf_s(filename, sizeof(filename), "camara4_image3.bmp"); break;
        default:sprintf_s(filename, sizeof(filename), "camara4_image1.bmp"); break;
        }

        char image_file_full_path[300];
        sprintf_s(image_file_full_path, sizeof(image_file_full_path), "%s%s", camara4_dir, filename);

        char temp_image_file[300];
        sprintf_s(temp_image_file, sizeof(temp_image_file), "%stemp.bmp", camara4_dir);
        char status_file[300];
        sprintf_s(status_file, sizeof(status_file), "%scamara4_latest.txt", camara4_dir);
        char temp_status_file[300];
        sprintf_s(temp_status_file, sizeof(temp_status_file), "%stemp_status.txt", camara4_dir);

        if (!buf->Save(temp_image_file, "-format bmp"))
        {
                fprintf(stderr, "Error al guardar imagen temporal\n");
                return;
```

```
	}

	if (remove(image_file_full_path) != 0)
	{
		if (errno != ENOENT)
		{
			perror("Error al eliminar archivo destino existente");
			return;
		}
	}
	if (!MoveFileEx(temp_image_file, image_file_full_path, MOVEFILE_REPLACE_EXISTING))
	{
		DWORD error = GetLastError();
		fprintf(stderr, "Error al mover archivo temporal: %lu\n", error);
		return;
	}

	FILE* f = fopen(temp_status_file, "w");
	if (f)
	{
		fprintf(f, "%s", filename);
		fclose(f);

		if (remove(status_file) != 0)
		{
			if (errno != ENOENT)
			{
				perror("Error al eliminar status file existente");
				return;
			}
		}
		if (!MoveFileEx(temp_status_file, status_file, MOVEFILE_REPLACE_EXISTING))
		{
			DWORD error = GetLastError();
			fprintf(stderr, "Error al renombrar el archivo de estado temporal: %lu\n", error);
			return;
		}
	}
	else
	{
		perror("Error al abrir archivo de estado temporal");
		return;
	}

	bufferIndex = (bufferIndex + 1) % 3;
	if (selectedCameraIndex == 4 && view) {
		view->Show();
	}
}

void SetCameraStartTime(SapAcqDevice* m_AcqDevice, UINT64 starttime)
{
	BOOL bIsAvailable = FALSE;
	if (m_AcqDevice->IsFeatureAvailable("timestampModuloStartTime", &bIsAvailable) && bIsAvailable)
	{
		m_AcqDevice->SetFeatureValue("timestampModuloStartTime", starttime);
	}
}
```

```
void ResetCameraTimestamp(SapAcqDevice* m_AcqDevice)
{
	BOOL bIsAvailable = FALSE;
	if (m_AcqDevice->IsFeatureAvailable("ptpMode", &bIsAvailable) && bIsAvailable)
	{
		m_AcqDevice->SetFeatureValue("ptpMode", "Off");

		BOOL bIsAvailable2 = FALSE;
		if (m_AcqDevice->IsFeatureAvailable("timestampControlReset", &bIsAvailable2) && bIsAvailable2)
		{
			m_AcqDevice->SetFeatureValue("timestampControlReset", 1);
		}

		BOOL bIsAvailable3 = FALSE;
		if (m_AcqDevice->IsFeatureAvailable("ptpMode", &bIsAvailable3) && bIsAvailable3)
		{
			m_AcqDevice->SetFeatureValue("ptpMode", "Automatic");
		}
	}
}

tNIC init_NICinfos()
{
	tNIC NICinfos;
	NICinfos.network_name = NULL;
	NICinfos.NICIP = NULL;
	return NICinfos;
}

BOOL get_Adapters_Info(tNIC* NICinfos)
{
	ULONG flags = GAA_FLAG_INCLUDE_PREFIX;
	ULONG outBufLen = 15000;
	ULONG ulOutBufLen = sizeof(IP_ADAPTER_INFO);

	PIP_ADAPTER_ADDRESSES pAddresses = NULL;
	PIP_ADAPTER_ADDRESSES pCurrAddresses = NULL;
	PIP_ADAPTER_INFO pAdapterInfo = NULL;
	PIP_ADAPTER_INFO pAdapter = NULL;

	DWORD dwRetVal = 0;
	int i = 0;
	int KEYval;

	pAdapterInfo = (IP_ADAPTER_INFO*)MALLOC(sizeof(IP_ADAPTER_INFO));
	if (pAdapterInfo == NULL)
	{
		printf("\nError allocating memory needed to call GetAdaptersinfo\n");
		return FALSE;
	}

	if (GetAdaptersInfo(pAdapterInfo, &ulOutBufLen) == ERROR_BUFFER_OVERFLOW)
	{
		FREE(pAdapterInfo);
		pAdapterInfo = (IP_ADAPTER_INFO*)MALLOC(ulOutBufLen);
		if (pAdapterInfo == NULL)
		{
			printf("\nError allocating memory needed to call GetAdaptersinfo\n");
```

```
                return FALSE;
        }
}

do
{
        pAddresses = (IP_ADAPTER_ADDRESSES*)MALLOC(outBufLen);
        if (pAddresses == NULL)
        {
                printf("\nMemory allocation failed for IP_ADAPTER_ADDRESSES struct\n");
                return FALSE;
        }

        dwRetVal = GetAdaptersAddresses(AF_INET, flags, NULL, pAddresses, &outBufLen);
        if (dwRetVal == ERROR_BUFFER_OVERFLOW)
        {
                FREE(pAddresses);
                pAddresses = NULL;
        }
        else
                break;

        i++;
} while ((dwRetVal == ERROR_BUFFER_OVERFLOW) && (i < 3));

i = 0;
pCurrAddresses = pAddresses;
if ((dwRetVal = GetAdaptersInfo(pAdapterInfo, &ulOutBufLen)) == NO_ERROR)
{
        pAdapter = pAdapterInfo;
        while (pCurrAddresses && pAdapter)
        {
                if (pAdapter->Type == MIB_IF_TYPE_ETHERNET)
                        i++;

                pAdapter = pAdapter->Next;
                pCurrAddresses = pCurrAddresses->Next;
        }

        printf("\n\nSelecciona una de las redes detectadas:\n");

        if (i > 1)
                printf("Presiona 1 a %d o 'q' para salir.\n", i);
        else if (i == 1)
                printf("Presiona 1 o 'q' para salir.\n");
        else
                printf("\n\nNinguna red detectada, presiona 'q' para salir.\n");

        printf("..................................................\n");

        pCurrAddresses = pAddresses;
        pAdapter = pAdapterInfo;
        i = 1;

        while (pCurrAddresses && pAdapter)
        {
                if (pAdapter->Type == MIB_IF_TYPE_ETHERNET)
                {
```

```
                                printf("%d. %wS\n", i, pCurrAddresses->FriendlyName);
                                printf("    IP : %s\n", pAdapter->IpAddressList.IpAddress.String);
                                i++;
                        }
                        pAdapter = pAdapter->Next;
                        pCurrAddresses = pCurrAddresses->Next;
                }
        }
        else
                printf("GetAdaptersInfo failed with error: %d\n", dwRetVal);

        printf("..........\n");

        while (1)
        {
                KEYval = (char)getch();

                if (KEYval == 'q' || KEYval == 'Q')
                        return FALSE;

                pCurrAddresses = pAddresses;
                pAdapter = pAdapterInfo;

                int asciiIndex = 49; // '1'
                while (pCurrAddresses && pAdapter)
                {
                        if (pAdapter->Type == MIB_IF_TYPE_ETHERNET)
                        {
                                if (asciiIndex == KEYval)
                                {
                                        printf("\n La red '%wS' ha sido seleccionada.\n", pCurrAddresses-
>FriendlyName);
                                        NICinfos->network_name = pCurrAddresses->FriendlyName;
                                        NICinfos->NICIP = pAdapter->IpAddressList.IpAddress.String;
                                        return TRUE;
                                }
                                else
                                        asciiIndex++;
                        }
                        pAdapter = pAdapter->Next;
                        pCurrAddresses = pCurrAddresses->Next;
                }
                printf("Seleccion invalida!\nPor favor, elige otra o 'q' para salir.\n");
        }
}

BOOL send_Action_cmd(tNIC NICinfos, char cmd, BOOL sock_open)
{
        if (NICinfos.NICIP == NULL)
        {
                printf("Error: La direccion IP de la interfaz de red es Nula.\n");
                return FALSE;
        }

        if (!init_sockAPI())
                return FALSE;

        if (!sock_open)
        {
```

```
		if ((sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == SOCKET_ERROR)
		{
			printf("socket() failed with error code : %d", WSAGetLastError());
			return FALSE;
		}

		srand(time(NULL));
		UINT16 source_port = (rand() % 0x4000) + 0xC000;

		sock_source.sin_family = PF_INET;
		sock_source.sin_port = htons((UINT16)source_port);
		sock_source.sin_addr.S_un.S_addr = inet_addr(NICinfos.NICIP);

		if (bind(sock, (struct sockaddr*)&sock_source, sizeof(struct sockaddr_in)) == SOCKET_ERROR)
			printf("ERROR: Unable to bind UDP socket (error %d)\n\n\b", WSAGetLastError());
		else
			sock_open = TRUE;

		sock_destination.sin_family = PF_INET;
		sock_destination.sin_port = htons(GVCP_PORT);
		sock_destination.sin_addr.s_addr = inet_addr(NICinfos.NICIP);
	}

	FillCommandHeader(datagram_out);
	FillCommandPayload(datagram_out, cmd);

	broadcastMessage(datagram_out, GVCP_PACKET_SIZE, &sock, &sock_destination);

	struct timeval timeout;
	timeout.tv_sec = 0;
	timeout.tv_usec = 16666;

	while (receiveMessage(datagram_in, BUFLEN, &sock, &timeout));

	if (nb_ack > 0)
	{
		increment_req_id();
	}

	nb_ack = 0;
	nb_non_ack = 0;

	return sock_open;
}

BOOL init_sockAPI()
{
	WSADATA wsa;
	if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
	{
		printf("Failed. Error Code : %d", WSAGetLastError());
		return FALSE;
	}
	return TRUE;
}

BOOL broadcastMessage(UINT8* datagram, int size, SOCKET* sock, struct sockaddr_in* dest)
{
```

```
	int option = 1;
	setsockopt(*sock, SOL_SOCKET, SO_BROADCAST, (char*)&option, sizeof(option));

	struct sockaddr_in destination = *dest;
	destination.sin_addr.s_addr = INADDR_BROADCAST;

	if (!sendMessage(datagram, size, sock, &destination))
		return FALSE;

	option = 0;
	setsockopt(*sock, SOL_SOCKET, SO_BROADCAST, (char*)&option, sizeof(option));

	return TRUE;
}

BOOL sendMessage(UINT8* datagram, int size, SOCKET* sock, struct sockaddr_in* dest)
{
	int nb_bytes;
	int addrLen = sizeof(struct sockaddr_in);

	nb_bytes = sendto(*sock, (const char*)datagram, size, 0, (struct sockaddr*)dest, addrLen);
	if (nb_bytes < 0)
	{
		printf("ERROR: Error in sendto (error %d)\n\n\b", WSAGetLastError());
		return FALSE;
	}
	return TRUE;
}

BOOL receiveMessage(UINT8* datagram, int maxSize, SOCKET* sock, struct timeval* timeout)
{
	fd_set setRead;
	int notused = 0;
	int cnt;
	struct sockaddr_in sa;
	int sa_len = sizeof(sa);

	FD_ZERO(&setRead);
	FD_SET(*sock, &setRead);

	cnt = select(notused, &setRead, NULL, NULL, timeout);
	if (cnt == SOCKET_ERROR)
	{
		printf("\nERROR: select() returned an error (error %d)\n\n\b", WSAGetLastError());
		return FALSE;
	}

	if (cnt > 0)
	{
		if (FD_ISSET(*sock, &setRead))
		{
			memset(&sa, 0, sizeof(struct sockaddr_in));
			cnt = recvfrom(*sock, (char*)datagram, maxSize, 0, (struct sockaddr*)&sa, &sa_len);
			if (cnt > 0)
			{
				if (validate_ack(datagram))
					nb_ack++;
				else
					nb_non_ack++;
```

```
                        }
                }
        }
        else
                return FALSE;

        return TRUE;
}

void FillCommandHeader(UINT8* datagram)
{
        if (length[1] == 0x0C)
                GVCP_PACKET_SIZE = 20;
        else if (length[1] == 0x14)
                GVCP_PACKET_SIZE = 28;

        BOOL idOK = FALSE;
        for (int i = 0; i < GVCP_ID_LENGTH; i++)
        {
                if (req_id[i] != 0)
                        idOK = TRUE;
        }
        if (!idOK)
                req_id[1] = 0x01;

        memset(datagram, 0, GVCP_PACKET_SIZE);

        for (int i = GVCP_KEY_OFFSET; i < GVCP_KEY_LENGTH; i++)
                datagram[i] = key[i];

        int i, j;
        for (i = GVCP_FLAG_OFFSET, j = 0; i < GVCP_COMMAND_OFFSET && j < GVCP_FLAG_LENGTH; i++, j++)
                datagram[i] = flag[j];

        for (i = GVCP_COMMAND_OFFSET, j = 0; i < GVCP_LENGTH_OFFSET && j < GVCP_COMMAND_LENGTH;
i++, j++)
                datagram[i] = cmd[j];

        for (i = GVCP_LENGTH_OFFSET, j = 0; i < GVCP_ID_OFFSET && j < GVCP_LENGTH_LENGTH; i++, j++)
                datagram[i] = length[j];

        for (i = GVCP_ID_OFFSET, j = 0; i < GVCP_HEADER_SIZE && j < GVCP_ID_LENGTH; i++, j++)
                datagram[i] = req_id[j];
}

void FillCommandPayload(UINT8* datagram, char cmd_char)
{
        UINT8 cmd_hex = 0x00;
        switch (cmd_char)
        {
        case '1': cmd_hex = 0x01; break;
        default:
                cmd_hex = 0x01;
                break;
        }
        UINT8 group_mask[GROUP_MASK_LENGTH] = { 0x00, 0x00, 0x00, cmd_hex };

        for (int i = DEVICE_KEY_OFFSET, j = 0; i < GROUP_KEY_OFFSET && j < DEVICE_KEY_LENGTH; i++, j++)
```

```
		datagram[i] = device_key[j];

	for (int i = GROUP_KEY_OFFSET, j = 0; i < GROUP_MASK_OFFSET && j < GROUP_KEY_LENGTH; i++, j++)
		datagram[i] = group_key[j];

	for (int i = GROUP_MASK_OFFSET, j = 0; i < ACTION_TIME_H_OFFSET && j < GROUP_MASK_LENGTH; i++,
j++)
		datagram[i] = group_mask[j];

	if (GVCP_PACKET_SIZE == 28)
	{
		for (int i = ACTION_TIME_H_OFFSET, j = 0; i < ACTION_TIME_L_OFFSET && j <
ACTION_TIME_LENGTH; i++, j++)
			datagram[i] = action_time_H[j];

		for (int i = ACTION_TIME_L_OFFSET, j = 0; i < GVCP_PACKET_SIZE && j < ACTION_TIME_LENGTH;
i++, j++)
			datagram[i] = action_time_L[j];
	}
}

void increment_req_id()
{
	if (req_id[1] == 0xFF)
	{
		req_id[0]++;
		req_id[1] = 0x00;
	}
	else
		req_id[1]++;
}

BOOL validate_ack(UINT8* datagram)
{
	if (datagram[0] == GVCP_STATUS_H && datagram[1] == GVCP_STATUS_L)
	{
		if (datagram[2] == GVCP_ACK_H && datagram[3] == GVCP_ACK_L)
			if (datagram[4] == GVCP_LENGTH_H && datagram[5] == GVCP_LENGTH_L)
				if (datagram[6] == req_id[0] && datagram[7] == req_id[1])
					return TRUE;
	}
	return FALSE;
}
```

Código optimizado con Inteligencia Artificial. [76]

d. ANEXO 4: control_robot.py

```
#!/usr/bin/env python3

#librerias
import cv2
import numpy as np
import matplotlib.pyplot as plt
import cv2.aruco as aruco
import rospy
from geometry_msgs.msg import Twist
import time
from collections import deque
import sys
import os
import threading
from nav_msgs.msg import Odometry
import csv
from datetime import datetime
os.environ["SDL_AUDIODRIVER"] = "dummy"
os.environ["PYGAME_HIDE_SUPPORT_PROMPT"] = "1"
import pygame
from pygame.locals import *
import ast
import rosgraph
import math

# ----------------------------
# Clases (PID, actualización y dirección)
# ----------------------------

class PIDController:
    def __init__(self,
                 Kp, Ki, Kd,
                 setpoint=0.0,
                 integrator_limit=None,
                 output_min=None,
                 output_max=None,
                 deadband=0.0):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.setpoint = setpoint
        self.integrator = 0.0
        self.previous_error = 0.0

        # Límites
        self.integrator_limit = integrator_limit  # Máximo valor del integrador
        self.output_min = output_min              # Salida mínima del PID
        self.output_max = output_max              # Salida máxima del PID
        self.deadband = deadband                  # deadband (posiblemente no se use)

    def update(self, measurement, dt):
        # Error
        error = self.setpoint - measurement

        # Aplicar deadband
        if abs(error) < self.deadband:
```

```
            error = 0.0

        # Acción integral
        self.integrator += error * dt
        if self.integrator_limit is not None:
            self.integrator = max(min(self.integrator, self.integrator_limit), -self.integrator_limit)

        #Derivar
        derivative = (error - self.previous_error) / dt if dt > 0 else 0.0
        self.previous_error = error

        #Calcular salida
        output = (self.Kp * error) + (self.Ki * self.integrator) + (self.Kd * derivative)

        #corrección si no está dentro de los umbrales
        if self.output_min is not None:
            output = max(output, self.output_min)
        if self.output_max is not None:
            output = min(output, self.output_max)

        return output

class VelocityFilter:
    def __init__(self, size=5):
        self.linear_queue = deque(maxlen=size)
        self.angular_queue = deque(maxlen=size)

    def filter(self, lineal, angular):
        self.linear_queue.append(lineal)
        self.angular_queue.append(angular)
        linear_avg = np.mean(self.linear_queue)
        angular_avg = np.mean(self.angular_queue)
        return linear_avg, angular_avg

class HeadingEstimator:
    def __init__(self, size=5):
        self.heading_queue = deque(maxlen=size)

    def update(self, new_heading):
        self.heading_queue.append(new_heading)
        sin_sum = np.sum(np.sin(self.heading_queue))
        cos_sum = np.sum(np.cos(self.heading_queue))
        return np.arctan2(sin_sum, cos_sum)

# ----------------------------
# Variables Globales
# ----------------------------

odom_data = None
real_linear_speed = 0.0
real_angular_speed = 0.0

camera_base_dir = "/mnt/c/Users/user/Documents/Captura Imagenes Camaras/ModoReal/"
camera_names = ['camara1', 'camara2', 'camara3', 'camara4']

defective_image_count = 0
total_image_count = 0

velocity_records = []  # Lista de dict: {cmd_lin, cmd_ang, real_lin, real_ang}
```

```
error_records   = []

# ----------------------------
# Guardar/Cargar estado PID y Filtros
# ----------------------------

def save_controller_state(pid_linear, pid_angular, velocity_filter, heading_estimator, filename="controller_state.csv"):

  #Guarda el estado en un CSV


  try:
    with open(filename, 'w', newline='') as f:
      writer = csv.writer(f)
      # PID lineal
      writer.writerow(["linear_integrator", pid_linear.integrator])
      writer.writerow(["linear_previous_error", pid_linear.previous_error])
      # PID angular
      writer.writerow(["angular_integrator", pid_angular.integrator])
      writer.writerow(["angular_previous_error", pid_angular.previous_error])
      # Filtro de velocidad
      writer.writerow(["velocity_filter_linear", list(velocity_filter.linear_queue)])
      writer.writerow(["velocity_filter_angular", list(velocity_filter.angular_queue)])
      # Estimación de dirección
      writer.writerow(["heading_queue", list(heading_estimator.heading_queue)])
    #print(f"Estado de los controladores guardado en {filename}.")
  except Exception as e:
    print(f"Error al guardar controladores: {e}")

def load_controller_state(pid_linear, pid_angular, velocity_filter, heading_estimator, filename="controller_state.csv"):

  #Carga el estado desde un CSV, si existe.

  if not os.path.exists(filename):
    print(f"No existe {filename}, se inician controladores en estado por defecto.")
    return

  try:
    with open(filename, 'r', newline='') as f:
      reader = csv.reader(f)
      for row in reader:
        if len(row) < 2:
          continue
        key = row[0]
        val = row[1]

        if key == "linear_integrator":
          pid_linear.integrator = float(val)
        elif key == "linear_previous_error":
          pid_linear.previous_error = float(val)
        elif key == "angular_integrator":
          pid_angular.integrator = float(val)
        elif key == "angular_previous_error":
          pid_angular.previous_error = float(val)
        elif key == "velocity_filter_linear":
          queue_list = ast.literal_eval(val)
          velocity_filter.linear_queue.clear()
          velocity_filter.linear_queue.extend(queue_list)
```

```
            elif key == "velocity_filter_angular":
                queue_list = ast.literal_eval(val)
                velocity_filter.angular_queue.clear()
                velocity_filter.angular_queue.extend(queue_list)
            elif key == "heading_queue":
                heading_list = ast.literal_eval(val)
                heading_estimator.heading_queue.clear()
                heading_estimator.heading_queue.extend(heading_list)

        #print(f"Estado de los controladores cargado desde {filename}.")

    except Exception as e:
        print(f"Error al cargar controladores: {e}")

# ----------------------------
# Funciones Auxiliares
# ----------------------------

def odom_callback(msg):
    #Actualiza las velocidades
    global odom_data, real_linear_speed, real_angular_speed
    odom_data = msg
    real_linear_speed = msg.twist.twist.linear.x
    real_angular_speed = msg.twist.twist.angular.z

def detect_aruco_centers(image, aruco_dict_type, ids_to_find):
    # Detecta el centro los marcadores ArUco de la imagen
    aruco_dict = aruco.getPredefinedDictionary(aruco_dict_type)
    parameters = aruco.DetectorParameters()
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    corners, ids, _ = aruco.detectMarkers(gray, aruco_dict, parameters=parameters)
    centers = {}
    if ids is not None:
        for i, corner in enumerate(corners):
            id_val = ids[i][0]
            if ids_to_find is None or id_val in ids_to_find:
                center_x = int(np.mean(corner[0][:, 0]))
                center_y = int(np.mean(corner[0][:, 1]))
                centers[id_val] = [center_x, center_y]
                cv2.circle(image, (center_x, center_y), 5, (0, 255, 0), -1)
                cv2.putText(image, f"ID {id_val}", (center_x + 10, center_y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
    return centers, image

def calculate_homography(pts_real, cam_centers, output_width, output_height):
    #Calcula la homografía
    pts_real_scaled = np.array([
        [output_width - pt[0], output_height - pt[1]]
        for pt in pts_real
    ], dtype=np.float32)

    ids_order = [id_ for id_ in sorted(cam_centers.keys()) if id_ != 50]
    pts_src = np.array([cam_centers[id_] for id_ in ids_order], dtype=np.float32)
    pts_dst = np.array([pts_real_scaled[id_ - 1] for id_ in ids_order], dtype=np.float32)

    if len(pts_src) != len(pts_dst):
        return None

    H, status = cv2.findHomography(pts_src, pts_dst, cv2.RANSAC, 5.0)
```

```
    return H

def project_point(H, point):
    #Proyección del punto en la imagen a plano real
    point_homog = np.array([point[0], point[1], 1])
    projected_homog = np.dot(H, point_homog)
    if projected_homog[2] == 0:
        return [0, 0]
    projected_homog /= projected_homog[2]
    return [projected_homog[0], projected_homog[1]]

def project_point_homography(H, point):
    #Proyección del punto de un plano a otro plano (homografía entre planos)
    point_homog = np.array([point[0], point[1], 1.0], dtype=np.float32)
    projected_homog = H.dot(point_homog)
    if projected_homog[2] == 0:
        return [0.0, 0.0]
    return [projected_homog[0]/projected_homog[2], projected_homog[1]/projected_homog[2]]

def calcular_direccion(pos_actual, pos_objetivo, current_heading):
    #Calcula la distancia y ángulo de error al punto objetivo
    delta_x = pos_objetivo[0] - pos_actual[0]
    delta_y = pos_objetivo[1] - pos_actual[1]
    angulo_objetivo = np.arctan2(delta_y, delta_x)
    angulo_error = angulo_objetivo - current_heading
    angulo_error = np.arctan2(np.sin(angulo_error), np.cos(angulo_error))
    distancia = np.sqrt(delta_x**2 + delta_y**2)
    return angulo_error, distancia

def adjust_target_position(pos_objetivo, table_bounds):
    #Ajusta la posición objetivo si está fuera de los márgenes
    adjusted_x = max(min(pos_objetivo[0], table_bounds['x_max']), table_bounds['x_min'])
    adjusted_y = max(min(pos_objetivo[1], table_bounds['y_max']), table_bounds['y_min'])
    return [adjusted_x, adjusted_y]

def get_middle_image(camera_name):
    #Coge la imagen del medio según tiempo de modificación del triple búffer
    global defective_image_count, total_image_count
    camera_dir = os.path.join(camera_base_dir, camera_name)
    image_files = [f"{camera_name}_image1.bmp", f"{camera_name}_image2.bmp", f"{camera_name}_image3.bmp"]
    image_paths = []
    for img_file in image_files:
        img_path = os.path.join(camera_dir, img_file)
        if os.path.isfile(img_path):
            image_paths.append(img_path)
        else:
            defective_image_count += 1

    if not image_paths:
        defective_image_count += 1
        total_image_count += 1
        return None

    image_times = []
    for img_path in image_paths:
        try:
            mtime = os.path.getmtime(img_path)
            image_times.append((mtime, img_path))
```

```
            except OSError:
                continue

        if not image_times:
            first_available_image = image_paths[0]
            image = cv2.imread(first_available_image)
            total_image_count += 1
            return image if image is not None else None

        image_times.sort()
        num_images = len(image_times)
        middle_image_path = image_times[num_images // 2][1] if num_images >= 3 else image_times[-1][1]
        image = cv2.imread(middle_image_path)
        total_image_count += 1
        return image if image is not None else None

    def project_new_id120(cam_name, homographies, output_width, output_height, H_z20_to_z0):
        #Proyecta la posición del centro del marcador del robot (120) sobre el plano mesa
        aruco_dict_type = aruco.DICT_6X6_250
        ids_to_find_z20 = [120]

        if cam_name not in homographies or homographies[cam_name].get('H_z20') is None:
            return None
        try:
            image = get_middle_image(cam_name)
            if image is None:
                return None

            centers_new, _ = detect_aruco_centers(image.copy(), aruco_dict_type, ids_to_find_z20)
            if 120 not in centers_new:
                return None

            pos_id_120_img = centers_new[120]
            pos_real_z20 = project_point(homographies[cam_name]['H_z20'], pos_id_120_img)
            pos_real_z20_metros = np.array([
                (output_width - pos_real_z20[0]),
                (output_height - pos_real_z20[1])
            ], dtype=np.float32)

            pos_real_z0_metros = project_point_homography(H_z20_to_z0, pos_real_z20_metros)
            return pos_real_z0_metros
        except Exception:
            return None

    def cargar_y_detectar_markers(cam_name, image_path, aruco_dict_type, ids_to_find):
        #Carga a imagen desde su dirección y lanza detect_aruco_centers para buscar los marcadores y su centro
        if not image_path or not os.path.exists(image_path):
            return None, None
        img = cv2.imread(image_path)
        if img is None:
            return None, None
        centers, im_marked = detect_aruco_centers(img.copy(), aruco_dict_type, ids_to_find)
        return centers, im_marked

    # ----------------------------
    # PID y Robot Configuración
    # ----------------------------

    PID_CONFIG_FILE = 'pid_params.csv'
```

```
ROBOT_CONFIG_FILE = "robot_config.csv"

def cargar_parametros_pid():
    #Carga los parámetros del los PID

    # Valores por defecto
    parametros = {
        'lineal': {
            'Kp': 0.6,
            'Ki': 0.0,
            'Kd': 0.1,
            'integrator_limit': None,
            'output_min': None,
            'output_max': None,
            'deadband': 0.0
        },
        'angular': {
            'Kp': 1.0,
            'Ki': 0.0,
            'Kd': 0.2,
            'integrator_limit': None,
            'output_min': None,
            'output_max': None,
            'deadband': 0.0
        }
    }

    if os.path.exists(PID_CONFIG_FILE):
        try:
            with open(PID_CONFIG_FILE, mode='r', newline='') as csvfile:
                reader = csv.DictReader(csvfile)
                for row in reader:
                    ctrl = row['controller']
                    if ctrl in parametros:
                        parametros[ctrl]['Kp'] = float(row['Kp'])
                        parametros[ctrl]['Ki'] = float(row['Ki'])
                        parametros[ctrl]['Kd'] = float(row['Kd'])

                        # integrator_limit
                        if 'integrator_limit' in row and row['integrator_limit']:
                            parametros[ctrl]['integrator_limit'] = float(row['integrator_limit'])

                        # output_min
                        if 'output_min' in row and row['output_min']:
                            parametros[ctrl]['output_min'] = float(row['output_min'])

                        # output_max
                        if 'output_max' in row and row['output_max']:
                            parametros[ctrl]['output_max'] = float(row['output_max'])

                        # deadband
                        if 'deadband' in row and row['deadband']:
                            parametros[ctrl]['deadband'] = float(row['deadband'])

            print("Parámetros PID cargados.")
        except Exception as e:
            print(f"Error al cargar los parámetros PID: {e}")
            print("Usando valores por defecto.")
```

```
    else:
        print("Archivo de configuración PID no encontrado. Usando valores por defecto.")

    return parametros

def guardar_parametros_pid(parametros):

    #Guarda los parámetros de los PID

    try:
        with open(PID_CONFIG_FILE, mode='w', newline='') as csvfile:
            fieldnames = [
                'controller','Kp','Ki','Kd',
                'integrator_limit','output_min','output_max','deadband'
            ]
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()

            for ctrl, gains in parametros.items():
                row = {
                    'controller': ctrl,
                    'Kp': gains['Kp'],
                    'Ki': gains['Ki'],
                    'Kd': gains['Kd'],
                    'integrator_limit': gains.get('integrator_limit', ''),
                    'output_min': gains.get('output_min', ''),
                    'output_max': gains.get('output_max', ''),
                    'deadband': gains.get('deadband', '')
                }
                writer.writerow(row)

        print("Parámetros del PID guardados.")
    except Exception as e:
        print(f"Error al guardar los parámetros del PID: {e}")

def load_robot_config():
    #Carga la configuración de las opciones del robot y funcionamiento del programa
    default_config = {
        "velocidad_lineal_max": 0.15,
        "velocidad_angular_max": 1.0,
        "velocidad_lineal_fuera_margenes": 0.05,
        "velocidad_angular_fuera_margenes": 0.3,
        "umbral_distancia_marcador_nuevo": 0.05,
        "umbral_distancia_marcador": 0.2,
        "umbral_distancia_punto": 0.05,
        "margen_mesa": 0.1,
        "margen_interno": 0.15,
        "margen_externo": 0.05,
    }
    config = {}
    if os.path.exists(ROBOT_CONFIG_FILE):
        try:
            with open(ROBOT_CONFIG_FILE, mode="r", newline="") as csvfile:
                reader = csv.DictReader(csvfile)
                for row in reader:
                    param = row["parameter"]
                    try:
                        config[param] = float(row["value"])
                    except ValueError:
```

```
                config[param] = default_config.get(param, 0)
        except Exception as e:
            print("Error al cargar la configuración:", e)
            config = {}
    for key, value in default_config.items():
        if key not in config:
            config[key] = value
    return config

def save_robot_config(config):
    #Guarda la configuración
    try:
        with open(ROBOT_CONFIG_FILE, mode="w", newline="") as csvfile:
            fieldnames = ["parameter", "value"]
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
            writer.writeheader()
            for param, value in config.items():
                writer.writerow({"parameter": param, "value": value})
        print("Configuración guardada exitosamente.")
    except Exception as e:
        print("Error al guardar configuración:", e)

def configure_options_menu():
    #Muestra y lleva el funcionamiento del menú de opciones durante la ejecución
    config_options = {}
    config_options["pid"] = cargar_parametros_pid()
    robot_config = load_robot_config()
    config_options.update(robot_config)

    while True:
        print("\n--- Menú de Configuración ---")
        print("1. Configurar parámetros PID")
        print("2. Configurar velocidad lineal máxima (actual: {:.2f} m/s)".format(config_options["velocidad_lineal_max"]))
        print("3. Configurar velocidad angular máxima (actual: {:.2f} rad/s)".format(config_options["velocidad_angular_max"]))
        print("4. Configurar velocidad lineal máxima fuera de los márgenes (actual: {:.2f} m/s)".format(config_options["velocidad_lineal_fuera_margenes"]))
        print("5. Configurar velocidad angular máxima fuera de los márgenes (actual: {:.2f} rad/s)".format(config_options["velocidad_angular_fuera_margenes"]))
        print("6. Configurar tolerancia de distancia para marcadores nuevos (actual: {:.2f} m)".format(config_options["umbral_distancia_marcador_nuevo"]))
        print("7. Configurar tolerancia de distancia para marcadores existentes (actual: {:.2f} m)".format(config_options["umbral_distancia_marcador"]))
        print("8. Configurar tolerancia de distancia para puntos objetivo (actual: {:.2f} m)".format(config_options["umbral_distancia_punto"]))
        print("9. Configurar margen interno de seguridad (actual: {:.2f} m)".format(config_options["margen_interno"]))
        print("10. Configurar margen externo de seguridad (actual: {:.2f} m)".format(config_options["margen_externo"]))
        print("11. Restablecer valores por defecto")
        print("12. Continuar con la ejecución")

        choice = input("Seleccione una opción (1-12): ").strip()

        if choice == "1":
            print("\n--- Configuración de Parámetros PID ---")
            for controller, gains in config_options["pid"].items():
                print(f"\nControlador {controller.capitalize()}:")
                print(f"  Kp={gains['Kp']}, Ki={gains['Ki']}, Kd={gains['Kd']}")
                print(f"  integrator_limit={gains['integrator_limit']}")
```

```
            print(f"   output_min={gains['output_min']}, output_max={gains['output_max']}")
            print(f"   deadband={gains['deadband']}")

        respuesta = input("\n¿Desea cambiar los valores de los parámetros PID? (s/n/default): ").strip().lower()
        if respuesta == 's':
            for controller in config_options["pid"]:
                print(f"\nConfigurando controlador {controller.capitalize()}:")
                try:
                    Kp = float(input(f"  Introduce Kp para {controller} (actual={config_options['pid'][controller]['Kp']}): "))
                    Ki = float(input(f"  Introduce Ki para {controller} (actual={config_options['pid'][controller]['Ki']}): "))
                    Kd = float(input(f"  Introduce Kd para {controller} (actual={config_options['pid'][controller]['Kd']}): "))

                    # integrator_limit
                    int_lim = input(f"   Introduce integrator_limit (actual={config_options['pid'][controller]['integrator_limit']}, vacío para no cambiar): ")

                    if int_lim.strip():

                        if int_lim.strip().lower() == 'none':

                            int_lim_val = None
                        else:

                            int_lim_val = float(int_lim)
                    else:

                        int_lim_val = config_options['pid'][controller]['integrator_limit']


                    # output_min
                    out_min = input(f"   Introduce output_min (actual={config_options['pid'][controller]['output_min']}, vacío para no cambiar): ")
                    if out_min.strip() != '':
                        out_min_val = float(out_min)
                    else:
                        out_min_val = config_options['pid'][controller]['output_min']

                    # output_max
                    out_max = input(f"   Introduce output_max (actual={config_options['pid'][controller]['output_max']}, vacío para no cambiar): ")
                    if out_max.strip() != '':
                        out_max_val = float(out_max)
                    else:
                        out_max_val = config_options['pid'][controller]['output_max']

                    # deadband
                    db = input(f"   Introduce deadband (actual={config_options['pid'][controller]['deadband']}, vacío para no cambiar): ")
                    if db.strip() != '':
                        db_val = float(db)
                    else:
                        db_val = config_options['pid'][controller]['deadband']

                    config_options["pid"][controller]['Kp'] = Kp
                    config_options["pid"][controller]['Ki'] = Ki
                    config_options["pid"][controller]['Kd'] = Kd
                    config_options["pid"][controller]['integrator_limit'] = int_lim_val
                    config_options["pid"][controller]['output_min'] = out_min_val
                    config_options["pid"][controller]['output_max'] = out_max_val
```

```
                config_options["pid"][controller]['deadband'] = db_val

            except ValueError:
                print("Entrada no válida. Se mantienen los valores actuales.")

        guardar_parametros_pid(config_options["pid"])

    elif respuesta == 'default':
        # Asignar defaults
        config_options["pid"] = {
            'lineal': {
                'Kp': 0.6, 'Ki': 0.0, 'Kd': 0.1,
                'integrator_limit': None,
                'output_min': None,
                'output_max': None,
                'deadband': 0.0
            },
            'angular': {
                'Kp': 1.0, 'Ki': 0.0, 'Kd': 0.2,
                'integrator_limit': None,
                'output_min': None,
                'output_max': None,
                'deadband': 0.0
            }
        }
        guardar_parametros_pid(config_options["pid"])
        print("Se han restablecido los parámetros PID a los valores por defecto.")

    elif respuesta == 'n':
        print("Manteniendo los parámetros PID actuales.")

elif choice == "2":
    try:
        val = float(input("Introduce la velocidad lineal máxima (m/s): "))
        config_options["velocidad_lineal_max"] = val
    except ValueError:
        print("Entrada no válida. Se mantiene el valor actual.")

elif choice == "3":
    try:
        val = float(input("Introduce la velocidad angular máxima (rad/s): "))
        config_options["velocidad_angular_max"] = val
    except ValueError:
        print("Entrada no válida. Se mantiene el valor actual.")

elif choice == "4":
    try:
        val = float(input("Introduce la velocidad lineal máxima fuera de los márgenes (m/s): "))
        config_options["velocidad_lineal_fuera_margenes"] = val
    except ValueError:
        print("Entrada no válida. Se mantiene el valor actual.")

elif choice == "5":
    try:
        val = float(input("Introduce la velocidad angular máxima fuera de los márgenes (rad/s): "))
        config_options["velocidad_angular_fuera_margenes"] = val
    except ValueError:
        print("Entrada no válida. Se mantiene el valor actual.")
```

```
elif choice == "6":
    try:
        val = float(input("Introduce la tolerancia de distancia para marcadores nuevos (m): "))
        config_options["umbral_distancia_marcador_nuevo"] = val
    except ValueError:
        print("Entrada no válida. Se mantiene el valor actual.")

elif choice == "7":
    try:
        val = float(input("Introduce la tolerancia de distancia para marcadores existentes (m): "))
        config_options["umbral_distancia_marcador"] = val
    except ValueError:
        print("Entrada no válida. Se mantiene el valor actual.")

elif choice == "8":
    try:
        val = float(input("Introduce la tolerancia de distancia para puntos  objetivo(m): "))
        config_options["umbral_distancia_punto"] = val
    except ValueError:
        print("Entrada no válida. Se mantiene el valor actual.")

elif choice == "9":
    try:
        val = float(input("Introduce margen interno (m): "))
        config_options["margen_interno"] = val
    except ValueError:
        print("Entrada no válida. Se mantiene el valor actual.")

elif choice == "10":
    try:
        val = float(input("Introduce margen_externo (m): "))
        config_options["margen_externo"] = val
    except ValueError:
        print("Entrada no válida. Se mantiene el valor actual.")

elif choice == "11":
    # Restablece valores por defecto
    config_options["velocidad_lineal_max"] = 0.15
    config_options["velocidad_angular_max"] = 1.0
    config_options["velocidad_lineal_fuera_margenes"] = 0.05
    config_options["velocidad_angular_fuera_margenes"] = 0.3
    config_options["umbral_distancia_marcador_nuevo"] = 0.05
    config_options["umbral_distancia_marcador"] = 0.2
    config_options["umbral_distancia_punto"] = 0.05
    config_options["margen_interno"] = 0.15
    config_options["margen_externo"] = 0.05
    save_robot_config({
        "velocidad_lineal_max": config_options["velocidad_lineal_max"],
        "velocidad_angular_max": config_options["velocidad_angular_max"],
        "velocidad_lineal_fuera_margenes": config_options["velocidad_lineal_fuera_margenes"],
        "velocidad_angular_fuera_margenes": config_options["velocidad_angular_fuera_margenes"],
        "umbral_distancia_marcador_nuevo": config_options["umbral_distancia_marcador_nuevo"],
        "umbral_distancia_marcador": config_options["umbral_distancia_marcador"],
        "umbral_distancia_punto": config_options["umbral_distancia_punto"],
        "margen_interno": config_options["margen_interno"],
        "margen_externo": config_options["margen_externo"]
    })
    print("Se han restablecido los valores por defecto.")
```

```
        elif choice == "12":
            break

        else:
            print("Opción no válida. Intente de nuevo.")

        # Guarda la configuración tras cada cambio
        save_robot_config({
            "velocidad_lineal_max": config_options["velocidad_lineal_max"],
            "velocidad_angular_max": config_options["velocidad_angular_max"],
            "velocidad_lineal_fuera_margenes": config_options["velocidad_lineal_fuera_margenes"],
            "velocidad_angular_fuera_margenes": config_options["velocidad_angular_fuera_margenes"],
            "umbral_distancia_marcador_nuevo": config_options["umbral_distancia_marcador_nuevo"],
            "umbral_distancia_marcador": config_options["umbral_distancia_marcador"],
            "umbral_distancia_punto": config_options["umbral_distancia_punto"],
            "margen_interno": config_options["margen_interno"],
            "margen_externo": config_options["margen_externo"]
        })

    return config_options


def monitor_images(pub_cmd_vel, image_file, threshold=2):
    #Controla si tras X segundos no se ha actualizado la imagen y detiene el robot
    while not rospy.is_shutdown():
        try:
            if os.path.exists(image_file):
                mtime = os.path.getmtime(image_file)
                if time.time() - mtime > threshold:
                    rospy.logerr("La imagen no se ha actualizado (en el tiempo limite). Deteniendo robot por seguridad.")
                    twist = Twist()
                    twist.linear.x = 0.0
                    twist.angular.z = 0.0
                    pub_cmd_vel.publish(twist)
                    rospy.signal_shutdown("Imagen no actualizada")
                    break
            else:
                #rospy.logerr(f"No se encontró el archivo de imagen: {image_file}")
                continue
        except FileNotFoundError:
            #rospy.logerr(f"Archivo {image_file} no encontrado al obtener su tiempo de modificación.")
            continue
        time.sleep(0.1)


def mostrar_diagrama_mesa(posiciones_marcadores, posiciones_nuevos_marcadores):
    """
    Dibuja en un plano 2D las esquinas (1,2,3,4), el home (ID 50) y
    los marcadores nuevos, y espera hasta a que se cierre la ventana.
    """
    fig, ax = plt.subplots()
    ax.set_title('Mesa y Marcadores (2D)')
    ax.set_xlabel('X (m)')
    ax.set_ylabel('Y (m)')
    ax.grid(True)
    ax.axis('equal')
```

```
    # Esquinas
    corner_ids = [1, 2, 3, 4]
    corner_positions = np.array([posiciones_marcadores[id_] for id_ in corner_ids])
    corner_positions = np.vstack([corner_positions, corner_positions[0]])
    ax.plot(corner_positions[:, 0], corner_positions[:, 1], 'k-', label='Mesa')
    ax.plot(corner_positions[:-1, 0], corner_positions[:-1, 1], 'ks', label='Esquinas')

    for i, (x, y) in enumerate(corner_positions[:-1]):
        ax.text(x, y, f'ID {corner_ids[i]}', fontsize=9, ha='right')

    # Home (ID 50)
    if 50 in posiciones_marcadores:
        home_pos = posiciones_marcadores[50]
        ax.plot(home_pos[0], home_pos[1], 'bo', label='Home (ID 50)')
        ax.text(home_pos[0], home_pos[1], 'Home', fontsize=9, ha='left')
    else:
        print("No se encontró la posición de ID 50 en las imágenes.")

    # Marcadores nuevos
    for id_nuevo, pos_nuevo in posiciones_nuevos_marcadores.items():
        if id_nuevo not in [1,2,3,4,50]:
            ax.plot(pos_nuevo[0], pos_nuevo[1], 'go', label=f'Nuevo ID {id_nuevo}')
            ax.text(pos_nuevo[0], pos_nuevo[1], f'ID {id_nuevo}', fontsize=9, ha='left')

    ax.legend()
    plt.show()

# ----------------------------
# Movimiento PID
# ----------------------------

def mover_robot_hacia_marcador(pos_actual, pos_objetivo, current_heading, pub_cmd_vel,
                               pid_linear, pid_angular,
                               velocity_filter,
                               heading_estimator,
                               velocidad_lineal_max,
                               velocidad_angular_max,
                               umbral_distancia,
                               dt=0.1):
    #Mueve el robot hacia la coordenada objetivo
    global velocity_records
    global error_records

    twist_msg = Twist()
    try:
        angulo_error, distancia = calcular_direccion(pos_actual, pos_objetivo, current_heading)

        if distancia < umbral_distancia:
            twist_msg.linear.x = 0
            twist_msg.angular.z = 0
            pub_cmd_vel.publish(twist_msg)
            velocity_records.append({
                'cmd_lin': twist_msg.linear.x,
                'cmd_ang': twist_msg.angular.z,
                'real_lin': real_linear_speed,
                'real_ang': real_angular_speed
            })
            error_records.append({
                'dist_error': distancia,
```

```
            'ang_error':  angulo_error
          })
          return True

        linear_speed = pid_linear.update(distancia, dt)
        angular_speed = pid_angular.update(angulo_error, dt)

        filtered_linear, filtered_angular = velocity_filter.filter(linear_speed, angular_speed)

        twist_msg.linear.x = -max(min(filtered_linear, velocidad_lineal_max), -velocidad_lineal_max)
        twist_msg.angular.z = -filtered_angular
        twist_msg.angular.z = max(min(twist_msg.angular.z, velocidad_angular_max), -velocidad_angular_max)

        pub_cmd_vel.publish(twist_msg)
        velocity_records.append({
          'cmd_lin': twist_msg.linear.x,
          'cmd_ang': twist_msg.angular.z,
          'real_lin': real_linear_speed,
          'real_ang': real_angular_speed
        })
        error_records.append({
          'dist_error': distancia,
          'ang_error':  angulo_error
        })
        return False
      except Exception as e:
        rospy.logerr(f"Error en mover_robot_hacia_marcador: {e}")
        twist_msg.linear.x = 0
        twist_msg.angular.z = 0
        pub_cmd_vel.publish(twist_msg)
        velocity_records.append({
          'cmd_lin': twist_msg.linear.x,
          'cmd_ang': twist_msg.angular.z,
          'real_lin': real_linear_speed,
          'real_ang': real_angular_speed
        })
        error_records.append({
          'dist_error': distancia,
          'ang_error':  angulo_error
        })
        return False

    def procesamiento_imagenes(camera_base_dir, homographies, output_width, output_height, trayectoria, stop_event,
H_z20_to_z0):
      #Procesa las imágenes en tiempo real
      rate = rospy.Rate(30)
      while not stop_event.is_set() and not rospy.is_shutdown():
        nuevas_posiciones_z0 = []
        for cam_name in camera_names:
          if cam_name in homographies and homographies[cam_name].get('H_z20') is not None:
            pos_real_z0 = project_new_id120(cam_name, homographies, output_width, output_height, H_z20_to_z0)
            if pos_real_z0 is not None:
              nuevas_posiciones_z0.append(pos_real_z0)
        if nuevas_posiciones_z0:
          promedio_pos = np.mean(nuevas_posiciones_z0, axis=0)
          trayectoria.append(promedio_pos)
        rate.sleep()
```

```
# ----------------------------
# Control Manual
# ----------------------------

def manual_control(pub_cmd_vel,
                   trayectoria,
                   homographies,
                   camera_base_dir,
                   output_width,
                   output_height,
                   H_z20_to_z0,
                   velocity_filter,
                   heading_estimator,
                   config_options):

    global defective_image_count
    global total_image_count

    pygame.init()
    screen = pygame.display.set_mode((300, 200))
    pygame.display.set_caption("Control Manual")

    # Límites máximos
    max_lin = config_options["velocidad_lineal_max"]
    max_ang = config_options["velocidad_angular_max"]

    # Velocidades iniciales
    manual_speed_linear = 0.12
    manual_speed_angular = 1.2

    print("\n--- Modo Manual ---")
    print("Use las teclas:")
    print("  w/s => movimiento adelante/atrás")
    print("  a/d => giro izquierda/derecha")
    print("  q   => salir del modo manual")
    print("Ajuste de velocidad:")
    print(f"  [e/r] => +/- 0.02 vel. lineal base (inicial={manual_speed_linear:.2f} m/s; max={max_lin:.2f})")
    print(f"  [t/y] => +/- 0.05 vel. angular base (inicial={manual_speed_angular:.2f} rad/s; max={max_ang:.2f})\n")

    twist_msg = Twist()
    rate = rospy.Rate(30)  # 30 Hz

    start_time = time.time()
    start_position = None
    running = True

    try:
        while running and not rospy.is_shutdown():
            #Lectura de teclas
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                    break
                elif event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        print("Saliendo del modo manual por tecla 'q'.")
                        running = False
                        break
                    # Ajustes de velocidad lineal base
```

```
        elif event.key == pygame.K_e:
            manual_speed_linear += 0.02
            manual_speed_linear = min(manual_speed_linear, max_lin)
            print(f"Vel. lineal base => {manual_speed_linear:.2f} m/s")
        elif event.key == pygame.K_r:
            manual_speed_linear -= 0.02
            manual_speed_linear = max(manual_speed_linear, 0.0)
            print(f"Vel. lineal base => {manual_speed_linear:.2f} m/s")
        # Ajustes de velocidad angular base
        elif event.key == pygame.K_t:
            manual_speed_angular += 0.05
            manual_speed_angular = min(manual_speed_angular, max_ang)
            print(f"Vel. angular base => {manual_speed_angular:.2f} rad/s")
        elif event.key == pygame.K_y:
            manual_speed_angular -= 0.05
            manual_speed_angular = max(manual_speed_angular, 0.0)
            print(f"Vel. angular base => {manual_speed_angular:.2f} rad/s")


keys = pygame.key.get_pressed()

# Movimiento lineal
if keys[pygame.K_w]:
    vel_lineal = manual_speed_linear
elif keys[pygame.K_s]:
    vel_lineal = -manual_speed_linear
else:
    vel_lineal = 0.0

# Movimiento angular
if keys[pygame.K_a]:
    vel_angular = manual_speed_angular
elif keys[pygame.K_d]:
    vel_angular = -manual_speed_angular
else:
    vel_angular = 0.0


twist_msg.linear.x = vel_lineal
twist_msg.angular.z = vel_angular
pub_cmd_vel.publish(twist_msg)

velocity_records.append({
    'cmd_lin': twist_msg.linear.x,
    'cmd_ang': twist_msg.angular.z,
    'real_lin': real_linear_speed,
    'real_ang': real_angular_speed
})


#Detectar posición para gráficas
nuevas_posiciones_z0 = []
for cam_name in camera_names:
    # Verificamos si existe homografía para z20
    if cam_name in homographies and homographies[cam_name].get("H_z20") is not None:
        pos_real_z0 = project_new_id120(
            cam_name,
            homographies,
```

```
                output_width,
                output_height,
                H_z20_to_z0
            )
            if pos_real_z0 is not None:
                nuevas_posiciones_z0.append(pos_real_z0)

        if nuevas_posiciones_z0:
            # Media de las posiciones
            promedio_pos = np.mean(nuevas_posiciones_z0, axis=0)
            trayectoria.append(promedio_pos)

            if start_position is None:
                start_position = promedio_pos.copy()
        else:
            # Si no detectamos ninguna posición
            print("No se pudo estimar la posición actual del robot (no se detectó ID120).")

        pygame.display.flip()
        rate.sleep()

finally:
    # Parar el robot
    twist_msg.linear.x = 0.0
    twist_msg.angular.z = 0.0
    pub_cmd_vel.publish(twist_msg)

    velocity_records.append({
        'cmd_lin': 0.0,
        'cmd_ang': 0.0,
        'real_lin': real_linear_speed,
        'real_ang': real_angular_speed
    })

    print("\nRobot detenido.")

    end_time = time.time()
    elapsed = end_time - start_time

    # Resumen final
    print("\n--- RESULTADOS MODO MANUAL ---")
    if len(trayectoria) > 0:
        final_position = trayectoria[-1]
        if start_position is not None:
            print(f"· Posición inicial real: ({start_position[0]:.3f}, {start_position[1]:.3f})")
        else:
            print("· Posición inicial no disponible (ID120 no se detectó al principio).")

        print(f"· Posición final real:   ({final_position[0]:.3f}, {final_position[1]:.3f})")
    else:
        print("No se detectaron posiciones (trayectoria vacía).")

    print(f"· Tiempo transcurrido:   {elapsed:.2f} s")
    print(f"· Imágenes defectuosas / total: {defective_image_count} / {total_image_count}")
    print("-----------------------------------------------------\n")

    pygame.quit()

return trayectoria
```

```
    def seleccionar_punto_destino(posiciones_marcadores, posiciones_nuevos_marcadores):
        #Permite seleccionar el punto objetivo por clic o introduciéndolo manualmente
        fig, ax = plt.subplots()
        ax.set_title('Selecciona el punto de destino haciendo clic en la mesa')
        ax.set_xlabel('X (m)')
        ax.set_ylabel('Y (m)')
        ax.grid(True)
        ax.axis('equal')

        corner_ids = [1, 2, 3, 4]
        corner_positions = np.array([posiciones_marcadores[id_] for id_ in corner_ids])
        corner_positions = np.vstack([corner_positions, corner_positions[0]])
        ax.plot(corner_positions[:, 0], corner_positions[:, 1], 'k-', label='Mesa')
        ax.plot(corner_positions[:-1, 0], corner_positions[:-1, 1], 'ks', label='Esquinas')

        for i, (x, y) in enumerate(corner_positions[:-1]):
            ax.text(x, y, f'ID {corner_ids[i]}', fontsize=9, ha='right')

        for id_, pos in posiciones_nuevos_marcadores.items():
            ax.plot(pos[0], pos[1], 'go', label=f'Nuevo marcador ID {id_}' if id_ == list(posiciones_nuevos_marcadores.keys())[0]
else "")
            ax.text(pos[0], pos[1], f'ID {id_}', fontsize=9, ha='left')

        home_pos = posiciones_marcadores.get(50, [0.971, 0.7225])
        ax.plot(home_pos[0], home_pos[1], 'bo', label='Posición Home')
        ax.text(home_pos[0], home_pos[1], 'Home', fontsize=9, ha='left')

        punto_seleccionado = {'x': None, 'y': None}

        def onclick(event):
            if event.xdata is not None and event.ydata is not None:
                punto_seleccionado['x'] = event.xdata
                punto_seleccionado['y'] = event.ydata
                ax.plot(event.xdata, event.ydata, 'ro', label='Destino Seleccionado')
                fig.canvas.draw()
                plt.close(fig)

        cid = fig.canvas.mpl_connect('button_press_event', onclick)
        plt.legend()
        plt.show()

        if punto_seleccionado['x'] is not None and punto_seleccionado['y'] is not None:
            print(f"Punto de destino seleccionado: ({punto_seleccionado['x']:.2f}, {punto_seleccionado['y']:.2f})")
            return [punto_seleccionado['x'], punto_seleccionado['y']]
        else:
            print("No se seleccionó ningún punto.")
            return None

    # ----------------------------
    # Inicializar ROS
    # ----------------------------

    def wait_for_master_loop():
        """
        Verifica en bucle si el master de ROS está disponible.
        Si no lo está, pregunta al usuario si desea seguir intentando.
```

```
    """
    while True:
      master = rosgraph.Master('/robot_aruco_navigator')
      try:
        master.getSystemState()
        print("ROS Master encontrado.")
        return
      except Exception as e:
        print("No se encontró el ROS Master. Detalle:", e)
        decision = input("¿Deseas seguir intentando conectar con el ROS master? (s/n): ").strip().lower()
        if decision == 'n':
          print("Cerrando el programa por solicitud del usuario.")
          sys.exit(1)
        else:
          print("Reintentando en 2 segundos...")
          time.sleep(2)

  def initialize_ros_and_check_connection():
    #Comprueba la conexión con ROS
    #Esperar hasta que el master esté disponible (o hasta que el usuario cancele)
    wait_for_master_loop()

    rospy.init_node('robot_aruco_navigator', anonymous=True)

    print("Verificando conexión al maestro de ROS probando el recibimiento de mensajes")
    try:

      rospy.wait_for_message('/odom', Odometry, timeout=5)
      print("Conexión del robot con ROS correcta.")
    except rospy.ROSException:
      print("No se detectó el tópico /odom en el tiempo asignado (5s). Cerrando el programa.")
      sys.exit(1)

    pub_cmd_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
    rospy.Subscriber('/odom', Odometry, odom_callback)
    return pub_cmd_vel

  def verify_image_capture_program():
    #Pregunta por pantalla si el programa de captura de imágenes está en funcionamiento
    while True:
      mode = input("Asegúrate de que el programa de captura de imágenes está funcionando. Selecciona (s/n): ").strip().lower()
      if mode not in ['s', 'n']:
        print("Modo no válido. Por favor, ingresa 's' o 'n'.")
        continue
      elif mode == 'n':
        print("Programa finalizado. Por favor, ejecuta el programa de captura de imágenes.")
        sys.exit()
      break

  # ----------------------------
  # Calibración de Cámaras con homografía
  # ----------------------------

  def process_camera_calibration(aruco_dict_type, ids_to_find_z0, ids_to_find_z20, config_options):
    image_paths_z0 = {
      'camara1': '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionPlano0/camara1/camara1-foto1.bmp',
```

```
        'camara2': '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionPlano0/camara2/camara2-foto1.bmp',
        'camara3': '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionPlano0/camara3/camara3-foto1.bmp',
        'camara4': '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionPlano0/camara4/camara4-foto1.bmp'
    }
    image_paths_new_ids = {
        'camara1': '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/Calibracion_Nuevo_Marc/camara1/camara1-foto1.bmp',
        'camara2': '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/Calibracion_Nuevo_Marc/camara2/camara2-foto1.bmp',
        'camara3': '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/Calibracion_Nuevo_Marc/camara3/camara3-foto1.bmp',
        'camara4': '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/Calibracion_Nuevo_Marc/camara4/camara4-foto1.bmp'
    }
    image_paths_z20 = {
        'camara1': [
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara1/camara1-ID1.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara1/camara1-ID2.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara1/camara1-ID3.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara1/camara1-ID4.bmp'
        ],
        'camara2': [
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara2/camara2-ID1.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara2/camara2-ID2.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara2/camara2-ID3.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara2/camara2-ID4.bmp'
        ],
        'camara3': [
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara3/camara3-ID1.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara3/camara3-ID2.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara3/camara3-ID3.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara3/camara3-ID4.bmp'
        ],
        'camara4': [
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara4/camara4-ID1.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara4/camara4-ID2.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara4/camara4-ID3.bmp',
            '/mnt/c/Users/user/Documents/Captura Imagenes Camaras/CalibracionRobot/camara4/camara4-ID4.bmp'
        ]
    }
    pts_real_z0 = np.array([
        [1.901, 0],
        [0,    0],
        [0,    1.403],
        [1.901, 1.403]
    ], dtype=np.float32)
    posiciones_marcadores = {
        1: [1.901, 0],
        2: [0, 0],
        3: [0, 1.403],
        4: [1.901, 1.403]
    }
    homographies = {}
    posiciones_id_50 = []
    posiciones_nuevos_marcadores = {}
```

```
known_ids = [1, 2, 3, 4, 50, 120]

max_x = np.max(pts_real_z0[:, 0])
max_y = np.max(pts_real_z0[:, 1])
output_width = int(max_x)
output_height = int(max_y)

pts_real_z20 = pts_real_z0.copy()
pts_real_z20[:, 1] += 0.2
H_z20_to_z0, _ = cv2.findHomography(pts_real_z20, pts_real_z0, cv2.RANSAC, 5.0)

for cam_name in camera_names:
  img_z0_path = image_paths_z0.get(cam_name)
  centers_z0, _ = cargar_y_detectar_markers(cam_name, img_z0_path, aruco.DICT_6X6_250, ids_to_find_z0)
  if centers_z0 is None or not all(id_ in centers_z0 for id_ in [1, 2, 3, 4]):
    continue

  H_z0 = calculate_homography(pts_real_z0, centers_z0, output_width, output_height)
  if H_z0 is None:
    continue

  if cam_name not in homographies:
    homographies[cam_name] = {}
  homographies[cam_name]['H_z0'] = H_z0

  # ID 50
  if 50 in centers_z0:
    pos_id_50_img = centers_z0[50]
    pos_real_id_50 = project_point(H_z0, pos_id_50_img)
    pos_real_id_50_metros = np.array([
      (output_width - pos_real_id_50[0]),
      (output_height - pos_real_id_50[1])
    ], dtype=np.float32)
    posiciones_id_50.append(pos_real_id_50_metros)

  # Nuevos marcadores
  img_new_ids_path = image_paths_new_ids.get(cam_name)
  centers_new_ids, _ = cargar_y_detectar_markers(cam_name, img_new_ids_path, aruco.DICT_6X6_250, None)
  if centers_new_ids is not None:
    nuevos_ids_en_camara = [id_ for id_ in centers_new_ids.keys() if id_ not in known_ids]
    for id_nuevo in nuevos_ids_en_camara:
      pos_nuevo_img = centers_new_ids[id_nuevo]
      pos_real_nuevo = project_point(H_z0, pos_nuevo_img)
      pos_real_nuevo_metros = np.array([
        (output_width - pos_real_nuevo[0]),
        (output_height - pos_real_nuevo[1])
      ], dtype=np.float32)
      if id_nuevo not in posiciones_nuevos_marcadores:
        posiciones_nuevos_marcadores[id_nuevo] = [pos_real_nuevo_metros]
      else:
        posiciones_nuevos_marcadores[id_nuevo].append(pos_real_nuevo_metros)

  # z20 calibración
  z20_paths = image_paths_z20.get(cam_name, [])
  correspondences_z20 = []
  for idx_z20, z20_path in enumerate(z20_paths):
    centers_z20, _ = cargar_y_detectar_markers(cam_name, z20_path, aruco.DICT_6X6_250, ids_to_find_z20)
    if centers_z20 is None or 120 not in centers_z20:
      continue
```

```
            corresponding_corner_id = idx_z20 + 1
            if corresponding_corner_id not in centers_z0:
                continue
            real_pos_z0 = pts_real_z0[corresponding_corner_id - 1]
            real_pos_z20 = np.array([real_pos_z0[0], real_pos_z0[1] + 0.2], dtype=np.float32)
            real_pos_z20_scaled = np.array([
                output_width - real_pos_z20[0],
                output_height - real_pos_z20[1]
            ], dtype=np.float32)
            correspondences_z20.append((centers_z20[120], real_pos_z20_scaled))

        if len(correspondences_z20) >= 4:
            pts_src_z20 = np.array([pt[0] for pt in correspondences_z20], dtype=np.float32)
            pts_dst_z20 = np.array([pt[1] for pt in correspondences_z20], dtype=np.float32)
            try:
                H_z20, status = cv2.findHomography(pts_src_z20, pts_dst_z20, cv2.RANSAC, 5.0)
                if H_z20 is not None:
                    homographies[cam_name]['H_z20'] = H_z20
            except cv2.error as e:
                print(f"Error al calcular H_z20 para {cam_name}: {e}")

    if posiciones_id_50:
        posicion_home = np.mean(posiciones_id_50, axis=0)
        print(f"Posición promedio de ID 50 (posición home): {posicion_home}")
    else:
        posicion_home = [1.901/2, 1.403/2]
        print(f"No se pudo estimar la posición de ID 50. Usando posición por defecto: {posicion_home}")
    posiciones_marcadores[50] = posicion_home

    # Procesar promedios de nuevos marcadores
    for id_nuevo, posiciones in posiciones_nuevos_marcadores.items():
        promedio_pos = np.mean(posiciones, axis=0)
        posiciones_nuevos_marcadores[id_nuevo] = promedio_pos
        print(f"Posición promedio del nuevo marcador ID {id_nuevo}: {promedio_pos}")

    # Unimos todo en posiciones_marcadores
    posiciones_marcadores.update(posiciones_nuevos_marcadores)

    return homographies, posiciones_marcadores, posiciones_nuevos_marcadores, output_width, output_height, H_z20_to_z0


def define_table_bounds(posiciones_marcadores, margen=0.1):
    #Define los límites de la mesa si no hubiera valores de márgenes externo ni interno se toma 0.1 m
    x_positions = [pos[0] for pos in posiciones_marcadores.values()]
    y_positions = [pos[1] for pos in posiciones_marcadores.values()]
    table_bounds = {
        'x_min': min(x_positions),
        'x_max': max(x_positions),
        'y_min': min(y_positions),
        'y_max': max(y_positions)
    }
    table_bounds['x_min'] += margen
    table_bounds['x_max'] -= margen
    table_bounds['y_min'] += margen
    table_bounds['y_max'] -= margen
    table_bounds['x_center'] = (table_bounds['x_min'] + table_bounds['x_max']) / 2
    table_bounds['y_center'] = (table_bounds['y_min'] + table_bounds['y_max']) / 2
```

```
    return table_bounds

  def invert_homography(H):
    #Hace una homografía inversa
    if H is None:
      return None
    try:
      return np.linalg.inv(H)
    except np.linalg.LinAlgError:
      return None

  def project_point_real_to_pixel(H_inv, real_point, output_width, output_height):
    #Projecta el punto real en la imagen
    scaled_x = output_width - real_point[0]
    scaled_y = output_height - real_point[1]
    point_homog = np.array([scaled_x, scaled_y, 1.0])
    pixel_homog = H_inv.dot(point_homog)
    if pixel_homog[2] != 0:
      pixel_x = int(pixel_homog[0] / pixel_homog[2])
      pixel_y = int(pixel_homog[1] / pixel_homog[2])
    else:
      pixel_x, pixel_y = 0, 0
    return [pixel_x, pixel_y]


  def draw_trajectory_on_calibration_image(
    all_trajectories,
    all_objectives,
    posiciones_marcadores,
    posiciones_nuevos_marcadores,
    homographies,
    output_width,
    output_height,
    camera_name='camara1'
  ):
    #Dibuja la trayectoria en una imagen espacial (se usa una imagen de calibración de la cámara 1)
    if camera_name not in homographies or 'H_z0' not in homographies[camera_name]:
      print(f"No se encontró homografía H_z0 para la cámara {camera_name}.")
      return
    calibration_image_path = '/mnt/c/Users/user/Documents/Captura Imagenes
Camaras/CalibracionPlano0/camara1/camara1-foto1.bmp'
    if not os.path.exists(calibration_image_path):
      print(f"No se encontró la imagen para el mapeado desde una cámara: {calibration_image_path}")
      return

    img_calib = cv2.imread(calibration_image_path)
    if img_calib is None:
      print(f"No se pudo leer la imagen de mapeado: {calibration_image_path}")
      return

    H_z0 = homographies[camera_name]['H_z0']
    H_z0_inv = invert_homography(H_z0)
    if H_z0_inv is None:
      print("No se pudo invertir la homografía de z0.")
      return

    color_corners = (0, 255, 0)
    color_home = (255, 0, 0)
    color_new_markers = (0, 0, 255)
```

```
    color_trajectory = (255, 0, 255)
    color_objective = (0, 255, 255)

    # Esquinas
    corner_ids = [1, 2, 3, 4]
    for c_id in corner_ids:
        if c_id in posiciones_marcadores:
            px, py = project_point_real_to_pixel(H_z0_inv, posiciones_marcadores[c_id], output_width, output_height)
            cv2.circle(img_calib, (px, py), 5, color_corners, -1)
            cv2.putText(img_calib, f"ID {c_id}", (px+5, py-5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color_corners, 1)

    # Home (ID 50)
    if 50 in posiciones_marcadores:
        px_home, py_home = project_point_real_to_pixel(H_z0_inv, posiciones_marcadores[50], output_width, output_height)
        cv2.circle(img_calib, (px_home, py_home), 6, color_home, -1)
        cv2.putText(img_calib, "Home (50)", (px_home+5, py_home-5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color_home, 2)

    # Nuevos marcadores
    for id_nuevo, pos_nuevo in posiciones_nuevos_marcadores.items():
        if id_nuevo not in [1,2,3,4,50]:
            px_nuevo, py_nuevo = project_point_real_to_pixel(H_z0_inv, pos_nuevo, output_width, output_height)
            cv2.circle(img_calib, (px_nuevo, py_nuevo), 5, color_new_markers, -1)
            cv2.putText(img_calib, f"ID {id_nuevo}", (px_nuevo+5, py_nuevo-5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color_new_markers, 1)

    # Trayectorias
    for idx, trayectoria in enumerate(all_trajectories):
        for i in range(len(trayectoria)-1):
            real_point1 = trayectoria[i]
            real_point2 = trayectoria[i+1]
            px1, py1 = project_point_real_to_pixel(H_z0_inv, real_point1, output_width, output_height)
            px2, py2 = project_point_real_to_pixel(H_z0_inv, real_point2, output_width, output_height)
            cv2.line(img_calib, (px1, py1), (px2, py2), color_trajectory, 2)
        if len(trayectoria) > 0:
            last_pt = trayectoria[-1]
            px_last, py_last = project_point_real_to_pixel(H_z0_inv, last_pt, output_width, output_height)
            cv2.circle(img_calib, (px_last, py_last), 4, color_trajectory, -1)

    # Objetivos
    for obj in all_objectives:
        if obj[0] == 'marcador':
            pos_objetivo_ajustado = obj[3]
            px_obj, py_obj = project_point_real_to_pixel(H_z0_inv, pos_objetivo_ajustado, output_width, output_height)
            cv2.circle(img_calib, (px_obj, py_obj), 7, color_objective, -1)
            cv2.putText(img_calib, f"Obj ID {obj[1]}", (px_obj+5, py_obj-5), cv2.FONT_HERSHEY_SIMPLEX, 0.6, color_objective, 2)
        elif obj[0] == 'punto':
            pos_objetivo_ajustado = obj[2]
            px_obj, py_obj = project_point_real_to_pixel(H_z0_inv, pos_objetivo_ajustado, output_width, output_height)
            cv2.circle(img_calib, (px_obj, py_obj), 7, color_objective, -1)
            cv2.putText(img_calib, "Objetivo", (px_obj+5, py_obj-5), cv2.FONT_HERSHEY_SIMPLEX, 0.6, color_objective, 2)

    cv2.imshow("Trayectoria sobre imagen vista desde la camara 1", img_calib)
    while True:
        key = cv2.waitKey(30) & 0xFF
        if cv2.getWindowProperty("Trayectoria sobre imagen vista desde la camara 1", cv2.WND_PROP_VISIBLE) < 1:
            break
```

```
        if key == 27:  # ESC
            break
    cv2.destroyAllWindows()

# ------------------------------------------------------------
# Mostrar trayectoria seguida sobre plano 2D
# ------------------------------------------------------------

def visualize_trajectories(all_trajectories, all_objectives,
                           posiciones_marcadores, posiciones_nuevos_marcadores):

    #Se muestran las trayectorias y los objetivos.

    plt.figure()
    colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']

    labeled_ids = set()

    #Dibuja cada trayectoria
    for idx, trayectoria in enumerate(all_trajectories):
        trayectoria_array = np.array(trayectoria)
        color = colors[idx % len(colors)]
        label = f'Trayectoria {idx+1}'

        if idx < len(all_objectives):
            objective = all_objectives[idx]
            if objective[0] == 'marcador':
                id_obj = objective[1]
                label += f' a ID {id_obj}'
            elif objective[0] == 'punto':
                original_p = objective[1]
                label += f' a punto ({original_p[0]:.2f}, {original_p[1]:.2f})'

        plt.plot(trayectoria_array[:, 0], trayectoria_array[:, 1], color + '.-', label=label)

    #Dibuja cada objetivo
    for idx, obj in enumerate(all_objectives):
        if obj[0] == 'marcador':
            id_obj = obj[1]
            pos_orig = obj[2]
            pos_adj  = obj[3]

            if id_obj not in labeled_ids:

                label_text = f'ID {id_obj}'
                labeled_ids.add(id_obj)
            else:

                label_text = ''

            if not np.allclose(pos_orig, pos_adj):

                if label_text:
                    plt.plot(pos_orig[0], pos_orig[1], 'rx')
                    plt.text(pos_orig[0], pos_orig[1], f'{label_text}(fuera)', fontsize=8, ha='left')
                    plt.plot(pos_adj[0], pos_adj[1], 'ro')
                    plt.text(pos_adj[0], pos_adj[1], label_text, fontsize=8, ha='left')
                else:
                    plt.plot(pos_orig[0], pos_orig[1], 'rx')
```

```
            plt.plot(pos_adj[0], pos_adj[1], 'ro')
        else:
            if label_text:
                plt.plot(pos_orig[0], pos_orig[1], 'ro')
                plt.text(pos_orig[0], pos_orig[1], label_text, fontsize=8, ha='left')
            else:
                plt.plot(pos_orig[0], pos_orig[1], 'ro')

    else:
        pos_orig = obj[1]
        pos_adj  = obj[2]

        if not np.allclose(pos_orig, pos_adj):
            plt.plot(pos_orig[0], pos_orig[1], 'mx')
            plt.text(pos_orig[0], pos_orig[1], '(fuera)', fontsize=8, ha='left')
            plt.plot(pos_adj[0], pos_adj[1], 'mo')
            plt.text(pos_adj[0], pos_adj[1], 'Punto', fontsize=8, ha='left')
        else:
            plt.plot(pos_orig[0], pos_orig[1], 'mo')
            plt.text(pos_orig[0], pos_orig[1], 'Punto', fontsize=8, ha='left')

corner_ids = [1, 2, 3, 4]
corner_positions = np.array([posiciones_marcadores[id_] for id_ in corner_ids])
corner_positions = np.vstack([corner_positions, corner_positions[0]])
plt.plot(corner_positions[:, 0], corner_positions[:, 1], 'k-', label='Mesa')
plt.plot(corner_positions[:-1, 0], corner_positions[:-1, 1], 'ks', label='Esquinas')
for i, (x, y) in enumerate(corner_positions[:-1]):
    c_id = corner_ids[i]
    if c_id not in labeled_ids:
        plt.text(x, y, f'ID {c_id}', fontsize=9, ha='right')
        labeled_ids.add(c_id)
    else:
        pass

#Marcadores nuevos
for id_nuevo, pos_nuevo in posiciones_nuevos_marcadores.items():
    if id_nuevo not in corner_ids and id_nuevo != 50:
        if id_nuevo not in labeled_ids:
            plt.plot(pos_nuevo[0], pos_nuevo[1], 'go')
            plt.text(pos_nuevo[0], pos_nuevo[1], f'ID {id_nuevo}', fontsize=9, ha='left')
            labeled_ids.add(id_nuevo)
        else:
            plt.plot(pos_nuevo[0], pos_nuevo[1], 'go')

#Home (ID 50)
if 50 in posiciones_marcadores:
    home_pos = posiciones_marcadores[50]
    if 50 not in labeled_ids:
        plt.plot(home_pos[0], home_pos[1], 'bo', label='Home (ID 50)')
        plt.text(home_pos[0], home_pos[1], 'Home', fontsize=9, ha='left')
        labeled_ids.add(50)
    else:
        plt.plot(home_pos[0], home_pos[1], 'bo')

plt.xlabel('X (m)')
plt.ylabel('Y (m)')
plt.title('Trayectorias del Robot')
plt.legend()
```

```
    plt.grid(True)
    plt.axis('equal')

    plt.tight_layout()
    plt.savefig("trayectorias_2d_robot.png", dpi=300)
    plt.show()

# ----------------------------
# Se muestran las velocidades
# ----------------------------

def plot_velocities(velocity_records):
    if not velocity_records:
        print("No hay datos de velocidad para mostrar.")
        return
    cmd_lin = np.array([r['cmd_lin'] for r in velocity_records])
    cmd_ang = np.array([r['cmd_ang'] for r in velocity_records])
    real_lin = np.array([r['real_lin'] for r in velocity_records])
    real_ang = np.array([r['real_ang'] for r in velocity_records])
    t = np.arange(len(velocity_records))

    valid_idx_cmd = np.where(np.abs(cmd_ang) <= 5)[0]
    valid_idx_real = np.where(np.abs(real_ang) <= 5)[0]

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 6), sharex=True)
    ax1.plot(t, cmd_lin, label='Lineal comandada', color='blue')
    ax1.plot(t, real_lin, label='Lineal estimada', color='red', linestyle='--')
    ax1.set_ylabel('Vel. lineal (m/s)')
    ax1.grid(True)
    ax1.legend()

    ax2.plot(t[valid_idx_cmd], cmd_ang[valid_idx_cmd], label='Angular comandada', color='blue')
    ax2.plot(t[valid_idx_real], real_ang[valid_idx_real], label='Angular estimada', color='red', linestyle='--')
    ax2.set_xlabel('Muestras')
    ax2.set_ylabel('Vel. angular (rad/s)')
    ax2.grid(True)
    ax2.legend()

    plt.tight_layout()
    plt.savefig("velocidades_robot.png", dpi=300)
    plt.show()

def plot_errors(error_records, nombre_png="errores_trayecto.png"):
    """
    Traza la evolución del error en distancia y del error angular
    """
    if not error_records:
        print("No hay datos de error para mostrar.")
        return

    dist_err = np.array([e['dist_error'] for e in error_records])
    ang_err  = np.array([e['ang_error']  for e in error_records])
    t        = np.arange(len(error_records))

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 6), sharex=True)
    # --- Error de distancia ---
    ax1.plot(t, dist_err, label='|error| distancia', linewidth=1.5)
    ax1.set_ylabel('Error (m)')
    ax1.grid(True)
```

```
    ax1.legend()

    # --- Error angular ---
    ax2.plot(t, ang_err, label='error ángulo (rad)', linewidth=1.5)
    ax2.set_xlabel('Muestras')
    ax2.set_ylabel('Error (rad)')
    ax2.grid(True)
    ax2.legend()


    plt.tight_layout()
    plt.savefig("errores_robot.png", dpi=300)
    plt.show()

# ----------------------------
# Control Automático
# ----------------------------

def automatic_control(pub_cmd_vel,
                      homographies,
                      posiciones_marcadores,
                      posiciones_nuevos_marcadores,
                      output_width,
                      output_height,
                      pid_linear,
                      pid_angular,
                      H_z20_to_z0,
                      config_options,
                      velocity_filter,
                      heading_estimator):
    import rospy
    import time
    import numpy as np
    import math
    from geometry_msgs.msg import Twist

    global defective_image_count
    global total_image_count

    def is_outside(bounds, pos):
        return (pos[0] < bounds['x_min'] or
                pos[0] > bounds['x_max'] or
                pos[1] < bounds['y_min'] or
                pos[1] > bounds['y_max'])

    margen_interno  = config_options["margen_interno"]
    margen_externo  = config_options["margen_externo"]

    table_bounds_outer = define_table_bounds(posiciones_marcadores, margen_externo)
    table_bounds_inner = define_table_bounds(posiciones_marcadores, margen_interno)

    robot_state = "normal"
    print(f"Estado inicial del robot: {robot_state}")

    # IDs disponibles: corners (1,2,3,4), home (50) y marcadores nuevos
    ids_disponibles = [1, 2, 3, 4, 50] + list(posiciones_nuevos_marcadores.keys())

    print("\nModo automático.")
```

```
mostrar_diagrama_mesa(posiciones_marcadores, posiciones_nuevos_marcadores)

all_trajectories = []
all_objectives   = []

# ------------------------------------------------------------------
# NUEVA VARIABLE PARA COMPARAR DISTANCIA A HOME ENTRE ITERACIONES
# ------------------------------------------------------------------
distance_to_home_prev = None

while not rospy.is_shutdown():
  # ------------------------------------------------
  # 1) Elegir objetivo: 'marcador' o 'punto'
  # ------------------------------------------------
  while True:
    try:
      target_type = input("¿Desea ir a un 'marcador' o a un 'punto'? (marcador/punto): ").strip().lower()
      if target_type not in ["marcador", "punto"]:
        print("Opción no válida.")
        continue
      break
    except ValueError:
      print("Entrada no válida.")

  # ------------------------------------------------
  # 2) Si se elige marcador
  # ------------------------------------------------
  if target_type == "marcador":
    print("\nIDs disponibles:", ids_disponibles)
    while True:
      try:
        id_objetivo = int(input(f"Introduce el ID del marcador objetivo ({ids_disponibles}): "))
        if id_objetivo not in ids_disponibles:
          print("ID no válido.")
          continue
        break
      except ValueError:
        print("Entrada no válida.")

    pos_objetivo_original = posiciones_marcadores[id_objetivo]
    pos_objetivo_ajustado = adjust_target_position(pos_objetivo_original, table_bounds_outer)
    pos_objetivo = pos_objetivo_ajustado

    # Registramos en all_objectives para graficar luego
    all_objectives.append(("marcador", id_objetivo, pos_objetivo_original, pos_objetivo_ajustado))

  # ------------------------------------------------
  # 3) Si se elige un punto (manual o clic)
  # ------------------------------------------------
  else:  # 'punto'
    while True:
      try:
        sub_mode = input("¿Desea ingresar las coordenadas manualmente o por clic? (manual/clic): ").strip().lower()
        if sub_mode not in ["manual", "clic"]:
          print("Opción no válida.")
          continue
        break
      except ValueError:
        print("Entrada no válida.")
```

```
    if sub_mode == "manual":
        while True:
            try:
                x_input = float(input("Introduce la coordenada X: "))
                y_input = float(input("Introduce la coordenada Y: "))
                pos_objetivo_original = [x_input, y_input]
                pos_objetivo_ajustado = adjust_target_position(pos_objetivo_original, table_bounds_inner)
                pos_objetivo = pos_objetivo_ajustado
                all_objectives.append(("punto", pos_objetivo_original, pos_objetivo_ajustado))
                break
            except ValueError:
                print("Entrada no válida.")
    else:
        # Por clic en la gráfica
        punto_destino = seleccionar_punto_destino(posiciones_marcadores, posiciones_nuevos_marcadores)
        if punto_destino is not None:
            pos_objetivo_original = punto_destino
            pos_objetivo_ajustado = adjust_target_position(pos_objetivo_original, table_bounds_inner)
            pos_objetivo = pos_objetivo_ajustado
            all_objectives.append(("punto", pos_objetivo_original, pos_objetivo_ajustado))
        else:
            print("No se seleccionó punto.")
            continue

# ------------------------------------------------
# 4) Preparar bucle de movimiento al objetivo
# ------------------------------------------------
trayectoria       = []
current_heading   = 0.0
objetivo_alcanzado = False
rate = rospy.Rate(30)

start_time    = time.time()
start_position = None

print("Procesamiento en tiempo real. Presiona Ctrl+C para parar.\n")

try:
    # ------------------------------------------------
    # 5) Bucle principal de movimiento
    # ------------------------------------------------
    while not objetivo_alcanzado and not rospy.is_shutdown():
        # 5.1) Obtener posición actual ID120 de varias cámaras
        nuevas_posiciones_z0 = []
        for cam_name in camera_names:
            if cam_name in homographies and homographies[cam_name].get("H_z20") is not None:
                pos_real_z0 = project_new_id120(cam_name,
                                                homographies,
                                                output_width,
                                                output_height,
                                                H_z20_to_z0)
                if pos_real_z0 is not None:
                    nuevas_posiciones_z0.append(pos_real_z0)

        if nuevas_posiciones_z0:
            promedio_pos = np.mean(nuevas_posiciones_z0, axis=0)
            trayectoria.append(promedio_pos)
```

```
    if start_position is None:
        start_position = promedio_pos.copy()
else:
    print("No se pudo estimar la posición actual del robot (no se detectó ID120).")
    rate.sleep()
    continue

# 5.2) Calcular heading estimado
if len(trayectoria) >= 2:
    dx = trayectoria[-1][0] - trayectoria[-2][0]
    dy = trayectoria[-1][1] - trayectoria[-2][1]
    if dx != 0 or dy != 0:
        vision_heading = np.arctan2(dy, dx)
    else:
        vision_heading = current_heading
else:
    vision_heading = current_heading

current_heading = heading_estimator.update(vision_heading)

# 5.3) Determina si está fuera del margen
outside_outer = is_outside(table_bounds_outer, promedio_pos)
outside_inner = is_outside(table_bounds_inner, promedio_pos)

if robot_state == "normal":
    if outside_outer:
        robot_state = "out_of_bounds"
        print("Fuera del margen externo. Busco home (vel reducida).")
else:  # out_of_bounds
    if not outside_inner:
        robot_state = "normal"
        print("Dentro del margen interno. Retomo objetivo normal.")
        # Cuando pase a normal, reseteamos la comparativa
        distance_to_home_prev = None

# 5.4) Seleccionar objetivo según estado
if robot_state == "out_of_bounds":
    next_waypoint = posiciones_marcadores[50]  # HOME
    umbral_distancia = config_options["umbral_distancia_punto"]
    velocidad_lineal_max  = config_options["velocidad_lineal_fuera_margenes"]
    velocidad_angular_max = config_options["velocidad_angular_fuera_margenes"]


    dist_home_actual = math.dist(promedio_pos, next_waypoint)

    if distance_to_home_prev is not None and dist_home_actual > distance_to_home_prev:

        angulo_error, _ = calcular_direccion(promedio_pos, next_waypoint, current_heading)


        angular_speed = pid_angular.update(angulo_error, dt=0.1)
        linear_speed  = 0.0


        filtered_lin, filtered_ang = velocity_filter.filter(linear_speed, angular_speed)

        twist_msg = Twist()
        twist_msg.linear.x  = -max(min(filtered_lin, velocidad_lineal_max),
                            -velocidad_lineal_max)
```

```
            twist_msg.angular.z = -max(
            min(filtered_ang, velocidad_angular_max),
            -velocidad_angular_max
            )
            pub_cmd_vel.publish(twist_msg)

            velocity_records.append({
              'cmd_lin':  twist_msg.linear.x,
              'cmd_ang':  twist_msg.angular.z,
              'real_lin': real_linear_speed,
              'real_ang': real_angular_speed
            })
          else:

            mover_robot_hacia_marcador(
              pos_actual         = promedio_pos,
              pos_objetivo       = next_waypoint,
              current_heading    = current_heading,
              pub_cmd_vel        = pub_cmd_vel,
              pid_linear         = pid_linear,
              pid_angular        = pid_angular,
              velocity_filter    = velocity_filter,
              heading_estimator  = heading_estimator,
              velocidad_lineal_max  = velocidad_lineal_max,
              velocidad_angular_max = velocidad_angular_max,
              umbral_distancia     = umbral_distancia,
              dt = 0.1
            )

          # Actualizamos la variable para la próxima iteración
          distance_to_home_prev = dist_home_actual

        else:
          # Estado "normal": usamos el objetivo principal
          if target_type == "marcador":
            if id_objetivo in posiciones_nuevos_marcadores:
              umbral_distancia = config_options["umbral_distancia_marcador_nuevo"]
            else:
              umbral_distancia = config_options["umbral_distancia_marcador"]
          else:
            umbral_distancia = config_options["umbral_distancia_punto"]

          velocidad_lineal_max  = config_options["velocidad_lineal_max"]
          velocidad_angular_max = config_options["velocidad_angular_max"]

          mover_robot_hacia_marcador(
            pos_actual         = promedio_pos,
            pos_objetivo       = pos_objetivo,
            current_heading    = current_heading,
            pub_cmd_vel        = pub_cmd_vel,
            pid_linear         = pid_linear,
            pid_angular        = pid_angular,
            velocity_filter    = velocity_filter,
            heading_estimator  = heading_estimator,
            velocidad_lineal_max  = velocidad_lineal_max,
            velocidad_angular_max = velocidad_angular_max,
            umbral_distancia     = umbral_distancia,
            dt = 0.1
```

```
            )

        # 5.5) Comprobar si se llega al objetivo en modo normal
        if robot_state == "normal":
            # Calcula error para ver si alcanzó
            angulo_error, distancia = calcular_direccion(promedio_pos, pos_objetivo, current_heading)
            if distancia < umbral_distancia:
                end_time      = time.time()
                final_position = promedio_pos.copy()
                elapsed       = end_time - start_time

                print("\n--- RESULTADOS DEL MODO AUTOMÁTICO AL LLEGAR AL OBJETIVO ---")
                if start_position is not None:
                    print(f"· Posición inicial real: ({start_position[0]:.3f}, {start_position[1]:.3f})")
                print(f"· Posición final real:   ({final_position[0]:.3f}, {final_position[1]:.3f})")
                print(f"· Tiempo transcurrido:   {elapsed:.2f} s")
                print(f"· Imágenes defectuosas / total: {defective_image_count} / {total_image_count}")
                print("----------------------------------\n")

                objetivo_alcanzado = True


        rate.sleep()

    # ------------------------------------------------
    # 6) Guardamos la trayectoria
    # ------------------------------------------------
    all_trajectories.append(trayectoria)

    # ------------------------------------------------
    # 7) Preguntamos si queremos otro objetivo
    # ------------------------------------------------
    while True:
        try:
            plot_errors(error_records)
            error_records.clear()
            next_action = input("¿Desea ir a otra posición? (s/n): ").strip().lower()
            if next_action not in ["s", "n"]:
                print("Entrada no válida.")
                continue
            break
        except ValueError:
            print("Entrada no válida.")

    if next_action == "s":
        continue
    else:
        print("Finalizando el programa automático.")
        rospy.signal_shutdown("Se terminó el programa.")
        break

except rospy.ROSInterruptException:
    pass
except KeyboardInterrupt:
    print("\nDetenido por usuario.")
    twist_msg = Twist()
    twist_msg.linear.x = 0
    twist_msg.angular.z = 0
    pub_cmd_vel.publish(twist_msg)
```

```
            velocity_records.append({
                'cmd_lin': 0.0,
                'cmd_ang': 0.0,
                'real_lin': real_linear_speed,
                'real_ang': real_angular_speed
            })
            print("Robot detenido.")
            return

    # ------------------------------------------------
    # 8) Al terminar se muestran las gráficas
    # ------------------------------------------------
    if len(all_trajectories) > 0:
        visualize_trajectories(all_trajectories, all_objectives,
                               posiciones_marcadores, posiciones_nuevos_marcadores)
        draw_trajectory_on_calibration_image(
            all_trajectories,
            all_objectives,
            posiciones_marcadores,
            posiciones_nuevos_marcadores,
            homographies,
            output_width,
            output_height,
            camera_name='camara1'
        )
    else:
        print("No hay datos de trayectorias para mostrar.")

    print("Programa automático finalizado.")


# ----------------------------
# main
# ----------------------------

def main():
    print("Programa de control de robot mediante las imágenes tomadas por las cámaras iniciado.")
    # 1) Configuración y ROS
    config_options = configure_options_menu()
    pub_cmd_vel = initialize_ros_and_check_connection()
    verify_image_capture_program()

    # 2) Monitor de imágenes
    camera1_image = os.path.join(camera_base_dir, "camara1", "camara1_image1.bmp")
    monitor_thread = threading.Thread(target=monitor_images, args=(pub_cmd_vel, camera1_image, 10))
    monitor_thread.daemon = True
    monitor_thread.start()

    # 3) Crear PID + Filtros
    pid_linear = PIDController(
        Kp=config_options["pid"]["lineal"]["Kp"],
        Ki=config_options["pid"]["lineal"]["Ki"],
        Kd=config_options["pid"]["lineal"]["Kd"],
        setpoint=0.0,
        integrator_limit=config_options["pid"]["lineal"]["integrator_limit"],
        output_min=config_options["pid"]["lineal"]["output_min"],
        output_max=config_options["pid"]["lineal"]["output_max"],
        deadband=config_options["pid"]["lineal"]["deadband"]
```

```
)
pid_angular = PIDController(
    Kp=config_options["pid"]["angular"]["Kp"],
    Ki=config_options["pid"]["angular"]["Ki"],
    Kd=config_options["pid"]["angular"]["Kd"],
    setpoint=0.0,
    integrator_limit=config_options["pid"]["angular"]["integrator_limit"],
    output_min=config_options["pid"]["angular"]["output_min"],
    output_max=config_options["pid"]["angular"]["output_max"],
    deadband=config_options["pid"]["angular"]["deadband"]
)

velocity_filter = VelocityFilter(size=3)
heading_estimator = HeadingEstimator(size=3)

load_controller_state(pid_linear, pid_angular, velocity_filter, heading_estimator, filename="controller_state.csv")

# 4) Calibrar cámaras
aruco_dict_type = aruco.DICT_6X6_250
ids_to_find_z0 = [1, 2, 3, 4, 50]
ids_to_find_z20 = [120]
(homographies,
 posiciones_marcadores,
 posiciones_nuevos_marcadores,
 output_width,
 output_height,
 H_z20_to_z0) = process_camera_calibration(aruco_dict_type, ids_to_find_z0, ids_to_find_z20, config_options)

# 5) Selección de modo
while True:
    mode = input("Seleccione el modo de operación: 'manual' o 'automatico': ").strip().lower()
    if mode in ['manual', 'automatico']:
        break
    else:
        print("Modo no válido.")

if mode == 'manual':
    trayectoria = []
    trayectoria = manual_control(pub_cmd_vel, trayectoria,
                                 homographies, camera_base_dir,
                                 output_width, output_height, H_z20_to_z0,
                                 velocity_filter, heading_estimator,
                                 config_options)
    if trayectoria:
        all_trajectories = [trayectoria]
        all_objectives = []
        # Graficar en 2D
        visualize_trajectories(all_trajectories, all_objectives, posiciones_marcadores, posiciones_nuevos_marcadores)
        draw_trajectory_on_calibration_image(
            all_trajectories,
            all_objectives,
            posiciones_marcadores,
            posiciones_nuevos_marcadores,
            homographies,
            output_width,
            output_height,
            camera_name='camara1'
        )
```

```
        else:
            print("No hay datos de trayectoria.")

        plot_velocities(velocity_records)
        print("Modo manual finalizado.")

    else:
        automatic_control(pub_cmd_vel,
                          homographies,
                          posiciones_marcadores,
                          posiciones_nuevos_marcadores,
                          output_width,
                          output_height,
                          pid_linear,
                          pid_angular,
                          H_z20_to_z0,
                          config_options,
                          velocity_filter,
                          heading_estimator)
        plot_velocities(velocity_records)

    save_controller_state(pid_linear, pid_angular, velocity_filter, heading_estimator, filename="controller_state.csv")

    print("Programa finalizado.")


if __name__ == "__main__":
    main()
```

Código optimizado con Inteligencia Artificial. [76]

REFERENCIAS

[1] https://openwebinars.net/blog/robotica-movil-que-es-y-sus-aplicaciones/

[2] https://www.prometec.net/sensores-distancia-infrarrojos/#:~:text=B%C3%A1sicamente%20son%20un%20emisor%20y,f%C3%A1cil%20calcular%20la%20distancia%20recorrida.

[3] https://www.wenglor.com/es/Principio-de-funcionamiento-y-tecnologia-de-un-sensor-de-ultrasonidos/s/Funktionsprinzip+und+Technologie+eines+Ultraschall-Sensors?BranchenundIndustrienlinks=%3A1&BranchenundIndustrienrechts=%3A1

[4] https://innovacionvolvo.xataka.com/deteccion-satelites-yacimientos-arqueologicos-como-funciona-tecnologia-lidar-que-revolucionaria-seguridad-coche/

[5] https://www.seguridadprofesionalhoy.com/camaras-termicas-monoculares/

[6] https://es.wikipedia.org/wiki/Estereoscop%C3%ADa

[7] https://es.wikipedia.org/wiki/Multic%C3%A1mara

[8] https://es.wikipedia.org/wiki/Homograf%C3%ADa_(geometr%C3%ADa)

[9] https://www.dedibujo.net/proyectividad-homografia/

[10] https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html

[11] https://www.uco.es/investiga/grupos/ava/portfolio/aruco/

[12] https://docs.wpilib.org/es/stable/docs/software/vision-processing/apriltag/apriltag-intro.html

[13] https://es.wikipedia.org/wiki/C%C3%B3digo_QR

[14] https://es.wikipedia.org/wiki/Marcador_de_referencia

[15] https://www.cognex.com/es-es/resources/symbologies/2-d-matrix-codes/data-matrix-codes

[16] https://dle.rae.es/rob%C3%B3tico

[17] https://www.lavanguardia.com/vida/junior-report/20220610/8330874/historia-robotica.html

[18] https://es.wikipedia.org/wiki/Tres_leyes_de_la_rob%C3%B3tica

[19] https://www.flexbot.es/aplicaciones-de-la-robotica/

[20] https://futuroelectrico.com/tipos-de-robots/

[21] https://www.telefonica.com/es/sala-comunicacion/blog/tipos-de-robots-clasificacion-aplicaciones-y-ejemplos/

[22] https://acolita.com/como-funcionan-los-dispositivos-gps-trilateracion-vs-triangulacion/

[23] https://es.wikipedia.org/wiki/GPS

[24] https://es.wikipedia.org/wiki/Sistema_de_posicionamiento_en_interiores

[25] https://idus.us.es/items/80da9759-e230-49f0-9085-73231e24da4e

[26] https://idus.us.es/items/31d67ebc-de02-4125-9cec-4b74a28f74c9

[27] https://idus.us.es/items/e44f5535-d4a8-437c-a2fc-6f0792ed261c

[28] https://pyramidimaging.com/Teledyne-Dalsa-Genie-Nano-M1280-NIR-G3-GM12-M1280__p-70519.aspx

[29] https://komoto.com/products/L_Series/LV6-T_Series

[30] https://es.wikipedia.org/wiki/C%C3%A1mara_oscura

[31] https://es.wikipedia.org/wiki/C%C3%A1mara_estenopeica#

[32] https://piziadas.com/2012/04/transformaciones-geometricas-homografias-vs-correlaciones.html

[33] https://statsbomb.com/es/articulos/futbol/nuevas-tecnologias/ia-y-los-datos-deportivos-la-estimacion-de-homografia/#:~:text=Como%20se%20explic%C3%B3%20previamente%2C%20la,imagen%20B%20aplicando%20esa%20transformaci%C3%B3n.

[34] https://biblus.us.es/bibing/proyectos/abreproy/94484/

[35] https://diotronic.com/capsulas-ultrasonicas/8861-sensor-distancia-ultrasonido

[36] https://fjalejandre.blogspot.com/2014/03/erroresmedicionesGPS.html

[37] https://zapt.tech/es/blog/mapas-asombrosos/sistema-de-posicionamiento-interior-ips-qu%C3%A9-es-y-c%C3%B3mo-funciona/

[38] https://skyhighbullseye.com/blogs/news/apriltags-accurate-tracking-solution

[39] https://mecatronicauaslp.wordpress.com/2014/03/08/patrones-de-calibracion-para-opencv/

[40] https://medium.com/@aashuthoshm/did-leonardo-da-vinci-build-a-robot-29ecd899fcf1

[41] https://findyourasl.weebly.com/blog/isaac-asimov-i-robot-book

[42] https://www.15minutenews.com/article/98541310/the-marines-are-testing-a-robot-armed-with-a-machine-gun/

[43] https://www.campetella.com/pt/industrial-robots/

[44] https://www.rtve.es/noticias/20131201/androide-para-experimentar-descubrir-robotica/806724.shtml

[45] https://www.automationworld.com/factory/robotics/article/55130925/cobots-take-on-manufacturing-challenges

[46] https://unifotografia.com/2019/08/06/que-es-la-camara-oscura-aca-te-lo-explicamos/

[47] https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html

[48] https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

[49] https://es.wikibrief.org/wiki/Homography_%28computer_vision%29

[50] https://acodigo.blogspot.com/2018/06/deteccion-de-objetos-planos.html

[51] https://es.wikipedia.org/wiki/Robot_Operating_System

[52] https://www.codio.es/2023/09/ros-nodos-publisher-subscriber-topic-introduccion.html

[53] https://www.pccomponentes.com/tp-link-archer-t3u-adaptador-usb-de-red-wifi-ac1300?kgclid=EAIaIQobChMI9bv4i82figMV77xoCR0uzxAxEAQYASABEgJfN_D_BwE&gad_source=1

[54] https://www.eleconomista.es/tecnologia/noticias/11616742/02/22/Analisis-Samsung-Galaxy-S21-FE-un-movil-incomprendido-que-no-deberias-perder-de-vista.html

[55]https://www.researchgate.net/publication/324477357_Tecnica_Flexible_de_Zhang_para_el_calculo_de_parametros_intrinsecos_y_extrinsecos_en_el_ajuste_de_dietas_en_cultivos_de_Oreochromis_Sp

[56] https://es.wikipedia.org/wiki/Raz%C3%B3n_anarm%C3%B3nica

[57] https://www.olfer.com/sdr-75-12.html

[58] https://www.netgear.com/es/business/wired/switches/smart/ms510txpp/

[59] https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[60] https://learn.microsoft.com/es-es/windows/release-health/release-information

[61] https://www.teledynevisionsolutions.com/es-es/products/sapera-lt-sdk/?model=sapera-lt-sdk&vertical=tvs-dalsa-oem&segment=tvs

[62] https://visualstudio.microsoft.com/es/vs/community/

[63] https://es.wikipedia.org/wiki/Subsistema_de_Windows_para_Linux

[64] https://releases.ubuntu.com/focal/

[65] https://code.visualstudio.com/

[66] https://wiki.ros.org/noetic

[67] https://vcxsrv.com/

[68] https://es.wikipedia.org/wiki/Python

[69] https://es.wikipedia.org/wiki/C%2B%2B

[70] https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/

[71] https://emanual.robotis.com/docs/en/platform/turtlebot3/sbc_setup/

[72] https://emanual.robotis.com/docs/en/platform/turtlebot3/opencr_setup/

[73] https://emanual.robotis.com/docs/en/platform/turtlebot3/hardware_setup/

[74] https://emanual.robotis.com/docs/en/platform/turtlebot3/bringup/#bringup

[75] https://emanual.robotis.com/docs/en/platform/turtlebot3/basic_operation/

[76] https://openai.com/

ÍNDICE DE CONCEPTOS

Homografía: Transformación proyectiva para la relación de planos. 1
ArUco: Elemento utilizado para la estimación de posición en visión artificial. 2
Parámetros intrínsecos de una cámara: Características internas de una cámara. 2
Parámetros extrínsecos de una cámara: Características de la cámara respecto al entorno. 2
Estereoscopia: Disciplina que utiliza múltiples cámaras para la estimación tridimensional. 6
ID: Número de identificación del marcador ArUco. 25
Dirección IP: Conjunto de normas que rigen los datos de una red. 26
Dirección MAC: Dirección única para identificar un dispositivo en una red local. 29
Protocolo PTP: Sincroniza relojes con alta precisión 31
Frame Rate: cantidad de imágenes capturadas por segundo 46
Trigger: Metodología de activación para la captura de imágenes. 47
Búffer: Zona de memoria temporal. 49
PID: Control Proporcional, Integral y Derivativo. 55
Anti-windup: Técnica para evitar que la acción integral se acumule en un PID. 56

GLOSARIO

AGV: Automated Guided Vehicle. 1

IA: Inteligencia Artificial. 1

LIDAR: Light Detection and Ranging. 4

GPS: Global Positioning System. 5

IPS: Indoor Positioning System. 5

Wi-Fi: Wireless Fidelity. 5

UWB: Ultra-Wide Band. 5

QR: Quick Response. 6

SDK: Software Development Kit. 21

NIR: Near-Infrared. 21

ROS: Robot Operating System. 19

WSL: Windows Subsystem Linux. 25

IP: Internet Protocol. 26

MAC: Media Access Control. 29

LUT: Look-Up Table. 47

Z-N: Ziegler-Nichols. 104