eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Estudio de la cinemática de un móvil de dos ruedas y su control para mantener el equilibrio teledirigido por Bluetooth

Facultad de Informática

Grado de Ingeniería Informática

Ingeniería de computadores

informatika fakultatea

facultad de informática

Autor:

Markel Picado Ortiz

Director:

Luis Gardeazabal Montón.

Septiembre 2015

RESUMEN

Este proyecto se centra en el control de un robot con estabilidad dinámica más concretamente en un robot que responde al conocido modelo del péndulo invertido. Una de las aplicaciones más famosas del péndulo invertido es el *Segway*. El péndulo invertido del *Segway* se define solo en un eje, al tener dos ruedas paralelas como soporte.

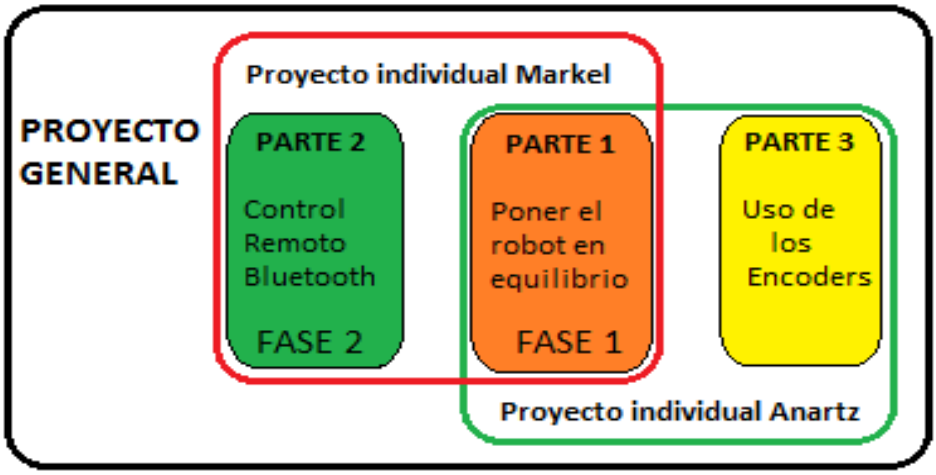
El proyecto general consta de tres partes, poner el robot en equilibrio, el control remoto del robot vía Bluetooth y el **uso de los *encoders* de los motores para manejar el robot.**

Poner el robot en equilibrio: Esta parte consiste en la lectura e interpretación de los datos que nos ofrece la Unidad de Medición Inercial (IMU) para conseguir que el robot se mantenga horizontalmente en todo momento utilizando un control PID y se ha desarrollado junto al compañero, Anartz Recalde.

Las otras dos partes están pensadas con el objetivo de darle más personalidad al proyecto individual de cada uno.

Control remoto vía Bluetooth: Con esta parte lo que se pretende es controlar el robot remotamente mediante el Bluetooth y ha sido desarrollada únicamente por mí.

Uso de los *encoders* de los motores para manejar el robot: Esta parte ha sido totalmente desarrollada por **Anartz Recalde**, y consiste en hacer uso de los *encoders* de los motores e interpretar los datos que proporcionan para sacar distintos datos como la distancia recorrida o la velocidad. De esta manera, intentará conseguir que el robot en todo momento se mantenga en la misma posición, es decir si alguna fuerza externa al robot hace que el robot se mueva, éste se tiene que situar en la misma posición en la que estaba antes de que se le hubiera ejercido dicha fuerza externa.



Por lo tanto, este proyecto consiste en construir un robot balancín, bastante similar a un *Segway* pero en una escala más pequeña, conseguir que se mantenga en equilibrio y manejarlo por control remoto mediante uso del Bluetooth.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

Ya que existen partes en común en el proyecto la redacción de esos apartados han sido escritos en conjunto con Anartz Recalde. Los apartados que se han escrito en conjunto son los siguientes:

AGRADECIMIENTOS

En primer me gustaría dar las gracias a algunas personas que han hecho posible la realización de este proyecto.

En primer lugar, dar las gracias a Anartz Recalde por toda la ayuda y el apoyo que me ha aportado durante el desarrollo del proyecto.

Me gustaría agradecer a todos los compañeros de Grado que me han mostrado su interés y que me han aportado consejos.

A Bego Ferrero por facilitarnos un aula en Las Escuelas Universitarias de Ingeniería Técnica Industrial (EUITI) para poder seguir trabajando durante el verano.

También a mis padres, mi hermana y a Leire por su apoyo y presencia incondicional.

Y por último, a mi tutor Luis Gardeazabal Montón por toda la infraestructura que me ha proporcionado para poder llevar a cabo este proyecto, gracias.

1

·1·

1. INTRODUCCIÓN

1.1. Visión general

A medida que pasan los años, la tecnología está cada vez más presente en la vida cotidiana de las personas.

La robótica es una rama de la tecnología que combina diversas disciplinas como son: la mecánica, la electrónica, la informática, la inteligencia artificial, la ingeniería de control y la física. Gracias a la robótica se pueden dar soluciones a distintos problemas cotidianos que aparecen en el día a día de cualquier persona.

Debido a esto, la robótica asume un papel importante, además de a nivel industrial, a un nivel particular, ofreciendo grandes ventajas y comodidades al ser humano así como nuevas formas de ocio.

Una de las principales diferencias a destacar entre los robots, en cuanto a la mecánica se refiere, es el tipo de estabilidad, que puede ser estática o dinámica. Un robot con estabilidad estática se refiere a un robot en el que su funcionamiento no afecta a su centro de gravedad mientras que en el caso de un robot con estabilidad dinámica sucede todo lo contrario.

La estabilidad dinámica trae bastantes ventajas respecto a la estática, como puede ser una mayor facilidad para evitar obstáculos o diseñar robots que tengan un aspecto más humanoide.

La robótica poco a poco se va integrando tanto en la industria como en la sociedad, amoldándose y dando solución a problemas que hace unos años serían imposibles de solucionar.

Por diversos factores, uno de los grandes problemas de la sociedad actual es el transporte. Disponemos de diferentes medios de transporte como son, los coches, los barcos, los trenes, los autobuses… en los cuales, cada vez es más frecuente el uso de la tecnología. Gran parte de esta nueva tecnología está basada en los microcontroladores que se empiezan a usar cada vez más debido a las prestaciones que pueden ofrecer, agilizando y facilitando alguno de los procesos de estos vehículos. Como ejemplo podemos pensar en el control de vuelo para un avión, entre otras muchas cosas.

Hoy en día, el transporte es de gran importancia ya que es el medio por el cual se pueden establecer conexiones entre distintas ciudades, pueblos… de manera rápida, cómoda y segura aparte de ayudar en la economía de un país con las importaciones y exportaciones o transportando a millones de personas a sus puestos de trabajo…

Debido a la falta de espacio a la hora de aparcar un coche, o a la contaminación que producen los diferentes medios de transporte o simplemente por comodidad o para mejorar la circulación, día tras día y gracias a la tecnología se desarrollan nuevos prototipos de vehículos para el transporte de personas. Entre estos nuevos medios de transporte encontramos el *Segway.*

El *Segway* es un vehículo de transporte ligero eléctrico de dos ruedas, a la vez de ser el primer dispositivo de transporte autobalanceado inventado por **Dean Kamen**. El ordenador y los motores situados en la base mantienen la base del *Segway* horizontal en todo momento.

Partiendo de la idea del *Segway* lo que se propone con este proyecto es realizar un estudio experimental de diferentes tecnologías como lo son los microcontroladores, el Bluetooth, los acelerómetros y los giroscopios, usando una estructura para el robot con unas dimensiones más pequeñas que las de un *Segway*.

1.2. Objetivos

El proyecto tiene dos objetivos principales, la puesta en marcha y en **equilibrio del robot balancín** y el **control remoto vía Bluetooth** del mismo.

El desarrollo de éstos, conllevan otros objetivos y competencias transversales como:

- Trabajo en equipo.

- Conocer las Unidades de Medición Inercial (IMU), que sensores llevan y como funciona cada uno.
- Conocer la teoría del péndulo invertido.
- Profundizar en la programación con microcontroladores PIC (I2C, PWM, RS232…).
- Conocer y aplicar la corrección de medidas mediante el filtro de Kalman.
- Comprender e implementar el sistema de equilibrio del robot mediante un control PID.

1.3. Planificación y gestión del proyecto

A continuación se mostrarán las distintas etapas que se han identificado y la forma en la que se han planificado en el tiempo para llevar a cabo de manera exitosa el proyecto.

En el proyecto hay que diferenciar dos partes, una primera relacionada con poner el robot en equilibrio, la cual se ha llevado a cabo junto a un compañero, Anartz Recalde. Una vez terminada esta fase, cada uno ha desarrollado un proyecto individual sobre el robot. En mi caso, la segunda parte ha consistido en el control remoto del robot por Bluetooth.

En la estructura de desglose de trabajo que se presenta en la Figura 1 se pueden diferenciar cuatro bloques importantes: **Planificación, Análisis, Desarrollo** y **Seguimiento y Control.**

En la parte de la **Planificación**, se trabaja la estrategia que se llevará a cabo para conseguir que el proyecto se satisfaga, apoyándonos en un diagrama de Gantt, en un plan de comunicación y en un plan de riesgos.

El **Análisis** consiste en analizar el problema al que nos enfrentamos, en este caso a la implementación de un robot balancín que se basa en la teoría del péndulo invertido. También se analizan los distintos componentes hardware que se utilizarán, como pueden ser la Unidad de Medición Inercial IMU, los Motores, el módulo Bluetooth… Esta parte engloba todo lo que es la parte teórica del proyecto.

El **Desarrollo** trata de poner en práctica todo lo que se ha analizado y aprendido en la parte de Análisis. En ocasiones tanto el Desarrollo como el Análisis se llevarán a cabo al mismo tiempo. Esta parte, hace referencia a la implementación del código del robot y de los distintos dispositivos que van conectados a éste como son la

Unidad de Medición Inercial (IMU), los controladores de potencia de los motores y el módulo Bluetooth.

Por ultimo en el **Seguimiento y Control**, se lleva acabo todo lo relacionado con la memoria del proyecto, el control de versiones tanto de código como de la documentación y la replanificación.

1.3.1. Estructura de desglose de trabajo

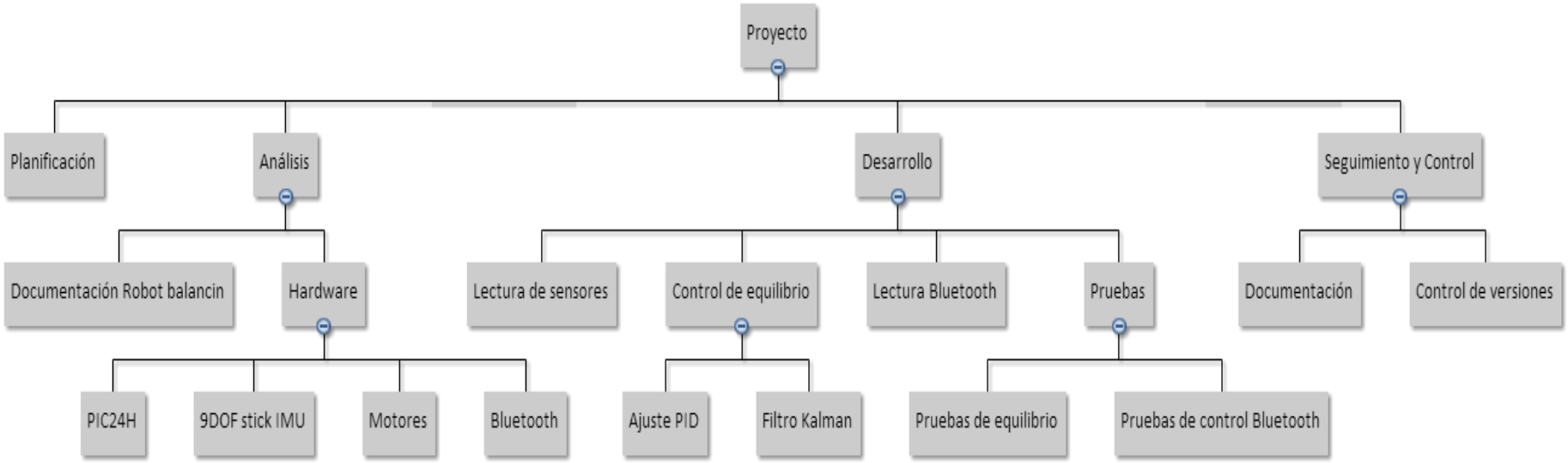
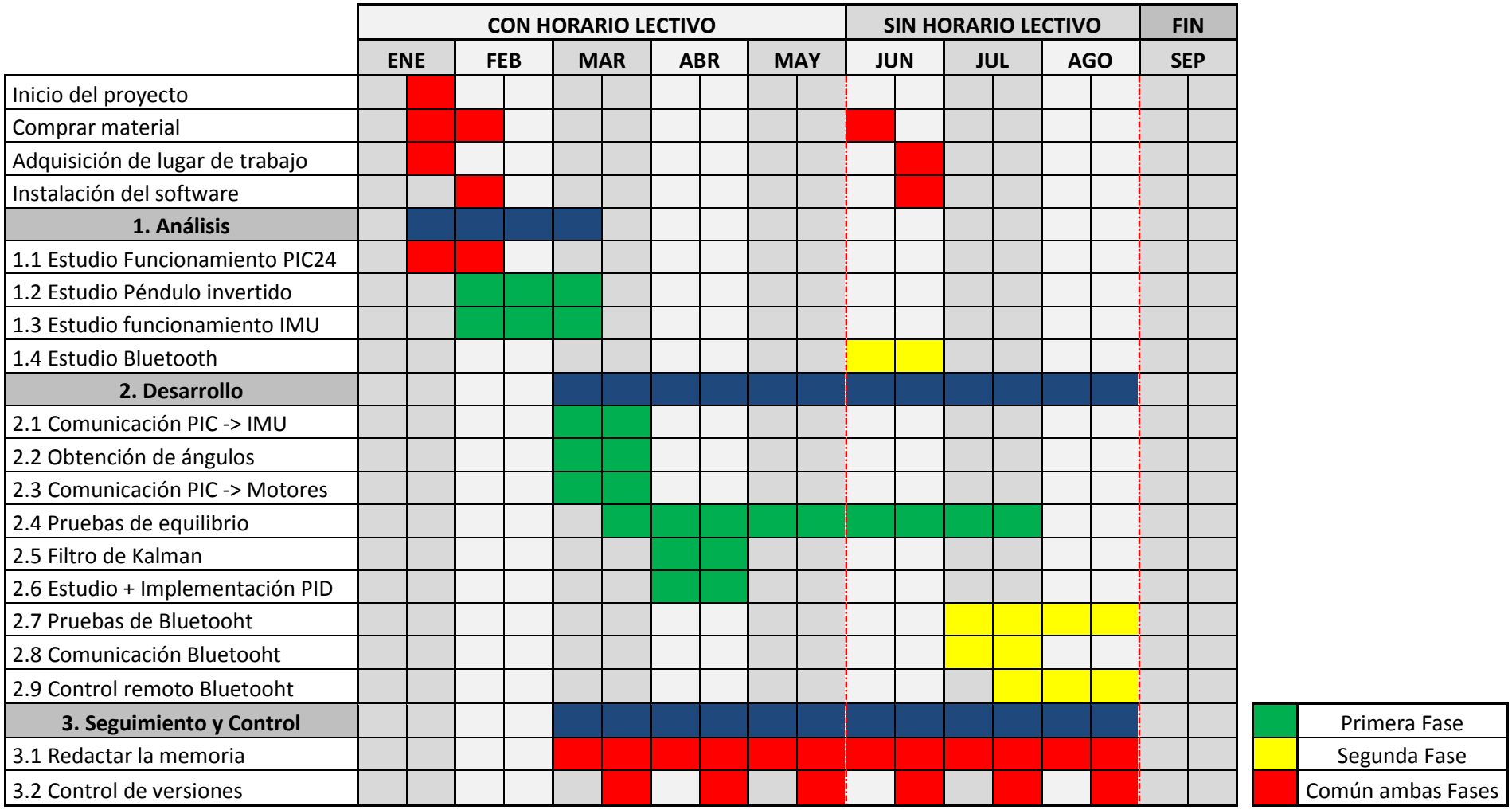


Figura 1: Estructura de desglose de trabajo (EDT)

1.3.2. Diagrama de Gantt



En el diagrama de Gantt se puede observar como la primera parte (la parte de poner en equilibrio el robot) se nos ha alargado mucho tiempo aun estando trabajando dos personas en ella. Ello ha sido debido a los distintos problemas mecánicos que han surgido en el desarrollo de esta parte y que se escapaban de nuestros conocimientos. Estos problemas nos han retrasado bastante el proyecto y dejado poco tiempo para la implementación de la segunda parte, la parte individual.

Los retrasos han sido debidos a la rotura de los motores, la rotura de dos controladores de potencia de los motores, incluso uno de ellos llego a incendiarse y finalmente también se nos estropeo una placa *Explorer 16.*

En la sección 5, se detallan estos problemas, y cómo se fueron detectando y resolviendo. Ya que aunque exista un problema, puede llegar a parecer durante un largo tiempo que todo funciona correctamente y ajustar unos parámetros (como los del controlador del PID) a un sistema que realmente está dañado que para cuando te das cuenta todo el trabajo empleado prácticamente no ha servido de nada. En esa sección también se describe como fueron solucionados los problemas una vez detectados.

1.3.3. Plan de comunicación

La manera de comunicación con el director del proyecto se puede diferenciar en dos partes, la primera parte está definida por la duración del cuatrimestre (desde Enero hasta Mayo) y la segunda parte en la que ya ha finalizado el curso (desde Junio hasta Agosto).

Primera parte:

En esta primera parte, al tener la vivienda al lado de la facultad y disponer de un laboratorio para trabajar en la facultad, la comunicación con el director era directa, bastaba con acudir a su despacho.

Segunda parte:

En esta segunda parte al terminar el curso y tener la vivienda fija en otra ciudad distinta (Bilbao) el plan que se ha llevado a cabo para la comunicación ha sido el siguiente.

- Reunirnos una vez a la semana (los jueves) en Donostia para el seguimiento y control del proyecto.

- En caso de que surja algún problema con el proyecto que no pueda solucionarse a priori o alguna novedad importante convocar una reunión en Donostia mediante el correo electrónico.

1.3.4. Plan de riesgos

Los principales riesgos que se han detectado han sido los siguientes:

- Perdida del código desarrollado.
- Perdida de la documentación desarrollada para la memoria.
- Rotura de alguna pieza.
- Existencia de un único robot.

Para evitar los dos primeros riesgos, simplemente se ha llevado un control de versiones tanto en Dropbox (para el código) como en Drive (para la memoria).

En caso de rotura de alguna pieza, se ha intentado disponer de repuestos para las piezas más débiles.

Por último, dos personas distintas estábamos trabajando sobre el mismo robot. Cuando estábamos en la facultad, al trabajar en común y en el mismo laboratorio los dos no teníamos ningún problema, pero al terminar el curso los dos nos volvimos a Bilbao y para poder seguir trabajando los dos a la vez en el mismo proyecto solicitamos un aula en Las Escuelas Universitarias de Ingeniería Técnica Industrial (EUITI) situada en Bilbao, la cual se nos concedió.

·2·

2. ROBOT BALANCIN

Un robot balancín o robot auto balanceado es un robot de dos ruedas alineadas a un mismo eje, que sostiene una estructura (en este caso el cuerpo del robot) en equilibrio sin ningún tipo de fuerza externa, únicamente ayudándose de los motores con las que mover las ruedas.

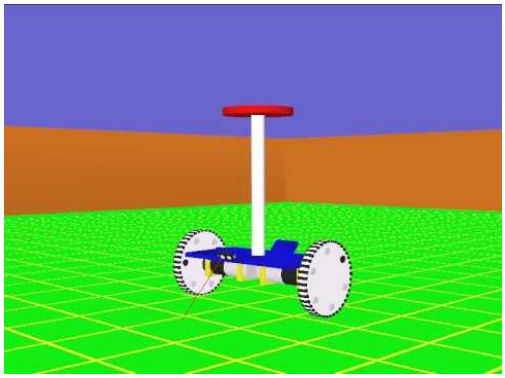


Figura 2: Robot balancín virtual.

Este robot responde como un sistema de carro-péndulo que es una variante del conocido problema del péndulo invertido en el que se utiliza la base móvil (el carro) para conseguir mantener la masa (el péndulo) en equilibrio.

2.1. Péndulo invertido

El problema físico del equilibrio de un péndulo invertido consiste en un péndulo cuyo centro de masas se encuentra situado por encima del punto o eje de balanceo. Esto crea al sistema una inestabilidad estática, que aplicando un momento o par determinado en el eje de giro, o mediante un movimiento en el plano horizontal puede ser compensada.

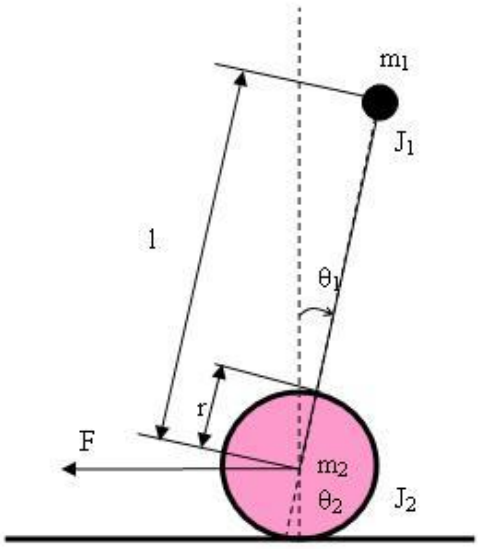


Figura 3: Fuerzas que afectan al problema del péndulo invertido.
Fuente: http://nxttwowheels.blogspot.com.es/

Para conseguir este equilibrio nos basamos en el ángulo formado por la vertical, y aplicaremos una fuerza a los motores, aplicando un control PID para que el ángulo formado por la vertical se mantenga a "0" grados.

Si consideramos un punto de balanceo fijo, y un sistema de movimiento en 2 dimensiones, obtenemos la siguiente ecuación de comportamiento del sistema. Una función en la que la aceleración (a) es dependiente de la gravedad (g), de la longitud del péndulo (l) y del seno del ángulo formado (θ).

$$a = \frac{g}{l} \sin\theta$$

$$\sum F = m \times a$$

En este proyecto y de una manera más simplificada (ya que no utilizaremos esa ecuación de comportamiento) trataremos de dar solución a un robot que tiene las características de un péndulo invertido y dotarlo de control remoto mediante Bluetooth.

2.2. Historia

Los robot balancín están basados en los *Segway Personal Transporter (Segway trasportador de personas),* por lo que podríamos considerar que nuestro robot es un tipo de Segway pero que en este caso no transporta personas.

Antes de su demostración pública del *Segway* en 1990, se podrían encontrar en los distintos medios, varios informes y rumores de un invento revolucionario, pero no existían detalles disponibles. El recibimiento inicial fue entusiasta: magnate de los negocios del sector informático, Steve Jobs, sugirió que las ciudades serían construidas alrededor de este nuevo método de transporte, y John Doerr predijo ventas que llegarían a los 100.000 millones de euros por lo que para afrontar la demanda esperada, la fábrica de Bedford fue diseñada originalmente para construir hasta 40.000 unidades mensuales.

Tras la demostración se vendieron tres ejemplares en una subasta en Amazon por más de 90.000 euros.

La compañía esperaba vender 50.000 unidades el primer año pero tras casi dos años solo se vendieron 6.000.

Tras sufrir varios contratiempos, incluido la retirada voluntaria de todos los *Segway PT* en 2003, la compañía Segway Inc. en el 2005 trabajó duro para incrementar su cuota de mercado y recuperar las inversiones en I+D y producción.

Pese a algunas publicaciones escépticas, es posible que el *Segway* aún sea un éxito comercial, una vez que se hayan cubierto las inversiones. De momento cuenta con más de 100 concesionarios y distribuidores internacionales.

El coste elevado de los *Segway* se considera el principal responsable de la baja demanda. Para eliminar la mala imagen asociada con el vehículo tras algún problema y su fracaso en el mercado, Segway Inc. ha abierto concesionarios por todos los Estados Unidos en los que la gente puede examinar y probar los *Segway PT* aún así no hay indicios de que el mercado vaya a considerarlo próximamente como un vehículo práctico para uso diario en lugar de como un juguete caro.

A pesar de que los creadores piensen que es ideal para las áreas urbanas densas y grandes ciudades, fueron diseñados para ser utilizados por autopistas y carreteras interestatales. Los lugares en los que el *Segway* podrían ser más exitosos parece que serían aquellas con una distancia corta entre el hogar, el trabajo y las áreas de ocio, por lo que para que tenga éxito en el mercado del transporte personal

dependerá en gran medida del estilo de desarrollo urbano o la capacidad de producir modelos más seguros y con mayores velocidades.

2.3. Actualidad

En la actualidad, el *Segway* sigue sin tener éxito como vehículo de transporte personal, y los factores para que esto sea así tampoco es que hayan cambiado mucho ya que siguen teniendo un precio demasiado elevado (entre 2.000 y 6.000 euros). De todos modos, se está abriendo hueco en muchos núcleos urbanos pero no tanto como para las personas que residen en él, si no que más bien para los turistas, ya que son utilizados para dar paseos por la ciudad acompañado de un guía y así recorrer la ciudad de una manera más cómoda y divertida.

Como es normal en este tipo de tecnologías, se ha seguido investigando y desarrollando nuevos modelos. Se ha utilizando un enfoque distinto pero manteniendo siempre la idea y la esencia en la que se basan este tipo de robots, el péndulo invertido. En la actualidad nos podemos encontrar distintos modelos que pueden llegar a ser más baratos, más caros, menos robustos, más fáciles de manejar, más pequeños incluso cómo no, como es en el caso de este proyecto robots auto balanceados y con control remoto diseñados desde un inicio para el ocio y no para el transporte personal.

Por lo que parece, se ha abierto una línea de trabajo en la que ya no se habla tanto de *Segway* y se empieza a abrir más el campo a vehículos o robots balancín. Esto hace que existan en el mercado distintos modelos de vehículos y robots cuya esencia está basada en la teoría del péndulo invertido y el autobalanceo.

A continuación se muestran distintos modelos desde el clásico *Segway* hasta el más novedoso, pasando por los robots a escala más pequeña controlados de forma remota y que no son utilizados para el transporte de personas.

De cada uno de los modelos que aparecen a continuación existen diferentes variantes dependiendo del fabricante, pero la idea siempre sigue siendo la misma y es que son vehículos o robots auto-balanceados y pueden variar más que nada en el chasis, la batería, la autonomía, el tiempo de carga, los colores…

Figura 4: Distintos modelos de vehículos auto balanceados.

***Segway* clásico:** Este modelo es el modelo más tradicional y está formado por dos ruedas unidas por el mismo eje, una plataforma en la que el pasajero se coloca de pie y dispone de un manillar para apoyar las manos. una vez montado en el *Segway*, basta con balancearte hacia adelante (para ir hacia adelante) o hacia atrás (para ir hacia atrás) para que el vehículo se ponga en marcha. Para girar, se hace uso del manillar, moviéndolo hacia el lado izquierdo (para girar hacia la izquierda) o moviéndolo hacia el lado derecho (para girar hacia la derecha)

Figura 5: *Segway* clásico.

Smart S1: Siguiendo la filosofía del *Segway* clásico, la compañía *Chic-Robot* ha diseñado este scooter eléctrico personal llamado Smart S1, es capaz de circular a una velocidad máxima, de 10 km, recorriendo una distancia de 20 kilómetros con una sola carga. Este scooter, de movilidad personal, tiene un control de giroscopio y un micro-

controlador que acciona el movimiento en cualquier dirección con los movimientos del pie.

En este caso, son dos ruedas unidas por un mismo eje pero que prescinde de manillar para hacer los giros, ya que dependiendo de la inclinación de cada pie gira hacia un lado o hacia el otro.

Figura 6: *Smart S.*

Ecovemp: Este modelo es un modelo de una única rueda y está basado en la misma filosofía que el *Segway* ya que es un vehículo para transportar personas auto balanceado. Tiene 25 kilómetros de autonomía, tarda hora y media en cargarse con 10 céntimos de electricidad, y pesa 13,5 kilogramos.

Figura 7: *Ecovemp*.

Robot balancín: Este es un robot que básicamente sigue la idea del *Segway* con una estructura muy similar pero con unas dimensiones más pequeñas que no sirven para transportar a personas. En este proyecto se ha diseñado un robot con estas propiedades cuyas características son definidas en las diferentes secciones de este documento.

Figura 8: Robot balancín

·3·

3. TECNOLOGÍA UTILIZADA

A continuación se muestran los detalles del Hardware y Software utilizado para el desarrollo del proyecto.

3.1. Hardware

Lo primero de todo aclarar que nos hemos basado en un robot que ya estaba desarrollado por una empresa (http://www.elecrow.com/freescale-mc9s12xs128mal-2wheel-selfbalancing-robot-p-878.html), por lo que adquirimos un robot de estos y sustituimos el microcontrolador que traía (porque no tenemos el sistema de desarrollo para ese procesador) y utilizamos el nuestro (por comodidad) junto a la placa *explorer 16*.

Debido a diversos problemas, tuvimos que cambiar los motores ya que se estropearon, por lo que prácticamente lo único que aprovechamos de ese robot fueron las ruedas, la batería y la estructura.

3.1.1. Motores

Los motores eléctricos que finalmente se han utilizado son unos motores Metal Gearmotor que incorporan unos encoders que tienen una resolución de 64 cuentas por vuelta.

Figura 9: Motor *Metal Gearmotor* con *encoder* incorporado.

Son unos motores con una carcasa de metal de 6,45 × 3,68 × 3,68 cm que funciona a 12V con una relación de 29:1, que pueden alcanzar hasta 350rpm.

3.1.2. Controladores de los motores

Como controladores de los motores hemos usado un puente H para cada motor, concretamente los *Simple H robot power*.

El *Simple H robot power* es un puente H robusto diseñado para controlar motores de corriente continua (CC). Gracias a éstos y a las señales de entrada que le mandemos desde el microcontrolador, conseguiremos que el motor gire en un sentido o en el otro con más o menos velocidad (dependiendo del *duty* del PWM)

Figura 10: *Simple H robot power.*

Las características del *Simple h robot power* son las siguientes:

Tensión de alimentación:	De 5V a 28V
Corriente de salida (continua):	20A
Frecuencia PWM:	20kHz
Interfaz lógica:	3V - 5V Min. Requiere 2 señales digitales para el control.

3.1.3. Microcontrolador

Hemos usado un microcontrolador PIC de 16 bits, en concreto el PIC24HJ256GP610A, con el que ya estamos familiarizados gracias a la asignatura *Diseño de Sistemas Integrados.*

Las características más interesantes para llevar a cabo este proyecto son las siguientes:

- Velocidad de CPU: 40 MIPS.
- 256 KB de memoria de programa.
- 2 buses I2C.
- 2 módulos RS232.
- 8 canales PWM.
- 9 *timers* de 16 bits o 4 de 32 bits.
- *Prescaler* ajustable para los *timers*.
- Capacidad de programar casi todo mediante interrupciones, para agilizar la ejecución.

Para la programación del microcontrolador y la depuración del código se ha utilizado el *debugger REAL ICE* y la placa de desarrollo *Explorer 16*, ambos del fabricante Microchip.

Figura 11: Diagrama de conexión para la programación del microcontrolador desde el PC utilizando la placa *Explorer 16.*

3.1.4. Módulo Bluetooth

Para llevar a cabo este proyecto se han utilizado dos módulos Bluetooth RN4020 de Microchip, funcionando uno en modo maestro y el otro en modo esclavo.

El módulo RN4020 cumple con la especificación de Bluetooth versión 4.1. Integra RF, un controlador de banda base, procesador de API de comandos, lo que lo convierte en una solución de bajo consumo Bluetooth completa.

Figura 12: Módulo Bluetooth RN4020.

Para la aplicación de control remoto que hemos diseñado, muchas de las funciones que ofrece este módulo no las hemos utilizado ya que para este proyecto no aportan ningún valor o función y solo nos pueden causar retrasos de tiempo con implementaciones de código más complejas.

Características

- Módulo versión 4.1 Bluetooth® totalmente homologado
- Pila 4.1 de bajo consumo Bluetooth en placa
- Interfaz de comandos ASCII API sobre UART
- Factor de forma compacto 11,5 x 19,5 x 2,5 mm
- Almohadillas SMT acanaladas para montar de forma fácil y fiable el PCB
- Respeta el medio ambiente y cumple con RoHS
- Certificaciones: FCC, IC, CE, QDID
- Actualización del firmware del dispositivo DFU) sobre UART o por aire (OTA)
- Perfil de datos de bajo consumo de Microchip (MLDP) para aplicaciones de datos serie
- Comandos remotos por aire
- Única tensión operativa: de 3,0 V a 3,6 V (3,3 V típica)
- Intervalo de temperatura: de −30 °C a 85 °C
- Bajo consumo de energía
- Interfaz UART
- Cristal integrado, regulador de tensión interno, circuitos adaptados, amplificador de memoria y antena PCB
- Varias E/S para control y estado
- Interfaz UART, GPIO, ADC
- 64 KB de flash serie interna

Características analógicas/RF

- Funcionamiento en banda ISM de 2,402 a 2,480 GHz
- Canales 0-39
- Sensibilidad Rx: -92,5 dBm a 0,1% VER

- Energía Tx: +7 dBm

MAC de monitor RSSI/Banda base/Mejores funciones de capa

- Cifrado AES128 seguro
- GAP, GATT, SM, L2CAP y perfiles públicos integrados
- Crear servicios personalizados con la API de comandos
- Autenticación de E/S con teclado
- Función configurable con software como periférico o central, cliente o servidor

3.1.5. Dispositivo para el control remoto

Para el control remoto por Bluetooth se pensaron dos soluciones posibles, una seria controlar el robot desde un dispositivo móvil con sistema operativo *Android* y la otra controlar el robot desde otra placa *Explorer 16* (junto al PIC24HJ256GP610A) distinta a la que utiliza el robot.

Las primeras pruebas se realizaron con un dispositivo *Android*, pero debido a problemas de sincronización y vinculación entre los dispositivos Bluetooth, que se analizan con más detalle más adelante. Finalmente se ha optado por utilizar la placa para desarrolladores *Explorer 16*.

Esta placa permite la evaluación a bajo coste y de forma eficiente de las prestaciones de estos nuevos controladores de señal digital (DSC) de 16 bits dsPIC33. Diseñada para poder trabajar con el depurador *REAL ICE* que utilizaremos para este proyecto. Las facilidades de depuración y la emulación en tiempo real nos ayudan a acelerar la evaluación de la aplicación.

Figura 13: Placa *Explorer 16.*

Además del conector para el módulo con el PIC24 o con dsPIC33, esta placa incluye conexión RS-232, conectividad USB mediante un PIC18F4550, el conector estándar para el REAL ICE, pantalla de cristal líquido de 2x16 caracteres, pulsadores de entrada para el usuario, 8 LED para indicación, área de expansión para prototipos, zócalo y conector de borde de placa para futuras placas de expansión *PicTail™ Plus*, entre otras muchas características.

3.1.6. Unidad de Medición Inercial (IMU)

La Unidad de Medición Inercial o IMU (*Inertial Measurement Unit*), es el dispositivo compuesto de tres diferentes sensores (acelerómetro, giroscopio y magnetómetro) que en nuestro caso se encarga de medir el ángulo de inclinación del robot balancín. Los datos que nos proporciona sirven para que el robot mantenga el equilibrio mediante el control PID.

En nuestro caso hemos utilizado la placa 9DOF *Stick* de *Sparkfun*, que contiene un acelerómetro, un giróscopo y un magnetómetro (o brújula), todos tridimensionales y conectados al bus I2C para su configuración y uso.

Figura 14: Unidad de Medición Inercial (IMU).

Las características principales de los sensores son las siguientes:

Acelerómetro ADXL345:

- Resolución ajustable de 10 a 13 bits.
- Lectura máxima: ±16g (con 13 bits).
- Sensibilidad de 4mg/LSB, permitiendo detectar cambios menores a 1° en la inclinación.
- Datos de salida formateados a 16bits en Complemento a 2.

- Soporta I2C a 100kHz y a 400kHz, trasmisiones simples y múltiples.

Giróscopo ITG-3200:

- 16 bits de resolución.
- Lectura máxima: ±2000°/s.
- Sensibilidad de 14,375 LSB/(°/s).
- Frecuencia de muestreo ajustable de 1 a 8kHz.
- Frecuencia de salida ajustable de 8.000 a 3,9 muestras por segundo.
- Datos de salida formateados a 16bits en Complemento a 2.
- Soporta I2C a 100kHz y a 400kHz, trasmisiones simples y múltiples, *SlewRate* necesario.
- Sensor de temperatura integrado.

Magnetómetro HMC5883L:

- 12 bits de resolución.
- Lectura máxima: ±8 Gauss.
- Sensibilidad de 2 miligauss/LSB, en grados de 1° a 2°.
- Frecuencia de muestreo máxima de 160 Hz.
- Varias muestras para calcular el valor de salida, ajustable de 1 a 8 muestras.
- Frecuencia de salida ajustable de 0.75Hz a 75Hz (por defecto 15Hz).
- Datos de salida formateados a 16bits en Complemento a 2.
- Soporta I2C a 100kHz y a 400kHz, trasmisiones simples y múltiples.

Fusionando mediante el filtro de Kalman los datos obtenidos del acelerómetro y del giróscopo intentaremos aproximar con la máxima exactitud la inclinación del robot.

3.1.7. Batería

La batería que hemos utilizado es la batería que proporcionaba el fabricante al que compramos el robot balancín.

Es una batería de litio de 12V y 5200mAh.

3.1.8. Estructura

Al igual que la batería, también hemos reutilizado la estructura que traía el robot que compramos, que está formada por tres placas de tamaño 15 cm x 7 cm

aunque una de ellas, la superior ha sido sustituida para poder sujetar con mayor estabilidad la placa *Explorer 16* y esta mide 14,6 cm x 12 cm.

Figura 15: Estructura utilizada para el robot en este proyecto.

En la Figura 15 aparece el robot que compramos, en la parte del centro tiene el microcontrolador y los sensores que utiliza, pero en nuestro caso esa parte del centro lleva los controladores de los motores y la Unidad de Medición Inercial (IMU). En la parte superior lleva la placa *Explorer 16*, el microcontrolador y el módulo Bluetooth.

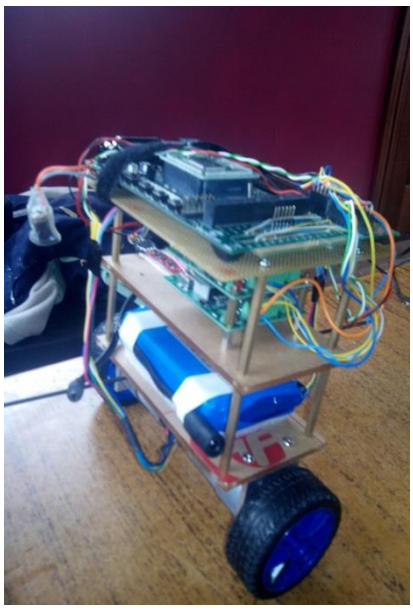


Figura 16: Estructura final del robot utilizado para el proyecto.

3.2. Software

3.2.1. Entorno de desarrollo

Hemos utilizado el IDE (Integrated Development Environment) suministrado por Microchip MPLAB X, que es una versión del IDE NetBeans (gratuito y de código abierto) modificada por Microchip para ajustarse a las necesidades del desarrollo en PIC.

3.2.2. Control de versiones

Para llevar un control de las versiones tanto del código como de la memoria y toda la parte del seguimiento y control del proyecto se han utilizado dos plataformas distintas, Dropbox y Drive.

Son plataformas que permiten a los usuarios almacenar y sincronizar archivos en línea y entre ordenadores y compartir archivos y carpetas con otros usuarios.

Dropbox

La plataforma de Dropbox ha sido utilizada para tener copias de seguridad del código desarrollado hasta el momento así como versiones distintas del código que funcionaban correctamente.

Drive

Para asegurar la información desarrollada en la memoria hasta el momento se ha utilizado la plataforma Drive, en la que se han ido guardando distintas copias de la memoria por motivos de seguridad.

El haber utilizado distintas plataformas para el código y para la memoria no proporciona ningún problema ya que con ambas plataformas estoy familiarizado y no me supone ninguna ventaja ni desventaja el uno respecto al otro.

·4·

4. SISTEMA DE CONTROL

4.1. Lecturas de sensor IMU

4.1.1. Bus de comunicación I²C

La primera tarea que hemos realizado en cuanto a la construcción del robot consiste en la lectura de los sensores. Esta lectura se realiza mediante el bus de comunicación I2C (*Inter-integrated circuit*). Este bus de comunicaciones es muy usado en sistemas integrados para la comunicación con sus periféricos como para comunicar varios circuitos integrados entre sí.

En el caso de nuestro robot, usaremos este bus de comunicaciones para comunicar el acelerómetro y el giroscopio. Para implementar este código lo que hemos decidido es reutilizar el código del I2C que teníamos de otro proyecto y simplemente adecuarlo al nuestro.

4.1.2. Lectura de datos de sensores acelerómetro y giroscopio.

Ahora que comprendemos cómo funciona el bus de comunicación I^2C procederemos a explicar cómo leemos estos datos. Para empezar tenemos que llamar a la función de leer los datos pero la primera pregunta que se nos plantea es ¿Cada cuánto debemos leer los datos? Tras investigar en diferentes proyectos sobre péndulos invertidos y robots balancines asumimos que leer los datos cada 10 milisegundos es un tiempo asequible para leerlos, ya que no leemos con una frecuencia demasiado alta como para que colapse el funcionamiento del robot y tampoco tan lento para que al robot no le dé tiempo a corregir la trayectoria.

Para que la lectura sea lo más fiel posible respecto a ese tiempo hemos decidido crear una interrupción y en ella gestionar la lectura de los datos y la interpretación de estos, puesto que de nada serviría interpretar los datos sin antes haberlos leído (tendríamos datos de la lectura anterior en vez de la actual que es la que nos interesa).

Una vez que decidimos cada cuanto se actualizan los datos relacionados con la IMU se procede a programar una interrupción con un temporizador donde se llama a las funciones de leer acelerómetro y leer giroscopio.

```
void _ISR _T7Interrupt (void)   // Prioridad 1
{
    I2C1CONbits.SEN = 1;
    if (estado_int==0){
         Leer_Acel();
         estado_int=1;
    }
    //disp=1;
    else{
        Leer_Giro();
.
.
.
```

Se puede observar que hemos creado una máquina de estados para que así cada llamada lea el acelerómetro o el giroscopio. Esto se debe a que el proceso de lectura de acelerómetro o giróscopo son costosas en cuanto a tiempo y por tanto si hacemos las dos llamadas juntas la posibilidad de que el robot se quedase bloqueado en esos bucles mientras programábamos aumentaban y por tanto nos daba más errores a la puesta en marcha después de programar.

4.1.3. Calculo de ángulos partiendo de la lectura de los sensores

Una vez que obtenemos los valores del giroscopio y del acelerómetro con un periodo de 10 milisegundos toca interpretar estos valores.

El giroscopio nos da cada vez que leemos la velocidad de giro respecto a la lectura anterior. Es decir, si leemos y a continuación giramos 90 grados y volvemos a leer el giroscopio nos dará un valor que corresponde a esos 90 grados. Pero el dato que nos da el sensor no está en º/seg (grados angulares entre segundos), que es lo que nos interesaría.

Para saber qué tipo de ángulo estamos leyendo es conveniente saber que es la sensibilidad (*Sensitivity*) de un sensor. La sensibilidad de un sensor es el menor cambio de magnitud que un sensor puede percibir, el cual se indica normalmente en

LSB/unidad (*Least Significant bit*). En caso del giroscopio nos indica cuantos º/seg equivale cada dato de salida del sensor. Según la ficha técnica del giroscopio la sensibilidad es de 14.375 LSB/ (º/seg), es decir, si obtenemos el valor de 2 en el giroscopio esto corresponderá a 2*1/14.375 º/seg.

Por tanto una vez leído el valor del giroscopio, tan solo falta multiplicar este valor por 1/14.375 para obtener el dato en º/seg, y para pasar este dato a un ángulo que al fin y al cabo es la unidad que nos interesa tan solo tenemos que multiplicar el dato por el tiempo que se tarda entre lectura y lectura el cual está fijado a 10 milisegundos.

Con todo ello, la formula final quedaría:

$$angulo(º) = X\ LSB * \frac{1\ {}^{º}/_{seg}}{14.375\ LSB} * 0.01\ seg$$

Donde X es el valor leído del giroscopio.

El ángulo obtenido mediante esta fórmula representa el ángulo que ha girado desde la última lectura. Por tanto habría que sumarle al ángulo anterior que teníamos y repitiendo este proceso podemos obtener el ángulo en cualquier momento. Cabe destacar que el giroscopio da valores respecto a los ejes X, Y y Z aunque a nosotros tan solo nos interesa el del eje X ya que es el que representa la inclinación del robot.

El giroscopio presenta dos problemas. El primero es que no sabe dónde están los “0” grados, es decir, tomará los cero grados en el estado en el que el robot se encienda. Y el segundo problema es el error de offset. El error de offset se define como la salida que se presenta cuando a la entrada se introduce el código correspondiente al cero, es decir en nuestro caso sería el valor que el giroscopio devuelve cuando no se ha movido el robot.

Este error hay que tenerlo muy en cuenta puesto que si no lo tomamos en cuenta, el ángulo que vamos leyendo, con cada lectura suma ese offset. Por tanto al de ciertas lecturas el ángulo empezará a desviarse e incrementará ese desvío a medida que el programa sigue en marcha. Para reducir ese error lo que hemos hecho es leer 5000 veces el giroscopio estando este quieto, cuando el robot se pone en marcha. Por tanto calculamos 5000 valores de errores offset. Con estos valores hacemos una media y en cada lectura que hacemos del giroscopio le restamos este offset. Con ello conseguimos que dicho error perjudique lo mínimo en el ángulo obtenido.

Para que el robot calcule el ángulo, no usamos solo el giroscopio puesto que si solo usásemos éste, llegaría un momento que el error acumulado nos haría que el robot cayese. Por ello usamos también el acelerómetro.

El acelerómetro es otro sensor de la IMU que usamos para calcular el ángulo. Este sensor sirve para medir la aceleración o cambio de velocidad y convertirlo en una señal eléctrica que podamos interpretar.

Igual que con el giroscopio una vez leamos el valor que recibimos del acelerómetro, este nos dará un valor que tenemos que interpretar. Para ello debemos conocer la sensibilidad (*Sensitivity*) del sensor. Mirando el manual podemos encontrar que la sensibilidad del acelerómetro es de 256 LSB/g, es decir, que el dispositivo puede leer 256 valores dentro de una unidad de la aceleración de la gravedad (9.81 m/s^2). Por tanto utilizando la siguiente formula podríamos descubrir la aceleración que recibe un eje:

$$aceleracion(m/s^2) = X\ LSB * \frac{1\ {}^{m}/_{s^2}}{256\ LSB}$$

**Donde X es el valor que el acelerómetro lee en uno de los ejes*

A diferencia del giroscopio en el que tan solo con leer el valor del eje X podíamos conocer el ángulo de giro, en el acelerómetro necesitamos dos valores (los correspondientes a los ejes X y Z) y a partir de estos calcular el ángulo. Para ello nos apoyaremos en la figura 16.

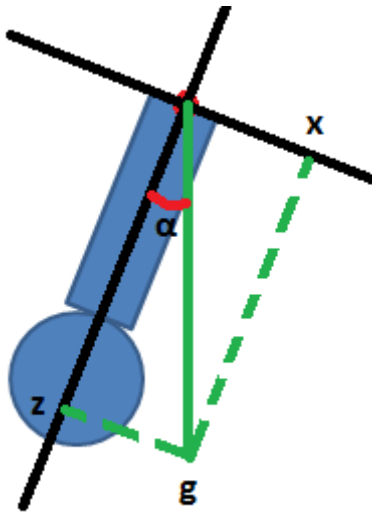


Figura 17: Efecto de la fuerza de la gravedad sobre el robot.

En este dibujo el robot representa la figura azul, el punto donde se cruzan las rectas negras representa donde se encuentra el acelerómetro, las rectas negras los ejes X y Z (el eje y seria perpendicular a estos) y la recta verde continua la fuerza que la gravedad aplica en el acelerómetro (la línea verde discontinua corresponde a la fuerza

de la gravedad distribuida en los ejes X y Z). Teniendo en cuenta que desde el acelerómetro podemos medir las fuerzas que la gravedad aplica sobre los ejes X y Z, resulta fácil calcular el ángulo α, puesto que este ángulo representa lo que el robot ha girado desde la posición vertical o bien es lo que hace falta que gire para llegar a la posición perpendicular respecto al suelo.

Para calcular dicho ángulo α debemos calcular el arco tangente de X entre Y, es decir:

$$\alpha(º) = \arctan(X/Y)$$

**Donde X e Y son los valores medidos por el acelerómetro en esos mismos ejes*

El ángulo así calculado no es muy exacto. Este es el mayor problema del acelerómetro, que el ángulo calculado no es exacto y además es extremadamente inestable, tanto que muchas de las veces no se corresponde con la realidad. Para ello, podemos ver la siguiente grafica en donde se ilustra el cálculo del acelerómetro (en rojo) respecto al valor real (azul):

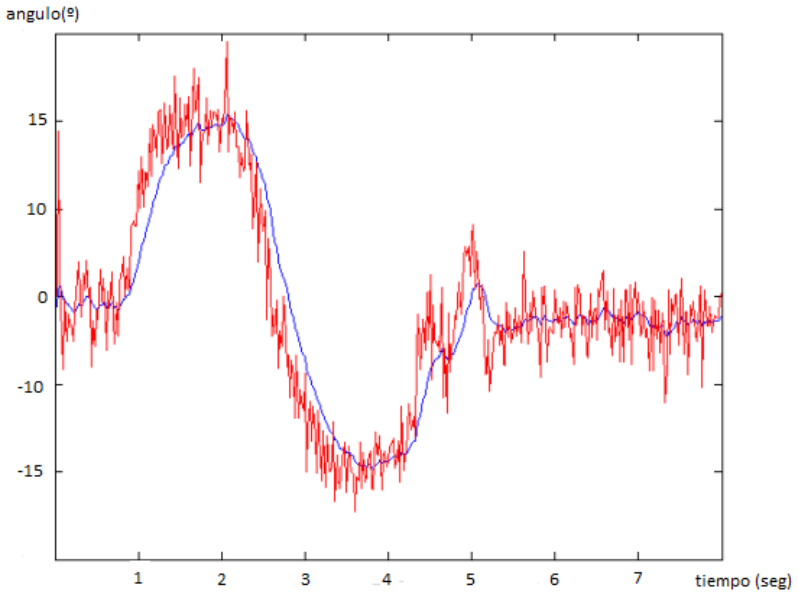


Figura 18: Relación entre los ángulos obtenidos por el acelerómetro y el ángulo real.

Aquí podemos observar que el ángulo medido a veces tiene más de 10º respecto a la realidad por lo que hay que tener mucho cuidado al utilizar este ángulo.

Dado que el ángulo obtenido del acelerómetro no es muy fiable, y el del giroscopio tiene varios errores, surge la idea de que hay que utilizar algún sistema para fusionar estos datos convenientemente, los filtros digitales.

Los filtros digitales son un sistema que, dependiendo de las variaciones de las señales de entrada en el tiempo y amplitud, realizan un procesamiento matemático

sobre dicha señal, obteniéndose en la salida el resultado del procesamiento matemático o la señal de salida.

En un comienzo, usamos los filtros más sencillos, en los cuales tan solo cogemos una parte de la señal que cada sensor calcula, por ejemplo:

$$y = anguloGiro * 0.95 + anguloAcel * 0.05$$

**Donde y es la señal de salida, anguloGiro es el ángulo calculado por el giroscopio y anguloAccel el ángulo calculado por el acelerómetro*

En el filtro digital que podemos observar aquí arriba damos más importancia al valor calculado por el giroscopio que al calculado por el acelerómetro. Esto se debe a que la señal del giroscopio es mucho más precisa que la del acelerómetro, que como hemos visto varía mucho, pero como al mismo tiempo el acelerómetro no tiene los errores que el giroscopio posee tampoco se puede excluir del todo.

Aun así, el comportamiento que obtenemos no es del todo el deseado, ya que el ángulo calculado con este filtro no es tampoco muy fiel a la realidad a pesar de que sea mejor que el de los sensores independientemente. Para mejorar este filtro se decide implementar un tipo de filtro muy común en este tipo de problemas, el filtro Kalman.

4.2. Filtro de Kalman

Desde su introducción en 1.960 el Filtro de Kalman ha llegado a ser un componente integral dentro de miles de sistemas de navegación tanto militares como civiles. Este algoritmo digital recursivo, aparentemente simple, es el favorito para integrar convenientemente o fusionar los datos de los sensores de navegación para alcanzar un rendimiento óptimo de todo el sistema.

En nuestro proyecto no entramos demasiado a fondo con el filtro de Kalman en cuanto a las fórmulas, debido a que este ya está bastante trabajado e implementado por mucha gente. Además, si quisiéramos entrar a fondo con el filtro de Kalman supondría un trabajo enorme, ya que la teoría del filtro de Kalman abarca muchos temas de las matemáticas. Se dispone de información más específica sobre la teoría del filtro de Kalman en el apéndice. Esta información no es necesaria para

implementar el filtro de Kalman debido a que ya existen muchísimas implementaciones del algoritmo en la red.

Para implementar este algoritmo hemos utilizado un modelo extraído del código que tenía el robot balancín del que partimos, en su microprocesador *Freescale MC9S12XS128MAL*. A partir de este modelo, lo hemos adaptado a nuestro robot para que se adecuase sus necesidades.

Una vez implementado este filtro notamos una gran mejoría en cuanto al ángulo de salida respecto al real, permitiendo que nuestro control del equilibrio del robot mejorase enormemente.

Gracias a la implementación del filtro de Kalman, logramos corregir los errores que el acelerómetro y el giroscopio tienen de por si como ya se ha comentado. Utiliza datos de ambos para conseguir el ángulo "0", con ello logra que el error del giroscopio al calcular el "0" se solucionará. Además, también se elimina parte del error del acelerómetro ya que al fusionarse los datos le da una mayor importancia al ángulo del giroscopio y por tanto, el ángulo logrado no oscila alrededor de un valor como lo hace si lo calculásemos sólo con el acelerómetro.

4.3. Control del equilibrio

En este apartado se detalla la manera en la que se consigue que el robot se mantenga en equilibrio. Para ello, hay tres cosas importantes. Los controladores de los motores: que entregan la potencia y el sentido en el que tienen que girar las ruedas, el control de los motores mediante PWM - *pulse-width modulation* (modulación por ancho de pulso) y el control PID: que se encarga de calcular el valor del ancho del pulso para el PWM).

4.3.1. Control PWM

El PWM, es una técnica que utilizamos para regular la velocidad de giro de los motores eléctricos. Mantiene el par motor constante y no supone un desaprovechamiento de la energía eléctrica. Aunque aparte de esto, mediante la técnica del PWM también podemos transmitir información analógica. Esto se suele utilizar junto a otros elementos para implementar un conversor ADC.

A diferencia con otros sistemas para el control de los motores, el PWM permite el máximo aprovechamiento de los motores. Los otros sistemas regulan la tensión eléctrica con lo que se disminuye el par motor. Sin embargo, en el PWM, la tensión eléctrica siempre es la misma, lo único que cambia es el tiempo en el que se está ejerciendo dicha energía. Para que se entienda mejor, con un sistema de control que regula la tensión, si queremos que el motor vaya despacio, reduciríamos la tensión eléctrica. Esto haría que el par motor se redujese y que no tuviera a penas fuerza para mover el motor. Mientras que usando la técnica del PWM, al motor siempre se le entregaría la misma tensión eléctrica por lo que el par motor siempre será el mismo. Si queremos que el motor vaya despacio simplemente se modificará el tiempo del ancho del pulso.

En el proyecto se gestionan los motores de manera similar, solo que se utiliza el módulo PWM implementado que ofrece el PIC24H, en el que tenemos que configurar el *Timer* que vamos a utilizar para generar los ciclos.

En el siguiente fragmento de código se puede ver la configuración de inicio del *Timer* que utilizamos para el módulo PWM.

```
    OC1CONbits.OCTSEL = 0; // Selección del Timer 2 como timer de
referencia
     T2CON = 0; // Reset del Timer
    // T2CONbits.TCS = 0; // Selecciona reloj de CPU (Fosc/2)
    // T2CONbits.TGATE = 0; // Modo Gated deshabilitado
     T2CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
     //T2CONbits.TCKPS0=0;
     //T2CONbits.TCKPS1=1;
     TMR2 = 0x00; // Borra Timer
     PR2 = PRT_PWM; // Determina periodo
//     PRT_PWM = 8000; // Determina periodo
     _T2IP = 0x01; // Define nivel de prioridad del Timer 2
     _T2IF = 0; // Borra el flag de interrupción Timer 2
     _T2IE = 1; // Habilita interrupción Timer 2
     T2CONbits.TON = 1; // Activa Timer
```

La configuración de inicio del módulo PWM es la siguiente:

```
    // Inicialización del módulo de Salida Comparada
    Motor_A2_Tris=0; //_TRISB10
    Motor_B1_Tris=0; //_TRISB11
    Motor_A1_Tris=0; //_TRISB12
    Motor_B2_Tris=0; //_TRISB13

    OC1CON = 0; // Reset del módulo
    OC2CON = 0;
    OC3CON = 0;
    OC4CON = 0;
    //OC1CONbits.OCTSEL = 0; // Selección del Timer 2 como timer de
referencia
    OC2CONbits.OCTSEL = 0;
```

```
    OC1R = 0;  // Define el duty cycle (ciclo de trabajo) para el
primer pulso PWM
    OC2R = 0;
    OC3R = 0;
    OC4R = 0;

    OC1RS = 0; // Define el duty cycle para el segundo pulso PWM
    OC2RS = 0;
    OC3RS = 0;
    OC4RS = 0;

    OC1CONbits.OCM = 0b110; // Selecciona modo PWM sin protección
    OC2CONbits.OCM = 0b110;
    OC3CONbits.OCM = 0b110;
    OC4CONbits.OCM = 0b110;
```

Tanto el OC1, OC2, OC3 y OC4 (módulos correspondientes al PWM implementados por el PIC) se configuran para funcionar con el *Timer 2* del microcontrolador. Si nos fijamos como está configurado el *Timer 2* vemos que cada 8000 pulsos de reloj se reiniciará es decir que tiene un periodo de 8000 pulsos por segundo.

```
     PR2 = PRT_PWM; // Determina periodo
//     PRT_PWM = 8000; // Determina periodo
```

Por lo tanto si queremos dar a los motores la máxima potencia y definir un ciclo de trabajo del 100% a las variables OC1RS, OC2RS, OC3RS y OC4RS que son las que definen el ciclo de trabajo deberíamos asignarles el valor 8000, sin embargo si queremos que el ciclo de trabajo sea del 50% (a la mitad de potencia) deberíamos asignar el valor 4000.

Resumiendo, una vez configurado tanto el Timer como los módulos PWM, basta con asignar a la variable un valor para el ciclo de trabajo, para que el PIC internamente y de forma transparente al usuario (con el módulo implementado del PWM) gestione los anchos de pulso con el ciclo de trabajo que hemos definido.

Para el caso de nuestros motores no será tan sencillo como definir un simple ciclo de trabajo estático, más bien serán asignaciones dinámicas que estarán en constante cambio, dependiendo del ángulo en el que se encuentre el robot (información extraída del IMU), para así poder mantenerlo en equilibrio. Para controlar esto existen distintos sistemas o controladores, en nuestro caso hemos utilizado un control PID (Proporcional Integrador y Derivativo), que va asignando distintos ciclos de trabajo para dar mayor o menor potencia a los motores y conseguir que el robot se mantenga en todo momento en posición vertical, dependiendo del ángulo del robot (mayor a “0” o menor a “0”) cambiara el sentido en el que giran las ruedas y el ancho de pulso.

El código que se muestra a continuación es un fragmento de código encargado de gestionar el ancho de pulso del PWM y la dirección en el que giran las ruedas.

```
if(outputPID >0){

    if(dir==1){
        dir=0;
        Integral=0.0;
    }
    // Motor 1
    OC1RS=abs(outputPID);
    OC3RS=0
    // Motor 2
    OC2RS=abs(outputPID);
    OC4RS=0;
}
else if (outputPID <0) {
   if(dir==0){
        dir=1;
        Integral=0.0;
    }
    // Motor 1
    OC1RS=0;
    OC3RS=abs(outputPID);
    // Motor 2
    OC2RS=0;
    OC4RS=abs(outputPID);
}
```

Como se ve en el código, el valor “outputPID” es el que se le asigna a los OCXRS. Este valor ha sido calculado previamente por el controlador PID. Como ya hemos dicho antes, cada motor es controlado por dos señales de PWM, en este caso OC1RS y OC3RS controlan un motor mientras que OC2RS y OC4RS controlan el otro motor, por lo que si el “outputPID” da un valor mayor a “0” (el robot está inclinado hacia un lado) la asignación del ancho del pulso es:

```
    // Motor 1
    OC1RS=abs(outputPID);
    OC3RS=0;
    // Motor 2
    OC2RS=abs(outputPID);
    OC4RS=0;
```

Esto hace que ambos motores giren en el mismo sentido y a la misma velocidad.

Por el contrario si la variable “outputPID” tiene un valor menor a “0” (el robot está inclinado hacia el otro lado) la asignación del ancho del pulso es:

```
    // Motor 1
    OC1RS=0;
    OC3RS=abs(outputPID);
```

```
            // Motor 2
            OC2RS=0;
            OC4RS=abs(outputPID);
```

4.3.2. Controladores de los motores

Para controlar los motores hemos utilizado dos controladores *Simple H robot power. A* cada uno de ellos está conectado un motor y gracias a estos controladores controlamos la velocidad y el sentido de cada motor.

Cada *Simple H robot power* recibe dos señales PWM del microcontrolador. Para uno de los motores, los pines OC1 y OC3 del microcontrolador están conectados a los pines PA y PB del controlador del motor, y para el otro motor los pines OC2 y OC4 del microcontrolador están conectados a los pines PA y PB del otro controlador del motor. La salida del *Simple H robot power* va conectado al motor.

Para que el motor se mueva, una de las dos señales debe estar activa (estado lógico alto), y la otra en baja. Para que esté parado ninguna de las dos señales debe estar activa. El sentido de giro del motor se controla mediante las dos señales de los pines PA y PB. Para que gire hacia un lado, la señal PA debe estar en estado lógico alto y la señal PB en estado lógico bajo y para que gire hacia el otro lado se intercambian los valores de PA y PB.

4.3.3. Controlador PID (PROPORCIONAL INTEGRAL DERIVATIVO)

Para conseguir que el robot se mantenga en equilibrio, hemos utilizado un controlador PID.

Un controlador PID es un mecanismo de control genérico sobre una realimentación de bucle cerrado. El funcionamiento del controlador PID es como sigue: al sistema, es decir, al controlador, le entra un error calculado a partir de la salida deseada y la salida obtenida. Ese error es utilizado como entrada en el sistema que queremos controlar. El controlador intenta minimizar el error ajustando la entrada del sistema.

Para el correcto funcionamiento de un controlador PID que regule un proceso o sistema se necesita, al menos:

- Un sensor, que determine el estado del sistema (termómetro, manómetro, acelerómetro…).
- Un controlador, que genere la señal que gobierna al actuador.

- Un actuador, que modifique al sistema de manera controlada (resistencia eléctrica, motor, válvula, bomba…).

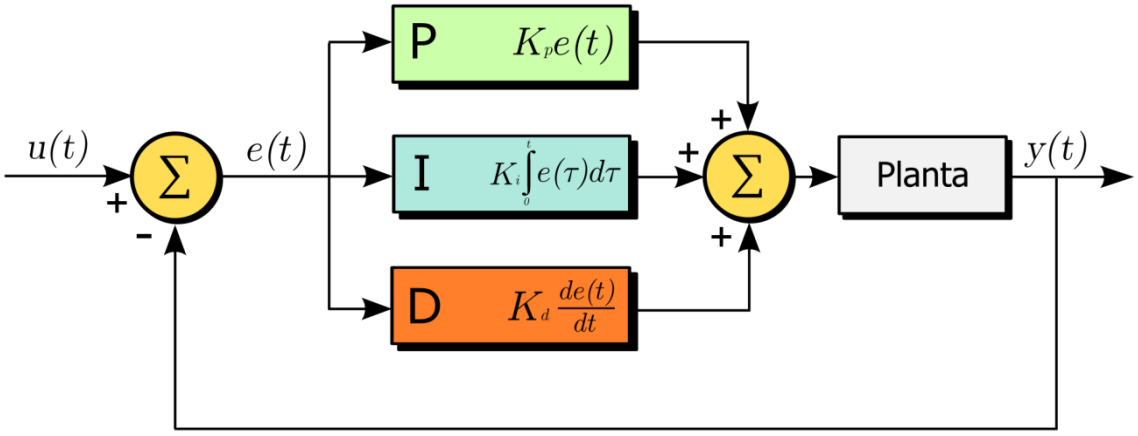


Figura 19: Diagrama de bloques de un controlador PID de lazo cerrado.

u (t): Entrada (Salida que se desea obtener).
y(t): Salida.
e(t): Error.
Planta: Sistema.

El controlador PID se ajusta en base a tres parámetros: el proporcional, el integral y el derivativo. Dependiendo la modalidad del controlador alguno de estos parámetros pueden valer "0". Podemos crear un controlador proporcional P, en el que el valor integral y el derivativo son "0", o un controlador proporcional e integral PI, en el que el valor derivativo es "0", o también un controlador proporcional derivativo PD en el que el factor integral es "0". Finalmente el que hemos utilizado para el proyecto es el controlador PID en el que los tres parámetros son distintos de cero. Cada uno de estos parámetros influye sobre alguna característica de la salida: tiempo de establecimiento, sobre oscilación, etc. También influye, en mayor o menor medida sobre las demás, por lo que por mucho que ajustemos los parámetros no encontraríamos un PID que redujera el tiempo de establecimiento a "0", la sobre oscilación a "0", el error a "0", etc… Se trata más de ajustarlo a un término medio cumpliendo las especificaciones requeridas.

Respuesta proporcional

La respuesta proporcional es la base de los tres modos de control, si los otros dos, integral y derivativo están presentes, éstos se suman a la respuesta proporcional. Se obtiene una respuesta proporcional, es decir se le aplica un factor multiplicador a la entrada del controlador. A este múltiplo se le llama "ganancia" del controlador.

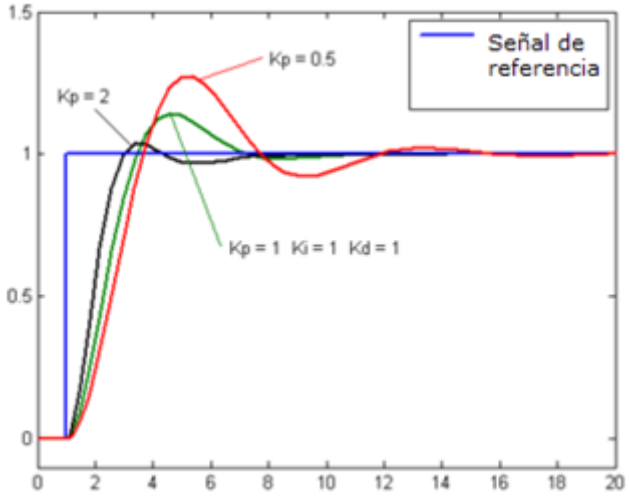


Figura 20: Respuesta proporcional.

Respuesta integral

La respuesta integral da una respuesta proporcional a la integral del error. Esta acción elimina el error en régimen estacionario, provocado por el modo proporcional. Por contra, genera un mayor tiempo de establecimiento, una respuesta más lenta y el periodo de oscilación es mayor que en el caso de la acción proporcional.

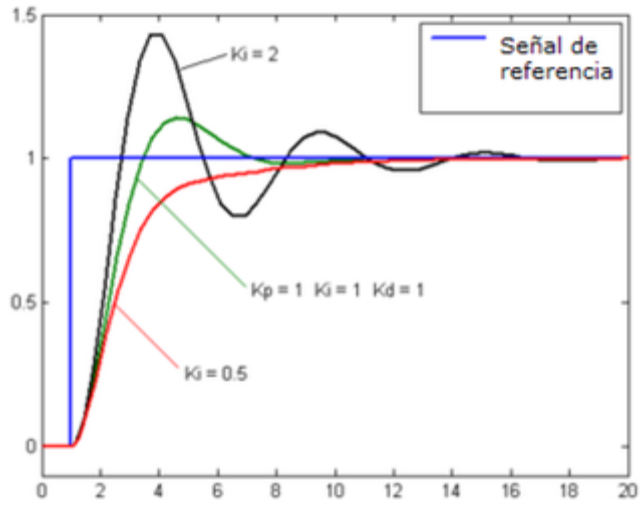


Figura 21: Respuesta integral.

Respuesta derivativa

La acción derivativa da una respuesta proporcional a la derivada del error o dicho de otra forma, velocidad de cambio del error. Añadiendo esta acción de control a las anteriores se disminuye el exceso de sobre oscilaciones.

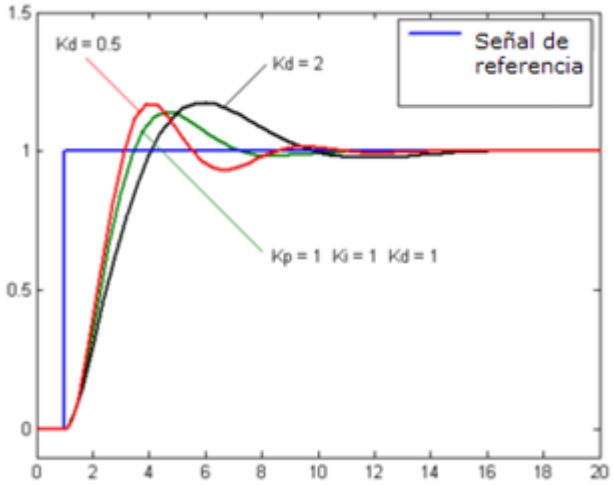


Figura 22: Respuesta derivativa.

En resumen, un controlador PID, lo que hace es: a partir de una señal de referencia, intenta ajustar la salida a la señal de referencia aplicando una función al error obtenido, entre la salida y la señal de referencia. Esa función se ajusta mediante tres parámetros P, I, D (Kp, Ki, Kd) y en base a esto, el sistema obtiene una respuesta mejor o peor.

$$\mathrm{y(t)} = \mathrm{MV(t)} = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de}{dt}$$

Para ajustar los parámetros existen distintos métodos: el método de Ziegler-Nichols, el método de Cohen-Coon, el método que sigue unas reglas heurísticas de ajuste y por último ajustándolo a ojo, que viene a ser algo parecido a seguir las reglas heurísticas de ajuste.

En nuestra solución, primero ajustamos el Kp, de forma que el robot se mantenga más o menos en equilibrio. Una vez encontrado ese valor, ajustamos el Kd, para que las oscilaciones disminuyan y finalmente ajustamos el Ki, para eliminar el error en régimen estacionario.

Tras diversas pruebas, los valores Kp, Kd y Ki con los que mejor respuesta hemos obtenido han sido los siguientes:

Kp = 330.0

Ki = 25.25

Kd = 1150.0

La señal de referencia para nuestro controlador PID es el ángulo de inclinación del robot, que idealmente debe tener siempre el valor 0º. Es decir, el robot se debe

encontrar en posición vertical. En caso de que se incline hacia un lado, tendrá un ángulo mayor a 0º y en caso de que se incline hacia el otro tendrá un ángulo menor a 0º. Como se ve en la siguiente figura el objetivo es que el robot mantenga en todo momento, respecto al eje vertical (Z), un ángulo (α) de 0º.

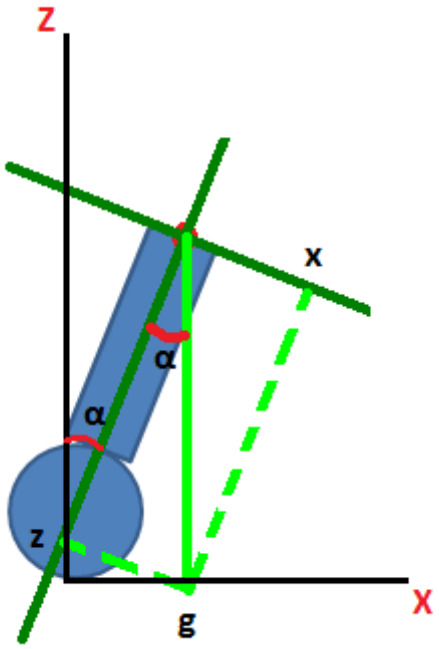


Figura 23: Fuerza de la gravedad sobre el robot, el ángulo α representa el ángulo que necesita corregirse para que el robot se quede en posición vertical.

La entrada del controlador PID, es el propio ángulo del robot, que previamente ha sido procesado por el filtro del Kalman, y el valor de referencia objetivo o *setpoint* es el valor "0" (0º).

El control PID que hemos implementando funciona de la siguiente manera:

1- Se obtiene como entradas la señal de referencia, y la salida obtenida anteriormente del sistema, que en este caso es el ángulo que tiene el robot.
2- Se calcula el error entre el valor de referencia y el ángulo y se aplica el control PID.
3- Se obtiene como salida un valor, ese valor es el ciclo de trabajo del PWM para los motores del robot. Es decir, se regula la velocidad de las ruedas del robot en base al controlador PID, y se le aplica a los motores para corregir el ángulo.
4- Se vuelve al paso 1.

La implementación es la siguiente:

```
void calcularPID(){
    error=anguloFiltro-SETPOINT;
    Proportional=Kp*error;
    Derivative=error-prev_error;

    if ( (abs(error)<=MARGEN_INTEGRADOR))
    {
      Integral+=Ki*error;
        if (Integral>LIMITE_PWM)Integral=LIMITE_PWM;
        else if(Integral < -LIMITE_PWM) Integral= -LIMITE_PWM;
    }
    //-----------------------------------------------------------------
    k_k=Proportional+ Integral +Kd * Derivative;

    //Limites de la salida en función de la configuración del timer
del PWM, para nuestro caso es 8000.
    if (k_k <-LIMITE_PWM)
    {
      k_k=-LIMITE_PWM;
    }else if (k_k>=LIMITE_PWM)
    {
      k_k=LIMITE_PWM;
    }

    auxoutputPID=(int)k_k;
    outputPID=auxoutputPID;
    prev_error=error;
}
```

La función *calcularPID(),* se la llama desde la rutina de atención de interrupción del *timer7* que tiene la siguiente configuración:

```
void Inic_Timer_7(void) //Función de inicialización del TIMER7
{
    // Inicializar y habilitar TIMER7
    T7CONbits.TON = 0; // Disable Timer
    T7CONbits.TCS = 0; // Select internal instruction cycle clock
    T7CONbits.TGATE = 0; // Disable Gated Timer mode
    T7CONbits.TCKPS = 0b10; // Selecionar Prescaler: 1:64
    TMR7 = 0x00; // Clear timer register
    PR7 = 3125; // Periodo Giroscopio: 10ms = 6250
//  _T7IP = 0x01; // Set Timer 7 Interrupt Priority Level
    _T7IP = 0x05; // Set Timer 7 Interrupt Priority Level
    _T7IF = 0; // Clear Timer 7 Interrupt Flag
    _T7IE = 1; // Enable Timer 7 interrupt
    T7CONbits.TON = 1; // Start Timer

        estado_int=0;
        t1=clock();
        errorFlag=1;
}
```

Por lo tanto a esta función se la llama cada 10 milisegundos.

4.3.4. Pruebas realizadas

Para conseguir que el robot se mantenga en equilibrio hemos hecho muchísimos ajustes para el controlador PID que han consistido en prueba y error. Finalmente hemos conseguido que el robot mantenga el equilibrio de forma bastante aceptable. Podemos diferenciar dos tipos de pruebas, **pruebas de equilibrio** y **pruebas de equilibrio frente a empujones**.

Las pruebas de equilibrio han consistido en intentar que el robot se mantuviera estable en todo momento por sus propios medios.
Las pruebas de equilibrio frente a empujones trataban de ajustar la respuesta del robot frente a fuerzas externas como lo puede ser un empujón. El objetivo de estas pruebas es ver la respuesta que tiene el robot cuando se le somete a alguna fuerza externa.

Ajustando el controlador PID, los valores Kp, Kd y Ki acabamos encontrando la combinación que más se amoldaba a nuestros objetivos.

El resultado final se puede ver en el siguiente video.

URL del video en YouTube: https://www.youtube.com/watch?v=cUhwLEih11c

4.4. Control Bluetooth

Para el control remoto he usado una placa *Explorer 16* que dispone de 4 botones que se han utilizado para dar órdenes al robot. Se ha dotado de un módulo Bluetooth RN4020 tanto a la *Explorer 16* del robot como a la *Explorer 16* que se ha usado para el control remoto. De esta manera ambas partes se han sincronizado mediante el programa implementado y se han podido intercomunicar mensajes de control.

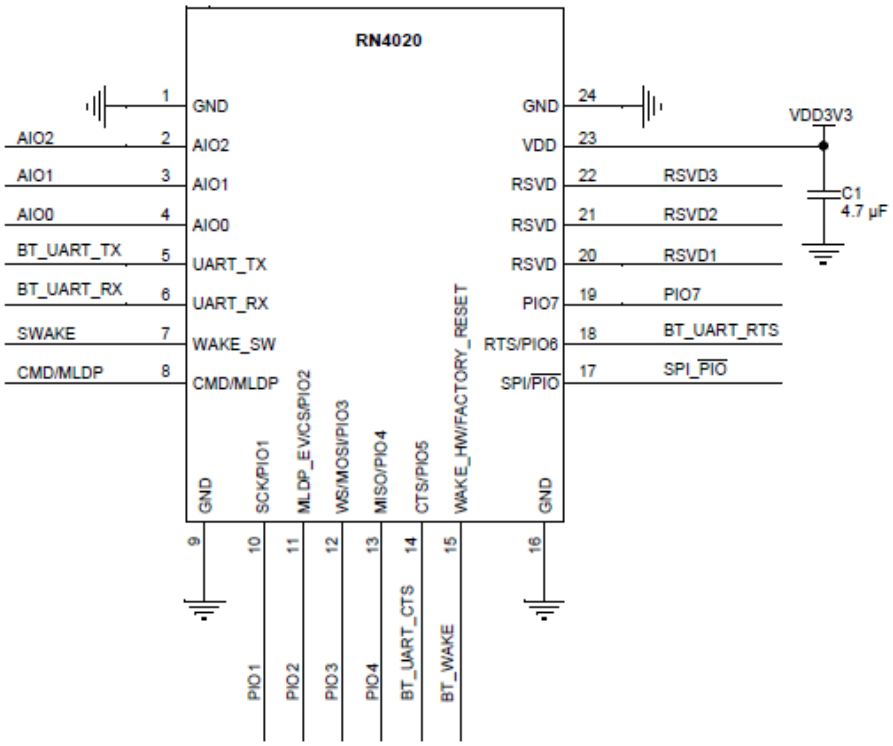


Figura 24: Diagrama de pines del módulo Bluetooth RN4020.

Existen dos modos en el módulo Bluetooth, el modo de comandos (CMD) y el de datos (MLDP). En el modo de comandos, se transmiten comandos por la UART hacia el módulo Bluetooth unos comandos que sirven para configurar el módulo Bluetooth a nuestras necesidades mientras que en el modo de datos, se envían datos desde la UART al módulo Bluetooth y luego éste lo retransmite hacia el otro dispositivo con el que está conectado. Estos datos que se envían ya son propios de la aplicación que hemos diseñado.

Para ponerlo en modo comandos la pata CMD/MLDP tiene que estar a 1 (*high*) pero si lo que queremos es ponerlo en el modo datos esta pata tiene que estar a "0" (*low*).

Para conectar ambos módulos vía Bluetooth, he utilizado los códigos de muestra proporcionados por Microchip y haciendo algunas modificaciones (en principio el código está implementado para el microcontrolador PIC24F y no para el PIC24H con el que se ha desarrollado el proyecto) he conseguido comunicar y conectar ambos módulos Bluetooth.

El módulo Bluetooth correspondiente a la parte del robot se ha configurado como esclavo, mientras que el otro módulo se ha configurado como maestro.

A continuación se describe la secuencia de comandos y procesos que se utilizan para cada configuración, se podría ver como una máquina de estados.

4.4.1. Configuración de los módulos Bluetooth

En este apartado se analiza la forma en la que están configurados los módulos Bluetooth, para poder configurarlos, como ya he dicho más arriba, la pata CMD/MLDP tiene que estar a 1 (*high*).

Para configurar ambos módulos el procedimiento es el mismo solo que la secuencia de comandos es distinta.

Existe en el programa un *array* en el que se define la secuencia de comandos que se tienen que ir ejecutando para la configuración "*gCommands_ToBeExecuted[]"*.

Gracias a una máquina de estados el procedimiento para la configuración se lleva acabo de la siguiente manera:

1. Si aún no hemos llegado al final del *array* "*gCommands_ToBeExecuted[]"* se obtiene del *array* el siguiente comando a ejecutar.
2. Se envía por la UART el comando para ejecutarlo.
3. Obtenemos la respuesta de la UART.
4. Obtenemos el último comando que hemos ejecutado.
5. En una secuencia de condiciones (if,else if, else if, …) en el que la expresión evaluada es el último comando ejecutado, entramos en la condición correspondiente al último comando ejecutado.
6. Dentro de la condición anterior, comprobamos la respuesta obtenida del módulo Bluetooth por la UART en el paso 2, que normalmente son ACKs, y si la respuesta es la que esperábamos volvemos al paso 1. Si no, se sigue intentando enviar el comando.

4.4.2. Secuencia de comandos para el esclavo (*peripheral*):

1- **"$$"** (*Command Mode)*:

No se trata de ningún comando en concreto, si no que más bien es para inicializar la comunicación con el módulo Bluetooth. Este primer paso se utiliza para comprobar que el pin correspondiente a CMD/MLDP este a 1 y que el Bluetooth está en modo Comandos.

2- **"SF,1"** (*Factory Reset*):

Este comando restablece las configuraciones a los valores predeterminados cuando se reinicie el módulo. Los parámetros para este comando pueden ser '1' y '2'.

Cuando el parámetro de entrada es '1', la mayoría de los ajustes se restaurarán a los valores por defecto de fábrica, pero algunos ajustes, como el nombre del dispositivo, información del dispositivo y servicios privados, permanecerán igual. Cuando el parámetro de entrada es '2', todos los parámetros son restaurados a los valores de fábrica

3- **"V"** (*Get Version Command)*:

Con este comando obtenemos la versión del firmware.

4- **"GR"** *(GR Command)*:

Devuelve un valor hexadecimal de 32 bits que representa la configuración de algunas de las funciones RN4020.

5- **"GN"** *(GN Command)*:

Devuelve la cadena ASCII del nombre del dispositivo.

6- **"SR,32000000"** *(SR Command)*:

Este comando establece las características compatibles de módulo RN4020 actual. La entrada es un parámetro que es un mapa de bits de 32 bits que indica la configuración del módulo. Después de cambiar las características, es necesario el reinicio para hacer efectivos los cambios.

En este caso 32000000, establece como dispositivo periférico con el aviso automático, y soporte para MLDP.

Nota: Para información más detallada acerca del mapa de bits o de las distintas configuraciones es recomendable acudir al manual que proporciona Microchip.

7- **"SN,MarkelPi"** *(CH Dname Command)*:

Este comando establece el nombre del dispositivo, en este caso el nombre del dispositivo está configurado como MarkelPi.

8- **"R,1"** *(Reboot Command)*:

Este comando fuerza un reinicio del dispositivo completo. Tiene el parámetro obligatorio '1'. Después se reinicia el módulo RN4020.

9- **""** *(Pair Request)*:

En este estado simplemente se espera a que el dispositivo se conecte, y depende de si ha tenido éxito o no la sincronización el dispositivo quedará finalmente conectado o no.

4.4.3. Secuencia de comandos para el maestro (*central*):

1- **“”** (*Command Mode)*
2- **“SF,1”** (*Factory Reset*)
3- **“V”** (*Get Version Command)*
4- **“GR”** *(GR Command)*
5- **“GN”** *(GN Command)*
6- **“SR,92000000”** *(SR Command)*

Este comando establece las características compatibles de módulo RN4020 actual. La entrada es un parámetro que es un mapa de bits de 32 bits que indica la configuración del módulo. Después de cambiar las características, es necesario el reinicio para hacer efectivos los cambios.

En este caso 92000000 , establece como dispositivo central con el aviso automático, y soporte para MLDP.

Nota: Para información más detallada acerca del mapa de bits o de las distintas configuraciones es recomendable acudir al manual que proporciona Microchip.

7- **“R,1”** *(Reboot Command)*
8- **“F”** *(Scan Request)*:

Este comando sólo está disponible para un dispositivo en modo central (maestro). Se utiliza para consultar los dispositivos periféricos antes de establecer una conexión.

El comando F tiene dos parámetros “F,<hex16>,<hex16>”.

Si no se proporciona ningún parámetro, el comando "F " inicia el proceso de escaneo activo por defecto con un intervalo de exploración de 375 milisegundos y una ventana de exploración de 250 milisegundos. El usuario tiene la opción de especificar el intervalo de exploración y la ventana de exploración primer y segundo parámetro, respectivamente, como un valor hexadecimal de 16 bits en milisegundos.

El comando “F” se detendrá después de que expire la ventana de tiempo de espera. El formato del resultado del análisis es:

<BTADDR>, <PRIVATE>, <BTName>, <UUID>, <RSSI>

<BTADDR>: Dirección MAC de 48bits.

<PRIVATE>: 1 bit que especifica el tipo de dirección (publica/privada).

<BTName>: Nombre opcional del dispositivo.

<UUID>: Identificador único universal (UUID).

<RSSI>: 8 bits que especifican el valor RSSI (la calidad de la señal).

9- **"X"** *(Stop Scan Request)*:

Este comando hace que se detenga el proceso de escaneo.

10- **"E,0,@MAC"** *(Establish Conn. Request)*:

El comando "E" inicia el proceso para establecer una conexión con un periférico.

Si el dispositivo central no está unido con el periférico, son necesarios dos parámetros de entrada para establecer la conexión con un dispositivo periférico. El primer parámetro es el tipo de dirección MAC, y el segundo parámetro es la dirección MAC del dispositivo periférico (esclavo). El tipo de dirección MAC es o bien "0" para la dirección pública o "1" para una dirección aleatoria. El tipo de dirección estará disponible en el resultado del escaneo del paso 8 utilizando el comando "F". El segundo parámetro es una dirección MAC de 6 bytes, que también está disponible como resultado de usar el comando "F".

11- **""** *(Pair Request)*

4.4.4. Protocolo para el control

Para controlar el robot remotamente tenemos 4 opciones, hacer que el robot ande hacia adelante, que ande hacia atrás, que gire a la izquierda o que gire a la derecha, por lo que aprovechando los 4 botones que trae la placa *Explorer 16* asignare a cada botón una función distinta.

He definido una seria de *tags* que serán los que forman el protocolo del programa para el control remoto.

TAG	**ASCII**	**Descripción**
ePERI_GO_FORDWARD	S3GF	Este comando se envía al apretar el botón S3 de la placa *Explorer 16,* cuando el robot recibe este *tag* dará potencia extra a los dos motores para que vallan hacia adelante.
ePERI_GO_BACK	S6GB	Este comando se envía al apretar el botón S6 de la placa *Explorer 16,* cuando el robot recibe este *tag* dará potencia extra a los dos motores para que vallan hacia atrás.
ePERI_TURN_RIGHT	S4TR	Este comando se envía al apretar el botón S4 de la placa *Explorer 16,* cuando el robot recibe este *tag* dará potencia extra al motor izquierdo para que gire hacia adelante mientras que al motor derecho le dará potencia extra para que gire hacia atrás. De esta manera conseguiremos que el robot de un giro hacia la derecha.
ePERI_TURN_LEFT	S5TL	Este comando se envía al apretar el botón S5 de la placa *Explorer 16,* cuando el robot recibe este *tag* dará potencia extra al motor derecho para que gire hacia adelante mientras que al motor izquierdo le dará potencia extra para que gire hacia atrás. De esta manera conseguiremos que el robot de un giro hacia la izquierda.
ePERI_STOP	STOP	Este comando se envía cuando se deja de apretar cualquiera de los botones de la *Explorer 16*, cuando el robot recibe este *tag* la potencia extra que da a cada motor es de “0”.

Nota: La representación ASCII es la que se envía por Bluetooth de un dispositivo a otro, y en base a eso el robot cuando analice la orden hará una cosa u otra.

En las siguientes imágenes se ve el funcionamiento de la aplicación, mientras uno de los botones esté pulsado se seguirá enviando el comando correspondiente a dicho botón y hasta que no se deje de pulsar el botón no se enviará el *tag* de “ePERI_STOP”, en ese momento el robot dejará de dar potencia extra a las ruedas y seguirá manteniéndose en equilibrio con el control PID para el equilibrio.

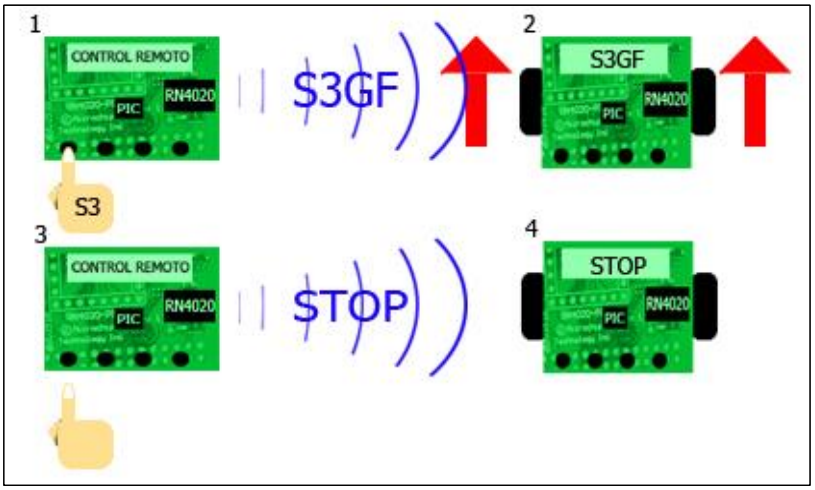


Figura 25: Envío del comando "S3GF" y "STOP".

Como se ve en la figura 24 pulsando el botón "S3" de la placa Explorer 16 (1) el módulo Bluetooth envía el comando "S3GF", cuando el módulo Bluetooth del robot (2) recibe ese comando, da potencia extra a ambas ruedas para que el robot ande hacia adelante. Tras soltar el botón "S3" (3) el módulo Bluetooth envía el comando "STOP", cuando el robot (4) recibe el comando, se le deja de dar potencia extra a las ruedas.

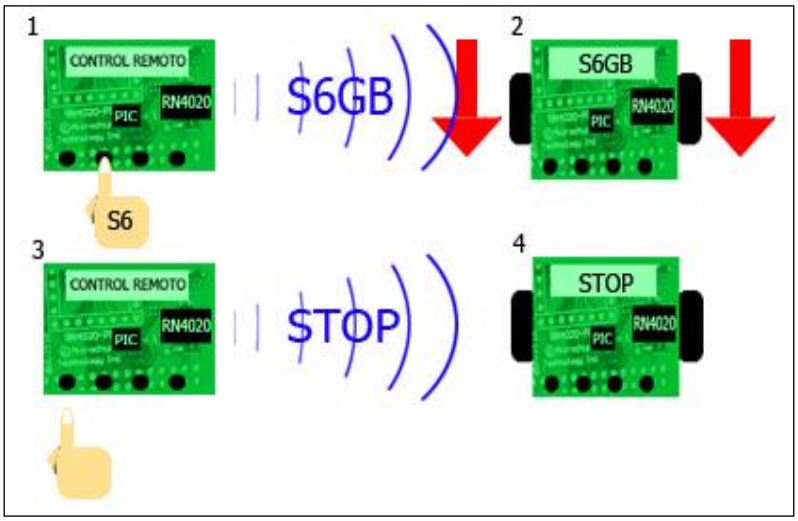


Figura 26: Envío del comando "S6GB" y "STOP".

Pulsando el botón "S6" de la placa Explorer 16 (1) el módulo Bluetooth envía el comando "S6GB", cuando el módulo Bluetooth del robot (2) recibe ese comando, da potencia extra a ambas ruedas para que el robot ande hacia atrás. Tras soltar el botón "S6" (3) el módulo Bluetooth envía el comando "STOP", cuando el robot (4) recibe el comando, se le deja de dar potencia extra a las ruedas.

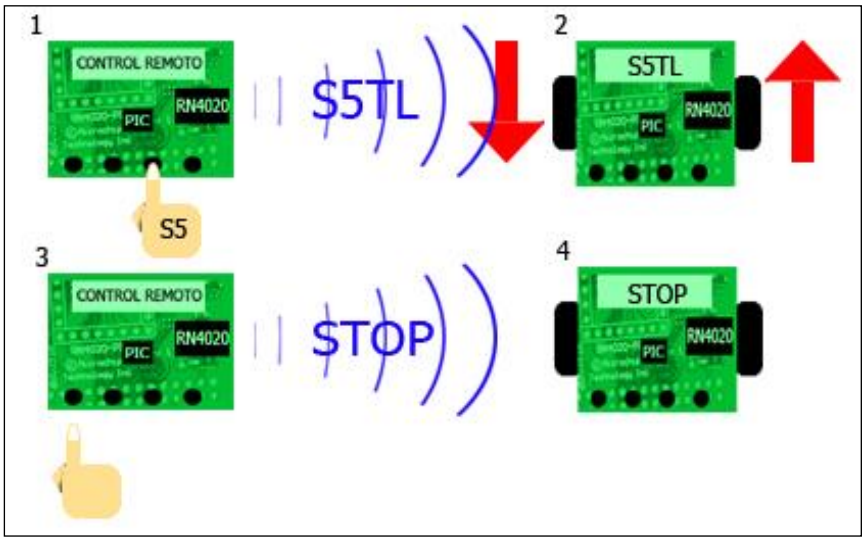


Figura 27: Envío del comando "S5TL" y "STOP".

Pulsando el botón "S5" de la placa Explorer 16 (1) el módulo Bluetooth envía el comando "S5TL", cuando el módulo Bluetooth del robot (2) recibe ese comando, da potencia extra al motor derecho para que gire hacia adelante mientras que al motor izquierdo le dará potencia extra para que gire hacia atrás. De esta manera conseguiremos que el robot de un giro hacia la izquierda. Tras soltar el botón "S5" (3) el módulo Bluetooth envía el comando "STOP", cuando el robot (4) recibe el comando, se le deja de dar potencia extra a la rueda.

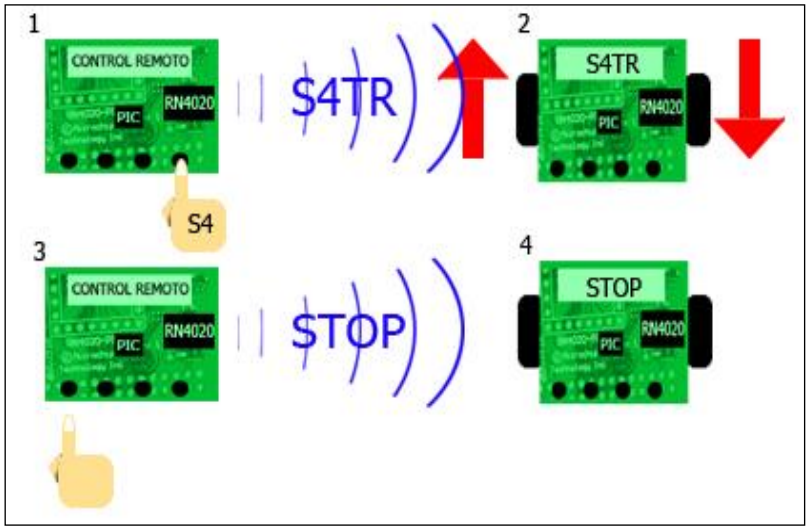


Figura 28: Envío del comando "S4TR" y "STOP".

En la figura 27 vemos que pulsando el botón "S4" de la placa Explorer 16 (1) el módulo Bluetooth envía el comando "S4TR", cuando el módulo Bluetooth del robot (2) recibe ese comando, da potencia extra al motor izquierdo para que gire hacia adelante mientras que al motor derecho le dará potencia extra para que gire hacia atrás. De

esta manera conseguiremos que el robot de un giro hacia la derecha. Tras soltar el botón "S4" (3) el módulo Bluetooth envía el comando "STOP", cuando el robot (4) recibe el comando, se le deja de dar potencia extra a la rueda.

Nota: La placa que aparece a la izquierda en las imágenes es la Explorer 16 utilizada para el control remoto. La encargada de enviar los tags al robot que está a la derecha de las imágenes.

El programa correspondiente a la parte de control es relativamente sencilla. Mientras se tiene pulsado alguno de los botones se envía el *tag* correspondiente al botón pulsado y cuando se deja de pulsar se envía el *tag* STOP que indica que no hay que darle potencia extra a los motores y por lo tanto no hay ningún tipo de control en ese momento sobre el robot aparte del control PID correspondiente al equilibrio. Debido a la falta de botones de la *Explorer 16* la velocidad tanto de giro como de avanzar o retroceder siempre va a ser fija.

El programa cargado al robot para el control Bluetooth es el encargado de darle una potencia extra a los motores del robot. Para conseguir esto, una vez que el robot analiza el *tag* que ha recibido por el otro dispositivo Bluetooth, se aplica la acción correspondiente a dicho mensaje. En todos los casos la acción, menos en el de la parada (STOP), es darle potencia a los motores, es decir, aumentar el *duty* del PWM. Existen dos variables, "velExtraIZQ" y "velExtraDER", que se sumarán al *duty* obtenido por el control PID para que el robot se mantenga en equilibrio

```
PWM_MOTOR_IZQ= duty_Control_PID + velExtraIZQ
PWM_MOTOR_DER= duty_Control_PID + velExtraDER
```

Pseudocódigo: Calculo del *duty* para el PWM de los motores usando el control remoto Bluetooth.

Las variables "velExtraIZQ" y "velExtraDER" tomarán los siguientes valores dependiendo de la acción:

Avanzar o retroceder:
velExtraIZQ = = velExtraDER >> 0
En este caso, para avanzar o retroceder lo que cambiará será el sentido en el que giran los motores.

Giro a la izquierda:
velExtraIZQ == velExtraDER
El motor derecho gira hacia adelante y el motor izquierdo hacia atrás.

Giro a la derecha:
velExtraIZQ >> velExtraDER
El motor derecho gira hacia atrás y el motor izquierdo hacia adelante.

Parar:
velExtraIZQ == velExtraDER == 0

4.4.5. Pruebas realizadas

Las pruebas que se han realizado, independientemente de las realizadas para intentar controlarlo desde un dispositivo *Android* que están descritas en el apartado de Problemas Encontrados, y han sido dos.

La primera prueba ha sido realizada desde una placa *Explorer 16* a otra placa. El objetivo de esta prueba es comprobar el funcionamiento del control Bluetooth en un entorno más "seguro" para el robot. De esta forma no se pone en peligro la integridad del robot. En ésta, el programa que se ha utilizado para cuando reciba los distintos *tags* del protocolo que hemos definido para el control Bluetooth, en vez de cambiar el *duty* de los motores, lo único que hace es encender los distintos *leds* de la placa *Explorer 16* que utiliza el robot. Así, cuando pulsamos un botón de la *Explorer 16* se enciende un *led* en la otra *Explorer 16*.

URL del video en YouTube: https://www.youtube.com/watch?v=xhY3xSIaGDY

Una vez se realizó la prueba con éxito y no se puso la integridad del robot en peligro, se pasó a hacer la segunda prueba. Para realizar esta segunda prueba lo que hay que hacer es fusionar el código del programa que comunica por Bluetooth los dos módulos para el control del robot con el programa que hace que el robot se mantenga en equilibrio, y una vez hecho esto en vez de encender un *led* lo que hacemos es modificar la variable correspondiente al *duty* de los motores.

Para empezar una vez fusionado el código hay que arreglar algún problema de compatibilidad. Tanto el programa del robot como el del Bluetooth utilizan los mismos recursos concretamente el *timer 2,* la UART y algunos pines. El programa del módulo Bluetooth de la parte del robot utilizaba el *timer 2* que colisionaba con el *timer 2* que utiliza el programa del robot para el PWM y para hacer algunos cálculos del control PID. Para evitar esta colisión, el *timer 2* que utilizaba el programa del Bluetooth se sustituyó por el *timer 1.*Este *timer* tiene las mismas características que el *timer 2*. Algo parecido pasó para el módulo de la UART, en este caso ambos programas utilizaban el módulo 2 UART del PIC. La solución fue hacer cambios en el programa del robot para que utilizara el módulo 1 UART de la PIC.

Antes de controlar el robot, puesto ya en marcha, primero se hace una prueba en la que el control de equilibrio esta desactivado. De esta manera se puede verificar que el robot responde de manera correcta según las especificaciones del protocolo de control que se ha definido.

URL del video en YouTube: https://www.youtube.com/watch?v=8t6GWoILzMc

Finalmente se pasa a la prueba definitiva. Esta prueba trata de controlar el robot por Bluetooth mientras se mantiene en equilibrio. La prueba se realizó con éxito, pero con algún problema. Mientras está en equilibrio, únicamente puede girar hacia ambos lados. Esto se debe a que cuando ordenamos al robot que ande hacia adelante o hacia atrás, éste pierde el equilibrio y el control PID lo corrige rápidamente. De esta manera no le deja ni avanzar ni retroceder. En el caso de los giros esto no pasa ya que al hacer un giro el robot no pierde el equilibrio ni inclina el robot. Debido a la falta de tiempo este problema no se ha podido solucionar.

URL del video en YouTube: https://www.youtube.com/watch?v=dg2dgAl6594

·5·

5. PROBLEMAS ENCONTRADOS

En este apartado se detallan los problemas que han ido surgiendo durante el desarrollo del proyecto, el retraso que nos han causado y la solución final que se le ha dado al problema.

Este apartado está dividido en dos partes, en la primera parte se describen los problemas encontrados en la primera fase, la fase que hemos hecho en conjunto Anartz Recalde y yo, y en la segunda parte los problemas relacionados con la segunda fase del proyecto.

Con este apartado lo que se pretende es que se comprenda los retrasos que hemos sufrido concretamente en la primera fase y que están reflejados en el diagrama de Gantt así como facilitar el trabajo a futuros estudiantes o ingenieros que quieran abordar un proyecto de estas características.

5.1. Relacionados con la fase 1 (equilibrio)

5.1.1. Controladores de los motores

Antes de acabar utilizando los controladores *Simple H robot power,* utilizamos otros modelos distintos, el TB6612FNG *Dual Moto Driver Carrier* y el L293D.

Para ambos modelos el problema era prácticamente el mismo, el sobrecalentamiento debido a algún problema que desconocemos, y que no llegaban a entregar la potencia suficiente a los motores. El problema de la potencia lo descubrimos demasiado tarde, ya que aunque los motores funcionasen en ambos sentidos, como no teníamos otro robot con el que poder comparar no sabíamos si realmente funcionaban a su potencia

máxima. Debido a esto, el robot no llegaba a rectificar el ángulo y a mantenerse en equilibrio.

Con el TB6612FNG *Dual Moto Driver Carrier,* mientras hacíamos las pruebas básicas para comprobar el comportamiento de los motores, moverse hacia adelante y hacia atrás el controlador prendió fuego. Como consecuencia directa el controlador dejó de funcionar y tuvimos que cambiar el controlador por otro, el L293D. Una vez hecho el cambio, nos dimos cuenta que aparte de quemar el controlador, también se quemaron algunas de las patas a las cuales estaba conectado el primer controlador, por lo que tuvimos que sustituir las conexiones.

Aún con el nuevo controlador, no conseguíamos que el robot se mantuviera en equilibrio, por lo que empezamos a sospechar que el problema podría ser de los motores.

Finalmente descubrimos que el problema era tanto del controlador porque no entregaba la potencia suficiente, como de los motores que no funcionaban correctamente, y los sustituimos por los que tenemos ahora.

Una diferencia entre los controladores actuales respecto a los anteriores es que los anteriores cada uno podía controlar dos motores, mientras que los actuales solo pueden controlar un motor cada uno y es por eso por lo que tenemos dos controladores para los motores, uno para cada rueda, proporcionando mucha más potencia.

5.1.2. Motores

Como se ha comentado en el apartado anterior, llegamos a las sospechas de que los motores no funcionaban correctamente. Para comprobar el funcionamiento de estos motores, volvimos a conectar la placa original que venía con el robot, la que compramos, para saber si aún seguía funcionando correctamente y así descartar el problema de los motores.

Para nuestra sorpresa, vimos que con la placa anterior, es decir, tal y como nos vino el robot según lo compramos, tampoco funcionaba. Los motores se comportaban de forma rara y el robot no llegaba a mantenerse en equilibrio.

Haciendo comprobaciones con distintas herramientas (osciloscopio, multímetro…) se descubrió que hacía falta un ciclo de trabajo del 50% para que los motores comenzaran a moverse, lo que nos limitaba a la hora de controlar el robot en inclinaciones cerca de los 0º y para cuando comenzaban a moverse los motores ya era

demasiado tarde porque el robot se caía. Aparte de esto, la potencia que le llegaba a los motores desde el controlador no era la máxima como ya se ha comentado.

Finalmente decidimos comprar unos motores nuevos que tuvieran unas características similares a los anteriores y adecuadas para este tipo de robot.

5.1.3. Explorer 16

La *Explorer 16* dispone de unos reguladores de voltaje los cuales impiden el paso de voltajes superiores de 5V y de 3.3V. Mientras trabajábamos nos dimos cuenta que estos reguladores se calentaban en exceso, hasta que un día se quemó uno de ellos. A pesar de ello la *Explorer 16* seguía funcionando correctamente y por ello seguimos usándola. Un mes después de que ocurriese esto y cuando ya se mantenía en equilibrio correctamente, la placa dejo de funcionar por motivos desconocidos, por lo que nuestras sospechas apuntan a que debió ser por el regulador roto.

Una vez cambiado el *Explorer 16* y volver a soldar todas las conexiones necesarias para que el robot funcionase correctamente, el comportamiento del robot cambió y ya no se mantenía solo en equilibrio. Debido a esto, tuvimos que cambiar la configuración del robot. Ya que este problema ocurrió bastante tarde, principios de julio, y no disponíamos del suficiente tiempo como para dejarlo tan bien como antes, y aunque el problema está solucionado parcialmente porque el robot se mantiene en equilibrio, no se mantiene tanto tiempo como antes que prácticamente podía estar en equilibrio permanentemente.

Creemos que este problema (al cambiar de una placa a otra placa) es debido a que los ajustes que teníamos realizados sobre la primera *Explorer 16* eran unos ajustes que ayudaban a corregir los defectos que esta placa tenía.

5.1.4. Ajuste PID

Como ya se ha mencionado, el ajuste de los valores Kp, Kd y Ki los hacemos a ojo, por lo que cada vez que hemos tenido un problema de los mencionados anteriormente, ha habido que reajustarlos, cosa que toma bastante tiempo.

Durante un largo tiempo, entre 2 y 3 meses, hemos estado trabajando en el ajuste del controlador PID sin resultados satisfactorios, hasta que descubrimos que teníamos problemas tanto con los motores como con los controladores.

5.1.5. Conversión de datos

En un principio el controlador PID nos proporcionaba valores en un formato *double*, los cuales eran muy grandes. Al realizar la conversión del tipo *double* al tipo entero (*int*), ya que el *double* es mucho más grande que el entero, 8 bytes el *double* y 4 *bytes* el entero, este no cabía en el entero dándole un valor desconocido que no era el valor que verdaderamente esperábamos.

En un principio teníamos limitado el valor del PID, la variable k_k, después de la conversión (`auxoutputPID=(int)k_k`) como se puede ver en el siguiente código:

```
k_k=Proportional+ Integral +Kd * Derivative;
auxoutputPID=(int)k_k;

if (k_k <-LIMITE_PWM)
    {
      k_k=-LIMITE_PWM;
    }else if (k_k>=LIMITE_PWM)
    {
      k_k=LIMITE_PWM;
```

Debido a ello los valores que obteníamos no eran los deseados y algunas veces incluso funcionaban hacia el lado contrario al deseado. Para solucionar esto se decidió limitar el valor que obteníamos del PID antes de realizar la conversión. Con ello acotamos el valor *double* en un rango de valores que al convertirlos a entero no da problemas de conversiones, con lo cual logramos el dato esperado.

```
  k_k=Proportional+ Integral +Kd * Derivative;


    if (k_k <-LIMITE_PWM)
    {
      k_k=-LIMITE_PWM;
    }else if (k_k>=LIMITE_PWM)
    {
      k_k=LIMITE_PWM;
    }

    auxoutputPID=(int)k_k;
```

5.2. Relacionados con la fase 2 (Bluetooth)

5.2.1. Sincronizar dispositivo *Android* con el módulo RN4020

Como ya se ha comentado, la idea principal del proyecto era controlar el robot desde un dispositivo *Android* más concretamente desde un teléfono móvil.

Lo que a priori parecía relativamente sencillo se empezó a complicar desde el inicio, ya que para poder conectarnos al módulo RN4020 del robot, nuestro móvil debe de tener implementado la versión 4.1 del Bluetooth, y esta versión salió en Diciembre del 2013, por lo que es una versión bastante nueva y los dispositivos fabricados antes de esa fecha no disponen de esa versión (ni muchos modelos fabricados después de esa fecha tampoco).

Por suerte, tenía que cambiar de teléfono móvil y aproveche a comprar uno que si traía la versión del Bluetooth 4.1, más concretamente el teléfono es el ***"Sony Xperia M4 Aqua"***.

Una vez que se dispuso de un dispositivo *Android* con la versión 4.1 Bluetooth podemos activar el Bluetooth del teléfono y hacer un escaneo para buscar dispositivos Bluetooth, el módulo RN4020 está programado según la configuración descrita anteriormente para que se comporte como un periférico (como un esclavo) y de esta manera ser visible para el resto de dispositivos Bluetooth.

Una vez que escaneamos desde el teléfono móvil encontramos el módulo RN4020 tal y como se muestra en la Figura 28.

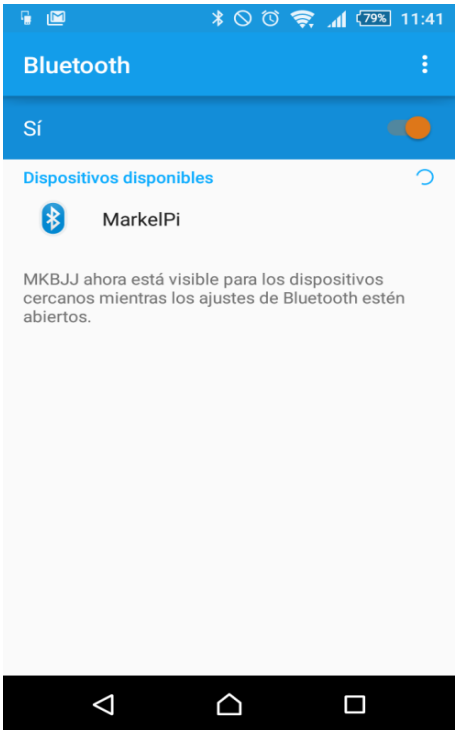


Figura 29: Resultado tras el escaneo de dispositivos Bluetooth. El escaneo se realiza desde el teléfono *Sony Xperia M4 Aqua.*

El problema es que a la hora de sincronizar el teléfono con el módulo RN4020 del robot, nos pide un PIN, que en todas las pruebas anteriores realizadas con el código proporcionado por Microchip para conectarnos desde un PIC con un módulo RN4020 a otro PIC con un módulo RN4020 no nos pedía. Por lo que lo primero que hago es mirar el código de la configuración del dispositivo maestro (o central) para ver si en algún momento se envía algún tipo de PIN para la conexión, pero no apareció nada de nada.

Mirando en internet decían que la mayoría de los dispositivos Bluetooth traían PIN por defecto y que normalmente era "0000" o "1234". Lo probé pero no dio resultado.

Entonces consulté en el manual del RN4020 proporcionado por Microchip a ver si ponía algo de algún PIN por defecto y tampoco aparecía nada. Tras un largo tiempo de búsqueda, en un foro de Microchip aparecía gente con el mismo problema a la hora de sincronizar un dispositivo *Android* con el módulo RN4020 aunque las soluciones aportadas por muchos de ellos no resultaran validas, había una que es la que terminé usando con la que conseguí sincronizar el teléfono con el módulo. La solución consiste en que a la hora de enviar la configuración del módulo en modo esclavo o periférico con el comando en vez de enviar **"SR,32000000"** hay que enviar **"SR,32006000"** y de esta forma no nos pide ningún PIN y en el móvil aparece sincronizado.

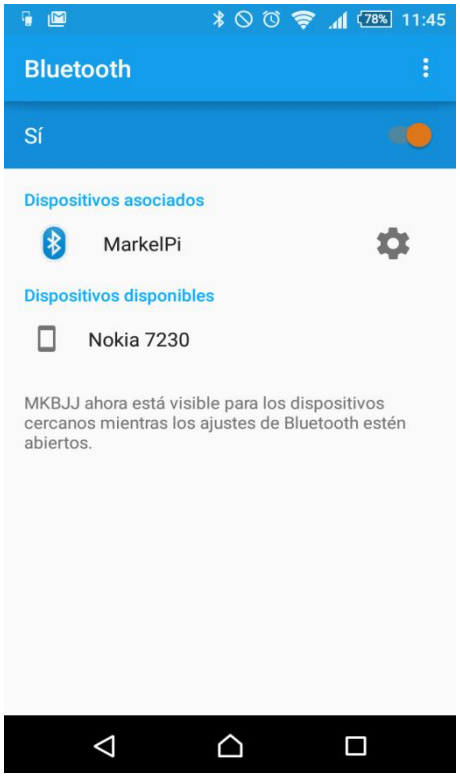


Figura 30: Sincronización del teléfono con el módulo Bluetooth RN4020, que tiene el nombre "MarkelPi".

Cuando parece que todo está bien, si nos fijamos en el LCD de la placa del robot el dispositivo está desconectado. Por lo que aunque se haya sincronizado con el teléfono no se ha llegado a conectar y por lo tanto no se pueden comunicar.

Figura 31: LCD de la placa *Explorer 16* tras intentar conectarse al módulo Bluetooth desde el teléfono. Se puede apreciar que pone *Disconnected..* (Desconectado..) por lo que la conexión no se llega a establecer.

En este punto tras muchos intentos y no conseguir sincronizarlos de manera correcta, se descarta la posibilidad de hacerlo desde un dispositivo *Android*.

5.2.2. Nombre dispositivo Bluetooth

Para comprobar que entendía los programas proporcionados por Microchip intento cambiar el nombre del módulo Bluetooth que actúa como periférico (esclavo). Así, cuando desde la otra placa *Explorer 16* cargada con el programa con la configuración central (maestro) haga el escaneo de los dispositivos Bluetooth, deberá de aparecer un nombre distinto en vez de la dirección MAC del dispositivo escaneado.

Tras un par de pruebas, noto por mucho que hiciera en el LCD nada cambiaba. Después de escanear, seguía apareciendo la dirección MAC del dispositivo en vez del nombre "MarkelPicado" con el que había configurado el módulo RN4020. Al final mirando en el código descubrí que el programa estaba implementado para que por el LCD se mostrara siempre la dirección MAC del dispositivo escaneado y no su nombre por lo que por mucho que cambiara no había ningún cambio. Escaneando desde el teléfono debería de aparecer algún cambio ya que en el teléfono sí que aparece el nombre, pero en este caso tampoco había ningún cambio. Tras muchas pruebas que la mayoría consistían tan solo en enviar un comando antes que otro o después que otro, para eso hay que volver a compilar todo y cargarlo de nuevo, cosa que tarda su tiempo, seguía sin cambiarse nada. Fruto de la desesperación se me ocurrió cambiar el nombre del dispositivo de "MarkelPicado" a "Markel" y de esta manera funcionó. En el manual de Microchip pone claramente que el nombre puede ser un valor alfanumérico de hasta 20 caracteres, cosa que "MarkelPicado" cumplía a la perfección. Al final, una vez descubierto que con "Markel" sí que funcionaba, empecé a hacer pruebas a ver si el problema estaba en la longitud, que aunque yo me ceñía a lo indicado en el manual del fabricante podría haber un error. Y exactamente, el máximo tamaño de longitud que me aceptaban era 8 caracteres alfanuméricos. Si por lo que fuera era más largo de 8 no se producía ningún cambio y por eso el nombre final del dispositivo es "MarkelPi".

5.2.3. Control Bluetooth

Como ya se ha mencionado, una vez que conseguida la comunicación Bluetooth con el robot, no he conseguido la manera de hacer que el robot ande hacia adelante y hacia atrás, mientras que los giros los hace perfectamente. Debido a la falta de tiempo no he podido investigar lo suficiente como para solucionarlo. El problema creo que reside en el control de equilibrio del robot. Cuando enviamos la orden al robot de que ande hacia adelante o hacia atrás, este mueve ambas ruedas pero esto afecta a la estabilidad del robot. Cuando el robot pierde la estabilidad, rápidamente el control PID lo corrige y lo vuelve a poner en equilibrio. De esta manera, le resulta imposible al robot avanzar hacia adelante o hacia atrás. Sin embargo, si lo que queremos es que el

robot gire, no hay ningún problema ya que el giro no pone en ningún compromiso a la estabilidad del robot (no en todos los casos).

·6·

6. RESULTADOS Y FUTUROS DESARROLLOS

Los resultados obtenidos tras el desarrollo del proyecto han sido satisfactorios. Si se analizan los objetivos que estaban definidos: **equilibrio del robot balancín** y el **control remoto vía Bluetooth,** han sido satisfechos, así como los objetivos secundarios y las competencias transversales que estos conllevan.

URL del video en YouTube: https://www.youtube.com/watch?v=IZ_7FxunIlo

Gracias a este proyecto he conseguido repasar y complementar conocimientos que he visto durante el grado. Así mismo, me he enfrentado a una infinidad de problemas desde el inicio del proyecto. He aprendido a ser autosuficiente, a abrir abanicos de posibilidades frente a las distintas dificultades que he tenido que superar y finalmente a mantener la calma problema tras problema en busca de soluciones aun a vísperas de la entrega del proyecto (Y no por culpa de una mala planificación del proyecto).

Este tipo de proyectos me parecen interesantes, ya que controlar por Bluetooth un robot de dos ruedas que mantiene el equilibrio por si solo me resulta bastante atrayente. Por eso, tras el desarrollo de este proyecto han aumentado aún más mis ganas por seguir investigando en este apasionante mundo de los sistemas embebidos, que une la electrónica con la informática.

Aunque los objetivos definidos se hayan cumplido, se puede seguir trabajando en distintos aspectos tanto como para aumentar funcionalidad al robot como para mejorar la que ya dispone.

Respecto al equilibrio del robot, aún se podría ajustar algo más. Aunque consiga mantenerse en equilibrio durante más de 10 minutos, tiene tendencia de ir avanzando muy poco a poco hacia adelante. Esto podría mejorarse o bien ajustando mejor a la placa la Unidad de Medición Inercial (IMU) o bien buscando una combinación para el controlador PID distinta de la que ya hay. Sin embargo considero más importante intentar mejorar la respuesta a los empujones. Cuando se somete al robot a una fuerza externa, normalmente, suele mantener la estabilidad. Dependiendo de la fuerza (No necesariamente una fuerza muy grande) a veces pierde el equilibrio y acaba cayéndose. Esto es algo importante a tener en cuenta ya que forma parte de las funcionalidades básicas que tiene que tener el robot.

En cuanto al control remoto, hay distintos aspectos en los que se puede trabajar. El primero de ellos es el del propio control. Como ya se ha dicho, solo he conseguido que el robot gire hacia ambos lados (aunque combinando ambos giros parezca que anda hacia adelante) pero no he conseguido que ande hacia adelante ni hacia atrás. Posiblemente esté relacionado con algún enfrentamiento entre el controlador PID y el control Bluetooth. Por otro lado, sería una muy buena mejora sustituir la *Explorer 16* que se utiliza como mando RC, por un dispositivo *Android*. De esta manera se volvería más portable el robot, ya que muchísima gente dispone de estos dispositivos. Por último, también intentaría cambiar el tipo de conexión para el control remoto y dotar al robot de conexión *WiFi*. Así, estando conectado a una red *WiFi*, podría ser controlado desde un ordenador a kilómetros de distancia, enviar datos a un servidor y poder mostrarlos a cualquier persona con tan solo acceder a un servidor web o incluso controlarlo desde el propio servidor web.

Teniendo en cuenta estas mejoras, sería interesante añadirle una cámara al robot, y así poder controlarlo desde cualquier punto del mundo con conexión a internet a la vez que vas viendo el terreno por el que se está moviendo.

Dejando de un lado el control remoto, también trabajaría en los aspectos relacionados con el autocontrol del robot. Añadiendo sensores de distancia o de detección de golpes para que el robot sea más independiente y evitar que acabe destrozándose contra una pared.

Por último para tener un robot mucho más completo se podrían unificar en un solo programa, la parte implementada por Anartz Recalde que utiliza los *encoders*, y la parte que he desarrollado yo.

·A·

A. APENDICE

En este apartado se profundizará en diferentes temas de los que hemos hablado en el proyecto, pero no hemos profundizado debido a que son temas que se han dado en clase o no veíamos convenientes introducirlos en la memoria.

A.1. PWM

El funcionamiento es simple, el ciclo de trabajo describe la cantidad de tiempo que la señal está en un estado lógico alto, se expresa como el porcentaje del tiempo total que está para completar un ciclo completo. La frecuencia determina lo rápido que se completa un ciclo (por ejemplo: 1000 Hz corresponde a 1000 ciclos en un segundo). Al cambiar una señal del estado alto a bajo a una tasa lo suficientemente rápida y con un cierto ciclo de trabajo, la salida parecerá comportarse como una señal analógica constante cuando esta se aplica a algún dispositivo.

Ejemplo: Para crear una señal de 3V dada una fuente digital que puede ser alta (5V) o baja (0V), se puede utilizar un PWM con un ciclo de trabajo del 60%. El cual generaría una señal de 5V el 60% del tiempo. Si la señal es conmutada lo suficientemente rápido, el voltaje visto en las terminales del dispositivo parecerá ser el valor promedio de la señal. Si el estado lógico bajo es 0V (que es el caso más común) entonces el voltaje promedio puede ser calculado multiplicando el voltaje que represente el estado lógico alto por el ciclo de trabajo, o 5V x 0.6 = 3V. Seleccionar un ciclo de trabajo del 80% sería equivalente a 4V.

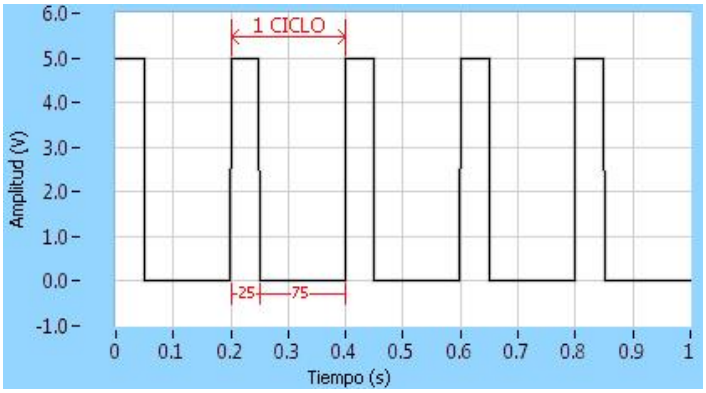


Figura 32: Ciclo de trabajo del 25%.

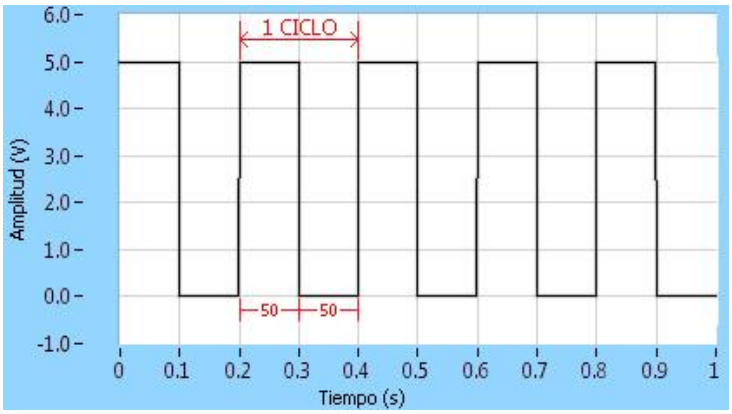


Figura 33: Ciclo de trabajo del 50%.

A.2. I²C

La principal característica del I^2C es que tan solo usa dos líneas para transmitir la información: una para los datos y otra para la señal del reloj. También hace falta una tercera línea, pero esta sólo es la referencia (0v). Estas tres líneas suelen usar las siguientes nomenclaturas:

- SCL (System Clock) es la línea de los pulsos de reloj que sincronizan el sistema.
- SDA (System Data) es la línea por la que se mueven los datos entre los dispositivos.
- GND (0v) común de la interconexión entre todos los dispositivos "enganchados" al bus.

Los dispositivos conectados al bus I^2C tienen una dirección única para cada uno.

Para que la comunicación tenga éxito debe seguir un protocolo de comunicación. En una conexión entre varios dispositivos a través del I^2C existen dos tipos de dispositivos: los dispositivos maestros y los dispositivos esclavos. El maestro es el Dispositivo que determina los tiempos y la dirección del tráfico en el bus. Es el único que aplica los pulsos de reloj en la línea SCL. Cuando se conectan varios dispositivos maestros a un mismo bus la configuración obtenida se denomina "multi-maestro". En cambio, el dispositivo esclavo no tiene la capacidad de generar pulsos de reloj. Los dispositivos esclavos reciben señales de comando y de reloj generados desde el maestro.

Para que el maestro comience una comunicación, el bus tiene que estar libre. Esto quiere decir que tanto la línea SDA como la SCL tienen que estar inactivas, es decir, presentar un estado lógico alto. Si no se da el caso de que el bus está libre el maestro no puede comenzar comunicación alguna con los dispositivos conectados mediante ese bus. Cuando el bus está libre, cualquier dispositivo maestro puede ocuparlo, estableciendo la condición de inicio, en la cual la línea de datos, SDA, toma un estado bajo mientras que la línea de reloj, SCL, permanece alta.

El primer byte que se transmite luego de la condición de inicio contiene siete bits que componen la dirección del dispositivo que se desea seleccionar, y un octavo bit que corresponde a la operación que se quiere realizar con él (lectura o escritura), aunque en nuestro caso siempre pretenderemos realizar la escritura (la operación de escritura se indica con el bit a nivel lógico bajo). Cada dispositivo tiene una dirección única, en nuestro caso la IMU tiene la dirección: 0xA6 en caso del acelerómetro y 0xD0 en caso del giroscopio, las cuales hemos averiguado gracias al proyecto que Omar Luque Rodríguez presentó sobre los cuadrucopteros en donde uso estos mismos dispositivos, pero aún así podríamos haberlos descubierto leyendo la ficha técnica del dispositivo IMU.

Tras enviar la condición *start* y los 8 bits si el dispositivo cuya dirección corresponde a la que se indica en los siete bits está presente en el bus, éste contesta con un bit en bajo, ubicado inmediatamente luego del octavo bit que ha enviado el dispositivo maestro. Este bit de reconocimiento (ACK) en bajo le indica al dispositivo maestro que el esclavo reconoce la solicitud y está en condiciones de comunicarse. Aquí la comunicación se establece en firme y comienza el intercambio de información entre los dispositivos.

Puesto que indicamos que se desea realizar la operación de escritura el dispositivo maestro envía datos al dispositivo esclavo. Esto se mantiene mientras continúe recibiendo señales de reconocimiento, y el contacto concluye cuando se

hayan transmitido todos los datos. El dispositivo maestro puede dejar libre el bus generando una condición de parada (stop en inglés).

La secuencia completa de comunicación seria la siguiente:

1. Enviar una secuencia de inicio
2. Enviar la dirección de dispositivo con el bit de lectura/escritura en bajo
3. Enviar el número de registro interno en el que se desea escribir
4. Enviar el byte de dato
5. [Opcionalmente, enviar más bytes de dato]
6. Enviar la secuencia de parada

A.3. Filtro de Kalman

El objetivo de este filtro será la obtención de un estimador óptimo de un sistema dinámico, basado en observaciones ruidosas y en un modelo de la incertidumbre de la dinámica del sistema. Para ello, haremos uso de dos ingredientes fundamentales: los conceptos de modelo del sistema y modelo de medida o de observación.

Para ir entendiendo la formulación vamos a considerar paralelamente la estimación de una constante (por ejemplo un voltaje) cuya medida lleva asociado un ruido con una desviación típica de 0.1 voltios. El voltaje será una constante y tenemos observaciones secuenciales ruidosas.

A.3.1. Modelo del sistema

El sistema físico se modeliza por un vector de estados x, llamado simplemente el estado, y un conjunto de ecuaciones llamado el modelo del sistema. El modelo del sistema es una ecuación de vectores que describe la evolución del estado con el tiempo. El tiempo de observación tiene la forma $t_k=t_0+k\Delta T$, k=0,1..., ΔT es el intervalo de muestreo y x_k el estado x (t_k). Vamos a suponer que ΔT es pequeño y que por tanto podemos utilizar un modelo del sistema lineal, es decir,

$$x_k = \varphi_{k-1}\, x_{k-1} + \xi_{k-1}$$

donde ξ_{k-1} es un vector aleatorio que modeliza el ruido aditivo. El subíndice k-1 en φ indica que la matriz de transición φ es (puede ser) una función del tiempo. Volvamos ahora a nuestro ejemplo. Como el voltaje es una constante podemos utilizar como modelo del sistema

$$x_k = x_{k-1} + \xi_{k-1}$$

Observemos que cuanto mayor sea k, en principio, más fiabilidad tendrá la estimación. Es importante analizar las características de ξ_{k-1} pero esto lo haremos con posterioridad. Casi siempre el ruido ξ_{k-1} se supone normal de media cero.

A.3.2. Modelo de medida

El segundo ingrediente en la teoría de la estimación es el modelo de medida. Suponemos que en cada instante t_k tenemos una observación ruidosa del vector de estados o al menos alguna de sus componentes mediante la siguiente relación

$$z_k = \mathrm{H}_k\, x_k + \mu_k$$

Z_k es el vector de medidas tomadas en el instante t_k. H_k es la llamada matriz de medidas y μ_k es un vector aleatorio que modeliza la incertidumbre asociada a las medidas. En nuestro caso tendríamos μ_k normales independientes de media "0" y desviación 0,1.

A.3.3. Formulación de la teoría de estimación lineal óptima del filtro de Kalman.

Esta sección es una recopilación de las dos anteriores.

El estado de un sistema dinámico en el instante t k está descrito por un vector n-dimensional x k llamado vector de estados. La evolución del sistema se modeliza mediante:

$$z_k = \varphi_{k-1}\, \mathrm{x}_{k-1} + \xi_{k-1}$$

Donde φ_{k-1} es una matriz de tamaño *n*x*n* llamada matriz de transición de estados y ξ_k es un *n*-vector que modeliza el ruido asociado al modelo del sistema.

En cualquier instante t k se obtiene un vector m-dimensional de medidas z_k. La relación entre el estado y las medidas es lineal y se modeliza mediante

$$z_k = \mathrm{H}_k\, x_k + \mu_k$$

H_k es una matriz dependiente del tiempo de tamaño *m*x*n* y μ_k es un m-vector aleatorio que modeliza la incertidumbre asociada a las medidas.

Los términos ξ_k y μ_k son vectores aleatorios gaussianos blancos de media cero y matrices de covarianza Q_k y R_k respectivamente.

Para el problema del ángulo de nuestro proyecto R_k=0.5 y la varianza del modelo de estados la fijamos nosotros. En nuestro caso hemos supuesto Q_k=0.001 aunque también hemos realizado pruebas con Q_k=0.003, pero vimos mejor resultado con el primero.

Supongamos que tenemos x_{k-1} y P_{k-1}. En el primer instante hemos de tener x_0 y P_0. Los valores iniciales x_0 y P_0 los introducimos a mano. ¿Cómo calculamos los nuevos estimadores iterativamente?

Tenemos un estimador x_{k-1} y su matriz de covarianzas P_{k-1}. Primero calculamos, antes de que llegue la observación z_k

$$P_k' = \phi_{k-1} P_{k-1} \phi_{k-1}^{t} + Q_{k-1}$$

Después la ganancia

$$K_k = P_k' H_k^{t} \left(H_k^{t} P_k' H_k + R_k \right)^{-1}$$

A continuación el estimador del estado k óptimo (aquí metemos la observación)

$$\underline{x}_k = \phi_{k-1} \underline{x}_{k-1} + P_k' H_k^{t} \left(H_k^{t} P_k' H_k + R_k \right)^{-1} \left(z_k - H_k \phi_{k-1} \underline{x}_{k-1} \right)$$

Y por último la matriz de covarianzas de este estimador

$$\begin{aligned} P_k &= P_k' - P_k' H_k^{t} \left(H_k^{t} P_k' H_k + R_k \right)^{-1} H_k P_k' = P_k' - K_k H_k P_k' \\ &= (I - K_k H_k) P_k' = (I - K_k) P_k' (I - K_k)^{t} - K_k R_k K_k^{t} \end{aligned}$$

·B·

B. REFERENCIAS

A continuación están algunas de las referencias más importantes que se han utilizado a la hora de desarrollar el proyecto. Cabe decir que para consultas más detalladas, hemos acudido al *datasheet* del fabricante de cada componente en la medida que ha sido posible.

B.1. Relacionados con el *Segway*

[1] http://www.tecnoneo.com/2014/10/chic-robot-smart-s1-scooter-para.html

[2] http://www.segway.es/

[3] https://es.wikipedia.org/wiki/Segway

[4] https://en.wikipedia.org/wiki/Segway_PT

[5] http://www.elmundo.es/madrid/2013/12/07/52a36fc80ab740a0768b4570.html

B.2. Péndulo invertido

[6] http://iimyo.forja.rediris.es/invpend/invpend.html

[7] http://www.ib.cnea.gov.ar/~instyctl/Tutorial_Matlab_esp/invpen.html

[8] http://eprints.ucm.es/16096/1/memoriaPFC.pdf

B.3. Proyectos similares

[9] http://electronica-iespolitecnico-cartagena.blogspot.com.es/2014/01/robot-equilibrista-con-servos-y-arduino.html

[10] http://nxttwowheels.blogspot.com.es/

[11] http://cienciaycacharreo.blogspot.com.es/2013/10/b-robot-un-robot-equilibrista-impreso_28.html

[12] http://www.todopic.com.ar/foros/index.php?topic=12748.0

[13] http://dse-urjc.blogspot.com.es/2015/04/robot-balancin.html

[14] http://www.iit.upcomillas.es/pfc/resumenes/4de4a8e596641.pdf

[15] http://www.elecrow.com/freescale-mc9s12xs128mal-2wheel-selfbalancing-robot-p-878.html

B.4. Controlador PID

[16] http://www.dia.uned.es/~fmorilla/MaterialDidactico/ajuste_empirico.pdf

[17] https://es.wikipedia.org/wiki/Controlador_PID

[18] http://www.lra.unileon.es/es/book/export/html/268

[19] http://control-pid.wikispaces.com/

B.5. Filtro de Kalman

[20] http://www.bccr.fi.cr/investigacioneseconomicas/metodoscuantitativos/Filtro_de_Kalman.pdf

[21] https://es.wikipedia.org/wiki/Filtro_de_Kalman

B.6. Relacionados con el Bluetooth

[22] http://www.mouser.es/new/microchip/microchip-rn4020-module/

[23] https://en.wikipedia.org/wiki/Bluetooth

B.7. Controladores de potencia

[24] http://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_2204082_-1

B.8. PWM

[25] https://es.wikipedia.org/wiki/Modulaci%C3%B3n_por_ancho_de_pulsos

[26] http://digital.ni.com/public.nsf/allkb/AA1BDEA4AA224E3E86257CE400707527

B.9. Unidad de Medición Inercial (IMU)

[27] https://es.wikipedia.org/wiki/Unidad_de_medici%C3%B3n_inercial

[28] https://ddd.uab.cat/pub/trerecpro/2012/hdl_2072_210290/PFC_CarlosPrecklerClemente.pdf

[29] http://fuenteabierta.teubi.co/2013/03/inclinometro-digital-con-arduino-uso-de.html

[30] http://forum.arduino.cc/index.php?topic=88956.0

·C·

C. CÓDIGOS ANEXOS

C.1. Controlador PID

```
void calcularPID(){

    errorFlag=16;

    //Angle = Angle + (((anguloFiltro-Angle) * 0.5 + vec_giro[0]) *
0.01);

    error=anguloFiltro-SETPOINT;

    Proportional=Kp*error;
    Derivative=error-prev_error;

    if ( (abs(error)<=MARGEN_INTEGRADOR))
    {
      Integral+=Ki*error;
        if (Integral>LIMITE_PWM)Integral=LIMITE_PWM;
        else if(Integral < -LIMITE_PWM) Integral= -LIMITE_PWM;
    }
    else {
        if(error>=0){
            Integral+=Ki*error/2;
        }
        else Integral+=Ki*error/2;
    }
     //Parte derivativa


    //-------------------------------------------------------------------
    k_k=Proportional+ Integral +Kd * Derivative;
   //k_k = Kp * anguloFiltro + Kd * vec_giro[0]+Integral;

    //k_k=k_k+300.0;
    //Limites de la salida.
    //auxoutputPID=auxoutputPID+turbo;
    if (k_k <-LIMITE_PWM)
    {
```

```
      k_k=-LIMITE_PWM;
    }else if (k_k>=LIMITE_PWM)
    {
      k_k=LIMITE_PWM;
    }

    auxoutputPID=(int)k_k;
    outputPID=auxoutputPID;
    prev_error=error;
}
```

C.2. Filtro de Kalman

```
float Q_bias, Angle_err;
float K_0, K_1, t_0, t_1;
float Pdot [4] = {0,0,0,0};
float PP [2] [2] = {{1, 0}, {0, 1}};
float Q_angle = 0.001;
float Q_gyro = 0.003; // small, adjust the tracking speed of the
waveform, but do not believe that prediction
float R_angle = 0.5;
float dt = 0.01; // dt for kalman filter sampling time;
char C_0 = 1;
float PCt_0, PCt_1, E;

void filtroKalman(float Accel, float Gyro){
anguloFiltro += (Gyro - Q_bias) * dt; // a priori estimate


Pdot [0] = Q_angle - PP [0] [1] - PP [1] [0]; // Pk- differential
priori estimate error covariance

Pdot [1] = - PP [1] [1];
Pdot [2] = - PP [1] [1];
Pdot [3] = Q_gyro;

PP [0][0] += Pdot [0] * dt; // Pk- priori estimate error covariance
integral differential
PP [0][1] += Pdot [1] * dt; // = a priori estimate error covariance
PP [1][0] += Pdot [2] * dt;
PP [1][1] += Pdot [3] * dt;

Angle_err = Accel - anguloFiltro; // zk- priori estimate

PCt_0 = C_0 * PP [0] [0];
PCt_1 = C_0 * PP [1] [0];

E = R_angle + C_0 * PCt_0;

K_0 = PCt_0 / E;
K_1 = PCt_1 / E;

t_0 = PCt_0;
t_1 = C_0 * PP [0] [1];
```

```
PP [0][0] -= K_0 * t_0; // after posteriori estimate error covariance
PP [0][1] -= K_0 * t_1;
PP [1][0] -= K_1 * t_0;
PP [1][1] -= K_1 * t_1;

anguloFiltro += K_0 * Angle_err; //posterior estimate after //
Q_bias += K_1 * Angle_err; //posterior estimate after //
vec_giro[0] = Gyro - Q_bias; // output value (a posteriori estimate) =
angular speed differential


//anguloFiltro=anguloFiltro+(((angulo-anguloFiltro)*0.5+vec_giro[0]) *
0.01);
Nop();
}
```

C.3. Unidad de Medición Inercial (IMU)

C.3.2. Eliminación del Offset del acelerómetro

```
void calibrar_acel()
{
    errorFlag=10;
    estado_calibracion = 0;
    int i;
    double vec_offset[3]={0.0,0.0,0.0};

    //Recolectar valores offset
    vec_offset[0]=0.0;
    errorFlag=11;
    for(i=0;i<5000;i++)
    {
        LDByteReadI2C_1(DirAcelW, Reg_DATAX0, aux_data, 6);
        vec_acel[1] = -(aux_data[0] + aux_data[1]*256)*CONV_ACEL;

        vec_offset[0]+=vec_acel[1];
        //sumar_vectores(vec_offset,vec_offset,vec_giro);
    }
    errorFlag=12;
    //Sacar la media
    vec_offset[0]=vec_offset[0]/5000.0;
    //vec_offset[0]+=0.20646405;

    errorFlag=13;

    ACEL_OFFSET_X=vec_offset[0];

    errorFlag=14;
}
```

C.3.2. Obtención del ángulo haciendo uso del acelerómetro

```
//Leer valores acelerometro
void Leer_Acel(/*unsigned int x, unsigned int * y, unsigned int * z*/)
{
    errorFlag=4;   // ???
    LDByteReadI2C_1(DirAcelW, Reg_DATAX0, aux_data, 6);

    acelX = aux_data[0] + aux_data[1]*256;
    acelY = aux_data[2] + aux_data[3]*256;
    acelZ = aux_data[4] + aux_data[5]*256;

    vec_acel[1] = -(aux_data[0] + aux_data[1]*256)*CONV_ACEL;
    vec_acel[0] = (aux_data[2] + aux_data[3]*256)*CONV_ACEL;
    vec_acel[2] = -(aux_data[4] + aux_data[5]*256)*CONV_ACEL;
    angulo=atan2(acelX,acelZ) * 180.0 / 3.14159265;
    if(angulo>0)angulo=angulo-errorAngulo;
    //angulo=angulo-errorAngulo;
    angulo=angulo*1.5;
    errorFlag=5;
}
```

C.3.3. Eliminación del Offset del giroscopio

```
void calibrar_giro()
{
    errorFlag=10;
    estado_calibracion = 0;
    int i;
    double vec_offset[3]={0.0,0.0,0.0};

    //Imprimir: Deja la placa quieta y pulsa "N" o "n"
    /*i=0;
    putRS232_2('\x1B');
    putRS232_2('[');
    putRS232_2('2');
    putRS232_2('J');
    while(TEXTOG1[i]!='\0'){
        putRS232_2(TEXTOG1[i]);
        i++;
    }*/

    //Esperar pulsacion
    //while(estado_calibracion==0);

    //Recolectar valores offset
    vec_offset[0]=0.0;
    errorFlag=11;
    for(i=0;i<5000;i++)
    {
        LDByteReadI2C_1(DirGiroW, Reg_XOUT_H, aux_data2, 6);
```

```
        vec_giro[1] = (aux_data2[1] +
aux_data2[0]*256)*CONV_GIRO1*DEG2RAD;
        vec_giro[0] = -(aux_data2[3] + aux_data2[2]*256)*CONV_GIRO1;
        vec_giro[2] = (aux_data2[5] +
aux_data2[4]*256)*CONV_GIRO1*DEG2RAD;
        vec_offset[0]+=vec_giro[0];
        //sumar_vectores(vec_offset,vec_offset,vec_giro);
    }
    errorFlag=12;
    //Sacar la media
    vec_offset[0]=vec_offset[0]/5000.0;
    vec_offset[1]=vec_offset[1]/5000.0;
    vec_offset[2]=vec_offset[2]/5000.0;
    vec_offset[0]+=0.20646405;

    errorFlag=13;

    GIRO_OFFSET_X=vec_offset[0];
    GIRO_OFFSET_Y=vec_offset[1];
    GIRO_OFFSET_Z=vec_offset[2];

    errorFlag=14;
}
```

C.3.4. Obtención del ángulo haciendo uso del giroscopio

```
//Leer valores giroscopio
void Leer_Giro(/*signed short x, signed short y, signed short z*/)
{
    //t2=clock();
    LDByteReadI2C_1(DirGiroW, Reg_XOUT_H, aux_data2, 6);

    vec_giro[1] = (aux_data2[1] +
aux_data2[0]*256)*CONV_GIRO1*DEG2RAD;
    vec_giro[0] = (-(aux_data2[3] + aux_data2[2]*256)*CONV_GIRO1)-
GIRO_OFFSET_X;
    vec_giro[2] = (aux_data2[5] +
aux_data2[4]*256)*CONV_GIRO1*DEG2RAD;


  //  tiempo=(t2-t1);
    angulogiro=angulogiro+0.01*vec_giro[0];
  // angulogiro=angulogiro+(int)((sec2-sec1)/1000)*vec_giro[0];
  //  t1=clock();
}
```

C.4. Protocolo de control Bluetooth

C.4.1. Código que corresponde a la parte del robot

```
void Process_Control(P_BTLE_PERI_CONTROL_T lBTLE_Resp)
{
    if (ePERI_GO_FORDWARD == lBTLE_Resp)
    {
        velExtra_RI=1000;
        velExtra_RD=1000;
        XPL16_D10_LED_LAT = XPL16_D10_LED_TurnON;
    }
    else if (ePERI_GO_BACK == lBTLE_Resp)
    {
        velExtra_RI=-1000;
        velExtra_RD=-1000;
        XPL16_D9_LED_LAT = XPL16_D9_LED_TurnON;
    }
    else if (ePERI_TURN_RIGHT == lBTLE_Resp)
    {
        velExtra_RI=1000;
        velExtra_RD=-1000;
        XPL16_D8_LED_LAT = XPL16_D8_LED_TurnON;
    }
    else if(ePERI_TURN_LEFT == lBTLE_Resp)
    {
        velExtra_RI=-1000;
        velExtra_RD=1000;
        XPL16_D7_LED_LAT = XPL16_D7_LED_TurnON;
    }
    else if(ePERI_STOP == lBTLE_Resp)
    {
        velExtra_RI=0;
        velExtra_RD=0;
        XPL16_D10_LED_LAT = XPL16_D10_LED_TurnOFF;
        XPL16_D9_LED_LAT = XPL16_D9_LED_TurnOFF;
        XPL16_D8_LED_LAT = XPL16_D8_LED_TurnOFF;
        XPL16_D7_LED_LAT = XPL16_D7_LED_TurnOFF;
    }
}
```

C.4.2. Código que corresponde a la parte de la *Explorer 16* (Mando RC)

```
void Process_Switches2(void)
{
   // Check the Switch status.
    char *lStringPtr = 0;
    if (!gBlueToothConnected)
    {
        Process_BTLE_Peripheral_Selection();
```

```
    }
    else
    {
        if (!Message_Tx_Packet.Msg_To_Be_Processed)
        {
            gHw_Switches.s1_sw_current = SWITCH_S1_PORT;
            gHw_Switches.s2_sw_current = SWITCH_S2_PORT;
            gHw_Switches.s3_sw_current = SWITCH_S3_PORT;
            gHw_Switches.s4_sw_current = SWITCH_S4_PORT;

            if (gHw_Switches.s1_sw_current != gHw_Switches.s1_sw_Old)
            {
                // Pass this information.
                if (gHw_Switches.s1_sw_current)
                {
                    lStringPtr =
                     (char
*)GetBTLE_RespStringFrom_BTLE_CONTROL_Strings(eCENTRAL_STOP);
                }
                else
                {
                    lStringPtr =
                       (char
*)GetBTLE_RespStringFrom_BTLE_CONTROL_Strings(eCENTRAL_GO_FORDWARD);
                }
                Message_Tx_Packet.Msg_To_Be_Processed = 1;
                gHw_Switches.s1_sw_Old = gHw_Switches.s1_sw_current;
            }
            else if (gHw_Switches.s2_sw_current !=
gHw_Switches.s2_sw_Old)
            {
                // Pass this information.
                if (gHw_Switches.s2_sw_current)
                {
                    lStringPtr =
                    (char
*)GetBTLE_RespStringFrom_BTLE_CONTROL_Strings(eCENTRAL_STOP);
                }
                else
                {
                    lStringPtr =
                    (char
*)GetBTLE_RespStringFrom_BTLE_CONTROL_Strings(eCENTRAL_GO_BACK);
                }
                Message_Tx_Packet.Msg_To_Be_Processed = 1;
                gHw_Switches.s2_sw_Old = gHw_Switches.s2_sw_current;
            }
            else if (gHw_Switches.s3_sw_current !=
gHw_Switches.s3_sw_Old)
            {
                // Pass this information.
                if (gHw_Switches.s3_sw_current)
                {
                    lStringPtr =
                    (char
*)GetBTLE_RespStringFrom_BTLE_CONTROL_Strings(eCENTRAL_STOP);
                }
                else
                {
```

```
                    lStringPtr =
                    (char
*)GetBTLE_RespStringFrom_BTLE_CONTROL_Strings(eCENTRAL_TURN_LEFT);
                }
                Message_Tx_Packet.Msg_To_Be_Processed = 1;
                gHw_Switches.s3_sw_Old = gHw_Switches.s3_sw_current;
            }
            else if (gHw_Switches.s4_sw_current !=
gHw_Switches.s4_sw_Old)
            {
                // Pass this information.
                if (gHw_Switches.s4_sw_current)
                {
                    lStringPtr =
                    (char
*)GetBTLE_RespStringFrom_BTLE_CONTROL_Strings(eCENTRAL_STOP);
                }
                else
                {
                    lStringPtr =
                    (char
*)GetBTLE_RespStringFrom_BTLE_CONTROL_Strings(eCENTRAL_TURN_RIGHT);
                }
                Message_Tx_Packet.Msg_To_Be_Processed = 1;
                gHw_Switches.s4_sw_Old = gHw_Switches.s4_sw_current;
            }
            if (lStringPtr && Message_Tx_Packet.Msg_To_Be_Processed)
            {
                strcpy((char *)Message_Tx_Packet.Message_Buffer,
lStringPtr);// Copia un string de origen a destino
            }
        }
    }
}
```

C.5. PWM

```
void calcularPWM(void){
    //calcularPID();
    if(estadoRobot==1){


        if(outputPID >0) // vamos hacia adelante
            if(dir==1){
                dir=0;
                Integral=0.0;
            }
        else if (outputPID <0)
           if(dir==0){
                dir=1;
                Integral=0.0;
            }
```

```
        outputPIDIzq=outputPID;
        outputPIDDch=outputPID;

        //se limita el valor que le daremos al potenciometro para que
no pase del minimo o maximo de potencia que podemos darle.


        if(outputPIDIzq+velExtra_RI>0){

            OC2RS=min(abs(outputPIDIzq+velExtra_RI),LIMITE_PWM);
            OC4RS=0;

        }else {
            OC2RS=0;
            OC4RS=min(abs(outputPIDIzq+velExtra_RI),LIMITE_PWM);

        }
        if(outputPIDDch+velExtra_RD>0){

            OC1RS=min(abs(outputPIDDch+velExtra_RD),LIMITE_PWM);
            OC3RS=0;

        }else {
            OC1RS=0;
            OC3RS=min(abs(outputPIDDch+velExtra_RD),LIMITE_PWM);

        }
        if (outputPIDDch==0 && outputPIDIzq==0){
            OC1RS=0;
            OC3RS=0;
            OC2RS=0;
            OC4RS=0;
        }

    }else{
            OC1RS=0;
            OC3RS=0;
            OC2RS=0;
            OC4RS=0;
             //Integral=0.0;
        }
}
```