Universidad Zaragoza
1542

Trabajo Fin de Grado

EXPLORACIÓN COOPERATIVA DE UN ENTORNO CON UN EQUIPO MULTI-ROBOT DE BAJO COSTE

Autor/es

Carlos Morales Jarque

Director/es

Rosario Aragüés Muñoz

Raúl Igual Catalán

Escuela Universitaria Politécnica de Teruel

2018

RESUMEN

El presente Trabajo de Fin de Grado consiste en la programación e implementación de un sistema formado por un equipo multi-robot de bajo coste, un sistema sensorial externo (cámara) y un ordenador encargado de la recepción, procesamiento y envió de la información. Esto implica la elección tanto de la plataforma de bajo coste como del protocolo de comunicación del sistema, además de la implementación y ajuste de un controlador que permita al robot alcanzar las coordenadas deseadas en función de la información sensorial recibida.

Finalmente, se desarrollaron dos algoritmos, el Algoritmo en 2 Fases, en el cual el robot rehúsa a recibir información una vez conocida su posición inicial, por lo que únicamente rota un ángulo deseado y avanza en línea recta hacia el objetivo la distancia deseada, y el Algoritmo *GoToGoal,* en el cual el robot consulta constantemente su posición y orientación, de forma que a través del controlador implementado es capaz de llegar al objetivo deseado.

Asimismo, se realizaron variantes del Algoritmo *GoToGoal* que permitieron, entre otras, utilizar más de un robot real en el sistema. Además, como ejemplo de aplicación se realizó un guiado del entorno multi-robot implementando una resolución por diagramas de Voronoi.

ABSTRACT

The following Final Project consists of the programming and implementation of a system formed by low cost multi-robot equipment, an external sensory system (camera) and a computer which receives, processes and sends the information. This implies both the selection of the low cost platform as the system communication protocol, and the implementation and adjustment of a controller which allows the robot to reach the desired coordinates based on the received sensorial information.

In the end, two algorithms were developed: the 2 Phases Algorithm, in which the robot refuses to receive any further information once the initial position is known, so that it only rotates to a desired angle and moves forward the desired distance towards the goal; and the *GoToGoal* Algorithm, in which the robot constantly checks its position and orientation in order to reach the goal through the implemented controller.

Additionally, variants of the *GoToGoal* Algorithm were developed, which allowed, for instance, the use of more than one real robot in the system. Furthermore, as an application example, a guidance of the multi-robot environment was performed by implementing a resolution by Voronoi diagrams.

CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1 Introducción

Los equipos multi-robot permiten realizar de forma más eficiente y robusta tareas de exploración autónoma, rescate y detección de víctimas, vigilancia de instalaciones, monitorización de recursos, o transporte cooperativo. En el TFG se desarrollará una aplicación de exploración cooperativa de un entorno con un equipo multi-robot de bajo coste ("low cost").

Para validar las diferentes estrategias muti-robot en un escenario controlado, es habitual utilizar un sistema de monitorización externo, es decir, una cámara. Esta configuración del sistema permite hacer pruebas sin hacer una gran inversión en hardware de otras plataformas robóticas más potentes y complicadas de poner en marcha. Todo ello en un entorno reducido como es el interior de un laboratorio.

En el caso del presente proyecto, se han utilizado los siguientes elementos en el sistema:

- **Cámara Kinect**. Dado que el tamaño de los robots es relativamente pequeño, se hará el uso de un trípode pues la altura que proporciona a la cámara es suficiente como para cubrir la zona de trabajo.
- **Ordenador**: Es la herramienta que procesa las imágenes capturadas por la cámara, ejecuta los algoritmos programados, realiza la toma de decisiones y es el que envía los comandos por bluetooth a los robots.
- **Robot Sparki**: Es el robot principal del proyecto, que a través del módulo bluetooth conectado a éste, recibirá las órdenes de navegación calculadas por el PC por lo que su potencia de cálculo es mucho menor.
- **Marcas ArUco**: Marcas visuales para localizar el robot y los puntos de interés del entorno.

A continuación, se muestra un ejemplo de disposición del sistema a controlar:

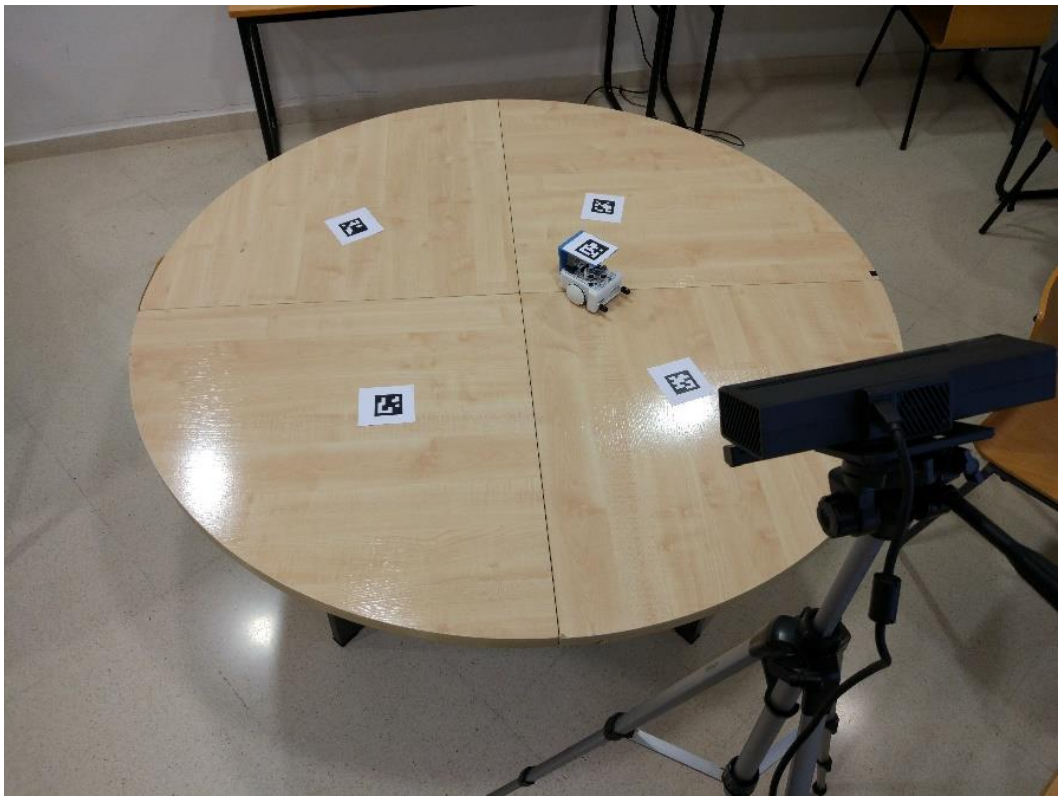

Ilustración 1: Disposición inicia l del sistema

1.1 Objetivos

A la vista del sistema de la ilustración 1, se pueden obtener una idea aproximada de los retos del proyecto. Este involucra diversas tecnologías, una variedad de lenguajes de programación, el uso de funciones documentadas, el establecimiento de mecanismos de control automático, etc. Al tratarse de un problema muy amplio, se acotó el mismo planteando los siguientes objetivos en el Trabajo de Fin de Grado:

- Elección de plataforma de bajo coste, basado en las diferentes alternativas tales como características de movimiento, tipos de sensores, métodos de conectividad y entornos de programación y control.
- Programación del protocolo de comunicación entre el robot y el ordenador.
- Implementación del uso de sensores externos para su utilización en el bucle de control del sistema de navegación del robot.
- Implementación y ajuste de un controlador que permita al robot alcanzar las coordenadas deseadas en función de la información sensorial recibida por el mismo.

Finalmente, durante su desarrollo se han abordado objetivos más ambiciosos dada la posibilidad de tratar cada uno de los mismos desde distintos enfoques. A lo largo de la memoria, se irán mostrando los distintos objetivos adicionales propuestos con sus respectivas soluciones, teniendo un repaso de los mismos en el apartado de Conclusiones. Asimismo, se contará con unos objetivos para un trabajo futuro, incluidos también en el apartado de Conclusiones.

1.2 Estructura del proyecto

La memoria del presente proyecto se organiza en 11 capítulos más anexos.

El primer capítulo consiste en una breve introducción del proyecto y de los objetivos a alcanzar en el mismo.

El segundo capítulo explica los parámetros que se tuvieron en cuenta para la elección del robot de bajo coste.

En el tercer capítulo se explican las características más destacadas del robot elegido (Robot Sparki) por su relevancia para este proyecto (sensores, programación, etc).

En el cuarto capítulo se describen los distintos procedimientos para poder realizar la comunicación entre el robot y el ordenador, así como la comunicación a través de Matlab y el robot.

En el quinto capítulo se describe el sistema sensorial externo utilizado para cumplimentar los objetivos del proyecto. Éste sistema incluye una cámara Kinect y una librería de marcas ArUco para la creación de un sistema de referencia.

En el sexto capítulo, una vez descrito el sistema, se muestra un ejemplo en el que se calculan las posiciones relativas entre dos marcas ArUco.

En el séptimo capítulo se desarrolla el algoritmo de control del robot Sparki.

En el octavo capítulo se describen los distintos algoritmos programados para el cálculo de la trayectoria del robot Sparki.

En el noveno capítulo se muestra un ejemplo de aplicación de los algoritmos expuestos previamente utilizando diagramas de Voronoi en un entorno multi-robot simulado.

En el décimo capítulo se describen los diferentes ensayos realizados con cada uno de los algoritmos programados a fin de estudiar su funcionalidad. Asimismo, se comentan los resultados obtenidos en los mismos.

En el undécimo capítulo se describen los problemas más destacados durante el desarrollo del proyecto, así como las soluciones aplicadas a los mismos.

En el último capítulo se corresponde a las conclusiones finales, así como líneas de trabajo futuro. Se exponen las conclusiones obtenidas sobre el diseño y desarrollo del sistema, así como las consideraciones finales a tener en cuenta.

Finalmente, se encuentran las referencias bibliográficas que se utilizaron para el desarrollo del proyecto. Los anexos se incluyen en un CD. En éstos se encuentran tanto los programas realizados, los resultados de los ensayos como los vídeos de muestra de cada uno de los algoritmos.

2 Elección de robot de bajo coste

Para la realización del proyecto se consideró la utilización de uno de los 3 robots de bajo coste que, clasificados según su programación, son:

- **Mbot**, cuyo entorno de programación está basado en Arduino.[1]
- **GoPiGo**, cuyo entorno de programación está basado en Raspberry Pi.[2]
- **Sparki Robot**, cuyo entorno de programación, al igual que el Mbot, está basado en Arduino. [3]

En la tabla (1) se muestra una comparativa entre las distintas especificaciones de cada uno de los modelos propuestos. Tras esto, debido a la falta de conocimiento de programación con el entorno de Raspberry Pi, el robot GoPiGo es descartado. Si bien, ambos robots Mbot y Sparki tienen especificaciones similares, se acaba eligiendo el robot Sparki ya que, además de tener conocimientos de programación en Arduino, se dispone de una gran cantidad de información en la red a la cual se puede acceder en caso de tener algún problema con la comunicación u otros parámetros.

Otra de las ventajas del Robot Sparki frente al Mbot es que éste cuenta con motores paso a paso a diferencia del Mbot, que cuenta con motores DC. La ventaja de los primeros se explicará en apartados posteriores. Además, el Robot Sparki no requiere un ensamblado previo evitando así problemas relacionados con la compatibilidad de los periféricos que puedan comunicarse con el robot.

En todos los modelos se permitía la posibilidad de utilizar una Toolbox de Matlab para ejecutar programas de Arduino que, sin embargo, no se ha llegado a utilizar pues el propio entorno de programación del Robot Sparki (*SparkiDuino*) contaba con las librerías necesarias para realizar las tareas requeridas en el proyecto.

Modelo	***Mbot*** 	***GoPiGo*** 	***Sparki Robot*** 
Precio aproximado	89.90 €	169.90€	125€
Ensamblaje requerido	Sí	Sí	No
Entorno de programación	Bloques(Basado en Scratch 2.0) o Arduino	Raspberry Pi: Python, Java, C/C++, etc	Bloques (Ardublock) o Arduino
Comunicación	Bluetooth o 2.4G	Bluetooth o 2.4G	Bluetooth
Disponibilidad de encoders	No	Sí	No
Tipo de carga	Pilas AA o Batería	Pilas AA o Batería	Pilas AA o Batería
Tipo de movimiento	Diferential Drive	Diferential Drive	Diferential Drive
Posibilidad de utilizar Matlab y Simulink	Sí, a través de una Toolbox para Android	Sí a través de una Toolbox para Raspberry Pi	Si, a través de una Toolbox para Android.
Fuentes de información	Foros y tutoriales dedicados a distintos robots basados en Arduino	Foros y tutoriales dedicados a modelos con Raspberry Pi	Foros y tutoriales dedicados exclusivamente a este modelo
Sensores disponibles	Destacan: 2 Motores DC, sensor de luminosidad, emisor y receptor de infrarrojos, pulsador programable, sensor de proximidad (ultrasonidos), sensor sigue-líneas, zumbador, posibilidad de conectar periféricos externos	Destacan: 2 Motores DC, 2 servomotores, sensor de proximidad (ultrasonidos), 1 puerto analógico, 1 puerto digital, 1 puerto serie I2C	Destacan: 3 Motores paso a paso, sensor de luminosidad, emisor y receptor de infrarrojos, sensor sigue-líneas, sensor de proximidad, 1 servomotor, pantalla LCD 128x64, posibilidad de conectar periféricos externos

Tabla 1: Elección de robot de bajo coste

3 Robot Sparki

A continuación, se mencionan las propiedades más relevantes del Robot Sparki entre las que destacan las características de movimiento del robot, los actuadores principales, su programación y conectividad.

3.1 Movimiento *diferential drive*.

Un robot con movimiento diferencial (*diferential drive)* es un robot móvil cuyo movimiento está basado en 2 ruedas separadas colocadas en ambos lados de éste. Puede así cambiar su dirección variando la velocidad angular de cada una de las ruedas[4]. Este tipo de movimiento predomina en los robots de bajo coste.

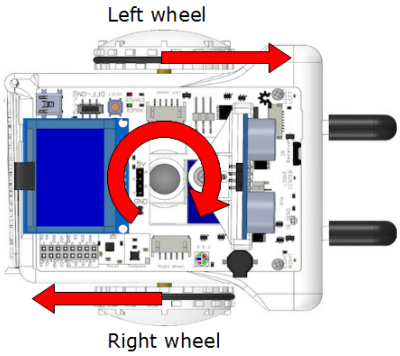

Ilustración 2: Movimiento Diferencial Robot Sparki

3.2 Actuadores principales del robot Sparki

El robot Sparki cuenta con más de 40 sensores y actuadores entre los que destacan, por su relevancia para la realización del proyecto, los motores paso a paso (*stepper*) y los pines para la conexión del módulo bluetooth (*HC-06*).

El robot Sparki, cuenta con 3 motores paso a paso (*stepper*), uno localizado en el *gripper* (la pinza) y los otros 2 en las ruedas:

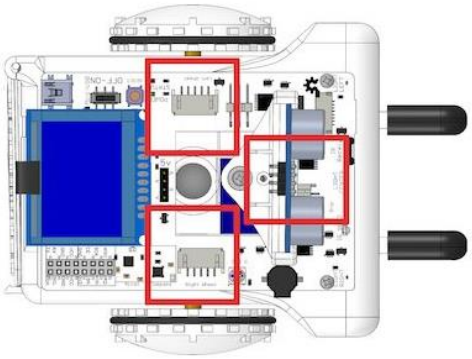

Ilustración 3: Motores de paso del Robot Sparki

Antes de explicar por qué se cuenta con motores paso a paso en lugar de motores DC, es necesario recurrir a la definición del éstos.

Un motor paso a paso (o "PAP") es un dispositivo electromecánico capaz de convertir una serie de impulsos eléctricos en desplazamientos angulares discretos. Esto significa que, a diferencia de un motor convencional (que gira de forma continua), éste es capaz de

avanzar una serie de grados (o pasos) a la vez, dependiendo del estado de sus entradas de control.[5]

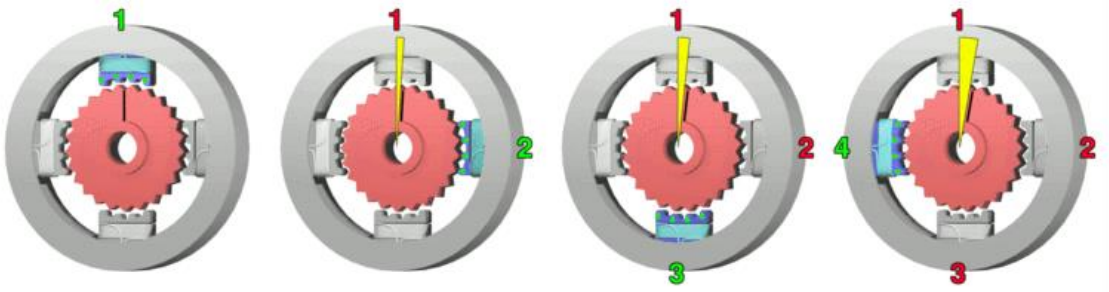

Ilustración 4: Motor paso a paso

De esta forma, el diseño de estos motores permite predecir el movimiento del robot. Los motores paso a paso del robot Sparki necesitan 4096 pasos para dar una vuelta completa, por lo que, haciendo un sencillo cálculo, conociendo el radio de cada rueda, se puede determinar la distancia deseada a recorrer por el robot.

Otra de las ventajas del uso de motores paso a paso es que a diferencia de los motores DC, éstos no necesitan una calibración previa, pues ésta es necesaria con los motores DC para que las velocidades en ambas ruedas sean iguales permitiendo así que el robot pueda ir en línea recta.

Además, durante la realización del proyecto se pudo apreciar que la velocidad de movimiento del robot no se veía afectada con la pérdida de potencia de las pilas, es decir, tras un tiempo prolongado de uso del robot, la velocidad de los motores paso a paso no se redujo, permitiendo así trabajar con éste en las mismas condiciones que con unas pilas nuevas, hasta el agotamiento de las mismas.

El robot cuenta además con 16 patillas que posibilitan la conexión de periféricos. Cuatro de las mismas serán utilizadas para conectar el módulo bluetooth HC-06.

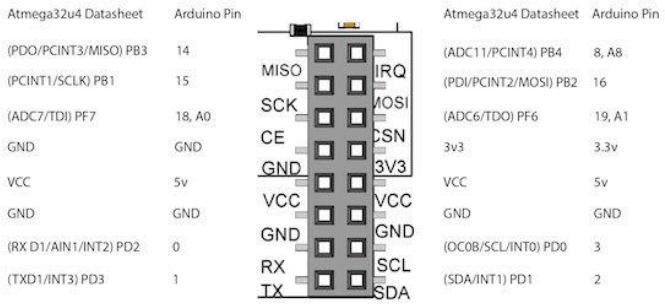

Ilustración 5: Patillas Robot Sparki

Otros sensores y actuadores del robot, como el sensor de ultrasonidos o los sensores infrarrojos no se utilizarán para el proyecto acometido, pero no se descarta su uso en aplicaciones futuras, como, por ejemplo, evitar obstáculos haciendo uso del sensor de ultrasonidos.

3.3 Entorno de programación del robot Sparki

El robot Sparki es un robot principalmente utilizado para fines educativos, por lo que cuenta con diferentes entornos de programación[6] en función del nivel de conocimiento del usuario:

- *SparkiDuino*: entorno de programación basado en Arduino, cuyo lenguaje es C/C++. Además, viene con todas las librerías y drivers necesarios instalados.
- *Ardublock*: destinado principalmente para los niños, este entorno únicamente requiere el uso de bloques para programar sin la necesidad de escribir una sola línea de código.

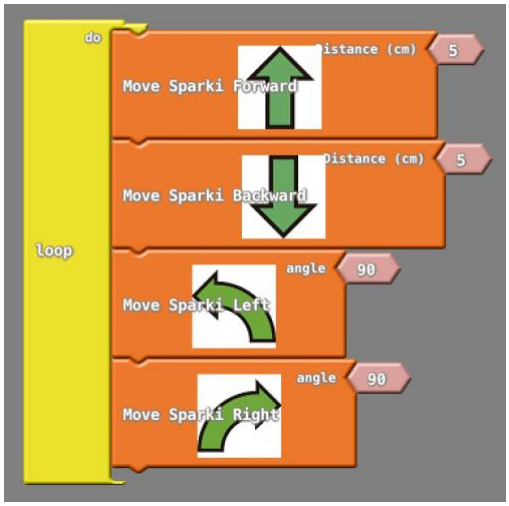

Ilustración 6: Ejemplo de programación en bloques

- *MiniBlock:* es la versión antigua de *Ardublock* y actualmente está obsoleta.

El entorno de programación elegido es SparkiDuino pues es el más avanzado y el que más flexibilidad proporciona al usuario. Además, no es necesario descargar ninguna librería adicional pues cuenta con todas las necesarias para poder programar al robot. En caso de necesitar utilizar el programa Arduino, es posible obtener las librerías desde la página oficial del robot Sparki. A pesar de ello, durante el desarrollo del proyecto no fue necesario utilizar el programa Arduino, ni la Toolbox de Arduino para Matlab.

3.4 Programación Arduino básica

Como se menciona anteriormente, se utilizará el programa *SparkiDuino* para la programación del robot. Como primer paso para programar el robot se requiere de la instalación de tanto los drivers del robot, de la propia placa del robot, como de las librerías necesarias para su funcionamiento.

3.5 Instalación de drivers y librerías

Para la instalación de los drivers, se requiere instalar el programa SparkiDuino, dado que, a pesar de prescindir de su uso para la realización del proyecto, éste realiza automáticamente la instalación de los drivers y librerías actualizadas evitando problemas de compatibilidad haciéndolo manualmente, lo que agiliza además éste proceso significativamente.

Una vez realizadas las instalaciones de todos los elementos mencionados anteriormente es el momento de comenzar con la programación básica.

3.6 Librería Sparki

La librería <sparki.h> cuenta con diferentes funciones relativamente sencillas para programar el robot para la realización de tareas básicas [7]. Algunas de las funciones más relevantes son:

- *sparki.moveForward(int cms)* -> El robot avanza hacia adelante los centímetros introducidos por el usuario con la velocidad de sus ruedas al máximo.

- *sparki.moveBackward(int cms)* -> El robot se mueve hacia detrás los centímetros introducidos por el usuario con la velocidad de sus ruedas al máximo.
- *sparki.moveRight(int grados)* o *sparki.moveLeft(int grados)* -> El robot rota hacia la derecha o hacia la izquierda un número de grados introducidos por el usuario, rotando ambas ruedas a máxima velocidad pero teniendo cada una un sentido distinto.
- *sparki.moveStop()* -> El robot se para.

Si bien el uso de estas funciones es relativamente simple, éstas no son adecuadas para la realización del proyecto pues se requieren distintas velocidades en cada una de las ruedas del robot para realizar giros de distinta abertura. Así pues, existen otras funciones más complejas que controlan el movimiento de cada uno de los motores paso a paso del robot Sparki, es decir, las 2 ruedas y el gripper:

- *sparki.motorRotate(int motor, int direccion, int velocidad, long steps)* ->Esta función permite controlar la rueda deseada (*int motor*), en la dirección deseada (*int dirección*) y a una velocidad deseada (*int velocidad*) comprendida entre 0 y 100, que es el porcentaje de la velocidad máxima de cada rueda. Opcionalmente se puede incluir el número de pasos de un stepper que se precisen para una rueda deseada (*long steps*). Sabiendo que 4096 pasos, conforman una vuelta completa es posible hacer que robot recorra una distancia deseada escribiendo los pasos necesarios. El número de pasos no se incluye en la realización del presente proyecto dado que no se pueden recibir el número de éstos en un momento preciso al carecer de encoders. Un ejemplo de esta función se muestra a continuación:

```
sparki.motorRotate(MOTOR_RIGHT, DIR_CW, 100, 8192);
sparki.motorRotate(MOTOR_LEFT, DIR_CCW, 50, 4096);
while(sparki.areMotorsRunning()){
  delay(1); //Espera de 1 milisegundo
}
sparki.moveStop();
```

Donde "`MOTOR_RIGHT`" y "`MOTOR_LEFT`" corresponden a las ruedas derecha e izquierda respectivamente. "`DIR_CW`" (*direction clockwise*) y "`DIR_CCW`" (*direction counterclockwise*) definen el sentido al que girarán las ruedas siendo en el sentido de las agujas del reloj y en sentido contrario respectivamente. Los números 100 y 50, corresponden a las velocidades de cada rueda, siendo la primera al 100% de su máxima y la segunda al 50%. Finalmente, los números 8192 y 4096 son el número de pasos que cada motor tiene que dar, si tenemos en cuenta el número de pasos por vuelta, que es 4096, la rueda derecha dará 2 vueltas mientras que la izquierda solo una. Con este fragmento de código, el robot se desplazaría formando una fracción del recorrido de una circunferencia en el sentido contrario de las agujas del reloj.

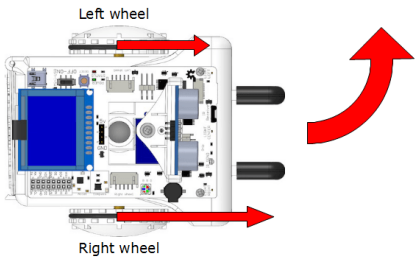

Ilustración 7: Movimiento del Robot Sparki

Para que ambos motores completen el recorrido programado, se precisa el uso de la función "*sparki.areMotorsRunning()"* que devuelve un TRUE si uno o ambos motores del robot están en funcionamiento asegurando así que cada rueda realizará el recorrido programado en la función *sparki.motorRotate()* .

4 Conectividad

Existen diversas formas de comunicación entre un robot de bajo coste y un ordenador. Los diversos modelos de robots que hay en el mercado, ofrecen desde comunicación por puerto serie USB hasta Bluetooth o 2.4G Wireless. Sin embargo, puesto que el robot elegido para el presente proyecto es el Sparki, y dado que se requiere que esta comunicación sea inalámbrica, ésta se realizará mediante bluetooth, pues es el único medio de comunicación inalámbrico que ofrece el robot y se considera suficiente para satisfacer los objetivos del presente proyecto.

La comunicación Bluetooth del robot con el ordenador es necesaria ya que se realizará el envío de comandos y recepción de lecturas de sensores a tiempo real. La conectividad del robot se realiza través del módulo *HC-06*, que irá conectado en las patillas TX, RX, GND y VCC del robot [8].

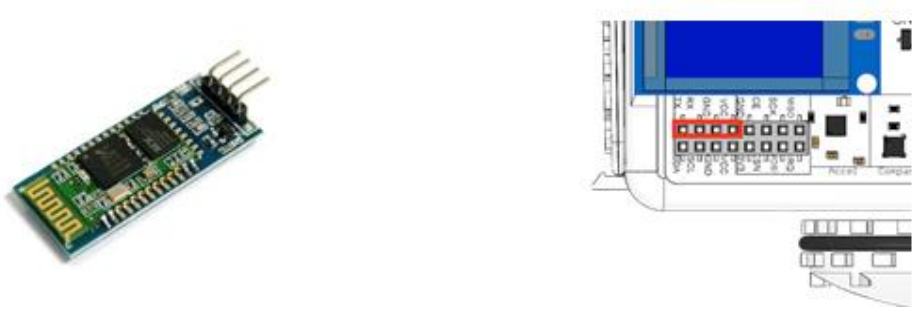

Ilustración 8: Módulo HC-06 y dónde colocarlo

La función de cada pin se detalla a continuación:

- **TX**: Pin que transmite la información procedente del robot. Como es el módulo Bluetooth quien recibe esa información, el pin RX del módulo HC-06 irá conectada al pin TX.
- **RX**: Pin que recibe la información procedente del módulo Bluetooth. Por esta razón, el pin TX del módulo HC-06 irá conectada al pin RX del robot.
- **GND**: Es el pin de tierra.
- **VCC**: Es el pin de suministro de voltaje. El módulo Bluetooth HC-06 requiere 3.3V para su correcto funcionamiento.

4.1 Conexión del robot Sparki con el ordenador a través del módulo Bluetooth.

La librería *<sparki.h>* cuenta con una serie de comandos para poder realizar la comunicación tanto serial por cable como por Bluetooth. En primer lugar, se deberá inicializar la comunicación a través del comando *serial1.begin(9600)*, siendo 9600 la velocidad de comunicación del módulo Bluetooth con el ordenador (9600 bits por segundo), además de ser la velocidad especificada por el fabricante.

Se cuenta con una serie de comandos para enviar y recibir información:

- *serial1.print()* ->Comando para escribir texto tipo *char* en el terminal Bluetooth.

- *serial1.read()*->Comando para leer texto tipo *char* procedente del terminal Bluetooth.
- *serial.available()*->Comando para comprobar la disponibilidad del terminal Bluetooth.

Para poder comprobar que la comunicación se realiza de forma exitosa, se utilizará como terminal Bluetooth el programa ***Termite*** para realizar la comunicación entre el robot y el ordenador. Posteriormente, la comunicación se realizará a través de Matlab.

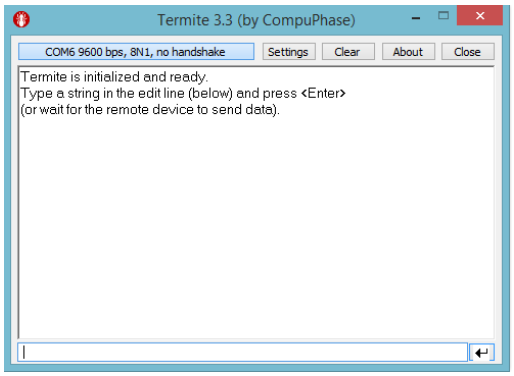

Ilustración 9: Terminal Bluetooth Termite 3.3

4.2 Envío de órdenes a través del terminal bluetooth

Dado que todo el proceso del control y programación del robot se realiza a través de Matlab, la información a enviar a través del terminal bluetooth consistirá únicamente en una serie de números que afectarán a diversos parámetros del robot como la velocidad de cada rueda o el ángulo de giro, entre otros.

Un problema asociado con el envío de órdenes es que el terminal Bluetooth solo lee y recibe texto tipo *char* y los valores a leer por el robot son tipo *int* (están asociados a las funciones por defecto del robot). Por lo cual, será necesario realizar una conversión de variables tipo *char* a *int*. Para ello se realiza un programa en el propio entorno Arduino que recibe los números procedentes del terminal y los transforma en variables tipo *int* para su posterior introducción en las funciones que controlan la velocidad de cada rueda.

4.3 Comunicación Bluetooth a través de Matlab.

Para la realización de la comunicación Bluetooth a través de Matlab es necesario identificar en primer lugar el módulo bluetooth a utilizar. Para ello se utiliza la función *Bluetooth('RemoteName', Channel)* para construir un objeto bluetooth, donde *RemoteName* se refiere al nombre del dispositivo bluetooth a conectar, en este caso 'HC_06'; mientras que en *Channel* se indicará el canal al que se desea conectar el dispositivo, que por defecto será fijado en 1.

Una vez creada la variable, será necesario conectarse a ella a través de la función *fopen()* y si la conexión se realiza con éxito, se podrá proceder a el envío de órdenes. Para escribir información en el terminal, se utiliza la función *fprintf(obj,'formato','cmd','modo')* Un ejemplo de código, donde se envían 2 velocidades, se muestra a continuación:

```
>> A=Bluetooth('HC-06',1)
>>fopen(A)
>> fprintf(A,['<50 100>'],'sync')
>> fprintf(A, ['<0 0>'],'sync')
```

En la función *fprintf* se distinguen distintos parámetros. El primer parámetro, *A,* es el objeto conectado al puerto bluetooth, en este caso el terminal HC-06. A continuación, se escribe el texto (en formato *string*) que se desea enviar, en este caso se escribe *['<50 100 >']* refiriéndose a los 2 valores de las velocidades. Por último, *sync* indica que el texto se escribirá de forma síncrona. Una vez se ha realizado la comunicación con el robot, si se desea terminar con ésta, se utilizará el comando *fclose(A)* que desconectará el dispositivo bluetooth.

Dependiendo del algoritmo a utilizar (*capítulo 8.- Algoritmos*) se le asignará a cada uno una serie de valores, ya sea velocidad, distancia al objetivo, así como número de algoritmo. De esta forma los datos a enviar diferirán en función del algoritmo implementado.

5 Sistema sensorial externo

5.1 Kinect

Para asegurar un óptimo funcionamiento del programa, es necesario conocer en todo momento la posición y orientación del robot para así poder calcular de forma precisa un controlador que permita guiar al robot hacia un punto deseado. Para ello, dado que el robot carece de encoders que permitan conocer parámetros como la distancia recorrida por cada rueda, se utilizará una cámara, que a través de Matlab podrá enviar los parámetros necesarios al ordenador.

La cámara elegida para el proyecto es una Kinect para Xbox One, pues su cámara RGB cuenta con una resolución de 1920x1080 píxeles, con lo que la precisión de medida será mayor. Además, entre otros aspectos, incluye un sensor de profundidad, *array* de micrófonos y sensor de infrarrojos (emisor y receptor) [9].

Al contar con sensor de profundidad se da la oportunidad de optimizar el proceso de medición de distancias en caso de que utilizando únicamente la cámara RGB se obtenga un error demasiado grande.

Para la realización del proyecto se utilizará únicamente la cámara RGB por haber mostrado una correcta precisión y se pospondrá el uso del sensor de profundidad para futuras aplicaciones, tales como la detección de obstáculos o la obtención de un mapa de una zona determinada.

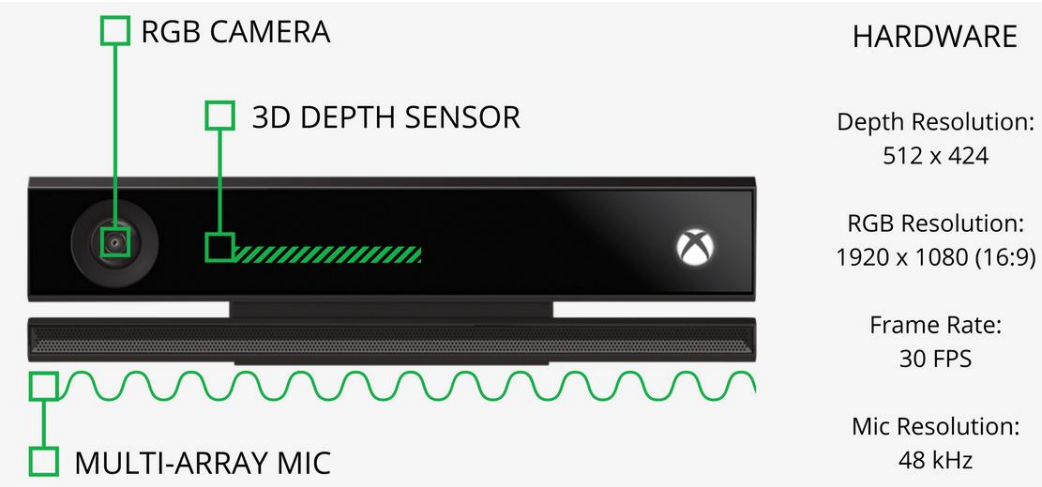

Ilustración 10: Características principales cámara Kinect

5.2 Instalación del sensor Kinect para Matlab

La cámara Kinect se comunicará con el robot a través de Matlab por lo que es necesaria tanto la instalación de los drivers necesarios como de la Toolbox "*Computer Vision System"* de *Mathworks*.

Una vez realizada la instalación de todos los drivers y Toolbox, se procede a la instalación de OpenCV y los paquetes de ArUco para su posterior uso en Matlab, para así poder realizar la lectura de las marcas para una correcta medición de distancias. [10]

Tras la instalación de todos paquetes mencionados anteriormente, se ha de redirigir la ejecución de los programas en Matlab al directorio de instalación de OpenCV ejecutando los siguientes comandos:

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

De forma que se realiza una compilación de todos los archivos situados en el presente directorio posibilitando así el uso de las librerías de ArUco.

5.3 Marcas ArUco

ArUco es una librería para detectar unas marcas determinadas en una imagen. A través de esta detección de marcas, es posible determinar la posición de la cámara con respecto a las mismas. Para la realización del proyecto se utilizarán marcas ArUco, incluidas en esta librería.

Una marca ArUco es un cuadrado compuesto por un borde negro y una matriz interior binaria que determina su identificación (ID). El borde negro facilita la detección de la marca en la imagen y la codificación binaria permite su identificación, así como la aplicación de detección de errores y técnicas de corrección. El tamaño de la marca determina el tamaño de su matriz interna. Por ejemplo, una marca de tamaño 4x4 se compone de 16 bits [11].

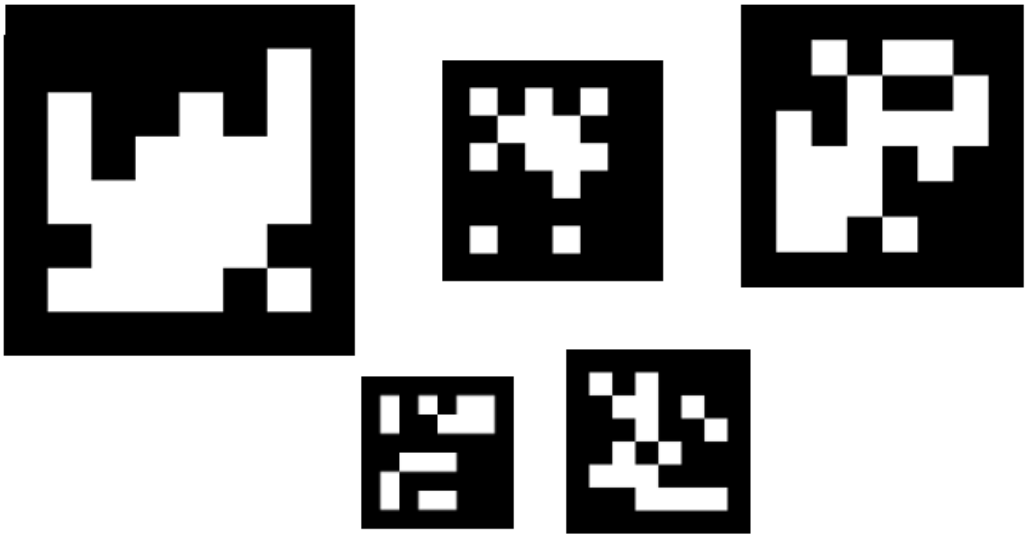

Ilustración 11: Marcas ArUco

Se ha de tener en cuenta que una marca puede aparecer rotada en el entorno de trabajo, sin embargo, el proceso de detección debe poder determinar su rotación original, de modo que

cada esquina se identifique inequívocamente. Esto también se realiza en base a su codificación binaria.

Cada tipo de marca tiene su propio diccionario, que sería el conjunto de marcas utilizados para una aplicación específica. Además, un diccionario tiene una serie de especificaciones que determinan las propiedades de una marca:

- *Dictionary size*. Es el número de marcas que componen el diccionario.
- *Marker size.* El tamaño de estas marcas (el número de bits).

Para la realización del proyecto se utilizarán diccionarios predefinidos de la librería ArUco.

5.4 Detección de marcas

El primer paso consiste en obtener el umbral de la imagen. Un ejemplo de imagen umbralizada se muestra a continuación:

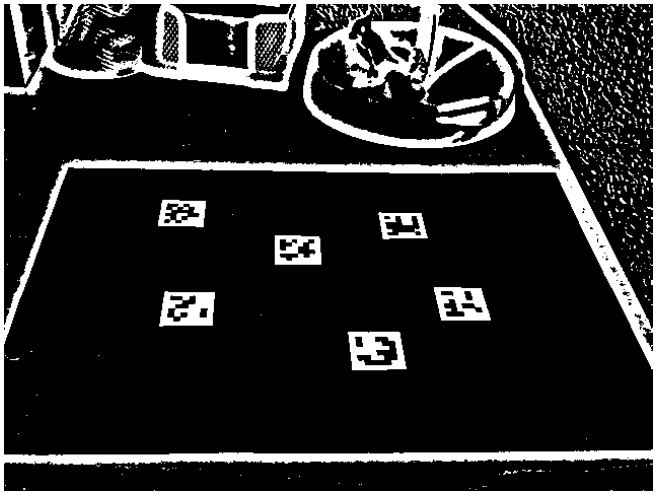

Ilustración 12: Imagen Umbralizada

El siguiente paso es detectar los contornos. Dado que no todos los contornos detectados son marcas ArUco, se debe realizar un filtrado para ir eliminando los no deseados, es decir, los que no tienen la forma de una marca.

Una vez detectados, el siguiente paso consiste en saber si los contornos detectados son marcas o no. Para ello se analizan los bits de cada candidato elegido.

Antes de analizar el código de éstos, es necesario extraer los bits de los mismos. Para ello, la distorsión producida por la perspectiva en la que se encuentra la marca se elimina y la imagen obtenida se umbraliza para separar los bits blancos y negros (1 y 0 respectivamente). Posteriormente la imagen se divide en una cuadrícula con el mismo número de celdas que bits tiene la marca según el diccionario especificado. En cada celda, se realiza un recuento del número de píxeles blancos y negros para decidir el bit que se asignará a cada una de las celdas.

Por último, se ha de comprobar si el código extraído pertenece al diccionario predefinido en un primer momento.

A continuación, se muestran 2 imágenes, la primera muestra la imagen obtenida tras eliminar la distorsión de la marca producida por su perspectiva, y la segunda la marca ya dividida en celdas para determinar sus bits.

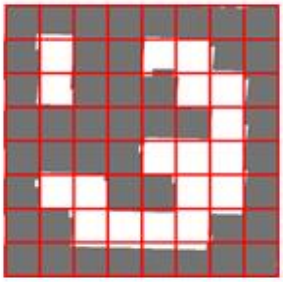

Ilustración 13: Mapa de bits de Marcas ArUco

5.5 Calibración de la cámara

La calibración de la cámara [12] es necesaria para la realización del proyecto, pues ésta será utilizada para estimar la posición y orientación real de las marcas y, por tanto, del robot respecto a éstas. Se requieren una serie de parámetros para calibrarla, que a su vez se clasifican en *intrínsecos* y *extrínsecos*.

Los parámetros *intrínsecos* son:

- f_x, f_y: la distancia focal de la lente de la cámara en ambos ejes. Normalmente expresados en píxeles.
- c_x, c_y: centro óptico de un sensor, expresado también en píxeles.
- k_1, k_2, p_1, p_2: coeficientes de distorsión.

En una cámara ideal, un punto 3D (X, Y, Z) en el espacio se proyectaría en el píxel:

$$x\left(X\frac{f_x}{Z}\right)+c_x\,;y=\left(Y\frac{f_y}{Z}\right)+c_y$$

Con estos parámetros se consigue la localización 3D de un punto respecto al sistema de referencia de la cámara. Como se requiere la proyección de un punto 3D respecto a un sistema de referencia arbitrario, se utilizan parámetros *extrínsecos*. Éstos son básicamente:

- Rotaciones 3D, representadas como $Rvecs = \{R_x, R_y, R_z\}$ y expresadas en radianes.
- Traslaciones 3D, representadas como $Tvecs = \{T_x, T_y, T_z\}$ y expresadas en metros.

Estos 2 parámetros son requeridos para poder realizar una traslación del sistema de referencia de la cámara a uno arbitrario. Los elementos de rotación se expresan con la fórmula de Rodrigues para poder obtener la matriz 3x3 de rotación.

Obtención de la matriz de rotación a partir de la fórmula de Rodrigues [13].

Una vez obtenidos los valores de *rvecs{rx,ry,rz},* se procede a la obtención de la matriz de rotación. Los valores de rvecs obtenidos son en realidad las rotaciones producidas respecto a la norma del vector, por lo que, para calcular la matriz, se requiere primero obtener los valores de rotación, para ello se calcula la norma del vector rvecs y posteriormente se divide este vector por su norma obteniendo así los ángulos:

$$r = \{rx, ry, rz\} \qquad (1)$$

$$\theta = norm(r) \tag{2}$$

$$r = \frac{r}{\theta} = \left\{\frac{rx}{\theta}, \frac{ry}{\theta}, \frac{rz}{\theta}\right\} \tag{3}$$

Posteriormente se aplicará la fórmula de rotación de Rodrigues:

$$R = I + \sin(\theta) * A + (1 - \cos(\theta)) * A^2 \tag{4}$$

Donde R es la matriz de rotación 3x3 deseada, *I* es una matriz identidad 3x3, y A es la matriz 3x3 denotada como:

$$A = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \tag{5}$$

Para representar la posición de un punto 3D respecto a un sistema de referencia arbitrario, se utiliza una matriz homogénea 4x4 que contiene tanto la matriz de rotación como el vector de traslación del punto.

$$T_{homogénea} = \begin{pmatrix} Rot_{matrix} & Trasl_{vec} \\ 0 & 1 \end{pmatrix} \tag{6}$$

Para la realización de la calibración de la cámara, es necesario imprimir previamente una tabla chArUco como la que se muestra a continuación:

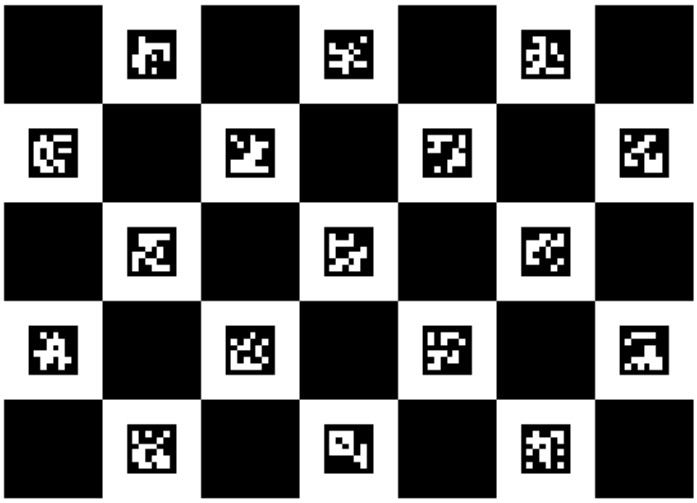

Ilustración 14: Tabla de calibración ChArUco

Ésta se puede obtener ejecutando en Matlab el programa aruco*_create_board_charuco.m* especificando entre otras propiedades, el número cuadros que queremos tanto en el eje X como en Y, el número de píxeles de las marcas y cuadros negros y el diccionario del cual se generarán las marcas para la tabla.

Una vez generada la tabla, se ha de imprimir sin variar su tamaño, pues la cámara tomará el tamaño de los píxeles introducidos anteriormente como referencia para la calibración. Una vez impresa, se ha de pegar en una pared lisa y ejecutar el programa *aruco_calibrate_camera_charuco.m*. En éste se utilizará la Kinect para tomar fotografías de la tabla chArUco desde distintas posiciones y ángulos. Se recomienda al menos tomar de 4 a 6 fotografías, sin embargo, para la realización del proyecto se tomaron más de 40 con la finalidad de mejorar la precisión a la hora de la toma de medidas. Para la toma de fotografías no es

necesario que aparezca la tabla en su totalidad, pues es posible tomar las fotos tapando distintas marcas sin que afecte a la posterior calibración.

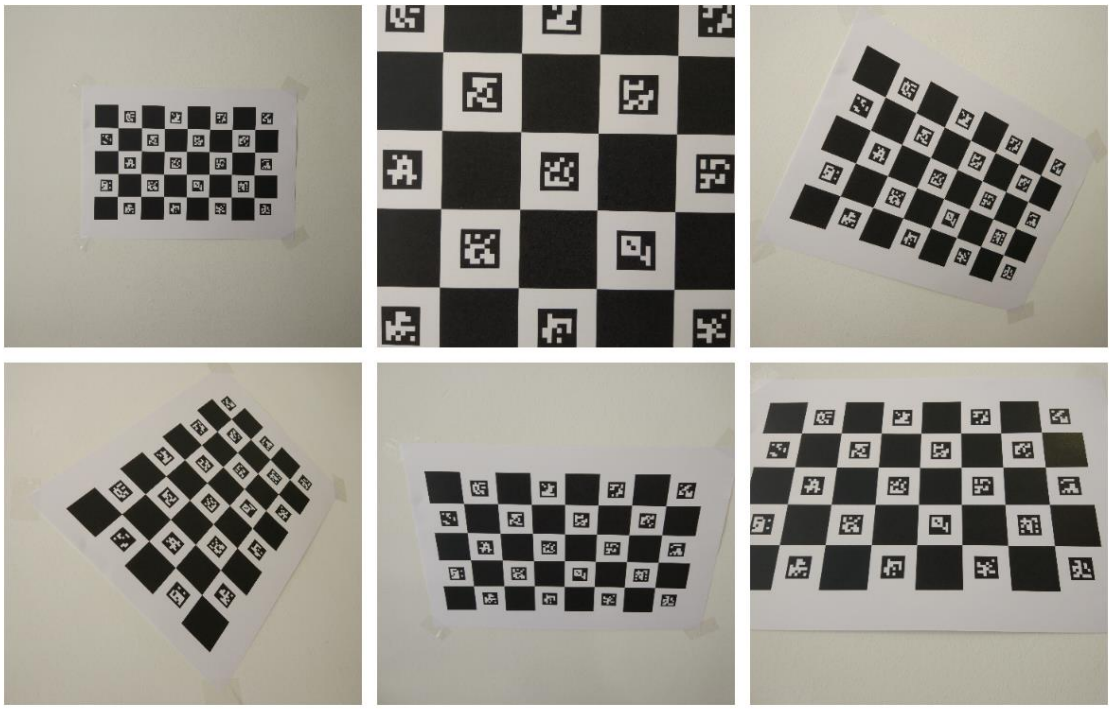

Ilustración 15: Ejemplo de toma de imágenes para calibración

Una vez realizada la toma de fotos, el programa genera automáticamente un fichero llamado *camera_parameters.mat* que incluye, entre otros, los coeficientes de distorsión y la matriz de rotación de la cámara, la cual será la matriz de referencia a la hora de estimar la posición de las marcas detectadas. Este fichero se utilizará durante la ejecución del programa para obtener las referencias de las marcas.

5.6 Funciones básicas OpenCV.

Antes de explicar el programa principal, es necesario explicar alguna de las funciones más relevantes que se utilizarán a lo largo del proyecto. Estas funciones permitirán entre otras cosas, detectar las marcas y estimar su posición gracias a las librerías instaladas.

- ***cv.detectMarkers***, esta función se encarga de detectar todas las marcas ArUco presentes en la imagen tomada por la cámara. Los parámetros de la función son, entre otros, la imagen a procesar y el diccionario de las marcas.
- ***cv.drawDetectedMarkers,*** la finalidad principal de esta función es dibujar sobre la marca ArUco detectada un recuadro para que el usuario pueda apreciar que esa marca ha sido detectada.
- ***cv.drawAxis,*** esta función dibuja los ejes X (azul), Y (verde), Z (rojo) sobre las marcas detectadas, estando siempre el eje Z perpendicular respecto a la marca.
- ***cv.estimatePoseSingleMarkers***, una vez detectados las marcas, esta función calcula la posición relativa de cada una de las marcas respecto a la referencia de la cámara. Los valores que devuelve son los 2 vectores *rvecs*$\{r_x, r_y, r_z\}$ (vector de rotación) y *tvecs*$\{t_x, t_y, t_z\}$ (vector de traslación).

A continuación, se muestra una imagen en la que se disponen 6 marcas ArUco que han sido detectadas por la cámara y posteriormente se les ha aplicado la ejecución de las funciones mencionadas anteriormente.

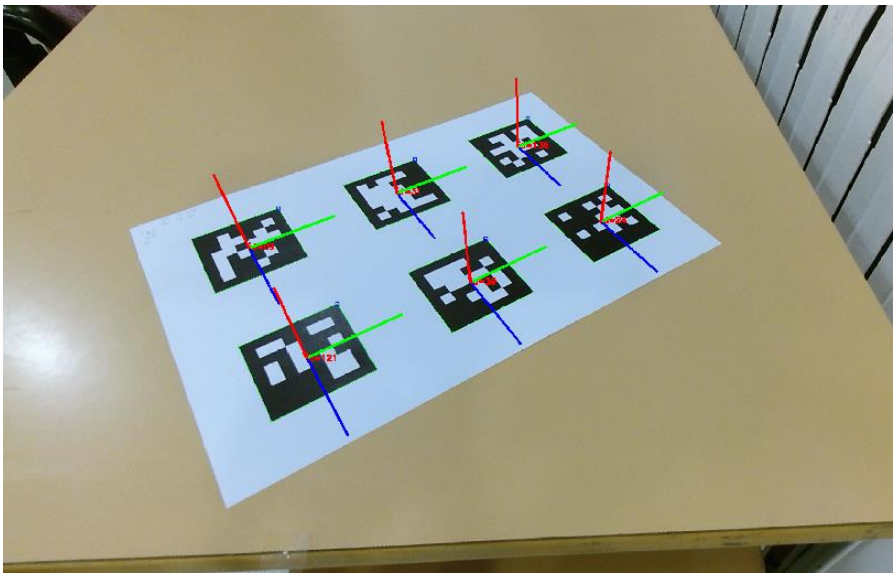

Ilustración 16: Posición de los ejes sobre las Marcas ArUco

5.7 Posición de marca ArUco sobre el robot Sparki

Para la detección de la posición y orientación del robot se debe colocar una marca ArUco encima de éste, para facilitar la colocación de la marca, se realizó el diseño de una pieza mediante el programa de diseño 3D *PTC Creo Parametric* que posteriormente se imprimiría en una impresora 3D del Laboratorio de Tecnologías Industriales de la escuela:

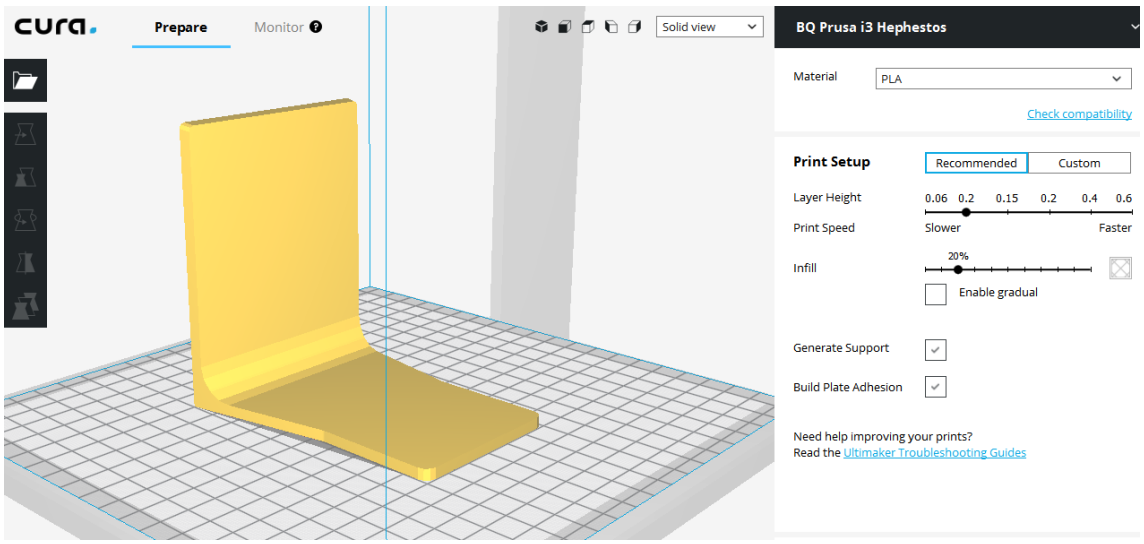

Ilustración 17: Visualización de pieza a través del programa CURA

Una vez impresa la pieza, ésta se adherirá a la parte trasera del robot con cinta adhesiva, pues éste es de uso temporal y no conviene hacer ninguna modificación a la carcasa del mismo. La disposición final del robot, con la pieza ya impresa con la marca en su superficie queda así:

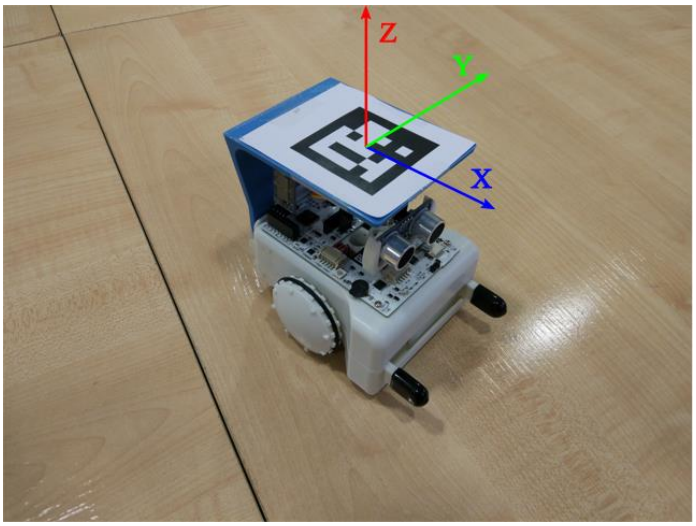

Ilustración 18: Montaje de la pieza sobre el Robot Sparki

La marca ArUco se encontrará a una altura aproximada de unos 11.5 centímetros. Sin embargo, esto afectará levemente a la posición y orientación del robot pues solo se requiere saber su posición en el plano XY.

6 Determinación de posiciones relativas entre 2 marcas (marca de referencia y robot)

A continuación, se muestra un ejemplo de cómo determinar la posición relativa, es decir, posición y orientación, entre 2 marcas conocidas (ID conocido), siendo una la marca de referencia y la otra la colocada sobre el robot. Partiendo del esquema:

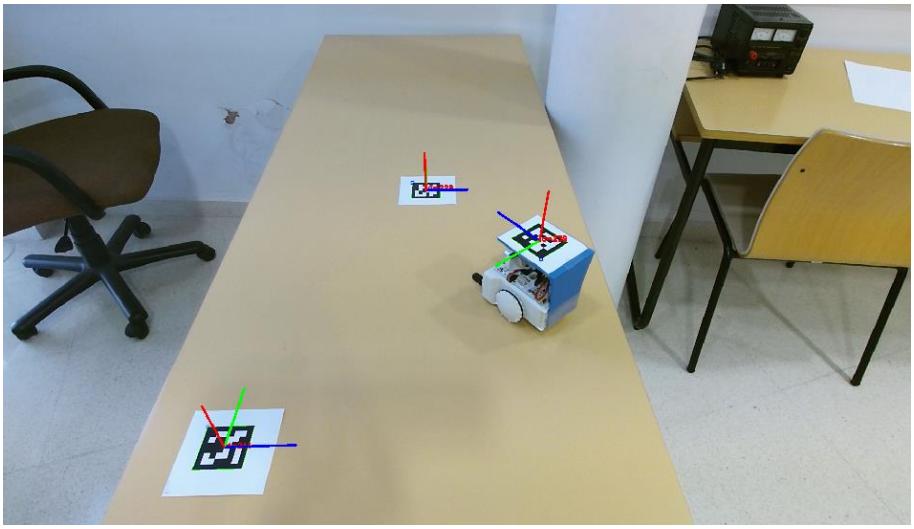

Ilustración 19: Disposición inicial de ejemplo

Tras ejecutar el programa, se obtienen 3 vectores de *rvecs* y *tvecs* de las 3 marcas que aparecen en la imagen. El primer paso es elegir qué marca se tomará como referencia, en este caso se elegirá la marca situada en la parte inferior izquierda de la imagen. Es necesario conocer el id de la marca elegida para hallar sus vectores *rvecs* y *tvecs*. Para ello se hará uso de la función *find(ids==id_marca_ref)* obteniendo así la posición de los vectores *rvecs* y *tvecs* pertenecientes a la marca elegida por el usuario, ya que, al haber 3 marcas, éstos vectores pueden estar en las posiciones 1, 2 o 3.

Una vez conocidos los vectores *rvecs* y *tvecs* de la marca elegida, (cabe recordar que son respecto a la cámara) para saber la posición de ésta con respecto a la marca perteneciente al robot se necesita previamente obtener la matriz de transformación homogénea de cada una de ellas, es decir ${}^{cámara}T_{referencia}$ y ${}^{cámara}T_{robot}$. Para ello, una vez conocidos los vectores *rvecs* y *tvecs* de cada una de ellas, se crea una matriz 4x4 como la que se muestra a continuación:

$$ {}^{cámara}T_{referencia} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (7) $$

Posteriormente, se añade la matriz de rotación utilizando la función *cv.Rodrigues(rvecs{posicion_referencia})* que forma una matriz 3x3 en función de los ángulos pertenecientes al vector *rvecs* a través de la fórmula de Rodrigues. Y finalmente, se añade en la cuarta columna, el vector $tvecs\{t_x, t_y, t_z\}$ que determina la posición de la marca de referencia respecto a la cámara (en metros). Por tanto, la matriz de transformación homogénea que representa la posición y orientación de la marca de referencia respecto a la cámara queda así:

$$ {}^{cámara}T_{referencia} = \begin{pmatrix} 0.999 & -0.038 & 0.012 & -0.2727 \\ -0.0157 & -0.6516 & -0.7584 & 0.2028 \\ 0.0368 & 0.7576 & -0.651 & 0.5916 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (8) $$

Se debe repetir el proceso con la marca de robot, obteniendo así la matriz de transformación homogénea de la posición y orientación del robot con respecto a la cámara:

$$ {}^{cámara}T_{robot} = \begin{pmatrix} -0.662 & -0.7476 & 0.046 & 0.096 \\ -0.5307 & 0.425 & -0.733 & -0.034 \\ 0.528 & -0.510 & -0.678 & 0.696 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (9) $$

Y finalmente, para obtener la posición relativa del robot con respecto a la marca de referencia, basta con multiplicar la inversa de la matriz ${}^{cámara}T_{referencia}$ por la matriz ${}^{cámara}T_{robot}$:

$$ {}^{referencia}T_{robot} = ({}^{cámara}T_{referencia})^{-1} * {}^{cámara}T_{robot} = \begin{pmatrix} -0.634 & -0.772 & 0.032 & 0.376 \\ 0.771 & -0.635 & -0.038 & 0.220 \\ 0.050 & 0.001 & 0.998 & 0.116 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (10) $$

De donde observando la cuarta columna, se puede obtener los valores de la traslación en x,y,z del robot respecto a la marca de referencia.

$$ Traslación({}^{referencia}T_{robot}) = \{0.376, 0.220, 0.116\} $$

Para obtener los ángulos de rotación rx,ry y rz a partir de la matriz de rotación 3x3, deberemos realizar la inversa del método de Rodrigues [13].

Para ello, se debe obtener primero el ángulo, y posteriormente la matriz n sobre la que se produce esa rotación.

Para ello, primero se aplica la siguiente fórmula:

$$\theta = \arccos\left(\frac{trace(R) - 1}{2}\right) \tag{11}$$

Donde R es la matriz de rotación y $trace(R)$ es la suma de los valores en diagonal de la matriz R. En este caso, $\theta = 2.25\, rad$.

Una vez obtenido el ángulo θ, se obtiene la matriz n mediante la fórmula:

$$n = \frac{1}{2\sin(\theta)}(R - R^T) \tag{12}$$

Obteniendo una matriz n cuyos valores se definen tal que:

$$n = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} = \begin{bmatrix} 0 & -0.999 & -0.011 \\ 0.999 & 0 & -0.0254 \\ 0.011 & 0.0254 & 0 \end{bmatrix} \tag{13}$$

Donde r_x, r_y, r_z son los elementos del vector de rotación, que multiplicados por el valor de θ, nos definen el valor de los elementos del vector que define la matriz de rotación, es decir los valores de los parámetros de *rvecs*$\{r_x, r_y, r_z\}$. Así pues, la información relativa a la rotación del robot respeto a la marca de referencia ($^{referencia}T_{robot}$) se representa mediante un vector rvecs:

$$rvecs = \{0.0573, 0.0252, 2.258\}\, rad \tag{14}$$

Puesto que el robot es considerando que se mueve en una superficie lisa, en ausencia de ruido, se debería obtener un vector de valor (0,0,1) donde todos los valores estarían multiplicados por el ángulo θ . En la práctica se obtienen valores despreciables en las componentes rx y ry. Por tanto, en rz contiene el valor del ángulo de rotación θ en el eje Z de 2.258 radianes o lo que es lo mismo, 129.37º.

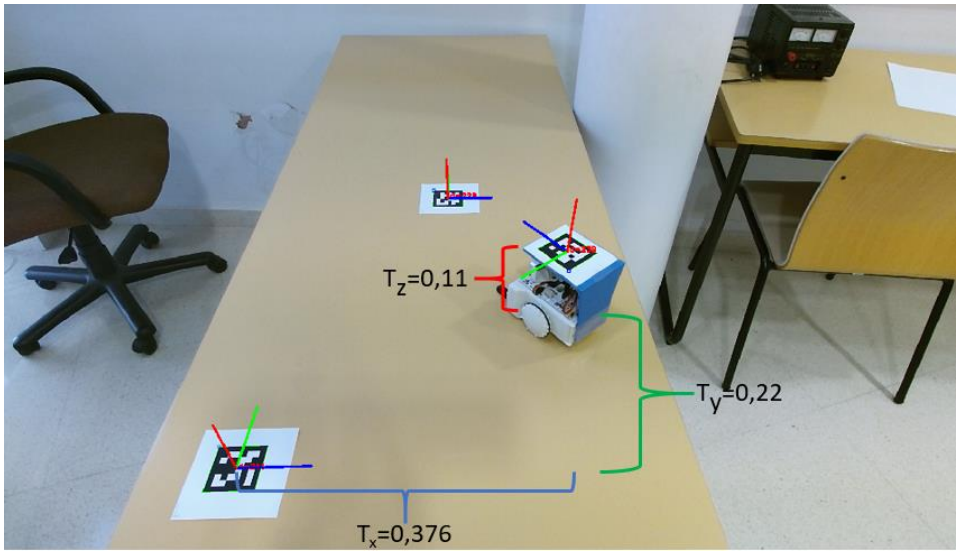

Ilustración 20: Disposición inicial con las medidas realizadas

7 Control del robot

Para el modelado del controlador del robot [14] se contarán con ciertas hipótesis como, por ejemplo, se asume que el robot se desplazará por una superficie plana, que no se ejercerá ninguna fuerza externa sobre éste, y que además los ejes de las ruedas se situarán paralelamente al suelo.

Es necesario conocer las dimensiones del robot para la realización del cálculo del controlador. El robot Sparki cuenta con los siguientes parámetros:

- Radio de las ruedas = 0.025 m.
- Distancia entre ruedas = 0.0851 m
- Máxima velocidad lineal del robot (la deseada) =0.02781 m/s.

A continuación, se muestra un plano de la disposición del robot respecto a un punto deseado (estrella):

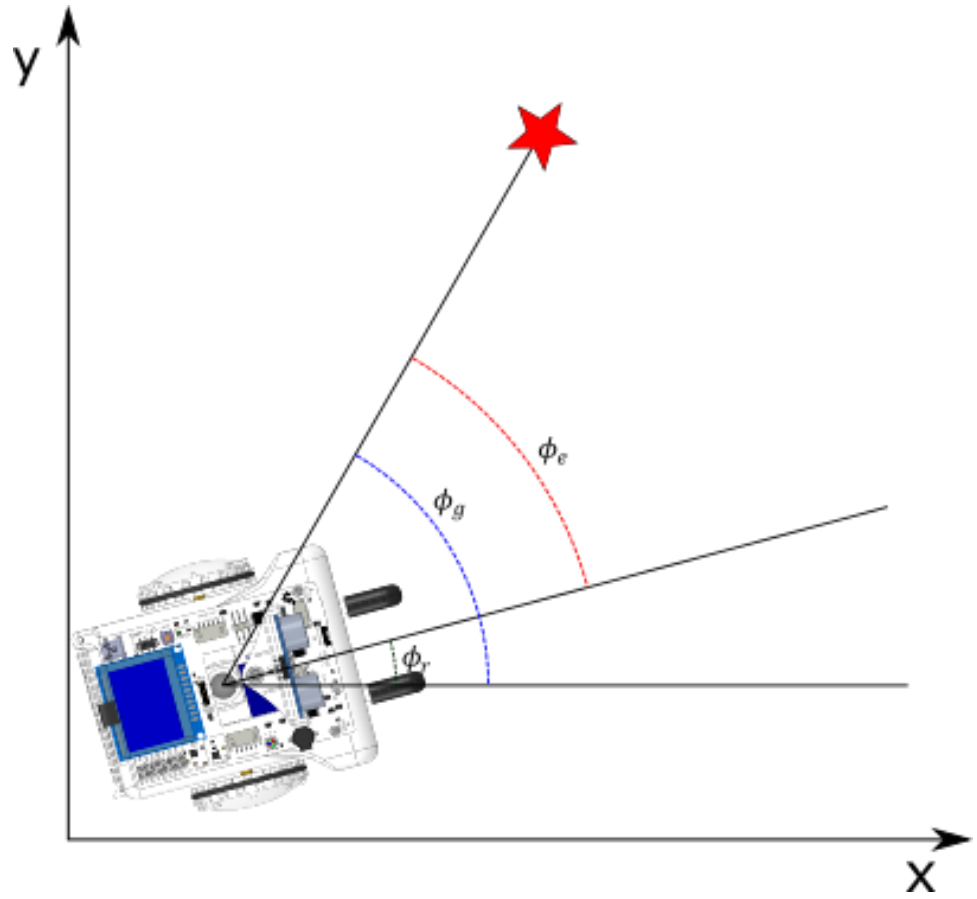

Ilustración 21: Posición y orientación del Robot Sparki en el plano XY

Para definir el sistema de control de este sistema, primero veremos cómo la salida (v_r, v_l) correspondiente a las velocidades de las ruedas derecha e izquierda, afecta a las entradas del sistema (X, Y, ϕ), correspondientes a la posición y orientación. Para que el robot se desplace en línea recta, es necesario que ambas velocidades de sus ruedas sean iguales, y que éstas sean proporcionales al radio de las ruedas.

$$v = R\frac{\omega_r + \omega_l}{2} \tag{15}$$

Donde ***R*** se refiere al radio de las ruedas, ω_r y ω_l a la velocidad angular de cada una de las ruedas del robot y ***v*** a la velocidad lineal del robot.

Para que el robot pueda realizar un movimiento de rotación sobre su eje, la velocidad de las ruedas debe ser igual, pero con sentido diferente. Además, la velocidad de las ruedas será proporcional al radio de ellas e inversamente proporcional a la distancia entre éstas. De forma que la ecuación queda:

$$\omega = \frac{R}{L}(\omega_r - \omega_l) \tag{16}$$

Donde ω se refiere a la velocidad angular del robot.

Las ecuaciones que definen la dinámica del robot se muestran a continuación:

$$\begin{cases} \dot{x} = v\cos(\phi) \\ \dot{y} = v\sin(\phi) \\ \dot{\phi} = \omega \end{cases} \tag{17}$$

A continuación, para definir el modelo del robot, se reemplazan las ecuaciones (15) y (16) en (17) obteniendo el siguiente modelo:

$$\begin{cases} \dot{x} = \frac{R}{2}(\omega_r + \omega_l)\cos(\phi) \\ \dot{y} = \frac{R}{2}(\omega_r + \omega_l)\sin(\phi) \\ \dot{\phi} = \frac{R}{L}(\omega_r - \omega_l) \end{cases} \tag{18}$$

De forma que si comparamos con las ecuaciones del modelo anterior y las aplicamos a (17) podemos deducir que:

$$v = \frac{R}{2}(\omega_r + \omega_l) \tag{15}$$

$$\omega = \frac{R}{L}(\omega_r - \omega_l) \tag{16}$$

Por lo que es posible determinar la velocidad angular de cada rueda aplicando los cálculos correspondientes a las ecuaciones (15) y (16), obteniendo así:

$$\omega_r = \frac{2v + \omega L}{2R} \tag{19}$$

$$\omega_l = \frac{2v - \omega L}{2R} \tag{20}$$

Como la velocidad del robot se medirá en porcentaje de su velocidad lineal, es necesario convertir la velocidad angular de cada rueda en velocidad lineal. Recordamos que:

$$v_r = \omega_r R \tag{21}$$

$$v_l = \omega_l R \tag{22}$$

Por tanto, aplicando las formulas (21) y (22) a las ecuaciones (19) y (20), obtenemos:

$$v_r = v + \frac{\omega L}{2} \tag{23}$$

$$v_l = v - \frac{\omega L}{2} \tag{24}$$

Que serán las fórmulas que nos permitirán calcular la velocidad lineal de cada rueda.

8 Algoritmos

8.1 Método en 2 fases

Éste algoritmo se considera como una primera solución sin tener en cuenta un controlador calculado. En su desarrollo solamente se consideran 2 etapas:

- **Rotación:** En esta primera etapa, se ha de calcular el error de orientación (ecuación (25)), que no es más que la diferencia entre el ángulo proyectado del punto deseado respecto al robot, y el ángulo del robot respecto al sistema de referencia

$$e = (\phi_g - \phi_r) \tag{25}$$

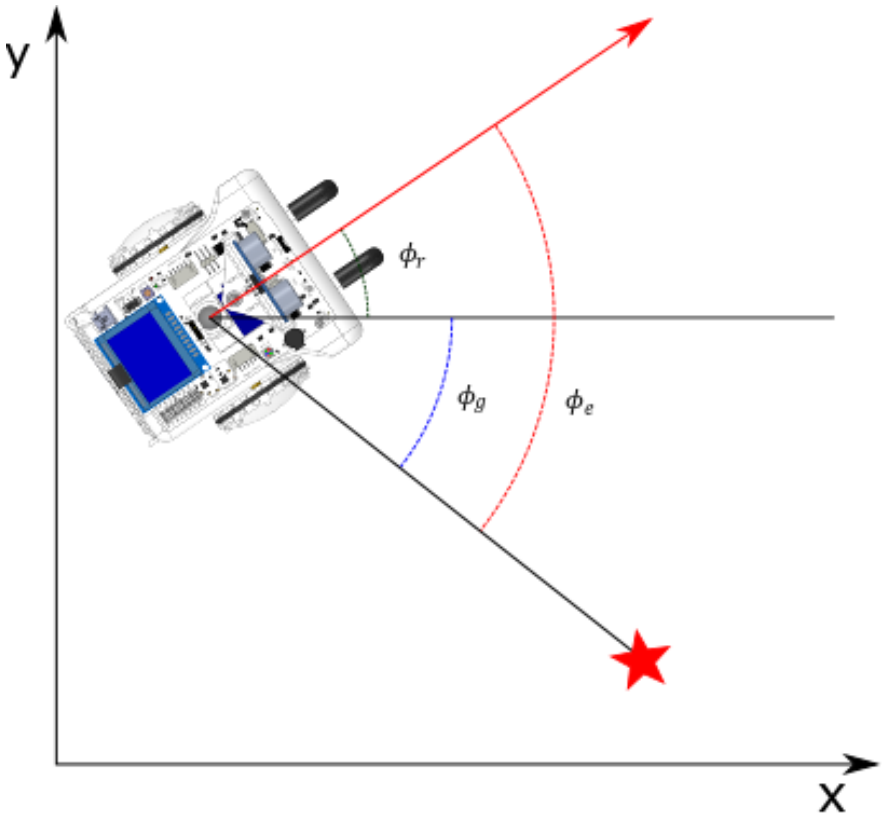

Ilustración 22: Posición y orientación del Robot Sparki en el plano XY

Una vez obtenido el error, el robot deberá corregir ese error mediante el algoritmo *sparki.moveRight(degrees)* en caso de que el ángulo a corregir sea negativo, o *sparki.moveLeft(degrees)*, si el ángulo a corregir es positivo.

- **Traslación:** Una vez corregido el ángulo, se calculará la distancia entre el robot y el punto, y a través del algoritmo *sparki.moveForward(cms),* el robot recorrerá en línea recta los centímetros necesarios para alcanzar dicho punto.

Caso especial

Puesto que para el cálculo de ángulos se utiliza la función *atan2* de Matlab, que comprende los ángulos entre (-π,π), es posible que se produzca una situación como la mostrada a continuación:

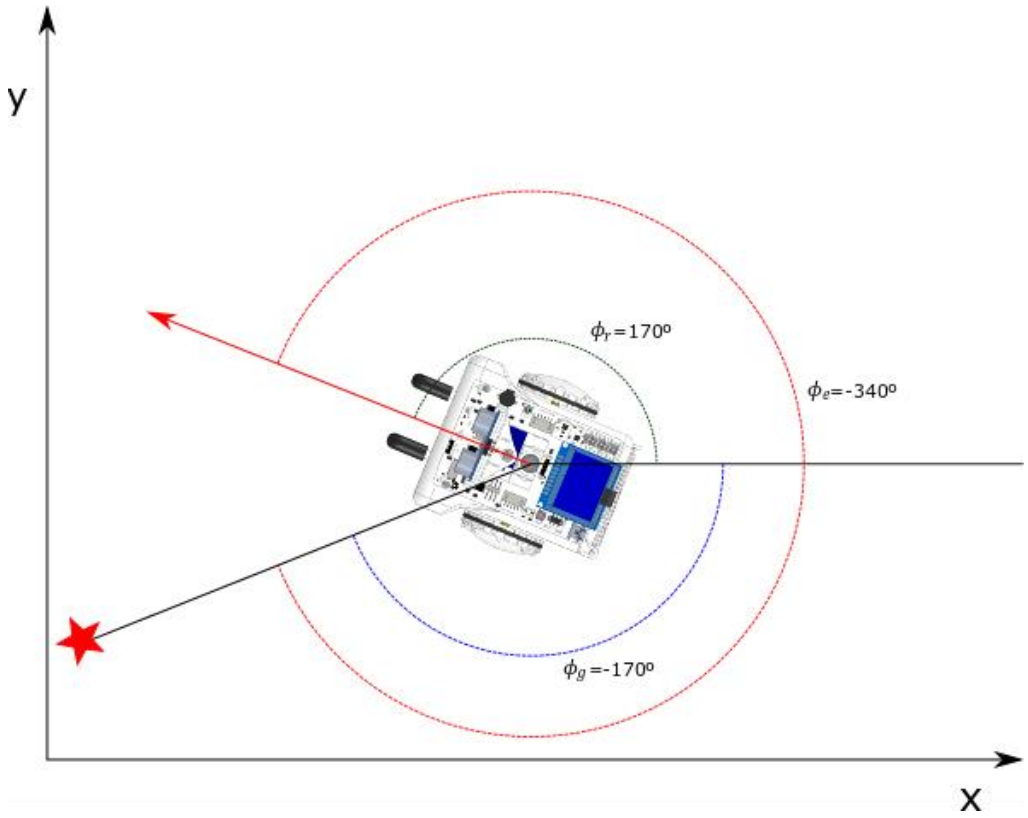

Ilustración 23: Disposición en caso especial

Como se puede observar, el ángulo del punto deseado (ϕ_g) es de -170º mientras que el del robot (ϕ_r) es de 170º, por lo que si aplicamos (25) obtenemos:

$$e = (\phi_g - \phi_r) = -170 - 170 = -340º$$

Por lo que el robot rotará hacia la derecha 340º en lugar de hacer un giro hacia la izquierda de tan solo 20º. Para ello, se ajustará el ángulo de forma que el ángulo de rotación del robot esté entre (-π,π):

```
if(error>pi)
    error = error - 2*pi);
elseif(error<-pi)
        error= error + 2*pi;
end
```

De forma que si el error obtenido es mayor que 180º o menor que -180º, éste se corregirá a fin de obtener un giro de un ángulo menor optimizando el recorrido del robot. En el caso del ejemplo propuesto, el robot rotaría 20 grados a su izquierda.

A continuación, se muestra un diagrama de flujo del programa principal ejecutando el algoritmo en 2 fases:

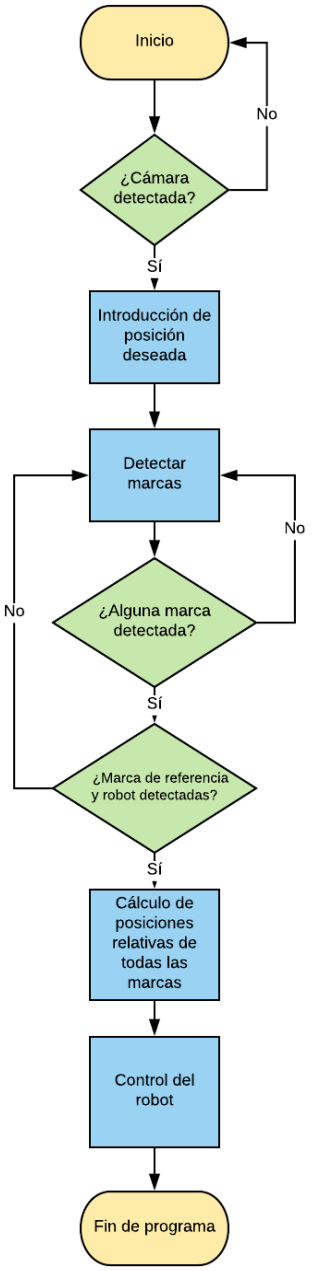

Ilustración 24: Diagrama de flujo del algoritmo en 2 fases

A pesar de la sencillez del algoritmo, éste presenta diversos problemas de precisión, pues una vez conocida la posición inicial del robot, éste rehúsa a recibir nueva información por parte de la cámara por lo que la precisión del mismo dependerá exclusivamente tanto de los motores paso a paso como de la cámara a la hora de medir distancias.

8.2 *GoToGoal*

Dado que la función del robot será alcanzar un punto deseado, se requiere la inclusión de un controlador PID que calcule una acción de control en función del error de ángulo que tendrá respecto al punto deseado. Este error se calcula en base a 2 elementos, el ángulo del robot respecto al punto deseado ϕ_g y el ángulo del robot respecto al plano ϕ_r (Ilustración 22).

El controlador PID(e) que regula la velocidad angular del robot se muestra a continuación:

$$\omega = K_P e + K_I \int e d\tau + K_D \dot{e} \tag{26}$$

Donde el error en el sistema se define como:

$$e = (\phi_g - \phi_r) \tag{25}$$

Como se ha mencionado anteriormente, se considera un sistema ideal, sin perturbaciones. Por tanto, se utilizará únicamente una ganancia proporcional, de forma que controlador[15] resultante queda:

$$\omega = K_P(\phi_g - \phi_r) \tag{27}$$

Una vez obtenido el valor de ω y asignando un valor de v que será la velocidad lineal máxima del robot, es posible calcular los valores de las velocidades de cada rueda aplicando las ecuaciones (23) y (24).

Para comprobar la eficacia de las fórmulas y poder así elegir un valor de la ganancia K_P adecuado, se han realizado una serie de pruebas con distintas posiciones del robot respecto al punto deseado. A continuación, se muestra un ejemplo de cómo se realiza el proceso de cálculo de velocidades del robot en base al controlador calculado previamente. Para ello se coloca el robot a 55º aproximadamente del punto deseado.

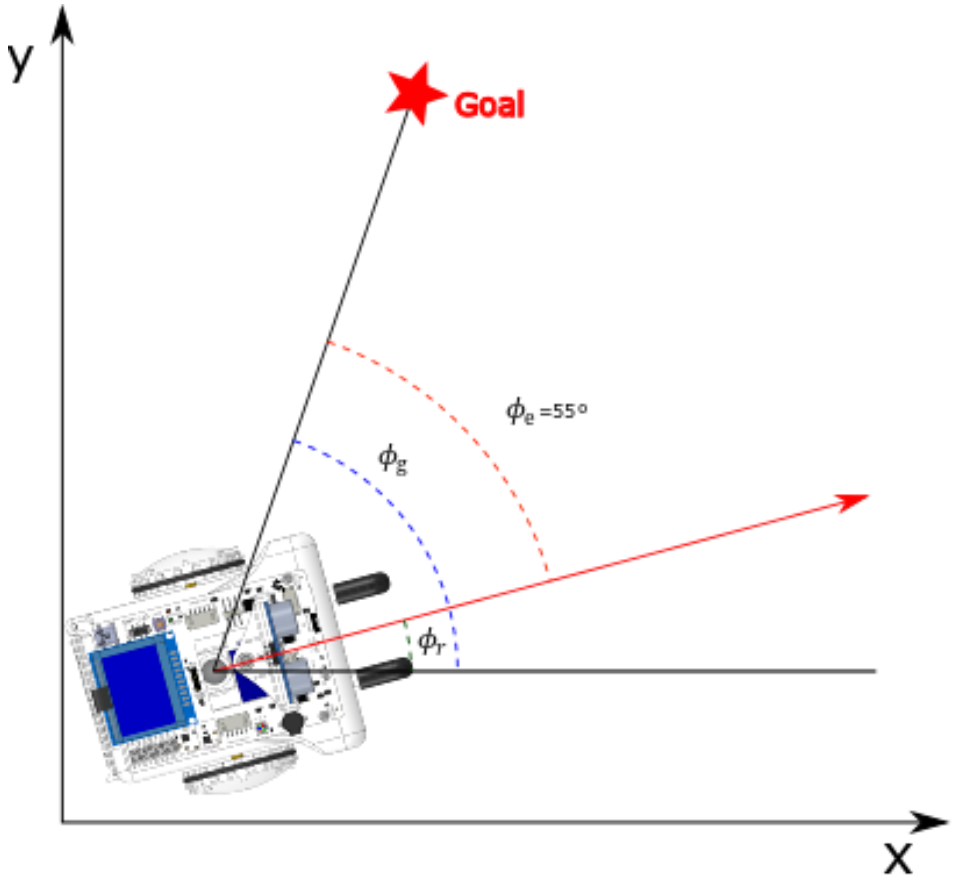

Ilustración 25: Disposición inicial a 55º

Tomando una ganancia $K_P = 1$, aplicando (27) obtenemos un valor de ω de 0,96 rad/s. A continuación, ese valor se aplica en las ecuaciones (23) y (24) donde v será 0.0278 m/s. Como resultado se obtiene:

$$v_r = 0.0686$$

$$v_l = -0.0131$$

Donde se puede apreciar que el valor de la velocidad lineal en la rueda derecha (v_r)es aproximadamente 2.5 veces mayor que la velocidad lineal máxima admisible. Teniendo en cuenta que:

$$v = \frac{v_r + v_l}{2} \tag{28}$$

Esta saturación de las velocidades es esperada pues si una de las 2 sale menor que la velocidad lineal máxima admisible, la otra saldrá n veces mayor, por lo que será necesario hacer un ajuste de éstas velocidades para evitar ésta saturación.

Para ello, se considera que la velocidad de mayor valor tendrá una velocidad del 100% de la máxima, posteriormente, la otra rueda tendrá una velocidad proporcional al valor obtenido siendo éste menor que el 100%.

De esta forma, los valores ajustados de las velocidades lineales quedarían:

$$v_r = 0.0278$$

$$v_l = -0.00521$$

Puesto que los valores de las velocidades del robot Sparki son introducidos en función del porcentaje de la velocidad lineal máxima, los valores recibidos por el robot serán:

$$v_r = 100$$

$$v_l = -19$$

Éste cálculo se realizará de forma periódica a medida que la cámara vaya capturando la posición del robot. Tras realizar diversos ensayos, se llegó a la conclusión de que el valor de ganancia óptimo teniendo en cuenta la velocidad del robot y el tiempo de muestreo de la cámara queda fijado en $\boldsymbol{K_P = 0.5}$.

A continuación, se muestra el diagrama de flujo del programa principal del proyecto ejecutando el algoritmo *GoToGoal*:

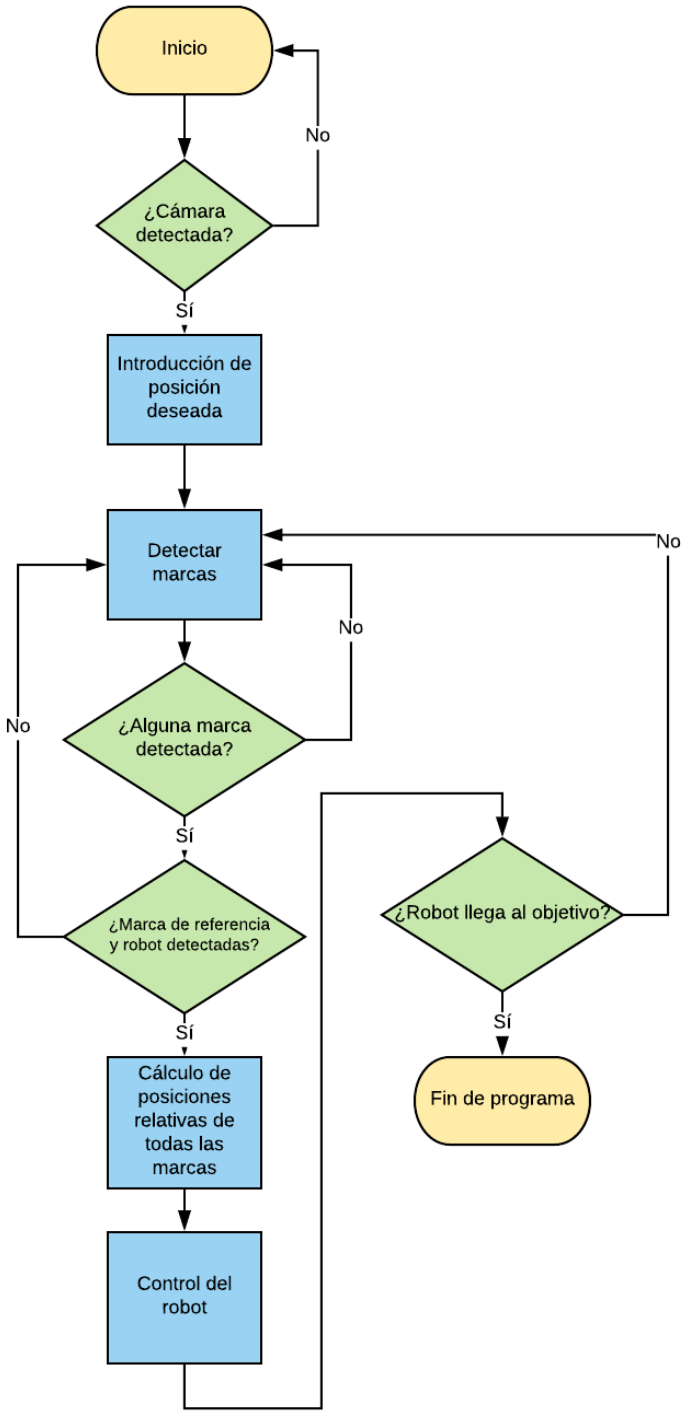

Ilustración 26: Diagrama de flujo del algoritmo GoToGoal

8.3 *GoToGoal* con múltiples marcas

Durante la ejecución de los algoritmos mencionados anteriormente, se observó que la precisión de la cámara a la hora de estimar la posición del robot, es mayor cuanto más cerca está éste de la marca ArUco de referencia.

Por ello se realizó otro programa, en el cual una vez fijada la posición del punto respecto a la marca de referencia, el robot disponía de varias marcas a tomar como referencia, tomando siempre la más cercana para así hacerlo con la mayor precisión.

Para la realización de este proyecto se optó por utilizar un máximo de 5 marcas ArUco, dado que a un mayor número de las mismas hace que el programa tarde más en procesar las imágenes (con la posición y orientación de cada marca), implicando así que se produzcan retardos en el envío y recepción de información. Esto provocaría así mayores errores de posición.

Aun así, es posible incluir tantas marcas ArUco como el usuario desee. Además, al igual que con el programa ***GoToGoal*** con una sola marca, en caso de tapar la marca de referencia, el robot parará, pues el punto deseado se sitúa únicamente respecto a la misma.

8.4 Otros algoritmos

GoToGoal Alternativo

El desarrollo de este algoritmo se desarrolló sobre papel y no sobre el robot por los motivos que se verán durante el desarrollo del mismo.

En este caso se utiliza un controlador proporcional para determinar la velocidad del robot en función de su posición y orientación respecto al punto deseado [15]:

$$v = k_v\sqrt{\left(x_g - x_r\right)^2 + \left(y_g - y_r\right)^2} \tag{29}$$

Donde k_v es la ganancia del sistema, x_g e y_g representa la posición del punto deseado y por último x_r e y_r representan la posición del robot en el sistema.

Posteriormente, para determinar el ángulo del robot respecto al punto, se utilizará la ecuación del apartado *GoToGoal* (eq. 27):

$$\omega = K_P\left(\phi_g - \phi_r\right)$$

Una vez obtenidos tanto el valor de ω como de v se aplicarán las fórmulas (23) y (24) para determinar la velocidad de cada rueda del robot.

$$v_r = v + \frac{\omega L}{2}$$

$$v_l = v - \frac{\omega L}{2}$$

De esta forma, las velocidades obtenidas irán disminuyendo a medida que el robot se aproxime al objetivo, alcanzando una $v = 0$ en el momento que éste haya alcanzado el punto.

Este algoritmo no se ejecutó en la práctica pues la velocidad máxima del robot es excesivamente baja por lo que en el momento en que éste empezase a acercarse al objetivo, si bien sus velocidades de lectura serían muy bajas (del orden de 1-10%) aparentemente el robot se mostrará parado, por lo que este algoritmo no es realmente efectivo para el proyecto.

Sin embargo, éste algoritmo se contará para futuras aplicaciones dado que en robots que funcionan a mayores velocidades, se logrará una mayor precisión a la par que menor tiempo de recorrido.

9 Ejemplo de aplicación del proyecto

Una vez cumplimentados todos los objetivos del presente proyecto, se procede a realizar un ejemplo de aplicación del mismo. Para ello se considerará un entorno multi-robot simulado contando aparte con el robot utilizado en el proyecto. El objetivo de esta aplicación consiste en guiar a cada uno de los robots a una posición deseada manteniendo siempre una distancia entre los mismos que permita su comunicación. De esta forma se facilitarían así las tareas de cooperación, así como el intercambio de datos o recepción de órdenes.

Para poder conseguir una distribución uniforme del equipo multi-robot, se llevará a cabo una resolución por diagramas de Voronoi.

9.1 Diagramas de Voronoi

Los Diagramas de Voronoi consisten en un método de interpolación basado en la distancia euclídea [16]. Se crean al unir los puntos entre sí, trazando las mediatrices de los segmentos de unión de los mismos. Las intersecciones de estas mediatrices determinan una serie de polígonos en un espacio alrededor de un conjunto de puntos de control, siendo en este caso los robots del proyecto.

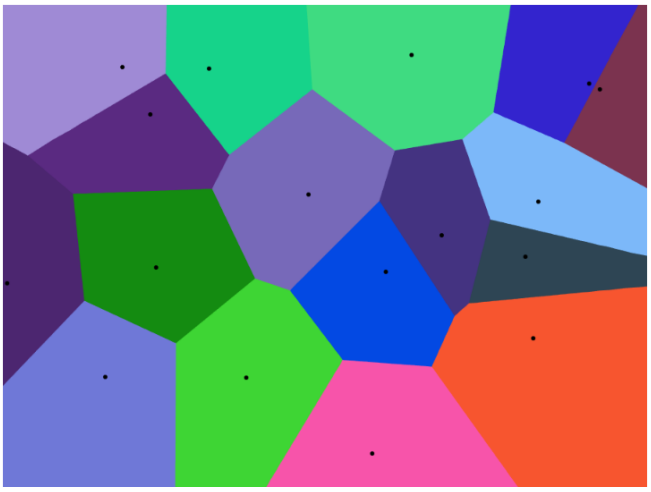

Ilustración 27: Ejemplo de diagramas de Voronoi

Para poder realizar una distribución uniforme de los robots, se calcularán los centroides de cada uno de los diagramas de Voronoi. Así pues, se tratará de un método iterativo en el cual las nuevas posiciones de los robots serán calculadas partiendo de sus posiciones actuales.

De esta forma, conocida la posición de cada uno de los robots, (incluido el físico) se les enviaran las coordenadas XY del punto a alcanzar. Posteriormente, el robot físico, utilizando tanto el algoritmo en 2 fases como el algoritmo *GoToGoal*, será capaz de alcanzar el objetivo fijado por el programa.

En este Trabajo de Fin de Grado se ha utilizado una implementación de una variación del método de Voronoi [17].

9.2 Aplicación utilizando 2 robots Sparki

Tras obtener un resultado satisfactorio de la aplicación mencionada previamente, se procedió a añadir al sistema otro robot físico para estudiar diversos factores tanto la conectividad, la precisión o la eficiencia del programa.

En el capítulo *5.- Conectividad* se menciona que el módulo bluetooth conectado al robot es el **HC-06**. Este módulo puede actuar únicamente como esclavo, siendo el maestro, el ordenador. En caso de contar con 2 robots, se ha de decidir de qué forma se realizará la comunicación bluetooth. Se cuenta con varias opciones, de las que se destacan 2:

- Colocar en uno de los robots un módulo bluetooth *HC-05* [18] dado que éste puede actuar tanto como maestro como esclavo. Por lo que el robot que llevase el módulo recibiría las órdenes desde el pc y enviaría las necesarias a los demás robots, que levarían el módulo *HC-06*.
- Seguir utilizando el ordenador como maestro, de forma que el programa en Matlab deba comunicarse con ambos robots enviando así información a 2 módulos bluetooth distintos, siendo ambos *HC-06*.

Finalmente se optó por la segunda opción dado que para la finalidad del proyecto se consideró suficiente utilizar a los 2 robots físicos como esclavos.

Para realizar la conectividad entre los 2 robots y el ordenador a través de Matlab, fue necesario en primer lugar, cambiar el nombre del dispositivo bluetooth *HC-06* de uno de los 2 robots pues ambos dispositivos vienen de fábrica con el mismo nombre, y Matlab en esta ocasión reconoce a los dispositivos bluetooth por su nombre, no por su dirección. Para cambiar el nombre de uno de los dispositivos fue necesario conectarlo a través de un Arduino UNO y ejecutar una serie de instrucciones a través de comunicación serial [19].

Finalmente, en Matlab, una vez configurados los dispositivos HC-06 de ambos robots, se almacenan en un vector los 2 objetos bluetooth conectados:

```
>> A(1)=Bluetooth('HC-06',1)
>> A(2)=Bluetooth('HC-06_2',1)
>>fopen(A(1))
>>fopen(A(2))
```

El desarrollo del programa con 2 robots se aplicará utilizando el algoritmo *GoToGoal* tomando como referencia una sola marca. El código del mismo se encuentra en el CD incluido en el proyecto.

10 Pruebas y resultados

A continuación, se procede a mostrar los resultados obtenidos en diferentes pruebas realizadas con los distintos algoritmos mencionados previamente. En éstas se tuvieron en cuenta los siguientes factores:

- **Posición inicial del robot**, pues la posición inicial del mismo respecto a la cámara puede variar en función de su distancia a la marca de referencia, lo que puede hacer que, dependiendo del algoritmo utilizado, el error de posición final sea elevado. Por tanto, en las diversas pruebas se colocó el robot a distintas distancias para observar si éste factor era realmente destacable.
- **Ángulo inicial del robot**. Este factor será relevante en el algoritmo en 2 fases, pues la posición y orientación del robot será tomada una sola vez, de forma que el resultado dependerá, entre otros factores, del ángulo inicial del mismo.

- **Posición final del robot.** Se refiere a la posición del robot una vez alcanzado el punto deseado. Este factor es crítico pues determinará la precisión del robot en función del algoritmo utilizado. Asimismo, se podrá comprobar el error de posición del robot una vez alcanzado el punto deseado.

  Es necesario considerar un radio objetivo sobre el punto deseado, pues al no ser un sistema ideal, si se considera que el robot debe alcanzar el punto exacto sin error, éste rara vez lo conseguirá ya sea por el valor de la ganancia o por la precisión de la cámara. Así pues, se consideró un radio de 5 centímetros respecto al punto deseado, por lo que en el momento que el robot entre en ese radio, éste se parará.

- **Distancia respecto al punto deseado.** Factor requerido para tener en cuenta el tiempo de respuesta del programa una vez alcanzado el radio objetivo del punto deseado (5 cm).

Otros factores como el tiempo que el robot tarda en alcanzar el punto deseado o la inclinación del terreno no se han tenido en cuenta, dada la baja velocidad máxima del robot (2.75 cm/s) y el uso de un controlador proporcional (donde no se tienen en cuenta perturbaciones externas).

Para los algoritmos utilizados se colocó al robot en las posiciones que se muestran a continuación, pues se consideraban suficientes para poder testear los factores mencionados anteriormente:

Posición inicial robot (metros)		Ángulo Inicial del robot (grados)	Posicion del punto deseado (metros)	
X	Y		X	Y
0,3	0	0	0,8	0,2
0,3	0	270	0,8	0,2
0,3	0	180	1,5	0,8
0,3	0	90	1,5	0,8
1,1	0,5	180	0,1	0,1
1,1	0,5	0	0,1	0,1
0	1	180	1	0
0	1	0	1	0

Tabla 2: Parámetros iniciales para realización de pruebas

A continuación, se detallarán los resultados obtenidos en cada algoritmo. Las tablas completas con todos los datos recopilados se encuentran en el **Anexo 1**.

10.1 Algoritmo en 2 fases

El algoritmo de 2 fases destaca principalmente por su no dependencia de la cámara una vez realizada la primera medida, pues el recorrido es calculado una sola vez. Así pues, en este caso no se requiere un radio objetivo, sino que se considera directamente el punto deseado pues el recorrido se calcula una sola vez. Además, la dependencia de los motores paso a paso puede provocar que en determinados casos se pueda producir un error de posición relativamente elevado, ya sea por el error de la rotación que realiza el robot como la distancia recorrida por éste.

Así pues, una vez realizados todos los recorridos mencionados anteriormente, se realiza la media aritmética tanto del error de posición como de la distancia al punto deseado.

Error de posición (metros)		Distancia al punto deseado (metros)	Error de ángulo (grados)	Radio de acción (metros)
X	Y			
0,0406625	0,03965	0,065160837	2,069715542	0

Tabla 3: Resultados Algoritmo en 2 Fases

Como se puede observar, si bien el error de posición tanto de X como de Y, es aproximadamente de 4 centímetros, la distancia al punto deseado es aproximadamente 6.5 centímetros. Como era de esperar, la precisión del algoritmo es relativamente baja, pues hay casos en los que la distancia al punto deseado llega a ser de 12 centímetros.

El error de posición producido con la ejecución del algoritmo en 2 fases se puede deber a diversos factores:

- **Error de posición de la cámara**, dado que, al localizarse el robot o la marca de referencia a una distancia alejada de la cámara, es posible que se pierda precisión a la hora de determinar la posición y orientación de cada uno de éstos.
- **Error de los motores paso a paso.** Es posible que, en caso de realizar un giro de un ángulo determinado, el error tienda a ser mayor cuan mayor sea el ángulo de giro del robot. Esto se puede deber, a que la precisión de los motores paso a paso del robot no sea ideal, por lo que es de esperar que se produzca un pequeño error de ángulo. Sin embargo, a pesar de que el error de ángulo pueda ser pequeño (en el caso de este apartado, de una media de 2,7 º), si la distancia a recorrer por el robot es muy elevada, se puede obtener un desvío considerable del robot respecto al punto deseado.
  Además, se comprobó que el error de la distancia a recorrer por parte del robot puede ser también debido a los motores paso a paso. Se realizaron varias pruebas con el robot recorriendo distancias de entre 20 y 150 cm, y se observó que a medida que se incrementaba la distancia a recorrer, se producía un error mayor. Es por ello que la acción de los motores paso a paso es posiblemente una de las principales causas del error de posición del robot.

10.2 Algoritmo *GoToGoal*

Para la ejecución de este algoritmo se ha considerado en primer lugar un radio objetivo de 5 centímetros teniendo en cuenta la precisión de la cámara y la ganancia elegida para el robot. Asimismo, se utilizó un solo robot y una marca ArUco como referencia, por lo que la posición final dependerá exclusivamente de la referencia tomada respecto a esa marca.

Además, a la hora de analizar los resultados del algoritmo, no se ha tenido en cuenta el ángulo final del robot respecto a la marca (a diferencia del algoritmo en 2 fases), dado que el programa calcula constantemente la trayectoria del robot, y por tanto, aunque se desvíe del objetivo en un primer momento, es capaz de llegar a éste tras recalcular el recorrido.

Así pues, tras realizar los test con los datos de la tabla 2, se obtienen los siguientes resultados de media:

Error de posición (metros)		Distancia al punto deseado (metros)	Radio de acción (metros)
X	Y		
0,0208	0,0144	0,027100405	0,05

Tabla 4: Resultados Algoritmo GoToGoal

Como se puede observar, el error de posición tanto de X como de Y es del orden de 2 centímetros, mientras que el error de distancia al punto deseado es cercano a 3 centímetros, siendo un valor menor que el radio de acción fijado (5 centímetros) y reduciéndose el error obtenido en el algoritmo en 2 fases a la mitad.

Sin embargo, en un sistema ideal, para que la precisión fuera exacta, el robot debería haberse parado a 5 centímetros del punto deseado, cuando en realidad lo hace a una media de 3 centímetros (0.0271 m). Por tanto, se puede considerar que la precisión del robot es de unos 2 centímetros de media (0.0228 m).

La precisión de este algoritmo es mayor dado que en este caso la posición y orientación del robot está siendo medida constantemente por la cámara, de forma que en cada iteración se calculan las velocidades del robot necesarias para alcanzar el objetivo. Además, en caso de producirse cualquier desvío del robot, ya sea porque éste ha sido movido por el usuario o porque se ha movido la cámara, es posible recalcular la posición final del objetivo y asimismo la del robot de forma que no se produzca un error elevado respecto a la posición final. De esta forma se pudo observar también que los resultados obtenidos no dependían realmente del ángulo inicial del robot.

Esto no era posible con el algoritmo en 2 fases puesto que la posición y orientación del robot se tomaba únicamente al principio de la ejecución del programa.

Los errores de posición producidos se pueden deber a distintos factores:

- **Error de precisión de la cámara**, dado que como con el algoritmo en 2 fases, es posible que se produzca un error en la toma de datos de la posición y orientación del robot, debido en mayor medida a la posición de la marca ArUco de referencia respecto a la cámara y al robot.
- **Tiempo de iteración del algoritmo**. Dado que en la ejecución del programa se deben calcular las posiciones relativas del robot respecto a la marca, además del controlador del robot y el envío de órdenes al mismo, se pueden producir una serie de retardos que afectan a la toma de medidas por parte de la cámara en tiempo real. Por tanto, aunque el robot esté realmente dentro del radio de acción, debido a estos retardos tarda un tiempo en ser consciente de la situación, y sigue moviéndose hacia el objetivo.

10.3 Algoritmo *GoToGoal* con varias marcas ArUco

Dado que en la aplicación del algoritmo *GoToGoal*, uno de los principales factores que provocan un error de posición en el robot es la distancia de la marca ArUco respecto a la cámara y al robot, se creó un programa en el cual se disponían en el sistema varias marcas ArUco. Durante la ejecución del programa el robot va tomando como referencia la marca ArUco más cercana de forma que se espere una precisión mayor a la hora de determinar la distancia relativa entre el robot y la marca de referencia elegida.

Así pues, una vez realizados los ensayos con los parámetros mostrados en la Tabla 2, se obtienen los siguientes resultados de media:

Error de posición (metros)		Distancia al punto deseado (metros)	Radio de acción (metros)
X	Y		
0,012775	0,0143375	0,01976068	0,05

Tabla 5: Resultados Algoritmo GoToGoal con varias marcas ArUco

Como se puede observar en la Tabla 5, tanto los valores de X como Y son próximos a 1,5 centímetros mientras que la distancia al punto deseado es próxima a 2 centímetros. Estos valores de media obtenidos son en mayor medida, menores que los obtenidos con los algoritmos anteriores. Sin embargo, en un principio se fijó que el robot debía pararse al alcanzar el radio de acción (5 centímetros) y éste sin embargo se para a los 2 centímetros de media (dentro del radio objetivo). Esto se traduce en que el robot se para en realidad a unos 3 centímetros del radio objetivo (aun estando dentro del mismo) por lo que la precisión resulta ser menor que la del algoritmo *GoToGoal* con una marca.

Este mayor error de posición se puede deber principalmente, al igual que en apartado anterior, al tiempo de iteración, pues en este caso se cuenta con un mayor número de marcas ArUco y en consecuencia, se puede tardar un mayor tiempo en calcular las posiciones relativas de cada marca respecto al robot.

10.4 Algoritmo *GoToGoal* con 2 robots

A continuación, se procede a comprobar que el algoritmo *GoToGoal* (con solo una marca ArUco como referencia) es válido también con 2 robots. En este caso se espera que la precisión de los robots sea menor pues el ordenador debe procesar y enviar al mismo tiempo órdenes para cada uno de los 2 robots. Esto puede provocar ciertos retardos en la cámara que se podrán ver reflejados en los resultados de precisión de los robots.

Para este caso, al contar con 2 robots, se generaron 3 posiciones iniciales para cada uno, sin tener en cuenta la orientación inicial, dado que en los resultados obtenidos en el algoritmo *GoToGoal* con 1 marca ArUco, se pudo observar que no era relevante. Además, se realizaron únicamente 3 pruebas porque se buscaba verificar el correcto funcionamiento de los robots con el algoritmo *GoToGoal*. En caso de querer obtener resultados basados en precisión, se habrían realizado un mayor número de pruebas. Las posiciones iniciales de los robots fueron las siguientes:

ROBOT 1				
	Posición inicial robot (metros)		Posicion del punto deseado (metros)	
	X	Y	X	Y
Prueba 1	0,3	0	0,8	0,2
Prueba 2	0,7	0,5	0,3	0
Prueba 3	0	0,8	1,2	0,2

ROBOT 2				
	Posición inicial robot (metros)		Posicion del punto deseado (metros)	
	X	Y	X	Y
Prueba 1	1	0,3	0,2	0,8
Prueba 2	1,2	0,2	0,3	0,4
Prueba 3	0,8	0	0,7	0,6

Tabla 6: Parámetros para pruebas con 2 robots

Tras realizar los ensayos con las distintas posiciones se obtuvieron los siguientes resultados de media:

Error de posición Robot 1 (metros)		Error de posición Robot 2 (metros)	
X	Y	X	Y
0,031333333	0,019333333	0,029666667	0,026666667

Distancia al punto deseado R1 (metros)	Distancia al punto deseado R2 (metros)	Radio de acción (metros)
0,041499015	0,043309906	0,05

Tabla 7: Resultados pruebas con 2 robots

Con lo que se puede observar a primera vista que los resultados son más precisos que los obtenidos con 1 robot y con el algoritmo *GoToGoal*. La distancia media al punto deseado es aproximadamente de 4 centímetros en ambos robots, lo que se acerca mucho al valor del radio de acción fijado (5 centímetros). Sin embargo, estos resultados no son relevantes dado el reducido número de pruebas realizadas.

El objetivo de este apartado era verificar el correcto funcionamiento del algoritmo *GoToGoal* con 2 robots y éste ha sido satisfactorio.

10.5 Velocidades teóricas y efectivas del robot Sparki

Para la comprobación de a precisión del robot a la hora de recorrer distancias, se realizaron una serie de ensayos para determinar si la distancia recorrida y la velocidad del robot se correspondían a las introducidas por el usuario.

Para ello se realizaron recorridos de diversas longitudes a distintas velocidades para comprobar que el robot las hacía en el tiempo esperado y cumpliendo con la distancia introducida en un principio. Se observó que, a mayor distancia a recorrer, mayor era el error que se producía por lo que se comprobó si las velocidades de las ruedas eran las adecuadas.

Así pues, se cronometró al robot recorriendo a determinadas velocidades una misma distancia y se observó que las velocidades programadas no se correspondían con las reales. Por ejemplo, recorriendo 15.7 cm de distancia, o lo que es lo mismo, el recorrido completo de una rueda, a la velocidad máxima, se tarda aproximadamente 5,64 segundos. Por tanto, se espera que recorriendo la misma distancia al 50% de su velocidad se debe obtener un tiempo aproximado a 11,28 segundos, es decir, el doble que el anterior. Sin embargo, tras programar el robot para que haga el recorrido a la mitad de la velocidad, éste tarda aproximadamente 9.6 segundos.

Para solventar este problema, se tomaron tiempos en el mismo recorrido a distintas velocidades y se representó en una gráfica la progresión de las mismas con respecto al tiempo obteniendo una curva aproximable mediante una línea de tendencia potencial.

La fórmula obtenida en la ecuación de la curva se correspondería con una aproximación a la función de transferencia de la velocidad de entrada (introducida por el usuario) con respecto a la de la salida (la real del robot). La función de transferencia en sí no se puede identificar, dado que con los sensores que cuenta el robot, resulta complicado hacer una identificación rigurosa de la velocidad que lleva el robot.

Las tablas con los cálculos realizados se encuentran en el CD como un documento Excel titulado "*Velocidades_robot.xlsx".*

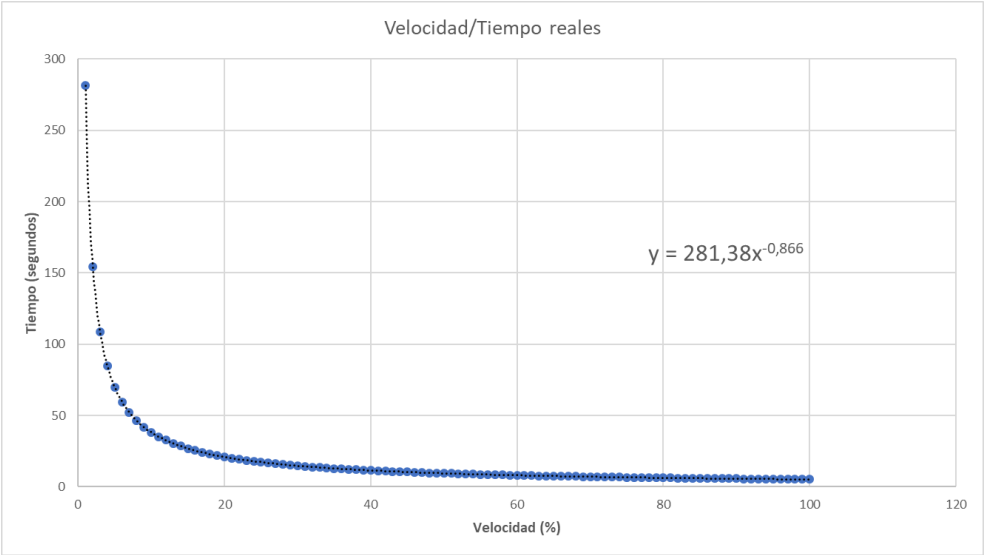

Tabla 8: Relación Velocidad/Tiempo reales

A través de la cual se puede obtener una aproximación más precisa de la velocidad real del robot. Como ejemplo de aplicación, para hacer que el realice un recorrido al 50% de su velocidad, o lo que es lo mismo, que recorra los 15.7 cm en 11,28 segundos, se aplica la fórmula de la línea de tendencia potencial:

$$Y = 281.38x^{-0.866} \tag{30}$$

Donde "*Y*" sería el tiempo deseado y "*x*" sería el porcentaje de velocidad, obteniendo así que la mitad de la velocidad es en realidad el **41%** de la máxima.

Posteriormente se procede a comprobarlo físicamente con el robot, y como resultado se obtiene que el rango de tiempos para las distintas velocidades se aproxima significativamente al esperado.

Si las velocidades fueran realmente el porcentaje de la total, es decir, que al 50%, el tiempo de recorrido fuera 11,28 segundos, se obtendría la siguiente tabla:

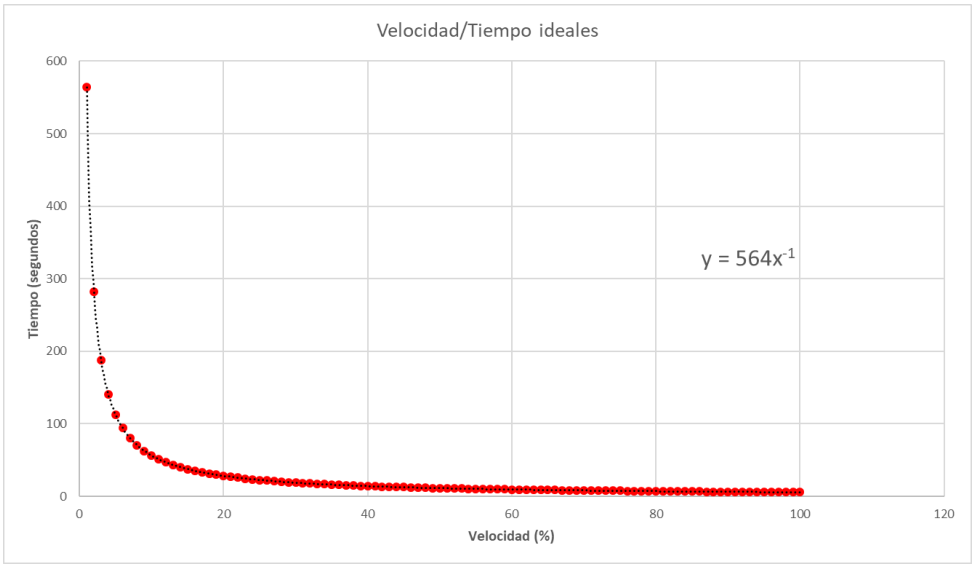

Tabla 9: Relación Velocidad/Tiempo ideales

Donde la ecuación asociada a la línea de tendencia obtenida sería:

$$Y = 564x^{-1} \tag{31}$$

A continuación, se procederá a aplicar está fórmula para obtener las velocidades correctas en cada caso. Para ello, se sustituye cada velocidad obtenida en (30), obteniendo el tiempo que se debe usar en (31) como *Y.* Finalmente, despejando la x se obtiene la velocidad correcta. Éstos cálculos se realizarán dentro de la programación del robot una vez recibidas las órdenes del ordenador.

10.6 Conclusión de resultados

Las conclusiones obtenidas en el apartado anterior, permitieron implementar en el robot un método para limitar el tiempo de muestreo del mismo en el algoritmo *GoToGoal* con sus respectivas variantes (con varias marcas, 2 robots, etc) debido a los retrasos producidos por la cámara. Este método se explicará más detalladamente en el capítulo *11.- Estudio de problemas y soluciones*.

Si bien los resultados de las tablas pueden ser aplicables, las medidas realizadas cuentan con un error de medida por parte del usuario pues el escenario del sistema es muy amplio y se pueden producir errores tanto de posición como orientación a la hora de medir tanto la posición inicial como la final del robot. Esto conlleva que el error de posición final del robot en cada algoritmo lleve asociado a su vez un error de medida por parte del usuario.

Por tanto, se recomienda realizar un mayor número de pruebas, ya que, con 8 pruebas por algoritmo, si bien se pudo determinar que éstos funcionaban correctamente, el número de pruebas realizado no fue relevante para poder determinar la precisión de los mismos. A pesar de ello, se obtuvieron errores de posición de menos de 5 centímetros de media.

Sin embargo, se pudo observar que el funcionamiento de cada algoritmo se veía afectado por los retardos producidos por la cámara, que a su vez se deterioraban al aumentar el número de operaciones que se tenían que ejecutar en cada algoritmo por iteración. Así pues, se procedió a medir el tiempo de iteración de media (en segundos) en cada uno de los programas realizados, llegando a obtener la siguiente tabla:

Algoritmo en 2 fases	***GoToGoal***	***GoToGoal* con múltiples marcas**	***GoToGoal* con 2 robots**
Irrelevante	0,594699474	0,935614199	0,782473196

Tabla 10: Tiempos de iteración de cada algoritmo en segundos

El algoritmo en 2 fases resultó ser irrelevante para esta medida pues la toma de información se realiza una sola vez. Sin embargo, para el resto de algoritmos se puede observar que el que tiene mayor tiempo de iteración es el algoritmo *GoToGoal* con múltiples marcas (0.9356 segundos), pues es en éste donde se presentan más marcas ArUco (un total de 5 incluyendo la del robot) sobre las cuales se debe determinar en cada iteración la posición y orientación de éstas respecto al robot, tomando como referencia la más cercana al mismo.

El algoritmo *GoToGoal con 2 robots* tiene un tiempo de iteración menor que el algoritmo mencionando anteriormente a pesar de tener que calcular el control de cada robot y de realizar el envío de datos a ambos. Esto se debe a que solo se dispone de una marca ArUco como referencia por lo que solo hay que calcular la posición de los 2 robots respecto a ésta, lo que se traduce en un menor tiempo de cálculo.

Por último, el algoritmo *GoToGoal* es el que tiene menor tiempo de iteración (0.594 segundos) debido a que únicamente se cuenta con un robot y una marca ArUco por lo que sólo se debe calcular la posición relativa de éste en cada iteración.

Por tanto, se puede concluir que uno de los principales factores que determinan el tiempo de iteración y, por tanto, que puede afectar a la precisión del robot a la hora de alcanzar el punto deseado, es el cálculo de las posiciones relativas del mismo. A un mayor número de marcas, se debe realizar un mayor número de cálculos por lo que el tiempo de iteración es mayor, y esto se ha podido observar en la tabla 9, donde el algoritmo cuyo sistema presenta más marcas es el que tiene un mayor tiempo de iteración.

11 Estudio de problemas y soluciones

Durante la realización del presente proyecto se encontraron diversos factores que alteraron la funcionalidad del mismo:

11.1 Saturación de velocidades

El robot Sparki cuenta con 2 motores paso a paso cuya velocidad máxima, como se menciona en apartados anteriores, es aproximadamente 2.78 cm/s. Durante la ejecución del algoritmo *GoToGoal*, el ordenador debe calcular las velocidades que debe enviar a cada rueda del robot en función de su posición y orientación a las marcas de referencia. Dependiendo del valor de la ganancia y de la velocidad lineal máxima establecida, las velocidades obtenidas aplicando las fórmulas del apartado *8.2.- GoToGoal* pueden ser, para ambas ruedas, de un valor inmensurablemente elevado, del orden de incluso 100 veces mayor que la máxima. Esto se puede deber a varios factores, entre los que destacan:

- **Fijar como velocidad lineal constante la velocidad máxima del robot.** En ese caso, si el robot tiene que ir en línea recta, no se produce problema alguno, pues cada rueda se desplaza a la misma velocidad (la máxima), y según la fórmula (28):

$$v = \frac{v_r + v_l}{2}$$

  La velocidad total del robot será también la máxima, sin aparecer ningún tipo de saturación. Sin embargo, en caso de ocurrir un elevado error de orientación, una de las velocidades tendrá que ser mayor que la otra para poder corregir ese error, lo que como resultado se obtendrán velocidades mucho mayores que las máximas.
- **Valor de la ganancia muy elevado.** Si el valor de la ganancia del sistema es muy elevado, en el momento de corregir el error de orientación se obtendrán velocidades muy elevadas pues a un mayor valor de ganancia se tiende a corregir el error en menor tiempo, por lo que el robot dará giros muy bruscos para corregir su posición.

  Como posibles soluciones se encuentran:

- **Reducir la velocidad lineal constante del robot.** Con esta solución habría menos probabilidad de que el robot alcanzara velocidades superiores a su máxima en caso de necesitar corregir su orientación. Sin embargo, teniendo en cuenta que todavía existiría la posibilidad de saturar, esta opción se descartó en su momento puesto que se decidió que el robot trabajara siempre a velocidad lineal máxima dado el bajo valor de la misma.

- **Reducir el valor de la ganancia K.** Ajustando el valor de la ganancia se lograría que el robot no saturara tanto en los giros, sin embargo, un valor muy bajo haría que el robot diera giros muy abiertos de forma que, si se encontrara cerca del objetivo, éste posiblemente no pudiera alcanzarlo en caso de intentar corregir su orientación.
- **Reducción tanto de la ganancia K como de la velocidad lineal máxima.** Combinando las dos soluciones mostradas anteriormente podría solucionar el problema. Sin embargo, dado que la velocidad máxima del robot es muy baja, el movimiento convencional del robot sería muy lento.
- **Ajustar las velocidades saturadas en función de la máxima.** Ésta solución es la elegida, pues consiste en considerar la mayor de las velocidades (esté o no saturada) como la velocidad máxima, y calcular la velocidad proporcional de la otra velocidad en función de la máxima [20].

11.2 Retrasos con la cámara

Uno de los problemas principales relacionados con el sistema sensorial externo está directamente relacionado con la captura de imágenes por parte de la cámara Kinect y el procesamiento de la información sensorial (extracción de marcas ArUco).

En un sistema ideal, el programa capturaría y procesaría una imagen (*frame*) para obtener la posición y orientación del robot sin producirse ningún retardo por tiempo de procesamiento de la imagen. Sin embargo, dada la elevada resolución de la cámara, y envergadura del programa, la imagen que se muestra en pantalla aparece con cierto retardo, actualizándose aproximadamente cada 0.8-0.9 segundos, y en muy pocas ocasiones, incluso cada 2-3 segundos.

Este elevado tiempo de iteración del programa supone un problema de precisión para el robot pues el periodo de muestreo de éste es muy elevado por lo que se asemeja a un bloqueador de orden cero a la entrada. Como consecuencia el robot se desviará de su objetivo pues estará ejecutando las mismas órdenes hasta recibir de nuevo información por parte de la cámara.

Para solventar este problema, se consideraron las siguientes soluciones:

- **Cambiar la cámara.** Puesto que la elevada resolución de la Kinect y su tiempo de procesamiento de imagen son una de las principales causas del elevado tiempo de iteración, una solución consistiría en utilizar una cámara con una menor resolución. De esta forma se reduciría notablemente el tiempo de iteración, sin embargo, al reducir la resolución de la cámara, existe la posibilidad de que la precisión a la hora de medir distancias se vea drásticamente reducida. Dado que la precisión es uno de los pilares del presente proyecto, se decidió no cambiar la cámara y buscar otras posibles soluciones.
- **Limitar el tiempo de muestreo del robot.** Como se mencionó previamente, el tiempo de muestreo del robot es demasiado alto por lo que una posible solución para paliar el problema producido por el bloqueador de orden cero sería reducir el tiempo de muestreo limitando la lectura del robot a un período fijo de aproximadamente 1-1,5 segundos. Esto sería posible a través de la programación del recorrido del robot en función del número de steps (pasos) que avanza cada motor paso a paso en función de su velocidad. Aplicando la fórmula obtenida en el apartado (10.5):

$$Y = 281.38x^{-0.866} \tag{30}$$

Que relaciona la velocidad real del robot con el período de una rueda (segundos x vuelta), es posible limitar en un periodo de tiempo la velocidad del robot, de forma que, si se produce un retardo en la información enviada por la cámara, el robot permanecerá estacionado hasta recibir nueva orden. Si bien aparentemente, el robot puede realizar su recorrido de forma no continuada (parándose de vez en cuando), se optimiza el programa pues en caso de corregir su orientación y no recibir más órdenes, éste no permanecerá en movimiento incrementando el error ya corregido, sino que parará tras hacer su recorrido de 1 segundo y esperará nueva orden.

Esta solución fue la elegida pues tras realizar numerosas pruebas se llegó a la conclusión de que era la más apropiada para mejorar la precisión del robot.

11.3 Ejes erróneos ArUco

Un problema puntual relacionado con el programa de detección de marcas ArUco es el producido por una mala proyección de los ejes de la marca a localizar. En cada marca detectada, se muestran los ejes XYZ sobre su superficie de forma que es posible determinar la orientación de la misma respecto a una marca o punto de referencia.

Sin embargo, ya sea por la lejanía de la marca respecto a la cámara, la luminosidad u otros factores, en determinadas ocasiones los ejes mostrados no correspondían con lo esperado en la realidad.

A continuación, se muestra una imagen donde la disposición de los ejes XYZ sobre el robot es correcta:

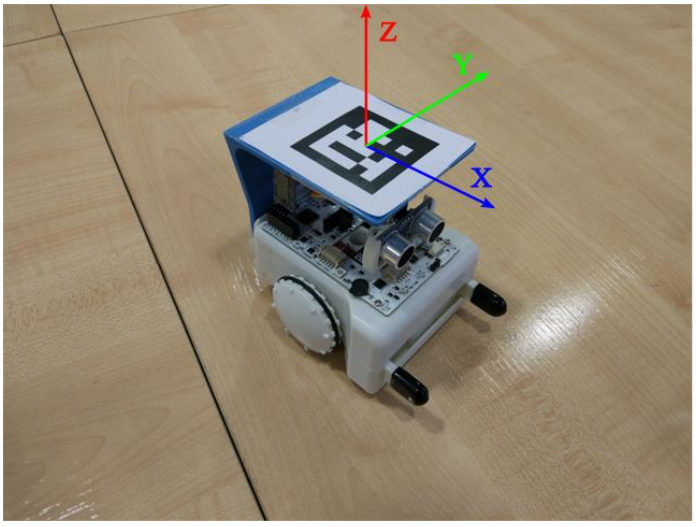

Ilustración 28: Robot Sparki con ejes correctos

En caso de producirse el problema mencionado, los ejes se disponen de la siguiente forma:

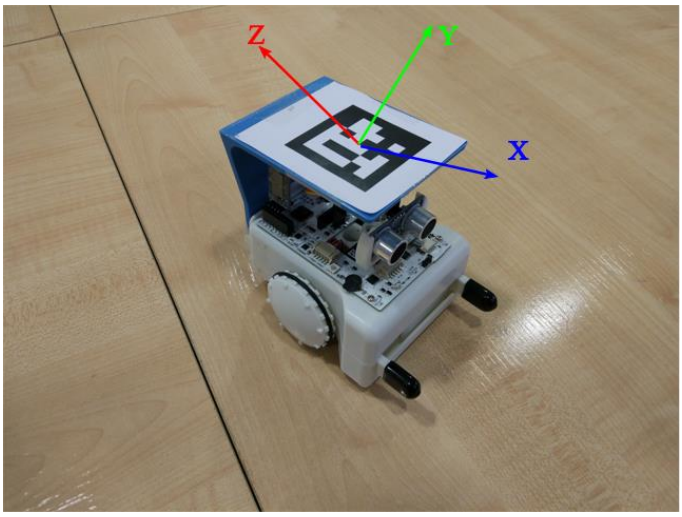

Ilustración 29: Robot Sparki con ejes erróneos

Donde se puede apreciar que todos los ejes aparecen levemente rotados. La causa de este problema se debe a que el programa interpreta que la marca está inclinada, puesto que, si ésta estuviera así, los ejes se dispondrían como en la imagen.

Este problema tenía como consecuencia la lectura errónea por parte de la cámara de la posición y orientación del robot, lo que se traducía en una desviación del robot de la ruta programada.

Se optaron por varias soluciones, entre las que destacaban:

- **Incrementar el tamaño de las marcas ArUco.** Esta solución se intentó implementar en un primer momento, pero se observó que para que surtiera algún efecto notorio, habría que crear unas marcas de unas dimensiones mayores que la propia superficie del robot, por lo que esta opción fue descartada.
- **Cambiar el comportamiento del robot.** Dado que este problema ocurría con una frecuencia muy baja, y durante períodos muy cortos de tiempo (1-1,5 segundos), se optó por hacer que el programa no procesara ninguna información en caso de recibir unos datos que no concordaran con lo habitual, (por ejemplo, tener el eje X o Y rotados). Aprovechando la solución propuesta con el problema del tiempo de muestreo, el robot permanecería parado hasta que la cámara consiga capturar una imagen con los ejes dispuestos correctamente. Sin embargo, esta solución fue descartada puesto que se podían dar casos en los que los ejes aparecieran rotados de forma indefinida (sobre todo en casos en los que el robot se encontraba muy alejado de la cámara) de forma que el robot permanecía parado de forma indefinida.
- **Cambiar la posición de la cámara.** Durante el desarrollo del proyecto, se utilizó un trípode para colocar la cámara en una posición elevada de forma que se pudiera abarcar cuanto más espacio en el sistema. Sin embargo, desde un principio se observó que la configuración óptima sería tener la cámara en el techo de forma que se minimizaran los problemas con los ejes del robot.
  Así pues, se procedió a colocarla de forma que estuviera enfocando el sistema desde un techo a una altura de unos 2 metros. La principal ventaja de tener la cámara en esta posición es que las marcas apenas tendrían inclinación de forma

que sería menos probable que se produjera el problema mencionado con los ejes.
Una vez realizada la configuración, se observó que el problema con los ejes seguía ocurriendo, pero con mucha menor frecuencia, además de que la precisión de medida de distancias por parte de la cámara era mayor. Por tanto, se eligió ésta como la solución más óptima para el proyecto.

Ilustración 30: Disposición final de la cámara

Otra solución posible habría sido implementar una versión propia del método para manejar las marcas ArUco, o pre-procesar las imágenes antes de ser usadas por el extractor de ArUco. Sin embargo, este proceso fue descartado pues no entraba dentro de los objetivos principales del presente proyecto.

12 Conclusiones y trabajo futuro

12.1 Conclusión

Tras la realización de este Trabajo de Fin de Grado se procede a mencionar las principales conclusiones del mismo.

Los objetivos propuestos al principio del proyecto se han alcanzado satisfactoriamente. Se ha conseguido a partir de un robot educativo de bajo coste y una cámara Kinect, elaborar un sistema en el cual el robot puede alcanzar un objetivo deseado conociendo su posición a través de la propia cámara en tiempo real. La comunicación entre los distintos elementos del sistema (ordenador, robot y cámara) ha sido posible gracias al uso de un módulo bluetooth HC-06 que comunicaba al robot con el ordenador. Asimismo, la cámara enviaba toda la información necesaria al ordenador mediante conexión USB de forma que los sistemas quedaban interconectados entre sí.

Se lograron programar 2 algoritmos, el algoritmo en *2 fases*, en el cual, una vez conocida la posición inicial del robot, éste rehúsa a recibir más información por parte de la cámara alcanzando, sin embargo, el objetivo deseado; y el algoritmo *GoToGoal* con varias versiones y

modificaciones en el cual, la posición del robot es recogida en todo momento por la cámara de forma que, a través del controlador calculado, se puede guiar al robot con mayor precisión.

Además de alcanzar los objetivos propuestos, se lograron otros como:

- A fin de mejorar la precisión del programa de *GoToGoal*, se realizó una modificación en la cual se le permitía al usuario utilizar varias marcas ArUco de forma que el robot pudiera tomar como referencia la más cercana incrementando así la precisión a la hora de tomar medidas.
- Se realizó un ejemplo de aplicación con un entorno multi-robot simulado aplicando diagramas de Voronoi, de forma que se podía aplicar cualquiera de los algoritmos programados en un entorno de más de un robot.
- Se consiguió implementar la aplicación del entorno multi-robot con 2 robots reales, pudiendo realizar una satisfactoria comunicación entre éstos y el ordenador, la adaptación del algoritmo *GoToGoal* para su funcionamiento con 2 robots reales, y la aplicación de todo el conjunto al entorno simulado con diagramas de Voronoi.

Para terminar con las conclusiones generales, este proyecto ha supuesto todo un reto en la implementación software como hardware pues involucraba la aplicación de diversas ramas de la ingeniería como robótica, visión por computador, control automático, así como diseño e impresión 3D de piezas.

12.2 Trabajo futuro y consideraciones finales

A pesar de haber alcanzado los objetivos propuestos del Trabajo de Fin de Grado, se proponen una serie de trabajos futuros dado el abanico de opciones que ofrece.

En primer lugar, se propone el uso de ordenador más potente que permita ejecutar los programas con varios robots físicos sin producirse problemas de rendimiento, así como los retrasos producidos por la cámara.

Asimismo, se propone realizar un mayor número de ensayos con los distintos algoritmos para justificar la precisión de cada uno de los mismos ya que en el desarrollo de este Trabajo de Fin de Grado se han realizado solo 8 ensayos por cada algoritmo.

Se propone además para un trabajo futuro, la programación del sistema de forma que se favorezca la evasión de obstáculos por parte de los robots, dado que es posible que 2 robots colisionen durante la ejecución de sus respectivos programas. Para ello se hará uso tanto de los sensores incluidos en los robots (sensor de infrarrojos) como de la propia cámara.

Por último, otro de los elementos a con los que trabajar en el proyecto es en la inclusión de un PID al algoritmo de forma que se puedan corregir perturbaciones externas, como por ejemplo la inclinación del terreno.

13 Bibliografía

[1] **Mbot** (último acceso 5 de junio de 2018)

https://makeblock.es/productos/robot_educativo_mbot/

[2] **GoPiGo** (último acceso 5 de junio de 2018)

https://www.dexterindustries.com/gopigo3/

[3] **Arcbotics Sparki** (último acceso 5 de junio de 2018)

http://arcbotics.com/products/sparki/

[4] **Movimiento diferencial** (último acceso 5 de junio de 2018)

https://en.wikipedia.org/wiki/Differential_wheeled_robot

[5] **Motores paso a paso** (último acceso 5 de junio de 2018)

http://www.neoteo.com/motores-paso-a-paso/

[6] **Lenguajes de programación Sparki** (último acceso 5 de junio de 2018)

http://arcbotics.com/lessons/sparki-install-software/

[7] **Comandos librería <sparki.h>** (último acceso 5 de junio de 2018)

http://arcbotics.com/products/sparki/parts/

[8] **Conexión HC-06** (último acceso 5 de junio de 2018)

http://arcbotics.com/products/sparki/parts/bluetooth-module/

[9] **Cámara Kinect** (último acceso 5 de junio de 2018)

http://www.kinectfordevelopers.com/es/2012/11/06/que-es-el-dispositivo-kinect/

[10] **Instalación OpenCV** (último acceso 5 de junio de 2018)

https://github.com/kyamagu/mexopencv/wiki

[11] **Marcas ArUco** (último acceso 5 de junio de 2018)

https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html

[12] **Calibración cámara** (último acceso 5 de junio de 2018)

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

[13] Francesco Bullo and Stephen L. Smith. **Rodrigues Lectures on Robotic Planning and Kinematics**, Version v0.91(d) (7 Apr 2016).

https://en.wikipedia.org/wiki/Rotation_matrix#Determining_the_axis (último acceso 5 de junio de 2018)

https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula (último acceso 5 de junio de 2018)

[14] Sandeep Kumar Malu & Jharna Majumdar, **Kinematics, Localization and Control of Differential Drive Mobile Robot**. Vol14, 2014.

[15] Nathan Sprague, **Controlling Physical Systems**. Version 1.0, 2016.

[16] **Diagramas de Voronoi** (último acceso 5 de junio de 2018)

https://es.wikipedia.org/wiki/Pol%C3%ADgonos_de_Thiessen

[17] Aragües, Rosario, Carlos Sagues, and Youcef Mezouar. "**Triggered minimum spanning tree for distributed coverage with connectivity maintenance.**" Control Conference (ECC), 2014 European. IEEE, 2014.

[18] **Módulo HC-05** (último acceso 5 de junio de 2018)

https://www.prometec.net/bt-hc05/

[19] **Cambio de nombre a módulo HC-06** (último acceso 5 de junio de 2018)

http://www.instructables.com/id/How-to-Change-the-Name-of-HC-06-Bluetooth-Module/

[20] Alessandro De Luca, Giuseppe Oriolo, Marilena Vendittelli**, Control of Wheeled Mobile Robots: An Experimental Overview**, Dipartimento di Informatica e Sistemistica, Università degli Studi di Roma “La Sapienza”, Italy.

ANEXO I

Tablas de resultados de las pruebas realizadas con los distintos algoritmos

1 Algoritmo en 2 fases

	Posición inicial robot (metros)		Posicion del punto deseado (metros)		Posición final robot (metros)		Error de posición (metros)	
	X	Y	X	Y	X	Y	X	Y
Prueba 1	0,3	0	0,8	0,2	0,805	0,185	-0,005	0,015
Prueba 2	0,3	0	0,8	0,2	0,8032	0,201	-0,0032	-0,001
Prueba 3	0,3	0	1,5	0,8	1,5943	0,7818	-0,0943	0,0182
Prueba 4	0,3	0	1,5	0,8	1,591	0,7857	-0,091	0,0143
Prueba 5	1,1	0,5	0,1	0,1	0,059	0,1252	0,041	-0,0252
Prueba 6	1,1	0,5	0,1	0,1	0,0492	0,1635	0,0508	-0,0635
Prueba 7	0	1	1	0	1,03	-0,12	-0,03	0,12
Prueba 8	0	1	1	0	0,99	0,06	0,01	-0,06

	Ángulo Inicial del robot (grados)	Ángulo giro (grados)	Ángulo giro (grados)	Error ángulo (grados)	Distancia al punto deseado (metros)	Ángulo Inicial del robot (grados)
Prueba 1	0	21,80140949	20,11963314	1,681776346	0,015811388	0
Prueba 2	-90	111,8014095	111,7739201	0,027489405	0,003352611	-90
Prueba 3	180	-146,3099325	-148,8666356	2,556703141	0,096040252	180
Prueba 4	90	-56,30993247	-58,67539035	2,365457881	0,092116719	90
Prueba 5	180	21,80140949	19,80082531	2,000584176	0,048125253	180
Prueba 6	0	-158,1985905	-162,2432607	4,044670178	0,081319678	0
Prueba 7	180	135	132,6029715	2,39702853	0,123693169	180
Prueba 8	0	-45	-43,51598532	1,484014676	0,060827625	0

Error de posición (metros)		Distancia al punto deseado (metros)	Error de ángulo (grados)	Radio de acción (metros)
X	Y			
0,0406625	0,03965	0,065160837	2,069715542	0

2 Algoritmo *GoToGoal* con una marca ArUco

	Posición inicial robot (metros)		Posicion del punto deseado (metros)		Posición final robot (metros)		Error de posición (metros)		Distancia al punto deseado (metros)
	X	Y	X	Y	X	Y	X	Y	
Prueba 1	0,3	0	0,8	0,2	0,7971	0,1784	0,0029	0,0216	0,021793806
Prueba 2	0,3	0	0,8	0,2	0,7782	0,1907	0,0218	0,0093	0,023700844
Prueba 3	0,3	0	1,5	0,8	1,4766	0,8264	0,0234	-0,0264	0,035277755
Prueba 4	0,3	0	1,5	0,8	1,46	0,8087	0,04	-0,0087	0,040935193
Prueba 5	1,1	0,5	0,1	0,1	0,1064	0,1001	-0,0064	-1E-04	0,006400781
Prueba 6	1,1	0,5	0,1	0,1	0,0649	0,1234	0,0351	-0,0234	0,04218495
Prueba 7	0	1	1	0	0,9911	-0,0005	0,0089	0,0005	0,008914034
Prueba 8	0	1	1	0	1,0279	0,0252	-0,0279	-0,0252	0,037595877

Media	0,027100405

Error de posición (metros)		Distancia al punto deseado (metros)	Radio de acción (metros)
X	Y		
0,0208	0,0144	0,027100405	0,05

3 Algoritmo *GoToGoal* con varias marcas ArUco

	Posición inicial robot (metros)		Posicion del punto deseado (metros)		Posición final robot (metros)		Error de posición (metros)		Distancia al punto deseado (metros)
	X	Y	X	Y	X	Y	X	Y	
Prueba 1	0,3	0	0,8	0,2	0,782	0,1856	0,018	0,0144	0,023051247
Prueba 2	0,3	0	0,8	0,2	0,7856	0,1792	0,0144	0,0208	0,025298221
Prueba 3	0,3	0	1,5	0,8	1,493	0,7832	0,007	0,0168	0,0182
Prueba 4	0,3	0	1,5	0,8	1,4856	0,8234	0,0144	-0,0234	0,027475808
Prueba 5	1,1	0,5	0,1	0,1	0,0932	0,1103	0,0068	-0,0103	0,012342204
Prueba 6	1,1	0,5	0,1	0,1	0,0894	0,112	0,0106	-0,012	0,016011246
Prueba 7	0	1	1	0	0,995	0,005	0,005	-0,005	0,007071068
Prueba 8	0	1	1	0	0,974	0,012	0,026	-0,012	0,028635642

Media	0,01976068

Error de posición (metros)		Distancia al punto deseado (metros)	Radio de acción (metros)
X	Y		
0,012775	0,0143375	0,01976068	0,05

4 Algoritmo *GoToGoal* con 2 robots

	ROBOT 1								
	Posición inicial robot (metros)		Posicion del punto deseado (metros)		Posición final robot (metros)		Error de posición (metros)		Distancia al punto deseado (metros)
	X	Y	X	Y	X	Y	X	Y	
Prueba 1	0,3	0	0,8	0,2	0,786	0,206	0,014	-0,006	0,015231546
Prueba 2	0,7	0,5	0,3	0	0,275	0,048	0,025	-0,048	0,054120237
Prueba 3	0	0,8	1,2	0,2	1,145	0,196	0,055	0,004	0,055145263

Media Robot 1	0,041499015

	ROBOT 2								
	Posición inicial robot (metros)		Posicion del punto deseado (metros)		Posición final robot (metros)		Error de posición (metros)		Distancia al punto deseado (metros)
	X	Y	X	Y	X	Y	X	Y	
Prueba 1	1	0,3	0,2	0,8	0,192	0,789	0,008	0,011	0,013601471
Prueba 2	1,2	0,2	0,3	0,4	0,265	0,398	0,035	0,002	0,035057096
Prueba 3	0,8	0	0,7	0,6	0,654	0,667	0,046	-0,067	0,081271151

Media Robot 2	0,043309906

Error de posición Robot 1 (metros)		Error de posición Robot 2 (metros)		Distancia al punto deseado R1 (metros)	Distancia al punto deseado R2 (metros)	Radio de acción (metros)
X	Y	X	Y			
0,031333333	0,019333333	0,029666667	0,026666667	0,041499015	0,043309906	0,05