Universidad Zaragoza
1542

Trabajo Fin de Grado

Desarrollo de un sistema de control multirobot centralizado basado en visión

Autor

Sergio Pérez Lloret

Director/es

Gonzalo López Nicolás

Grado en Ingeniería de Tecnologías Industriales
Escuela de Ingeniería y Arquitectura
2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Sergio Pérez Lloret,

con nº de DNI 773133776D en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Desarrollo de un sistema de control multirobot centralizado basado en visión.

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 23 de Noviembre 2017

Fdo: Sergio Pérez Lloret

Desarrollo de un sistema de control multirobot centralizado basado en visión

Resumen

El objetivo principal de este proyecto es desarrollar e implementar un sistema de control multi-robot con visión artificial. En concreto, un ordenador central, reconoce e identifica a cada robot con la ayuda de una cámara y de un patrón impreso en papel para cada robot. Los robots se desplazan utilizando un control desde su posición inicial, hasta una serie de posiciones indicadas previamente por el usuario.

El primer paso para la realización del proyecto, ha sido un estudio de las distintas opciones que se pueden usar para la visión y los patrones impresos. A partir de las opciones elegidas, se ha seleccionado la plataforma de programación en la cual se va a realizar el control del sistema.

Elegir una plataforma de bajo coste implica que tanto el robot como los demás componentes son lo más sencillos posibles, por lo que en un principio se desconocen en detalle sus características técnicas. Por ello se ha realizado un estudio del funcionamiento de los distintos componentes del sistema, así como de sus características básicas, realizando pruebas para familiarizarse con los componentes utilizados. Esto nos ha permitido diseñar e implementar la integración de los distintos componentes, permitiendo la comunicación entre ellos y el funcionamiento deseado.

Ya instalado todo el software necesario y probadas las comunicaciones entre componentes, se ha realizado un primer planteamiento del control y movimiento de un robot. Con este modelo y control, se ha comprobado la viabilidad y respuesta real del definir un sistema de control que produzca un comportamiento adecuado.

Una vez se ha conseguido obtener un sistema de control viable y funcional, se han estudiado los parámetros relevantes del control, depurándolo y aumentando la velocidad dentro de lo posible. Finalmente, se ha aplicado el control de un robot, a un sistema de control multirobot. Una vez logrado el objetivo de un control multirobot, se plantea el diseño de algoritmos para evitar colisiones entre robots durante la ejecución del programa.

Por último, se puede concluir que, con las tareas realizadas y los resultados obtenidos en el sistema, se ha conseguido el objetivo del proyecto, obteniendo información de utilidad, y creando una base de la cual se pueden crear otras aplicaciones, o profundizar este proyecto.

Contenido

1 Introducción

1.1 Contexto y motivación

La visión por computador es uno de los campos de investigación más importantes del momento. Es una tecnología con infinidad de usos y aplicaciones, con un gran potencial. Se trata de una tecnología barata de aplicar, y que proporciona una gran cantidad de información. Su utilidad se puede apreciar claramente, ya que poco a poco, se le está dando más aplicaciones de uso en la industria.

Asimismo, la robótica móvil, es también un campo de investigación muy importante ya que pueden ser de gran utilidad, y proporcionan una gran flexibilidad en posibles aplicaciones. Se ha podido ver en estos últimos años el uso de estos robots en distintos entornos, como el militar, el doméstico con el famoso "Roomba", o en la exploración espacial.

Uno de los mayores problemas de la robótica móvil, es el de saber en todo momento la posición en el espacio que tiene el robot respecto a una referencia. Esto lleva a modelos complejos del robot, y del control de éste, o a componentes caros, para saber con cierta precisión, la posición del robot. Por eso la combinación de la visión artificial con la robótica móvil, es tan interesante. Permite localizar y controlar un robot, de forma barata, y sin tener que realizar un modelo complejo de éste, ya que se puede saber en cualquier momento, la posición del mismo respecto a una referencia, donde la precisión viene marcada por la calidad de la cámara.

Por lo tanto, el resultado de este proyecto puede ser de gran utilidad, ya que el sistema multirobot implementado basado en visión puede servir para diseñar diversas aplicaciones.

1.2 Objetivo y alcance

El objetivo principal del proyecto es el de controlar varios robots al mismo tiempo con la ayuda de la visión por computador y patrones predefinidos. La adquisición de la información visual se efectúa en un ordenador central, desde donde también se le indicará a cada uno de los robots, los comandos de movimiento necesarios para alcanzar una posición determinada con cada robot, correspondiente a la configuración deseada en el sistema multi-robot. Por último, se planteará un sistema que se pueda implementar para evitar que los robots choquen entre sí. También se impone como objetivo que la realización del proyecto se realice en una plataforma de bajo coste y todo lo que se realice, se intente simplificar lo máximo posible.

Para conseguirlo, se han de evaluar las distintas alternativas que se pueden usar, así como el modo de comunicación entre componentes. Se ha de asegurar que todo funciona correctamente, y los límites de los componentes.

El elemento más importante, ha sido la realización de un modelo básico de movimiento del robot, para la realización de un programa de control, a partir del cual se puedan extraer resultados y conclusiones, modificando el modelo y el control cuando se ha visto necesario.

En resumen, en este proyecto se estudian las distintas alternativas de software que se pueden aplicar, el hardware proporcionado, la implementación de éste, y el diseño de un sistema capaz de dirigir varios robots hacia direcciones independientes, simultáneamente.

1.3 Metodología y entorno del trabajo

Para la realización de este proyecto ha sido necesario conocer y entender las opciones que se han presentado durante la realización del proyecto, eligiendo la que se ha considerado que facilita su realización.

Para el estudio de las herramientas de visión, y de las distintas plataformas de programación, ha sido necesario instalar algunas de ellas como VisualStudio o Matlab, buscar información en internet, e instalar librerías. En el caso de la comunicación entre componentes, se han buscado alternativas, y probado que funcionasen.

En la realización del control, se ha dividido el problema en partes más pequeñas, las cuales se han ido resolviendo progresivamente. Una vez resueltas cada una de las partes, se han unido y se han resuelto los problemas de compatibilidad entre ellas.

El trabajo ha sido desarrollado en un ordenador personal, realizando pruebas en un área amplia y adecuada para ellas. Los robots, han sido proporcionado por la universidad de Zaragoza.

1.4 Organización de la memoria

El primer punto de la memoria excluyendo la introducción es el capítulo 2, en el que se exponen los principales componentes de hardware utilizados en el proyecto, que han sido proporcionados desde el primer momento. Estos componentes incluyen el ordenador, la cámara para la visión y los componentes del robot. Se explican las características básicas que se han tenido que tener en cuenta para la realización del proyecto.

El capítulo 3, se centra en el sistema de visión. En un primer lugar se plantean las distintas alternativas para éste, empezando por la librería principal de visión, pasando por la librería de los patrones, y acabando con la calibración del sistema para que trabaje con medidas reales. Una vez se han presentado las alternativas a utilizar y su razón, se explica cómo se ha planteado el funcionamiento del sistema de visión en su conjunto, además de cómo se ha utilizado para localizar el robot.

En el capítulo 4 se contemplan las diferentes plataformas de programación que se han tenido en cuenta para implementar el control y la comunicación de todo el sistema. En la primera parte de este punto se exponen las diferentes alternativas que hay para plataformas, aportando tanto puntos fuertes como débiles, y se explican las herramientas de utilidad que pueden aportar cada uno al proyecto. Por último, se razona la elección realizada. Después viene una explicación de cómo se ha conseguido comunicar los diferentes elementos, como el ordenador, el robot y los motores. Se detallan aspectos a tener en cuenta para el correcto funcionamiento, así como los pasos previos a realizar.

Una vez se han establecido las bases del proyecto, se empieza a plantear el movimiento del robot. En el capítulo 5 se estudia cómo se comporta el robot frente a

comandos de movimiento, y los defectos que pueden afectar a su control. También se presenta un estudio simple de posibles modelos de movimiento del robot que se han estudiado o deducido para el control.

El capítulo 6 contiene la explicación de los distintos controles que se han planteado para un solo robot, así como las conclusiones sacadas de cada uno de ellos, apoyados con datos experimentales. Se detalla cómo se ha afinado el control del modelo que funciona, además de justificar las decisiones que se han tomado para mejorarlo. Se explican también algunos cambios necesarios en otras partes del sistema.

Cuando ya se tiene un control que funcione para un robot, se puede pasar a controlar varios. El capítulo 7 contiene el modo en el que se procesa la información necesaria para controlar varios robots y se plantea el algoritmo anticolisiones.

Para acabar, en el capítulo 8 se exponen las conclusiones que se han sacado del trabajo.

2 Hardware Utilizado

En este apartado se exponen los diferentes elementos de Hardware utilizados en la realización del proyecto. Entre estos elementos se incluyen, el robot y los elementos que lo componen, el sistema utilizado para la visión y la cámara.

2.1 Robot

Los robots prestados por la universidad de Zaragoza son unos Pirobots. Éste es el término con el que se denomina a estos robots que utilizan una raspberrypi como CPU. Son de fabricación casera, utilizando distintos componentes comprados por separado. Son muy baratos frente a robots comerciales. Concretamente, el modelo utilizado es un robot con ruedas de direccionamiento diferencial, compuesto por una raspbery pi, dos ruedas, dos motores de corriente continua, un microcontrolador, una batería y el chasis del robot (ver figura 2.1).

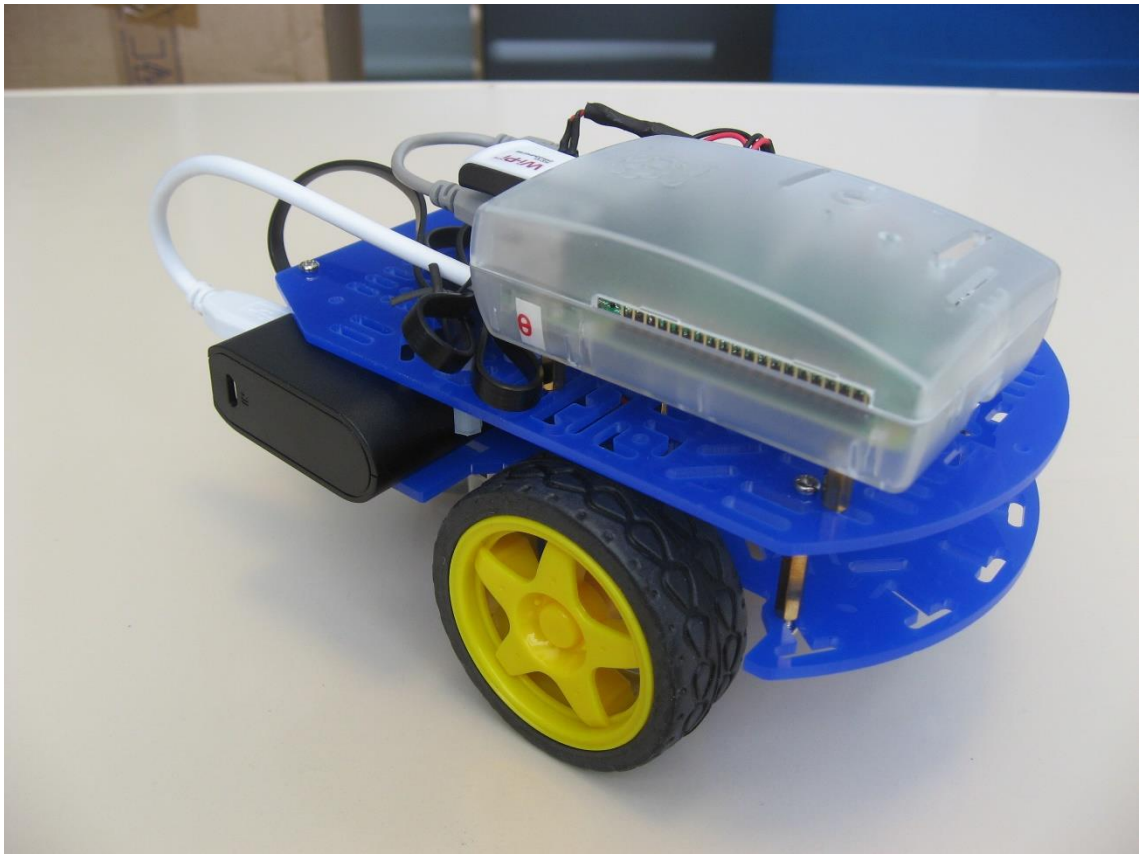

Figura 2.1 Modelo del robot utilizado en la realización del proyecto.

2.1.1 Raspberry Pi

Una Raspberry Pi es una computadora de placa reducida muy extendida en el mercado, que opera con el sistema operativo Linux, con un precio bastante reducido frente a un ordenador estándar **[Raspberry Pi]**. El modelo utilizado en el robot es la Raspberry Pi 2B, contando con 4 puertos USB, 40 pins GPIO, 1GB de RAM, un procesador de 900 MHz y una ranura para una tarjeta SD, necesaria para funcionar (ver figura 2.2).

Figura 2.2 Modelo de la Raspberry Pi 2B. Utilizada en el Robot.

2.1.2 Ruedas y motores

El robot cuenta con dos ruedas de plástico unidas a dos motores de corriente continua con reductora incluida, teniendo ambos un precio bastante reducido. Todos los componentes son de DAGU robotics, siendo la referencia del motor con reductora DG01D 48:1. Gracias a este par de motores con ruedas, que son totalmente independientes, el robot se puede mover con velocidad dependiente a la tensión con la que se alimenta. Esto implica que, si alimentan estos motores de forma independiente con una señal PWM, las ruedas girarán con velocidades distintas y controlables, lo que permite que el robot se desplace y gire (ver figura 2.3).

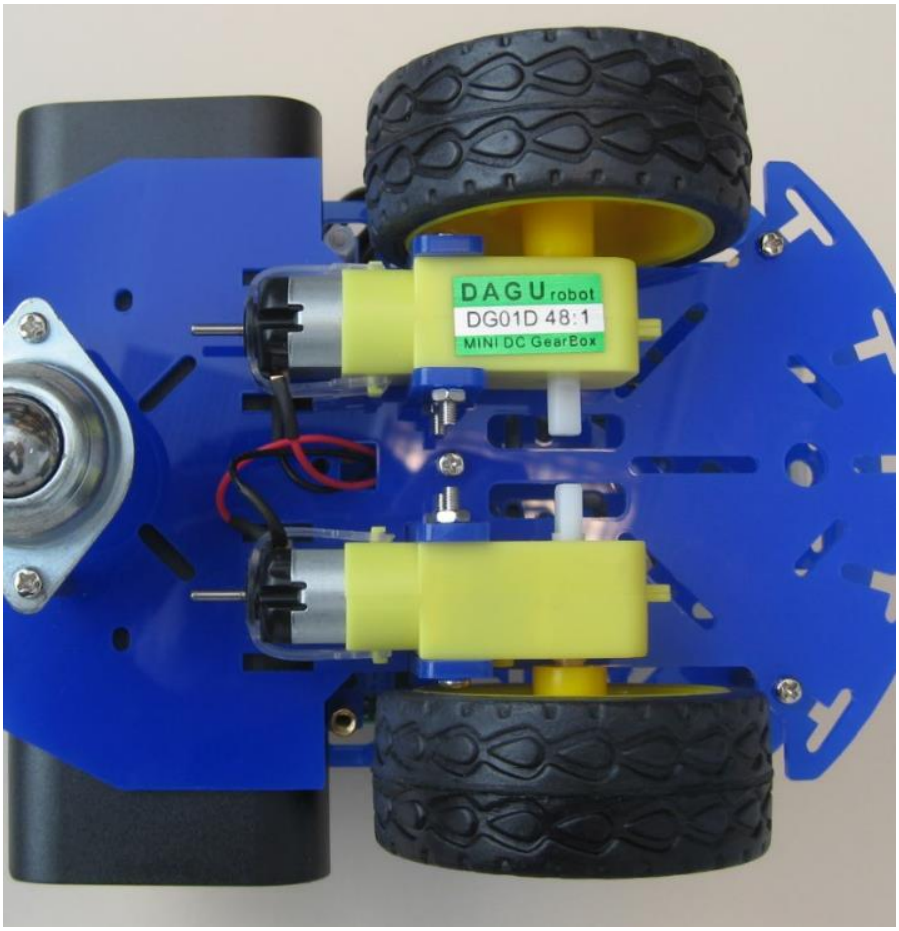

Figura 2.3 Fotografía de la parte posterior del robot, donde se aprecian las ruedas y los motores.

2.1.3 Dagu Mini Driver

Aunque la propia Raspberry Pi es capaz de generar una señal PWM, la salida de esta no es lo suficientemente potente como para alimentar a los motores. Por ello el robot cuenta con un micro controlador llamado Dagu Arduino mini Driver (ver figura 2.4). Este pequeño micro permite crear una PWM para cada motor. Su mayor ventaja, aparte de su reducido tamaño, es que se puede programar como a un Arduino, lo que facilita su implementación.

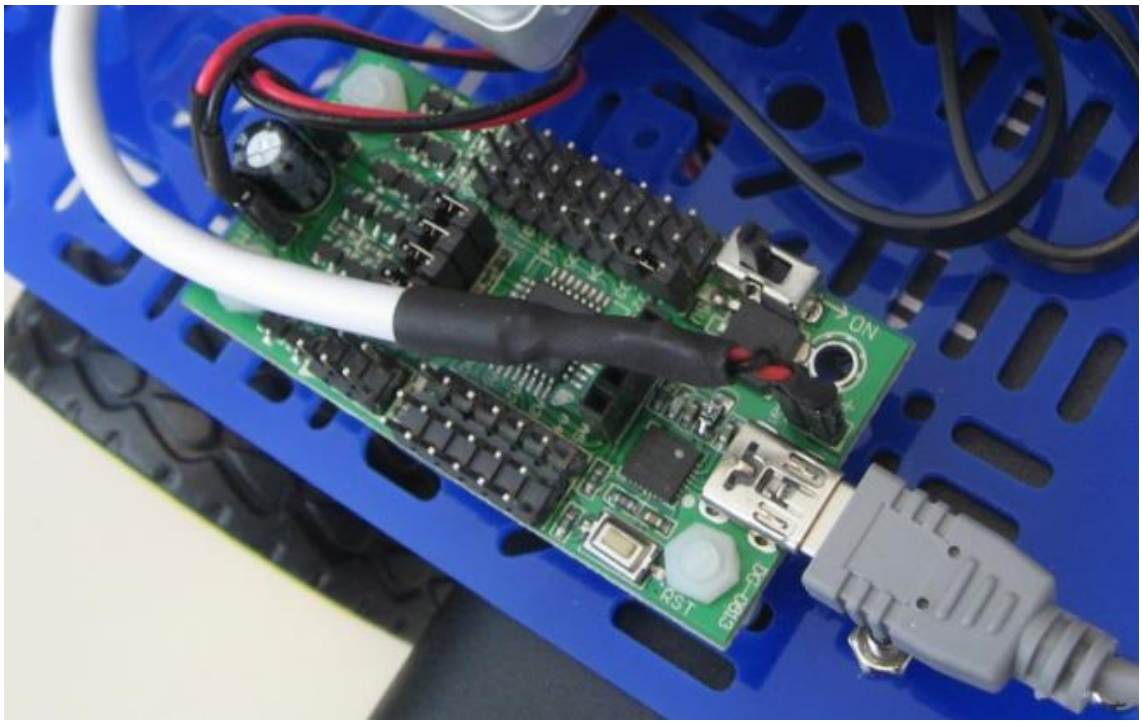

Figura 2.4 Fotografía del Mini Driver en el Robot

2.1.4 Batería

Dado que el robot necesita una fuente de electricidad para funcionar, y no sería apropiado tenerlo conectado a una toma de corriente, éste cuenta con una batería recargable propia. La batería en cuestión es de la marca ISmart, modelo RP-PB07 y cuenta con una capacidad de 10400 mAh, 5V y 2A y dos salidas USB.

2.2 Ordenador central

El sistema que soporta el sistema de visión, es un ordenador portátil de gama media-alta, que dispone de un procesador Intel i7 y de una memoria RAM de 4 GB. Aunque se podría haber utilizado la raspberry pi del propio robot para la visión, al tratarse de un sistema multi-robot, un ordenador dedicado a la visión y que mande la información correspondiente a cada robot es la opción más eficiente.

2.3 Cámara

La cámara usada en este proyecto es una webcam comercial. En concreto una Logitech c270 HDWebcam (ver figura 2.5), con cámara de 3MP y 720p de resolución en video, con conexión a través del puerto USB. Se trata de una cámara barata y fácil de conectar a un PC, convirtiéndola en la mejor opción para este proyecto.

Figura 2.5 Modelo de la cámara utilizada

3 Sistema de visión

En este apartado se detalla la selección del software necesario para el sistema de visión, comparándolo con las alternativas, así como una explicación de algunas de sus características. Por último, se explica cómo funciona el sistema de visión en general.

3.1 Librería de procesamiento de visión: OpenCV

La herramienta principal para el procesamiento de las imágenes por ordenador es la librería libre OpenCV. Esta librería desarrollada originalmente por Intel, dispone de más de 500 funciones optimizadas relacionadas con el proceso de visión. OpenCV está ampliamente extendida, por lo que cuenta con una gran comunidad de usuarios, y una gran cantidad de manuales de ayuda. Aunque desarrollado para C y C++, OpenCV se puede utilizar también en Java, Python y Matlab.

Existen varias versiones de OpenCV disponible para su descarga ya que se sigue actualizando la librería en la actualidad. Por lo general podemos quedarnos con dos versiones de las librerías dependiendo del uso, una librería básica y una librería completa. La librería básica tiene la ventaja de ser más ligera que la completa, pero cuenta con las herramientas básicas de visión por computador, frente a todas las herramientas disponibles en la completa. Si se hubiese trabajado en un dispositivo de memoria limitada, la librería ligera sería la opción más adecuada, pero dado que el proceso de visión se ha realizado en un ordenador personal, se dispone de memoria suficiente, por lo que se ha optado por la versión completa.

Para más información ver **[OpenCV]** en la bibliografía.

3.2 Reconocimiento del patrón impreso

Los patrones impresos o *targets (ver figura 3.1)*, son imágenes predefinidas que un algoritmo es capaz de reconocer en una imagen. Un ejemplo de patrón predefinido cuyo uso se ha extendido ampliamente son los códigos QR. Los *targets* además de ser reconocidos por un código, también contienen información codificada, como puede ser una página web en el caso de los códigos QR. Por lo tanto, su uso nos permite tanto rastrear como identificar cada uno de los robots a través de una imagen.

En la actualidad hay librerías gratuitas creadas por distintas personas o grupos para la impresión y el reconocimiento de estos *targets*. De las disponibles, se han elegido las que utilizan OpenCV como base, y que están diseñadas para robótica, es decir, lo más robustas y simples posibles. Finalmente, se han barajado las siguientes tres librerías diferentes:

- **[TargetDetector]**, creada por Léo Baudouin, que ha participado en colaboraciones de la Universidad de Clermont Ferrand con la Universidad de Zaragoza. Esta librería creada en entorno Linux, se basa en la detección de patrones circulares. Una vez rastreada e identificados los targets, el programa te devuelve la posición del centro del target en pixeles, el ángulo del *target* con respecto a la vertical y un número correspondiente con el patrón. Estos resultados son muy ventajosos, ya que nos devuelve exactamente la información que necesitamos para el control. Otro punto a favor es que el programa es robusto cuando la cámara no

es perpendicular al plano del *target*. Desafortunadamente, hubo problemas con la compatibilidad del programa con Windows, y se tuvo que descartar.

- **[ArUco]**, un proyecto desarrollado por el grupo AVA de la Universidad de Córdoba, basado en rastreo de patrones binarios cuadrados. La información que devuelve este programa de la imagen procesada, son las coordenadas de las cuatro esquinas en pixeles del *target*, así como el número que le corresponde *(ver figura 3.2)*. ArUco cuenta también con varías librerías de *targets* propias y es capaz de reconocer patrones de otros programas de rastreo. La mayor ventaja que presenta, es que OpenCV incluye un módulo de funciones extras de expansión para la última versión, el cual incluye la librería de ArUco, y se dispone de manuales y ejemplos en la página oficial de OpenCV. En general es una librería muy robusta, y con facilidad para instalar, que presenta unas funciones interesantes.

- **[AprilTags]**, es otra librería basada en el rastreo de patrones binarios cuadrados desarrollada por la Universidad de Michigan. La base de funcionamiento es la misma que la de ArUco, siendo una librería robusta.

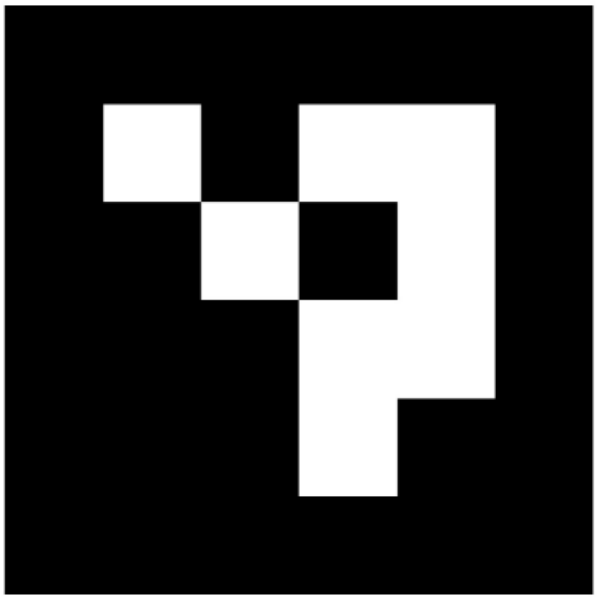

Figura 3.1 Target de la librería de ArUco correspondiente al número 0

Por lo tanto, se ha decidido utilizar ArUco, ya que al colaborar con OpenCv, la instalación de la librería es más sencilla y además se evitan posibles problemas de compatibilidad que pueden surgir con la otra alternativa viable, AprilTags, que al ser tan parecida a ArUco, no ofrece nada que supere esta ventaja. Sin embargo, se ha de advertir que la instalación de la última versión de OpenCv con los módulos extra requiere seguir unos pasos específicos, y que se necesita una gran cantidad de espacio libre en la memoria del dispositivo en la que se realiza, resultando en una librería con un tamaño aproximado de 12GB, aunque no es necesario la instalación de todos los módulos extra.

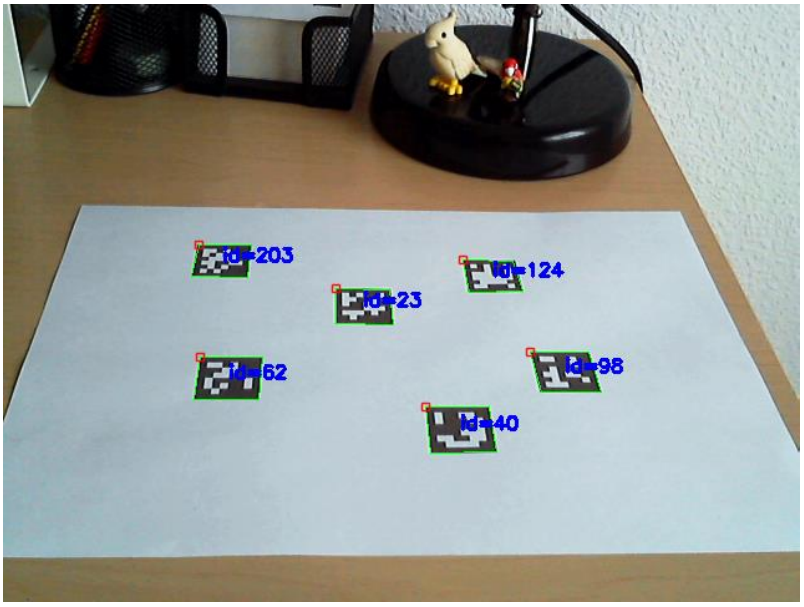

Figura 3.2 Hoja impresa con Targets detectados de ArUco

3.3 Calibración de la visión

Uno de los problemas principales de la visión artificial es obtener dimensiones reales de una imagen. No se puede sin información previa, conocer las dimensiones de un objeto con una sola imagen. Además, hay que tener en cuenta que las cámaras no son perfectas, y que las imágenes están distorsionadas por la lente de la cámara, aunque esta distorsión es una constante de la cámara. Durante la realización del proyecto, se han considerado dos maneras de resolver este problema, una función de ArUco que necesita la cámara calibrada, y la Homografía.

3.3.1 Método de calibración de la cámara

ArUco cuenta con una función que calcula el vector posición y el vector rotación respecto a la cámara, del centro de cada *target* reconocido, siendo la dirección Z de las coordenadas del target, perpendicular al plano de éste *(ver figura 3.3)*. Para obtener esta información, se le ha de proporcionar el posicionamiento del patrón en la imagen original, la medida de los lados del patrón impreso en las unidades que nos interese, la matriz de parámetros intrínsecos de la cámara, y los coeficientes de distorsión. Excepto por el posicionamiento del *target* en la imagen, los demás parámetros son constantes, siendo la longitud del lado del *target* el más fácil de encontrar.

La matriz de parámetros de la cámara y los coeficientes de distorsión son parámetros intrínsecos de la cámara. Para obtener estos parámetros, hay que calibrar la cámara, operación que se puede realizar con ayuda de otra de las funciones que proporciona ArUco. En esta operación, se realizan varias fotos desde distintos ángulos de una tabla de patrones impresos, de los cuales se conocen las dimensiones. Con estos datos, el programa es capaz de obtener los parámetros de la cámara y los coeficientes de distorsión, que se pueden guardar en un archivo, y cargar siempre que se necesiten.

El problema de este método, es que la función de ArUco solo es capaz de obtener los valores en dimensiones reales del *target*, y no de los demás puntos de la imagen. Este inconveniente complica el control, ya que no se puede referenciar el punto al cual

se quiere llevar el robot. Se podría suponer el vector rotación del punto obtenido, pero esto llevaría a un error. Por esta razón, este método ha sido descartado.

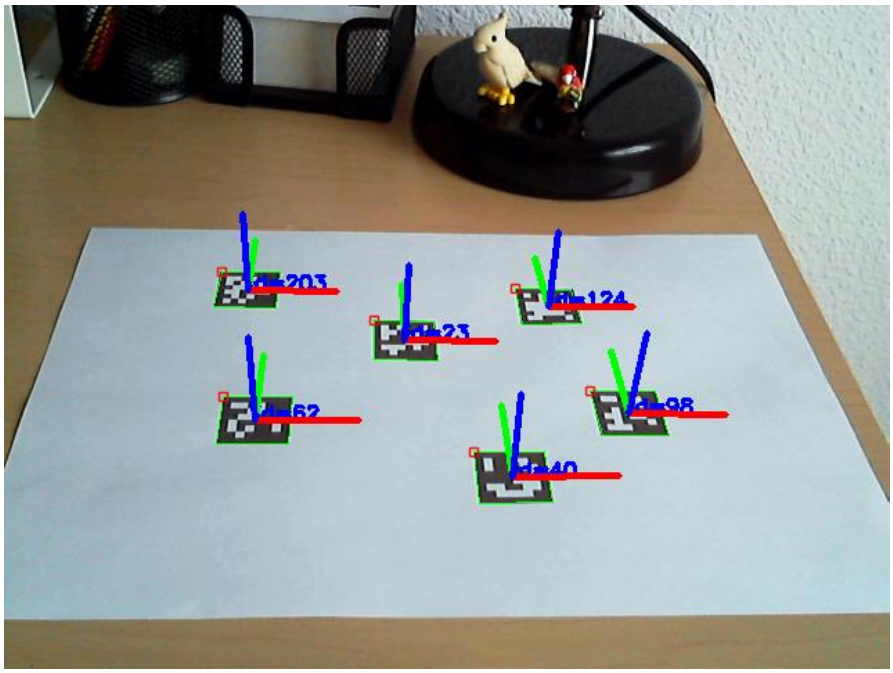

Figura 3.3 Hoja impresa con target en la cual se muestra dibujada la posición y orientación gracias al vector de posición y el de rotación.

3.3.2 Homografía

El método de la calibración por homografía consiste en realizar una transformación proyectiva a la imagen gracias a una matriz H obtenida. Esta imagen obtenida es la proyección perspectiva de un plano. Es decir, la imagen transformada es homogénea en un plano, siendo la relación entre la medida real por pixel, igual en todos los objetos que pertenecen al plano en el que se ha realizado la homografía.

Para obtener la matriz H se ha utilizado un código creado a partir del **[Hartley]**, que, para devolver la matriz, necesita que se le introduzcan cuatro puntos de un cuadrado. A partir de estos puntos, el algoritmo devuelve la matriz que permite realizar la transformación proyectiva, respecto al plano del cuadrado. Si se utilizan los 4 puntos obtenidos del rastreo del *target*, y se introducen en el algoritmo, seremos capaces de transformar la imagen para que sea homogénea en el plano del *target*, que es paralelo al suelo. Si se utiliza esta imagen transformada para el control, se puede suponer el problema en dos dimensiones no distorsionadas, simplificándolo de manera considerable. Si además se conoce el lado de los *targets*, se puede saber la relación de distancia por pixel. Por lo tanto, se ha elegido la homografía como el método de calibración de la visión.

Figura 3.4 Imagen normal, frente a imagen rectificada con homografía

3.4 La visión en el sistema

El primer paso para la puesta en marcha del sistema de visión es la colocación de la cámara. Interesa que la cámara esté lo suficientemente alta para que capte el espacio suficiente para que se muevan varios robots a la vez. Además, la lente de la cámara tiene que estar lo más perpendicular al suelo posible, para que la deformación causada por la homografía sea lo menor posible.

Al solo poder hacerse la homografía respecto a un plano, en el sistema multi-robot, se ha de decidir respecto a qué robot se realiza. Para tomar esta decisión, se ha creado un algoritmo, que estudia las homografías creadas por cada uno de los robots presentes, comprobando en cuál de ellas se observan los demás *targets* como cuadrados, a través de un cociente. Dado que repetir esta operación cada vez que se realiza una foto supondría mucho tiempo computacional, solo se realiza una vez al principio del control, y guardando la matriz H de transformación, y aplicándola a las imágenes posteriores. Mientras no se mueva la cámara, no será necesario recalcular la homografía.

El rastreo de los targets con ArUco de la imagen rectificada, devuelve la posición en pixeles de las cuatro esquinas de todos los *targets*. Pero esta información no basta, ya que lo que interesa conocer es un centro del robot y una dirección que se considera de avance. Para calcular el centro, se calcula la media de las coordenadas de las cuatro esquinas, obteniendo un punto imaginario como centro del robot. Las esquinas que devuelve ArUco, siempre están en el mismo orden, la primera que devuelve es siempre la misma, y las demás están ordenadas en el sentido de las agujas del reloj. Si se usa el punto medio entre las dos primeras esquinas, se obtiene un punto que, combinado con el centro del robot, da un vector que nos permite orientar el robot. Se ha de tener en cuenta en el control, que estos puntos son imaginarios, y pueden no coincidir con los centros y avances verdaderos del robot, ya que el target se sujeta pegado al robot ajustado a mano, por lo tanto puede no estar centrado o un poco girado.

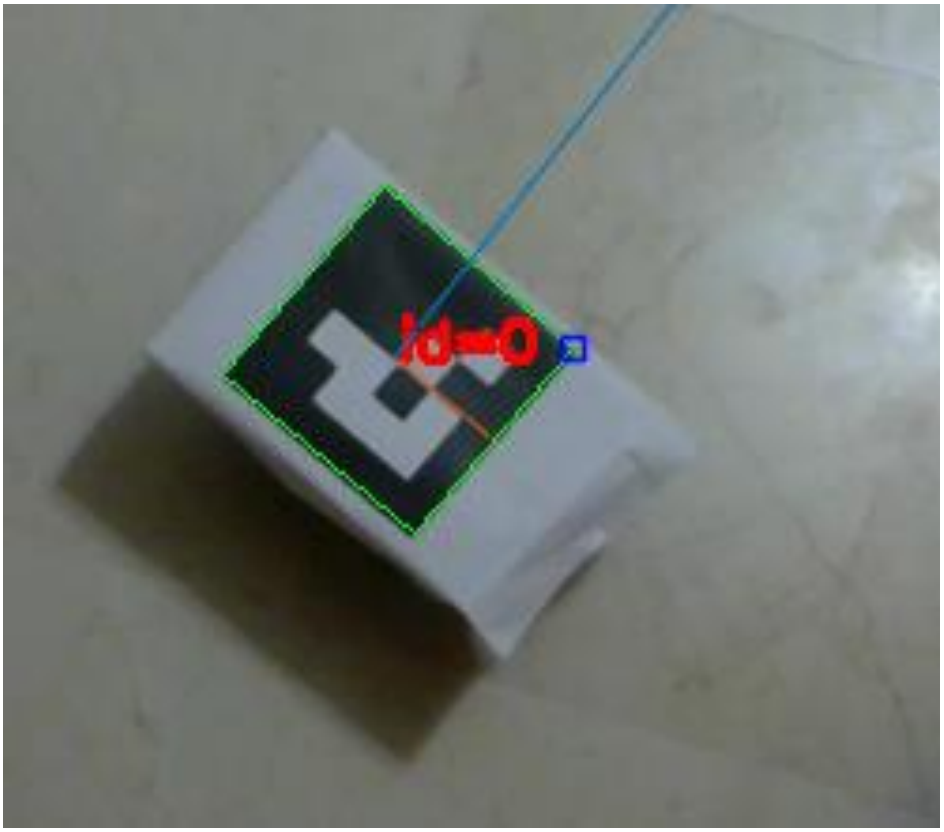

Figura 3.5 Robot detectado y orientado

4 Plataforma de programación y conexiones entre los dispositivos

La elección de la plataforma de programación, es una de las decisiones más importantes que se han tomado para este proyecto, ya que es el entorno con el que se va a realizar el control del sistema, por lo que se ha intentado seleccionar la que resulte más cómoda y ofrezca más ventajas. Además, esta plataforma debe ser compatible con OpenCV y el resto de librerías que se han seleccionado. De todas las opciones se han planteado dos, Matlab y C++.

En este punto se selecciona la plataforma de programación donde se va a realizar el control, así como explicar algunas de sus opciones que son de utilidad. Para más información sobre este punto, consulte la bibliografía de la plataforma elegida.

4.1 Matlab vs C++

C++ es un lenguaje de programación muy potente y flexible, que está muy extendido en el mundo de la programación. Su mayor ventaja es que tanto OpenCV como ArUco han sido diseñados para su uso en C# y C++, por lo que está más optimizado y es más rápido que Matlab. Su mayor desventaja es que la programación en C++ es más complicada y menos intuitiva que Matlab, con el cual hemos trabajado con anterioridad.

Matlab es una plataforma de desarrollo de software más centrado al ámbito de la investigación. Por ello, su entorno es más intuitivo, y tiene varias librerías de funciones que pueden facilitar la recogida de datos experimentales. Su inconveniente es que es más lento, y que se ha de instalar OpenCV, aunque gracias a una colaboración entre ambos, este último problema se resuelve con facilidad. **[Matlab]**

Debido a la familiaridad que se tiene con Matlab, y a que su entorno de desarrollo es más intuitivo y ofrece varias herramientas de utilidad, se ha optado por usarla como plataforma de programación frente a C++, a pesar de que pudiera ser más rápida.

4.2 Herramientas de Matlab

4.2.1 OpenCV y ArUco

Gracias a una colaboración entre Matlab y OpenCV, se puede instalar la última librería disponible de OpenCV en Matlab, de forma bastante simple, permitiéndo utilizar todas sus funciones básicas. El problema es que esta librería no incluye los módulos extra, por lo que si se instala de esta manera, no se tendrán las funciones de ArUco. Por esa razón se ha utilizado **[mexopencv]**, un programa que permite instalar la librería de OpenCV en Matlab con algunos módulos extra, entre los que incluye ArUco. De esta manera se pueden utilizar todas las funciones necesarias en la parte de la visión del control.

4.2.2 Módulo Raspberry Pi

Una de las herramientas más útiles que presenta Matlab, es el módulo de Raspberry Pi, que facilita la interacción entre Matlab y la Raspberry. Para utilizarla, se tiene que descargar el módulo desde Matlab, y dejar que se instalen unos archivos en la memoria micro SD de la Raspberry que se desee utilizar. Una vez instalados los archivos, se pueden enviar comandos a la Raspberry desde Matlab, aunque para ello, ambos dispositivos tienen que estar conectados a la misma red, y se ha de conocer la dirección

IP de la Raspberry. Este módulo permite además cargar un programa de Simulink en la propia Raspberry para que lo ejecute.

4.3 Interconexión de los dispositivos

En este apartado se explica cómo se ha conseguido establecer la comunicación entre la Raspberry Pi y el ordenador central, así como la comunicación entre la Raspberry Pi y el Mini Driver. Par más información mirar la bibliografía.

4.3.1 Conexión ordenador Robot

Para que el ordenador sea capaz de enviar consignas de movimiento al robot, se ha tenido que establecer un medio de comunicación entre el ordenador y el PC. Para ello se ha utilizado el módulo de Raspberry Pi para Matlab, que permite enviar comandos al mini ordenador desde el PC. Para que este módulo funcione, se ha de estar conectado a la misma red, así como conocer la dirección IP del robot. Por lo tanto, se han configurado todos los robots para que se conecten directamente a una red WiFi generada por el ordenador, con una capacidad de hasta 8 dispositivos. También se ha cambiado la configuración de las IP de los robots, de una IP variable a una IP fija. Esta IP, tiene una relación con el número de *target* que se le asigna (ver anexo II.5), evitando tener que guardar todas las IPs en una tabla y cargarla cada vez que sea necesario.

4.3.2 Conexión Raspberry Pi con Mini Driver

La conexión entre la Raspberry Pi y el Mini Driver, se ha realizado con la ayuda del módulo de Raspberry de Matlab, ya que cuenta con una función para conectarse por el puerto serie a otro dispositivo. También cuenta con dos señales, para leer y para escribir a través del puerto serie. En el lado del Mini Driver, se ha tenido que crear un programa para leer los comandos por el puerto serie y ejecutar la acción correspondiente en los motores. Estos comandos los lee en el formato de caracteres ASCII, es decir, el número correspondiente a ese carácter en la tabla. Por ello, y para simplificar el mensaje que se ha de enviar se ha creado una función en Matlab para enviar mensajes. (ver anexo II.6)

5 Modelo y comportamiento del robot

5.1 Modelo de movimiento

El robot con el que se trabaja dispone de dos grados de libertad de movimiento que corresponden con los dos motores independientes. Este modelo se conoce como robot de movimiento diferencial con ruedas. La base de este modelo, es que el robot es capaz de girar con diferente velocidad angular, o avanzar en línea recta, en dependencia a la velocidad que tienen las ruedas en ese momento.

Existen numerosos estudios sobre este modelo para determinar el comportamiento del robot, para conseguir que el robot siga una trayectoria deseada o para localizar el robot según las velocidades de las ruedas. Existen varios modelos, desde los más complejos, que tienen en cuenta una gran cantidad de variables, hasta los más sencillos. Pero la finalidad de todos ellos es conocer la posición del robot, en función de la velocidad de las ruedas en todo momento. Según uno de estos modelos, obtenido de un estudio realizado por **[G.W.Lucas]**, del Rossum Project, las fórmulas necesarias para un modelo sencillo serán:

$$\theta(t) = \frac{(v_r - v_l)}{L} * t + \theta_0$$

$$x(t) = x_0 + \frac{L * (v_r + v_{l)}}{2 * (v_r - v_{l)}} [sin(\frac{(v_r - v_l)}{L} * t + \theta_0) - sin(\theta_0)]$$

$$y(t) = y_0 + \frac{L * (v_r + v_{l)}}{2 * (v_r - v_{l)}} [\cos(\frac{(v_r - v_l)}{L} * t + \theta_0) - cos(\theta_0)]$$

Siendo θ el ángulo de giro del robot, v_r la velocidad de la rueda derecha, v_l la velocidad de la rueda izquierda y L la longitud entre ruedas.

En nuestro problema éstas ecuaciones no nos son de utilidad, porque no nos interesa predecir donde va a estar el robot, puesto que esta información se puede extraer con facilidad gracias a la visión. Por lo tanto, se ha decidido simplificar más aún el modelo, consiguiendo las ecuaciones de comportamiento lo más simples posibles, ya que es uno de los objetivos principales del proyecto realizar un control lo más sencillo posible. Se ha de mencionar, que estas ecuaciones serían de utilidad si en un futuro se quisiera ampliar el proyecto e implementar una predicción de la posición dentro del control e ir corrigiéndola con el control, para conseguir que el robot se mueva por un camino predeterminado, o para implementar un algoritmo que calcule el recorrido a seguir.

En el modelo que se utiliza para nuestro control, es necesario saber la v_r y la v_l, que son las velocidades de la rueda derecha e izquierda respectivamente, para enviar el comando de velocidad necesario al mini Driver. Para ello se ha desarrollado un modelo que para saber la velocidad necesaria en cada una de las ruedas si se quiere ir a una velocidad determinada y a girar con una velocidad angular determinada (ver anexo I). Las ecuaciones obtenidas son:

$$v_d = V + W\frac{L}{2}$$

$$v_l = V - W\frac{L}{2}$$

Donde V es la velocidad tangencial y W la velocidad angular deseadas en el robot.

5.2 Funcionamiento y comportamiento del Robot

El robot se mueve gracias a que se alimentan los motores con un PWM generado en un contador de 8 bits, por lo que se le tiene que fijar un número de 0 a 255. También se puede alimentar el motor con tensión negativa, consiguiendo que gire en el sentido contrario. Por lo tanto, para mover uno de los motores, se le ha de enviar al mini Driver por el puerto serie un comando de velocidad comprendido entre -255 y 255, indicando el símbolo negativo alimentación inversa.

Aunque la consigna que se le puede asignar al motor está entre 0 y 255, el robot no se comporta igual en todos los valores. Se han podido observar tres zonas distintas de comportamiento que se muestran en la tabla a continuación.

Consigna	Zona
0 … 50	Zona 1
51 … 80	Zona 2
81 … 255	Zona 3

Figura 5.1 Zonas de Trabajo del robot

Cuando el comando está en la zona 1, los motores no tienen la potencia suficiente para mover el Robot y este permanece parado.

La zona 2 es una zona media, ya que cuando la consigna está entre estos valores, el robot tiene un comportamiento irregular. El robot puede avanzar o permanecer quieto, acelerar y frenarse e incluso llegar a pararse en marcha, todo manteniendo la consigna constante.

La zona 3 contiene los valores con los que los motores consiguen la potencia mínima para moverse en todo momento, resultando en un comportamiento aproximadamente lineal y son los valores que se utilizan para el control.

La velocidad máxima aproximada del Robot, es de 160 cm/s, que se consigue cuando la consigna de los dos motores es de 255. Este valor se ha calculado aproximadamente, ya que no va a ser igual entre todos los Robots. Esto se debe a que aunque los motores sean del mismo modelo y del mismo fabricante, no tienen el mismo comportamiento frente a la tensión. Este comportamiento se observa claramente al enviarle la consigna máxima a ambos motores. Según el modelo, el robot tendría que ir en línea recta, pero se observa que se desvía, ya que los motores no son iguales, y no responden igual a la tensión, por lo que ninguno de los robots se comporta igual.

6 Creación y ajuste del control para un solo robot

En este apartado se explica en primer lugar cómo funcionan los controles que se han creado, y que es lo que se ha tenido que modificar para conseguir un control funcional. Después se explica cómo se ha ajustado y cuáles son los parámetros más importantes a tener en cuenta de ese control. Todo esto se realiza únicamente para un solo robot.

6.1 Control proporcional

Puesto que el objetivo es hacer un control lo más simple y robusto posible para manejar varios robots al mismo tiempo, se ha optado por un control proporcional. El primer paso en el programa de control es la inicialización de todo, empezando por la utilización de las funciones de OpenCV para obtener imágenes a través de la cámara, seguido de la obtención de la matriz de transformación para la homografía. Una vez obtenida esta matriz, se trabaja solo con imágenes transformadas respecto a esa matriz, y se calcula el tamaño real del pixel. Por último, se conecta el ordenador con el robot a través de la red WiFi y se seleccionan por pantalla los puntos que se desea que alcance el robot. (Para más detalle ver anexo II.1).

Una vez se ha inicializado el sistema, se entraría dentro del bucle de control, donde se procedería a detectar la posición del robot, actualizarla y actuar sobre él. Gracias al sistema de visión somos capaces de conocer la posición del centro imaginario del robot, así como de un vector que indica la dirección de avance. Con el centro del robot, el punto al que se tiene que llegar y el tamaño de pixel, se puede saber fácilmente la distancia real entre estos dos puntos. Si además se crea un vector entre estos dos puntos, se puede sacar el ángulo de desfase entre el vector de avance del robot y el punto al que se quiere ir (ver anexo III.3). Conociendo entonces el ángulo que se tiene que girar y la distancia a la que se tiene que ir se pude aplicar un control de la forma:

$$V = Distancia * Kv$$
$$W = Ángulo * Kw$$

Si esta velocidad angular y tangencial se introduce en las ecuaciones del movimiento obtenidas en el apartado 6.1, se obtiene la velocidad necesaria en cada rueda para obtener esas velocidades. Si a esta velocidad en cada rueda se le aproxima con una consigna de esta manera:

$$V_{consigna} = \frac{V_{rueda}}{V_{max}} * 255$$

Hay que tener en cuenta que, si la velocidad de la rueda es mayor que 255, se tiene que limitar de alguna manera. En este caso se fijará el valor en 255, y se le dará prioridad al giro que al avance.

El principal problema de este modelo, es que, al ser los motores distintos, no se puede aproximar la velocidad de la rueda tan fácilmente, ya que se comportan de forma distinta frente a la tensión. Esto implicaría que se tendría que realizar un modelo de cada uno de los motores de los robots a utilizar y almacenarlo, y ya que se quiere obtener un control lo más simple posible, no nos interesa hacerlo. Por lo tanto, este modelo se ha descartado.

6.2 Control secuencial conmutado

Este segundo modelo se basa en el primero, la única diferencia es que se divide el control en dos fases. Una fase donde el robot gira sobre sí mismo y se orienta, y una segunda en la que el robot avanza. Para la comprobación de este control, primero se estudia el giro y luego el avance (ver anexo II.3, aunque no sea el mismo, el planteamiento es el mismo).

Para que el robot gire sobre sí mismo aproximadamente, hay que hacer que la consigna de una rueda y la otra sean $V_{derecha} = -V_{izquierda}$. La velocidad de estas ruedas es proporcional al ángulo de giro del robot respecto al objetivo, igual que en el primer control planteado. Pero hay que tener en cuenta que, si el ángulo que tiene que girar es muy pequeño, la consigna de velocidad en las ruedas será demasiado pequeña, y no se moverá ya que no se dispondría de par motor suficiente, y si se subiese la Kw, la acción sería demasiado grande. Por lo tanto, hay que ponerle un límite mínimo a la consigna de velocidad de las ruedas para que se mueva.

El principal problema de este control se debe a que el control es muy lento. El control para un robot, cuesta computacionalmente unos 0,14 segundos (ver anexo III). Aunque se depurase el programa a lo mínimo posible, solo la visión y el detectar el robot necesitan 0,05 segundos (ver anexo III), sin contar que se tendría que eliminar la información visual por pantalla, que no se quiere hacer ya que dificultaría la verificación de que funciona. Esto sería con solo un robot, con varios robots, dado que los cálculos se hacen en el ordenador, costará más tiempo por ciclo. Este tiempo y la velocidad mínima con la que gira es la precisión que se tendrá. Pero al ser el tiempo demasiado grande, la precisión baja. Ya que desde que detecta su posición hasta que actúa han pasado 0,05 segundos, tiempo durante el cual está avanzando con una consigna más alta. La consigna de velocidad para la que funcionaría con una precisión más o menos aceptable, estaría dentro de la zona 2 de la figura 5.1. Pero el comportamiento en esa zona es errático.

Por lo tanto, este control no puede funcionar adecuadamente de esta manera, y la mejor solución es cambiar el motor con reductora, por uno con un cociente de reducción más grande, para bajar la velocidad de giro de la rueda, aumentando el par motor que tiene. Esto permitiría que el robot fuese a velocidades más lentas y controlables. Otra opción sería cambiar a C++, que podría ser más rápido, o realizar los cálculos y el control dentro de la Raspberry Pi, pero no afectaría los resultados lo suficiente como para hacerlo plausible, además de que, se ha intentado hacer el control lo más simple posible.

6.3 Control proporcional de pulsos

Debido a que el tiempo de ciclo es demasiado largo, se ha decidido cambiar el modo de control. En vez de mantener una consigna hasta la siguiente orden, se mantiene la consigna durante un periodo de tiempo. Para ello se ha modificado en un detalle el programa del Mini Driver y la función de enviar mensajes, para que se reciba también el tiempo en milisegundos en los que se tiene que activar, que llamaremos pulso. Aunque esta forma es más lenta, se consigue una mayor robustez que con los otros métodos, y una precisión mayor. Hay que asegurarse de que el pulso tiene un

límite superior, para que no llegue la siguiente orden de movimiento sin que esta haya acabado, siendo este 0,03 segundo.

La base de este control es la misma que en el segundo método, está dividido en dos partes, primero se orienta, y luego avanza. La diferencia es que ahora el control proporcional no modula la consigna de velocidad, si no que modula el tiempo que se mantiene ésta. Cada una de estas dos fases se pueden dividir en tres subfases, si la distancia o el ángulo es mayor que un valor determinado, si está comprendido entre ese valor y uno aceptado, o si es menor que el valor aceptado. (ver figura 6.1)

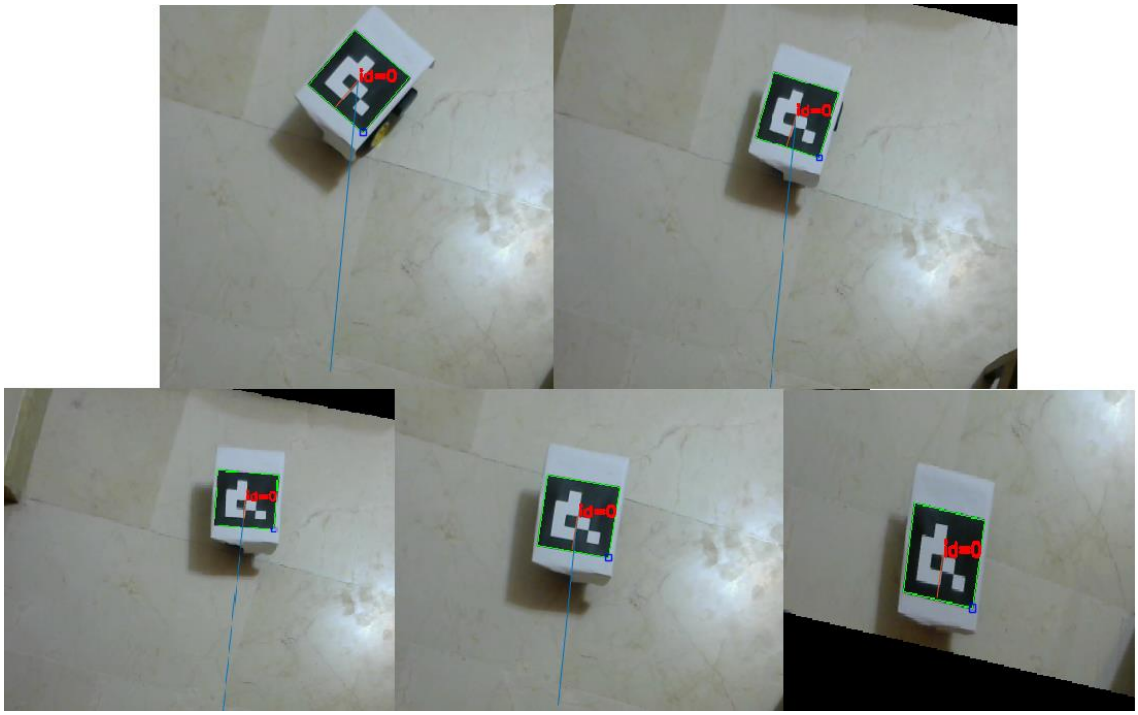

Figura 6.1 Giro y avance del robot hasta llegar a su punto.

6.3.1 Ajuste de orientación

El primer paso de la puesta en marcha consiste en estudiar que el robot se oriente con respecto al punto deseado. Se ha observado que un control proporcional en todo el sistema no es lo más rápido, y por lo tanto se ha dividido el control en los tres casos ya mencionados. El primer caso se produce cuando el ángulo es superior a un valor dado, dando como consigna el pulso máximo de 30 ms, y un valor de velocidad de ruedas de 200 de modulo sobre 255, con el signo dependiendo del lado de giro, ya que en este caso no es trascendental la precisión sino la velocidad. El segundo caso se produce cuando es menor que ese valor dado, pero el ángulo del robot no está dentro de las tolerancias en que se considera orientado. En este caso se aplica una consigna de velocidad de las ruedas de 160, y una consigna de pulso que sigue la ecuación $pulso = |ángulo| * Kw$, así se consigue una mayor precisión. Por último, si el ángulo es menor que un valor que se considera aceptable en tolerancias, se puede decir que el robot está orientado. Hay que tener en cuenta que el pulso no puede bajar tampoco de un valor mínimo, ya que no avanzaría el robot, siendo este valor 15 ms, de pruebas obtenidas con varios robots. (ver figura 6.2)

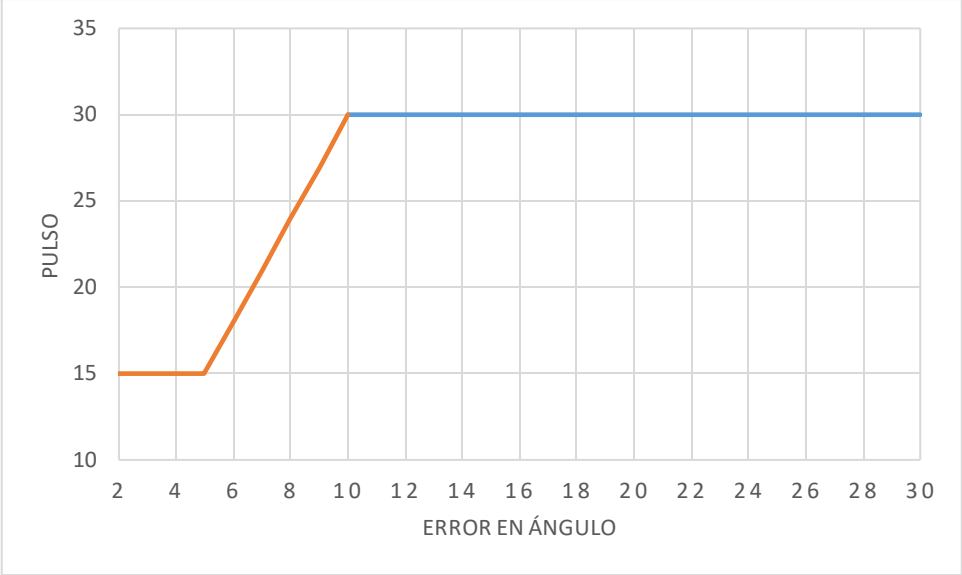

Figura 6.2 Comportamiento del control. Tiempo de espera frente al error en ángulo del robot. Pulso (ms) vs Ángulo (º). El azulo representa una consigna de velocidad de 200 frente a 255, el naranja de 160 frente a 255.

De este sistema se pueden sacar varios puntos importantes para la rapidez del control, la consigna máxima de velocidad, la consigna mínima de velocidad, el ángulo de acercamiento, el ángulo objetivo y la Kw, estando todos ellos relacionados. La consigna de velocidad máxima se ha seleccionado experimentalmente, eligiendo una lo bastante grande como para que el robot gire rápido, pero no lo suficientemente rápido como para que el sistema se pueda descontrolar. El ángulo de acercamiento escogido es 10, elegido experimentalmente. Se ha de coger el valor más bajo posible, para que se aproxime lo máximo, a la mayor velocidad que se pueda, sin perder luego precisión. La Kw se ha calculado, de manera que cuando el ángulo tenga el mismo valor que el de aproximación, el pulso sea igual que 30ms, dando una Kw de 3. El valor de la consigna mínima de velocidad es el que marca la precisión de la orientación, si se quiere una precisión muy alta o se prefiere menor precisión y mayor rapidez, y viene determinada por el ángulo de aproximación, y la tolerancia que se le da (ver Figura 6.3). En nuestro caso, la precisión se ha puesto a 3º, ya que en el avance se desviará debido a que las ruedas no se comportan igual.

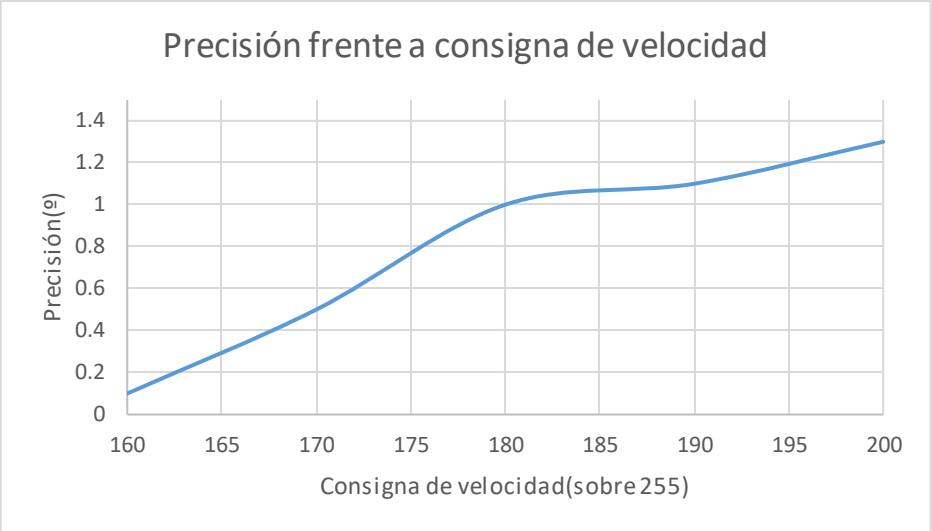

Figura 6.3 Precisión máxima alcanzable variando la consigna mínima de velocidad usada para la precisión. Ángulo vs Consigna de precisión (º)

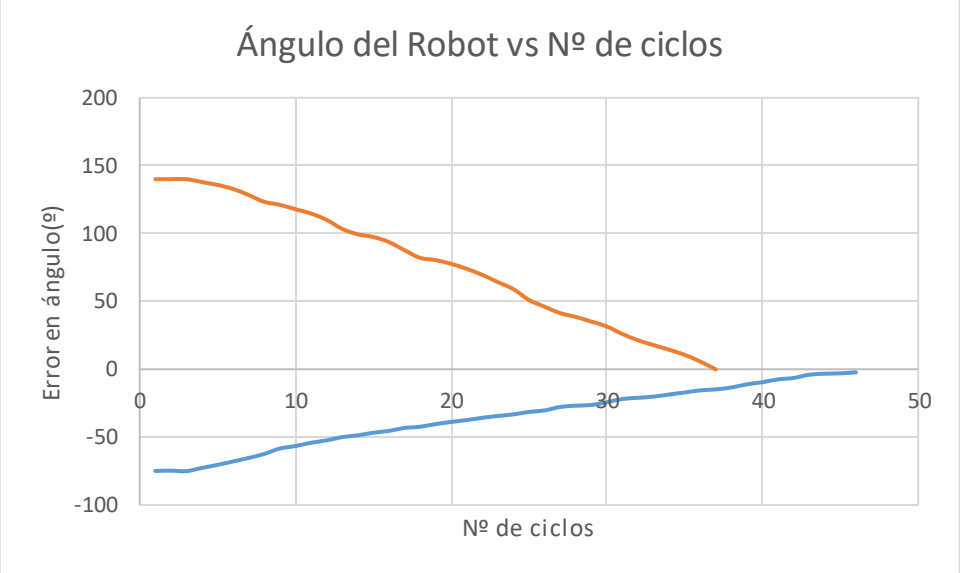

Figura 6.4 Comportamiento del control por número de ciclos. Ángulo vs Ciclo

Se puede observar un comportamiento distinto entre el giro hacia un sentido o hacia otro. Si se mira la Figura 6.4, se observa que, para el control elegido, uno es más rápido que otro, es decir gira más rápido en un sentido que en el otro. Además, no demuestra un comportamiento uniforme, ya que se ven tramos en los que gira más o menos con la misma consigna. Esto hace que sea muy difícil hacer un control perfecto del sistema, ya que se ha de tener muchos valores en cuenta, además de él lado en el que gire. En cuanto al porqué se representa el ángulo respecto al número de ciclos, esto se debe a que, al pararse el robot, cada cierto tiempo, ya no interesa el movimiento respecto al tiempo, si no como se mueve respecto a cada ciclo.

6.3.2 Ajuste del avance

Una vez el robot está orientado, avanzará. El control del avance es muy similar al del ángulo. Si la distancia es menor que un valor aceptable, se considera que ha llegado, si la distancia está entre el valor aceptable y un valor de aproximación, se avanzará con una consigna de velocidad constante de valor 200, y un pulso modulado

con la ecuación $pulso = |distancia| * Kv$. Si la distancia es mayor que el valor de aproximación, se avanzará con pulso de 30 ms y una consigna de velocidad de 250. (Ver figura 6.5)

A diferencia de la orientación, es muy difícil conseguir una buena precisión en el avance, ya que depende de cuestiones como dónde está el punto de avance y el centro del robot considerados, respecto del real. También hay que tener en cuenta cuánto se desvía cuando se le envía una consigna para que avance "recto", y eso depende de cada robot, por lo tanto, no ha sido posible hacer un estudio muy preciso del ajuste óptimo.

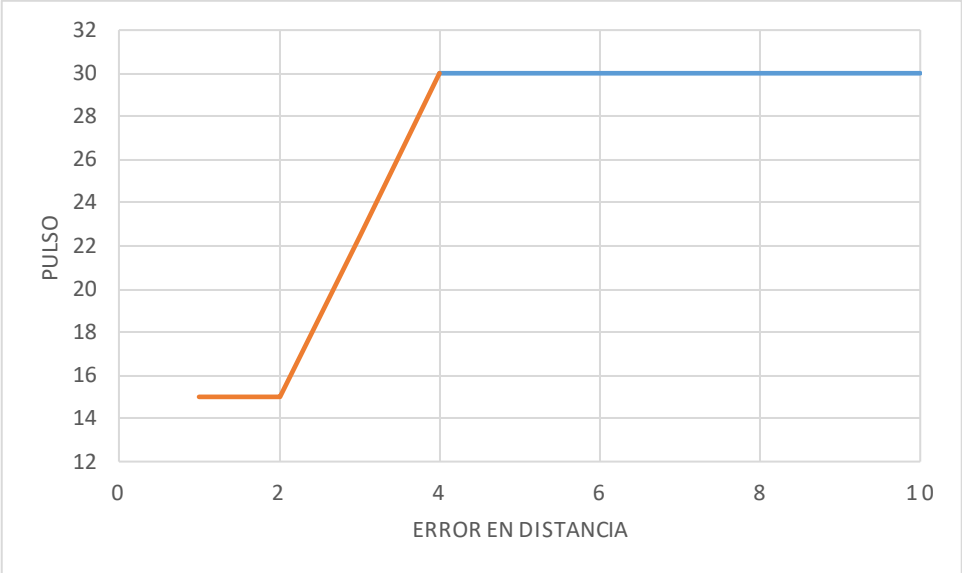

Figura 6.5 Comportamiento del control al modular el tiempo de espera frente al error en distancia del robot con el objetivo. Pulso(ms) vs Distancia(cm). El azulo representa una consigna de velocidad de 250 frente a 255, el naranja de 200 frente a 255.

Para la velocidad máxima que se utiliza como aproximación se ha seleccionado un valor muy grande, ya que como se va a desviar al avanzar (observar en la figura 6.6), ese desvío lo va a tener que corregir en cada ciclo, y al aumentar la velocidad, conseguimos que no tenga que corregir ese ángulo tantas veces por trayecto. La consigna de velocidad para cuando es necesario que el avance sea preciso y la distancia a partir de la cual se quiere avanzar con consignas máximas, se han cogido en base a la experiencia debido a la dificultad de afinar estos parámetros que dependen de muchos factores. La consigna de velocidad máxima, es una velocidad alta, ya que no se requiere tanta precisión en el avance, y la distancia escogida es de 4 cm. En cuanto a la Kv, se ha escogido 7,5, que es el valor que permite que cuando la distancia sea 4 cm, el pulso calculado por el controlador sea de 30 ms. Se ha cogido como distancia de llegada cuando el centro imaginario del robot está a 2cm o menos de distancia, ya que al ser el centro del robot considerado un punto imaginario, se puede aceptar ese margen.

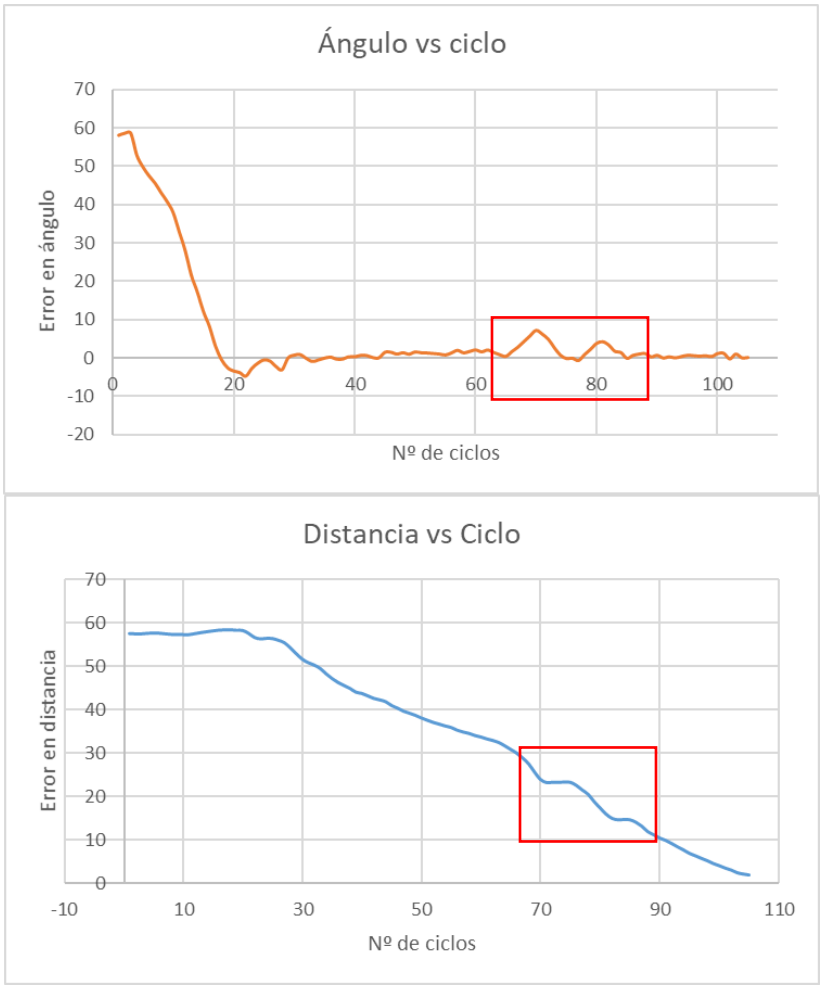

Figura 6.6 Comportamiento del Robot en Ángulo vs Número de ciclos y de Distancia (cm) vs Número de ciclos, donde se puede observar que se tiene que parar el robot a reorientarse.

7 Control Multirobot y planteamiento del algoritmo anticolisiones

En este apartado, se explica cómo se ha adaptado el control de un solo robot, a un control multirobot. Después, se plantea un modelo de algoritmo para evitar colisiones.

7.1 Control Multirobot

Una vez que se controla a un robot, el mayor problema del control multirobot es la gestión de la información dentro del control. Para que el control funcione con normalidad hacen falta 8 variables de Matlab con cada robot. Estas variables son: el número del *target* asociado al robot, la dirección IP del robot, una variable que se utiliza para comunicarse con el robot, otra para el puerto serie, coordenadas del centro actual, el punto de avance actual, los puntos por los que tiene que ir cada robot y los puntos por los que ya ha pasado. Para la gestión de esta información, se ha creado una variable de tipo array del mismo tamaño que el número de robots, que contiene un array de las variables del robot, consiguiendo tener la información de todos los robots en una variable, facilitando la generalización de las funciones con el uso del bucle iterativo. (ver anexo II.4).

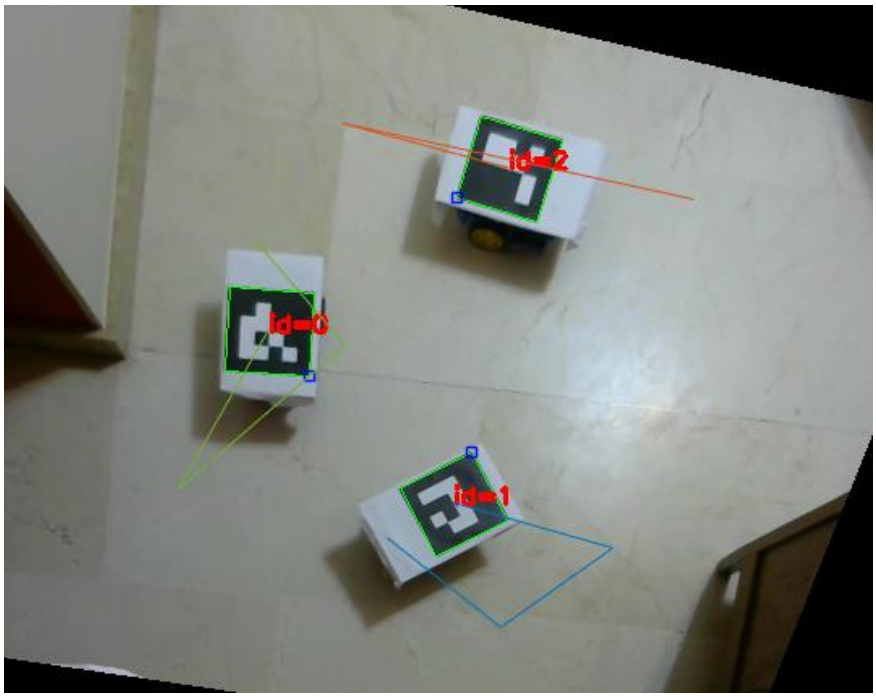

Figura 7.1. Selección de múltiples trayectorias para múltiples robots.

Otra de las cuestiones a tener en cuenta es la homografía, ya que se ha de seleccionar solo una de las posibles matrices de transformación. Para ello se comparan antes de iniciar el ciclo todas las posibilidades, y se mira cuál de ellas es la que mantiene mejor todos los *targets* como cuadrados. Con esta imagen rectificada general de todos los robots, se puede elegir por pantalla una trayectoria con múltiples puntos para cada uno de los robots (ver Figura 7.1).

Se ha de tener en cuenta que, al haber varios objetivos, el comportamiento de la curva de control cambia. Si nos fijamos en la figura 7.2, existen unos cambios bruscos en el error, esto se debe a que ha llegado a su objetivo y va al siguiente. Si nos fijamos solo en uno de los objetivos, la forma de la gráfica se parece a la figura 6.6.

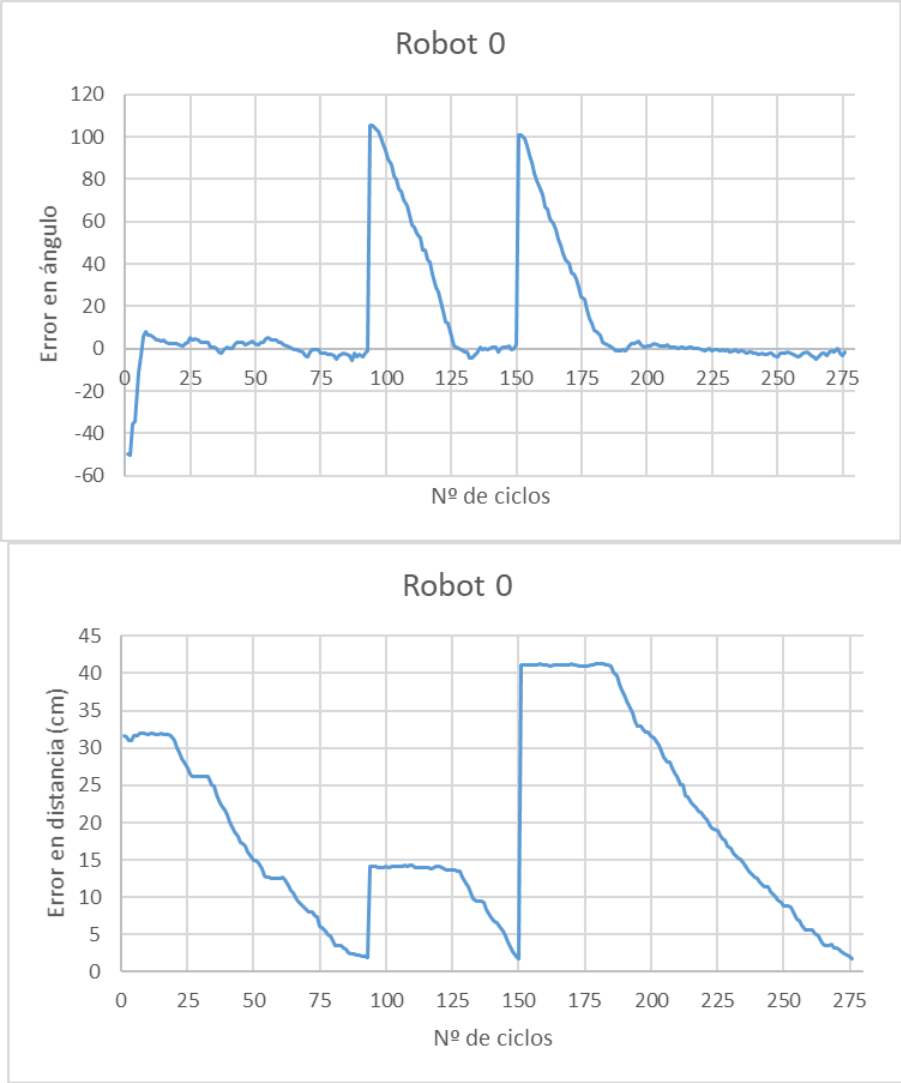

Figura 7.2 Comportamiento del desfase robot denominado 0, frente al número de ciclos transcurridos, donde se puede observar cómo llega y cambia de objetivo.

Se han adaptado el programa principal y las funciones secundarias para que en vez de con un solo robot, funcione con varios al mismo tiempo, maximizando el provecho que da el uso del array. En este proyecto no es necesario tener en cuenta el tiempo, ya que, al cambiar el modo de control al de pulsos, el sistema puede ser todo lo lento que quiera, ya que no afectará a la precisión.

Se ha de comentar que, aunque los robots están compuestos por los mismos componentes, y utilizan el mismo control, éstos no responden igual, por lo que siempre habrá diferencias entre ellos como se puede observar en la figura 7.3. Se puede ver claramente en las pendientes de las curvas que algunos robots son más rápidos que otros, siendo el robot 0 el más rápido seguido por el 2 y por el 1. El robot 1 es un caso particular, ya que además de ser el más lento, se puede ver en ambas gráficas, que tiene problemas al intentar avanzar recto.

Observando el comportamiento del Robot 1 en la figura 7.3, se pueden señalar dos anomalías en el desfase del ángulo, con dos cambios bruscos seguidos. Esta anomalía por lo que se ha visto experimentalmente, sucede en todos los robots, pero

sucede con más frecuencia en este robot en particular. Esta peculiaridad solo surge en el momento que tiene que empezar a girar, y el robot gira durante más tiempo de lo que debería. La causa se encuentra o en el Mini Driver o en los motores, debidos seguramente a su baja calidad. De todos modos, este error es despreciable frente al comportamiento total, ya que se descontrola el giro durante un instante pequeño de tiempo, y se estabiliza con rapidez. Si fuese el avance el que se descontrolará sería un problema.

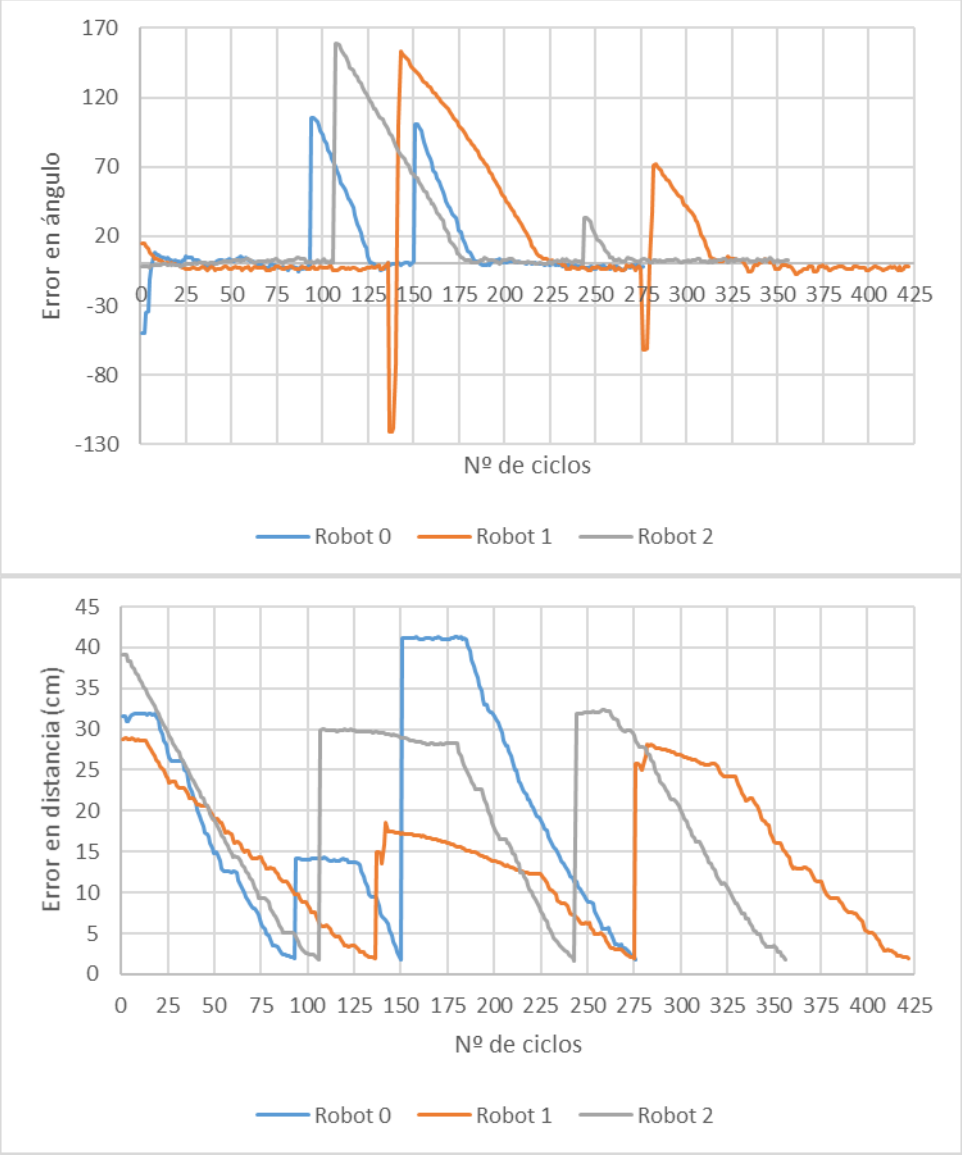

Figura 7.3. Donde se observan los desfases frente al número de ciclos, obtenidos de varios robots controlados simultáneamente.

Figura 7.4 Fotografías en las que se ve distintos momentos de un control con tres robots. Se puede observar que uno de los robots es más lento que el resto debido a la diferencia de motores.

Al final se ha conseguido realizar un control que funcione con robots tan distintos, por lo que se puede considerar un éxito, aunque quedando claro, que el control no reaccionará igual para todos los robots del sistema (ver figura 7.4).

7.2 Algoritmo anticolisiones

Se ha tratado de que el algoritmo planteado sea lo más simple posible, siguiendo con la mentalidad que se ha llevado a lo largo de todo el proyecto, por lo que no tendría sentido utilizar un algoritmo anticolisiones complejo en un control tan simple.

El algoritmo se implementaría justo después de actualizar las posiciones de cada robot y antes de enviar los comandos de movimiento. En primer lugar, se calcularía la distancia entre los centros de todos los robots, y si algún valor es menor que un valor predeterminado entraría en función el algoritmo, si no continuaría con el ciclo normal. Si entra en función el algoritmo y la distancia es menor que un valor crítico, se detendrán ambos robots y se consideraría que han llegado a su destino. En caso de que entre el afirmativo y no esté por debajo de ese valor crítico, se procedería a comprobar si las trayectorias de estos robots se cruzan, dato que se puede saber fácilmente ya que los robots se mueven en línea recta. En caso de que no se crucen, se ignorará, en caso de que sí se crucen, se pararían ambos robots y se consideraría que se ha llegado al destino.

8 Conclusiones y trabajo futuro

Puesto que se ha conseguido un control simple, barato y flexible, para que sea robusto con robots fabricados con componentes de bajo precio y de comportamiento muy distinto entre ellos, se considera que se ha alcanzado el objetivo, y por lo tanto, el proyecto puede considerarse un éxito. Esto implica que se puede controlar una cantidad elevada de robots sin apenas preparación previa, permitiendo intercambiar unidades que por ejemplo están averiadas, que se conseguiría simplemente insertando la tarjeta SD de la Raspberry Pi en otro robot, y no haría falta configurarlo. Si se hubiese utilizado un control más sofisticado, que requiriera el mínimo modelado del robot, esto no se podría realizar.

Hay que admitir que, aunque este modelo es rápido y sencillo de utilizar, si se requiriera un poco más de precisión en los movimientos se tendría que realizar un modelo de los motores del robot. Aunque se podría seguir utilizando el mismo modelo planteado, el único cambio sería que se tendría que realizar el modelo del robot previamente, y guardarlo en algún archivo.

También hay que señalar que, aunque se ha conseguido controlar el sistema, el modelo que se ha utilizado no es el más ideal. Por lo tanto, se tendría que buscar alguna forma de aumentar la velocidad de computación, y depurar el código dentro de lo posible. Otra opción que ya se ha comentado y que es fácil de realizar es el de intercambiar el modelo de motores con reductora con uno diferente. Aconsejamos utilizar el Dagu DG01D 120:1, frente al 48:1 que se ha utilizado, ya que, al tener un mayor par motor, se mueve con más lentitud y es más estable, siendo un mejor componente para el sistema. Y aunque su precio sea un poco superior al modelo utilizado, la diferencia es pequeña. Con estos cambios se podría aplicar perfectamente el segundo modelo planteado, que daría una mayor fluidez al control.

Hay que señalar que, aunque no se ha hecho ningún modelo de los motores, sería interesante utilizar este programa como base para crear uno que calculara el modelo de los motores a través del comportamiento de estos frente a las consignas.

Bibliografía

[Raspberry Pi]

Web oficial https://www.raspberrypi.org/

Hoja de especificaciones de Raspberry Pi 2B
https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

Guía para configurar la conexión automática a una red wifi
https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md

Guía para configurar IP estática https://www.modmypi.com/blog/tutorial-how-to-give-your-raspberry-pi-a-static-ip-address

[OpenCV]

Web oficial https://opencv.org/

Guía para la instalación de módulos extra
https://docs.opencv.org/3.1.0/de/d25/tutorial_dnn_build.html

Enlace de descarga de módulos extra https://github.com/opencv/opencv_contrib

Manual de ArUco en OpenCV
https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html

Enlace para descargar módulo de Matlab
https://github.com/opencv/opencv_contrib/tree/master/modules/matlab

[Matlab]

Web oficial https://es.mathworks.com/?s_tid=gn_logo

Instrucciones de instalación del paquete de Raspberry Pi
https://es.mathworks.com/help/supportpkg/raspberrypiio/ug/install-support-for-raspberry-pi-hardware.html

Información para calcular el ángulo de dos vectores
https://es.mathworks.com/matlabcentral/answers/180131-how-can-i-find-the-angle-between-two-vectors-including-directional-information

[TargetDetector]

Enlace de descarga para el detector de patrones
https://github.com/lbaudouin/TargetDetector

Enlace para el generador de patrones https://github.com/lbaudouin/TargetGenerator

Enlace para la página personal del creador http://lbaudouin.fr/

[ArUco]

Artículos relacionados

Automatic generation and detection of highly reliable fiducial markers under occlusion *Pattern Recognition*, Vol. 47, No. 6. (June 2014), pp. 2280-2292, doi:10.1016/j.patcog.2014.01.005 by S. Garrido-Jurado, R. Muñoz Salinas, F. J. Madrid-Cuevas, M. J. Marín-Jiménez

Automatic generation and detection of highly reliable fiducial markers under occlusion *Pattern Recognition*, Vol. 47, No. 6. (June 2014), pp. 2280-2292, doi:10.1016/j.patcog.2014.01.005 by S. Garrido-Jurado, R. Muñoz Salinas, F. J. Madrid-Cuevas, M. J. Mar'ın-Jiménez

Web oficial https://www.uco.es/investiga/grupos/ava/node/26

[AprilTags]

Web oficial https://april.eecs.umich.edu/wiki/AprilTags

[Hartley]

Multiple View Geometry in Computer Vision Second Edition*

R Hartley, A Zisserman - Cambridge University Press, 2000

[mexopencv]

Página de descarga de GitHub con instrucciones de instalación https://github.com/kyamagu/mexopencv/wiki/Installation-%28Windows%2C-MATLAB%2C-OpenCV-3%29

[G.W.Lucas]

Enlace de articulo por G.W.Lucas del Rossum Project http://rossum.sourceforge.net/papers/DiffSteer/

Anexos

Anexo I Modelo Cinemático del Robot

En este anexo se describe brevemente el modelo cinemático utilizado para obtener el comportamiento del robot.

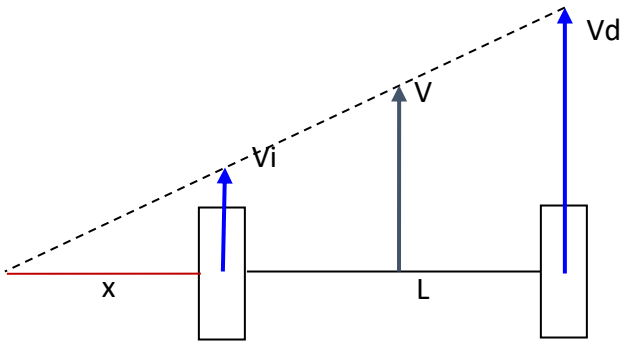

Figura I.1 Modelo cinemático del robot con velocidades diferenciales en las dos ruedas.

Si se observa la figura I.1, se puede ver el modelo de un robot de distancia entre rueda y rueda L, con velocidad Vi en la rueda izquierda y Vd en la rueda derecha, distinta una de otra siendo mayor Vd, provocando un giro en sentido anti horario alrededor del centro instantáneo de rotación que está a una distancia x de la rueda izquierda. En este modelo se considera la V, la velocidad angular W y la L como conocidas, y la x, Vi y Vd como incógnitas.

Si se considera que el punto medio entre las ruedas tiene una velocidad V. Por lo tanto, la velocidad angular W será:

$$W = \frac{V}{\frac{L}{2} + x}$$

Con esta ecuación, somos capaces de despejar x, la distancia de la rueda izquierda al centro instantáneo de rotación.

$$x = \frac{V}{W} - \frac{L}{2}$$

Si planteamos la velocidad en cada rueda en función de la velocidad angular se obtiene:

$$V_i = W * x$$

$$V_d = W * (L + x)$$

Sustituyendo la x y despejando se obtiene:

$$V_i = V - W * \frac{L}{2}$$

$$V_d = V + W * \frac{L}{2}$$

Siendo estás las velocidades necesarias en cada rueda para obtener la V y la W deseada.

Anexo II Programas utilizados en el control

Este anexo consiste en fragmentos del programa que se consideran importantes para entender el proyecto.

Anexo II.1 Programa que inicializa el entorno para el control antes del ciclo

Este programa inicializa todos los parámetros y datos que se necesitan para empezar el ciclo de control, así como la decisión de los puntos por los que tiene que pasar el robot.

```
%Sergio Perez Lloret, abril 2017
%Para que funcine bien hay que borrar todo
clear all
close all


%Se inicia la captura
video = cv.VideoCapture(0);
%se define el diccionario de patrones a usar
dict = cv.dictionaryDump('4x4_100');


while (video.grab())            % get a new frame from camera

    image = video.retrieve();
    imshow(image);
    k=get(gcf,'CurrentCharacter');
    if k=='q', break; end

end

%se detectan los targets
[corners, ids] = cv.detectMarkers(image, dict);


%Si detecta algun target continua el programa, si no sale
if (size(ids,2) > 0)
    %obtiene la matriz H para la Homografía
    H= GetH( corners,ids );

    %Transformamos la imagen original para buscar los puntos iniciales
    %De cada robot para meterlos como información
    tform = projective2d(H');
    fondo=0;
    image2= imwarp(image, tform, 'fill', fondo);

    %conseguimos los nuevos corners y ids de imagen rectificada
    [corners, ids] = cv.detectMarkers(image2, dict);
    image = cv.drawDetectedMarkers(image2, corners,'IDs',ids);
    %Inicializamos la matriz de información del robot
    RobotInfo = GetRoboInfoIni(ids,corners);

    %Damos la opción de seleccionar caminos para que siga el robot
    RobotInfo = PathSelector(RobotInfo,image);
```

```
    %Calculamos el tamaño del pixel
    TaPixel=13.1/norm(corners{1}{1}-corners{1}{2});

    %Conectamos los robots
    RobotInfo=ConectRobots(RobotInfo);

    %iniciamos el ciclo
    Tfg_Ciclo

else
    fprintf('No Hay targets detectados.\n')
    fprintf('Cerrando Programa.\n')

end
```

Anexo II.2 El ciclo de control

En este programa es donde se actualiza la posición del robot y se calcula en una función la acción a realizar.

```
%Sergio Pérz Lloret, mayo 2017
%Ciclo principal de control de robots


 while (video.grab()) %Bucle principal de visión

     %Variable para saber si todos los robots han llegado a su destino
     comproDest=0;

     %Actualiza la imagen de la camara, la transforma con la H
     %y detecta los target en la imagen transformada
     image = video.retrieve();
     tform = projective2d(H');
     fondo=0;
     image2= imwarp(image, tform, 'fill', fondo);

     %Se dibujan los targets para verlos mejor
     [corners, ids] = cv.detectMarkers(image2, dict);
     image = cv.drawDetectedMarkers(image2, corners,'IDs',ids);
     imshow(image)
     hold on
     %actualizamos el centro actual de la matriz y realizamos el
control
     for i=1:size(ids,2)
         for j=1:size(RobotInfo,2)
             if ids(i)==RobotInfo{j}{1}

[RobotInfo{j}{5},RobotInfo{j}{6}]=GetCenyAv(corners{i});
                 if size(RobotInfo{j}{7},1)>=1

plot([RobotInfo{j}{5}(1),RobotInfo{j}{7}(1,1)],[RobotInfo{j}{5}(2),Rob
otInfo{j}{7}(1,2)]);

plot([RobotInfo{j}{5}(1),RobotInfo{j}{6}(1)],[RobotInfo{j}{5}(2),Robot
Info{j}{6}(2)]);
                     [RobotInfo{j}] =
Get_and_Send_Robot_speed(RobotInfo{j},TaPixel);
                     break
```

```
                else
                    comproDest=comproDest+1;
                end
            end
        end
    end
    hold off

    %Pausa para asegurarse de que ya se han movido todos los robots
    pause(0.01)
    %si llegamos al final del todo que se cierre
    if comproDest==size(RobotInfo,2)
        break
    end

end
```

Anexo II.3 Función de control

Esta es la función que se utiliza para calcular la acción en cada robot y mandar el comando al robot.

```
%Sergio Pérez Lloret, Mayo 2017

function [  RobotInfo ] = Get_and_Send_Robot_speed( RobotInfo,TaPixel)

            %Calculamos la distancia
            Vobj=RobotInfo{7}(1,:)-RobotInfo{5}(1,:);
            Dist= norm(Vobj)*TaPixel;

            %Distancia entre centro y dirección
            Vav=RobotInfo{6}(1,:)-RobotInfo{5}(1,:);

           %Vectores de posición en 3D
            va=[Vobj,1]; % vector de direccion objeto
            vb=[Vav,1]; % vector de dirección robot

            %Calculo de
            alphat= atan2d( cross(va,vb) , dot(va,vb) );
            alpha=alphat(3);

          %Comprobamos si ha llegado a su destino
          if Dist<=2
                    %ha llegado a su destino
                    RobotInfo{8}(end+1,:)= RobotInfo{7}(1,:);

                    %eliminamos la posición que ya ha estado
                    RobotInfo{7}(1,:)=[];

                    %salimos de la función
                    return
          else
                %comprobamos si el angulo es mayor que el adecuado
                if abs(alpha)>3

                    % Si el angulo es muy grande se usa la maxima
accion
```

```
                    if alpha>10
                        pulso=30;
                        vi=-200*sign(alpha);
                        vd=200*sign(alpha);
                    else

                        %Si no se usa un control proporcional con
velocidad
                        %menor

                        pulso=abs(alpha)*3;

                        if pulso<=15
                        pulso=15;
                        end

                vi=-160*sign(alpha);
                vd=160*sign(alpha);

                    end

                %si está ya orientado
                else
                    %su es mayor que una distancia señalada, se pone
                    %consigna maxima
                    if Dist>4
                        pulso=30;
                        vi=250;
                        vd=250;
                    else
                     % si no se calcula con un control y consigna
menor
                     pulso=abs(Dist)*7.5;

                     if pulso>30
                        pulso=30;
                     end

                     if pulso<=15
                        pulso=15;
                     end

                     vi=200;
                     vd=200;

                    end
                end
           end
        %enviamos el comando de velocidad aqui para ganar tiempo, y no
        %repetir las comprobaciones
              pulso=round(pulso);
              res=mensajepi(vd,vi,pulso,RobotInfo{3});

end
```

Anexo II.4 Vector de variables de robot

Función que inicializa las variables y describe para que sirve cada una.

```
%Sergio Perez Lloret, abril 2017

function [ RobotInfo ] = inicializarRoboInfo( RobotInfo,Ip,Id )
%inicializarRoboInfo Inicializamos una fila del vector
%   Detailed explanation goes here

    RobotInfo{1}=[Id]; %Identidad del target asociado
    RobotInfo{2}=[Ip]; %Ip del robot
    RobotInfo{3}=[]; %Comunicacion con puerto serie
    RobotInfo{4}=[]; %conexion con robot
    RobotInfo{5}=[]; %Centro actual
    RobotInfo{6}=[];  %Punto de avance
    RobotInfo{7}=[];    %Puntos a los que tiene que ir
    RobotInfo{8}=[];    %Puntos a los que ha ido
    RobotInfo{9}=[];    %estado del robot
end
```

Anexo II.5 Vinculación de la IP con el target

Función para saber la IP del robot en función del número asignado al *target*.

```
%Sergio Perez Lloret, abril 2017

function [ ipRobot ] = getIp( id )
%getIp Devuelve la Ip de un robot según la ID del target
%
    ipRobot=strcat('192.168.137.',num2str(id)+3);
end
```

Anexo II.6 Función para enviar mensajes por el puerto serie

Función que ayuda a mandar mensajes al puerto serie.

```
%Sergio Pérez Lloret, Marzo 2017


function [ ok ] = mensajepi( iz,dr,pulso,myserialdevice)
%Crea un vector de numeros ASCII para comunicación directa de la
%Raspberrypi con el pwm, envía el comando de velocidad a esta
%Y comprueba que se ha recibido correctamente
 comando = [uint8(':') uint8(num2str(iz)) uint8(',')
uint8(num2str(dr)) uint8(',') uint8(num2str(pulso)) 10];
 write(myserialdevice,comando,'uint8')
 res = read(myserialdevice,16);
    if isequal(res,comando)==0
        ok=0;
    else
        ok=1;
    end
end
```

Anexo III Tiempo de un ciclo

En este anexo se recogen los tiempos medios de un solo robot. Los tiempos obtenidos para varios robots son muy parecidos a los obtenidos para uno solo, aumentando solo el tiempo de control.

Tiempo medio de recolección de imagen (segundos)	0,001890165
Tiempo medio transformación imagen (segundos)	0,042482376
Tiempo medio de detección de targets (segundos)	0,00839114
Tiempo dibujar targets (segundos)	0,002471345
Tiempo medio de control (segundos)	0,031370612
Tiempo medio de ciclo (segundos)	**0,14153686**

Figura III.1 Tabla con los tiempos medios de distintas partes del código

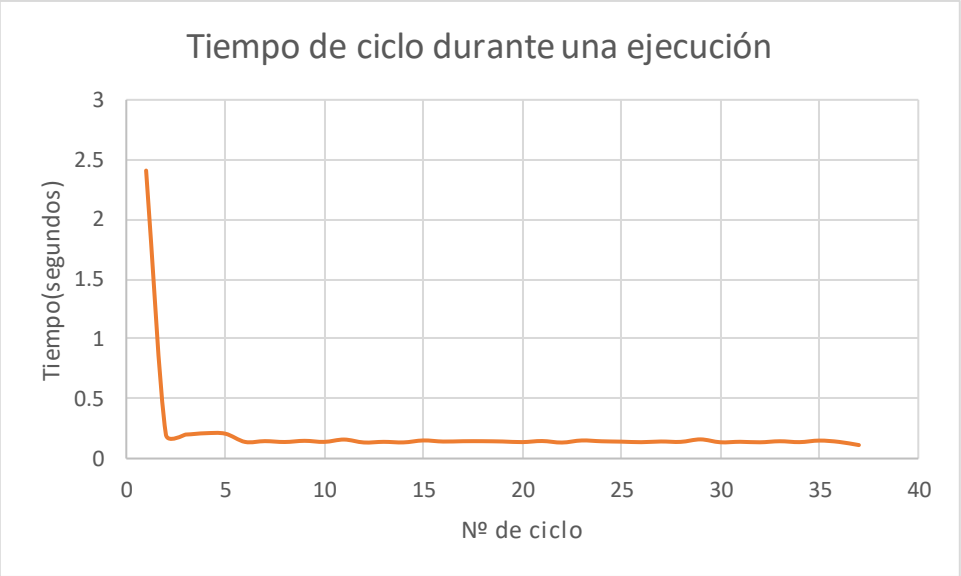

Figura III.2 Grafica de tiempo de computación del ciclo frente a número de ciclos

Si se observa la figura III.2 se puede ver que, en los primeros ciclos, sobre todo el primero, el tiempo de ciclo es mucho mayor, pero luego se estabiliza. Esto se debe a las tareas de inicialización y reserva de memoria que hace el programa al arrancar el programa de control, y sucede en todas las pruebas realizadas.

Anexo IV Manual de usuario

Este anexo explica de manera resumida cómo montar y poner en marcha el sistema completo. Para ello se presupone que se tiene el modelo del robot o uno similar desde un principio. Se empieza explicando cómo se instalan todas las librerías utilizadas por el sistema, seguido por la configuración que se ha de realizar en todos los dispositivos para dejarlos listo. Por último se explica cómo interactuar con el programa creado.

Anexo IV.1 Instalación de librerías

El primer paso es la descarga de todos los archivos necesarios para las librerías, siendo estos la librería de OpenCV versión 3.1 como mínimo, los módulos extra correspondientes de OpenCV y mexOpenCV. La librería de OpenCV y los módulos extra se pueden descargar o en la página oficial de OpenCV, proporcionada en la bibliografía o, en su perfil de GitHub. MexOpenCV, solo se puede descargar de la página de GitHub del usuario **kyamagu** también proporcionada en la bibliografía.

Para la instalación en Matlab, hacen falta algunos programas extra para poder crear, compilar y ejecutar la librería. Estos programas serían Cmake, o similar, y un compilador de C++, además de temer en cuenta Matlab.

Una vez se tiene todos los archivos y programas necesarios se puede empezar la instalación. El primer paso sería utilizar Cmake para crear la librería que contenga OpenCV y los módulos de expansión. Cmake creará un ejecutable, que si se ejecuta y compila en C++, se creará una librería. Una vez se ha compilado la librería, se procederá a ejecutar el siguiente comando en Matlab.

```
>> cd('C:\dev\mexopencv')
>> addpath('C:\dev\mexopencv')
>> addpath('C:\dev\mexopencv\opencv_contrib')
>> mexopencv.make('opencv_path','C:\dev\build\install', 'opencv_contrib',true)
```

Como es un proceso en el que hay que seguir una serie de pasos meticulosos, es recomendable seguir la guía de instalación de kyamagu, disponible en su perfil de GitHub, para compilar, e instalar OpenCV con los módulos adicionales de Matlab.

También hay que instalar el módulo de RaspberryPi en Matlab, para ello se ha de acceder a Matlab, clicar en la pestaña Add On, y buscar el módulo. Existe un enlace en la bibliografía con instrucciones más claras.

Anexo IV.2 Configuración de dispositivos

Antes de poner en marcha el programa hay que configurar los dispositivos del sistema, empezando por los robots. El primer paso con el robot es la instalación en la microSD de los archivos para funcionar con el módulo de Matlab. Este archivo se puede instalar durante la instalación de dicho modulo en Matlab, o más tarde con el comando " targetupdater". Siguiendo las instrucciones que te indican durante la instalación, configurarás la Raspberry Pi para trabajar con Matlab. También hay que configurar la Raspberry Pi para que se conecte automáticamente a una red WiFi generada en otro dispositivo, y con una IP estática. Para ello hay que conectarse al entorno de la Raspberry

con un teclado y a una pantalla. Para configurarlo bien, se aconseja seguir las dos guías que están provistas en la bibliografía. Recordad que la IP de cada robot tiene que tener una relación con el número asociado al *target* del robot.

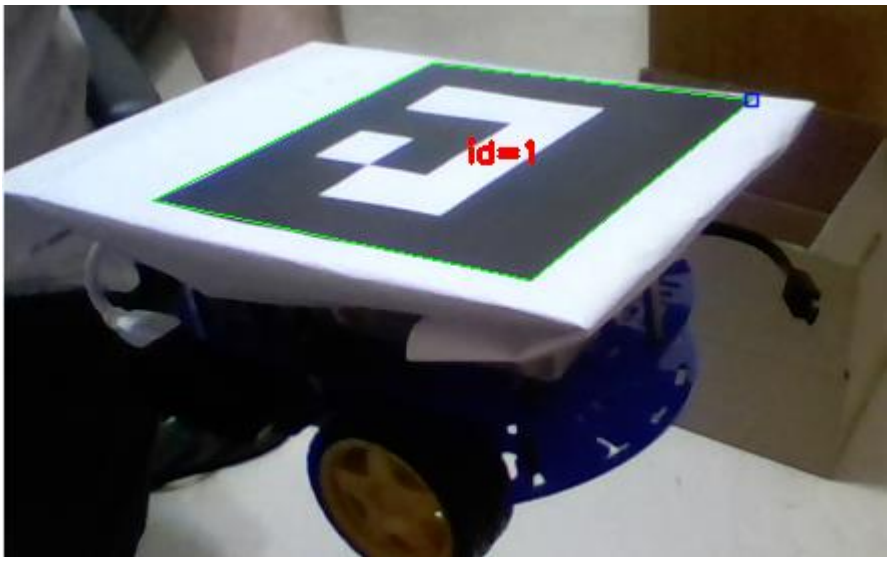

Figura IV.1 Robot con target *colocado en la orientación correcta*

Por último, hay que colocar el target correspondiente a cada robot en la dirección adecuada (ver figura IV.1) y generar una red WiFi con el ordenador central. Si se dispone de Windows 10, el propio sistema puede generar la red. Para acceder se tiene que seguir estos pasos: Pulsar botón de Windows+i. Seleccionar red e internet, y de ahí vamos a Zona con cobertura inalámbrica móvil, donde se puede generar la red que se necesita.

Anexo IV.3 Interfaz con el usuario

Una vez está todo instalado y listo para funcionar, se ejecuta el programa Main de todos los utilizados por el sistema, asegurándonos antes de que la cámara está conectada al ordenador. Matlab procederá a mostrar lo que capta la cámara por pantalla, permitiendo ajustar la posición de la cámara a una que nos sea favorable. Cuando se tiene la cámara en la posición deseada, se pulsa la tecla "q" para continuar. Asegúrese de que los robots están listos para trabajar. Si durante la ejecución del programa, algún robot da error de conexión se parará.

Si el programa no detecta ningún robot, esté se cerrará, de lo contrario, te volverá a mostrar la imagen captada por la cámara, pero esta vez rectificada, y te pedirá que selecciones el robot al que le quieres asignar los puntos. Para seleccionar un robot, simplemente clica con el botón izquierdo del ratón alrededor del centro del robot. Una vez lo tengas seleccionado, te pedirá que cliques en la imagen, los puntos a los que quieres que vaya ese robot, y te pedirá que confirmes. Una vez confirmado, se te dará la opción de coger más puntos con el mismo robot, si dices que no, se te dará la opción de seleccionar un robot de nuevo, o de ir al control. Una vez se ha salido de esté menú, el programa correrá solo hasta que todos los robots hayan pasado por los puntos indicados, que se parará el programa.