Universidad Zaragoza
1542

Trabajo Fin de Grado

Navegación de un robot móvil basada en odometría utilizando Encoder diferencial e IMU

Navigation of a mobile robot based on odometry using Differential Encoder and IMU

Autor

Pedro Ruiz Altabella

Director

Gonzalo López Nicolás

Ingeniería de Tecnologías Industriales

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura

Mayo 2018

Navegación de un robot móvil basada en odometría utilizando Encoder diferencial e IMU.

RESUMEN

La robótica es un sector que ha experimentado un gran crecimiento en los últimos años. Los robots han pasado de usarse exclusivamente para tareas muy específicas en fábricas a usarse para multitud de tareas en las que su incorporación pueda aportar alguna ventaja económica, lúdica o de seguridad. Un ejemplo son los robots aspiradores conocidos como "roomba". Ejemplos como éste han impulsado que los robots domésticos y los robots de ocio aparezcan con fuerza en el mercado llegando a ser habitual encontrarlos en muchos hogares.

Un requerimiento habitual es la capacidad del robot de navegar de forma precisa siguiendo trayectorias predefinidas. En este Trabajo Fin de Grado se dispone de un robot móvil a controlar por un dispositivo Raspberry Pi (denominándose el conjunto Pi Robot). El objetivo principal del trabajo es desarrollar un sistema de control basado en sensores propioceptivos para dicho robot que permita realizar trayectorias definidas por el usuario. Para la consecución de esta tarea se implementarán dispositivos de bajo coste que permitan que la plataforma robótica sea accesible en coste, facilitando la posibilidad de crear equipos de múltiples robots. En particular, se aborda la instalación y uso de dos dispositivos: Encoder diferencial y unidad de medición inercial (IMU).

Con estos dispositivos se estudia y analiza el comportamiento dinámico del robot definiendo un modelo adecuado. El modelo se valida experimentalmente analizando su eficacia ante comportamientos no lineales como la saturación de la acción o zona muerta. Utilizando este modelo, se desarrolla e implementa un control que permite realizar movimientos con suficiente precisión como para realizar las trayectorias deseadas por el usuario. En particular, se diseñan algoritmos de control con los que se implementan diferentes controles (de velocidad y de posición) para un funcionamiento robusto. Para conseguir un ajuste de parámetros adecuados y un funcionamiento robusto, se analiza el uso de los diferentes tipos de información adquirida con los sensores mediante los distintos controles.

Además de estudiar y crear programas para el Pi Robot utilizado, los programas que definen el modelado del sistema se implementan de forma general. Por lo tanto, su uso es directamente utilizable en otros robots con distintos parámetros dinámicos mediante el uso de las funciones de calibración. La evaluación experimental de los algoritmos de control se realiza con una serie de programas que controlan al robot desde su posición actual hasta una posición definida por el usuario o siguiendo una trayectoria predefinida.

Índice general

Parte I: Memoria

Parte II: Apéndices

Memoria

Capítulo 1

Introducción

1.1 Contexto y motivación

A día de hoy las tecnologías avanzan a un ritmo fulgurante. Lo que antes tardaba días, incluso semanas, se puede obtener ahora en pocos minutos debido a la eficiencia de los equipos con los que se trabaja, y la amplia información que se tiene en casi cualquier campo.

Esto hace que continuamente se esté avanzando, y lo que antes era novedoso en un corto periodo de tiempo pasa a quedarse obsoleto. Eso ocurre con los robots; hace unos años su única misión era simplificar el trabajo en las factorías, eran piezas voluminosas y de precios desorbitados. Hoy en día, hay robots para casi todo, y de ser una herramienta únicamente usada en la industria ha pasado a ser usada también en el hogar e incluso para aplicaciones lúdicas y de ocio.

Este trabajo fin de grado usa uno de estos robots de uso lúdico. El robot que se usará se observa en la figura 1.1 y será un robot controlado por una Raspberry Pi (conocido como Pi Robot). Con el trabajo descrito en esta memoria se pretende controlar su comportamiento, conocer más acerca de él, de cómo funciona y hacerlo una herramienta precisa y fiable con la incorporación de dispositivos de bajo coste.

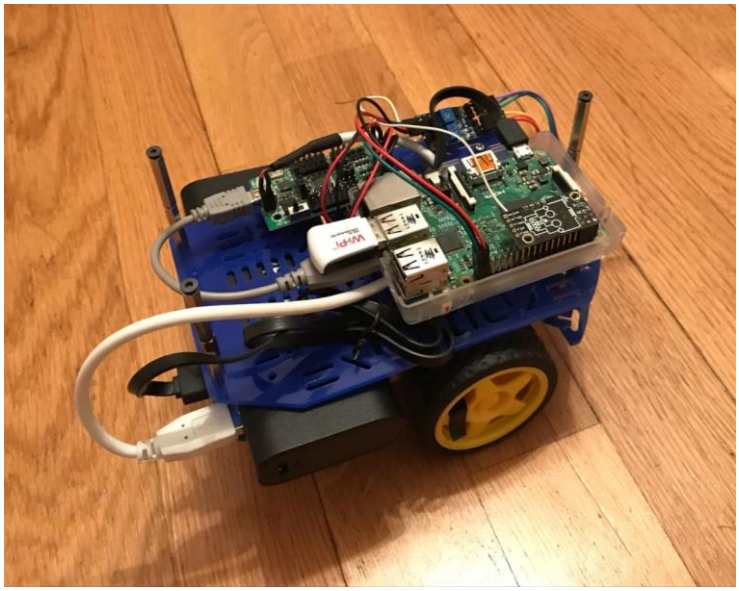

Figura 1.1. Imagen del Pi Robot usado en este trabajo con todos los dispositivos instalados.

Este trabajo puede extrapolarse a robots autónomos de uso industrial debido a que la motivación de este trabajo fin de grado es conseguir realizar trayectorias fiables ya sea en un hogar, en una gran fábrica o en un almacén.

1.2 Objetivos

El objetivo principal de este trabajo fin de grado es desarrollar una serie de programas que hagan que podamos hacer trayectorias precisas con nuestro Pi Robot. Estas trayectorias pueden ir desde una trayectoria en línea recta en la que se recorra una distancia definida por el usuario, hasta que el Pi Robot vaya a una determinada coordenada. De igual modo se puede ordenar una trayectoria curva indicándole alguna característica precisa de la misma como el radio.

Como se verá más adelante se hará un control de velocidad de manera que se podrá introducir la velocidad a la que se desea mover nuestro robot. Este robot puede aceptar entradas en tensión adaptadas y discretizadas entre 0 a 255. El valor 0 corresponderá a la entrada más baja posible a la que el Robot no recibiría tensión (0 V) y en consecuencia no se movería; mientras que 255 sería la entrada que le proporcionaría la máxima velocidad, equivalente a proporcionarle 5 V. Cada motor tendrá su propia entrada de modo que si se desean mover ambas ruedas del Pi Robot se deberán mandar dos entradas.

Otro objetivo complementario que forma parte de la realización de este proyecto es el de analizar y modelar el comportamiento de los motores que impulsan este tipo de robots. A través de dispositivos de medida asequibles económicamente se harán experimentos que nos darán valiosa información para poder modelarlo con funciones y herramientas informáticas.

Con este Trabajo Fin de Grado se buscan todos estos objetivos, y no solo en este Pi Robot en particular, sino también en cualquier robot que tenga características semejantes a las del nuestro. Es por ello que los programas que definen el modelado del sistema son de carácter general. De manera que introduciendo los programas en otro robot se conseguiría calibrar y modelar dicho robot, aunque con otros parámetros diferentes a los de nuestro Pi Robot.

A continuación se dejarán dos enlaces en los que se podrá ver en primera instancia el comportamiento del robot sin control cuando realiza dos trayectorias básicas como pueden ser una línea recta y un círculo. Los vídeos mostrarán muy claramente los problemas que presenta este dispositivo si no es controlado por ningún dispositivo. Los enlaces a los videos son:

Video realizando el Pi Robot una trayectoria recta sin control:

https://www.dropbox.com/home/TFG/videos_trayectoria_recta?preview=Recta_2m_NC.mp4

Video realizando el Pi Robot una trayectoria circular sin control (círculo a realizar delimitado por cinta aislante en el suelo):

https://www.dropbox.com/home/TFG/videos_trayectoria_curva?preview=Curva_r50cm_10s_2circulos_NC.mp4

1.3 Alcance y entorno de trabajo

Para la realización de este trabajo se han implementado en nuestro robot dos dispositivos que ayudarán a conseguir movimientos más precisos y controlados.

Los dos dispositivos principales en los que se apoyará la realización de este proyecto son dos:

(i) un encoder de dos puertos de la industria británica “Pridopia”[1] y
(ii) una unidad de medición inercial, conocida por su abreviatura (IMU) de la firma Ozzmaker[2].

Los lectores del encoder se colocarán cerca de los motores para poder medirlos (esta colocación se explica en el apéndice A) mientras que el chip se colocará atornillado en la placa. En lo referente a la IMU, como su misión es medir ángulos, debe ir en una posición fija de manera que no pueda moverse ni oscilar. Para ello se colocará en los primeros 6 pines de la placa Raspberry Pi como indica la figura 1.2. Este Trabajo Fin de Grado no alcanza, por ejemplo, la implementación de una cámara con la que se podría hacer alguna aplicación relacionada con visión y superación de obstáculos.

En cuanto al entorno en el que se va a desarrollar este proyecto, se cuenta con un Pi Robot, donde su parte más importante es su ordenador, una Raspberry Pi 3. Cuenta con diversos puertos y conexiones que hacen muy fácil la comunicación con otros dispositivos como por ejemplo cuatro puertos USB a los que poder conectar distintos periféricos así como un dispositivo que permita conexiones a una Wi-fi, etc. Otro puerto de gran utilidad es el HDMI. Para la instalación y los primeros pasos se ha usado como referencia [1].

Figura 1.2. Colocación de la IMU en los puertos de la Raspberry Pi. La IMU se coloca en los 6 primeros puertos.

Además, se cuenta con motores de la marca “Dagu”[3]. Son pequeños motores de 5V, que en Python pueden recibir una entrada comprendida entre 0-255, siendo 255 el equivalente a 5V. Estos motores tienen una relación de transformación de 48:1 y son controlados por la Raspberry Pi. Ésta es la encargada de mandarle las acciones que son entradas escalón.

Por otra parte, nuestro Pi Robot cuenta con 3 ruedas. Las dos ruedas de plástico corresponden a las ruedas motrices, mientras que la última es una rueda loca de metal en la parte delantera (basada en una rótula).

[1] http://www.pridopia.co.uk/pi-motor-encoder-2p-2w.html
[2] http://ozzmaker.com/berryimu-quick-start-guide/
[3] http://www.dagurobot.com/goods.php?id=166

Las ruedas motrices tienen un diámetro aproximadamente de 6.3 cm, por lo que la longitud de su circunferencia será de 19.79 cm aproximadamente. Dado que los discos medidores de nuestro encoder tienen 20 ranuras (dientes), cada ranura equivaldrá a una distancia en centímetros de 0.9895 cm/ranura. Esta conversión se usará con mucha frecuencia en cualquier programa por lo que se le asignará a una constante denominada "*cm*".

El chasis estará formado, como se ve en la figura 1.3, por dos láminas de plástico azul. Estas tablas tienen infinidad de huecos o aberturas por donde pasarán los diferentes cables que vayan a algún dispositivo, o al motor etc. Ambas estarán unidas mediante varios tornillos y en su interior, justo encima de la rueda loca se alojará una batería modelo "PowerBank" que será la que suministre energía al conjunto.

Figura 1.3. Chasis de la estructura donde se puede observar las dos placas que sirven como cuerpo principal unidas por tornillos y agujereadas ambas para dejar pasar entre ellas los distintos cables del conjunto Pi Robot

Para la programación usaremos el lenguaje de programación Python (versión 3.3) , un lenguaje que en los últimos años está teniendo un crecimiento espectacular sobrepasando ya a lenguajes como Java o C++ según artículos como [2].

1.4 Estructura de la memoria

Esta memoria presenta todos los pasos descritos para abordar el objetivo deseado y se estructura en 3 partes bien diferenciadas:

- La primera parte de la que forma parte el presente capítulo es una introducción al lector acerca del objetivo del proyecto, de las herramientas usadas, etc. Es decir, el contexto sobre el que girará el Trabajo Fin de Grado. Esta primera parte ocupará 3 capítulos. El primero es una breve introducción del proyecto. En el segundo se nombran los dispositivos de medida usados, mientras que en el tercero se estudian los motores del robot, los cuales serán las piezas sobre las que habrá que actuar.

- La segunda parte de esta memoria es la parte más importante debido a que trata cómo solucionar el problema. Esta parte está enteramente destinada al control. En particular a analizar cómo funcionan los programas y en qué se basan para poder lograr el objetivo. Mientras que el capítulo 4 es más teórico y explicativo, el capítulo 5 describe los experimentos y pruebas realizadas para llegar al objetivo final, llegando a tratar también sobre la precisión de los instrumentos usados.
- Por último está la tercera parte relacionada con la conclusión. Esta se muestra en el capítulo 6, así como las futuras líneas de investigación o las posibles mejoras e implementaciones que se le pueden hacer a este Pi Robot.

Capítulo 2

Tecnologías de los sensores

Para abordar el problema que se plantea en este proyecto es necesario captar dos tipos de información diferente. Una vez analizadas, se sacarán conclusiones acerca de cómo se comporta el conjunto, y se trazaran estrategias para minimizar el error y hacer una herramienta precisa.

La primera información que se obtendrá, y la más importante, será la velocidad. Ésta se obtendrá gracias a un sensor encoder. La segunda información será la referente a la orientación del robot. Con esta se podrá determinar cuánto ha girado el robot en un determinado movimiento. Cabe destacar que los sensores usados durante este Trabajo son sensores propioceptivos, los cuales miden variables internas del robot. Más concretamente, se trata de sensores odométricos.

2.1 Tecnología del encoder

En cualquier aplicación para el control de velocidad de un objeto móvil (y de su trayectoria) es necesario saber cómo se mueve el objeto que queremos controlar. Para esta misión usamos un encoder diferencial de dos puertos, uno para cada motor.

Como se puede apreciar, se trata de un sensor fotoeléctrico. Estos sensores (figura 2.1) funcionan por medio de la detección de un cambio en la cantidad de luz que es reflejada o bloqueada por el objeto que se desea detectar. El cambio de luz puede ser debido a la presencia del objeto o por su ausencia. En [3] se encuentra más información sobre estos dispositivos.

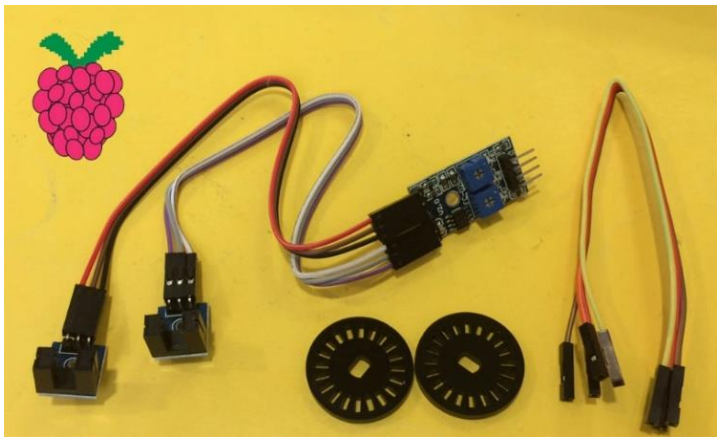

Figura 2.1. Partes que componen el encoder. De izquierda a derecha: Sensores unidos a placa base, ruedas medidoras y cables

La detección correcta de un sensor fotoeléctrico requiere que el objeto que se desea detectar ocasione un cambio suficiente en el nivel de luz detectado por el sensor. El cómo se realiza esta tarea será el tema del que trata el Apéndice A. El encoder leerá un disco con una serie de espacios y, por asociación a la rueda correspondiente, nos dará información de cuánto se mueve en una determinada cantidad de tiempo, es decir, de la velocidad.

2.2 Tecnología IMU

Además de información acerca de la velocidad, para según qué aplicaciones hay que obtener información sobre la orientación de nuestro Robot. Esta información se necesitará principalmente cuando se quiera hacer trayectorias que contengan rotaciones puras. La información necesaria sobre los primeros pasos a dar con este sensor están disponibles en [4]. Para ello usaremos una unidad de medición inercial, la usada para este proyecto contará con:

1) Acelerómetro
2) Giróscopo
3) Magnetómetro

De los cuales solo se usará la información recopilada por el giróscopo, en particular del giro respecto del eje vertical (eje Z). Este instrumento cuenta con un repositorio de programas[4] tanto en lenguaje C, como en lenguaje Python. El programa usado para ello en lenguaje Python (programa completo disponible en [5]) resulta ser un programa de gran extensión, sintácticamente hablando, el cual da distintas medidas usando todos y cada uno de los instrumentos incorporados en él. Un ejemplo de ejecución es la mostrada en la figura 2.2. El arranque de este dispositivo se analizará en el Apéndice C.

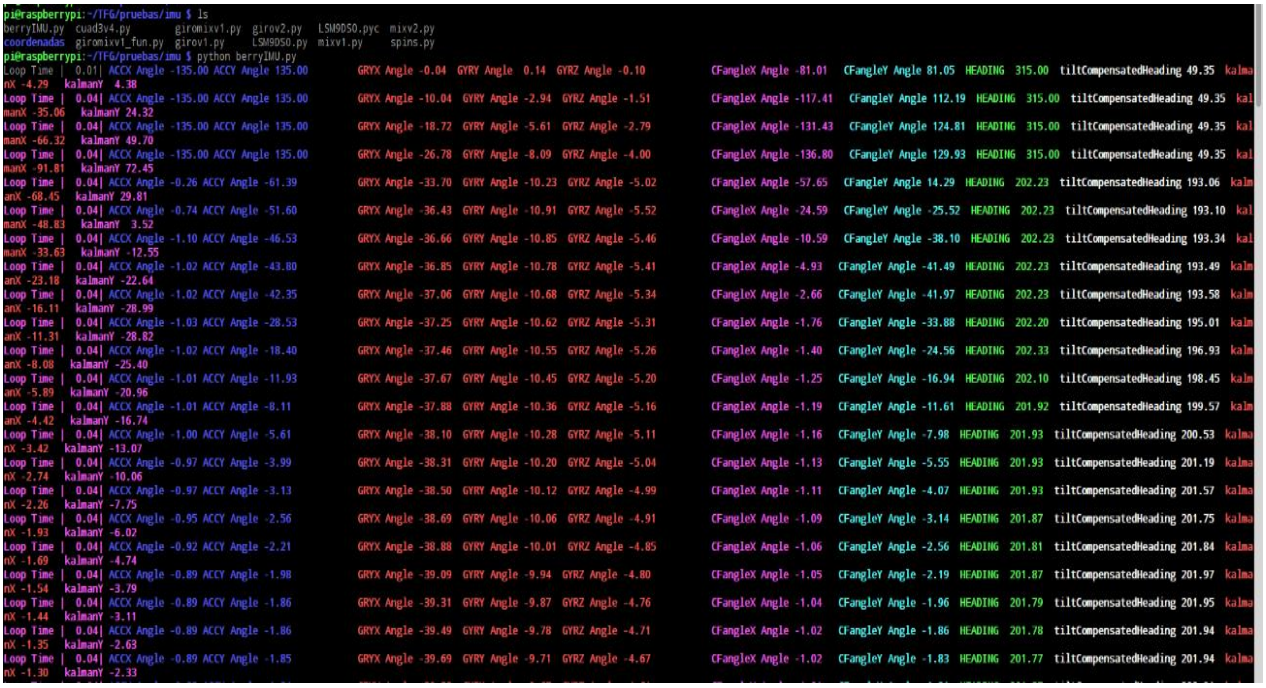

Figura 2.2. Resultado mostrado por pantalla con todos los parámetros definidos y medidos por defecto del programa original berryIMU.py encontrado en el repositorio GIthub.

El acelerómetro (variables *ACCX y ACCY)* da información acerca de los ejes X e Y por lo cual no nos es de utilidad. Mientras, el giróscopo (variables *gyroXangle, gyroYangle, gyroZangle*) informa acerca de los tres ángulos existentes. Para cualquier ejecución se usará *gyroZangle* para hallar el ángulo girado por nuestro Pi Robot. Además de usar estos instrumentos para dar lecturas, la IMU tiene una serie de filtros que dan una respuesta sin tanto ruido.

[4]https://github.com/mwilliams03/BerryIMU

Estos filtros como se puede observar en la imagen 2.2 (*Kalman X, Kalman Y, Complementary Filter X y Complementary Filter Y)* dan información sobre ejes que no nos interesan por lo que no se tendrán en cuenta.

En consecuencia, de entre todos los dispositivos mencionados, solo nos interesa uno. Razón por la cual se ha estudiado la sintaxis facilitada en el repositorio y se ha definido un programa en lenguaje Python (figura 2.5) acorde a nuestras necesidades. Este programa, más conciso, más intuitivo y más fácil de seguir sólo muestra la información capturada por el giróscopo y de esa información, solo se usa la relacionada con el eje Z.

Los giróscopos digitales están cada día más presentes en nuestras vidas; un ejemplo de ello son su implantación en los teléfonos móviles como se comenta en [6].

Como ya es sabido, el giróscopo es un dispositivo que mide el movimiento de rotación. Lo miden a través de la velocidad angular, en grados por segundo (º/s) o en revoluciones por segundo. El giróscopo digital que usaremos es un pequeño sensor de bajo coste, que mide la velocidad angular y que puede medir la rotación alrededor de 3 ejes: x, y, z. En la figura 2.3 se pueden ver los ángulos de este dispositivo. La información comentada a continuación sobre el funcionamiento del giróscopo está disponible en [7].

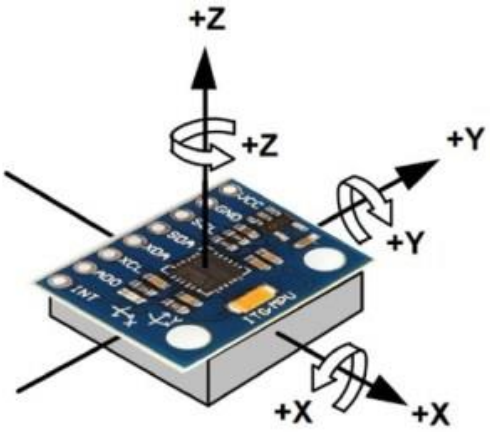

Figura 2.3. Relación de posición de los distintos ejes medidos por el sensor

El funcionamiento de este dispositivo se basa en que, cuando se le hace girar, una pequeña masa se desplaza a medida que cambia su velocidad angular. Este movimiento convierte las señales eléctricas de muy baja corriente que pueden ser amplificadas y leídas por un microcontrolador. En la figura 2.4 se observa el funcionamiento antes descrito.

Como pasa con todos los sensores, los valores dados contendrán algo de error. Estos errores pueden deberse a múltiples factores, como por ejemplo la temperatura, y se les conoce comúnmente como ruido de medida.

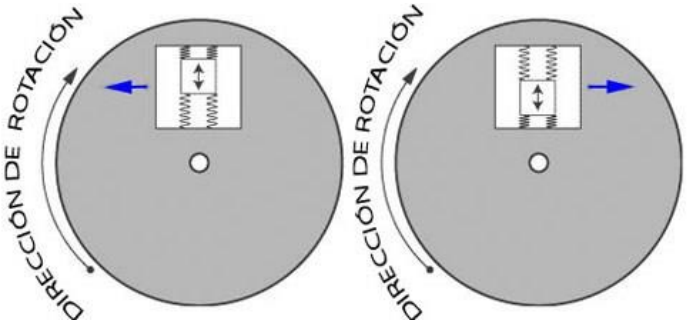

Figura 2.4. Funcionamiento teórico del giróscopo

Como se puede observar en la figura 2.2 de este mismo capítulo y se puede intuir en la foto de la sintaxis del programa (función) resultante tras la reducción, la medida de los ángulos leídos por el giróscopo aumentan aun estando parado el dispositivo. Este incremento está relacionado con el tiempo (unos pocos milisegundos) que tarda la placa base en ejecutar toda la función de lectura de los nuevos ángulos.

Estos cambios suponen una alteración en la medición restando un punto de exactitud. De modo que se podría dar el caso de que nuestro robot quisiera girar 90 grados y tras una ejecución haber girado un ángulo de ± 3 grados, estar bloqueado y que la IMU siguiera haciendo iteraciones del ángulo. Según como midiera la IMU y si el bloqueo del robot siguiera durante mucho tiempo, el robot iría variando poco a poco su valor pudiendo llegar a marcar que el ángulo girado hubiera llegado al objetivo siendo que no, el ángulo girado distaba en ± 3 del objetivo.

Una vez vistas las características de ambos dispositivos y sus posibles aplicaciones llega la hora de decidir qué dispositivo se encarga de según qué tarea. La medición de la velocidad será asumida por el encoder. En cuanto a las rotaciones puras hay más debate pudiendo servir ambos instrumentos de medida.

El encoder puede ser usado para la realización de rotaciones puras también, solamente se necesita hacer antes un ensayo en que se midan las marcas equivalentes a un determinado ángulo de giro (por ejemplo, girar 90 grados con solo una rueda equivalen a que esa misma rueda debe avanzar 17 marcas) . Una vez sabida la conversión, se le daría fuerza al motor de la rueda exterior y continuaría la acción mientras no se llegase a ese número exacto de marcas. Este tipo de ejecución presenta una ventaja notable como es que la medida nunca va a variar, siempre se van a recorrer 17 marcas, ni más, ni menos. Por otro lado presenta un gran inconveniente y es que para cada ángulo habría que hacer antes el ensayo teniendo que ensayar una gran cantidad de ángulos. También se puede deducir que el encoder para la realización de rotaciones puras carece de la precisión necesaria respecto a la IMU (el encoder no diferenciaría un ángulo de 50 grados de otro de 53). En definitiva, aunque alternativa, esta aplicación del encoder tendrá poca utilidad en la vida real dejando esta tarea a la IMU. No obstante, se ha probado a realizar un cuadrado siendo el encoder el encargado de controlar la rotación pura con el fin de compararlo con la ejecución de la IMU. Esta ejecución podrá verse llegado el capítulo de experimentación (apartado 5.3).

```
def writeGRY(register,value):
        bus.write_byte_data(GYR_ADDRESS, register, value)
        return -1
def readGYRx():
        gyr_l = bus.read_byte_data(GYR_ADDRESS, OUT_X_L_G)
        gyr_h = bus.read_byte_data(GYR_ADDRESS, OUT_X_H_G)
        gyr_combined = (gyr_l | gyr_h <<8)

        return gyr_combined  if gyr_combined < 32768 else gyr_combined - 65536
  def readGYRy():
        gyr_l = bus.read_byte_data(GYR_ADDRESS, OUT_Y_L_G)
        gyr_h = bus.read_byte_data(GYR_ADDRESS, OUT_Y_H_G)
        gyr_combined = (gyr_l | gyr_h <<8)

        return gyr_combined  if gyr_combined < 32768 else gyr_combined - 65536
def readGYRz():
        gyr_l = bus.read_byte_data(GYR_ADDRESS, OUT_Z_L_G)
        gyr_h = bus.read_byte_data(GYR_ADDRESS, OUT_Z_H_G)
        gyr_combined = (gyr_l | gyr_h <<8)

        return gyr_combined  if gyr_combined < 32768 else gyr_combined - 65536
#initialise the gyroscope
writeGRY(CTRL_REG1_G, 0b00001111) #Normal power mode, all axes enabled
writeGRY(CTRL_REG4_G, 0b00110000) #Continuos update, 2000 dps full scale

gyroXangle = 0.0
gyroYangle = 0.0
gyroZangle = 0.0

def giroIMU():
                                global gyroZangle
                                global gyroXangle
                                global gyroYangle
                                global a

                                #Read the gyroscope values
                                GYRx = readGYRx()
                                GYRy = readGYRy()
                                GYRz = readGYRz()
                                ##Calculate loop Period(LP). How long between Gyro Reads
                                b = datetime.datetime.now() - a
                                a = datetime.datetime.now()
                                LP = b.microseconds/(1000000*1.0)
                                #print "Loop Time | %5.2f|" % ( LP ),
                                #Convert Gyro raw to degrees per second
                                rate_gyr_x =  GYRx * G_GAIN
                                rate_gyr_y =  GYRy * G_GAIN
                                rate_gyr_z =  GYRz * G_GAIN
                                #Calculate the angles from the gyro.
                                gyroXangle+=rate_gyr_x*LP
                                gyroYangle+=rate_gyr_y*LP
                                gyroZangle+=rate_gyr_z*LP

                                #slow program down a bit, makes the output more readable
                                #time.sleep(0.03)

IMU_upside_down = 1     # Change calculations depending on IMu orientation.
                        # 0 = Correct side up. This is when the skull logo is facing down
                        # 1 = Upside down. This is when the skull logo is facing up
RAD_TO_DEG = 57.29578
M_PI = 3.14159265358979323846
G_GAIN = 0.070          # [deg/s/LSB]  If you change the dps for gyro,
                        #you need to update this value accordingly
```

Figura 2.5. . Sintaxis modificada del programa original berryIMU.py para adaptarse a las necesidades de nuestro programa.

Capítulo 3

Modelado

Uno de los pilares fundamentales de este trabajo es el de confeccionar un control de velocidad robusto, sencillo y eficaz que sea capaz de llegar rápidamente a la velocidad demandada por el usuario y que para las siguientes iteraciones presente poca oscilación. Será uno de los más usados a lo largo de las distintas etapas de diseño ya que en todas se requiere que el robot se mueva a una determinada velocidad.

Antes de nada, el primer paso será obtener información acerca de los motores. Un experimento que puede arrojar valiosa información sobre cómo se comporta el robot se mostrará en el Apéndice B. Otro experimento será el hallazgo de las respectivas funciones de transferencia para cada motor. En este capítulo se explicarán los métodos para la obtención de dichas funciones los cuales serán cálculos y programas de carácter general.

La función de transferencia relaciona la salida del sistema con la entrada, como puede verse en la figura 3.1 que muestra un esquema teórico de lo comentado.

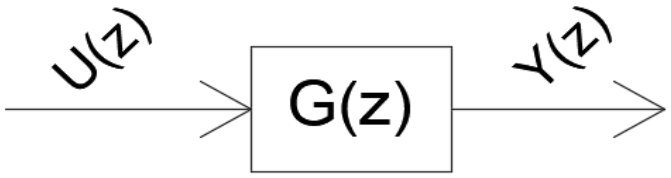

Figura 3.1. Esquema teórico de función de transferencia donde U(z) es la acción mandada, G(z) representa nuestro sistema e Y(z) la salida. Todo ello en dominio discreto.

En nuestro caso, la entrada (U(z)) serán "unidades" que oscilarán entre ± 255 y la salida (Y(z)) será velocidad medida en centímetros/segundo. Lo primero que se debe hacer es averiguar qué sistema se tiene. Para ello se hace un experimento en bucle abierto para ver cómo se comporta el motor. El experimento consiste en que, dada una entrada en escalón cualquiera (entrada rueda derecha: 255 || entrada rueda izquierda: 255) , ver cuánto le cuesta al robot llegar a una velocidad constante, al régimen permanente. Este experimento, dado que se usa también para hallar la componente τ, se explicará en el apartado 3.2.

A la vista de los resultados, se observa que se tiene un sistema de primer orden. Este sistema tiene la peculiaridad de que en la ecuación general aparece solamente la derivada primera del bucle abierto como se comenta en [8][10], por lo tanto la ecuación genérica queda como se muestra en la siguiente ecuación:

$$\tau * \frac{dy(t)}{dt} + y(t) = K * u(t)$$

Aplicando a ambos lados la transformada de Laplace y considerando condiciones iniciales nulas, se llega a la expresión general de una función de transferencia de 1º orden, mostrada a continuación:

$$G(s) = \frac{Y(s)}{U(s)} = K * \frac{1}{1 + \tau * s}$$

Donde K es la ganancia local del proceso y τ es la constante de tiempo del sistema.

3.1 Cálculo del parámetro K

Como se ha visto anteriormente, la función de transferencia relaciona la entrada del motor con su salida. Viendo la expresión, "K" es un parámetro que depende linealmente de la salida. Para hallar este parámetro se usarán los programas "*calculo_K_L.py" y "calculo_K_R.py" (Programas disponibles en apéndice I),* y se hará lo siguiente:

- Se dejará al robot rodar en línea recta durante 2 segundos hasta alcanzar el permanente.
- Dicha ejecución de 2 segundos se dividirá en 4 partes contando las marcas leídas en cada una de estas etapas de 0.5 segundos, de modo que se tendrán 4 medidas de la velocidad.
- Sabiendo las 4 velocidades se hará la media para tener la velocidad media de esa ejecución (en $\frac{cm}{s}$)
- Este procedimiento se repetirá 3 veces para cada entrada suministrada por el usuario
- Se evaluarán todas las entradas desde 100 hasta el máximo, 255, en intervalos de 10 unidades. Acabando en 250 y añadiendo una más con valor 255.
- Una vez se tienen las tablas con las entradas y sus respectivas velocidades (en cm/s) se hace lo siguiente: se divide la entrada entre la velocidad.
  Con esta operación se saca la K propia de cada experimento. Si se estuviera en un sistema perfectamente lineal todas las ganancias deberían dar el mismo valor, dado que nos encontramos en un sistema no ideal estos valores varían un poco entre ellos.
  Esto es así debido al siguiente razonamiento disponible en [9],[10]:

  La salida de un sistema puede expresarse así:

  $$Y(s) = G(s) * U(s)$$

  Siendo Y(s) la salida, G(s) la función de transferencia que modela nuestro sistema y U(s) la entrada (en este caso las unidades que se le mandan al motor)

  Ahora se necesita saber el valor de la salida ($y(t \to \infty)$) cuando el tiempo tiende a infinito por lo que se trabajará con el Teorema del Valor Final:

  $$y(t \to \infty) = \lim_{s \to 0} s * Y(s) = \lim_{s \to 0} s * \big(G(s) * U(s)\big) = \lim_{s \to 0} s * \frac{K}{1 + \tau * s} * \frac{255}{s}$$

  Siendo $\frac{255}{s}$ la entrada escalón (U(s)) de valor 255 para el caso concreto de este ensayo.

Se simplifican los términos quedando la siguiente igualdad:

$$\lim_{s\to 0} \cancel{s} * \frac{K}{1 + \tau * s} * \frac{255}{\cancel{s}} = \lim_{s\to 0} \frac{K}{1 + \tau * s} * 255 = 255 * K$$

$$K = \frac{y(t \to \infty)}{255}$$

Es decir, el resultado de la K de cada experimento será el valor de su salida (velocidad en cm/s) dividida entre su entrada (que en este ensayo, es de 255 unidades).

El resultado de este experimento se muestra a continuación en las figuras siguientes

entradas	100	110	120	130	140	150	160	170	180
vels	19,625	23,748	26,881	29,85	33,478	34,962	38,425	40,404	42,384
K	0,19625	0,21589	0,22401	0,22962	0,23913	0,23308	0,24016	0,23767	0,23547
entradas	190	200	210	220	230	240	250	255	
vels	45,847	47,496	49,145	51,454	52,938	55,412	58,126	59,37	
K	0,24130	0,23748	0,23402	0,23388	0,23017	0,23088	0,23250	0,23282	

Figura 3.2. Resultado del experimento para hallar la K sobre la rueda derecha donde las entradas son adimensionales y las velocidades están medidas en cm/s

entradas	100	110	120	130	140	150	160	170	180
vels	20,285	23,913	27,706	30,18	33,643	35,622	38,261	39,91	42,054
K	0,20285	0,21739	0,23088	0,23215	0,24031	0,23748	0,23913	0,23476	0,23363
entradas	190	200	210	220	230	240	250	255	
vels	44,198	46,506	48,815	50,794	52,443	54,093	57,556	59,535	
K	0,23262	0,23253	0,23245	0,23088	0,22801	0,22539	0,23022	0,23347	

Figura 3.3. Resultado del experimento para hallar la K sobre la rueda izquierda donde las entradas son adimensionales y las velocidades están medidas en cm/s

Para ver más claro cómo se comportan los motores según la entrada que reciben se mostrarán estos mismos datos en las siguientes gráficas relacionando la entrada (eje de abscisas) con la velocidad producida (eje de ordenadas).

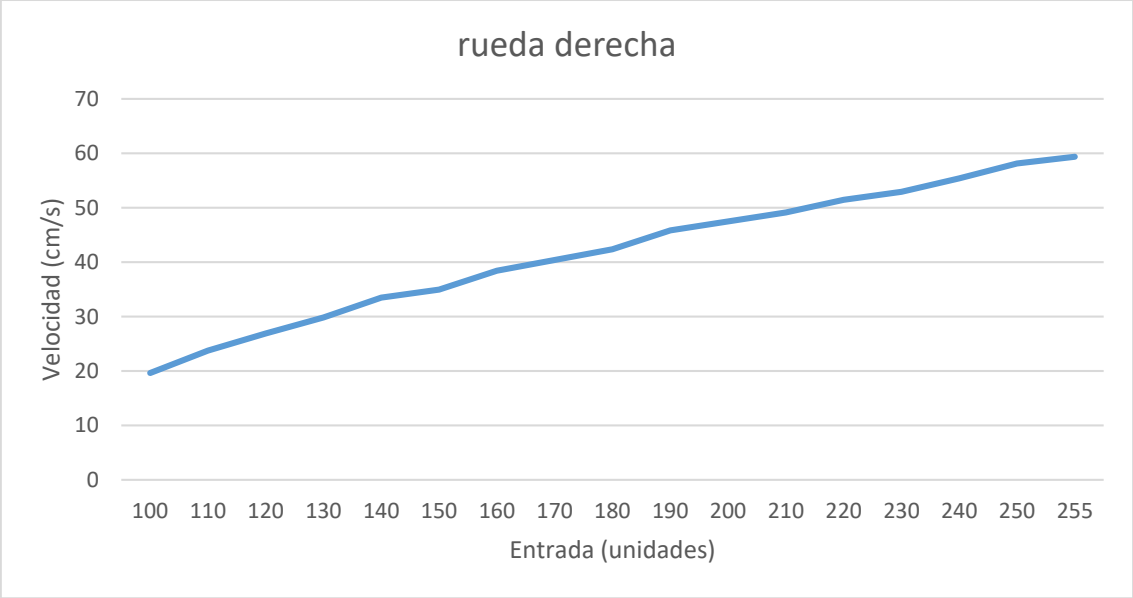

Figura 3.4. Gráfica de los resultados mostrados en la figura 3.2

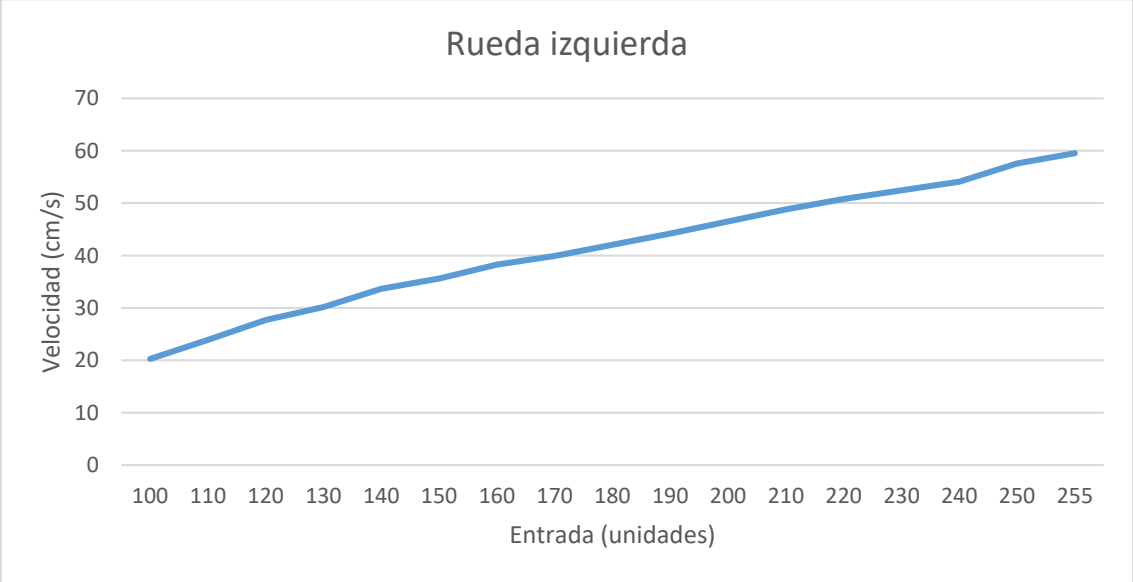

Figura 3.5. Gráfica de los resultados mostrados en la figura 3.3

Como se concluye tras el experimento realizado en el Apéndice B, las entradas elegidas empiezan en el valor de 100 uds. debido a que es la entrada mínima que garantiza un movimiento fluido y sin tirones. Este experimento requiere que el tipo de movimiento sea homogéneo (movimientos sin tirones, ni paradas por falta de fuerza de los motores, etc.). Cumpliéndose esto se cumplirá que todos los valores de $K_{entrada}$ serán bastante parecidos los unos de los otros.

Ejecutar el estudio con una entrada de (80,80) no garantiza en ningún caso que el movimiento del robot vaya a ser el deseado, pudiendo en alguna ocasión no poder realizarlo ya sea porque la batería que surte de energía al conjunto este baja, por un mayor agarre del suelo que hace que se necesite más fuerza para realizar el mismo movimiento etc. Esto provoca que no pueda incluirse en el estudio dado que podría falsear el resultado obteniendo unas ganancias no acordes con el funcionamiento normal del robot.

Esto será una restricción para programas como por ejemplo el capaz de generar circunferencias dado que se verá limitada su velocidad. Esto se explicará más adelante.

Una vez tenemos todas las ganancias propias, debemos encontrar una K global para dicho motor, ya que no se puede tener una K diferente en función de la entrada que se recibe. Para ello se hace la mediana de los valores de los que se dispone dando los siguientes resultados:

$$K_{rueda\ derecha} = 0.23308 \text{ (adimensional)}$$

$$K_{rueda\ izquierda} = 0.23245 \text{ (adimensional)}$$

Una vez obtenidos estos valores solo nos faltará calcular la constante de tiempo (τ) de cada motor para tener todos los parámetros de nuestras funciones G(s) .

3.2 Cálculo del parámetro τ

El procedimiento para calcular la constante de tiempo del sistema se hace de manera diferente. Mientras que antes se analizaba el sistema en permanente, ahora se analizará el transitorio. Se usará el programa mencionado antes, que nos ayudaba a averiguar que los motores eran unos sistemas de primer orden.

Para hallar este parámetro se usarán los programas "*finversa_L.py" y "finversa_R.py"* y se hará lo siguiente:

- Se mandará una entrada a ambas ruedas para que se muevan durante 1 segundo aproximadamente (normalmente la máxima velocidad).
- Durante este tiempo los encoder medirán marcas, midiendo además el tiempo que transcurre entre marca y marca (milisegundos).
- Dividiendo la conversación (cm/marca) entre el tiempo medido (seg/marca) se obtendrá la velocidad instantánea en ese momento. Con ayuda de la extrapolación de dichos datos a una gráfica se verá cómo la velocidad aumenta hasta llegar al régimen permanente.

Tanto este programa como el usado para calcular la ganancia del sistema se muestran en el apéndice I (apartados I.4 e I.3 respectivamente). Una ejecución de este procedimiento sin ningún tipo de filtrado ni suavizado se puede observar en la figura 3.6. Además del programa antes mencionado que será el que obtenga los datos numéricos, se recurrirá a una aplicación del programa Matlab para conseguir un resultado con mejor precisión.

Una vez tenemos los datos capturados siguiendo el esquema anterior, se pueden representar gráficamente como se ha comentado anteriormente. Estos datos generarían una gráfica atribuible a un sistema de 1º orden. En el dominio del tiempo, la gráfica podría aproximarse por una función exponencial de dos sumandos como la mostrada a continuación:

$$\text{Velocidad} = a * e^{b*X} + c * \ e^{d*X}$$

El siguiente paso sería comprobar en qué momento del eje X (tiempo), la gráfica entra en el intervalo del ± 5% del valor en permanente y no sale de ahí (La información relacionada con este razonamiento se ha extraído de [10]. Como la gráfica que se representa en la figura 3.6 presenta muchos picos, no se puede usar para medir este tiempo de respuesta con precisión. Para solucionarlo se recurre a una herramienta de Matlab que suaviza la curva. Todo el procedimiento desarrollado en Matlab se explicará en el Apéndice E.

Con la gráfica corregida sí que se puede medir claramente el momento en que la gráfica entra en el margen del ±5% del valor en permanente y ya no vuelve a salir de ahí. Este valor será el tiempo de respuesta del sistema (t_r) y con él se hallará la constante de tiempo (τ) usando la siguiente fórmula al tratarse de un sistema de primer orden:

$$\tau = \frac{t_r}{3}$$

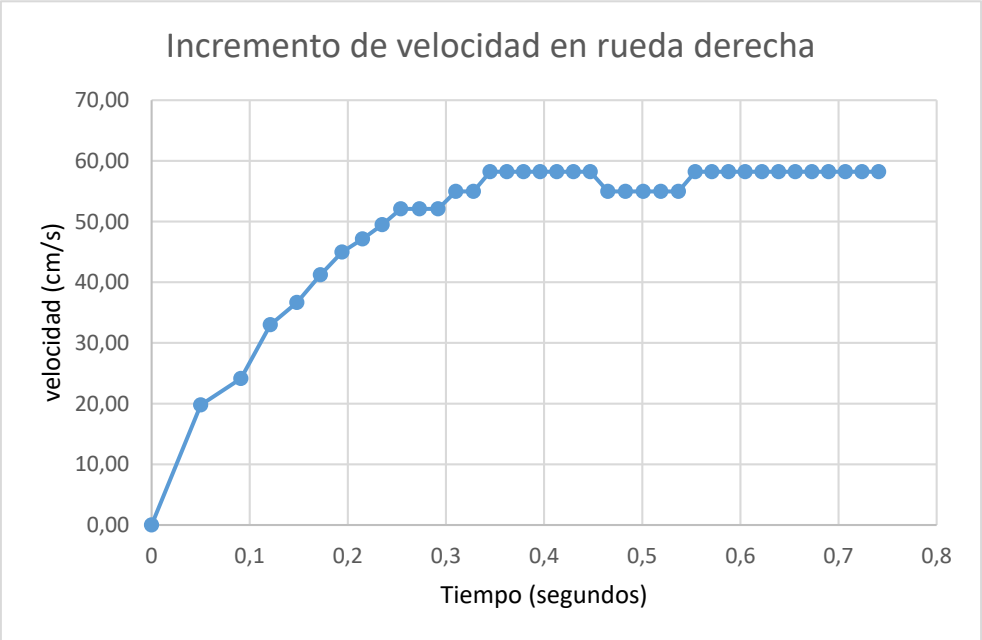

Figura 3.6. Gráfica donde se muestra el incremento sufrido solamente por la rueda derecha (aceleración) cuando sobre ambas ruedas ella se aplica la máxima entrada posible (255 unidades) siendo el experimento en bucle abierto.

Para tener un valor contrastado de la constante de tiempo se repite este experimento 5 veces y se hace la media de sus valores. Los valores de las constantes de tiempo de cada rueda se muestran a continuación.

$$\tau_{rueda\ derecha} = 0.1336 \text{ segundos}$$

$$\tau_{rueda\ izquierda} = 0.1373 \text{ segundos}$$

3.3 Cálculo de G(s)

Una vez se tienen los parámetros que modelan el comportamiento de los motores es inmediato sacar las funciones de transferencia:

$$G(s)_{rueda\ derecha} = 0.23308 \frac{1}{1 + 0.1336 * s}$$

$$G(s)_{rueda\ izquierda} = 0.23245\ \frac{1}{1 + 0.1373 * s}$$

Dadas las características de nuestro sistema basado en un microcontrolador (discreto), debemos pasar nuestras funciones de transferencia del dominio continuo al dominio discreto. Antes de nada, se debe fijar un tiempo de muestreo razonable el cual no sea ni muy lento donde la acción tarde mucho en aparecer, ni muy rápido, ya que habrá muy poca información con la que comparar. El tiempo de muestreo elegido ha sido de 0.1 segundos. La elección del periodo de muestreo y el modo de obtención se analiza en profundidad más adelante, en el apartado 4.1.

Para sacar las funciones en tiempo discreto (G(z)) se usará un comando de la herramienta matemática Matlab, la cual convierte una función en tiempo continuo en una función en tiempo discreto. Esta herramienta se usa escribiendo en la ventana de comandos de Matlab lo siguiente:

G(z) = c2d[G(s), tiempo de muestreo]

Siendo c2d la abreviatura de “continue to discrete”. las funciones obtenidas han sido:

$$G(z)_{rueda\ derecha} = \frac{0.1228}{z - 0.4731}$$

$$G(z)_{rueda\ izquierda} = \frac{0.1202}{z - 0.4827}$$

También se pueden obtener de manera manual con la ayuda de tablas de conversión de dominio en “s” a dominio en “z” [11].

3.4 Simulación

Una vez se ha hallado la función de transferencia de cualquiera de nuestros motores, una manera de comprobar que se va en la dirección correcta es mediante simulación. De esta manera se simulan ambas G(s) en algún programa informático (figura 3.9) y se ejecutan teniendo que dar datos parecidos a los obtenidos hasta el momento. El programa que se usará para esta misión será "*Simulink*" , una aplicación dentro de las muchas que ofrece Matlab.

En este programa se creará a nivel teórico la estructura de nuestros motores y se ejecutará teniendo que dar por resultado algo semejante a lo visto en la realidad. La entrada que se simulará será la máxima, un escalón de 255 unidades y el resultado se comparará con alguna ejecución de las usadas para el cálculo de τ que como esta, es una ejecución a la máxima entrada posible mirando en cada instante como aumenta la velocidad de la rueda.

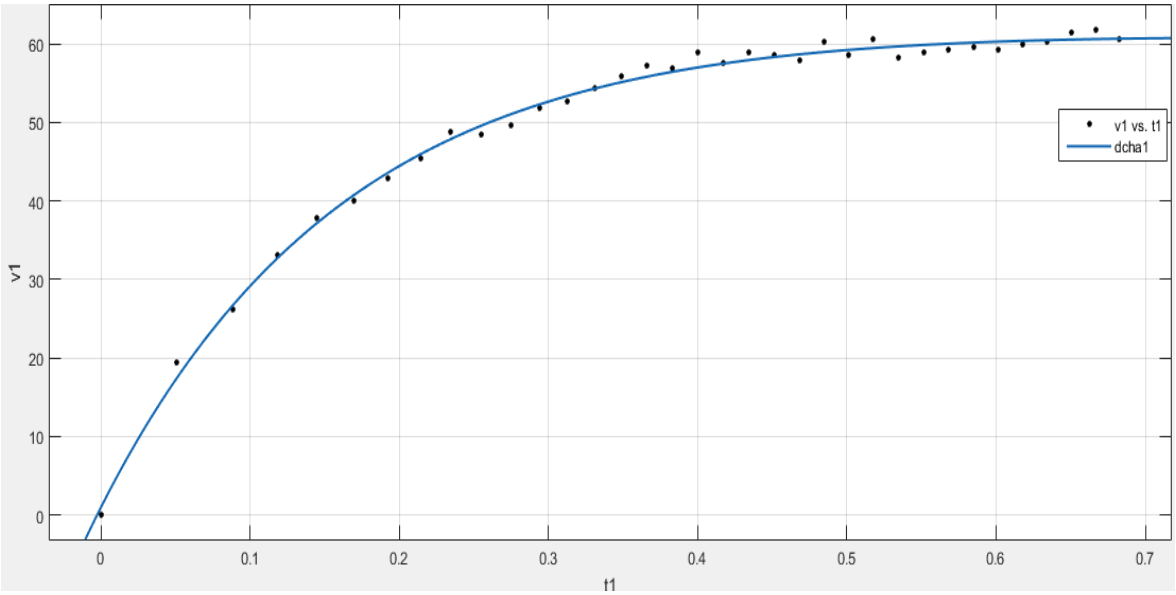

Figura 3.7. Gráfica suavizada de la ganancia de velocidad del robot a entrada máxima en ambas ruedas (255,255) siendo el eje de abscisas el tiempo (segundos) y el eje de ordenadas la velocidad (cm/s). En esta gráfica se puede observar la ganancia de velocidad en la rueda derecha.

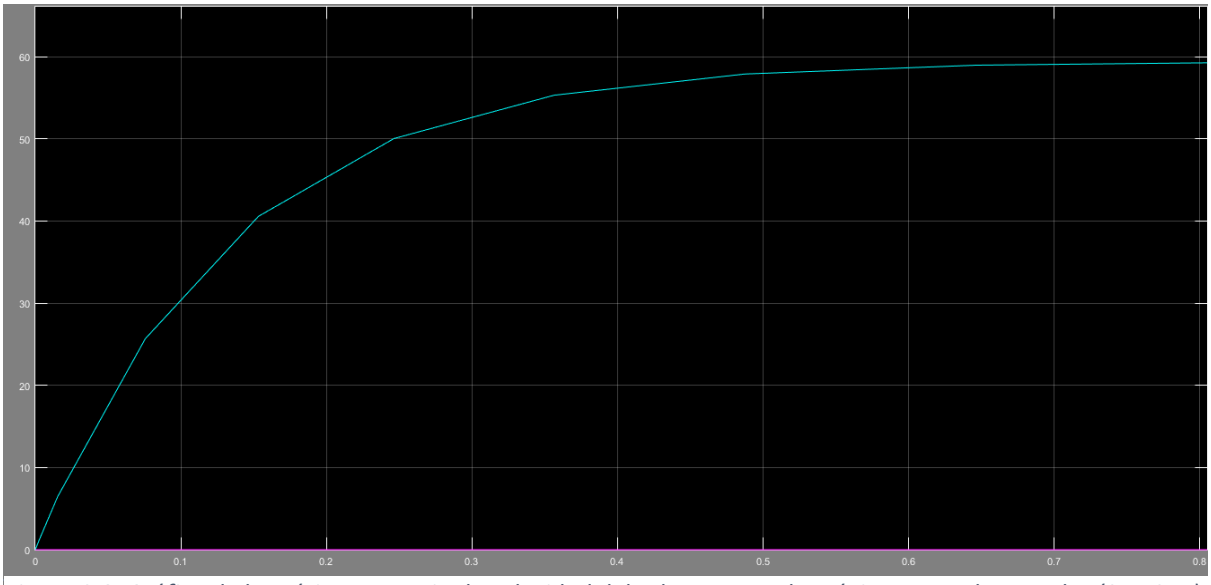

Figura 3.8. Gráfica de la teórica ganancia de velocidad del robot a entrada máxima en ambas ruedas (255,255) según Simulink con la G(s) dada del motor de la rueda derecha. Siendo el eje de abscisas el tiempo (segundos) y el eje de ordenadas la velocidad (cm/s)

Como se observa, ambas imágenes son muy parecidas. Ambas llegan a un valor muy parecido del régimen permanente, cercano a la velocidad de 60 cm/s y ambas lo hacen en un tiempo prácticamente idéntico.

Esto no viene más que a confirmar el buen trabajo realizado a la hora de modelar nuestro sistema el cual con los experimentos realizados se ha sido capaz de llevar un sistema real al plano teórico. Esto será de gran ayuda aquí y en implementaciones más complicadas ya que antes de probar algo en la realidad se podrá probar de manera teórica sabiendo que los resultados mostrados por ambos experimentos no distarán mucho el uno del otro.

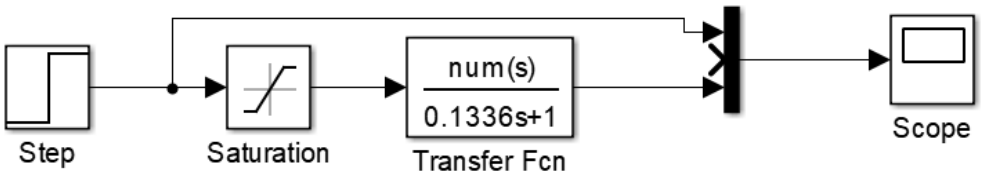

Figura 3.9. Esquema realizado en Simulink del motor de la rueda derecha. Este experimento dará como resultado la figura 3.8 vista anteriormente. Experimento en bucle abierto.

3.5 Coexistencia de dos periodos de muestreo

Como se comenta más adelante (apartado 4.2.2 caso C), con el objetivo de darle un mayor rango de velocidades a nuestro Raspberry Pi Robot se implementa un nuevo tiempo de muestreo. Este nuevo T se usará únicamente para ejecuciones a "mínima velocidad" .

El tiempo de muestreo será de 0.3 segundos para que el robot sea capaz de leer más marcas y conseguir ser capaz de conseguir ejecuciones a una velocidad inferior.

El cambio en el periodo de muestreo supone que las ecuaciones en tiempo discreto ya no sean las mostradas en el apartado 3.3 por lo que surge la necesidad de recalcularlas en vistas a calcular el regulador que actuaría cuando se diera esta circunstancia. Ayudados por Matlab, las nuevas expresiones se detallarán a continuación:

$$G(z)_{rueda\ derecha} = \frac{0.12084}{z - 0.1059}$$

$$G(z)_{rueda\ izquierda} = \frac{0.2063}{z - 0.1125}$$

Capítulo 4

Diseño del Control

En este capítulo se presentan todos los pasos relativos al diseño de los diferentes controles usados durante este trabajo. La primera parte del capítulo hablará enteramente del control de velocidad en el que se abarcarán temas como el porqué del control elegido, la implementación, el tiempo de muestreo elegido, etc. La segunda parte irá destinada al control usado para las translaciones puras focalizándose en las partes más importantes del mismo.

4.1 Elección del tipo de control

4.1.1 Introducción

Una vez sabido qué control queremos hacer, en este caso un control de velocidad, surgen diversas preguntas a las que hay que hacer frente: el tiempo de muestreo, el tipo de control (si es proporcional, integral…).

Lo más importante en este tipo de control es cómo medir la velocidad para trabajar con ella, poder compararla y en función de los resultados obtenidos hacer una u otra acción. Para ello, antes de nada hay que encontrar un equilibrio para el tiempo de muestreo.

Idealmente cuanto más pequeño es el tiempo de muestreo, más rápido se corrigen los posibles fallos y normalmente se tiene un control más rápido. El problema es que debido a la poca resolución del encoder (solo 20 marcas) en un tiempo de muestreo muy pequeño la lectura o no de una marca tiene un impacto mayor. Como se puede imaginar, hacer un cálculo de velocidad con una marca resulta un cálculo con muy poca precisión[5]. La solución más rápida a ese problema sería aumentar el tiempo de muestreo a, por ejemplo 0.5 segundos, así una diferencia en una marca supondría una diferencia de aproximadamente 2 cm/s entre las velocidades de las ruedas ruedas y no los 10 cm/s que resultarían anteriormente con T = 0.1 s.

En este nuevo tiempo de muestreo se contarían muchas más marcas que antes (5 veces más) por lo cual la velocidad dada sería más precisa. El principal problema de esta solución y que no se puede admitir a la hora de diseñar nuestro control es la lentitud. Todo lo que se gana en un cálculo más exacto de la velocidad, se pierde en rapidez de respuesta que es lo que se quiere como objetivo principal. Se puede apreciar tal diferencia en las figuras 4.2 y 4.3.

Esto descarta la opción de aumentar el periodo de muestreo a 0.5 segundos; siendo el tiempo de respuesta más adecuado para nosotros el pensado inicialmente, es decir, 0.1 segundos. Este periodo de muestreo sigue siendo lento para el sistema motor, que tiene un tiempo de respuesta cercano a medio segundo. Por eso el tipo de control más adecuado para nosotros sea el denominado control Deadbeat.

[5] El cálculo original con periodo de muestreo (T= Δt) sería $velocidad = \frac{n*0.9895\,\frac{cm}{marca}}{\Delta t}$ siendo n→ número de marcas leídas en el Δt

Este control Deadbeat es usado debido a que se está ante un sistema muy rápido y se desea un muestreo muy rápido para no perderse nada de información. Debido a la poca resolución del enconder que se tiene, esto no es posible.

4.1.2 Control Deadbeat

El control deadbeat busca que el comportamiento sea lo más rápido posible sin error, por lo que la expresión teórica de la función de transferencia en bucle cerrado será: $D(z) = \frac{K_{sistema}}{z}$

Esto viene a decir que habrá un polo muy rápido (z) y que se deberá ajustar la $K_{regulador}$ para que la ganancia del sistema con el regulador en bucle cerrado sea 1 ($K_{sistema} = 1$). Toda la información relativa el control Deadbeat esta disponible en [12].

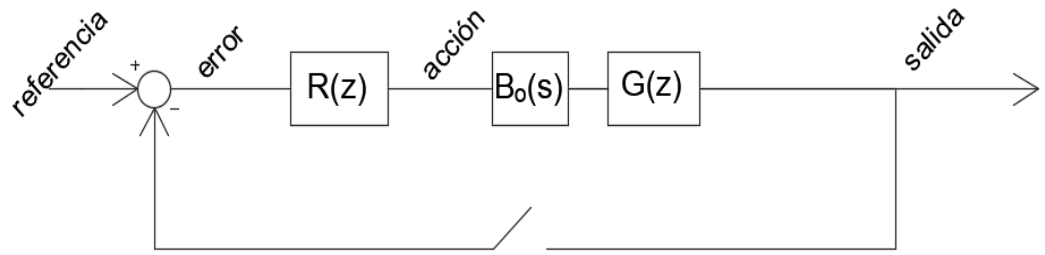

Figura 4.1. Diagrama de bloques teórico de control a implementar donde se muestran todas las partes desde la referencia que es el objetivo pedido por el usuario, al error entre lo pedido y lo dado por el robot, pasando por la acción, dato calculado según el error cometido, y bloqueador de orden 0, G(z)

El control deadbeat busca diseñar un controlador para que se obtenga una respuesta temporal finita para un sistema en bucle cerrado sometido a una excitación escalón. Para este trabajo se tiene que la respuesta temporal finita será en T segundos ya que éste es el tiempo de muestreo. Cada este tiempo, el sistema recogerá información, la comparará con lo que quiere y en función de los resultados hará una acción u otra. Por otro lado, la excitación escalón será la entrada mandada en "unidades" que se irá actualizando cada T.

Este controlador es un controlador digital que coloca todos los polos de bucle cerrado en el origen. Sus características más importantes y la razón por la que se ha elegido este tipo son:

- $t_{respuesta}\ y\ t_{asentamiento}$ son mínimos
- $e_p = 0$

Otro parámetro que caracteriza a este control es "n". Este parámetro muestra el exceso de polos sobre ceros en la función de transferencia de la planta. El controlador concentra los polos y los ceros de la planta y coloca "n" polos en el origen.

Una vez averiguadas las características principales que tiene este control sobre los demás, se procede a mostrar las expresiones más importantes y las que se usarán para dar forma a las expresiones que más adelante controlaran nuestro sistema:

- Función de transferencia en bucle cerrado → $D(z) = z^{-n}$
- Controlador → $R(z) = \frac{1}{G(z)} * \frac{D(z)}{1-D(z)}$

Un inconveniente que presenta es la existencia de sobreoscilación (Se observa en las figuras 4.2 y 4.3 como en la primera parte de la ejecución, la velocidad alcanzada es mayor que la pedida en ambos casos). Otro es la necesidad de un muy buen modelado de la planta, ya que de lo contrario, el control no funciona como se esperaría de él. Como quedó patente en el apartado 3.4, se consiguió un muy buen modelado de la planta por lo que esto no será un inconveniente.

Para concluir, como ventaja se puede decir que sirve para controlar aquellos sistemas que no son lineales. Este sistema como se puede ver en el cálculo de la K (Capítulo 3), o en la gráfica de la zona muerta (Anexo B) no es un sistema perfectamente homogéneo ni perfectamente lineal.

Con este Δt, la mayor limitación es la medición de la velocidad debido a la poca resolución del encoder. La solución tomada para hacer frente a este contratiempo y medir una velocidad precisa se explicará en Anexo D donde se detallará la sintaxis y la idea principal.

A continuación, se mostrará una comparativa (figuras 4.2 y 4.3) del control de velocidad en línea recta entre un control implementado con un tiempo de muestreo de 0.5 segundos y otro con un tiempo de 0.1 segundos.

Como puede observarse, el control con el T más pequeño llega antes a la velocidad objetivo. En particular, mientras que con 0.1 segundos, alcanza la velocidad objetivo a los 0.6 segundos de haber empezado la trayectoria aproximadamente, no es hasta los 2-2.5 segundos cuando ocurre lo mismo con el otro control. Esto hace que el control con T = 0.5 segundos sea más lento y dé peores resultados, otra razón más para elegir como periodo de muestreo T = 0.1 segundos.

A continuación se explicará detalladamente los dos programas principales en los que se usará este control. Dicho control se usará como base para realizar una trayectoria recta y para realizar una circunferencia de radio preestablecido (trayectoria curva). Ambos programas son esencialmente el mismo, solo que cambiarán las referencias y la manera de introducir datos. Se empezará explicando la trayectoria en línea recta.

4.2 Implantación del controlador Deadbeat para realizar trayectorias rectas

El primer tipo de programa que se estudiará será el programa que permita la realización de trayectorias rectas precisas[6]. Este programa será uno de los que más peso tenga dentro del algoritmo implementado dado que el realizar una trayectoria recta será uno de los componentes fundamentales de la navegación y será una ejecución que se repetirá con gran asiduidad. Su mayor utilización estará dentro del programa de navegación directa a un punto[7] dado que siempre que el robot quiera ir a una coordenada distinta del origen, deberá moverse en línea recta.

[6] Programa "*dead_dual_5a.py"* disponible en el Apéndice I
[7] Programa "*coord_newconcept_5a.py"* disponible en el Apéndice I

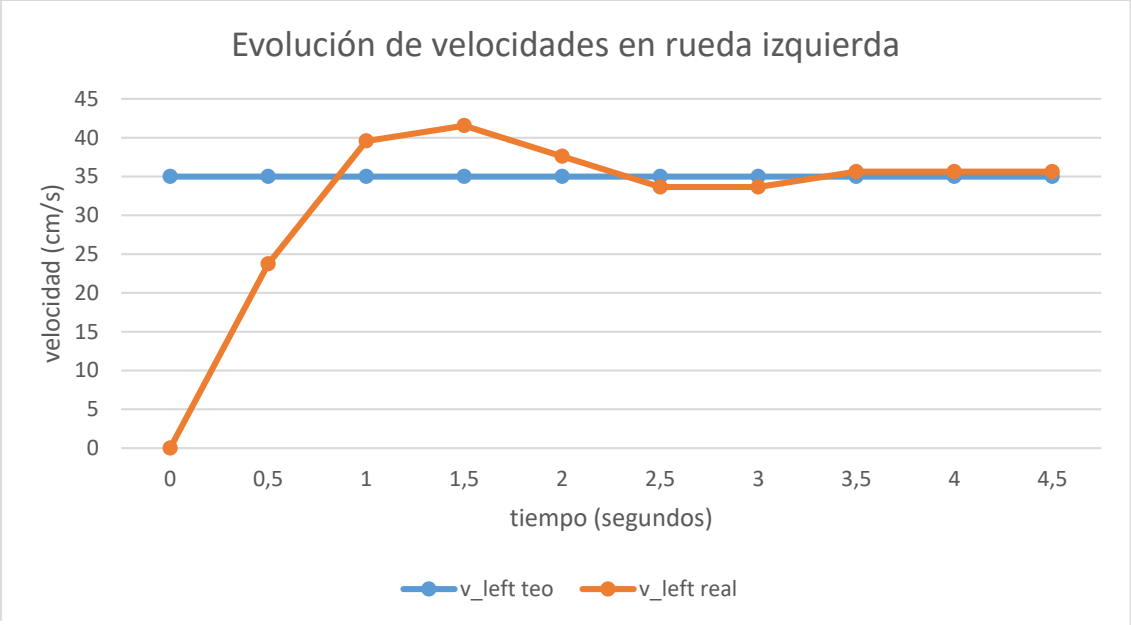

Figura 4.2. Gráfica que muestra la evolución en la rueda derecha desde el reposo hasta la velocidad objetivo con un T=0.5 segundos donde la línea azul representa la velocidad pedida por el usuario (35 cm/s) y la línea naranja la velocidad llevada por el robot en cada instante de tiempo.

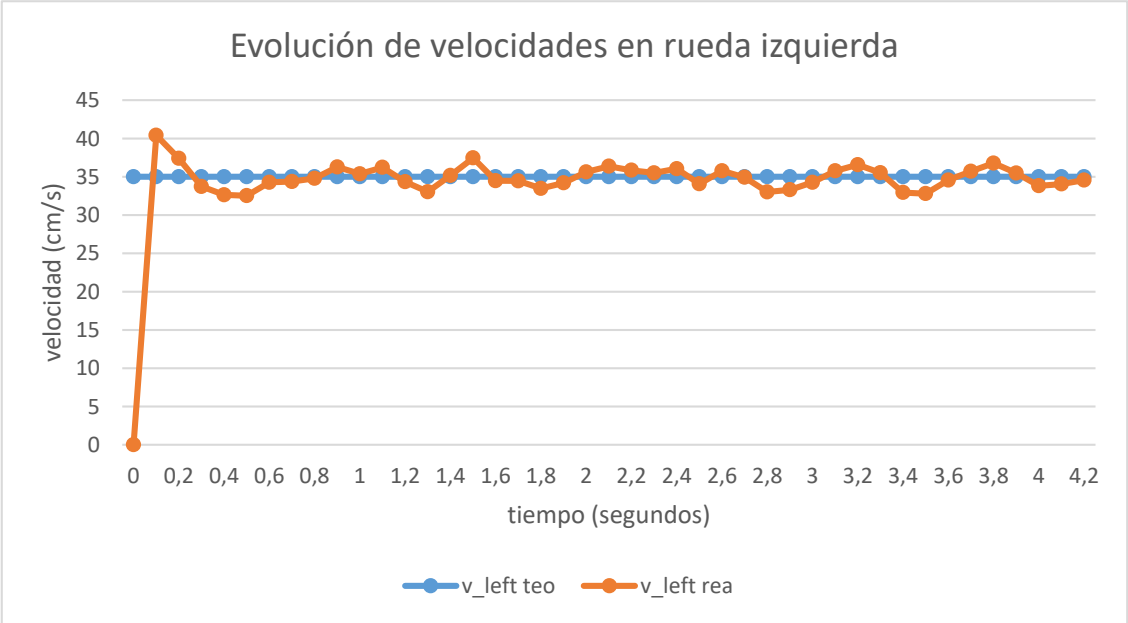

Figura 4.3. Gráfica que muestra la evolución en la rueda derecha desde el reposo hasta la velocidad objetivo con un T=0.1 segundos donde la línea azul representa la velocidad pedida por el usuario (35 cm/s) y la línea naranja la velocidad llevada por el robot en cada instante de tiempo..

4.2.1 Descripción

Para que resulte más fácil la comprensión del programa, se dividirá la explicación en varias partes. Las partes referentes a la entrada de argumentos y al cuerpo principal del programa se detallarán en el apéndice I.1, dejando para este capítulo todo lo relacionado con la función de control que gobierna este programa, la función *deadbeat ()* (figura 4.4).

Una vez definidas las características que tendrá nuestra línea recta se deberá pasar al control como tal de dicha trayectoria. Como se ha especificado a lo largo de los anteriores capítulos, se hará un control sobre la velocidad del robot buscando que converja a la velocidad deseada. Mientras no se haya recorrido la totalidad de la trayectoria, el programa entrará en la función deadbeat ()[8] en la cual se hará lo siguiente:

Inicialización
Bucle de control:
    Para cada T
        Medir marcas (obtener "iempo")
        Obtener vRight
        Calcular error
        Calcular acción correctora y mandarla al Pi Robot
    Fin

Este es el esquema teórico para la rueda derecha; siendo equivalente el de la rueda izquierda. A continuación se explicará este esquema con más exactitud.

Cada periodo de muestreo (T segundos) el programa entrará en las respectivas funciones "cuenta-marcas"[9] y leerá todas las marcas que se registren en ese tiempo. Cada vez que se detecte una marca, se actualizarán las variables que miden el tiempo en milisegundos que ha habido entre la detección de la marca anterior y la actual (variables iempo y *tiempoL)*. Una vez haya acabado este tiempo de muestreo, se cogerá el último valor registrado de las anteriores variables y se hallará la velocidad que lleva el robot en ese preciso instante. Para calcular la velocidad se coge el último dato y no la media de valores de velocidades que hayan podido obtenerse durante este periodo de muestreo. La razón es simple, si fuera un dato en el que hubiera mucho ruido y la interpretación de un solo valor pudiera llevar a error se haría la media con todos los valores detectados en ese intervalo. Dado que los datos de tiempos (variables *iempo* y *tiempoL)* son valores precisos que dan valores fiables acerca de la velocidad del robot en ese preciso instante, se escoge trabajar con el último valor de tiempo registrado.

Una vez tenemos la velocidad actual del Pi Robot, se compara con la teórica que se le ha fijado y la diferencia entre ambas constituirá el error. Cada rueda tendrá su error particular.

Después de obtener el error se resolverán las ecuaciones que nos darán la nueva acción que se deberá mandar a cada rueda para disminuir ese error y llegar a la velocidad fijada en el menor tiempo posible. A continuación se explicará teóricamente el procedimiento seguido para sacar la expresión de ambas acciones para luego acabar mostrando ambas expresiones [12].

[8] Nombre que recibe la función de control en el programa, se podrá ver en el Apéndice antes indicado
[9] Explicadas en el Anexo D, cómo funcionan y qué salida dan

Al principio se tiene para cada rueda la función de transferencia G(s). Dado que estamos en el dominio discreto es necesario para ambas expresiones al dominio Z. Para la rueda derecha se tiene por ejemplo*:*

$$G(s)_{rueda\ derecha} = 0.23308\frac{1}{1 + 0.1336 * s}$$

A continuación se declararán nuevas variables donde *Ks derecha = 0.23308 y* $\tau_{derecha} = 0.1336$

Operando se logrará *G(z):*

$$G(z) = \frac{K_{z\ derecha}}{z - \tau_{z\ derecha}}$$

$$K_{z\ derecha} = K_s * \left(1 - e^{\frac{-T}{\tau_{derecha}}}\right)$$

$$\tau_{derecha} = e^{\frac{-T}{\tau_{derecha}}}$$

$$G(z)_{rueda\ derecha} = \frac{0.1228}{z - 0.4731}$$

La expresión del regulador discreto diseñado es de la forma:

$$R(z) = \frac{U(z)}{E(z)} = \frac{1}{G(z)} * \frac{D(z)}{1 - D(z)}$$

Recordemos que D(z) = z^{-n} . Operando quedaría*:*

$$R(z) = \frac{z - \tau_{derecha}}{K_{z\ derecha} * (z - 1)} = \frac{U\ (z)}{E(z)}$$

Transformamos la expresión anterior para poder implementarla en forma de algoritmo.

$$K_{z\ derecha} * (z - 1) * U(z) = (z - \tau_{derecha}) * E(z)$$

Ahora se dividen ambos lados entre Z.

$$K_{z\ derecha} * U(k) - K_{z\ derecha} * U(k - 1) = E(k) - \tau_{derecha} * E(k - 1)$$

Finalmente la acción se calcula de la siguiente forma*:*

$$U(k) = U(k - 1) + \frac{1}{K_{z\ derecha}} * E(k) - \frac{\tau_{derecha}}{K_{z\ derecha}} * E(k - 1)$$

Particularizando la expresión para ambas ruedas:

$$U_{derecha}(k) = U(k - 1) + \frac{1}{0.1228} * E(k) - \frac{0.4731}{0.1228} * E(k - 1)$$

$$U_{izquierda}(k) = U(k - 1) + \frac{1}{0.1202} * E(k) - \frac{0.4827}{0.1202} * E(k - 1)$$

Por último, y antes de enviar las nuevas entradas al robot, se comprueba que estas acciones estén dentro de los límites posibles del robot y en caso de no estarlo, se corrigen. Estas comprobaciones se hacen pensando en la primera o primeras iteraciones ya que en ellas va a ser donde se alcance un mayor error respecto a la referencia pedida por el usuario. Este error tan sumamente grande acarreará importantes acciones que pueden llegar a saturar. Al final del apartado se estudiarán los casos límite, la mayoría de los cuales tienden a ocurrir en el arranque.

Una vez mandada la nueva entrada, las variables se sobrescriben pensando ya en la siguiente iteración. Las variables que se modifican para la iteración "k+1" son los errores y las acciones anteriores de ambas ruedas que toman los valores de la iteración en la que aún se está, es decir la iteración "k". También la variables *"t_antR" y "t_antL" (que son los valores de las variables tiempoR y tiempoL de la iteración anterior)* las cuales son usadas para cerciorarse de que el robot se mueve. Si ambos valores son iguales quiere decir que el robot no ha leído nuevas marcas, es decir, no se ha movido.

4.2.2 Análisis de comportamiento

Una vez definido el control llega la hora de probarlo en la realidad para darse cuenta de los posibles fallos de concepto o los supuestos casos que no se han contemplado y que vendría bien tenerlos en cuenta de cara a hacer nuestro programa aun más fiable y robusto.

El funcionamiento teórico y normal de este programa hace que prácticamente todo el tiempo ambas ruedas vayan a la misma velocidad. Sin embargo pueden ocurrir ciertos supuestos que, de darse las pertinentes circunstancias arruinarían nuestro control.

El caso más claro y más perjudicial es el arranque. Es el momento en que más lenta va la rueda y se podrían dar dos sucesos fatales:

a) Dado que en el arranque está acelerándose, y como el tiempo de muestreo es tan corto puede darse el caso de que una rueda no lea marcas o haya variaciones entre las lecturas de ambas ruedas, originando una situación anómala (figura 4.5).

   Para esta ejecución se ha pedido que el robot vaya a una velocidad de 25 cm/s. La primera ejecución el robot detecta que está todo parado, por lo que el error es el máximo posible y la acción mandada es muy elevada (alrededor de 205 unidades).

   En la siguiente ejecución ambas ruedas se están moviendo ya que registran marca, pero es a partir de esta primera marca cuando se cuenta el tiempo de manera que si se lee otra marca se obtendrá la primera velocidad. Mientras que en la rueda derecha se han leído dos marcas y se tiene una velocidad, en la rueda izquierda solo se ha podido capturar una marca. Esto hace que a ojos del sistema, esa rueda sigue estando parada (no ha sobreescrito el valor de "*tiempoL").*

   Esto hace que mientras la acción de la rueda derecha empieza a decrecer (de 203 a 167), la acción de la rueda izquierda sube hasta incluso llegar a saturar. Se crea una descompensación entre ambas ruedas que impide ya la correcta ejecución, siendo que en la iteración 2 iban ambas ruedas "prácticamente" a la misma velocidad con la salvedad de que la segunda marca no ha llegado a tiempo.

```
def deadbeat():
    contR=contL=0
    v_antR=v_antL=1

    iterr=iterr+1
    #Bucle principal de la funcion
    st=time.time()
    while (time.time()-st)<T:
        trRight()
        trLeft()

    if (tiempoR==tiempoantR):
        v_actualR=0
    else:
        v_actualR= cm/tiempoR

    if (tiempoL==tiempoantL):
        v_actualL=0
    else:
        v_actualL= cm/tiempoL

    errR= v_objetivo - v_actualR
    errL= v_objetivo - v_actualL

    u_R = u_antR + (1/( k_right*(1-math.exp(-T/tau_right) )))*errR - (math.exp(-T/tau_right))/(k_right*(1-math.exp(-T/tau_right)))*err_antR
    u_L = u_antL + (1/( k_left *(1-math.exp(-T/tau_left ) )))*errL - (math.exp(-T/tau_left ))/(k_left *(1-math.exp(-T/tau_left )))*err_antL
    ##########################################
    ######## CONDICIONES ESPECIALES ###########
    # 1) SATURACION
    if (u_R>255):
        print "Rueda derecha satura con accion de " + str(u_R)
        vRight= 255
    else:
        vRight= int(u_R)

    if u_L>255:
        print "Rueda izquierda satura con accion de " + str(u_L)
        vLeft= 255
    else:
        vLeft= int(u_L)


    # 2) ACCIONES NEGATIVAS
    if u_R<=0 or u_L<=0:
        print "ACCION NEGATIVA"
        print "Se recalculan las acciones"
        err_antR= 0
        err_antL= 0

        u_R = u_antR + (1/( k_right*(1-math.exp(-T/tau_right) )))*errR - (math.exp(-T/tau_right))/(k_right*(1-math.exp(-T/tau_right)))*err_antR
        u_L = u_antL + (1/( k_left *(1-math.exp(-T/tau_left ) )))*errL - (math.exp(-T/tau_left ))/(k_left *(1-math.exp(-T/tau_left )))*err_antL


   # 3) AUMENTO DEL PERIODO DE MUESTREO EN ITERACION 2
   if iterr==1:
       T=2*T
   else:
       if v_objetivo>=30:
           T=0.1
       else:
           T=0.3

   if iterr==2:
           print "EXCEPCIONALMENTE AMPLIAMOS TIEMPO DE MUESTREO A 2*T PARA LA ITERACION 2"
```

```
velocidad = ":" + str(vRight) + "," + str(vLeft) + "\n"
s.write(velocidad)

print ('iteracion --> ', iterr)
print ('right --> ', contR, ' ||| left --> ',contL)
print ('tiempoR --> ',"{0:.5f}".format(tiempoR), ' ||| tiempoL --> ', "{0:.5f}".format(tiempoL))
print ('tantR --> ', "{0:.5f}".format(tiempoantR), ' ||| tantL --> ', "{0:.5f}".format(tiempoantL))
print ('velocidad actual right --> ', "{0:.2f}".format(v_actualR), ' (', v_objetivo,')')
print ('velocidad actual left  --> ', "{0:.2f}".format(v_actualL), ' (', v_objetivo,')')
print(velocidad + '\n')

#guardamos en vectores
vec_velR.append(round(v_actualR,2))
vec_velL.append(round(v_actualL,2))
#vec_error.append(err)
vec_uR.append(round(u_R))
vec_uL.append(round(u_L))

#cambiamos el valor de las variables (k-1)
err_antR= errR
u_antR= u_R
err_antL= errL
u_antL= u_L
tiempoantR=tiempoR
tiempoantL=tiempoL
```

Figura 4.4. Sintaxis en Python completa de la función Deadbeat() la cual lleva implementada el control con el mismo nombre. En ella se pueden observar las diferentes partes que presenta además de seguir las explicaciones sobre este programa con mucha más facilidad. Para que la imagen tuviera menos extensión se han suprimido todas las variables globales que intervienen siendo estas obligatorias en la sintaxis si se quiere que el programa funcione

Como medida extraordinaria lo que se hace es crear un contador de iteraciones de manera que si estamos en la ejecución 2, el tiempo de muestreo sea T'= 2*T. De esta manera doblamos el tiempo que el programa está leyendo marcas haciendo que sea más fácil que capture más de una marca.

b) También en el arranque se puede dar el siguiente suceso (figura 4.6). Imaginemos que se le pide al robot una velocidad intermedia, del orden de entre 25-30cm/s. Como se ha dicho antes, la ejecución empezaría con una acción de gran amplitud dado que hay mucho error. Esta acción provocaría que en las primeras iteraciones el robot se moviera mucho más rápido de lo demandado. Al detectar el robot que se está por encima del objetivo, sus acciones irán en la línea de intentar frenar el robot, pudiendo llegar incluso un momento de pedir una acción negativa, la cual arruinaría la ejecución.

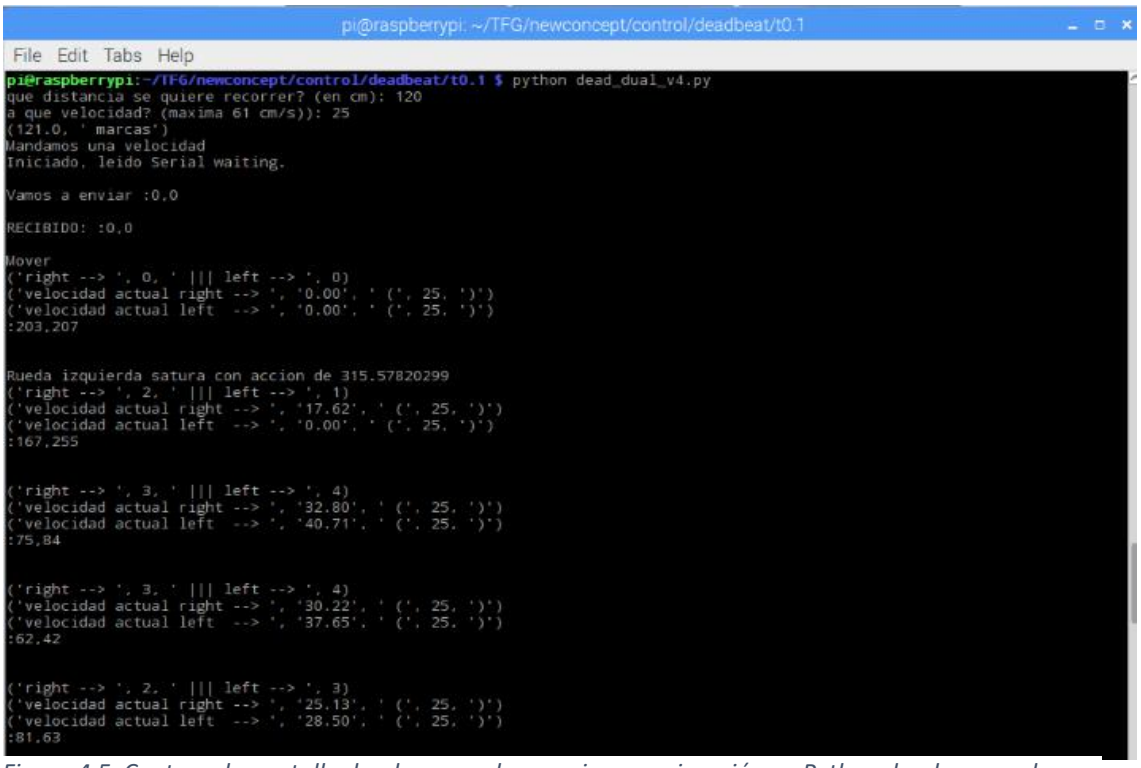

Figura 4.5. Captura de pantalla donde se puede apreciar una ejecución en Python donde se produce un error en el arranque el cual es crítico para el correcto funcionamiento del programa.

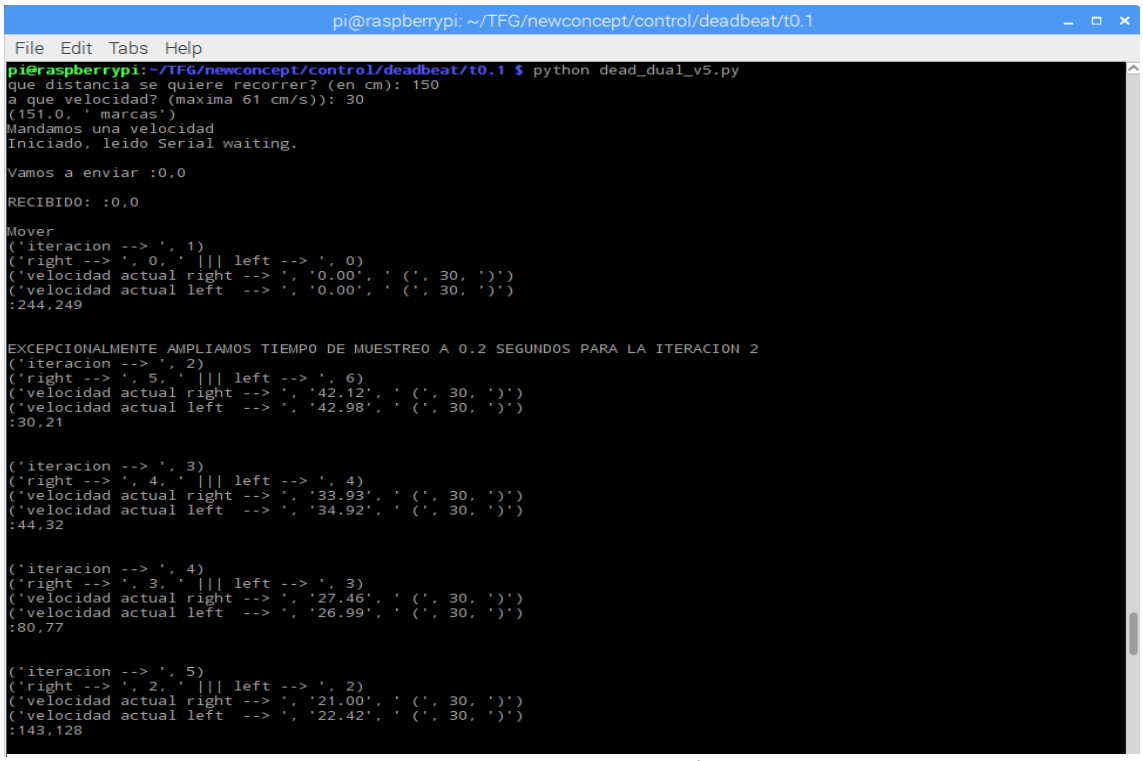

Figura 4.6. Captura de pantalla donde se puede apreciar la acción tan elevada al comienzo con la que se consiguen grandes velocidades y las acciones posteriores que disminuyen mucho su valor para intentar frenar el robot.

Como se observa en la imagen anterior se ha intentado hacer una trayectoria recta a una velocidad de 30 cm/s. Para obtener dicha velocidad la acción inicial es tremendamente elevada y para la segunda iteración se obtienen valores de velocidad alrededor de los 42 cm/s. Esto hace que se pase de unas acciones muy próximas al límite superior a acciones casi nulas.
En el caso de que la corrección implicara una acción negativa que arruinase la ejecución, como solución se ha propuesto recalcular la acción sin tener en cuenta el error anterior (*err_ant=0).*

Una vez vistos los dos problemas relacionados con el arranque, se comentará el último caso con el que habrá que tener especial cuidado, las ejecuciones a baja velocidad.

c) Otro problema que se tenía a la vista de los experimentos realizados para la obtención de las ganancias (apartado 3.1), era que las velocidades mínimas que podía desarrollar el robot de manera fluida eran aproximadamente de 20cm/s. Se intentaron hacer pruebas a velocidades menores de 20 cm/s y se encontraron los problemas comentados en el apartado a) de este capítulo.

   El único problema era el tiempo de muestreo, demasiado corto. Como solución se pensó en aumentar el tiempo de muestreo pero sólo para este tipo de ejecuciones. Así, para velocidades de intermedias a muy rápidas el tiempo de muestreo sería el pensado desde un primer momento (T=0.1 segundos) y para las ejecuciones "*a velocidad mínima"* el nuevo tiempo de muestreo sería T=0.3 segundos.

   Esto implicaba unas nuevas ecuaciones de control y hacer una distinción entre un método de control y otro dependiendo de la velocidad deseada por el usuario. Las nuevas ecuaciones se muestran a continuación:

$$U_{derecha}(k) = U(k-1) + \frac{1}{0.2084} * E(k) - \frac{0.1059}{0.2084} * E(k-1)$$

$$U_{izquierda}(k) = U(k-1) + \frac{1}{0.2063} * E(k) - \frac{0.1125}{0.2063} * E(k-1)$$

Esto solo se tiene en cuenta en el programa "*dead_dual_5a.py"* no usándose en la implementación de círculos. Esto es así dado que al haber ahí dos velocidades diferentes se puede dar el caso de que el programa le mande a una rueda ir a 27cm/s (T=0.1 segundos) y a otra a 16 cm/s (T=0.3 segundos) viéndose dos tiempos de muestreo diferentes dentro de una misma ejecución.

Una vez explicadas las principales características del control deadbeat aplicado a la realización de una trayectoria recta, se pasará a continuación a explicar el mismo control para realizar una trayectoria curva. La estructura y el control serán los mismos por lo que no se volverán a comentar, haciendo solo hincapié en las diferencias, las cuales se deben sobre todo la entrada de referencias y los posibles problemas que pueden surgir ya que de las no linealidades del sistema resultan en una mayor complejidad al diseñar estas.

4.3 Implantación del controlador Deadbeat para realizar trayectorias curvas

Una vez solucionada la implementación del control para realizar una línea recta fiable e intuitiva, surge la necesidad de hacer otro tipo de trayectoria, la trayectoria en línea curva que permita la realización de giros. El programa hecho en Python (apéndice I.2) servirá para la realización de todo tipo de circunferencias a cualquier velocidad (dentro del rango admisible por el robot), además de dar la opción de dar tantas vueltas en uno u otro sentido como el usuario desee. A continuación se detallará en diferentes partes toda la información necesaria para la comprensión de este programa.

4.3.1 Descripción

En este tipo de robots las ruedas motrices son independientes la una de la otra, por lo que no están conectadas mediante un mecanismo de dirección como ocurre en los turismos convencionales. Por lo tanto, para poder realizar un giro la única solución es que la rueda exterior vaya a una velocidad superior a la interior, ya que la rueda exterior deberá recorrer mayor distancia en el mismo tiempo.

Como se hacía anteriormente, para explicar las partes más importantes del programa como pudiera ser la introducción de las características del círculo a realizar o ver qué condición de parada se adopta se recurrirá al apartado I.2 de dicho apéndice. La única diferencia que hay entre ambos programas es que mientras que en la línea recta ambas ruedas tienden a la misma velocidad, en el giro cada una tenderá a una velocidad diferente siendo la velocidad de la rueda exterior siempre mayor.

4.3.2 Análisis de comportamiento

Una vez escrito y compilado el programa de control llega la hora de probarlo y ser consciente de cómo funciona realmente nuestro programa y qué parámetros son críticos, si los hay. Este programa a diferencia del anterior tiene más circunstancias especiales y más particularidades.

Las dos circunstancias críticas reflejadas anteriormente en el arranque se tienen también en cuenta aquí, dado que el arranque (4.2.2 a) sigue siendo el punto más crítico de la ejecución. Como solución se aumenta el periodo de muestreo para que en la iteración crítica se capturen más marcas. Para el otro problema (4.2.2 b), en caso de darse, no se tiene en cuenta el error anterior y se continua con la ejecución.

Las demás circunstancias son propias solo de este programa y son las siguientes:

a) En primer lugar la situación que se comentará a continuación también tiene que ver con el arranque. Imaginemos una circunferencia en la que los errores iniciales sean tan altos que provoquen que la primera acción mandada a cada rueda sature, como ocurre en la figura 4.7.

   En ese instante ambas ruedas irán a la máxima velocidad (255,255). No será hasta un par de ejecuciones después cuando cada rueda llegue a su velocidad objetivo y vaya adecuándose a ella. Esto hace que en las primeras iteraciones el Robot vaya prácticamente en línea recta, hecho que es más notable cuanto más corto es el radio de la circunferencia.

En este ejemplo puede verse el problema tratado en el apartado 4.4.2 b). Al inicio, las acciones son altísimas sobre todo si lo miramos desde el punto de vista de la rueda interior. En esta rueda, en la segunda iteración se tiene una velocidad de 43,14 cm/s cuando su objetivo es de 35. Esto hace que haya errores tan grandes que lleguen incluso a hacer girar la rueda en el otro sentido.

Para solucionar ambos problemas de golpe se ha pensado en dar a cada rueda la entrada que teóricamente debería de llevar para lograr esa velocidad. Estas entradas teóricas se sacarán del estudio realizado para sacar las ganancias cuando se hizo el modelado de los motores (apartado 3.1).

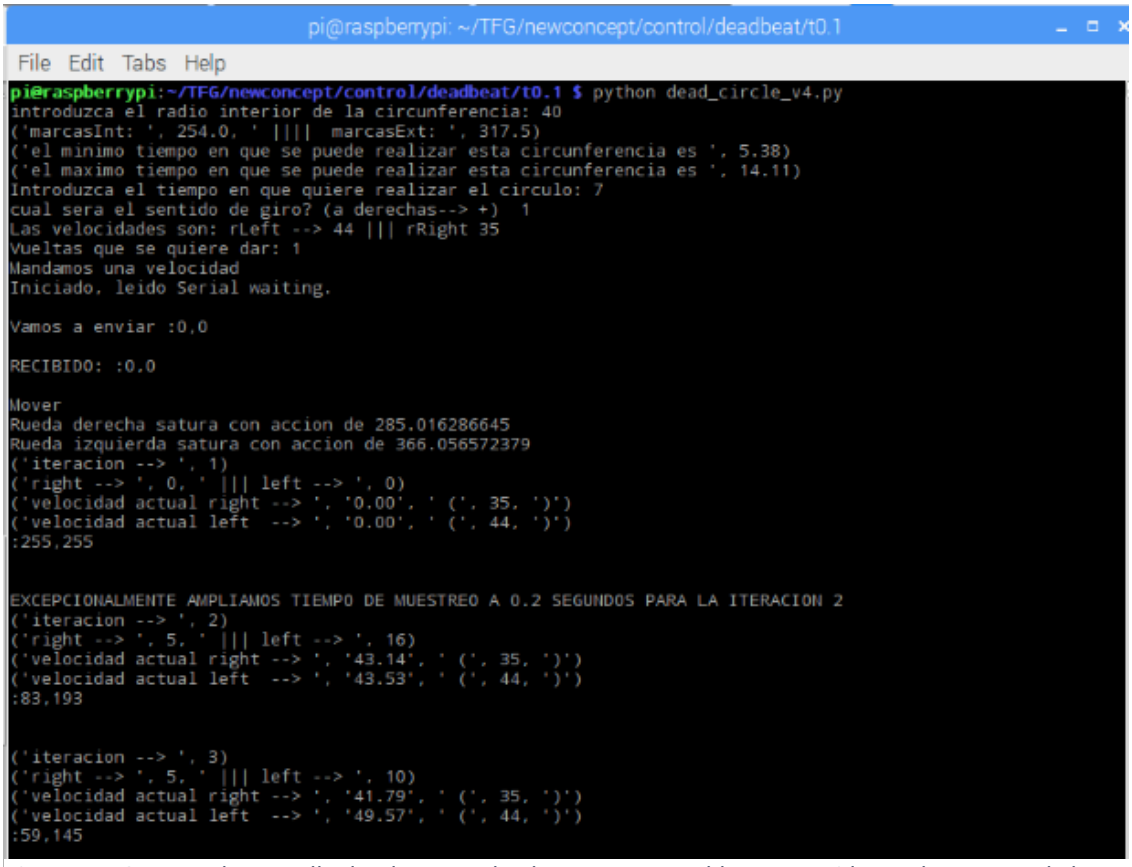

Figura 4.7 Captura de pantalla donde se puede observar otro problema ocurrido en el arranque de la ejecución de un círculo. Aquí, debido a lo altas que van a ser las velocidades, ambas acciones saturan siendo imposible en primera instancia la realización de un circulo.

Sabiendo qué velocidad le corresponde a cada rueda, el programa interpolará entre estas velocidades dando la entrada teórica que debería llevar la rueda en cuestión. Con esta solución se evitan los dos problemas antes mencionados. Las entradas iniciales no serán tan elevadas consiguiendo que la velocidad en los motores aumente más suavemente y sea difícil encontrar errores tan grandes que provoquen acciones negativas. Por otro lado, al ser acciones menos elevadas esto ayudará también a que estas acciones iniciales no saturen eliminando la posibilidad de hacer una línea recta en los primeros instantes de la ejecución.

b) Otro problema surgido es que el robot tenderá a hacer una circunferencia un poco más amplia de lo exigido por el usuario, por lo que completará la distancia exigida en menos distancia de la requerida.

   Un radio ligeramente superior conlleva una circunferencia ligeramente superior, por lo que el Pi Robot recorrerá la distancia calculada antes de llegar al punto de partida. El error apreciado es que el robot recorre el 90% de la distancia pedida. Dado que el número de marcas se va leyendo durante la ejecución, es relativamente sencillo calcular la corrección para completar el número de vueltas deseado.

4.4 Implantación de un control proporcional para realizar rotaciones puras

Una vez estudiados los controles que se encargan de que nuestro robot haga trayectorias precisas y a unas velocidades determinadas, se centrará la atención en implementar un tipo de control que funcione muy puntualmente. Su único cometido será el de garantizar que nuestro Pi Robot haga rotaciones puras lo más precisas posibles.

Con la implementación de este control se estará en disposición de hacer un sinfín de trayectorias y movimientos en el que el único límite será la imaginación del usuario. A continuación se explicará el tipo de control elegido y el porqué de esta elección además de las limitaciones y de las características del mismo. En el Apéndice H se reflejarán los problemas acaecidos con este control desde la idea inicial que se tuvo hasta el último método de control pasando por todas las circunstancias intermedias y las propias limitaciones del Robot que han hecho bastante complicado la implementación de dicho control.

Lo primero será definir como será el movimiento en un giro y para la realización de éste se ha optado porque solo se mueva una de las dos ruedas. En este caso se moverá siempre hacia adelante la rueda que en el giro resulte ser la exterior, de modo que para realizar una rotación de 90º en sentido horario, se moverá la rueda de la izquierda.

4.4.1 El control

Para este control se optará por un control proporcional. Es un control sencillo, el cual se basa en que la acción mandada es proporcional a un error, como se verá en la siguiente expresión y no se requiere nada más sofisticado para alcanzar el objetivo (Figura 4.8)

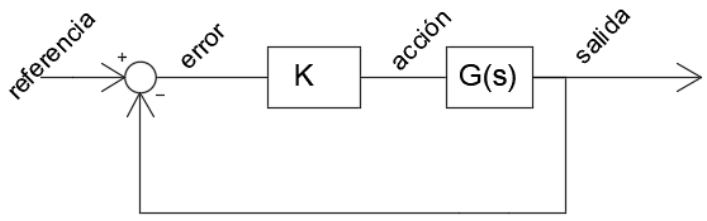

Figura 4.8. Esquema teórica del control proporcional en el dominio "s". Se puede ver claramente como la acción mandada sobre el sistema es proporcional al error medido por el sensor. Esta K será la que habrá que averiguar mediante experimentación

Aquí se debe buscar un equilibrio con la K. Una K muy grande podría hacer que para errores muy pequeños, se diera una acción elevada que hiciera al robot desplazarse de modo que estuviera todo el rato sobreoscilando intentando corregir errores minúsculos. Decir además que cuanto más sobreoscile el robot más difícil será la lectura del ángulo por parte de la IMU, dando una ejecución muy pobre. Por otro lado, con una K demasiado pequeña podría ocurrir que para errores aún considerables (20 grados), diera acciones tan minúsculas que no consiguiera hacer mover la rueda quedando la ejecución bloqueada sin solución aparente. En ese caso, el uso del integrador tampoco es adecuado porque las no linealidades del sistema lo hacen muy sensible a sobreoscilaciones para ese caso tal y como se describe en el siguiente apartado.

Por lo tanto, queda patente que la elección de una K adecuada será de suma importancia para que el control implementado tenga éxito.

4.4.2 Integrador

De entre las dos opciones expuestas anteriormente se escoge una K que tienda a ser un valor reducido. Prefiriendo encontrarse antes el fallo de intentar mover una rueda y debido a una entrada reducida, no poder hacerlo.

Para solucionar este problema introducimos un integrador de manera que si se le manda una acción y se detecta que el robot no se mueve se active este integrador. Una vez activado, el integrador introducirá un nuevo sumando a la expresión del cálculo de la acción siendo la expresión resultante la siguiente.

$$U = K_{regulador} * error + K_{integrador} * I_e$$

Siendo "*Ie*" el error integral. Este error cada vez que se active el integrador cogerá el error anterior y el actual. Esto provocará que este segundo sumando sea de mayor magnitud que el primero teniendo que ser su ganancia (k_i) menor para que una vez activado no sature.

De este manera se consigue que una vez el Robot se pare a falta de 30 grados (por ejemplo) en vez de acabar ahí la ejecución, se pueda realizar esta parte final a "tirones" y de poco en poco.

Como se ha dicho antes, esta expresión se activa cuando el control detecta que el Robot no se mueve. En principio la condición para entrar en este bucle sería que la velocidad angular fuese igual a cero, pero esto no es así dado que en cada ejecución el ángulo medido por la IMU cambia aun estando quieto el dispositivo. Ese cambio, aunque pequeño, no deja de ser una variación. Para ver cuánto cambiaba el ángulo (o la velocidad angular ya que se mide el ángulo durante un cierto T) se dejó el Pi Robot quieto y se midió la velocidad angular. Se observó que en ningún momento superaba los 2.5 grados/segundo por lo que la condición pasó a ser que mientras el robot no tuviera una velocidad angular menor de 2.5 esta solución no entraría a escena.

4.4.3 Movimiento de aproximación

Una vez implementado el control proporcional con el integrador para los casos especiales en los que tuviera que aparecer se probó la ejecución. La ejecución en sí iba como se había supuesto. Por ejemplo, para la realización de un ángulo de 60 grados al principio sólo actuaba el control proporcional pero al llegar a un error pequeño (25 grados aprox.) el robot ya no era capaz de actuar por sí mismo y entraba el integrador.

El problema era que cuando llegaba a valores cercanos al objetivo (5 grados), la variable "*Ie*" era tan grande (muchas iteraciones sumando el error) que aunque su multiplicador fuera pequeño proporcionaba una acción tal que el robot iba rápido en la parte última de la trayectoria pudiendo hacer que se pasase por bastante del ángulo pedido y se tuviera el problema de la sobreoscilación al final de la ejecución.

Como alternativa se propuso hacer algo parecido a lo hecho por los robots en la industria hoy día. Un ejemplo de esto podrían ser los brazos robóticos. El brazo robótico describe una trayectoria para llevar una pieza de un lado a otro por ejemplo. Una vez está cerca de la posición final su velocidad cambia y se reduce. Entra ahí un movimiento de aproximación al objetivo a mucha menor velocidad de la habitual consiguiendo más precisión en los últimos metros.

Este movimiento de aproximación llevado a nuestro Robot sería un movimiento para los últimos 20 grados de un giro. Una vez el error fuera menor de estos 20 grados el Robot giraría a una velocidad constante determinada para el sistema robot (entradas de (80,0) para giros en sentido horario, y de (0,80) para el otro sentido) de manera que siempre fuera la misma y no diera acelerones bruscos. En caso de pasarse del valor deseado se movería de manera opuesta hasta llegar al objetivo.

Capítulo 5

Evaluación experimental

Una vez se ha visto cómo va a funcionar el control para cada una de las diversas situaciones que se van a afrontar en el futuro, llega la hora de probarlo en la realidad. Hasta ahora todo lo que se ha pensado ha sido desde el plano teórico por lo que para que tenga validez, esa teoría debe ser llevada a la práctica.

En este capítulo se harán diferentes experimentos para cada control de los presentados en el capítulo 4 de esta memoria. Estos consistirán en:

1) El control en línea recta se probará solicitando al Pi Robot a hacer una línea recta durante una distancia prefijada.
2) Con el control en línea curva se buscará que nuestro robot haga un círculo de radio predefinido y ese giro lo repetirá tantas veces como quiera el usuario.
3) El control de giro busca realizar una rotación pura de un determinado ángulo de manera precisa por lo que se le introducirá un ángulo objetivo cualquiera y el robot controlado por los algoritmos desarrollados debería ser capaz de abordar el problema sin mayor dificultad.
4) Por último se probará el funcionamiento de ambos controles funcionando de manera conjunta en la realización de diferentes trayectorias como pueden ser un cuadrado y un movimiento de ir a una posición cualesquiera.

5.1 Trayectoria recta

Una vez lanzado el programa en Python, éste pedirá al usuario una serie de variables que caractericen la ejecución a realizar (distancia, velocidad, etc). Después de introducir estos datos el robot iniciará su ejecución, actuando como se ha explicado en el punto 4.2.1.

Como se ha comentado previamente, cada rueda buscará ir a la velocidad objetivo que el usuario haya introducido con anterioridad y tendrá que variar su acción dependiendo, entre otras cosas, de la iteración anterior. La expresión de la acción es la que corresponde al control deadbeat y es la explicada con anterioridad en el apartado 4.2.1. Cuando ambas ruedas consigan moverse a la velocidad objetivo el robot se moverá en línea recta ya que, si ignoramos posibles perturbaciones como deslizamientos de las ruedas con el suelo es inmediato pensar que si dos ruedas se mueven a la misma velocidad y sentido, van en línea recta.

El periodo de muestreo es de T= 0.1 segundos por lo que cada 0.1 segundos el programa analiza lo que ha pasado y manda una orden en consonancia. Como se comentó en el modelado (ver apartado 3.5) , según la velocidad pedida por el usuario este periodo de muestreo puede ser también T= 0.3 segundos , siendo la ejecución análoga al caso de T=0.1 segundos.

```
print ('iteracion --> ', iterr)
print ('right --> ', contR, ' ||| left --> ',contL)
print ('tiempoR --> ',"{0:.5f}".format(tiempoR), ' ||| tiempoL --> ', "{0:.5f}".format(tiempoL))
print ('tantR --> ', "{0:.5f}".format(tiempoantR), ' ||| tantL --> ', "{0:.5f}".format(tiempoantL))
print ('velocidad actual right --> ', "{0:.2f}".format(v_actualR), ' (', v_objetivo,')')
print ('velocidad actual left  --> ', "{0:.2f}".format(v_actualL), ' (', v_objetivo,')')
print(velocidad + '\n')
```

Figura 5.1. Sintaxis de Python de los diferentes textos que se van a mostrar por pantalla en cada iteración para dar información al usuario acerca de la ejecución actual.

Para que sea más fácil el seguimiento para el usuario, el programa enviará por pantalla cierta información acerca de la ejecución que se esté haciendo en ese preciso instante. Esta información es la mostrada en la figura 5.1 y consta de:

1) La iteración en la que se encuentra el programa.
2) Las marcas totales que se han registrado en cada rueda en el periodo de muestreo anterior (en la primera iteración serán las marcas registradas en 2*T tal como se explica en el apartado 4.2.2).
3) El último valor medido de las variables *tiempoR* y *tiempoL.* Con este valor se calculará la velocidad actual de las respectivas ruedas. También se mostrará esta misma variable pero relativa a la iteración (k-1).
4) La velocidad actual de cada rueda, mostrándose al lado la velocidad teórica a la que tienen que llegar de modo que el usuario pueda ver cómo de cerca están las ruedas de su objetivo.
5) Por último, se imprime por pantalla las acciones para cada rueda (entrada rueda dcha., entrada rueda izq.). Normalmente y con lo visto en algún ensayo, la rueda derecha es un poco más lenta por lo que es normal ver acciones más grandes sobre esta rueda.

Una vez vista la información que el programa presenta por pantalla se va a mostrar una ejecución completa del funcionamiento real del programa. En ella se podrá ver la evolución sufrida por ambas ruedas en su búsqueda de ir a la velocidad demandada por el usuario. Los datos que componen esta gráfica son los correspondientes a la ejecución de un línea recta de 2 metros de longitud y una velocidad de 35 cm/s.

Después de la gráfica (imagen 5.2) se explicarán los resultados. La ejecución mostrada en la figura es la ejecución que se puede ver en uno de los videos hechos durante los experimentos[10].

[10] https://www.dropbox.com/home/TFG/videos_trayectoria_recta?preview=Recta_2m_35cms.mp4

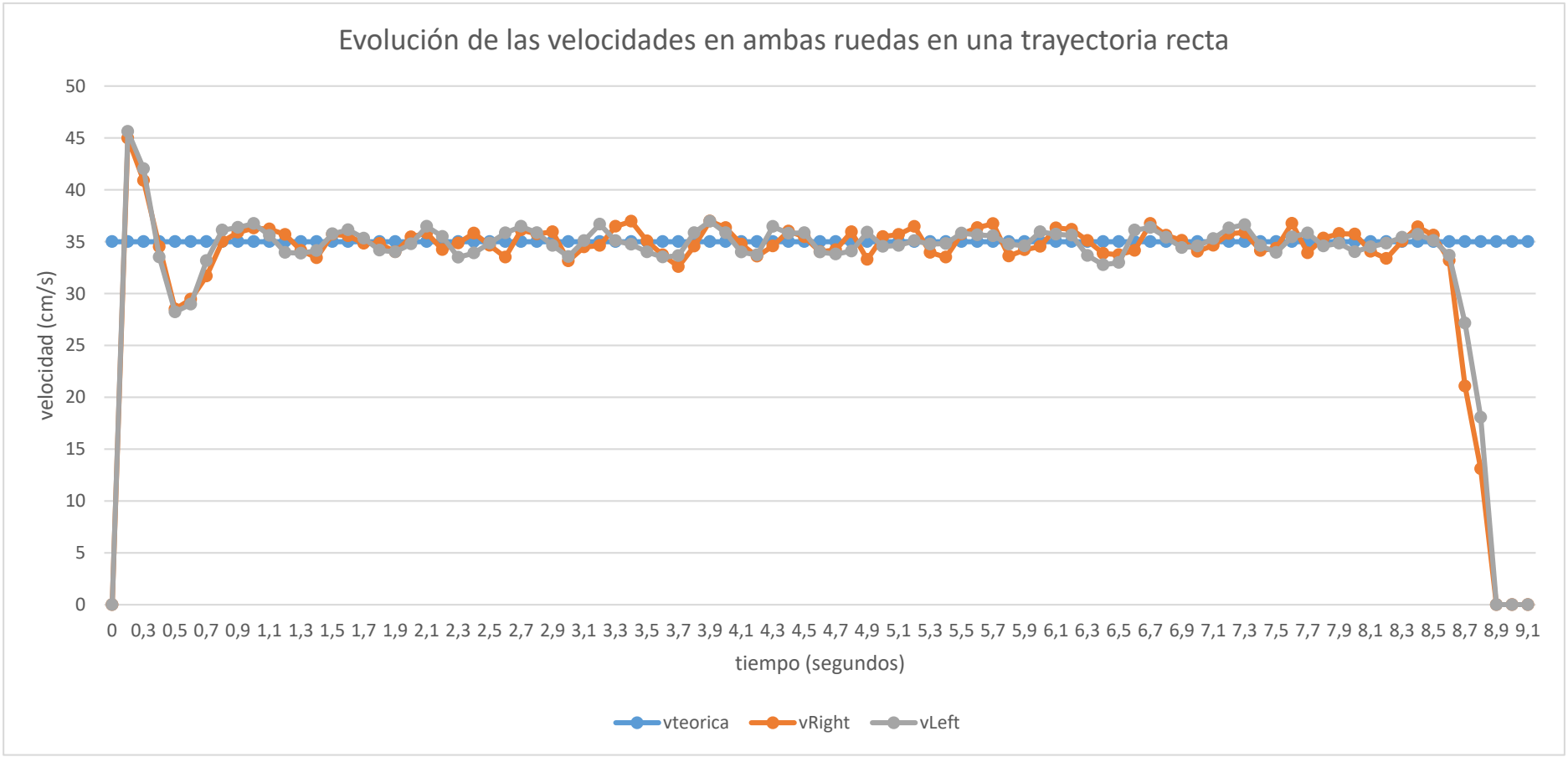

Figura 5.2. Evolución de las velocidades de ambas ruedas en el experimento de realizar una trayectoria recta. En este gráfico la línea azul indica la velocidad a la que deberían tener ambas ruedas y las líneas naranja y gris reflejan las velocidades llevabas a cabo por los motores de la rueda derecha e izquierda respectivamente.

Cómo se ha comentado, la ejecución hace referencia a una trayectoria recta de 2 metros de longitud que se debe recorrer a una velocidad de 35 cm/s. Al tratarse de una velocidad alta, la acción mandada inicialmente para lograr dicho objetivo es la más alta posible, motivo por el cual se llega a velocidades tan altas, llegando incluso a los 45 cm/s.

Una vez se disminuye la acción y se frena el robot, el robot tiende durante toda su ejecución a la velocidad pedida por el usuario (35 cm/s) estando durante los cerca de 9 segundos que dura la ejecución rodando en valores muy cercanos a este valor.

Por último, una vez se ha completado la distancia, el robot recibe la entrada nula (entrada rueda derecha = 0, entrada rueda izquierda = 0) con la intención de frenar su ejecución. Esta disminución de la velocidad no es instantánea tardando unas pocas iteraciones en llevarse a cabo. Al final de la ejecución el Pi Robot se consigue detener por completo.

5.2 Trayectoria curva

Ahora se analizará el programa[11] con el que se puede hacer una trayectoria del tipo circunferencia. Una vez lanzado el programa en Python, él mismo realiza una serie de preguntas para recabar información acerca de la circunferencia a realizar.

Una vez introducidos todos los parámetros necesarios para definir una circunferencia el programa empezará a ejecutarse con una dinámica equivalente a la que se tenía en el caso de trayectoria recta. En la figura 5.3 de la siguiente página se muestra un ejemplo del programa, en el que se reflejarán los datos obtenidos de la primera circunferencia de las dos que tenía que hacer en el ensayo, dado que más iteraciones no suponían mucha más información.

En cuanto a la ejecución del programa, se quiere realizar una circunferencia de 50 cm de radio. Cada círculo deberá completarse en unos 10 segundos y se desean dar 2 giros. Esta ejecución se puede ver en video[12]. Con estos datos, el programa calcula las velocidades que debería llevar cada rueda (31 cm/s para la rueda interior y 37 cm/s para la exterior). Con ayuda de las tablas obtenidas en los experimentos de modelado (apartado 3.1), el programa interpola para obtener las entradas equivalentes y las manda. A partir de ahí se va ejecutando el programa buscando que ambas ruedas vayan cumpliendo el objetivo de velocidad que se les ha impuesto.

En esta ejecución se pueden apreciar varias diferencias respecto a la ejecución de una trayectoria recta. La primera es que el arranque es distinto. Mientras que al inicio de la ejecución de una línea recta hay error máximo y la acción resultante es de gran amplitud, aquí la acción está impuesta (como se ha explicado en 4.3.2 a) por lo que el arranque es mucho menos brusco. El Pi Robot va progresivamente ganando velocidad, algo que puede observarse en que prácticamente no hay sobreoscilación.

[11] Dead_circle_5a.py

[12] https://www.dropbox.com/home/TFG/videos_trayectoria_curva?preview=Curva_r50cm_10s_2circulos.mp4

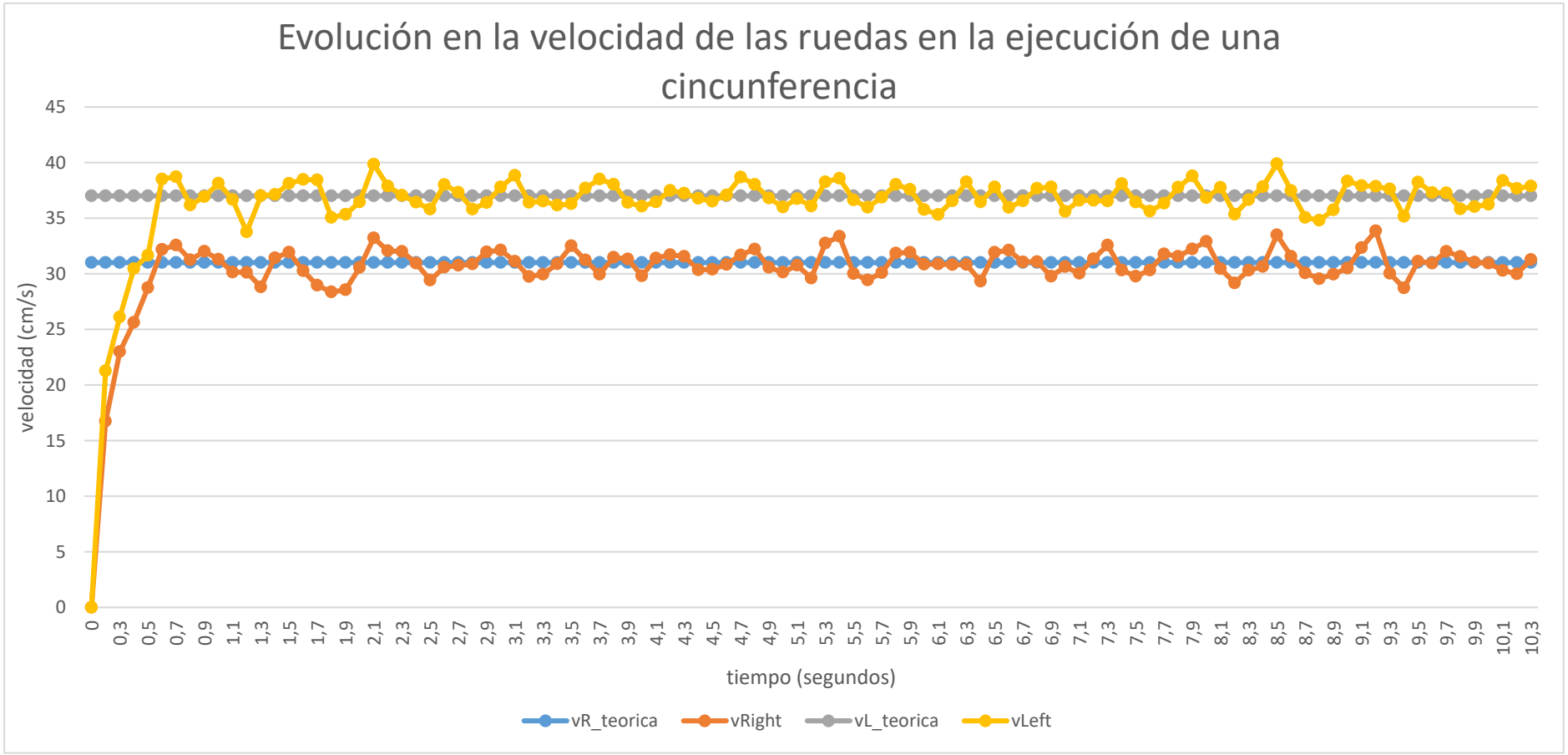

Figura 5.3. Evolución de las velocidades de ambas ruedas en el experimento de realizar una trayectoria curva. En este gráfico las líneas azul y gris indican la velocidad a la que deberían tener ambas ruedas y las líneas naranja y amarilla reflejan las velocidades llevadas a cabo por los motores de la rueda derecha e izquierda respectivamente. Se muestra 1 de las 2 vueltas realizadas en este ensayo.

El arranque es menos brusco, esto aumenta el control que se tiene sobre la salida y es igual de rápido que el control sin ”la asistencia en el arranque” en llegar a las proximidades de la velocidad pedida. Esto es así porque la lentitud que nos aporta este arranque suave se compensa con que el robot no tiene que gastar iteraciones en ajustarse debido a que no hay sobreoscilación (tarda unos 0.8 segundos como pasaba en la figura 5.2).

Una vez transcurrido el primer segundo de ejecución, los motores se van ajustando a lo demandado dando siempre valores muy cercanos a lo pedido. Esto pone de manifiesto la buena implementación que se ha hecho consiguiendo obtener un control de velocidades robusto y eficaz capaz de mover al robot a cualquier velocidad con una precisión alta.

Con el objetivo de mostrar más información al usuario y que vea en tiempo directo cómo trabaja el Robot se muestra la misma información por pantalla que para el caso de una trayectoria recta.

5.2.1 Efecto arrastre

Por último comentar una situación un tanto peculiar. Para explicar este fenómeno se mostrarán las acciones mandadas por el robot durante la ejecución del experimento anterior (figura 5.4). Aquí se puede apreciar cómo mientras las acciones mandadas sobre la rueda izquierda se mantienen cercanas al valor teórico, las de la rueda interior (derecha) son menores. Esta diferencia es alrededor de unas 15 unidades por debajo del valor teórico a simple vista.

Recopilando todos los valores de acciones mandadas durante los 10 primeros segundos de ejecución, se puede obtener la entrada media mandada. Estas medias son las siguientes:

a) Acción media sobre la rueda exterior = 154.76 (teórica = 155 uds.)
b) Acción media sobre la rueda interior = 116.33 (teórica = 133 uds.)

Las acciones sobre la rueda exterior muestran valores normales, no siendo así para la rueda interior (derecha). En esta ejecución, con una acción media de 116 unidades se alcanza una velocidad media de 31 cm/s.

Si ahora se va al punto 3.1 de la memoria, figura 3.2 (experimento en rueda derecha) se ven las velocidades que se obtienen con las diferentes entradas. Aquí se observa que para una entrada entre 110 y 120 unidades, la velocidad que le correspondería sería alrededor de 25.31 cm/s (no 31cm/s). O dicho de otra manera, para alcanzar una velocidad de 31 cm/s, la entrada teórica que tendría dicha rueda sería de entre 130 y 140 unidades en vez de las 116 unidades.

Este hecho viene a decir que en la rueda lenta, se logran velocidades superiores a las que le correspondería con esa acción mandada. Eso es debido al conocido como fenómeno arrastre.

Este fenómeno consiste en que la rueda interior recibe dos empujes. El primer empuje es el debido a la entrada mandada sobre el motor, en el ejemplo contemplado, la acción media mandada es de 116 unidades. La segundo empuje que habría que tener en cuenta sería la de la rueda exterior que gira a una velocidad superior. Este exceso de velocidad arrastra a la otra rueda a rodar a una velocidad mayor que la que teóricamente le correspondería.

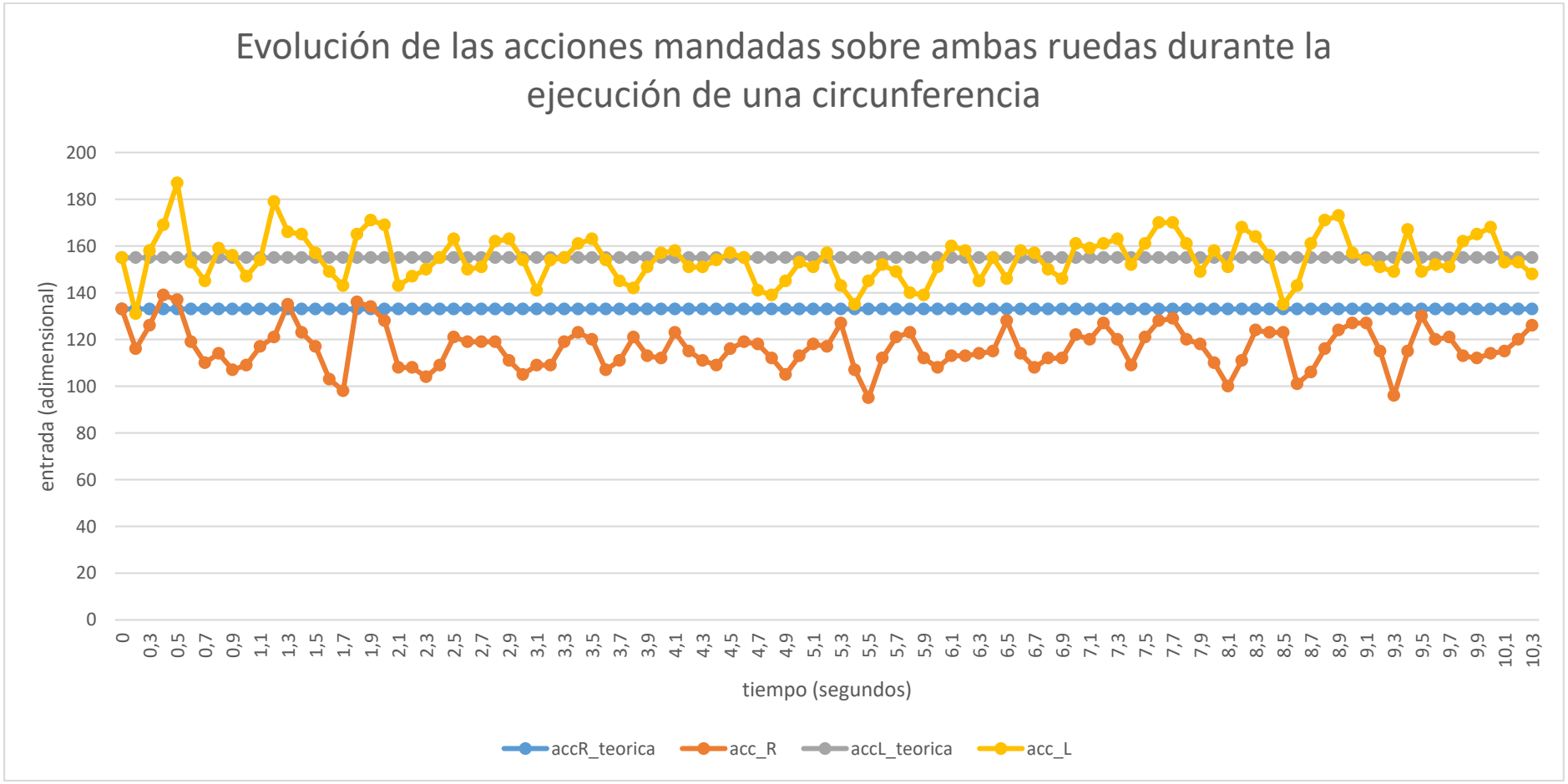

Figura 5.4. Evolución de las acciones en el experimento de realizar una circunferencia donde se puede ver la relación entre la velocidad real que deberían llevar las ruedas y la velocidad que realmente llevan. Las velocidad reales se representan mediante los colores naranja (rueda derecha) y amarilla (rueda izquierda) mientras que las teóricas en azul y gris respectivamente. En esta imagen también se puede apreciar el efecto arrastre que sufre la rueda lenta, la rueda interior (rueda derecha)

Este fenómeno no ocurre en la trayectoria en línea recta porque ambas ruedas van prácticamente a la misma velocidad, no hay una diferencia sustancial como para que una rueda contagie a la otra con ese extra de velocidad de más.

Pasando este fenómeno a datos numéricos, y fijándonos en la tabla de la figura 3.2 del punto 3.1 de la memoria , se ve que la acción exacta que correspondería teóricamente a 31 cm/s es:

$$\frac{31 - 29.85}{33.478 - 29.85} = \frac{U - 130}{140 - 130}$$

$$U = 133.17\ unidades$$

La acción teórica para que la rueda derecha alcanzara tal velocidad sería aproximadamente 133 unidades (que es con la acción con la que se empezó el experimento como se aprecia en la gráfica). Viendo el gráfico superior, nosotros logramos esa velocidad con una acción aproximadamente de 116 unidades por lo tanto el efecto arrastre como tal logra incrementar la velocidad teórica a una determinada acción en un 13% aproximadamente.

5.3 Programa cuadrado

Una vez vistas las ejecuciones por separado llega la hora de ver una combinación de ellas. Esta sucesión de instrucciones se asemejará más a lo que tenga que hacer el Pi Robot en su funcionamiento normal.

El objetivo pensado para llevar a cabo esta prueba será la ejecución de un cuadrado. El usuario definirá la longitud del lado L del cuadrado en centímetros y la velocidad a la que se moverá en cada línea recta que describa (en cm/s). No será necesario definir el ángulo a girar dado que siempre será el mismo (90 grados), lo que sí habrá que definir será el sentido de giro (horario o anti horario) . Una vez introducidos todos estos datos el robot iniciará la ejecución.

A continuación se mostrarán diferentes capturas de pantalla referentes al experimento, en el que se verá al Pi Robot haciendo todas las operaciones necesarias para lograr el objetivo. En este experimento los giros fueron controlados por la IMU (video[13]). Aunque como se dijo en el apartado 2.2, también podría encargarse de esta tarea el encoder en casos muy aislados como éste, donde el giro a realizar siempre es el mismo (video[14]).

[13] https://www.dropbox.com/home/TFG/videos_cuadrado/cuadrado_IMU?preview=Cuad_IMU.mp4

[14] https://www.dropbox.com/home/TFG/videos_cuadrado/cuadrado_encoder?preview=Cuad_Encoder.mp4

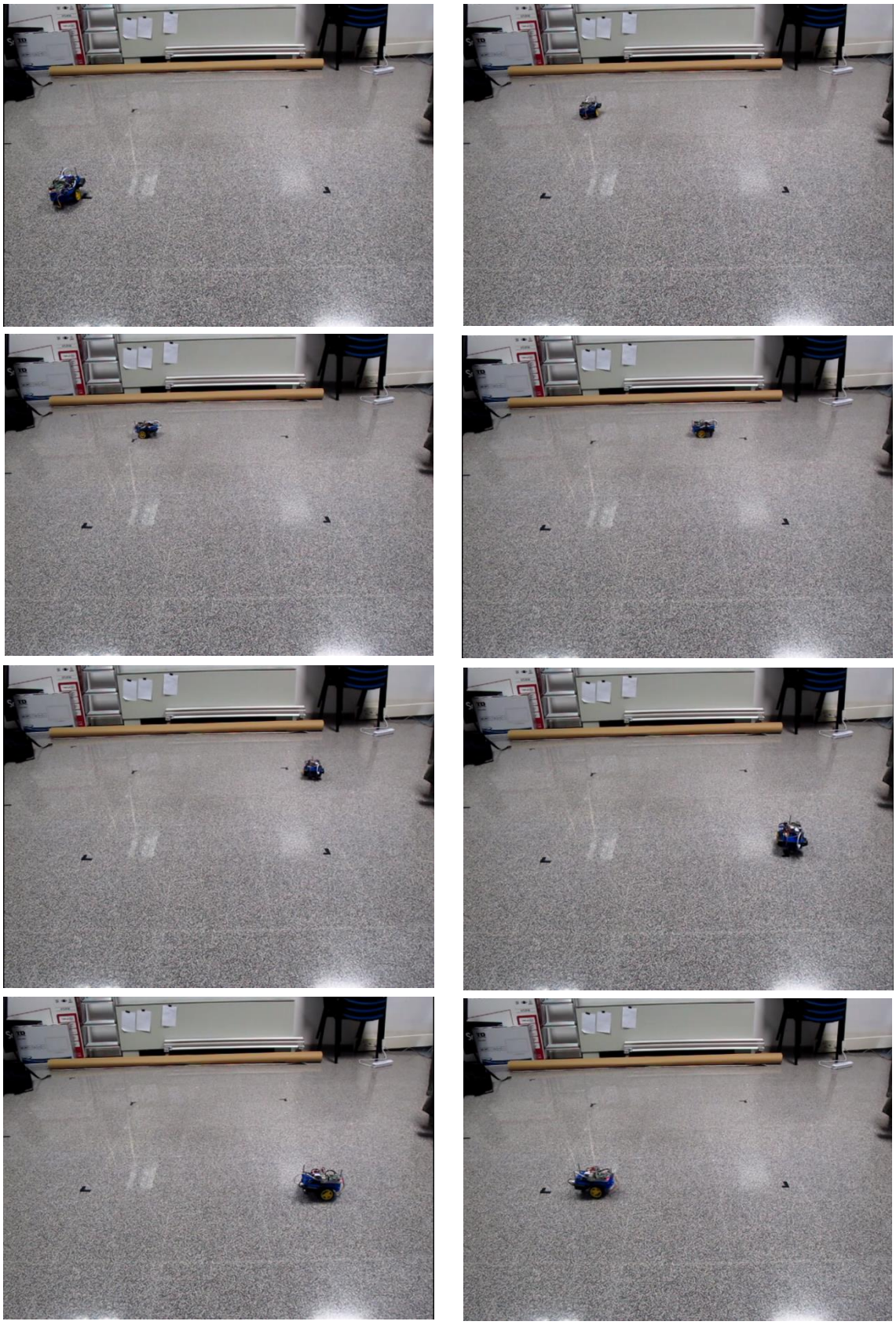

Figura 5.5. Sucesión de fotografías donde puede verse la ejecución de un cuadrado. Estas 8 fotografías han sido extraídas de un ensayo real realizado en un laboratorio de la Universidad de Zaragoza.

5.4 Programa coordenadas

Este programa como ya se ha comentado anteriormente nace con la misión de hacer que nuestro robot sea capaz de alcanzar cualquier punto del entorno con la unión de dos movimientos, un giro sobre si mismo y después una trayectoria recta.

Este programa combina los dos tipos de programas explicados durante toda esta memoria de manera que este programa es, de alguna manera, el objetivo final de la memoria. Conseguir que nuestro Robot vaya a un destino cualquiera impuesto por el usuario a través de trayectorias precisas que contengan giros y rectas.

Los controles que usa este programa no se explicarán dado que se basan directamente en lo explicado anteriormente. Lo que sí se explicará será la entrada de datos al programa. Esta entrada de parámetros usará el método de "ortogonalización de Gram-Schmidt" para el cálculo del ángulo de la trayectoria deseada. Esta técnica se explicará con detalle en el apéndice G.

A continuación se muestra dicho experimento en una sucesión de fotografías extraídas del video original. En ellas se podrá apreciar paso a paso la ejecución de este programa.

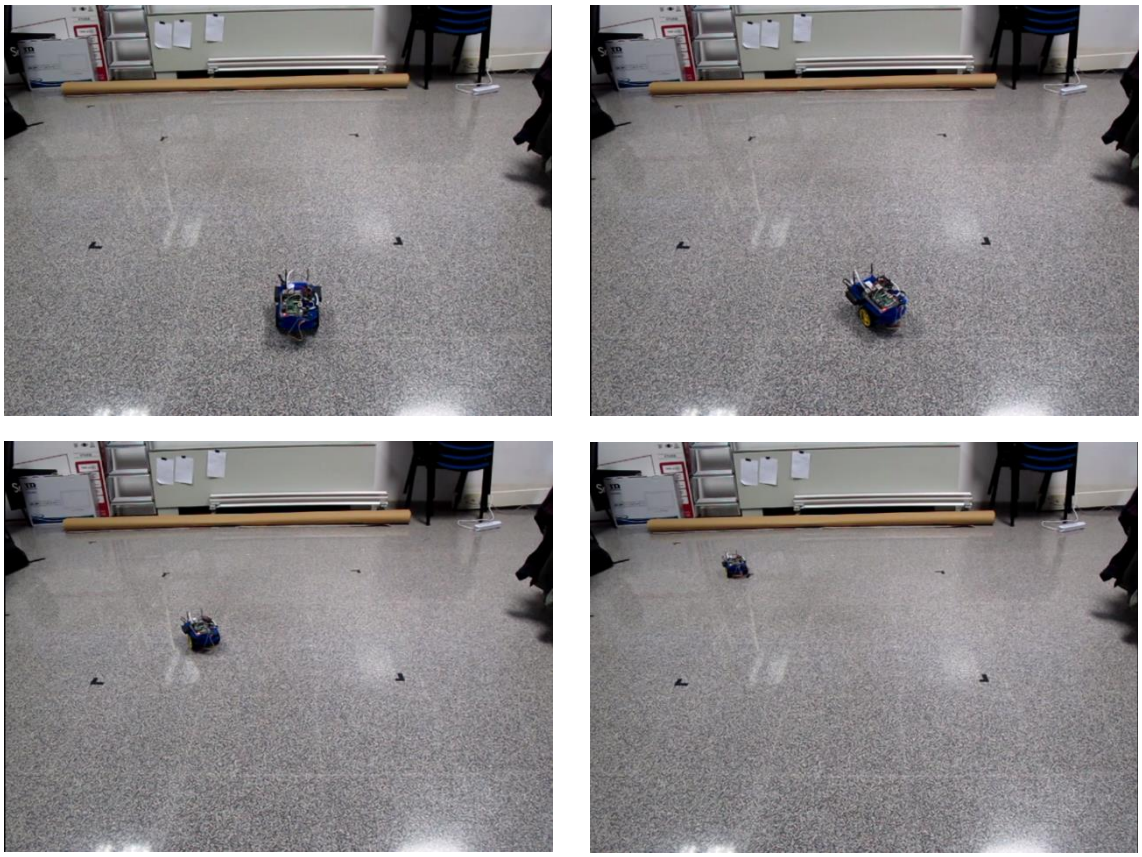

Figura 5.6 Sucesión de fotografías donde puede verse una ejecución en la que el robot debe alcanzar un punto definido por el usuario. En este caso el punto se encuentra a 75 cm a la izquierda del robot y a 160 cm por delante.

Este experimento podrá verse en el siguiente video[15]. De la misma manera, se ha realizado otro experimento consistente en dos ejecuciones. Este segundo experimento disponible en otro video[16] busca ir a una posición cualesquiera y acto seguido volver a la posición de salida.

5.5 Análisis de dispositivos

Una vez implementados los distintos tipos de programa y probados en sucesivas ejecuciones la siguiente pregunta que nos surge es si estos dispositivos de medición son precisos. En caso de no ser precisos los aparatos con los que se mide, no se puede garantizar la eficacia o los buenos resultados de nuestro trabajo. Esta pregunta se enfoca no desde el punto de vista del control de velocidad, sino desde el punto de vista de las condiciones de parada, esto es, desde el punto de vista del control de posición.

Como se recordará cada tipo de programa tenía su condición de parada. Mientras que en los programas de trayectoria recta, trayectoria curva.. (los programas donde el robot se desplazaba) la condición de parada era que el Robot mantenía su ejecución mientras no se llegase a la distancia pedida (apéndice I.1, parte III), esto no ocurría con los programas de rotación pura. En ellos la ejecución acababa cuando el error entre el ángulo girado y el ángulo objetivo era menor de ± 1 grado (figura 5.10).

Es inmediato pensar que para los programas donde el robot tenía que desplazarse, la condición de parada corría a cargo del enconder, mientras que en los programas de giro donde se miraban ángulos iniciales, ángulos girados etc., el instrumento utilizado era la unidad de medición inercial (IMU). En los siguientes apartados de este punto se analizarán ambos dispositivos ofreciendo al lector datos concluyentes acerca de la exactitud de tales instrumentos.

5.5.1 Precisión encoder

Tanto para los programas donde se realiza una trayectoria recta o una trayectoria curva, el usuario debe definir una distancia (o un radio, que haciendo la conversión pasa a ser una distancia). Esta distancia se pasa a su equivalente en marcas (gracias a la variable "*cm"* explicada en el punto 1.3) y a partir de ahí, el encoder ya tiene un valor con el que comparar. Con las marcas que tiene que recorrer el robot definidas, el programa no acabará (no se le enviará al robot la entrada (0,0)) hasta que no haya llegado a tal cantidad de marcas.

El experimento que se hará para ver la exactitud del encoder consistirá en recorrer una serie de distancias a distintas entradas y medir con un distómetro la verdadera distancia recorrida.

Las distancias que se medirán irán desde los 40 cm, considerando así las pequeñas distancias, hasta los 3 metros, considerando así distancias más importantes. También se estudiarán distancias de 1 y 2 metros. En cuando a las entradas evaluadas, se evaluarán las siguientes: 100, 140, 170 y 200 unidades, abarcando desde entradas relativamente lentas (100 uds), hasta entradas de bastante consideración (200 uds).

---

15 https://www.dropbox.com/home/TFG/videos_coordenadas?preview=(+75%2C160)_coordenadas_25cms.mp4

16 https://www.dropbox.com/home/TFG/videos_coordenadas?preview=IdaVuelta_30cms.mp4

La manera de proceder será la siguiente. Se empezará con la entrada más baja (100 uds) y se estudiarán todas las distancias antes mencionadas. Para cada distancia se harán 3 pruebas, para tener un resultado más estable. Después se hará la media de estas 3 mediciones y se calculará el error respecto a la distancia pedida. Una vez hayamos concluido con una entrada, se pasará a la siguiente y así sucesivamente.

En cuanto al aparato de medida se usará como se ha comentado anteriormente un distómetro digital (figura 5.7), facilitado por el área de Robótica de la Universidad de Zaragoza. Las características acerca de este medidor pueden encontrarse en [13].

Figura 5.7. Distómetro laser haciendo una medición como las hechas por nosotros para el estudio de exactitud.

Los resultados se anotarán en la tabla que se muestra en la figura 5.8 y se comentarán. La colocación del distómetro en el robot para conseguir medir las distancias se muestra en la figura 5.9. Se colocará encima de la placa Raspberry Pi y se buscará colocar lo más recto posible de manera que su rayo vaya en la dirección en la que se va a mover el robot. Deberá ir lo más estable posible debido a que los arranques y frenazos del robot suelen ser bruscos y podrían desestabilizar el medidor y con ello entregar información errónea.

Como se puede apreciar en la figura 5.8, en líneas generales nuestro dispositivo es bastante preciso. Como es lógico, a mayor velocidad siempre debería de haber un poco más de error debido a que con la inercia y el deslizamiento es más difícil el arranque y la parada por lo que recorre unos pocos centímetros de más. Lo más reseñable de este experimento es que el encoder no se comporta de manera lineal. La lógica puede llevar al lector a pensar que si por ejemplo, a entrada 200, cuando le pedimos que recorra 40 centímetros recorre 44, cuando le pidamos que recorra 100, el error será el equivalente, pero esto no ocurre así.

Se puede observar que prácticamente para todas las entradas probadas, da igual la distancia y siempre se pasa por aproximadamente la misma distancia. Esto provoca que conforme aumentemos la distancia del ensayo, el error será más pequeño en porcentaje dado que 4 cm pesan más en una distancia de 40 cm que en una distancia de 2 metros.

Exactitud en la distancia recorrida						
Entrada=100 uds						
Distancia en metros del ensayo	Distancia en m(distómetro)					
	medida 1	medida 2	medida 3		media (cm)	error (%)
0,4 m	0,405	0,416	0,402		0,408	1,917
1 m	1,011	1,011	1,016		1,013	1,267
2 m	2,01	2,02	2,02		2,017	0,833
3 m	3,035	3,009	3,013		3,019	0,633
Entrada=140 uds						
Distancia en metros del ensayo	Distancia en m(distómetro)					
	medida 1	medida 2	medida 3		media (cm)	error (%)
0,4 m	0,419	0,427	0,425		0,424	5,917
1 m	1,033	1,035	1,038		1,035	3,533
2 m	2,033	2,04	2,037		2,037	1,833
3 m	3,026	3,035	3,048		3,036	1,211
Entrada=170 uds						
Distancia en metros del ensayo	Distancia en m(distómetro)					
	medida 1	medida 2	medida 3		media (cm)	error (%)
0,4 m	0,424	0,425	0,433		0,427	6,833
1 m	1,038	1,039	1,032		1,036	3,633
2 m	2,042	2,032	2,039		2,038	1,883
3 m	3,047	3,05	3,039		3,045	1,511
Entrada=200 uds						
Distancia en metros del ensayo	Distancia en m(distómetro)					
	medida 1	medida 2	medida 3		media (cm)	error (%)
0,4 m	0,457	0,432	0,432		0,440	10,083
1 m	1,042	1,051	1,045		1,046	4,600
2 m	2,051	2,034	2,041		2,042	2,100
3 m	3,064	3,05	3,051		3,055	1,833

Figura 5.8. Tabla en la que se recogen los datos medidos en el ensayo de exactitud de la distancia recorrida por el robot. Consta de la misma prueba hecha a diferentes velocidades y donde se ejecutan hasta 4 diferentes distancias. Estas distancias se evalúan 3 veces, obteniendo su media y viendo el error en porcentaje.

A la vista de los resultados, se concluye que son bastante coherentes y bastante exactos y que demuestran que nuestro Pi Robot es suficientemente preciso a la hora de ejecutar distancias, tanto en línea recta como en línea curva.

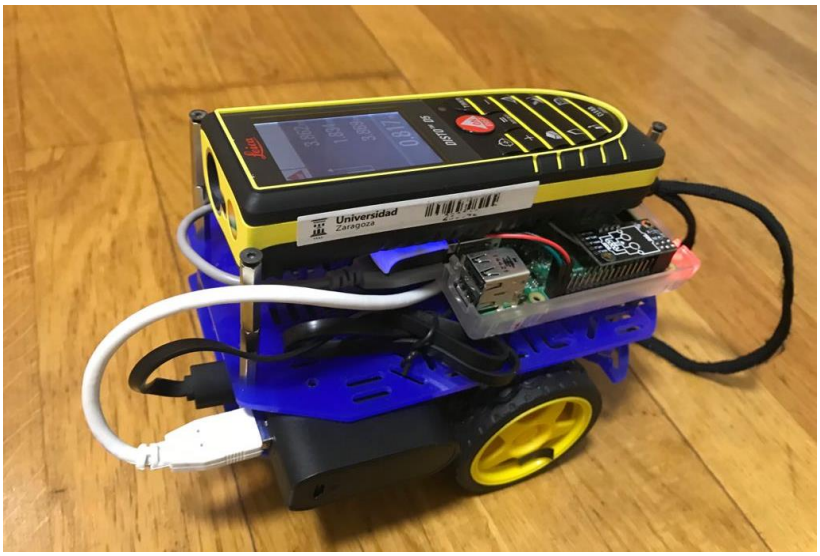

Figura 5.9. Colocación del distómetro para el análisis.

5.5.2 Precisión IMU

En definitiva, la unidad de medición inercial solo se usará para programas donde se requieran giros precisos (Programa coordenadas, Programa ida y vuelta, Programa cuadrado,…). Para estos programas, como es lógico, la condición de parada la marcará la IMU siendo que mientras no se llegue al ángulo objetivo, el programa siga ejecutándose. En la figura 5.10 se muestra la condición de parada para este tipo de programas.

Mientras se esté ejecutando el programa le mandará al Robot una serie de acciones para completar la misión de girar los grados deseados. Una vez el ángulo girado esté en el margen de ± 1 grado, el programa saldrá del bucle y enviará al Robot las entradas (0,0) consiguiendo detener el aparato.

```
while abs(-inigiro + gyroZangle - ang_objetivo)>1:
        spins()

s.write(':0,0\n')
print ('ha girado ', round((gyroZangle-inigiro2),3), ' grados en la ejecucion')
```

Figura 5.10. Sintaxis Python referente a la condición de parada cuando la condición la marca la IMU.

El experimento con el cual se verá la exactitud de la IMU consistirá en la realización de numerosos ángulos. Estos ángulos tendrán dos mediciones, la propia de la IMU que indicará los grados girados al final de la ejecución y la medición por parte de la app "compass" de un teléfono móvil. Este se colocará sobre nuestro Pi Robot como muestra la figura 5.11 de manera que se medirá el ángulo inicial y final y se comparará con el objetivo.

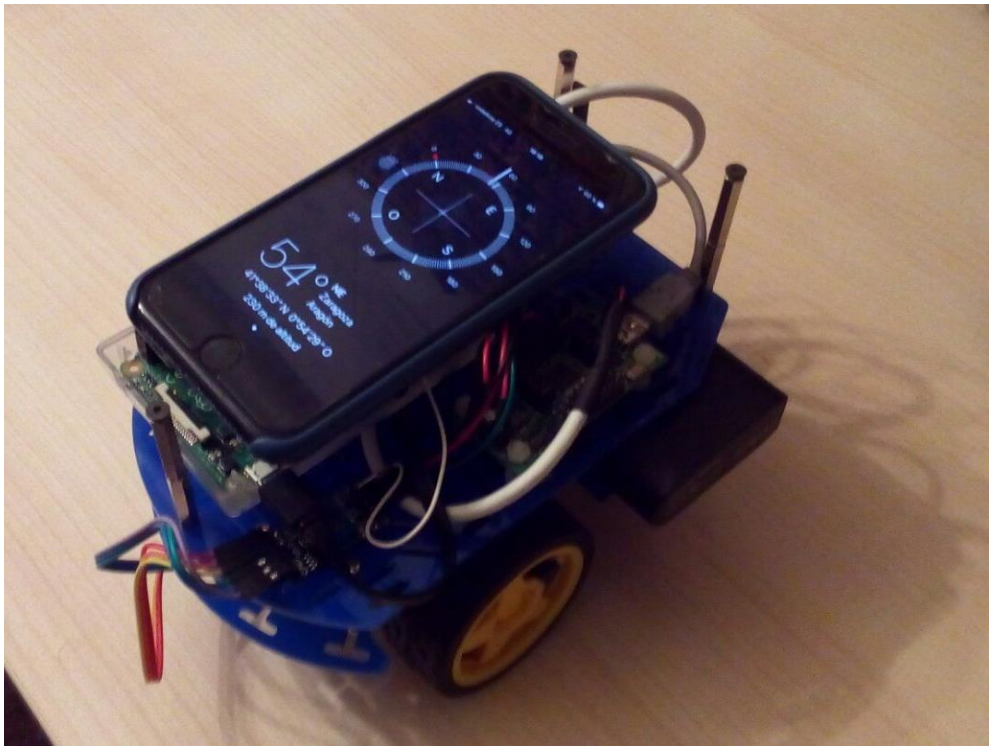

Figura 5.11. Colocación del teléfono móvil para el estudio de exactitud del ángulo girado.

Los ángulos que se ensayarán serán de lo más variado buscando cubrir cualquier solicitud. El experimento abarcará desde ángulos muy pequeños como puede ser el de 15 grados, hasta ángulos de mayor magnitud como pudiera ser el de 180 grados. Los demás ángulos que se estudiarán serán los ángulos de 30, 45, 60, 90, 105, 120 y 150 grados buscando que la mayor diferencia entre ellos sea de 30 grados y en algunos casos únicamente de 15 grados.

Cada ángulo se probará 3 veces con el objetivo de tener lecturas más uniformes. Los datos se adjuntarán en una tabla donde se podrá ver el error que se comete en cada caso. Al lado del resultado mostrado por la brújula, se mostrará el resultado mostrado por la propia IMU. De manera que el lector pueda ver que aunque una ejecución realmente haya girado más o menos de lo esperado, a ojos de la IMU se ha girado lo que se quería en un principio haciendo ver que el programa como tal funciona correctamente.

INTEGRADOR con velocidad de aproximación cuando se acerca al objetivo									
Kr=3 Ki=0,55	T=0,025 seg								
A dchas	mediciones								
angulo	med 1	medida IMU	med 2	medida IMU	med 3	medida IMU	media	IMU_media	error_real (%)
15	20	14,856	18	14,33	19	15,191	19,00	14,793	26,667
30	33	29,209	32	29,98	35	30,162	33,33	29,78233333	11,111
45	46	44,422	47	44,70	50	45,216	47,67	44,77833333	5,926
60	66	60,724	68	60,85	62	59,071	65,33	60,21466667	8,889
90	98	90,744	91	90,78	96	89,399	95,00	90,30766667	5,556
105	111	104,326	110	104,82	109	104,01	110,00	104,386	4,762
120	114	119,654	126	120,34	126	119,227	122,00	119,739	1,667
150	158	150,762	161	149,56	152	149,302	157,00	149,873	4,667
180	191	180,329	183	179,48	188	180,583	187,33	180,131	4,074

Figura 5.12. Estudio de exactitud de la IMU. Para este experimento se evalúan diferentes ángulos (columna izquierda) tres veces cada uno. Estos 3 ensayos son medidos por la brújula y por la propia IMU que al acabar la ejecución muestra por pantalla lo que percibe este dispositivo que ha girado. Luego se hacen las medias y se calcula el error en porcentaje.

A la vista de los resultados parece una herramienta de precisión intermedia, con errores de poca envergadura en cualquiera de los giros testeados. Si echamos la vista atrás, esta gráfica se podría asemejar algo a la expuesta anteriormente al hablar de la exactitud del encoder.

Como pasaba entonces, este error no se trata tanto de un error lineal sino de un error constante dependiendo de la ejecución. Como pasaba anteriormente, los errores más grandes ocurren en los ángulos más pequeños, dado que un error de 4 grados cuando el objetivo es 15 grados es mucho más notable que cuando se le solicita un giro de 105 grados. Por regla general el error medio más o menos es de unos 3 a 5 grados más del ángulo que nos pide, hasta llegar al ángulo de 150 y 180 donde ahí el error sube hasta los 7 grados.

Si uno se fija en lo que muestra por pantalla la IMU se puede apreciar como a diferencia del experimento anterior, aquí unas veces se rebasa el valor solicitado pero otras no se alcanza.

Como conclusión se puede afirmar que es una herramienta que cumple con su función de dar información del giro pero que carece de la exactitud necesaria para conseguir giros precisos y exactos. El ejemplo más claro lo encontramos en el ensayo de ángulo 90 grados. Aquí mientras que la IMU manifiesta que se ha girado poco más de los 90 grados pedidos por el usuario, la realidad nos hace ver que se han girado casi 100 en esa ejecución concreta (98 grados).

Esto hace que si se quiere realizar un programa donde el robot pueda hacer tantos cuadrados como le pida el usuario, sea muy difícil llevarla a buen puerto dado que no se garantiza que cuando el usuario le pida 90 grados, el Robot vaya a girar esos grados. Esto como se ha comentado no es problema de la implementación ni del programa ya que se ejecuta adecuadamente y según la IMU el objetivo se cumple pero queda patente que lo mostrado en la realidad no corresponde con lo dicho por la unidad de medición inercial.

Capítulo 6

Conclusiones

Como conclusiones a este Trabajo Fin de Grado considero que se han cumplido con los objetivos propuestos satisfactoriamente. Se ha conseguido convertir el movimiento extremadamente impreciso de un Pi Robot en un movimiento preciso para cualquier distancia y para cualquier velocidad (dentro del rango admisible que tiene el robot), consiguiendo una aplicación donde el robot puede ir a un punto deseado.

Esto también se ha visto favorecido por la buena precisión del encoder resultando ser una herramienta fiable pese a su bajo coste. No puede decirse lo mismo de la unidad de medición inercial. Este dispositivo no ha tenido un comportamiento tan preciso y exacto como el encoder, haciendo muy difícil su uso en la implementación. El resultado ha sido que las trayectorias que dependían de esta herramienta no fueran tan precisas. Pese a ello, se ha conseguido la realización de cualquier tipo de rotación pura aunque con una calidad algo inferior debido a que el comportamiento de este instrumento no ha dado pie a una mejor implementación.

En el ámbito personal comentar que ha sido todo un reto este Trabajo Fin de Grado. Verse sólo ante tantas complicaciones y el tratar de dar solución a todas y cada una de ellas ha hecho que creciera en ámbitos tales como el estudio y análisis de información tanto en español como en inglés (leyendo artículos como [14]) . Cabe destacar la gran diferencia que supone trabajar con un sistema que presenta importantes no linealidades, frente a los sistemas clásicos lineales vistos en el Grado.

He aumentado también mis conocimientos en ámbitos de ingeniería de control además de aprender un nuevo lenguaje de programación (Python, la información acerca de cómo dar los primeros pasos se muestra en [14]) en un entorno nuevo del tipo Linux [16]. Esto me ha hecho crecer en el campo de la programación dado que ha habido que hacer un sinfín de pruebas de medición, de operaciones de medida de tiempo, de captura de información, de envío de información a otros archivos, de resolución de problemas ayudado de foros como [17] etc. Toda la documentación referente a programación citada en el presente Trabajo Fin de Grado está disponible en el grupo de Robótica para posibles trabajos futuros.

Me siento muy satisfecho con lo aprendido y con lo entregado al final. Este trabajo me ha enseñado hasta donde soy capaz de llegar si me propongo algo. Esta claro que mi tutor, Gonzalo López Nicolás ha sido de grandísima ayuda dándome las pautas de cómo actuar y como solventar según qué problemas. Aconsejándome desde el inicio y superando ambos, etapas pasito a pasito, siempre con actitud receptiva. Él ha sido parte fundamental de este trabajo y no podía acabar esta memoria sin dedicarle estas palabras de agradecimiento.

6.1 Trabajo futuro

Queda patente que en este Trabajo Fin de Grado, aunque amplio, no se han contemplado todas las posibilidades que ofrece este Pi Robot. En mi opinión hay dos grandes campos o circunstancias que pueden ser de interés para trabajos futuros.

La primera de ellas es la ejecución en un terreno irregular. En este tipo de terreno, con baches y demás obstáculos se puede conseguir que ambas ruedas vayan a la misma velocidad pero resulta mucho más complicado mantenerlo recto. La razón es que el bache puede provocar un desvío en la trayectoria y el robot, al no tener medio para medir este error, no detectarlo y continuar con una ejecución no deseada. Otro problema con el firme irregular podría ser el encallamiento del robot en un hoyo. Aquí, el robot seguiría girando las ruedas pero al estar atrapado no podría salir del agujero. Al estar girando las ruedas el robot intuiría que está recorriendo distancia cosa que no es correcta.

Estas limitaciones se deben al contar con sensores propioceptivos como se indicó en el capítulo 2. Estos problemas podrían ser subsanados con sensores esteroceptivos, los cuales miden variables del entorno del Pi Robot.

Un segundo campo de trabajo futuro es la detección de obstáculos, mediante la implementación de alguna herramienta de visión.

Con una visión cenital también se podrían evitar los problemas antes expuestos referentes al terreno irregular. Ahí sí se podría ver que el robot no cumple con los objetivos de trayectoria o distancia, mandando acciones correctoras para intentar reconducir al robot o sacarlo de alguna situación comprometida.

Estas situaciones quedan fuera del alcance de este Trabajo Fin de Grado, dejándolas pendientes a otros trabajos que trabajen con visión.

Volviendo a nuestro Trabajo, y como se comentó al inicio de esta memoria, estos programas no solo son válidos para este Pi Robot sino que la modelización y las pautas para enviar datos, leerlos, transformarlos sirven para cualquier Pi Robot de características similares al estudiado. Por lo cual el trabajo desempeñado en este Trabajo Fin de Grado podrá ser usado por más personas para conseguir movimientos fiables y repetitivos de las ejecuciones que ellos deseen.

De hecho, este Trabajo Fin de Grado ha sido usado por Carlos Rubio, alumno que está realizando en estos momento su Trabajo Fin de Máster de nombre "Sistema multi-robot para cobertura persistente" (Figura 6.1). Él, utiliza un equipo multi-robot formado por Pi Robots similares al desarrollado aquí y que funcionan correctamente gracias al control desarrollado en este trabajo. Él, desde Matlab, calcula a que velocidad debería ir cada rueda y mediante una escucha sincronizada por medio de un servidor TCPIP lo envía y mi programa controla las ruedas.

Este Trabajo Fin de Grado es pues, es el primer paso hacia poder realizar un control real de los robots. Se puede decir que mi programa es el control al más bajo nivel, el de mi compañero Carlos sería el de medio y un control utilizando un planificador de trayectorias sería el control de más alto nivel.

Figura 6.1. Sistema multi-robot usado por mi compañero Carlos Rubio en su TFM de nombre "Sistema multi-robot para cobertura persistente"

Apéndices

Apéndice A

Encoder

En este apéndice se describen los encoder utilizados y se analizan sus características. Como se ha indicado en el apartado 2.1 de la memoria, el encoder es un sensor fotoeléctrico. Estos sensores basan su funcionamiento en la detección de un cambio en la intensidad de luz que es reflejada o bloqueada. En la figura A.1 se muestra cómo se coloca dicho dispositivo.

Figura A.1. Colocación del sensor del encoder y de los discos medidores

El motor al recibir un voltaje empieza a girar. Gracias a los engranajes que lleva en el se consigue que esta velocidad se reduzca y pueda impulsar la rueda que lleve acoplada. El centro de la rueda se conecta con la salida del sistema de reducción (recordemos que la relación de transmisión de este motor es 48:1) mediante una barra de color blanco que incluso sobresale como se puede observar en la figura anterior. En esta prolongación habrá que insertar nuestro disco medidor. De esta manera se conseguirá tener conectado el movimiento de la rueda con el movimiento de un medidor el cual sí se puede medir.

Aquí entra la importancia de tener un disco medidor de la máxima resolución posible para obtener lecturas lo más precisas posibles, razón por la que se desecharon unos discos medidores de 16 ranuras que funcionaban correctamente (apéndice A.3).

El encoder se encargará de medir este disco colocándose de la manera que se indica en la anterior figura. Debido a la importancia de esta operación, se deberá prestar especial cuidado a la hora de su colocación. Evitando el roce de cualquier material o una distancia incorrecta de manera que pueda producir lecturas erróneas.

A.1 Cómo se conecta

Lo primero que se debe hacer es observar la disposición que tiene la placa base del dispositivo. Esta placa se muestra en la figura A.2. La información acerca de cómo conectar el dispositivo con la Raspberry Pi se encuentra disponible en [18].

Figura A.2. Vista superior e inferior de la placa base de encoder

Como se puede observar, la parte superior tiene seis pines o salidas y la parte inferior solo cuatro. Las 6 salidas son para conectar la placa base con los sensores (3 pines para cada sensor). Cada pin tiene un dibujo ilustrativo que indica la finalidad de dicho pin (tierra, señal en bajo y señal en alto). Por otra parte se tiene la parte con cuatro pines que conectan la placa base con la Raspberry Pi. Como pasaba anteriormente cada pin tiene su función [DO2 (medidor 2), DO1 (medidor 1), GND (tierra) y VCC (entrada voltaje)]. Aquí se deberá prestar atención cuidado a cómo se conecta cada pin dentro de la Raspberry Pi dado que si se conecta mal el encoder no funcionará como se espera de él. Esta información ha sido obtenida de [19].

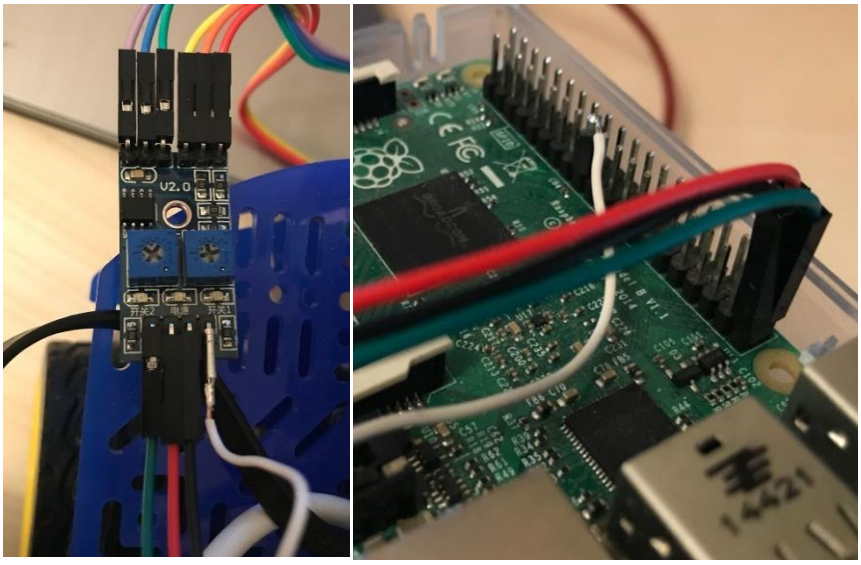

Figura A.3. Colocación de los respectivos cables con la placa base y con la Raspberry Pi para un funcionamiento óptimo del dispositivo

Para tener bien diferenciadas las conexiones y dado que no se podía masificar la zona superior de la Raspberry Pi (está la IMU) se decidió concentrar todos los cables en la parte inferior como se muestra en la imagen anterior.

Como se muestra en la figura A.3, la disposición de los cables ha sido:

Canal	*Color*	*Puerto Raspberry*
DO2	Verde	GPIO 21 (puerto 40)
DO1	Rojo	GPIO 20 (puerto 38)
GND	Negro	GND (puerto 39)
VCC	Blanco	3V3 (puerto 17)

Como los cables que unen los sensores con la placa base ya venían montados, no se precisó ninguna acción sobre ellos. Esta disposición de cables es de vital importancia dado que sino se hace con detenimiento puede provocar el fallo de dicho aparato de medida.

A.2 Cómo mide

Una vez conectado a la Raspberry Pi, el encoder recibe electricidad y empieza su funcionamiento. El encoder estará continuamente emitiendo un haz de luz; y dependiendo de si la señal llega al receptor o no, se emitirá un pulso eléctrico que la placa base de este dispositivo tendrá que encargarse de leer. Atendiendo a lo señalado en la hoja de especificaciones, disponible en [19], las características básicas de su funcionamiento son las siguientes:

Cuando llegue el haz de luz al receptor, la respuesta será una respuesta baja (el 0), mientras que cuando este cubierto y no llegue el haz de luz, la salida será alta (el 1). Para este dispositivo el 0 equivale a 0 voltios, mientras que el 1 equivale a 3,3 V. La figura A.4 muestra lo que aparecería en pantalla si conectáramos un osciloscopio a la salida.

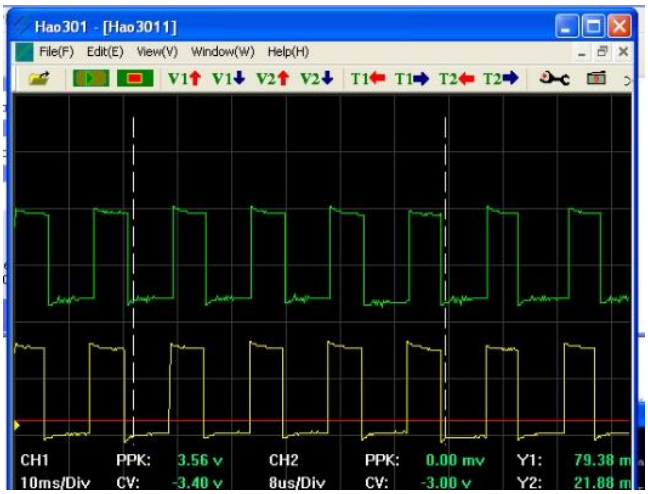

Figura A.4. Ejemplo de la salida que se vería si esta estuviera conectada a un osciloscopio. Aquí se puede diferenciar con claridad la señal en alto y en bajo de ambas señales.

A.3 Función original cuenta-marcas en lenguaje Python

Una vez conocido cómo mide el encoder, se deberá escribir en código Python las instrucciones necesarias que conviertan los sucesivos pulsos, o los valores en bajo y en alto (0 y 1), en variables con las que se pueda trabajar.

Para ello la manera más fácil es con la sentencia "if", que nos permite identificar el cambio que se produce en la señal (comparando el valor actual de la salida con el anterior para identificar el momento en el que cambia la señal ya que sino, no se puede saber si la rueda avanza o si permanece quieta). En la figura A.5 se muestra la sintaxis original usada para detectar marcas. Esta fue la 1º idea para capturar marcas y a partir de ahí, trabajar con ellas. Conforme se ha ido ganando experiencia con Python, esta sintaxis se ha visto modificada aunque la esencia es la misma. A continuación se detallará en que consiste este código de Python.

El apéndice D se dedicará única y exclusivamente a la descripción de la función usada en los programas para capturar marcas. Una vez llegado a ese punto, el lector podrá echar la vista atrás y comparar ambas funciones.

A continuación se detallará la sintaxis usada para este primer programa:

```
import time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setup(21,GPIO.IN)
GPIO.setup(20,GPIO.IN)

def rRight():
        global contR
        global v_antR

        pul21=GPIO.input(21)
        if pul21==True and v_antR==0:
                contR=contR + 1
                v_antR=pul21

        else:
                contR=contR
                v_antR=pul21

def rLeft():
        global contL
        global v_antL

        pul20=GPIO.input(20)
        if pul20==True and v_antL==0:
                contL=contL + 1
                v_antL=pul20

        else:
                contL=contL
                v_antL=pul20
```

Figura A.5. Sintaxis original del programa cuenta-marcas escrito en Python

Primero se conectan los pines a la Raspberry Pi y acto seguido se definen como entrada a ojos del programa. En este caso los pines GPIO 21 y GPIO 20 serán las entradas. Hecho esto, el valor de la entrada se asignará a una variable. La variable"*pul21"* (diminutivo de pulso entrada 21) va a coger la entrada de la rueda derecha que estará conectada al puerto GPIO número 21, mientras que la variable"*pul20*" dará información acerca de la rueda izquierda.

En cuanto a la captura del movimiento, la sintaxis usada en el programa indica que si el valor actual de salida (iteración i) de una rueda es True (=1) y el anterior (iteración i-1) es =0, existe un movimiento del disco medidor. En el caso de que así sea, se generan dos instrucciones:

- se incrementa en uno el contador de marcas de la rueda correspondiente y
- se cambia el valor anterior (v_ant) pensando en la siguiente iteración.

En caso de no darse esta circunstancia, el programa entraría en el "else" en el cual el contador permanecería constante y el valor anterior pasaría a ser el actual. El procedimiento es análogo para ambas ruedas.

A.4 Selección del disco medidor

Como se detalló en el primer punto de este apartado, el encoder medirá un pequeño disco medidor el cual va acoplado a su motor correspondiente. El disco se colocará entre medio del sensor para que alternativamente bloquee y deje pasar la luz. A continuación se comentará los diversos problemas acaecidos con esta rueda de plástico. Los primeros discos de los que se dispuso fueron como los mostrados en la imagen A de la figura A.6.

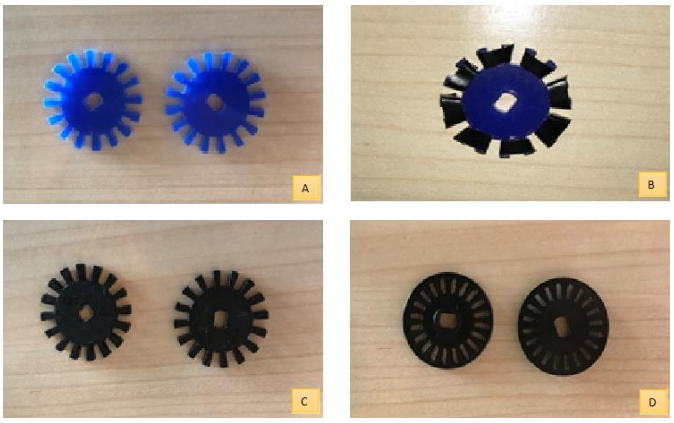

Figura A.6. Colección de fotografías donde se ilustran las diferentes ruedas usadas como disco medidor: a) Ruedas dadas con el encoder (16 huecos), b) Ruedas con cinta aislante, c) Ruedas pintadas en negro mate, d) Ruedas finales, de 20 huecos, las usadas.

El primer experimento para ver que efectivamente el encoder medía correctamente el giro de la rueda del robot, era un programa en el cual la sintaxis era algo similar a lo que aparece en la figura A.5. Se ejecutaba el programa mientras se giraba manualmente la rueda analizada. La sorpresa vino cuando por mucho que se girase la rueda de plástico, la salida que se mostraba por pantalla era el siguiente diálogo: "0 marcas leídas".

La primera razón que se pensó que podía ser la causa fue que el sensor fotoeléctrico estuviera mal colocado, demasiado alejado o demasiado cerca. Tras cambiarlo de posición en numerosas ocasiones el resultado seguía siendo el mismo.

Otra posible razón que se pensó era que el diente del disco medidor fuera demasiado estrecho por lo que el encoder no podía leerlo bien. Como solución a ese problema se pensó en unir dos dientes consecutivos con cinta aislante de color negro como se muestra en la imagen B de la figura A.4. Con este recurso se volvió a repetir el experimento comentado anteriormente obteniendo como resultado que el encoder por fin empezó a medir valores. Esto, aunque positivo, no era una solución óptima pues seguía presentando inconvenientes que dificultaban mucho la continuación de las mediciones. Primeramente era muy impreciso, debido a que la cinta aislante se despegaba, era irregular, etc. y en segundo lugar y más importante, se había reducido la precisión de 16 a 8 marcas. En cualquier caso, esta modificación lo que dejó patente es que el encoder funcionaba, ya que se pensó que había salido defectuoso.

Dado que no se encontraba una respuesta lógica a tal comportamiento (no detección de marcas con 16 dientes y detección cuando se ensanchaban) se pensó en ver la salida del encoder a través de un osciloscopio digital con los discos medidores azules mostrados en la imagen A de la figura A.6 (discos originales sin modificar).

Haciendo las conexiones pertinentes y ejecutando un programa que hacía que las ruedas del robot se movieran solas (ya no las tenía que mover el usuario), el osciloscopio arrojó los resultados mostrados en la figura A.7.

Figura A.7. Resultados del osciloscopio al medir dos ruedas de las denominadas A) en la figura A.6.

En la imagen se puede ver que el valor bajo corresponde a unos 120 mV mientras que el alto está alrededor de los 200 mV (Recordemos que las especificaciones decían que el valor bajo era 0 V y el alto a 3,3 V). Esta diferencia tan baja (80 mV entre el valor bajo y el alto), hacía que el programa fuera incapaz de detectar la diferencia entre ambas. En resumen, el disco original ofrecía una diferencia imperceptible.

A la vista de que con los discos con cinta aislante sí que se conseguía medir marcas, como experimento complementario se puso un disco medidor sin modificar en una rueda, mientras que en la otra se puso uno de los discos modificados con cinta aislante. El objetivo era ver si la salida era diferente, siendo la figura A.8 el resultado.

Aquí se puede ver como el canal 1 (disco normal), sigue arrojando la misma medición de antes mientras la de la cinta aislante muestra unos valores semejantes a lo que debería aparecer teóricamente. A la luz de estos resultados se pensó que el problema residía única y exclusivamente en el color del disco medidor; por ello se optó por pintar los discos medidores de color negro mate, como los mostrados en la imagen C de la figura A.6. Llegados a este punto comentar la concordancia de los resultados ya que la rueda de cinta aislante tiene un pulso cada dos pulsos de la rueda normal debido a que tiene la mitad de resolución. Como se ha comentado, el encoder arroja su salida en alto cuando se está bloqueando el haz de luz del sensor.

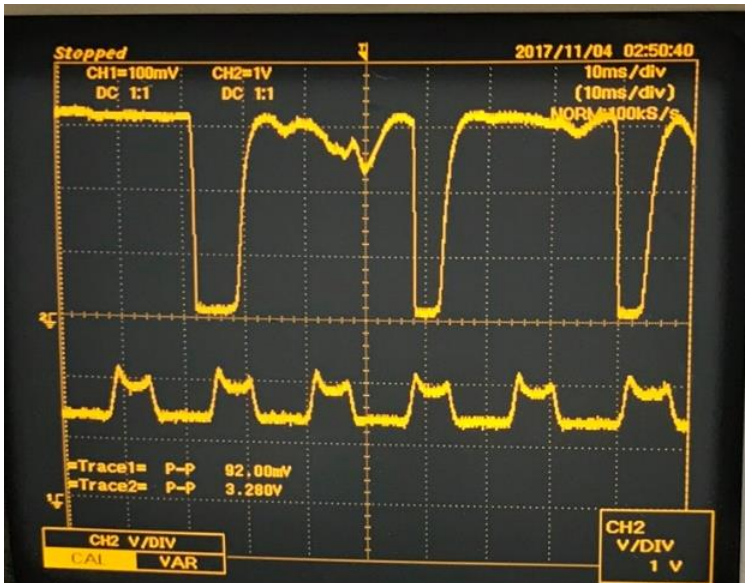

Figura A.8. Resultados del osciloscopio al medir, por un lado una de las ruedas denominadas como A) y otra como B) en la figura A.6. El canal 1 mide las ruedas sin modificar (A), mientras que el canal 2 mide las ruedas con cinta aislante (B)

Volviendo a hacer el experimento con las ruedas ya pintadas, las lecturas fueron las idóneas por lo que el color era la causa del fallo. Ya más adelante se consiguieron discos medidores de 20 huecos con lo que la resolución aumento en un 25%. Imagen D de la figura A.4.

El color de las ruedas originales era un azul translúcido y no se interrumpía del todo el haz de luz, de ahí la poca diferencia entre valores altos y bajos. Al cambiar a negro mate la situación cambio observándose por fin una diferencia reseñable con la que sí funcionaba el enconder.

Apéndice B

Zona muerta de los motores

En este apéndice se detallará un experimento adicional realizado al inicio del proyecto con el objetivo de dar información al usuario sobre el comportamiento en primera instancia de estos motores.

Este experimento consistía en analizar la zona muerta de los motores para dar al usuario una idea general de los motores por separado y en conjunto.

Con este simple experimento se podía dar respuesta a preguntas como:

- ¿Son ambos motores exactamente iguales?
- ¿Se comportan igual ante la misma entrada?
- ¿Se comportan igual los motores funcionando independientemente (funcionamiento en vacío) que funcionando en situación normal?
- …

B.1 Experimento

A la hora de poder mover el Robot nos surge una pregunta. ¿A qué velocidad (o entrada) puedo moverlo? La elección de la entrada influirá en el comportamiento futuro. Una entrada demasiado pequeña hará que el robot se mueva sumamente despacio o no se mueva, debido a que mover todos los engranajes y la rueda requiere una fuerza que un voltaje muy pequeño no va a ser capaz. Mientras, una entrada cercana al máximo hará un Robot muy veloz, difícilmente controlable y provocará una situación en la que un mínimo error, debido a la gran velocidad que pueda llevar el dispositivo, pueda provocar un error bastante más considerable.

Aparte de responder a las preguntas arriba expuestas, otro de los objetivos de este experimento es ver a partir de qué entrada el robot se mueve y ver el comportamiento en todo el rango de velocidades disponibles.

El programa usado para este experimento consistirá en:

Se le dará a nuestro robot todas las entradas comprendidas entre 0 y 300 con una separación entre ellas de 10 unidades. Esta entrada se mantendrá durante 3 segundos haciendo que el robot avance. Durante este tiempo los encoder estarán leyendo continuamente. Al final de la ejecución se tendrá por un lado el número total de marcas que ha leído cada encoder (mediante la conversión se podrá saber la distancia recorrida) y por otro se sabrá el tiempo de ejecución, 3 segundos. Sabidos estos dos datos se obtendrá la velocidad para esa ejecución. Con el fin de obtener lecturas contrastadas y que no den lugar a error cada entrada será repetida 3 veces. Acto seguido se hará la media y se apuntará en una hoja de cálculo para luego poder sacar la gráfica de manera que resulte más cómodo para el estudio. Dicho ensayo ha tenido dos partes.

B.1.1 Vacío

En esta primera parte se probarán todas las velocidades estando el robot en una posición de vacío, es decir, sin que las ruedas del robot estén en contacto con el suelo (libres de rozamiento). De esta manera las ruedas no tendrán ninguna resistencia excepto la mostrada por el aire y por sus engranajes internos.

En esta posición se podrá estudiar el comportamiento de cada rueda de manera independiente. Estando el robot en la posición normal de funcionamiento, puede darse el caso, como se verá más adelante, que aunque una rueda no gire, la otra sí este girando y arrastre a la primera de manera que marcará velocidad aunque en realidad, esa rueda por sí sola no ha podido moverse. Aquí, cada rueda solo se moverá debido a la fuerza que le proporcione su motor, no dependerá de nada más.

Ejecutamos el programa analizando cada velocidad y una vez acabada la ejecución lo llevamos a una Hoja de Cálculo para obtener la gráfica y verlo con más claridad (figura B.1). En las gráficas siguientes el eje X corresponderá con las distintas entradas dadas, mientras que el eje de las Y mostrará la velocidad (cm/s) del robot para cada caso.

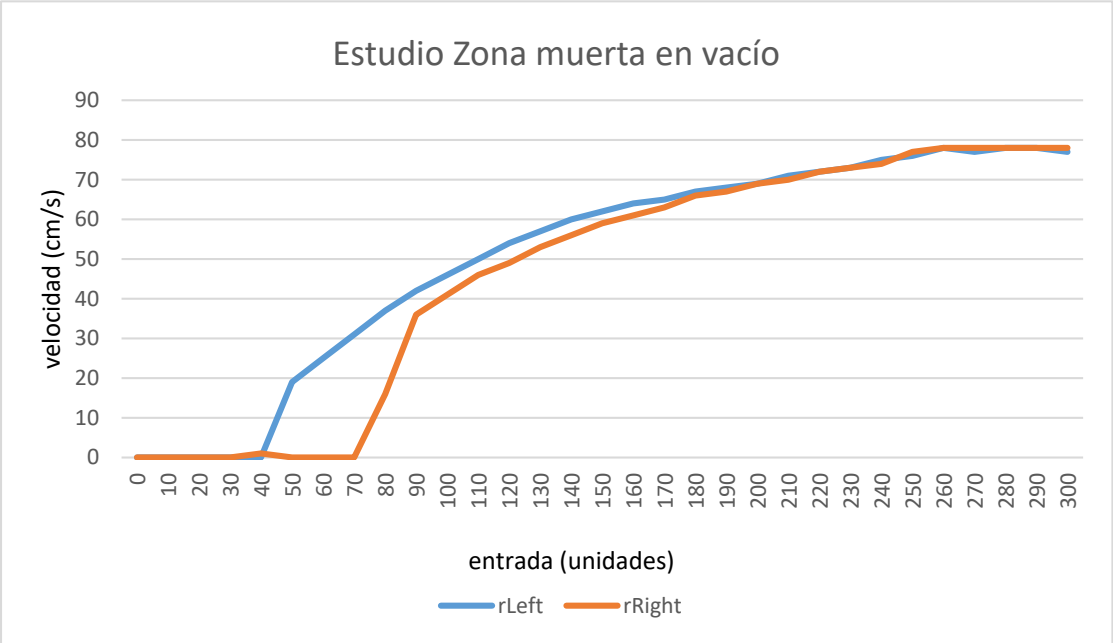

Figura B.1. Resultado del experimento en zona muerta realizado sobre ambas ruedas en la que la rueda izquierda esta representada mediante la línea azul y la derecha mediante la línea naranja.

A la vista de los resultados obtenidos en este experimento son de destacar los aspectos que se señalan a continuación:

a. En primer lugar, queda patente que los dos motores no tienen el mismo comportamiento. Mientras que el de la rueda izquierda empieza a rodar con tan solo 50 de entrada, no es hasta 80 cuando se empieza a mover la rueda derecha.

b. También se puede observar que no tienen un comportamiento similar hasta que no llega a 200, aproximadamente. Esto indica que desde la entrada de 50 hasta la de 200, el motor de la rueda izquierda se mueve más rápido.
c. Por último, se puede ver cómo para valores de entrada por encima del máximo (entre 260 y 300), la cantidad de marcas/segundo de ambas ruedas permanece prácticamente constante. Esto es debido a que el motor satura y aunque el usuario le pida 270 de entrada, este solamente es capaz de entregar 255.

B.1.2 Comportamiento normal (en condiciones de uso)

En esta segunda parte se probarán todas las entradas estando el robot en posición normal de funcionamiento, es decir, con las ruedas en contacto con el suelo. Las dos principales consecuencias de este hecho son:

a. La primera de ellas, y más importante, es que ahora habrá más impedimentos al movimiento de nuestro robot ya que, a la resistencia generada por el aire y por los engranajes internos, ahora se debe sumar la resistencia provocada por el rozamiento con el suelo; también el propio peso de la estructura que hará que sea más difícil mover la misma. Esto implicará que las lecturas serán más lentas que las anteriores (el número de marcas será menor), como puede verse en la figura B.2.

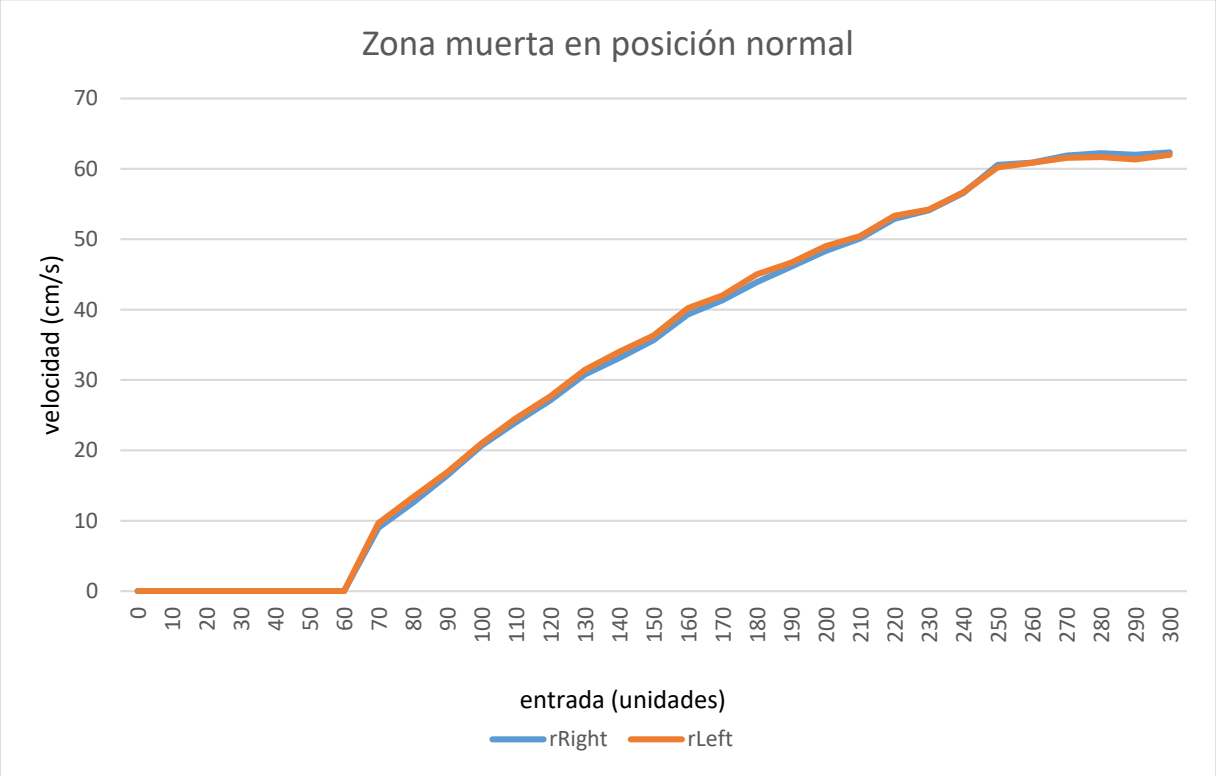

Figura B.2. Resultado del experimento realizado en al zona muerta en la posición de normal funcionamiento donde la rueda derecha se representa mediante la línea azul y la izquierda mediante la línea naranja.

b. Por otra parte, al estar todo el conjunto en contacto con el suelo, el movimiento de una rueda ejercerá fuerza sobre todo el bloque y empujará a la otra rueda a girar. Ahora la rueda derecha empieza a moverse a una entrada menor que anteriormente debido a que a esa entrada se mueve primero la rueda izquierda, y como está en contacto con el suelo arrastra a la otra.

Puede verse que prácticamente para cada entrada las lecturas de ambas ruedas son las mismas debido a que hay más impedimentos al movimiento y las diferencias se atenúan. Para reflejar con mayor detalle la diferencia, lo que se ha hecho en este caso ha sido coger el número total de marcas que se han capturado durante los 9 segundos que ha durado cada experimento. Luego se ha llevado a una tabla y se han hecho las operaciones pertinentes para dar el resultado, en centímetros/segundo. La ventaja de esta manera de actuar ha sido que mientras que en el caso anterior después de cada ejecución se sacaba su velocidad y luego se hacía la media de las 3 velocidades calculadas, aquí se coge la distancia total recorrida y luego se divide entre 9, de manera que solo se hace un cálculo, y no 3.

Se puede ver como hasta 230 aproximadamente, la velocidad de la rueda izquierda es un poco superior a la de la derecha, pero con diferencias mínimas; estas diferencias prácticamente desaparecen en los valores cercanos al máximo, llegando a sobrepasar la rueda derecha en algún momento cuando el robot ya satura.

B.1.3 Conclusión

A la vista de ambos resultados ya se está en posición de contestar a las preguntas antes formuladas y a que el lector saque sus propias conclusiones.

En cuanto a que si ambos motores eran iguales ha quedado demostrado que esto no ocurre y que ante una misma entrada no funcionan igual ni siquiera en el caso normal de funcionamiento, donde las diferencias entre ambos motores se reducían bastante.

Atendiendo a las gráficas tampoco tienen el mismo comportamiento, como era de esperar, en vacío y en condiciones normales. Sí que se puede decir que llegados a valores cercanos al máximo el comportamiento de ambos motores se convierte en un comportamiento muy parecido. En esta zona las velocidades son prácticamente las mismas independientemente de si se está en vacío o no. Para esta zona sí que podría decirse que los motores se comportan de una manera relativamente similar, comportamiento que no se manifiesta en ningún intervalo de velocidades menor.

Viendo los resultados, y en vistas a modelar el comportamiento de los motores se limitará la entrada que se le introduzca a nuestro Pi Robot para conseguir un comportamiento más preciso. La acotación de la entrada se realizará entre 100 y 255, alejándonos de velocidades extremadamente lentas que pueden ocasionar algún fallo.

Como se verá mas adelante cuando se detalle el modelado de los motores la entrada mínima se establecerá en 100 unidades que dará una velocidad cercana a los 20 cm/s. Esto no quiere decir ni mucho menos que la velocidad mínima de nuestro robot sea esa. Esto quiere decir que la mínima velocidad con la que se puede garantizar un movimiento fluido, sin parones y sin anomalías es con esa entrada.

Como se detallará en capítulos siguientes el rango de velocidades con los que nuestro Pi Robot puede moverse en línea recta irá desde los 10 cm/s hasta prácticamente los 60 cm/s. Para moverse a 10 cm/s de manera ininterrumpida harán falta una serie de acciones especiales las cuales se detallarán a lo largo de este Trabajo Fin de Grado.

El hecho de que los motores no sean precisamente iguales hace aun más necesaria la existencia de un control de velocidad que posibilite que ambas ruedas puedan girar a velocidades iguales.

Apéndice C

Inicialización de la IMU

En este apéndice se explicará el proceso que se ha de seguir para inicializar la IMU antes de cada ejecución. Este proceso es fundamental para elegir un ángulo Z inicial válido, que nos permita comparar ese valor con los demás y hacer un giro lo más exacto posible de acuerdo a lo que pide el usuario.

Este procedimiento solo es usado en el control del giro debido a que es el único control que utiliza la IMU. En este programa original (programa muy parecido al disponible en [5]), una vez introducido el giro deseado por el usuario, y con el Robot aún parado se hacía una primera iteración de la función *giroIMU()* para obtener el primer valor de la variable *gyroZangle.*

Este valor se almacenaría en una variable y a partir de ese instante, se mandaría la entrada a las ruedas y el Robot empezaría a girar. Mientras girase, la función *giroIMU()* estaría continuamente sacando valores del ángulo Z actual del Robot. Este bucle duraría hasta que la diferencia entre el valor de *gyroZangle* menos el valor inicial fuera el ángulo pedido por el usuario en ±1 grado.

Cuando se probó se vio que nuestro Raspberry Pi Robot hacía giros que no se parecían en nada a la instrucción que se le había pasado por pantalla. Esto hizo que se investigara más a fondo las lecturas dadas por la IMU. Los resultados se esta prueba se muestran en la imagen C.1.

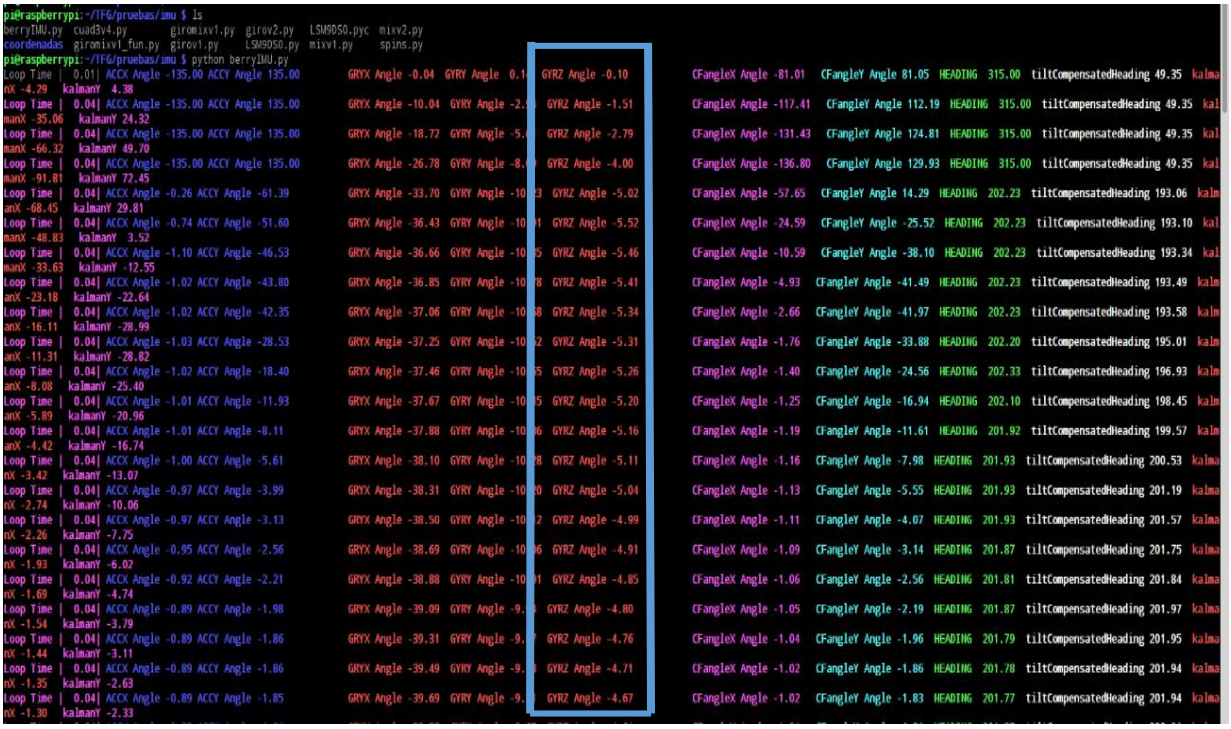

Figura C.1 Resultado mostrado por pantalla con todos los parámetros definidos y medidos por defecto del programa original berryIMU.py encontrado en el repositorio GIthub.

Como se puede observar en la imagen C.1, las primeras iteraciones de todas las variables arrojan datos que no tienen nada que ver con los siguientes, incluida la variable que nos interesa (GYRZ); y no es hasta la quinta iteración cuando los valores de los parámetros se actualizan y marcan las verdaderas magnitudes que están midiendo. En el caso de nuestra variable *GyroZAngle* este error es menor que el que se puede encontrar en otras variables como por ejemplo "*heading*". Aun así, estando el robot quieto, entre la primera iteración y la quinta el ángulo medido varía en 5 grados (de 0.10º a 5.02º).

Este hecho explica el problema comentado anteriormente donde el Robot era incapaz de realizar los giros pedidos por el usuario. Como medida para solventar este problema en vez de hacer una iteración antes de iniciar el giro, se deben hacer varias iteraciones de la función *giroIMU()* siendo este número superior al número de iteraciones que dan un resultado erróneo, en este caso mayor o igual a 5. Nosotros hemos elegido como valor 5.

Apéndice D

Función cuenta marcas

Como se explicó en el apartado 4.1.1 de la memoria, el imponer un periodo de muestreo tan sumamente pequeño traía consigo el problema de la poca precisión a la hora de obtener la velocidad actual de cualquiera de las ruedas.

El método inicial para hallar la velocidad era:

$$\text{Velocidad} = \frac{marcas\ leídas * 0.9895\ \frac{cm}{marca}}{T\ segundos}$$

Donde T es el periodo de muestreo (0,1 segundos antes de introducir la opción de mover el robot a "mínima velocidad"). Esto tenía muy poca precisión, ya que la diferencia de velocidad entre si se leía una marca o se leían dos era de:

- Velocidad leyendo 1 marca con T=0.1 segundos → 9.89 cm/s
- Velocidad leyendo 2 marcas con T=0.1 segundos→ 19.79 cm/s

Es decir, una variación de prácticamente 10 cm/s si se leía una marca o dos, por lo que este método de cálculo era muy impreciso y se hacía imperiosa la necesidad de busca un método alternativo de calcular la velocidad actual en las ruedas. La solución que se pensó se describirá a continuación ayudándose de un pequeño esquema (Figura D.1) y de la sintaxis en Python para una rueda.

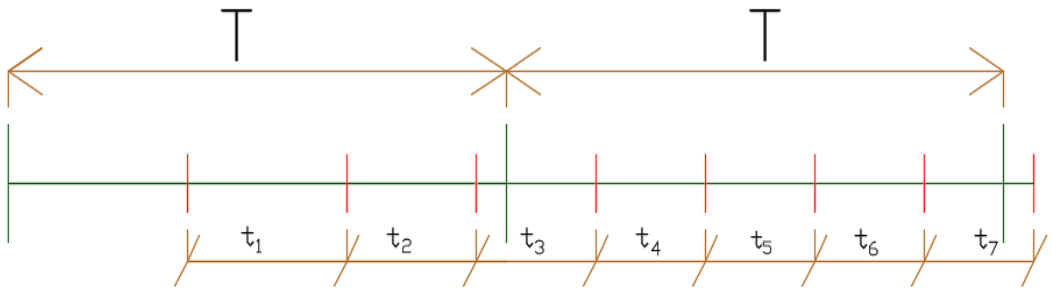

Figura D.1. Esquema teórico del planteamiento pensado para calcular la velocidad de manera más exacta. Donde las líneas verticales rojas indican los instantes donde se detecta marca, estos instantes no son iguales unos a otros como es obvio por lo que puede variar el número de marcas detectadas entre dos iteraciones consecutivas.

La idea consiste en que una vez leída una marca, se iniciaba un reloj que contaba el tiempo hasta que se detectara una nueva marca. De esta manera se obtenían t_1, t_2, t_3 … y haciendo la conversión necesaria se podía obtener la velocidad actual de la rueda en ese instante de tiempo.

Una vez acabado el tiempo de muestreo de la iteración "i", si el enconder había leído marca, el valor de t_i se sobrescribía. Para calcular la nueva velocidad se usaba el último valor de t_i . Así, si $t_i \neq t_{i-1}$ significaba que la rueda se había movido. En la memoria se explica el porqué de este planteamiento y no de hacer una media con las velocidades leídas durante T segundos.

A continuación se mostrará el código de la función creada para contar las marcas y los tiempos en una rueda, en este caso la función cuenta marcas de la rueda derecha. La función para la rueda izquierda es prácticamente igual. Se puede ver cómo esta implementación dista un poco de su original mostrada con anterioridad en el apéndice A, figura A.3.

```
def trRight():
        global contR
        global contTOTR
        global v_antR
        global stR
        global tiempoR

        pul21=GPIO.input(21)
        if pul21==True and v_antR==0:
                contR=contR + 1
                tiempoR=time.time()-stR
                stR=time.time()
                contTOTR=contTOTR + 1
                v_antR=pul21
        else:
                contR=contR
                v_antR=pul21
```

Figura D.2. Sintaxis Python de la función actual usada para contar marcas. En esta imagen se representa la función utilizada para la rueda derecha. La diferencia encontrada con la mostrada en la figura A.3 es que una vez ha detectado marca, se inicia un reloj que se detiene la siguiente vez que el bucle entre en el "if".

Las variables *tiempoR* y *stR* se inicializan al principio y empieza a ejecutarse la primera vez que el encoder lee una marca. A partir de ahí se mide siempre los segundos (milisegundos) que le cuesta al encoder leer dos marcas consecutivas. Cuando se sale del bucle, el valor recogido en la variable *tiempoR* corresponde con el último dato conocido y es con el que se trabajará durante esa iteración.

Apéndice E

Herramienta “Curve Fitting Tool”

Esta herramienta es la usada para el cálculo de la constante de tiempo τ de ambos motores. Este parámetro es necesario a la hora de calcular las funciones de transferencia en tiempo continuo. Información adicional sobre esta herramienta puede encontrarse en [20].

De acuerdo con lo mencionado en el apartado 3.2 de la memoria, una vez se tiene una serie de datos acerca del tiempo que tarda el motor en llegar al régimen permanente ante una entrada dada, es necesario calcular el tiempo de respuesta. Este tiempo de respuesta se define como el tiempo que tarda el motor en llegar a un margen del ±5% del valor pedido y ya no sale de ahí. Como la gráfica que tenemos en un principio muestra muchos picos resulta complicado trabajar con ella, motivo por el cual se recurre a esta herramienta del programa informático Matlab. A continuación se explicará qué hace esta herramienta y el procedimiento usado para el cálculo del tiempo de respuesta y en definitiva de la constante de tiempo τ.

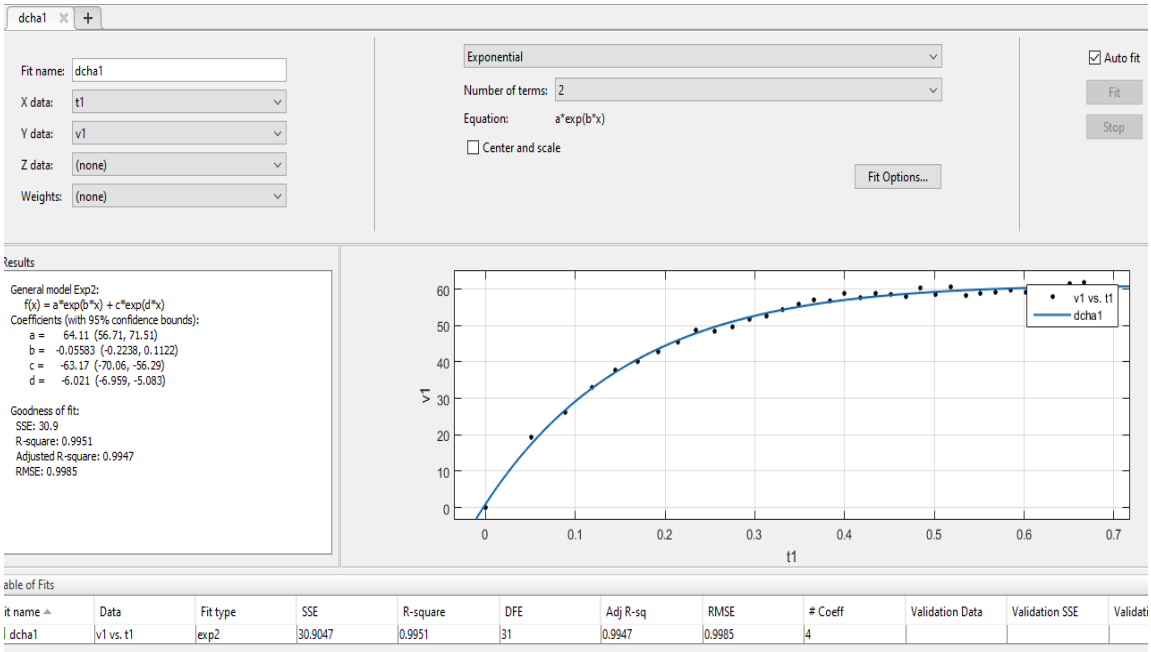

Figura E.1. Ventana principal de la aplicación "cftool" de Matlab

Introducimos en la ventana de Matlab los mismos datos obtenidos del ensayo real. Es decir, por un lado se meterá el vector de tiempos (t1) y por otro lado el vector de velocidades (v1). Estos vectores se asignarán a una variable de manera que Matlab pueda trabajar con ellos. Una vez realizado esto se llama a la herramienta mediante la siguiente instrucción:

```
>> cftool
```

Una vez introducidos los vectores y estando en la herramienta, se define qué vector va al eje de abscisas (tiempo en segundos) y cual al eje de ordenadas (velocidad en cm/s), y aparecen los mismos puntos que aparecían en la tabla. A continuación la herramienta permite ajustar la gráfica por diversos métodos, en nuestro caso se hará un ajuste exponencial por ser el comportamiento esperado en un sistema de primer orden.

Todas estas instrucciones se muestran en la imagen anterior, figura E.1. A continuación el programa pide al usuario el número de términos para ajustar. Para este caso se elige el ajuste mediante dos términos de modo que la ecuación que modelará la gráfica tendrá la forma siguiente.

$$velocidad\ = a * e^{b*X} + c *\ e^{d*X}$$

Una vez hecho esto el programa calcula la nueva curva y los coeficientes de la ecuación quedando la gráfica como se muestra en la imagen E.2.

Como se comenta a lo largo de la descripción del controlador DeadBeat, el uso de este control requiere un muy buen modelado de la planta, motivo por el cual esta operación se hará cinco veces para cada motor. Estas cinco ejecuciones darán cinco tiempos de respuesta distintos, pero todos ellos muy próximos entre sí dado que el tiempo que tarda el robot en llegar a la velocidad objetivo es siempre similar. Acto seguido, mediante una media de los cinco valores se obtendrá el tiempo de respuesta medio y en consecuencia la constante de tiempo τ.

Recordemos que ambas variables se relacionan mediante al siguiente expresión en los sistemas de primer orden:

$$\tau = \frac{t_{respuesta}}{3}$$

En esta gráfica se pueden observar los puntos negros que corresponden con los datos aportados por el usuario sobre una ejecución del programa. Puede observarse como al inicio, la distancia horizontal es mucho mayor que al final dado que al inicio el robot lleva una velocidad mucho mas lenta y por tanto el tiempo entre lecturas es mucho mas grande.

Una vez se tiene una gráfica como la mostrada en la imagen E.2 se amplía la parte asemejable a una recta horizontal (régimen permanente) final para ver a qué valor tiende. En este caso como se puede apreciar, el valor es cercano a 60 cm/s llegando a superarlo mínimamente. Una vez obtenido ese valor exacto al que tiende la velocidad, se calcula el valor que corresponde a un -5% para obtener el tiempo de respuesta.

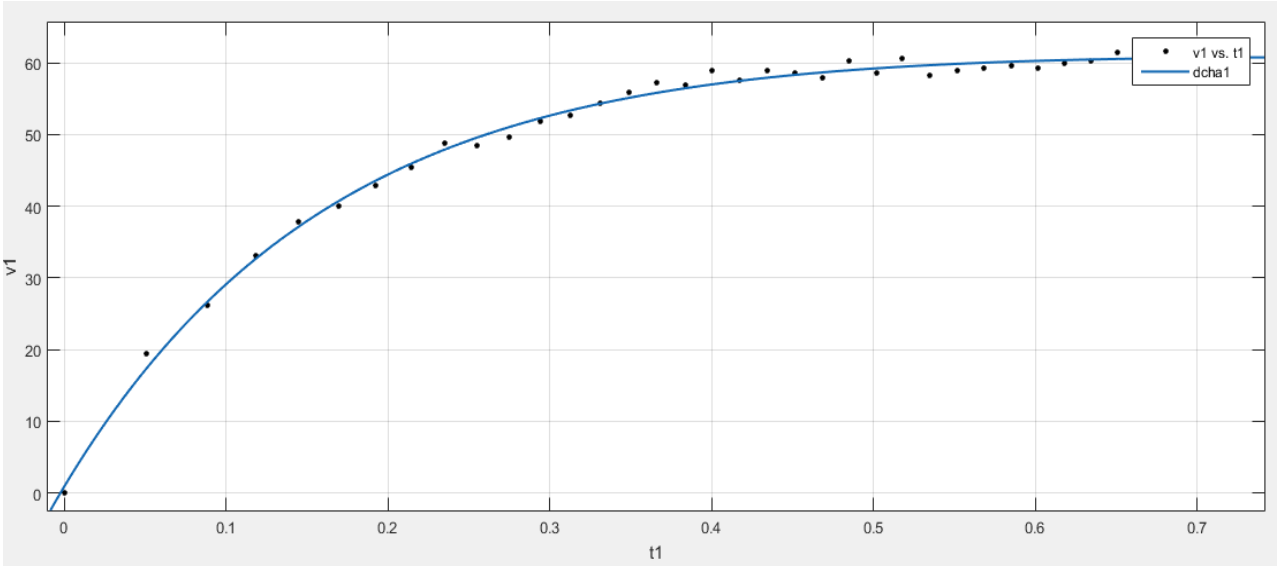

Figura E.2. Gráfica corregida mediante la herramienta Curve Fitting Tool. Los puntos negros indican los puntos exactos atribuibles a los datos recogidos del ensayo y mediante aproximación exponencial se obtiene la gráfica de la línea azul, mucho más detallada.

Una vez conocido este valor, se busca en la gráfica para ver a qué instante de tiempo corresponde y se anota. Esta ejecución se realiza 5 veces con el objetivo de tener un valor de la constante de tiempo contrastado y eficaz y que nos permita un buen modelado de nuestras funciones de transferencia en el dominio de "s".

Apéndice F

Desviación en la línea recta

Una vez se tiene un control para la ejecución de líneas rectas fiables y precisas y que va a la velocidad pedida por el usuario, lo único que queda por comprobar para dar por válido esta implementación es la posible desviación que pueda tener dicha línea recta. Esta será la última comprobación y se estudiará dado que con solo ver que el control de velocidad funciona correctamente no se puede saber si la trayectoria es correcta. Esto es así dado que en el arranque, punto crítico, el Robot puede desviarse mínimamente de la ejecución y un grado, en una ejecución de 2 metros conduciría a un error mucho mayor al final.

Este análisis estudiará la desviación que sufre el robot entre el punto inicial y al punto final medido respecto a la posición de la rueda loca. Para ello, al inicio de la ejecución se colocará la rueda loca en el centro de un listón de madera (no se colocará entre dos listones de madera para que esta juntura no tuviera incidencia sobre la ejecución intentando que la rueda loca siguiente dentro de ella). Al finalizar la ejecución se medirá la distancia existente entre el punto final de la rueda loca y el punto de referencia. Los datos del estudio se muestran en la figura F.1.

Exactitud en la rectitud en una trayectoria recta			
100	Distancia en (cm) que		
(cm)	se desvia a los lados	error (%)	error total(%)
200	2,5	1,25	1,72
	2,3	1,15	
	5,5	2,75	
130	Distancia en (cm) que		
(cm)	se desvia a los lados	error (%)	error total(%)
200	5,1	2,55	2,27
	0	0	
	8,5	4,25	
160	Distancia en (cm) que		
(cm)	se desvia a los lados	error (%)	error total(%)
200	5	2,5	1,60
	1,4	0,7	
	3,2	1,6	
190	Distancia en (cm) que		
(cm)	se desvia a los lados	error (%)	error total(%)
200	2,6	1,3	1,10
	3,2	1,6	
	0.8	0.4	

Figura F.1. Tabla que refleja los datos recogidos sobre la desviación en la línea recta. En ella se puede ver que se evalúan 4 velocidades y solo una distancia (2 metros). Cada prueba se realiza 3 veces, midiendo el desvío sufrido por el robot al final de la ejecución. Por último se calcula el error en porcentaje.

En este estudio se evaluará únicamente una distancia, 2 metros la cual se considera ya una distancia importante y se estudiará a diferentes entradas como pasaba con la exactitud de la condición de parada de este mismo programa. Las cuatro entradas analizadas serán 100, 130, 160 y 190. Para cada entrada se realizará la prueba 3 veces recogiendo los 3 valores de manera que nos quede un resultado más preciso del que quedaría si solo se contemplara un dato.

Cómo se puede observar en la tabla adjunta la desviación sufrida es independiente de la velocidad y es mínima llegando en algunos casos a ser 0. Esto da como conclusión que se cuenta con un programa capaz de realizar trayectorias rectas con una exactitud remarcable, con un control fiable, robusto y con todos los supuestos casos críticos estudiados y solventados.

Apéndice G

Ortogonalización de Gram-Schmidt

En este apéndice se desarrollan los cálculos para obtener el ángulo que debe girar el robot para orientarse a una posición del entorno definida por el usuario. A la hora de implementar una función que recoja los datos de la ejecución del programa *"coord_newconcept_5a.py"* tiene que ser consciente que se pueden dar muchos supuestos en función de donde quiera el usuario que vaya nuestro Robot.

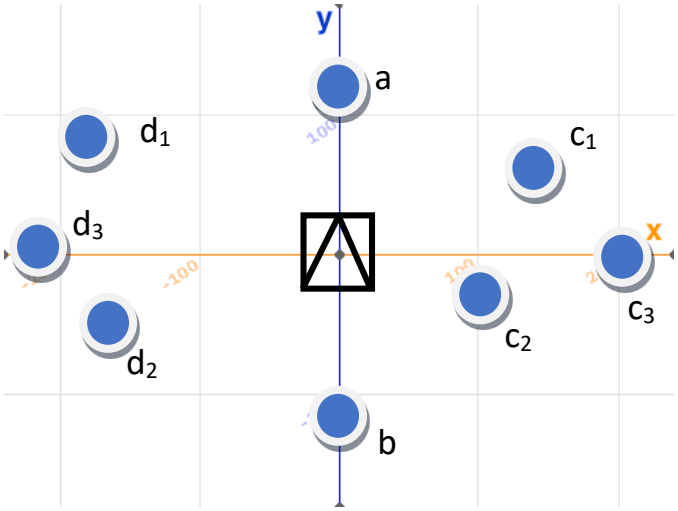

Figura G.1. Mapa bidimensional con el Pi Robot en el centro y posibles puntos alcanzables por él.

Todos estos supuestos se recogen en la imagen G.1 y se representan por puntos. Estos son:

a) Línea recta sin ángulo
b) Línea recta con ángulo de 180º
c) Punto situado en 1º o 3º cuadrante
d) Punto situado en 2º o 4º cuadrante

Tener todas estas situaciones en cuenta implicaría la necesidad de usar gran cantidad de clausulas condicionales que harían un programa mucho más largo y menos elegante a la hora de presentar.

Como solución se pensó en usar la ortogonalización de Gram-Schmidt (toda la información relativa a este metido puede encontrarse en [21]) , el cual es un método compacto y contrastado que ofrece grandes resultados. Se basa en que el usuario manda al programa una coordenada alcanzable por el robot y el método calcula qué ángulo es el necesario especificando además el sentido.

Este método se basa en la utilización de la función arcotangente (atan2) la cual convierte las coordenadas rectangulares (x,y) en coordenadas polares (r, θ).

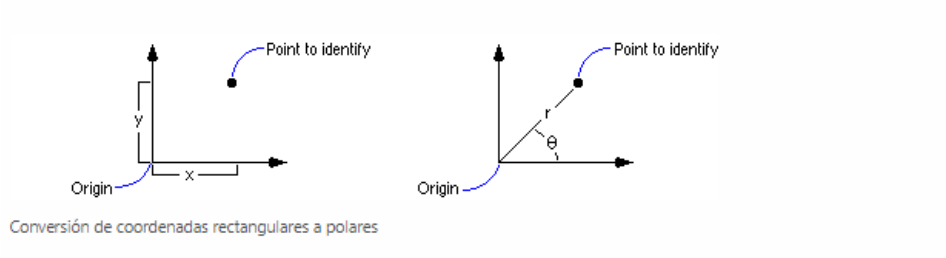

Figura G.2. Conversión del sistema de unidades rectangulares a polares

El algoritmo que habrá que implementar para poder hacer esta operación será:

Definir coordenadas de destino

Con estos datos se define el vector $u = \begin{pmatrix} X \\ Y \\ 0 \end{pmatrix}$

Se convierte este nuevo vector en vector unitario $u_1 = \begin{pmatrix} X_1 \\ Y_1 \\ 0 \end{pmatrix}$

Definimos vector $v = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

Para obtener el primer argumento que se le pasa a la función "atan2" se debe hacer el producto vectorial de los vectores u_1 y v y el argumento será el único valor no nulo del resultado: $u_1 \; X \; v = \begin{pmatrix} 0 \\ 0 \\ \alpha_t \end{pmatrix}$

Para obtener el segundo término, se hará el producto escalar de estos dos mismos vectores: $u_1^T * v = (X_1 \quad Y_1 \quad 0) * \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

El resultado se dará en radianes respecto a la vertical por lo que tendrá que hacerse a conversión a grados para poder pasárselo al control de la IMU.

Como se ha comentado, para llevar a cabo dicha tarea se necesita la función *"atan2"* la cual se encuentra en una librería externa de Python por lo que será necesario importarla escribiendo al inicio del programa "import math".

A continuación se mostrará una imagen de la función *"inicio()"* del programa *"coord_newconcept_5a.py":*

```
def inicio():
    global ang_objetivo
    global cm
    global marcas
    global v_objetivo
    global T

        ######################################################################
        ######################### CALCULO GIRO ###############################
    coord_X= int(input('Determine la coordenada X del destino final? (negativo--> lado izq): '))
    coord_Y= int(input('Determine la coordenada Y del destino final? (negativo--> atras): '))
    if coord_X==0 and coord_Y==0:
        print "DATOS INVALIDOS"
        coord_X= int(input('Determine la coordenada X del destino final? (negativo--> lado izq): '))
        coord_Y= int(input('Determine la coordenada Y del destino final? (negativo--> atras): '))

    coord_Z= 0
    print('La coordenada Z del destino final es 0')
    unit= math.sqrt((coord_X**2)+(coord_Y**2)+(coord_Z**2))
    u1= coord_X/unit
    u2= coord_Y/unit
    u3= 0

    v1= 0
    v2= 1
    v3= 0

    alfa_t= u1*v2 - u2*v1
    alfa_2= u1*v1 + u2*v2 + u3*v3

    angulo_giro= math.atan2(alfa_t,alfa_2)
    conver=180/(math.pi)
    ang_objetivo= angulo_giro*conver

    v_objetivo= int(input('a que velocidad? (maxima 61 cm/s)): '))

    print('\n')
    print('RESUMEN: ')
    if ang_objetivo>0:
        print ("el angulo que tendra que girar sera de ", ang_objetivo, " grados a la derecha")
    elif ang_objetivo<0:
        print ("el angulo que tendra que girar sera de ", abs(ang_objetivo), " grados a la izquierda")
    else:
        print ("el robot no realizara giro alguno")


    #Variacion del tiempo de muestreo T en funcion de la velocidad que vaya a llevar nuestro Robot
    if v_objetivo>=25:
        T=0.1
        print "El periodo de muestreo T sera de 0.1 segundos"
    else:
        T=0.3
        print "El periodo de muestreo T sera de 0.3 segundos"

    marcas=round(int(unit/cm))
    print ("Distancia a recorrer: ", unit , ' cm  (',marcas,' marcas )')
```

Figura G.3. Sintaxis Python de la función inicio() del programa coord_newconcept_5a.py donde se implementa el método de la ortogonalización de Gram-Schmidt en código Python.

Apéndice H

Limitaciones de los dispositivos de medida

En este capítulo se enumerarán los contratiempos con los que se ha tenido que lidiar referentes a los aparatos de medida. Estos aparatos rara vez presentan problemas en la lectura, malos funcionamientos, etc, pero sí que alguno de los problemas explicados aquí se ha repetido en el tiempo. Como normal general, la parte donde más problemas ha habido o la más costosa de implementar ha sido el control llevado a cabo por la IMU. Por su parte, en el encoder solo se ha encontrado un problema muy poco frecuente, el cual por otra parte, ha resultado tener difícil solución. En este anexo se explicarán las diferentes fases por las que se ha pasado y los diferentes problemas surgidos dando como solución la alternativa implementada actualmente.

H.1 Problemas IMU

Se empezará a hablar de la unidad de medición inercial, la cual como se ha comentado durante parte de la memoria tenía varios inconvenientes juntos lo que hacía muy difícil su implementación y su aplicación a trayectorias precisas, que era el objetivo final. A continuación se detallarán los dos casos más llamativos.

H.1.1 Translación pura

Lo primero que se intentó implementar fue una translación pura(una rueda describía un movimiento hacia adelante, mientras la otra, hacía ese mismo movimiento pero en dirección contraria), es decir un programa donde se girara usando las dos ruedas al mismo tiempo. El tipo de control podía seguir siendo el control deadbeat debido a los buenos resultados obtenidos de su uso para la trayectoria recta. Para ello se debería modelar de nuevo los motores debido a que la velocidad sería otra (ya no era cm/s, sino grados o radianes/segundo).

Se intentó implementar de manera análoga a como se había hecho para la línea recta es decir, calcular las nuevas ganancias, los nuevos tiempos de respuesta, etc. para conseguir una nueva expresión $G(s)_{giro}$. Mientras que en la línea recta se tenía un control para cada rueda y cada rueda era independiente, aquí el error (grados hasta destino) era común, de manera que la acción que se iba a mandar sobre las ruedas se calculaba únicamente con un dato.

Seguidamente se vio que un control de velocidad de giro no tenía mucho sentido aquí dado que las distancias a cubrir eran minúsculas y el control iba a tardar más tiempo en llegar al objetivo de velocidad que en llegar a la posición deseada. En cuanto a la condición de parada se tenía que mientras no se llegara al objetivo, el programa siguiera actuando, otra opción que se barajo fue la posibilidad de contar un giro como un número de marcas, por lo menos en el giro de 90 grados. De esta manera se sabía que se conseguía un ángulo de 90 grados cuando la rueda exterior giraba 16-17 marcas (o cuando una leía 8 marcas y la otra otras 8, en direcciones opuestas).

Esta condición de parada se desechó dado que entre periodos de muestreo se podía perder la referencia de modo que el robot podía pasarse un montón de marcas (grados), siendo muy difícil que convergiera al resultado que se buscaba. Esto dejaba como única solución que la condición de parada dependiera únicamente de la IMU, como así finalmente ha sido.

En cuanto a la decisión de por qué no implementar un control de giro usando translación pura la respuesta es muy sencilla: El poco margen que hay para las velocidades. Esto se explicará detalladamente a continuación. Este problema es el mayor que se tiene actualmente a la hora de desarrollar el control de la IMU, pero en una translación pura se manifestaba por partida doble.

Se daba la circunstancia de que a lo mejor con batería baja, a entradas (entrada para la rueda derecha = 80 uds., entrada para la rueda izquierda = -80 uds.), nuestro Robot no tenía fuerza suficiente para conseguir mover el Robot, pero a (rueda derecha = 100 uds., rueda izquierda = -100 uds.), el movimiento del robot era fluido y giraba a muchos grados/segundo. Esto hacía que fuera prácticamente imposible jugar con las entradas dado que el margen eran 20-30 unidades, con entradas muy pequeñas el robot no se desplazaba, con entradas 30 unidades más altas, se movían dos ruedas a la vez (mayor velocidad aun) y era muy complicado hacer mediciones.

Conforme más rápido se movía, menor tiempo de muestreo se requería dado que si no, el error podía crecer mucho entre una iteración y otra, por lo que el periodo de muestreo tenía que ser de unos pocos milisegundos, nada que ver con los 100 milisegundos (1 décima) que se tenía en la línea recta. Otro inconveniente muy importante, es que la IMU ante variaciones muy rápidas de su posición reacciona mal, dando medidas imprecisas.

Todas estas circunstancias hacían que un giro moviéndose al unísono ambas ruedas resultara imposible con el material con el que se disponía. Esto hizo que se cogiera la otra opción, la realización del giro usando solamente una rueda, la rueda exterior, que se movería hacia adelante mientras la rueda interior no recibiría tensión durante la ejecución.

H.1.2 Giro actual

Con este tipo de ejecución, el Robot giraba de manera más lenta debido a que solamente se desplazaba una rueda, por lo que el abanico de posibles entradas aumentaba algo.

También como consecuencia trajo que no se pensara ya en la implementación del control deadbeat, sino que ahora la idea fue intentar implementar un control proporcional. Este tipo de control como se sabrá es el más sencillo de todos. Consiste en estudiar el error y dar una acción que sea proporcional a dicho error, esto hacía que se tuviera que jugar con la ganancia.

Esto resultó más difícil de lo pensado debido a las limitaciones con las que se contaba. El problema más importante era a la hora de aproximarse al destino, una vez el error disminuía, se requería un movimiento lento pero continuo. Con el control proporcional el robot se paraba dado que errores pequeños generaban entradas pequeñas con las que el Robot era incapaz de actuar.

Si por el contrario se implementaban ganancias altas, capaces de dar respuesta en estos casos donde se necesitaba exactitud y respuesta por parte del robot, el usuario se encontraba de lleno con otro problema y no era otro que el Robot se movía en exceso al llegar a posiciones cercanas al objetivo.

Esto hacía que estuviera mucho rato oscilando (si el sensor detectaba que el robot se había pasado del objetivo, la acción compensatoria hacia mover la rueda interior hacia adelante, dejando la rueda exterior quieta), y se perdiera exactitud. Otro problema añadido era que si se jugaba con ganancias altas, en los primeros instantes donde el error es el máximo, se conseguían entradas altísimas, capaces incluso de saturar, por lo tanto volvía a aparecer el problema antes visto de la altísima velocidad presentada por el Robot con la que hacía imposible un control sobre la ejecución. Esto hacía que fuera cual fuera la acción, la entrada máxima debía ser limitada, llegando a falsear un poco el control dado que durante las primeras iteraciones daba igual que ganancia se hubiera implementado dado que lo más seguro es que pasara este límite y entrara en escena el limitador. Por lo que si en la translación pura se tenía como límite de entrada alrededor de 100 unidades, aquí, el límite era 120 aproximadamente.

Estos problemas hacían que entre las dos opciones se escogiera la primera pero con algún ligero cambio. El cambio llegaba obviamente en las posiciones de error pequeño. Ahora, en estas circunstancias entraría un integrador, que en el caso de detectar que el robot no se movía, añadiría un sumando al cálculo de la acción, de manera que la acción crecería iteración a iteración hasta dar un valor tal que hiciera que el Robot se moviera.

De esta manera se conseguía un movimiento suave producido por una ganancia pequeña durante los primeros dos tercios de la ejecución, y un movimiento a tirones en la última fase de la misma. El problema que traía consigo esta implementación era que llegado el punto, al haber tantas iteraciones en las que intervenía el integrador, al final, el sumando que dependía del integrador, se hacía muy grande, dando entradas relativamente altas cuando se estaba a escasos grados de alcanzar el objetivo, por lo que esta, aunque buena, no podía ser tampoco la solución definitiva para nuestro control.

Como solución se echó la vista a cómo se comportaban los robots (brazos robóticos entre otros) en la industria. Aquí, los robots tienen escrita una trayectoria la cual tienen que repetir indefinidamente, pero su comportamiento varía entre si está, por ejemplo, cerca del objeto que desea coger, o en pleno transporte de dicho objeto hasta su nueva ubicación. Mientras que el movimiento de transporte es rápido, eficiente y repetitivo, el movimiento de aproximación se hace a una velocidad mucho menor que el otro. En este movimiento se necesita mucha precisión, por lo que el Robot va lento intentando no dañar nada y por si hubiera cualquier inconveniente, no agrandar mucho el error. Esto se pensó para nuestro caso, implementar un movimiento de aproximación a velocidad constante cuando se estuviera relativamente cerca del ángulo deseado para que no se dieran casos de acción puntuales muy grandes, acciones tan pequeñas que provocaban que el Robot no se moviera, etc.

Por último, este "movimiento de aproximación" entraba cuando se estaba a menos de 20 grados del objetivo, por lo que se jugó con ello para darle valor a la ganancia. De manera que entre un error de 21 grados (ahí intervenía la ganancia), la respuesta dada por el programa fuera similar a la que se encontraría en la siguiente iteración, donde entraría ya el movimiento de aproximación.

H.2. Problemas Encoder

Una vez detallados todos los problemas acaecidos con la IMU se pasará a hablar en este apartado de los problemas encontrados con el encoder.

A decir verdad solo se ha detectado un problema en la medición con el enconder. Este problema, (como se puede ver en la figura H.1) resultaba que en un instante determinado, hacía una lectura errónea de los resultados, mandando acciones desproporcionadas al motor correspondiente y arruinando la ejecución.

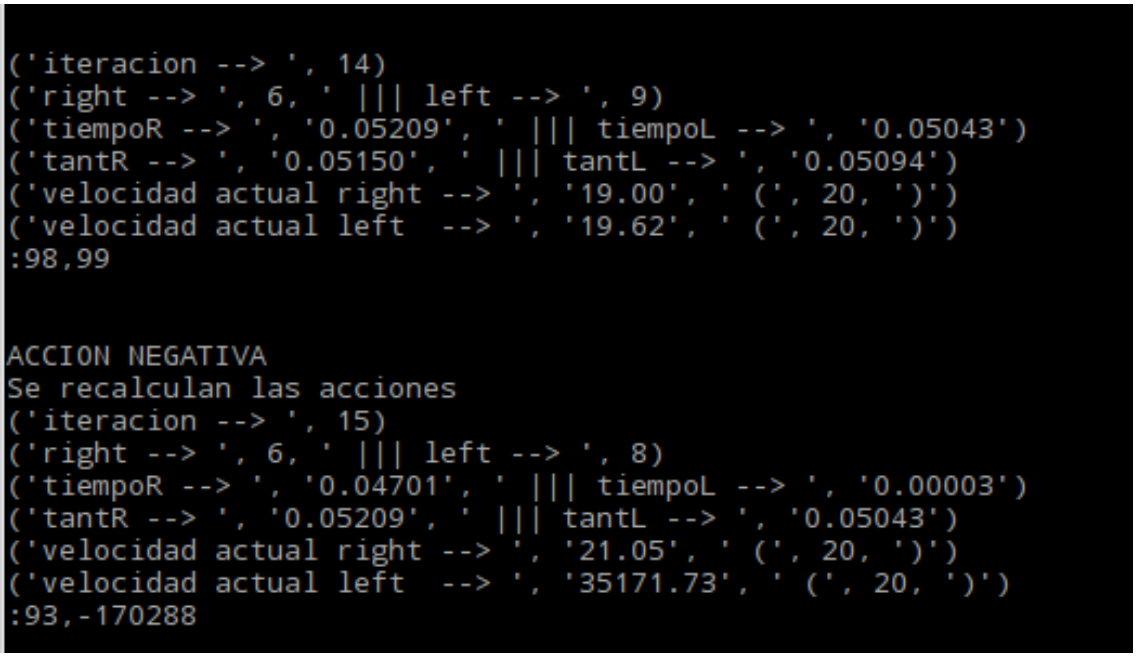

Figura H.1. Captura de pantalla en una ejecución en línea recta donde se puede apreciar un error en la medida en el sensor de la rueda izquierda.

Cómo se puede apreciar, en esta ejecución en concreto se está realizando una línea recta de una longitud cualesquiera en la que se ha pedido al robot que se desplace a una velocidad de 20 cm/s. En la iteración número 14 se observa que el sistema funciona correctamente, los tiempos medidos en cada rueda son parejos, las velocidades están cerca del objetivo y las acciones están próximas las unas a las otras (algo lógico dado que se trata de una trayectoria recta). Sin embargo en la iteración número 15 el sensor tiene un error en la medida.

Algo afecta a su lectura, siendo el tiempo entre marcas mucho más bajo que lo que cabría esperar. Este tiempo tan sumamente pequeño da por consiguiente una velocidad extremadamente grande. En consecuencia, el control entiende que la rueda va muy por encima de las exigencias pedidas por el usuario y le pide reducir en consecuencia (es decir, mucho en este caso) su acción, por lo que la acción ahora es negativa y se pierde todo el control sobre la trayectoria.

La causa del fallo puede deberse a la suciedad que pueda tener el disco medidor. En ese preciso instante ha podido coger una mota de polvo o alguna partícula que haya ocasionado una lectura errónea por parte del encoder.

También se ha podido deber a algún problema de conexiones. Algún cable puede no haber transmitido bien su señal eléctrica y de ahí que venga la lectura errónea. Lo cierto es que las veces que ha ocurrido este fallo se ha revisado en detenimiento las conexiones estando todas en perfecto orden por lo que no se cree que pueda ser esta la causa.

Sea como fuere, este problema, aunque muy causal, ha aparecido en distintas ocasiones durante el tiempo que ha durado este TFG presentando otro problema añadido a la ya de por sí difícil implementación del conjunto Pi Robot.

Apéndice I

Programas desarrollados

En el siguiente apéndice se mostrarán una serie de programas en Python para que el lector pueda ver las metodologías usadas y las soluciones propuestas a los diferentes sucesos que se pueden llegar a dar.

Los programas que tratan sobre trayectorias rectas y curvas se dividirán en diferentes partes para un mayor entendimiento por parte del lector. De estos programas se explicarán estas partes diferenciadas menos la parte correspondiente a control dado que esa parte tiene un capítulo propio en la memoria de este trabajo fin de grado (capítulo 4).

Los primeros programas que se mostraran serán los programas donde se usa el enconder, es decir, aquellos programas donde el usuario definirá una trayectoria, ya sea recta o circular, y nuestro Pi Robot, deberá ser capaz de cumplir con tal cometido.

I.1 Trayectorias rectas

A continuación se mostrará la sintaxis correspondiente al programa que genera trayectorias rectas a partir de la información definida por el usuario. Mas concretamente se analizará el programa "*dead_dual_5a.py*".

El programa se mostrará a continuación y como se ha comentado anteriormente, para una mayor facilidad de comprensión se dividirá el programa en 3 partes:

i) La entrada de parámetros (Figura I.1)
ii) El control (explicado en el apartado 4.2.1 de la memoria)
iii) El cuerpo principal del programa

El programa pedirá al usuario cierta información sobre la recta que va a tener que realizar. Esta información comprenderá la distancia a recorrer expresada en centímetros y la velocidad, en cm/s, a la que queremos que se desplace nuestro robot móvil. A la vez que el programa pregunta la velocidad, se mostrará por pantalla el rango de velocidades alcanzables por el robot para que, en ningún caso, el usuario introduzca un valor que imposibilite la realización. Este rango irá entre los 10 y los 60 cm/s aproximadamente.

Este valor que frustre la ejecución puede ser bien porque se le pide una velocidad tan elevada que el robot es incapaz de alcanzarla con sus propios medios o por pedir una velocidad tan lenta que colapse el programa, que en el periodo de muestreo dado no lea marcas y de errores como los mencionados en otros apartados de esta memoria.

```
def inicio():
    global v_objetivo
    global marcas
    global cm
    global T

    distancia= int(input('que distancia se quiere recorrer? (en cm): '))
    v_objetivo= int(input('a que velocidad? (min 10cm/s ||||max 62 cm/s)): '))

    #Variacion del tiempo de muestreo T en funcion de la velocidad que vaya a llevar nuestro Robot
    if v_objetivo>=25:
        T=0.1
        print "El periodo de muestreo T sera de 0.1 segundos"
    else:
        T=0.3
        print "El periodo de muestreo T sera de 0.3 segundos"


    marcas=round(int(distancia/cm))
    print(marcas, ' marcas')
```

Figura I.1. Sintaxis Python de la función inicio() del programa dead_dual_v8.py donde se puede apreciar por ejemplo la elección de uno u otro tiempo de muestreo según la velocidad pedida.

Según la velocidad suministrada por el usuario el programa adoptará un periodo de muestreo u otro. A lo largo del programa y para velocidades "normales" de funcionamiento como siempre se ha dicho, se tendrá el periodo de muestreo de T=0.1 segundos. Para posibilitar la realización de trayectorias a velocidades inferiores se eligió este periodo de muestreo T=0.3 segundos dado que al tratarse de trayectorias lentas, para un mismo periodo de tiempo va a entrar menos información por lo cual para conseguir velocidades menores habrá que aumentar T, aun a riesgo de conseguir un control más lento y donde puede haber más error ya que la corrección se hace en un tiempo el triple de lento.

Por último, se mostrará por pantalla la cantidad total de marcas que deberán recorrer ambas ruedas. En *dead_dual_5a.py* solo está implementado el contador total de marcas en la función cuenta-marcas de una rueda (rueda derecha, figura D.2, variable contTOT). Se procede de esta manera dado que en una trayectoria rectas ambas ruedas deben recorrer la misma distancia.

Parte III. Cuerpo principal del programa:

Por último toca hablar del cuerpo principal del programa el cual cuenta con una sola condición de parada. Una vez el usuario pide que se recorra una cierta distancia, ya se puede saber cuantas marcas van a tener que recorrer ambas ruedas y este dato se usará como condición de parada como se observa en la figura I.2.

La condición de parada establece que, mientras el contador total (que está en la función cuenta-marcas de la rueda derecha) de marcas no llegue al valor estipulado por el usuario, el programa de control se ejecutará ininterrumpidamente. Una vez haya recorrido tal distancia, se le mandará la instrucción de que se detenga (velocidad= (0,0)).

```
if prueba == 'Serial waiting.\n':
        print "Iniciado, leido " + prueba
        print "Vamos a enviar " + velocidad

        s.write(velocidad)
        eco = s.readline()

        if len(eco)!=0:
                print "RECIBIDO: " + eco
                if eco==velocidad:
                        print "Mover"
                else :
                        print "error!"
                k=0
                contTOT=0
                while (contTOT<marcas):
                    deadbeat()

                s.write(':0,0\n')
                contR=0
                contL=0
                while k<5:
                    k=k+1
                    stop()

                print(contTOT)

else:
        print "Error al iniciar"
        s.close()

#Escribimos en el .txt los resultados
f.write('Datos obtenidos:' + '\n')
f.write('velocidad objetivo: ' + str(v_objetivo) + '\n')
f.write('velocidades rRight: ' + str(vec_velR) + '\n')
f.write('acciones en rRight: ' + str(vec_uR) + '\n')
f.write('velocidades rLeft: '  + str(vec_velL) + '\n')
f.write('acciones en rLeft: '  + str(vec_uL) + '\n')
f.write('\n')
```

Figura I.2. Sintaxis Python del cuerpo principal del programa dead_dual_v8.py donde se puede apreciar la condición de parada entre otros detalles.

Una vez recorrida la distancia objetivo, el robot guardará los datos que ha ido recopilando durante la ejecución en un archivo .txt. Si se analizaran esos datos se vería que la última velocidad medida sería una velocidad cercana a la velocidad objetivo pero eso carece de sentido. La última velocidad leída debería ser 0. Para conseguir esto, una vez acabada la ejecución del programa, los contadores locales de ambas ruedas se inicializan y se ejecutan 5 iteraciones de una función similar a la *"deadbeat"* con la salvedad que la nueva función solo lee velocidades, no se asignan acciones. Así se asegura que la última velocidad medida es 0 cm/s ya que a lo largo de estas cinco iteraciones el robot irá frenando llegando a detenerse.

I.2 Trayectorias curvas

Una vez explicado a fondo el programa que permite elaborar trayectorias precisas en línea recta, es hora de hablar del programa que permite hacer círculos de radio cualesquiera con innumerables repeticiones. En concreto se hablará del programa "*dead_circle_5a.py".*

Se procederá de la misma manera, en este apartado se detallarán la función que entrega los datos al programa, y el cuerpo principal del programa dado que el control se explica en la memoria con exactitud y detalle.

Parte I, la entrada de parámetros:

A diferencia del programa anterior donde se introducían dos parámetros aquí se deben introducir cuatro. Estos describirán el comportamiento que tendrá el robot durante la ejecución de la trayectoria. La función *inicio()* se mostrará en la figura I.3.

Se empezará pidiendo el radio que describirá la rueda interior del robot, obteniendo con este dato la distancia que tendrán que recorrer ambas ruedas ya que se conoce la distancia entre ambas ruedas (variable *"dist_ruedas"*).

Después se pedirá al usuario que diga en cuánto tiempo quiere que se realice esta trayectoria. Para ello se habrá mostrado anteriormente los tiempos mínimo y máximo en los cuales el robot puede hacer la circunferencia. Las velocidades máximas y mínimas se obtendrán de los estudios realizados para el hallazgo de la K_{local_ruedas} descrito en el apartado de modelado de la memoria. Por lo que a ojos de este programa la velocidad mínima a la que puede girar el robot es 20 cm/s. Para evitar complicaciones solo se gestiona un tiempo de muestreo (T=0.1 segundos) dado que podría darse el caso de una rueda tener que girar a 30 cm/s y la otra tener que girar a 19 cm/s. A ojos del programa anterior, una rueda debería llevar un control con periodo de muestreo cada décima de segundo, mientras que la otra rueda iría con otra dinámica y requeriría otro tipo de implementación.

Acto seguido se deberá especificar el sentido de giro. Se introducirá un numero positivo cualquiera en el caso de querer realizar un giro en sentido horario, y negativo si por el contrario, se desea un giro que vaya en contra de las agujas del reloj.

Sabiendo la distancia que se quiere recorrer y el tiempo en que hay que hacerlo, el programa es capaz de obtener las velocidades que debe llevar cada rueda para llevar a buen puerto la ejecución del programa. Para evitar fallos, acciones desmesuradas al inicio, evitar problemas como la posibilidad de acciones negativas etc., como se explica en el apartado 4.3.2 se opta porque el robot empiece con velocidad. Esta velocidad será la teórica que le correspondería según el estudio hecho para el modelado de motores (capítulo 3 de la memoria). Para obtenerla, se interpola la velocidad que ha calculado el programa con las unidades y dará la entrada que llevará el robot en esta primera iteración. De esta manera se evitan arranques bruscos y prácticamente en línea recta, hecho que producía circunferencias con bastante error.

Por último, el programa pedirá que se especifiquen cuantas vueltas se quieren dar a la misma circunferencia, ya que se ha hecho un programa que dé la posibilidad de hacer tantas vueltas a una circunferencia como el usuario desee.

```
def inputs():
    global pi
    global v_objetivoR
    global v_objetivoL
    global vLeft
    global vRight
    global n
    global marcExt
    global sentido
    global cm
    global dist_ruedas
    global t
    global radioInt
    global vec_uR
    global vec_uL

    entradas=[100,110,120,130,140,150,160,170,180,190,200,210,220,230,240,250,255]
    vels_right=[19.625, 23.748, 26.881, 29.85, 33.478, 34.962, 38.425, 40.404, 42.384, 45.847, 47.496, 49.145, 51.454, 52.938, 55.412, 58.126, 59.37 ]
    vels_left =[20.285, 23.913, 27.706, 30.18, 33.643, 35.622, 38.261, 39.91,  42.054, 44.198, 46.506, 48.815, 50.794, 52.443, 54.093, 57.556, 59.535]
    radioInt= int(input('introduzca el radio interior de la circunferencia: '))

    marcInt=round(((2*pi*radioInt)/cm),1)
    marcExt=round(((2*pi*(radioInt+dist_ruedas))/cm),1)
    print ('marcasInt: ', marcInt,' ||||  marcasExt: ',marcExt)

    tmin= round((marcExt/59),2) #siendo 59cm/s la maxima velocidad, esta cuenta te dira el tiempo minimo en que se puede realizar el giro
    tmax= round((marcInt/20.285),2) #siendo 20cm/s la minima velocidad que puede llevar una rueda fijandonos en los registros de los vectores
    print('el minimo tiempo en que se puede realizar esta circunferencia es ',tmin)
    print('el maximo tiempo en que se puede realizar esta circunferencia es ',tmax)

    t= int(input('Introduzca el tiempo en que quiere realizar el circulo: '))
    sentido= int(input('cual sera el sentido de giro? (a derechas--> +)  '))

    if sentido>0:
        v_objetivoL =int(round((marcExt*cm/t),2))
        v_objetivoR =int(round((marcInt*cm/t),2))
        print "Las velocidades son: rLeft --> " + str(v_objetivoL) + " ||| rRight " + str(v_objetivoR)


    if sentido<0:
        v_objetivoL =int(round((marcInt*cm/t),2))
        v_objetivoR =int(round((marcExt*cm/t),2))
        print "Las velocidades son: rLeft --> " + str(v_objetivoL) + " ||| rRight " + str(v_objetivoR)

    x=0
    while vels_right[x]<v_objetivoR:
            x=x+1
            print ('x=', x)
    vRight= (((v_objetivoR - vels_right[x-1]) / (vels_right[x] - vels_right[x-1])) * (entradas[x] - entradas[x-1])) + entradas[x-1]
    vRight= int(round(vRight))
    print "La entrada que corresponderia a tal velocidad seria " + str(vRight) + " en la rueda derecha"

    y=0
    while vels_left [y]<v_objetivoL:
        y=y+1
        print ('y=', y)
    vLeft=  (((v_objetivoL - vels_left[y-1])  / (vels_left[y]  -  vels_left[y-1])) * (entradas[y] - entradas[y-1])) + entradas[y-1]
    vLeft=  int(round(vLeft))
    print "La entrada que corresponderia a tal velocidad seria " + str(vLeft)  + " en la rueda izquierda"

    vec_uR.append(vRight)
    vec_uL.append(vLeft)
    n=int(input('Vueltas que se quiere dar: '))
```

Figura I.3. Sintaxis python de la función inicio() en el programa dead_circle_v6.py donde se puede observar la gran cantidad de operaciones necesarias para llevar a buen puerto la ejecución.

Una vez mandados los datos, el programa buscará alcanzar y mantenerse en las velocidades objetivo en el mínimo tiempo posible , haciendo exactamente el mismo control que el explicado con anterioridad en las trayectorias en línea recta.

Parte III. Cuerpo principal del programa

Para este tipo de trayectoria, la condición de parada será como antes, que una rueda recorra la distancia pedida. En este caso, la rueda sobre la que se comprobará la condición de parada siempre será la exterior, esto indica que puede haber dos posibles escenarios, que el giro sea en el sentido de las agujas del reloj y por tanto, el contador usado sea el de la rueda izquierda o la situación inversa.

```
                stime=time.time()
                print "VELOCIDAD INICIAL"
                velocidad = ":" + str(vRight) + "," + str(vLeft) + "\n"
                s.write(velocidad)
                print(velocidad + '\n')
                contTOTL=0
                contTOTR=0
                if sentido>0:
                    while contTOTL<(n*marcExt):
                        deadcircle()

                if sentido<0:
                    while contTOTR<(n*marcExt):
                        deadcircle()


                s.write(':0,0\n')
                fin=time.time()-stime
                print "La ejecucion se ha hecho en " + str(round(fin,3)) + " segundos"

else:
        print "Error al iniciar"
        s.close()

#Escribimos en el .txt los resultados
f.write('Datos obtenidos:' + '\n')
f.write('velocidad objetivo right: ' + str(v_objetivoR) + '\n')
f.write('velocidades rRight: ' + str(vec_velR) + '\n')
f.write('acciones en rRight: ' + str(vec_uR) + '\n')
f.write('velocidad objetivo left: ' + str(v_objetivoL) + '\n')
f.write('velocidades rLeft: '  + str(vec_velL) + '\n')
f.write('acciones en rLeft: '  + str(vec_uL) + '\n')
f.write('\n')
```

Figura I.4. Sintaxis en Python del cuerpo principal del programa dead_circle_v6.py

Esto obliga a tener dos contadores de marcas totales, uno para cada rueda. La condición de parada como ocurría anteriormente será la misma, se ejecutará el programa de control mientras no se hayan alcanzado las marcas correspondientes.

La distancia se define como $n * marcExt$ siendo "n" el número de vueltas que queremos que dé el robot.

I.3 Programa coordenadas

En este apartado se ensañará parte del código Python referente al programa que, como se ha dicho durante buena parte de la memoria, engloba el objetivo final del Trabajo Fin de Grado. Este programa será capaz de guiar al Pi Robot hasta la coordenada predefinida por el usuario a través de sucesiones de movimientos giro + recta.

En este apartado solo se mostrará la información relativa al cuerpo principal del programa (figura I.5) dado que cómo se controla una trayectoria recta o una translación pura se ha visto con anterioridad tanto en la memoria como en apéndices (Entrada de parámetros y cuerpo principal del programa para trayectoria recta en apéndice I.1).

```
if prueba == 'Serial waiting.\n':
        print "Iniciado, leido " + prueba
        print "Vamos a enviar " + velocidad

        s.write(velocidad)
        eco = s.readline()

        if len(eco)!=0:
                print "RECIBIDO: " + eco
                if eco==velocidad:
                        print "Mover"
                else :
                        print "error!"
        if (ang_objetivo==0):
                print "solo realizara trayectoria recta"
                print ('\n')
                contTOT=0
                inigiro = 0
                fingiro = 0
                while (contTOT<marcas):
                    deadbeat()

                s.write(':0,0\n')
                print (contTOT)

        else:
                print "giro + trayectoria recta"
                print ('\n')
                        ##################################################
                        ################### GIRO #########################
                #TRAYECTORIA CIRCULAR
                cont=0
                a = datetime.datetime.now()
                while cont<6:
                    cont=cont+1
                    giroIMU()

                inigiro = abs(gyroZangle)
                inigiro2 = gyroZangle
                print "inigiro --> " + str(inigiro)
                time.sleep(0.5)

                while abs(inigiro + gyroZangle - ang_objetivo)>1:
                                spins()

                s.write(':0,0\n')
                fingiro = gyroZangle-inigiro2
                print ('ha girado ', round((fingiro),3), ' grados en la ejecucion')
                time.sleep(1.5)

                        ##################################################
                        ################### RECTA ########################
                contTOT=0
                while (contTOT<marcas):
                    deadbeat()

                s.write(':0,0\n')
                print (contTOT)
```

Figura I.5. Cuerpo principal del programa coord_newconcept_5a.py el cual es el programa que dada una coordenada alcanzable por el robot es capaz de conducirlo hasta esta nueva posición

Como se puede observar en la sintaxis el programa reconoce dos escenarios posibles:

1) la coordenada a alcanzar este justo en frente del robot, en ese caso no habría giro en la ejecución la cual se ejecutaría como una trayectoria recta
2) Si que haya que precisar de un giro para alcanzar el nuevo destino

En el caso de estar en la opción 2) primero se ejecutará el giro y acto seguido la trayectoria recta. El giro habrá sido calculado con anterioridad gracias al método de Gram-Schmidt (apéndice G). Antes de empezar el giro, es necesario saber la orientación exacta del Pi Robot, para ello se procede del modo explicado en el apéndice C.

Se hace una serie de iteraciones de la función giroIMU() estando el robot quieto y se obtiene la orientación actual. Sabiendo estos datos ya se puede empezar con la ejecución de la translación pura hasta el ángulo objetivo. Una vez se ha llegado, se imprime por pantalla el ángulo exacto que ha girado el dispositivo y se empieza con la trayectoria recta. Ésta no tiene diferencia alguna con la explicada en el apartado 5.1 de la memoria. Anteriormente se ha obtenido la distancia a recorrer y se ha definido la velocidad por lo que se tienen todos los datos para hacer una ejecución óptima.

Con este programa se podrán hacer (en varias ejecuciones) programas del tipo de realizar un cuadrado de lado L, ir hasta una posición y volver al inicio, etc.

I.4 Programas de modelado

Una vez explicados ciertos fragmentos de los programas que se han conseguido implementar durante la realización de este Trabajo Fin de Grado se mostrarán a continuación los programas en los que se basa todo lo mostrado anteriormente.

Estos programas son los referentes al modelado de los motores (memoria capítulo 3). Gracias a estos dos programas se han podido calcular las ganancias y las constantes de tiempo de los dos motores que forman el sistema motriz del Pi Robot.

Se empezará explicando el programa con el cual es posible obtener la ganancia en el dominio de “s” del motor. Este programa se llama “calculo_K_L.py” (figura I.6).

Acto seguido se hará lo propio con el otro programa. Este a través de estudiar la ganancia de velocidad en el arranque nos dará información del tiempo de respuesta del sistema y en definitiva de la constante de tiempo. Este programa se llamará “finversa_L.py”. (Figura I.7)

Ambos programas para obtener resultados contrastados harán varias repeticiones del mismo experimento buscando un resultado robusto y lo más parecido a la realidad posible.

```
import sys
import serial
import time
import RPi.GPIO as GPIO
import math
import numpy as np
import matplotlib.pyplot as plt


GPIO.setmode(GPIO.BCM)
GPIO.setup(20,GPIO.IN)

vLeft=100
vRight=100

#########################################################################
#################### declaracion de funciones ###########################
#########################################################################
def rLeft():
        global contL
        global v_antL

        pul20=GPIO.input(20)
        if pul20==True and v_antL==0:
                contL=contL + 1
                v_antL=pul20

        else:
                contL=contL
                v_antL=pul20

#########################################################################################
############################## FIN FUNCIONES ###########################################
#########################################################################################

print "Mandamos una velocidad"
Puerto = '/dev/ttyUSB0'
velocidad = ":" + str(vRight) + "," + str(vLeft) + "\n"

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()

#inicializo variables
v_tot=0
vmed=0

#defino vectores
speeds=[100,110,120,130,140,150,160,170,180,190,200,210,220,230,240,250,260]
vec_K=[]
vec_vel=[]

fecha=time.strftime("%c")
f= open ('calibracion_K_Izq.txt', 'a')
f.write('\n')
f.write(fecha + '\n')
f.write('Caracterizacion del motor rueda izquierda:'+ '\n' + '\n')
```

```
if prueba == 'Serial waiting.\n':
        print "Iniciado, leido " + prueba
        print "Vamos a enviar " + velocidad

        s.write(velocidad)
        eco = s.readline()

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

                for j in range (17):
                        for x in range (3):
                                velocidad=":" + str(vLeft) + "," + str(vRight) + "\n"
                                s.write(velocidad)

                                st=time.time()
                                while (time.time()-st)<2:
                                    contL=0
                                    v_antL=1

                                    stt=time.time()
                                    while(time.time()-stt)<0.5:
                                        rLeft()

                                    v_ensayo=round(((contL*0.9895)/0.5),3) #velocidad en cm/s
                                    v_tot= v_tot + v_ensayo

                                s.write(':0,0\n')
                                v= round((v_tot/4),3)
                                print('la velocidad media es --> ' , v)
                                v_tot=0
                                vmed= vmed + v
                                time.sleep(6)
                        velocidad=round((vmed/3),3)
                        print('velocidad a entrada ', vLeft , ' es --> ', velocidad)

                        K= velocidad/vLeft
                        print('K-->  ',K)
                        print('\n')

                        vmed=0
                        vec_vel.append(velocidad)
                        vec_K.append(K)

                        vRight=vRight+10
                        vLeft=vLeft+10
                        velocidad=":" + str(vRight) + "," + str(vLeft) + "\n"
                        s.write(velocidad)


else:
        print "Error al iniciar"
        s.close()

#Escribimos en el .txt los resultados
f.write('Datos obtenidos:' + '\n')
f.write('entradas testeadas: ' + str(speeds) + '\n')
f.write('velocidades:        ' + str(vec_vel) + '\n')
f.write('K:                  ' + str(vec_K) + '\n')
f.write('\n')


time.sleep(1)
s.write(':0,0\n')
s.readline()
s.close()
```

Figura I.6. Sintaxis completa del programa calculo_K_L.py con el cual se consigue obtener la ganancia del motor de la rueda izquierda en este caso. Este programa solo sirve para dicho objetivo y puede ser usado para cualquier robot, no solo el Pi Robot de este TFG. Una vez acaba la ejecución el programa vuelca los datos obtenidos en un .txt para poder trabajar con ellos en el futuro.

```
import sys
import serial
import time
import RPi.GPIO as GPIO
import numpy as np
import matplotlib.pyplot as plt

#Creado 17-enero
#Sacamos la "funcion de transferencia" con la v_inst
#y el tiempo acumulado de la rueda IZQUIERDA
#Con la repeticion de este programa se saca la tau del sistema IZQUIERDO

GPIO.setmode(GPIO.BCM)
GPIO.setup(20,GPIO.IN, GPIO.PUD_UP)

print "Mandamos una velocidad"
Puerto = '/dev/ttyUSB0'
velocidad = ':255,255\n'

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()

contL=t_antL=tiempoL=t_acumL=0
v_antL=1
vecL=[0]
vec2L=[0]

fecha=time.strftime("%c")
f= open ('ftransferenciainversa_L.txt', 'a')
f.write('\n')
f.write(fecha + '\n')
f.write('estudio de la funcion de transferencia de manera inversa en rLeft del Raspberri Pi robot'+ '\n' + '\n')

if prueba == 'Serial waiting.\n':
        print "Iniciado, leido " + prueba
        print "Vamos a enviar " + velocidad

        s.write(velocidad)
        eco = s.readline()

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

                stt=time.time()
                stL=time.time()
                while (time.time()-stt<0.75):
                        pul20=GPIO.input(20)
                        if pul20==True and v_antL==0:
                                t_antL=tiempoL
                                tiempoL=round((time.time()-stL),4)
                                contL=contL + 1
                                v_antL=pul20
                                stL=time.time()
                                if t_antL!=tiempoL and tiempoL!=0:
                                        vec2L.append(tiempoL)
                                        velL=round((0.9895/tiempoL),2)
                                        vecL.append(velL)
                        else:
                                contL=contL
                                v_antL=pul20

                s.write(':0,0\n')
                t=round((time.time()-stt),6)
                print(t,' segundos')
                print(contL,' marcas')
```

```
else:
        print "Error al iniciar"
        s.close()

time.sleep(0.5)

f.write('frecuencia de lectura de marcas en rLeft y velocidad instantanea:' + '\n')
f.write(str(vec2L) + '\n')
f.write(str(vecL)+ '\n' + '\n')

print(vec2L)
print(vecL)

s.readline()
s.close()
```

Figura I.7. Sintaxis completa del programa finversa.py con el cual se consigue obtener la constante de tiempo (tau) del motor de la rueda izquierda en este caso. Este programa solo sirve para dicho objetivo y puede ser usado para cualquier robot, no solo el Pi Robot de este TFG. Una vez acabada la ejecución guarda los datos en un archivo .txt externo de manera que se trabajan con datos reales extraídos de la propia ejecución

Bibliografía

A continuación se mostrarán los enlaces de Internet, libros especializados o artículos a los que se ha recurrido en algún momento concreto para obtener información:

[1] https://www.raspberrypi.org/help/

[2] https://stackoverflow.blog/2017/09/06/incredible-growth-python/

[3] https://es.wikipedia.org/wiki/Sensor_fotoel%C3%A9ctrico

https://www.keyence.com.mx/ss/products/sensor/sensorbasics/photoelectric/info/

[4] http://ozzmaker.com/berryimu-quick-start-guide/

[5] https://github.com/mwilliams03/BerryIMU/blob/master/python-BerryIMU-gryo-accel-compass-filters/berryIMU.py

[6] https://hipertextual.com/2016/08/acelerometro-giroscopio

[7] https://www.5hertz.com/index.php?route=tutoriales/tutorial&tutorial_id=13

http://www.ingmecafenix.com/automatizacion/giroscopio/

[8] http://www.cartagena99.com/recursos/alumnos/apuntes/7_FUNCION_DE_TRANSFERENCIA_PRIMER_ORDEN.pdf

[9] http://www.elai.upm.es/webantigua/spain/Asignaturas/Servos/Apuntes/6_AnaTemp_1_2.pdf

Análisis temporal. Tema 3. Sistemas automáticos. Pag 6

[10] Análisis temporal. Tema 3. Sistemas automáticos. Pag 7-8

[11] formulario examen. Sistemas automáticos. Pag 34

[12] Diseño de controladores digitales. Tema 3. Ingeniería de control. Pags 14-17

[13] http://www.demaquinasyherramientas.com/herramientas-de-medicion/introduccion-al-medidor-de-distancia-a-laser

[14] Hung-shiang chuang, Ying-Chun Chuang, Chun-Hsiang Yang “Development of a low-cost platform for control engineering education”. Pags 444-448

Yoonsoo Kim, “Control systems lab using a LEGO Mindstroms NXT motor system”. Pags 452-461

[15] https://www.python.org/

https://es.wikihow.com/comenzar-a-programar-en-Python

https://docs.python.org/3/library/

https://recursospython.com/guias-y-manuales/calcular-tiempo-de-ejecucion/

https://www.pybonacci.org/2013/10/10/teoria-de-control-en-python-con-scipy-i/

http://cs231n.github.io/python-numpy-tutorial/

[16] https://www.nerion.es/soporte/tutoriales/comandos-basicos-de-programacion-linux/

[17] https://stackoverflow.com/

[18] http://www.pridopia.co.uk/pi-motor-encoder-2p-2w.html

[19] https://www.raspberrypi.org/documentation/usage/gpio/

[20] https://es.mathworks.com/products/curvefitting.html

[21] https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process