CONTROL REMOTO DE UN DRONE

PROYECTO FINAL DE GRADO EN SISTEMAS AUDIOVISUALES
ESCOLA D'ENGINYERIA DE TERRASSA (EET), UPC

2015-2016

AUTOR: JESÚS MARIO CALLEJA HERNÁNDEZ
TUTOR: JUAN MON GONZÁLEZ

AGRADECIMIENTOS

Me gustaría mostrar mi más sincero agradecimiento al tutor del proyecto, Juan Mon, por involucrarse tanto en el desarrollo de éste, trabajando en la investigación, resolviendo dudas y teniendo una actitud proactiva que ha sido clave en el desarrollo del proyecto. También quiero agradecer el apoyo y paciencia que me han ofrecido mis familiares y mi pareja durante todos estos meses.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

RESUMEN

En este proyecto se presenta una alternativa al control remoto convencional de un drone cuadricóptero. En este caso, se ha desarrollado un sistema que permite controlar un drone mediante el movimiento de una sola mano.

La idea se basa en poder controlar el cuadricóptero con los movimientos naturales de la mano. Si se quiere elevar el drone, se debe elevar la mano. Así mismo, para rotar el drone, rotar la mano. Y por último, si se desea inclinar el drone, simplemente se debe inclinar la mano.

Para poder capturar estos movimientos se requiere de una unidad de medición inercial o IMU (la cual debe incorporar como mínimo un acelerómetro y magnetómetro de tres ejes) y un sensor de ultrasonidos. Añadir también, que para gestionar estos periféricos y otros adicionales se necesita un microcontrolador.

Por último se diseña una placa de circuito impreso o PCB donde conectar el microcontrolador con los diferentes periféricos necesarios. También se ha diseñado y fabricado un chasis donde se aloja todo el sistema.

Palabras clave: control remoto, drone, ultrasonido, acelerómetro, arduino

1. INTRODUCCIÓN

1.1. Objetivos

El trabajo presente tiene como propósito controlar un drone cuadricóptero mediante el movimiento de la mano, reemplazando así la utilización de un mando de control convencional. El hándicap principal es conseguir que éste reproduzca en el aire un movimiento idéntico al de la extremidad en cuestión.

La elección del tema de este proyecto surge de la convergencia de varios motivos. En primer lugar, he de decir que desde la infancia he sentido una gran pasión por el tema del aeromodelismo (sobre todo del ámbito de los helicópteros) que en parte fue transmitida por mi padre, el cual estaba involucrado en este tema desde su adolescencia. Eso me ha llevado hasta día de hoy a obtener varios modelos de helicópteros y drones, así como a estar a la vanguardia de todas las innovaciones que aparecen en torno a este tema. En segundo lugar, tengo que decir que estoy mucho más familiarizado con los drones desde que hace pocos años se han vuelto más accesibles a nivel usuario, de forma que es mucho más fácil comprarlos y/o probarlos a un precio razonable. También ha contribuido a la elección de esta temática el hecho de que el semestre pasado realicé

una asignatura en la universidad (Sistemas Electrónicos Musicales Interactivos) que me cautivó, y a raíz de estudiar el entorno de desarrollo Arduino me di cuenta de que me gustaba mucho cómo funcionaba. Por último, y no menos importante, me percaté de que muchos de mis amigos tenían dificultad para pilotar los drones, quizás porque el hecho de controlar dos *joysticks* a la vez y que cada uno de estos se mueva en dos direcciones, siendo cada dirección un parámetro diferente a controlar, requiere bastante entrenamiento. Todas estas razones son las que me han impulsado a llevar a cabo este proyecto con mucha ilusión y muchas ganas.

1.2. Estado del arte

Si bien es cierto que no existe ningún trabajo en el que se haya desarrollado un control remoto como el que se trata en este proyecto, es importante hacer mención de dos autores que han hecho investigaciones en relación a esta temática. Por un lado, el blogger DZL [1] publicó en 2013 el código para hacer posible la conexión con el drone *Sky Walker* mediante el microcontrolador Arduino. Por otro lado, en el blog de Sebastian Brandes se explicó en 2014 cómo un equipo de investigadores consiguieron conectar el *AR Drone* al SDK de *Kinect* para Windows via wifi y controlar el cuadricóptero mediante los movimientos de las manos procesados por *Kinect* [2]. El inconveniente principal es que es necesario

tener un ordenador con Windows cerca y el accesorio *Kinect*, de manera que para llevárselo donde se requiera es poco práctico y se necesitan las dos manos para controlarlo, además de que los movimientos están pautados por una serie de patrones.

Justamente en ese sentido el trabajo precedente intenta cubrir mucho mejor algunas carencias evidentes en los dos anteriores. Puesto que queremos hacer mucho más fácil el pilotaje del drone, se ofrece la posibilidad de controlarlo mediante una sola mano, de forma que es mucho más práctico y cómodo, además de que el control es mediante movimientos nativos de la mano, mucho más intuitivo que seguir unos patrones establecidos. Cabe añadir que no es necesaria la utilización de un ordenador, sino que el proceso es simplemente encender el drone y ya automáticamente pilotarlo.

1.3. Estructura del proyecto

A continuación se comenta la aportación de cada uno de los siguientes capítulos del proyecto:

En el **capítulo 2** se detalla el diseño del sistema, incluyendo la conexión con el drone, el establecimiento del ángulo de rotación de la mano (*pitch*, *roll* y *yaw*), y por último cómo se establece la velocidad del drone (*throttle*) mediante el sensor de ultrasonidos.

En el **capítulo 3** se aborda con más detalle la implementación del software del sistema.

En el **capítulo 4** se hace hincapié en la implementación hardware del sistema, incluyendo el sistema de alimentación, el diseño de la PCB (Printed Circuit Board) y el chasis que alberga todo el sistema.

En el **capítulo 5** para finalizar con el desarrollo del proyecto se presentan las conclusiones y trabajos futuros.

2. DISEÑO DEL SISTEMA

Tal y como se ha especificado en la parte introductoria de este trabajo, la idea del proyecto es crear un sistema que sustituya una emisora convencional, de manera que éste funcione interpretando los gestos naturales de la mano y transformándolos en órdenes enviadas al drone, facilitando la curva de aprendizaje del pilotaje de éste.

Para ello, necesitamos poder controlar la altura de la mano y la rotación de ésta en los ejes X, Y y Z. Además, debemos poder conectarnos al drone y gestionar todos los periféricos y órdenes mediante un microcontrolador.

Los componentes electrónicos necesarios para la implementación del sistema de control remoto son:

- Un **microcontrolador** que actúe como cerebro del sistema.
- Una **IMU (Inertial Measurement Unit)** que incorpore un acelerómetro y un magnetómetro, necesarios para poder establecer la rotación de la mano.
- Un **sensor de ultrasonidos** para fijar la velocidad con la que se desplazará el drone, a partir de la distancia entre la mano y el suelo.

- Dos **servos** para asegurar que el sensor de ultrasonidos esté siempre paralelo al suelo.
- Un **módulo RF** (Radio Frequency) para establecer la comunicación con el drone y así poderle enviar los valores para establecer el rumbo y la velocidad del cuadricóptero (pitch, roll, yaw y throttle).

2.1. Entorno de desarrollo Arduino

El entorno software y hardware elegido para la implementación de este proyecto es Arduino. Esta plataforma destaca por su sencillez de programación, asequibilidad de sus placas de desarrollo y periféricos, escalabilidad tanto de software como de hardware y la gran cantidad de documentación disponible gracias a la popularidad de esta plataforma y al hecho de ser totalmente libre.

Dentro del abanico de placas de desarrollo disponibles, el entorno hardware escogido como cerebro del sistema es el Arduino Mini [3] (Figura 1).

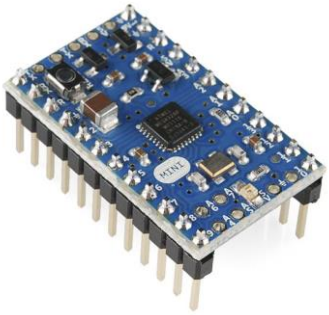

Figura 1: Placa de desarrollo Arduino Mini.

La prioridad absoluta era que fuese una placa lo más pequeña posible, pero con los puertos de entradas/salidas suficientes para poder gestionar todos los periféricos necesarios.

El Arduino Mini, con un tamaño de 30 mm x 18 mm consta de 14 entradas/salidas digitales (de las cuales 6 se pueden utilizar como salidas PWM (Pulse Width Modulation)) y 8 entradas analógicas (de las cuales 4 se pueden configurar como salida). Incorpora un microcontrolador ATmega328 (muy popular en el mercado). La placa de desarrollo se alimenta con una tensión entre 7 V y 9 V, aunque su voltaje operacional es de 5 V. La intensidad máxima de cada puerto es de 40 mA. Trabaja a una frecuencia de reloj de 16 MHz.

2.2. Conexión RF

En este punto se detalla cómo se ha realizado la comunicación con el receptor del drone utilizando la placa de desarrollo Arduino Mini y un módulo RF. De esta manera, es posible sustituir la emisora que viene incluida con el cuadricóptero.

Este es el punto más crítico del proyecto, es necesario conocer el protocolo de conexión, así como qué módulos RF son compatibles o incluso si el hecho de conectarse a un drone comercial con un Arduino es posible o no.

No obstante, antes de inclinarme definitivamente por este proyecto, al investigar acerca del tema fue posible encontrar un post de un blog en el cual el autor había conseguido *hackear* la conexión entre un drone y su emisora utilizando un Arduino y así controlarlo con éste. [1]

La conexión se había realizado en concreto con el drone con nombre comercial *Sky Walker*. Como módulo RF se utilizaba el que incorporaba la emisora que venía incluida con éste y que se muestra en la Figura 2 [4].

Figura 2: Módulo RF BK2421 incluido en la emisora de SkyWalker Drone

En [1] se detalla paso a paso cómo se debe conectar este módulo al Arduino e incluso se dispone de una librería (HCD) con ejemplos para el entorno Arduino, la cual se puede descargar y probar fácilmente.

Realizando modificaciones en la librería, esta también se puede adaptar para trabajar con módulos RF comerciales compatibles con el *BK2421*, como por ejemplo el módulo *NRF24L01*. Así que éste ha sido el módulo seleccionado.

2.2.1. Módulo *NRF24L01*

En la Figura 3 se muestra el módulo *NRF24L01* [5] seleccionado para establecer la comunicación entre el microcontrolador y el drone.

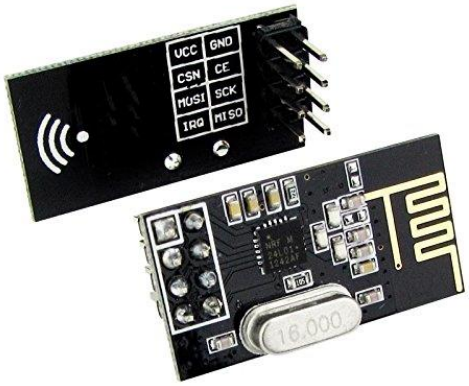

Figura 3: Módulo NRF24L01

Este módulo consiste en un transceiver que trabaja con una frecuencia de 2,4 GHz y se alimenta con una tensión de 3,3V. Se comunica con el Arduino mediante un bus SPI (Serial Peripheral Interface). Su conexionado es el que se muestra en la Figura 4.

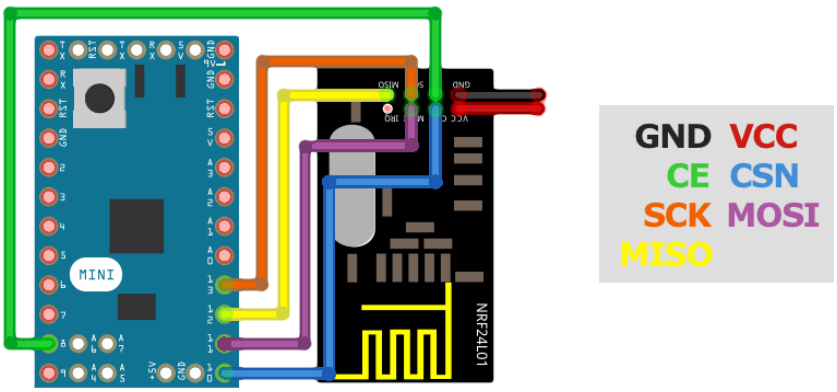

Figura 4: Conexionado Módulo NRF24L01

- **VCC:** Alimentación del módulo a 3,3V.
- **GND:** Masa.
- **CE (Chip Enable) → Digital Pin 8**: Activa o desactiva el chip en función del nivel de tensión presente, nivel alto o nivel bajo.
- **CSN (Chip Select Not) → Analog Pin 10**: El master selecciona qué slave (esclavo) va a utilizar o lo activa. En este caso sólo hay un slave (el propio NRF24L01).
- **SCK (Serial Clock) → Analog Pin 13**: Señal de reloj. Es la señal que marca la sincronización. Con cada pulso de este reloj se lee o se envía un bit.
- **MOSI (Master Output Slave Input) → Analog Pin 11**: Salida de datos del Master y entrada de datos del Slave.
- **MISO (Master Input Slave Output) → Analog Pin 12**: Salida de datos del Slave y entrada de datos del Master.

2.3. IMU (Pitch, Roll y Yaw)

En este apartado se aborda la utilización de una IMU para capturar los valores de rotación de la mano del usuario en el eje X, Y y Z, para transformarlos en los valores de Pitch, Roll y Yaw que se enviarán al drone.

Nuestro objetivo es que cuando inclinemos la mano hacia delante o atrás, el drone se desplace hacia delante o atrás respectivamente **(Pitch)**.

Dicho movimiento se puede observar en la Figura 5. Así mismo, cuando inclinemos la mano a derecha o izquierda, el drone también se desplace a derecha o izquierda **(Roll)**, según se muestra en la Figura 6. Por último, cuando rotemos la mano sobre el eje vertical, el drone realice la misma rotación **(Yaw)**, tal y como se muestra en la Figura 7.

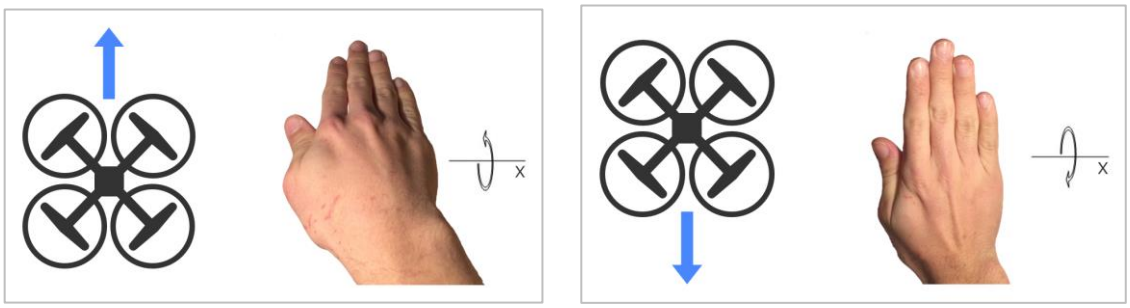

Figura 5: Rotación de la mano sobre el eje X (Pitch).

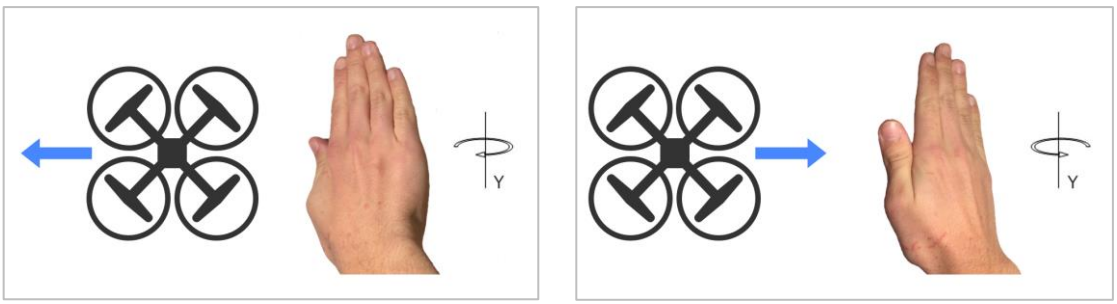

Figura 6: Rotación de la mano sobre el eje Y (Roll).

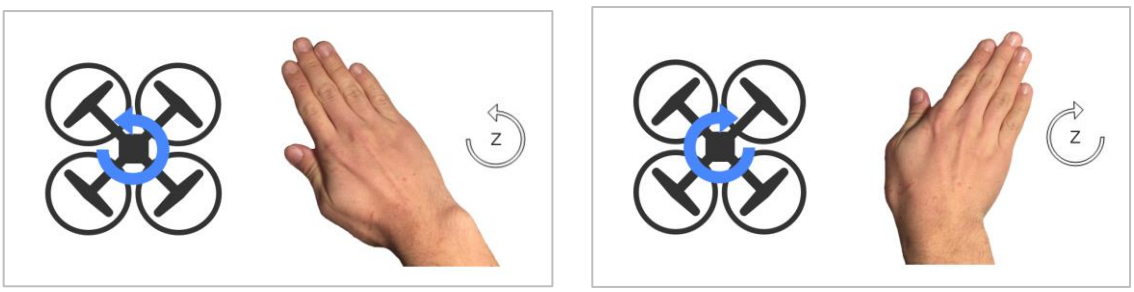

Figura 7: Rotación de la mano sobre el eje Z (Yaw).

2.3.1. Módulo *GY-85*

En la Figura 8 se muestra la IMU seleccionada para este proyecto. Su voltaje de operación está entre los 3 V y los 5 V. El protocolo de comunicación es el I²C (Inter-Integrated Circuit).

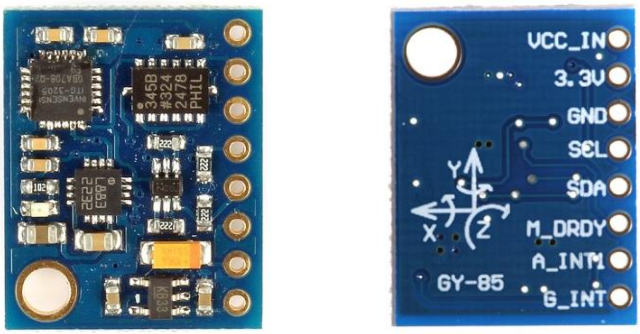

Figura 8: Módulo GY-85.

Este módulo incorpora:

- **ADXL345** (Acelerómetro de 3 ejes) [6]
- **HMC588L** (Magnetómetro o Sensor de campo magnético de 3 ejes) [7]
- **ITG3205** (Giroscopio de 3 ejes) (No lo utilizaremos, ya que no nos hace falta para los valores que necesitamos) [8]

En la Figura 9 se establece la conexión entre la placa de desarrollo Arduino Mini y el módulo *GY-85*.

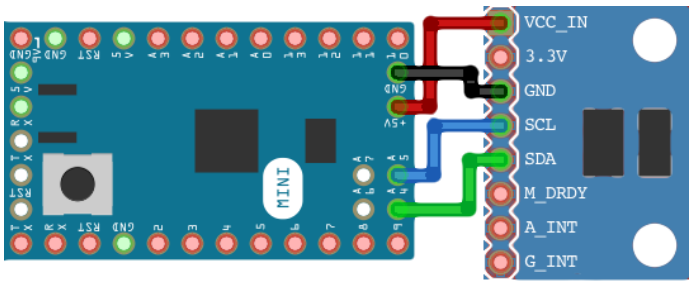

Figura 9: Conexionado Módulo GY-85.

- **VCC:** Alimentación del módulo a 5V.
- **GND:** Masa.
- **SCL (I²C Clock Line) → Analog Pin 5:** Señal de reloj. Es el pulso que marca la sincronización. Con cada pulso de este reloj se lee o se envía un bit.
- **SDA (I²C Data Line) → Analog Pin 4:** Datos enviados por la IMU al entorno Arduino y viceversa.

2.4. Sensor de Ultrasonidos

Otro de los grandes retos de este proyecto es poder calcular la distancia vertical de la mano respecto al suelo. Después de investigar varias opciones, se decidió que la más acertada era un sensor por ultrasonidos.

Estos tipos de sensores trabajan de manera similar a los sónar de los submarinos, barcos, etc. Envían un pulso ultrasónico y miden el tiempo que transcurre entre que lo envían y lo reciben, tal y como se muestra en la Figura 10.

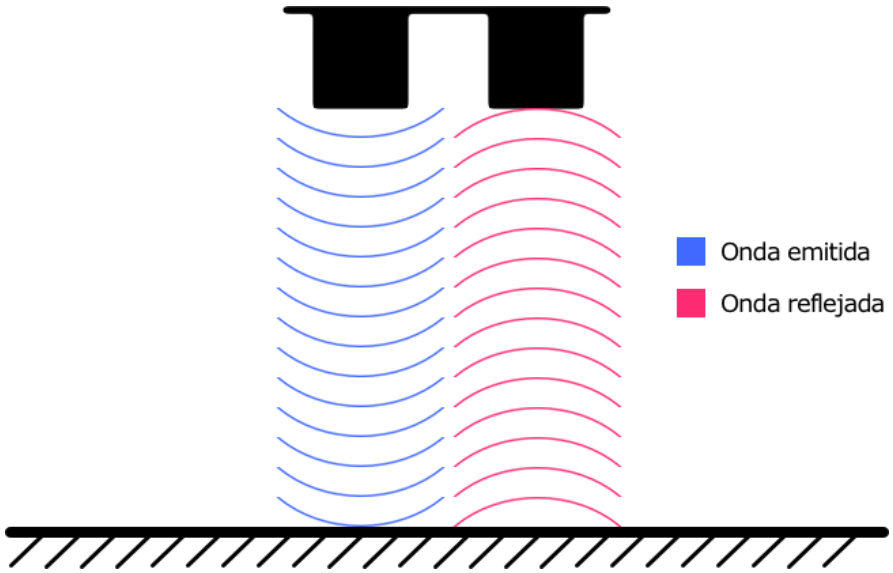

Figura 10: Funcionamiento del sensor de ultrasonidos.

Una vez medido el tiempo transcurrido en microsegundos, se calcula la distancia obtenida en la unidad deseada. Si por ejemplo, queremos calcular la distancia en centímetros y el medio de transmisión es el aire, multiplicaremos el resultado obtenido por 0,017. Este factor es el resultado de dividir la velocidad del sonido a temperatura ambiente ($34 \cdot 10^3$ cm/s) entre 10^6, ya que 1 segundo = 10^6 microsegundos, y de volver a dividir este resultado entre 2, ya que el tiempo devuelto es el que la onda tarda en ir y volver, y nosotros sólo necesitamos uno de ellos.

2.4.1. Sensor *HC-SR04*

En la Figura 11 se muestra el sensor de ultrasonidos seleccionado para este proyecto [9].

Figura 11: Sensor de ultrasonidos HC-SR04.

Este módulo se alimenta a 5V (+-0,5V). Tiene dos transductores (un altavoz y un micrófono) que calculan el tiempo entre la transmisión y recepción del pulso ultrasónico. Tiene una resolución muy buena (aproximadamente 3 mm) y es estable en las lecturas. Su ángulo de medición es de 15º y no se ve afectado por la luz solar o el material negro (aunque materiales acústicamente suaves como las telas pueden ser difíciles de detectar). La longitud máxima a la que sus lecturas son estables es unos 2 metros aproximadamente [10].

En la Figura 12 se establece la conexión entre la placa de desarrollo Arduino Mini y el sensor HC-SR04.

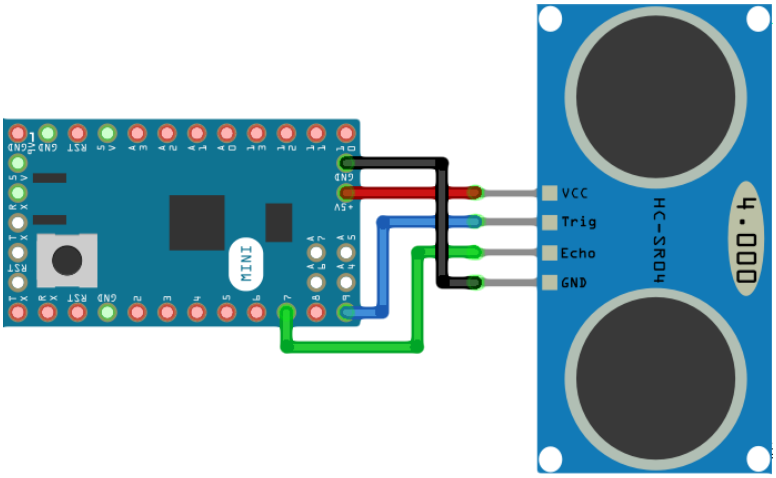

Figura 12: Conexionado del sensor de ultrasonidos HC-SR04.

- **VCC:** Alimentación del módulo a 5V.
- **GND:** Masa.
- **Trig (Trigger) → Digital Pin 9:** Genera la onda de 40kHz.
- **Echo → Digital Pin 7:** Cambia de estado cuando se envía la onda y vuelve a cambiar cuando se recibe ésta.

3. IMPLEMENTACIÓN SOFTWARE DEL SISTEMA

El software del sistema se ha desarrollado en el entorno Arduino. El programa utilizado para crear, compilar y cargar el código a la placa de desarrollo es el Arduino IDE, concretamente la versión 1.6.5.

La estructura del proyecto se divide en un programa principal (el cual se encarga de recibir, procesar y enviar los datos) y las librerías externas correspondientes a los módulos y componentes utilizados.

3.1. Librerías

Las librerías son un papel principal del software del proyecto. Ellas se encargan de manejar la conexión con los módulos y periféricos conectados a la placa de desarrollo. Las podemos dividir en tres grandes bloques: Drone, IMU y Servos.

La librería correspondiente al drone es la `HCD` [11]. Su función es gestionar el objeto Drone, incluyendo su conexión y escritura de datos. Ésta a su vez incluye los ficheros fuente `HCD.cpp`, `bk2421.cpp` (correspondiente al

módulo RF) y los ficheros de cabecera `HCD.h`, `bk2421.h` y `blocks.h`. En el fichero `blocks.h` se especifica la configuración por defecto del módulo RF *NRF24L01*.

Las librerías correspondientes al IMU son la librería estándar `Wire` (I²C) y las librerías `ADXL345` (acelerómetro) y `HMC5883L` (magnetómetro). Ellas se encargan de la conexión de la placa de desarrollo con la IMU y gestionar las lecturas de las coordenadas [12].

Por último, la librería estándar `Servo`, que se encarga de transmitir la información de giro necesaria al servo, así como de consultar su estado, etc.

En la Figura 13 se puede observar la distribución de estas librerías.

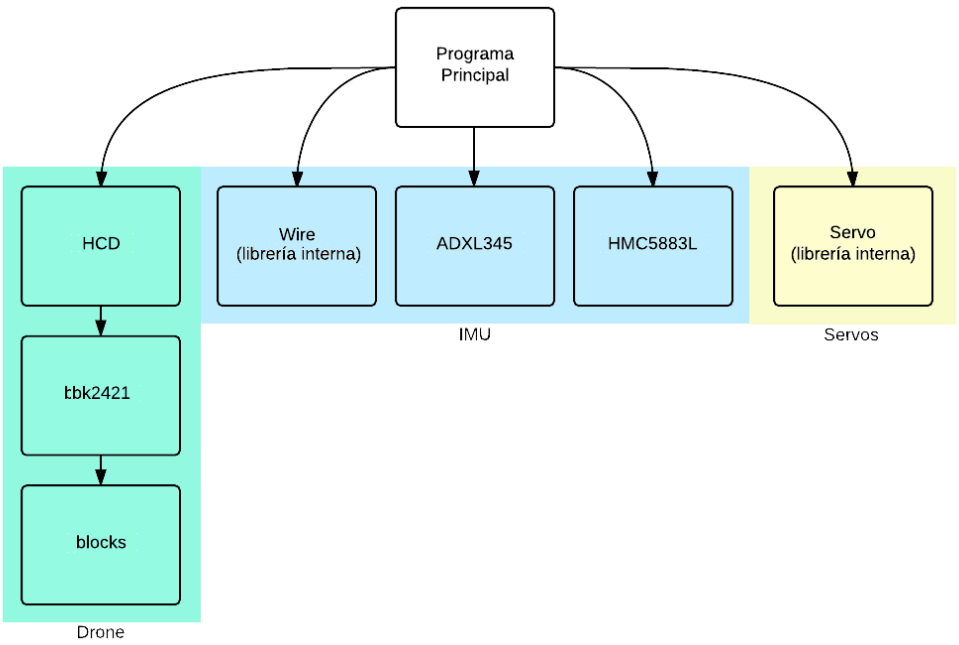

Figura 13: Esquema de distribución de librerías del proyecto.

3.2. Programación de los diferentes módulos

A continuación se especifican los principales pasos a seguir para trabajar con los principales módulos que conforman el sistema; el módulo de RF *NRF24L01*, la IMU *GY-85* y el sensor de ultrasonidos *HC-SR04*.

3.2.1. NRF24L01

Una vez hemos incluido la librería `HCD` descargada previamente, la sintaxis para conectarnos con el drone es la que se detalla a continuación.

Creamos un objeto del tipo HCD.

```
HCD drone;
```

Creamos un *array* de 4 números cualquiera que utilizaremos como identificador del drone.

```
unsigned char ID[] = {0x16, 0x01, 0x55, 0x11};
```

Nos conectamos con el drone.

```
drone.bind[ID];
```

Una vez estamos conectados, necesitamos enviarle los valores de movimiento al drone. Estos se envían cada 20 milisegundos en forma de paquetes de 8 bytes. Para ello se seguirá el siguiente orden:

- **Byte 0 = Throttle (0 - 255):** Este valor corresponde a la velocidad del drone. Incrementando este valor, aumentamos la velocidad de rotación de todos los motores a la vez. Si el valor es '0', los motores están parados. Si el valor es '255', los motores están a la máxima potencia.
- **Byte 1 = Yaw (0 - 255):** Este valor corresponde al giro del drone sobre el eje Z. Si enviamos un '0', el drone girará en sentido anti-horario a máxima velocidad. Si enviamos '255', el drone girará en sentido horario a máxima velocidad. Si enviamos '128', el drone no girará, ya que este es el valor central.
- **Byte 2 = Yaw trim (0 - 128):** Los valores *trim* indican un *offset* inicial. Si por ejemplo el drone tiene tendencia a girar un poco a la derecha, compensaremos enviando valores entre '0' y '64' hasta llegar al punto de equilibrio. En este proyecto no controlaremos los trim, así que siempre enviaremos un '64'.
- **Byte 3 = Pitch (0 - 255):** Este valor corresponde a la inclinación del drone sobre el eje X. De esta manera controlamos el desplazamiento del drone hacia delante o atrás. Si enviamos '255', el drone se moverá hacia delante a máxima velocidad. Si enviamos '0', el drone se desplazará hacia atrás a máxima velocidad. Si enviamos '128', el drone no se moverá, ya que este es el valor central.
- **Byte 4 = Roll (0 - 255):** Este valor corresponde a la inclinación del drone sobre el eje Y. De esta manera controlamos el desplazamiento del drone hacia la derecha o izquierda. Si enviamos '255', el drone se

moverá hacia la derecha a máxima velocidad. Si enviamos '0', el drone se desplazará hacia la izquierda a máxima velocidad. Si enviamos '128', el drone no se moverá, ya que este es el valor central.

- **Byte 5 = Pitch trim (0 - 128):** Al igual que Yaw trim, enviaremos un '64'.
- **Byte 6 = Roll trim (0 - 128):** Al igual que Yaw trim, enviaremos un '64'.
- **Byte 7 = Fly/Run mode (0/16):** Este drone en particular tiene 2 modos de control, uno para volar y otro para moverse como si fuera un coche. Nosotros utilizaremos el modo de volar, así que siempre enviaremos un '0'.

Para enviar estos valores utilizaremos el método `update()` disponible en la librería `HCD`. Supongamos que simplemente queremos elevar el drone a cierta altura sin ningún desplazamiento, utilizaremos:

```
drone.update(200,128,64,128,128,64,64,0);
```

Los valores que modificaremos según lo que captemos por los periféricos serán los bytes 0, 1, 3, 4.

3.2.2. *GY-85*

Para poder utilizar el acelerómetro y el magnetómetro del módulo *GY-85*, tenemos que incluir las librerías correspondientes al acelerómetro y magnetómetro (`ADXL345` y `HMC588L`), así como los ficheros de cabecera `ADXL345.h`, `HMC588L.h` y `Wire.h`, para la comunicación I²C.

Seguidamente, se deben realizar los siguientes pasos:

Creamos los objetos correspondientes a cada clase.

```
ADXL345 acc;
HMC5883L compass;
```

Activamos los dispositivos utilizando el método correspondiente a cada uno de los objetos creados.

```
acc.powerOn();
compass = HMC5883L();
```

El acelerómetro lo utilizaremos con los parámetros por defecto, pero el magnetómetro lo configuraremos en modo de lectura constante y ajustaremos su escala.

```
compass.setMeasurementMode(Measurement_Continuous);
compass.SetScale(1.3);
```

Para leer los datos del acelerómetro, creamos las variables correspondientes a cada eje `(ax, ay, az)` y pasamos su puntero como argumento en el método de lectura.

```
acc.readAccel(&ax, &ay, &az);
```

Para leer los datos del magnetómetro, creamos el objeto del tipo correspondiente donde se almacenarán.

```
MagnetometerScaled = compass.ReadScaledAxis();
```

Y leemos el valor de cada una de las coordenadas del siguiente modo:

```
x = scaled.YAxis;
y = scaled.YAxis;
z = scaled.YAxis;
```

Una vez hacemos las lecturas en cada iteración, se debe proceder a tratar los datos antes de enviárselos al drone. Los datos del acelerómetro, se deben procesar para obtener valores correspondientes a la rotación de la mano en los ejes X e Y. Utilizando una combinación de las ecuaciones de [13] y [14] se obtiene el Pitch (Ecuación 1) y Roll (Ecuación 2). El cálculo se reduce a la arcotangente de las coordenadas X e Y y viceversa obtenidas por el acelerómetro.

$$Pitch = -\left(180 \cdot \frac{tan^{-1}\left(\frac{Y}{\sqrt{X^2 \cdot Z^2}}\right)}{\pi}\right) \quad (1)$$

$$Roll = -\left(180 \cdot \frac{tan^{-1}\left(\frac{X}{\sqrt{Y^2 \cdot Z^2}}\right)}{\pi}\right) \quad (2)$$

Uno de los problemas que presenta el magnetómetro es que varía sus coordenadas en función de la inclinación del eje Z. Si éste es perpendicular al suelo y se rota la IMU sobre éste eje Z, las lecturas no se ven afectadas, pero si inclinamos la IMU se puede apreciar cómo el magnetómetro empieza a devolver valores erróneos. Para compensar esta inclinación se ha utilizado una combinación de las ecuaciones de [15] y [16], donde X, Y y Z son las coordenadas del magnetómetro (Ecuación 3).

$$Yaw = tan^{-1}\left(\frac{X \cdot sin(Roll) \cdot sin(Pitch) + Y \cdot cos(Roll) - Z \cdot sin(Roll) \cdot cos(Pitch)}{X \cdot cos(Pitch) + Z \cdot sin(Pitch)}\right) \quad (3)$$

Tal y como se indica en [17] otra opción es calcular la aceleración en los ejes X, Y y Z con el giroscopio, ponderar los resultados con el acelerómetro y posteriormente hacer la corrección.

Para comprobar la exactitud de los datos obtenidos no es suficiente imprimirlos por el monitor serie, ya que no es una manera intuitiva de evaluar si se está obteniendo la posición deseada. Así que se ha desarrollado una aplicación utilizando el entorno *Processing*, la cual interpreta los datos recibidos por el puerto serie y transforma la posición

3D de una imagen de referencia, enseñando exactamente qué posición se ha detectado combinando los valores del acelerómetro y magnetómetro. Se puede observar una captura de la aplicación en la Figura 14.

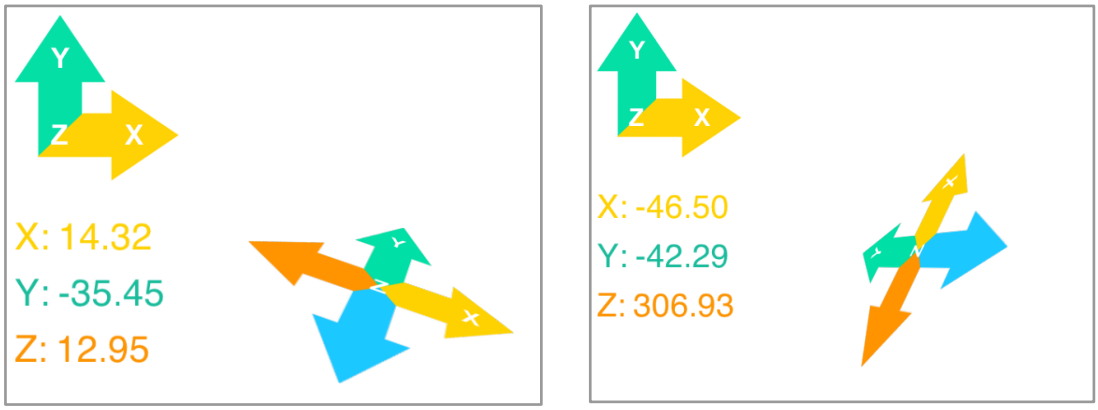

Figura 14: Aplicación en Processing para representar los datos del IMU.

3.2.3. *HC-SR04*

Para utilizar este sensor no es necesario incluir ninguna librería. Lo primero que debemos hacer es declarar las variables de los pines que harán de `trigger` y `echo`. Es muy importante que el pin que actuará como `trigger` sea una salida digital PWM.

```
const int triggerPin = 9;
const int echoPin = 7;
```

Seguidamente procedemos a activar la salida `trigger`, de esta manera el sensor envía la onda y esperamos 10 microsegundos a que éste realice dicha operación.

```
digitalWrite(triggerPin, HIGH);
delayMicroseconds(10);
```

Para acabar, a través de la función `pulseIn()` medimos el ancho del pulso de la señal presente en el pin `echo`, tal y como se muestra en la Figura 15. Este pulso cambia de estado cuando se envía la onda y vuelve a cambiar cuando se recibe. De esta manera, obtenemos el tiempo en milisegundos que ha tardado la onda en ir y volver.

```
long distance = pulseIn(echoPin, HIGH);
```

Una vez obtenemos este tiempo, podemos procesarlo como más nos convenga. En este caso, he añadido un filtro que estabiliza las pequeñas variaciones y otro los cambios bruscos en los cuales los valores pasan a 0 o 255 (valor máximo).

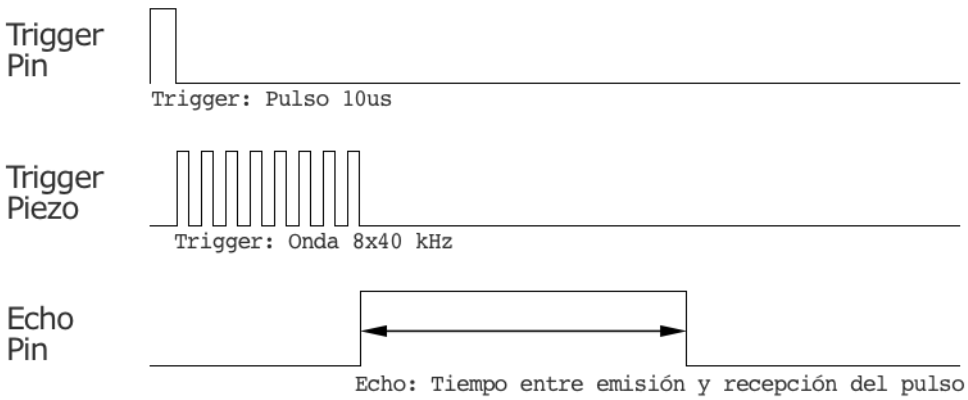

Figura 15: Cronograma correspondiente al envío y recepción del pulso.

Para comprobar la estabilidad de las lecturas del sensor una vez tratados los datos, se ha procedido a modificar el programa en *Processing* mencionado en el subapartado anterior, añadiendo un *slider* que muestra la distancia vertical calculada. Esta modificación se puede ver en la Figura 16.

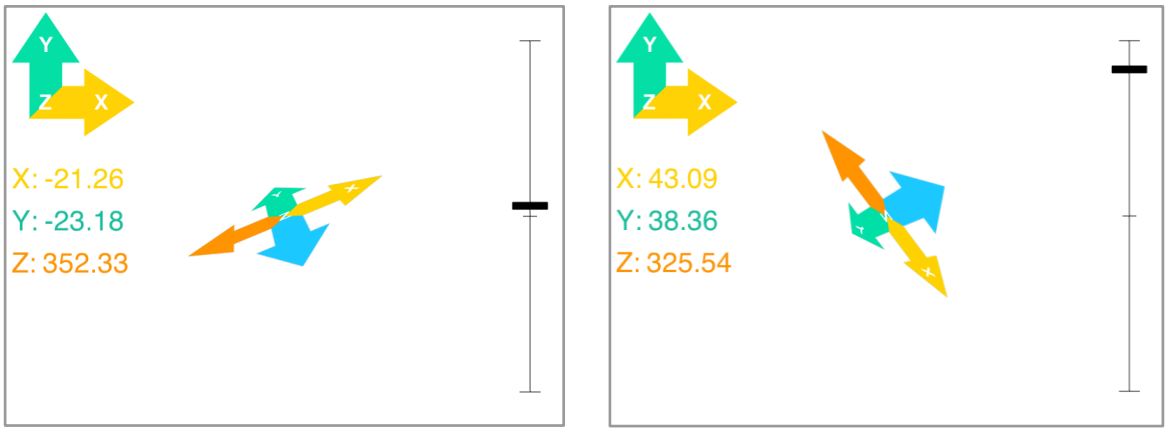

Figura 16: Modificación de la aplicación en Processing para mostrar los valores del sensor de ultrasonidos.

3.3. Diagrama de flujo del programa principal

Como cualquier programa desarrollado para Arduino, éste tiene dos funciones principales, `setup()` y `loop()`. La función `setup()` se ejecuta una sola vez y se suele utilizar para inicializar parámetros, activar módulos, etc. La función `loop()` se ejecuta justo después de `setup()` durante un número infinito de veces (o hasta que se apague la placa). En ésta es donde irá la mayor parte del código.

En nuestro programa principal lo primero que se hace es incluir las cabeceras de librerías y crear las variables globales del proyecto.

Seguidamente, en la función `setup()` se declaran todas las entradas y salidas de la placa de desarrollo, se inicializa la IMU y se calculan los valores de *offset* e iniciales del acelerómetro, magnetómetro y sensor de ultrasonidos, así como otras tareas secundarias.

Después, en la función `loop()` se leen los valores del acelerómetro, magnetómetro y sensor de ultrasonidos, se procesan y se envían al drone.

Todo este proceso se puede apreciar en el diagrama de la Figura 17.

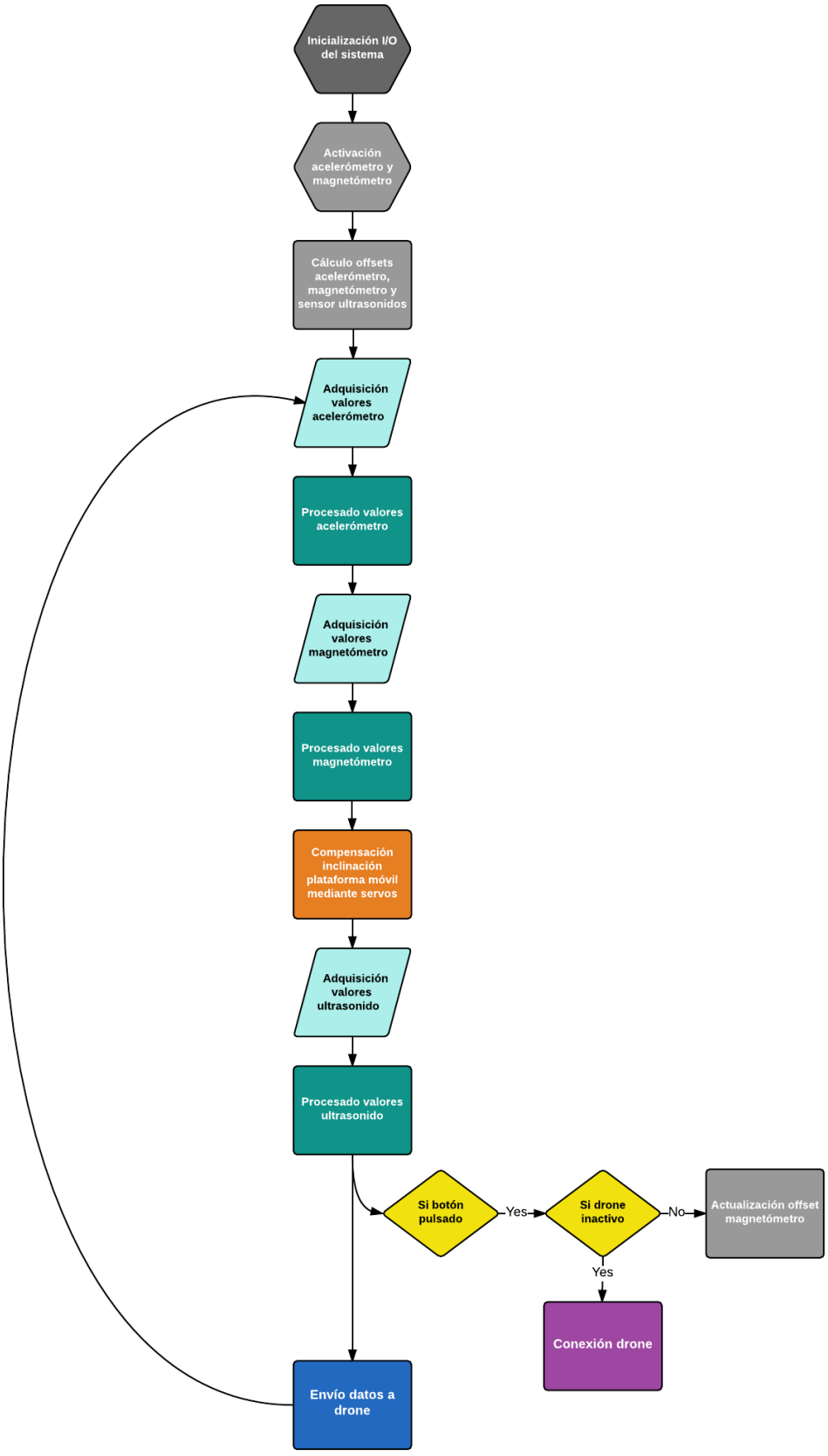

Figura 17: Diagrama de flujo del programa principal.

4. IMPLEMENTACIÓN HARDWARE DEL SISTEMA

En esta parte se explica todo lo correspondiente al diseño y creación de la PCB y chasis del sistema, así como la alimentación de éste y el montaje final del conjunto.

4.1. Alimentación

La alimentación del sistema es otro punto clave de este proyecto. Para poder elegir qué tipo de batería utilizar, primero se deben conocer las tensiones de todos los elementos así como su consumo.

Por un lado, tenemos la placa de desarrollo Arduino Mini, que se alimenta a una tensión entre 7 V y 9 V. Por otro lado, tenemos los servos, que para funcionar correctamente necesitan una tensión de 6 V aproximadamente y son unos de los periféricos que más consumen (300 mA/servo). A todo esto hay que añadir el sensor por ultrasonidos y la IMU, que ambos se

alimentan a 5V y el módulo RF, que funciona a 3,3V. Así que es necesario disponer de una batería que proporcione como mínimo 7 V y tenga una capacidad de carga elevada, aproximadamente de 700 mAh.

Finalmente, se ha optado por una batería Li-Po que proporciona una tensión de 7,4 V y posee una carga de 850 mAh, suficiente para hacer funcionar el equipo varios vuelos, dado que la duración media de vuelo del drone es de 5 minutos.

Como se ha mencionado anteriormente, el Arduino Mini se alimenta entre 7 V y 9 V. No obstante, su voltaje operacional es de 5 V, con lo cual tiene varias salidas que proporcionan esta tensión y se utilizarán para alimentar la IMU y el sensor de ultrasonidos. Esta placa de desarrollo incorpora un regulador de tensión de 5 V y 150 mA.

Para alimentar los servos se emplea un regulador de tensión que reduce la tensión de la batería a 6V.

Por último, es necesario conseguir los 3,3 V para alimentar el módulo RF. Para ello, también se ha utilizado un regulador de tensión. En este caso se obtienen los 3,3 V a partir de los 6 V de la alimentación de los servos.

En la Figura 18 se puede observar el flujo de regulación de las diferentes tensiones de alimentación.

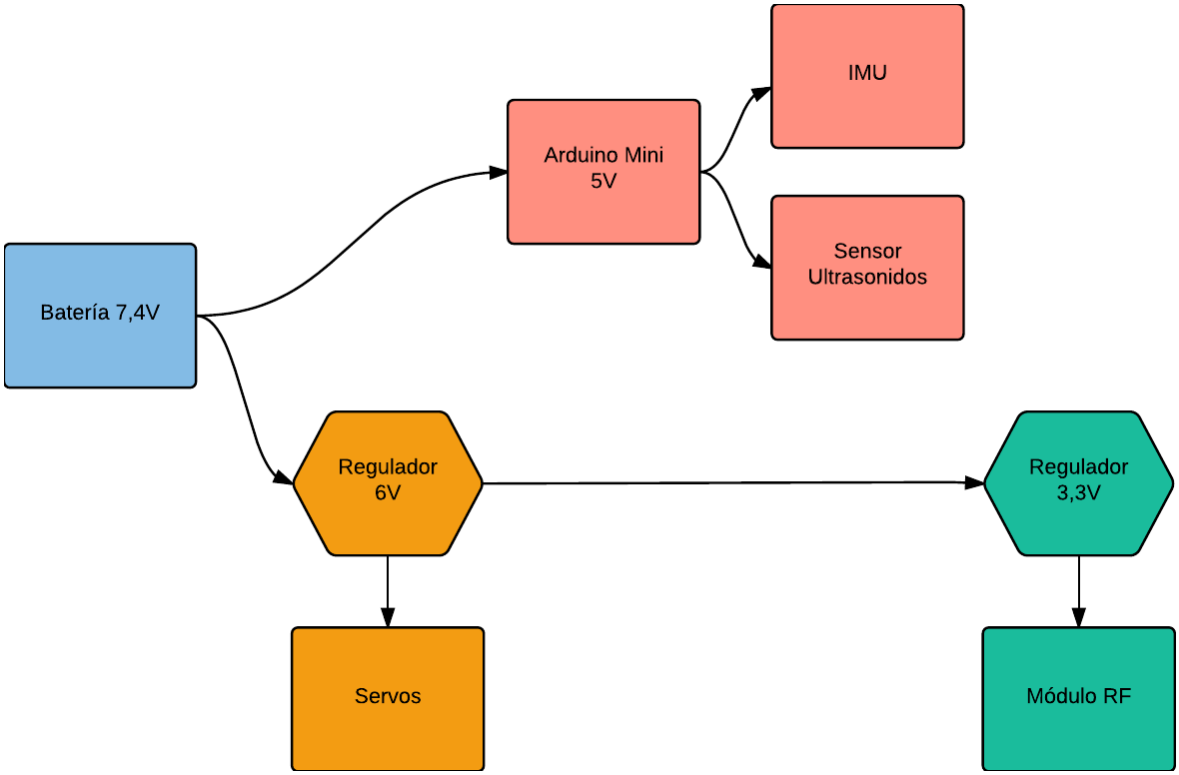

Figura 18: Diagrama de regulación de voltajes del sistema.

4.2. PCB

Otro reto a añadir a este proyecto es el diseño y creación de la PCB para el sistema. Para esta tarea se ha optado por el software gratuito *Fritzing*.

El primer paso es realizar el esquemático con todos los componentes y conexiones, tal y como se muestra en la Figura 19.

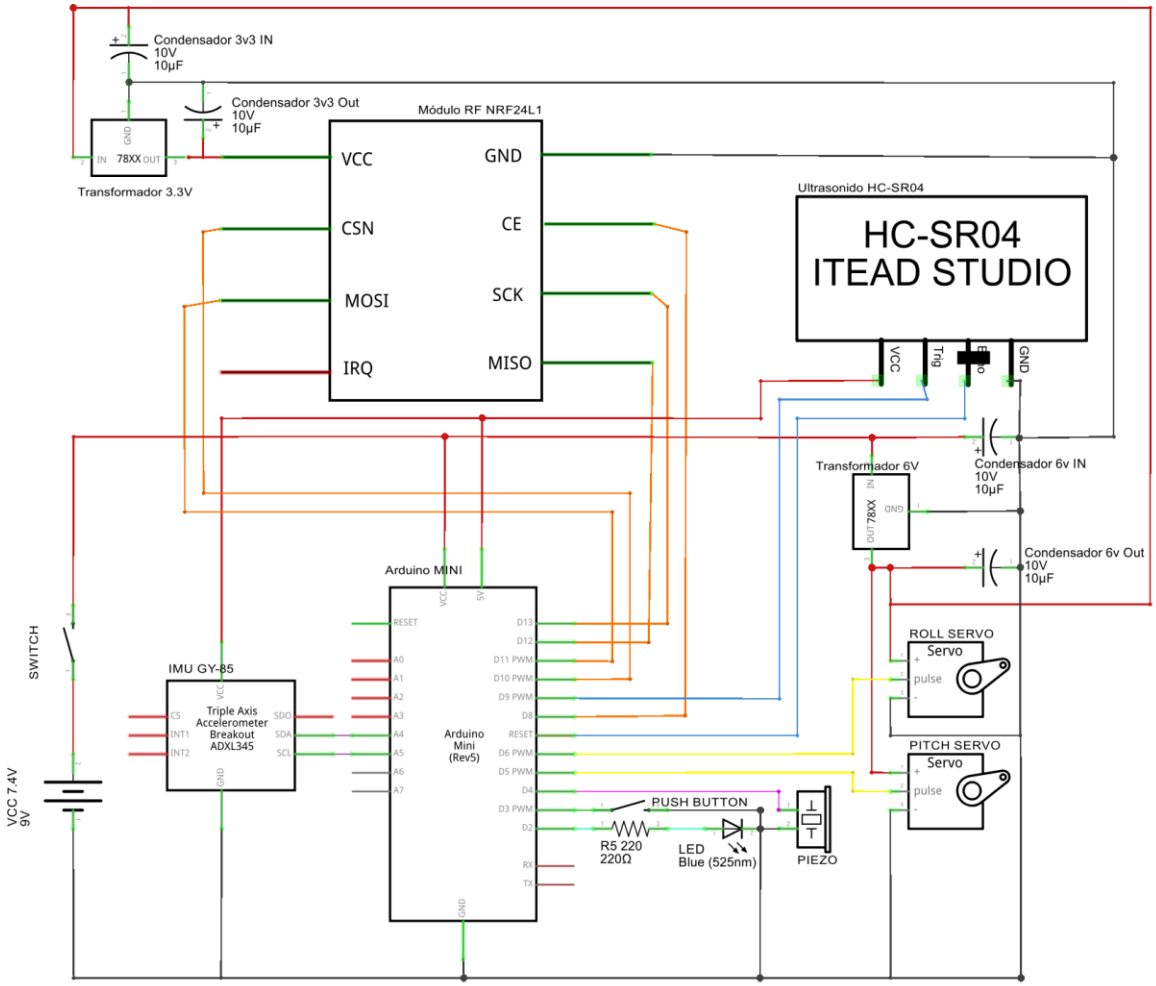

Figura 19: Esquemático del sistema.

Una vez finalizado, *Fritzing* inserta automáticamente los elementos en el apartado de diseño de la PCB. El siguiente paso es crear los ruteados de las pistas manualmente y ajustar las posiciones y tamaños de los elementos así como de los agujeros y pistas.

Se utilizaron las partes sobrantes de ambas caras como planos de masa, de esta manera se evita tener que sustraer esa gran cantidad de material y mejora la integridad de las señales, ya que permite mantener cortas las

conexiones de masa lo que proporciona conexiones de baja inductancia, además de simplificar la conexión de los diferentes componentes a masa.

En la Figura 20 se muestran las dos caras de la PCB diseñada, a la izquierda se observa la cara superior y a la derecha la cara inferior.

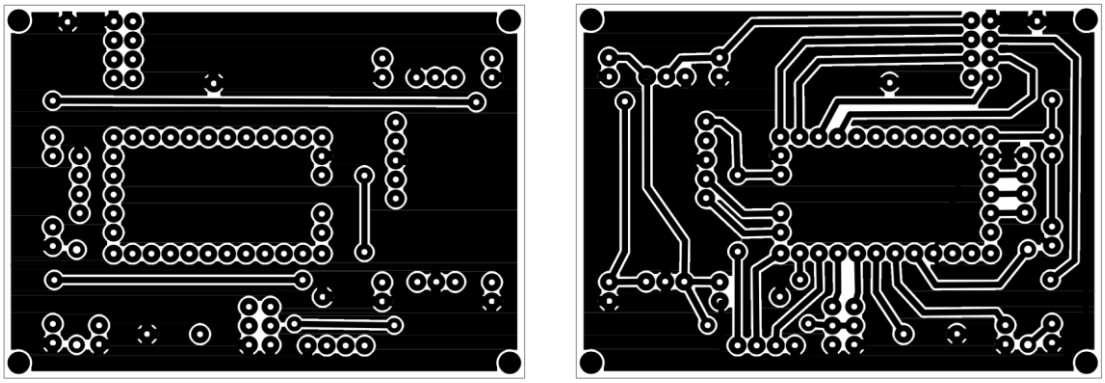

Figura 20: Diseño final de la PCB del sistema.

Seguidamente, en la Figura 21 se aprecian las dos caras de la PCB fabricada.

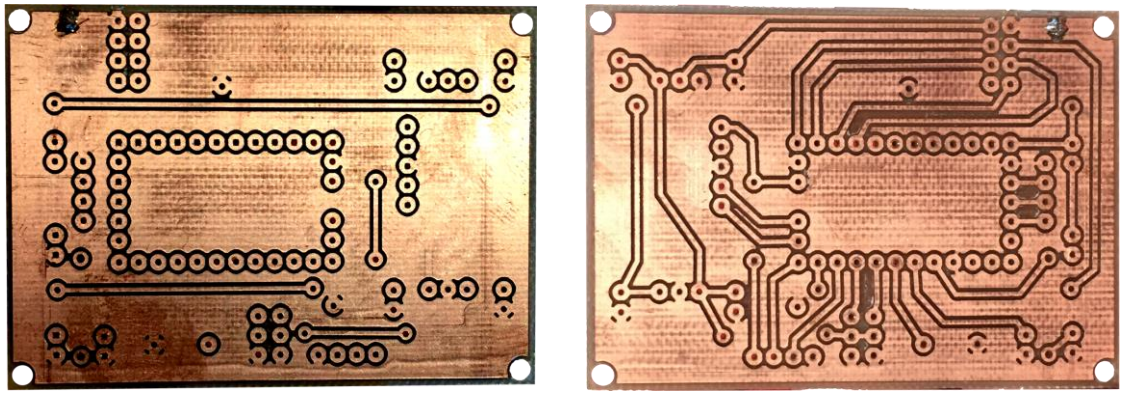

Figura 21: PCB fabricada.

Finalmente se optó por insertar pines hembra en las posiciones de la placa de desarrollo Arduino Mini, módulo RF e IMU en vez de soldar estos componentes a la PCB directamente, así se pueden reutilizar estas piezas en un futuro si es necesario, o sustituirlas fácilmente.

Los periféricos como los servos y el sensor de ultrasonidos que se encuentran en la base móvil se conectan mediante cables a los pines macho soldados en la PCB.

Para el interruptor encendido/apagado, el pulsador y el led se optó por soldar los cables directamente a la PCB.

En la Figura 22 se puede observar la placa con todos los elementos colocados.

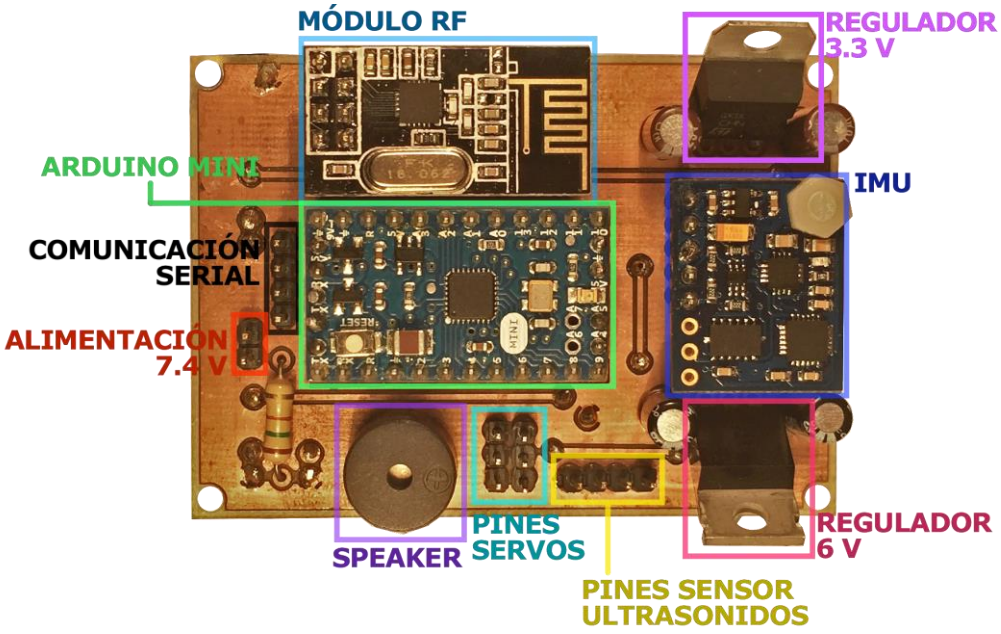

Figura 22: PCB con todos los elementos soldados.

4.3. Chasis

Para poder construir el sistema se quería ir un paso más allá y no dejarlo en un simple prototipo montado en una protoboard, así que se decidió diseñar un chasis en 3D para poder imprimirlo posteriormente en una impresora 3D.

4.3.1. Diseño (Autodesk 123D)

El programa elegido para el diseño del chasis es el *Autodesk 123D Design*. Este software se caracteriza por ser totalmente gratuito y tener una interfaz muy simple e intuitiva. No ofrece las mismas posibilidades que otras herramientas como *3DMax*, *Maya* o *Blender*, pero para diseñar piezas básicas es una opción muy acertada.

4.3.2. Plataforma móvil + Soporte servos

Una de las características mencionadas anteriormente sobre el sensor de ultrasonidos es que tiene un ángulo de medición de 15º. Esto supone un problema ya que el usuario estará moviendo la mano continuamente para controlar el drone, haciendo que el sensor permanezca inclinado en varias posiciones la mayor parte del tiempo, obteniendo lecturas erróneas sobre la altura de la mano.

Para poder solucionar este problema y mantener el sensor de ultrasonidos enfocando al suelo constantemente, se optó por diseñar una plataforma móvil controlada por dos micro servos que compensan la inclinación de la mano en los ejes X e Y con los valores leídos por la IMU.

La plataforma consta de dos partes, la superficie móvil donde va alojado el sensor de ultrasonidos y la parte fija, donde se encuentran sujetos los servos y que está acoplada a la carcasa.

Estos servos se encargan de compensar la inclinación de los ejes X e Y (cada servo controla un eje). El valor de giro de los servos se calcula cambiando el signo del valor de inclinación de la IMU en el eje del servo correspondiente (ya que se está contrarrestando) y multiplicado por un factor 1,5. Este factor compensa el hecho de que un movimiento de 1º del servo es aproximadamente 0,66º de movimiento de la plataforma.

Antes de empezar a diseñar la pieza con el software, se fabricó un prototipo con un bolígrafo en 3D (similar a una impresora 3D pero en forma de bolígrafo) y algunas piezas de juguetes antiguos para ver si la idea era funcional. Se puede ver esta plataforma en la Figura 23.

Figura 23: Plataforma móvil fabricada con bolígrafo 3D.

Una vez comprobado el correcto funcionamiento de la plataforma, se dispuso a hacer el diseño en 3D. Para asegurar que los componentes encajasen a la perfección, se crearon también en 3D y se dibujaron las piezas en función de éstos.

En la Figura 24 se puede observar el diseño en 3D de la plataforma móvil.

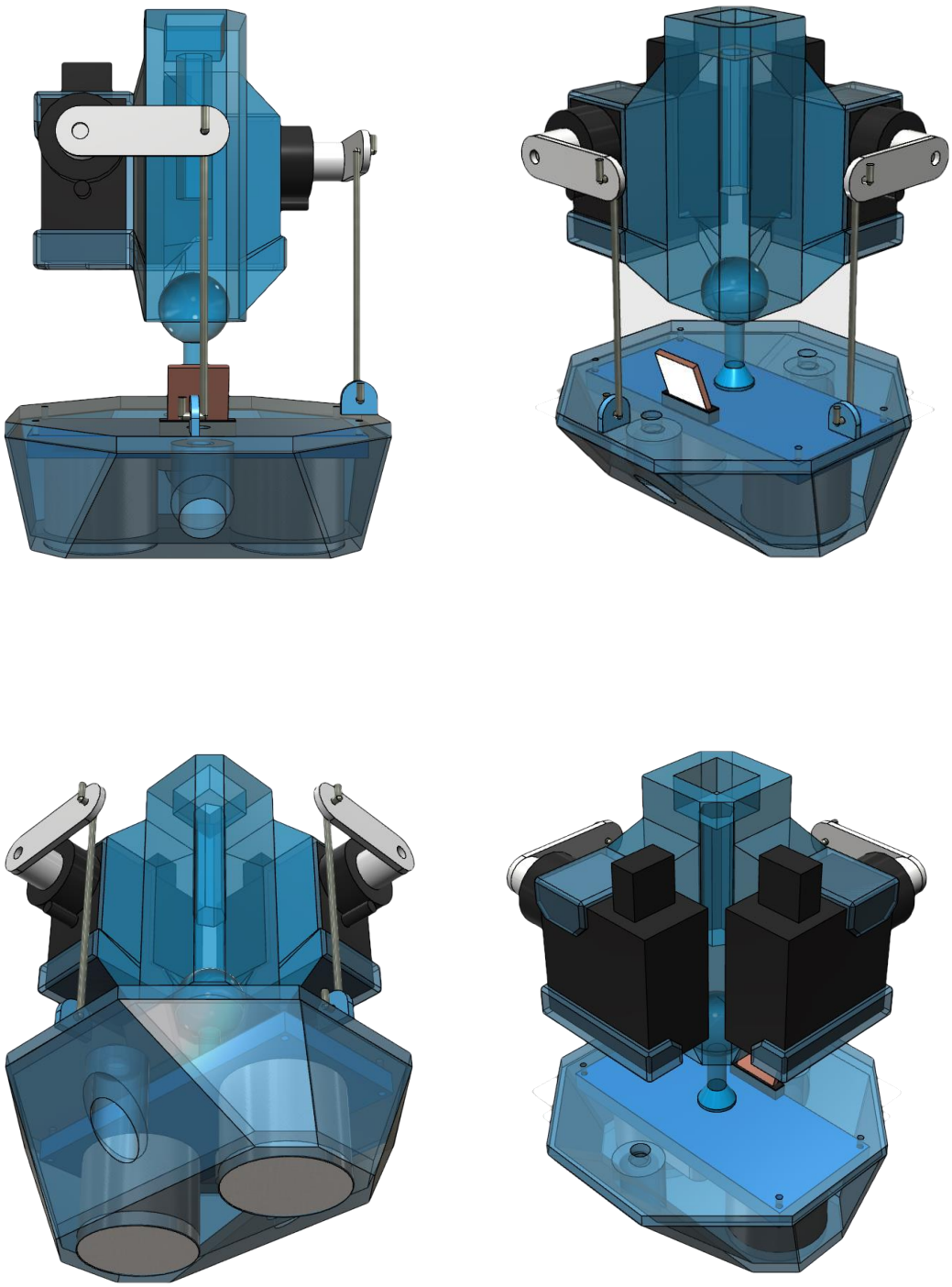

Figura 24: Diseño 3D plataforma móvil.

4.3.3. Carcasa

La otra pieza clave del sistema es la carcasa. Esta sirve para proteger el conjunto y sostener la PCB, batería, botones, etc, y poder sujetarse a la mano, de manera que al moverla, el conjunto se desplace con ella.

Como el diseño de esta pieza no es tan complejo, se optó por dibujarla directamente en 3D (tal y como se aprecia en la Figura 25). La máxima prioridad era que ocupase el menor tamaño posible, pero teniendo en cuenta que la plataforma móvil necesita espacio para poder inclinarse sin ningún obstáculo. Otro punto a tener en cuenta, es que la carcasa debe ser lo suficientemente amplia para contener la PCB con los periféricos necesarios y todo el conjunto tiene que poder sujetarse a la mano firmemente.

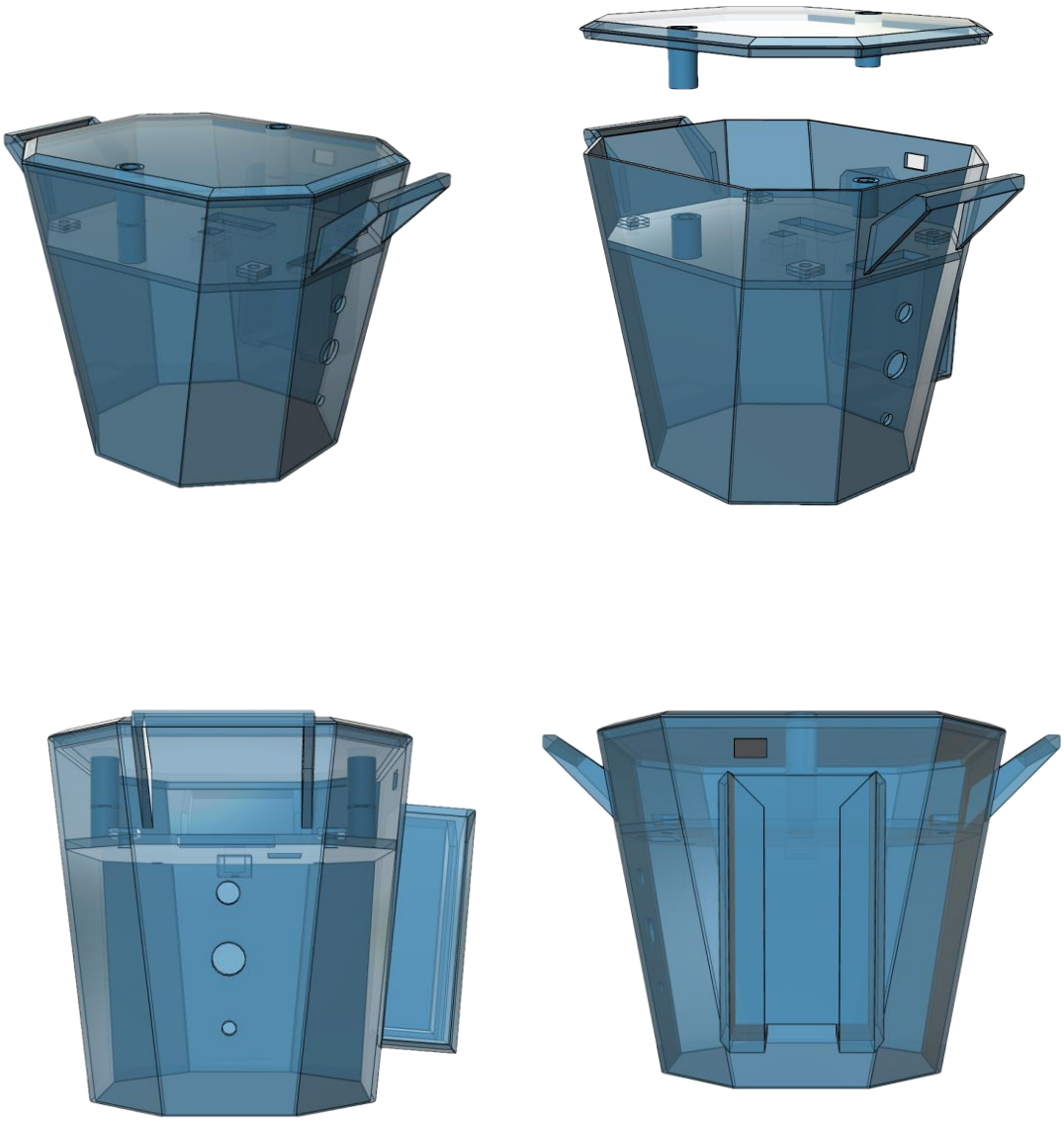

Figura 25: Diseño 3D carcasa.

El conjunto entero queda tal y como se muestra en la Figura 26.

Figura 26: Diseño 3D del sistema completo.

4.3.4. Fabricación

El material con el que se han construido las piezas es PLA (Poliácido Láctico). Es un tipo de polímero constituido por moléculas de ácido láctico con propiedades semejantes al conocido PET (Tereftalato de polietileno), el cual se utiliza para hacer envases. Es un material que posee cierta flexibilidad y es biodegradable (se degrada fácilmente con agua y óxido de carbono).

La impresora con la que se han fabricado las piezas es la BCN3D+. Posee una resolución de 0,4 mm, una altura de capa mínima de 0,075 mm y su volumen máximo de impresión es de 252x200x200 mm.

En la Figura 27 se puede observar el resultado final después de realizar la impresión 3D.

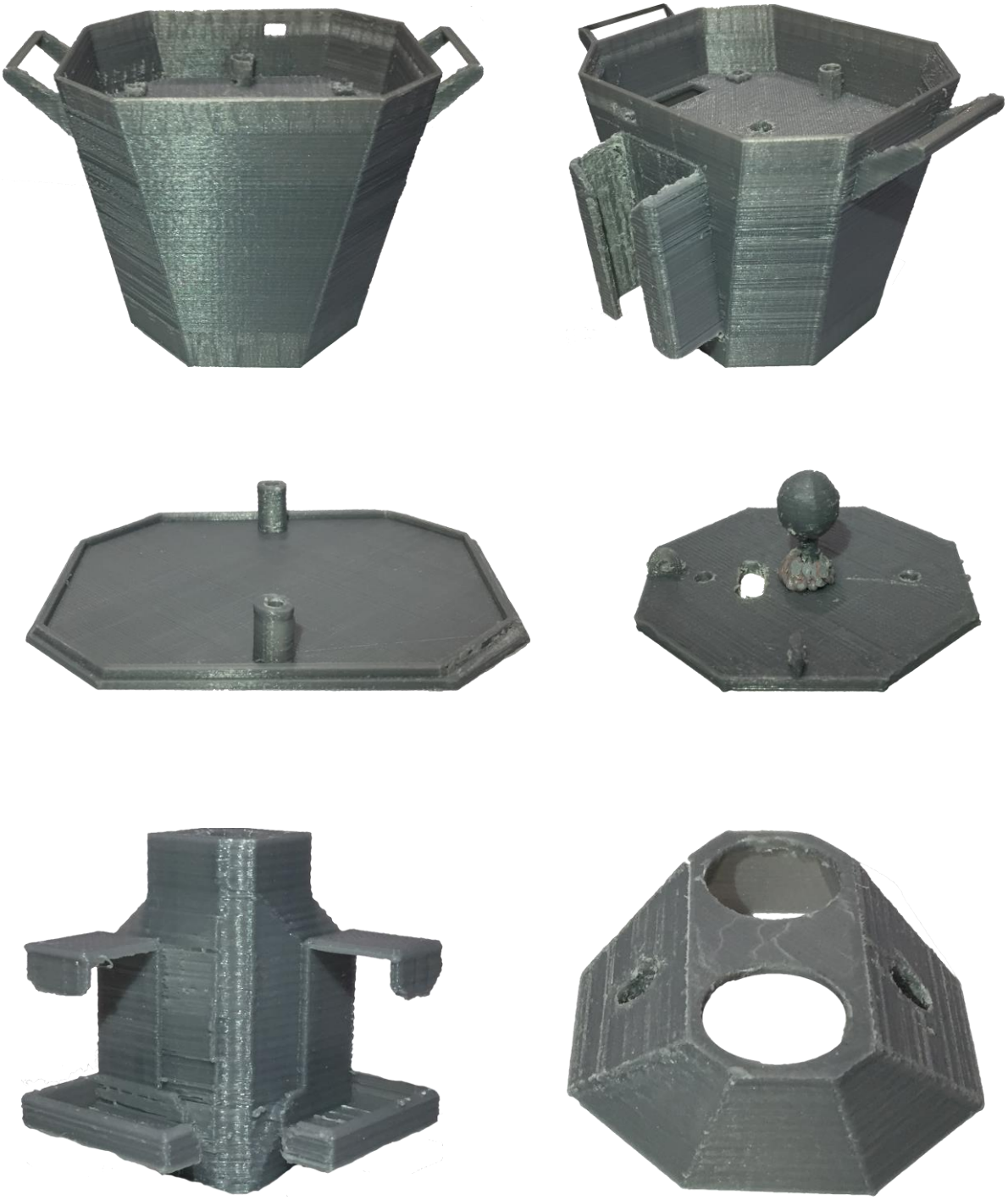

Figura 27: Piezas fabricadas con impresora 3D.

4.3.5. Montaje del sistema

Pese a la alta resolución de la máquina, sus acabados no fueron del todo precisos, ya que una vez fabricadas las piezas, éstas no encajaban entre sí tal y como lo hacían en el diseño. Se procedió a lijar casi todas las uniones y modificar algunas piezas para poder encajarlas y que los componentes pudieran ser colocados en su lugar. También se pintó todo el conjunto de color negro, acorde con la imagen del drone y eliminando el color blanco procedente de las partes lijadas de las piezas.

Se reforzó la unión entre la bola que permite el giro de la plataforma móvil y la base de ésta, dado que es un punto que sufre mucha tensión y podía terminar rompiéndose. También se optó por eliminar los brazos que sujetaban los servos, dado que no ofrecían un agarre lo suficientemente fuerte y la colocación de estos era muy tediosa. En su lugar, los servos se sujetaron con bridas, tal y como se aprecia en la Figura 28.

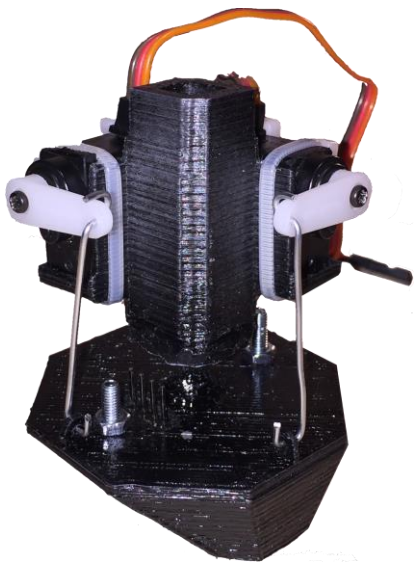

Figura 28: Plataforma móvil finalizada.

La unión entre la plataforma móvil y la carcasa se realiza mediante un tornillo (Figura 29) situado en el centro de la carcasa que atraviesa a lo largo el pilar central de la pieza donde se alojan los servos.

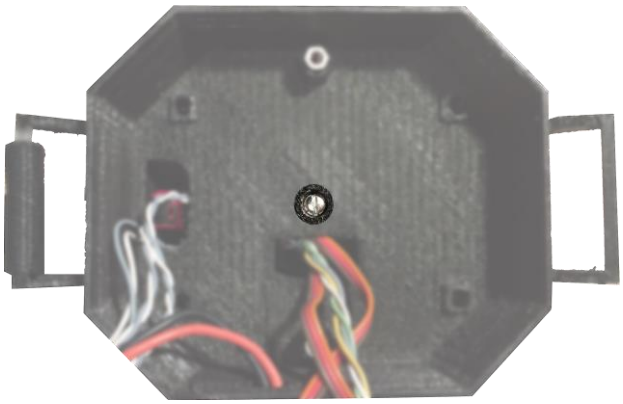

Figura 29: Tornillo encargado de unir la plataforma móvil con la carcasa.

Una vez unidas ambas partes, insertados el resto de componentes y atornillado la tapa, el conjunto queda tal y como se muestra en la Figura 30.

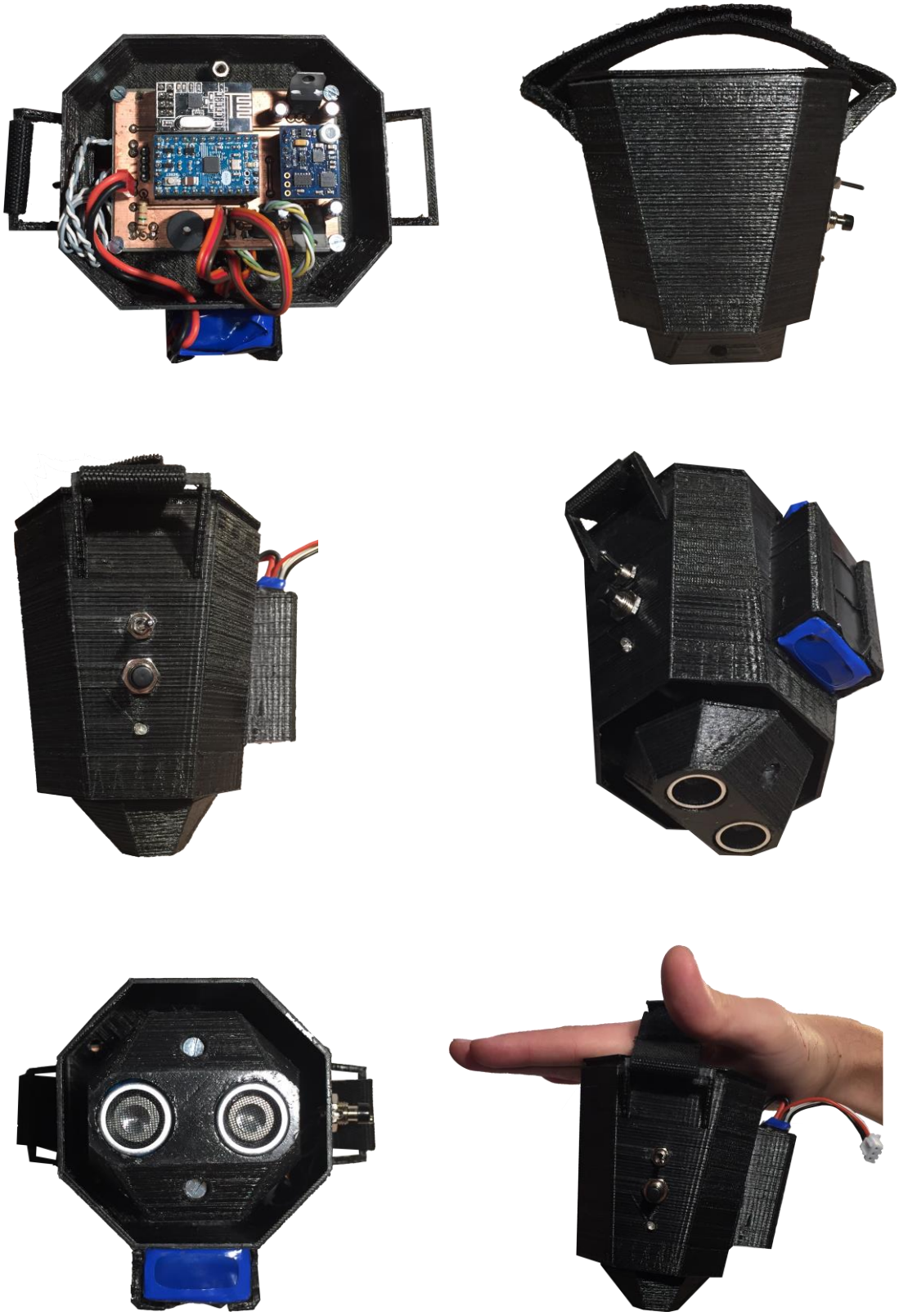

Figura 30: Montaje finalizado.

5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1. Conclusiones

Los resultados finales nos muestran que es posible controlar el drone mediante los movimientos naturales de la mano, cumpliendo de este modo con las expectativas iniciales. Se ha concluido también que este tipo de control del drone es menos preciso que el de un mando convencional, pero al mismo tiempo es mucho más fácil de utilizar para ese público que no tiene conocimientos previos sobre pilotaje de drones. No obstante, es necesario seguir investigando sobre la compensación del magnetómetro con el acelerómetro, así como la conexión entre el sistema y el drone.

5.2. Trabajos futuros

El punto clave a seguir investigando en este proyecto es el sistema para calcular la distancia vertical de la mano, ya que el actual, al requerir dos servos para mantener el sensor de ultrasonidos paralelo al suelo, incrementa y complica el tamaño y diseño del chasis. Con una solución eficiente que no incluyese servos, se reduciría notablemente el tamaño

del sistema, se simplificarían las piezas y se disminuiría el consumo de la batería, pudiéndose sustituir por otra más pequeña y con menos capacidad.

Otro aspecto importante a seguir investigando también en este proyecto, es perfeccionar el algoritmo que compensa las lecturas del magnetómetro con los valores del acelerómetro (y giroscopio, si es necesario). De esta manera, se obtendría un control más preciso del drone, mejorando la experiencia de vuelo.

También sería interesante tratar con más detalle la conexión con el drone. Actualmente esta conexión es en un sólo sentido, de manera que sólo nosotros enviamos datos al drone, pero desconocemos en todo momento las señales que éste nos envía. Un punto positivo entonces sería poder habilitar la recepción de datos del drone. Así mismo, otro factor interesante sería ampliar el rango de modelos de drones habilitados para funcionar con este control remoto, estandarizando el algoritmo de conexionado.

Por último, se debería estudiar el hecho de diseñar y fabricar una propia placa de desarrollo. Esto ayudaría a reducir costes en la fabricación del sistema y se tendría un control más preciso de los componentes implicados.

BIBLIOGRAFÍA

[1] *Dzl's Evil Genius Lair: Controlling toy quadcopter(s) with Arduino* (12 de Noviembre de 2013). Última consulta 15 de Septiembre de 2015, recuperado de http://dzlsevilgeniuslair.blogspot.dk/2013/11/more-toy-quadcopter-hacking.html

[2] *Tutorial: Fly your Parrot AR.Drone 2.0 with a Kinect for Windows* (4 de Diciembre de 2014). Última consulta 22 de Octubre de 2015, recuperado de https://sebastianbrandes.com/2014/12/tutorial-parrot-ar-drone-kinect/

[3] *Arduino - ArduinoBoardMini* (s.f.). Última consulta 1 de Noviembre de 2015, recuperado de https://www.arduino.cc/en/Main/ArduinoBoardMini

[4] *BK2421 - Low Power High Performance 2.4 GHz GFSK Transceiver* (18 de Marzo de 2010). Última consulta 26 de Noviembre de 2015, recuperado de https://halckemy.s3.amazonaws.com/uploads/document/file/1337/Beken-BK2421.pdf

[5] *NRF24L01+ Single Chip 2.4GHz Transceiver Preliminary Product Specification v1.0* (Marzo de 2008). Última consulta 26 de Noviembre de 2015, recuperado de https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf

[6] *3-Axis, ±2 g/±4 g/±8 g/±16 g Digital Accelerometer* (s.f.). Última consulta 6 de Octubre de 2015, recuperado de http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf

[7] *3-Axis Digital Compass IC HMC5883L* (Marzo de 2011). Última consulta 6 de Octubre de 2015, recuperado de http://cdn.sparkfun.com/datasheets/Sensors/Magneto/HMC5883L-FDS.pdf

[8] *ITG-3200 3-Axis Gyro Evaluation Board Application Note Rev 1.1* (s.f.). Última consulta 6 de Octubre de 2015, recuperado de https://cdn.sparkfun.com/datasheets/Sensors/Gyros/3-Axis/EB-ITG-3200-00-01.1.pdf

[9] *Ultrasonic Ranging Module HC - SR04* (s.f.). Última consulta 25 de Diciembre de 2015, recuperado de http://www.micropik.com/PDF/HCSR04.pdf

[10] *Todo lo que tienes que saber sobre: HC-SR04 Sensor Ultrasónico* (4 de Enero de 2016). Última consulta 2 de Enero de 2016, recuperado de http://bkargado.blogspot.com.es/2013/09/todosobrehc-sr04.html

[11] *dzlonline/HCDlib · GitHub* (22 de Abril de 2014). Última consulta 16 de Noviembre de 2015, recuperado de https://github.com/dzlonline/HCDlib

[12] *Arduino and GY-85 9DOF (Accelerometer ADXL345, Gyroscope ITG3200 and Magnetometer HMC5883) + Angle Information Comparison* (21 de Marzo de 2015). Última consulta 20 de Septiembre de 2015, recuperado de http://www.himix.lt/arduino/arduino-and-gy-85-9dof-accelerometer-adxl345-gyroscope-itg3200-and-magnetometer-hmc5883-angle-information-comparison/

[13] *Arduino IMU: Pitch & Roll from an Accelerometer* (24 de Septiembre de 2012). Última consulta 2 de Diciembre de 2015, recuperado de http://theccontinuum.com/2012/09/24/arduino-imu-pitch-roll-from-accelerometer/

[14] *madc/GY-85 · Github* (27 de Diciembre de 2015). Última consulta 2 de Diciembre de 2015, recuperado de https://github.com/madc/GY-85

[15] *GY80/Compass.ino at master · vshymanskyy/GY80 · GitHub* (26 de Diciembre de 2012). Última consulta 2 de Diciembre de 2015, recuperado de https://github.com/vshymanskyy/GY80/blob/master/AHRS_9DOF_arduino/Compass.ino

[16] *Tilt Compensation Magnetometer (HMC5883L) - Parallax Forums* (19 de Diciembre de 2013). Última consulta 2 de Diciembre de 2015, recuperado de http://forums.parallax.com/discussion/152502/tilt-compensation-magnetometer-hmc5883l

[17] *The quadcopter: how to compute the pitch, roll and yaw* (19 de Septiembre de 2012). Última consulta 2 de Diciembre de 2015, recuperado de http://theboredengineers.com/2012/09/the-quadcopter-get-its-orientation-from-sensors/

GLOSARIO

CE	Chip Enable
CSN	Chip Select Not
GND	Ground
ID	Identifier
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
I²C	Inter-Integrated Circuit
Li-Po	Lithium Polymer
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
PCB	Printed Circuit Board
PET	Polyethylene Terephthalate
PLA	Polylactic Acid
PWM	Pulse Width Modulation
RF	Radio Frequency
SCK	Serial Clock
SCL	I²C Clock Line
SDA	I²C Data Line
SDK	Software Development Kit
SPI	Serial Peripheral Interface
WIFI	Wireless Fidelity

ANEXOS

1. Código programa principal Arduino

```
/*
Control Remoto de un Drone
Proyecto de Final de Grado.
EET UPC TERRASSA.
Autor: Jesús Mario Calleja Hernández
*/

#include <HCD.h>
#include <Wire.h>
#include <ADXL345.h>
#include <HMC5883L.h>
#include <Servo.h>

//IMU variables
ADXL345 acc;
HMC5883L compass;
int ax, ay, az;
int rawX, rawY, rawZ;
float X, Y, Z;
float rollDegrees, pitchDegrees, yawDegrees;
float lastPitchMove = 0.0, lastRollMove = 0.0;
float rollRadians, pitchRadians;
float Xh, Yh;
const float declinationAngle = 0.0457;
float yaw;
int pitchDrone, rollDrone, yawDrone;
int error = 0;
float aOffsetX = 0.0, aOffsetY = 0.0, aOffsetZ = 0.0;
float yawOffset = 0.0;

//Ultrasonic variables
long initialDistance = 0.0, distance, lastDistance,
throttleDrone;
const int triggerPin = 9;
const int echoPin = 7;

//Servo variables
Servo pitchServo, rollServo;
const int pitchServoPin = 5;
const int rollServoPin = 6;
```

```
const int servoMinPulse = 720;
const int servoMaxPulse = 2400;
const float servoAlpha = 1.5;

//Drone variables
HCD drone0;
unsigned char ID0[]={0x16,0x01,0x55,0x11};

//Led variables
const int ledPin = 2;

//Button variables
const int buttonPin = 3;
boolean connectionReady = false;

//Speaker variables
const int speakerPin = 4;
int onMelody[] = {1175, 1175};
int onMelodyDurations[] = {16, 16};
int connectedMelody[] = {587, 784, 1175};
int connectedMelodyDurations[] = {16, 16, 16};


void setup() {
  Serial.begin(19200);

  //Inputs/Outputs
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH);
  pinMode(buttonPin, INPUT);
  digitalWrite(buttonPin, HIGH);

  //Servos
  pitchServo.attach(pitchServoPin, servoMinPulse,
servoMaxPulse);
  rollServo.attach(rollServoPin, servoMinPulse,
servoMaxPulse);
  pitchServo.write(90);
  rollServo.write(90);

  //Ultrasound initial value
  pinMode(triggerPin, OUTPUT);
  pinMode(echoPin, INPUT);
  for(int i=0; i<100; i++){
    initialDistance += calcDistance();
  }
  initialDistance /= 100;

  //IMU (activation and offsets)
  acc.powerOn();
  compass = HMC5883L();
```

```
  compass.SetScale(1.3);
  compass.SetMeasurementMode(Measurement_Continuous);
  for (int i = 0; i <= 200; i++) {
    acc.readAccel(&ax, &ay, &az);
      aOffsetX += ax;
      aOffsetY += ay;
      aOffsetZ += az;
  }
  aOffsetX /= 200;
  aOffsetY /= 200;
  aOffsetZ /= 200;
  for (int i = 0; i <= 200; i++) {
    MagnetometerScaled scaled = compass.ReadScaledAxis();
    yawOffset += atan2(scaled.YAxis, scaled.XAxis);
  }
  yawOffset = yawOffset/200;

  playSong(speakerPin, onMelody, onMelodyDurations);

  delay(1000);
}


unsigned long timer=0;


void loop() {
  /***********************/
  /**** ACCELEROMETER ****/
  /***********************/
  acc.readAccel(&ax, &ay, &az);

  rawX = ax - aOffsetX;
  rawY = ay - aOffsetY;
  rawZ = az  - (255 - aOffsetZ);

  X = rawX / 256.00;
  Y = rawY / 256.00;
  Z = rawZ / 256.00;

  pitchDegrees = - (180 * (atan(Y / sqrt(X * X + Z * Z))) /
PI);
  rollDegrees = - (180 * (atan(X / sqrt(Y * Y + Z * Z))) /
PI);

  pitchDrone = map(pitchDegrees, -90, 90, 0, 255);
  rollDrone = map(rollDegrees, -90, 90, 0, 255);
```

```
  /**********************/
  /**** MAGNETOMETER ****/
  /**********************/
  MagnetometerScaled scaled = compass.ReadScaledAxis();

  X = constrain(X, -1, 1);
  Y = constrain(Y, -1, 1);

  rollRadians = asin(Y);
  pitchRadians = asin(-X);

  rollRadians = constrain(rollRadians, -0.78, 0.78);
  pitchRadians = constrain(pitchRadians, -0.78, 0.78);

  Xh = scaled.XAxis*cos(pitchRadians) +
scaled.ZAxis*sin(pitchRadians);
  Yh = scaled.XAxis*sin(rollRadians)*sin(pitchRadians) +
scaled.YAxis*cos(rollRadians) -
scaled.ZAxis*sin(rollRadians)*cos(pitchRadians);

  yaw = atan2(Yh,Xh);

  yaw += declinationAngle;
  yaw -= yawOffset;

  //For Processing
  yawDegrees = yaw * 180 / M_PI;
  if (yawDegrees < 0) yawDegrees += 360;
  if (yawDegrees > 360) yawDegrees -= 360;
  if(yawDegrees >= 0 && yawDegrees <= 40) yawDrone =
yawDegrees;
  if(yawDegrees > 40 && yawDegrees < 180) yawDrone = 40;
  if(yawDegrees >= 180 && yawDegrees < 320) yawDrone = -40;
  if(yawDegrees >= 320) yawDrone = yawDegrees - 360;
  yawDrone = map(yawDrone, -40, 40, 0, 255);



  /********************/
  /****** SERVO ******/
  /********************/
  if(abs(pitchDegrees - lastPitchMove) > 5){
    pitchServo.write(constrain(-pitchDegrees * servoAlpha, -
90, 90) + 90);
    lastPitchMove = pitchDegrees;
  }
  if(abs(rollDegrees - lastRollMove) > 5){
    rollServo.write(constrain(-rollDegrees * servoAlpha, -90,
90) + 90);
    lastRollMove = rollDegrees;
  }
```

```
  /********************/
  /**** ULTRASONIC ****`/
  /********************/
  throttleDrone = calcDistance();
  throttleDrone = constrain(throttleDrone, initialDistance,
initialDistance+1500);
  throttleDrone = map(throttleDrone, initialDistance,
initialDistance+1500, 0, 250);
  if(throttleDrone == 0 && lastDistance > 50) throttleDrone =
lastDistance;
  if(throttleDrone > lastDistance + 10 || throttleDrone <
lastDistance - 10) lastDistance = throttleDrone;
  else throttleDrone = lastDistance;



  /********************/
  /****** BUTTON ******/
  /********************/
  if(digitalRead(buttonPin)==LOW) {
    if(!connectionReady){
      connectionReady = true;
      drone0.bind(ID0);
      playSong(speakerPin, connectedMelody,
connectedMelodyDurations);
    }
    else updateYawOffset();
  }



  /********************/
  /****** DRONE ******/
  /********************/
  if(millis()>=timer){
    timer+=20;

drone0.update(throttleDrone,yawDrone,64,pitchDrone,rollDrone,6
4,64,0);
  }



  /********************/
  /**** PROCESSING ****/
  /********************/
  processingOutput();
}
```

```
int calcDistance(){
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(5);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  long _distance = pulseIn(echoPin, HIGH);
  return _distance;
}


void playSong(int pin, int notes[], int durations[]){
  for (int thisNote = 0; thisNote <= sizeof(notes);
thisNote++) {
    int noteDuration = 1000 / durations[thisNote];
    tone(pin, notes[thisNote], noteDuration);
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    noTone(pin);
  }
}


void updateYawOffset(){
  yawOffset = 0.0;
  yawDrone = 128;
  for (int i = 0; i <= 50; i++) {
    MagnetometerScaled scaled = compass.ReadScaledAxis();
    yawOffset += atan2(scaled.YAxis, scaled.XAxis);
  }
  yawOffset = yawOffset/50;
}


void processingOutput(){
  Serial.print(pitchDegrees);
  Serial.print(",");
  Serial.print(rollDegrees);
  Serial.print(",");
  Serial.print(yawDegrees);
  Serial.print(",");
  Serial.println(throttleDrone);
}
```

2. Código programa Processing

```
import processing.serial.*;

Serial port;
String x = "", y = "", z = "", up = "";
Float roll = 0.0, pitch = 0.0, yaw = 0.0, slide = 0.0;
PImage arrows, reference;
PFont font;

void setup(){
  size(800,600, P3D);

  arrows = loadImage("arrows.png");
  reference = loadImage("reference.png");

  font = loadFont("Helvetica-Light-48.vlw");
  textFont(font,48);

  port = new Serial(this, Serial.list()[1], 19200);
  port.bufferUntil('\n');
}

void draw(){
  background(255);
  textFont(font, 40);

  //REFERENCE TEXT
  fill(254, 210, 2);
  text("X: "+x,10,260);
  fill(26, 188, 156);
  text("Y: "+y,10,320);
  fill(252, 148, 3);
  text("Z: "+z,10,380);

  //REFERENCE ARROWS
  pushMatrix();
  imageMode(CORNERS);
  image(reference,10,10);
  popMatrix();

  //ARROWS
  pushMatrix();
  imageMode(CENTER);
  translate(width/2, height/2);
  rotateX(radians(pitch+90.0));
  rotateY(radians(roll));
  rotateZ(radians(yaw));
  image(arrows,0,0);
```

```
  popMatrix();

  //SLIDER
  line(width-50, 50, width-50, height-50);
  line(width-65, 50, width-35, 50);
  line(width-65, height-50, width-35, height-50);
  line(width-60, height/2, width-40, height/2);
  fill(0, 0, 0);
  rectMode(CENTER);
  rect(width-50, (255-slide)*2+45, 50, 10);
}

void serialEvent (Serial p){
  try{
    String data = p.readStringUntil('\n');
    if(data != null){
      data = data.substring(0, data.length()-1);
      String[] dataList = split(data, ',');

      y = dataList[0];
      x = dataList[1];
      z = dataList[2];
      up = dataList[3];

      pitch = Float.parseFloat(y);
      roll = Float.parseFloat(x);
      yaw = Float.parseFloat(z);
      slide = Float.parseFloat(up);
    }
  }
  catch(RuntimeException e) {}
}
```