Universidad Zaragoza
1542

Trabajo Fin de Grado

Control orbital de robots móviles basado en visión por ordenador

Computer-vision based orbital control for mobile robots

Autor

Lucía Sánchez Artigas

Director

Gonzalo López Nicolás

Ingeniería de tecnologías industriales

Departamento de informática e ingeniería de sistemas

Escuela de ingeniería y arquitectura

2020

Control orbital de robots móviles basado en visión por ordenador

Resumen

A mediados del siglo pasado, la industria se vio revolucionada por el desarrollo de la robótica. Los brazos mecánicos permitieron la automatización de infinidad de procesos, aumentando la precisión y la producción.

Poco a poco, estas tecnologías, utilizadas casi exclusivamente en fábricas, fueron ampliando su área de influencia. Empezaron a diseñarse robots móviles que sustituyeran a las personas en trabajos en ambientes peligrosos o difícilmente accesibles, y pronto se descubrieron las ventajas de este tipo de máquinas. Hoy en día, no es extraño que un hogar cuente con asistentes de limpieza robóticos o que una empresa utilice robots móviles en su logística.

En este trabajo de fin de grado se detalla la creación de un control de movimiento que permite a un robot móvil orbitar alrededor de un punto de manera autónoma. Para ello, se utiliza una cámara que toma imágenes del entorno de manera continua, y una serie de marcadores visuales cuya detección y posterior procesamiento proporciona información relativa a la posición y orientación del robot. De esta manera, se puede corregir su trayectoria en tiempo real. La plataforma utilizada es un robot constituido por una Raspberry Pi y un par de motores que se encargan de mover las ruedas tractoras. Los marcadores a detectar provienen de la librería Aruco, y el código está escrito en Python 3. La cámara es el único sensor utilizado.

En el trabajo desarrollado, se tratan dos casos principales: rotación alrededor de una figura con marcadores en su superficie y rotación alrededor de un único marcador colocado en el suelo. El funcionamiento de ambos es similar: el robot adquiere información de los marcadores y calcula la distancia a la que se encuentra del centro de giro. A partir de dicha información, el sistema desarrollado calcula el error y modifica su trayectoria para ajustarse al radio deseado.

Como complemento, se exponen programas con distintas funciones: reconocimiento de marcadores, guiado del robot hasta un marcador mediante trayectorias rectilíneas, cálculo de los sistemas de referencia a utilizar y giro en torno a una figura mediante líneas rectas y giros sin avance.

Una vez diseñadas, las aplicaciones se probaron experimentalmente para comprobar su funcionamiento y ajustar manualmente, si fuera necesario, sus parámetros. Todas las gráficas y figuras contenidas en esta memoria pertenecen a casos reales de movimiento del robot.

Tabla de contenido

Control orbital de robots móviles basado en visión por ordenador 2
Resumen 2
1. Introducción 5
1.1. Objetivos 5
1.2. Trabajos previos 5
1.3. Alcance 5
1.4. Metodología y cronología 6
1.5. Estructura de la memoria 7
2. Herramientas utilizadas 8
2.1. Pi Robot 8
2.2. Python 10
2.3. Marcadores Aruco 11
2.4. OpenCV 12
2.5. Entorno virtual 12
2.6. VNC 13
2.7. Prisma hexagonal 13
3. Sistemas de referencia 14
3.1. Sistema de referencia marcador 14
3.2. Sistema de referencia cámara 14
3.3. Sistema de referencia robot 15
4. Implementación del código sin control 17
4.1. Detección de marcadores y proyección de resultados 17
4.2. Movimientos en línea recta 18
4.3. Obtención de sistemas de referencia 20
4.4. Aplicación de los sistemas de referencia al movimiento 21
4.5. Introducción del movimiento alrededor de una figura 22
5. Implementación del código con control 23
5.1. Control 23
5.2. Giro alrededor del prisma hexagonal 26
5.3. Giro alrededor de un marcador horizontal 31
6. Conclusión 34
ANEXOS 35
A1. Código de los programas. 35
A1.1. DetectAxis.py 35
A1.2. VeAMarca3.py 37

A1.3. Ref.py 40
A1.4. VeLat.py 42
A1.5. GiraHex3.py 47
A1.6. MarcaV.py 56
A1.7. MarcaH.py 61
A2: Calibración de la cámara 68
A2.1. Fundamentos 68
A2.2. Software 69
A2.3. Proceso de calibración 69
A3: Problemas encontrados 70
A3.1. Versiones de python 70
A3.2. Resolución de la cámara 70
A3.3. Problemas de calibración 70
A3.4. Raspberry pide la contraseña una y otra vez al inicio 70
A3.5. Memoria insuficiente 71
A3.6. Problemas de conexión con el monitor 71
A4: Manual de usuario. 72
Bibliografía 75

1. Introducción

1.1. Objetivos

El objetivo principal de este trabajo es el desarrollo de un sistema de navegación que permita a un robot móvil rotar en torno a un objeto o punto de manera completamente autónoma. La información necesaria para ello se tomará de manera visual, a través de marcadores Aruco (1) (2) y utilizando el lenguaje de programación Python 3. Se considerarán dos casos de giro distintos: en torno a una figura con marcadores adheridos a sus caras verticales, y giro en torno a un único marcador colocado en el suelo.

Como objetivos secundarios se diseñarán programas que detecten los marcadores, calculen la posición y orientación del robot con respecto a ellos y lo guíen hasta una distancia introducida por el usuario. También se diseñará un control proporcional que calcule el error entre el radio de giro deseado y el radio de giro del robot en el momento para modificar la trayectoria del robot.

1.2. Trabajos previos

Este proyecto se encuentra dentro de la línea de trabajo de los PiRobots del departamento de informática e ingeniería de sistemas de la Escuela de ingeniería y arquitectura. A lo largo de los años, otros alumnos han utilizado este tipo de dispositivos para distintas aplicaciones que, en mayor o menor medida, se parecen a la de este trabajo, y que se han tomado como referencia.

En concreto, dos de ellos han servido de gran ayuda: *Sistema multi-robot para cobertura persistente* (3) y *Navegación de un robot móvil basada en odometría utilizando Encoder diferencial e IMU* (4). En un principio, se pensaba adaptar el control desarrollado en este último trabajo a la navegación orbital, pero al final se optó por realizar uno propio basado únicamente en la información recogida por la cámara, sin ayuda de sensores adicionales como el encoder.

1.3. Alcance

En este trabajo se han diseñado programas que permiten el giro controlado alrededor de un marcador o figura de manera autónoma por parte del robot móvil. Para llegar hasta ellos, se crearon programas de movimiento en línea recta y se estudió su funcionamiento sin control y sus propiedades.

Posteriormente, se implementó el control basado en visión mencionado en el apartado anterior y, con él, se diseñaron los programas de giro finales. Es decir, durante este trabajo se ha desarrollado, por un lado, un módulo de percepción y, por otro, uno de control en giro, que después se sometieron a evaluaciones experimentales. Cualquier aplicación adicional, como el control en trayectorias rectas, quedan fuera del alcance de este trabajo.

1.4. Metodología y cronología

Todo el tiempo empleado en este proyecto se puede resumir en tres etapas diferentes:

- Búsqueda, instalación y puesta en marcha del software necesario. Familiarización con el hardware.
- Familiarización con las herramientas a utilizar y creación de los primeros programas. Resolución de problemas no tenidos en cuenta en la primera etapa.
- Programación del controlador y creación de los programas objetivo.

La primera etapa se caracterizó por la búsqueda de información y la planificación de los pasos a seguir. Se tomó contacto con el equipo de trabajo y se reunieron las herramientas que serían utilizadas posteriormente. Fue, probablemente, la etapa más costosa debido a la inexperiencia y a los distintos problemas encontrados[1] a lo largo de la puesta a punto.

Durante la segunda etapa se comenzó a trabajar con los marcadores y las funciones necesarias para el procesamiento de la información obtenida. Hizo falta instalar módulos adicionales y trabajar con conceptos que no se tuvieron en cuenta en la primera fase, como los sistemas de referencia.

La etapa final consistió en programar e implementar el control diseñado, así como experimentar y ajustarlo de manera manual. La figura 1 muestra este proceso más en detalle:

Figura 1: Diagrama temporal del proceso

[1] Para más información, consultar el anexo 3.

1.5. Estructura de la memoria

Este documento se divide en seis apartados, más los anexos. El actual apartado uno hace las veces de introducción. El apartado dos presenta las herramientas utilizadas; en él se detalla el hardware y software a utilizar, así como algunos aspectos a tener en consideración en cuanto a la manera de utilizarlos. En el apartado tres se muestran los sistemas de referencia utilizados a lo largo del trabajo y su método de obtención. En el apartado cuatro, se expone la implementación de aquellos programas que no utilizaron control y, en el cinco, se expone la implementación de aquellos que sí utilizaron. También se comenta el funcionamiento del control. Finalmente, el apartado seis sirve como conclusión.

En los anexos se podrá encontrar el código de los programas y comentarios acerca de aspectos concretos del uso del robot, como la calibración de la cámara, una lista de errores encontrados y un pequeño manual para futuros usuarios.

2. Herramientas utilizadas

2.1. Pi Robot

Es el elemento principal de este proyecto. Se compone, a su vez, de las partes numeradas en las figuras 2 y 3, desarrolladas más adelante:

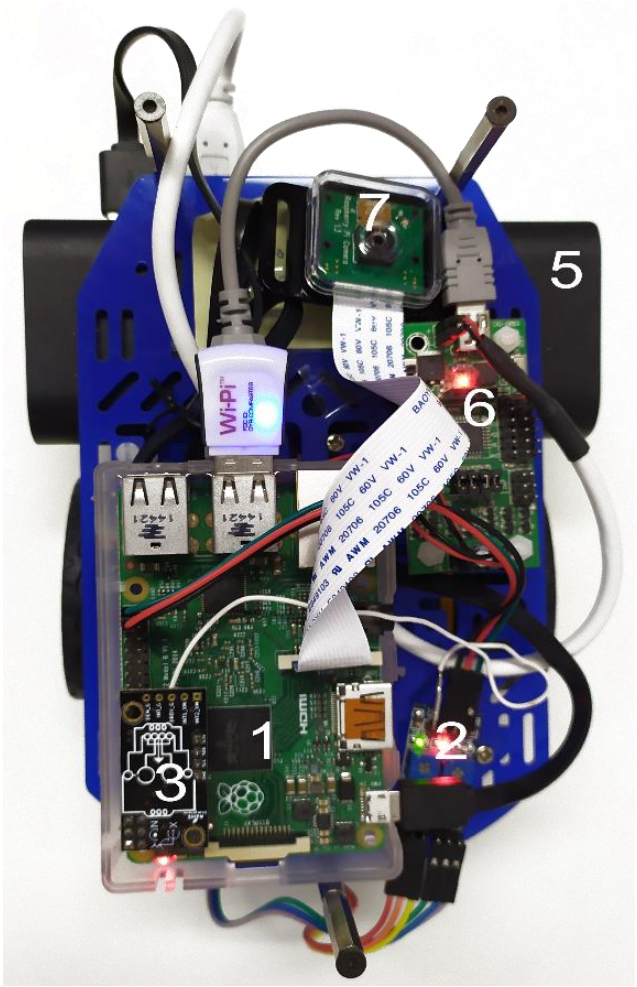


Figura 3: PiRobot visto desde arriba

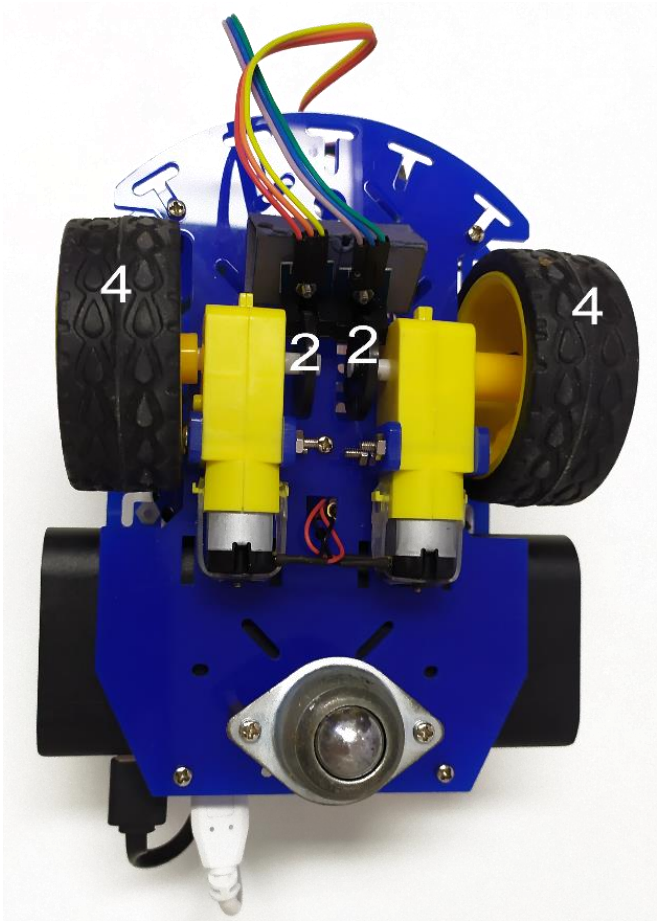


Figura 2: PiRobot visto desde abajo

2.1.1. Raspberry Pi modelo B, V1

Es un ordenador de placa única, es decir, se encuentra integrado en un solo circuito. Cuenta con:

- Cuatro puertos USB, a los que se pueden conectar periféricos como ratones o teclados. Inicialmente, se empezó el proyecto ya con dos de ellos ocupados por un adaptador de red USB (Wi-Pi) que permitirá la conexión wifi del robot y un puerto de salida que conecta con el controlador de potencia Dagu.
- Un puerto HDMI, utilizado para conectar un monitor.
- Un puerto micro USB utilizado para conectar la alimentación.
- Cuarenta pines GPIO (entradas/salidas de propósito general), que se pueden programar para recibir y transmitir información a los sensores.

El sistema operativo de este modelo es Raspbian Jessie (V8), basado en Debian. Incluye de serie un conjunto de software optimizado para Raspberry Pi, como Python 2.7.9.

2.1.2. Encoder

Sirve para medir las revoluciones de las ruedas. No se utilizará en este proyecto.

2.1.3. IMU

También llamada unidad de medición inercial, es un dispositivo que mide la velocidad y las fuerzas gravitacionales. No se utilizará en este proyecto.

2.1.4. Ruedas

El robot viene equipado con dos pares de ruedas de 6.2 centímetros de diámetro, controladas por motores, y una rueda loca que sirve como apoyo.

2.1.5. Batería

Sirve para alimentar tanto a los motores como a la Raspberry Pi.

2.1.6. Dagu mini driver

Se encarga de aumentar el voltaje de las señales mandadas por la Raspberry Pi a los motores para su correcto funcionamiento.

2.1.7. Picamera

Es la cámara utilizada para tomar imágenes. Se colocó en tres configuraciones distintas, dependiendo de la aplicación deseada:

- Configuración 1 (Figura 4): en la parte delantera del robot, para programas que requieran localizar una marca y avanzar hasta situarse a cierta distancia.
- Configuración 2 (Figura 5): Adherida a la batería, en uno de los laterales. Se encuentra ligeramente inclinada hacia arriba para mejorar su ángulo de visión. Utilizada para programas marcadores verticales.
- Configuración 3 (Figura 6): En un soporte elevado, en uno de los laterales del robot. Se encuentra inclinada hacia el suelo para mejorar la visión de las marcas horizontales.

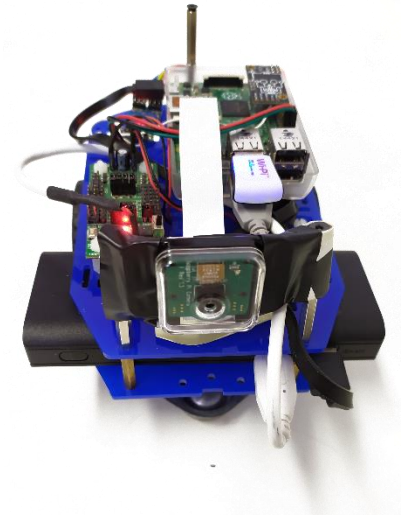


Figura 4: Configuración 1

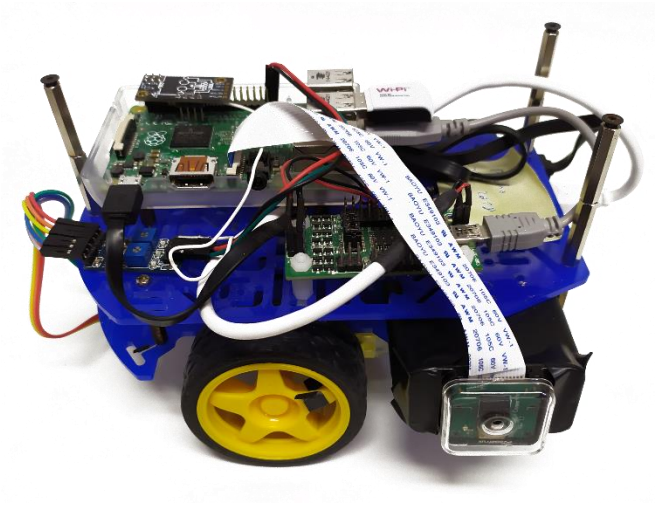


Figura 5: Configuración 2

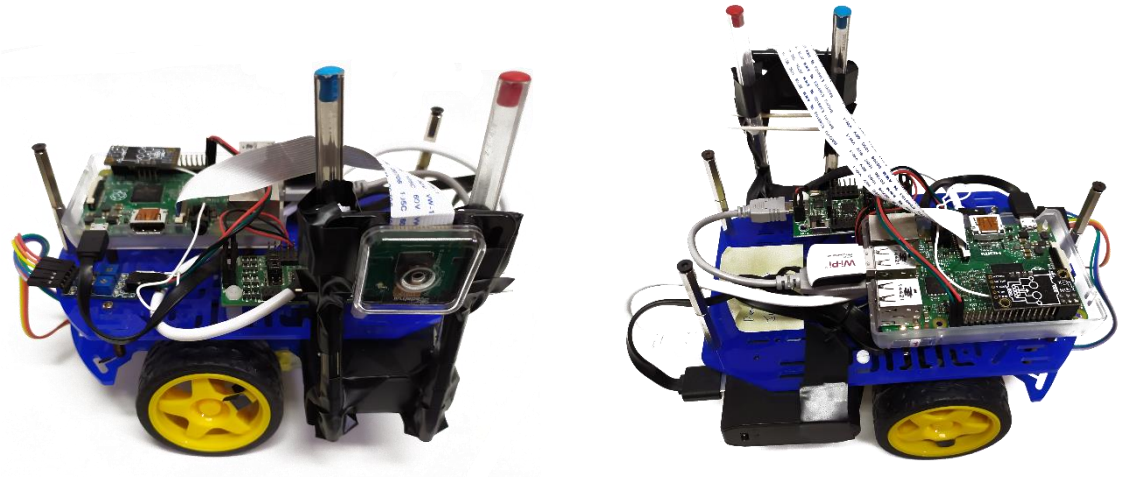


Figura 6: Configuración 3 vista desde frente y desde atrás

2.2. Python

Es un lenguaje de programación interpretado creado a finales de los ochenta. Aunque la versión más reciente es la 3.8.3, en este proyecto se utilizó la 3.4.2 para la creación de todos los programas.

Además, cuenta con módulos, que son conjuntos de funciones que se pueden importar y ejecutar. Para esta aplicación, los módulos utilizados fueron:

- **numpy**: permite realizar operaciones con matrices y vectores en Python. Suele abreviarse nada más ser importado como *np*.
- **math**: permite realizar operaciones matemáticas, como senos, cosenos, tangentes, arcotangentes…
- **cv2**: es el módulo de OpenCV para Python. Se utilizó para mostrar las imágenes captadas por la cámara en la pantalla.

- **cv2.aruco**: abreviado como *aruco* al ser importado, es el módulo que permite trabajar con las funciones de los marcadores Aruco. Probablemente es el más utilizado, ya que se encarga de detectar y procesar las imágenes.
- **time**: Como su nombre indica, sirve para realizar operaciones con tiempo. Por ejemplo, esperar un número determinado de segundos.
- **picamera**: para todo lo relacionado con la cámara y la obtención de imágenes.
- **RPi.GPIO**: controla los pines generales de entrada/salida
- **serial**: permite leer y escribir información en los puertos.
- **sys**: para editar parámetros específicos del sistema

2.3. Marcadores Aruco

Aruco es una librería de código abierto diseñada para detectar marcadores dentro de imágenes. Fue inicialmente desarrollada por investigadores de la universidad de Córdoba (1) (2), y permite, además, calcular la posición de la cámara respecto a las marcas.

Existen distintos tipos de marcadores, dependiendo del diccionario al que pertenezcan. En este proyecto se utilizó un diccionario Aruco, que contiene marcadores como el expuesto en la figura 7.

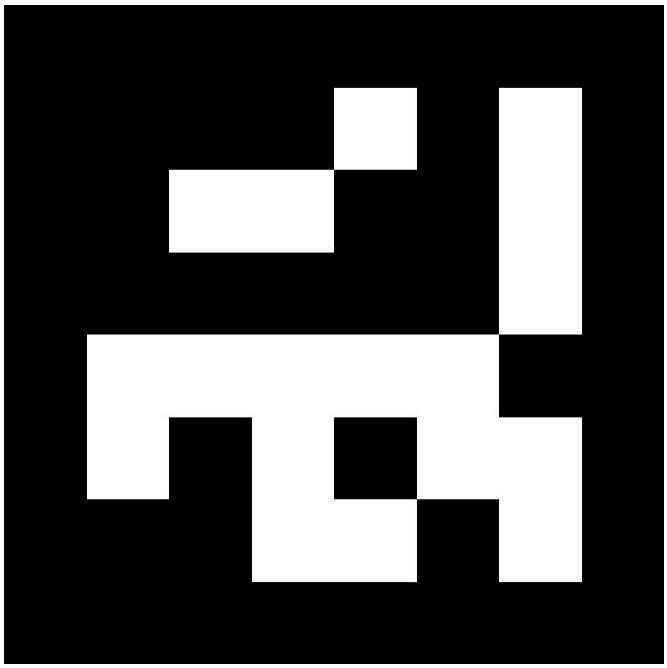


Figura 7: Marcador Aruco, id 2.

Como se puede observar, están constituidos por un borde negro externo dentro del cual hay un patrón binario de cuadrados más pequeños. Este patrón es único para cada marcador, lo que sirve para distinguir unos de otros. A mayor cantidad de cuadrados, más bits, y menor probabilidad de confusión. Sin embargo, también aumenta la resolución necesaria para detectarlos, por lo que se debe escoger el tamaño adecuado para cada cámara. Para este proyecto se utilizaron marcadores de 6x6 bits, del diccionario DICT_6X6_250. Estos diccionarios se encuentran disponibles en *Sourceforge*. (5)

Aunque no se aprecia en la figura, todas las marcas deben contar con un recuadro blanco exterior para ser reconocidas por la cámara.

A la hora de programar, cada marcador queda representado por un vector de cuatro puntos, cada uno de los cuales representa una esquina; un número de identidad o *id*, su tamaño en metros y los parámetros de rotación y traslación que relacionan su posición con la de la cámara.

Además, cada marca tiene asociado un sistema de referencia con origen en su centro geométrico, tal y como se explica en el apartado “Sistemas de referencia”.

2.4. OpenCV

OpenCV (Open Computer Vision) es una librería de software libre dedicada a la visión artificial. Cuenta con más de 2500 algoritmos orientados a reconocer formas, objetos e incluso facciones a partir de imágenes.

En este proyecto se ha utilizado su versión 3.4.0 para reconocer los marcadores Aruco.

2.5. Entorno virtual

Un entorno virtual es una zona de trabajo aislada del resto del equipo que cuenta con sus propios paquetes, módulos y librerías. Así pues, permite trabajar con distintas versiones de una herramienta a la vez y dentro de un mismo equipo.

En este proyecto en concreto, se creó un entorno de trabajo que utiliza la versión de Python 3.4.2[2] Dentro del mismo, se instaló OpenCV[3] y todos los módulos y paquetes necesarios para el funcionamiento de los programas.

Para activarlo, se deben teclear en la terminal los comandos de la figura 8:

1. source ~/.profile
2. workon cv

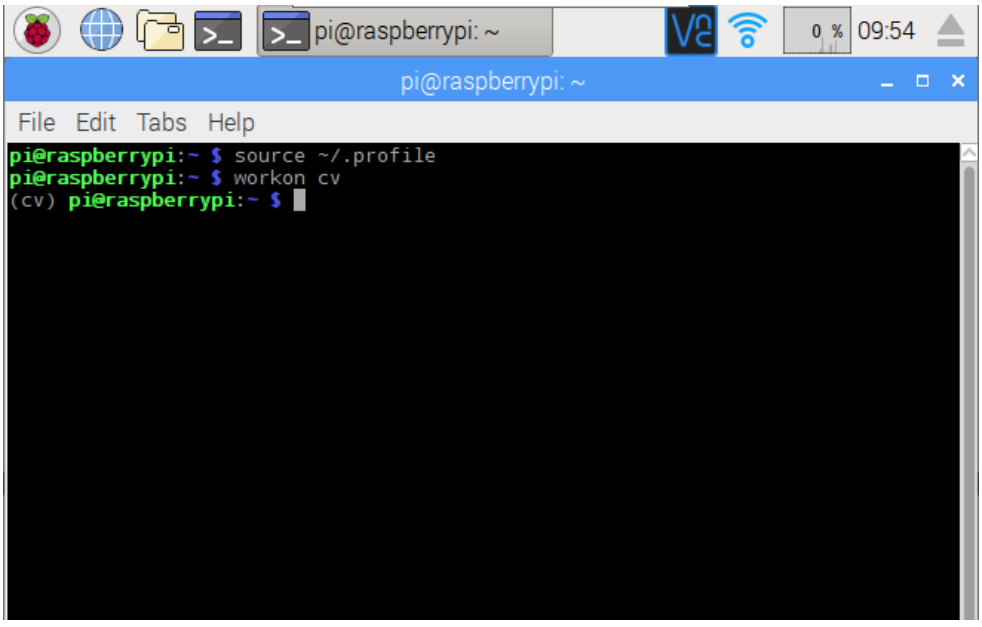


Figura 8: Activación del entorno virtual. El símbolo (cv) indica que se está trabajando dentro del mismo.

[2] Para más información sobre el entorno virtual, ir al anexo *problemas encontrados*.
[3] Para la instalación del entorno y de OpenCV se siguió el tutorial de Pyimagesearch (8).

2.6. VNC

Virtual Network Computing (VNC) es un programa de software libre que permite compartir pantalla y controlar un ordenador remotamente desde otro equipo. En este caso, se utilizó para controlar el robot sin necesidad de conectarle un monitor u otros periféricos, permitiendo así su libre movimiento por la sala de trabajo.

2.7. Prisma hexagonal

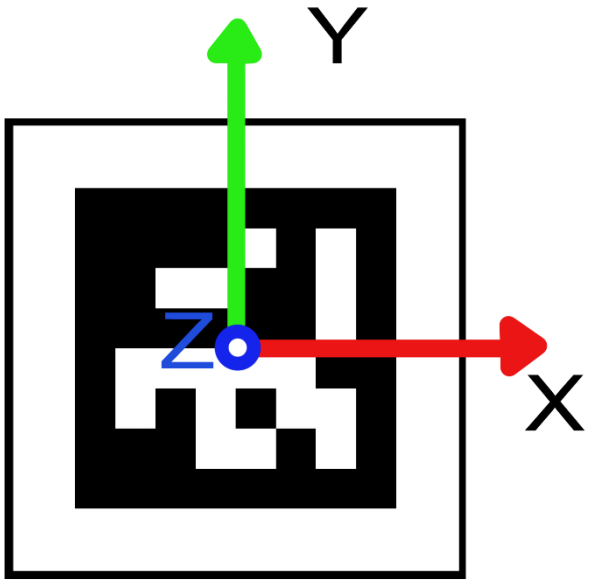
Es la figura alrededor de la cual el robot gira en los ejercicios con la cámara en la segunda configuración. Tiene marcadores con distintos *ids* adheridos a cada una de sus caras, ordenados de manera que, al girar en sentido contrario a las agujas del reloj, los *ids* van aumentando de uno en uno de 0 a 5. Mide 31 centímetros de alto, y sus bases tienen una apotema de 13 centímetros y un lado de 15, tal y como muestra la figura 9:

Figura 9: Prisma hexagonal

3. Sistemas de referencia

OpenCV y Aruco permiten conocer los vectores de posición y rotación de la cámara con respecto a la marca, y viceversa. A partir de ellos se pueden obtener las matrices de transformación entre un sistema de referencia y otro fácilmente[4].

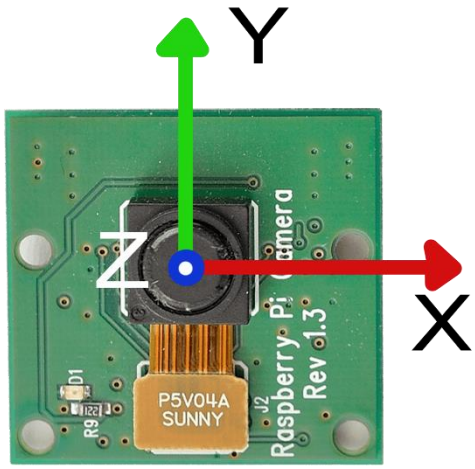
3.1. Sistema de referencia marcador



Cada marcador cuenta con un sistema de referencia propio con origen en su centro geométrico. El eje Z, normalmente representado en azul, es perpendicular al plano del marcador. El eje Y, por otro lado, suele representarse en color verde y va en dirección perpendicular a éste y perpendicular al suelo, cuando el marcador se coloca verticalmente. Por último, el eje X es perpendicular a los anteriores y paralelo al plano del suelo cuando la marca se coloca verticalmente.[5] La figura 10 representa este sistema:

Figura 10: Sistema de referencia Marcador

3.2. Sistema de referencia cámara



El sistema de referencia de la cámara viene determinado de serie. El origen se encuentra en su centro óptico, y el eje Z se coloca de tal manera que "sale" de la cámara. Es decir, mide la profundidad, o la distancia, que hay entre el origen del sistema de la cámara y cualquier objeto que se coloque delante. El resto de ejes pueden observarse en la figura 11.

Figura 11: Sistema de referencia Cámara

[4] Para más información, ir a la sección 4.3. *Ref.py*
[5] Para más información, consultar la documentación de Aruco (9)

3.3. Sistema de referencia robot

Como ya se ha comentado, el soporte de la cámara tuvo que inclinarse en mayor o menor medida para ampliar su ángulo de visión. Esto ocasionó una rotación de los ejes de la cámara alrededor de su eje X, que provocó que el eje Z de la cámara dejara de ser paralelo al suelo. Aunque esto no impide la medición de los datos de localización de la cámara con respecto al marcador, los dificulta en gran medida, por lo que se optó por crear un nuevo sistema de referencia que forzara que el eje Y fuera perpendicular al plano del suelo y los ejes Z y X fueran paralelos al mismo: el sistema de referencia Robot.

Además de estas características, interesaba que el eje X del sistema fuera paralelo a la velocidad de movimiento, es decir, que fuera paralelo al eje longitudinal del robot, tal y como se indica en la figura 12:

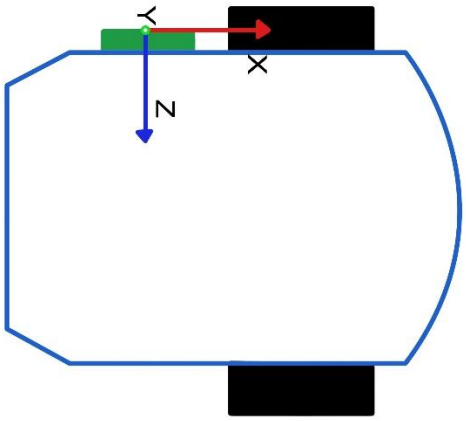


Figura 12: Sistema de referencia Robot

El origen de coordenadas de este sistema podría haberse colocarse en cualquier punto del espacio con tal de que se moviera junto al robot; sin embargo, se colocó coincidiendo con el origen del sistema de referencia cámara para facilitar su obtención. Para lograr la orientación de los ejes deseada, se creó un programa[6] capaz de obtener las distintas matrices de traslación entre sistemas a partir de una fotografía, y de separar sus respectivas partes de rotación y de traslación.

Luego, se colocó un marcador en una pared vertical, con su eje X completamente paralelo al plano del suelo. Después se colocó al robot – con la cámara fija a una de sus superficies – con su dirección de movimiento lo más alineada posible a dicho eje y se sacó una fotografía, a partir de la cual el programa anteriormente mencionado “copió” los ejes del sistema de referencia Marcador al origen del sistema de referencia cámara. A partir de ahí, se obtuvo la matriz de transformación del sistema de referencia Cámara al sistema de referencia Robot, que permanece constante con el movimiento y puede exportarse a otros programas para, así, tener todos los sistemas de referencia ya definidos.

Esta matriz cambia con cada configuración nueva de la cámara, por lo que tras cada ajuste de su posición o cambio de soporte hay que repetir todo el proceso.

[6] Ref.py, cuyo código y se encuentra completo en el anexo y cuyo funcionamiento se explica en el apartado 4.3.

La figura 13 muestra el montaje llevado a cabo para obtener el sistema de referencia robot:

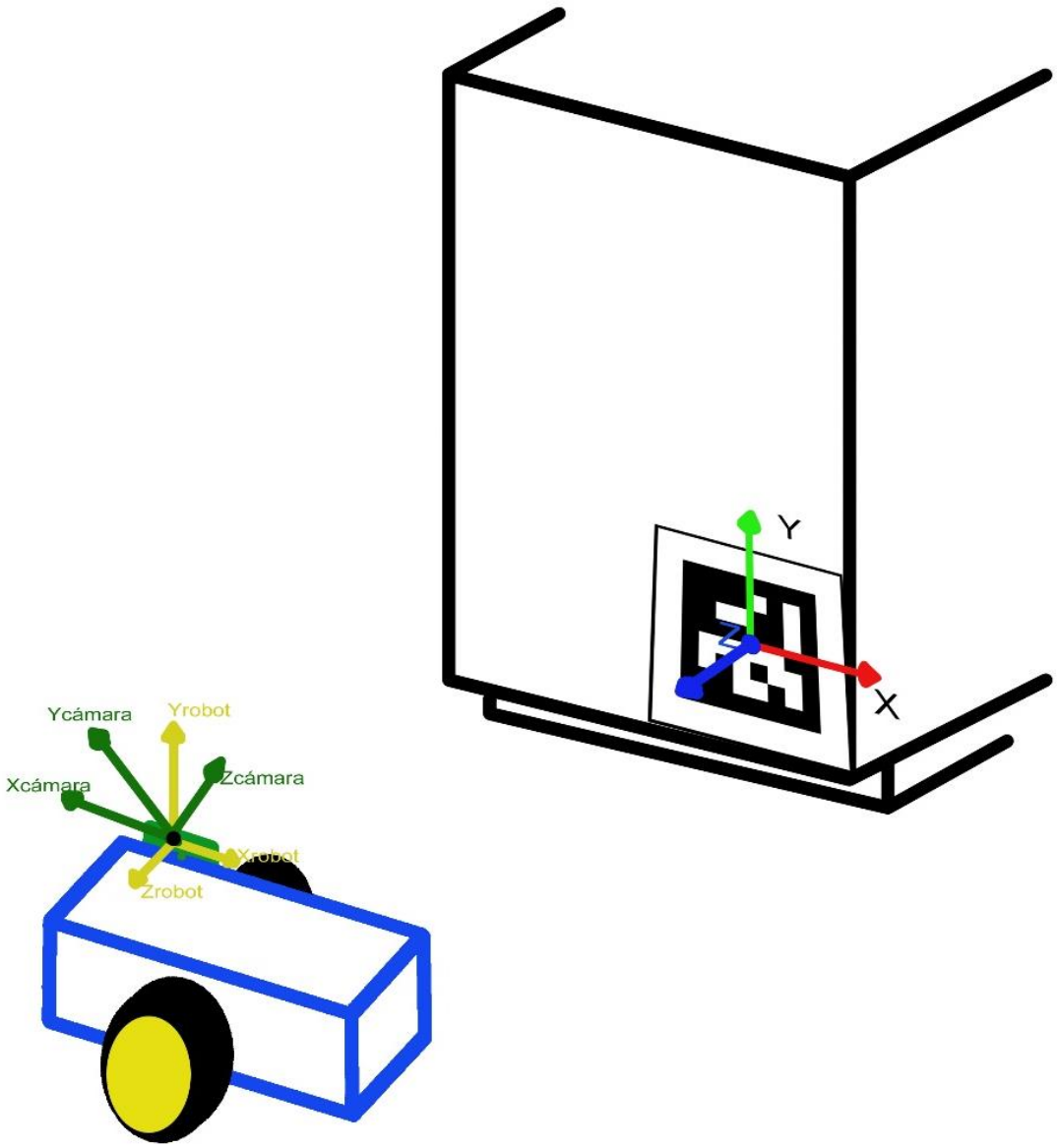


Figura 13: Montaje para la obtención del sistema de referencia Robot

4. Implementación del código sin control

En esta sección se introducen las primeras aplicaciones programadas, que funcionan sin realimentación. También se detallará el funcionamiento del programa utilizado para obtener los sistemas de referencia.

4.1. Detección de marcadores y proyección de resultados

Los primeros programas realizados se centraron en la localización de las marcas y la proyección de los resultados por pantalla. Se basan en la captación continua de imágenes de la cámara y en el análisis de cada una de ellas en busca de marcas reconocibles. Gracias a las funciones del módulo aruco de Opencv, es posible devolver por pantalla la misma imagen con los marcadores recuadrados y sus ejes dibujados.

De todos ellos, el ejemplo más relevante es el programa DetectAxis.py, cuyo código completo se puede encontrar en los anexos.

El procedimiento de búsqueda de marcadores, dibujado y obtención de parámetros se lleva a cabo mediante funciones del módulo Aruco de OpenCV:

- aruco.detectMarkers(image, diccionario): detecta los marcadores de cierto diccionario en una imagen.
- aruco.estimatePoseSingleMarkers(esquinas, longMarca , cameraMatrix, distCoeffs): una vez detectados los marcadores de una imagen, estima la pose de los mismos a partir de sus esquinas, la longitud de su lado y los parámetros de calibración de la cámara. Devuelve los vectores de rotación y traslación.
- aruco.drawDetectedMarkers(image, esquinas, ids,4): dibuja un recuadro alrededor de los marcadores detectados en una imagen a partir de sus esquinas y su id.
- aruco.drawAxis(image, cameraMatrix, distCoeffs, rvec[i], tvec[i], 0.05): dibuja los ejes de un marcador a partir de una imagen, los parámetros de calibración de la cámara y los vectores de rotación y traslación del marcador encontrado.

En la figura 16 se puede observar el marcador, recuadrado en verde, con sus ejes representados: X en rojo, Y en verde y Z en azul. También se dibuja el número o id del marcador, en este caso 2.

Al cambiar la posición de la cámara, tanto el recuadro como los ejes cambian con ella, de manera que Z siempre se dirige hacia el exterior del marcador.

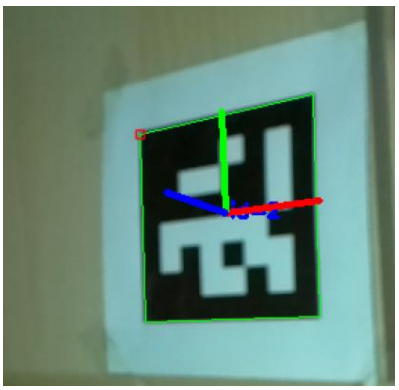


Figura 14: Marcador detectado

4.2. Movimientos en línea recta

Tras la creación de programas de detección de marcadores se procedió a estudiar el movimiento del robot. El programa resultante fue VeAMarca3.py[7], que combina la detección de marcas con movimientos sencillos para llevar al robot hasta un marcador en línea recta, utilizando la configuración 1 de la cámara. Como la cámara no está inclinada, en este caso solamente se necesita saber la distancia en su eje Z, que coincidirá con la distancia del robot al marcador. Este parámetro se obtiene directamente del vector de traslación que devuelve la función aruco.estimatePoseSingleMarkers, así que no es necesario obtener todos los sistemas de referencia explicados anteriormente, y el funcionamiento se simplifica.

Aunque la precisión de la cámara en esta posición es relativamente buena, la ausencia de controlador hace que las rectas se desvíen. Si se coloca el robot inicialmente a una distancia lo suficientemente alejada del marcador, tras iniciar el avance es posible que pierda de vista la marca y necesite volver a girar sobre sí mismo para localizar el marcador de nuevo.

Las siguientes gráficas (Figuras 17 y 18) muestran la distancia lateral al marcador vista desde el punto de vista de la cámara, de manera que el eje X positivo apunta a la izquierda del robot y el eje X negativo, hacia su derecha. En ambas, la distancia recorrida es de 78 centímetros, pero la primera lo hace a una velocidad de 12.5 cm/s y la segunda, al doble. Al inicio, la cámara se colocó alrededor de 8 centímetros a la izquierda del marcador, mirando de frente al mismo para evitar giros a mitad de trayectoria. Se tomaron los datos de distancias en el eje X a lo largo de la ejecución del programa y se representaron en función de la distancia avanzada.

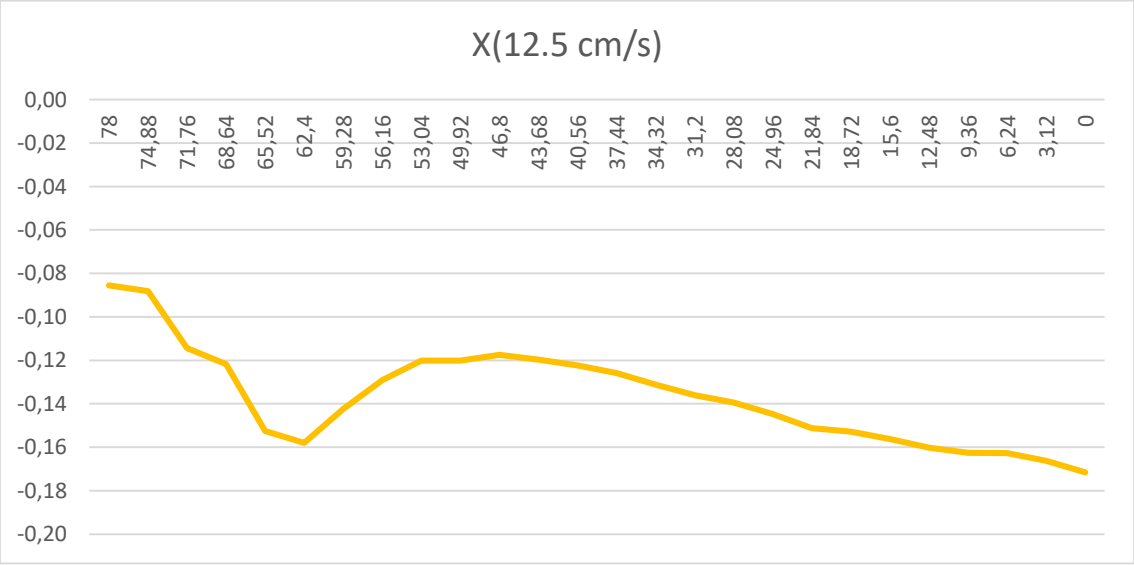


Figura 15: Desviación en el eje X(centímetros) respecto al avance en Z (metros), 12.5 cm/s, recorrido corto

[7] Existe un vídeo de este programa, que se podrá encontrar en el departamento de Informática e ingeniería de sistemas de la Universidad de Zaragoza.

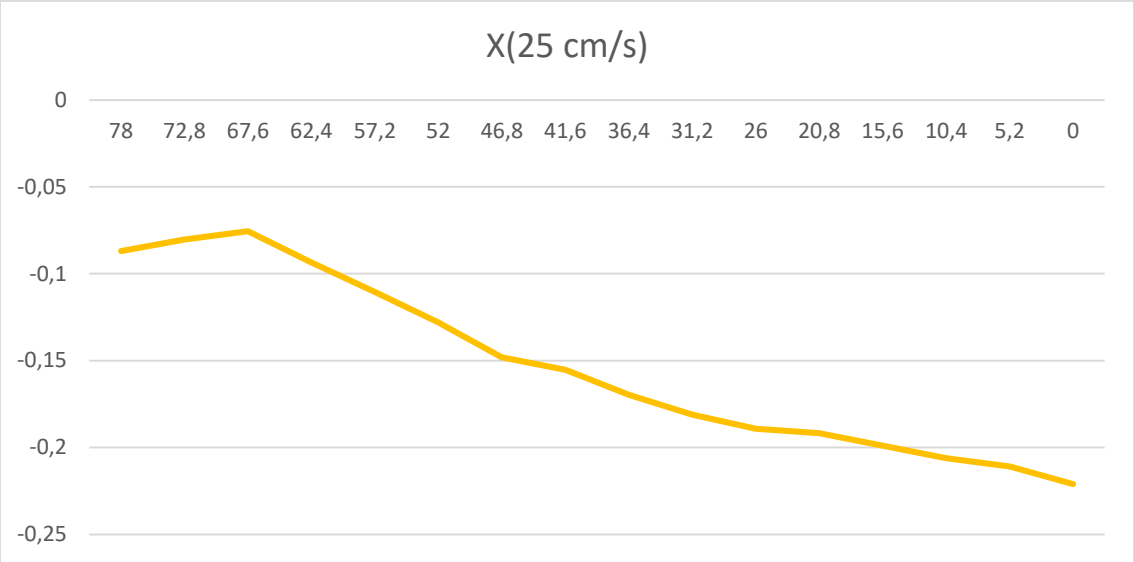


Figura 16: Desviación en el eje X(centímetros) respecto al avance en Z(metros), 25 cm/s, recorrido corto

En una gráfica que mostrase el funcionamiento ideal, la función debería ser y = -0,08 durante todo el recorrido. Sin embargo, en las dos gráficas la recta tiene una pendiente negativa, representando la desviación del robot hacia su derecha. En la primera, la distancia final de la cámara al marcador es de 17 centímetros, lo que supone una desviación total de 0,11 metros. En la segunda, la distancia final es de unos 22 centímetros, lo que supone una desviación total de 0,14 metros. En todos los experimentos llevados a cabo, la desviación es mayor en las ejecuciones a mayor velocidad. Esto se debe a que los impulsos que deben mandar los motores a las ruedas es mayor, propiciando picos de tensión y haciendo más notable la diferencia entre el funcionamiento de uno y otro.

También se aprecian las oscilaciones del robot –en este caso, mayores en la gráfica lenta – introducidas por las ruedas.

En trayectorias más largas, estas desviaciones provocan que el robot pierda de vista al marcador y tenga que girar sobre sí mismo para volver a encontrarlo y poder avanzar de nuevo hacia él. En la figura 19 se refleja este fenómeno mediante el cambio brusco del parámetro X. Las condiciones iniciales esta vez fueron ligeramente distintas; la marca se colocó diagonalmente a la cámara por lo que los valores de X no se corresponden directamente con la desviación. Además, el recorrido fue notablemente más largo: 118 centímetros. En esta gráfica se aprecian mejor los picos producidos por el paso de velocidad cero a velocidad 12.5 cm/s, tanto al inicio como después del giro.

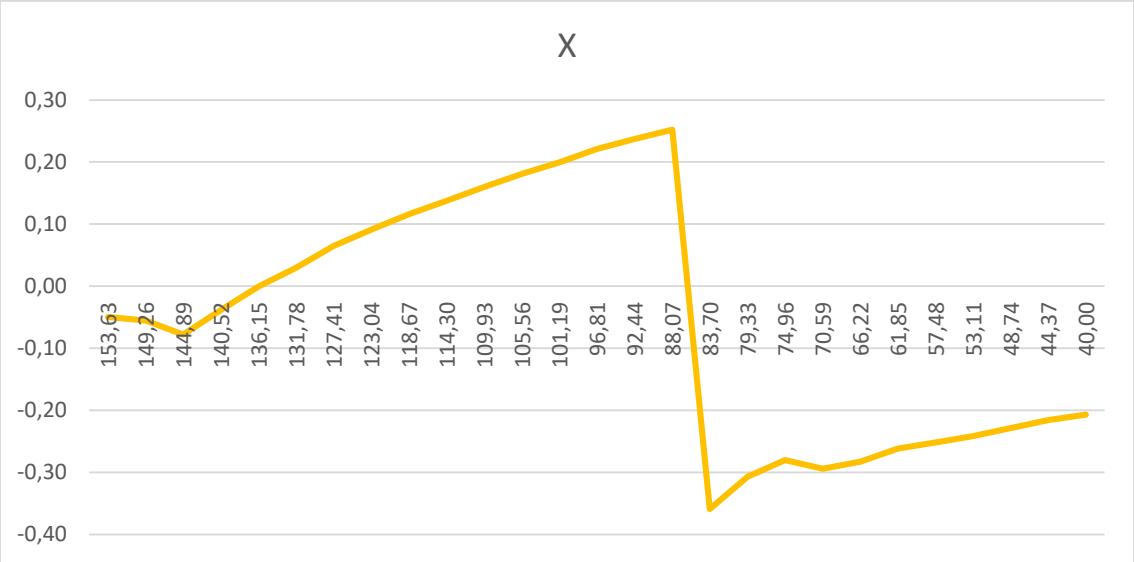


Figura 17: Desviación en X respecto al avance en Z, 12.5 cm/s

4.3. Obtención de sistemas de referencia

A continuación, se explica en detalle el fundamento de Ref.py, el programa a partir del cual se obtiene el sistema de referencia Robot. Tal y como se comentó en el apartado *Sistemas de referencia*, el objetivo de este sistema es hacer más fácil la lectura de la posición del robot. Tiene su origen de referencia en el mismo punto que el sistema *cámara*, pero sus ejes se encuentran girados de manera que el plano XZ queda completamente paralelo al suelo, y el eje X tenga la misma dirección que la velocidad del robot.

Para ello, primero es necesario realizar el montaje explicado en el apartado 3.3 y después pulsar la tecla “c”. Entonces el programa obtiene los vectores de rotación y traslación que relacionan la cámara y el marcador modelo. Con estos datos construye la matriz ${}^{C}T_{M}$, que representa el sistema de referencia Marcador con respecto al sistema de referencia Cámara. Invirtiendo esta matriz se consigue ${}^{M}T_{C}$, que representa el sistema de referencia de la cámara con respecto al sistema de referencia Marcador. Estas matrices constan de dos partes, representadas en la figura 20:

$$ {}^{A}\mathbf{T}_{\mathbf{H}} = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{n} & \mathbf{o} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0} & 1 \end{pmatrix} $$

Figura 18: matriz de transformación del sistema de referencia H con respecto al sistema de referencia A

Tal y como indica la figura, las columnas “n”, “o” y “a” definen la rotación de los ejes del sistema H con respecto a los ejes del sistema A, mientras que la columna “p” representa la traslación de su origen. La columna “n” indica la rotación del eje X de H según los ejes

de A; la columna “o” indica la rotación del eje Y de H según los ejes de A, y la columna “a” indica la rotación del eje Z de H respecto a los ejes de A.

Como el sistema de referencia Robot se caracteriza por compartir origen con el sistema Cámara y por presentar la misma orientación de ejes, se fuerza a que la matriz ${}^{M}T_{R}$ tenga la forma indicada en la figura 21:

$$\begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figura 19: Ejemplo de matriz ${}^{M}T_{R}$

Donde d_x, d_y y d_z son los parámetros de traslación de la matriz ${}^{M}T_{C}$, lo que hace que ambos orígenes coincidan con respecto a la marca, y donde los ejes de R no presentan rotación alguna respecto a los de M.

Esto se realiza separando los datos de traslación de la matriz ${}^{M}T_{C}$ anteriormente calculada y combinándolos con una matriz unidad. A partir de estos datos, el programa calcula la matriz ${}^{C}T_{R}$ deseada multiplicando ${}^{C}T_{M}*{}^{M}T_{R}$.

Finalmente, imprime los datos relevantes por pantalla.

4.4. Aplicación de los sistemas de referencia al movimiento

Para introducir los conceptos del apartado anterior en el movimiento del robot, se programó VeLat.py. Este fue un programa de aprendizaje y familiarización con los comandos de giro y sistemas de referencia, cuya misión es llevar el robot a una distancia determinada del marcador, en este caso con la cámara colocada en uno de sus laterales. Fue el primero en implementar el sistema de referencia robot, y el primero en el que fue necesario calcular el ángulo con respecto al marcador.

Su funcionamiento es parecido a VeAMarca3, pero con pasos adicionales. Inicialmente, hace girar al robot en torno a sí mismo hasta que localiza un marcador. Después, gira hasta alinear el eje X del robot con el de la marca, y avanza en línea recta hasta encontrarse en el centro de la marca. Recalcula la distancia y gira de nuevo hasta situarse perpendicular a ella para luego avanzar hasta llegar a la distancia especificada por el usuario. Entonces, gira una última vez para volver a colocarse paralelo al marcador.

Este programa podría haberse diseñado de forma más sencilla. Una vez localizada la marca, podría haberse calculado el ángulo a girar el robot para avanzar directamente hasta el centro del marcador y, después, haber girado de nuevo para situarlo paralelo. Sin embargo, de esta manera se trabajan mejor los comandos de giros y paradas, aunque el resultado es brusco y poco preciso.

4.5. Introducción del movimiento alrededor de una figura

El paso posterior al estudio de funcionamiento de los sistemas de referencia en movimiento fue aplicarlos a la detección de múltiples marcadores y al movimiento alrededor de los mismos. Para ello se creó GiraHex.py, cuyo objetivo es introducir el movimiento en torno al prisma hexagonal mediante líneas rectas y giros.

En teoría, el robot empieza detectando un marcador y moviéndose paralelamente a su eje x y, por extensión, a la cara del prisma. En cuanto detecta un marcador con un *id* mayor, avanza hasta colocarse a cierta distancia de su eje Z y luego se para y gira para colocarse paralelo a este nuevo marcador.

Así, se consigue rodear la figura de manera sencilla. Sin embargo, al no contar con realimentación, el robot tiende a desviarse hacia la figura más de la cuenta, lo que ocasiona pérdidas de visión de las marcas. Para evitarlo, se ajustó manualmente el tiempo de giro, de manera que se contrarrestara el movimiento lateral no deseado. El resultado es caótico y poco duradero, puesto que los ajustes no solucionan el problema.

El siguiente paso, por lo tanto, fue diseñar un control de giro sencillo.

5. Implementación del código con control

Tras comprobar las deficiencias presentadas por el giro en el apartado anterior, se decidió programar un control. En esta sección se exponen los fundamentos de diseño de las distintas aplicaciones que lo utilizan, así como los resultados obtenidos de las evaluaciones experimentales y las conclusiones de dichos experimentos[8].

El primer apartado describe las bases del control, mientras que los dos últimos se combinan los ejercicios anteriores con el control para conseguir los objetivos del trabajo.

5.1. Control

El control diseñado para sustituir al que se iba a utilizar en un principio es un control proporcional simple, que calcula el error entre el radio deseado y el radio en cada momento y aplica una constante de corrección. En la figura 14 se representan los radios teóricos que describe el robot en su trayectoria circular. En amarillo, se representa el círculo que describe la rueda derecha, sobre el que actúa el controlador. El mayor es el que describe la rueda izquierda, y el restante, el del robot. Se va a considerar que el radio obtenido mediante la medición de la posición de la cámara es igual al radio que describe la rueda derecha por su proximidad.

[8] Existen vídeos de los apartados 5.2 y 5.3, que estarán disponibles en el departamento de informática e ingeniería de sistemas de la Universidad de Zaragoza.

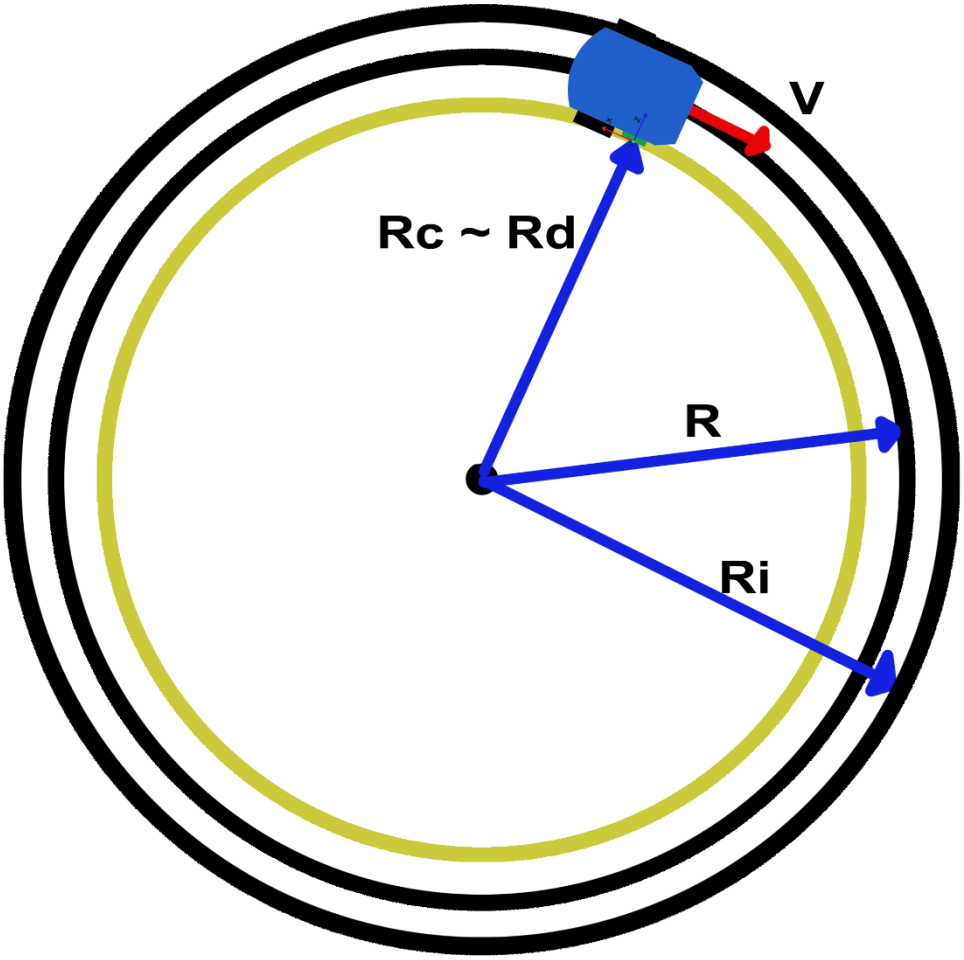


Figura 20: Esquema del robot y sus radios de giro

Para el cálculo del control primero se elige la velocidad lineal que debe llevar el punto medio de la rueda izquierda, que permanecerá constante, así como el radio medio deseado. Sabiendo la anchura del robot, L, se pueden calcular los radios a describir por parte de las ruedas derecha e izquierda. Partiendo de que la velocidad angular (w) en torno al punto central debe ser igual para todos los puntos del robot, dicha velocidad se puede calcular con el radio a describir por la rueda izquierda y la velocidad escogida:

$$w = Vi0/Ri$$

Una vez hallada la velocidad angular, se puede calcular la velocidad lineal que debe llevar el punto central de la rueda derecha, y obtener su velocidad lineal inicial, mediante la misma fórmula:

$$Vd0 = wRd$$

Esta velocidad será la velocidad inicial, que se irá modificando según sea necesario.

Una vez calculado el radio de giro de la cámara (y, por ende, el de la rueda derecha) se calcula el error:

$$error = Rd - rd(t)$$

Este dato, multiplicado por la constante de proporcionalidad, K, da como resultado la velocidad de control, que, sumada a la velocidad inicial de la rueda derecha, se transforma en la nueva velocidad a mandar al motor:

$$vcontrol = K * error$$

$$vd = vd0 + vcontrol$$

En este caso, el control se realiza sobre la velocidad lineal del punto medio de cada rueda, ya que es la que se puede medir con mayor facilidad. En caso de que se quisiera realizar sobre la velocidad angular de las ruedas, debería multiplicarse por el radio de la propia rueda.

El resultado de la aplicación de este control se puede apreciar en las ejecuciones de aquellos programas que lo implementan, en los cuales la trayectoria del robot oscila para mantenerse lo más próxima posible al radio deseado. En la figura 15 se puede observar al robot ampliando su radio de giro para ajustarse a la trayectoria deseada:

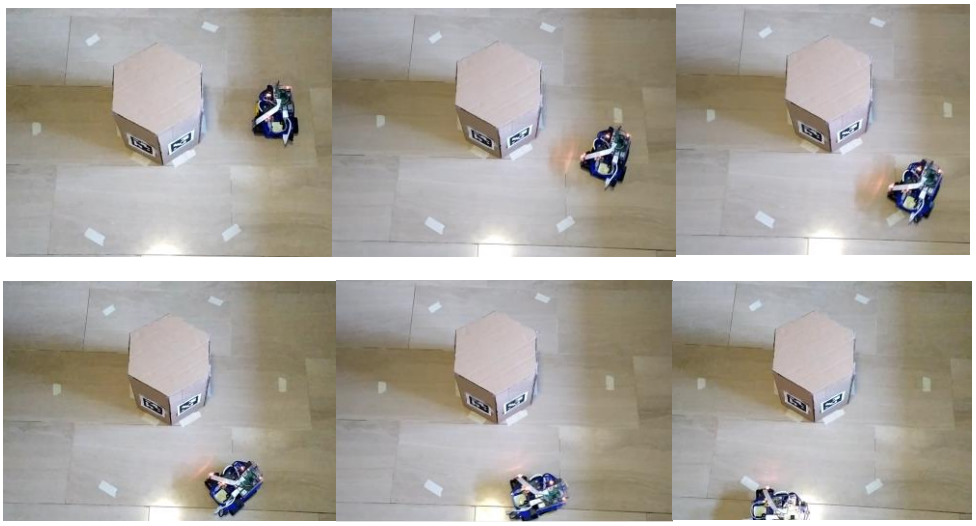


Figura 21: Robot ajustándose al radio de giro deseado (marcado con trozos de cinta)

Esta figura está constituida por fotogramas de uno de los vídeos tomados en un experimento. La circunferencia deseada está marcada en el suelo mediante cinta, y sirve solamente de referencia visual; el robot no utiliza esa información en ningún momento. En el centro se coloca el prisma con los marcadores en sus caras. En la primera foto, que corresponde con la posición inicial, el robot se encuentra demasiado cerca de la figura, por lo que en cuanto comienza el avance debe alejarse de la misma, girando al mismo tiempo. Conforme van transcurriendo las fotografías, se puede observar cómo el robot va ampliando su radio de giro hasta hacer coincidir el radio deseado con la rueda derecha, que es la que se controla. Una vez alcanzado dicho radio, el robot es capaz de mantenerse girando alrededor del prisma indefinidamente, aunque, debido a las perturbaciones introducidas por las ruedas y otros factores, no permanece demasiado tiempo a la distancia ideal de los marcadores. Por lo tanto, oscila alrededor del radio deseado en mayor o menor medida.

Aunque la habitación contaba con luz natural, se utilizó también una lámpara colgante cuyo foco se refleja en el suelo para intentar homogeneizar la iluminación de todas las caras del prisma.

5.2. Giro alrededor del prisma hexagonal

En esta etapa las rectas y giros de GiraHex.py se sustituyen por los movimientos circulares de MarcaV.py. Este programa utiliza realimentación para corregir las desviaciones del robot y así poder mantenerlo girando alrededor del prisma indefinidamente.

Para ello, se fija una velocidad en la rueda exterior, que no varía, y se va ajustando la velocidad de la rueda interior según la distancia a la marca en el momento y la distancia deseada. Si la velocidad calculada es demasiado baja o demasiado alta para la que puede proporcionar el motor, se ajusta al máximo o al mínimo.

En este caso no es necesario que los marcadores se encuentren ordenados con ids ascendentes o decrecientes, puesto que se mide la distancia al centro de la figura y no la distancia a los marcadores. Sea cual sea el marcador a partir del cual obtiene su distancia, el radio calculado será el mismo.

La figura 22 es un esquema de la obtención del radio instantáneo del robot. Las apotemas del hexágono se representan en naranja, el radio de la cámara en amarillo y los parámetros de traslación del robot con respecto al sistema de referencia de los marcadores colocados en las caras, en negro y gris.

Al medir la distancia a los marcadores se obtienen dz y dx (o dz’ y dx’). Como la apotema del hexágono permanece constante, se puede calcular el ángulo entre dz y el radio de la cámara para, posteriormente, obtener el radio de la cámara por trigonometría. Finalmente, para hallar el radio de giro de cada rueda en torno al centro del prisma, sólo hace falta sumar la distancia de cada rueda a la cámara.

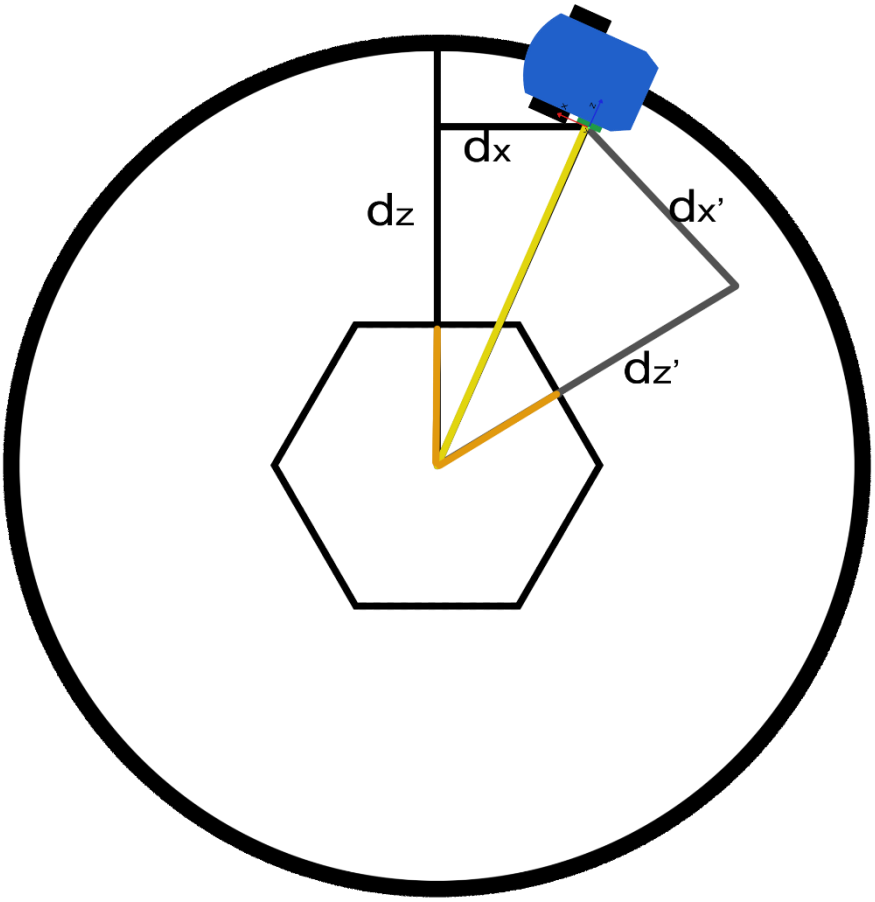


Figura 22: Obtención del radio de giro

Este sistema presenta ciertos problemas a la hora de la ejecución. En primer lugar, si hay fuentes de luz intensa en la zona de trabajo, se producen contrastes de iluminación entre marcadores que dificultan su detección y retrasan el cálculo del programa. También se pueden producir oscilaciones en la medida del radio producidas por la diferencia de posición entre los marcadores: aunque se midió dónde debían colocarse, es muy difícil que su centro coincida exactamente con la apotema de las caras, y que su altura sea siempre la misma. Además, la figura se hizo a mano y no es perfecta; sus dimensiones pueden variar. Por último, las medidas de los parámetros de posición y orientación no siempre son fiables. Tal y como se observa en la figura 23, que representa el radio del robot al girar alrededor del marcador, de vez en cuando se registran saltos de radio demasiado grandes y rápidos para ser reales:

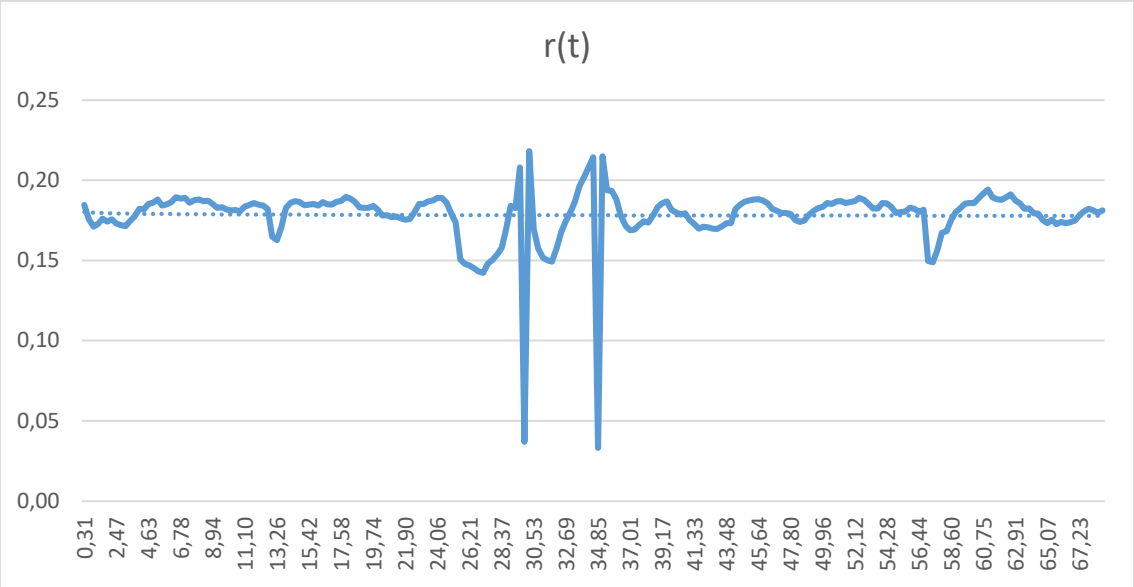


Figura 23: Representación del radio del robot (m) con respecto al tiempo (segundos) en giro.

A partir de ahora, todas las gráficas presentadas excluirán los radios espurios.

En la figura 24 se representa el radio con respecto al tiempo registrado cuando el robot se coloca inicialmente cerca del radio deseado de la marca. La línea naranja marca el radio deseado. Se puede observar que, si bien la línea de tendencia se acerca cada vez más al radio deseado, los radios a lo largo del tiempo no se estabilizan. Este hecho es imposible, puesto que el robot se encuentra constantemente sometido a perturbaciones: tanto por parte de los motores, que no funcionan de manera uniforme, como por parte del suelo, ya que se pueden producir deslizamientos o puede haber obstáculos en el trayecto.

Una vuelta al prisma lleva, aproximadamente, un tiempo entre 20 y 40 segundos. Los cambios bruscos de radio se deben a ajustes de la trayectoria repentinos por parte del control.

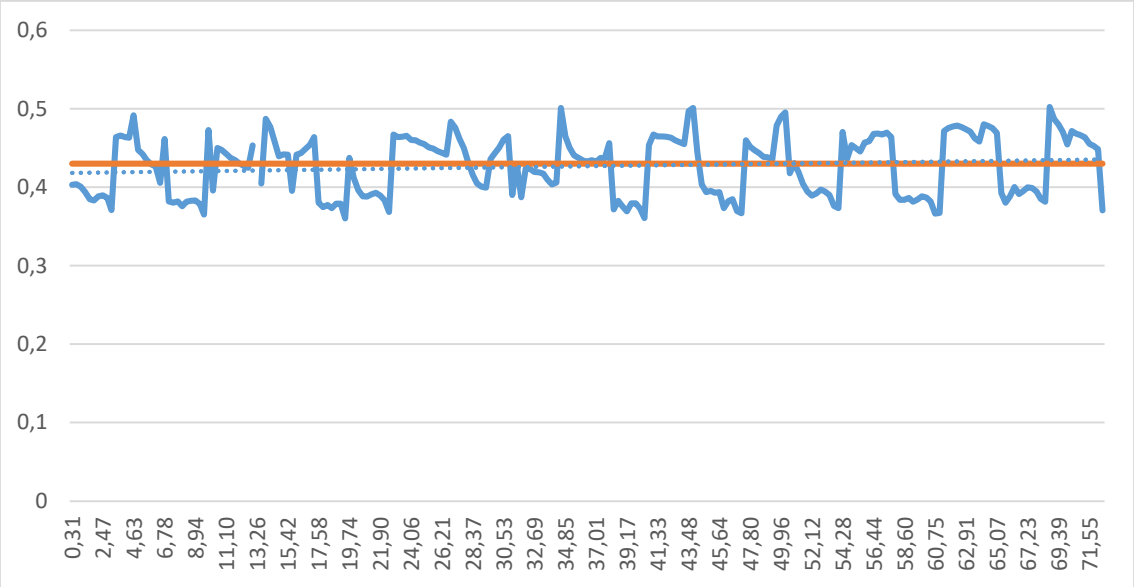


Figura 24: Radio del robot (m) con respecto al tiempo (s), en azul, con su línea de tendencia (puntos) y radio deseado (naranja).

Las siguientes gráficas (figuras 25 y 26) muestran los mismos parámetros que la anterior, pero con posiciones iniciales diferentes. En la primera, el robot se encontraba demasiado lejos de la figura, mientras que en la segunda se encontraba demasiado cerca. En ambos casos el radio va disminuyendo –o aumentando – hasta que su tendencia es igual al radio deseado:

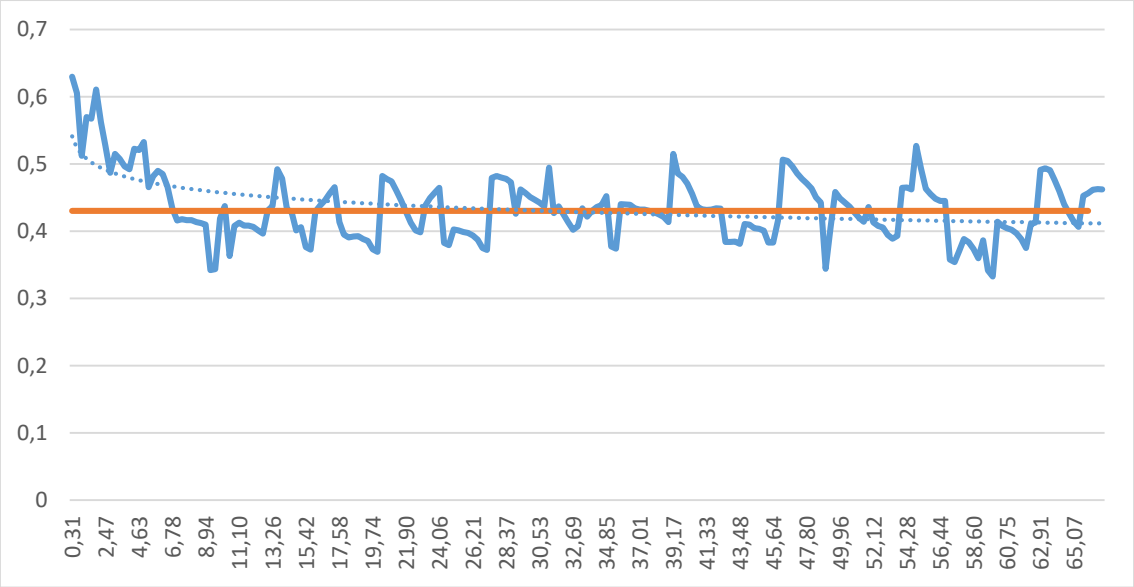


Figura 25: Radio medio del robot (m) con respecto a tiempo (s), radio inicial mayor al deseado. La línea de puntos marca la tendencia y la naranja el radio deseado.

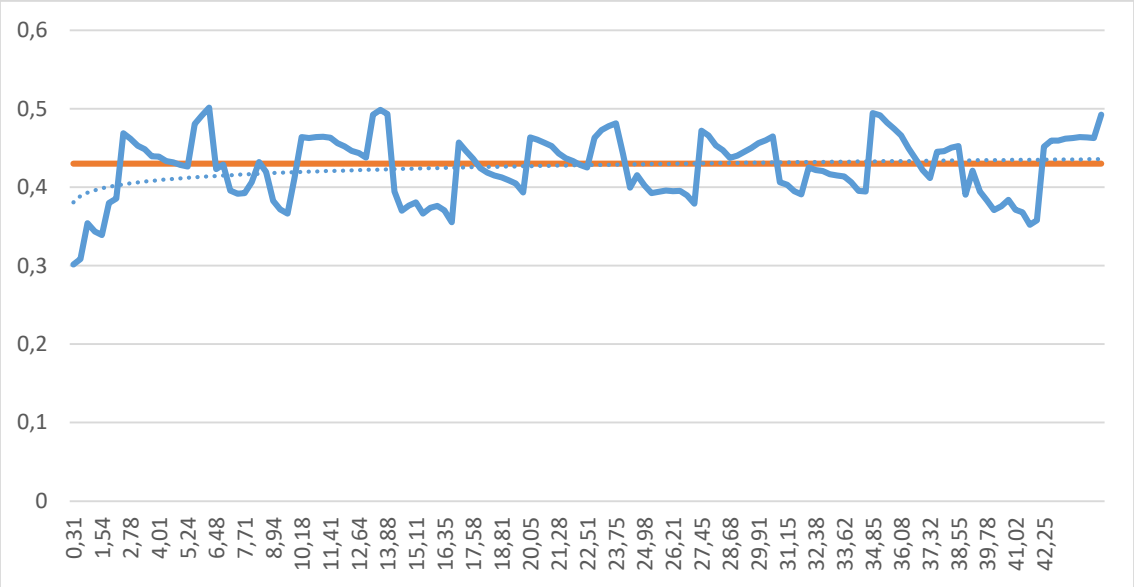


Figura 26: Radio medio del robot (m) con respecto al tiempo (s), radio inicial menor al deseado. La línea de puntos marca la tendencia y la naranja el radio deseado.

La figura 27 muestra una serie de fotografías extraídas del vídeo de una ejecución de este código. Concretamente, la correspondiente a la gráfica 23, en la que el robot se coloca lejos del radio deseado y debe hacer maniobras de aproximamiento:

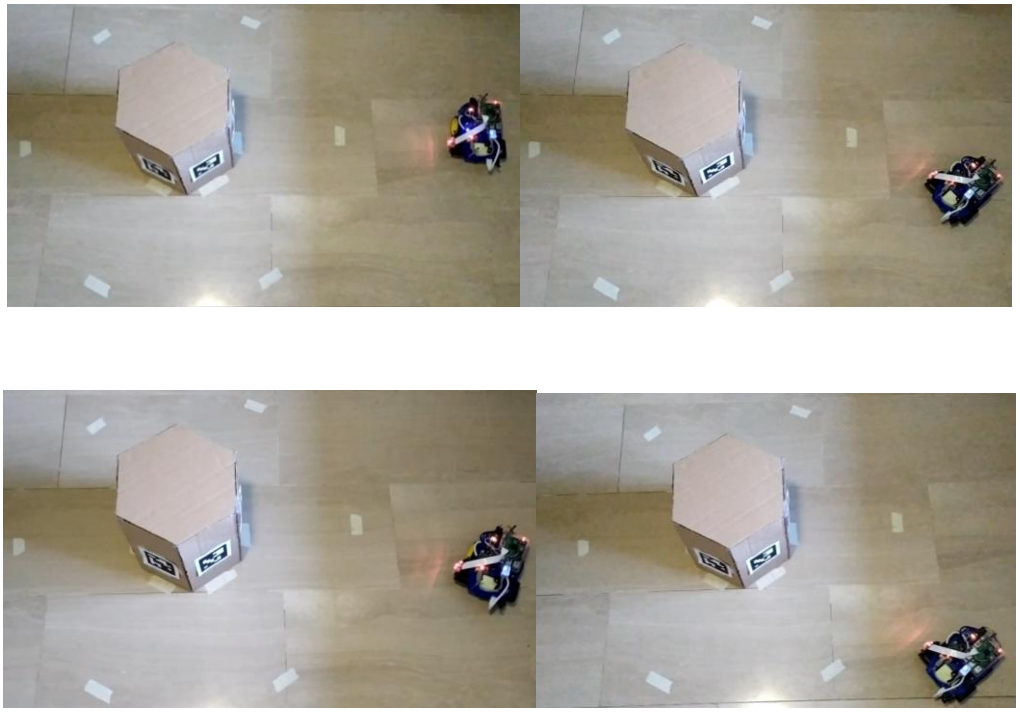


Figura 27: Maniobras del robot para ajustarse al radio deseado

Este experimento es similar al mostrado anteriormente, con la diferencia de que el robot se encuentra más lejos de lo deseado en la situación inicial. En este caso, el robot pierde

de vista los marcadores poco tiempo después de comenzar a moverse, debido a que debe cerrar demasiado el radio. Tras esa situación, se detiene y maniobra dando marcha atrás y marcha adelante hasta que alcanza un punto en el cual al girar no pierde de vista el marcador, y puede ajustar su trayectoria conforme se mueve. A veces, esto da lugar a giros en torno a sí mismo y otros problemas de ajuste que acaban solucionándose cuando vuelve a tener en el punto de mira al marcador.

5.3. Giro alrededor de un marcador horizontal

Finalmente, se intentó sustituir la figura por un único marcador colocado en el suelo. El programa resultante fue MarcaH.py, para el cual se necesita ajustar la cámara a la configuración 3, elevándola todo lo posible e inclinándola hacia el suelo.

El mayor problema de este sistema es la tendencia a dejar de detectar el marcador, ya que, si la cámara se encuentra demasiado cerca, no se captura entero en una imagen y, si se aleja demasiado, se deforma tanto que no es reconocible. Como resultado, el radio de giro en torno a la marca no puede variar tanto como puede hacerlo si se utilizan marcadores verticales. Esto podría mejorarse aumentando la longitud del cable de la cámara para situarla más alta, o utilizando una con mayor resolución.

Por otro lado, los problemas de iluminación que presentaba el giro alrededor de marcadores verticales desaparece, ya que no hay contrastes fuertes entre medidas. También desaparecen las oscilaciones debidas a cambios de marcador, por lo que el radio es más estable que en el caso de los marcadores verticales, tal y como indican las gráficas de las figuras 28 y 29:

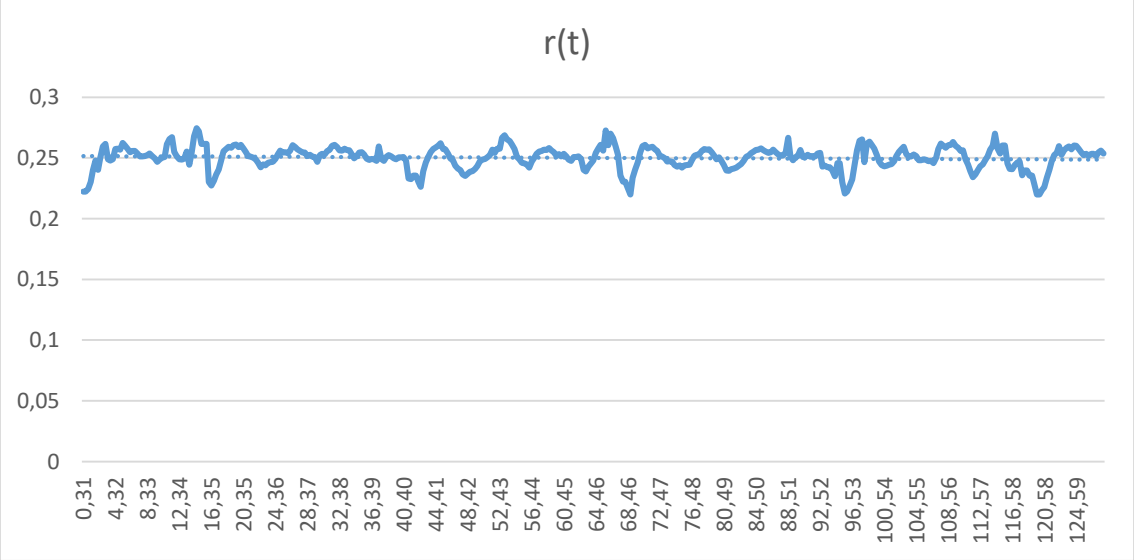


Figura 28: Radio del robot con respecto al tiempo. El eje y muestra el radio en metros y el eje x, el tiempo en segundos.

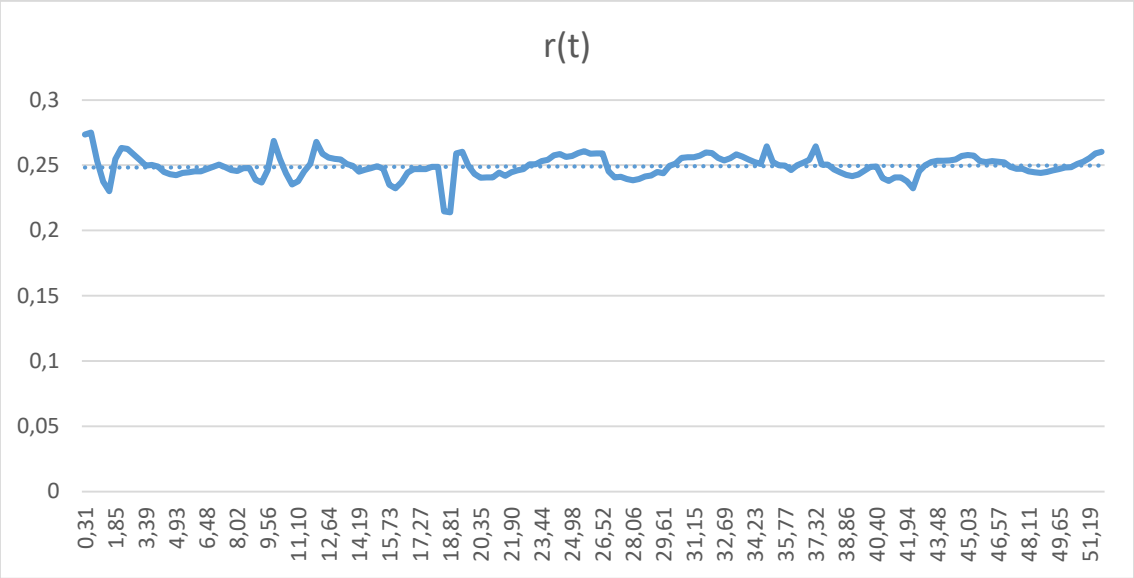


Figura 29: Radio del robot con respecto al tiempo. El eje y muestra el radio en metros y el eje x, el tiempo en segundos.

En la primera, el robot se colocó inicialmente a una distancia inferior al radio deseado. En la segunda, por el contrario, se colocó a una distancia mayor. Al igual que en las gráficas del programa anterior, ninguna alcanza el equilibrio, pero la línea de tendencia se encuentra en torno a 25 centímetros, el radio deseado, en ambas.

La figura 30 muestra una de las ejecuciones de este código, extraída del vídeo:

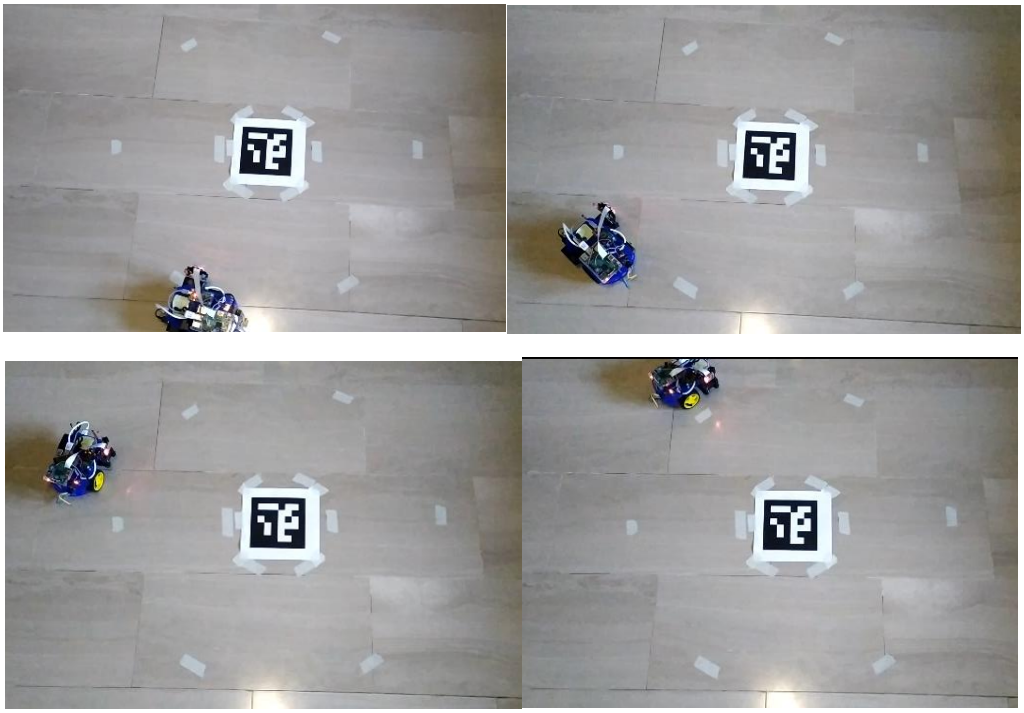


Figura 30: Ejecución de MarcaH.py

En este experimento se puede observar cómo el prisma hexagonal ha sido sustituido por un solo marcador, aunque el resto del montaje no haya cambiado. Cabe destacar que el radio deseado en este caso es ligeramente mayor al marcado por las cintas, debido a que fue necesario encontrar el rango de radios para el cual el marcador se detectaba adecuadamente. Como consecuencia, la posición inicial del robot no puede ser tan lejana ni tan cercana como la del experimento anterior. También es necesario tener en cuenta que el cálculo de la posición del robot es menos preciso en esta configuración que en configuraciones anteriores debido a la deformación óptica del marcador.

6. Conclusión

A lo largo de este trabajo se ha demostrado que es posible construir un sistema de navegación basado en visión por ordenador. En concreto, se han creado dos programas que permiten el giro autónomo en torno a un punto o figura.

Además, se ha diseñado un control sencillo que permite utilizar los datos de posición y orientación que proporcionan las marcas para ajustar la trayectoria del robot y, así, contrarrestar las perturbaciones introducidas por los motores. A pesar de que la calidad de los materiales utilizados no fue demasiado buena y de la simplicidad del control, el sistema es capaz de cumplir los objetivos de diseño con una precisión razonable, y con un bajo coste.

En un futuro, se podría estudiar cómo combinar el control inicial (4) con la detección de marcas visuales. De esta manera, si se colocaran marcadores a lo largo de las paredes de una vivienda, por ejemplo, el robot podría programarse para realizar distintos circuitos. Así, podría añadirse la dificultad de tener que detectar sólo los marcadores pertenecientes a la ruta especificada. Otra opción sería poner los marcadores en el techo o en el suelo, pero en este caso debería alargarse el cable de la cámara o mejorar su resolución para facilitar la detección. También sería posible asociar a cada marcador un movimiento específico del robot, como giros en torno a sí mismo, movimientos rectos en un sentido u otro… En cualquier caso, se podría crear un sistema de navegación flexible mediante el cambio de lugar de los marcadores, con aplicaciones en logística industrial, por ejemplo.

En conclusión, la idea inicial del trabajo no se ha llegado a corresponder totalmente al resultado final, pero, desde el punto de vista docente, ha sido un éxito. Se ha aprendido a programar en un lenguaje multiuso, se ha tenido la oportunidad de aplicar conceptos teóricos a usos tangibles y se han desarrollado cualidades como la resolución de problemas y la creatividad.

Espero que este trabajo sirva para contribuir al desarrollo del proyecto PiRobot y para ayudar a futuros alumnos.

ANEXOS

A1. Código de los programas.

A1.1. DetectAxis.py

El pseudocódigo del programa es el siguiente:

Inicio de la captación continua de imágenes de la cámara.

Para cada imagen captada:

- Buscar marcadores.
- Si no hay marcadores en la imagen:
  - Mostrar por pantalla la imagen captada sin modificaciones.
- Si hay marcadores en la imagen:
  - Obtener los vectores de posición y orientación de los marcadores.
  - Mostrar por pantalla los marcadores recuadrados y/o dibujar sus ejes.
  - (Opcional) Imprimir los datos referentes a distancias y posiciones.
- Si se pulsa la tecla q:
  - Finalizar.

El código completo es:

```
# Este programa activa la cámara y devuelve en pantalla los marcadores
# detectados, recuadrados en verde, junto con los ejes.

import numpy as np
import cv2
import cv2.aruco as aruco
import time
from picamera.array import PiRGBArray
from picamera import PiCamera

longMarca = 0.043 #No es necesario editar esta variable (en este programa)
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
#camera.rotation = 180 #Activar cuando se ponga la cámara al revés.
rawCapture = PiRGBArray(camera, size=(640, 480))
time.sleep(0.1)

diccionario = aruco.Dictionary_get(aruco.DICT_6X6_250)

cameraMatrix = np.array([[610.22177936,   0,        368.54175257],[  0,        610.54590987, 346.66406953],[  0,
0,          1,        ]])
distCoeffs = np.array([[-1.12957874,  1.97328403, -0.06451624, -0.00856775, -1.31922536]])

print(cameraMatrix)
print(distCoeffs)

i = 0
j = 0
```

```
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
        image = frame.array
        esquinas, ids, reject = aruco.detectMarkers(image, diccionario)
        rvec, tvec, _objPoints = aruco.estimatePoseSingleMarkers(esquinas, longMarca , cameraMatrix,
distCoeffs)

        key = cv2.waitKey(1) & 0xFF


        if (np.all(rvec) and np.all(tvec)) == None:          #Si no se encuentra ningún marcador en la imagen...

                resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
                cv2.resizeWindow('Image', 1200,1200)
                cv2.imshow('Marcadores detectados',resultado)


        elif ((np.all(rvec) and np.all(tvec)) == True):      #Si se encuentra algún marcador en la imagen...

                if j == 0:  #La primera vez
                        rvecs = np.array(rvec)
                        tvecs = np.array(tvec)
                        j = j + 1
                        resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                        resultado = aruco.drawAxis(resultado, cameraMatrix, distCoeffs, rvec[i], tvec[i], 0.05)
                        cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
                        cv2.resizeWindow('Image', 1200,1200)
                        cv2.imshow('Marcadores detectados',resultado)
                        i = i+1

                else:
                        rvecs = np.vstack((rvecs,rvec))
                        tvecs = np.vstack((tvecs, tvec))
                        resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                        resultado = aruco.drawAxis(resultado, cameraMatrix, distCoeffs, rvecs[i], tvecs[i],
0.05)
                        cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
                        cv2.resizeWindow('Image', 1200,1200)
                        cv2.imshow('Marcadores detectados',resultado)
                        tx=(tvecs[i])[0,0] #Por si hay que imprimir alguno de ellos
                        ty=(tvecs[i])[0,1]
                        tz=(tvecs[i])[0,2]
                        rx=(rvecs[i])[0,0]
                        ry=(rvecs[i])[0,1]
                        rz=(rvecs[i])[0,2]

                        #if (key == ord("c")):
                        #       print("Tz: ", tz)

                        i = i+1
        rawCapture.truncate(0)      #Elimina la imagen actual antes de mostrar otra.

        if key == ord("q"):
                break
```

A1.2. VeAMarca3.py

El pseudocódigo del programa es:

Inicio de la captación continua de imágenes de la cámara.

Girar alrededor de un punto fijo hasta detectar un marcador.

Si se detecta un marcador:

Estimar la distancia a la que se encuentra.

Obtener la distancia en el eje Z

Si se encuentra a una distancia mayor a la establecida por el usuario:

Avanzar

Si se encuentra a una distancia igual o menor a la introducida por el usuario:

Detenerse

Si se pulsa la tecla “q”:

Finalizar el programa.

EL código completo es:

```
# Este programa hace que el robot gire sobre sí mismo hasta que
# encuentre un marcador, y entonces lo hace avanzar hasta quedarse
# a una distancia indicada por el usuario.

import numpy as np
import math
import cv2
import cv2.aruco as aruco
import time
from picamera.array import PiRGBArray
from picamera import PiCamera

import sys
import serial
import RPi.GPIO as GPIO


print("¿A qué distancia debe quedarse el robot de la marca? (m)")
distAMarca = float(input('> '))

print("¿Cuál es la longitud del lado de la marca, en metros?")
print("Para dejar la distancia predeterminada de 9.6 cm, introduce 'd'")
respuesta = input('> ')
if respuesta == 'd':
        longMarca = 0.096
else:
        longMarca = float(respuesta)

#vfinal = np.array([[0,0,distAMarca,1]])
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
camera.rotation = 180
rawCapture = PiRGBArray(camera, size=(640, 480))
time.sleep(0.1)

diccionario = aruco.Dictionary_get(aruco.DICT_6X6_250)
```

```
cameraMatrix = np.array([[610.22177936,   0,        368.54175257],[  0,        610.54590987, 346.66406953],[  0, 0, 1, ]])
distCoeffs = np.array([[-1.12957874,  1.97328403, -0.06451624, -0.00856775, -1.31922536]])

print(cameraMatrix)
print(distCoeffs)

i = 0
j = 0
tx = 1000

############
Puerto = '/dev/ttyUSB0'


try:
        s = serial.Serial(Puerto, 57600)
        s.timeout=5;

except serial.SerialException:

        sys.stderr.write("Error al abrir puerto (%s)\n" % str(Puerto))

        sys.exit(1)

time.sleep(10)

prueba = s.readline()
print ("PRUEBA: ", prueba.decode('ascii'))

f = 0
############


for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
        image = frame.array
        esquinas, ids, reject = aruco.detectMarkers(image, diccionario)
        rvec, tvec, _objPoints = aruco.estimatePoseSingleMarkers(esquinas, longMarca , cameraMatrix, distCoeffs)

        key = cv2.waitKey(1) & 0xFF


        if ((np.all(rvec) and np.all(tvec)) == None): #Si no detecta marcadores, gira sobre sí mismo

                resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
                cv2.resizeWindow('Image', 1200,1200)
                cv2.imshow('Marcadores detectados',resultado)

                velocidad = ':-65,0\n'
                if prueba == ('Serial waiting.\n').encode('utf-8'):

                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                if eco.decode('ascii')==velocidad:
                                        print ("Moviendo")
                                else :
                                        print ("error!")


        elif ((np.all(rvec) and np.all(tvec)) == True):

                if j == 0:
                        rvecs = np.array(rvec)
                        tvecs = np.array(tvec)
                        j = j + 1
                        resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                        resultado = aruco.drawAxis(resultado, cameraMatrix, distCoeffs, rvec[i], tvec[i], 0.05)
                        cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
```

```
                cv2.resizeWindow('Image', 1200,1200)
                cv2.imshow('Marcadores detectados',resultado)
                i = i+1
                print ("Marca encontrada")

        elif j > 0: #Si está a una distancia mayor a la especificada se mueve.
                rvecs = np.vstack((rvecs,rvec))
                tvecs = np.vstack((tvecs, tvec))
                resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                resultado = aruco.drawAxis(resultado, cameraMatrix, distCoeffs, rvecs[i], tvecs[i], 0.05)
                cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
                cv2.resizeWindow('Image', 1200,1200)
                cv2.imshow('Marcadores detectados',resultado)
                tx=(tvecs[i])[0,0]
                ty=(tvecs[i])[0,1]
                tz=(tvecs[i])[0,2]
                rx=(rvecs[i])[0,0]
                ry=(rvecs[i])[0,1]
                rz=(rvecs[i])[0,2]


                if ((tz > distAMarca) and (f == 0)):
                        velocidad = ':90,90\n'
                        if (i == 1):
                                print("Moviendo.")

                        if prueba == ('Serial waiting.\n').encode('utf-8'):
                                print("Prueba = serialwaiting")
                                s.write(velocidad.encode('utf-8'))
                                eco = s.readline()

                                if len(eco.decode('ascii'))!=0:
                                        print ("RECIBIDO: " + eco.decode('ascii'))
                                        if eco.decode('ascii')==velocidad:
                                                print ("Mover")
                                        else :
                                                print ("error!")


                ####################################
                if ((tz <= distAMarca) and (f == 0)):
                        s.write((':0,0\n').encode('utf-8'))
                        s.readline()
                        s.close()
                        f = f + 1
                        print("El robot ha llegado a su destino")

                i = i+1

        else:
                cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
                cv2.resizeWindow('Image', 1200,1200)
                cv2.imshow('Marcadores detectados',image)
rawCapture.truncate(0)
if key == ord("q"):
        s.write((':0,0\n').encode('utf-8'))
        s.readline()
        s.close()
        break
```

A1.3. Ref.py

Su código completo es el siguiente:

```
#Este programa establece el sistema de referencia del robot con respecto
#al de la cámara. Además, calcula la matriz de transferencia CTR.
#Para ello saca una foto de una marca, colocada perfectamente
#perpendicular al suelo.

import numpy as np
import math
import cv2
import cv2.aruco as aruco
import time
from picamera.array import PiRGBArray
from picamera import PiCamera

import sys
import serial
import RPi.GPIO as GPIO



longMarca = 0.096 #Editar este valor cuando se cambie de marca


camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
camera.rotation = 180
rawCapture = PiRGBArray(camera, size=(640, 480))
time.sleep(0.1)

diccionario = aruco.Dictionary_get(aruco.DICT_6X6_250)

cameraMatrix = np.matrix ([[549.4858516 ,  0.      ,  231.95990112],
 [  0.     ,  545.71020483, 119.3904947 ],
 [  0.     ,    0.     ,    1.        ]] )
distCoeffs = np.array([[-1.07019571,  1.76457253,  0.04606337,  0.0281203,  -1.10438634]])


i = 0
j = 0
tx = 1000

#time.sleep(10)

f = 0

key = cv2.waitKey(1) & 0xFF


for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
        image = frame.array
        esquinas, ids, reject = aruco.detectMarkers(image, diccionario)
        rvec, tvec, _objPoints = aruco.estimatePoseSingleMarkers(esquinas, longMarca , cameraMatrix, distCoeffs)

        key = cv2.waitKey(1) & 0xFF


        if ((np.all(rvec) and np.all(tvec)) == None):

                resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
                cv2.resizeWindow('Image', 1200,1200)
                cv2.imshow('Marcadores detectados',resultado)

        elif ((np.all(rvec) and np.all(tvec)) == True):

                if j == 0:
                        rvecs = np.array(rvec)
```

```
                tvecs = np.array(tvec)
                j = j + 1
                resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                resultado = aruco.drawAxis(resultado, cameraMatrix, distCoeffs, rvec[i], tvec[i], 0.05)
                cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
                cv2.resizeWindow('Image', 1200,1200)
                cv2.imshow('Marcadores detectados',resultado)
                i = i+1
                print ("Marca encontrada")


        elif j > 0:
                rvecs = np.vstack((rvecs,rvec))
                tvecs = np.vstack((tvecs, tvec))
                resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                resultado = aruco.drawAxis(resultado, cameraMatrix, distCoeffs, rvecs[i], tvecs[i], 0.05)
                cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
                cv2.resizeWindow('Image', 1200,1200)
                cv2.imshow('Marcadores detectados',resultado)
                tx=(tvecs[i])[0,0]
                ty=(tvecs[i])[0,1]
                tz=(tvecs[i])[0,2]
                rx=(rvecs[i])[0,0]
                ry=(rvecs[i])[0,1]
                rz=(rvecs[i])[0,2]


                if key == ord("c"):
                        #Rodrigues
                        CTM_rot = np.zeros(shape=(3,3))
                        cv2.Rodrigues(rvecs[i], CTM_rot)
                        CTM = np.matrix([[CTM_rot[0][0],CTM_rot[0][1],CTM_rot[0][2], tx],
                        [CTM_rot[1][0],CTM_rot[1][1],CTM_rot[1][2],ty],
                        [CTM_rot[2][0],CTM_rot[2][1],CTM_rot[2][2],tz],
                        [0,0,0,1]])
                        MTC = np.linalg.inv(CTM)
                        MTR =
np.matrix([[1,0,0,MTC.item((0,3))],[0,1,0,MTC.item((1,3))],[0,0,1,MTC.item((2,3))],[0,0,0,1]])

                        CTR = CTM*MTR
                        RTM = np.linalg.inv(MTR)

                        print("La matriz RTM es:")
                        print(RTM)
                        print("La matriz MTC es:")
                        print(MTC)
                        print("La matriz MTR es:")
                        print(MTR)
                        print("La matriz CTM es:")
                        print(CTM)
                        print("La matriz CTR es: ")
                        print(CTR)

                        print("El robot se encuentra a una distancia z del centro de la marca de:")
                        print(MTR.item((2,3)))
                        print("El robot se encuentra a una distancia x del centro de la marca de:")
                        print(MTR.item((0,3)))
                        print("El robot se encuentra a una distancia y del centro de la marca de:")
                        print(MTR.item((1,3)))

                        print("IDS:")
                        print(ids)


                        Nz = np.array([MTR.item((2,0))])
                        Nx = np.array([MTR.item((0,0))])
                        Ny = np.array([MTR.item((1,0))])


                        Z = np.arctan2(Ny,Nx)
```

```
                                        Y = np.arctan2(-Nz, Nx*math.cos(Z)+Ny*math.sin(Z))


                                i = i+1


                        else:
                                cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
                                cv2.resizeWindow('Image', 1200,1200)
                                cv2.imshow('Marcadores detectados',image)
                rawCapture.truncate(0)
                if key == ord("q"):
                        break
```

A1.4. VeLat.py

El código completo es:

```
#Gira y va hasta la marca en la configuración de cámara lateral.

import numpy as np
import math
import cv2
import cv2.aruco as aruco
import time
from picamera.array import PiRGBArray
from picamera import PiCamera

import sys
import serial
import RPi.GPIO as GPIO

def calculoT(tx,ty,tz,rx,ry,rz):
        CTM_rot = np.zeros(shape=(3,3))
        cv2.Rodrigues(rvecs[i], CTM_rot)
        CTM = np.matrix([[CTM_rot[0][0],CTM_rot[0][1],CTM_rot[0][2], tx],
        [CTM_rot[1][0],CTM_rot[1][1],CTM_rot[1][2],ty],
        [CTM_rot[2][0],CTM_rot[2][1],CTM_rot[2][2],tz],
        [0,0,0,1]])
        MTC = np.linalg.inv(CTM)
        CTR = ([[ 9.87489937e-01, 7.61112482e-02, -1.38096712e-01,  0.00000000e+00],
         [ 1.46089088e-02, -9.16186053e-01, -4.00486824e-01,  0.00000000e+00],
         [-1.57003834e-01,  3.93459266e-01, -9.05836410e-01, -5.55111512e-17],
         [ 0.00000000e+00 , 0.00000000e+00,  0.00000000e+00,  1.00000000e+00]])

        MTR = MTC*CTR


        return MTR

def calculoAngulo(MTR):
        Nz = np.array([MTR.item((2,0))])
        Nx = np.array([MTR.item((0,0))])
        Ny = np.array([MTR.item((1,0))])
        Z = np.arctan2(Ny,Nx)
        Y = np.arctan2(-Nz, Nx*math.cos(Z)+Ny*math.sin(Z))

        return (Y)

#longMarca = 0.15
#distAMarca = 0.4

print("¿A qué distancia debe quedarse el robot de la marca? (m)")
distAMarca = float(input('> '))

print("¿Cuál es la longitud del lado de la marca, en metros?")
print("Para dejar la distancia predeterminada de 8.3 cm, introduce 'd'")
respuesta = input('> ')
if respuesta == 'd':
        longMarca = 0.083
```

```
else:
        longMarca = float(respuesta)

vfinal = np.array([[0,0,distAMarca,1]])
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
camera.rotation = 180
rawCapture = PiRGBArray(camera, size=(640, 480))
time.sleep(0.1)

diccionario = aruco.Dictionary_get(aruco.DICT_6X6_250)

cameraMatrix = np.matrix ([[549.4858516 ,  0.      , 231.95990112],
 [  0.     ,  545.71020483, 119.3904947 ],
 [  0.     ,    0.     ,    1.        ]] )
distCoeffs = np.array([[-1.07019571,  1.76457253,  0.04606337,  0.0281203,  -1.10438634]])

print(cameraMatrix)
print(distCoeffs)

i = 0
j = 0
n = 0
tx = 1000

############
Puerto = '/dev/ttyUSB0'


try:
        s = serial.Serial(Puerto, 57600)
        s.timeout=5;

except serial.SerialException:

        sys.stderr.write("Error al abrir puerto (%s)\n" % str(Puerto))

        sys.exit(1)

time.sleep(10)

prueba = s.readline()
print ("PRUEBA: ", prueba.decode('ascii'))

f = 0
i = 0
u = 0

Fase0 = True
Fase1 = False
Fase2 = False
Fase3 = False
Fase4 = False
Fase5 = False
Fase6 = False
Final = False
############


key = cv2.waitKey(1) & 0xFF

for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
        image = frame.array
        esquinas, ids, reject = aruco.detectMarkers(image, diccionario)
        rvec, tvec, _objPoints = aruco.estimatePoseSingleMarkers(esquinas, longMarca , cameraMatrix, distCoeffs)

        key = cv2.waitKey(1) & 0xFF


        #El robot empieza girando sobre sí mismo para buscar el marcador.
```

```
if (Fase0 == True):
        resultado = image
        velocidad = ':-65,0\n'
        print("VELOCIDAD: ", velocidad)
        if prueba == ('Serial waiting.\n').encode('utf-8'):

                s.write(velocidad.encode('utf-8'))
                eco = s.readline()

                if len(eco.decode('ascii'))!=0:
                        if eco.decode('ascii')==velocidad:
                                print ("Moviendo")
                        else :
                                print ("error!")


if ((np.all(rvec) and np.all(tvec)) == True):
        if j == 0: #La primera vez que detecta el marcador
                rvecs = np.array(rvec)
                tvecs = np.array(tvec)
                j = j + 1
                encontrada = True
                Fase0 = False
                Fase1 = True
                resultado = image
                s.write((':0,0\n').encode('utf-8'))
                time.sleep(1)

        elif j > 0:
                rvecs = np.vstack((rvecs,rvec))
                tvecs = np.vstack((tvecs, tvec))
                resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                resultado = aruco.drawAxis(resultado, cameraMatrix, distCoeffs, rvecs[i], tvecs[i], 0.05)


        cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
        cv2.resizeWindow('Image', 1200,1200)
        cv2.imshow('Marcadores detectados',resultado)
        tx=(tvecs[i])[0,0]
        ty=(tvecs[i])[0,1]
        tz=(tvecs[i])[0,2]
        rx=(rvecs[i])[0,0]
        ry=(rvecs[i])[0,1]
        rz=(rvecs[i])[0,2]

        MTR = calculoT(tx,ty,tz,rx,ry,rz)
        Y = calculoAngulo(MTR)

        i = i + 1

        dx = MTR.item((0,3))
        dy = MTR.item((1,3))
        dz = MTR.item((2,3))

elif ((np.all(rvec) and np.all(tvec)) == None):
        #resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
        resultado = image
        cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
        cv2.resizeWindow('Image', 1200,1200)
        cv2.imshow('Marcadores detectados',resultado)


if (Fase1 == True): #En esta fase, gira para colocarse paralelo al eje x del marcador
        if (n == 0):

                if (dx > 0):
                        velocidad = ':-65,0\n'
                elif (dx < 0):
                        velocidad = ':-65,0\n'
```

```
                n = n + 1

        if (((Y > -0.07) and (Y < 0.07)) == False):

                print("Colocando el robot. Y=", Y)
                if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else :
                                        print ("error!")
        elif (((Y > -0.07) and (Y < 0.07)) == True):
                print("Fase 1 terminada. Robot paralelo a la marca")
                s.write((':0,0\n').encode('utf-8'))
                time.sleep(1)

                Fase1 = False
                Fase2 = True

elif (Fase2 == True): #En esta fase, avanza hasta el centro de la marca
        if (u == 0): #La primera vez que mida la distancia x con respecto a la marca...
                if (dx < 0):
                        velocidad = ':90,90\n'
                elif (dx > 0):
                        velocidad = ':-90,-90\n'
                u = 1

        if (((dx < 0.03) and (dx > -0.03)) == False):
                print("Avanzando hasta el centro de la marca. tx=", tx)

                if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else :
                                        print ("error!")

        elif (((dx < 0.03) and (dx > -0.03)) == True):
                print("Robot en el centro de la marca")
                s.write((':0,0\n').encode('utf-8'))
                time.sleep(1)
                Fase2 = False
                Fase3 = True


#Gira para ponerse perpendicular a la marca
elif (Fase3 == True):
        velocidad = ':65,0\n'
        print("Colocando el robot")

        if prueba == ('Serial waiting.\n').encode('utf-8'):
                print("Prueba = serialwaiting")
                s.write(velocidad.encode('utf-8'))
                eco = s.readline()

                if len(eco.decode('ascii'))!=0:
                        print ("RECIBIDO: " + eco.decode('ascii'))
                        if eco.decode('ascii')==velocidad:
                                print ("Mover")
                        else :
                                print ("error!")
```

```
                time.sleep(2.4)
                s.write((':0,0\n').encode('utf-8'))
                time.sleep(1)
                Fase3 = False
                Fase4 = True


#Comienza a avanzar hacia la marca, una vez en su centro.
elif (Fase4 == True):
                velocidad = ':-90,-90\n'
                print("Avanzando hacia la marca")

                if prueba == ('Serial waiting.\n').encode('utf-8'):
                                print("Prueba = serialwaiting")
                                s.write(velocidad.encode('utf-8'))
                                eco = s.readline()

                                if len(eco.decode('ascii'))!=0:
                                                print ("RECIBIDO: " + eco.decode('ascii'))
                                                if eco.decode('ascii')==velocidad:
                                                                print ("Mover")
                                                else :
                                                                print ("error!")

                time.sleep(dz/0.125 - 2)
                s.write((':0,0\n').encode('utf-8'))
                time.sleep(1)
                Fase4 = False
                Fase5 = True

elif (Fase5 == True):
                velocidad = ':-65,0\n'
                if prueba == ('Serial waiting.\n').encode('utf-8'):
                                print("Prueba = serialwaiting")
                                s.write(velocidad.encode('utf-8'))
                                eco = s.readline()

                                if len(eco.decode('ascii'))!=0:
                                                print ("RECIBIDO: " + eco.decode('ascii'))
                                                if eco.decode('ascii')==velocidad:
                                                                print ("Mover")
                                                else :
                                                                print ("error!")
                time.sleep(2.5)
                Fase6 = True
                Fase5 = False
                Final = True


elif (Final == True):
                s.write((':0,0\n').encode('utf-8'))
                s.readline()
                s.close()


rawCapture.truncate(0)
if key == ord("q"):
                s.write((':0,0\n').encode('utf-8'))
                s.readline()
                s.close()
                break
```

A1.5. GiraHex3.py

El pseudocódigo del programa es el siguiente:

Inicio de la captación continua de imágenes de la cámara.

Si se detecta uno o varios marcadores:

Mostrar por pantalla la vista de la cámara con los marcadores recuadrados

Calcular posición, orientación e ids.

Si (mover == Verdadero):

Avanzar hasta detectar un id superior

Mover = Falso

CogerDistancia = Verdadero

Si (CogerDistancia == Verdadero):

Avanzar hasta que el robot se encuentre a cierta distancia del eje Z del nuevo marcador

CogerDistancia == Falso

Girar == Verdadero

Si (Girar == Verdadero):

Girar tiempo definido para cada fase

Girar = Falso

Mover = Verdadero

Si se pulsa la tecla “q”:

Finalizar el programa.

El código completo es:

```
#Este programa hace que el robot gire alrededor de una figura hexagonal
#mediante trayectorias rectas y giros sobre sí mismo

import numpy as np
import math
import cv2
import cv2.aruco as aruco
import time
from picamera.array import PiRGBArray
from picamera import PiCamera

import sys
import serial
import RPi.GPIO as GPIO

#Definición de funciones

def calculoT(tx,ty,tz,rx,ry,rz,rvec):
```

```
        MTC_rot = np.zeros(shape=(3,3))
        cv2.Rodrigues(rvec, MTC_rot)
        MTC = np.matrix([[MTC_rot[0][0],MTC_rot[0][1],MTC_rot[0][2], tx],
        [MTC_rot[1][0],MTC_rot[1][1],MTC_rot[1][2],ty],
        [MTC_rot[2][0],MTC_rot[2][1],MTC_rot[2][2],tz],
        [0,0,0,1]])
        CTM = np.linalg.inv(MTC)
        CTR = ([[ 0.9996148,  -0.00458056, -0.02737291,  0.       ],
        [-0.01463449, -0.92502031, -0.37963569,  0.       ],
        [-0.02358155,  0.37989004, -0.92473102,  0.       ],
        [ 0.       ,  0.       ,  0. ,        1.       ]])

        MTR = MTC*CTR


        return MTR

def calculoAngulo(MTR):
        Nz = np.array([MTR.item((2,0))])
        Nx = np.array([MTR.item((0,0))])
        Ny = np.array([MTR.item((1,0))])
        Z = np.arctan2(Ny,Nx)
        Y = np.arctan2(-Nz, Nx*math.cos(Z)+Ny*math.sin(Z))

        return (Y)

def gira60():
        velocidad = ':65,0\n'
        if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else :
                                        print ("error!")
        time.sleep(1)
        s.write((':0,0\n').encode('utf-8'))

#Cosas para inicializar GPIO

Puerto = '/dev/ttyUSB0'


try:
        s = serial.Serial(Puerto, 57600)
        s.timeout=5;

except serial.SerialException:

        sys.stderr.write("Error al abrir puerto (%s)\n" % str(Puerto))

        sys.exit(1)

time.sleep(10)

prueba = s.readline()
print ("PRUEBA: ", prueba.decode('ascii'))

#Inicialización de cámara

camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
camera.rotation = 180
rawCapture = PiRGBArray(camera, size=(640, 480))
time.sleep(0.1)
```

```
diccionario = aruco.Dictionary_get(aruco.DICT_6X6_250)

cameraMatrix = np.matrix ([[549.4858516 ,  0.      , 231.95990112],
 [  0.     ,  545.71020483, 119.3904947 ],
 [  0.     ,   0.     ,   1.        ]] )
distCoeffs = np.array([[-1.07019571,  1.76457253,  0.04606337,  0.0281203,  -1.10438634]])

longMarca = 0.077
distAMarca = 0.2

#Variables Variables Variables Variables Variables Variables

moviendo = False
girando = False
j = 0

nmarcas = 0
u = 0
o = 0
d = 0
e = 1

Fase0 = True
Fase1 = False
Fase2 = False
Fase3 = False
Fase4 = False
Fase5 = False

ids = np.array([[0],[0]])
#Un bucle que mueve y gira
cogiendoDistancia = False

for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
        image = frame.array
        esquinas, ids, reject = aruco.detectMarkers(image, diccionario)
        rvec, tvec, _objPoints = aruco.estimatePoseSingleMarkers(esquinas, longMarca , cameraMatrix, distCoeffs)

        key = cv2.waitKey(1) & 0xFF
        if ((np.all(rvec) and np.all(tvec)) == True):
                if j == 0: #La primera vez que detecta el marcador
                        #rvecs = np.array(rvec)
                        #tvecs = np.array(tvec)
                        j = j + 1
                        encontrada = True
                        resultado = image
                        print("Primera marca detectada")
                        resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                        #time.sleep(0.5)
                        moviendo = True
                        Fase0 = True
                        tx=(tvec[0])[0,0]
                        ty=(tvec[0])[0,1]
                        tz=(tvec[0])[0,2]
                        rx=(rvec[0])[0,0]
                        ry=(rvec[0])[0,1]
                        rz=(rvec[0])[0,2]

                        rvecM = rvec[0]
                        if (len(ids) == 1):
                                IDS = ids[0][0]
                        else:
                                IDS = ((ids[0][0]) or (ids[1][0]))

                elif j > 0:
                        #rvecs = np.vstack((rvecs,rvec))
                        #tvecs = np.vstack((tvecs, tvec))
                        resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                        #if (len(ids) == 1):
                        #       print("Una marca. Rvecs, tvecs: ")
                        #       rvecss = rvecs[i]
```

```
                        #           tvecss = tvecs[i]
                        #elif (len(ids) > 1):
                        #           print("Dos marcas. Rvecs, tvecs:")
                        #           print(rvecs)
                        #           print(tvecs)
                        #           rvecss = (rvecs[i])[1]
                        #           tvecss = (tvecs[i])[1]
                        if (len(ids) == 1):
                                    resultado = aruco.drawAxis(resultado, cameraMatrix, distCoeffs, rvec[0], tvec[0], 0.05)
                                    tx=(tvec[0])[0,0]
                                    ty=(tvec[0])[0,1]
                                    tz=(tvec[0])[0,2]
                                    rx=(rvec[0])[0,0]
                                    ry=(rvec[0])[0,1]
                                    rz=(rvec[0])[0,2]

                                    rvecM = rvec[0]
                                    IDS = ids[0][0]
                        elif(len(ids) > 1):
                                    #resultado = aruco.drawAxis(resultado, cameraMatrix, distCoeffs, rvec[len(ids)-1],
tvec[len(ids)-1], 0.05)
                                    resultado = aruco.drawAxis(resultado, cameraMatrix, distCoeffs, rvec[0], tvec[0], 0.05)
                                    tx=(tvec[0])[0,0]
                                    ty=(tvec[0])[0,1]
                                    tz=(tvec[0])[0,2]
                                    rx=(rvec[0])[0,0]
                                    ry=(rvec[0])[0,1]
                                    rz=(rvec[0])[0,2]

                                    rvecM = rvec[0]
                                    IDS = ((ids[0][0]) or (ids[1][0]))


                        #tx=(tvecss)[0,0]
                        #ty=(tvecss)[0,1]
                        #tz=(tvecss)[0,2]
                        #rx=(rvecss)[0,0]
                        #ry=(rvecss)[0,1]
                        #rz=(rvecss)[0,2]
                                                #if (len(ids) == 1):
                                                #           tx=(tvecs[i])[0,0]
                                                #           ty=(tvecs[i])[0,1]
                                                #           tz=(tvecs[i])[0,2]
                                                #           rx=(rvecs[i])[0,0]
                                                #           ry=(rvecs[i])[0,1]
                                                #           rz=(rvecs[i])[0,2]
                                                #elif (len(ids) > 1):
                                                #           print(tvecs)
                                                #           print("ids:")
                                                #           print(ids)
                                                #           tx=(tvecs[i][1])[0,0]
                                                #           ty=(tvecs[i][1])[0,1]
                                                #           tz=(tvecs[i][1])[0,2]
                                                #           rx=(rvecs[i][1])[0,0]
                                                #           ry=(rvecs[i][1])[0,1]
                                                #           rz=(rvecs[i][1])[0,2]

                        MTR = calculoT(tx,ty,tz,rx,ry,rz,rvecM)
                        Y = calculoAngulo(MTR)

                        #i = i + 1

                        dx = MTR.item((0,3))
                        dy = MTR.item((1,3))
                        dz = MTR.item((2,3))

                        nmarcas = len(ids)
            elif ((np.all(rvec) and np.all(tvec)) == None):
                        resultado = image
                        nmarcas = 0
                        if (j == 0):
```

```
                    time.sleep(1)
                IDS = 999

        #Obtención de la posición del robot respecto de la marca.
        cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
        cv2.resizeWindow('Image', 1200,1200)
        cv2.imshow('Marcadores detectados',resultado)
        print("IDS:")
        print(ids)


#########################################################################################################
        if (moviendo == True):
                if (Fase0 == True):
                        p = 0
                        print("Fase0")
                        if ((IDS == 1) == False):
                                velocidad = ':-70,-70\n'
                                if prueba == ('Serial waiting.\n').encode('utf-8'):
                                        print("Prueba = serialwaiting")
                                        s.write(velocidad.encode('utf-8'))
                                        eco = s.readline()

                                        if len(eco.decode('ascii'))!=0:
                                                print ("RECIBIDO: " + eco.decode('ascii'))
                                                if eco.decode('ascii')==velocidad:
                                                        print ("Mover")
                                                else :
                                                        print ("error!")
                        elif ((((ids[0][0])or(ids[0][1]))== 1) == True):

                                velocidad = ':-70,-70\n'
                                if prueba == ('Serial waiting.\n').encode('utf-8'):
                                        print("Prueba = serialwaiting")
                                        s.write(velocidad.encode('utf-8'))
                                        eco = s.readline()

                                        if len(eco.decode('ascii'))!=0:
                                                print ("RECIBIDO: " + eco.decode('ascii'))
                                                if eco.decode('ascii')==velocidad:
                                                        print ("Mover")
                                                else :
                                                        print ("error!")

                                        print("HA localizado la marca 1. Dz esperada: ")
                                        print(distAMarca)
                                        cogiendoDistancia = True
                                        Fase0 = False



                if (Fase1 == True):
                        p = 1
                        print("Fase1")
                        if ((IDS == 2) == False):
                                velocidad = ':-70,-70\n'
                                if prueba == ('Serial waiting.\n').encode('utf-8'):
                                        print("Prueba = serialwaiting")
                                        s.write(velocidad.encode('utf-8'))
                                        eco = s.readline()

                                        if len(eco.decode('ascii'))!=0:
                                                print ("RECIBIDO: " + eco.decode('ascii'))
                                                if eco.decode('ascii')==velocidad:
                                                        print ("Mover")
                                                else :
                                                        print ("error!")
                        elif ((((ids[0][0])or(ids[0][1]))== 2) == True):

                                velocidad = ':-70,-70\n'
```

```
                    if prueba == ('Serial waiting.\n').encode('utf-8'):
                            print("Prueba = serialwaiting")
                            s.write(velocidad.encode('utf-8'))
                            eco = s.readline()

                            if len(eco.decode('ascii'))!=0:
                                    print ("RECIBIDO: " + eco.decode('ascii'))
                                    if eco.decode('ascii')==velocidad:
                                            print ("Mover")
                                    else :
                                            print ("error!")

                            print("HA localizado la marca 2. Dz esperada: ")
                            print(distAMarca)
                            cogiendoDistancia = True
                            Fase1 = False


if (Fase2 == True):
        p = 2
        print("Fase2")
        if ((IDS == 3) == False):
                velocidad = ':-70,-70\n'
                if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else :
                                        print ("error!")
        elif ((((ids[0][0])or(ids[0][1]))== 3) == True):

                velocidad = ':-70,-70\n'
                if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else :
                                        print ("error!")

                        print("HA localizado la marca 3. Dz esperada: ")
                        print(distAMarca)
                        cogiendoDistancia = True
                        Fase2 = False


if (Fase3 == True):
        p = 3
        print("Fase3")
        if ((IDS == 4) == False):
                velocidad = ':-70,-70\n'
                if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else :
```

```
                                                print ("error!")
        elif ((((ids[0][0])or(ids[0][1]))== 4) == True):

                velocidad = ':-70,-70\n'
                if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else :
                                        print ("error!")

                        print("HA localizado la marca 4. Dz esperada: ")
                        print(distAMarca)
                        cogiendoDistancia = True
                        Fase3 = False


if (Fase4 == True):
        p = 4
        print("Fase4")
        if ((IDS == 5) == False):
                velocidad = ':-70,-70\n'
                if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else :
                                        print ("error!")
        elif ((((ids[0][0])or(ids[0][1]))== 5) == True):

                velocidad = ':-70,-70\n'
                if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else :
                                        print ("error!")

                        print("HA localizado la marca 5. Dz esperada: ")
                        print(distAMarca)
                        cogiendoDistancia = True
                        Fase4 = False


if (Fase5 == True):
        p = 5
        print("Fase4")
        if ((IDS == 0) == False):
                velocidad = ':-70,-70\n'
                if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
```

```
                                        print ("RECIBIDO: " + eco.decode('ascii'))
                                        if eco.decode('ascii')==velocidad:
                                                print ("Mover")
                                        else :
                                                print ("error!")
                elif ((((ids[0][0])or(ids[0][1]))== 0) == True):

                        velocidad = ':-70,-70\n'
                        if prueba == ('Serial waiting.\n').encode('utf-8'):
                                print("Prueba = serialwaiting")
                                s.write(velocidad.encode('utf-8'))
                                eco = s.readline()

                                if len(eco.decode('ascii'))!=0:
                                        print ("RECIBIDO: " + eco.decode('ascii'))
                                        if eco.decode('ascii')==velocidad:
                                                print ("Mover")
                                        else :
                                                print ("error!")

                                print("HA localizado la marca 0. Dz esperada: ")
                                print(distAMarca)
                                cogiendoDistancia = True
                                Fase5 = False



        if cogiendoDistancia == True:
                velocidad = ':-70,-70\n'
                if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else :
                                        print ("error!")

                        print(distAMarca)
                print("DUrmiendo")
                time.sleep(((distAMarca*100)-7)/9.7222)
                print("Parando")

                velocidad = ':0,0\n'
                if prueba == ('Serial waiting.\n').encode('utf-8'):
                        print("Prueba = serialwaiting")
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()

                        if len(eco.decode('ascii'))!=0:
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else:
                                        print ("error!")
                cogiendoDistancia = False
                girando = True


if (girando == True):
        print("Girando")
        velocidad = ':65,0\n'
        if prueba == ('Serial waiting.\n').encode('utf-8'):
                print("Prueba = serialwaiting")
                s.write(velocidad.encode('utf-8'))
                eco = s.readline()

                if len(eco.decode('ascii'))!=0:
```

```
                                print ("RECIBIDO: " + eco.decode('ascii'))
                                if eco.decode('ascii')==velocidad:
                                        print ("Mover")
                                else :
                                        print ("error!")

        if (p == 0):
                time.sleep(0.93)

        elif ( p == 1):
                #time.sleep (0.7)
                time.sleep (0.93)

        elif (p == 2):
                #time.sleep(0.6)
                time.sleep (0.93)

        elif (p == 3):
                #time.sleep(0.5)
                time.sleep (0.7)

        elif ( p == 4):
                #time.sleep(0.6)
                time.sleep (0.7)

        elif (p == 5):
                time.sleep(1)

        #time.sleep(0.93)
        velocidad = ':0,0\n'
        if prueba == ('Serial waiting.\n').encode('utf-8'):
                print("Prueba = serialwaiting")
                s.write(velocidad.encode('utf-8'))
                eco = s.readline()

                if len(eco.decode('ascii'))!=0:
                        print ("RECIBIDO: " + eco.decode('ascii'))
                        if eco.decode('ascii')==velocidad:
                                print ("Mover")
                        else :
                                print ("error!")
        moviendo = True
        girando = False


        if (p == 0):
                Fase1 = True

        elif (p == 1):
                Fase2 = True


        elif (p == 2):
                Fase3 = True


        elif (p == 3):
                Fase4 = True


        elif (p == 4):
                Fase5 = True

        elif (p == 5):
                Fase0 = True



print("FASES:")
print(Fase0)
print(Fase1)
```

```
print(Fase2)
print(Fase3)
print("Ps")
print(p)
rawCapture.truncate(0)
if key == ord("q"):
        s.write((':0,0\n').encode('utf-8'))
        s.readline()
        s.close()
        break
```

A1.6. MarcaV.py

Su funcionamiento simplificado es el siguiente:

Inicio de la captación continua de imágenes de la cámara.

Si se detectan marcadores:

Calcular parámetros de posición y orientación del robot.

Si se han detectado marcadores alguna vez y se detectan ahora:

Calcular el radio actual

Calcular error y parámetros de corrección

Calcular la nueva velocidad de la rueda interior

Si la velocidad interior calculada se encuentra en un rango adecuado:

Avanzar con la velocidad calculada

SI la velocidad interior es demasiado alta:

La velocidad interior se sustituye por la velocidad máxima

Si la velocidad interior es demasiado baja:

La velocidad interior se sustituye por la velocidad mínima

Si se han detectado marcadores alguna vez y no se detectan ahora:

Si el ángulo del robot con respecto al marcador es menor a 0 (el robot ha girado demasiado hacia el prisma):

Dar marcha atrás medio segundo

Si no (el robot ha girado demasiado alejándose del prisma):

Aumentar la velocidad de la rueda exterior para que el robot gire hacia el prisma durante medio segundo

Si se pulsa la tecla “q”:

Finalizar el programa.

Su código completo es el siguiente:

```
#Programa que hace un circulo controlado alrededor de un objeto,
#guiandose por las marcas colocadas en vertical en su superficie.


import numpy as np
import math
import cv2
import cv2.aruco as aruco
import time
from picamera.array import PiRGBArray
from picamera import PiCamera

import sys
import serial
import RPi.GPIO as GPIO

#K//////////////////////////////////////////////////////////////////

K = 0.0009

#Funciones/////////////////////////////////////////////////////////

def calculoT(tx,ty,tz,rx,ry,rz,rvec):



            CTM_rot = np.zeros(shape=(3,3))
            cv2.Rodrigues(rvec, CTM_rot)
            CTM = np.matrix([[CTM_rot[0][0],CTM_rot[0][1],CTM_rot[0][2], tx],
            [CTM_rot[1][0],CTM_rot[1][1],CTM_rot[1][2],ty],
            [CTM_rot[2][0],CTM_rot[2][1],CTM_rot[2][2],tz],
            [0,0,0,1]])
            MTC = np.linalg.inv(CTM)
            MTR = np.matrix([[1,0,0,MTC.item((0,3))],[0,1,0,MTC.item((1,3))],[0,0,1,MTC.item((2,3))],[0,0,0,1]])
            CTR = ([[ 9.97807577e-01,  1.57754743e-02, -6.42742016e-02 , 0.00000000e+00],
 [-8.76196818e-03, -9.31138067e-01 ,-3.64561555e-01 , 0.00000000e+00],
 [-6.55992873e-02,  3.64325451e-01, -9.28958395e-01, -5.55111512e-17],
 [ 0.00000000e+00 , 0.00000000e+00 , 0.00000000e+00 , 1.00000000e+00]])



            MTR = MTC*CTR


            return MTR

def calculoAngulo(MTR):
            Nz = np.array([MTR.item((2,0))])
            Nx = np.array([MTR.item((0,0))])
            Ny = np.array([MTR.item((1,0))])
            Z = np.arctan2(Ny,Nx)
            Y = np.arctan2(-Nz, Nx*math.cos(Z)+Ny*math.sin(Z))

            return (Y)

def mover(velocidad):
            if prueba == (('Serial waiting.\n').encode('utf-8')):
                        s.write(velocidad.encode('utf-8'))
                        eco = s.readline()
            else:
                        print("Error en serialwaiting")


#Inicializacion del puerto y el GPIO ////////////////////////////////

Puerto = '/dev/ttyUSB0'


try:
```

```
            s = serial.Serial(Puerto, 57600)
            s.timeout=5;

except serial.SerialException:

            sys.stderr.write("Error al abrir puerto (%s)\n" % str(Puerto))

            sys.exit(1)

time.sleep(5)

prueba = s.readline()
print ("PRUEBA: ", prueba.decode('ascii'))

key = cv2.waitKey(1) & 0xFF

#Inicialización de la cámara ////////////////////////////////////////////

camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
camera.rotation = 180
rawCapture = PiRGBArray(camera, size=(640, 480))
time.sleep(0.1)

diccionario = aruco.Dictionary_get(aruco.DICT_6X6_250)

cameraMatrix = np.matrix ([[549.4858516 ,  0.      , 231.95990112],
 [  0.     ,  545.71020483, 119.3904947 ],
 [  0.     ,   0.     ,   1.       ]] )
distCoeffs = np.array([[-1.07019571,  1.76457253,  0.04606337,  0.0281203,  -1.10438634]])

#Inicialización de variables ////////////////////////////////////////////

longMarca = 0.077
distAMarca = 0.33

apotema = 0.13
L = 0.103
Rrueda = 0.062/2
Drueda = 0.062

V = 0.150 #Velocidad lineal (no de la rueda)
R = distAMarca + apotema
w = V/R #Velocidad angular del robot (no de la rueda)


vd0 = V - (L/2)*w
vi0 = V + (L/2)*w

Rd = R - (L/2)
Ri = Rd + L
m = 0.125/90
vi0b = vi0/m             #Velocidad a comunicarle al robot
vd0b = vd0/m

print("Vi0:")
print(vi0b)
print(vi0)
print("Vd0:")
print(vd0b)
print(vd0)

error = 0
vcontrol = 0


i = 0
j = 0
h = 0
```

```
#Inicio del bucle de la función ////////////////////////////////////////

for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
        image = frame.array
        esquinas, ids, reject = aruco.detectMarkers(image, diccionario)
        rvec, tvec, _objPoints = aruco.estimatePoseSingleMarkers(esquinas, longMarca , cameraMatrix, distCoeffs)

        key = cv2.waitKey(1) & 0xFF
        h = 0

        if ((np.all(rvec) and np.all(tvec)) == True):

                if j == 0:
                        j = j + 1
                        encontrada = True
                        resultado = image
                        print("Primera marca detectada")
                        resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)

                        tx=(tvec[0])[0,0]
                        ty=(tvec[0])[0,1]
                        tz=(tvec[0])[0,2]
                        rx=(rvec[0])[0,0]
                        ry=(rvec[0])[0,1]
                        rz=(rvec[0])[0,2]

                        rvecM = rvec[0]
                        if (len(ids) == 1):
                                IDS = ids[0][0]
                        else:
                                 IDS = ((ids[0][0]) or (ids[1][0]))

                elif j > 0:
                        resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                        if (len(ids) == 1):

                                tx=(tvec[0])[0,0]
                                ty=(tvec[0])[0,1]
                                tz=(tvec[0])[0,2]
                                rx=(rvec[0])[0,0]
                                ry=(rvec[0])[0,1]
                                rz=(rvec[0])[0,2]

                                rvecM = rvec[0]
                                IDS = ids[0][0]
                        elif(len(ids) > 1):

                                tx=(tvec[0])[0,0]
                                ty=(tvec[0])[0,1]
                                tz=(tvec[0])[0,2]
                                rx=(rvec[0])[0,0]
                                ry=(rvec[0])[0,1]
                                rz=(rvec[0])[0,2]

                                rvecM = rvec[0]
                                IDS = ((ids[0][0]) or (ids[1][0]))


                MTR = calculoT(tx,ty,tz,rx,ry,rz,rvecM)
                Y = calculoAngulo(MTR)

                dx = MTR.item((0,3))
                dy = MTR.item((1,3))
                dz = MTR.item((2,3))

                nmarcas = len(ids)

        elif ((np.all(rvec) and np.all(tvec)) == None):
                resultado = image
```

```
                nmarcas = 0
                if (j == 0):
                                time.sleep(1)
                IDS = 999


#Hasta aquí, se obtienen los parámetros de posición del robot //////
#Parámetros del controlador a continuación

if ((j > 0) and ((np.all(rvec) and np.all(tvec)) == None) == False):
                distACentro = apotema + dz
                angulo = np.arctan2(dx,distACentro)
                r = dx/math.sin(angulo)


                error = R - r
                vcontrol = K*error
                vcontrolb = vcontrol/m
                vd = vd0 + vcontrolb
                vdb = vd/m



                if ((vdb < 65) and((vdb > - 65))):

                                if (vdb > 0):
                                                vdb = 65
                                elif (vdb < 0):
                                                vdb = -65

                                velocidad = ":" + str(round(vdb)) + "," + str(round(vi0b-20)) + "\n "

                                mover(velocidad)

                                time.sleep(0.5)



                if ((vdb < -255) or ((vdb > 255))):


                                if (vdb > 0):
                                                vdb = 255
                                elif (vdb < 0):
                                                vdb = -255



                if (((vdb >= 65) or (vdb <= - 65)) and ((vi0b >= 65) or (vi0b <= - 65))):

                                velocidad = ":" + str(round(vdb)) + "," + str(round(vi0b)) + "\n "

                                mover(velocidad)

                                i = i + 1

                if ((i % 30) == 0):
                                print("Radio actual:", r)
                                print("Radio deseado:", R)
                                print("Velocidad de la rueda derecha:", vdb)
                                print("Velocidad de la rueda izquierda:", vi0b)


elif ((j > 0) and ((np.all(rvec) and np.all(tvec)) == None)):
                                print("Se ha dejado de ver la marca")
                                print("ÚLTIMO Y = ", Y)


                                if ((h == 0) and (Y < 0)):
```

```
                velocidad = ":-65,-150\n "

                mover(velocidad)

                time.sleep(0.5)

                h = 1

        elif (h == 0):

                velocidad = ":65,150\n "

                mover(velocidad)

                time.sleep(0.5)
                h = 1


cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
cv2.resizeWindow('Image', 1200,1200)
cv2.imshow('Marcadores detectados',resultado)
rawCapture.truncate(0)

if key == ord("q"):
        s.write((':0,0\n').encode('utf-8'))
        s.readline()
        s.close()
        break
```

A1.7. MarcaH.py

El pseudocódigo es el siguiente:

Inicio de la captación continua de imágenes de la cámara.

Si se detectan marcadores:

Calcular parámetros de posición y orientación del robot.

Si se han detectado marcadores alguna vez y se detectan ahora:

Calcular el radio actual

Calcular error y parámetros de corrección

Calcular la nueva velocidad de la rueda interior

Si la velocidad interior calculada se encuentra en un rango adecuado:

Avanzar con la velocidad calculada

SI la velocidad interior es demasiado alta:

La velocidad interior se sustituye por la velocidad máxima

Si la velocidad interior es demasiado baja:

La velocidad interior se sustituye por la velocidad mínima

Si se han detectado marcadores alguna vez y no se detectan ahora:

Si el ángulo del robot con respecto al marcador es mayor a 0.3 (el robot ha girado demasiado hacia el prisma):

Dar marcha atrás medio segundo

Si el ángulo del robot con respecto al marcador es mayor a 0 y menor a 0.3:

Aumentar la velocidad de la rueda exterior y disminuir al máximo la velocidad de la rueda interior para que el robot disminuya el radio

SI el ángulo del robot con respecto al marcador es menor a 0:

Aumentar mucho la velocidad de la rueda exterior y disminuir al máximo la velocidad de la rueda interior para disminuir el radio rápidamente.

Si se pulsa la tecla “q”:

Finalizar el programa.

Su código completo es el siguiente:

```
#Programa que hace un circulo controlado alrededor de una marca colocada
#en el suelo.


import numpy as np
import math
import cv2
import cv2.aruco as aruco
import time
from picamera.array import PiRGBArray
from picamera import PiCamera

import sys
import serial
import RPi.GPIO as GPIO

#K//////////////////////////////////////////////////////////////////

K = 0.0009

#Funciones////////////////////////////////////////////////////////////

def calculoT(tx,ty,tz,rx,ry,rz,rvec):
        CTM_rot = np.zeros(shape=(3,3))
        cv2.Rodrigues(rvec, CTM_rot)
        CTM = np.matrix([[CTM_rot[0][0],CTM_rot[0][1],CTM_rot[0][2], tx],
        [CTM_rot[1][0],CTM_rot[1][1],CTM_rot[1][2],ty],
        [CTM_rot[2][0],CTM_rot[2][1],CTM_rot[2][2],tz],
        [0,0,0,1]])
        MTC = np.linalg.inv(CTM)
        MTR = np.matrix([[1,0,0,MTC.item((0,3))],[0,1,0,MTC.item((1,3))],[0,0,1,MTC.item((2,3))],[0,0,0,1]])
        CTR = ([[ 9.97807577e-01,  1.57754743e-02, -6.42742016e-02 , 0.00000000e+00],
 [-8.76196818e-03, -9.31138067e-01 ,-3.64561555e-01 , 0.00000000e+00],
 [-6.55992873e-02,  3.64325451e-01, -9.28958395e-01, -5.55111512e-17],
 [ 0.00000000e+00 , 0.00000000e+00 , 0.00000000e+00 , 1.00000000e+00]])

        MTR = MTC*CTR


        return MTR

def calculoAngulo(RTM):
        Nz = np.array([RTM.item((2,0))])
        Nx = np.array([RTM.item((0,0))])
        Ny = np.array([RTM.item((1,0))])
        Z = np.arctan2(Ny,Nx)
```

```
        Y = np.arctan2(-Nz, Nx*math.cos(Z)+Ny*math.sin(Z))

        return (Y)

def mover(velocidad):
        if prueba == (('Serial waiting.\n').encode('utf-8')):
                s.write(velocidad.encode('utf-8'))
                eco = s.readline()
        else:
                print("Error en serialwaiting")


#Inicializacion del puerto y las GPIO ///////////////////////////////////

Puerto = '/dev/ttyUSB0'


try:
        s = serial.Serial(Puerto, 57600)
        s.timeout=5;

except serial.SerialException:

        sys.stderr.write("Error al abrir puerto (%s)\n" % str(Puerto))

        sys.exit(1)

time.sleep(5)

prueba = s.readline()
print ("PRUEBA: ", prueba.decode('ascii'))

key = cv2.waitKey(1) & 0xFF

#Inicialización de la cámara ///////////////////////////////////////////

camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
camera.rotation = 180
rawCapture = PiRGBArray(camera, size=(640, 480))
time.sleep(0.1)

diccionario = aruco.Dictionary_get(aruco.DICT_6X6_250)

cameraMatrix = np.matrix ([[549.4858516 ,  0.      , 231.95990112],
 [  0.     ,  545.71020483, 119.3904947 ],
 [  0.     ,    0.     ,    1.       ]] )
distCoeffs = np.array([[-1.07019571,  1.76457253,  0.04606337,  0.0281203,  -1.10438634]])

#Inicialización de variables ///////////////////////////////////////////


longMarca = 0.15 #Longitud del lado del marcador Aruco.
distAMarca = 0.22

L = 0.103
Rrueda = 0.062/2
Drueda = 0.062

V = 0.125
R = distAMarca
P = 2*math.pi*(R)
tiempoCirculo = P/V
w = V/R

vd0 = V - (L/2)*w
vi0 = V + (L/2)*w

m = 0.125/90
vi0b = vi0/m
```

```
vd0b = vd0/m

print("Vi0:")
print(vi0b)
print(vi0)
print("Vd0:")
print(vd0b)
print(vd0)

error = 0
vcontrol = 0


i = 0
j = 0
h = 0


#Inicio del bucle de la función ////////////////////////////////////////

for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
        image = frame.array
        esquinas, ids, reject = aruco.detectMarkers(image, diccionario)
        rvec, tvec, _objPoints = aruco.estimatePoseSingleMarkers(esquinas, longMarca , cameraMatrix, distCoeffs)

        key = cv2.waitKey(1) & 0xFF
        h = 0


        if ((np.all(rvec) and np.all(tvec)) == True):

                if j == 0:
                        j = j + 1
                        encontrada = True
                        resultado = image
                        print("Primera marca detectada")
                        resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)

                        tx=(tvec[0])[0,0]
                        ty=(tvec[0])[0,1]
                        tz=(tvec[0])[0,2]
                        rx=(rvec[0])[0,0]
                        ry=(rvec[0])[0,1]
                        rz=(rvec[0])[0,2]

                        rvecM = rvec[0]
                        if (len(ids) == 1):
                                IDS = ids[0][0]
                        else:
                                 IDS = ((ids[0][0]) or (ids[1][0]))

                elif j > 0:
                        resultado = aruco.drawDetectedMarkers(image, esquinas, ids,4)
                        if (len(ids) == 1):

                                tx=(tvec[0])[0,0]
                                ty=(tvec[0])[0,1]
                                tz=(tvec[0])[0,2]
                                rx=(rvec[0])[0,0]
                                ry=(rvec[0])[0,1]
                                rz=(rvec[0])[0,2]

                                rvecM = rvec[0]
                                IDS = ids[0][0]
                        elif(len(ids) > 1):

                                tx=(tvec[0])[0,0]
                                ty=(tvec[0])[0,1]
                                tz=(tvec[0])[0,2]
                                rx=(rvec[0])[0,0]
                                ry=(rvec[0])[0,1]
```

```
                                rz=(rvec[0])[0,2]

                                rvecM = rvec[0]
                                IDS = ((ids[0][0]) or (ids[1][0]))


                MTR = calculoT(tx,ty,tz,rx,ry,rz,rvecM)
                RTM = np.linalg.inv(MTR)
                Y = calculoAngulo(RTM)

                dx = MTR.item((0,3))
                dy = MTR.item((1,3))
                dz = MTR.item((2,3))

                print("Dx", dx)
                print("Dy", dy)
                print("Y", Y)

                nmarcas = len(ids)

elif ((np.all(rvec) and np.all(tvec)) == None):
                resultado = image
                nmarcas = 0
                if (j == 0):
                                time.sleep(1)
                IDS = 999
                if (j > 0):
                                print("No se ve la marca. Los últimos parámetros son: ")
                                print("Dx", dx)
                                print("Dy", dy)
                                print("Y", Y)


#Hasta aquí, se obtienen los parámetros de posición del robot //////
#Parámetros del controlador a continuación

if ((j > 0) and ((np.all(rvec) and np.all(tvec)) == None) == False):


                angulo = np.arctan2(dy,dx)

                if ((dx>0) and (dy>0)):
                                angulo = angulo

                elif ((dx>0) and (dy<0)):
                                angulo = (2*math.pi)+angulo

                elif ((dx<0) and (dy<0)):
                                angulo = 180 + angulo

                elif ((dx<0) and (dy>0)):
                                angulo = (math.pi/2) + angulo


                r = dy/math.sin(angulo)
                rm = r + 0.0715

                error = R - r
                vcontrol = K*error
                vcontrolb = vcontrol/m
                vd = vd0 + vcontrolb
                vdb = vd/m
                print("Radio actual:", r)



                if ((vdb < 65) and((vdb > - 65))):
                                #print("La v derecha es demasiado baja, se ha cambiado a 65")

                                if (vdb > 0):
                                                vdb = 65
```

```
                elif (vdb < 0):
                        vdb = -65

                velocidad = ":" + str(round(vdb)) + "," + str(round(vi0b-20)) + "\n "

                mover(velocidad)

                time.sleep(0.5)


        if ((vdb < -255) or ((vdb > 255))):
                #print("La v derecha demasiado alta, se ha cambiado a 255")

                if (vdb > 0):
                        vdb = 255
                elif (vdb < 0):
                        vdb = -255


        if (((vdb >= 65) or (vdb <= - 65)) and ((vi0b >= 65) or (vi0b <= - 65))):

                velocidad = ":" + str(round(vdb)) + "," + str(round(vi0b)) + "\n "


                mover(velocidad)

                i = i + 1

        if ((i % 30) == 0):
                print("Radio actual:", r)
                print("Radio deseado:", R)
                print("Velocidad de la rueda derecha:", vdb)
                print("Velocidad de la rueda izquierda:", vi0b)


elif ((j > 0) and ((np.all(rvec) and np.all(tvec)) == None)):
                print("Se ha dejado de ver la marca")
                print("ÚLTIMO Y = ", Y)


                if ((h == 0) and (Y > 0.3)):

                        velocidad = ":-65,-100\n "

                        mover(velocidad)

                        time.sleep(0.5)
                        h = 1

                #elif ((h == 0) and ((Y < 0.3) and (Y > 0))): #Se cierra

                elif ((h == 0) and ((Y > 0)) and (Y < 0.3)): #Se cierra
                        velocidad = ":65,90\n "

                        mover(velocidad)

                        time.sleep(0.5)
                        h = 1


                elif ((h == 0) and ((Y < 0))): #Se cierra

                        velocidad = ":65,125\n "

                        mover(velocidad)

                        time.sleep(0.5)
                        h = 1
```

```
                    print("Velocidad", velocidad)

cv2.namedWindow('Marcadores detectados', cv2.WINDOW_NORMAL)
cv2.resizeWindow('Image', 1200,1200)
cv2.imshow('Marcadores detectados',resultado)
rawCapture.truncate(0)

if key == ord("q"):
        s.write((':0,0\n').encode('utf-8'))
        s.readline()
        s.close()
        break
```

A2: Calibración de la cámara

A2.1. Fundamentos

Las cámaras de bajo coste suelen introducir distintos tipos de distorsión en las imágenes que toman. Esto afecta negativamente al reconocimiento de objetos y al cálculo de posiciones y distancias por lo que, antes de empezar con las aplicaciones de visión artificial, es necesario calibrar la cámara para eliminar estos defectos.

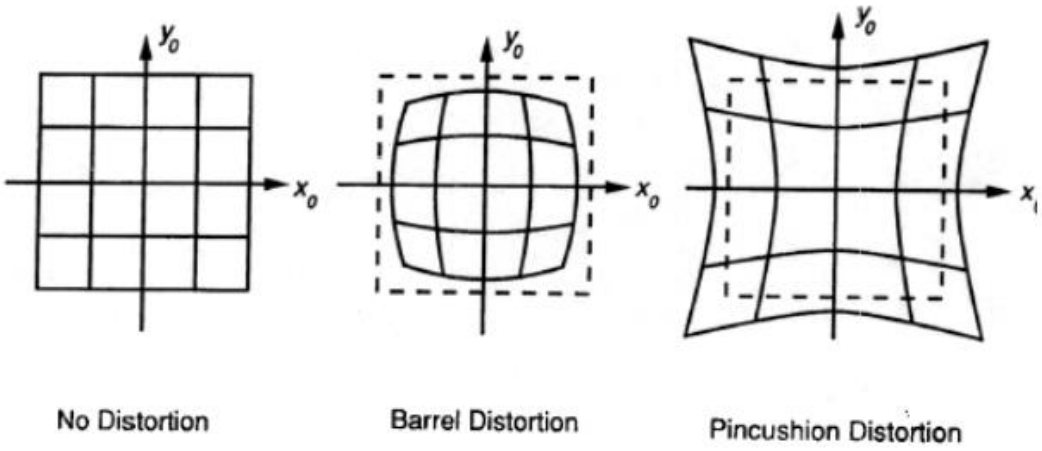


Figura 31: Imagen sin distorsión, con distorsión radial y con distorsión tangencial.

Como se puede observar en la figura 31, hay dos tipos principales de distorsión: la radial y la tangencial. En la distorsión radial, las líneas que deberían ser rectas aparecen curvadas, dando la impresión de que la imagen tiene forma de barril. En la distorsión tangencial, ocurre lo contrario. Para corregir estos defectos, se aplican cuatro ecuaciones, dos para cada tipo de distorsión:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

Donde las dos primeras corresponden a la distorsión radial y las dos últimas, a la tangencial.

El proceso de calibración es aquel que permite obtener los cinco parámetros desconocidos que aparecen en estas fórmulas, llamados *coeficientes de distorsión* o *distortion coefficients:*

$$Distortion\ coefficients = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

Además, mediante este procedimiento se obtienen los parámetros intrínsecos y extrínsecos de la cámara, con los que se puede conformar la matriz de la cámara o *camera matrix.* Los parámetros intrínsecos aportan información sobre la distancia focal, centros ópticos o configuración del dispositivo. Los parámetros extrínsecos, por otro lado, son vectores de rotación y traslación utilizados para generar sistemas de coordenadas.

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

A no ser que se cambie la configuración de la cámara, estos datos no cambiarán, por lo que no será necesario calibrar la cámara más de una sola vez.

A2.2. Software

Tanto OpenCV como el módulo Aruco cuentan con funciones diseñadas para hallar estos parámetros, y devolver al usuario un vector con los coeficientes de distorsión y una matriz de cámara.

El programa utilizado para calibrar la cámara, llamado Calibracion.py, es una modificación del código que se puede encontrar en la bibliografía (6). Este programa crea una lista de fotografías, previamente obtenidas con Generación.py, que luego utiliza como argumento de entrada en la función aruco.calibrateCameraAruco. Cuando termina el proceso, devuelve la matriz de la cámara y los coeficientes de distorsión por pantalla.

A2.3. Proceso de calibración

La primera vez, es necesario crear una tabla de marcadores Aruco del diccionario que se quiera utilizar, mediante el programa *GenerateBoard.py*. Si ya está creada, se debe imprimir y colocar en una superficie lisa y bien iluminada.

Primero, se deben colocar los programas *Calibración.py* y *Generación.py* en una misma carpeta. Después, se deben sacar alrededor de 50 fotografías de la tabla, en distintas posiciones y ángulos, mediante *Generación.py*. Estas fotografías se guardan numeradas en la carpeta en la que se encuentra el programa, y en ellas debe ser posible la detección de, al menos, un marcador. Finalmente, se ejecuta *Calibración.py*, que devuelve los coeficientes de distorsión y la matriz de la cámara. Se recomienda guardar ambos en un archivo de texto para ir introduciéndolos en todos los programas que los requieran.

A3: Problemas encontrados

Este anexo muestra una lista de problemas que fueron surgiendo a lo largo de la realización de este trabajo junto con su solución (si se encontró), con el objetivo de ahorrar tiempo a futuros usuarios que vuelvan a encontrarse con ellos.

A3.1. Versiones de python

Como ya se ha mencionado, la versión de Python que trae de serie el sistema operativo es la 2.7.9, lo que quiere decir que todas las librerías y módulos utilizados se encuentran adaptados a Python 2, así como el controlador que se iba a utilizar inicialmente.

Sin embargo, para este proyecto fueron necesarias funciones que venían asociadas a versiones de Opencv basadas en Python 3 o posterior, por lo que en un principio se optó por actualizar el sistema en su totalidad. Poco después de hacerlo, se intentó ejecutar uno de los programas del controlador y se descubrió que, a pesar de seguir teniendo instalado Python 2, los módulos requeridos por el programa daban error, lo que no ocurría en programas de Python 3.

Para solucionar este problema, se revirtió el sistema a su situación original mediante una copia de seguridad y se instaló una versión de OpenCV dentro de un entorno virtual, que mantiene separadas las librerías y módulos de Python 2 y Python 3. Todos los módulos y librerías necesarias para este proyecto se encuentran también dentro de dicho entorno.

A3.2. Resolución de la cámara

Por defecto, la distancia focal de la cámara viene ajustada a infinito. Esto provoca que las imágenes cercanas se vean borrosas, y obstaculiza la detección de marcas cuando el robot se encuentra lejos de ellas.

Es posible ajustar la distancia focal manualmente (7), pero durante este trabajo no se disponía de las herramientas necesarias, por lo que no fue posible. Se solucionó manteniendo el marcador relativamente cerca de la cámara.

A3.3. Problemas de calibración

Tras calibrar cinco veces y comprobar los resultados, se descubrió que ninguna combinación de matriz de cámara + coeficientes de distorsión era completamente exacta. Todas tenían, en mayor o menor medida, fallos a la hora de calcular las distancias hasta el marcador. Algunas eran muy precisas al medir longitudes en el eje Z de la cámara, pero muy defectuosas en su eje X, y otras fallaban unos pocos centímetros en ambos. Para el programa VeAMarca3.py, se utilizó una de las primeras, ya que sólo era necesario medir el eje Z, mientras que para el resto se utilizó una de las segundas. Como resultado, el robot no se ajusta al radio exacto que introduce el usuario.

A3.4. Raspberry pide la contraseña una y otra vez al inicio

Este problema surge cuando el ordenador se queda “atascado” en el usuario root. Al iniciarse, pide la contraseña (que por defecto es “raspberry”) una y otra vez, mostrando

una pantalla en negro sin dejar entrar al escritorio. En este estado, sólo es posible acceder mediante la consola, pulsando CTRL+ALT+F6 y escribiendo sudo startx.

Para solucionarlo, se siguieron estos pasos:

- Encender la raspberry. En la pantalla de contraseña, pulsar CTRL+ALT+F2 e introducir el nombre de usuario (pi) y la contraseña (raspberry)
- Teclear sudo chown pi:pi .Xauthority
- Teclear startx.

Tras esto, la pantalla se pone negra. Esperar un rato y, si sigue así, apagar y volver a encender. Ya debería funcionar adecuadamente.

Para evitar este problema, se debe salir del modo root (si se ha utilizado *sudo su* para acceder a él) antes de apagar el ordenador.

A3.5. Memoria insuficiente

A la hora de instalar OpenCV, uno de los errores más frecuentes se debe a la insuficiente memoria. Para solucionarlo, se cambió la tarjeta de 8 GB por una de 32, y después se expandió a través de la configuración de la tarjeta.

A3.6. Problemas de conexión con el monitor

A veces, al enchufar la raspberry a un monitor, éste no mostraba nada por pantalla, por mucho que se desconectase y se volviese a conectar. Al final se descubrió que esto sólo ocurría cuando la raspberry se había iniciado sin ningún monitor enchufado. El 100% de las veces que se iniciaba con el cable HDMI conectado, luego podía desconectarse libremente y volver a conetarse, y el monitor mostraba la pantalla adecuadamente.

Por otro lado, a la hora de usar VNC, si la raspberry se inicia sin un monitor conectado, la ventana que aparece es muy pequeña. Si se inicia con un monitor conectado, VNC muestra la resolución de dicho monitor, haciendo más cómodo el trabajo.

A4: Manual de usuario.

En este anexo se detalla cómo ejecutar un programa desde cero.

Primero, hay que inicial la raspberry pulsando el botón de la batería o conectándole el cable de alimentación si no lo tenía conectado previamente. No debería pedir contraseña para el usuario pi. Si lo hace, introducir *raspberry*[9].

Si el programa que se va a ejecutar hace que el robot se traslade, es recomendable utilizar VNC para realizar los pasos siguientes. En este caso, hay que encender el equipo desde el cual se va a controlar el robot y ejecutar VNC conectarse a la raspberry, desconectar el monitor y cualquier tipo de cable conectado al robot y seguir los pasos siguientes. Si el programa no va a producir movimiento, no es necesario utilizar VNC (aunque es posible).

Una vez en el escritorio, se abre la terminal y se inicia el entorno virtual tal y como se ha visto anteriormente:

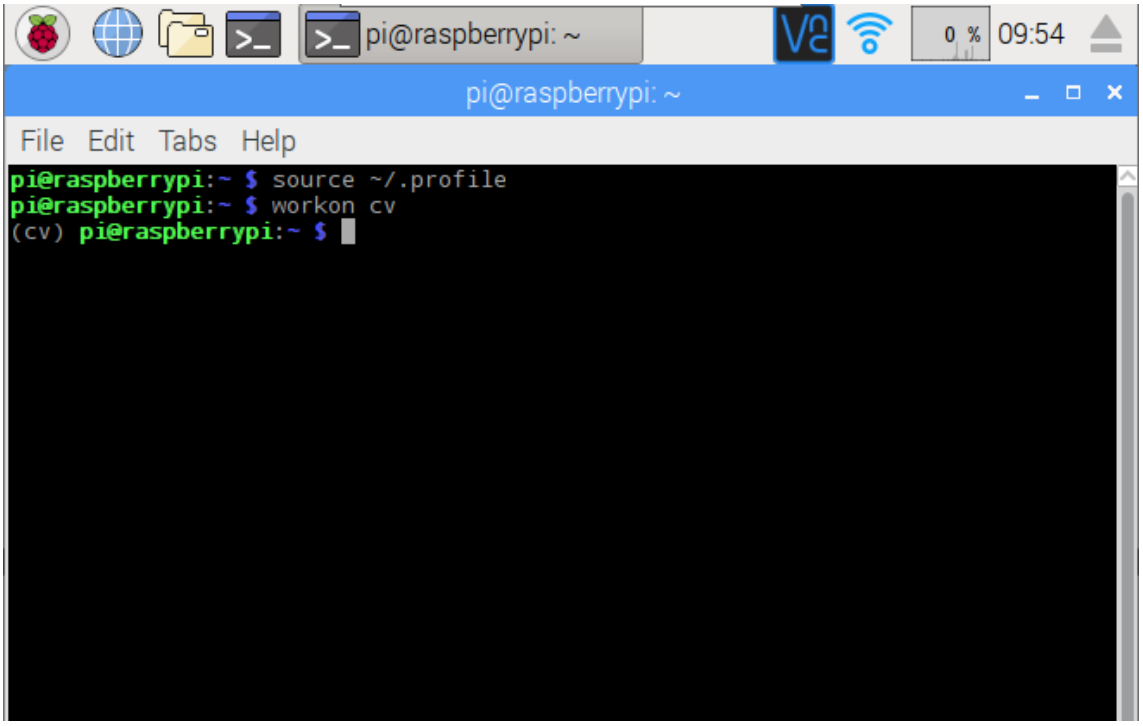


Figura 32: Activación del entorno virtual

Una vez activado, aparecerá al inicio de las líneas el icono (cv).

Posteriormente, hay que situarse en el directorio en el que se encuentra el programa a ejecutar. Para ello, teclear la orden *cd* seguida del directorio. El directorio puede copiarse de la parte superior de la carpeta y pegarse seguidamente[10].

Finalmente, hay que escribir *python* seguido del nombre del archivo, acabado en *.py*, y pulsar enter. Dentro del entorno virtual no hace falta especificar la versión, puesto que se ejecuta Python 3 por defecto. Fuera del entorno debería escribirse *python2* o *python3.*

[9] Si no funciona, ver A3.4.

[10] Para copiar o pegar algo en la terminal no es suficiente con CTRL + C o CTRL + V, hace falta pulsar también SHIFT (flecha hacia arriba).

Por ejemplo, si se quiere ejecutar MarcaV.py, dentro de /home/pi/Lucia/Movimiento…

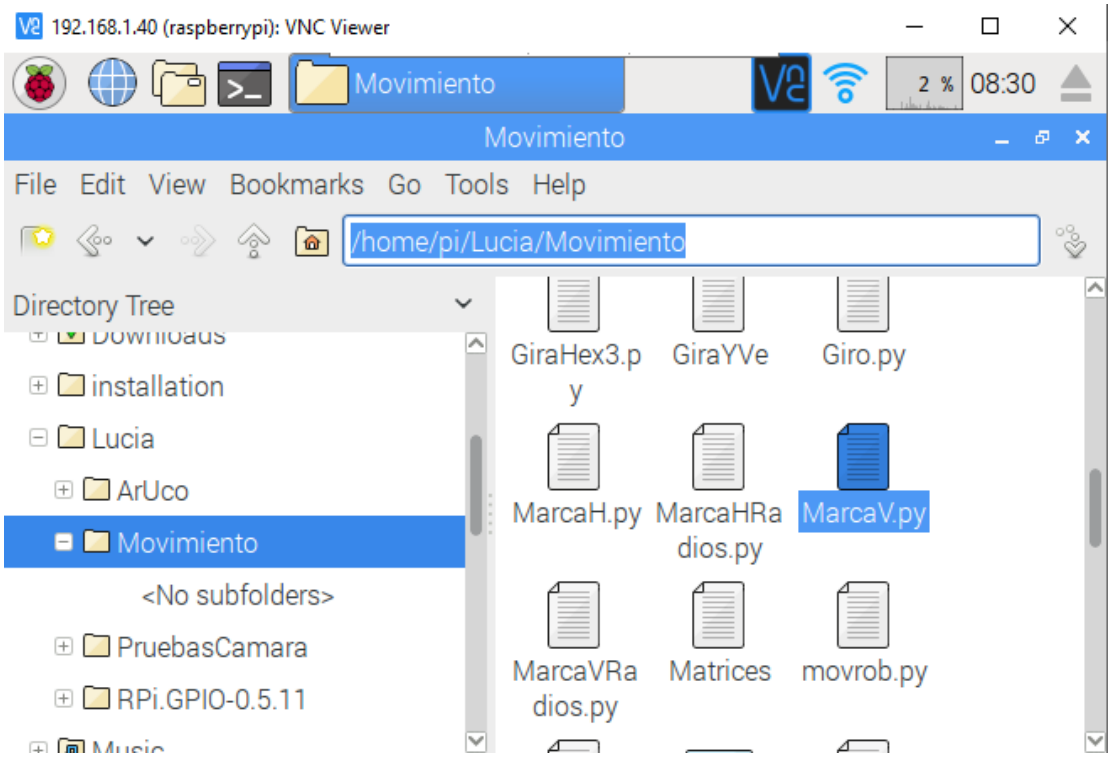


Figura 33: Obtención del directorio

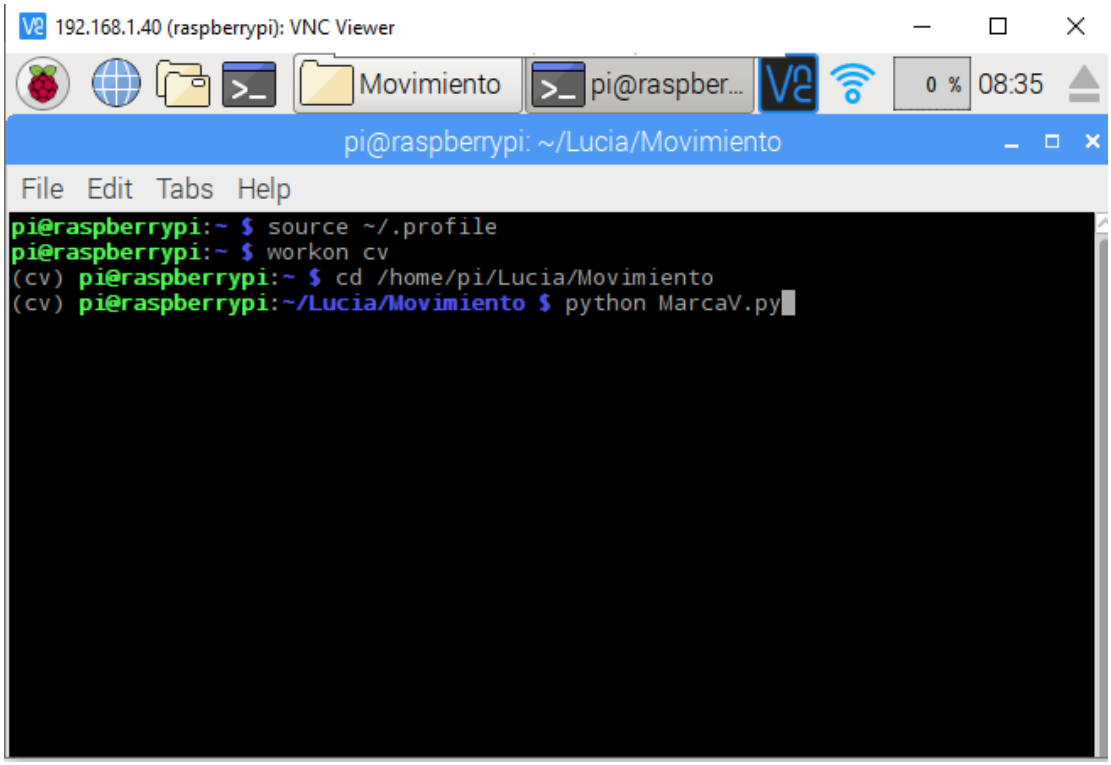


Figura 34: Inicialización del programa

Para ejecutarlo fuera del entorno virtual hay que hacer lo mismo, evitando introducir los dos primeros comandos.

La figura 34 muestra unos segundos de ejecución de MarcaV.py, tras los cuales se abortó el programa:

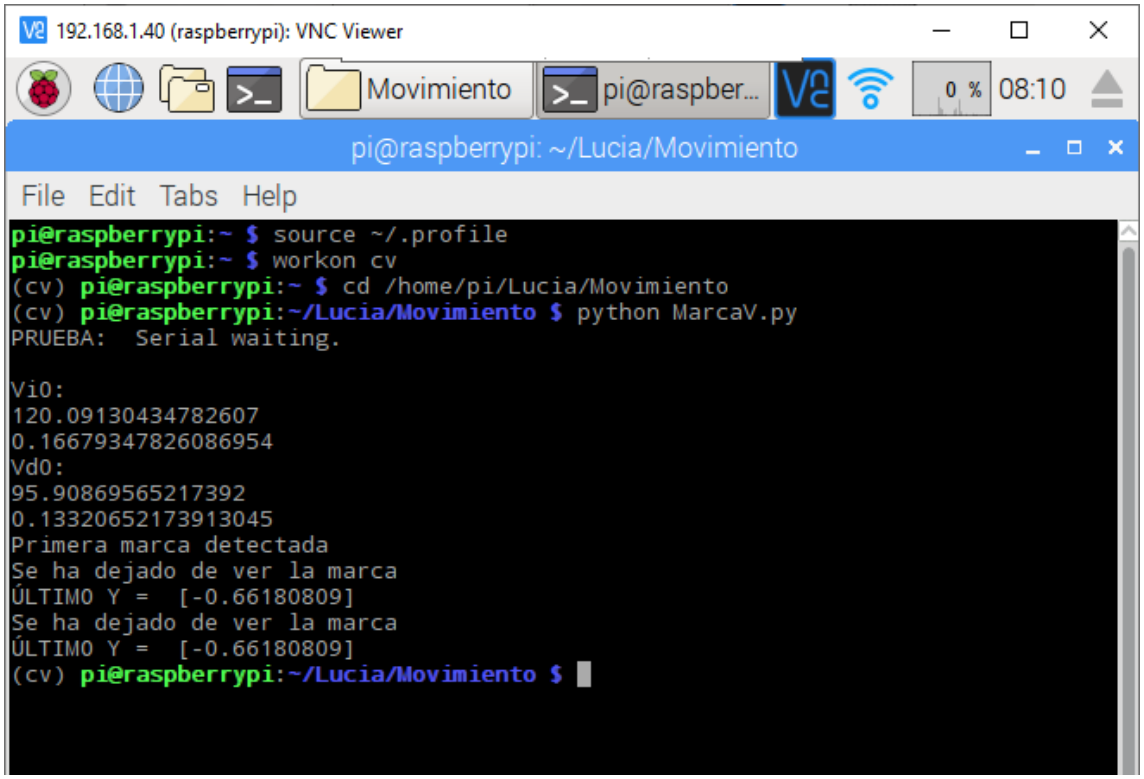


Figura 35: Ejecución de MarcaV.py. Durante el programa, además de lo mostrado, aparece una ventana adicional que retransmite lo que ve la cámara.

Bibliografía

1. *Speeded up detection of squared fiducial markers.* **Francisco J.Romero-Ramírez, Rafael Muñoz Salinas, Rafael Medina-Carnicer.** 2018, Image and Vision Computing, Vol. 76, págs. 38-47.

2. *"Generation of fiducial marker dictionaries using mixed integer linear programming".* **S. Garrido-Jurado, R. Muñoz Salinas, F.J. Madrid-Cuevas, R. Medina-Carnicer.** 2016, Pattern Recognition, Vol. 51, págs. 481-491.

3. **Rubio Roig, Carlos.** Sistema multi-robot para cobertura persistente. *Universidad de Zaragoza.* [En línea] 2018. http://zaguan.unizar.es/record/76330?ln=es.

4. **Ruiz Altabella, Pedro.** Navegación de un robot móvil basada en odometría utilizando Encoder diferencial e IMU. *Universidad de Zaragoza.* [En línea] Mayo de 2018. https://zaguan.unizar.es/record/78687?ln=es.

5. **Rafael Muñoz Salinas, Sergio Garrido, AVA Group.** Sourceforge. [En línea] https://sourceforge.net/projects/aruco/files/.

6. **Padalkar, Abhishek.** GitHub. [En línea] https://github.com/abhishek098/camera_calibration.

7. **Geerling, Jeff.** Jeff Geerling. [En línea] 16 de Junio de 2017. https://www.jeffgeerling.com/blog/2017/fixing-blurry-focus-on-some-raspberry-pi-camera-v2-models.

8. **Rosebrok, Adrian.** pyimagesearch. [En línea] 23 de Febrero de 2015. https://www.pyimagesearch.com/2015/02/23/install-opencv-and-python-on-your-raspberry-pi-2-and-b/.

9. **Universidad de Córoba.** AVA (Aplicaciones de la Visión Artificial). [En línea] https://www.uco.es/investiga/grupos/ava/node/26.