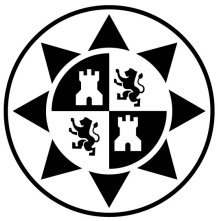

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Electrónica, Tecnología de Computadoras
y Proyectos

DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE CONTROL DE UN VEHÍCULO AÉREO NO TRIPULADO (UAV)

Trabajo Fin de Grado en Ingeniería Electrónica
Industrial y Automática

Autor: José Alberto Zapata del Baño
Director: Manuel Sánchez Alonso
Curso Académico: 2017-2018

Resumen

En este trabajo vamos a presentar el diseño y la implementación del sistema de control de un Vehículo Aéreo no Tripulado (UAV). Concretamente, se pensó en realizar el diseño de un UAV de cuatro motores (cuadricóptero), con la finalidad de mejorar los sistemas de control existentes basados en Arduino. El trabajo se va a descomponer en varias partes, a través de las cuales podremos profundizar en aspectos relevantes como pueden ser: su descripción, historia (relativamente reciente), y aplicaciones. Como es obvio trataremos el sistema de control para realizar un control de estabilidad, que posteriormente, con algunas modificaciones, se puede extender al manejo operativo del UAV.

Por motivos de seguridad, tiempo y recursos, se ha optado por implementar sólo la mitad de un cuadricóptero. La decisión no ha sido por comodidad, sino estrictamente por motivos de operatividad. Se implementan dos motores (de los cuatro) en un balancín que proporciona su seguridad de operación, y se le aplican algoritmos de control de estabilidad en base a las lecturas de los periféricos asociados.

Se explicará él porqué se escoge un microcontrolador concreto, en nuestro caso el PIC16F876A. Esta elección proporciona serias ventajas sobre otros modelos y fabricantes, siendo lo más destacable el bajo consumo, que realmente está más condicionado al conjunto de dispositivos periféricos complementarios, necesarios para la captura de información y la actuación sobre los motores del dispositivo UAV.

Por último, se expondrá una serie de resultados de las pruebas, y las conclusiones a las que se ha llegado en este TFE, así como las vías futuras de desarrollo y mejora de este proyecto.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Descripción

El proyecto consistirá en diseñar y construir un prototipo para simular el control de vuelo de un UAV tipo cuadricóptero. Pese a que no vamos a realizar un cuadricóptero completo debido al presupuesto y los problemas de seguridad que podría conllevar, modificar el código de este prototipo para conseguir el control de vuelo de un cuadricóptero completo es relativamente sencillo de implementar.

Este prototipo consta de dos motores en dos extremos de un balancín, el microcontrolador calculará y establecerá la potencia de ambos motores, con el objetivo de que el sistema se estabilice en una inclinación deseada. El microcontrolador necesitará obtener medidas a través de un sensor y enviará a un ordenador los datos obtenidos. Los detalles serán explicados en los posteriores temas.

1.2. Objetivos

El objetivo principal de este trabajo es realizar una implementación de sensores y actuadores gestionados a través de un microcontrolador PIC para realizar el control en el vuelo de un UAV. Este objetivo da pie a adquirir una serie de competencias muy interesantes, como programar microcontroladores PIC en un lenguaje de alto nivel, gestión de errores, comunicarlos a través de la UART y del protocolo I2C, etc.

Otro objetivo es realizar un UAV desde un enfoque distinto, ya que existen diversos proyectos de control de UAV basados en Arduino, pero el Arduino tiene unas prestaciones limitadas, además que la placa es relativamente grande para un UAV, mientras

que si lo enfocamos a realizarlo con un PIC la programación es distinta, ya que tenemos control sobre los registros. Además tenemos una gran variedad de modelos para elegir el más afín a nuestras necesidades, y si queremos aumentar la potencia del PIC o incluir nuevas funciones podemos portar el programa a un PIC más potente.

No podemos olvidar las competencias transversales que suponen un proyecto práctico como éste, no solo tendremos que realizar el proyecto si no que tendremos que tomar decisiones para diseñarlo, además a lo largo del proyecto tendremos que enfrentarnos a multitud de fallos y problemas que tendremos que solucionar, este proceso es parte del aprendizaje y aportará una experiencia de trabajo muy valiosa.

1.3. Motivación

El motivo principal para realizar este trabajo es la curiosidad, los UAVs son sistemas que están ganando popularidad, y claramente involucran muchas tecnologías (baterías, microcontroladores, motores, materiales...), su auge en gran medida es debido a los avances en dichas tecnologías. Siempre he sentido curiosidad por cómo funcionan los sistemas tecnológicos, en especial los sistemas informáticos y los automóviles, por lo que desde que los UAVs han empezado a popularizarse también he sentido curiosidad por conocer su funcionamiento.

Por otro lado en el Grado en Ingeniería Electrónica Industrial y Automática se estudia muchas de las tecnologías detrás de los UAVs, como electrotecnia, teoría de control, ingeniería de materiales, microcontroladores, etc., así que otra razón para realizar este proyecto es que dispone de un estupendo marco, ya que muchas asignaturas estudiadas durante el grado convergen en este campo.

Por último, los sistemas comerciales que nos rodean sean más complejos y difíciles de abordar por una sola persona, por lo que otra ventaja de este proyecto es que podemos aislar las partes que nos interesan, es decir, de toda la complejidad que puede tener el código de un UAV comercial, nosotros podemos centrarnos en la parte del control, consiguiendo que el proyecto sea más accesible.

1.4. Definición

Antes de profundizar el proyecto vamos a ver un poco de contexto. Un vehículo aéreo no tripulado, UAV por sus iniciales en inglés (Unmanned Aerial Vehicle) y comúnmente conocido como dron, es una aeronave sin piloto humano a bordo. Su diseño tiene una amplia variedad de formas, tamaños, configuraciones y características. El vuelo de UAVs puede tener diferentes grados de autonomía, ya sea poca autonomía, siendo controlado de forma remota por un humano o de forma autónoma por computadoras a bordo, con un programa precargado.

La historia del desarrollo de los UAV siempre ha estado ligada a los usos militares. Su desarrollo comenzó a principios del siglo XX, y originalmente se centraron en proporcionar objetivos de práctica para el entrenamiento del personal militar. El desarrollo de UAVs continuó durante la Primera Guerra Mundial. El primer vehículo a control remoto a escala fue desarrollado en 1935 y continuaron su uso durante la Segunda Guerra Mundial; se usaron tanto para entrenar artilleros antiaéreos como para volar misiones de ataque. Los motores a reacción entraron en servicio después de la Segunda Guerra Mundial en vehículos como el australiano GAF Jindivik.

Figura 1.1: UAV GAF Jindivik controlado remotamente por radiofrecuencia.

Durante los siguientes años EEUU empezó a utilizar UAV en guerras en las que estaba involucrado, provocando el desarrollo de estos, y a partir de 1980 comenzó la miniaturización de estos, cada vez acercándose más a los UAV como los conocemos hoy en día. Y con los avances tecnológicos que se han producido en las últimas décadas y la investigación realizada por numerosos países hemos llegado a los UAV como los conocemos hoy en día.

Los UAV han ganado bastante popularidad en los últimos años gracias al desarrollo de las baterías y la bajada de precio de sus componentes que permiten que cada vez más personas puedan disponer de uno, se pueden utilizar simplemente para volarlos como entretenimiento o pueden tener otros fines como control de incendios forestales, cartografía, etc. También siguen siendo útiles en usos militares.

1.5. Tipos y aplicaciones

Existe una gran variedad de UAV, por lo que tenemos diferentes formas de clasificarlos. Nosotros los clasificaremos por morfología, tamaño y aplicaciones.

1.5.1. Morfología

1. **Multirrotores**

   Los multirrotores es la morfología más extendida actualmente y más representativa de los drones. Proporciona una gran versatilidad y eficacia en las operaciones por su simpleza a la hora de ser pilotados. Sus motores se encuentran equidistantes del centro, lo que le hace muy estable, y variando adecuadamente la velocidad de giro cada motor conseguimos gran maniobrabilidad.

   Según la cantidad de motores los clasificamos en tricópteros (3 motores), cuadricópteros (4 motores), hexacópteros (6 motores) y octocópteros (8 motores). Y según la configuración de los brazos los clasificamos en "Y", "X", "H", "plus" (+). Por otra parte, también existen los multirrotores coaxiales, es decir, dos motores por brazo. Esto proporciona un ahorro en el peso del aparato por contar con la mitad de brazos, pero por el contrario se resta eficiencia aerodinámica.

   A la hora de elegir un multirrotor u otro, debemos de tener en cuenta que cuantos más brazos tendremos más estabilidad y más seguridad, mientras que cuantos más motores tengamos más propulsión y consumo. Los multirrotores tienen la autonomía como una desventaja importante, ya que de media no suele superar los 15 minutos de vuelo, y esto puede suponer un impedimento para muchas operaciones.

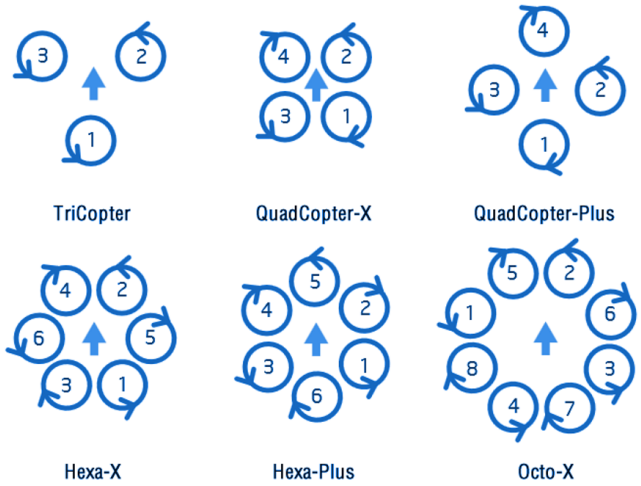

Figura 1.2: Diferentes configuraciones de multirrotores.

2. **Helicópteros**

   Se componen de un rotor en la parte superior y otro en la cola para contrarrestar el par rotor del motor superior que lo haría al cuerpo girar. Poseen una gran autonomía. El helicóptero es mucho más eficiente aerodinámicamente que un multirrotor, ya que el helicóptero funciona a revoluciones fijas, y lo que hace que un helicóptero ascienda o descienda es la variación en el ángulo de ataque que se da a las palas del rotor, mientras que el multirrotor varía las revoluciones del motor para mantenerse estable.

   Otra ventaja muy destacable es que si lo dotamos de un motor de explosión, podemos permanecer en el aire al rededor de 1 hora, lo que es perfecto para operaciones como la fotogrametría. Sin embargo, los helicópteros son bastante complejos a nivel mecánico, también es bastante complicado a la hora de ser pilotado.

3. **Ala fija**

   El ala fija es el claro ganador en lo que a autonomía se refiere, puede permanecer en el aire varias horas. Es la plataforma perfecta para trabajos que abarquen una gran extensión de terreno. Por otra parte, es el más eficiente aerodinámicamente hablando, ya que con la configuración adecuada, puede permanecer bastante tiempo sin necesidad de utilizar el motor gracias al planeo. Por otra parte, el hecho de poder planear hace que sea una plataforma mucho más segura, ya que en un supuesto fallo de motor puede planear hasta llegar al punto de aterrizaje.

Sin embargo, el ala fija, está preparado para unos fines muy específicos, lo que le resta versatilidad a la hora de ser utilizado. Su principal desventaja es el tema del aterrizaje y el despegue., que no puede realizarlas de forma vertical. El no poder aterrizar y despegar verticalmente nos obliga a tener que acotar un extensión bastante grande de terreno (unos 60m), y que ésta sea plana y sin obstáculos. Por otra parte, un dron de ala fija no permite hacer un vuelo estacionario[1], lo que nos impide poder realizar un sinfín de operaciones.

1.5.2. Tamaño

1. **Muy pequeños**

   Se pueden diseñar con un rango de tamaño común que varía desde un par de centímetros a unos de 50 cm de largo.

2. **Pequeños**

   Tienen un tamaño un poco más grande que los micro, lo que significa que superará los 50 cm pero tendrá una dimensión máxima de 2 m. Aquí se encuentran la mayoría de UAV comerciales.

3. **Medianos**

   Estos drones pueden llevar un peso de hasta 200 kg y tener una capacidad de vuelo promedio de 5 a 10 minutos.

4. **Grandes**

   Se puede decir que son comparables con el tamaño de una aeronave y su uso más común es en aplicaciones militares.

1.5.3. Aplicaciones

En cuanto a las aplicaciones de UAV tenemos el uso recreativo y el uso profesional. Actualmente existe un gran mercado en el uso recreativo, entre los que tenemos los de juguete (suelen ser muy sencillos), los destinados a grabación de foto y video, los de

[1] Cuando hablamos de "vuelo estacionario" nos referimos a mantener la aeronave en vuelo fija sobre un punto, sin avanzar en ninguna dirección

carreras y los de acrobacias. Sin embargo los crecientes intereses de los usuarios y empresas han propiciado que los UAV se empiecen a implementar en el mundo profesional. Nombraremos las más significativas, aunque algunas de estas son todavía recientes.

1. **Fotografía aérea**

   Uno de los usos más comunes de los UAV con cámaras avanzadas es capturar contenido multimedia de lugares inaccesibles. Cada vez se está siendo más utilizado por medios de comunicación.

2. **Agricultura**

   Pueden monitorizar terrenos de gran tamaño y así ayudar a los agricultores. También pueden obtener información y rociar fertilizantes, pesticidas y agua para cultivos en los momentos adecuados.

3. **Envío y entrega**

   Ahora se pueden utilizar para aplicaciones de envío y entrega en el comercio online.

4. **Ingeniería**

   las empresas están empezando a utilizar UAV para para supervisar sus proyectos en profundidad, como cables de transmisión, oleoductos e inspecciones de mantenimiento, pero sobretodo destaca el mapeado 3D.

Además de estos usos, tenemos los usos militar, entre los que podemos destacar:

1. **Blanco**

   sirven para simular aviones, u objetos voladores en general, para los sistemas de defensa de tierra o aire.

2. **Búsqueda y rescate**

   Los UAV en estos días están equipados con sensores térmicos para que puedan ubicar la posición de las personas perdidas. También pueden trabajar en la oscuridad y dentro de un terreno desafiante y en lugares remotos.

3. **Reconocimiento**

   Se utilizan para obtener y enviar información militar. Entre estos destacan los MUAV (Micro Unmanned Aerial Vehicle, en español Micro vehículo aéreo no tripulado) con morfología de avión o de helicóptero.

4. **Combate**

   Se utilizan para sustituir a los pilotos de combate, ya que son misiones que suelen ser muy peligrosas para ser realizadas por humanos.

Para determinar la aplicación de un UAV no solo debemos elegir una morfología y tamaño adecuados, lo más importante para esto es complementarlo con los sensores adecuados. Se están implementando muchos nuevos sensores para que la operación de los UAV pueda ser altamente optimizada, en función de la aplicación es conveniente agregar unos y otros, como por ejemplo GPS o sistemas FPV (First Person View, en español Visión en Primera Persona), pero esto va a gusto del consumidor.

1.6. Dinámica

Como ya indicamos anteriormente, nosotros tomaremos como referencia los UAV multirrotores, concretamente los de tipo cuadricóptero. Esta decisión ha sido tomada debido a que son los más comunes y su dinámica es la más intuitiva de comprender.

Un cuadricóptero es un multirrotor que, como su nombre indica, cuenta con cuatro rotores que lo sostienen y lo propulsan, generalmente están colocados en las extremidades de una cruz. Gracias a estos 4 motores se puede conseguir que se desplace con seis grados de libertad.

En un sistema tridimensional, seis grados de libertad se refieren a los movimientos adelante/atrás (forward/back), arriba/abajo (up/down), izquierda/derecha (left/-right), cabecear (pitch), guiñar (yaw), rodar (roll).

Si queremos tener claro como vuela un cuadricóptero lo primero que debemos tener claro es el sentido en el que debe girar cada motor. Podemos deducir que si el sentido en el que giran los motores no está compensado el cuadricóptero no será estable, en este caso al ser cuatro motores la solución es simple: colocamos dos motores que giren

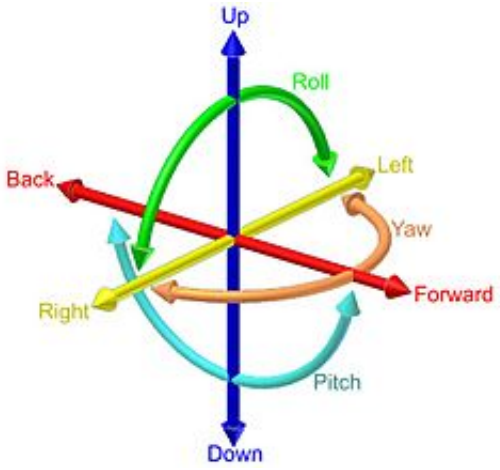

Figura 1.3: Ejes de los seis grados de libertad.

en sentido horario (CW[2]) y otros dos en sentido antihorario (CCW[3]), tal y como se muestra en la imagen.

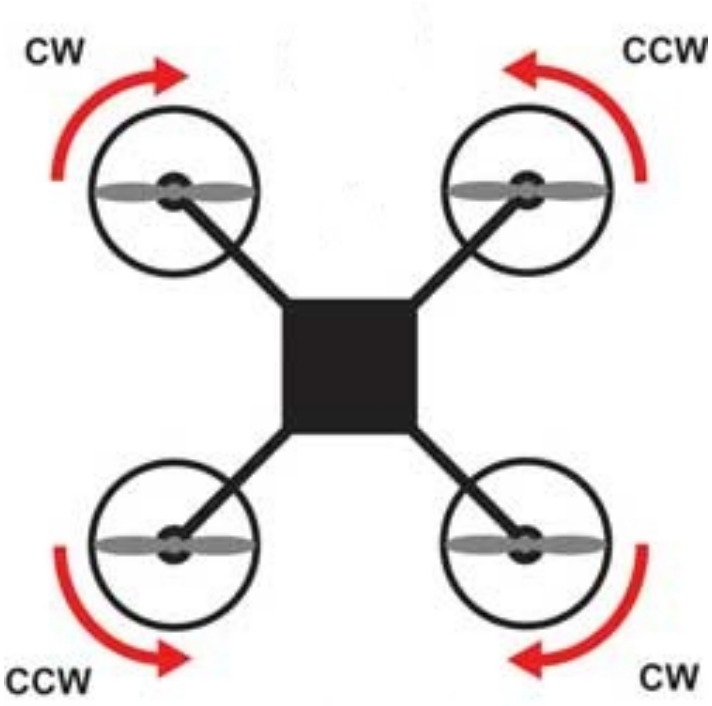

Figura 1.4: Sentido de giro de los motores de un cuadricóptero.

A continuación explicaremos como obtener los distintos movimientos en un cuadricóptero:

[2]CW es la sigla de Clock Wise, en español sentido horario
[3]CCW es la sigla de Counterclock Wise, en español sentido antihorario

1. **Movimiento arriba/abajo**

   Para producir esto los cuatro motores deben girar a la misma velocidad, de manera que producen una fuerza en sentido descendente que venza el peso del cuadricóptero, cuanto mayor sea la velocidad mayor será la fuerza. Para producir su descenso basta con disminuir esta velocidad suavemente para que la fuerza ejercida sea menor y descienda por la fuerza de la gravedad.

2. **Movimiento adelante/atrás**

   Para ello simplemente debemos girar los dos motores delanteros a distinta velocidad de los dos motores traseros. Al producir un desequilibrio de fuerzas obtendremos un movimiento de cabeceo para compensar la componente vertical del empuje de los cuatro motores. La inclinación del cuadricóptero generará una fuerza horizontal que causará el movimiento hacia adelante, si los motores traseros empujan mas que los delanteros, y hacia detrás en caso contrario.

3. **Movimiento izquierda/derecha**

   Se basa en lo mismo que el movimiento adelante/atrás pero agrupando en dos los motores del lado derecho e izquierdo.

4. **Guiñada**

   Anteriormente vimos que para que el cuadricóptero sea estable los motores en sentido horario y antihorario debían estar compensados. Si rompemos este equilibrio lo podemos usar a nuestro favor para lograr que el cuadricóptero gire sobre su eje vertical, logrando un movimiento de guiñada. Si los motores CW tienen mayor velocidad, el cuadricóptero girará en sentido antihorario, si los motores CCW tienen mayor velocidad, girará en sentido horario.

   A la hora de realizar este movimiento tenemos que tener en cuenta que si aumentamos la velocidad de los motores de un sentido, debemos reducir las del otro sentido para que el empuje total de los cuatro motores se mantenga.

Podemos ya observar que si coordinamos cuatro motores controlados de forma independiente podemos obtener que un cuadricóptero se desplace con cuatro grados de libertad.

Capítulo 2

Componentes y prototipo

2.1. Introduccion

En este capítulo vamos a comenzar a definir nuestro proyecto, recordamos que hemos elegido estudiar los multirrotores tipo cuadricóptero, pero como estos se componen de diversas partes, cada una de ellas con diferentes variaciones, puede ser difícil realizar un prototipo sin conocer estas partes. Así que lo primero que haremos es realizar un estudio de los componentes fundamentales que componen un cuadricóptero comercial y explicaremos un poco su funcionamiento, y después adaptaremos estos componentes para nuestro prototipo, para ello realizaremos un balancín con dos motores que posteriormente podrá ser aplicado a un cuadricóptero completo.

2.2. Componentes

Antes de construir nuestro prototipo necesitamos conocer los componentes que necesitamos. Los componentes principales de un cuadricóptero son los motores, las hélices, los variadores de velocidad, la placa controladora, la batería, la emisora, y la estructura. Los cuadricópteros pueden llevar mas elementos (cámaras, detectores de calor, etc.) pero estos son secundarios y en función del cuadricóptero, por lo que no los implementaremos en nuestro proyecto y no serán objeto de estudio, pues son objeto de posteriores mejoras.

2.2.1. Motores

Los motores son sin duda una de las partes de un cuadricóptero, ya que son los únicos que le proporciona el empuje para que pueda levantarse del suelo y desplazarse en la dirección que elijamos. Un cuadricóptero necesita unos motores específicos y bien dimensionados para funcionar de la manera más óptima, así que primero explicaremos un poco sobre los motores para entender las particularidades de los motores de un cuadricóptero.

Un motor es un sistema que convierte una forma determinada de energía en energía mecánica de rotación o par, esta energía mecánica es capaz de realizar un trabajo. Existen diversos tipos, siendo los más comunes los motores térmicos y los eléctricos. Nuestro sistema debe ser ligero y con un bajo consumo, o al menos que sea lo menor posible, por lo que la mejor opción para los UAV son los motores eléctricos.

Un motor eléctrico convierte la energía eléctrica en fuerzas de giro por medio de la acción mutua de los campos magnéticos. Todos los tipos de motores eléctricos se componen por un estátor y un rotor. El estátor es la parte fija, y en los motores es la parte inductora, y el rotor es la parte inducida, que gira respecto al rotor. Esto se consigue al pasar corriente eléctrica por unas bobinas, produciendo un campo magnético que se aprovecha para generar movimiento.

Debido a que los cuadricópteros deben llevar consigo su fuente de energía (la batería), los motores deben ser de corriente continua. Las relaciones entre el campo eléctrico y el campo magnético está recogidas en las Ecuaciones de Maxwell, pero a nosotros la propiedad que nos interesa para entender cómo funcionan los motores es, que si circula una corriente eléctrica por un campo magnético, se crea una fuerza perpendicular a ambas.

Para entender el funcionamiento del motor vamos a simplificar las bobinas e imaginar que es una espira. Al meter la corriente por la espira obtenemos dos conductores enfrentados (por uno entra la corriente y por el otro sale), un lado de la espira subirá y el otro bajará, ya que por un lado la corriente entra y por el otro lado de la espira la corriente sale. Esto produce un par de fuerzas con sentido contrario que provocan que gire la espira, pero no es suficiente para crear un motor, ya que cuando la espira gira media vuelta, el sentido de las fuerzas cambian, por lo que el motor daría media vuelta en un sentido y retrocedería, volvería a avanzar y retroceder sucesivamente.

Para solucionar esto tenemos las escobillas, la base de la espira se desliza a través de las escobillas que están estáticas, como podemos apreciar en la figura 2.1, por lo que el flujo eléctrico en el interior de la espira cambiará de sentido cada media vuelta, por lo que las fuerzas generadas tendrán siempre el mismo sentido y el motor se moverá en un sentido. El inconveniente de las escobillas es que generan un rozamiento que provoca una perdida de rendimiento. En motores grandes las pérdidas producidas por este rozamiento pueden ser despreciables, incluso en motores muy pequeños para cuadricópteros baratos también podemos despreciar el rozamiento, pero para un cuadricóptero no debemos despreciar estas pérdidas y debemos recurrir a un motor más eficiente.

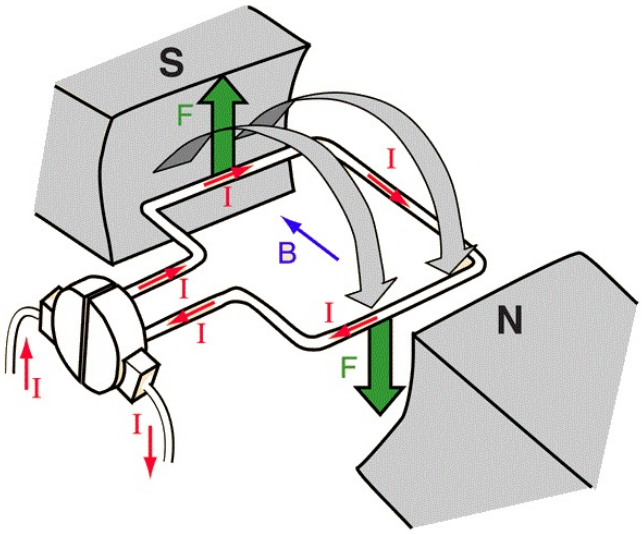

Figura 2.1: Esquema de funcionamiento de un motor con escobillas.

Sin embargo existe un tipo de motor que no posee escobillas y por tanto posee un mayor rendimiento y es ideal para un cuadricóptero, es el llamado BLDC (Brushless DC electric motor, motor eléctrico de corriente continua sin escobillas) o simplemente motor brushless. Las bobinas están dispuestas en el estátor de forma circular, si conseguimos alternar en sentido circular las bobinas que reciben corriente, alternamos las bobinas que generan campos magnéticos y por tanto habrá un campo magnético total que se desplazará de forma circular, y los imanes permanentes del rotor girarán siguiendo la polaridad de las bobinas, produciendo el movimiento del motor. Se suelen alternar en tres fases.

El inconveniente de estos motores es que ese control de las bobinas que reciben corriente se realiza fuera del motor, por lo que no se pueden conectar directamente a

Figura 2.2: Motor sin escobillas.

corriente continua. Sin embargo, los avances en informática y control han hecho que el control de estos motores sea viable para un cuadricóptero, por lo que son la mejor opción y estudiaremos el componente más adelante.

Ahora veremos cómo elegir el motor indicado. Si observamos un motor de este tipo veremos que aparece una numeración que corresponde a sus características esenciales, y que va a definir de una manera clara su comportamiento y capacidades. Las cuatro primeras cifras se leen de dos en dos y se corresponden con la longitud del motor y con su altura. Las cuatro siguientes cifras se corresponden con los Kv, que se refiere a la velocidad constante de un motor (no debe confundirse con "kV", la abreviatura de kilovoltio).

Para calcular Kv se mide el número de revoluciones por minuto a la que gira un motor cuando se aplica 1 V sin ninguna carga conectada al motor. Un motor de bajo Kv tiene más devanados de alambre más delgado, consiguiendo más voltios a menos amperios, producirá un mayor par motor y hará mover una hélice más grande. Un motor de alto Kv tiene menos devanados de cable más grueso que llevan más amperios a menos voltios y hacen girar una hélice más pequeña a altas revoluciones; también aumenta el consumo.

Conocer el número de Kv de un motor es útil para determinar qué motor utilizar para cada tipo. Un cuadricóptero de carreras, por ejemplo, requiere altas rpm para alta velocidad, por lo que se usaría un motor de alto Kv y una hélice de pequeño diámetro. Por otro lado, en un multirotor de elevación pesada se utilizaría un motor de Kv más bajo porque quiere girar una hélice grande a menos rpm y obtener un par alto. Tenemos

otras características como el peso del motor, su calidad de fabricación, etc., pero no suelen existir muchas diferencias entre motores y suelen ir ligadas al factor del precio.

2.2.2. Hélices

Todos los UAV multirrotores utilizan hélices (que no deben confundirse con las hélices de los helicópteros) para lograr la sustentación. Las hélices se conectan a los motores del cuadricóptero y, cuando el motor gira, también lo hacen las hélices. Al igual que en los marcos de cuadricópteros, las hélices se pueden fabricar a partir de una amplia variedad de materiales (incluso existen hélices de madera), y también existen en formas diferentes, siendo bipala y tripala, y en tamaños diferentes, que irá en función de la finalidad del UAV.

La mayoría de los cuadricópteros vienen con hélices de tres palas o dos palas, siendo la configuración más común dos. Las hojas más pequeñas (aquellas con diámetros más pequeños) tienden a ser más fáciles de reducir la velocidad y acelerar, lo que es útil si para realizar vuelos acrobáticos. Las hojas más grandes, o aquellas con diámetros más grandes, son más adecuadas para vuelos más estables, ya que es más difícil acelerar o ralentizar las cuchillas. Las hélices se componen de cuatro números, los dos primeros indican la longitud en pulgadas, y los dos siguientes la inclinación en grados, mayor inclinación supone mayor empuje y consumo.

Todas las hélices que utilizas para construir un dron vienen diseñadas para girar en sentido horario (CW) o en sentido antihorario (CCW). Hay que tener cuidado ya que la hélice debe ser acorde al motor, de lo contrario no funcionará de la forma prevista. Además de esto, es importante que pueda saber qué parte de la hélice debe mirar hacia arriba y qué parte de la hélice debe orientarse hacia abajo. También debemos elegir el material más adecuado, los más comunes son plástico y fibra de carbono.

Plástico: Es, con mucho, la opción más popular para hélices. Esto se debe principalmente a su bajo coste y durabilidad respetable. Desafortunadamente, las hélices de plástico tienen sus desventajas, como que si recibe algún golpe es muy probable que se dañen, pero no debería ser un problema por su bajo coste. Al aprender a construir un cuadricóptero son la mejor opción.

Fibra de carbono: Se utilizan cuando se busca algo que sea de muy alta calidad y que no se rompa, son muy difíciles de romper y ofrecen mucha más flexibilidad que

una hélice de plástico estándar, pero no son irrompibles así que no son indicadas para principiantes. Otra desventaja es el precio, muy superior a las de plástico.

Figura 2.3: Hélices de fibra de carbono.

2.2.3. Variadores de velocidad

Como vimos en el apartado de los motores, el control de los motores brushless debe realizarse fuera de los motores, para ello necesitamos los variadores de velocidad. Estos son más conocidos como ESC (Electronic Speed Controller, controlador de velocidad electrónico), y son unos circuitos integrados que constan de transistores FETs que se encienden y apagan a gran velocidad y de forma controlada para conseguir la salida deseada. Necesitan una señal de referencia para producir esa salida.

Un ESC consta de su propio PCB donde se encuentra su electrónica, dos cables para recibir la alimentación de la batería, tres cables para proporcionar los pulsos al motor y un conector con dos o tres cables para recibir la señal de referencia. Este tercer cable opcional es de alimentación y sirve por si queremos alimentar al circuito que proporciona la señal, los otros dos reciben una señal PWM (Pulse-Width Modulation, modulación por ancho de pulsos).

Una señal PWM es una técnica que modifica el ciclo de trabajo de una señal periódica, es decir, variamos la relación entre el tiempo que la señal se encuentra activa y el periodo total. Si la señal nunca está activa el ciclo de trabajo será 0, y si se encuentra

activa constantemente el el tiempo que se encuentra activa será igual al periodo, por lo que el ciclo de trabajo será 1. El resto de valores que puede tomar varía entre 0 y 1.

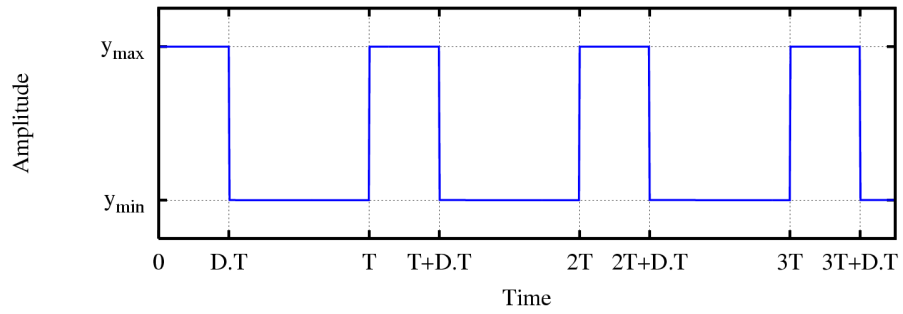

Figura 2.4: Señal PWM.

Los ESC han adoptado tradicionalmente la forma de transmitir señales que hacen los servos. Un servo envía la señal con una frecuencia de 50 Hz, es decir, cada 20 ms refrescamos el pulso que controla la posición del motor, y el el ancho de pulso determina la posición angular del servo, los pulsos con una duración de 1 ms corresponden a una posición de 0 grados, una duración de 1,5 ms a 90 grados y de 2 ms a 180 grados. En el caso de los motores 1ms corresponde al 0 % del acelerador y 2ms corresponde al 100 % del acelerador.

Los softwares más comunes para los ESC son BLHeli y Simonk, estos disponen de un nuevo protocolo llamado Oneshot125, que es 8 veces más rápido que la forma anterior, es decir, un ancho de pulso 125 us corresponde a 0 % del acelerador y 250 us corresponde al 100 % del acelerador. Para asegurarnos de que nuestro ESC soporta Oneshot125 debemos mirar la versión de software que dispone y consultar la referencia del software, puede ser que deba ser activado configurando el ESC.

Los motores brushless suelen ser bastante potentes, por lo que sería muy peligroso que comenzasen a funcionar nada más recibir una señal PWM, por eso los ESC implementan una secuencia previa llamada armado, este armado consiste en partir de una señal de 0 % de acelerador e ir subiendo, una vez adquiera cierta cantidad hay que volver a llevar la señal al 0 %. Los ESC enviarán una señal a los motores durante el armado para que estos emitan unos pitidos y podamos saber como se encuentra el proceso de armado. La figura 2.5 muestra la secuencia de armado para el software BLHeli.

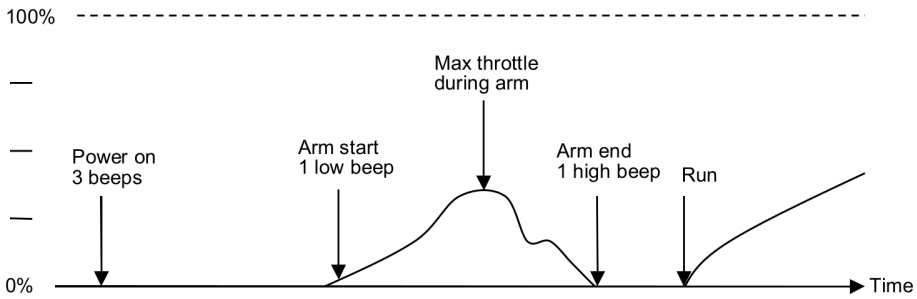

Figura 2.5: Secuencia de armado.

2.2.4. Placa controladora

Los cuadricópteros por naturaleza no son tan estables. Como cada aspa, además de proporcionarán sustentación, ejerce un par de torsión, e puede pensar que al tener cuatro hélices, dos girando en una dirección y dos girando en la dirección opuesta, terminan cancelando el par de torsión, pero no es tan sencillo. En el aire, un cuadricóptero está sometido a varias fuerzas diferentes. Soportará la fuerza del aire que el cuadricóptero está empujando a través de sus hélices, el viento, la presión del aire.

La placa controladora o controladora de vuelo permite enviar a cada motor de forma independiente su velocidad, además también incluye sensores para captar estas perturbaciones y, con el fin de contrarrestarlas, las tiene en consideración para calcular la señal que envía a cada motor, por lo que los motores no funciona del mismo modo en un entorno ideal o en un entorno real con sus posibles perturbaciones. Todo esto permite al cuadricóptero ser estable en el aire mientras vuela. También disponen de puertos para conectar otros periféricos, por lo que podemos decir que es el cerebro del cuadricóptero y todos necesitan de una para funcionar.

Por lo tanto, una placa controladora es un circuito integrado normalmente compuesto por un microcontrolador, sensores y pines de entrada y salida. Intentan ocupar el menor espacio posible para no aumentar el peso del cuadricóptero y se puedan colocar fácilmente. Para ello las dimensiones suelen ser estandar y llevan componentes SMD soldados en hornos. Una controladora de vuelo no conoce el tipo de UAV específico en el que está, por lo que es necesario establecer ciertos parámetros en el software.

Las placas puedes disponer varios sensores, pero el principal es una IMU (del inglés Inertial Measuerement Unit, Unidad de Medición inercial). Consiste en un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales de un aparato, usando una combinación de acelerómetros y giróscopos. Son

necesarios para que el microcontrolador sepa en todo momento la orientación espacial del dispositivo y elabore las señales de corrección concretas. El microcontrolador debe saber como leer los datos de que mide la IMU.

Un acelerómetro es un dispositivo que es capaz de detectar la aceleración en los tres ejes, mientras que un giróscopo o giroscopio es un dispositivo que es capaz de medir la rotación de un dispositivo. La diferencia es que el acelerómetro puede obtener la inclinación respecto al suelo, pero si se produce una fuerza en el cuadricóptero el acelerómetro no es capaz de distinguirla de la fuerza gravitacional, por lo que necesita apoyo del giróscopo.

Respecto a los microcontroladores, ara crear estas placas se puede optar por diferentes tipos, por ejemplo el ATmega644 de Atmel, microcontrolador de 8 bits de arquitectura AVR, pero los más extendidos en uso son la familia STM32 de STMicroelectronics, microcontroladores que implementan la arquitectura ARM 32 bits. Dentro de la familia STM32 existen se utilizan las series F1, F3, F4 y F7, un mayor número indica una mayor velocidad de procesamiento.

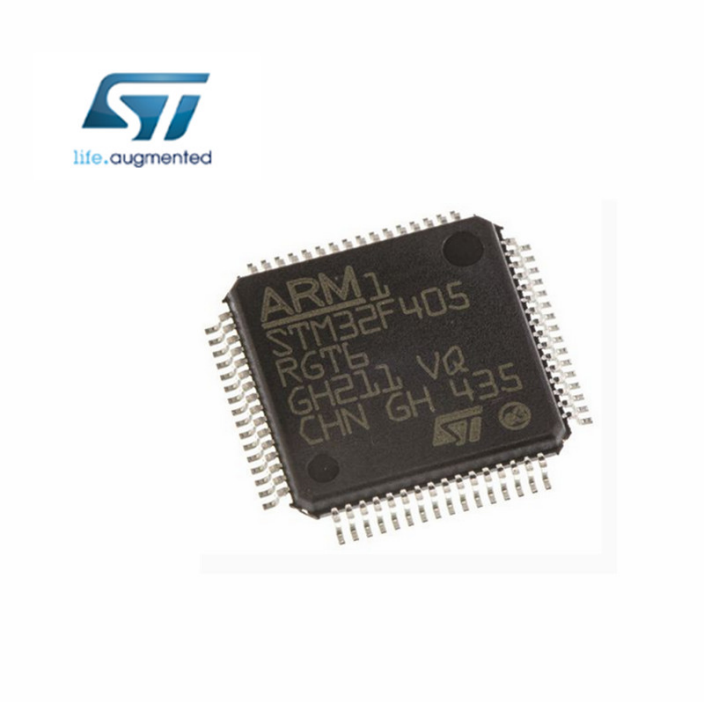

Figura 2.6: Procesador STM32F405 de STMicroelectronics.

Se puede obtener un cuadricóptero con una placa F1, pero actualmente se considera desfasada, y softwares como Betaflight dejarán de actualizarlo sus versiones para micro-

controladores F1. Los procesadores F3 tienen la misma velocidad que los procesadores F1, tienen una mayor memoria (más espacio para almacenar el código de firmware) y pueden admitir un UART adicional (tres de la F3 contra dos de la F1), además tiene un puerto USB para conectarlo a un ordenador, en las placas F1 es habitual dejar una UART libre para conectarse al ordenador.

La placas con procesador F4 tienen mejores prestaciones que las F3, Aunque ambas son una buena opción hoy en día. Lo que más destaca es la mayor memoria, la frecuencia de funcionamiento (180MHz de la F4 contra 72MHz de la F3) y los puertos UART (cinco de la F4 contra tres de la F3). Estos características extras deben ser tenidos en cuenta, ya que con algunas características nuevas de software la memoria del chip F3 se queda escasa.

Los procesadores F7 también tienen mayor velocidad de procesamiento y más puertos UART, pero las IMUs actuales limitan la velocidad del microcontrolador y no se puede aprovechar esa potencia extra que ofrecen, igual que en la mayoría de casos no es necesario disponer de todos los puertos UARTs que ofrecen, también suponen una inversión mayor, un mayor tamaño de la placa, por lo que es una opción pensada para el futuro más que para ahora.

El hardware por si solo no puede controlar un cuadricóptero, por eso necesita interaccionar con el software, este envía instrucciones que el hardware ejecuta, haciendo posible el funcionamiento, en nuestro caso, del cuadricóptero. Existen muchos softwares distintos, algunos de código cerrado que están pensadas para determinadas placas controladoras, pero también existen otros de código abierto que los fabricantes de placas puedan implementarlo, entre estos destaca BetaFlight y CleanFlight, que está respaldado por una gran comunidad.

En las primeras placas el hardware del controlador de vuelo estaba bien, ya que el software no aprovechaba al máximo las capacidades. Cada cuadricóptero tenía que pasar una gran cantidad de tiempo ajustándose para asegurarse de que el controlador enviaba los comandos correctos a los motores. Actualmente, el software es tan potente que los cuadricópteros vuelan bastante bien sin mucha afinación. De hecho, la tendencia parece ser que los controladores de vuelo te permiten conectarte de inmediato y volar sin problemas.

2.2.5. Batería

Para almacenar la energía que necesita un cuadricóptero necesita llevar consigo una batería. Una batería es un dispositivo que consta de una o más celdas electro-químicas que pueden convertir la energía química almacenada en corriente eléctrica. Cada celda consta de un electrodo positivo, o cátodo, un electrodo negativo, o ánodo, y electrolitos que permiten que los iones se muevan entre los electrodos, permitiendo que la corriente fluya fuera de la batería para llevar a cabo su función, alimentar un circuito eléctrico. Las baterías se presentan en muchas formas y tamaños, siendo las más comunes las de litio.

Las baterías de polímero de iones de litio, conocidas como baterías LiPo, son las más utilizadas en los UAV, y funcionan siguiendo el mismo principio que las baterías de iones de litio, el intercambio de electrones entre el material del electrodo negativo y el material del electrodo positivo mediante un medio conductor. Para evitar que los electrodos se toquen directamente, se coloca entre ellos un material con poros microscópicos que permite tan sólo los iones (y no las partículas de los electrodos) migren de un electrodo a otro.

El avance y bajada de precio en las baterías durante los últimos años han dado un impulso al mercado de los UAV. Las baterías LiPo se caracterizan por ser ligeras y por poder almacenar una gran cantidad de energía. Una batería LiPo puede almacenar una gran cantidad de energía y puede fabricarse en medidas personalizadas, y ofrecen una tasa de descarga muy alta. Esto último es realmente importante para los UAV. Las baterías LiPo vienen definidas por el número de celdas en serie y paralelo, la capacidad y la tasa de descarga.

1. **Número de celdas**

   El número de celdas de una batería se define con un número seguido de una S, siendo ese número la cantidad de celdas que tiene la batería conectada en serie: 3S, 4S... Cada celda es la encargada de contener el voltaje de la batería, y a mayor sea el número de celdas, mayor voltaje contendrá la batería. Cada celda tiene un voltaje de 3,7 V, un mayor voltaje total proporcionará al UAV una mayor potencia, siendo las 3S muy comunes entre la gente que se inicia. También encontramos la letra P que indica cuantos bloques de celdas en serie están conectadas en paralelo.

2. **Capacidad**

   La capacidad de una batería mide cuánta energía puede almacenar en su interior, se mide en mAh, es decir, mili amperios por hora; esta unidad indica la cantidad de consumo necesaria para descargar la batería en una hora. Para aumentar la duración de la batería debemos aumentar su capacidad.

3. **Tasa de descarga**

   La tasa de descarga representa la velocidad con la que la batería puede liberar su voltaje de manera segura. Se mide en C y este número viene respecto a la capacidad, es decir, con 30 y una capacidad de 1500 mAh indica que se puede descargar 45 A a la vez. La C indica los amperios constantes que puede descargar, y suele ir acompañado por un número mayor que es el pico de amperios que soporta, necesario porque los motores eléctricos tienen un pico elevado de consumo durante el arranque. También suele ir indicada la tasa de carga.

   Las baterías LiPo son bastante delicadas, una batería con buen uso y bien mantenida puede llegar a realizar más de 300 ciclos de carga y descarga, mientras que una batería mal cuidada puede no llegar ni a los 50 ciclos. Es necesario cargar las LiPo con un cargador específico, preferiblemente con un cargador balanceador que cargue cada celda de manera independiente para que las baterías estén equilibradas. Usar un cargador inadecuado puede dañar la batería y hacer que se incendie.

   Por último, comentaremos unas cuantas cosas para manejar las baterías de forma segura y para alargar la vida útil de las mismas, como no dejar desatendidas las baterías mientras se cargan, no cargar las baterías por encima del voltaje indicado por el fabricante, almacenarlas a un 40 % de carga si va a estar mucho tiempo almacenada, preferiblemente en fundas ignífugas, no cargarlas si se encuentran muy frías (menos de 5°C) o muy calientes.

2.2.6. Emisora

Pese al sistema de control que puede implementar un cuadricóptero, necesitamos alguna forma de comunicarnos con este, para indicarle si queremos que ascienda, se desplace, etc. Para esto tenemos las emisoras de radio. Las emisoras disponen de un

número de canales, cada orden ocupa un canal, y para volar un cuadricóptero debemos controlar el acelerador, yaw, pitch y roll, por lo que como mínimo necesitamos cuatro canales.

Los canales adicionales normalmente se llaman canales auxiliares, en forma de interruptores, palanca y potenciómetros. Estos se usan para cambiar modos de vuelo o activar ciertas funciones, como armar el cuadricóptero (esto suele ser generalmente necesario). Las emisoras funcionan a 2,4 GHz y ahora no hay que preocuparse por recibir frecuencias de otro cuadricóptero y vienen con un receptor que hay que conectar con la placa controladora. Por último, podemos elegir entre unas palancas que se controlen con los dedos pulgares o con varios dedos.

2.2.7. Estructura

Para concluir hablaremos de la estructura, elemento que da forma al cuadricóptero. Todos los componentes deben ser adecuados para la finalidad del cuadricóptero pero especialmente la estructura debe ser acorde, ya que la finalidad del cuadricóptero va ligada a su tamaño y morfología, aporta gran parte del peso total. La calidad de la estructura también es importante porque su forma nos proporciona la aerodinámica del cuadricóptero y debe proteger los componentes electrónicos.

Al igual que las hélices, la estructura pueden fabricarse en plástico o en fibra de carbono, siendo el plástico más barato pero la fibra de carbono de mayor calidad. Además de estas opciones también tenemos el aluminio, metal que se caracteriza por su ligereza y relación calidad precio, además también es flexible. Se puede utilizar para construir la estructura en su totalidad, o para construir ciertas partes. También se ve la madera pero es menos común.

2.3. Prototipo

Ahora ya tenemos una visión global de un cuadricóptero, el siguiente paso será realizar nuestro prototipo del balancín. Para ello debemos elegir un microcontrolador PIC y el resto de componentes, procurando que no se excedan demasiado de precio debido al presupuesto limitado del que disponemos. Algunos de estos componentes podrán ser los mismos de un cuadricóptero y para otros tendremos que adaptar los componentes

para que operen con dicho microcontrolador. La elección del microcontrolador se estudiará en el próximo capítulo ya que es la parte principal de este proyecto y debe ser tratada en profundidad.

1. **Motores**

   Elegiremos los EMAX 2208 II de 2000 Kv. Para el balancín necesitaremos 2 motores, por lo que utilizaremos uno de sentido CW y otro de CCW. Estos motores tienen una relación calidad precio muy buena, y pueden utilizarse perfectamente en un cuadricóptero comercial.

Motor type	The voltage (V)	Propeller size	current (A)	thrust (G)	power (W)	efficiency (G/W)	speed (RPM)
MT2208 II - 2000KV	8	HQ5040	4.8	220	38.4	5.7	14000
		HQ6045	8.8	370	70.4	5.3	12300
	12	HQ5040	8.4	430	100.8	4.3	19900
		HQ6045	15.2	690	182.4	3.8	16720
	14.8	HQ5040	11.3	590	167.2	3.5	23500
		HQ6045	19.4	910	287.1	3.2	19320
MT2208 II - 1500KV	11.1	HQ6045	7.7	430	85.5	5.0	13400
		8*4.5 CF	15.8	700	175.4	4.0	18700
	14.8	HQ6045	11.5	690	170	4.1	16800

Figura 2.7: Tabla de características del motor MT2208 II.

2. **Hélices**

   Para las hélices elegimos las Gemfan 5030, unas hélices sencillas de plástico pero que para nuestro propósito cumplen su función perfectamente. Necesitamos una en sentido horario y otra en sentido antihorario.

3. **Batería**

   Utilizaremos una batería 3S1P de 1600mAh y 30C. Al tener tres celdas su voltaje total es de 11,1V. Es una batería asequible ideal para principiantes, y al utilizar la mitad de motores no debemos preocuparnos por la duración. En la tabla anterior podemos comprobar que con una hélice de 5 pulgadas, 40 grados y 12 voltios el consumo es de 8,4 A, por lo que nuestro batería es adecuada para mover incluso cuatro motores con estas hélices. También necesitaremos un cargador.

4. **ESC**

   Utilizaremos los LittleBee 30A-S de la marca Favourite, estos implementan un procesador Silabs y el software BLHeli en su versión 16 que se puede actualizar, para ello necesitamos que esté disponible una nueva versión para este modelo. Estos ESC permiten el paso de hasta 30A, por lo cual son válidos para nuestros motores y hélices.

5. **Placa controladora**

   Esta será sustituida por el microcontrolador que elegiremos en el próximo capítulo y por la IMU MPU6050 de InvenSense. Esta dispone de seis ejes, tres del acelerómetro y tres del giroscopio. Hemos elegido esta IMU debido a su bajo coste, popularidad y que se comunica a través de I2C, a diferencia de otra IMU popular como la MPU6000 también de InvenSense que se comunica a través de SPI.

6. **Emisora**

   En lugar de utilizar algún sistema de radio emplearemos comunicación bluetooth. Para ello utilizaremos un módulo HC-05 que se conectará al microcontrolador a través del puerto UART, y para enviar instrucciones utilizaremos un ordenador. Hemos optado por esta opción debido a la diferencia de precio, y porque además de enviar instrucciones podemos utilizarla para recibir datos del programa del PIC, muy importante a la hora de desarrollar un programa.

7. **Estructura**

   Para realizar las pruebas del cuadricóptero con dos motores necesitaremos un balancín con un eje en el centro para que realice el balanceo. Para ello utilizaremos madera de barca debido a su bajo coste, un eje cilíndrico de metal y silicona para fortalecer las uniones de madera.

Además de estos componentes necesitaremos algunos otros componentes de menor importancia para fabricar el prototipo del balancín, pero ello lo detallaremos en el capítulo 5 en el que probaremos el prototipo. Para extender el balancín a un cuadricóptero necesitaríamos adaptar el código, el microcontrolador, disponer de un a estructura y por supuesto incluir dos motores y hélices extra.

Capítulo 3

Microcontroladores

3.1. Introduccion

En el capítulo anterior elegimos todos los componentes a excepción del microcontrolador, esta decisión no es trivial y por eso le dedicamos este capítulo. Los microcontroladores tienen una gran importancia en el presente, están presentes en nuestro trabajo, en nuestra casa y en nuestra vida, gobiernan la mayor parte de los aparatos que fabricamos y usamos los humanos, como ratones y teclados de los ordenadores, los microondas, televisores, etc. En este capitulo explicaremos un poco acerca de los microcontroladores y algunas de sus familias, y por último elegiremos los más apropiados.

3.2. ¿Qué es un microcontrolador?

Un microcontrolador (abreviado MCU) es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Un microcontrolador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento (CPU del inglés Central Processing Unit), memoria y periféricos de entrada/salida. Se emplea para controlar el funcionamiento de una tarea determinada y, debido a su reducido tamaño, suele ir incorporado en el propio dispositivo al que gobierna. Esta característica es la que le da la denominación de controlador incrustado o empotrado (del inglés embedded controller) y es su principal diferencia con los microprocesadores, que sólo implementa la CPU.

Un microcontrolador sólo necesita un mínimo de circuitos externos de apoyo, es

un sistema que contiene un computador completo y sus prestaciones no se pueden modificar. La idea es que el circuito integrado se coloque en el dispositivo, enganchado a la fuente de energía y de información que necesite, y eso es todo. Otra diferencia es la arquitectura, los microprocesadores utilizan la arquitectura de Von Neumann en la que la memoria de datos y memoria de programas se almacenan juntas, y los microcontroladores emplean la arquitectura Harvard, donde los datos e instrucciones están separados y el acceso a una memoria u otra depende de la instrucción.

Los microcontroladores disponen de una gran variedad de dispositivos de Entrada/Salida, como Convertidor Analógico Digital, temporizadores, UARTs y buses de interfaz serie especializados, como I2C y CAN. El tamaño de la Unidad Central de Procesamiento, la cantidad de Memoria y los periféricos incluidos dependerán de la aplicación. Por todo esto cada fabricante de microcontroladores oferta un elevado número de modelos diferentes, y al elegir sólo lo que necesitamos reducimos el coste económico y el consumo de energía del sistema en el que se apliquen.

Cuando se fabrica un microcontrolador no contiene datos en la memoria ROM. Para que pueda controlar algún proceso es necesario crear y luego grabar algún programa, el cual puede ser escrito en lenguaje ensamblador u otro lenguaje de alto nivel (C, Java, C++). Para que el programa pueda ser grabado en la memoria del microcontrolador debe ser codificado en sistema numérico hexadecimal, que es finalmente el código que entiende el microcontrolador.

3.3. Microcontroladores PICs

Para nuestro proyecto hemos utilizado un microcontrolador de Microchip, concretamente la familia PIC. Los PICs disponen principalmente de tres gamas de productos clasificados por los bits de su arquitectura (8, 16 y 32 bits), y también dispone de una gama llama DSCs (controladores de señal digital) de 16bits. Dentro de los 8 bits también podemos hacer varias clasificaciones, entre la gama básica, la gama media, la gama media mejorada y la gama alta.

Podemos decir que Microchip proporciona un rendimiento escalable, ya que con el mismo entorno de desarrollo podemos programar desde aplicaciones sencillas para microcontroladores modestos hasta aplicaciones en tiempo real para microcontroladores más potentes, esto es ideal porque para nuestro proyecto no necesitamos un tiempo real

crítico y podemos elegir un controlador de 8 bits. Además los microcontroladores de 8 bits de Microchip son muy populares en el mercado.

3.3.1. Gama básica

Los microcontroladores de gama básica (o baseline en inglés) utilizan una palabra de programa de 12 bits y proporcionan la cantidad adecuada de características y opciones para minimizar los gastos y hacer el trabajo correctamente. Con tantas opciones disponibles, elegir el microcontrolador Baseline Flash PIC correcto para una aplicación es rápido y fácil. Los PIC10F2xx, PIC12F5xx, PIC16F5x y PIC16F5xx pertenecen a esta clase.

Algunas características son:

- 33 instrucciones de 12 bits de ancho.
- Memoria de programa de 2K instrucciones.
- Máximo de 144 bytes de memoria RAM.
- Pila de 2 niveles de profundidad.
- 1 Registro Seleccionador de Registros (FSR) de 8 bits.
- Factores de forma más pequeños posible.

3.3.2. Gama media

Los microcontroladores PIC de rango medio (o Mid-Range en inglés) son el siguiente nivel en rendimiento y características de nuestros microcontroladores Baseline PIC. Utilizando una palabra de instrucciones de 14 bits, estos ya disponen de multitud de periféricos, siendo ideales para aplicaciones multidimensionales que requieren un mayor nivel de control integrado, pero sólo cuentan con 35 instrucciones para aprender, lograr un rendimiento óptimo del sistema sigue siendo una tarea fácil. Aquí incluimos a los PIC10F3xx, PIC12F6xx, PIC12F7xx, PIC16F6xx, PIC16F7xx, PIC16F8xx, PIC16F9xx.

Algunas características son:

- 35 instrucciones de 14 bits de ancho.
- Memoria de programa de 8K instrucciones.
- Máximo de 368 bytes de memoria RAM.
- Pila de 8 niveles de profundidad.
- 1 Registro Seleccionador de Registros (FSR) de 9 bits.
- Manejo de interrupción de hardware.
- Otras características integradas, que incluye EEPROM, LCD, comunicaciones serie, etc.

3.3.3. Gama media mejorada

Los microcontroladores PIC de rango medio mejorado (o Enhanced Mid-Range en inglés) tiene la misma base que la gama media pero proporciona un rendimiento adicional, al tiempo que mantiene la compatibilidad con la gama media para lograr la migración de productos. Son los PIC12F1xxx y PIC16F1xxx.

Algunas características son:

- 49 instrucciones de 14 bits de ancho.
- Memoria de programa de 32K instrucciones.
- Máximo de 4KB de memoria RAM.
- Pila de 16 niveles de profundidad.
- 2 Registro Seleccionador de Registros (FSR) de 16 bits.
- Manejo de interrupción de hardware con ahorro de contenido.
- Conjunto de funciones avanzadas, comunicaciones seriales múltiples y capacidad de control del motor, etc.

3.3.4. Familia PIC18

La familia PIC18 combina el nivel máximo de rendimiento e integración con la facilidad de uso de una arquitectura de 8 bits. Son los microcontroladores más potentes de Microchip.

Algunas características son:

- Oscilador interno de hasta 64 MHz.
- Memoria de programa de 128K instrucciones.
- Máximo de 1 kB de memoria EEPROM.
- Máximo de 8 kB de memoria SRAM.
- Pila de 32 niveles de profundidad.
- Manejo de interrupción de vectores (VIC).
- ADC de 12 bits, hasta 43 canales.
- Indicador de temperatura en el chip.
- Comunicación: UART, SPI, I2C, CAN, USB, Ethernet, LCD.
- Disponible en paquetes de 18 a 100 pines.

3.4. Microcontrolador elegido

Para el balancín necesitaremos como mínimo una UART, un puerto I2C y 2 módulos PWM, por lo que utilizaremos el PIC16F876A, que es un microcontrolador de 8 bits Mid-Range muy utilizado entre los usuarios y "hermano mayor" del PIC16F84A, que también es bastante utilizado. Además el PIC16F876A incluye tres temporizadores/contadores, siendo uno de ellos necesario para el PWM.

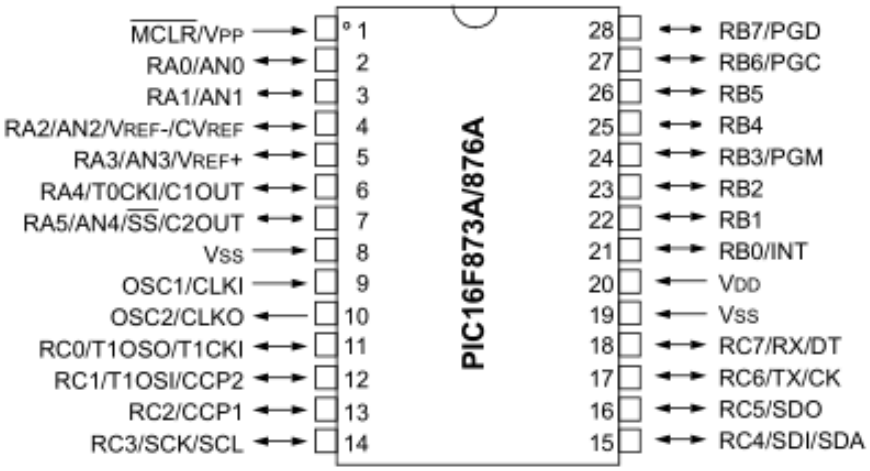

Figura 3.1: Pines PIC16F876A.

Si necesitasemos cuatro salidas PWM podríamos elegir el PIC16F1827, de 8 bits que pertenece a los Mid-Range Enhanced y tiene un encapsulado menor que el del PIC16F876A.

Capítulo 4

Recursos

4.1. Introduccion

En el Capítulo 2 elegimos los componentes que vamos a necesitar para el prototipo y en el Capítulo 3 el microcontrolador que lo gobierna, así que ya tenemos todos los componentes para realizar nuestro balancín. Este capítulo lo dedicaremos a explicar los recursos que necesitamos para hacer funcionar nuestro prototipo, que son el Timer0, la UART, el I2C y los PWM.

Para la programación utilizaremos el lenguaje C, y nos permite crear bibliotecas para reutilizar código, estas agrupan funciones en uno o varios archivos, lo cual permite que en archivo principal solamente tengamos que llamar a la función sin preocuparnos de cómo funciona por dentro. Esto es ideal para centrarnos primero en los recursos (en este capítulo) y en el programa en sí (siguiente capítulo).

Cada biblioteca tiene un archivo cabecera (.h) donde se declaran las funciones y un archivo donde se definen las funciones (.c), y el programa en sí se encuentra en el archivo main.c. Este capítulo estará dividido en un apartado para cada recurso y en cada uno explicaremos cómo funciona y hemos realizado su biblioteca. En el Anexo A podemos ver el código del prototipo y en el Anexo B el código de estas bibliotecas.

4.2. Configuración

Este no es un recurso del micro en sí pero crearemos un archivo .h para la configuración del PIC. A esta la llamaremos “configuration.h” y no contiene funciones, ya que

se encarga de seleccionar los bits de la palabra de configuración. Además utilizaremos este archivo para seleccionar la frecuencia del PIC e indicar que usaremos la biblioteca de Microchip xc.h.

REGISTER 14-1: CONFIGURATION WORD (ADDRESS 2007h)

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
CP	—	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	—	—	$\overline{\text{PWRTEN}}$	WDTEN	Fosc1	Fosc0
bit 13													bit0

Figura 4.1: Palabra de configuración.

La palabra de configuración es un registro de 14 bits, por lo tanto estará codificada en un número binario de 14 bits, que tiene su equivalente en hexadecimal de 4 dígitos de longitud. Cada uno de esos 14 bits almacena un 1 o un 0, por lo que por ejemplo la opción CP estará activado o desactivado en función de si CP vale 1 o 0. Para facilitarnos esta tarea podemos asociar CP a ON u OFF, (mucho más legible para nosotros) y el compilador lo traducirá. Algunos bits trabajan en parejas y disponen de más opciones que un ON o un OFF. A continuación explicaremos cada bit.

- CP: Flash Program Memory Code Protection bit, protege el código para que no podamos sobreescribirlo, lo desactivamos.
- DEBUG: In-Circuit Debugger Mode bit, activa el debugger en los pines RB6 y RB7, lo desactivamos.
- WRT: Flash Program Memory Write Enable bits, protege areas de la memoria flash, lo desactivamos.
- CPD: Data EEPROM Memory Code Protection bit, protege la memoria EEPROM, lo desactivamos.
- LVP: Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit, si lo activamos RB3 se utiliza para programar con bajo voltaje, si no se usa como puerta de Entrada/Salida, lo desactivamos.
- BOREN: Brown-out Reset Enable bit, resetea el PIC si baja cierto nivel el voltaje, lo desactivamos.
- PWRTEN: Power-up Timer Enable bit, proporciona un delay para encenderse después de un reset, lo desactivamos.

- WDTEN: Watchdog Timer Enable bit, habilita el Watchdog, lo desactivamos.
- FOSC: Oscillator Selection bits, elegimos el tipo de oscilador que queremos, nosotros elegiremos un cristal de 16MHz, por lo que debemos seleccionar HS. Pondremos unos condensadores de 22 pF tal como muestra el datasheet.

TABLE 14-2: CAPACITOR SELECTION FOR CRYSTAL OSCILLATOR

Osc Type	Crystal Freq.	Cap. Range C1	Cap. Range C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
	8 MHz	15-33 pF	15-33 pF
	20 MHz	15-33 pF	15-33 pF

Figura 4.2: Tipos de osciladores de cristal.

Aprovechamos este punto para explicar el conexionado básico del PIC, debemos conectar el pin VDD a 5V y los pines VSS a 0V, MCLR debe ir a VDD ya que tiene lógica inversa, y el cristal debe conectarse entre los pines OSC1 y OSC2, y estas dos patillas deben tener un condensador cada una que vaya a 0V. Los leds que podamos necesitar no necesitan resistencia ya que los pines del microcontrolador suministran poca intensidad, y los pulsadores necesitan un acondicionamiento con un Pull-Down para evitar rebotes.

4.3. Interrupciones y timer

Después de esta toma de contacto prepararemos una biblioteca para el Timer0 con una interrupción, que llamaremos “timer.h”. Los programas suelen ser secuenciales, por lo que si en algún momento queremos comprobar un estado, el microcontrolador deberá recorrer todo el programa hasta llegar a la línea de código donde hace la comprobación, esto puede ser un problema en programas largos y las interrupciones buscan

solucionarlo. Las interrupciones nos permiten que ante una determinada condición el programa detenga lo que está realizando y salte a la rutina de interrupción, donde escribimos cómo debe gestionarla. Al resolver la interrupción continúa el programa donde lo había detenido.

Prepararemos las interrupciones de tiempo por seguridad, debido a que en las bibliotecas generaremos varias funciones con bucles que esperan la recepción de datos y ante alguna interferencia pueden ser infinitos, dejando el programa atascado en el bucle. La interrupción salta cuando detecta que está demasiado tiempo en el bucle,y le permite salir de éste.

Para generar esta biblioteca necesitaremos 3 registros, el registro INTCON, el OPTION_REG y el TMR0. El registro INTCON es el que se encarga de habilitar o deshabilitar las interrupciones del PIC, el registro OPTION_REG se encarga de administrar el módulo Timer0 (también se podría usar como contador). TMR0 es el contador que se va incrementando, la interrupción se produce cuando el valor de TMR0 pasa de FFh a 00h, así que ajustando el valor inicial del TMR0 podemos ajustar el tiempo temporizado.

1. Del registro OPTION_REG los bits que debemos configurar son:

    - T0CS: TMR0 Clock Source Select bit, sirve para seleccionar si funciona como temporizador o contador del pin T0CKI, si vale 0 funciona como temporizador.
    - PSA: Prescaler Assignment bit, permite elegir si asignar el prescaler al WDT o al Timer0, si vale 0 se asigna al temporizador.
    - PS2:PS0: Prescaler Rate Select bits, 3 bits que permiten elegir el valor del prescaler, elegiremos 8.

2. Del registro INTCON los bits que debemos configurar son:

    - GIE: Global Interrupt Enable bit, habilita el uso de interrupciones en general, si vale 1 se permiten.
    - TMR0IE: TMR0 Overflow Interrupt Enable bit, habilita la interrupción cuando se desborda el TMR0. Si vale 1 se permite.

PS2:PS0: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Figura 4.3: Tabla PS2:PS0.

- TMR0IF: TMR0 Overflow Interrupt Flag bit, indica si se ha producido el desbordamiento del TMR0 y el programa debe limpiar el valor, si vale 1 se ha producido desbordamiento.

3. En el registro TMR hay que cargarle un valor para que realizamos la temporización deseada, nosotros elegiremos 10ms y si queremos temporizaciones mayores podemos realizarlo con un contador que sume uno en la rutina de interrupción.

$$Temp = 4\frac{1}{Fosc}(256 - TMR0) \cdot (PS2 : PS0)$$

Despejando:

$$TMR0 = 256 - Fosc\frac{Temp}{4(PS2 : PS0)}$$

Para los valores de Fosc = 16MHz, Temp = 0.2ms y PS2:PS0 = 8 obtenemos TMR0 = 156.

La biblioteca que crearemos consta de 6 funciones, la primera para iniciar el timer, la segunda para contar tiempo sin límite, la tercera para saber cuando hemos contado 20ms, la cuarta para aumentar los contadores, la quinta para reiniciar el primer contador y la ultima función para reiniciar el contador de 20ms. La función para atender la interrupción la incluiremos en main.c.

4.4. PWM

Como vimos en el Capítulo 2 los ESC necesitan recibir una señal PWM para generar la alimentar los motores, para ello el PIC dispone de los registros CCP (Capture/Compare/PWM Module), al tener dos módulos CCP se llaman CCP1 y CCP2. En estos módulos tenemos que configurarlos para que se comporten como PWM, que necesita el apoyo del Timer2. Para ello crearemos una biblioteca "pwm.h".

Para generar un PWM debemos tener en cuenta el ciclo de trabajo (con 10 bits de resolución) y el periodo del PWM. Necesitamos algunos bits de 7 registros distintos: CCPR1L, CCPR2L, CCP1CON, CCP2CON, TMR2, PR2, T2CON. CCPR1L y CCPR2L para el ciclo de trabajo, CCP1CON y CCP2CON configura el módulo CCP1 y CCP2 respectivamente, T2CON configura el Timer2, PR2 es el registro para el periodo y TMR2 realiza la cuenta. Además TRISC2 y TRISC1 deben valer 0.

1. De los registros CCP1CON y CCP2CON los bits que configuramos son:

   - CCPxX:CCPxY: PWM Least Significant bits, son los dos bits menores del ciclo de trabajo.
   - CCPxM3:CCPxM0: CCPx Mode Select bits, sirven para elegir entre capturador, comparador o PWM. Poniendo los dos primeros a 1 seleccionaremos el modo PWM.

2. En los registros CCPR1L y CCPR2L hay que cargar los 8 bits mayores del ciclo de trabajo, para obtener el ciclo de trabajo deseado seguimos la siguiente fórmula:

   $$PWM(activo) = (CCPRxL : CCPxCON(5:4)) \cdot Tosc \cdot Prescaler$$

   Despejando:

   $$(CCPRxL : CCPxCON(5:4)) = \frac{PWM(activo)}{Tosc \cdot Prescaler}$$

   Estos valores los tendremos que actualizar constantemente ya que con ellos variaremos la velocidad de los motores.

3. Del registro T2CON los bits que configuramos son:

- TMR2ON: Timer2 On bit, permite el uso del timer. Si vale 1 lo habilitamos.
- T2CKPS1:T2CKPS0: Timer2 Clock Prescale Select bits, aquí seleccionamos el valor del preescaler, con 01 pondremos el prescaler a 4.

4. En el registro PR2 seleccionamos el periodo del PWM con la siguiente fórmula:

$$PWM(periodo) = (PR2 + 1)4 \cdot Tosc \cdot Prescaler$$

Despejando:

$$PR2 = \frac{PWM(activo)}{4 \cdot Tosc \cdot Prescaler} - 1$$

Para los valores de Fosc = 16MHz, Prescaler = 4, Para un periodo de 250 us obtenemos un valor para PR2 de 249.

5. El registro TMR2 temporiza y cuando coincide con PR2 se limpia y vuelve a empezar a contar.

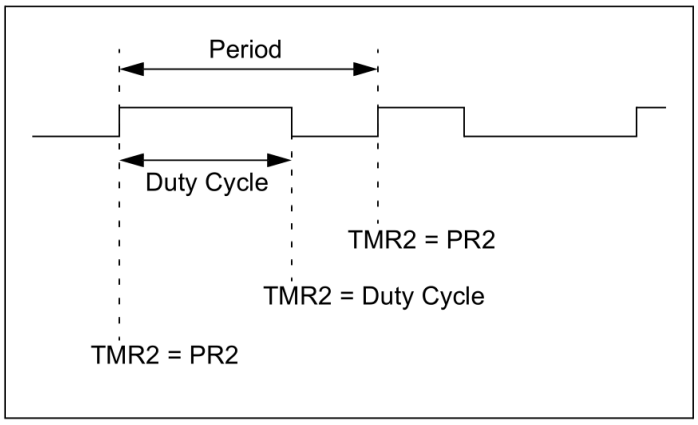

Figura 4.4: Ciclo de funcionamiento del PWM.

La biblioteca “pwm.h” tiene 6 funciones, las dos primeras para iniciar los módulos PWM 1 y 2, las dos siguientes para asignarle el valor de salida a cada una de ellas y las dos últimas para detener ambos módulos.

4.5. UART

El siguiente paso que realizaremos será configurar el módulo USART del PIC (Universal Synchronous Asynchronous Receiver Transmitter), es una de las partes más importantes del proyecto, ya que nos permite comunicarnos con el microcontrolador, tanto enviándole datos como recibirlos. En este apartado volveremos a crear una nueva biblioteca, en este caso la de la UART ("uart.h"), que sirve para que el PIC pueda recibir una nueva referencia, y de paso enviaremos las medidas que realicemos al ordenador. Esto es posible gracias al módulo bluetooth HC-05.

La USART también es conocida como SCI (Serial Communications Interface, en español Interfaz de Comunicaciones Serie), y cual puede ser configurada como asíncrona, síncrona (maestro) o síncrona (esclavo). Estos tres métodos utilizan las patillas RC7 y RC6 pero la diferencia es la señal de reloj; la comunicación asíncrona no dispone de ella por lo que RC7 recibe datos (RX) y RC6 envía datos (TX), mientras que en la síncrona RC7 se utiliza para datos (DT) mientras que RC6 se usa para generar la señal de reloj (CK). La diferencia entre maestro y esclavo es solamente si genera o recibe la señal de reloj.

En nuestro caso usaremos la configuración asíncrona, que se conoce como UART. A la hora de conectar las patillas, si una patilla sirve para enviar datos debe conectarse a una que sirva para recibir datos, por eso a la patilla RX debe conectarse con TX y viceversa. El protocolo serie asíncrono tiene unas reglas para evitar el uso de reloj, pero no hay una forma única de enviar los datos, así que debemos establecer algunos criterios según estas reglas.

1. **Tasa de Baudios (Baud Rate)**

   Indica la velocidad de transmisión en una linea serie, normalmente en bits por segundo (bps). El requisito es que el dispositivo que envía y el que recibe tengan la misma tasa, pero hay unos valores normalizados, como 4800, 9600, 19200... Nosotros utilizaremos 38400.

2. **Bits de sincronización**

   Son el bit de inicio al principio del bloque de bitsy los de parada al final del bloque, sólo hay un bit de inicio que se indica bajando la línea del valor 1 al 0, en cambio los de parada se puede elegir entre uno o dos bits, que dejan la linea en el valor 1. Lo más común es 1 bit y nosotros lo haremos así.

3. **Bits de paridad**

   Es un bit para comprobar errores, puede ser bit de paridad par o impar. Van detrás delos bits de datos y antes del bit de parada. Es opcional y pensado para medios ruidosos, así que nosotros no lo utilizaremos.

4. **Bits de datos**

   Son los bits que contienen la información que queremos enviar. Su tamaño puede ser variable pero nosotros elegiremos 8 bits. También importa el orden, puede empezar del más a menos signifivo o viceversa. Lo normal si no se especifica es primero el menos significativo.

Estos parámetros se pueden resumir escribiendo 9600 8N1, que indica 9600 bauds, 8 bits de datos, sin paridad y un bit de parada.

Para nuestro prototipo queremos que el microcontrolador envíe continuamente las medidas que toma al ordenador, para así poder supervisar el proceso, pero también queremos que cuando queramos cambiar la referencia de posición el microcontrolador lea el dato que le enviamos inmediatamente, y como podemos enviar la nueva referencia en cualquier instante necesitaremos nuevamente una interrupción.

Para crear nuestra biblioteca necesitamos utilizar 8 registros distintos: TXSTA, RCSTA, SPBRG, TXREG, RCREG, INTCON, PIE1 y PIR1, aunque no vamos a necesitar de todos los bits. TXSTA y RCSTA sirve para configurar la USART, SPBRG sirve para generar el Baud Rate deseado, con TXREG enviamos, con RCREG recibimos, con PIR1 sabemos el estado de TXREG y RCREG y con INTCON y PIE1 elegimos las interrupciones. Además TRISC7 y TRISC6 deben valer 1.

1. Del registro TXSTA los bits que debemos configurar son:

   - TX9: 9-bit Transmit Enable bit, permite la transmisión de 9 bits, a 0 lo deshabilitamos.
   - TXEN: Transmit Enable bit, bit de habilitación de transmisión, a 1 lo habilitamos.
   - SYNC: USART Mode Select bit, permite elegir entre síncrono y asíncrono, a 0 establecemos asíncrono.

- BRGH: High Baud Rate Select bit, permite elegir entre alta velocidad y baja velocidad, a 1 establecemos alta velocidad.
- TRMT: Transmit Shift Register Status bit, hemos dicho que en el registro PIR1 tenemos los bits para saber el estado de TXREG y RCREG, pero TXREG no envía directamente el bit, sino que alimenta un buffer llamado TSR, por lo que para saber el estado de la transmisión debemos conocer el estado de TSR. TRMT permite saber si el buffer está lleno o vacío, cuando la transmisión se complete se pondrá a 1. Bit sólo de lectura.

2. Del registro RCSTA los bits que debemos configurar son:

   - SPEN: Serial Port Enable bit, configura RC7 y RC6 para la comunicación serie, a 1 lo habilitamos.
   - RX9: 9-bit Receive Enable bit, permite la recepción de 9 bits, a 0 lo deshabilitamos.
   - CREN: Continuous Receive Enable bit, permite la recepción continua de datos, a 1 lo habilitamos.

3. En el registro SPBRG cargamos un valor para generar el Baud Rate, para ello tenemos una fórmula que relaciona ambos valores, y la fórmula seleccionada depende del bit BRGH del registro TXSTA (que pusimos a 0). Queremos generar un Baud Rate de 38400.

   La fórmula para BRGH = 1

$$BaudRate = \frac{Fosc}{16(SPBRG + 1)}$$

   Despejando:

$$SPBRG = \frac{Fosc - 16 \cdot BaudRate}{16 \cdot BaudRate}$$

   Para los valores de Fosc = 16MHz y Baud Rate = 38400, obtenemos un valor para SPBRG de 25,04166, así que igualamos SPBRG a 25.

4. En TXREG cargamos el byte que vamos a enviar.

5. De RCREG leemos el byte que hemos recibido.

6. Del registro PIR1 sólo necesitamos utilizar un bit:

   - RCIF: USART Receive Interrupt Flag bit, indica que hemos recibido un byte en RCREG, además si tenemos habilitada la interrupción indicará que se ha producido. Si vale 1 indica que hemos recibido el byte y se limpia cuando RCREG se vacía.

7. Del registro INTCON los bits que debemos configurar son:

   - GIE: Global Interrupt Enable bit, permite las interrupciones, a 1 lo habilitamos.
   - PEIE: Peripheral Interrupt Enable bit, habilita las interrupciones de los periféricos (registros PIE1 y PIE2), a 1 lo habilitamos.

8. Del registro PIE1 los bits que debemos configurar son:

   - RCIE: USART Receive Interrupt Enable bit, permite la interrupción cuando recibimos un dato, a 1 lo habilitamos.

La biblioteca para la UART necesita 5 funciones, una para inicializar la UART, dos para leer y escribir un byte, otras dos para leer y escribir strings (compuestas por las funciones de leer y escribir un byte), pero nosotros operamos con números y la UART envía caracteres ASCII, así que necesitamos 4 funciones adicionales, dos para convertir un char a un número y viceversa, y otras dos para convertir un string a un número y viceversa (apoyadas en las dos anteriores). De nuevo la interrupción se atenderá en el main.c.

El PIC debe ser conectado a otro dispositivo, en nuestro caso lo conectaremos a un ordenador, podríamos conectar los dos cables con el ordenador a través de un adaptador USB, pero para poder controlarlo remotamente usaremos el HC-05 que hace de intermediario, pero el PIC debe cablearse hasta el HC-05, además también debe ser alimentado.

El HC-05 tiene dos modos, el de datos (para funcionar) y el de comandos (para configurarlo). Para configurarlo fácilmente podemos utilizar un arduino, debemos alimentar el módulo después de que el pin "EN" esté alimentado. En este modo tenemos que enviarle unos códigos llamados comandos AT, y configuraremos el bluetooth como esclavo (ya que el bluetooth del ordenador actua como maestro) y a una tasa de 38400 bauds. Con estos comandos también podemos cambiar el nombre del módulo o su código de emparejamiento (por defecto 1234).

4.6. I2C

El protocolo Circuito Inter-integrado (I2C, del inglés Inter-integrated Circuit) es un protocolo desarrollado por Philips para permitir que uno o más circuitos digitales "maestros"se comuniquen con multiples circuitos .ºesclavos.º con uno también. El PIC tiene un módulo llamado MSSP (Master Synchronous Serial Port) que se encarga de gestionar los protocolos I2C y SPI (Serial Peripheral Interface). El I2C es necesario porque es el protocolo en el que la IMU envía valores al PIC, y en este caso tendremos que crear dos bibliotecas, la del I2C ("i2c.h"), y la de la IMU ("mpu6050.h"). Esta última utilizará las funciones de la librería I2C.

Al igual que SPI, está diseñada solo para comunicaciones de corta distancia dentro de un solo dispositivo, pero I2C lo mejora en que sólo necesita dos cables, como pasa en la UART, y mejora a la UART ya que esta es adecuada para las comunicaciones entre sólo dos dispositivos. Con sólo dos cables I2C pueden admitir hasta 1008 dispositivos esclavos, y la mayoría de los dispositivos I2C pueden comunicarse a 100 kHz o 400 kHz, aunque por cada 8 bits de datos se debe transmitir 1 de reconocimiento (el bit ACK).

Las dos lineas del bus I2C son SCL y SDA. SCL es la señal del reloj, y SDA es la señal de datos, la señal de reloj siempre la genera el maestro de bus. La gran propiedad del bus I2C es que sus lineas son "Open-drain", lo que significa que el maestro o el esclavo pueden poner la línea de señal baja, pero ninguno de los dos puede ponerla a nivel alto. Cada línea de señal tiene una resistencia de pull-up, para restaurar la señal a alto cuando ningún dispositivo la quiere poner a nivel bajo.

La información se codifica dividiéndose en varios bloques.

1. **Condición de inicio**

Para iniciar la comunicación el dispositivo maestro deja SCL alto y baja SDA, esto indica a los esclavos que la transmisión va a empezar. Si dos dispositivos maestros desean ganar el control del bus, lo logra el dispositivo que primero baja el nivel de SDA.

2. **Dirección**

   La dirección siempre es lo primero que se envía al empezar una comunicación nueva, y permite seleccionar el esclavo. Las direcciones pueden ser de 7 bits o 10 bits. Nosotros usaremos direcciones de 7 bits, y primero se envía el bit más significativo (MSB), seguido por el bit R/W que indica si se trata de una operación de lectura (1) o de escritura (0). El noveno bit es el bit ACK para comprobar que ha sido recibido, después de enviar 8 bits le toca al receptor de datos manipular la linea SDA, si baja la línea SDA antes del noveno pulso de reloj se confirma que ha recibido el byte, de lo contrario el intercambio se detiene.

3. **Datos**

   Después de que se haya enviado la dirección, los datos pueden comenzar a transmitirse. El maestro simplemente continuará generando pulsos de reloj, y dependiendo del bit R/W los datos serán colocados en SDA por el maestro o el esclavo. No debemos tener en cuenta el número de bytes que queremos enviar, la mayoría de esclavos incrementarán automáticamente el registro para que las lecturas o escrituras posteriores se realicen en siguiente registro. Después de cada byte es necesario es necesario confirmar utilizando el ACK bit.

4. **Condición de parada**

   Una vez que se hayan enviado todos los datos, el maestro generará una condición de detención. Las condiciones de detención se definen por una transición de 0 a 1 en SDA después de una transición de 0 a 1 en SCL. Durante la transmisión de datos el valor en SDA no debe cambiar cuando SCL es alto, para evitar condiciones de parada falsas.

5. **Condición de reinicio**

   Sirve para que el maestro no ceda el control del bus y pueda cambiar de esclavo, o pueda cambiar de lectura a escritura o viceversa. Para realizarlo SDA cambia de 0 a 1 y después lo hace SCL (orden contrario a la condición de parada) y después

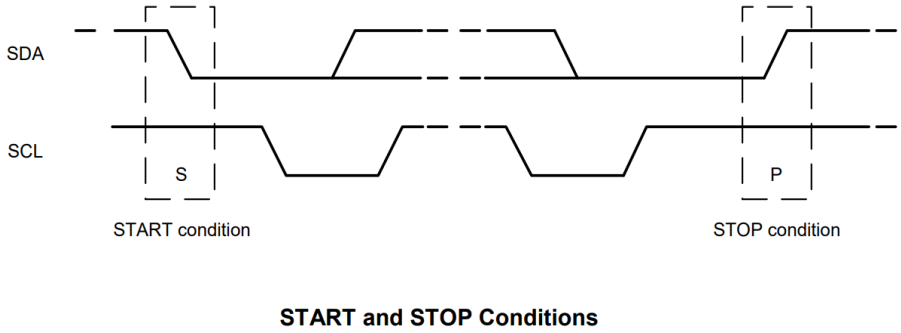

Figura 4.5: Condición de inicio y de parada.

se realiza una condición de inicio. Debido a que no se produce una condición de parada el maestro no pierde el control del bus. Debe ser seguido otra vez por la dirección y después por los datos.

Antes de comenzar a leer y escribir datos a través del I2C debemos saber cuando leer datos y cuando escribir, así que lo siguiente que haremos es consultar la documentación del MPU6050. El MPU6050 se comporta siempre como esclavo, por lo que nuestro PIC deberá ser el maestro, también puede ser programado al escribirle valores en los registros, en el giroscopio podemos elegir entre 250, 500, 1000 o 2000 grados por segundo, y en el acelerómetro entre 2, 4, 8 o 16 g. Elegiremos 250 dps y 2g.

La dirección del esclavo es de 7 bits y puede ser 0x68 o 0x69, ya que en binario es b110100x, siendo el valor de x el que tome la patilla AD0, pero debemos de llevar cuidado ya que no enviamos solo la dirección, los 7 bits van seguidos del bit R/W, por lo que en el caso de que x sea 0, enviamos b11010000 para escribir o b11010001 para leer, es decir 0xD0 o 0xD1. La IMU guarda los valores del acelerómetro y osciloscopio con 16 bits, en dos registros seguidos de 8 bits. Los valores de los tres ejes del acelerómetro y los tres ejes del giroscopio se encuentran seguidos, entre medias los dos registros de la temperatura, siendo el primero de ellos los 8 mayores bits del eje x del acelerómetro (ACCEL_XOUT_H, con dirección 0x3B), y como la dirección del registro se incrementa automáticamente cada vez que queramos tomar medidas podemos leer 14 veces seguidas sin cambiar de dirección.

La secuencia para escribir varios datos seguidos quedaría así:

- Maestro: Condición inicial.
- Maestro: Dirección del esclavo I2C + bit de escritura (0).

- Esclavo: bit ACK.
- Maestro: Enviar dirección del registro interno a escribir.
- Esclavo: ACK.
- Maestro: Dato.
- Esclavo: ACK.
- Maestro: Dato.
- Esclavo: ACK.
- ...
- Maestro: Condición de parada.

Y la secuencia para leer varios datos seguidos quedaría así:

- Maestro: Condición inicial.
- Maestro: Dirección del esclavo I2C + bit de escritura (0).
- Esclavo: bit ACK.
- Maestro: Enviar dirección del registro interno a leer.
- Esclavo: ACK.
- Maestro: Reinicio.
- Maestro: Dirección del esclavo I2C + bit de lectura (1).
- Esclavo: ACK.
- Maestro: Dato.
- Esclavo: ACK.
- ...
- Maestro: Dato.
- Esclavo: NACK.

- Maestro: Condición de parada.

Para crear la biblioteca I2C necesitamos utilizar 6 registros distintos: SSPCON1, SSPCON2, SSPADD, I2C, SSPSTAT y SSPBUF, nuevamente no necesitamos todos los bits. SSPCON1 y SSPCON2 son para la configuración y control del I2C, SSPADD para el Baud Rate, SSPSTAT para el estado de la comunicación, SSPBUF leer y escribir, y PIR1 para saber si llegan los bits.

1. Del registro SSPCON1 los bits que debemos configurar son:

   - SSPEN: Synchronous Serial Port Enable bit, habilita el puerto serie y configura los pines SDA y SCL. Para hacerlo debe ponerse a 1.
   - SSPM3:SSPM0: Synchronous Serial Port Mode Select bits, selecciona el modo de operación del I2C. Con el valor 1000 lo configuramos como maestro

2. Del registro SSPCON2 los bits que debemos configurar son:

   - ACKSTAT: Acknowledge Status bit, en una transmisión indica si el esclavo ha recibido el byte. 0 indica que se ha recibido.
   - ACKDT: Acknowledge Data bit, en una recepción indica que se ha recibido el byte. 0 indica que se ha recibido.
   - ACKEN: Acknowledge Sequence Enable bit, inicia la transmisión del bit ACKDT. Al ponerlo a 1 se realiza.
   - RCEN: Receive Enable bit, permite recibir datos. Al ponerlo a 1 se habilita.
   - PEN: Stop Condition Enable bit, bit para realizar la condición de parada. Al ponerlo a 1 comienza a realizarla.
   - RSEN: Repeated Start Condition Enabled bit, bit para realizar la condición de reinicio. Al ponerlo a 1 comienza a realizarla.
   - SEN: Start Condition Enabled bit, bit para realizar la condición de inicio. Al ponerlo a 1 comienza a realizarla.

3. En el registro SSPADD cuando el I2C es maestro los 7 menores dígitos sirven para generar el Baud Rate y en el modo esclavo sirve para guardar la dirección de éste. Su fórmula es:

$$FrecI2C = \frac{Fosc}{4(SSPADD + 1)}$$

Despejando:

$$SSPADD = \frac{Fosc}{4 \cdot FrecI2C}$$

Con los valores de Fosc = 16MHz y Frec I2C = 100 kHz obtenemos que en SSPADD debemos cargar el valor 39.

4. Del registro SSPSTAT los bits que debemos configurar son:

   - SMP: Slew Rate Control bit, selecciona la velocidad del I2C. Al ponerlo a 1 podemos elegir una velocidad de 100 kHz o 1 MHz.
   - P: Stop bit, indica que se ha detectado el bit de parada. En 1 indica que se ha detectado.
   - S: Start bit, indica que se ha detectado el bit de inicio. En 1 indica que se ha detectado.
   - R/W: Read/Write bit information, indica si hay una transmisión en proceso. En 1 indica que se está realizando.
   - BF: Buffer Full Status bit, indica el estado del buffer tanto para transmitir como para recibir. 1 indica que está lleno y 0 que está vacío.

5. Con el registro SSPBUF cargamos el byte a enviar o leemos el byte recibido.

6. Del registro PIR1 sólo necesitamos utilizar un bit:

   - SSPIF: Synchronous Serial Port Interrupt Flag bit, se pone a 1 cuando se produce una de estas condiciones:
     - Se ha realizado una transmisión o recepción.

- Se ha realizado la condición de inicio.
- Se ha realizado la condición de parada.
- Se ha realizado la condición de reinicio.
- Se ha realizado la comprobación del bit ACK.

La biblioteca para el I2C necesita 6 funciones, una para iniciar el I2C, una para la condición de inicio, otra para la condición de parada, dos para enviar y recibir datos y última para la condición de reinicio. La biblioteca MPU6050 necesitará 2 funciones, una para iniciar el MPU6050 y configurar sus registros y otra para empezar un proceso de lectura de la IMU.

Capítulo 5

Gestión del control

5.1. Introducción

En el Capítulo 4 hemos aprendido a utilizar el Timer, los PWM, la UART y el I2C, y hemos desarrollado las bibliotecas para hacerlas funcionar, además en el Capítulo 2 elegimos los componentes que vamos a necesitar para el prototipo y en el Capítulo 3 el microcontrolador que lo gobierna, así que ya tenemos todos los componentes para realizar. En este capítulo explicaremos el programa principal, que necesita estas bibliotecas.

El grueso del programa es tomar medidas y con ellas realizar un control y producir una salida adecuada, así que primero explicaremos estas dos etapas y después recapitularemos con la secuencia total que sigue el programa.

5.2. Medidas

Como nuestro proyecto tiene dos motores en un único eje sólo necesitamos leer el eje Y del acelerómetro y el X del giroscopio, estando el eje Y sobre el eje que forman los 2 motores. Con el I2C leemos los datos en binario, por lo que primero debemos escalar los 16 bits que leemos a las medidas del acelerómetro y giroscopio dividiendo entre la senibilidad. Para el giroscopio la sensibilidad es 131 y para el acelerómetro es de 16384, además en el acelerómetro convertimos las fuerzas G a inclinación, para ello aproximamos multiplicamos por 90.

También sería conveniente que las medidas se realicen en un intervalo de tiempo regular, por lo que hemos hecho pruebas del bucle de control y en una línea medimos

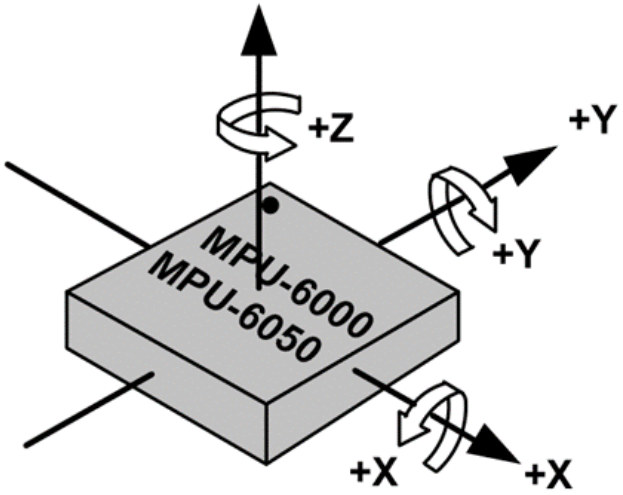

Figura 5.1: Ejes MPU6050.

cuanto tarda respecto al mismo instante en la ejecución anterior del bucle, tomando unas medidas vemos que el cada ciclo tarda un poco menso de 20ms, por lo que elegiremos un tiempo de muestreo de 20ms, al comienzo del bucle comprobaremos si con el timer hemos llegado a 20ms y si se cumple reiniciamos el timer y ejecutamos el código.

El giroscopio mide velocidad, pero nosotros queremos posición, así que como tenemos el tiempo entre medidas y la velocidad podemos reconstruir la posición. Hemos girado el balancín, en unos instantes con mayor amplitud y menor velocidad y otros de mayor velocidad y menor amplitud, tomando más de 2000 medidas para acelerómetro y giroscopio y las hemos enviado al ordenador.

Podemos observar que el acelerómetro toma medidas de cambios lentos con bastante precisión, sin embargo cuando producimos giros bruscos mide más inclinación de la que debería en realidad. El giroscopio no la mide bien los cambios lentos de posición ya que que utilizamos medidas discretas, y no medimos lo que realmente sucede entre cada medida, por lo que con el paso del tiempo vamos acumulando un error, sin embargo en cambios de posición rápidos tenemos mejor precisión que en el acelerómetro.

Es por ello que si queremos quedarnos con lo mejor de cada señal tenemos que filtrar ambas y sumarlas para obtener nuestra señal real de posición, al acelerómetro le aplicamos un filtro pasa bajos para eliminar los picos producidos por los cambios bruscos y al giroscopio le aplicamos un filtro pasa altos para eliminar el error y cambios lentos y quedarnos con los cambios bruscos.

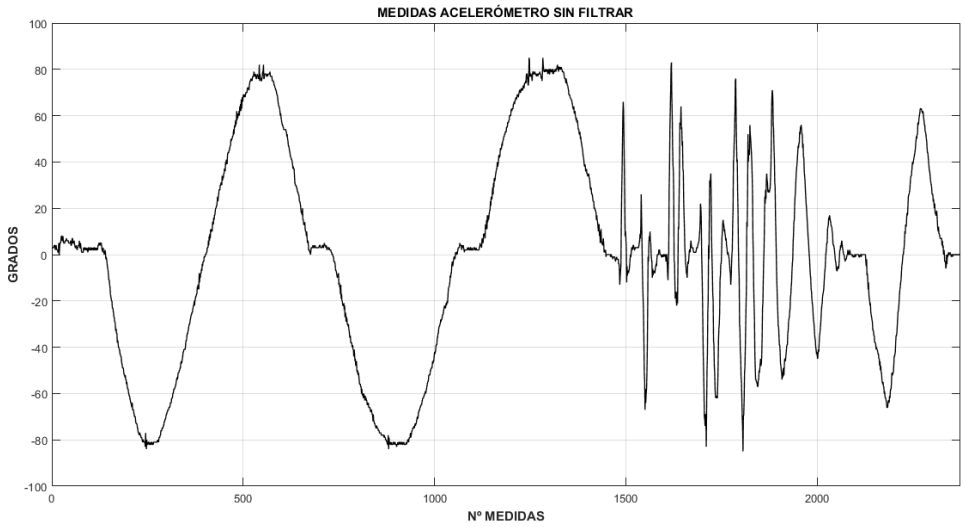

Figura 5.2: Medidas acelerómetro sin filtrar.

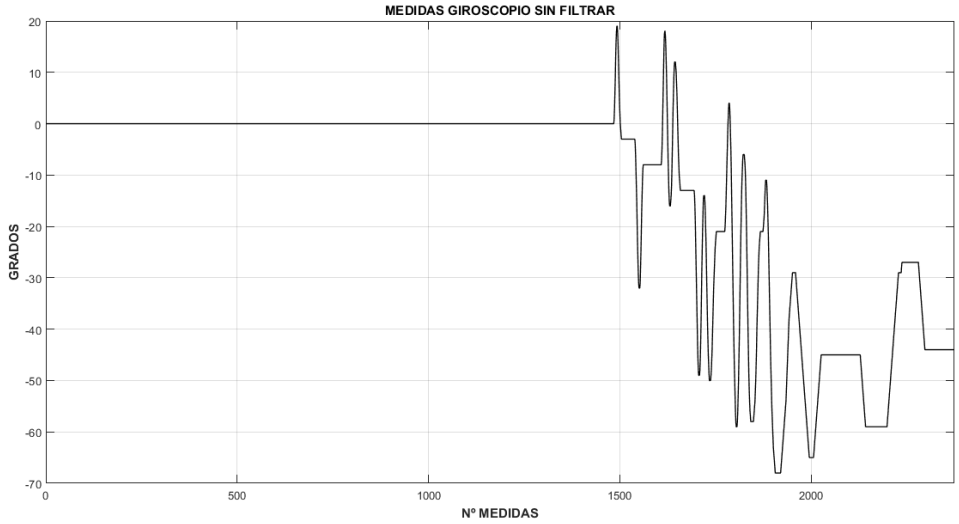

Figura 5.3: Medidas giroscopio sin filtar.

La ecuación para el filtro pasa bajos para el acelerómetro, siendo y[n] la medida filtrada en un instante n y x[n] la medida real en un instante n, es:

$$y[n] = (1 - \alpha) \cdot x[n] + \alpha \cdot y[n - 1]$$

Para realizar un filtro pasa altos para el giroscopio basta con utilizar la misma

ecuación de filtro pasa bajos y restarsela a la medida sin filtrar. El filtro depende de alfa para, después de varias pruebas lo estableceremos a 0.8.

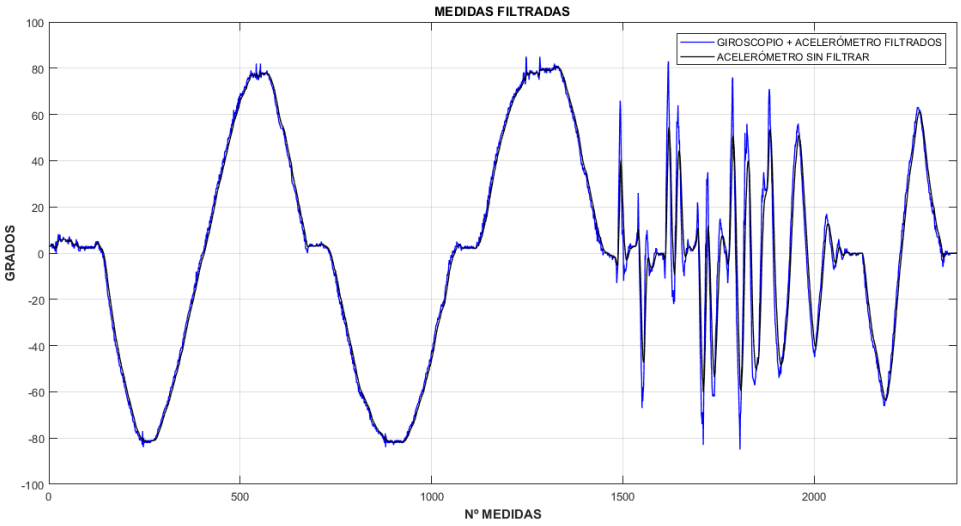

Figura 5.4: Medidas filtradas vs medidas acelerómetro.

Podemos observar como la señal filtrada es más suave pero sigue al acelerómetro bastante bien, y en los cambios bruscos la señal tiene menor amplitud, que es lo que buscábamos. La señal filtrada tiene un ligero retraso aceptable.

5.3. Control PID

Para el control de los motores hemos optado por el conocido control PID. Un controlador PID (Controlador Proporcional-Integral-Derivativo) calcula el error entre un valor medido y un valor deseado. Consta de tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor Proporcional depende del error actual. El Integral depende de los errores pasados y el Derivativo es una predicción de los errores futuros. La suma de estas tres acciones es usada para ajustar al proceso en el que aplicamos el control PID.

El control PID es el más indicado cuando no se tiene conocimiento del proceso o de las ecuaciones que controlan el proceso. Ajustando estas tres variables en el algoritmo de control del PID, el controlador puede proveer una acción de control específica para nuestro proceso. Algunas aplicaciones pueden solo requerir de una o dos variables del PID, siento entonces un controlador PI, PD, P o I. Su fórmula es:

$$U(t) = p + i + d = Kp \cdot e(t) + ki \cdot \int e(t) + kd \cdot \frac{de(t)}{dt}$$

Donde en nuestro caso el error la diferencia entre en angulo que queremos (lo enviamos desde el ordenador) y el medido (nos lo proporciona la IMU). Los parámetros Kp, Ki y Kd debemos ajustarlos varias veces hasta obtener los valores que hagan nuestro control preciso. La parte proporcional consiste en el producto entre el error y la constante proporcional Kp, cuanto mayor sea el error, mayor será la diferencia de velocidad entre ambos motores. En la mayoría de sistemas de control, el valor proporcional sólo es óptimos en un determinado rango, existiendo un valor límite en el que algunas veces el sistema alcanza valores superiores a los deseados.

La parte integral tiene como propósito disminuir y eliminar el error en estado estacionario (offset), actúa cuando hay una desviación entre la medida y la referencia, integra esta desviación en el tiempo y es multiplicada por la constante Ki. La acción derivativa se manifiesta cuando hay un cambio en el valor del error, la función de la acción derivativa es mantener el error al mínimo corrigiéndolo proporcionalmente con la misma velocidad que se produce, de esta manera evita que el error se incremente. Se deriva con respecto al tiempo y se multiplica por una constante Kd. La integral del error es igual a la suma de errores pasados y la derivada del error es la velocidad con la que se produce este error.

Debemos probar Kp, Ki y Kd con distintos valores para determinar los valores adecuados, el no depende sólo de la fuerza de los motores, ya que si los dos motores giran a la misma velocidad el momento angular total se anulará, así que nos tenemos que preocupar de crear una diferencia de velocidad entre ambos motores, para ello ambos motores tendrán una velocidad base, y la salida del PID será una variable que contiene una variación de velocidad (positiva o negativa), esta variación se le sumará a un motor y al motor se le restará, provocando una diferencia de velocidad.

5.4. Programa Microcontrolador

Ya sabemos cómo realizar el programa para el microcontrolador, recordamos que antes de iniciar el control debemos armar los motores, y el código en sí se puede ver en el anexo A. La secuencia del programa sería:

1. Iniciar Módulos.

2. Armado.
3. Tomar medidas de la IMU a través del I2C.
4. Escalar las medidas.
5. Filtrar las medidas.
6. Comparar valor obtenido con valor deseado.
7. Algoritmo PID.
8. Salida: PWM.
9. Enviar valores al ordenador a través de la UART.
10. Volver al punto 3.

Si en el punto 4 se produce algún error el timer lo detectará y abortará esta medida, volviendo al punto 3, y si enviamos algún dato desde el ordenador saltará la interrupción y el valor de la referencia se actualizará. Por seguridad si medimos una inclinación mayor de 50º y menor de -50º los motores se detendrán.

5.5. Programa Ordenador

Por último necesitaremos un programa en el ordenador para comunicarnos con el PIC, y poder leer las medidas que envía el PIC por la UART y poder cambiar la referencia. Para ello utilizaremos Processing, que es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java. Lo hemos elegido debido a que es sencillo de utilizar, está enfocado a hacer programas gráficos y proporciona muy buenos resultados, y dispone bibliotecas para enviar y recibir datos a través del bluetooth del ordenador, así que sólo debemos implementarlas en el programa y emparejar el HC-05 con el bluetooth del ordenador. Podemos ver el código de este programa en el Anexo C.

Capítulo 6

Resultados

6.1. Planos

Para construir el prototipo no hemos realizado un plano como tal de la estructura, hemos realizado un brazo para colocar los componentes y después un soporte para este brazo. Sin embargo debemos conocer la distancia entre los motores, ya que cuanto mayor sea, mayor será el par que generará cada motor, hemos utilizado como referencia el plano de la estructura F450 de la marca DJI, escogiendo la distancia menor entre los motores, redondeando a 380mm.

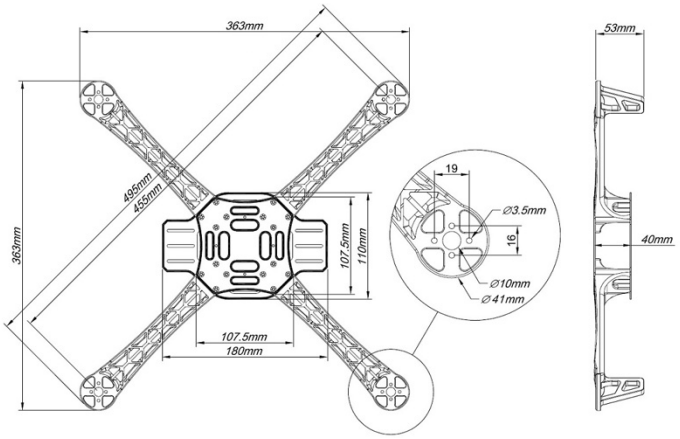

Figura 6.1: Estructura F450.

La protoboard la colocamos encima del brazo procurando que la IMU se encuentre lo más centrada posible y con un led que indique si nos encontramos en la fase de armado. El prototipo se alimentará con el programador, por lo que de la protoboard salen los cables de alimentación y los de las dos señales PWM. Podríamos haber co-

locado solamente la IMU y el resto de la protoboard fuera del brazo, pero esto tiene el inconveniente de que debemos aumentar la longitud de los cables entre el PIC y la IMU aumentando considerablemente los errores de medida.

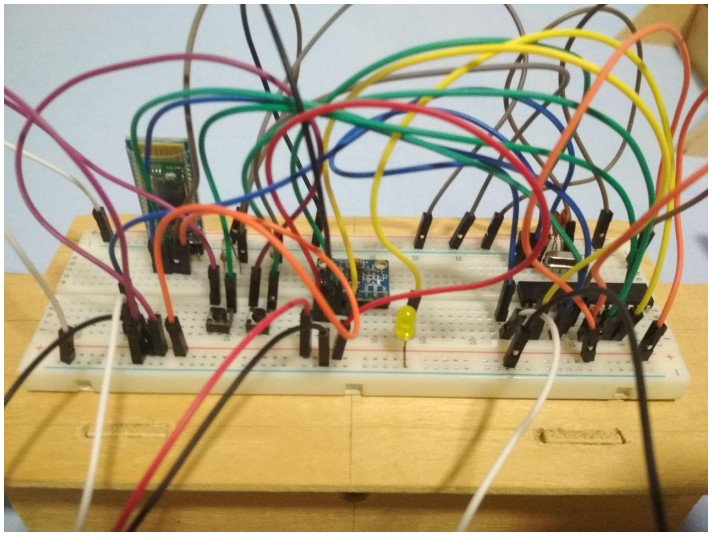

Figura 6.2: Protoboard.

Para colocar los motores en el brazo hemos utilizado los tornillos que disponía el motor, haciendo los agujeros siguiendo el plano de la figura 6.1, añadiendo unas arandelas para evitar dañar la madera y utilizando unos cables de alimentación suficientemente largos para que pueda girar. Además hemos necesitado un agujero central para evitar el roce del motor con la madera.

Además del brazo necesitamos un soporte que le permita girar, la unión entre el brazo y el soporte se realiza mediante un cilindro roscado que es el que le permite girar. Para esta unión el brazo necesita dos trozos de madera perpendicular al eje de los motores para que el cilindro lo atraviese. El prototipo final quedaría de la siguiente forma:

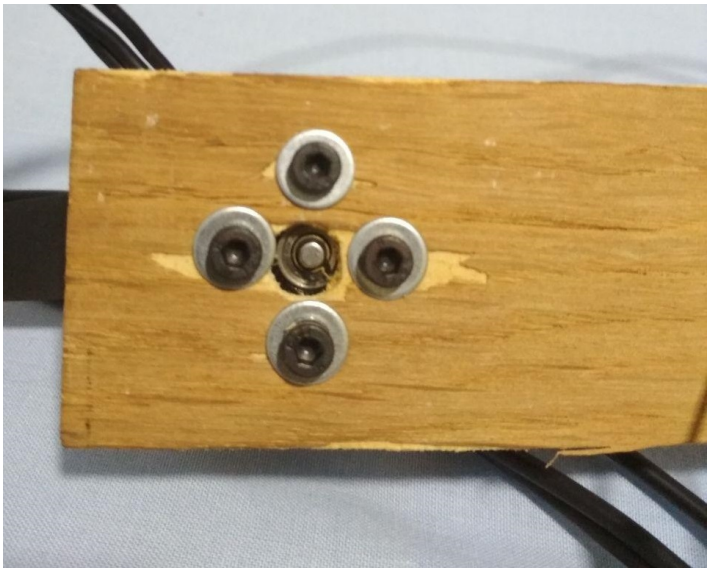

Figura 6.3: Unión motor-brazo.

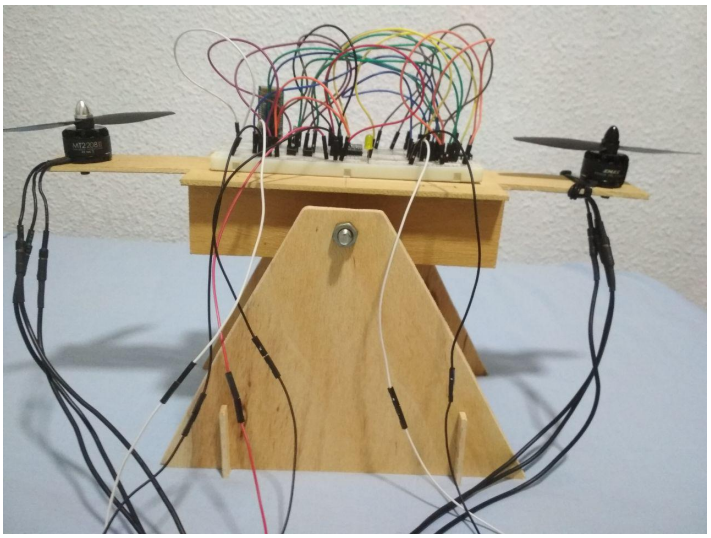

Figura 6.4: Prototipo final.

6.2. Interfaz gráfica

En cuanto al programa hemos preferido que muestre sólo la posición actual en la que se encuentra, pudiendo enviar una nueva referencia de consigna. Además indicamos que cuando sube el motor izquierdo y baja el derecho el ángulo de giro es positivo, en el caso contrario es negativo.

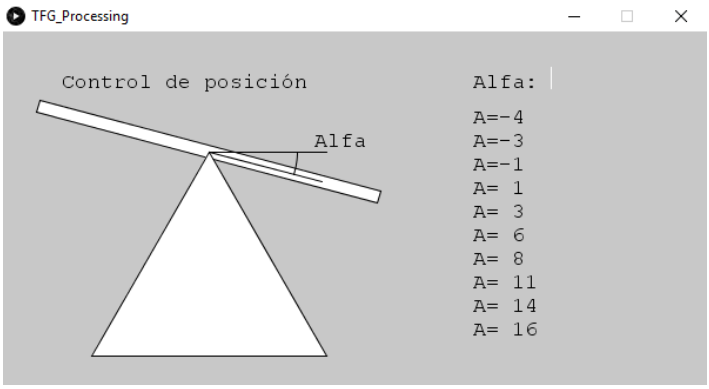

Figura 6.5: Programa realizado con Processing.

6.3. Respuesta de control

El programa se comporta correctamente, respondiendo ante perturbaciones y cambios de referencias, sin embargo el ciclo de control conlleva una duración de casi 20ms, y el filtro no consigue eliminar pequeñas variaciones lentas, por lo que el algoritmo tiene una histéresis en pocos grados en torno al valor de consigna, y no podemos implementar un filtro más potente si queremos que el periodo de nuestro algoritmo es de 20ms. Pese a esto la respuesta es satisfactoria y más si tenemos en cuenta que utilizamos un PIC de 8 bits, bastaría con aumentar la potencia del microcontrolador y mejorar el filtro para mejorar la salida.

Capítulo 7

Conclusiones

Una vez realizado el desarrollo, podemos concluir que, para realizar un sistema de control para un UAV, no es necesario utilizar microcontroladores sofisticados si lo que se desea es solamente controlar su vuelo. Otro asunto es cuando se le incorporan otros dispositivos como cámaras, sensores de calor, y todo tipo de detectores que al usuario se le ocurra. En ese caso siempre sería conveniente utilizar controladores específicos independientes del sistema de control de vuelo.

Una de las características del trabajo es que nos ha permitido establecer una base de código abierto (open source), que partiendo de la comunicación serie, es fácilmente aplicable a otros sistemas de transmisión como Bluetooth, WIFI, señales RF, etc. El futuro usuario sólo necesita incorporar las "funciones" o "rutinas" adecuadas para cada tecnología de comunicación.

Por último, destacar que aunque los cálculos se han realizado para dos motores, lo interesante es que la estructura y el control es "replicable", lo que significa que con pequeñas modificaciones del código y añadiendo los componentes necesarios, podríamos aplicarlo a un número par de motores, con los únicos límites que impone el consumo y el peso del dispositivo.

7.1. Vías futuras

Para adaptar este prototipo a un cuadricóptero no es necesario aumentar la potencia de cómputo del PIC, aunque sería interesante utilizar un dsPIC para esta tarea. En cualquier caso debemos hacer una serie de modificaciones en el prototipo:

1. Utilizar un PIC con 4 salidas PWM.
2. Utilizar realizar el control en un segundo eje para aumentar sus grados de libertad.
3. Adaptar el programa de ordenador o utilizar otro sistema para enviar datos.
4. No limitar la potencia de los motores sólo al control, ya que es necesaria para ganar altura.
5. Crear funciones para el movimiento del cuadricóptero en todas las dimensiones.
6. Implementar un conversor CC/CC para alimentar el PIC con la batería de los motores.
7. Implementar un PCB y una estructura para disminuir el peso y aumentar la eficiencia del sistema.

Apéndice A

Código Principal

A.1. main.c

```
#include "configuration.h"

#include "uart.h"
#include "i2c.h"
#include "mpu6050.h"
#include "timer.h"
#include "pwm.h"

void uart();

char skip=0;

int Ay=0,Gx=0,Ay1=0,Gx1=0,Ay3=0,Gx3=0;
char Ay2=0,Gx2=0;

int Ab,Gb,Gh;

int s1 = 600, s2;

int T=0;
int angle, ref = 0, error, error2, ctrl;
```

```
float p, i, d;
float kp = 0.45, ki = 0.5, kd = 0.1;

char stringRef[] = "0000000";
char stringSend[] = "0000000";

void main(void){
    TRISB = 0x03; //RB0 y RB1 entradas
    PORTB = 0x04; //RB2 = 1;

    uartInit(38400);

     //el armado
    while(!RB0);
    RB2 = 0;

    pwm1Init(4000);
    pwm2Init(4000);

    while(s1<650){
        if(RB0 == 1){
            s1 += 1;
            __delay_ms(10);
            pwm1Duty(s1);
            pwm2Duty(s1);
            uartWriteStr("Armado = ");
            numberToStr(stringSend, s1);
            uartWriteStr(stringSend);
            uartWrite(0x0A);
        }
    }

    __delay_ms(500);
    RB2 = 1;

    while(s1>511){
```

```
        if(RB1 == 1){
            s1 -= 1;
            __delay_ms(10);
            pwm1Duty(s1);
            pwm2Duty(s1);
            uartWriteStr("Armado = ");
            numberToStr(stringSend, s1);
            uartWriteStr(stringSend);
            uartWrite(0x0A);
        }
    }

    __delay_ms(500);

    timerInit();
        mpu6050Init();

    while(1){
        if(timerus() >= 20000){
            T = timerus();
            timerusRestart();
            RB2 = 0;

            skip = mpuStart();
            if(skip){
                i2cStop();
            }
            else{
                i2cRead(0);
                i2cRead(0);
                Ay1 = i2cRead(0);
                Ay2 = i2cRead(0);
                i2cRead(0);
                i2cRead(0);
                i2cRead(0);
                i2cRead(0);
```

```
Gx1 = i2cRead(0);
Gx2 = i2cRead(0);
i2cRead(0);
i2cRead(0);
i2cRead(0);
i2cRead(1);
i2cStop();

Ay3 = (Ay1<<8) | (int)Ay2;
Gx3 = (Gx1<<8) | (int)Gx2;
Ay = (float)Ay3*90/16384.0;
Gx3 = Gx3/131.0;
Gx += Gx3*0.02;

//filtrado
Ab = Ay*0.2 + Ab*0.8;
Gb = Gx*0.2 + Gb*0.8;
Gh = Gx-Gb;

angle = Ab + Gh;

error = ref - angle;
p = kp * error;
i += ki * error * 0.02;
d = kd * (error-error2)*50;
error2 = error;
ctrl = p + i + d;
if (ctrl > 50)
    ctrl = 50;
else if (ctrl < -50)
    ctrl = -50;
s1 = 650 - ctrl;
s2 = 650 + ctrl;

if(Ay < -50 && Ay > 50)
{
```

```
                    s1 = 511;
                    s2 = 511;
                }

                pwm1Duty(s1);
                pwm2Duty(s2);

                uart();
            }
        }
    }
}

void __interrupt() inter(){
    if(RCIF==1){
        uartReadStr(stringRef, 3);
        ref = strToNumber(stringRef, 2);
    }
    if(TMR0IF){
        TMR0IF = 0;
        TMR0 = 156;
        timerCounter();
    }
}

void uart(){
    uartWriteStr("A=");
    numberToStr(stringSend, angle);
    uartWriteStr(stringSend);
    uartWrite(0x0A);
}
```

Apéndice B

Bibliotecas

B.1. configuration.h

```
#ifndef CONFIGURATION_H
#define CONFIGURATION_H

#define _XTAL_FREQ 16000000
#include <xc.h>

#pragma config CP = OFF
#pragma config DEBUG = OFF
#pragma config WRT = OFF
#pragma config CPD = OFF
#pragma config LVP = OFF
#pragma config BOREN = OFF
#pragma config PWRTE = OFF
#pragma config WDTE = OFF
#pragma config FOSC = HS

#endif
```

B.2. timer.h

```
#ifndef TIMER_H
#define TIMER_H

int tus,t20=0;

void timerInit();
int timerus();
int timer20ms();
void timerCounter();
void timerusRestart();
void timer20msRestart();

#endif
```

B.3. timer.c

```
#include "timer.h"
#include "configuration.h"

void timerInit(){
    T0CS = 0;
    PSA = 0;
    PS2 = 0;
    PS1 = 1;
    PS0 = 0;
    GIE = 1;
    TMR0IE = 1;
    TMR0IF = 0;
    TMR0 = 156;
}

int timerus(){
    int T = 200*tus;
    return T;
}

int timer20ms(){
    if (t20>=100){
        return 1;
    }
    else
        return 0;
}

void timerCounter(){
    tus++;
    t20++;
}
```

```
void timerusRestart(){
    tus = 0;
}

void timer20msRestart(){
    t20 = 0;
}
```

B.4. pwm.h

```
#ifndef PWMH
#define PWMH

#define TMR2PRESCALE 4

void pwm1Init(long);
void pwm2Init(long);
void pwm1Duty(unsigned int);
void pwm2Duty(unsigned int);
void pwm1Stop();
void pwm2Stop();

#endif
```

B.5. pwm.c

```
#include "pwm.h"
#include "configuration.h"

int maxDuty;

void pwm1Init(long freq){
    CCP1M3 = 1;
    CCP1M2 = 1;
    #if TMR2PRESCALE == 1
        T2CKPS0 = 0;
        T2CKPS1 = 0;
    #elif TMR2PRESCALE == 4
        T2CKPS0 = 1;
        T2CKPS1 = 0;
    #elif TMR2PRESCALE == 16
        T2CKPS0 = 1;
        T2CKPS1 = 1;
    #endif
    TMR2ON = 1;
    TRISC2 = 0;
    PR2 = (_XTAL_FREQ/(freq*4*TMR2PRESCALE)) - 1;
    maxDuty = _XTAL_FREQ/(freq*TMR2PRESCALE);
}

void pwm2Init(long freq){
    CCP2M3 = 1;
    CCP2M2 = 1;
    #if TMR2PRESCALE == 1
        T2CKPS0 = 0;
        T2CKPS1 = 0;
    #elif TMR2PRESCALE == 4
        T2CKPS0 = 1;
        T2CKPS1 = 0;
```

```
    #elif TMR2PRESCALE == 16
        T2CKPS0 = 1;
        T2CKPS1 = 1;
    #endif
        TMR2ON = 1;
        TRISC1 = 0;
    PR2 = (_XTAL_FREQ/(freq*4*TMR2PRESCALE)) - 1;
    maxDuty = _XTAL_FREQ/(freq*TMR2PRESCALE);
}

void pwm1Duty(unsigned int duty){
  if(duty < 1024){
    duty = ((float)duty/1024)*maxDuty;
    CCP1X = duty & 2;
    CCP1Y = duty & 1;
    CCPR1L = duty>>2;
  }
}

void pwm2Duty(unsigned int duty){
  if(duty < 1024){
    duty = ((float)duty/1024)*maxDuty;
    CCP2X = duty & 2;
    CCP2Y = duty & 1;
    CCPR2L = duty>>2;
  }
}

void pwm1Stop(){
  CCP1M3 = 0;
  CCP1M2 = 0;
}
void pwm2Stop(){
  CCP2M3 = 0;
  CCP2M2 = 0;
}
```

B.6. uart.h

```
#ifndef UART_H
#define UART_H

void uartInit(long);
char uartRead();
void uartWrite(char);
void uartReadStr(char*, unsigned int);
void uartWriteStr(char*);
int charToNumber(char);
int strToNumber(char*, int length);
char numberToChar(int);
void numberToStr(char*, int);

#endif
```

B.7. uart.c

```
#include "uart.h"
#include "configuration.h"

void uartInit(long baudrate){
    TRISC7 = 1;       //RX
    TRISC6 = 1;       //TX
    BRGH = 1;
    SPBRG = (_XTAL_FREQ - baudrate*16)/(baudrate*16);
    SYNC = 0;
    SPEN = 1;
    CREN = 1;
    TXEN = 1;

    GIE = 1;
    PEIE = 1;
    RCIE = 1;
}

char uartRead(){
    while(!RCIF);
    return RCREG;
}

void uartWrite(char data){
    while(!TRMT);
    TXREG = data;
}

void uartReadStr(char *str, unsigned int length){
    unsigned int i;
    for(int i=0;i<length;i++)
        str[i] = uartRead();
    str[length] = '\0';
```

```
}

void uartWriteStr(char *str){
    int i;
    for(i=0;str[i]!='\0';i++){
        uartWrite(str[i]);
    }
}

int charToNumber(char str){
    switch(str){
        case '1': return 1;
        break;
        case '2': return 2;
        break;
        case '3': return 3;
        break;
        case '4': return 4;
        break;
        case '5': return 5;
        break;
        case '6': return 6;
        break;
        case '7': return 7;
        break;
        case '8': return 8;
        break;
        case '9': return 9;
        break;
        case '0': return 0;
        break;
    }
}

int strToNumber(char *str, int length){
    unsigned char i;
```

```
    int x = 0;
    for(i=1;i<=length;i++)
        x = x*10 + charToNumber(str[i]);
    if(str[0] == '-')
        x = -x;
    return x;
}

char numberToChar(int number){
    switch(number){
        case 1: return '1';
        break;
        case 2: return '2';
        break;
        case 3: return '3';
        break;
        case 4: return '4';
        break;
        case 5: return '5';
        break;
        case 6: return '6';
        break;
        case 7: return '7';
        break;
        case 8: return '8';
        break;
        case 9: return '9';
        break;
        case 0: return '0';
        break;
    }
}

void numberToStr(char *str, int number){
    unsigned char i,j=0;
    int n = number;
```

```
    if(number<0){
        str[0] = '-';
        number = -number;
        n = -n;
    }
    else
        str[0] = ' ';
    while(n > 0){
        n = n/10;
        j++;
    }
    if(number == 0)
        j=1;
    for(i = j; i > 0 ;i--){
        str[i] = numberToChar(number%10);
        number = number/10;
    }
    str[j+1] = '\0';
}
```

B.8. i2c.h

```
#ifndef I2C_H
#define I2C_H

#define I2C_CLOCK 100000

void i2cInit();
void i2cStart(char);
char i2cStop();
char i2cWrite(unsigned char);
char i2cRestart(char);
char i2cRead(char);

#endif
```

B.9. i2c.c

```
#include "i2c.h"
#include "configuration.h"
#include "timer.h"

#include "uart.h"

void i2cInit(){
    TRISC4 = 1;       //SDA
    TRISC3 = 1;       //SCl
    SMP = 1;
    SSPEN = 1;
    SSPM3 = 1;
    SSPM2 = 0;
    SSPM1 = 0;
    SSPM0 = 0;
    SSPADD = ((_XTAL_FREQ/(4*I2C_CLOCK))-1);
    SSPIF = 0;
}

void i2cStart(char slave_write_address){
  while(1){
    SEN = 1;
    while(SEN);
    SSPIF = 0;
    if(!SSPSTATbits.S)
        continue;
    i2cWrite(slave_write_address);
    if(ACKSTAT){
        i2cStop();
        continue;
    }
    break;
  }
```

```
}

char i2cStop(){
    PEN = 1;
    while(PEN);
    SSPIF = 0;
    if(!SSPSTATbits.P)
        return 0;
}

char i2cWrite(unsigned char data){
    SSPBUF = data;
    timer20msRestart();
    while(!SSPIF && !timer20ms());
    if(timer20ms()){
        //uartWriteStr("Cuidado!\n");
        RB2 = 1;
        return 0;
    }
    SSPIF=0;
    if (ACKSTAT) /* Devuelve 1 si ha llegado, si no 0*/
        return 0;
    else
        return 1;
}

char i2cRestart(char slave_read_address){
    RSEN = 1;
    while(RSEN);
    SSPIF = 0;
    if(!SSPSTATbits.S)
        return 0;
    i2cWrite(slave_read_address);
    if (ACKSTAT) /* Devuelve 1 si ha llegado, si no 0*/
        return 0;
    else
```

```
        return 1;
}

char i2cRead(char flag){ /* 0 = Ack, 1 = Nack */
    char buffer;
    RCEN = 1;
    while(!SSPSTATbits.BF);
    buffer = SSPBUF;
    if(flag==0){
        ACKDT = 0;
        ACKEN = 1;
        while(ACKEN);
    }
    else{
        ACKDT = 1;
        ACKEN = 1;
        while(ACKEN);
    }
    while(!SSPIF);
    SSPIF=0;
    return(buffer);
}
```

B.10. mpu6050.h

```
#ifndef MPU6050_H
#define MPU6050_H

#define SMPLRT_DIV 0x19
#define CONFIG 0x1A
#define GYRO_CONFIG 0x1B
#define PWR_MGMT_1 0x6B
#define INT_ENABLE 0x38

#define ACCEL_XOUT_H 0x3B
#define ACCEL_XOUT_L 0x3C
#define ACCEL_YOUT_H 0x3D
#define ACCEL_YOUT_L 0x3E
#define ACCEL_ZOUT_H 0x3F
#define ACCEL_ZOUT_L 0x40
#define TEMP_OUT_H 0x41
#define TEMP_OUT_L 0x42
#define GYRO_XOUT_H 0x43
#define GYRO_XOUT_L 0x44
#define GYRO_YOUT_H 0x45
#define GYRO_YOUT_L 0x46
#define GYRO_ZOUT_H 0x47
#define GYRO_ZOUT_L 0x48

void mpu6050Init();
char mpuStart();

#endif
```

B.11. mpu6050.c

```
#include "mpu6050.h"
#include "i2c.h"
#include "configuration.h"

void mpu6050Init(){
    i2cInit();
    __delay_ms(200);
    i2cStart(0xD0);
    i2cWrite(SMPLRT_DIV);
    i2cWrite(0x07);
    i2cStop();

    i2cStart(0xD0);
    i2cWrite(PWR_MGMT_1);
    i2cWrite(0x01);
    i2cStop();

    i2cStart(0xD0);
    i2cWrite(CONFIG);
    i2cWrite(0x00);
    i2cStop();

    i2cStart(0xD0);
    i2cWrite(GYRO_CONFIG);
    //i2cWrite(0x18);
    i2cWrite(0x00);
    i2cStop();

    i2cStart(0xD0);
    i2cWrite(INT_ENABLE);
    i2cWrite(0x01);
    i2cStop();
}
```

```
char mpuStart(){
    char x;
    i2cStart(0xD0);
    x = i2cWrite(ACCEL_XOUT_H);
    if(!x) /* Devuelve 1 si falla */
        return 1;
    x = i2cRestart(0xD1);
    if(!x) /* Devuelve 1 si falla, si no 0*/
        return 1;
    else
        return 0;
}
```

Apéndice C

Programa Ordenador

C.1. Processing

```
import processing.serial.*;
String letters = "";
String uuu, uuu2;

String[] uart = {"", "", "", "", "", "", "", "", "", ""};
String Angle = "";


Serial port;

void setup(){
  size(1000,300);
  //size(1200,300);
  PFont font; // Declare the variable
  font = loadFont("Monospaced.plain-18.vlw"); // Load the font
  textFont(font); // Set the current text font
  port = new Serial(this, Serial.list()[2], 38400);

}

void draw() {
```

```
  background(200);
  balancin();
  stroke(255);
  fill(0);
  texto();
  salida(uart);
  if ( port.available() > 0)
  {
    uuu = port.readStringUntil('\n');
    if (uuu != null) {
      if(uuu.charAt(1)=='='){
        Angle += uuu.substring(2, uuu.length()-1);
        Angle += ' ';
        uuu2 = "Angle = " + uuu.substring(2, uuu.length()-1);
        ordenar(uart);
        println(uuu2);
      }
      else{
        uuu2 = uuu;
        ordenar(uart);
        print(uuu2);
      }
    }
  }
}

void keyPressed() {
  if (key == 'T') {
    println("---Angle---");
    println(Angle);
  }
  if (key == BACKSPACE) {
    if (letters.length() > 0) {
      letters = letters.substring(0, letters.length()-1);
    }
  }
```

```
  else if ((letters.length() == 0) && (key == '+' || key == '-')){
    letters = letters+key;
  }
  else if ((letters.length() < 2) && (key <= 57 && key >= 48)){
    letters = letters+key;
  }
  else if (key <= 57 && key >= 48){
    if((letters.charAt(0) == '+' || letters.charAt(0) == '-') && letters.le
    letters = letters+key;
  }



  else if(key == ENTER){
    if(letters.length() == 0){
      letters = "0";
    }
    if(letters.charAt(0) != '-' && letters.charAt(0) != '+'){
      letters = '+'+letters;
    }
    while(letters.length() < 3) {
      letters = letters.substring(0,1)+0+letters.substring(1, letters.lengt
    }
    port.write(letters);
  }
}

void balancin(){
  stroke(0); //lineas negras
  fill(255);
  float x1 = 175 + 5*sin(PI/12) - 150*cos(PI/12);
  float y1 = 102 - 5*cos(PI/12) - 150*sin(PI/12);
  float x2 = 175 + 5*sin(PI/12) + 150*cos(PI/12);
  float y2 = 102 - 5*cos(PI/12) + 150*sin(PI/12);
  float x3 = 175 - 5*sin(PI/12) + 150*cos(PI/12);
  float y3 = 102 + 5*cos(PI/12) + 150*sin(PI/12);
  float x4 = 175 - 5*sin(PI/12) - 150*cos(PI/12);
```

```
  float y4 = 102 + 5*cos(PI/12) - 150*sin(PI/12);
  float x5 = 175 + 100*cos(PI/12);
  float y5 = 102 + 100*sin(PI/12);
  quad(x1,y1,x2,y2,x3,y3,x4,y4);
  triangle(75,275,275,275,175,102);
  fill(255,0);
  arc(175,102,150,150,0,PI/12);
  fill(0);
  text("Alfa", 265, 98);
  line(175, 102, 275, 102);
  line(175, 102, x5, y5);
}

void texto(){
  text("Control de posicion", 50, 48);
  text("Alfa: ", 400, 48);
  float cursorPosition = textWidth(letters) + textWidth("Alfa: ");
  line(cursorPosition + 400, 30, cursorPosition + 400, 48);
  text(letters, textWidth("Alfa: ") + 400, 48);
}

void salida(String[] uart){
  for(int i = 0; i < uart.length; i++){
    text(uart[i], 400, 78+20*i);
  }
}

void ordenar(String[] uart){
  for(int i = (uart.length - 1); i > 0; i--){
    uart[i] = uart[i-1];
  }
  uart[0] = uuu2;
}
```

Bibliografía

[1] Alfonso José Luna Alcolea, Trabajo de la signatura Microrobótica: Unidad de medición inercial (IMU) MPU6050.

[2] https://www.scribbr.es/category/estructura/

[3] http://picguides.com/beginner/introduction.php

[4] http://microchipdeveloper.com/

[5] https://github.com/jrowberg/i2cdevlib

[6] https://electrosome.com/led-pic-microcontroller-mplab-xc8/

[7] https://es.wikipedia.org/wiki/Seis_grados_de_libertad

[8] https://filmora.wondershare.com/drones/drone-applications-and-uses-in-future.html

[9] http://beginnerflyer.com/build-a-drone/

[10] https://es.wikipedia.org/wiki/Motor_el%C3%A9ctrico

[11] http://www.areatecnologia.com/electricidad/tipos-de-motores-electricos.html

[12] https://github.com/bitdump/BLHeli/tree/master/BLHeli_S%20SiLabs

[13] http://fvt-littlebee.com/#features

[14] https://quadmeup.com/pwm-oneshot125-oneshot42-and-multishot-comparison/

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

[16] https://fpvfrenzy.com/quadcopter-flight-controller-shootout/

[17] http://www.electronicwings.com/pic/pic18f4550-pwm

[18] https://electrosome.com/pwm-pic-microcontroller-mplab-xc8/

[19] https://learn.sparkfun.com/tutorials/serial-communication/rules-of-serial

[20] https://learn.sparkfun.com/tutorials/serial-communication/wiring-and-hardware

[21] https://learn.sparkfun.com/tutorials/i2c

[22] https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

[23] https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdfhttps://learn.sparkfun.com/tutorials/i2c

[24] http://www.electronicwings.com/pic/pic18f4550-i2c

[25] https://www.luisllamas.es/arduino-paso-bajo-exponencial/

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

[27] https://processing.org/

[28] https://learn.sparkfun.com/tutorials/connecting-arduino-to-processing