UNIVERSIDAD POLITÉCNICA DE VALENCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

TRABAJO FIN DE MÁSTER
INGENIERÍA MECATRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN ROBOT AUTOBALANCEADO

AUTOR:
Jorge Ferrer Gómez

DIRECTOR:
Juan José Serrano Martín

Septiembre 2017

ÍNDICE

1. RESUMEN GENERAL

1.1 RESUMEN (ESPAÑOL)

Según la teoría darwiniana llegó un momento de la evolución en que el antepasado del ser humano comenzó a andar a dos piernas. A su semejanza se ha deseado realizar un proceso similar en el caso de un robot, con la diferencia de que en lugar de piernas, éste posea únicamente ruedas. Se presenta, por tanto, el problema del péndulo invertido llevado a la automática con la cualidad de desplazarse como un vehículo con ruedas.

Inicialmente se reflexionó cual era el problema a plantear y los medios necesarios para llevarlo a cabo. Se necesitaba que una estructura pudiera mantener el equilibrio únicamente estando apoyada sobre dos ruedas y a su vez pudiera desplazarse sobre una superficie. Para la realización de este objetivo primero se pensaron los diferentes casos que podían darse. Para funcionar correctamente necesitaba de un controlador (En este caso se empleó un microcontrolador STM32). Necesitaba la existencia de una estructura. Ésta debería moverse en l una superficie, por lo tanto necesitaría de ruedas y unos motores. Eran necesarios una serie de sensores que fueran los sentidos del robot y le informaran sobre lo que sucedía en cada momento. Un ejemplo de éstos es la unidad de medida inercial (IMU, modelo MKI124V1), la cual le informa sobre las posiciones angulares de la estructura. De esta forma el propio robot puede auto reajustarse y cumplir su función. Para la correcta implementación de todos los elementos anteriores y los no nombrados en este resumen, pero que se verán a lo largo de esta memoria, fueron necesarias una serie de pruebas y estudios con sus diferentes implicaciones. Se realizó desde el diseño, el montaje hasta la programación del robot.

Con la presente actividad se desea conocer el funcionamiento de este tipo de automatismo el cual, hoy en día, está teniendo cada vez mayor presencia en el mercado. Sus futuras proyecciones aún son amplias y le queda todavía un buen recorrido. Además es un ejemplo ideal para aplicar las diferentes materias que componen la mecatrónica. Con este trabajo se desea realizar una introducción a dicho campo, el cual facilite al autor y a aquel que lo desee poder conocer este tipo de robots.

1.2 ABSTRACT (ENGLISH)

According to the Darwinian Theory there was a moment in evolution when the human ancestor started to walk on two feet. Similarly, it is hoped to make a similar process with a robot, but using wheels instead of legs. Therefore, the inverted pendulum problem has been taken to automatics with the trait of moving as a wheeled vehicle.

At first, it was considered which problem to formulate and the necessary means to carry it out. A structure which could keep its balance and move on a surface with only two wheels was needed. For the fulfilment of this goal, the different possibilities that might appear were considered. It needed a controller to make it work correctly (In this case it was using a STM32 microcontroller). A structure was also required. In addition, it should move on a surface so it would also need wheels and motors. It also needed sensors to be aware and inform it about what happens every moment. An example of this is the Inertial Measurement Unit (IMU,

MKI124V1 model). It informs about the angular position of the structure. This way the robot can readjust itself and achieve its aim. Multiple tests and researches were needed, with their different implications, for the correct implementation of every previously listed element and the ones not yet mentioned but explained throughout this report. The design and the mounting, including the programming of the robot, have been made.

This activity looks forward to knowing the functioning of this kind of automatism which nowadays is increasingly more present in the market. Its future repercussions are still broad and it has a long way to go. Besides, it is an ideal example of the different subjects included the mechatronics. This project is determined to make an introduction to that field, making the author and anyone who wishes, to be able to know about this kind of robot.

1.3 RESUM (VALENCIÀ)

Segons la teoria darwiniana va arribar un moment de l'evolució en què l'avantpassat de l'ésser humà va començar a caminar a dues cames. A la seua semblança s'ha desitjat realitzar un procés similar en el cas d'un robot, amb la diferència que en lloc de cames, aquest posseïsca únicament rodes.

Es presenta, per tant, el problema del pèndol invertit portat a l'automàtica amb la qualitat de desplaçar-se com un vehicle amb rodes. Inicialment es va reflexionar com era el problema a plantejar i els mitjans necessaris per a dur-ho a terme. Es necessitava que una estructura poguera mantenir l'equilibri únicament estant recolzada sobre dues rodes i al seu torn poguera desplaçar-se sobre una superfície. Per a la realització d'aquest objectiu primer es van pensar els diferents casos que podien donar-se. Per a funcionar correctament necessitava d'un controlador (En aquest cas es va utilitzar un microcontrolador STM32). Necessitava l'existència d'una estructura. Aquesta hauria de moure's en una superfície, per tant necessitaria de rodes i d'uns motors. Eren necessaris una sèrie de sensors que foren els sentits del robot i li informaren sobre el que succeïa a cada moment. Un exemple d'aquests, és la unitat de mesura inercial (IMU, model MKI124V1), la qual li informa sobre les posicions angulars de l'estructura. D'aquesta forma el propi robot pot auto- reajustar-se i complir la seua funció. Per a la correcta implementació de tots els elements anteriors i els no nomenats en aquest resum, però que es veuran al llarg d'aquesta memòria, van ser necessàries una sèrie de proves i estudis amb les seues diferents implicacions. Es va realitzar des del disseny, el montatge fins a la programació del robot.

Amb la present activitat es desitja conèixer el funcionament d'aquest tipus d'automatisme el qual, hui en dia, té cada vegada major presència en el mercat. Les seues futures projeccions encara són àmplies i li queda encara un bon recorregut. A més és un exemple ideal per a aplicar les diferents matèries que componen la mecatrónica. Amb aquest treball es desitja realitzar una introducció a aquest camp, el qual facilite a l'autor i a aquell que ho desitge poder conèixer aquest tipus de robots.

2. INTRODUCCIÓN

¿Por qué se ha realizado este proyecto de la forma que se ha ejecutado y planteado? Ésta es una de las preguntas que puede hacerse el lector de esta memoria, y tiene una respuesta.

Con el presente trabajo no se desea simplemente seguir una serie de instrucciones predefinidas y conseguir un resultado excepcional gracias al trabajo de otras personas. Lo que se busca es seguir el camino del ingeniero. Aunque este tipo de robots ya existan, se siguen investigando. Se desea conocer cómo funciona desde sus inicios y aprender de este proceso. Como alumno del Máster en Ingeniería Mecatrónica, pretendo poder absorber estos conocimientos desde una perspectiva más experimental y práctica, a la vez que aplico los conocimientos obtenidos durante el transcurso del máster y la investigación realizada para este proyecto, finalmente siendo esto reflejado en la creación de un prototipo. A partir del resultado, se podrá obtener una serie conclusiones, las cuales ayudarán a la ejecución de un próximo prototipo o producto.

2.1 OBJETO

Con el presente proyecto se desea poder ver y aplicar a un caso real los distintos campos y conocimientos adquiridos durante el desarrollo del Máster en Ingeniería Mecatrónica y de este trabajo. Con todo esto se desea crear un prototipo vehículo autobalanceado, sistema mecatrónico.

El robot deberá tener como objetivo sostenerse en pie mediante únicamente el uso de dos ruedas, situadas en sus dos extremos inferiores, asemejándose al caso del péndulo invertido.

Deberá ser de un tamaño relativamente pequeño el cual haga su visualización y manejo sencillo.

Podrá enviar información desde el robot y recibirla en un ordenador.

Se deberá tener en cuenta los materiales y los componentes necesarios para el correcto ensamblaje del proyecto.

Alcance:

- Diseño, construcción y montaje físico del robot, incluida búsqueda y obtención de los materiales necesarios.
- Diseño, desarrollo e implementación de la programación necesaria para el funcionamiento correcto del todos los componentes del robot y de éste mismo en su conjunto.

2.2 JUSTIFICACIÓN

El presente proyecto se presenta con el objetivo de usarse como "Trabajo Final de Máster" de forma que sirva como colofón final a los estudios del título de "Máster en Ingeniería Mecatrónica" de la "Universidad Politécnica de Valencia".
Durante este trabajo se aplicarán de forma real los conocimientos adquiridos durante el transcurso del máster.
Los componentes físicos empleados para el desarrollo del trabajo son una mezcla entre componentes nuevos, modificados y reciclados de formas que se consigue un ahorro en cuestión de costes de ejecución de proyecto. Para la valoración se tendrá en el supuesto como si el proyecto se hiciera fuera y sin recursos de la universidad.

3. FUNDAMENTACIÓN TEÓRICA

3.1 ¿QUÉ ES UN ROBOT AUTOBALANCEADO?

Un robot autobalanceado, como el que se realiza en este caso, se basa en el principio de un péndulo invertido con dos ruedas. Posee una superficie que debe mantenerse en equilibrio para que el vehículo pueda mantenerse en pie y moverse sobre la superficie.

El problema del péndulo invertido ha atraído en interés de muchos investigadores a lo largo de muchos años. Hoy en día se está buscando llevar esta idea cada vez más a una aplicación más móvil. Algún ejemplo serían algunos robots humanoides, medios de transporte personales o sillas de ruedas robóticas.

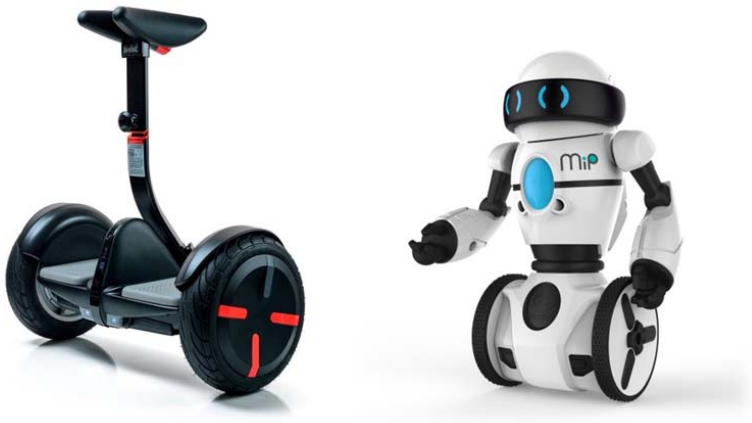


Figura 1 Imágenes de robot autobalanceados

Principio de funcionamiento:

Para poder mantener el robot autobalanceado en una posición rectilínea erecta, los motores deben contrarrestar la caída del robot. Esto quiere decir, que cada vez que el robot se incline en un sentido y otro, los motores deberán ser capaces de actuar de forma que las ruedas giren en sentido de la cabeza ¿Recuerda usted cuando jugaba de pequeño a mantener un palo en la mano de forma que no cayera? El principio es el mismo, el de péndulo invertido, aunque en este caso en lugar de manos tenemos ruedas y motores. Para realizar esta acción es necesaria

una retroalimentación que informe del estado del robot y un factor de corrección que arregle los posibles errores que surjan. En este caso se ha empleado el MKI124V1, unidad electrónica que, entre otras cosas, incluye un acelerómetro y un giróscopo. Éstos informan al microcontrolador sobre la orientación del robot. El factor de corrección de dicha inclinación será en este caso la combinación de motores y ruedas.

Para este caso cuanto más alto se encuentre el centro de gravedad mayor facilidad de control tendrá.

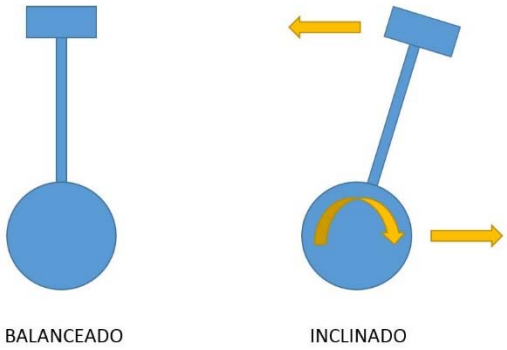


Figura 2Movimiento Robot Autobalanceado

3.2 CONTROL DEL ROBOT

A continuación se muestra y realiza una pequeña introducción a los distintos elementos principales responsables del control del vehículo autobalanceado.

3.2.1 Señal PWM

Para poder mover el motor según consideremos es necesario implementar un control de su velocidad mediante PWM en el caso que se nos presente de motores CC.

Mediante la implementación de una PWM (Pulse Modulation With) es posible establecer una regulación por ancho de pulso del motor CC. Se controla la alimentación del motor mediante el uso de una onda cuadrada. Según el periodo de tiempo que la onda cuadrada se mantenga en alto (Con tensión) el motor se moverá más rápido o menos. Cuando la onda se encuentra en la parte baja la corriente es cero. Mediante esta señal se puede controlar y modificar la cantidad de energía enviada a la carga, en este caso un motor eléctrico.

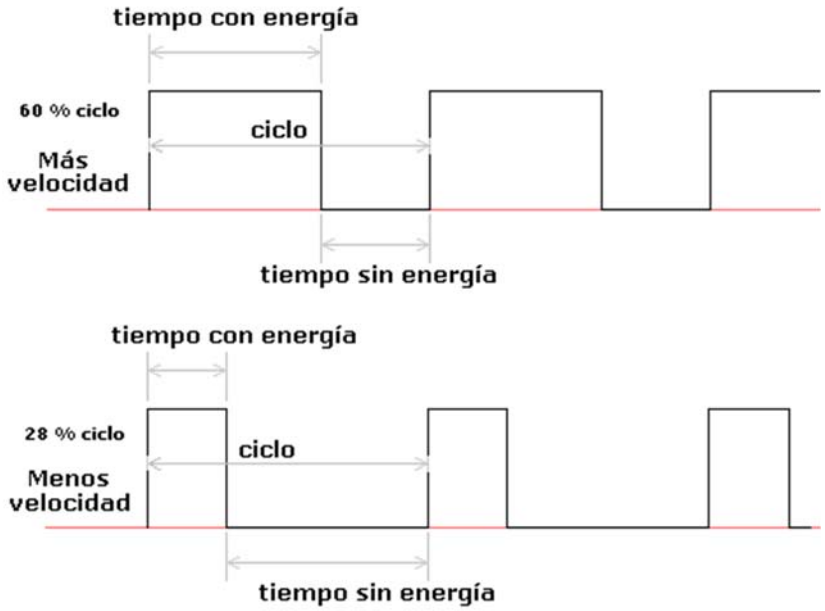


Figura 3 Características PWM

Para nuestro caso particular, después de varios ensayos y pruebas, se ha optado por usar un rango de 0 a 1000 pulsos por ciclo de trabajo.

Para ver más en detalle el desarrollo y obtención de este apartado váyase al apartado de su ensayo correspondiente, "Ensayo 2", en el punto "Programación / Desarrollo del robot" de esta memoria.

3.2.2 Control PD

Se desea poder conseguir que el prototipo reaccione a los cambios que sufra y pueda ejercer un control sobre sí mismo de forma que mantenga el equilibrio sobre sus dos ruedas manteniendo la estructura vertical (A modo de péndulo invertido). Para poder conseguir este objetivo se debe modificar su comportamiento dinámico, pudiéndose hacerlo de dos formas:

- Intervenir en las características físicas del prototipo.
- Emplear un elemento que modifique su relación entrada/salida.

En este caso nos concentraremos en la segunda opción, resumiéndose en un control de tipo Lazo Cerrado PD que explicaremos en este apartado.

En un caso ideal donde se conocen todas las características del sistema a controlar y su entorno siendo también ideal el proceso a controlar sería conocido de forma precisa, simplemente con el modelo matemático implementado el sistema sería capaz de autorregularse en bucle abierto sin información alguna del exterior. Sin embargo, esta situación no se da en nuestro caso, por lo que se hace necesaria una retroalimentación que nos diga lo que está ocurriendo en cada instante.

El tipo de controlador más difundido es el de tipo PID, viniendo su nombre de las siglas de sus tres posibles formas de actuación: Proporcional, Integral y Derivada. Este controlador consiste en ajustar los parámetros de cada parte del propio regulador para cumplir las especificaciones

deseadas. Las acciones básicas que ejerce este tipo de regulador dependen de la cantidad de información recibida. Cuanta más información y mayor sea su capacidad de cálculo, más precisas serán las actuaciones realizadas sobre el proceso.

El controlador decide las acciones a tomar según la diferencia que haya entre la variable de interés, o referencia, y donde se encuentra en ese momento el proceso. Según estas consideraciones el controlador mandará la orden de corregir en mayor o menor grado dicho error entre la referencia y la situación actual.

Tal como se ha comentado, para un control PID existen tres tipos de acciones diferentes que puede realizar:

- **Acción proporcional (P):** Llamada así porque dicha acción es proporcional al valor de la señal de error en cada instante del tiempo. Se tiene en cuenta, por tanto, el signo y la magnitud de la señal. Cuando el error es pequeño la acción reguladora será pequeña, pero sin embargo, cuando el error sea grande la acción reguladora también será de mayor magnitud. Su fórmula viene dada como sigue:

$$u_p(t) = K_p \cdot e(t) \qquad (1)$$

  Siendo "K_p" la constante por la que se multiplica la señal de error e(t) para, así, obtener la acción proporcional.

- **Acción Integral:** Se tiene en cuenta el error acumulado que es el que determina esta acción. Dicha acción contempla que cuanto más tiempo ha pasado sin que el error llegue a su referencia mayor importancia debe tener éste. Usado cuando la acción proporcional se hace demasiado pequeña antes de tiempo y no se acerca lo suficiente a su variable de interés. Su fórmula es la siguiente:

$$u_I(t) = K_I \cdot \int_{\tau=0}^{t} e(\tau)\, d\tau \qquad (2)$$

  Tal como se puede comprobar esta acción es proporcional a la integral de la señal de error, donde su ganancia integral K_i. Dicho integrador, es pues un acumulador de errores que consigue que la acción de control sea cada vez mayor mientras exista error.

- **Acción Derivada (D):** mediante esta acción se tiene en cuenta la tendencia del error, es decir, la velocidad con la que este cambia y como lo hace. Implica realizar una predicción de la situación futura si ésta no cambiara. De esta forma se pueden corregir los errores antes de que ocurran. Para este caso se utiliza esta acción, la cual es proporcional a la derivada de la señal de error y donde se determina la cantidad de acción derivada en el controlador mediante la ganancia "K_d". La fórmula que la caracteriza es tal que:

$$u_D(t) = K_D \frac{d\, e(t)}{dt} \qquad (3)$$

  Se tiene, por tanto, a la derivada de la señal como una predicción de la señal del error.

En nuestro caso usaremos un **control PD**, esto es, Proporcional-Derivativo. La razón de usar este control en nuestro caso es principalmente porque:

- La acción Proporcional ayuda a llegar al valor a la referencia deseado.
- La acción Integral, dada la gran cantidad de oscilaciones, y como se comprobará en el apartado de desarrollo del prototipo, vuelve más inestable el prototipo.
- La acción Derivada ayuda a mejorar la estabilidad del vehículo.

Este regulador aplica, por tanto, las acciones de control proporcional y derivada al error a la vez. Al añadir la acción derivada a la proporcional se mejora la respuesta transitoria del sistema.

A continuación se muestra a grandes rasgos un esquema de cómo funcionaría este tipo de control en el sistema a regular:

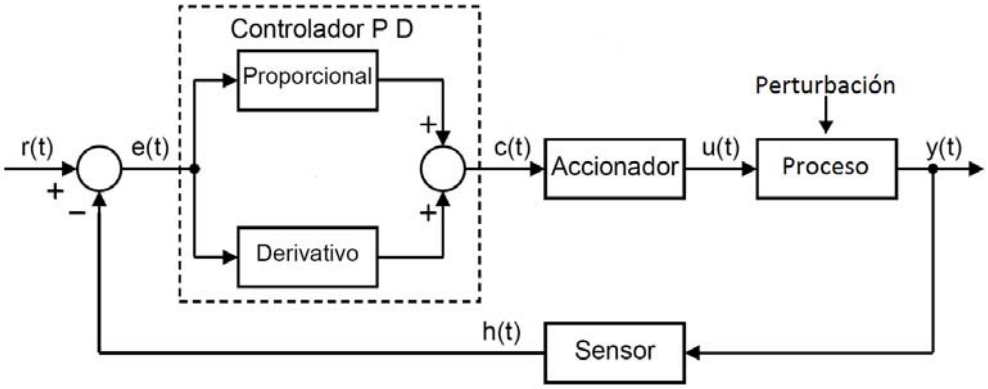


Figura 4 Esquema Control PD

Entre las diferentes formas de aplicar el control anterior al microcontrolador se pueden distinguir dos: El LQR (Regulador Lineal Cuadrático) o el método heurístico.

LQR (Linear-Quadratic Regulator):

Tiene sus orígenes en el trabajo de N.Wiener sobre el filtrado en promedio cuiadrático, aunque no es hasta 1969 que R.E.Kalman desarrolla la teoría del LQR y LQG. Kalman también introdujo la representación en espacio de estados para un sistema dinámico y definió la controlabilidad y observabilidad en dicho marco.

Este tipo de regulador tiene la función de buscar y encontrar de forma automática una realimentación de estado del controlador adecuada y minimizar de este modo la función de coste. Este regulador busca orientarse al control óptimo de señales para nuestro sistema de control con el menor coste de energía posible.

Los ajustes del controlador para regular el sistema se encuentran usando algoritmos matemáticos que minimizan el índice de funcionamiento sopesando los factores aportados a éste. El índice de funcionamiento suele estar definido como una suma de las desviaciones de las medidas clave deseadas.

Por tanto, a la hora de aplicar este tipo de regulador es necesario obtener lo siguiente:

- Un estudio dinámico robusto del sistema a tratar.
- Conversión a función en espacio de estados de dicho sistema.

Se refiere, pues, a un sistema lineal y a un rendimiento cuadrático.

Una vez obtenida dicho información es posible obtener los diferentes factores necesarios para un correcto control.

Sin embargo, este tipo de control no se amolda a nuestro caso y no se hace posible su aplicación por lo siguiente:

- El sistema necesita de un estudio dinámico del sistema, el cual no se puede obtener de una forma fiable, pues hacen falta datos de los motores por parte del fabricante y al poseer un modelo creado de la nada hay gran cantidad de elementos que afectan a las características dinámicas del sistema. Sin tener un estudio exhaustivo de las características físicas de cada componente y del prototipo entero no es posible conseguir el modelo matemático deseado pues falta información necesaria.
- Este tipo de regulador no es muy robusto frente a perturbaciones, sobretodo, en el caso de perturbaciones continuas inherentes a las características del robot. Es decir, fallos por calidades de los materiales o montaje (Motores CC) o perturbaciones inesperadas y no valoradas en los cálculos.

Método heurístico:

Al contrario que en métodos más analíticos donde los parámetros del PID se obtienen mediante modelos matemáticos complejos, este método es más empírico.

El método heurístico consiste en el método de prueba y error siguiendo una serie de normas generales de forma que mediante ensayos se va ajustando poco a poco los parámetros hasta obtener un rendimiento adecuado a nuestros deseos. Este método es usado ampliamente en este tipo de reguladores y su sintonización mejora con la experiencia del ingeniero que los regule.

Este es el método empleado finalmente para la obtención de los valores de las ganancias Proporcional y Derivada. Se han realizado numerosas pruebas con diferentes valores hasta conseguir los valores más adecuados para el control de estabilidad.

Implementación del Control PD:

Para obtener un mejor control del robot se decide implementar dos controles PD, los cuales al sumarse entre ellos darán lugar a la acción correctora de la estabilidad del sistema para que mantenga el equilibrio.

Control PD - Angular:

Este control se basa en el ángulo obtenido por el IMU. Es decir, según la desviación producida en el ángulo del eje vertical principal de la estructura respecto a su referencia, siendo ésta el ángulo 0º, coincidiendo con la vertical. Según el sentido en el que gire la estructura, el ángulo será negativo o positivo. Cuanto más se aleje de la referencia el ángulo aumentará, a su vez el error y por tanto la acción de control aplicada.

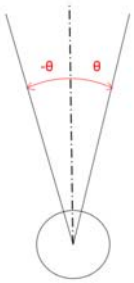


Figura 5 Ángulo estructura

La entrada es el ángulo de la estructura, siendo el error su desviación respecto al ángulo 0º. La salida del sistema es un valor "p_{ang}" el cual sumado al siguiente control forman el valor a aplicar en forma de PWM en los motores. Como la referencia es 0 se opta por usar directamente el ángulo actual como error.

Control PD - Desplazamiento:

Este control se encarga de ver cuando el robot se ha desplazado demasiado sobre el plano en el que se mueven sus ruedas (El suelo, por ejemplo) y devolverlo a su posición original.

Como entrada a este control se introducen los valores de longitud de desplazamiento sobre la superficie medidos mediante el encoder. Se establece como referencia el lugar donde comienza a moverse el robot de forma que cuanto más se desplace en un sentido u otro aumentará el error y por tanto la acción a aplicar.

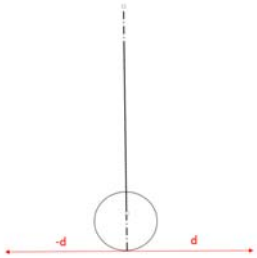


Figura 6 Desplazamiento robot

Como salida al regulador obtenemos los valores de "p_{desp}" el cual junto a la salida del control PD angular se convertirán en la salida final de este doble regulador.

Suma de los dos controles:

Se suman las salidas "p_{ang}" y "p_{desp}" dando lugar al valor "p". Este último valor que va de 0 a 1000, será el que se introduzca como entada para las PWM necesarias para accionar los motores CC y controlar su velocidad y aceleración.

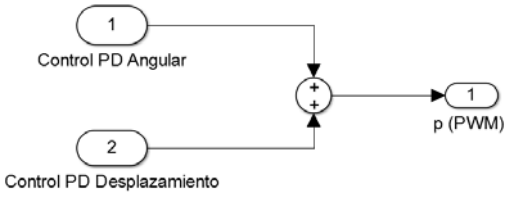


Figura 7 Suma controles PD

Desglosado el control PD del sistema se ve de la siguiente manera:

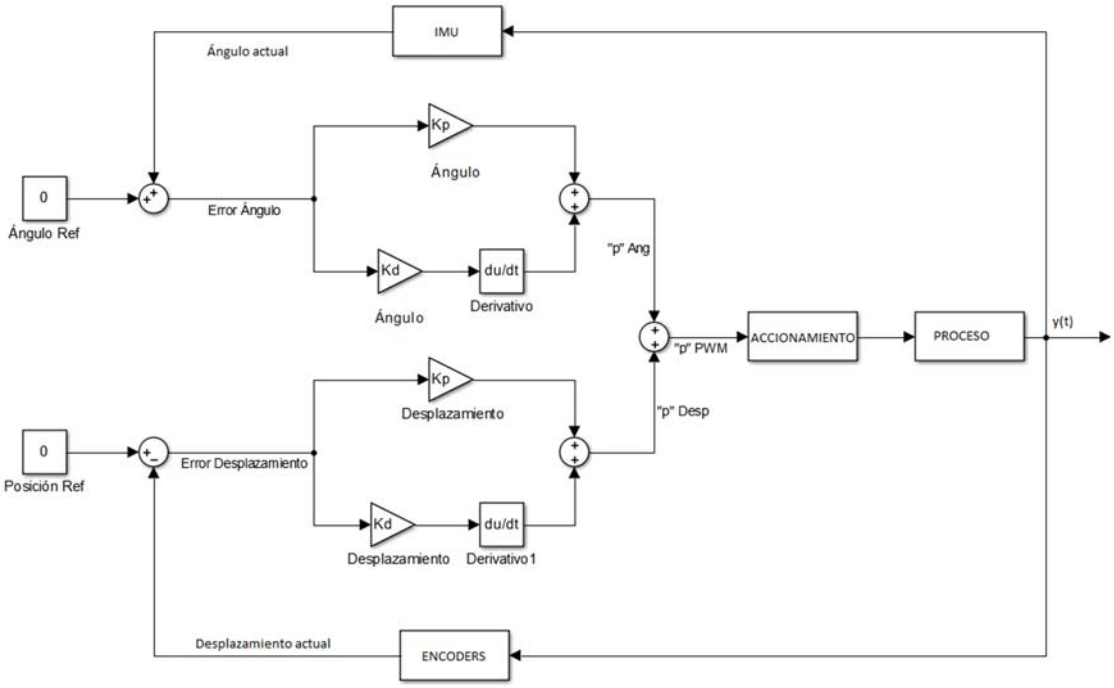


Figura 8 Esquema Control PD del robot

3.3 MODELO MATEMÁTICO

El robot autobalanceado tiene un comportamiento equivalente a un péndulo invertido sobre ruedas. Se han analizado por separado la dinámica de las ruedas y la de la estructura. Esto nos llevará a la obtención de las ecuaciones que describen el movimiento del vehículo.

Para comenzar se analiza las ecuaciones referentes a las ruedas. Como los dos casos son análogos estudiaremos el caso para un rueda, en concreto la derecha.

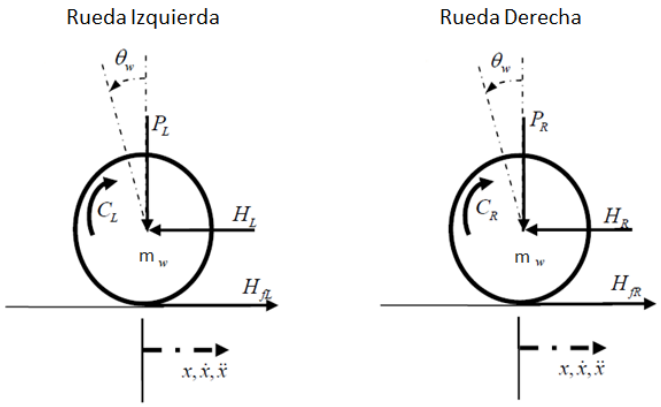


Figura 9 Diagrama de Cuerpo Libre Ruedas

Usando la ley del movimiento de Newton la suma de las fuerzas en el eje horizontal "x" es tal que sigue:

$$\sum F_x = m_w \ddot{x} \quad \rightarrow \quad m_w \ddot{x} = H_{fR} - H_R \tag{4}$$

$$\sum M_o = I \cdot \alpha \quad \rightarrow \quad I_w \ddot{\theta} = HC_R - H_{fR} r \tag{5}$$

Según la dinámica del motor CC, el par del motor puede expresarse:

$$T_m = I_R \frac{d\omega}{dt} + T_a \tag{6}$$

Par de salida de las ruedas es:

$$C = I_R \frac{d\omega}{dt} = \frac{-k_m k_e}{R} \dot{\theta}_w + \frac{k_m}{R} V_a \tag{7}$$

Aplicando las ecuaciones (5) y (7):

$$I_w \ddot{\theta} = \frac{-k_m k_e}{R} \dot{\theta}_w + \frac{k_m}{R} V_a - H_{fR} r \tag{8}$$

$$H_{fR} = \frac{-k_m k_e}{Rr} \dot{\theta}_w + \frac{k_m}{Rr} V_a - \frac{I_w}{r} \ddot{\theta}_w \tag{9}$$

Sustituimos con (9) y (4):

Para la rueda derecha:

$$m_w \ddot{x} = \frac{-k_m k_e}{Rr} \dot{\theta}_w + \frac{k_m}{Rr} V_a - \frac{I_w}{r} \ddot{\theta}_w - H_R \tag{10}$$

Para la rueda izquierda:

$$m_w \ddot{x} = \frac{-k_m k_e}{Rr} \dot{\theta}_w + \frac{k_m}{Rr} V_a - \frac{I_w}{r} \ddot{\theta}_w - H_L \tag{11}$$

Dado que el movimiento lineal actúa en el centro de la rueda, la rotación angular puede transformarse en movimiento lineal según a siguiente relación:

$$\ddot{\theta}_w r = \ddot{x} \quad \rightarrow \quad \ddot{\theta}_w = \frac{\ddot{x}}{r} \tag{12}$$

$$\dot{\theta}_w r = \dot{x} \quad \rightarrow \quad \dot{\theta}_w = \frac{\dot{x}}{r} \tag{13}$$

Sustituimos (13) y (12) en (11) y (10):

Rueda derecha:

$$m_w \ddot{x} = \frac{-k_m k_e}{Rr^2} \dot{x} + \frac{k_m}{Rr} V_a - \frac{I_w}{r^2} \ddot{x} - H_R \tag{14}$$

Rueda izquierda: *(15)*

$$m_w\ddot{x} = \frac{-k_m k_e}{Rr^2}\dot{x} + \frac{k_m}{Rr}V_a - \frac{I_w}{r^2}\ddot{x} - H_L$$

Sumando (14) y (15):

$$2m_w\ddot{x} = \frac{-2k_m k_e}{Rr^2}\dot{x} + \frac{2k_m}{Rr}V_a - \frac{2I_w}{r^2}\ddot{x} - H_R - H_L$$

Simplificando: *(16)*

$$2\left(m_w + \frac{I_w}{r^2}\right)\ddot{x} = \frac{-2k_m k_e}{Rr^2}\dot{x} + \frac{2k_m}{Rr}V_a - (H_R + H_L)$$

La estructura del robot puede estudiarse como un péndulo invertido.

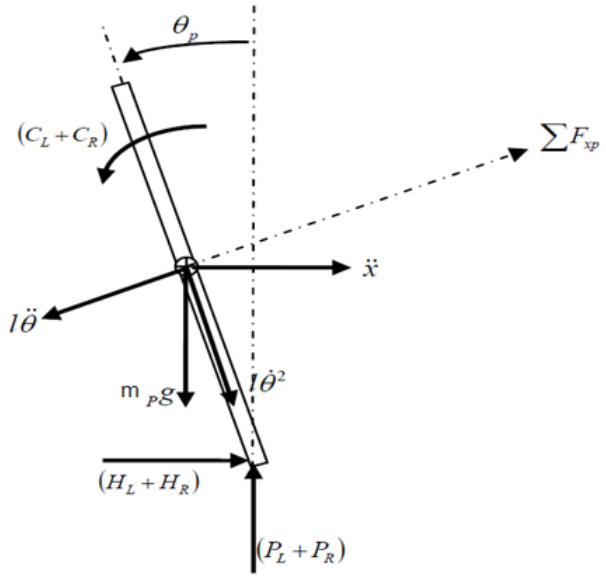


Figura 10 Diagrama de Cuerpo Libre Estructura

Usando la ley del movimiento de Newton la suma de las fuerzas en el eje horizontal "x" es tal que sigue:

$$\sum F_x = m_p\ddot{x} \qquad (17)$$

$$m_p\ddot{x} = (H_R + H_L) - m_p l\,\ddot{\theta}_p cos\theta_p + m_p l\dot{\theta_p}^2 sen\theta_p$$

Por tanto:

$$(H_R + H_L) = m_p\ddot{x} + m_p l\,\ddot{\theta}_p cos\theta_p - m_p l\dot{\theta_p}^2 sen\theta_p \qquad (18)$$

Suma de las fuerzas perpendiculares al péndulo:

$$\sum F_{xp} = m_p\ddot{x}cos\theta_p \qquad (19)$$

$$(H_R + H_L)cos\theta_p + (P_R + P_L)sen\theta_p - m_p g sen\theta_p - m_p l\ddot{\theta}_p = m_p\ddot{x}\,cos\theta_p$$

Suma de fuerzas alrededor del centro de masas:

$$\sum M_o = I\alpha \qquad (20)$$
$$-(H_R + H_L)lcos\theta_p - (P_R + P_L)lsen\theta_p - (C_R + C_L) = I_p\ddot{\theta}_p$$

El par aplicado tal como se define en (7) aplicando (13) teniendo en cuenta ambas ruedas:

$$C_R + C_L = \frac{-2k_m k_e}{Rr}\dot{x} + \frac{2k_m}{R}V_a \qquad (21)$$

Sustituyendo (21) en (20):

$$-(H_R + H_L)lcos\theta_p - (P_R + P_L)lsen\theta_p - \left(\frac{-2k_m k_e}{Rr}\dot{x} + \frac{2k_m}{R}V_a\right) = I_p\ddot{\theta}_p \qquad (22)$$

Por tanto:

$$-(H_R + H_L)lcos\theta_p - (P_R + P_L)lsen\theta_p = I_p\ddot{\theta}_p - \frac{2k_m k_e}{Rr}\dot{x} + \frac{2k_m}{R}V_a \qquad (23)$$

Multiplicamos (19) por "-l" de forma que podemos realizar el siguiente paso:

$$-\left[(H_R + H_L)lcos\theta_p + (P_R + P_L)lsen\theta_p\right] + m_p glsen\theta_p + m_p l^2\ddot{\theta}_p = -m_p\ddot{x}lcos\theta_p \qquad (24)$$

Entonces sustituimos (23) en (24):

$$I_p\ddot{\theta}_p - \frac{2k_m k_e}{Rr}\dot{x} + \frac{2k_m}{R}V_a + m_p glsen\theta_p + m_p l^2\ddot{\theta}_p = -m_p\ddot{x}lcos\theta_p \qquad (25)$$

Sustituimos (18) en (16):

$$2\left(m_w + \frac{I_w}{r^2}\right)\ddot{x} = \frac{-2k_m k_e}{Rr^2}\dot{x} + \frac{2k_m}{Rr}V_a - \left(m_p\ddot{x} + m_p l\,\ddot{\theta}_p cos\theta_p - m_p l\dot{\theta_p}^2 sen\theta_p\right) \qquad (26)$$

Reajustamos (25) y (26), obteniendo las ecuaciones no lineales del sistema:

$$\left(I_p + m_p l^2\right)\ddot{\theta}_p - \frac{2k_m k_e}{Rr}\dot{x} + \frac{2k_m}{R}V_a + m_p glsen\theta_p = -m_p\ddot{x}lcos\theta_p \qquad (27)$$

$$\frac{2k_m}{Rr}V_a = \frac{2k_m k_e}{Rr^2}\dot{x} + \left(2m_w + \frac{2I_w}{r^2} + m_p\right)\ddot{x} + m_p l\,\ddot{\theta}_p cos\theta_p - m_p l\dot{\theta_p}^2 sen\theta_p \qquad (28)$$

Se puede linealizar las ecuaciones (27) y (28) simplificando, asumiendo "$\theta_p=\pi+\varphi$", donde "φ" representa un pequeño ángulo desde la vertical en dirección hacia arriba. De esta forma hace posible un modelo lineal para obtener los controladores del espacio de estados lineal.

Por consiguiente:

$$cos\theta_p = -1; \quad sen\theta_p = -\varphi; \quad \dot{\theta_p}^2 = \left(\frac{d\theta_p}{dt}\right)^2 = 0$$

Las ecuaciones linealizadas son tal que:

$$\left(I_p + m_p l^2\right)\ddot{\varphi} - \frac{2k_m k_e}{Rr}\dot{x} + \frac{2k_m}{R}V_a + m_p gl\varphi = m_p\ddot{x}l \qquad (29)$$

$$\frac{2k_m}{Rr}V_a = \frac{2k_m k_e}{Rr^2}\dot{x} + \left(2m_w + \frac{2I_w}{r^2} + m_p\right)\ddot{x} - m_p l\ddot{\varphi} \quad (30)$$

De forma que se haga posible realizar la representación del espacio de estados del sistema reordenamos (29) y (30):

$$\ddot{\varphi} = \frac{m_p l}{(I_p + m_p l^2)}\ddot{x} + \frac{2k_m k_e}{Rr(I_p + m_p l^2)}\dot{x} - \frac{2k_m}{R(I_p + m_p l^2)}V_a - \frac{m_p g l}{(I_p + m_p l^2)}\varphi \quad (31)$$

$$\ddot{x} = \frac{2k_m}{Rr\left(2m_w + \frac{2I_w}{r^2} + m_p\right)}V_a - \frac{2k_m k_e}{Rr^2\left(2m_w + \frac{2I_w}{r^2} + m_p\right)}\dot{x} + \frac{m_p l}{\left(2m_w + \frac{2I_w}{r^2} + m_p\right)}\ddot{\varphi} \quad (32)$$

Sustituyendo (31) en (30) y (32) en (29) se puede obtener de forma algebraica la ecuación de espacio de estados que seguiría el robot balanceado según el tipo de sistema dinámico que es:

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{\varphi}\\ \ddot{\varphi}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & \frac{2K_M K_e(m_p l\, r - I_p - m_p\, l^2)}{r^2 R\,\alpha} & \frac{{m_p}^2\, g\, l^2}{\alpha} & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{2K_M K_e(r\,\beta - m_p l)}{r^2 R\,\alpha} & \frac{m_p\, g\, l\,\beta}{\alpha} & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ \varphi\\ \dot{\varphi}\end{bmatrix} + \begin{bmatrix}0\\ \frac{2K_m(I_p + m_p\, l^2 - m_p l\, r)}{r\, R\,\alpha}\\ 0\\ \frac{2K_M(m_p - r\,\beta)}{r\, R\,\alpha}\end{bmatrix}V_a \quad (33)$$

Dónde:

$$\beta = \left(2m_w + \frac{2I_w}{r^2} + m_p\right) \quad (34)$$

$$\alpha = \left(I_p\beta + 2m_p\, l^2\left(m_w + \frac{I_w}{r^2}\right)\right) \quad (35)$$

Siendo:

x Desplazamiento.

$\dot{x}$ Velocidad desplazamiento.

$\ddot{x}$ Aceleración desplazamiento.

φ Ángulo

$\dot{\varphi}$ Velocidad angular

$\ddot{\varphi}$ Aceleración angular

4. DESARROLLO Y METODOLOGÍA

4.1 COMPONENTES

4.1.1 Hardware

El hardware es la parte física del sistema, siendo estos los componentes eléctricos, electrónicos, electromecánicos y mecánicos. Para este caso en concreto lo hemos empleado con el significado de "todo aquel componente físico y tangible del sistema".

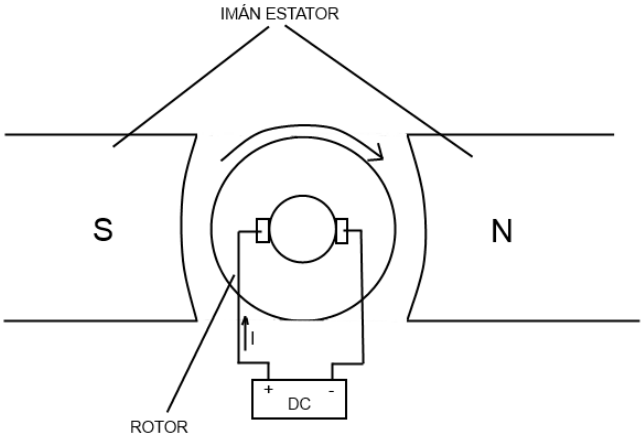
4.1.1.1 Motores CC y Encoders

Figura 11 Motor CC 12V y Encoder

Para este proyecto se han empleado dos motores eléctricos de corriente continua con encoders incluidos.

El **motor CC** convierte la energía eléctrica (Siendo ésta de tipo continuo) en mecánica y está compuesto por dos partes principales:

- **Estator:** parte fija mecánica donde se sitúan los polos del imán.
- **Rotor:** parte móvil con presencia de un devanado y núcleo alimentado de corriente mediante el uso de escobillas.



-

Figura 12 Motor CC

A la circular corriente eléctrica por el devanado el rotor, se genera un campo electromagnético que interactúa con el campo magnético de los imanes del estator. Debido a esto se crea un par de fuerza el cual hace que el motor gire.

Según el sentido de la corriente inducida en el rotor se puede controlar el sentido de giro. Según la polaridad el rotor girará en un sentido o en otro.

También, según la frecuencia de la señal de entrada el motor girará más deprisa o más lento.

Los motores empleados, poseían como única información la tensión a la que debían alimentarse de 12V, la velocidad máxima sin carga de 110 rpm y la configuración de los pines para la conexión de motor y encoder. Respecto a otra información no era dada por el fabricante y toda la que ha sido necesaria emplear ha sido obtenida experimentalmente mediante la realización de pruebas.

Después de los ensayos se ha llegado a la conclusión que los motores empleados poseen un solo par de polos.

Junto al motor viene acoplado un **encoder**. Éste se encarga de medir cuanto gira en el tiempo el eje del motor.

Está acoplado al final del eje del motor y está compuesto principalmente por un disco acoplado a éste y por dos sensores efecto Hall.

Las señales del encoder están en cuadratura. Según el giro del eje del motor se generan dos ondas cuadradas desfasadas 90º, una de cada sensor. Según llegue una onda desfasada antes o después indicará el sentido de giro del motor. Según la frecuencia de dichas señales se podrá saber la velocidad de dicho eje. Contado el número de flacos podemos saber el número de vueltas y cuando las realiza el eje del motor (Contando la totalidad de señales enviadas por el encoder, es un total de cuatro por vuelta)

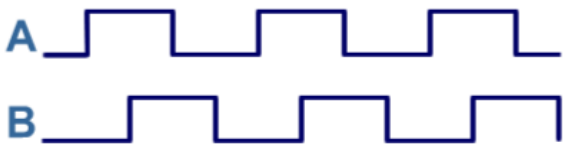


Figura 13 Señales Encoder

Estas son las señales Vout A yB.

Además posee una **reductora**. Mediante ella se reduce la velocidad a la que se mueve el eje de la rueda a la vez que se aumenta el par.

El coeficiente de reducción no venía dado por el fabricante, por lo que fue necesario realizar pruebas experimentales para su obtención.

Nota: A lo largo de la memoria nótese que siempre se hace referencia a dos ejes.

- **Eje del motor:** Eje situado en el rotor del motor CC y del que se mide los pulsos referentes a los giros de éste mediante el encoder.
- **Eje de la rueda:** Eje donde se acopla la rueda. Es el eje de salida de la reductora.

A continuación se observa la imagen de un motor con una disposición igual a la del motor empleado.

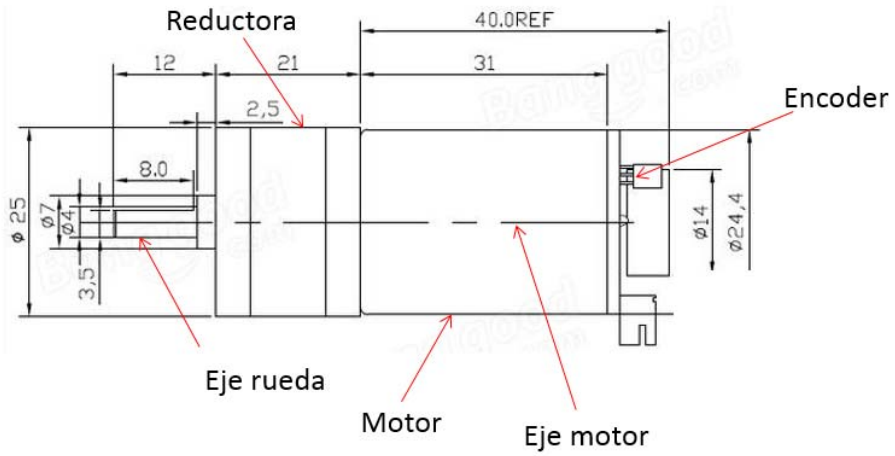


Figura 14 Partes motor CC

4.1.1.2 Etapa de potencia

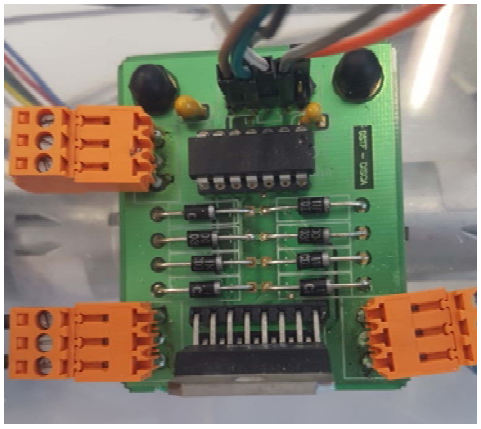


Figura 15 Etapa de potencia

Para el correcto control de los motores CC mediante el STM32 se hace necesaria la presencia de una etapa de potencia o driver. Se hace necesario para poder proporcionarle una mayor corriente al motor mediante una fuente externa, a la vez que podemos controlar la velocidad y sentido el motor mediante el microcontrolador. La mayor tensión de salida que tiene éste es de 5V, mientras que con driver se puede alimentar a los motores con mayor tensión.

En este caso se usa un puente en H. Este sistema sirve para controlar el **sentido de giro** del motor usando 4 transistores. Un ejemplo simple de como estaría configurado es el siguiente:

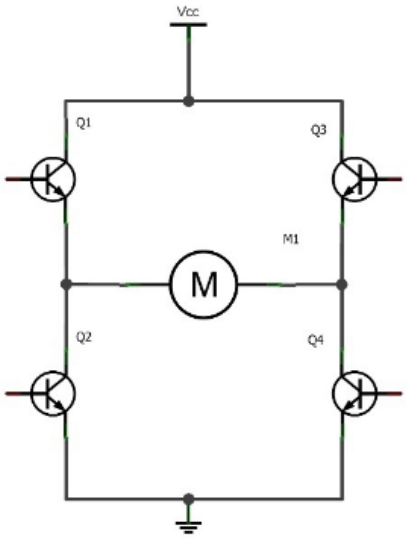


Figura 16 Esquema puente H Etapa de Potencia

Si la corriente circulara por los transistores Q1 y Q4, conduciendo éstos, y Q2 y Q3, estando en corte, el motor giraría en un sentido.

Lo mismo se daría para el sentido contrario, pero siento en este caso Q2 y Q3 los que condujeran y Q1 y Q3 los que estuvieran en corte.

Como la etapa de potencia para este proyecto está diseñada para dos motores, posee dos puentes en H, uno para cada motor.

Para el **control de velocidad** se usa una señal PWM. Ésta se introduce por dos pines creados para tal fin (Uno para cada motor)

4.1.1.3 Batería – Fuente de alimentación

Se ha habilitado un espacio para poder colocar una batería para futuros proyectos. Se tiene en cuenta que es un elemento que será útil y que ejercerá influencia sobre el modelo. Por esta razón se ha incorporado su presencia como elemento que modifica las condiciones físicas (c.d.g) aunque no se emplee para alimentar al vehículo.

Para el caso actual, el robot se alimenta directamente desde la fuente de alimentación para los motores y el puerto USB para el microcontrolador.

Se ha ajustado la salida de la Fuente de alimentación a los 12V necesarios para hacer funcionar los motores según especificaciones.

Figura 17 Fuente de alimentación

La batería se propone implementarla en la parte superior del robot (Para elevar su c.d.g, pues es un elemento que pesa).

Figura 18 Batería

Según la batería, que se ha usado como referencia para el prototipo, de 7.4V de tensión de salida y 1800mAh de capacidad, no sería posible conectarla al microcontrolador ni al motor, pues para e primero la tensión sería demasiado grande y para el segundo, lo contrario. Para solucionar esto será necesario el uso de dos tipos de convertidor CC-CC. Uno tipo “Buck”, el cual bajará la tensión continua a 5V para alimentar el microcontrolador y otro tipo “Boost”, el cual se encargaría de subir a tensión a 12V para alimentar los motores.

4.1.1.4 Microcontrolador

Una de las partes principales a tener en cuenta en el desarrollo de este proyecto es los sistemas embebidos. Un sistema embebido o empotrado es un dispositivo, combinación de software y hardware, diseñado para realizar funciones específicas.

En concreto, un microcontrolador, se trata de un circuito integrado que contiene en su interior lo siguiente:

- **CPU:** Unidad central de procesamiento.
- **RAM y ROM:** Unidades de memoria. La primera, RAM, se emplea para lectura y escritura donde se carga todo aquello necesario durante el momento de funcionamiento. La segunda, ROM, es solo de lectura y generalmente no puede ser

modificada, y se guarda el programa principal. Un ejemplo de esta última es la memoria Flash.

- **GPIOs:** Puertos de entrada y salida.
- **Periféricos:** dispositivo externo conectado al computador que no forma parte de su núcleo básico y que le sirven para comunicarse con el exterior (Sensores, radio, etc)

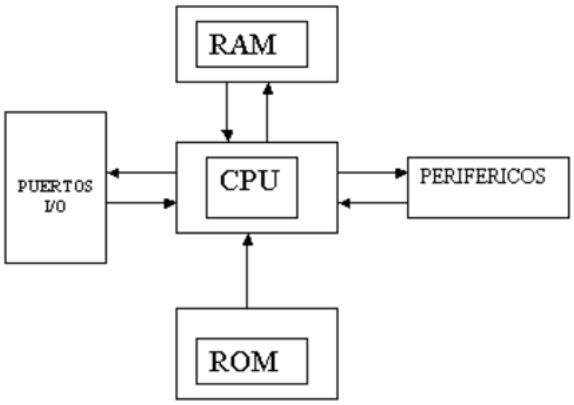


Figura 19 Esquema partes MCU

El microcontrolador se encarga de leer y ejecutar los programas introducidos en él. De esta forma puede controlar todo el proceso que se está dando en el sistema, convirtiéndose así en el cerebro del robot.

Para este caso en concreto se ha empleado un **STM32F411RET6** de 64 pines, perteneciente a la familia de los STM32 Nucleo.

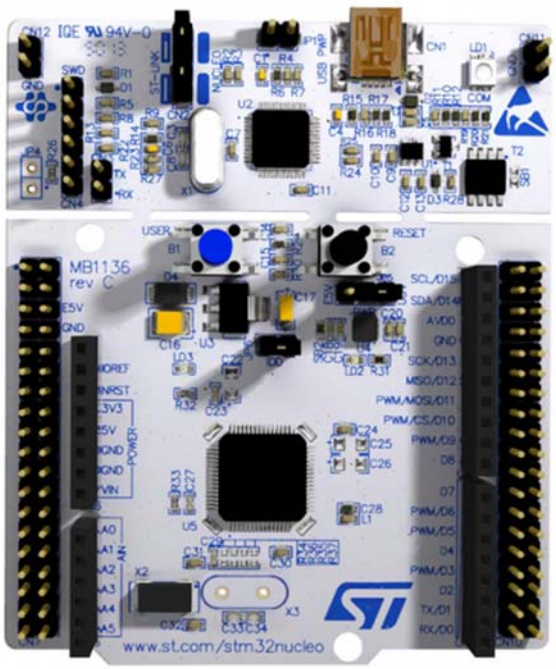


Figura 20 STM32 Núcleo

Características básicas:

- CPU ARM 32 bit Cortex-M4
- 100MHz de frecuencia de CPU máxima.
- 512 kB Flash, 128 kB SRAM.
- Dos tipos de conector: “Arduino” y ST Morpho.
- VDD de 1.7 V a 3.6 V (Aimentación por USB o fuente externa de 5V)

Sus conexiones vienen dadas tal que:

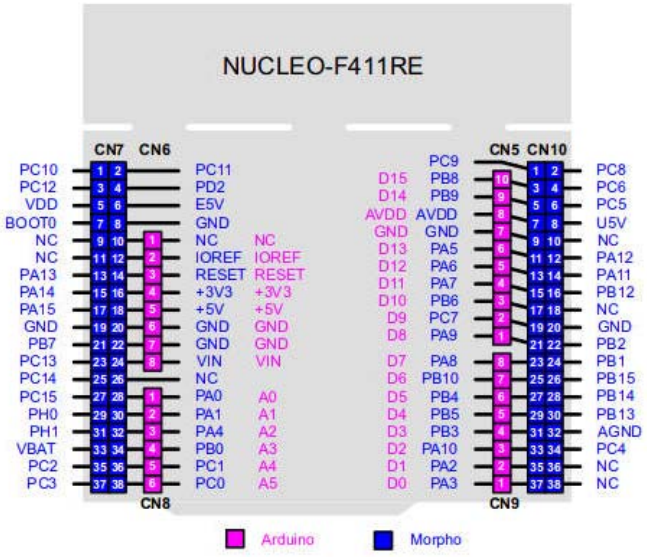


Figura 21 Pines STM32F411RE

Más adelante, en el apartado “Conexiones”, se amplia y concreta en mayor medida este tema.

4.1.1.5 Botonera

Elemento auxiliar empleado durante los ensayos para obtención de resultados. Se ha empleado para comprobación de funcionamiento y de errores del proyecto. Se explica su aplicación más a fondo en el apartado de desarrollo del robot.

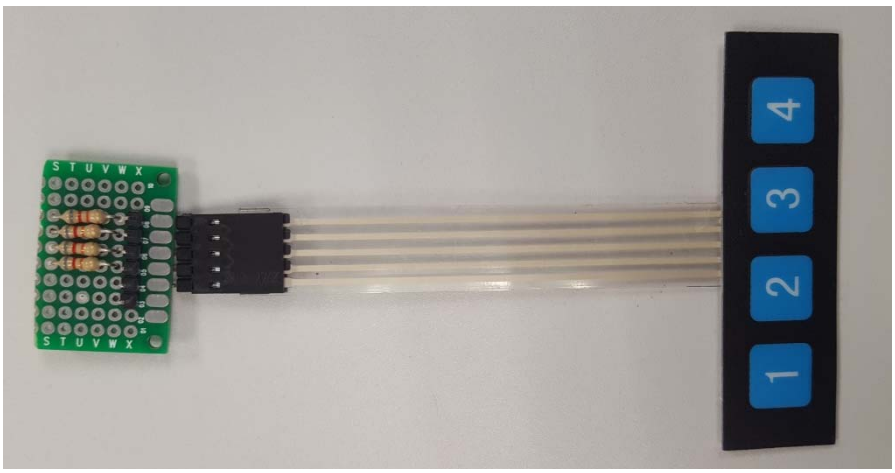


Figura 22 Circuito más botonera

La botonera se emplea para introducir los comandos deseados por el usuario.

Se crea y suelda un circuito para poder utilizar la botonera y conectar los respectivos pines, manteniéndolos activos o inactivos según situación.

A continuación se puede ver un **esquema unifilar** de éste.

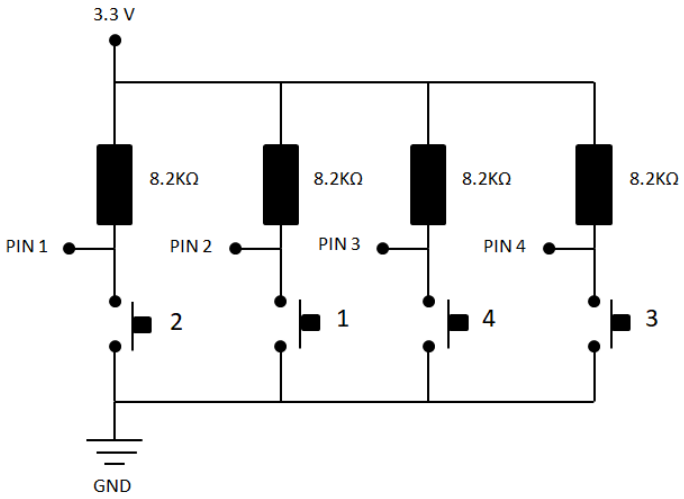


Figura 23 Esquema circuito para botonera

Siendo los números indicativos de cada botón marcado en la botonera, tal como se puede ver en la imagen primera de este punto. Con un multímetro se obtuvo que el orden de los pines de izquierda a derecha era GND, 2, 1, 4 y 3.

Ciruito real:

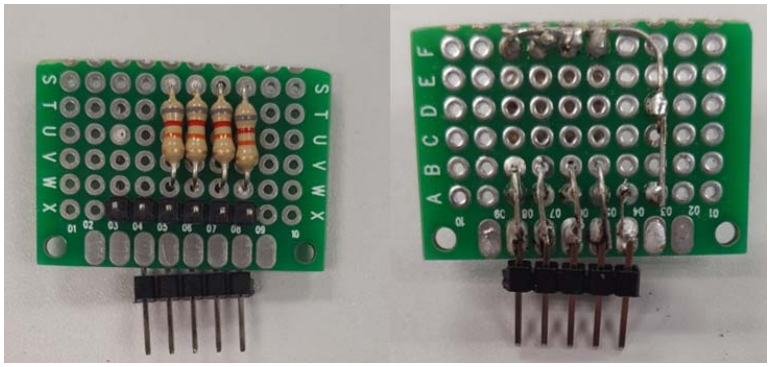


Figura 24 Circuito botonera visto desde arriba y abajo

4.1.1.6 Unidad de Medida Inercial (IMU)

La unidad de medida Inercial con sus siglas en inglés IMU (Inertial Measurement Unit) es un circuito integrado con la capacidad de medir e informar sobre velocidad, orientación y fuerzas gravitacionales.

Para este caso se ha empleado el módulo **STEVAL-MKI124V1**.

De este dispositivo se ha usado para las mediciones dos componentes:

- **L32GD20:** Giroscopio.

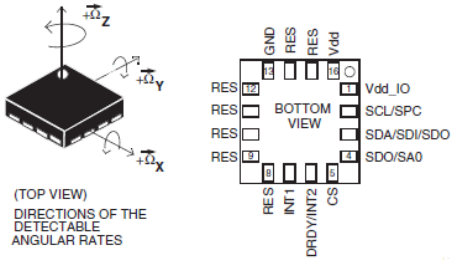


Figura 25 Pines L32GD20

El giroscopio mide la velocidad angular, a partir de la cual se puede obtener el ángulo. Es más preciso que el acelerómetro pero acumula error con el tiempo.

- **LSM303DLHC:** Acelerómetro.

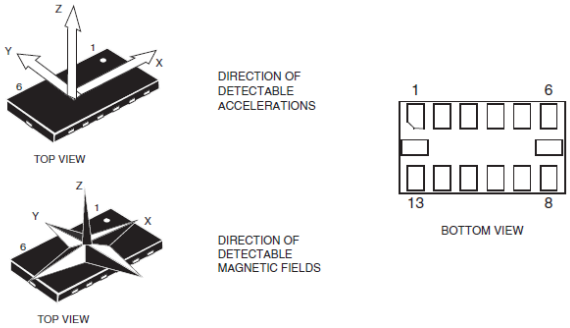


Figura 26 Pines LSM303DLHC

El acelerómetro, como su nombre indica, mide aceleraciones. En nuestro caso por caída del vehículo en un sentido u otro. Como más adelante se comprobará se podrá obtener la posición angular respecto a estos datos según una serie de relaciones físicas. Es menos preciso que el giroscopio en movimiento, su señal es ruidosa pero no acumula más error en el tiempo como el giroscopio.

El IMU se comunica con el microcontrolador mediante I2C.

Sistema I2C:

Es un bus serie de datos (inter-Integrated Circuit) empleado para comunicar diferentes partes de un circuito. En este caso se ha utilizado para comunicar el IMU con el microcontrolador, STM32. Dentro de los que es el sistema de comunicación I2C el SDA es la señal de datos. Y el SCL es la señal de reloj.

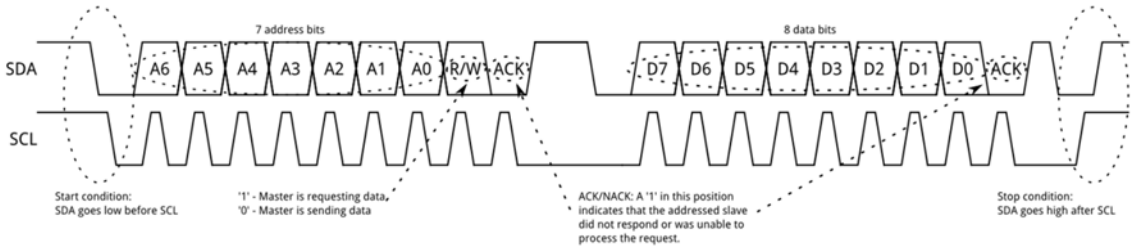


Figura 27 I2C

La señal de arranque viene dada en el SDA siendo un flanco descendente y, a su vez, el SCL se encuentra en nivel alto.

Dado que no poseía **resistencias de "Pull-up"** hubo que añadírselas para que pudiera funcionar. Conectan la entrada Vdd con las salidas SDA y SCL, del I2C. Estas resistencias tienen como función elevar una tensión de entrada o salida de un circuito lógico mientras este esté en reposo. Sirven para evitar lecturas erróneas si el pin ya no tiene nada conectado o no está recibiendo señal.

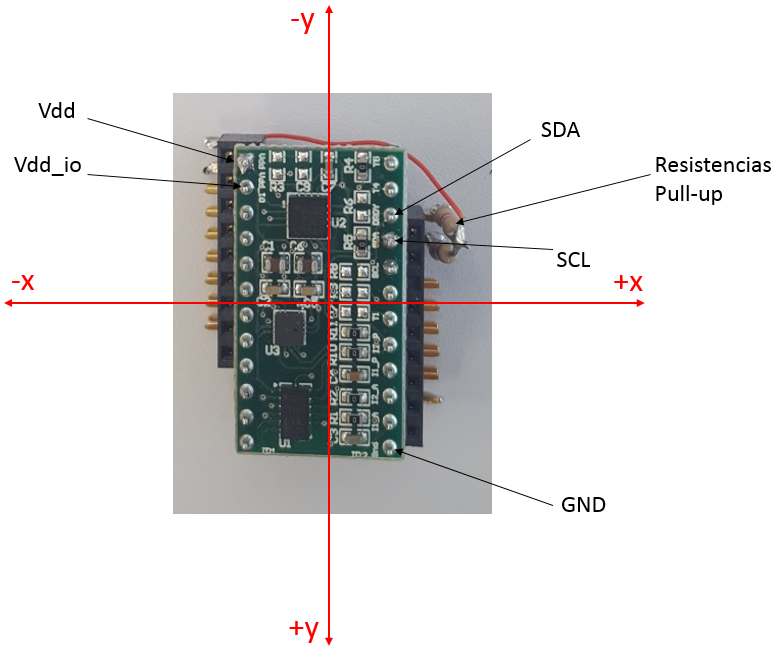


Figura 28 Pines y orientación IMU

En esta imagen se muestra la localización de las diferentes conexiones. Vdd y Vdd_io siendo la alimentación del STEVAL-MKI124V1 y sus entradas/salidas respectivamente. En rojo se muestra los sentidos de los ejes planos (x,y). Según hacia donde incline el signo de la señal cambia. El eje z se sitúa perpendicular a la placa.

4.1.1.7 Módulo de radio

Figura 29 Módulo RF XBee

Este tipo de módulo de radiofrecuencia (RF) proporciona conectividad inalámbrica de largo alcance para redes digitales. Puede utilizar frecuencias de hasta 868MHz. Posee prácticamente ninguna interferencia a la hora de comunicarse entre módulos.

Estos módulos RF se han empleado para comunicar el vehículo autobalanceado (STM32F4) con el usuario.

Las unidades empleadas y estaban pre-configuradas mediante su programa específico y se le asigna la frecuencia a la que transmitirán.

Se conectan a una antena para poder emitir su señal y ser transmitida correctamente.

La configuración física empleada es tal que así:

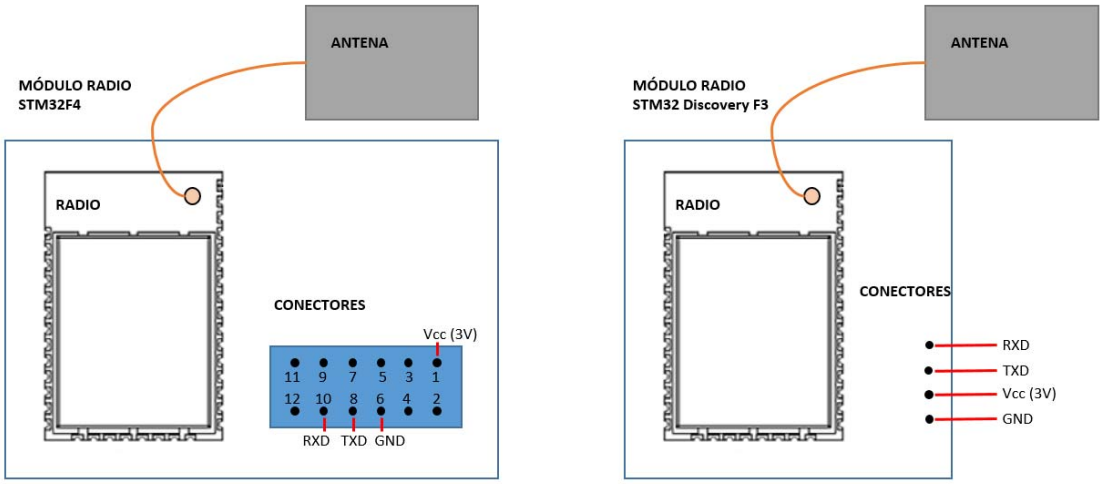


Figura 30 Conectores Módulo RF

Para este proyecto se han empleado dos módulos "**XBee XB8-DMUS-002**" con sus respectivas antenas. El que se encuentra dentro del vehículo, conectado al STM32F4 hace de emisor y el que se encuentra conectado al STM32 Discovery F3 hace de receptor para pasar la información a pantalla por el convertidor USB. Ambos módulos RF pueden funcionar tanto como receptor como transmisor.

4.1.1.8 Estructura

Como estructura entendemos a todo aquello que compone la parte física y tangible del robot y a la configuración, diseño y tipo de montaje de éste.

Se ha optado por una estructura flexible en su diseño. Con esto no se quiere puntuar que sus elementos tengan como propiedad inherente la flexibilidad, sino que se trata de un formato de montaje de piezas fácil de montar y desmontar y en caso de ser necesaria poder realizar alguna modificación de éste si la situación así lo requiere.

Posee dos ruedas en su base con los motores que las accionan. Éstas harán la función de desplazar el vehículo.

La estructura conectada a las ruedas es alargada, en forma de estantería, de forma que se pueda poner, quitar y observar de forma sencilla los diferentes elementos de los que consta el robot.

El prototipo está configurado principalmente de la siguiente forma:

- Base estantería: Motores, ruedas y etapa de potencia.
- 1er piso estantería: Microcontrolador y radio.
- 2º piso estantería: Batería, IMU.

La razón de crear la estructura principal de forma alargada es la de elevar el c.d.g. lo máximo posible. La importancia de este aspecto viene dada por lo siguiente:

Cuanto más alto esté el c.d.g. más lento se vuelve el sistema, por lo que es necesaria una acción menos rápida para compensar la caída. Por tanto, cuanto más bajo esté el c.d.g. más rápido es el sistema y caerá con mayor velocidad, lo que hará que sea necesaria una acción más rápida.

4.1.2 Software

Entendemos por software al conjunto de componentes lógicos empleados para la realización de tareas específicas. Esto quiere decir, aquellas aplicaciones informáticas empleadas en el transcurso del proyecto.

Los programas de ordenador principalmente usados para la consecución del prototipo son los siguientes:

- **STM32CubeMX:** Programa de desarrollo de programación de STM. Se emplea para la pre-configuración del microcontrolador. Se establecen GPIOs, se configuran TIMERs, etc. Mediante este programa se genera el código base para programar el microcontrolador y demás archivos necesarios para su funcionamiento.

- **Keil µVision:** Entorno de programación “C”. Mediante esta aplicación se puede programar el microcontrolador, montar, depurar y cargar el código deseado en éste.

Integrated Development Environment

- **Termite:** Programa empleado para la visualización en pantalla de las variables enviadas por el STM32.

- **Logic:** Aplicación usada para la visualización de señales mediante el analizador lógico a modo de osciloscopio simplificado.

Además de estos programas se han empleado otros tantos para labores auxiliares, como por ejemplo la escritura de esta memoria. También se debe hacer mención a la cantidad de drivers obtenidos desde las webs de los fabricantes y que son necesarios para el correcto funcionamiento de periféricos y aplicaciones informáticas empleadas

4.2 CONEXIONES

Se han realizado conexiones de los periféricos a dos microcontroladores.

- El primero y principal, el **STM32F411RE**. Este se encuentra dentro del vehículo autobalanceado y sirve como base para todo el sistema. A lo largo de la memoria, cuando se haga referencia al STM32 se estará haciendo mención a este microcontrolador.
- El secundario, el **STM32 Discovery F3**. Este se usa de forma auxiliar. En nuestro caso se ha empleado para alimentar la radio receptora y el convertidor USB conectado a ésta. Se ha usado una implementación ya existente, como se puede observar en la imagen del Ensayo 9, de la radio. Esto es, pues para ya tener una pre-configuración para las ampliaciones por parte de otros alumnos del presente proyecto. A este microcontrolador se le puede conectar un "joystick" para controlar por radio el robot, por ejemplo, enviar órdenes por botonera…o para cualquier otra aplicación que se tenga en cuenta. A lo largo de la memoria, en caso de querer hacer referencia a este microcontrolador se hará de forma específica y con todo su nombre.

Conexiones STM32F411RE:

Como eje central para la programación del prototipo tenemos que tener en cuenta las conexiones a realizar con el cerebro del robot, esto es, con el microcontrolador.

A continuación se pueden observar dos imágenes donde se muestra la localización física de los diferentes tipos de conectores del STM32F411RE.

Pines tipo ST Morpho:

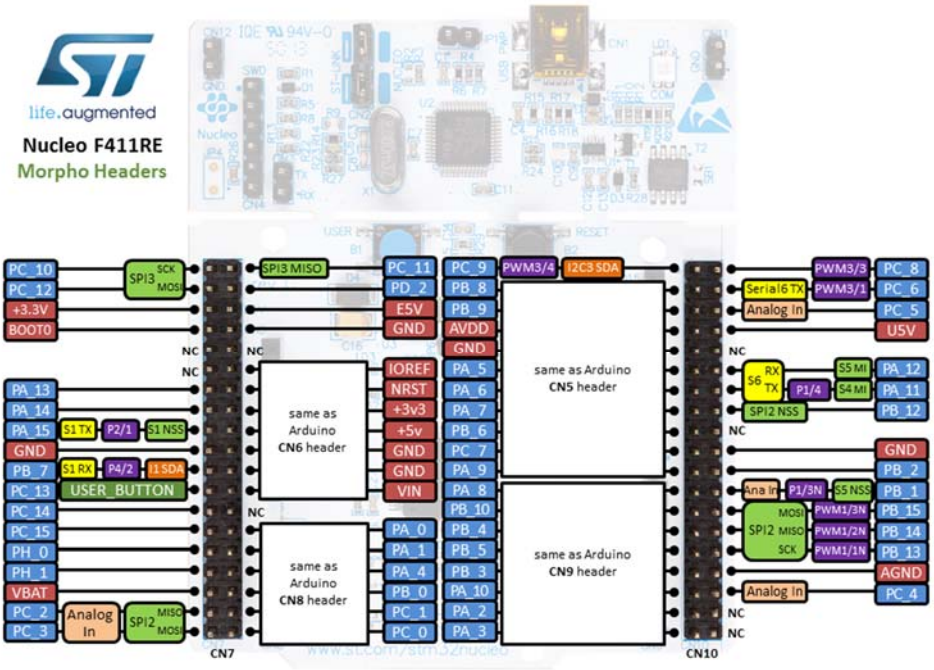


Figura 31 Pines ST Morpho STM32F4

Pines tipo Arduino:

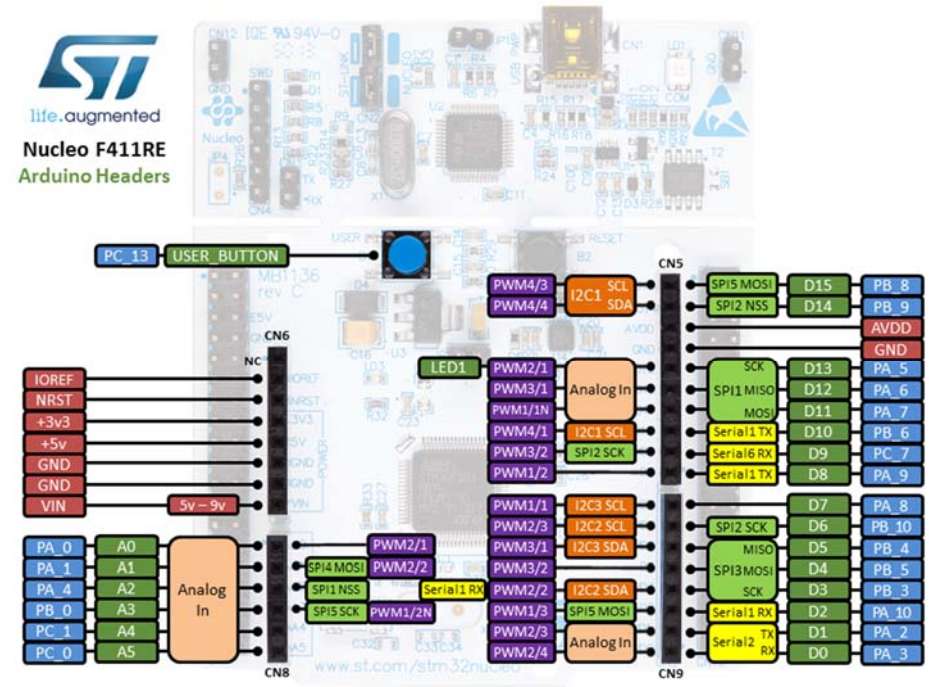


Figura 32 Pines Arduino STM32F4

Conexiones STM32 Discovery F3:

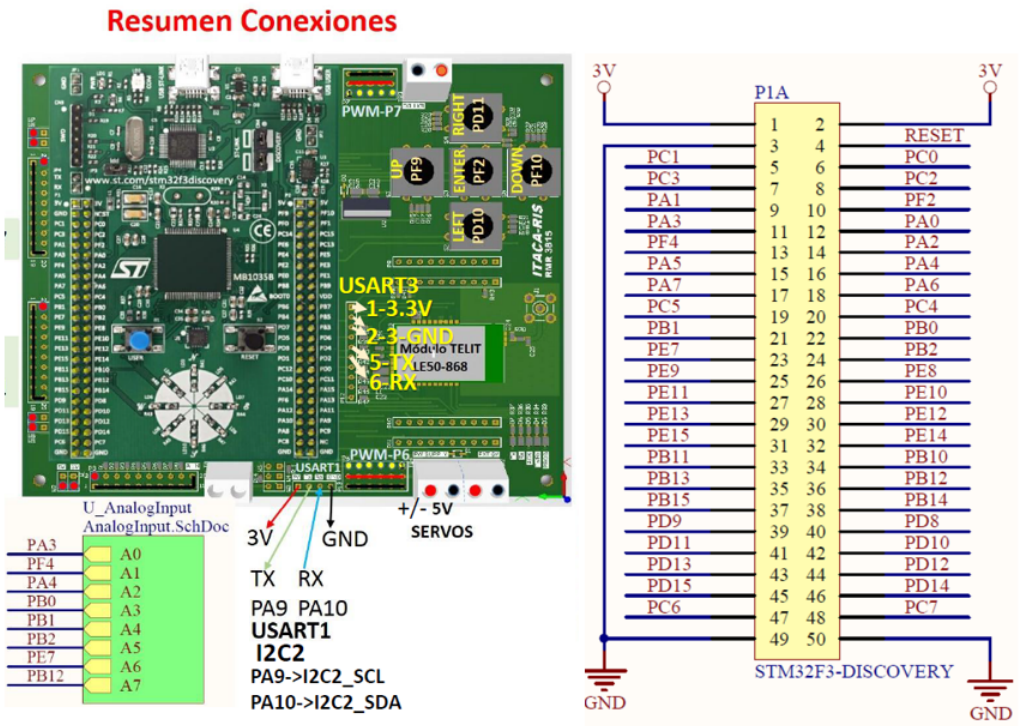


Figura 33 Pines STM32 Discovery F3

Para la totalidad del proyecto, se ha realizado una **tabla resumen** que clarifica las distintas conexiones realizadas para realizar el prototipo en su totalidad.

CONEXIONES STM32F411RE

MONTAJE FINAL

	PERIFERICO 0	Descripcion 0	PERIFERICO 1	Descripcion 1	STM32	CN	Descripcion STM
MOTOR 1	1	M1+	V1+	Etapa de Potencia			
	2	M1-	V1-	Etapa de Potencia			
	3	GND S.Hall			6	6	GND
	4	5V S.Hall			5	6	5V
	5	Vout (A) S.Hall			23	10	PA8 (TIM1_CH1)
	6	Vout (B) S.Hall			21	10	PA9 (TIM1_CH2)
MOTOR 2	1	M2+	V2+	Etapa de Potencia			
	2	M2-	V2-	Etapa de Potencia			
	3	GND S.Hall			20	7	GND
	4	5V S.Hall			18	7	5V
	5	Vout (A) S.Hall			28	7	PA0 (TIM2_CH1)
	6	Vout (B) S.Hall			30	7	PA1 (TIM2_CH2)

STM32 INTERNAS	8 (CN5)	PA8 (TIM1_CH1)			10	5	PB8 (TIM10_CH1)
	1(CN6)	PA0 (TIM2_CH1)			9	5	PB9 (TIM11_CH1)
ETAPA DE POTENCIA	V1+	V1+	1	M1+			
	V1-	V1-	2	M1-			
	V2+	V2+	1	M2 +			
	V2-	V2-	2	M2 -			
	12V	12V Entrada Motores	V1+, V2+	Etapa de Potencia			
	GND	GND Entrada Motores	V1 -, V2 -	Etapa de Potencia			
	2	5V			8	10	5V
	3	Sentido 1			34	10	PC4 (GPIO_Out)
	5	Sentido 2			6	10	PC5 (GPIO_Out)
	7	PWM Motor 1			4	10	PC6 (TIM3_CH1)
	9	PWM Motor 2			19	10	PC7 (TIM3_CH2)
	10	GND			20	10	GND
IMU	1	VDD			16	7	3V3
	2	VDD_IO			5	6	3V3
	13	GND			7	6	GND
	20	SCL			17	10	PB6 (I2C1_SCL)
	21	SDA			21	7	PB7 (I2C1_SDA)
RADIO (Nucleo)	1	3V3			7	10	3V3
	6	GND			19	7	GND
	8	TXD			33	10	PA10 (USART1_RX)
	10	RXD			17	7	PA15 (USART1_TX)
RADIO (Dicovery F3)	1	3V3			3V	USART1 (Modulo F3)	3V3
	6	GND			GND	USART1 (Modulo F3)	GND
	8	TXD	RX	Conversor USB (Discovery F3)			
	10	RXD					
CONVERSOR USB (Discovery F3)	GND	GND			3	P1A	GND
	RX	RX	TXD	Radio (Discovery F3)			

OTRAS CONEXIONES (Ensayos)

	PERIFERICO 0	Descripcion 0	PERIFERICO 1	Descripcion 1	STM32	CN	Descripcion STM
CONVERSOR USB (Nucleo)	GND	GND			19	7	GND
	RX	RX			17	7	PA15 (USART1_TX)
BOTONERA	1	3V3			16	10	3V3
	2	GND			22	7	GND
	3	BOTON 2			36	7	PC1 (GPIO_Input)
	4	BOTON 1			38	7	PC0 (GPIO_Input)
	5	BOTON 4			37	7	PC3 (GPIO_Input)
	6	BOTON 3			35	7	PC2 (GPIO_Input)
SENSOR HALL	SIG	Salida Señal			1	7	PC10 (GPIO_Input)
	Vcc	Vcc			5	6	3V3
	GND	GND			7	6	GND

Leyenda Tablas Conexiones

Periférico 0	Periférico de origen
Periférico 1	Periférico de destino
STM32	Nº de pin en el STM32
CN	Bloque de pines del STM32, excepto especificación Discovery F3
#	Nº de motor DC
S.Hall	Sensores efecto Hall (Encoder)
M# +	Entrada alimentación motor positiva
M# -	Entrada alimentación motor negativa
V# +	Salida alimentación motor positiva
V# -	Salida alimentación motor negativa
Sentido #	Cambio de sentido de giro
3V3	3.3 V
GND	Masa (Ground)

Además, se puede comprobar en el apartado de anexos el documento generado mediante el STM32CubeMX donde entre otros aspectos se puede confirmar este aspecto.

A continuación se puede ver la configuración de los GPIOs del STM32F4.

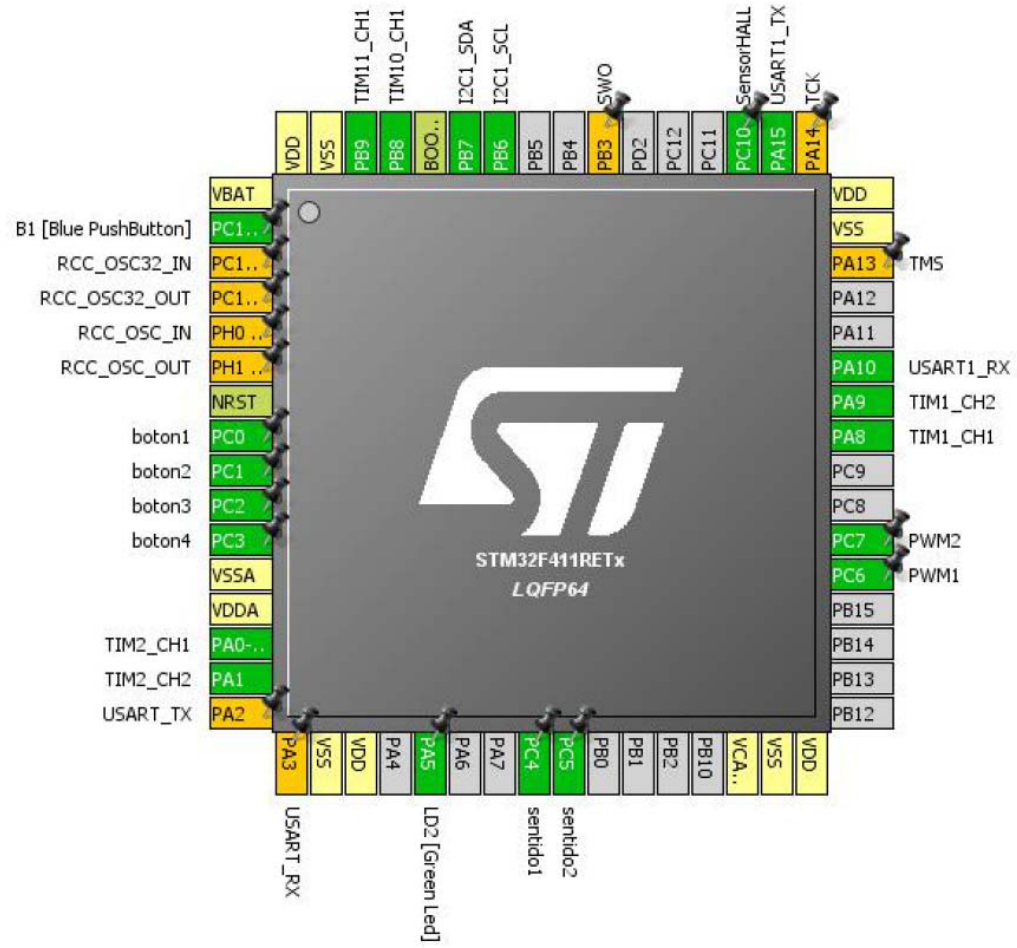
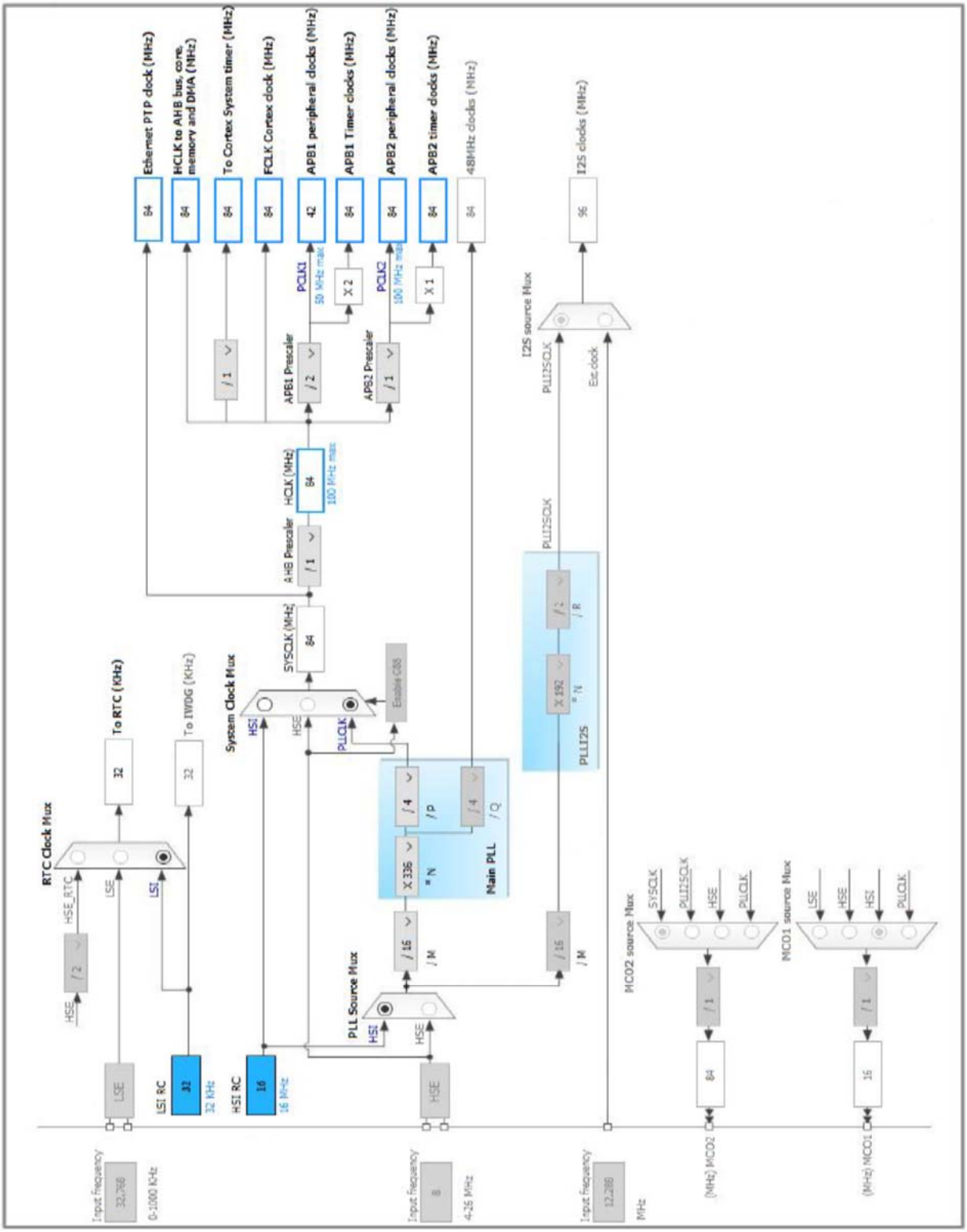


Figura 34 CubeMX - Configuración de los pines

Además, se puede comprobar en el siguiente esquema la disposición del árbol de “clocks” del microcontrolador.



4.3 PROGRAMACIÓN / DESARROLLO DEL ROBOT

4.3.1 Planteamiento general

Durante el desarrollo del prototipo, y dado el gran número de incógnitas a manejar, ha sido necesario una serie de numerosos ensayos para poder:

- Conocer funcionamiento de la herramienta de programación y microcontrolador.
- Ajuste y conocimiento del funcionamiento de periféricos y accionamientos.
- Obtención de características de elementos no dadas por el fabricante.
- Configuración y obtención de variables internas del microcontrolador.
- Realización de medidas y calibraciones.
- Corrección de errores.
- Pruebas de funcionamiento del prototipo y sus elementos.
- Pre-configuración de partes del vehículo y creación elementos de programación final de éste.

Durante el transcurso de esta empresa se han tenido que hacer gran número de cambios. Por esta razón se ha optado por un modelo flexible que permita el cambio de piezas, corrección de errores y calibraciones tanto de hardware como de software en tiempo real. De esta forma se agiliza mucho más el desarrollo del prototipo y los diferentes ensayos necesarios para su culminación.

Aunque en algunos casos los ensayos se han tenido que hacer de forma aislada para una mejor ejecución, de forma general se han desarrollado de forma escalonada y progresiva. Esto quiere decir que cada nuevo ensayo era un sumatorio del anterior como base más él mismo de forma que cada vez se iba ampliando el programa y sumando nuevas funcionalidades. Según lo requiriera cada ensayo, se bloqueaban o eliminaban partes de programa anterior.

En el programa final han sido necesarias múltiples revisiones, para optimizarlo. De esta forma, se han eliminado variables innecesarias (Sí necesarias en los ensayos anteriores, pero ya no útiles en la versión final, pues ya han cumplido su función previa). También se ha optimizado el código, eliminando partes innecesariamente largas o con procesos que, simplificados, obtienen el mismo resultado consumiendo menor capacidad del procesador.

Para la parte de hardware, según el experimento realizado y sus requerimientos, se usaba unos periférico/accionamientos u otros. Únicamente ajustándose a cada situación, uno podía obtener los mejores resultados posibles para cada caso y así poder, al final, englobarlo todo en lo que es el presente proyecto.

El programa final está compuesto por varios “script” de los cuales, unos ya venían predefinidos por el programa CubeMX cuando éste genera el proyecto, y otros has sido creados, añadidos o modificados.

Para finalizar y comenzar con la explicación de cada ensayo, se desea mostrar al lector las siguientes **consideraciones previas** respecto a éstos:

- Los ensayos, en orden general, se muestran en orden cronológico a su realización.
- Algunos poseen “sub-ensayos”, es decir, se han realizado varios ensayos distintos sobre el mismo tema central. Por esta razón, se ha optado en englobarlos en un mismo grupo.

- A pesar de lo antes mencionado, y con carácter extraordinario, algunos test, aunque distintos se han realizado en paralelo por exigencias de la situación (Por ejemplo, configuración del Controlador y modificación de la estructura del robot).

Nota: Para consulta de conexiones váyase al apartado previo correspondiente a tal temática.

4.3.2 Ensayo 1: LED controlado mediante una Botonera

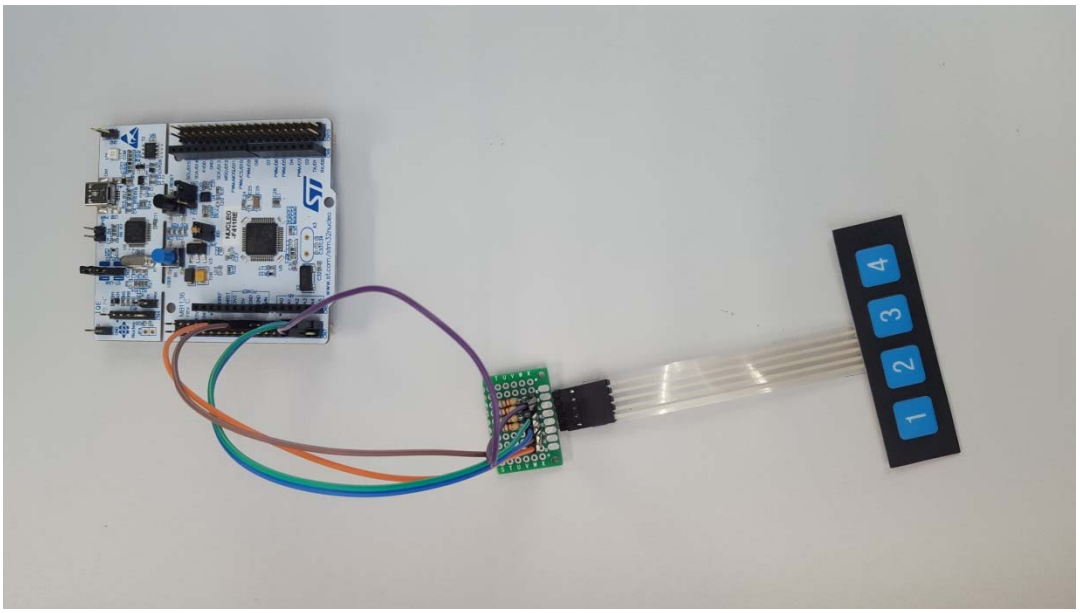


Figura 35 Montaje Ensayo 1

Componentes empleados:

- STM32.
- Botonera.
- Circuito con resistencias.
- MCU con LED incorporado.

La botonera se emplea para introducir los comandos deseados por el usuario. Ésta será necesaria para futuros ensayos, por lo que se hace necesaria su configuración e incorporación temporal.

El circuito se crea para poder utilizar la botonera y conectar los respectivos pines, manteniéndolos activos o inactivos según situación.

El microcontrolador servirá como base para el control, procesamiento y conexión de los diferentes elementos, incluido el LED, que ya lleva incorporado de fábrica.

Desarrollo:

Éste se trata de un programa sencillo para familiarizarse con el entorno de programación C, las diferentes herramientas de programación y conocer la fisionomía y comportamiento del microcontrolador.

La única intervención física es la conexión de los pines y la creación del circuito de resistencias para la conexión de la botonera.

Para comenzar, se establecen las entradas y salidas tanto en el plano físico como en el programa "STM32CubeMX" para su posterior configuración.

Una vez escogidas se genera un nuevo código para el presente ensayo. Este código se gestiona, desarrolla y modifica mediante el programa "Keil µVision".

El programa consta de dos partes principales:

- Programa para encender el LED.
- Programa para leer información desde la botonera.

La función principal del programa es que el LED se comporte de diferente forma según el botón pulsado. Esto es:

- **Botón 1:** Enciende LED.
- **Botón 2:** Apaga LED.
- **Botón 3:** Parpadea LED.

Se comprueba experimentalmente el correcto funcionamiento del programa y la botonera. Ésta última se ha testeado previamente mediante un multímetro, de forma que se confirma que los pines se activan o desactivan según el botón presionado.

4.3.3 Ensayo 2: Control de velocidad de un motor CC mediante Botonera

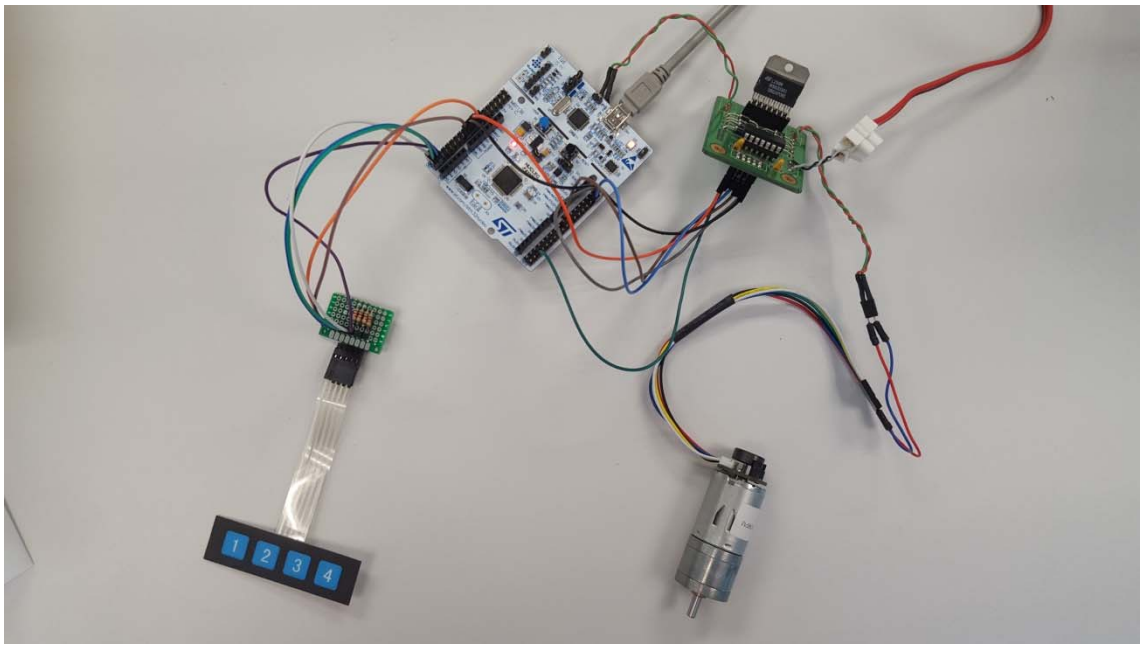


Figura 36 Montaje Ensayo 2

Componentes empleados:

- STM32.
- Botonera.
- Fuente de alimentación (12 V).
- Etapa de potencia.
- Motor CC.

La fuente de alimentación se emplea para poder alimentar el motor de corriente continua. Éste se conecta a la etapa de potencia y de ésta a los motores.

Se hace necesaria la incorporación de una etapa de potencia para el control del motor CC.

Se usa como accionamiento un motor CC el cual solo tiene conectadas las entradas de alimentación a la etapa de potencia.

En la siguiente imagen se muestran las conexiones del motor y encoder para este ensayo y próximos.

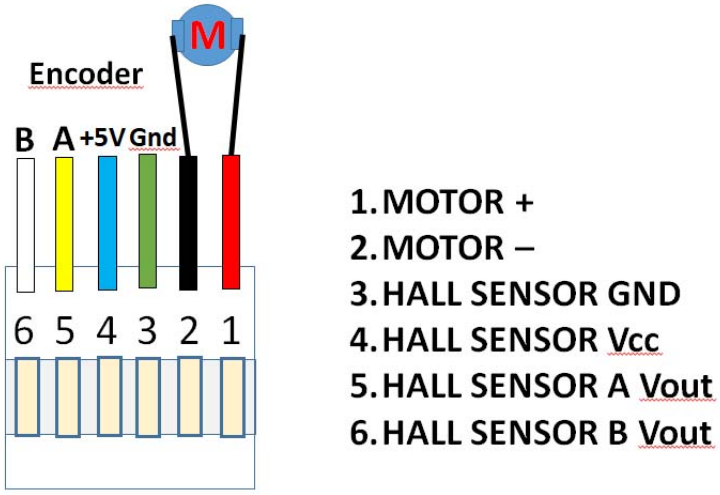


Figura 37 Pines motor CC

Desarrollo:

En este ensayo se tiene como objeto configurar la onda PWM para poder mover el motor CC. Para su comprobación se crea un programa con las siguientes funcionalidades, basándose en los botones pulsados cada vez:

- Boton 1: Aumentar velocidad.
- Botón 2: Disminuir velocidad.
- Botón 3: Cambio de sentido de giro.
- Botón 4: Parar motor.

Para comenzar, se establecen las entradas y salidas tanto en el plano físico como en el programa “STM32CubeMX” para su posterior configuración, además e configurar el Timer 3, ch1&2 en modo PWM (Un canal para cada motor, pensando en que en el futuro se emplearán dos, aunque en esta prueba solo se use uno).

Después se debe calcular el valor del "Autoreload" y el "Prescaler" para configurar la señal PWM, responsable de la regulación de velocidad del motor. La obtención de este apartado se explica más adelante en este mismo punto de la memoria.

Una vez escogidas se genera un nuevo mediante el programa "Keil µVision"donde se termina de crear el código necesario para el funcionamiento del sistema.

Se comprueba experimentalmente la correcta generación de la onda PWM mediante un "Analizador Lógico". En el caso particular de este proyecto, se trataba de un "Sealae Logic".

Además, se confirma experimentalmente que efectivamente, el motor realiza las acciones deseadas, aumentando, disminuyendo, parando o cambiando de sentido según deseo.

Obtención de la señal PWM:

Para poder mover el motor y modificar su velocidad, se usará un control por PWM. Controlando el tiempo en el que la señal se mantiene activa (Alta) o inactiva (Baja) podremos mover el motor según las especificaciones que deseemos. Esto es, cuanto más grande sea el ciclo de trabajo, o "duty" en inglés, más rápido irá el motor.

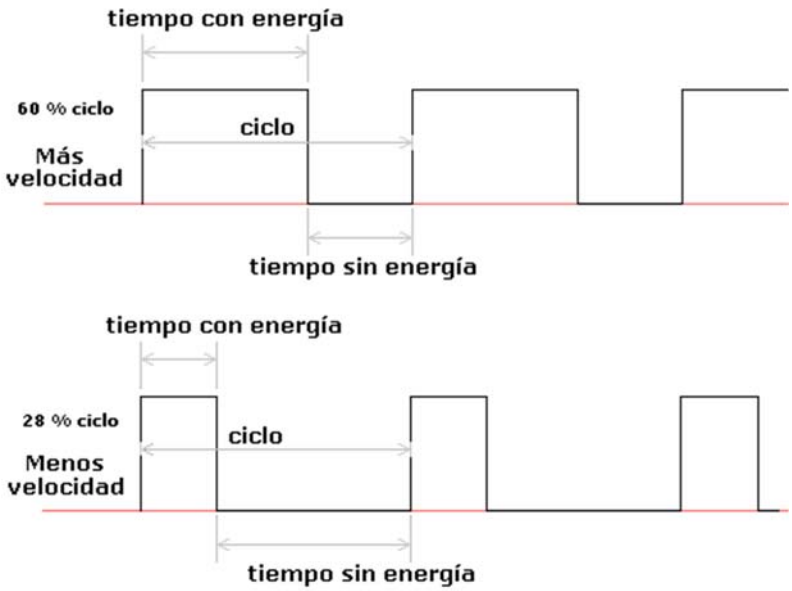


Figura 38 Señal PWM

Para procedes a configurar los parámetros necesarios para obtener la señal PWM deseada se ha de tener en cuenta lo siguiente:

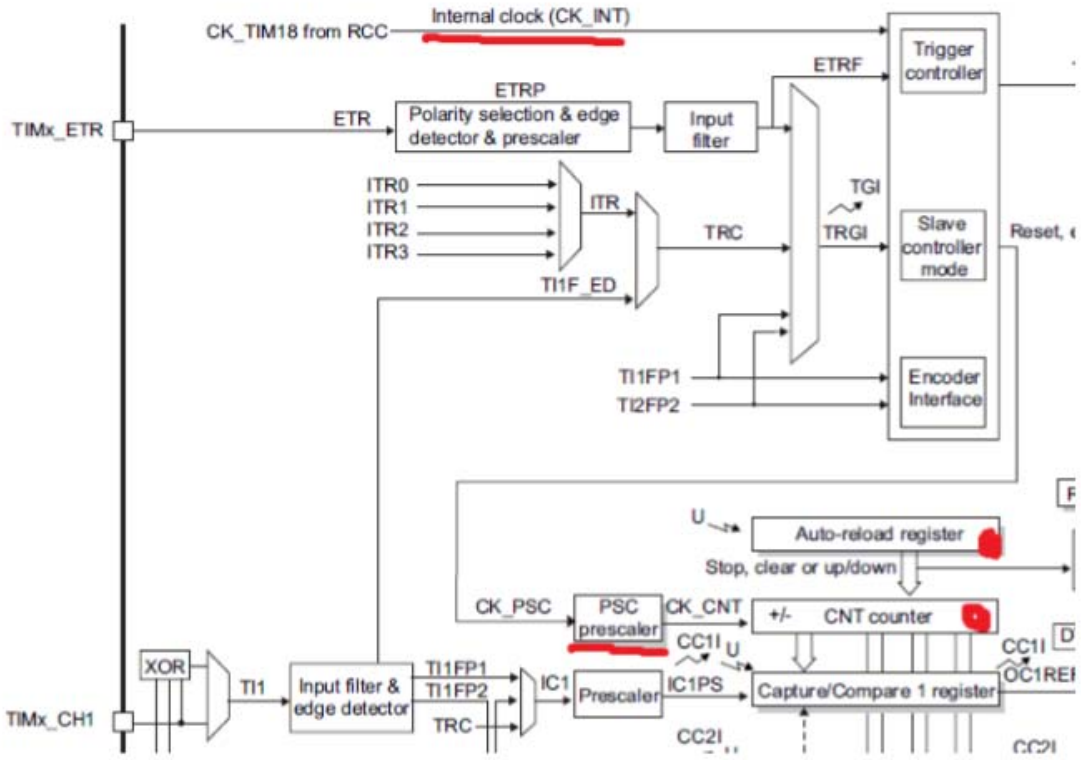


Figura 39 Esquema interno CLK STM32

Cojamos como base la siguiente imagen simplificada:

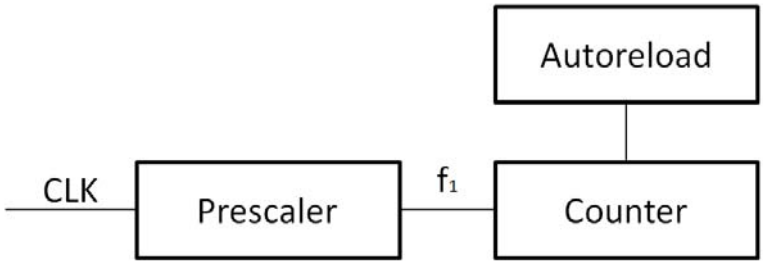


Figura 40 Esquema Simplificado CLK

Para empezar se debe considerar usar un Timer, en este caso el Timer 3, usando los canales 1 y 2, uno para cada motor. Se debe conocer que el reloj, CLK (Clock) interno del microcontrolador, en este caso el STM32F411RE, tiene una frecuencia de 84MHz. Esto quiere decir que el microcontrolador posee un reloj que va generando un flanco de subida de señal cuadrada con una frecuencia de 84MHz.

Tengamos ahora en cuenta los siguientes aspectos dentro de la programación:

- **Contador (Counter):** tal como su nombre indica, va aumentando valores o restándolos, desde 0 hasta el valor máximo o viceversa. Una vez el contador llega a su valor máximo o mínimo reinicia la cuenta, coincidiendo esto con el inicio de un nuevo ciclo de trabajo.
- **Prescaler:** Es el divisor de frecuencia para el contador. Para este caso, puede tener valores que van de 0 a 65535.

- **Autoreload:** O CounterPeriod, indica el máximo de cuentas a realizar por el "counter". Al igual que el prescaler, puede tener valores que van de 0 a 65535.

Estos valores se deben configurar según el resultado que deseemos.

Inicialmente se quería generar una PWM con las siguientes características:

- Una frecuencia de 10kHz
- Un periodo que contemplara 100 cuentas. Esto se debe a que se deseaba en un inicio que fueran equivalentes el número de cuentas con el porcentaje del ciclo de trabajo deseado (0-100%).

Los valores a introducir para modificar el valor de la PWM son equivalentes al valor comprendido por el Autoreload, siendo en este caso entre 0 y 100. "T" es el valor del periodo entre señales.

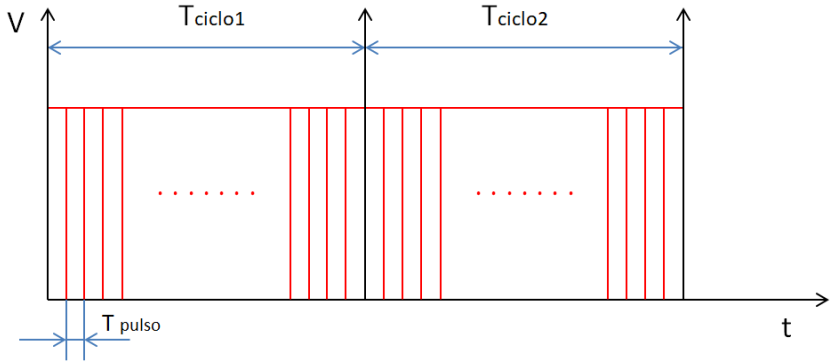


Figura 41 T de PWMs

Para obtener nuestro objetivo:

$$f_{counter} = 10\ kHz \quad (36)$$

$$T_{ciclo} = \frac{1}{f_{counter}} = \frac{1}{10 \cdot 10^3} = 0.0001 = 100\ \mu s \quad (37)$$

Para este tiempo de 10ms se tiene que conseguir hacer un total de 100 pulsos, que es el número de cuentas que deseamos introducir en el Autoreload.

$$T_{pulso} = \frac{T_{ciclo}}{Valor\ Autoreload} = \frac{100 \cdot 10^{-6}}{100} = 0.000001 = 1\ \mu s \quad (38)$$

Aquí podemos saber que cada pulso generado, esto es, cada valor del contador durará un total de 1 µs. Por tanto,

$$f_1 = \frac{1}{T_{pulso}} = \frac{1}{1 \cdot 10^{-6}} = 1000000 = 1MHz \quad (39)$$

De aquí obtenemos finalmente que:

$$Valor\ Prescaler = \frac{f_{CLK}}{f_1} = \frac{84 \cdot 10^6}{1 \cdot 10^6} = 84 \quad (40)$$

Por tanto, los valores que debemos introdujimos inicialmente en el programa son:

- Prescaler = 84.
- Autoreload = 100.

Puesto que los valores introducidos para la variación de la PWM han de ser siempre enteros, como más adelante se observará, si existen muchos decimales, estas pequeñas variaciones se pierden, por lo se hace muy poco exacto el valor introducido, pues se redondea directamente al valor entero. Para mayor sensibilidad y exactitud a la hora de introducir, se ha optado por modificar el valor del Autoreload y por consiguiente el valor del Prescaler, de forma que se mantengan las cualidades deseadas, se pierde un poco de velocidad de generación de picos (No afecta al sistema, es despreciable) pero obteniendo a cambio una mayor precisión a la hora de aplicar las salidas obtenidas de implementar el control de estabilidad en el robot. Por tanto, los valores quedarán al final del proyecto tal que:

- Prescaler = 84.
- Autoreload = 1000.

Con este cambio conlleva lo siguiente:

$$f_1 = \frac{f_{CLK}}{Valor\ Prescaler} = \frac{84 \cdot 10^6}{84} = 1MHz \quad (41)$$

$$T_{pulso} = \frac{1}{f_1} = \frac{1}{1 \cdot 10^6} = 1 \cdot 10^{-6} = 1\,\mu s \quad (42)$$

$$T_{ciclo} = T_{pulso} \cdot Valor\ Autoreload = 1 \cdot 10^{-6} \cdot 1000 = 1 \cdot 10^{-3} = 1\,ms \quad (43)$$

$$f_{counter} = \frac{1}{T_{ciclo}} = \frac{1}{1 \cdot 10^{-3}} = 1\,kHz \quad (44)$$

```
void MX_TIM3_Init(void)
{
  TIM_MasterConfigTypeDef sMasterConfig;
  TIM_OC_InitTypeDef sConfigOC;

  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 84;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 1000;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
```

“Init.Period” es el equivalente al Autoreload en el código.

Analizador lógico:

El analizador lógico tiene una función similar a la que realiza un osciloscopio. Se trata de un instrumento de medida para la visualización de señales en múltiples canales. Permite la

comprobación del correcto funcionamiento de un sistema digital midiendo cambios de nivel, números de estados lógicos, etc.

Para nuestros casos concretos, en este ensayo y posteriores, se ha empleado principalmente a modo de osciloscopio. Para la comprobación de errores en la señal, correcta generación de señales cuadradas o PWM, y medición de tiempos y frecuencias en señales.

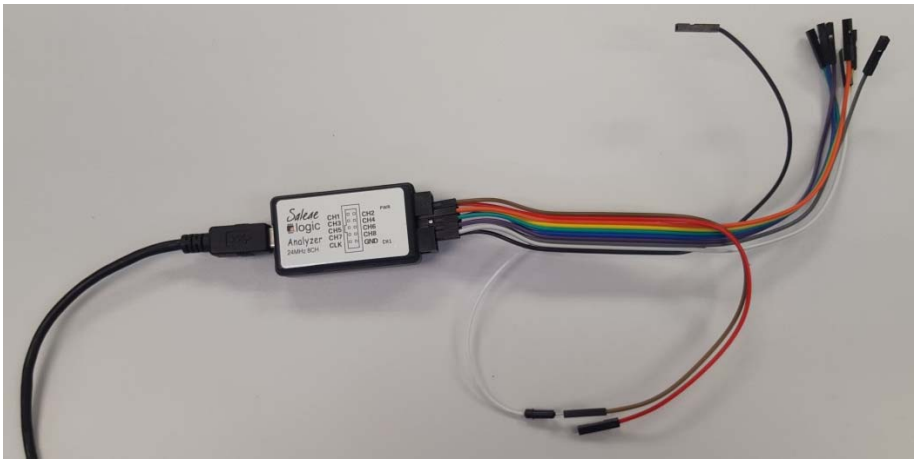


Figura 42 Analizador lógico

Para su funcionamiento requería de las siguientes acciones:

- La primera es conectar los canales deseados a sus respectivos pines del circuito a medir.
- Conectar a GND el cable correspondiente.
- Conectar la salida USB al ordenador empleado para la visualización.
- Configurar el tipo de señal a leer y los canales en el programa “Logic” del analizador.

Se realiza la comprobación de señal PWM generada mediante la visualización de ésta en el programa “Logic” del ordenador.

Nota: Durante el transcurso de este experimento se ha descubierto que el motor CC posee una zona muerta en vacío que llega desde su 0% al 30% de su ciclo de trabajo.

También se ha notado que al girar el eje del motor existe cierta holgura en los engranajes de la reductora, lo cual imposibilita un cambio de sentido suave. Crea otra zona muerta, esta vez producida por ese hueco en el que el eje de la rueda no será capaz de moverse inmediatamente hacia el sentido contrario y le generará incrementos en el momento de inercia situaciones en las que no debería

Aunque en este ensayo aún no se pudiera detectar de forma clara el impacto que generarían estos desperfectos, sobretodo el segundo se sabe por ensayos posteriores lo siguiente: Esto generará una dificultad añadida cuando más tarde se desee realizar el control de estabilidad del robot. Y como luego se comprobará, el efecto de estos defectos de fábrica, se incrementará sobremanera cuando posea el peso añadido de toda la estructura y necesite moverla.

4.3.4 Ensayo 3: Obtención de número de pulsos desde Encoder

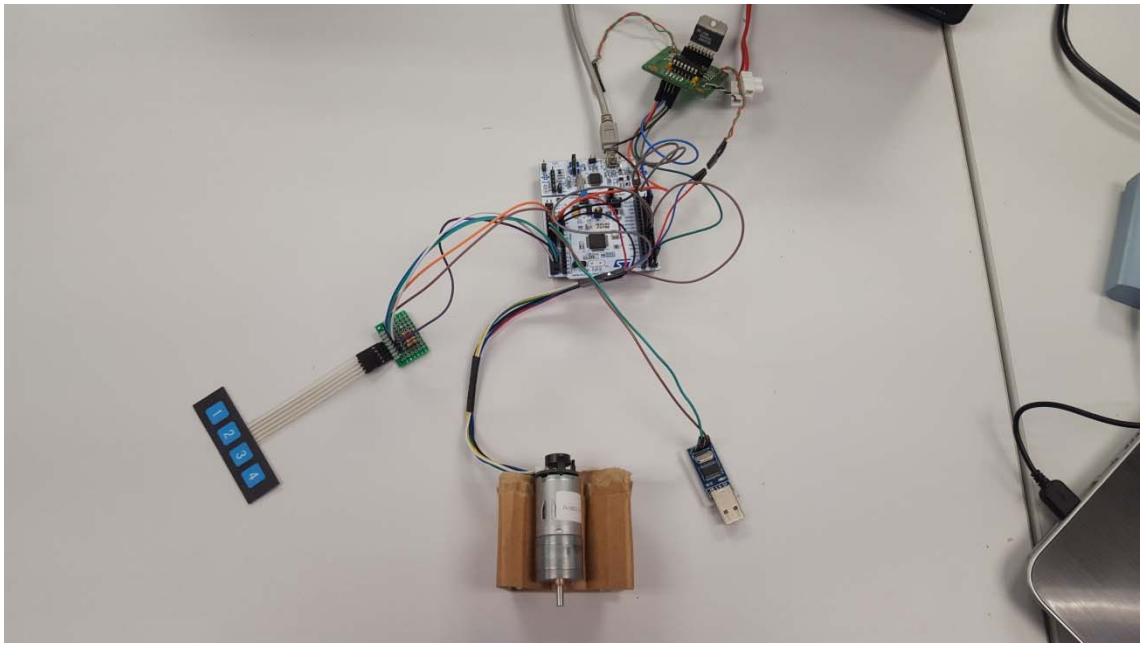


Figura 43 Montaje Ensayo 3

Componentes empleados:

- STM32.
- Botonera.
- Fuente de alimentación (12 V).
- Etapa de potencia.
- Motor CC y Encoder.
- Convertidor USB.

El encoder se empleará para medir el número de pulsos generado en éste producido por el giro del eje del motor. Por tanto, en esta ocasión sí se conectan todos los pines que salen del motor al microcontrolador.

El convertidor USB se encarga de transmitir la información mediante un puerto tipo **USART** (Universal Synchronous/ Asynchronous Receiver/ Transmitter). Se trata de un tipo de microchip que facilita la comincación a través de un puerto serie usando el protolo RS-232C, usado como estándar para transmisión de datos comunicando computadores y aparatos relacionados.

Para este caso debemos tener en cuenta dos partes principales.

- Conectar el GND del puerto Usart.
- Poner el pin encargado de la transmisión de datos del STM32 en modo TX (Transmisor) pues es el que emite la información y conectar ese pin al conector RX (Receptor) del puerto USB, pues este recibe la información del microcontrolador.

Este puerto se usará para transmitir las variables que deseemos y visualizarlas durante el experimento en pantalla.

Para la visualización de las variables se emplea el programa “Termite”.

Desarrollo:

En esta prueba se desea conocer el funcionamiento del encoder y configurar el programa de forma que sea capaz de leer los pulsos generados por éste.

Tal como se ha comentado en su apartado correspondiente, se debe tener en cuenta que el encoder envía dos señales desfasadas entre ellas. Para leerlas se emplea un Timer para cada motor, y de cada Timer dos canales (ch1&2) para capturar la señal Vout A y B del encoder.

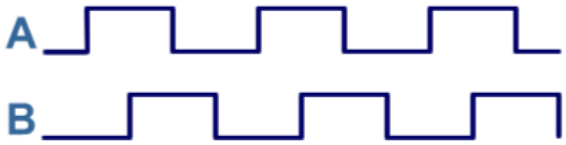


Figura 44 Señales Encoder

Configuramos en el STM32CubeMX como modo Encoder. Esto hará que compare las señales de forma que cuando gire en un sentido y se presente antes el flanco de subida de la señal A que el de la B o viceversa aumente o reste valores enteros de un contador.

A cada contador se le asigna un Autoreload de 10000 valores, es decir, cuando el contador llegue a 10000 o a 0, éste se reiniciará a 0 o 10000 respectivamente. Este se hace porque el contador del timer no puede llegar a infinitos valores, y como ya se ha comentado antes, tiene un valor límite ya pre-establecido. A la hora de programar se tiene en cuenta este aspecto de forma que corrija las cuentas y vaya acumulando los valores sumados o restados. Los puntos críticos en esta cuenta se dan cuando el contador llega a sus límites máximo o mínimo.

Se comprueba mediante el analizador lógico que, efectivamente, el encoder está trasmitiendo las ondas cuadradas pertinentes desfasadas entre ellas.

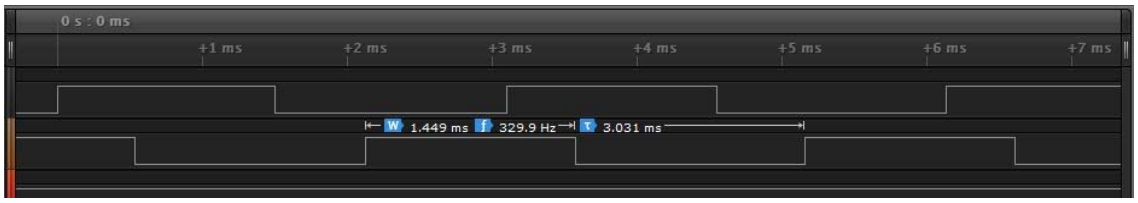


Figura 45 Muestra mediante analizador lógico

La señal de arriba es la Vout A y la de abajo es la B. Se comprueba, por tanto, que el encoder genera las dos señales deseadas.

Los datos de cuenta obtenidos se transmiten mediante el puerto Usart1 Tx y se recogen mediante convertidor a USB “Profilic” donde luego es procesado por un programa escogido para recibir los mensajes, siendo en este caso el “Termite”.

Ahora ya podemos conocer la posición en el eje del motor.

4.3.5 Ensayo 4: Obtención de frecuencia de pulsos desde Encoder

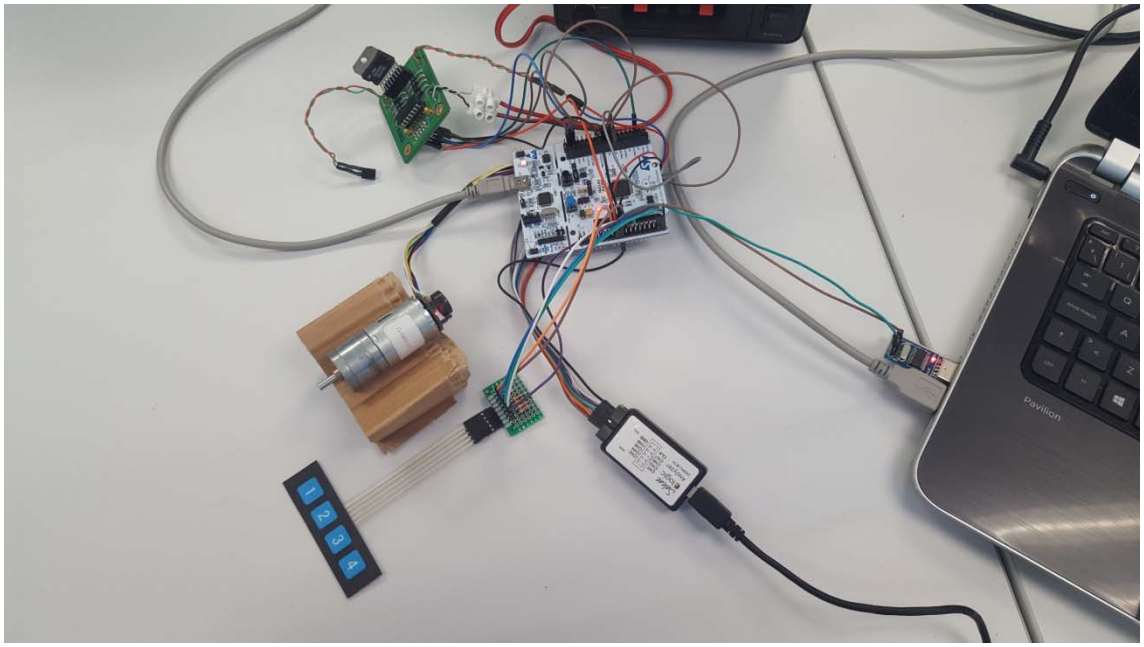


Figura 46 Montaje Ensayo 4

Componentes empleados:

- STM32.
- Botonera.
- Fuente de alimentación (12 V).
- Etapa de potencia.
- Motor CC y Encoder.
- Convertidor USB.

Desarrollo:

En esta ocasión se conseguirá medir la frecuencia a la que llegan los pulsos del encoder para, así, obtener la velocidad a la que gira el eje del motor. Más tarde, en el siguiente apartado, se podrá conocer esta misma información trasladada al eje de la rueda.

Para este caso se conectan las salidas de los pines de una de las dos señales (A o B), en nuestro caso la Vout A, a los Timers 10 y 11 del STM32, uno para cada motor.

Figura 47 Señal Encoder A

Estos timers son los responsables de medir el tiempo que tarda en aparecer un nuevo pulso proveniente del encoder, indicando en cada momento el periodo que tarda en dar una vuelta. Para ello cada timer conectado a su respectivo motor mide en que instante aparece un flanco de subida del pulso. Según el prescaler escogido, y sabiendo la frecuencia generada, se obtendrá un mayor o menor número de pulsos del timer por pulso del encoder. Este contador

se reinicia cada vez que aparece un flanco de subida, y por tanto, cada vez que aparece un pulso de la señal del encoder. De aquí se podrá obtener el tiempo / frecuencia de cada pulso del encoder.

A estos timers se le ha configurado los siguientes parámetros:

- Prescaler = 84.
- Autoreload = 40000.

Experimentalmente se ha comprobado que las cuentas nunca llegar a superar o acercarse al valor de 40000. Por esta razón se le ha asignado dicho valor, de forma que no se reinicie en momentos que no debe y simplificando así la programación para la obtención de los valores deseados.

Con el valor asignado del Prescaler se tiene una frecuencia de muestreo para la señal del encoder de **f_1 = 1 MHz** y por tanto:

$$T_{muestreo} = \frac{1}{f_1} = \frac{1}{1 \cdot 10^6} = 1 \cdot 10^{-6} = 1\ \mu s \qquad (45)$$

Para saber cuánto tarda un periodo de pulso de señal del encoder basta con multiplicar el número de pulsos contados por el timer de muestreo TIM10 y TIM11 por el periodo de muestreo. Por tanto, el periodo de ciclo de cada pulso se calcula según la siguiente fórmula.

$$T_{ciclo} = T_{muestreo} \cdot N_{pulsos}$$

Siendo N_{pulsos} el número de pulsos de muestreo contados en cada ciclo.

Para la comprobación de este experimento se ha comprobado para dos velocidades distintas. Una a p=40 y otra a p=80. Siendo “p” el valor introducido en la PWM que va de 0 a 100, coincidiendo con el valor porcentual del duty de 40% y 80% respectivamente.

Tomamos de dos referencias distintas las medidas, de forma que se pueda contrastar la información y comprobar que, efectivamente, es correcta.

- **Medida desde encoder:** Esto es, muestreo de la señal del encoder y cálculo de su periodo mediante el microcontrolador. Se muestra por pantalla las medidas realizadas.
- **Medida desde analizador:** Medición de la frecuencia y periodo de la señal del encoder mediante el analizador lógico.

Se les ha asignado a dos botones de la botonera un valor del duty distinto. Para el botón 1, p=40 y para el botón 2, p=80. De esta forma se hace más cómodo y sencillo la obtención de resultados.

Por tanto, obtenemos los siguientes resultados:

Mediciones Analizador:

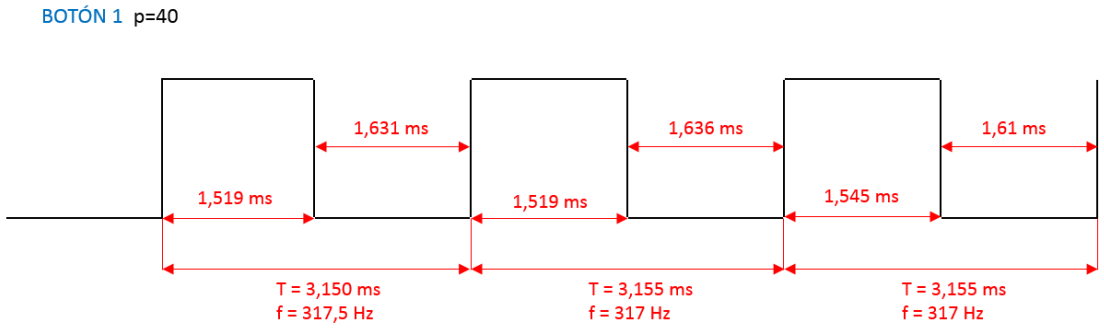


Figura 48 Medición Analizador Lógico p=40

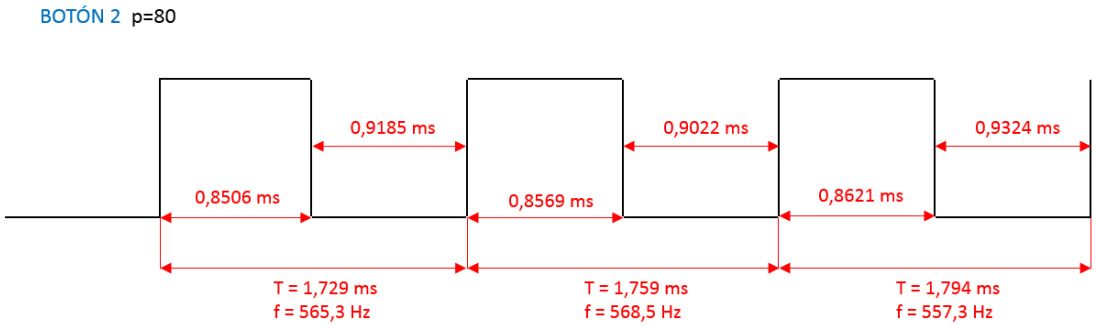


Figura 49 Medición Analizador Lógico p=80

Siendo el eje vertical la tensión y el horizontal el tiempo medido.

Mediciones microcontrolador (µC) y comparación:

BOTÓN 1	p = 40
Npulsos	Tpulso
3299	3,299 ms
3320	3,320 ms
3386	3,386 ms
3401	3,401 ms
3421	3,421 ms
3365	3,365 ms
3345	3,345 ms
3356	3,356 ms

Media µC	3,362 ms
Media Analizador	3,153 ms

BOTÓN 2	p = 80
Npulsos	Tpulso
1746	1,746 ms
1743	1,743 ms
1752	1,752 ms
1745	1,745 ms
1788	1,788 ms
1765	1,765 ms
1769	1,769 ms
1738	1,738 ms

Media µC	1,756 ms
Media Analizador	1,761 ms

Como toda medida, siempre surgirán pequeñas variaciones en la medida de ésta. Estas variaciones no son importantes a no ser que aumenten demasiado y se genere mucho ruido. No es el caso que se nos presenta. Comparados los periodos de ciclo de la señal del encoder, se comprueba por tanto que las señales medidas son correctas, pues la diferencia entre las dos mediciones es despreciable y prácticamente igual.

Dado que el motor CC presenta una zona muerta, es lógico pensar, y como se ha comprobado experimentalmente, que si se pone un duty del doble que el anterior, éste no dará exactamente una velocidad el doble de la anterior. Para esto hay que tener en cuenta el offset generado por la zona muerta.

Tenemos, por tanto, un programa capaz de leer el periodo de la señal del encoder y la base para obtener la velocidad de giro. Más adelante, transformando esta señal se podrá obtener la velocidad de giro.

Nota: Nótese que se emplean valores de "p" de 0 a 100 en la PWM coincidiendo con el duty escogido. No será hasta llegar al ensayo del control PD del robot donde se cambien estos valores de 0 a 1000 por necesidades del experimento y finalmente del funcionamiento prototipo.

4.3.6 Ensayo 5: Obtención de relación entre Eje Motor y Eje Rueda

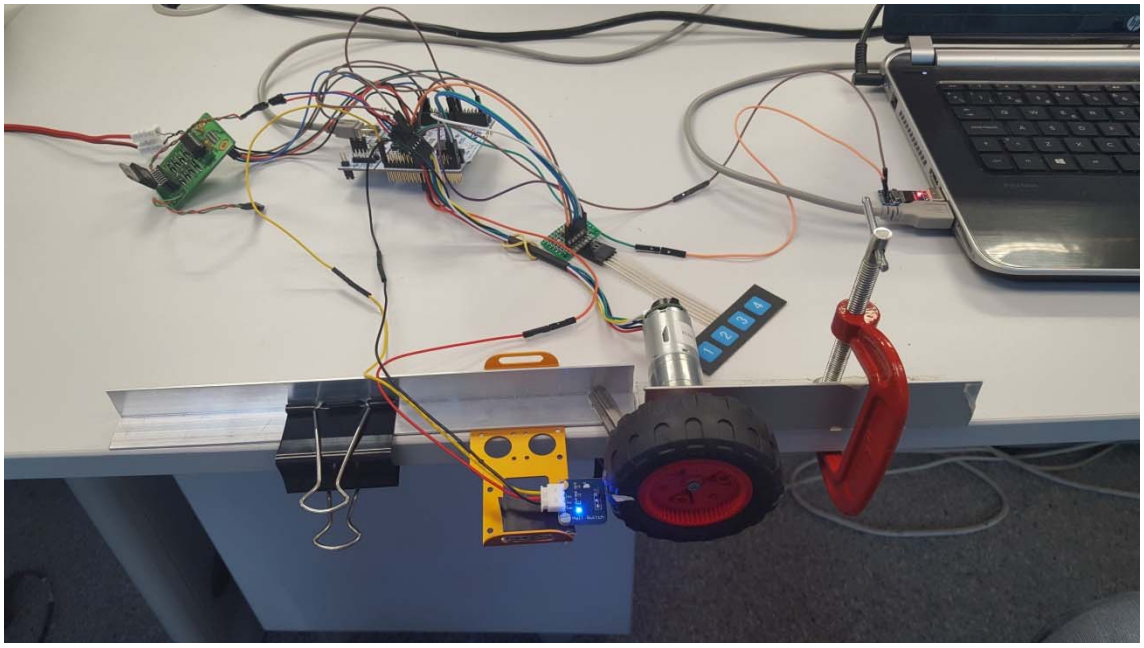


Figura 50 Montaje Ensayo 5

Componentes empleados:

- STM32.
- Botonera.
- Fuente de alimentación (12 V).
- Etapa de potencia.
- Motor CC y Encoder.
- Convertidor USB.
- Rueda.
- Imán.
- Sensor efecto Hall.

Los componentes antes nombrados son los mismos para las dos pruebas que se explican a continuación en lo sub-apartados de este punto.

La rueda se une al “eje de la rueda” y se le pega un imán en un punto de la cara que sirve de apoyo cuando esta gira sobre una superficie, es decir, el imán irá colocado paralelo al eje con sus polos perpendiculares a éste.

El sensor de efecto Hall se emplea para medir el momento en el que el imán pegado a la rueda pasa por su posición. Funciona a modo interruptor, cada vez que el campo magnético del imán activa el sensor envía una señal al microcontrolador.

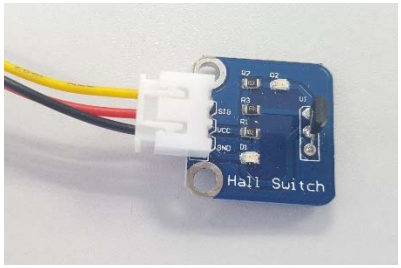


Figura 51 Sensor efecto Hall

Desarrollo:

Ahora que ya sabemos los valores que envía el encoder concernientes a la posición y la velocidad, se desea poder procesar dicha información a valores conocidos y útiles para el proyecto. Se desea conocer su equivalencia en el eje de la rueda.

Este ensayo ha sido necesario por necesidades del proyecto y falta de información por parte del proveedor. Esta es una situación que nos podemos encontrar en cualquier momento de nuestra vida profesional y que requiere de una solución por nuestra parte.

En este caso, no se ha sido proporcionada la información concerniente a la relación de transmisión de la reductora y, por tanto, se hace necesaria una comprobación experimental.

Además, se quiere comprobar a qué velocidad se mueve realmente el eje de la rueda con respecto al eje del motor.

Los pasos previos a la realización de los siguientes experimentos son los referentes a la **configuración del sensor Hall**:

- El sensor se ha configurado como una entrada GPIO inicialmente para la realización de la primera prueba deposición. Según esté en “High” (Activo) o en “Low” (Desactivado) indicará que ha sufrido una perturbación por parte del imán y por tanto, que se ha dado una vuelta de rueda.
- El sensor Hall posee un led que se activa cada vez que recibe una perturbación (En nuestro caso es el paso del imán). Comprobamos la polaridad correcta del imán haciéndolo pasar por delante del sensor y viendo cuando se activa.
- Se comprueba que el sensor de efecto Hall envía una señal con el analizador lógico.
- Se coloca el imán en la rueda, se conecta el sensor a los GPIOs correspondientes del STM32 y se procede a la realización de las siguientes pruebas y mediciones.

4.3.6.1 Ensayo 5.1: Obtención relación de posición ejes motor/rueda

Se procede a la obtención de la relación entre el número de pulsos generados en el eje del motor en una vuelta del eje de la rueda.

Para conocer dicha relación se usa el imán pegado a la rueda adyacente al sensor de efecto Hall puesto como GPIO de entrada en el STM32. De esta forma se puede saber el momento exacto en el que empieza una vuelta de la rueda y cuando acaba. Así se podrá obtener de forma precisa cuantos pulsos (Vueltas) desde el encoder llegan en una vuelta completa de la rueda.

Se realiza el código necesario para que cuente de forma automática los pulsos equivalentes a una vuelta del eje del motor producidos en el transcurso de una vuelta de rueda (Medido por el sensor de efecto Hall). Se muestran los pulsos medidos en cada vuelta en pantalla.

Una vez obtenida una muestra se descarta el primer valor obtenido, pues la posición del imán al comienzo del experimento puede ser cualquiera y así nos aseguramos de que se mide de forma correcta vueltas completas del eje.

Se ha tomado en total una muestra de 100 mediciones de vueltas de rueda. De esta medida se obtiene luego la media entre los pulsos generados en el encoder respecto a las vueltas dadas por la rueda. A su vez se comprueba que las medidas son todas similares entre ellas.

Se obtiene la siguiente tabla como resultado:

Promedio Pulsos	Grados Rueda / Pulso Encoder (º)	Grados Rueda / Pulso Encoder (Rad)
967	**0,3723**	**0,00649**

Donde 967 es el número de pulsos medio dado en 1 vuelta del eje de la rueda. Por tanto, la relación de posición entre el motor y el eje de la rueda es de: 1/967. Siendo 1, las vueltas del eje de la rueda y 967, los pulsos generados por el encoder referentes al eje del motor.

De esta forma se puede obtener a su vez la posición angular a la que se encuentra la rueda en cada momento, sabiendo la equivalencia entre pulsos y grados. La proporción es tal que sigue:

$$Grados_{pulso} = \frac{360º}{967} = 0.3723º \quad (46)$$

O en radianes:

$$radianes_{pulso} = \frac{2\pi}{967} = 0.00649º \quad (47)$$

Usaremos más adelante como referencia en el programa los grados desplazados.

4.3.6.2 Ensayo 5.2: Obtención relación de velocidad ejes motor/rueda

En esta ocasión se busca la obtención del coeficiente de reducción entre la velocidad del eje motor y el eje de la rueda.

Para este caso se usa la misma configuración que anteriormente excepto por un detalle. Se cambia la conexión del sensor de efecto Hall y se conecta a un timer, en este caso el TIM4, para poder obtener la frecuencia de giro de la rueda. Las características del timer se ajustan de la siguiente forma:

- Prescaler = 8400.
- Autoreload = 60000.

Por tanto:

$$T_{muestreo} = \frac{1}{f_1} = \frac{1}{1 \cdot 10^4} = 1 \cdot 10^{-4} = 100\ \mu s \qquad (48)$$

Como en el caso anterior se pone una valor de autoreload muy grande para evitar que desborde y se reinicie antes de la siguiente cuenta. Estos cambios se realizan porque el giro de la rueda con respecto al encoder es mucho más lento y por tanto no hace falta una frecuencia de muestreo tan elevada y se evita complicar el programa.

Con este experimento se desea obtener la frecuencia desde el encoder y la de giro del eje de la rueda. De esta forma podemos obtener la velocidad en cada uno de los casos y, por tanto, la constante de reducción del motor. Pudiendo sacar la relación de velocidades entre la información obtenida por el encoder del eje del motor y la velocidad real de la rueda.

Tanto para los ensayos anteriores como para este, la rueda no tiene rozamiento alguno con ninguna superficie, estando en voladizo tal como se muestra la fotografía anterior.

Se realizan mediciones para 3 situaciones diferentes, de forma que se pueda ver y comprobar el sistema en casos distintos y obtener un resultado más fiable. Esto es, se obtienen mediciones y se calculan datos para un duty del 40%, 80% y del 100% (Máxima velocidad). Tomando gran cantidad de muestras, se calcula los periodos, frecuencias de giro y velocidades. Al final se obtiene un promedio de todos estos datos como se puede comprobar en la siguiente tabla resumen:

PROMEDIOS	M Medida	R Medida	M Tiempo (s)	R Tiempo (s)	M Frecuencia (Hz)	R Frecuencia (Hz)	M Velocidad (rpm)	R Velocidad (rpm)	M -> R Cte reducción	R-> M Cte reducción
DUTY: 40%	2185,2399	11404,7315	0,0022	1,1405	457,7883	0,8768	27467,2960	52,6099	0,00192	522,0943
DUTY: 80%	1300,0772	6795,5487	0,0013	0,6796	769,2759	1,4716	46156,5561	88,2931	0,00191	522,7646
DUTY: 100%	1125,5721	5881,2785	0,0011	0,5881	888,5106	1,7003	53310,6361	102,0187	0,00191	522,5579

	M -> R Cte reducción	R-> M Cte reducción
PROMEDIO Constantes reducción:	0,00191	522,4723

Siendo “M” el eje del motor t “R” el eje de la rueda.

Se ha obtenido los datos de la tabla anterior de la siguiente forma:

Se ha muestreado con mediante su timer correspondiente cada eje.

Frecuencia base Timers:	8,40E+07 Hz

	Encoder (Motor)	Sensor (Rueda)
Prescaler	84	8400
Frecuencia cuenta (Hz)	1,00E+06	1,00E+04
Periodo cuenta (s)	1,00E-06	1,00E-04

Medida (M): Timer 10; $f_{muestreo}$=1MHz; $T_{muestreo\ M}$ = 1µs

Medida (R): Timer 4; $f_{muestreo}$= 10 kHz; $T_{muestreo\ R}$ = 100µs

Se calculan los periodos de cada medida, es decir, el tiempo entre pulso y pulso para cada eje:

$$T_{ciclo} = T_{muestreo\ i} \cdot Medida \quad (49)$$

Siendo “$T_{muestreo\ i}$” el de cada timer respectivo e “i” el eje estudiado y “Medida” el número de pulsos de timer da cada medida.

Transformamos estos valores de periodo en frecuencia en Hz para cada eje:

$$f_i = \frac{1}{T_{muestreo\ i}} \quad (50)$$

Para obtener los valores de velocidad del motor CC en ambos seguimos la siguiente fórmula:

$$n = \frac{60 \cdot f}{p} \quad (51)$$

Siendo “n” la velocidad de giro en r.p.m. (Revoluciones por minuto), “f” la frecuencia y “p” el número de polos. Por falta de información por parte de fabricante se deduce lo siguiente. Teniendo en cuenta que el motor empleado es el motor CC más simple, se llega a la conclusión de que el motor posee un único par de polos, es decir, p=1. Esto se comprueba experimentalmente cuando al aplicar la fórmula con este valor de “p”, comparando la velocidad máxima a duty 100% en el eje de la rueda de n=102 rpm, la más cercana a la referencia sin carga dada por el fabricante de 110 rpm.

A continuación obtenemos las constantes (Acortando, “ctes”) de reducción:

Constante de reducción de Motor a Rueda:

$$k_{red\ M-R} = \frac{n_R}{n_M} \quad (52)$$

Constante de reducción de Rueda a Motor:

$$k_{red\ R-M} = \frac{n_M}{n_R} \quad (53)$$

Siendo "n_M" la velocidad en rpm del eje del motor y "n_R" la velocidad en rpm del eje de la rueda.

Una vez tomadas estas medidas para cada caso con duty del 40%, 80% y 100%, se hace una media de los resultados obtenidos, teniendo así lo mostrado en la tabla:

$k_{red\ M-R} = 0.00191\ rpm$

$k_{red\ R-M} = 522.4723\ rpm$

En el código usaremos la primera constante de relación de motor a rueda, pues durante el proceso conoceremos únicamente la velocidad del eje del motor mediante sensor, encoder, y la velocidad del a rueda tendrá que ser observada mediante transformación de estos valores.

4.3.7 Ensayo 6: Lectura conjunta de posición y velocidad desde Encoder

Componentes empleados:

- STM32.
- Botonera.
- Fuente de alimentación (12 V).
- Etapa de potencia.
- Motor CC y Encoder.
- Convertidor USB.

Desarrollo:

Ya sin necesidad de sensores externos (Sensor de efecto Hall) y con los datos anteriormente obtenidos, se realiza un programa que calcule automáticamente desde el encoder las variables de posición y velocidad. Se tratara de un compendio de los resultados y conclusiones obtenidos en el Ensayo 5.

Se realiza un código mediante el "Keil µVision" que aúne los dos tipos de medida, la de posición y la de velocidad.

Principalmente nos interesa los valores dados en el eje de la rueda, los del eje del motor seguimos visualizándolos para controlar que el proceso se realiza correctamente. En este ensayo se obtienen los valores para el motor 1 de forma que el programa los procese:

- Eje del motor: número de vueltas, velocidad rpm
- Eje de la rueda: número de vueltas, posición angular, velocidad rpm.

En el ejemplo de abajo se puede ver una de las pruebas realizadas, de forma que el lector pueda hacerse una idea de cómo se pueden mostrar por pantalla las mediciones:

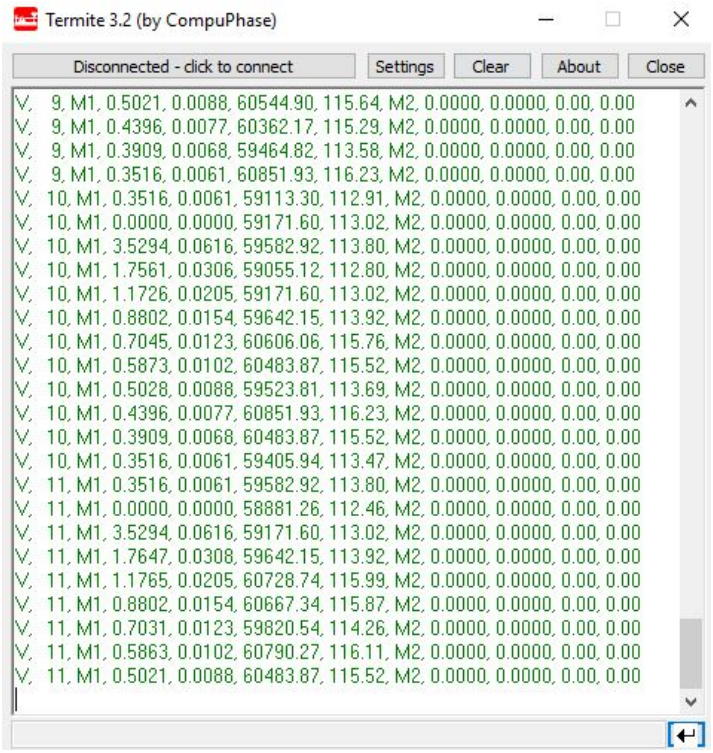


Figura 52 Muestra Termite

De izquierda a derecha, se muestra separadas por una coma las siguientes medidas: la vuelta actual (V #), el motor medido (M# #), muestra de grados girados entre pulso y pulso del encoder en grados y en radianes (#, #), velocidad en rpm del eje del motor (#) y velocidad en rpm del eje de la rueda (#).

4.3.8 Ensayo 7: Configuración RTC

Componentes empleados:

- STM32.

Desarrollo:

Este ensayo se ha desarrollado en paralelo al siguiente ensayo 8 de configuración del IMU. A la vez que se desarrolla el siguiente experimento éste se implementará dentro de él.

El objeto de este ensayo es el de ser capaz de saber en tiempo real cuánto tarda el microcontrolador en procesar el programa entre vuelta de bucle a la siguiente vuelta de bucle. Esto significa saber cuánto se tarda desde que se hace un cálculo o se emite una orden hasta que se repite la misma acción.

Después de haber implementado el programa final, se sabrá que el tiempo que tardará en realizar una vuelta de bucle será de 8 ms.

Función de lectura del RTC:

```
static void RTC_Segundero(void)            //Real Time Clock, devuelve segundos y subsegundos (3...4ms)
{
  //RTC_DateTypeDef sdatestructureget;

  RTC_TimeTypeDef sTime;
  RTC_DateTypeDef sDate;

  /* Get the RTC current Time */
  HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
  /* Get the RTC current Date */
  HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);
  /* Display time Format: hh:mm:ss */

  seg = sTime.Seconds;
  subseg = sTime.SubSeconds;

}
```

Primero se comprueba que se miden los segundos y los sub-segundos por el RTC (reloj de Tiempo Real).

Según especificaciones, se sabe que para un segundo hay 255 sub-segundos.

$$t = \frac{1\ seg}{255\ subseg} = 0.00392\ s \approx 4\ ms \qquad (54)$$

Siendo "t" el tiempo entre sub-segundos. Se comprueba que las variables se almacenan de forma correcta y se puede operar de forma fiable con ellas para obtener el tiempo entre cálculos. El RTC posee un error inherente, pues este reloj no es tan preciso como el de la orden "HAL_Delay" (Para retardos) del STM32, lo cual hace necesaria la aplicación de un factor de corrección para que corrija dicho offset.

De forma experimental se obtiene que:

Coef.Corrección = 1.33

Por tanto, se establece un coeficiente que multiplica a los valores obtenidos del RTC en sub-segundos para saber cuánto tiempo se ha tardado. A su vez se le añade un factor de escala para que de su resultado en milisegundos.

$$K_{RTC} = \frac{1000}{255} 1.33 \qquad (55)$$

Como en otros casos hay que tener en cuenta que el contador de sub-segundos al llegar a valores máximos se reinicia y debe añadirse este aspecto a la lógica del programa.

Llamaremos al tiempo calculado por el programa de una vuelta de bucle "t_c".

4.3.9 Ensayo 8: Configuración del IMU

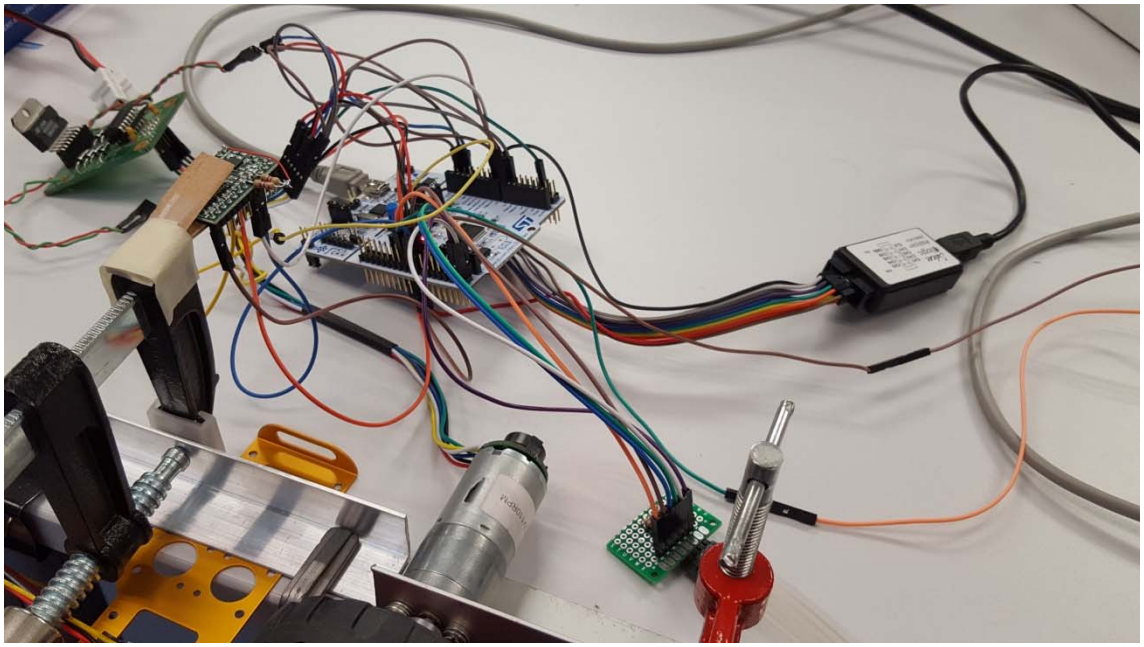


Figura 53 Montaje Ensayo 8

Componentes empleados:

- STM32.
- Botonera.
- Fuente de alimentación (12 V).
- Etapa de potencia.
- Motor CC y Encoder.
- Convertidor USB.
- IMU

Se conecta el IMU para poder medir la posición angular a la que se encontrará en cada momento el robot cuando caiga en un sentido u otro. Servirá como base para uno de los controles principales del vehículo autobalanceado. Por tanto, es necesaria su configuración.

Desarrollo:

Para este ensayo se desea configurar el IMU de forma que el microncontrolador sea capaz de, basándose en las medidas dadas por el giroscopio y el acelerómetro, medir la posición angular de la estructura en todo momento.

Esta parte es muy importante para el futuro control de estabilidad del robot. Si mide que su inclinación es de 0º el vehículo se encontrará de pie y en posición correcta, si mide que los ángulos aumentan, tanto es su vertiente positiva como en la negativa (Caída hacia un lado u otro, según ejes del IMU) habrá que aportarle una señal de control y mover las ruedas consecuentemente.

Primero comprobamos la correcta conexión y funcionamiento del IMU. Al realizar esta comprobación se ve que no posee resistencias de pull-up, necesarias para su funcionamiento. Se conectan a las dos salidas SDA y SCL a la entrada de tensión Vdd.

Se implementa la programación necesaria para procesar los datos obtenidos desde el giroscopio y el acelerómetro. Se puede llegar a obtener los ángulos en los distintos ejes x, y, z, pues el STEVAL-MKI124V1 tiene 3GDL cada sensor.

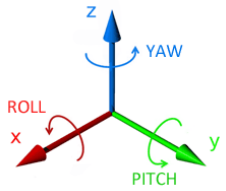


Figura 54 Ejes IMU

Diferenciamos en varias partes el ajuste de la configuración del IMU y la transformación de sus datos en otros procesables por el controlador y entendibles por la persona que desee verlos, es decir, transformación de las señales del giroscopio y acelerómetro en ángulos.

Acelerómetro:

El acelerómetro es capaz de mediar la aceleración en cada uno de sus ejes (x,y,z). En una posición paralela al suelo y si el IMU no se moviera, obtendríamos la aceleración de la gravedad en el eje z y 0 en los demás. Si se moviera el IMU los valores de estos vectores varían, de forma que mediante cálculos trigonométricos podemos obtener el ángulo en el que se encuentra.

La fórmula aplicar es tal que sigue:

$$Ángulo_x = \arctan\left(\frac{X_{acelerómetro} - X_{offset}}{Z_{acelerómetro} - Z_{offset}}\right)\frac{180}{\pi} \quad (56)$$

$$Ángulo_y = \arctan\left(\frac{Y_{acelerómetro} - Y_{offset}}{Z_{acelerómetro} - Z_{offset}}\right)\frac{180}{\pi} \quad (57)$$

Siendo $X_{acelerómetro}$, $Y_{acelerómetro}$, y $Z_{acelerómetro}$ las mediciones obtenidas por el dispositivo. Los offset son constantes que se verá más adelante como se obtienen.

La multiplicación por 180/π se aplica para pasar a grados la medida, que es lo que nos interesa. De otro modo estaría en radianes.

Se ha calculado para ambos casos, eje x y eje y, por si se diera la situación de que por necesidades futuras de alguna modificación en la estructura hubiera que girar el IMU y obtener información del otro eje que estábamos usando.

Giroscopio:

El giroscopio mide la velocidad angular. Sus medidas son más precisas que las del acelerómetro, aunque como veremos tiende a acumular un error con el tiempo. La forma de calcular el ángulo a partir de la velocidad angular se basa en la integración de las lecturas.

Sigue como base las siguientes fórmulas:

$$Ángulo_X = \int \omega_x \cdot dt \quad (58)$$

$$Ángulo_y = \int \omega_y \cdot dt \quad (59)$$

Siendo "ω_i" la velocidad angular.

Sin embargo, no se puede aplicar esta fórmula tal cual en el código, por lo que se hace necesario aplicarlo de diferente forma. Teniendo en cuenta estas fórmulas como base, las especificaciones en cuanto a transformación de los valores medido por el giroscopio dados por el fabricante, sensibilidad, y que el tiempo entre medida y medida procesada por el microcontrolador es igual a "t_c", tiempo del RTC ms, concluimos las siguientes fórmulas:

$$memX_n = X_{giroscopio} + X_{offset} \quad (60)$$

$$memY_n = Y_{giroscopio} + Y_{offset} \quad (61)$$

$$Ángulo_{x,n} = (memX_n + memX_{n-1}) \frac{t_c}{2000000} \quad (62)$$

$$Ángulo_{y,n} = (memY_n + memY_{n-1}) \frac{t_c}{2000000} \quad (63)$$

Siendo "memX" y "memY" la lectura con el offset añadido del giroscopio y "n" la vuelta de bucle en la que nos encontramos.

De esta forma podemos calcular el ángulo actual mediante el giroscopio.

Cálculo offset:

Los offsets, o desviaciones del valor referencia, se han obtenido de forma experimental. Se ha medido las señales tanto del acelerómetro como del giroscopio se realizarle transformación alguna.

El proceso seguido en orden cronológico ha sido el siguiente:

- Se ha colocado el IMU en posición horizontal, asegurándose de que la cposición era correcta y de que no sufría movimiento alguno.
- Se ha tomado una serie de muestras de los valores medidos para cada eje (x,y,z).
- Se ha saca la media de estos valores para marcar así el valor referencia para "0º".
- Se ha obtenido el valor máximo y mínimo de estas medidas.
- Para conseguir lo valores máximos y mínimos de desviación se ha calculado la diferencia entre la referencia y los valores máximos y mínimos medidos.
- Una vez obtenidas la desviaciones máximas (Por encima del valor de referencia) y mínimas (Por debajo), se comprueba su similitud y se calcula la media entre ellas.

De esta forma se obtiene los valores de offset, como se puede comprobar en las siguientes tablas:

Acelerómetro

	Xacel	Yacel	Zacel
Promedio Medida	869,964	474,982	15285,680
Max	1344,000	832,000	16640,000
Min	384,000	-64,000	14720,000
Difmax	474,036	357,018	1354,320
Difmin	485,964	538,982	565,680
Promedio Dif	480,000	448,000	960,000

Giroscopio

	Xgiro	Ygiro	Zgiro
Promedio Medida	-2199,261	1165,486	-157,146
Max	-1085,000	3797,500	6492,500
Min	-4725,000	-6982,500	-4462,500
Difmax	1114,261	-2632,014	-6649,646
Difmin	2525,739	8147,986	4305,354
Promedio Dif	1820,000	2757,986	-1172,146

Siendo el valor de "Promedio Dif" el valor final de los offset de cada eje.

Filtro complementario:

Con respecto a los dos elementos de medición comentados se hace notar lo siguiente. Los acelerómetros se ven influenciados por los movimientos del sensor y tienen lecturas bastante ruidosas, por lo que pierden utilidad y es necesario separar lo que es ruido de lo que realmente es la señal deseada. Los giroscopios obtienen mediciones más precisas y fiables, sin embargo con el tiempo acumulan error (Deriva o "drift").

A la hora de obtener las señales se desea, pues, obtener unas lecturas lo más reales y estables posibles. Es por esta razón que, una vez obtenidos los datos deseados, le aplicamos un filtro.

Un filtro importante y famoso, que se ha usado en aparatos para navegación o control de vehículos, como por ejemplo en satélites, es el **filtro de Kalman**. Este filtro sirve como procedimiento óptimo para estimar el estado de un sistema dinámico lineal a la vez que se minimiza el valor cuadrático medio del error cometido en esa estimación. Está compuesto por una serie de ecuaciones matemáticas que proveen una solución recursiva eficiente del método de mínimos cuadrados. En resumen, este filtro realiza una estimación del valor futuro de la medición y después compara mediante un análisis estadístico el valor real para compensar el error en mediciones futuras.

Este filtro sería ideal para nuestro caso si no fuera por, en resumen, lo siguiente. Necesita del conocimiento del sistema y su dinámica, cosa que sabemos cómo funciona pero no disponemos de datos sufrientes para aplicarlo. Posee un mayor coste computacional, lo cual retrasaría el procesado de la información y necesitaría de una mayor capacidad computacional que el filtro que estamos estudiando en este punto.

Por tanto, hemos escogido el uso del **filtro complementario**, más simple y con un menor coste computacional. Al combinar las mediciones de los dos dispositivos podemos obtener mediciones de orientación más precisas que la de cada elemento por separado.

A la hora de aplicar este filtro hay que tener en cuenta la importancia que se le va a dar a cada medida. Su fórmula sería la siguiente:

$$Ángulo_n = K \cdot \left(Ángulo_{n-1} + Ángulo_{giroscopio}\right) + (K-1) \cdot Ángulo_{acelerómetro} \quad (64)$$

Siendo “n” el instante en que se ha realizado la media y “K” el coeficiente en forma de porcentaje en valor decimal que se quiere dar a cada parte del filtro, es decir, su valor se comprenderá entre 0 y 1 y se variará su valor según deseemos darle mayor o menor importancia a cada parte del filtro.

En nuestro caso se le ha asignado un valor de K=0.98.

Este filtro se comporta, por tanto, como un filtro de paso alto para la medición del giroscopio y un filtro de paso bajo para la señal del acelerómetro. Es decir, a corto plazo mandará la señal del giroscopio, pero a largo plazo será el acelerómetro el que domine.

Otros filtros que pueden aplicarse:

- **Filtro por redondeo:**
  Este filtro se trata de multiplicar por un valor la medición, en este caso el ángulo. Una vez multiplicado el valor del ángulo se redondea la cifra obtenida. A este resultado se vuelve a dividir por el valor que se había multiplicado inicialmente, de forma que se ha conseguido eliminar algunos decimales considerados como ruido en la señal. Según el tamaño de la constante multiplicadora/divisora, la cantidad de decimales eliminados será mayor o menos. Este es un filtro empleado para “limpiar” el valor del ángulo obtenido en cada medida. En este caso, dicha constante tiene un valor de 50.
- **Filtro IIR:**
  Este filtro, en su versión de primer orden, se aplicaría para los momentos en los que el sistema no esté en reposo, que es continuamente, pues siempre está corrigiendo y obteniendo valores de ángulo. Llamado IIR (Infinite Impulse Response) donde su entrada es un impulso y la salida es un número ilimitado de términos no nulos. Depende de las entradas pasadas y actuales, por tanto se realimenta.

$$y[n]=\sum_{k=0}^{M} b_k x[n-k]+\sum_{k=1}^{N} a_k y[n-k] \qquad (65)$$

4.3.10 Ensayo 9: Configuración de la Radio

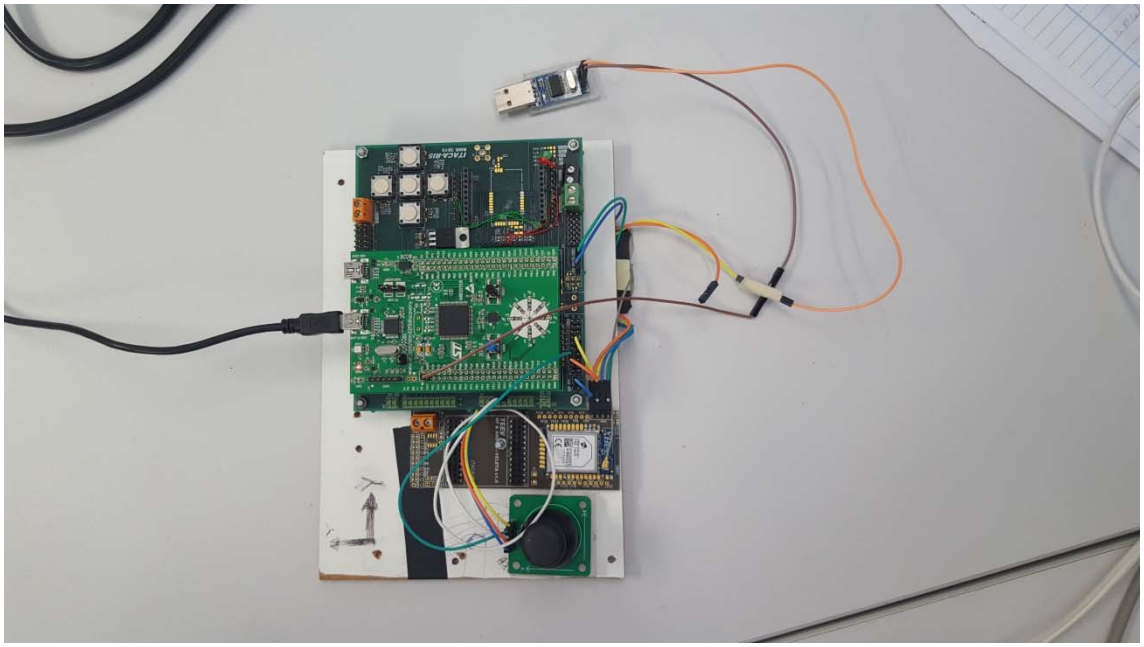


Figura 55 Montaje Ensayo 9

Componentes empleados:

- STM32.
- Botonera.
- Fuente de alimentación (12 V).
- Etapa de potencia.
- Motor CC y Encoder.
- Convertidor USB.
- IMU
- Módulo RF.

El módulo Rf será el encargado de transmitir la información mediante radio frecuencia entre el robot y el exterior.

Desarrollo:

Durante esta prueba se desea configurar y comprobar el funcionamiento de los módulos RF encargados de la transmisión de información entre el STMF4 del robot autobalanceado y el exterior.

Para la consecución de este objetivo serán necesarios dos módulos RF. En nuestro caso, y tal como se ha explicado en su apartado correspondiente, se ha usado el modelo “XBee XB8-DMUS-002”.

Un módulo RF va conectado el STM32F411RE y el otro se alimenta del STM32 Discovery F3 y transmite a la computadora mediante el convertidor USB para luego visualizar las variables por pantalla.

La función de estos dispositivos es la de transmitir la información dada por el STM32F4 y se ha implementado de forma que se pueda usar en el futuro para conectar otros dispositivos que

transmitan información el robot, como un joystick para controlar el movimiento o una botonera para enviar comandos.

Los módulos usados ya estaban pre-configurados, por lo que no fue necesario el uso del programa específico para conectarlos entre sí y que transmitan a la misma frecuencia.

Para la transmisión y recepción de datos se emplea tipo los conectores para comunicación USART. RX es el pin receptor y TX el pin transmisor.

Dado que la transmisión de datos es abierta, únicamente se debe averiguar a qué velocidad de transmisión de datos.

Ajustamos en la aplicación informática de visualización por pantalla “Termite”.

Probamos varias velocidades en el Termite, las cuales se asignan en Baudios. Si la velocidad de transmisión asignada no es correcta, el programa no será capaz de interpretar la información que le llega y por pantalla solo se verán símbolos inconexos o fallos de comunicación.

En este caso, la velocidad de transmisión es de **38400 Bd**.

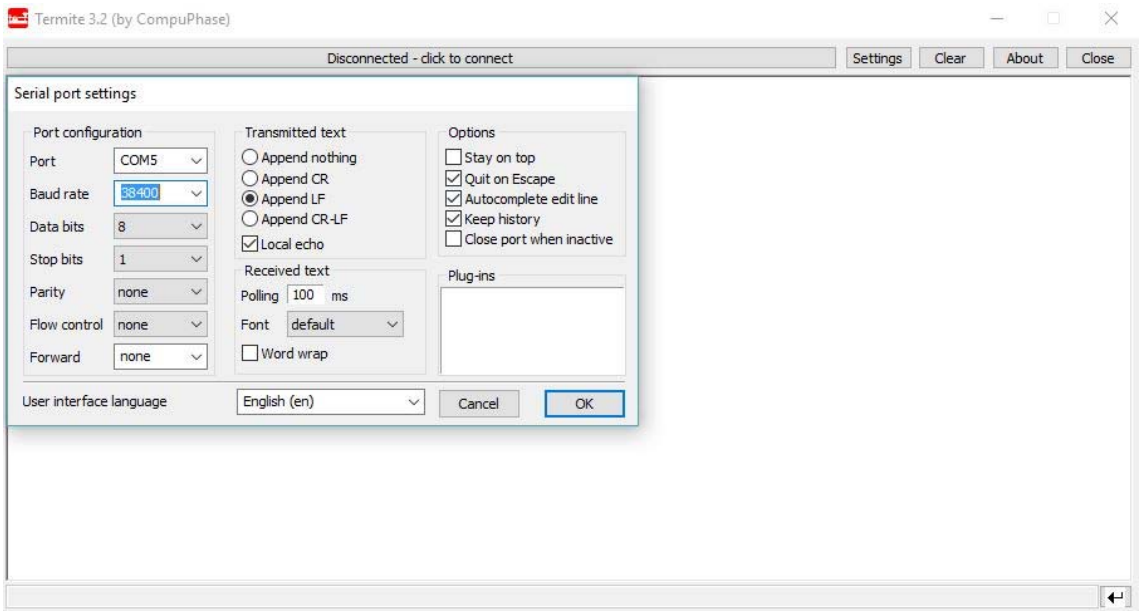


Figura 56 Configuración Baudios Termite

Una vez ajustada la velocidad se comprueba que efectivamente se transmite información desde el robot al ordenador.

4.3.11 Ensayo 10: Calibración de los motores

Componentes empleados:

- STM32.
- Botonera.
- Fuente de alimentación (12 V).
- Etapa de potencia.
- Motor CC y Encoder.
- Convertidor USB.
- Módulo RF.

Desarrollo:

Con este ensayo se quiere conseguir que las dos ruedas giren a la misma velocidad y sentido en las diferentes situaciones que más tarde le deberá ordenar el control PD del sistema.

Para este ensayo se han modificado los conectores de la etapa de potencia soldando unos nuevos y haciéndolos más sólidos y estables. Por tanto, se comprueba que funciona correctamente:

- Se conecta el pin de sentido a masa para ponerlo a 0.
- Se conectan las señales PWM de los motores a 5V para ponerlo en “high” y comprobar que los ejes se mueven.

Una vez comprobado el correcto funcionamiento de la etapa de potencia se comprueba con el analizador lógico que se generan las señales PWM por los pines asignados a ello para los dos motores.

Al conectar los dos motores con el mismo valor de “p” para la señal PWM confirmamos que los motores no se mueven exactamente a la misma velocidad. Esto es lógico, pues los motores, pese a ser el mismo modelo, siempre surgen algunas pequeñas desviaciones generadas por pequeñas variaciones surgidas en la fabricación de cada accionamiento.

Por tanto, para igualar el funcionamiento entre los motores se hace necesario la calibración de estos de forma que se pueda controlar esa diferencia de velocidad, compensarla e igualarla.

Existen varios métodos para conseguir este objetivo, en nuestro caso se ha realizado de la siguiente forma, aunque también puede hacerse como, por ejemplo, el método que se explica al final de este punto.

Se usa un control proporcional tipo maestro- esclavo. Se considera el motor 1 como maestro y el motor 2 como esclavo.

Se miden las velocidades de ambos motores, de las cuales la velocidad del motor 2 debe igualarse con la del motor 1. Para conocer esto se calcula el error entre las medidas de velocidad en valor porcentual tal como se ve en la siguiente fórmula:

$$Error_{velocidad} = 100 - \frac{v_{ejeM1}}{v_{ejeM2}} 100 \quad (66)$$

Siendo las velocidades del eje del motor 1 (ejeM1) y del motor2 (ejeM2).

	DiferenciaVel Ejes	Error %
MAX	4306	9
MIN	-3842	-13
PROMEDIO	-829,566	2

Comprobamos que el error medio en las velocidades de la muestra es de un 2%.

Sabiendo este error entre velocidades se calcula el nuevo valor para la señal PWM del motor 2, "p_2":

$$p_2 = p - \frac{p \cdot Error_{velocidad}}{100} \quad (67)$$

Siendo "p" el valor de la PWM del motor 1.

Puesto que a veces se producen picos en la señal debidos a la calidad de los materiales empleados, se filtra la señal añadiéndole límites superiores e inferiores a modo filtro de paso banda, de forma que estos picos no produzcan aceleraciones o deceleraciones bruscas en momentos donde no deben.

Se comprueba experimentalmente. Se añade una tercera rueda fija delante del robot para que mantenga la estabilidad y se le hace mover sobre una superficie plana para comprobar que, efectivamente, se mueve en trayectoria rectilínea y por tanto los dos motores se mueven al unísono.

Se puede observar también como desventaja de este método que si se bloquea la rueda que hace de maestra la segunda rueda irá frenando del mismo modo pero, sin embargo, es la rueda que hace de esclava la que es bloqueada, la rueda maestra seguirá girando como si no hubiera sucedido nada.

Otro método:

Otra posibilidad que se propone para introducir los valores de las señales PWM sería la siguiente. Realizar una serie de muestras en las que se obtendría un offset promedio entre los dos motores. Una vez con esta información, se leería la información de los encoders, ser haría una media y después de añadirle el offset a ésta, se introduciría el correspondiente valor de la PWM en cada motor. Con este método no se tiene en cuanta las posibles variaciones en el offset entre un motor y otro, suponiéndolo constante. Esto conlleva a un aumento en el error en los valores a emplear y, por consiguiente, en la diferencia de velocidad entre los motores. Por el contrario, con este método se gana una mayor independencia entre los motores, evitando el método "maestro-esclavo" empleado. De esta forma se conseguiría que si cualquiera de las dos ruedas variara la velocidad súbitamente la otra rueda hiciera lo mismo.

4.3.12 Ensayo 11: Configuración de los Controles PD

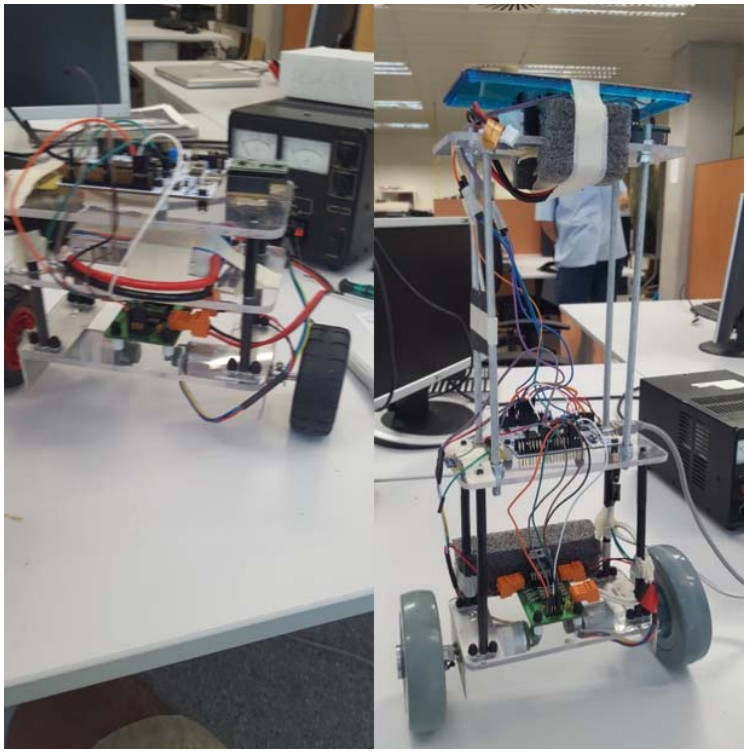


Figura 57 Montaje Ensayo 11

Componentes empleados:

- STM32.
- Botonera.
- Fuente de alimentación (12 V).
- Etapa de potencia.
- Motor CC y Encoder.
- Convertidor USB.
- IMU
- Módulo RF.

Desarrollo:

En este ensayo busca como objetivo conseguir que el robot consiga autorregularse y conseguir mantenerse de pie y balanceado. Para ello se estudia la implementación de dos controladores.

- Control PD angular.
- Control PD desplazamiento.

Este ensayo se realiza una vez montada ya toda la estructura para comprobar si puede mantenerse en la vertical el robot. También se hacen ensayos sin apoyar en la superficie las ruedas para verificar que los motores reaccionan como deberían ante los diferentes estímulos.

Este ensayo va unido a otros en paralelo, pues a la hora de implementar el control se ha visto necesarias varias revisiones de tests anteriores y ejecución de otros tantos nuevos.

Se verifican las conexiones de los diferentes elementos y dispositivos.

Se comprueba cual es el ángulo de no retorno, es decir, aquel ángulo de caída que una vez sobrepasado, aun yendo a máxima velocidad no se puede recuperar la posición.

$\theta_{límite} = 10º$

Siendo "θ" el ángulo de la estructura respecto a la vertical.

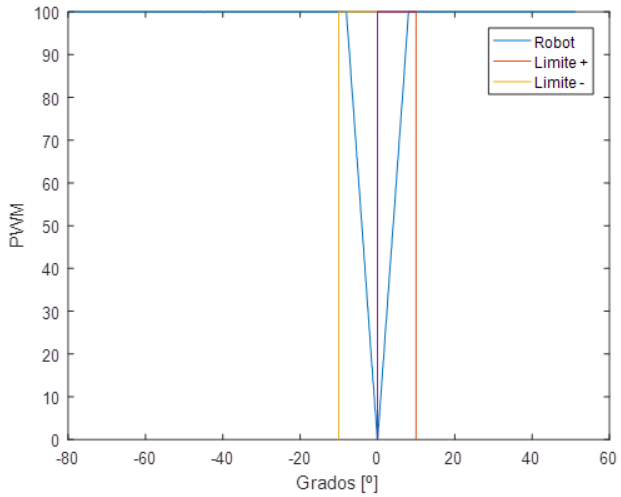


Figura 58 Límite ángulo y comportamiento Control P

En la gráfica se ve uno de los resultados con control P. Sin tocar el suelo se movía la estructura desde una posición original tumbada hasta realizar 180º. De esta forma se podía ver con que Kp se llegaba a la velocidad máxima antes de llegar al ángulo límite de no retorno. Al tratarse de un sistema no lineal, la respuesta lineal de velocidad mostrada en la gráfica no es adecuada para el control del prototipo. Es por esto que se opta por los controles especificados en este punto.

A la hora de Ajustar las ganancias Kp y Kd Se ha procedido de la siguiente forma:

- Se han puesto ambas ganancias con valor 0.
- Se ha ido incrementando el valor de Kp comprobando sobre la superficie la mejora de la estabilidad o lo contrario.
- Una vez obtenida una Kp satisfactoria, se incrementaba el valor de kd hasta que se veía un aumento de estabilidad. En ocasiones, al superarse demasiado el valor de la acción derivada, surtía el efecto contrario, produciendo mayores inestabilidades en el sistema.
- Si no se mantenía de pie después de este proceso se repetía de nuevo desde el principio con otros valores nuevos. Además se comprobaban posibles causas físicas que pudieran generar inestabilidad al sistema y se corregían.

Durante la en desarrollo de la aplicación de estos controles también se observa que el control no funciona de la forma que se esperaría. Después de varias pruebas se va observando los errores cometidos y ajustándose éstos:

- La estructura no es completamente adecuada para poder realizar un control eficaz y se hace necesario aumentar la altura y cambiar pesos de forma que el centro de gravedad esté más alto.
- Se observa aceleraciones bruscas cuando no debería. Esto es debido principalmente a dos factores, los cuales se corrigieron después de una observación, experimentación. Con estos cambios se consigue un movimiento mucho más fluido y estable y se puede proceder a ajustar de forma correcta el PID principal:
    1. Al procesar los datos, a veces el tiempo se superpone y realiza varias vueltas de cálculos, antes de enviar la información y procesarla. Esto es, el reloj de tiempo real tarda en dar su información y por lo tanto falsea algunos picos de tiempo, lo cual varía los cálculos momentáneamente e desestabiliza el robot. Se observa cual es la constante de tiempo que tarda el microcontrolador en realizar una vuelta de bucle, siendo ésta de 8ms y se pone constante dentro del PID.
    2. Se observa que existe cierto ruido cuando se calcula las proporciones entre velocidades de ruedas. Esto hace que de vez en cuando haya una lectura errónea y la rueda que sigue a la principal de un pico de velocidad momentáneo, lo cual hace que l robot pierda estabilidad. Esto se corrige poniendo un filtro de paso banda limitando el máximo del valor a corregir en la velocidad de dicha rueda

Durante los ensayos, también se ha probado a aumentar el peso en la parte superior mediante plomos para elevar el c.d.g. todavía más. Si se ponía demasiado dicho objetivo, aunque a su vez se aumentaba la masa, lo cual reducía a su vez la capacidad de respuesta de los motores sin conseguir mejorar visibles. Por esta razón al final se opta por no incluir pesos excesivos. Esto es, se coloca una serie de 3 plomos de 40g de masa en la parte superior del prototipo.

Nota: Dadas las características del robot se hace muy complicada la obtención de gráficas del desempeño de éste hasta que no se mantenga de pie durante un periodo largo de tiempo. Nuestro primer objetivo es que se mantenga de pie a modo de péndulo invertido, nos basamos en el comportamiento del robot durante los experimentos, observando visualmente su comportamiento y comparando éste con las variables obtenidas por pantalla. Si durante el experimento el robot cae, los datos no son fiables a la hora de ajustarlos. Es por esta razón que se ha trabajado de una forma más experimental y visual, tal como se ha explicado.

Ensayo 11.1: Control PD – Angular

A la hora de obtener los valores de las ganancias, antes se ha visualizado los valores que daba cada acción. De esta forma se obtiene una idea de la escala a la que se debe introducir los valores de las ganancias Kp y Kd.

Durante el transcurso del experimento se ha probado también la aplicación de la acción Integral, siendo un control tipo PID, pero tal como se ha explicado anteriormente únicamente se conseguía aumentar la inestabilidad del sistema.

Se toma como referencia el ángulo cuando el IMU está horizontal, el cual es cero. Esto se da cuando la estructura está completamente vertical.

A la hora de implementar en el código este control se ha creado una función con dicho propósito. Dicha función sigue las siguientes pautas.

$$P = \theta_{actual} \cdot K_p \quad (68)$$

$$D = \left((\theta_{actual} - \theta_{anteriorl})/\ t_c\right)K_D \quad (69)$$

$$Y = P + D \quad (70)$$

Siendo “P” la acción proporcional y “D” la acción derivada. “Θ” es el ángulo obtenido mediante el IMU. “tc” es el tiempo entre aplicación del regulador anterior hasta la actual. “Y” es la salida del regulador.

Además estas señales se filtran mediante un filtro de paso bajo, en el que se limita el valor de salida a 1000. Esto se hace porque la señal de la PWM va del rango de 0 a 1000 y no puede ser superior.

Se comprueba que inclinando hacia un sentido u otro las ruedas giran hacia el sentido que deben y que cuanto más se incrementa el ángulo, la acción de control y por lo tanto la velocidad de los motores aumenta.

Se confirma el grado de estabilidad sobre una superficie.

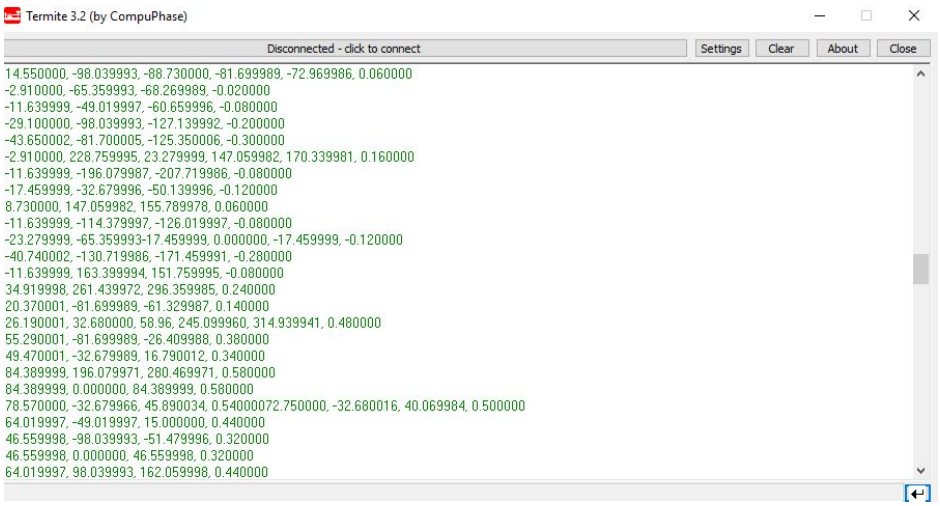


Figura 59 Control PD Angular– Termite

De izquierda a derecha los datos obtenidos son: “P” acción proporcional, “D” acción derivada, Salida del control PD angular, Ángulo actual.

Valores finales de las ganancias para el control:

Ganancias Control PD Angular

Kp	Kd
145,5	6,536

4.3.12.1 Ensayo 11.2: Control PD – Desplazamiento

Inicialmente se empezó este experimento estudiando únicamente la posibilidad de controlar el vehículo basándose en el ángulo de caída. Sin embargo, se observa la necesidad de aplicar un control añadido al ya conocido. Es decir, además del control basado en el ángulo del robot, se aplica otro basado en el desplazamiento de las ruedas. Este último control realizará una acción según la cantidad de giros que haya dado la rueda, siendo éstos, completos o parciales (Distancia total recorrida en un sentido u otro)

Se toma como referencia el punto de partida, es decir, desplazamiento cero.

Para calcular el desplazamiento del vehículo sobre la superficie, es decir, la distancia recorrida por éste en un sentido o en otro se usa la siguiente fórmula.

$$d = 2\pi R\left(\left(\frac{\beta_{rueda}}{360º}\right) + N_{vueltas}\right) \quad (71)$$

Siendo "d" la distancia recorrida (desplazamiento), "β" el ángulo de giro de la rueda y "N" el número de vueltas desde que se inició el desplazamiento de.

Este control se aplica de la siguiente forma:

$$e = R - d \quad (72)$$

$$P = e \cdot K_p \quad (73)$$

$$D = \left((e_{actual} - e_{anteriorl})/\, t_c\right)K_D \quad (74)$$

$$Y = P + D \quad (75)$$

Siendo "e" el error entre la medida de desplazamiento "d" y la referencia "R". "P" es la acción proporcional y "D" la acción derivada. "tc" es el tiempo entre aplicación del regulador anterior hasta la actual. "Y" es la salida del regulador.

Se siguen los mismos pasos de comprobación que en el control anterior.

Al intentar aplicar el segundo control se observa ciertos problemas a la hora de leer la correcta posición de la rueda. Según el sentido suma o resta de forma correcta los grados recorridos, sin embargo, si el cambio es muy brusco el microcontrolador cree que ha realizado varias vueltas simultáneamente antes de empezar a funcionar de forma correcta. Sabiendo que una vuelta son 967 pulsos del encoder, que el autoreload tiene un límite de 10000 cuentas y se resetea y que si realiza un cambio brusco, dicha cuenta de pulsos asciende de repente a una cifra mucho mayor a 967 porque piensa que ha realizado una vuelta completa instantánea, se aplica un filtro de paso bajo. En éste solo se dejan pasar señales un poco por encima de 967, de forma que el ruido generado por cambios no afecte al sistema y crea que la rueda ha realizado muchas más vueltas de las reales.

Muestra de valores obtenidos mediante el termite cuando se ejecutaba el programa y el robot se autobalanceaba.

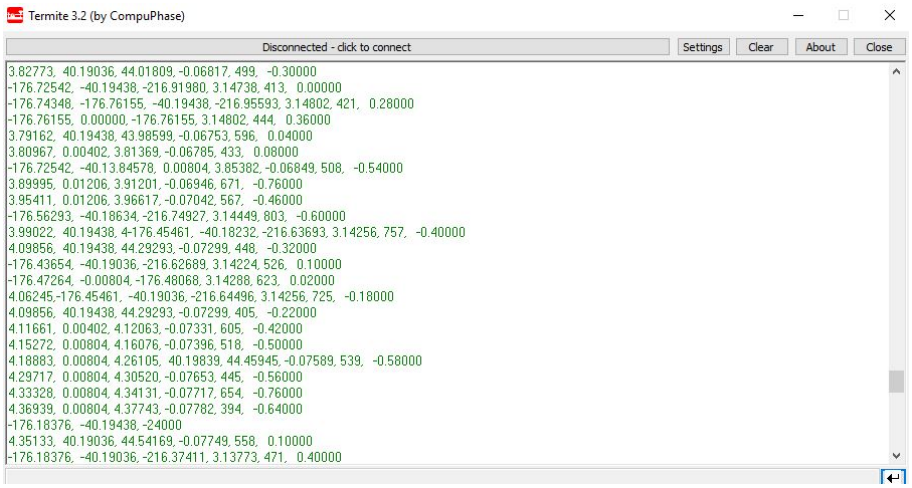


Figura 60 Control PD Desplazamiento – Termite

De izquierda a derecha los datos obtenidos son: “P” acción proporcional, “D” acción derivada, Salida del control PD de desplazamiento, distancia recorrida por la rueda, “p” de la PWM, Ángulo actual.

Valores finales de las ganancias para el control:

Ganancias Control PD Desplazamiento

Kp	Kd
56,15	0,1

Nota: Durante el transcurso de este ensayo se han realizado numerosas depuraciones del código (Reajustes de lógicas, adición de filtros, etc). Además ha sido necesario cambiar las características de la estructura. Entre otras se ha aumentado su altura y c.d.g. consiguiendo hacer el sistema dinámico más lento y por tanto su control más sencillo, consiguiendo una mayor estabilidad.

4.3.13 Ensayo 12: Estructura del robot

Este ensayo se desarrolla a lo largo de todo el proceso de la creación del prototipo incluido en los diversos test antes mencionados y explicados.

Según las necesidades, se iba ampliando la estructura. Al principio solo se trataba de componentes electrónicos conectados unos a otros. Al final, toda una estructura física que contiene todos los componentes, tanto fijos como móviles.

Se comenta y explica en más detalle en el siguiente apartado “Construcción del robot”.

4.4 CONSTRUCCIÓN DEL ROBOT

Al tratarse de un prototipo, durante el transcurso de su desarrollo se previó la necesidad de que hacer modificaciones en las proporciones y estructura. Por esta razón se ha optado por un modelo que sea de sencillo y rápido montaje y despiece, de forma que pueda quitarse, añadirse o modificarse según las necesidades que vayan surgiendo y observándose.

Por tanto, y como ya se comentó al inicio del apartado anterior, se trata de un modelo flexible, que ayuda a poder realizar los cambios y modificaciones que exija la situación en cada momento.

Además, ha sido necesaria la incorporación, modificación o creación de elementos electrónicos para el completo funcionamiento del prototipo. Es decir, ha sido necesaria la creación de un circuito para implementar una botonera, otro para poder conectar los motores al microcontrolador (Etapa de potencia), soldaduras, conexiones varias, etc.

Algunos de los componentes y como se han empleado ya se han enumerado en cada ensayo del apartado anterior.

Como base antes de definir un modelo hemos de tener claro nuestro objetivo. Según las necesidades ya comentadas a lo largo de la memoria llegamos a la siguiente conclusión.

Debemos tener una estructura alargada donde podamos colocar los diferentes dispositivos electrónicos, de fácil acceso a ellos en caso de tener que realizar modificaciones, donde podamos elevar el centro de gravedad. Necesitamos unir los motores en un solo eje de forma que la estructura pueda mantenerse en vertical con dos ruedas, esto es, las ruedas se colocan en paralelo para que puedan moverse al mismo tiempo como si estuvieran conectadas por un único eje.

Se opta finalmente por una estructura de tipo estantería donde en cada piso se podrán colocar los diferentes elementos necesarios para la construcción del prototipo.

Los materiales a emplear inicialmente previstos, por necesidades del proyecto se aumentan a medida que se avanza en el progreso de éste.

Una lista de los materiales empleados es la siguiente:

- Dispositivos electrónicos del robot.
- Motores CC y batería.
- Cableado.
- Tornillería (Incluidos tornillos y tuercas)
- Planchas de plástico transparente.
- Planchas de metal en L.
- Ruedas
- Pegamento
- Cinta adhesiva
- Plomos
- Estaño para soldaduras

Herramientas empleadas para la construcción del prototipo además de destornilladores, pie de rey, alicates, etc:

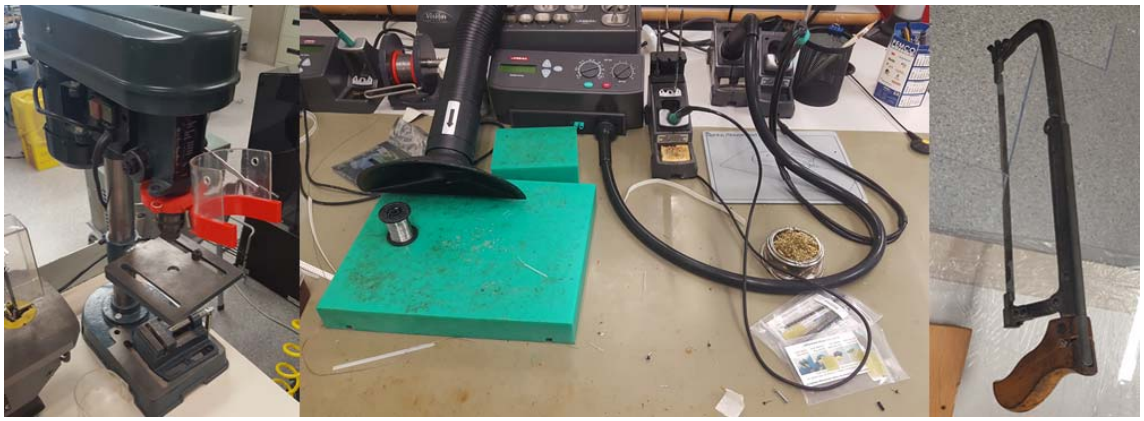


Figura 61 Taladradora, Puesto de soldadura, Sierra manual

A continuación se muestra una serie de imágenes en orden cronológico del montaje del prototipo.

Empezamos por la base. Necesitamos unir los motores con la estructura de forma que se mantengan estables en el sitio. Para ello cortamos dos perfiles en L de aluminio de tamaño 30x30x1,5mm y 7cm de longitud y los enfrentamos uno al otro. Se le practican los orificios mediante la taladradora para que coincidan el eje y los tornillos de sujeción del motor.

Se corta mediante el uso de la sierra manual las planchas de plástico que harán la función de bases para cada piso de la estructura. Al comienzo se probó con una sierra eléctrica para acelerar el proceso de corte, sin embargo se descubrió que al cortar tan rápido, se generaba demasiado calor, lo cual derretía el material empleado e imposibilitaba el correcto cortado de la pieza.

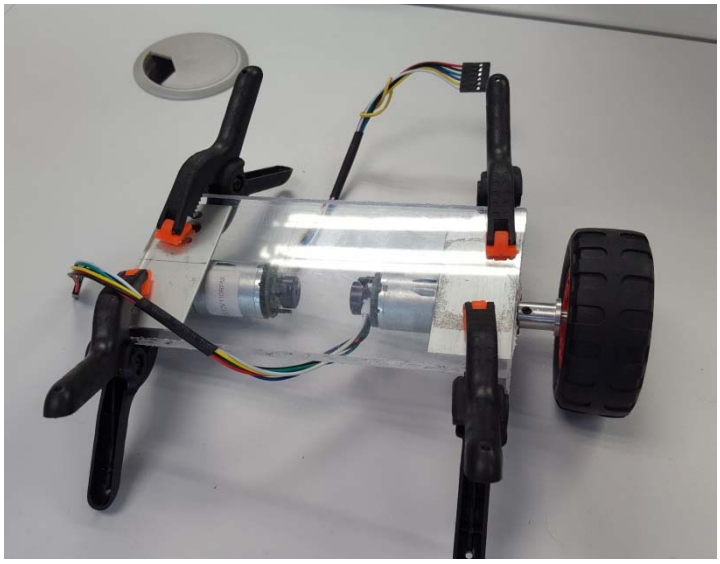


Figura 62 Construcción 1

Se pega mediante el uso de una pistola de encolar en caliente las dos perfiles en L de aluminio, una vez pegados se atornillan para asegurarlas del todo a la estructura. Se comprueba el correcto acople de la rueda con el eje del motor y la alineación de éstos. Este aspecto es importante para que luego el vehículo pueda desplazarse de forma correcta.

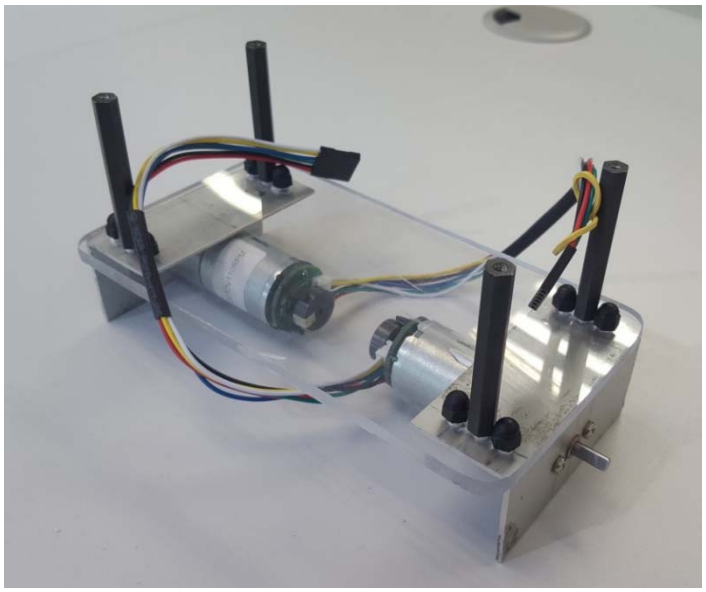


Figura 63 Construcción 2

Una vez acoplados los motores a la base, se comienza a montar la estructura en sí. Para comenzar se unen las tuercas espaciadoras a modo de columnas y ayudan a reforzar todavía más la unión entre la base u las planchas de metal en L.

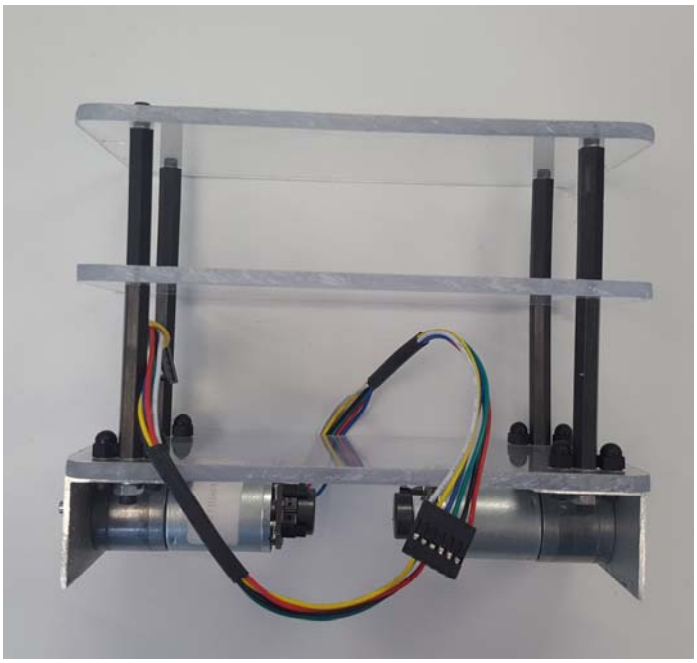


Figura 64 Construcción 3

Se procede del mismo modo atornillando hasta tener un total de 3 estanterías deseadas.

A cada plancha de plástico transparente se le han practicado, usando la taladradora, los orificios necesarios para poder acoplar los diferentes elementos.

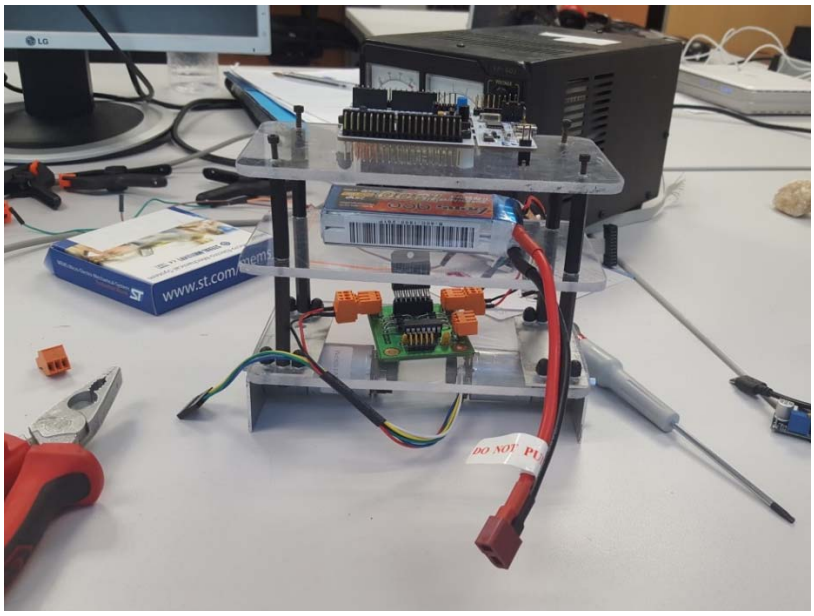


Figura 65 Construcción 4

Una vez montada la estructura principal se comienza a unir los diferentes dispositivos y elementos que harán que el robot pueda funcionar.

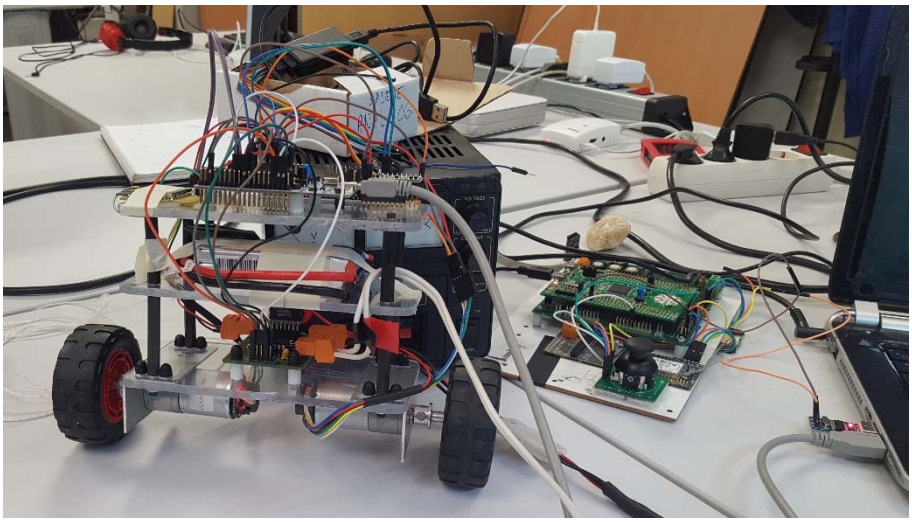


Figura 66 Construcción 5

La primera disposición escogida es la de:

- **1º Piso (Base):** Etapa de potencia (Por encima) y motores CC (Por debajo).
- **2º Piso:** Batería.
- **3º Piso:** STM32F411RE, IMU y Módulo RF.

El IMU se coloca en la parte superior pues es el emplazamiento donde notará en mayor medida la variación del ángulo de caída.

Una vez montado todo el robot, incluidas ruedas, se comprueba su funcionamiento y rendimiento.

Se constata que el sistema dinámico es demasiado rápido para la capacidad de acción de los motores y el control.

Además se nota que la batería no está bien sujeta a la estructura y se mueve cuando el robot está en marcha.

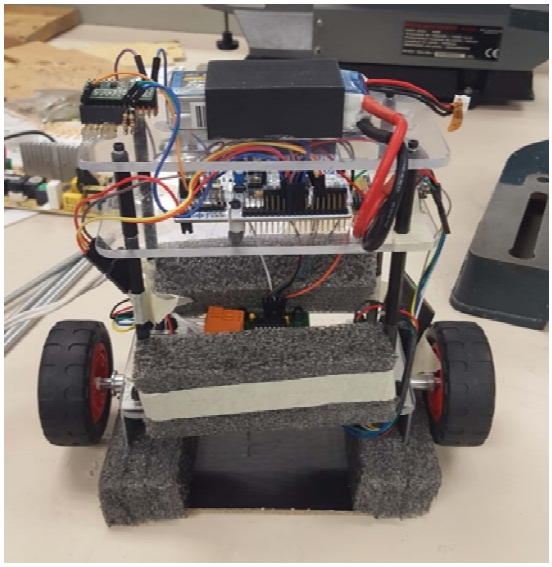


Figura 67 Construcción 6

Se le añaden cubos rectangulares de espuma de poliuretano para reducir el impacto de las caídas sobre la estructura y componentes.

Se construye una base para soportar el prototipo cuando no se desea que las ruedas toquen el suelo o para cuando repose y no caiga.

Se modifica la estructura añadiendo 4 tuercas espaciadoras más, es decir, se hacen más altas las columnas inferiores de forma que se eleve el centro de gravedad.

Además se cambia la disposición de los elementos.

La segunda disposición escogida es la de:

- **1º Piso (Base):** Etapa de potencia (Por encima) y motores CC (Por debajo). Incremento de longitud de sus columnas.
- **2º Piso:** STM32F411RE y Módulo RF.
- **3º Piso:** Batería, IMU.

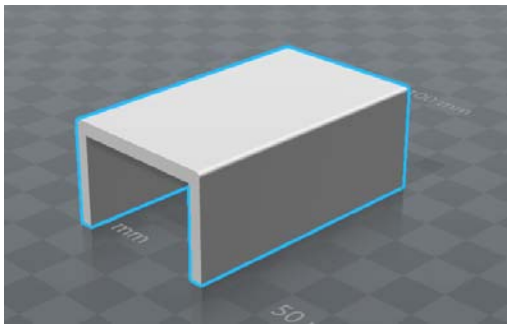


Figura 68 Caja 3D

Se le añade a la parte superior un módulo impreso en 3D al cual llamaremos “Caja” para poder introducir la batería dentro y que esta quede bien fija de forma que no se mueva del sitio cuando el robot se balancee. Además permite extraer la batería del sito, si se desea, fácilmente.

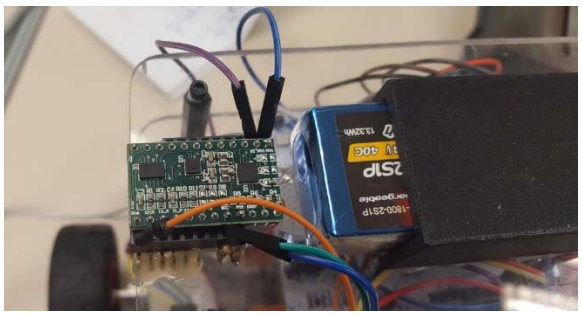


Figura 69 Construcción 7

En la imagen anterior se puede ver más de cerca la implementación del IMU y la batería en la estructura. El IMU se acopla a unos conectores que se han predispuesto para ser pegados a la superficie superior de forma que dicho dispositivo no sufra oscilaciones fuera del balanceo del vehículo y además sea más sencillo el acople de los diferentes conectores necesarios para su funcionamiento.

Una vez más, en esta segunda versión del prototipo se comprueba que el sistema sigue siendo demasiado rápido para la capacidad de los motores empleados. Sin poder cambiar éstos, nos queda únicamente la posibilidad de volver a hacer más alta la estructura y elevar todavía más el c.d.g.

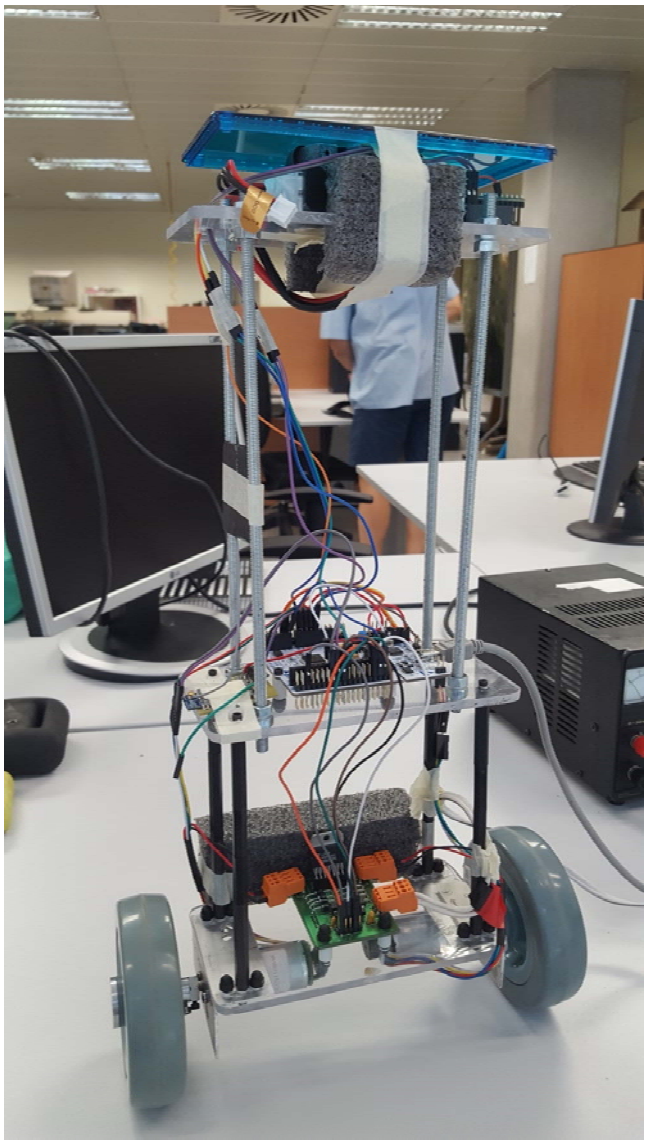


Figura 70 Construcción 8

La tercera disposición escogida es la de:

- **1º Piso (Base):** Etapa de potencia (Por encima) y motores CC (Por debajo). Incremento de longitud de sus columnas.
- **2º Piso:** STM32F411RE y Módulo RF. Adición de nuevas columnas de vara roscada.
- **3º Piso:** Batería, IMU.

Las columnas que antes unían el tercer piso con el segundo se trasladan al primero para aumentar la altura del segundo piso.

El añadido de las columnas de vara roscada permite poder elevar o disminuir la altura del último piso según se desee.

Inicialmente se había empleado unas ruedas de plástico del juego "Meccano" de 7 cm de diámetro. Éstas eran ligeras por lo que reducían el peso en la base, pero sin embargo no tenían un buen agarre. Por esta razón, necesidad del control se ha cambiado las ruedas.

Las nuevas ruedas son ruedas de gomas elásticas de caucho. Tienen un buen rendimiento de rodadura y, aunque su peso y diámetro (10cm) sean superiores, bajando el c.d.g., ofrecen un

mejor agarre a la superficie y suavizan el movimiento del robot inhibiendo ligeramente el efecto producido por las perturbaciones inherentes a los motores. Se comprueba que se mejora el control del robot al aplicar estas nuevas ruedas. Se añade al código del programa de control el nuevo radio de las ruedas para el cálculo de distancia recorrida.

Se descubre que por efecto de las vibraciones algunos elementos de tornillería empiezan a aflojarse. Se aumenta el par de apriete de éstos para evitar dicho efecto.

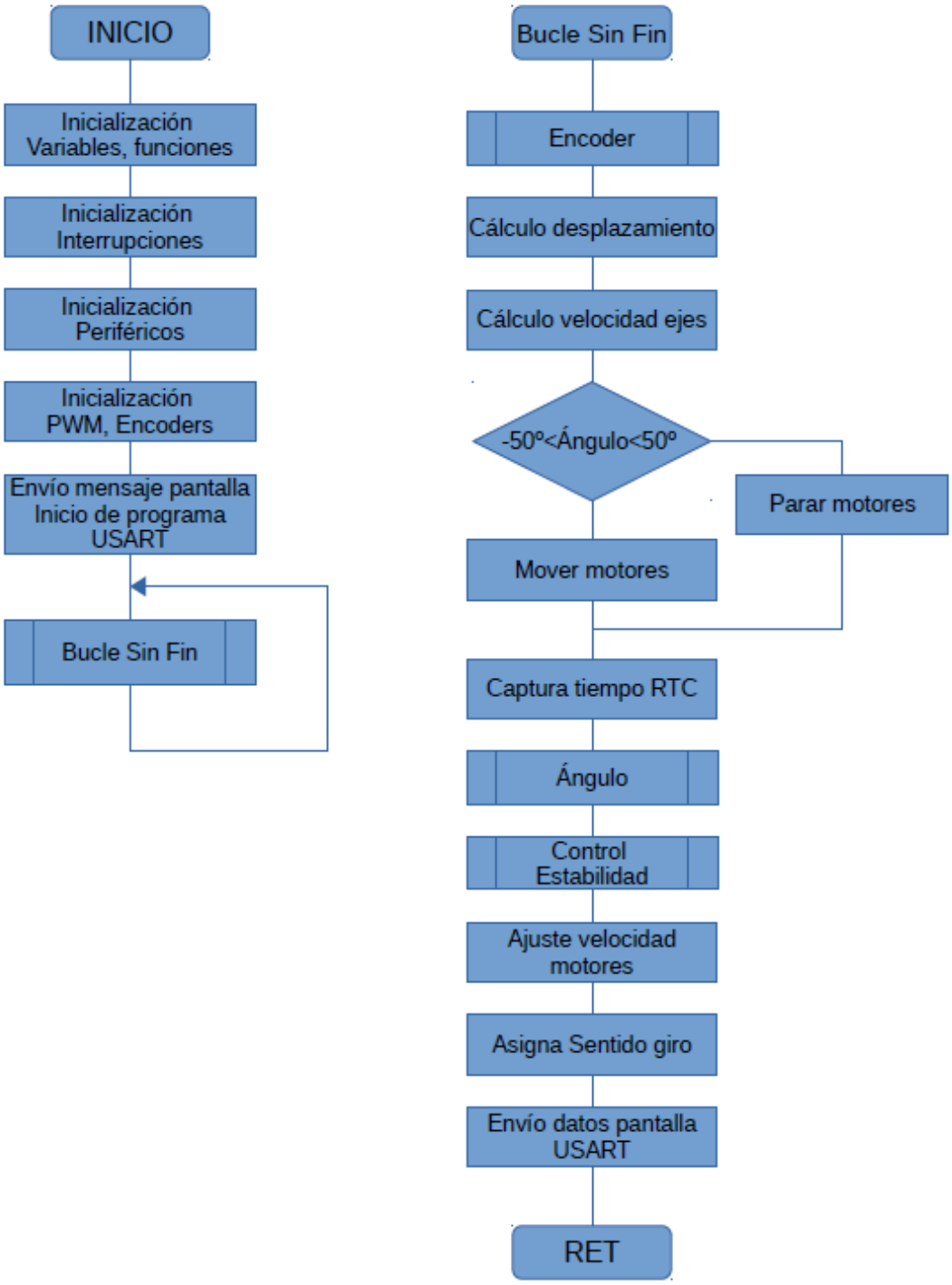
No se detectan problemas en la robustez de la estructura.

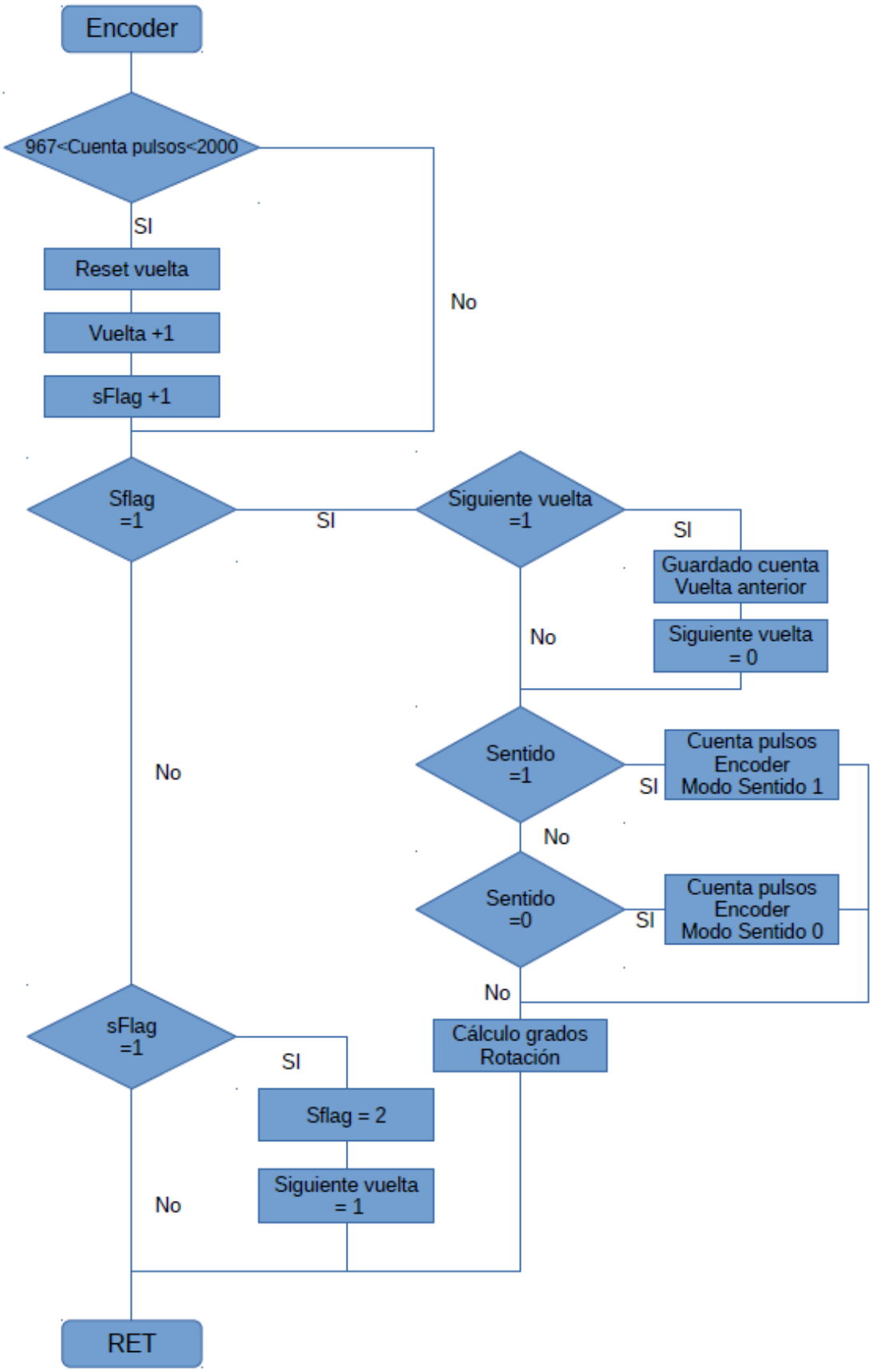
Se añaden 3 piezas de plomo de 0,5cmx3,5cm que pesan 40 gramos cada uno haciendo un total de 160g extra de masa. Se colocan en la parte superior de la "Caja" donde va fijada la batería.

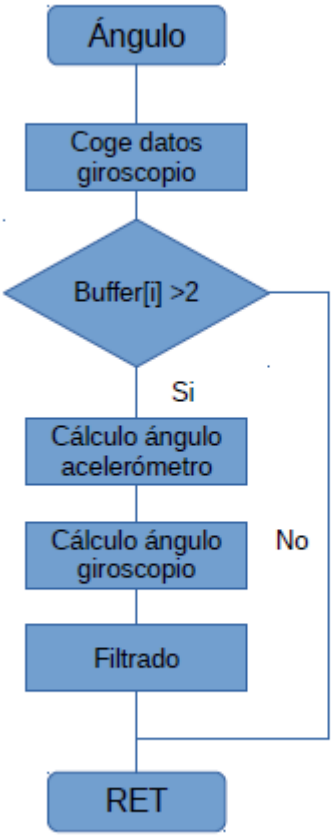
Encima de la "caja" se ha colocado una plancha de plástico de 8x16x0,5 cm a modo de plataforma para colocar objetos encima. Esta plataforma es empleada únicamente para ensayos y va pegada a la "caja" de la batería. Encima de ésta se han realizado probar con diferentes pesos para comprobar el comportamiento del sistema dinámico.

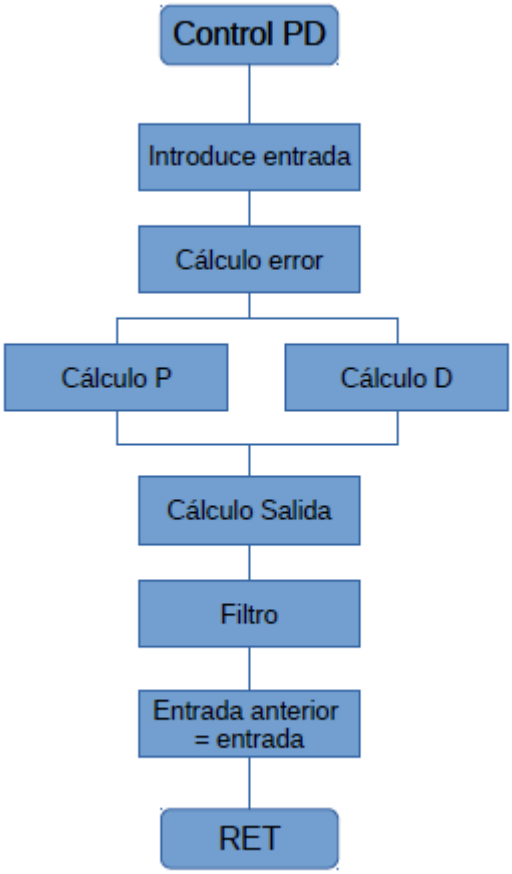
4.5 FLUJOGRAMAS

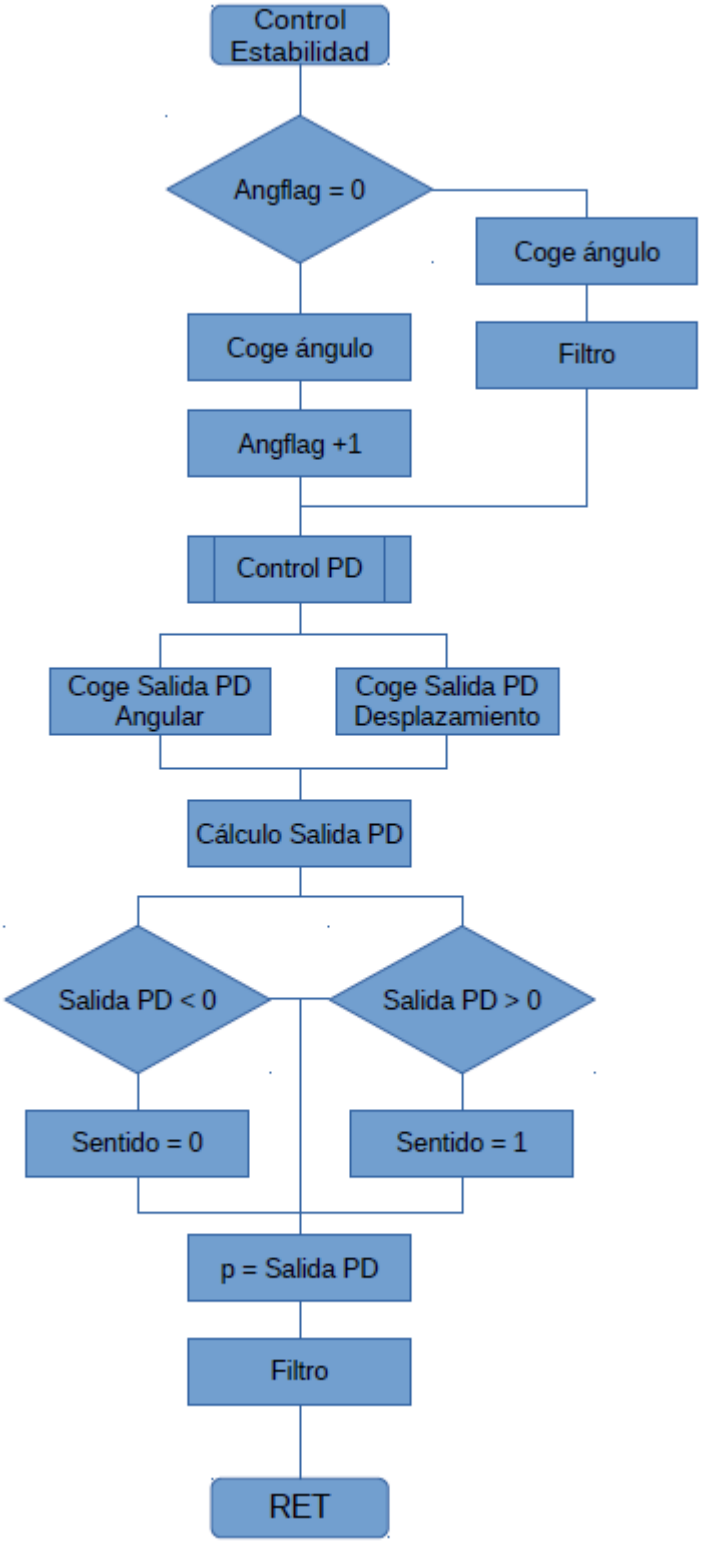
Para entender de forma simplificada y esquemática el programa final se pueden observar los siguientes flujogramas.




Encoder
967<Cuenta pulsos<2000
SI
No
Reset vuelta
Vuelta +1
sFlag +1
Sflag =1
SI
Siguiente vuelta =1
SI
Guardado cuenta Vuelta anterior
Siguiente vuelta = 0
No
Sentido =1
SI
Cuenta pulsos Encoder Modo Sentido 1
No
Sentido =0
SI
Cuenta pulsos Encoder Modo Sentido 0
No
Cálculo grados Rotación
No
sFlag =1
SI
Sflag = 2
Siguiente vuelta = 1
No
RET


Ángulo
Coge datos giroscopio
Buffer[i] >2
Si
No
Cálculo ángulo acelerómetro
Cálculo ángulo giroscopio
Filtrado
RET


Control PD
Introduce entrada
Cálculo error
Cálculo P
Cálculo D
Cálculo Salida
Filtro
Entrada anterior = entrada
RET


Control Estabilidad
Angflag = 0
Coge ángulo
Filtro
Coge ángulo
Angflag +1
Control PD
Coge Salida PD Angular
Coge Salida PD Desplazamiento
Cálculo Salida PD
Salida PD < 0
Salida PD > 0
Sentido = 0
Sentido = 1
p = Salida PD
Filtro
RET

4.6 PROBLEMAS ENCONTRADOS

La creación de un prototipo, en este caso de un vehículo autobalanceado, no está exenta de la aparición de problemas durante su desarrollo.

A lo largo de los diferentes ensayos se han ido encontrando diferentes incógnitas, imprevistos y problemas varios los cuales ha sido necesario solventar y superar.

A continuación se puede comprobar una lista general de cuáles son los problemas que han sucedido y se han solucionado y pueden servir de guía para aquel que quiera desarrollar este mismo tipo de proyecto y pueda preverlos.

- Errores a la hora de programar (Fallos de escritura, falta de librerías, no existencia de códigos, funciones, necesarios…)
- Obtención previa de información, fichas técnicas sobre los distintos elementos a utilizar. Difícil acceso.
- Falta de información por parte del proveedor (Ejemplo: Fichas técnicas de los motores con toda la información posible)
- Errores en alguna ficha técnica.
- Al no poseer datos exactos del proveedor de los motores DC han sido necesarias varias pruebas para conocer el comportamiento de éstos. A su vez, los motores han presentado algunos problemas los cuales solo se ha podido suponer ciertas hipótesis sobre las causas.
- Conocimiento sobre los cálculos o procedimientos a realizar en cada momento. Investigación y aprendizaje de nuevos conceptos.
- Fallos con algunos elementos del hardware (Malas conexiones, elementos de motor atascados, cables con mala conexión…)
- Mala o nula ejecución del programa y búsqueda de errores que puedan haberlo ocasionado (En ocasiones han sido de hardware y otras de software).
- Necesidad búsqueda y utilización de programas y aparatos específicos para realizar medidas, actualizar drives, etc (ST Link, Saleae logic, multímetro, osciloscopio…)
- Necesidad de búsqueda e instalación de drivers muy concretos para la utilización de programas y periféricos.
- Fallos de precisión o capacidad de obtención de variables por parte de los sensores(Ejemplo, sensor efecto Hall a veces no medía bien)
- Fallos de comunicación entre los periféricos, el microcontrolador y el ordenador.
- Fallos de funcionamiento de algunos periféricos o malas conexiones.
- Incompatibilidad entre versiones de programas.

5. RESULTADOS

Este apartado se puede considerar como una recopilación de los demás ensayos, su consecución y realización en un prototipo.

5.1 MONTAJE FINAL

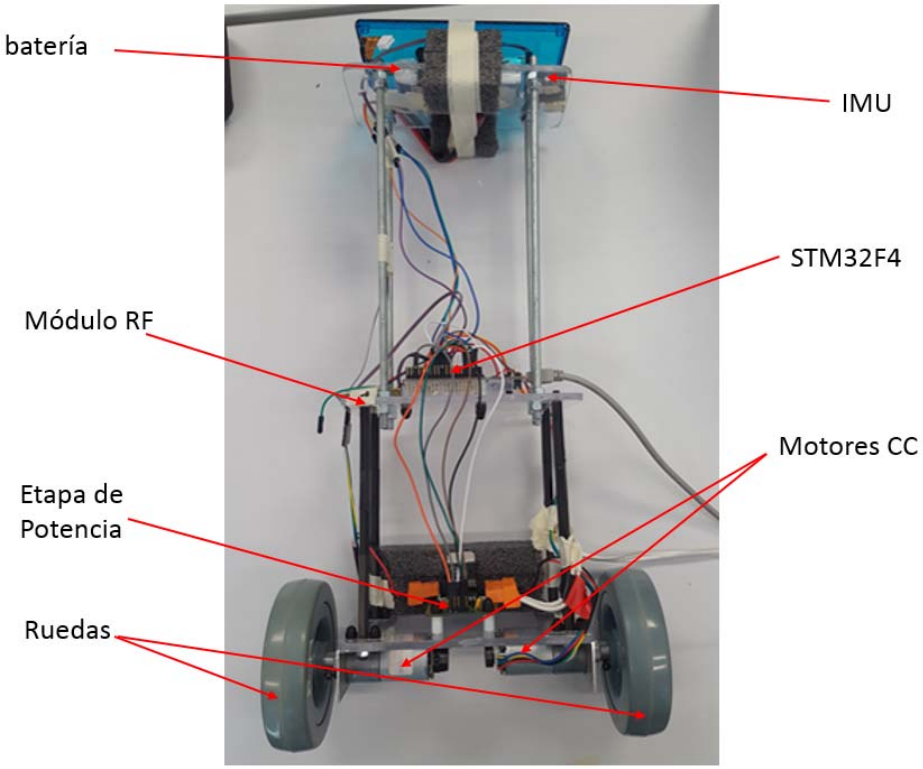


Figura 71 Alzado Robot

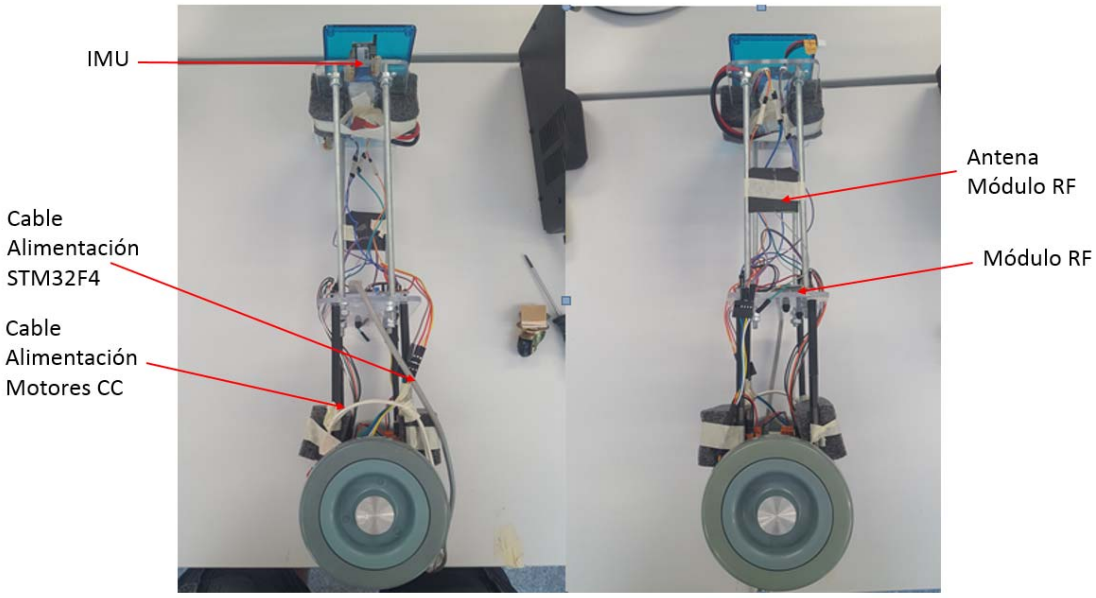


Figura 72 Perfiles del robot

La disposición final escogida es la de:

- **1º Piso (Base):** Etapa de potencia (Por encima) y motores CC (Por debajo).
- **2º Piso:** STM32F411RE y Módulo RF.
- **3º Piso:** Batería, IMU.

Se trata de la tercera disposición explicada al final del punto de "Construcción del Robot".

Las medidas de este modelo pueden observarse en los planos anexos.

5.2 PUESTA EN MARCHA

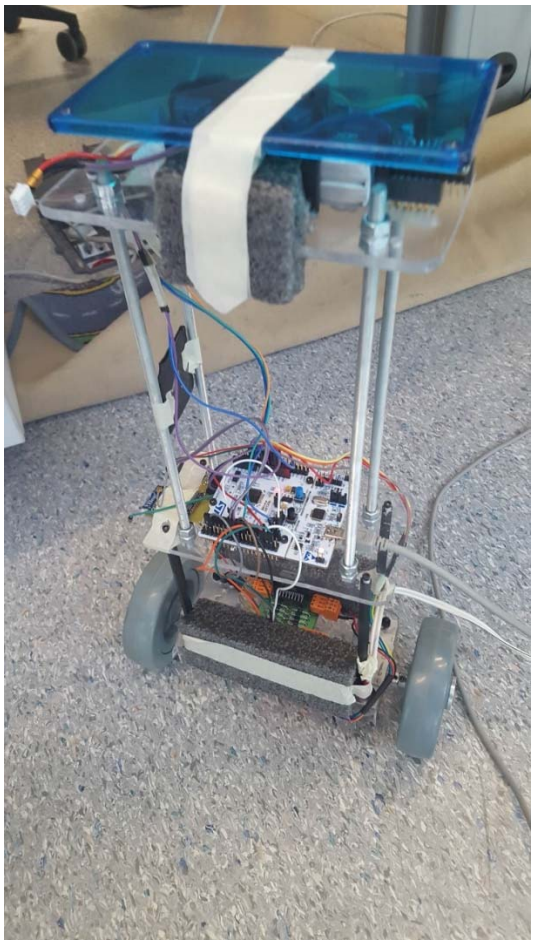


Figura 73 Montaje Final en funcionamiento

La puesta en marcha se puede considerar como un resumen del Ensayo 12.

Se verifican las conexiones de los diferentes elementos del sistema y se comprueba el correcto ensamblaje de la estructura. Se depura el código implementado dentro del STM32F411RE.

El funcionamiento final del robot se ve condicionado por:

- Las mediciones realizadas por los sensores IMU y Encoders.
- Capacidad de respuesta de los motores.
- Robustez y características de la estructura .
- Control Automático implementado.

Para el control se aplica la suma de los dos controles PD, “Angular” y de “Desplazamiento” deviniendo en un control más preciso y capaz de mantener el equilibrio en vertical del robot.

Una muestra de las medidas obtenidas al funcionar el robot con su programa final es la siguiente:

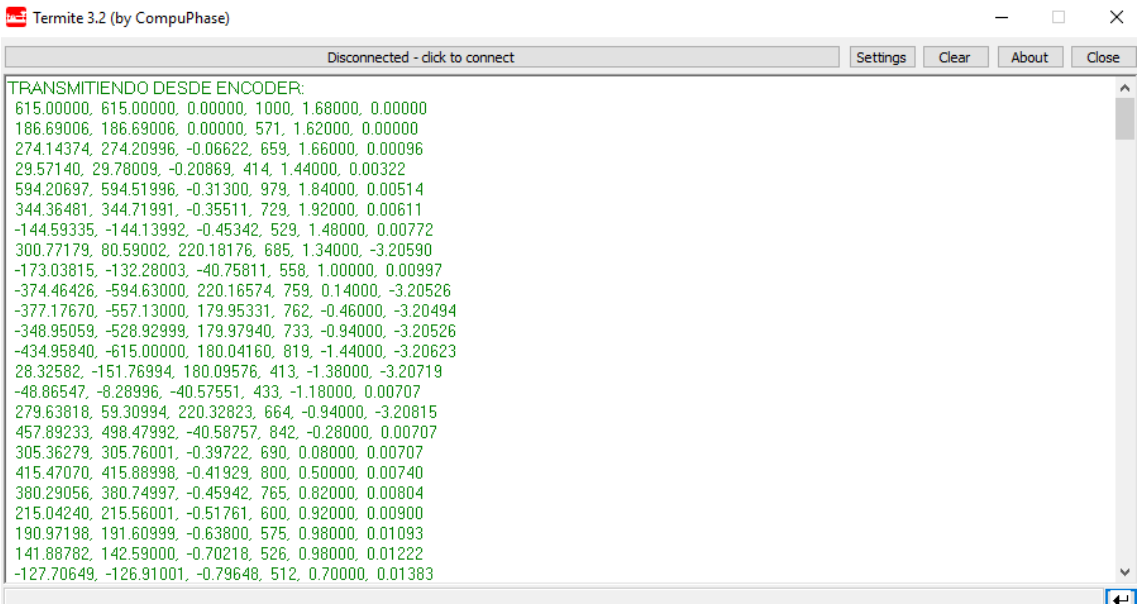


Figura 74Muestra Salidas Control y Mediciones

De izquierda a derecha: Salida final del PD, Salida del PD angular, Salida del PD desplazamiento, valor PWM “p”, Ángulo actual y Distancia recorrida respecto a la referencia.

Finalmente se ha conseguido mantener el robot de pie durante un corto periodo de tiempo de unos 5 segundos. Las razones de ello se explican en el apartado de conclusiones y en el ensayo 12.

6. VALORACIÓN ECONÓMICA

El objeto de este apartado es determinar y estimar el coste total que supone la realización de este proyecto.

Se ha realizado bajo el supuesto de que todos, o casi todos los materiales empleados para la terminación de este proyecto han sido necesario comprarlos expresamente.

A la hora de realizar el presupuesto se ha tenido en cuenta como un proyecto de investigación exclusivo para un cliente. Se destina a un ingeniero trabajando exclusivamente para este proyecto, además de los costes indirectos que dicho trabajo puede generar.

Por tanto, para la realización de este apartado se ha tenido en cuenta tanto los costes de material, software, construcción, mano de obra e indirectos que pueda suponer.

Nº	Descripción	Unidades	Cantidad	Coste
	Estructura			
1	Tornillo cabeza avellanada M2. Aleación de acero con cabeza hueca hexagonal (Lote)	Unidad	50	2,07 €
2	Tornillo acero Inoxidable M2 x 10mm (Lote)	Unidad	10	1,40 €
3	Tornillo cabeza plana Allen acero M3 x 10mm (Lote)	Unidad	100	3,02 €
4	Tornillo cabeza plana Allen acero M3 x 16mm (Lote)	Unidad	100	3,56 €
5	Varilla roscada acero inoxidable SIN975 M7, L=1m, D=6mm	Unidad	1	2,02 €
6	Tuerca acero zincado DIN 934 M7 (Lote)	Unidad	100	1,65 €
7	Tuerca métrica de nylon plástico hexagonal DIN1587 M7 (Lote)	Unidad	20	4,02 €
8	Plancha metacrilato transparente 700x500x5mm	Unidad	1	13,45 €
9	Perfil en L aluminio 300x300x15mm y 1m de largo	Unidad	1	2,77 €
10	Espaciadores hexagonales nylon negro con tuerca M3 (Lote)	Unidad	1	2,26 €
11	Espaciador cilíndrico nlylon M3x15mm (Lote)	Unidad	20	2,81 €
12	"Caja" en color negro impresa en 3D 20x35x60mm	Unidad	1	4,96 €
13	Lastre plomo 40 gramos	Unidad	3	1,64 €
14	Rueda goma elástico gris (Caucho). D=100mm	Unidad	2	9,01 €
			Subtotal:	45,64 €
	Electrónica			
15	Motor corriente continua 12V y 110rpm max. Se incluye encoder con doble sensor de efecto hall	Unidad	2	33,74 €
16	Batería de litio 1500mAh-7,4V recargable	Unidad	1	15,69 €
17	Etapa de potencia. Incluidos pines de conexión al microcontrolador, 2 motores CC y entrada alimentación 12V	Unidad	1	10,74 €
18	Microcontrolador STM32F411RE Núcleo	Unidad	1	9,50 €
19	Microcontrolador STM32F3 Discovery	Unidad	1	11,93 €
20	Modulo interruptor Sensor efecto Hall Sunfounder. Incluido 3 cables pin anti-reverso	Unidad	1	4,13 €
21	Unidad de medida Inercial STEVAL-MKI124V1. Incluye módulos L3GD20 (Giroscopio), LSM303DLHC (Acelerómetro) y LPS331AP (Sensor presión/altitud)	Unidad	1	23,29 €
22	Módulo de radiofrecuencia XBee XB8-DMUS-002	Unidad	2	45,74 €

23	Adaptador USB a USART para microcontrolador	Unidad	1	1,52 €
24	Material electrónico: resistencias, PCBs, pines, estaño, varillas metálicas	Unidad	1	18,60 €
25	Cables pines 20cm hembra-hembra para conexión microcontrolador (Lote)	Unidad	40	4,12 €
26	Cables pines 20cm macho-hembra para conexión microcontrolador (Lote)	Unidad	40	4,12 €
27	Cables pines 20cm macho-macho para conexión microcontrolador (Lote)	Unidad	40	4,12 €
28	Botonera digital para microcontrolador. 4botones	Unidad	1	3,76 €
			Subtotal:	190,99 €
	Software			
29	Saleae Logic. Analizador lógico para realización de medidas. Incluido software específico.	Unidad	1	102,82 €
30	ARM Keil µVision software aplicado al proyecto	Unidad	1	435,07 €
			Subtotal:	537,89 €
	Otros			
31	Empleo y alquiler de herramientas (taladradoras, Puesto soldadura, sierras, etc)	Unidad	1	116,18 €
32	Mano de obra y costes indirectos	Unidad	1	3.068,18 €
			Subtotal:	3.184,36 €
			BASE IMPONIBLE:	3.958,88 €
			IVA 21%	831,36 €
			TOTAL:	4.790,25 €

El precio final, incluido IVA, es por tanto:

CUATRO MIL SETECIENTOS NOVENTA EUROS CON 25 CÉNTIMOS.

7. CONCLUSIONES

7.1 CONCLUSIONES

Después de haber finalizado y durante el desarrollo de este prototipo se ha llegado a una serie de conclusiones sobre el proyecto en sí. Algunas conclusiones ya han sido nombradas y argumentadas durante la explicación de los diferentes ensayos y otras pueden verse mostradas en este apartado. Como resumen de éstas puede afirmarse lo siguiente.

- La realización de este tipo de proyectos ayuda al ingeniero en formación a motivarse y ejerce una influencia positiva sobre él consiguiendo que en el transcurso de su desarrollo se consiga repasar conceptos aprendidos durante el desarrollo del máster, ampliarlos e incluso descubrir nuevos (Filtros, modos de control, tipos de dispositivos en el mercado, mejora en lógica de programación, etc)
- Se ha constatado la importancia que tiene la información. Cuanto mejor sea la capacidad de acceso a información mayores probabilidades de éxito tiene un proyecto tanto de investigación, económico o de cualquier materia. También se confirma, por tanto, la importancia de una buena formación.

Respecto al prototipo:

- El robot es capaz de mantenerse de pie, cumpliendo con el objetivo de mantenerse perpendicular a la superficie de apoyo de las ruedas durante un corto periodo de tiempo a modo de péndulo invertido. Esto es debido principalmente a las siguientes razones:
    1. Existe holgura en la reductora de los motores CC lo cual añade una inercia extra cada vez que cambia de sentido además de retardar este cambio, pues debe recorrer el tramo vacío de dicha holgura en los engranajes antes de empezar a moverse la rueda. Esto crea retardos en la acción real de control y genera inestabilidades en el sistema que de funcionar de forma fluida no existirían.
    2. Por las características generales del sistema y defectos varios de los componentes de los motores, éstos no tienen la suficiente potencia para ejercer las acciones de control de estabilidad de forma adecuada.
    3. Ruido en las señales de entrada por parte de los sensores.
    4. Poco margen en el límite ángulo de caída acortando el rango de acción posible.

  En este punto se resume la importancia de la fiabilidad y calidad de los componentes a la hora de crear un diseño de calidad.

- Basándonos en la afirmación anterior podemos concluir que es necesario el cambio de motores para un correcto funcionamiento y facilidad del control. Éstos no deberán tener ningún tipo de holgura entre el eje del motor y el eje de la rueda y deberá moverse con fluidez. Además, los motores deberán tener una mayor potencia si se sigue usando la misma estructura o de características similares.

- Cuanto más elevado está el centro de gravedad el sistema dinámico es más inestable, pero sin embargo, se vuelva más lento. Esto hace el control sea más sencillo y por tanto más eficaz. Otra confirmación de la falta de potencia de los motores se muestra cuando se añaden plomos para elevar el c.d.g. todavía más. Se consigue dicho objetivo, aunque a su vez se aumentaba la masa, lo cual reduce a su vez la capacidad de respuesta de los no siendo capaces de realizar una acción adecuada si el peso es excesivo.

- Sin un modelo dinámico claro y un sistema de control robusto que lo respalde se incrementa la dificultad de obtención de estabilidad en el modelo. Sin embargo, se comprueba que en ocasiones no es posible aplicar este tipo de sistemas en su totalidad, y a aunque se pueda apoyar en ellos, hay variables que pueden hacer necesarias ampliaciones mediante métodos empíricos. Es importante tener en cuenta el efecto de las perturbaciones, tanto externas como internas al sistema.

- La señal del sensor del IMU presenta ruido. Esto es debido a varias razones.

    1. La propia sensibilidad del sensor. Detecta variaciones muy pequeñas de ángulo lo cual hace que le afecte en mayor medida las vibraciones.
    2. El sensor empleado presentaba desviaciones inherentes en sus medidas, los cuales se hace necesario calibrar.
    3. Presencia de gran cantidad de ruido en la señal.
    4. La presencia de vibraciones en el robot genera lecturas falsas de ángulo generando más ruido.

- Se comprueba, por tanto, la importancia de una buena configuración de los periféricos que componen un sistema. Éstos hacen la función de los “sentidos” y “músculos” de él. Sin información fiable del exterior no es posible ejercer un buen control del elemento a regular por muy buen procesador que tenga. Sin unos accionamientos adecuados no es posible ejercer de forma correcta las órdenes mandadas por el regulador, por muy correctas o precisas que éstas sean.

- La aplicación de un microcontrolador en este tipo de sistemas supone una ventaja. Por una parte se reduce el coste económico y consumo energético del sistema. El espacio empleado por él es reducido. Al estar focalizado a pocas funciones procesa mejor los datos y es mucho más rápido, por lo tanto es una gran opción para un proceso específico como el nuestro.

- Una buena programación del microcontrolador y los diferentes elementos es importante. Un código depurado y eficiente siempre dará mejores resultados. Por ejemplo, la aplicación de filtros en el código mejoran el rendimiento, acotan señales y ayudan a eliminar información falsa que lo único que haría sería desviar al control de su verdadero objetivo.

Respecto a normativa:

- Una vez asentado los conceptos físicos que asientan las bases para la creación del producto final, aunque no se incluye en el alcance de este prototipo, se deberá tener en cuenta y comprobar las diferentes normativas vigentes respecto a este tipo de robots. Hoy en día, al tratarse de un producto bastante moderno existe poca regulación de éstos. Existen algunas normas respecto a la circulación de este tipo de robots, usados a modo de vehículo para personas, como ordenanzas según localidad. Respecto a seguridad y requerimientos por estándares de calidad que debe requerir, únicamente se ha encontrado información sobre la norma UL 2272 que reúne los diferentes test y características constructivas que debe reunir un "hoverboard", aparato similar al robot autobalanceado, donde la "estructura" es la propia persona. Para testeo eléctrico (Sobrecarga, cortocircuito, aislamientos, etc), mecánico (Vibraciones, choque, resistencia a esfuerzos, etc), medioambiental (Exposición al agua, cambios de temperatura), materiales empleados, modo de construcción.

Respecto a la vertiente económica:

- Se comprueba el coste que puede llegar a generar un proyecto de investigación. Esto es importante conocerlo pues se trata de una inversión a medio-largo plazo. En este tipo de proyectos se espera tener "pérdidas" cuando se desarrolla un nuevo producto, pues éste sale mucho más caro. Sin embargo el retorno de la inversión se vislumbra una vez el prototipo está terminado, corregido y verificado y se puede lanzar al mercado. Estas nuevas unidades tendrán un coste mucho menor y de ellas se podrá obtener un beneficio.

En resumen, se ha creado un prototipo de forma que cumple con sus objetivos en el grado de conocer e investigar las diferentes disciplinas que comprenden este tipo de proyecto obteniendo los puntos flacos y fuertes de éste, aprendiendo durante el proceso. Se ha podido conocer en mayor profundidad su comportamiento y estructura de manera que se ha conseguido cumplir con el objeto de asentar unas bases para futuros proyectos similares agilizando y mejorando la calidad de éstos. Además se ha podido realizar una introducción a las características del mercado en el que se incluye.

7.2 PROPUESTAS

A continuación se muestra y enumera una serie de proposiciones, ideas, tanto para mejoras como ampliaciones del proyecto que se ha descrito en esta memoria.

- Estructura modelada en 3D
- Estudio de una estructura dinámica y aplicación real de ésta.
- Aplicación del modelo LQR para control.
- Aplicación del filtro de Kalman.
- Cambio de motores por otros de más potencia y/o mayor suavidad de movimiento.
- Adición de sensores (Por ejemplo, ópticos o de ultrasonido).
- Ajuste e implementación de una batería
- Control remoto por radio mediante joystick, botonera, comandos, etc.
- Control remoto por GPS.

8. BIBLIOGRAFÍA

- SERRANO MARTÍN, JUAN JOSÉ. Apuntes de "PoliformaT" de la asignatura de "Sistemas Embebidos" del "Máster en Ingeniería Mecatrónica" de la UPV.

- HURTADO PÉREZ, ELÍAS J. 2007. *Máquinas Eléctricas Dinámicas*. Valencia. ISBN: 978-84-611-5693-1.

- HARBISON, SAMUEL P.; STEELE JR., GUY L. 1991. *C: A Reference Manual*. 3rd ed. New Jersey: Englewood Cliffs. ISBN: 0-13-110941-3.

- http://robots-argentina.com.ar/MotorCC_ControlAncho.htm

- SALT LLOBREGAT, JULIÁN J.; CUENCA LACRUZ, ÁNGEL; CASANOVA CALVO, VICENTE; CORRECHER SALVADOR, ANTONIO. 2015. *Control Automático Tiempo Continuo y Tiempo Discreto.* Valencia. ISBN: 978-84-291-4751-3.

- ST. UM1730 User Manual.

- Fichas técnicas de los diferentes elementos empleados.

- http://www.pieter-jan.com/node/11

- http://www.naylampmechatronics.com/blog/45_Tutorial-MPU6050-Aceler%C3%B3metro-y-Giroscopio.html

9. ANEXOS

- **Glosario.**
- **Planos.**
- **Código principal.**

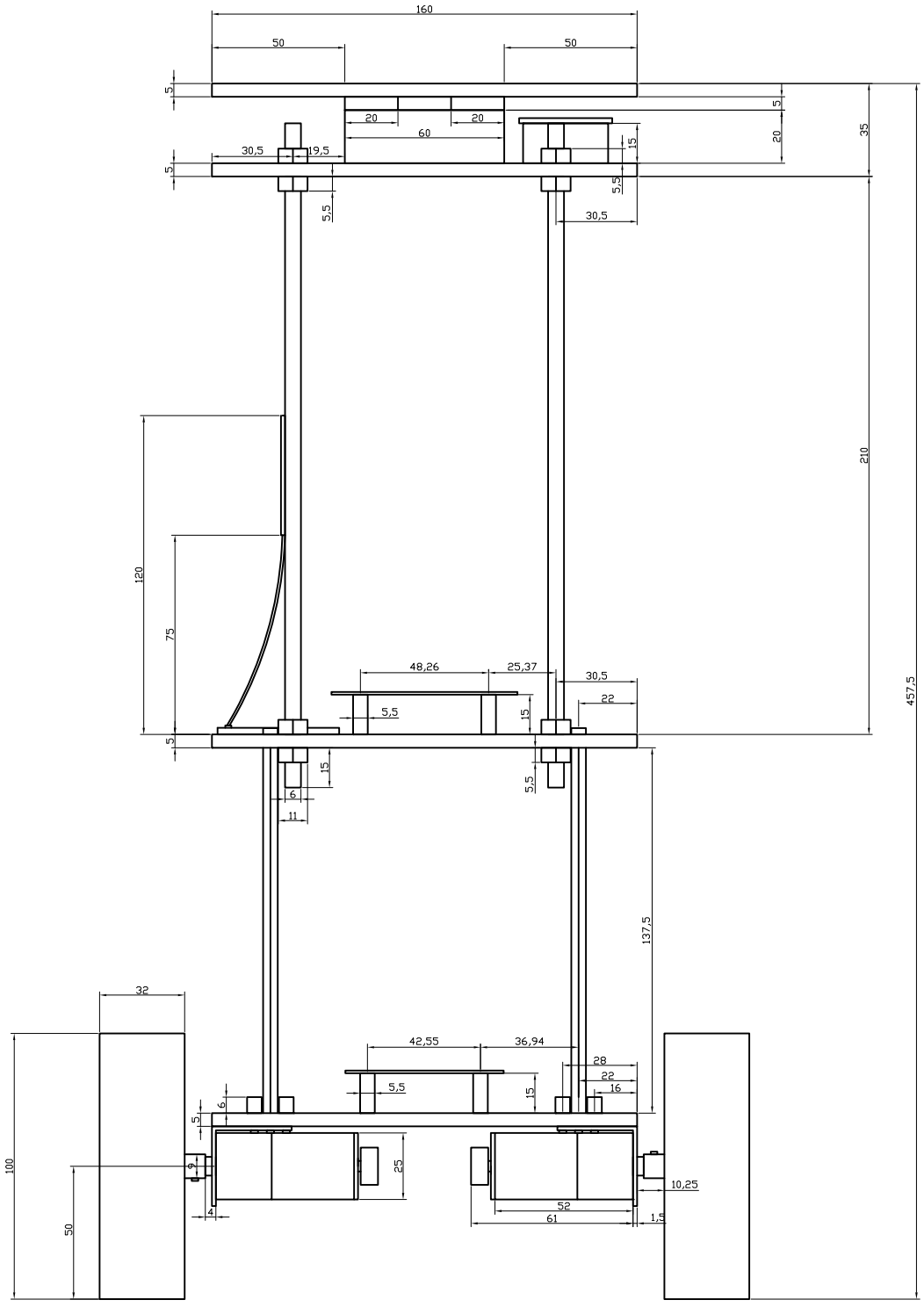
9.1 Glosario

IMU	Inertial Meauremente Unit (Unidad de Medida Inercial)	α	Aceleración angular (rad/s2)
MCU / µC	Microcontroller Unit (Microcontrolador)	θ_w	Ángulo de rotación de las ruedas (rad)
LED	Light Emitting Diode (Diodo emisor de luz)	θ_p	Ángulo de rotación del péndulo (rad)
CC	Corriente Continua	I_w	Momento de inercia de la rueda (kgm2)
STM32 / STM32F4	Microcontrolador STM32F411RE Nucleo	I_p	Momento de inercia de l péndulo (kgm2)
USB	Universal Serial Bus	T_w	Par motor (Nm)
TIM	Timer (Temporizador, reloj)	I_R	Inercia de rotor (kgm2)
ch	Channel (Canal)	T_a	Par de carga (Nm)
GPIO	Pin de entrada/salida	CR, CL	Par aplicado por el motor a las ruedas (Nm)
GND	Ground (Masa)	k_m	Constante de par (Nm/A)
c.d.g	Centro de gravedad	k_e	Constante de fcem (Vs/rad)
mw	Masa de la rueda (kg)	R	Resistencia nominal (Ω)
mp	Masa del péndulo (kg)	V_a	Tensión aplicada a los motores (V)
H_R, H_L, P_R, P_L	Fuerzas de reacción entre la rueda y la estructura (N)	r	radio de la rueda (m)
H_{fR}	Fuerza de rozamiento entre el suelo y la rueda (N)	g	Aceleración de la gravedad (m/s2)
θ	Posición angular (rad)	l	Longitud entre el centro de las ruedas y el c.d.g del péndulo (m)
ω	Velocidad angular (rad/s)		

9.2 Planos

En las próximas páginas pueden observarse los planos de referencia creados del prototipo.

- **Plano 1:** Alzado del robot autobalanceado.
- **Plano 2:** Perfil del robot autobalanceado.



	Fecha	Nombre
Dibujado	Septiembre 2017	JFG
Comprobado	Septiembre 2017	JFG
Nombre plano	Alzado Robot Autobalanceado	

Escuela Técnica Superior de Ingeniería del Diseño

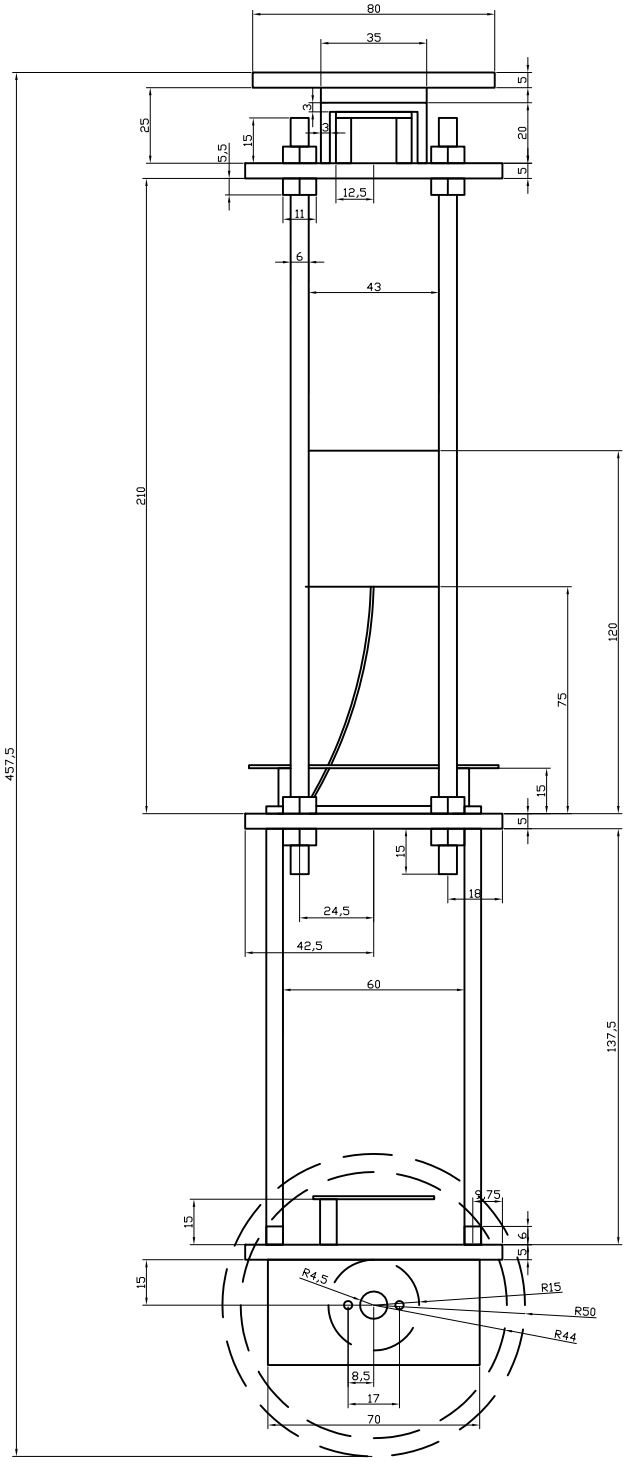
Escala: 1:2.5

DISEÑO E IMPLEMENTACIÓN DE UN ROBOT AUTOBALANCEADO

Plano nº: 1

Trabajo Fin de Máster

Máster en ingeniería Mecatrónica



	Fecha	Nombre
Dibujado	Septiembre 2017	JFG
Comprobado	Septiembre 2017	JFG
Nombre plano	Perfil Robot Autobalanceado	

Escuela Técnica Superior de Ingeniería del Diseño

Escala: 1:2.5

DISEÑO E IMPLEMENTACIÓN DE UN ROBOT AUTOBALANCEADO

Plano nº: 2

Trabajo Fin de Máster

Máster en ingeniería Mecatrónica

9.3 Código

A continuación se muestra el código o “script” principal implementado en el robot autobalanceado junto a sus comentarios pertinentes.

```c
/**
  ******************************************************************************
  * File Name          : main.c
  * Description        : Main program body
  *
  * Author:                      : Jorge Ferrer Gomez
  * Project:                     : Diseño e implementacion de un robot autobalanceado (TFM)
  ******************************************************************************
  *
  * COPYRIGHT(c) 2017 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_hal.h"
#include "i2c.h"
#include "rtc.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* USER CODE BEGIN Includes */
#include "Funciones_propias.h"
#include <string.h>
#include "accelerometer.h"
#include "gyroscope.h"
#include <math.h>
#include <stdlib.h>

/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
uint16_t in_captura3, in_captura4;
#define PI 3.14159265                                   //Numero Pi
#define Ten 0.000001                                    //Periodo muestreo encoder
#define Kred 0.00191                                    //Coeficiente de reducción de velocidad Eje Motor a Eje Rueda

extern RTC_HandleTypeDef hrtc;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
int MaxPWM = 1000, MinPWM = 0, offsetPWM = 385;     //Limites del PWM
char transform[32]; //vector de caracteres usado para mostrar valores a lo largo del programa
float gyroData[3];
char buffer[5];
```

```
69  char memory[5];
70  int16_t accelBuf[3];
71  float memX = 0;
72  float memY = 0;
73  float angleX = 0;
74  float angleY = 0;
75  float angleXant, angleXfin;             //Angulo anterior y del filtro IIR
76  float X, Y;
77  int angflag;
78
79  float offsetIMU = 1.68; //Offest de las medidas tomadas por el giroscopio y
    acelerometro
80
81  int prueba=0;               //Prueba
82  char help[5];
83  double pitch = 0;
84  uint8_t reverselegIK = 0;
85  uint8_t rotating = 0;
86  float rAngleX, rAngleY;
87  float roll, X, Y;
88  float kp2 =1;           //cte proporcional de ajuste velocidad entre ruedas
89  //Variables RTC
90  uint8_t seg;
91  uint32_t subseg;
92  uint32_t tc;
93
94  //Variables PD
95  int angflag = 0;                    //Para inicializar angulo anterior y actual en
    la misma posicion 0-Inicio, 1-resto de calculos
96  float AngActual, AngAnterior;
97
98  //K control 1 - Ang
99  float kp = 145.5;                   //Constante proporcional
100 float kd = 6.536;                   //Constante derivativa
101
102 //K control 2 - Desp
103 float kpp = 56.15;                  //Constante proporcional
104 float kdd = 0.1;                    //Constante derivativa
105
106 float AngCamb = 90;                                         //Angulo en el que se
    cambia de control para tipo por secciones (Ensayos)
107 float kc = 1000;                                           //Constante para
    correccion de impulso generado por cambio de sentido y holgura en reductora (Ensayos)
108
109 float P, D;
110 float Ppos, Ipos, Dpos, E;
    //Proporcional, Integral, Derivativo, Error =ref - medida
111 static float salidaPD, salidaPDang, salidaPDpos,salidaPDposAnt, Iant = 0, Eant = 0;
112 static float refpos = 0;
113 float Isum = 0;
114 int MaxPD = 615, MinPD = -615;  //MaxPWM - offsetPWM
115
116 /* USER CODE END PV */
117
118 /* Private function prototypes -----------------------------------------------*/
119 void SystemClock_Config(void);
120 void Error_Handler(void);
121
122 /* USER CODE BEGIN PFP */
123 /* Private function prototypes -----------------------------------------------*/
124 void IMUread(void);
125 static void RTC_Segundero(void);
126 float PD(float dt, float entrada, float kp, float kd);
127 float PDpos(float dt, float refp, float entrada, float kpp, float kdd)      ;
128 float PID(float dt, float entrada, float kp, float ki, float kd);
129 float limite(float a, float amax, float amin);
130
131 /* USER CODE END PFP */
132
133 /* USER CODE BEGIN 0 */
134
```

```
135  void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef
     *htim)                                    //Funcion captura e interrupcion velocidad
     desde encoder
136      {
137          if
         (htim->Instance==TIM10)
                                                   //Timer 10, encoder motor 1
138              {
139                  in_captura3 = __HAL_TIM_GetCompare(&htim10,
                     TIM_CHANNEL_1);                        //Captura valor señal
                     encoder motor 1 (Numero de pulsos que indican la frecuecia)
140                  __HAL_TIM_SetCounter(&htim10,
                     0);
                                //Reinicia el counter despues de la interrupcion de
                     entrada de captura ocurra
141              }
142
143          if
         (htim->Instance==TIM11)
                                                 //Timer 11, encoder motor 2
144              {
145                  in_captura4 = __HAL_TIM_GetCompare(&htim11,
                     TIM_CHANNEL_1);                        //Captura valor señal
                     encoder motor 2 (Numero de pulsos que indican la frecuecia)
146                  __HAL_TIM_SetCounter(&htim11,
                     0);
                                //Reinicia el counter despues de la interrupcion de
                     entrada de captura ocurra
147              }
148  }
149
150  /* USER CODE END 0 */
151
152  int main(void)
153  {
154
155    /* USER CODE BEGIN 1 */
156
157          //Variables
158          static int flag=5,
             sflag=1;
                            //Flags
159          static int  p=0, p2, p2ant=0, sentido, sentidoAnt, sentidoEncoder;
             //Variables del motor DC
160          static int vuelta=0;
             //Variable numero de vuelta (Para test sensor HALL)
161          static int nextvuelta=1, cuenta1v_ant=0,cuenta2v_ant=0;
             //Para valor de pulsos por vuelta
162
163          int d = 100;
164          //Variables transmision para el encoder, Timer 1 y 2 (Posicion)
165          static int32_t cuenta1=0, cuenta2=0, cuenta11,
             cuenta22;
166          static int32_t c1ant = 0, c2ant=0, c1ant2=0, c2ant2 = 0;
167          //Variables calculo
168          static double gradosr1, gradosr2, gradosrm, radr1, radr2,f1, f2, vel1, vel2,
             velr1, velr2, velmed, comp, difpor, distr1, distr2;
169
170          //Variable mensaje TX
171          char c_cuenta1[128];
172
173          //Variables RTC
174          int t1=0, t2=0;
175          float krtc;                                //Constante de ajuste del Real
             Time Clock
176
177          //Calculo krtc
178          krtc=1000/255*1.33;                        //El RTC da 255 pulsos por
             segundo. Ajuste de escala para tener valores reales, en ms. 1.25 factor de
             correccion offset RTC
179
```

```c
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();
  MX_USART1_UART_Init();
  MX_TIM10_Init();
  MX_TIM11_Init();
  MX_I2C1_Init();
  MX_RTC_Init();

  /* USER CODE BEGIN 2 */

    BSP_ACCELERO_Init();
    BSP_GYRO_Init();


    //Timers para las cuentas de vueltas
    HAL_TIM_Base_Start(&htim1);
    HAL_TIM_Base_Start(&htim2);

    //Mensaje programa del encoder
    Print_USART1("TRANSMITIENDO DESDE MICRO: \n");

    //Señales PWM
    HAL_TIM_PWM_Start (&htim3, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start (&htim3, TIM_CHANNEL_2);

    //Timer para contar pulsos velocidad desde encoder (TIM10 y TIM11)
        //Modo captura de entrada con interrupcion
    HAL_TIM_IC_Start_IT(&htim10,TIM_CHANNEL_1);
    HAL_TIM_IC_Start_IT(&htim11,TIM_CHANNEL_1);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */


                    /*Programa Encoder*/

                          if((cuenta11 > 967 && cuenta11 < 2000) || (cuenta22 >
                          967 && cuenta22 < 2000)){         //1vuelta de rueda =
                          967 pulsos encoder
                        sflag++;
                        cuenta11=0;
                        cuenta22=0;

                              if(sentido ==
                              1)vuelta++;
                                                        //incremento en 1 para
                              saber en que vuelta se esta
                              if(sentido == 0)vuelta--;
                    }


```

```
246                     switch(sflag){
247                     case 1:
248                         if(nextvuelta==1){
249                                     cuenta1v_ant=cuenta1;
250                                     cuenta2v_ant=cuenta2;
251                                         nextvuelta=0;
252                             }
253
254                         //Encoder, cuenta TIM1 y TIM2,
255                             cuenta1 =
                                __HAL_TIM_GetCounter(&htim1);
                                            //Cuenta motor 1 (Timer 1, Ch1&2)
256                             cuenta2 =
                                __HAL_TIM_GetCounter(&htim2);
                                            //Cuenta motor 2 (Timer 2, Ch1&2)
257
258
259                             if (sentido==1){
260
261                                 cuenta11 = cuenta1 - cuenta1v_ant;
262                                 cuenta22 = cuenta2 - cuenta2v_ant;
263
264                                 //En caso de llegar a valor maximo del
                                    Autoreload y se obtienen valores negativos
                                    respecto a la
                                    referencia
265                                     if(cuenta1 - cuenta1v_ant < 0){
266                                             cuenta11 = cuenta1 + 10000 -
                                                cuenta1v_ant;                   //Si
                                                el Timer 1 se resetea porque llega
                                                al maximo valor del autoreload
                                                (10000) obtiene el valor real de
                                                pulsos en esa vuelta
267                                         }
268
269                                         if(cuenta2 - cuenta2v_ant < 0 ){
270                                             cuenta22 = cuenta2 + 10000 -
                                                cuenta2v_ant;                   //Si
                                                el Timer 2 se resetea porque llega
                                                al maximo valor del autoreload
                                                (10000), obtiene el valor real de
                                                pulsos en esa vuelta
271                                         }
272
273                             }
274
275                             if (sentido==0){
276
277                                 cuenta11 = cuenta1v_ant - cuenta1;
278                                 cuenta22 = cuenta2v_ant - cuenta2;
279
280                                 //En caso de llegar a valor minimo del
                                    Autoreload y se obtienen valores negativos
                                    respecto a la referencia
281                                     if(cuenta1v_ant - cuenta1 < 0){
282                                         cuenta11 = cuenta1v_ant + 10000 -
                                            cuenta1;                        //Si el
                                            Timer 1 se resetea porque llega al
                                            minimo valor del autoreload (0 de
                                            10000), obtiene el valor real de pulsos
                                            en esa vuelta
283                                         }
284
285                                         if(cuenta2v_ant - cuenta2 < 0){
286                                             cuenta22 = cuenta2v_ant + 10000 -
                                                cuenta2;                    //Si el Timer
                                                2 se resetea porque llega al minimo
                                                valor del autoreload (0 de 10000),
                                                obtiene el valor real de pulsos en esa
                                                vuelta
287                                         }
```

```
288
289
290                         }
291                         //Obtencion de datos de posicion rotacion eje ruedas
292
293                         if (sentido==1){
294                         gradosr1=360.0*cuenta11/977;
                            //Grados eje rueda 1
295                         radr1=
                            gradosr1*PI/180.0;
                            //Radianes eje rueda 1
296
297                         gradosr2=360.0*cuenta22/977;
                            //Grados eje rueda 2
298                         radr2=
                            gradosr2*PI/180.0;
                            //Radianes eje rueda 2
299                         }
300
301                         if (sentido==0){
302                         gradosr1= - 360.0*cuenta11/977;
                            //Grados eje rueda 1
303                         radr1= - gradosr1*PI/180.0;
                            //Radianes eje rueda 1
304
305                         gradosr2= - 360.0*cuenta22/977;
                            //Grados eje rueda 2
306                         radr2= - gradosr2*PI/180.0;
                            //Radianes eje rueda 2
307                         }
308
309                     break;
310
311                     case 2:
312
                            sflag=1;
                                                    //Reseteo del sflag
313                         nextvuelta=1;
314
315                     break;
316                 }   //Fin del switch
317
318                 //Obtencion distancia recorrida segun las vueltas de rueda
319
320                 distr1 = 2 * PI * 5* 0.01 * ((gradosr1/360) + vuelta); //Radio
                    ruedas 3 = 5 cm
321
322                 //Obtencion velocidades de ejes
323
324                 f1 = 1.0/(in_captura3*Ten);                 //Frecuencia eje
                    motor 1 = vel rps (1 par de polos)
325                 f2 = 1.0/(in_captura4*Ten);                 //Frecuencia eje
                    motor 2 = vel rps (1 par de polos)
326                 vel1=f1*60.0;
                    //Velocidad rpm eje motor 1,    (n=f*60/p) siendo p pares de
                    polos (solo 1)
327                 vel2=f2*60.0;
                    //Velocidad rpm eje motor 2, igual que antes
328
329                 velr1 = vel1*Kred;                                  //Velocidad
                    rpm eje rueda 1
330                 velr2 = vel2*Kred;                                  //Velocidad
                    rpm eje rueda 2
331
332                 /*Fin programa Encoder*/
333
334                 //Comandos para mover los motores segun el valor de p introducido
335
336                 __HAL_TIM_SetCompare(&htim3,
                    TIM_CHANNEL_1,p);
                        //Mueve motor 1, Timer 3 Ch1
```

```
337                     __HAL_TIM_SetCompare(&htim3,
                        TIM_CHANNEL_2,p2);
                            //Mueve motor 2, Timer 3 Ch2
338
339                         if(angleX<-50 || angleX >50){                    //Para
                            motores si se cae el robot
340                         __HAL_TIM_SetCompare(&htim3,
                            TIM_CHANNEL_1,0);
                                //Mueve motor 1, Timer 3 Ch1
341                     __HAL_TIM_SetCompare(&htim3,
                        TIM_CHANNEL_2,0);
                            //Mueve motor 2, Timer 3 Ch2
342                         }
343
344
345                     //Inicializo la captura de informacion del RTC
346
                        t2=t1;
                                                //Guardo t1 anterior como t2
347
348                     RTC_Segundero();
349                     t1=subseg*krtc;
350
351                     if(t1<t2)tc=t2-t1;
352
                        if(t1>t2)tc=t2+255*krtc-t1;
                                //En caso llegue a final de lista de segundo y reinicie
                        contador
353
354                     //Lectura del IMU: giroscopio y acelerometro
355                     IMUread();
356
357
358                     velmed = (velr1 + velr2)/2;     //Velocidad media entre los dos
                        motores para visualizacion
359
360                     /*Programa PD*/
361                     //Se ejecuta el control de estabilidad
362
363                         if(angflag == 0) {
364
365                             angleXfin = angleX+offsetIMU;
366                             angflag++;
367                         }
368
369                         else angleXfin = 0.5*(angleX+offsetIMU) +
                            0.5*angleXfin;            //Filtro IIR PB
370
371
372                         salidaPDang = PD(tc,angleXfin, kp,
                            kd);                          //Control segun Angulo
373
374                         salidaPDpos = PDpos(tc, refpos, distr1, kpp, kdd);
                            //Control segun distancia desplazada
375
376                         salidaPD = salidaPDang + salidaPDpos;
377
378                         angleXant = angleXfin;
379
380
381                     //Cambio de sentido segun caida
382
383                         if(0 > salidaPD ){
384                             sentido = 0;
385                         }
386                         if(0 < salidaPD ){
387                             sentido = 1;
388                         }
389
390                     //Se introduce los valores del controlador como valor de la PWM
391                         p = fabs(salidaPD) +
```

```
                        offsetPWM;
                            //Valor absoluto de la salida del PD
392                     p = limite(round(p),MaxPWM,
                        MinPWM);                                        //Se
                        redondea al entero mas cercano al valor calculao de p para
                        mayor precision
393
394                 /*Fin control PD*/
395
396                 //Control ajuste de velocidad de la rueda Motor 2
397                 comp = vel1/vel2*100;
398                 difpor = 100 -
                    vel1/vel2*100;
                    //Diferencia velocidades en ejes de los motores
399
400                 if(difpor>-2 && difpor<2){
401                     p2 = (p -
                        p*difpor/100)*kp2;
                        //Valor nuevo para que la velocidad del motor 2 sea igual a
                        la del motor 1;
402                     p2 = limite (round(p2),MaxPWM,MinPWM);
                        //Acotamos (0-1000 PWM) y volvemos entero el valor para el
                        motor 2
403                 p2ant=p2;
404                 }
405                 else
                    if(p-p2<0)p2=limite(round(p+12),MaxPWM,MinPWM);
                        //filtro para evitar picos
406                 else
                    if(p-p2>0)p2=limite(round(p-12),MaxPWM,MinPWM);
                        //filtro para evitar picos
407
408
409
410                     //Seleccion de sentido e los motoreS
411                     if(sentidoAnt!=sentido){
412                             if(sentido==0){
413
                                    HAL_GPIO_WritePin(sentido1_GPIO_Port,sentido1_
                                    Pin,GPIO_PIN_SET);            //Pone motores
                                    en Sentido de giro +
414
                                    HAL_GPIO_WritePin(sentido2_GPIO_Port,sentido2_
                                    Pin,GPIO_PIN_SET);
415 //                              p = p -50;
416                                 sentidoAnt = 0;
417                             }
418                             if(sentido==1){
419
                                    HAL_GPIO_WritePin(sentido1_GPIO_Port,sentido1_
                                    Pin,GPIO_PIN_RESET);      //Pone motores en
                                    Sentido de giro -
420
                                    HAL_GPIO_WritePin(sentido2_GPIO_Port,sentido2_
                                    Pin,GPIO_PIN_RESET);
421 //                              p = p -50;
422                                 sentidoAnt = 1;
423
424                             }
425                     }
426
427                 /*Envio de mensajes a pantalla por radio - Escoger segun prueba*/
428
429 //                      sprintf(transform, "angx %d, velmed %d, sentido %d, tc %d, P
    %.4f, I %.4f, D %.4f, salidaPD %.4f, p %d, %d \n",(int) angleX, (int)velmed,
    sentido, tc,P ,I , D, salidaPD, p, p2);
430 //                      Print_USART1(transform);
431 //                      sprintf(transform, "AngActual %f, AngAnterior %f \n \n",
    AngActual, AngAnterior);
432 //                      Print_USART1(transform);
433
```

```
434                         //PID
435   //                        sprintf(transform, "  %.5f,  %.5f,  %.5f,  %d,  %.5f,
      %.5f,  %f,  %f,   %f \n", salidaPD, salidaPDang, salidaPDpos, p, AngActual, distr1,
      kp, kd, kpp);
436   //                        Print_USART1(transform);
437
438   //                        sprintf(transform, "  %.5f,  %.5f,  %.5f,  %d,  %.5f,
      %.5f \n", salidaPD, salidaPDang, salidaPDpos, p, AngActual, distr1);
439   //                        Print_USART1(transform);
440
441   //                        sprintf(transform, "%.5f,  %.5f, %.5f, %.5f, %d,
      %.5f   \n", Ppos, Dpos, salidaPDpos, distr1, p, AngActual);
442   //                        Print_USART1(transform);
443
444   //                        sprintf(transform, "%f, %f, %f,P: %f,Ppos: %f, %d, %f
      \n", P, I, D, salidaPDang, Ppos, p, AngActual);
445   //                        Print_USART1(transform
446
447                           sprintf(transform, "%f, %f, %f, %f \n", P,  D,
                              salidaPDang,  AngActual);
448                           Print_USART1(transform);
449
450   //                        sprintf(transform, "%d,  %f,  %f \n", p, D, AngActual);
451   //                        Print_USART1(transform);
452
453                         //Velocidad ruedas
454   //                        sprintf(transform, "%f, %d, %d, %f, %f \n",difpor ,p ,
      p2, velr1, velr2);
455   //                        Print_USART1(transform);
456
457                         //Sentido y angulo
458   //                        sprintf(transform, "%d, %f \n", sentido, AngActual);
459   //                        Print_USART1(transform);
460
461                         //Control de distancia recorrida
462   //                        sprintf(transform, "%d,  %d, %f, %d, %f \n", cuenta1,
      cuenta11, gradosr1, vuelta, distr1 );
463   //                        Print_USART1(transform);
464
465   //                        sprintf(transform, "%d,  %f, %f \n", vuelta, gradosr1,
      AngActual );
466   //                        Print_USART1(transform);
467
468                         /*Fin envio de mensajes*/
469
470                         AngAnterior = AngActual;
471     } //Fin del while(1)
472     /* USER CODE END 3 */
473
474   }
475
476   /** System Clock Configuration
477   */
478   void SystemClock_Config(void)
479   {
480
481     RCC_OscInitTypeDef RCC_OscInitStruct;
482     RCC_ClkInitTypeDef RCC_ClkInitStruct;
483     RCC_PeriphCLKInitTypeDef PeriphClkInitStruct;
484
485       /**Configure the main internal regulator output voltage
486       */
487     __HAL_RCC_PWR_CLK_ENABLE();
488
489     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
490
491       /**Initializes the CPU, AHB and APB busses clocks
492       */
493     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI;
494     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
495     RCC_OscInitStruct.HSICalibrationValue = 16;
```

```
496     RCC_OscInitStruct.LSIState = RCC_LSI_ON;
497     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
498     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
499     RCC_OscInitStruct.PLL.PLLM = 16;
500     RCC_OscInitStruct.PLL.PLLN = 336;
501     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
502     RCC_OscInitStruct.PLL.PLLQ = 4;
503     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
504     {
505       Error_Handler();
506     }
507
508       /**Initializes the CPU, AHB and APB busses clocks
509       */
510     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
511                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
512     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
513     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
514     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
515     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
516
517     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
518     {
519       Error_Handler();
520     }
521
522     PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_RTC;
523     PeriphClkInitStruct.RTCClockSelection = RCC_RTCCLKSOURCE_LSI;
524     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
525     {
526       Error_Handler();
527     }
528
529       /**Configure the Systick interrupt time
530       */
531     HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
532
533       /**Configure the Systick
534       */
535     HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
536
537     /* SysTick_IRQn interrupt configuration */
538     HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
539   }
540
541   /* USER CODE BEGIN 4 */
542
543   static void RTC_Segundero(void)                         //Real Time Clock, devuelve
      segundos y subsegundos (3...4ms)
544   {
545     //RTC_DateTypeDef sdatestructureget;
546
547       RTC_TimeTypeDef sTime;
548     RTC_DateTypeDef sDate;
549
550     /* Get the RTC current Time */
551     HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
552     /* Get the RTC current Date */
553     HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);
554     /* Display time Format: hh:mm:ss */
555
556       seg = sTime.Seconds;
557       subseg = sTime.SubSeconds;
558
559   }
560
561
562   void IMUread(void)                                    //Funcion lectura y calculo del
      IMU
563   {
564       float roll, pitch, X, Y;
```

```
565
566
567         BSP_ACCELERO_GetXYZ((int16_t *) accelBuf);
568
569         BSP_GYRO_GetXYZ((float*) gyroData);
570
571         if (accelBuf[2] > 0)
572         {
573             if (rotating == 0)
574             {
575
576                 roll = atan2((accelBuf[1] - 480), (accelBuf[2] - 960))* 180 / PI;
577                 pitch=atan2((accelBuf[0] - 448), (accelBuf[2] - 960))* 180 / PI;
578
579
580                 X = (gyroData[1] + 1820 + memX) * tc/ 2000000;
581                 Y = (gyroData[0] + 2757.986 + memY) * tc / 2000000;
582
583
584                 memX = gyroData[1] + 1820;
585                 memY = gyroData[0] + 2757.986;
586
587
588                 angleX = 0.98 * (angleX + X) + 0.02 * roll;
589                 angleY = 0.98 * (angleY + Y) + 0.02 * pitch;
590
591             }
592         }
593     }   //Fin funcion
594
595     float limite(float a, float amax, float amin){      //Funcion para poner limites y
    acotar variables
596         if (a > amax)a = amax;
597         if (a < amin)a = amin;
598
599         return(a);
600     }
601
602     float PD(float dt, float entrada, float kp, float kd)           //Funcion control PD
    para estabilidad
603         {
604             float salida;
605             dt=8*0.001; //mseg a seg
606
607             AngActual = round(entrada*50)/50;
608
609             if(angflag == 0){                       //Inicio de programa cuando aun no
            tenemos un angulo anterior
610                 AngAnterior = AngActual;
611                 angflag = 1;
612             }
613
614             P = AngActual * kp;
            //Proporcional
615             P = limite(P,MaxPD,MinPD);
616
617
618             D = ((AngActual - AngAnterior) / dt) * kd;  //Derivada
619             D = limite(D,MaxPD,MinPD);
620
621             salida = P + D;
622             salida = limite(salida,MaxPD,MinPD);
623
624             return (salida);
625
626     }//Fin de la funcion PD
627
628     float PDpos(float dt, float refp, float entrada, float kpp, float kdd)
    //Funcion control PD con referencia de desplazamiento posicion
629         {
630             float salidaPos;
```

```
631           dt=8*0.001; //mseg a seg
632 
633 
634           E = refp - entrada;                       //Error
635 
636           Ppos = E * kpp;
637           Ppos = limite(Ppos,MaxPD,MinPD);
638 
639           Dpos = ((E - Eant) / dt) * kdd;
640           Dpos = limite(Dpos,MaxPD,MinPD);
641 
642           salidaPos = Ppos + Dpos;
643           salidaPos = limite(salidaPos,MaxPD,MinPD);
644 
645           Eant = E;
646           return (salidaPos);
647 
648   }//Fin de la funcion PD posicion
649 
650 
651 
652   /* USER CODE END 4 */
653 
654   /**
655     * @brief  This function is executed in case of error occurrence.
656     * @param  None
657     * @retval None
658     */
659   void Error_Handler(void)
660   {
661     /* USER CODE BEGIN Error_Handler */
662     /* User can add his own implementation to report the HAL error return state */
663     while(1)
664     {
665     }
666     /* USER CODE END Error_Handler */
667   }
668 
669   #ifdef USE_FULL_ASSERT
670 
671   /**
672      * @brief Reports the name of the source file and the source line number
673      * where the assert_param error has occurred.
674      * @param file: pointer to the source file name
675      * @param line: assert_param error line source number
676      * @retval None
677      */
678   void assert_failed(uint8_t* file, uint32_t line)
679   {
680     /* USER CODE BEGIN 6 */
681     /* User can add his own implementation to report the file name and line number,
682       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
683     /* USER CODE END 6 */
684 
685   }
686 
687   #endif
688 
689   /**
690     * @}
691     */
692 
693   /**
694     * @}
695   */
696 
697   /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
698 
```