
UNIVERSIDAD
DE MÁLAGA


**ESCUELA DE INGENIERÍAS INDUSTRIALES**

**Departamento de Tecnología Electrónica**

**Área de Conocimiento: Lenguajes y Sistemas Informáticos, Tecnología Electrónica y Electrónica.**

# TRABAJO FIN DE GRADO

## DESARROLLO DEL SOFTWARE PARA ADAPTACIÓN DEL RECEPTOR 3D AMFITECH EN DISPOSITIVOS PLUG&PLAY

Grado en

Ingeniería Electrónica Industrial

Autor: Álvaro José Ávila-Casanova Ureña

Tutor: Francisco José Sánchez Pacheco

MÁLAGA, 02 Enero de 2.020

# RESUMEN

El presente proyecto parte de una iniciativa por parte de la empresa PREMO para utilizar el kit de la empresa Amfitech (llamado Amfitrack), incorporarlo en su línea de productos y ofrecer al usuario final una herramienta de desarrollo para aplicaciones de posicionamiento 3D en entornos acotados.

Este dispositivo se integra dentro de la plataforma Arduino con una placa impresa modular que permite ampliar la capacidad de telecomunicación del microcontrolador (le llamaremos USB Host Shield) convirtiéndose en un dispositivo *Plug&Play* para poder conectar éste fácilmente a proyectos tecnológicos que precisen del posicionamiento espacial y ayudar al usuario a familiarizarse con la tecnología de seguimiento (del inglés, *tracking*), en concreto del kit Amfitrack. Para ello, durante el proyecto se desarrolló una librería de software libre en lenguaje C++ que será descargado por el usuario final del dispositivo teniendo el TFG un alto contenido de *software* y de protocolo de comunicación USB.

# ABSTRACT

This Project draws on an initiative from PREMO company to use Amfitech's kit (also known as Amfitrack), in order to incorporate it to their products's line and to offer the final user a development tool for 3D positioning applications in delimited environments.

This device is integrated in the Arduino platform with a shield board that allows for broading microcontroller's telecommunication capacity (we will call it USB Host Shield) turning it into a *Plug&Play* device to connect it easier to other technological projects that need this spatial tracking and to help the user to get used to tracking technology, specifically with Amfitrack. For that purpose, in this project was developed a free-software library in C++ language that will be downloaded by the final user of this device, so this thesis has a high content of software and USB protocol.

# ÍNDICE DE CONTENIDOS

# TABLA DE FIGURAS

# 1. MOTIVACIÓN Y OBJETIVOS

El presente trabajo fin de grado surge tras un primer contacto con la empresa Premo y Amfitech en la competición "*PREMO Motion Tracking for Virtual Reality Challenge*" de 2018 celebrada en la UMA. En ésta me introduje en el campo del tracking electromagnético ideando junto a mi equipo un sistema de posicionamiento 3D basado en el que actualmente posee PREMO y Amfitech. Posteriormente transformamos la idea en proyecto para crear una empresa emergente llamada SAADER (Sistema para el Aterrizaje Automático de Drones en Estaciones Remotas) y conseguimos uno de los premios Spin Off 2018 de la UMA. Todo este trabajo ha desembocado en mi TFG, cuyo objetivo es expandir la tecnología de Amfitrack a Arduino, creando un nuevo producto del tipo Plug&Play con *hardware* y *software* libre.

El objetivo de esta expansión es la de acercar la tecnología Electro-Magnetic Tracking Systems (desde ahora EMTS) de Amfitech y PREMO a los proyectos Arduino que precisen de un *tracking* de alta precisión, pudiendo ser utilizado por ejemplo en proyectos que abarquen el aterrizaje de drones, seguimiento de robots inteligentes en la industria 4.0, etc, y aumentar mis conocimientos en este tipo de tecnología para poder seguir perfeccionando el sistema SAADER junto a mis compañeros.

# 2. INTRODUCCIÓN

## 2.1. Principio físico de los EMTS

Los Sistemas Electromagnéticos se basan en los cálculos de posicionamiento descritos por la física de los campos magnéticos producidos por transductores o bobinas. Para producir un campo magnético estable puede utilizarse el ejemplo del campo ejercido por una espira circular en un punto cualquiera del espacio por su extendido uso y por la semejanza que tiene con una bobina comercial, si se crea un solenoide con la espira. Este ejemplo se encuentra publicado en la web de la *Universidad del País Vasco* [1]. Si se introduce una corriente I por dicha espira, se obtiene un vector campo magnético **B** (*inducción magnética* [2], [3]), medido en Teslas (T), según la *Ley de Biot* [4]. Dicho vector se encuentra en un punto del espacio P. La ley es la siguiente:

$$\boldsymbol{B} = \frac{\mu_0 i}{4\pi} \oint \frac{u_t x u_r}{r^2}\, dl \qquad (1)$$

Donde *dl* representa la longitud infinitesimal de la espira circular, $\mu_0$ es la permeabilidad magnética del vacío y *r* distancia del radio respecto a *dl*. El vector de esta distancia es normal a la longitud infinitesimal. $u_t$ es un vector unitario que indica el punto P, que puede estar en el eje coaxial de la espira o fuera de este, en un punto cualquiera del espacio. $u_r$ es un vector unitario que indica la posición de P respecto del elemento de corriente $\frac{\mu_0}{4\pi}$. μ0 es la permeabilidad magnética en el vacío, medida en Henrio (H). El campo producido tiene simetría axial. El modelo matemático para un punto cualquiera en el espacio se representa en la *figura 1*.

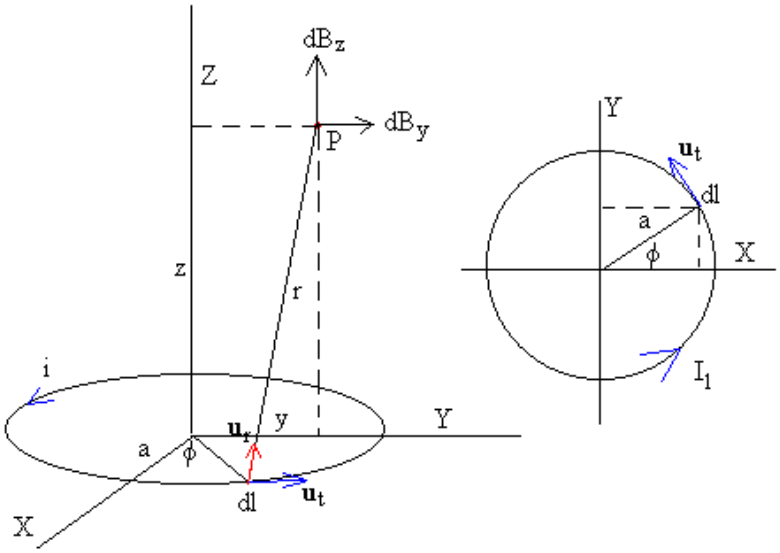



*Figura 1. Campo magnético producido por una corriente circular en un punto fuera del eje coaxial.*

Siendo:

$$r = \sqrt{\left(a^2 + y^2 + z^2 - 2ay\ sen(\varphi)\right)} \quad (2)$$

$$u_r = \frac{-a \cdot \cos(\varphi) \cdot \mathrm{i} + \left(\mathrm{y} - \mathrm{a}\ \mathrm{sen}(\varphi)\right) \cdot j + z \cdot k}{r} \quad (3)$$

$$u_t = -sen(\varphi) \cdot \mathrm{i} + \cos(\varphi) \cdot \mathrm{j} \quad (4)$$

Siendo las 3 componentes del campo magnético según las *ecuaciones (5), (6) y (7)*:

$$B_x = \frac{\mu_0}{4\pi} i \cdot a \cdot z \int_0^{2\pi} \frac{\cos(\varphi)}{\mathrm{r}^3} d\varphi \quad (5)$$

$$B_y = \frac{\mu_0}{4\pi} i \cdot a \cdot z \int_0^{2\pi} \frac{sen(\varphi)}{\mathrm{r}^3} d\varphi \quad (6)$$

$$B_z = \frac{\mu_0}{4\pi} i \cdot a \int_0^{2\pi} \frac{a - y \cdot sen(\varphi)}{\mathrm{r}^3} d\varphi \quad (7)$$

En este modelo se observa que el campo magnético depende de la inversa de la distancia r al cubo, es decir, a mayor distancia menor campo. Esta propiedad es relevante cuando el usuario debe elegir entre usar EMTS o Sistemas Ópticos "OPS" (del inglés, *Optical Systems*).

Para el diseño de un sistema de posicionamiento por magnetismo previamente se ha de estudiar el comportamiento del circuito como un dipolo magnético. Para ello se utilizan dos antenas: una emisora compuesta por **tres bobinas** isométricas y una segunda

receptora compuesta de otras tres **bobinas isométricas**. Cada una de ellas se comporta como un dipolo magnético. El modelo queda representado en la *figura 2*:

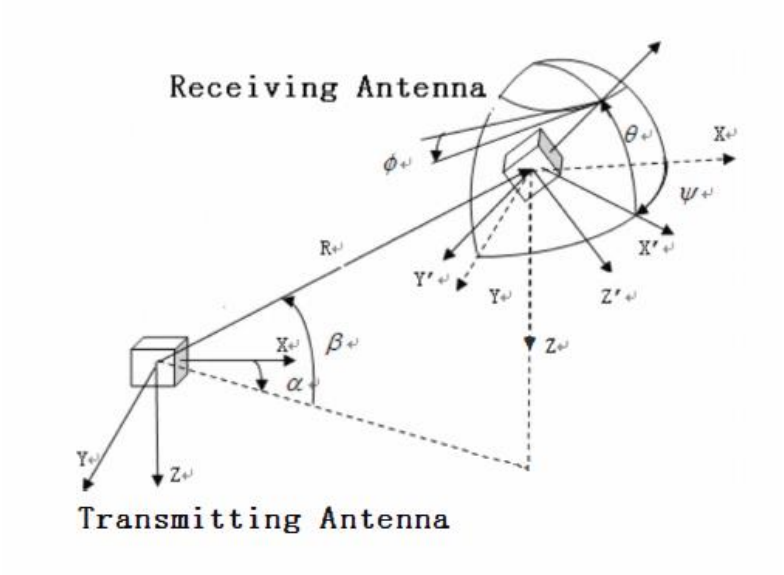



*Figura 2. Representación de las coordenadas de posición y orientación del modelo.*
*(Imagen obtenida de Research on Six-degree-of-freedom Electromagnetic Tracking System)* [5]

La antena receptora se define por los siguientes parámetros:

- R: Distancia de la antena receptora respecto a la antena emisora.
- α: Acimut.
- β: Ángulo de elevación.
- ψ: Ángulo de desviación. Utilizado para definir la orientación de la antena.
- θ: Ángulo del momento.

Los ejes de la antena emisora son excitados en la misma frecuencia y fase cuando es encendido su circuito. El patrón de dicha excitación se compone de 3 estados secuenciales, mientras que el patrón recibido se compone de 3 estados simultáneos que, para un período de excitación, la antena receptora recibe 9 señales, formando así la *matriz receptora* **Y**. La ecuación (8) define dicha matriz.

$$Y \cdot X^{-1} = -\frac{G}{R^3} H \cdot M \qquad (8)$$

## 2.2. Configuración del modelo de seis grados de libertad en los EMTS.

Esta configuración se adopta en los sistemas EMTS (*Electromagnetic Tracking Systems*) que usan el *método de la bobina-dipolo*, el cual usa corriente alterna de baja frecuencia en la generación de las bobinas de la antena emisora. El esquema del modelo está representado en la *figura 3*:

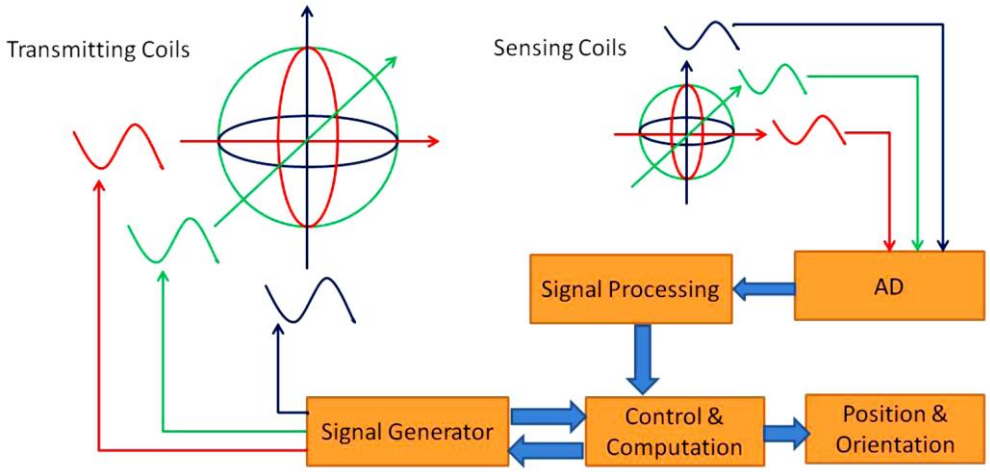



*Figura 3. Esquema de bloques del EMTS. (Imagen obtenida de Rotating Uniaxial Coil With Sparse Points and Closed Form Analytic Solution.* [6]

Este sistema se compone de un emisor formado por 3 bobinas isométricas (3 ejes), que son las generadoras del campo magnético y 3 bobinas isométricas (3 ejes) que funcionan como sensores. Estas últimas son capaces de detectar a las primeras utilizando métodos de acoplamiento de frecuencia. El sensor receptor detecta todas las señales del emisor magnético. Como puede verse en la *figura 3* el emisor transmite a tres frecuencias distintas de baja frecuencia.

Se parte de las siguientes coordenadas:

- Antena emisora: (a, b, c).
- Antena-sensor receptor: (x, y, z).
- Orientación del sensor: (α, β, γ).


*Figura 4. Coordenadas de la antena emisora (izquierda) y receptora (derecha).*

Simplificando las *figuras 3* y *4* se obtiene el siguiente modelo teórico:

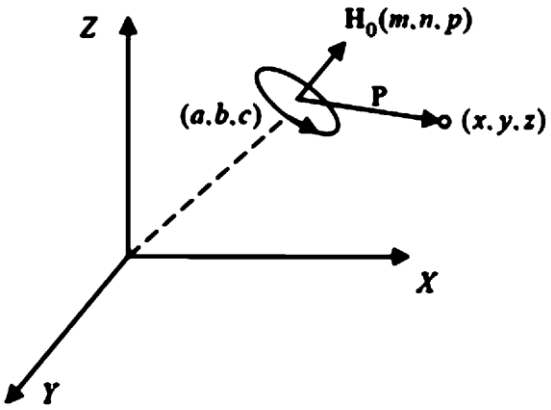



*Figura 5. Modelo teórico del EMTS simplificado. (Imagen obtenida del documento Electromagnetic Motion Tracking. Fundamentals - Algorithm - Basic Driving Electronics PREMO).* [7]

La antena emisora se comporta como un dipolo magnético cuando está alimentada, y la dirección a la que apunta viene indicada por la *ecuación (9)*:

$$H_0 = \begin{pmatrix} m \\ n \\ p \end{pmatrix} \quad (9)$$

Para calcular la posición del receptor, previamente ha de calcularse el campo magnético:

$$B = B'_x\, i + B'_y\, j + B'_z\, k \quad (10)$$

Donde $B'_x$, $B'_y$ y $B'_z$ son las componentes de B a lo largo de los ejes isométricos. Se tiene que el campo magnético B es:

$$B = \frac{\mu_r \mu_0 M_T}{4\pi}\left(\frac{3(H_0 \cdot P)P}{R^5} - \frac{H_0}{R^3}\right) \quad (11)$$

Siendo:

- MT: Constante que define la intensidad magnética del dipolo.
- $P = \begin{pmatrix} x-a \\ y-b \\ z-c \end{pmatrix}$, punto receptor respecto al origen del dipolo (antena emisora).
- $R = \sqrt{(x-a)^2 + (y-b)^2 + (z-c)^2}$, define la longitud del sensor receptor.

Las componentes del campo emisor quedarían según la *ecuación (12)*:

$$B'_X = B_T\left(\frac{3[m(x-a)+n(y-b)+p(x-a)]}{R^5}-\frac{m}{R^3}\right)$$
$$B'_y = B_T\left(\frac{3[m(x-a)+n(y-b)+p(y-b)]}{R^5}-\frac{n}{R^3}\right) \quad (12)$$
$$B'_z = B_T\left(\frac{3[m(x-a)+n(y-b)+p(z-c)]}{R^5}-\frac{p}{R^3}\right)$$

Donde:

$$B_T = \mu_r\mu_0\frac{M_T}{4\pi} \quad (13)$$

Este campo magnético es captado por el sensor receptor. El receptor capta las 3 frecuencias, obteniéndose las 9 señales comentadas en el *apartado 2.1*. De ellas se obtienen las 9 ecuaciones de campo magnético y con una serie de cálculos algebráicos se obtiene la matriz de rotación **R**. Esta poseerá la posición y orientación del receptor en el espacio.

## 2.3. Arquitectura del sistema EMTS utilizado en Amfitrack.

La arquitectura utilizada en el sistema electromagnético utilizado por el kit Amfitrack se muestra en la *figura 6*, la cual es muy similar al esquema de la *figura 3*.

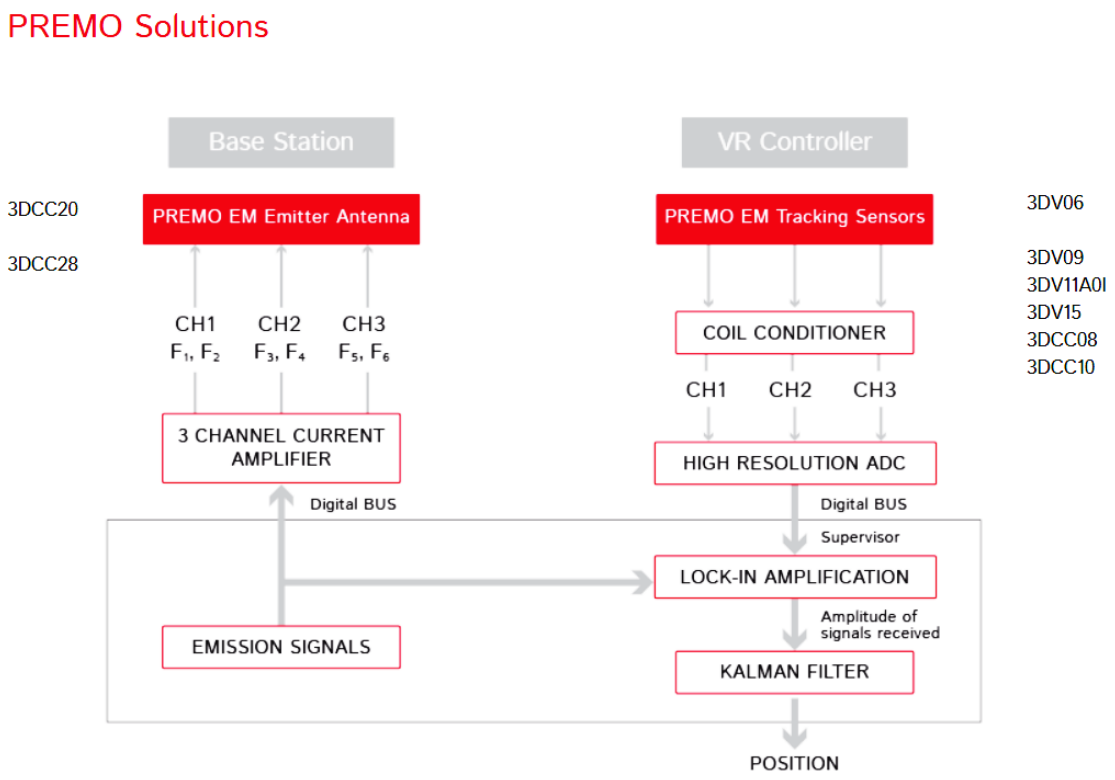



*Figura 6. Arquitectura del EMTS de PREMO-Amfitech. (Imagen obtenida de la web de Grupo PREMO).*

El área acotada por el contorno de color gris representa el "supervisor", el cual se encarga de controlar las señales que emite la antena emisora y las recibidas por el sensor receptor. Este supervisor puede diseñarse utilizando una FPGA. Para ejercer dicho control utiliza el bus de datos (DB) para enviar las órdenes, además de utilizar la técnica "Lock-in amplification" para extraer por modulación de frecuencia la amplitud de señal y eliminar el ruido de cada señal del sensor [8]. Estas 3 señales de baja frecuencia son previamente digitalizadas por un dispositivo ADC de alta resolución. Por último, el supervisor tiene un filtro de tipo Kalman [9], [10] que sirve para estimar con precisión la posición y orientación absoluta durante el movimiento relativo del emisor.

Este no es el único diseño que existe para los EMTS, pero sí es el más eficiente ya que mejora la resolución que es capaz de captar el supervisor. Un ejemplo de diseño es el de la *figura 5* en el *apartado 2.2*, o el diseño de la *figura 7*.

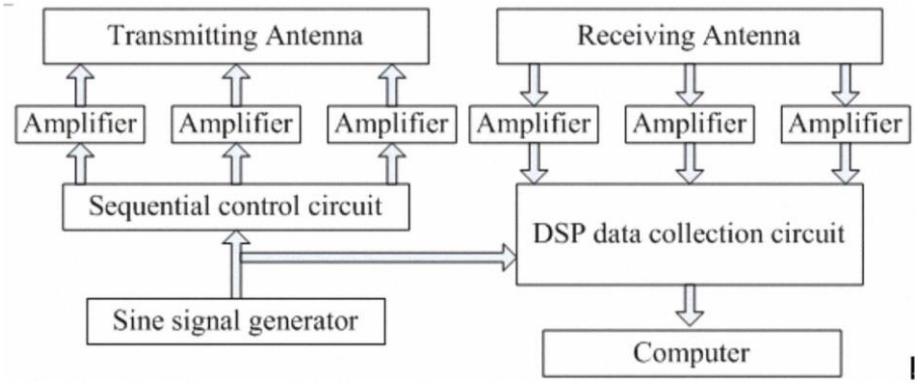



*Figura 7. Esquema de bloques del EMTS desarrollado en el estudio Research on Six-degree-of-freedom Electromagnetic Tracking System.*[5]

## 2.4. Kit Amfitrack

El kit utilizado en el presente TFG fue desarrollado por la empresa Amfitech con tecnología del Grupo Premo. Este está formado por los siguientes dispositivos:

- Fuente EMF: encargada de emitir el campo electromagnético.
- Driver de la fuente EMF: se alimenta a 5V a través del puerto USB. Se comunica con el sensor EMF y el dispositivo Hub RF (radiofrecuencia) usando 2.4GHz.
- Sensores EMF: Son 3. Recogen el campo electromagnético y calculan sus respectivas posiciones con 6 grados de libertad desde la fuente EMF. Estos se comunican con el driver de la fuente EMF y el USB RF Hub usando radiofrecuencia a 2.4GHz. Solo uno de ellos provee el filtro Kalman.
- USB RF Hub: Recoge los 2.4GHz de comunicación entre la fuente EMD y el sensor EMF, y transmite los datos de posición del sistema a través de su puerto USB al PC.
- Plug-in de 5V USB de alimentación: puede usarse para alimentar el driver de la fuente EMF. Puede además usarse para cargar los sensores EMF.
- Adaptadores plug-in de 5V USB: permiten usar el Plug-in de alimentación en distintos países.
- 2 cables USB – Micro USB.

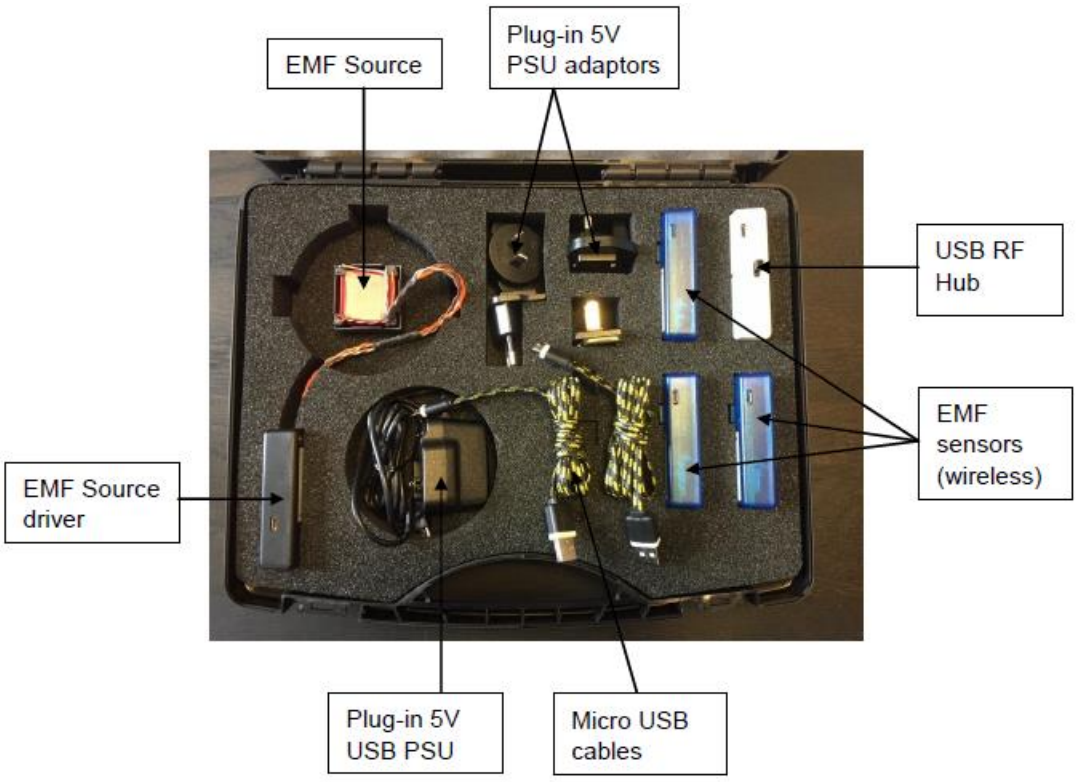



*Figura 8. Fotografía del kit Amfitrack.*

# 3. ANTECEDENTES

## 3.1. Tecnología OPS y EMTS

La tecnología Motion Tracking (del español Trazabilidad del Movimiento) tuvo sus orígenes a inicios de 1980 cuando Tom Calvert, profesor de la Simon Fraser University, creó un sistema de captura de movimiento (lo llamaremos Mo-Cap, del inglés Motion Capture [11]) colocando potenciómetros en el cuerpo de un actor para registrar sus movimientos con el fin de utilizarlos en sus estudios de animación por computador. En 1983 los investigadores Ginsberg y Maxwell del MIT utilizaron el innovador sistema Op-Eye para el desarrollo del sistema "scripting-by-enactment" [12]. Este sistema fue un adelanto en la tecnología OTS (Optical Tracking Systems [13]), esta utilizaba leds unidos al cuerpo del sujeto y una serie de cámaras colocadas alrededor grababan el movimiento y calculaban la posición de los distintos leds en el espacio.

No fue hasta 1989 que se creó la primera película utilizando imágenes generadas por ordenador o CGI (del inglés Computer-Generated-Imagery) con la película "Don't Touch Me". Y es a partir del 2000 que el CGI y el Mo-Cap comenzaron a usarse de forma extendida en la industria del cine y los videojuegos quedando como obra de referencia la película "Avatar" de James Cameron [14].

Desde entonces se sigue innovando y desarrollando alternativas en el área de los Motion Tracking Systems, explorando nuevas tecnologías para reducir costes y mejorar la precisión. Naciendo así la tecnología EMTS cuya ventaja es el reducido coste y la inexistencia de puntos ciegos con respecto a la tecnología OTS. Empresas como PREMO Group están actualmente trabajando en la tecnología EMTS.

# 4. ESTADO DEL ARTE

## 4.1. Aplicaciones de la tecnología Electromagnetic Tracking Systems

La tecnología EMTS está actualmente en pleno auge gracias al desarrollo de la industria de la realidad virtual (la llamaremos RV), la realidad aumentada (la llamaremos RA) y de la industria médica [15]. Los dispositivos más utilizados en la industria médica se pueden dividir en 3 grupos [16]:

- *Tracking de Corriente Alterna*.
- *Tracking de Corriente Continua.*
- *Sistemas pasivos o transpondedores.*

Los sistemas de *tracking* con corriente continua o alterna sirven para la localización de herramientas médicas fuera y dentro del cuerpo humano, muy útiles durante las operaciones. Los problemas de distorsión debido a cuerpos magnéticos no afectan a los sistemas basados en corriente continua. Por otro lado, los transpondedores EMTS son pequeños sensores que pueden implantarse dentro del tejido blando del cuerpo para trazar, por ejemplo, tumores.

En lo que respecta al *motion tracking* en la industria de la realidad virtual [17], su objetivo es trazar en tiempo real todos los movimientos de un sujeto u objeto para que la acción pueda ser reproducida en un medio digital [18]. Esto sirve en la grabación de películas que precisen efectos especiales y en videojuegos de RV donde el jugador utiliza un avatar que sigue sus movimientos. Por otro lado, este sistema de seguimiento del movimiento se puede utilizar para aplicaciones de RA como las utilizadas con el sistema Kinect [19], pero utilizando tecnología EMTS.

*Figura 9. Control para videojuegos con 3Dcoil de PREMO (imagen propiedad de Grupo Premo)*

# 5. METODOLOGÍA Y RESULTADOS

## 4.1. Presentación del modelo

A continuación, se muestra el esquema de conexionado para el dispositivo Plug&Play:

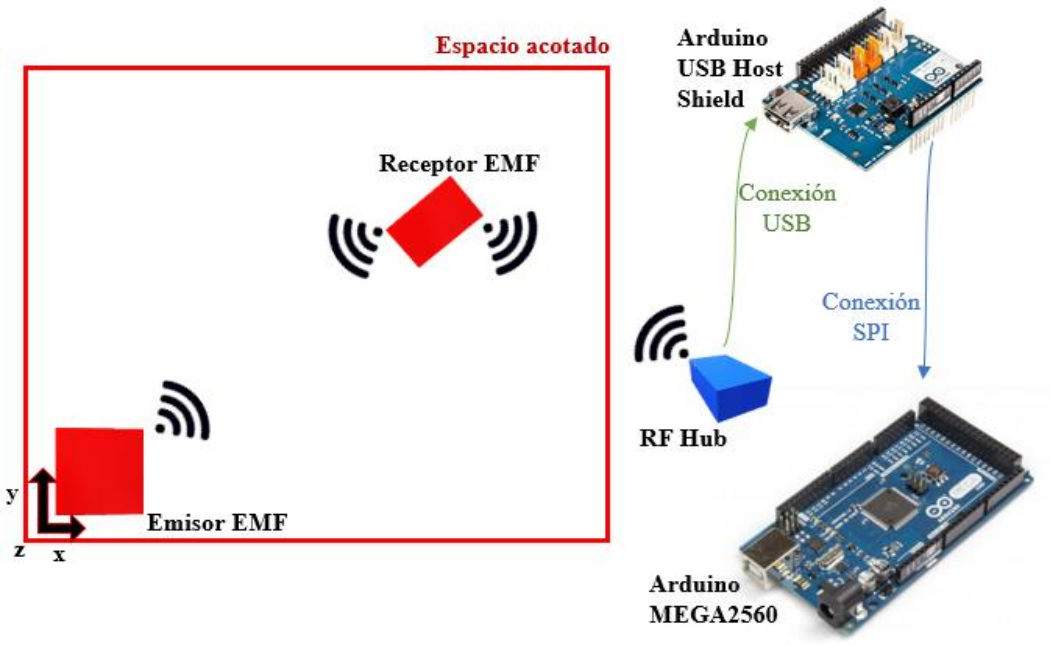



*Figura 10. Modelo del proyecto desarrollado.*

La fuente emisora EMF emite hasta cierta distancia creando un espacio acotado, el receptor EMF capta la señal del emisor, calcula su propio posicionamiento tridimensional respecto del emisor y manda las coordenadas al dispositivo RF Hub. Este las transmite por conexión USB al USB Host Shield que está montado encima del microcontrolador Arduino MEGA 2560. Los paquetes de datos son transmitidos al microcontrolador a través de las patillas SPI. Este paquete de datos es decodificado con el software desarrollado en el presente TFG estando listo para ser utilizado por los usuarios de Arduino.

## 4.2. Estrategia seguida en el proyecto

Para afrontar el reto tecnológico se optó por seguir la siguiente estrategia:

i. Estudiar la tecnología del kit Amfitrack.
   En primer lugar, se estudió los principios físicos en los que se fundamentan el emisor y receptor para la realización del tracking y, además, las distintas implementaciones que tienen, descritas en el capítulo de Estado del Arte.

ii. Estudiar el tipo de comunicación a utilizar en el proyecto.
En este punto se abordó el tipo de comunicación que se utilizaría en el proyecto para conectar el dispositivo Hub con la placa Arduino. A petición de PREMO se optó por la comunicación USB.
Se llegó a la conclusión de que esta era la mejor solución debido a la velocidad de transmisión de información (Full Speed: 12Mbps), por el extendido uso de la comunicación USB en la actualidad y por el reciclaje de código ya existente para la conexión del dispositivo Hub con el ordenador. Esta comunicación USB es de la clase HID (Human Interface Device) [20] el cual está muy extendido en dispositivos del tipo mouse, teclados e instrumental médico.

iii. Búsqueda del hardware necesario para la realización del proyecto y estudio de su implementación.
Debido a que el microcontrolador Arduino MEGA2560 se comporta como esclavo (lo llamaremos *device* para seguir el lenguaje del protocolo USB) al igual que el dispositivo Hub, se llegó a la conclusión de que era necesario el uso del USB Host Shield para Arduino y su correspondiente librería de software libre. Esta nueva placa convierte al microcontrolador Arduino en maestro (lo llamaremos *host* para seguir la jerga utilizada en el protocolo USB). Ambas placas de Arduino se conectan por SPI.

iv. Análisis del código Python de Amfitech para su implementación en la placa Arduino.
Antes del desarrollo de las librerías, el cuál es más complejo debido al estudio previo que se tenía que realizar sobre esto, se decidió desarrollar la función encargada de decodificar el paquete de datos para traducirlo a posicionamiento de manera que fuese útil para el futuro usuario de Amfitrack.

v. Análisis de la librería USB Host Shield 2.0 de Cirtuits@Home.
Previamente a desarrollar la librería para el desarrollo del dispositivo Plug&Play se necesitó analizar y estructurar la librería que utiliza el USB Host Shield de Arduino. Una vez estudiada se localizaron las funciones y variables necesarias para la transmisión del paquete de datos.

vi. Implementación de la librería ArduPREMO.
Una vez realizados los pertinentes estudios se procedió a escribir el código C++, creando para ello los correspondientes archivos header (extensión .h) y cplusplus (extensión .cpp) en el compilador Code::Blocks y la parte de código en el IDE de Arduino donde encuentra el ejemplo para ayudar al usuario a implementar el archivo.

vii. Pruebas experimentales.

Una vez compilado el código se procedió a realizar las correspondientes pruebas para confirmar que las posiciones enviadas por el receptor a Arduino eran las correctas. Así se probaría el extracto de código encargado de traducir los paquetes de bytes en posicionamiento e inclinación 3D. Para ello se dibujaron sobre un plano de tamaño A2 distintas posiciones en un radio de 1 metro. Esta prueba sobre el plano se repitió a dos alturas distintas respecto al emisor para poder probar así el posicionamiento tridimensional. Esto se realizó tanto con Arduino como con el kit de Amfitech y su programa "Amfitrack Viewer".

## 4.3. Entorno de trabajo.

Para desarrollar el presente proyecto se han utilizado las siguientes herramientas.

### 4.3.1. Arduino

Arduino es una plataforma de electrónica "open-source" basada en simplificar el uso de *hardware* y *software*. Esta plataforma tiene una serie de productos llamadas Arduino Boards con los cuáles se pueden hacer proyectos electrónicos dentro de un marco académico y de innovación [21]. Por otro lado, la plataforma Arduino se compone por un foro específico para el desarrollo de sus placas.

#### 4.3.1.1. Placas de Arduino

Arduino proporciona en su catálogo un gran número de placas, pero para este proyecto se ha optado por el uso de dos:

- Arduino MEGA 2560.


*Figura 11. Placa microcontrolador Arduino MEGA 2560. (Imagen propiedad de Arduino).*

- Arduino USB Host Shield.

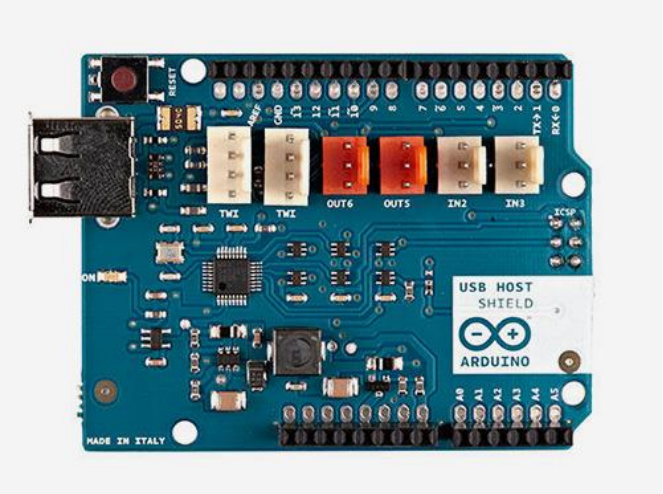



*Figura 12. Placa USB Host de Arduino. (Imagen propiedad de Arduino).*

Arduino MEGA 2560 [22] es una de las placas que más memoria RAM y memoria para *sketchs* contiene, además de ser una de las más utilizadas junto al Arduino UNO por la comunidad de usuarios Arduino, siendo ésta última característica la que hizo que se escogiese Arduino MEGA 2560 como microcontrolador para el proyecto. La placa Arduino USB Host Shield [23] se conecta al Arduino MEGA 2560 montándolo encima de esta, encajando así las patillas del USB Host Shield con las del MEGA 2560. Esta placa se encarga de expandir el hardware del microcontrolador MEGA 2560 para permitirle participar como *host* en las comunicaciones vía USB. La comunicación entre ambas placas se realiza mediante las patillas SPI. Este *shield* se basa en el controlador device/host MAX3421E de Maxim Integrated [24].

#### 4.3.1.2. Arduino IDE

Uno de los entornos de programación utilizados en el presente proyecto es Arduino IDE [25], un entorno de desarrollo escrito en Java y basado en *software* libre. Es un IDE (del inglés Integrated Development Environment) muy sencillo de utilizar y está enfocado en proyectos para placas Arduino. En este trabajo se utilizó para el análisis de los ejemplos existentes en la librería USB para Arduino y para crear parte del código de programación.

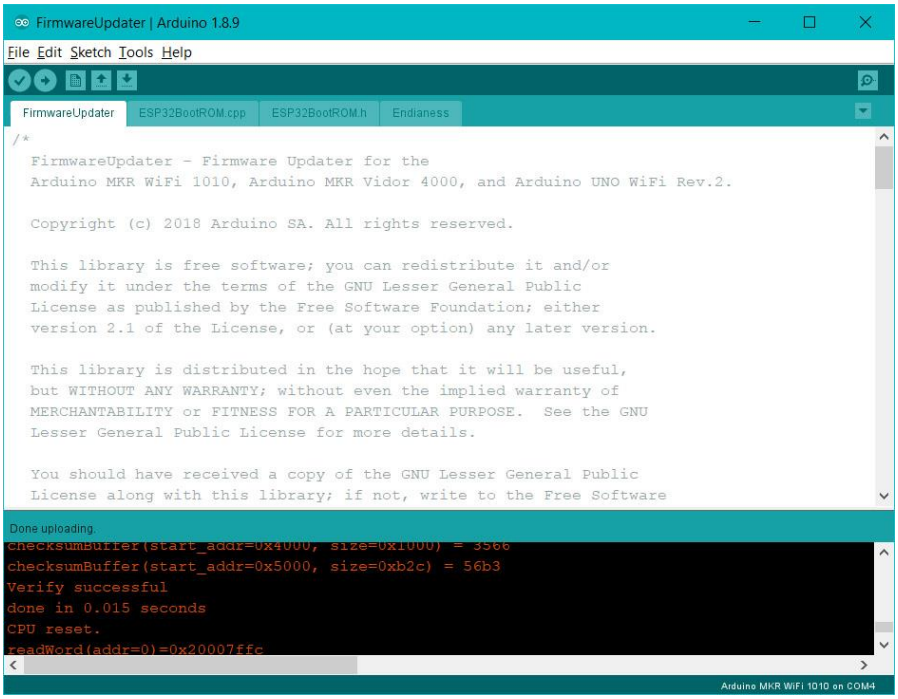



*Figura 13. Entorno de programación de Arduino. (Imagen propiedad de Arduino).*

#### 4.3.1.3. Librería USB HOST SHIELD 2.0 de Arduino

La librería USB Host Shield 2.0 para Arduino fue creado por un grupo de desarrolladores y compartida a través de diferentes plataformas de desarrollo de *software* como GitHub. Esta librería permite el uso a los usuarios de la placa USB host de Arduino para que puedan incorporarla en sus proyectos.

### 4.3.2. JetBrains PyCharm

PyCharm [26] es un IDE para lenguaje Python desarrollado por la compañía JetBrain. Este entorno fue utilizado para analizar el código del Amfitrack facilitado por PREMO, el cual se utiliza como parte del *software* de la interfaz del programa de Amfitech llamado Amfitrack Viewer.

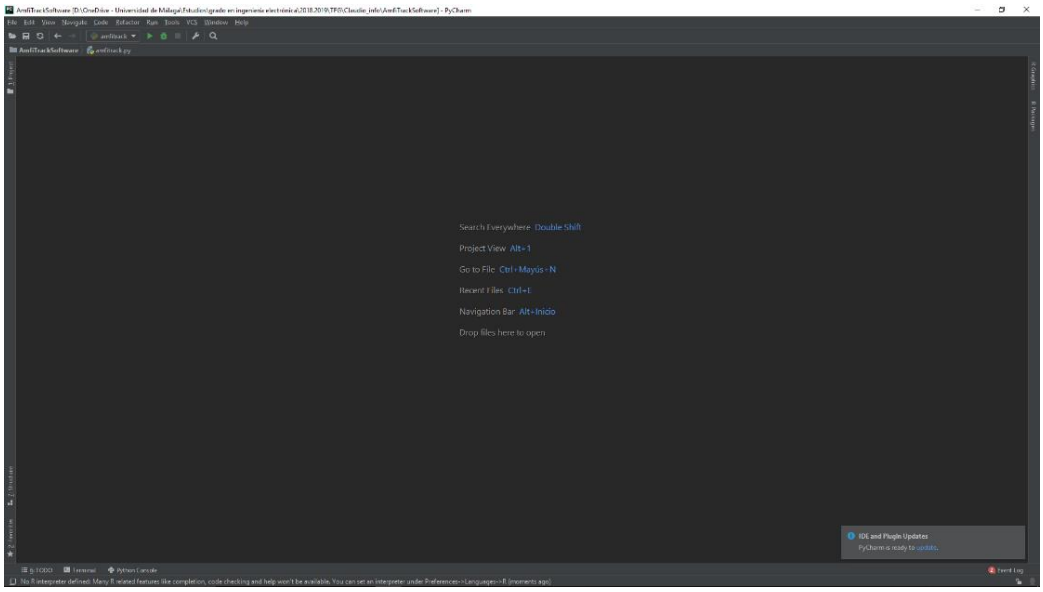



*Figura 14. Entorno de programación de PyCharm.*

### 4.3.3. Code::Blocks

Code::Blocks [27] es un IDE desarrollado en C++ y orientado a la programación C, C++ y Fortran. Dicho entorno se ha utilizado para la creación de las librerías que han sido desarrolladas en el presente TFG.

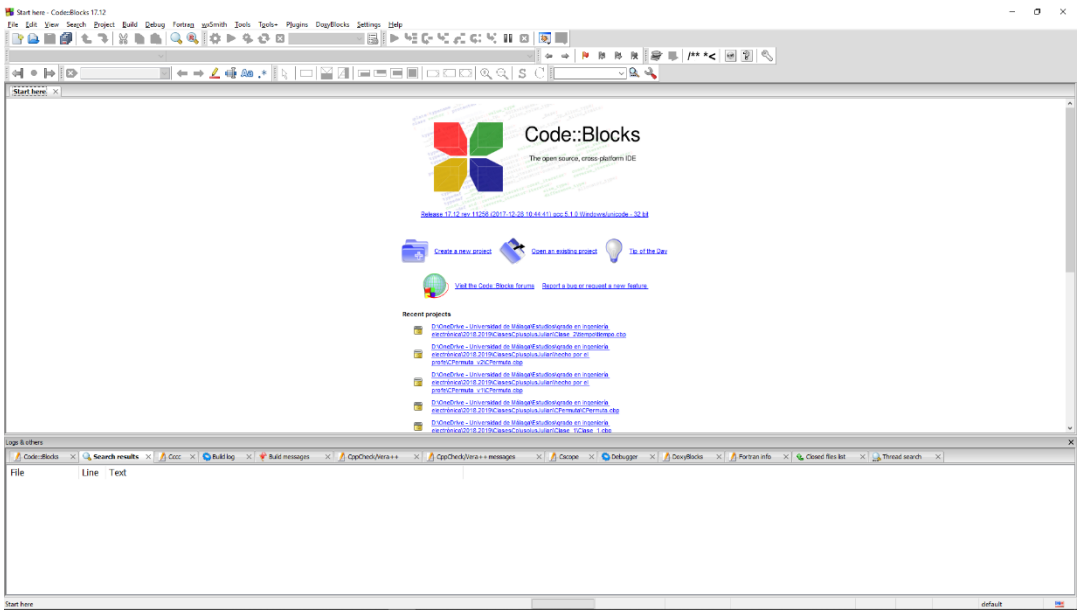
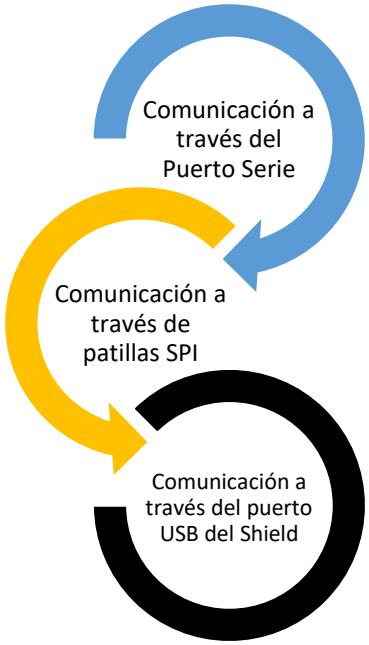



*Figura 15. Entorno de programación de Code::Blocks.*

## 4.4. Descripción de la comunicación del hardware

Para saber cómo se comunica el Hub Amfitrack con el Arduino utilizado en el proyecto se estudió el hardware que lo hace posible. Para entenderlo, en esta memoria se dividirá en tres bloques la comunicación:




Hay que indicar que, aunque el diagrama comienza por la comunicación por puerto serie, este proceso es bidireccional.

### 4.4.1. Comunicación a través del Puerto Serie

La comunicación por Puerto Serie es muy utilizada en los proyectos con Arduino debido a que es necesaria para poder cargar el programa en la memoria *flash* de la placa Arduino, además de utilizarse para la comunicación entre una computadora y Arduino. Es decir, el Puerto Serie se utiliza como puerto de consola. Este puerto es del tipo USART (Universal Synchronous/Asynchronous Receiver Transmitter) para los Arduinos con ATMega328p o con ATMega2560 como el utilizado en el presente TFG. USAR es un protocolo empleado en comunicaciones duales [28], y al igual que UART utiliza protocolos de comunicación como el RS-232.

Por otro lado, este módulo está conectado a una interfaz USB con una toma del tipo B.

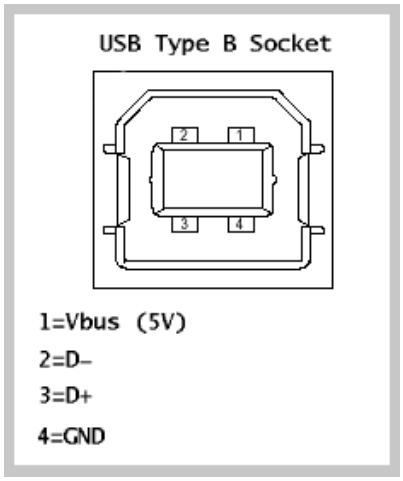



*Figura 16. USB Socked Type B. (Imagen propiedad del blog Aprendiendoarduino).*

Para poder unir ambos sistemas de comunicación, las placas Arduino poseen un segundo microcontrolador (ATMega 16u2) que dispone de interfaz USB y de un *firmware* para conseguir la conversión entre sistemas de comunicación [29]. Con ello la placa Arduino puede ser conectada al ordenador siempre que este tenga instalados los correspondientes *drivers*. El programa ejemplo diseñado para el usuario necesita este bloque de comunicación para poder imprimir por pantalla los datos de posicionamiento calculados (Ver punto 3.5.2).

### 4.4.2. Comunicación a través de patillas SPI

La comunicación entre el Arduino MEGA2560 y la USB Host Shield Arduino se realiza a través de las patillas SPI (Serial Peripheral Interface). Este es un protocolo de comunicación síncrono y *full duplex* basado en cuatro señales, dividiendo las líneas según sea para datos o para el reloj. Esta comunicación se gestiona a través del dispositivo maestro, en el caso de este proyecto el maestro es el controlador host MAX3421E que posee el Shield y que se comunica con el esclavo, el ATMega2560.

Normalmente, hay tres líneas comunes a todos los dispositivos:


*Figura 17. Patillas de conexión SPI de Arduino. (Imagen propiedad de Arduino.cc)*

**MISO**. Master In Slave Out. Línea de transmisión de datos del dispositivo esclavo hacia el dispositivo maestro.

**MOSI**. Master Out Slave In. Línea de transmisión de datos del dispositivo maestro hacia el dispositivo esclavo.

**SCK**. Serial Clock. Pulsos del reloj que sincronizan la transmisión de datos generado por el dispositivo maestro.

Y una línea específica para cada dispositivo:

**SS**. Slave Select. Patilla que tiene cada dispositivo esclavo para ser activado o desactivado por el dispositivo maestro.

Un ejemplo de comunicación SPI se puede observar en la siguiente figura:

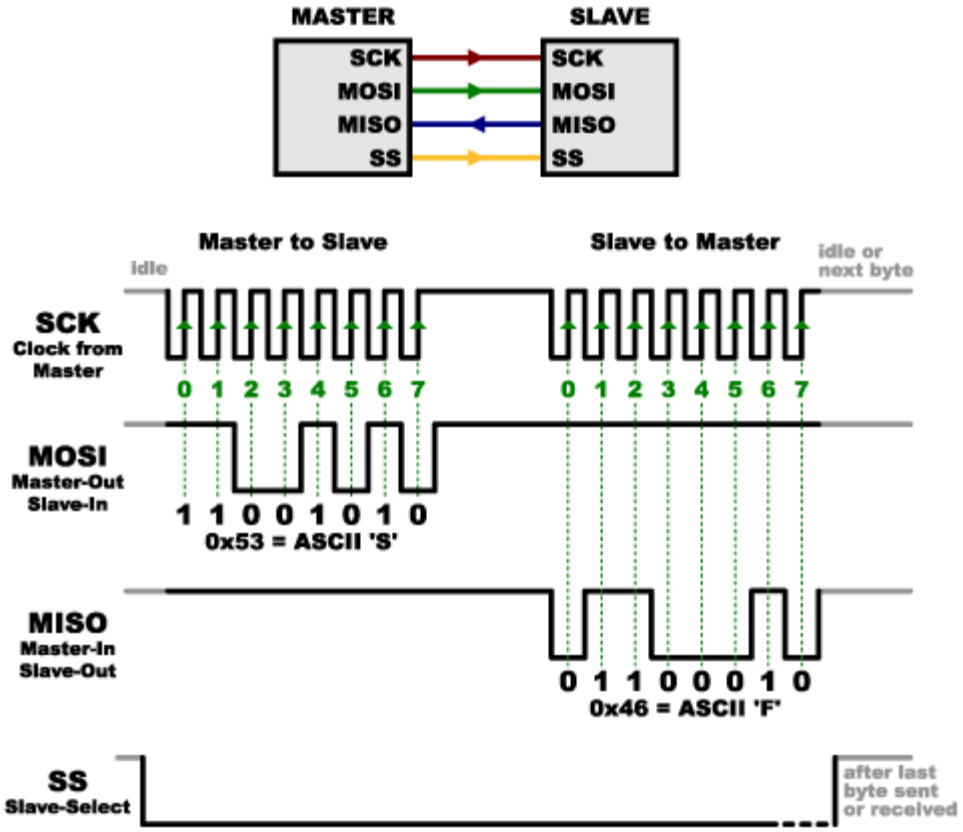



*Figura 18. Ejemplo de comunicación SPI. (Imagen propiedad de Aprendiendoarduino).*

De esta forma, el Shield y el Arduino MEGA2560 se envían los paquetes de datos, según establezca el programa que se haya cargado en el microcontrolador.

### 4.4.3. Comunicación a través del puerto USB del Shield

El tercer y último bloque se centra en la comunicación de la interfaz USB que posee el Shield y que es conectado al Hub Amfitrack. Esta comunicación la controla el controlador *host* MAX3421E utilizando las librerías USB Host Shield 2.0, las cuáles se basan en la programación a bajo nivel de dicho controlador. Esta documentación se encuentra en la web de MAX Integrated [30]. El conexionado es el descrito en el punto 4.4.1, con la diferencia de que el enchufe que posee el Shield en vez de ser del tipo B es del tipo A.

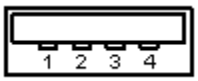



*Figura 19. USB socked type B. (Imagen de Usb in a NutShell).*

La velocidad de comunicación es FullSpeed o 12Mbits/s. La comunicación USB depende del contenido de las transacciones, que están divididas en cuatro tipos de paquetes diferentes [31]:

- **Token Packet**. Es la cabecera de la transacción y define qué es lo que viene en el paquete. Este lo genera el *host* para describir si la transacción es de lectura o escritura y qué dirección de dispositivo y *endpoint* han sido designados. Este tipo de paquete está formado por tres tipos de paquetes distintos:
  - **In**. Informa al dispositivo USB que el *host* va a leer información.
  - **Out**. Informa al dispositivo USB que el *host* va a enviar información.
  - **Setup**. Usado para comenzar transferencias de control.

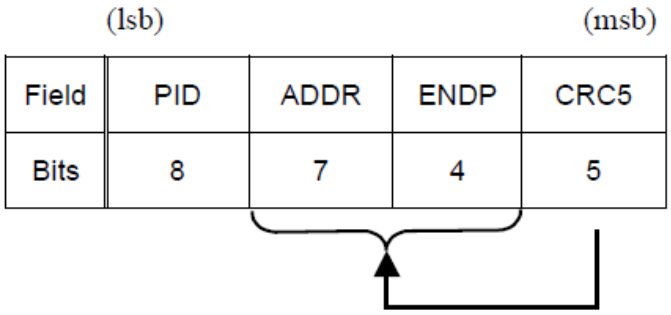



*Figura 20. Formato del Token Packet por usb.org .*

- **Optional Data Packet**. Paquete de datos que sigue al Token Packet y que contiene el *payload*. Existen dos tipos de paquetes de datos, cada uno capaz de transmitir 1MB.
    - **Data0**.
    - **Data1**.

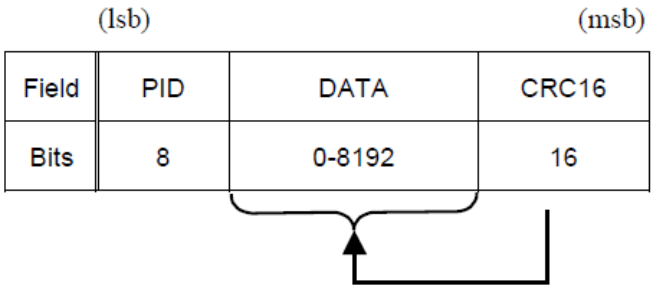



*Figura 21. Formato del Data Packet por usb.org .*

- **Status Packet**. Es el *handshake* y se usa para determinar si existe algún error en la transmisión del paquete o si se ha transmitido correctamente. Existen tres tipos de *handshakes* en *FullSpeed*:
    - **ACK.** Acknowledgment (del inglés, conocimiento o constancia). El paquete fue correctamente recibido. Es una confirmación.
    - **NAK**. No Acknowledgment (del inglés, desconocimiento). El paquete no ha sido enviado o recibido. Se usa también en las transacciones por interrupción para informar que el host no tiene datos para enviar.
    - **STALL**. El dispositivo necesita intervención del *host*. Si el *host* no interviene, no será posible la comunicación. Esta clase de error apareció durante el desarrollo de la librería ArduPREMO. Para evitar esta espera por parte del dispositivo Hub Amfitrack se optó por realizar un *powercycle* para así obligar a realizar una reenumeración. De esta forma se eliminaba cualquier clase de error STALL o NAK.

(lsb) (msb)

| Field | PID |
|---|---|
| Bits | 8 |

*Figura 22. Formato del Token Packet por usb.org .*

- **Start of Frame Packets (SOF)**. Indica el comienzo de un nuevo *frame*. Estos paquetes son emitidos por el *host* y consiste en un PID (Packet Identifier) indicando el tipo de paquete seguido por un número de frames de 11b:

(lsb) (msb)

| Field | PID | FrameNumber | CRC5 |
|---|---|---|---|
| Bits | 8 | 11 | 5 |

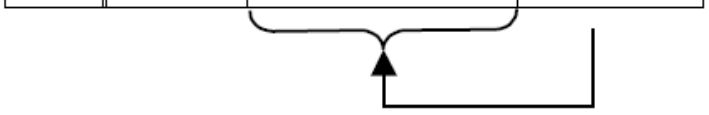


*Figura 23. Formato del SOF Packet por usb.org .*

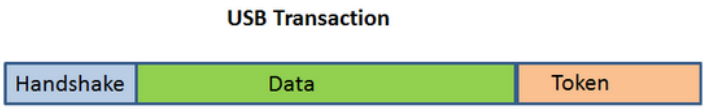



*Figura 24. Disposición de los paquetes dentro de una transacción USB. (Imagen de Microchip Developer Help).*

Los paquetes sirven para que tanto el *host* como el *device* consigan comunicarse siguiendo un orden. Para establecer ese orden se utilizan los e*ndpoints*. Estos se pueden entender como canales de datos, pueden existir hasta 32 *endpoints* y se cuentan por pares ya que uno es IN (entrada a *Host*) y el otro OUT (salida desde el *Host*). El *endpoint* 0 lo tienen todos los dispositivos debido a que su función es la de recibir todas las peticiones de *control* y *status* durante la fase de enumeración y mientras el dispositivo esté operativo en el bus.

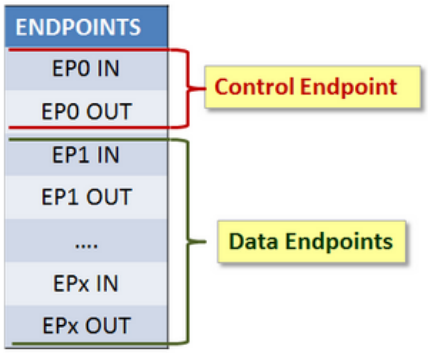



*Figura 25. Disposición de los Endpoints. (Imagen de Microchip Developer Help).*

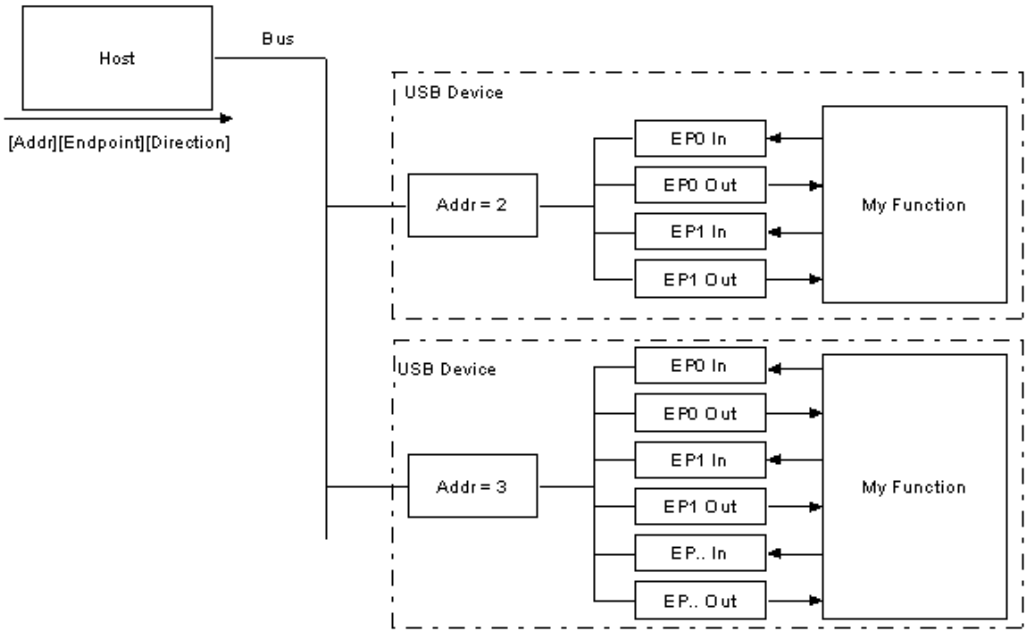



*Figura 26. Conexionado teórico entre el bus y el dispositivo USB. (Imagen propiedad de USB in a NutShell).*

Una vez visto esto, se procede a explicar la fase de enumeración. Esta fase es un proceso donde el *Host* detecta la presencia de un dispositivo y enumera una serie de pasos para asegurar que los *endpoints* del dispositivo son añadidos a la lista de *endpoints* para el *Host* [32]. Una vez la enumeración llega al punto de Dispositivo Configurado (o Running, dependiendo del desarrollador del *firmware*) el dispositivo está preparado para usarse.

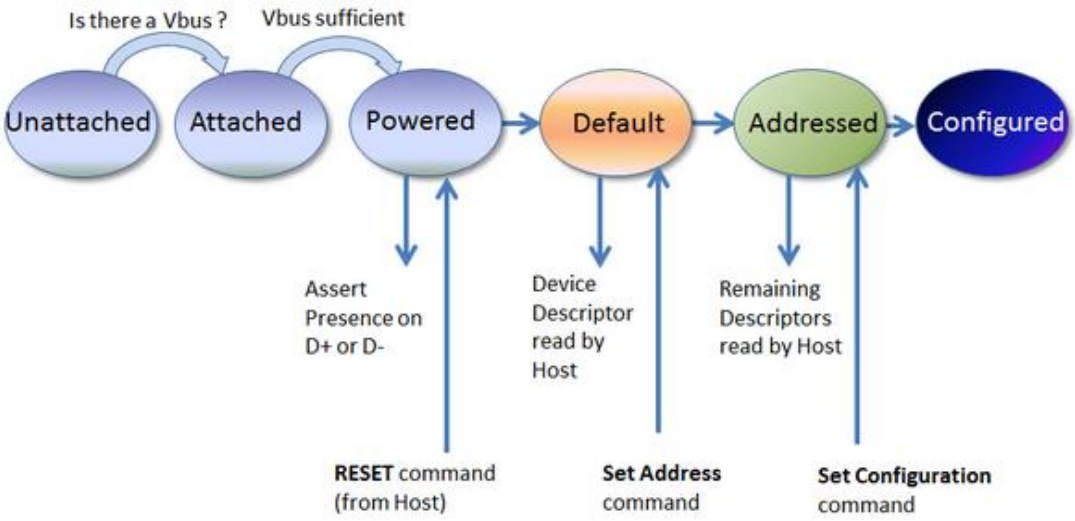



*Figura 27. Enumeración USB según Microchip Developer Help.*

En el presente proyecto, debido a que surge el error STALL, se necesita desconectar y volver a conectar (powercycle) el Hub Amfitrack del Arduino USB Host Shield, obligando así que se repita desde el principio la Enumeración. Esta solución puede ser mejorada si se altera el hardware del Shield y añadiendo una función de la librería Usb.cpp con el propósito de desconectar y conectar digitalmente el Hub Amfitrack cuando se necesite. La función es **USB::vbusPower()**.

Como se indicó previamente, la comunicación entre el Arduino y el Hub Amfitrack es constante, pasando así los paquetes de datos por los tres bloques según dicte el software desarrollado por las librerías USB Host Shield 2.0, SPI, Serial y ArduPREMO.

## 4.5. Descripción del desarrollo del *software*

En este punto se describe el código desarrollado (que desde ahora llamaremos librería ArduPREMO). Además, se indican las clases creadas, como las heredadas y el ejemplo en archivo Arduino (extensión .ino) para la utilización del Amfitech.

Todo el código está publicado en el Anexo I.

### 4.5.1. Librerías de software libre heredadas

#### 4.5.1.1. SPI

La librería SPI, del inglés Interfaz Periférica Serie (del inglés, Serial Peripheral Interface), permite la comunicación con los dispositivos SPI, teniendo a la placa Arduino como maestro [33]. Este es un protocolo usado por los microcontroladores para la comunicación con uno o más periféricos o microcontroladores. Esta librería se utiliza dentro de la librería USB Host Shield 2.0 para la comunicación entre el *host shield* de Arduino que comunica al controlador *host* MAX3421E que incorpora este con el Arduino MEGA2560 utilizado en el proyecto [34].

#### 4.5.1.2. USB_HOST_SHIELD_2.0

La librería ArduPREMO utiliza directa e indirectamente distintos tipos de clase creados para la correcta comunicación de los dispositivos por USB. Para saber qué archivos se necesitan para el presente proyecto se ha recurrido a árboles de herencia, creados por los propios desarrolladores de las librerías Usb Host Shield 2.0. Algunos de estos se pueden encontrar en el Anexo.

Las librerías directamente utilizadas de esta carpeta son:

- **hiduniversal.h**

  Esta librería utiliza la clase HIDUniversal. Una de las funciones que se usan es Poll() que llama a la función ParseHIDData de la librería ArduPREMO. Además, se hace uso de la función Init() cuya instrucción final es OnInitSuccesful() de la clase ArduPREMO. Con esta función se indica la inicialización del dispositivo HID, en este caso el Hub Amfitrack.

Las librerías más importantes utilizadas indirectamente de esta carpeta son:

- **UsbCore.h**
- **Usb.h**
- **usbhub.h**
- **hiddescriptorparser.h**
- **usbhid.h**

Cuando se incluyen librerías como hiduniversal.h, las distintas funciones implementadas en el archivo de extensión .cpp utiliza distintas clases y funciones creadas en las anteriores librerías, entre otras, para conseguir establecer una comunicación entre el *shield* y el dispositivo. Cabe destacar la librería hidescriptorparser.h, donde se encuentra la clase ReportDescParser2. Esta clase contiene la función ParseItem, que analiza los Items del receptor.

### 4.5.2. Librería ArduPREMO

Esta es la librería desarrollada en el presente TFG y donde se han utilizado las distintas librerías de USB Host Shield 2.0. Para ello se crearon dos archivos:

- **ArduPREMO.h**: es donde se ha creado la clase ArduPREMO y han sido definidas las funciones y variables miembro.
- **ArduPREMO.cpp**: es donde se han implementado las funciones miembros para el funcionamiento de la librería.

#### 4.5.2.1. Estructuras Position y Quaternion

Estas estructuras se definen en el archivo *header* y fuera de la clase ArduPREMO. Su finalidad es la de crear una serie de variables de posicionamiento (*position*) y orientación (*quaternion*) del tipo flotante (número con decimales) y calcularlos en milímetros, según marca el protocolo USB de Amfitech recogido en el Anexo del presente documento.

#### 4.5.2.2. Variables de la clase ArduPREMO

En esta sección se describen las variables creadas en la clase ArduPREMO. Las variables son las siguientes:

- **uint8_t TxUID**

  Variable que sirve para identificar la ID del receptor 3DCoild conectado al Hub Amfitrack. Si no hay ningún receptor conectado el programa imprime un aviso por pantalla. El byte TxUID está contenido en el paquete de datos que envía el Hub Amfitrack.

- **long int posX**
- **long int posY**
- **long int posZ**
- **long int quatX**
- **long int quatY**
- **long int quatZ**

- **long int quatW**

  Estas siete variables son el posicionamiento y la orientación del receptor obtenido tras el cálculo de la función *intFromBytes*. Tienen una extensión long int (entero largo) porque el receptor envía el posicionamiento 3D para los ejes tanto positivos como negativos, necesitando así un mayor número de *bytes*. **Se aconseja al usuario no utilizar estas variables directamente y sí implementar las funciones *getPosition* y *getQuaternion* descritas más abajo**.

- **uint8_t* pBUFF**

  La variable pBUFF es un puntero de datos. Este contiene el paquete de datos que el Hub envía al Arduino MEGA2560. Hay que señalar que este puntero se crea consiguiendo los datos del puntero Buf, el cual no siempre posee el paquete de datos. Esto depende del proceso de enumeración y no contiene el paquete hasta que esa enumeración ha sido recorrida con éxito.

- **bool TxUID_bool**

  Este *booleano* sirve para indicar si se ha conectado (True) o no (False) un receptor 3D *coil*.

- **Definición CYCLE_TIME**

  Esta definición sirve para el ejemplo creado *Amfitrack_Viewer*. Con él, el usuario puede ralentizar la obtención de las variables para poder leer mejor la consola.

#### 4.5.2.3. Funciones de la clase ArduPREMO

En esta sección se detallan las funciones creadas en la clase ArduPREMO. Las funciones son las siguientes:

- **Void ParseHIDData**

  Esta función se usa para analizar los datos del USB HID. Fue reimplementada de la clase HIDUniversal, es decir, HIDUniversal contiene un método virtual llamado ParseHIDData y se utiliza en la librería desarrollada. Un método virtual es una función asociada a un objeto que indica cómo ha de definirse una función, pero no indica cómo resuelve el problema que se plantee [35]. Por ello, cada librería tiene una función ParseHIDData distinta.

En el contexto del presente proyecto, el método ParseHIDData de la clase ArduPREMO se encarga de copiar el *buffer* y de confirmar que se trata del puntero de datos buscado.

- **Void Screening**

  *Screening* es un filtro para el paquete de datos. Con ello se puede confirmar que el puntero de datos obtenido es el especificado en el protocolo USB proporcionado por Amfitech, se puede encontrar en el Anexo.

- **Bool connected**

  Esta función es un *booleano* que resulta verdadero cuando el ID Vendor y el ID Product del Hub coinciden con los definidos dentro del archivo ArduPREMO.h. Si es falso, no se imprimirá ningún puntero de datos.

- **Long intFromBytes**

  Función para pasar los *bytes* del paquete de datos a decimal. Además, permite al usuario seleccionar si el paquete está en Little Endian o Big Endian, si necesita el posicionamiento con signo o sin signo.

- **Const uint8_t* Getbuf**

  Esta función sirve para hacer una copia del puntero de datos. Es del tipo constante para prevenir que sea alterado en alguna otra parte del programa.

- **Float getQuaternion** y **Float getPosition**

  Ambas funciones son las que necesita utilizar el usuario si quiere recibir los resultados del cálculo de posicionamiento y orientación del Sensor Receptor Amfitrack. Estas funcionan con un *case* en el que, si el usuario en su código escribe 0, 1 o 2, le entregará la orientación o posición respecto a los ejes X, Y o Z, respectivamente. El case 3 solo sirve para la orientación, ya que entrega el Momento.

- **Quaternion getQuaternion** y **Position getPosition**

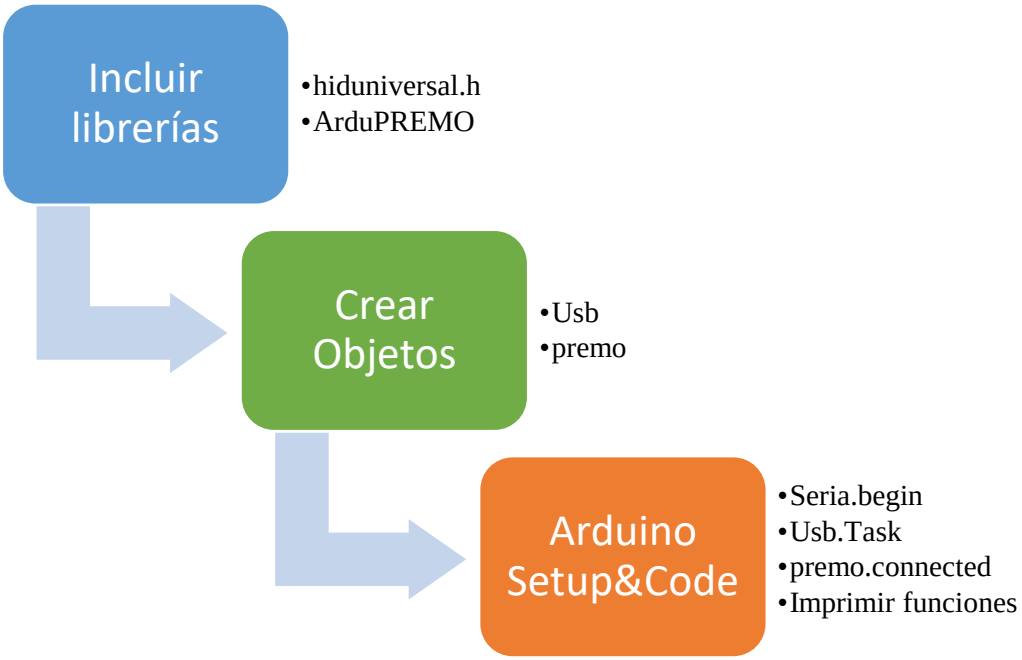
  Estas funciones sirven para realizar una copia de las variables calculadas. Así el usuario no debe preocuparse de si elimina por

error dichas variables durante la manipulación de la librería en el momento de la implementación en su respectivo proyecto.

### 4.5.3. Programa example Arduino

Este programa es un ejemplo donde el usuario final puede probar e implementar el código a su proyecto. Este programa es el principal y se encarga de unificar todas las librerías descritas anteriormente e implementar las funciones necesarias para construir la comunicación entre el Hub, el Shield y la propia placa Arudino.

El programa se estructura de la siguiente forma:




#### 4.5.3.1. Inclusión de librerías

Al principio del código se incluyen las librerías que se usarán directamente. Estas son la librería hiduniversal.h que contiene la clase HIDUniversal y la librería ArduPREMO, con su respectiva clase.

#### 4.5.3.2. Creación de objetos

En esta parte del código se crean los objetos Usb de la clase USB cuya librería ha sido incluida dentro de ArduPREMO.h y por ello se hereda en este código. Seguidamente, se crea el objeto premo de la clase ArduPREMO. Este objeto se creó pasando el objeto Usb como parámetro. Esto es así porque ArduPREMO es una clase que se basa en la clase HIDUniversal y esta fue desarrollada para ser iniciada con el parámetro del objeto Usb, para así poder utilizar los parámetros miembro de la clase USB de la librería Usb.cpp.

#### 4.5.3.3. Configuración y función de lazo cerrado de Arduino

Esta etapa se divide en dos funciones:

- Void setup()

  En esta función se realiza la configuración de la Comunicación *Serial* (del inglés, Serie) y Usb. La Comunicación *Serial* se inicializa para comenzar la comunicación por puerto serie entre la placa Arduino y el ordenador. Esta se configura a una velocidad de transmisión de 115200 bits por segundo, la cual es la configurada por los desarrolladores de Usb Host Shield 2.0. Posteriormente hay un condicional donde llama a la función Init() de la clase USB que se encarga de inicializar la comunicación USB entre el Hub Amfitrack y la Shield. Si esta no se puede inicializar, enviará un mensaje de error y entrará en un bucle infinito hasta que conecte.

- Void loop()

  Esta función es un lazo infinito, es decir, una vez finaliza la última instrucción que contiene, vuelve a correr la primera instrucción que contiene. La primera instrucción es una llamada a la función Task() de la clase USB. Esta función se encarga de realizar el proceso contenido en el protocolo USB llamado Enumeración. Una vez completada la etapa de enumeración se llama al método getUsbTaskState que consigue el estado que devuelve Task(). Si este coincide con el USB_STATE_RUNNING o estado de funcionamiento de USB, se procederá a la siguiente condicional. Si el método connected de la clase ArduPREMO es verdadero significa que el dispositivo conectado coincide con el ID Vendor y el ID Product especificados, por lo tanto se procede a leer el condicional premo.TxUID_bool. Si este último es falso, significa que ningún sensor 3D coild del Amfitrack está conectado al Hub Amfitrack y no procedería a imprimir el seguimiento 3D. No obstante, si esta condición resulta verdadera, se imprimirán las variables de posicionamiento y orientación en formato decimal con coma flotante y con signo.

## 4.6. Documentación de las pruebas finales

Para demostrar el correcto funcionamiento del software se realizó una prueba que consistía en colocar la fuente emisora del kit Amfitrack en un punto localizado sobre la superficie de un papel de tamaño A2, colocando además uno de los sensores del kit con filtro Karman. Utilizando una regla se medía la posición en ejes X, Y y Z del sensor respecto a la fuente. Los resultados llegaron a ser satisfactorios, cometiendo un pequeño margen de error debido al *hardware* del *kit* y a las herramientas de medida.

### a) Medición a 400 milímetros de la fuente en el eje X.

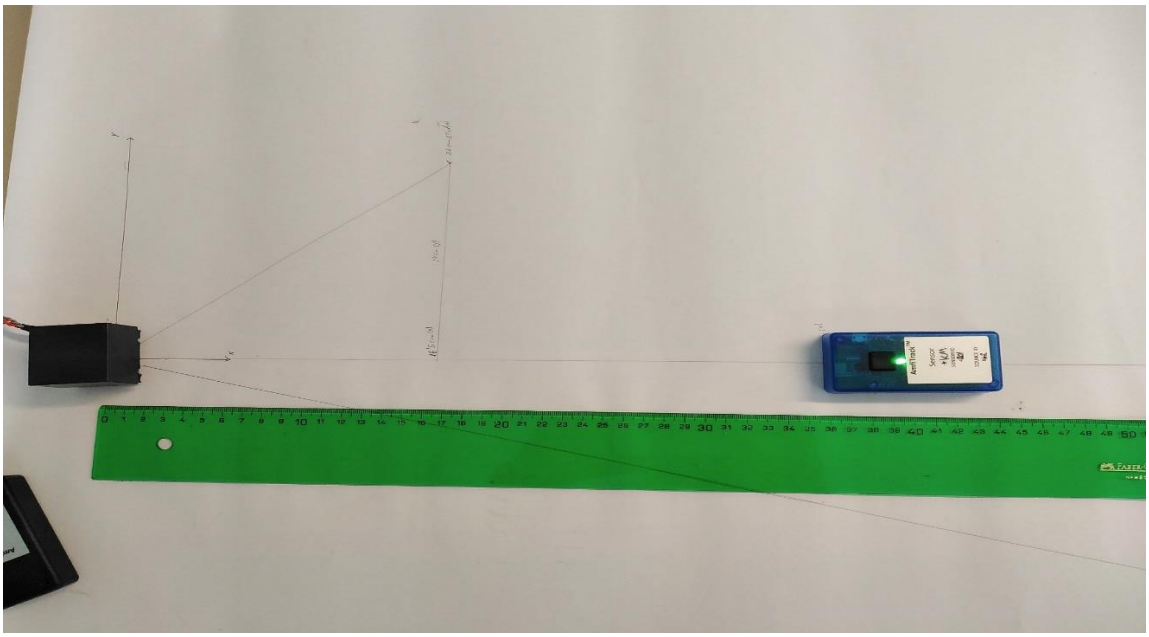


*Figura 28. Medición real en x=400mm, y = 0, z = 0.*

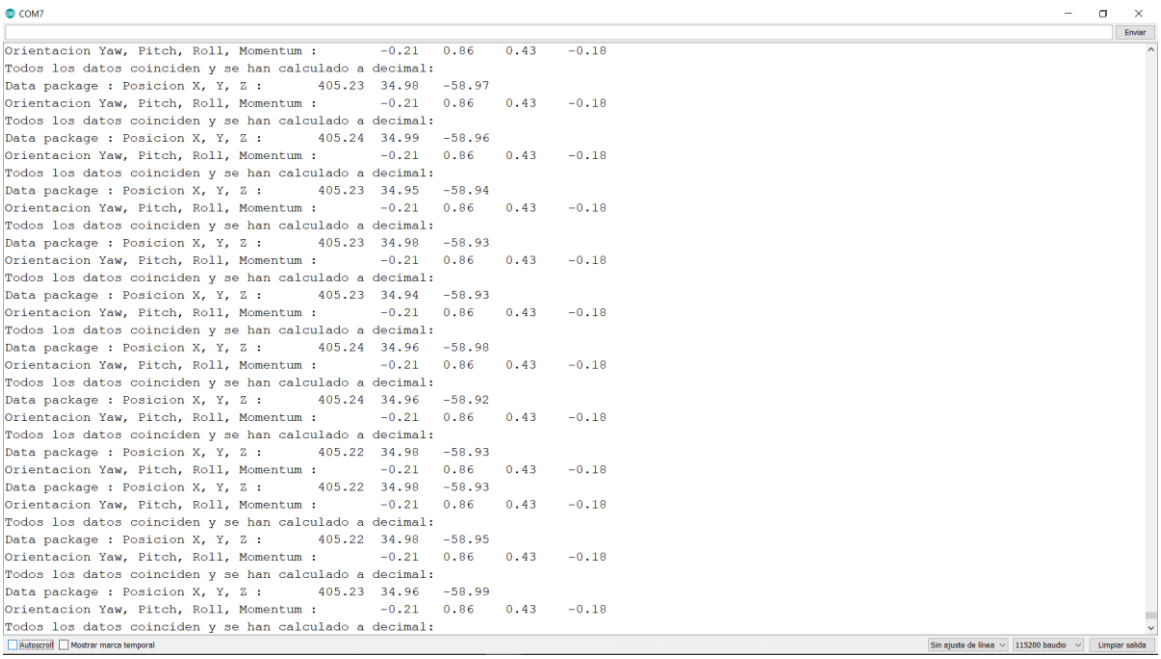



*Figura 29. Medición obtenida por ArduPREMO en x=400mm, y = 0 y z = 0.*

## b) Medición a 400 milímetros de la fuente en eje X y 140 milímetros en eje Y

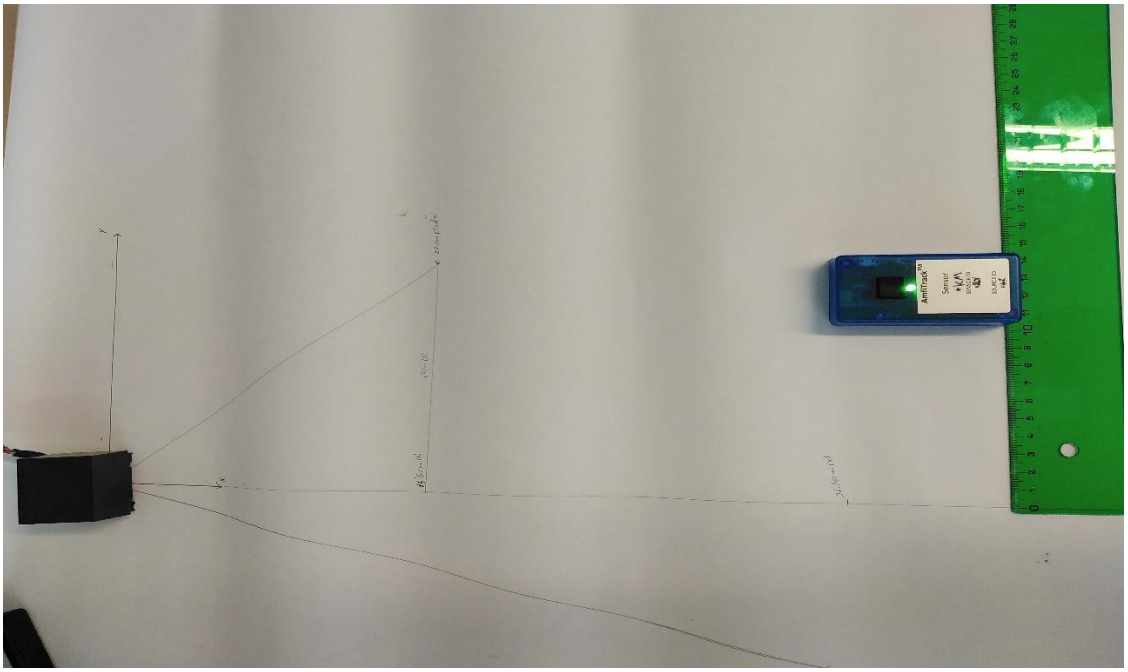


*Figura 30. Medición real en y=130mm, x = 400mm y z = 0.*

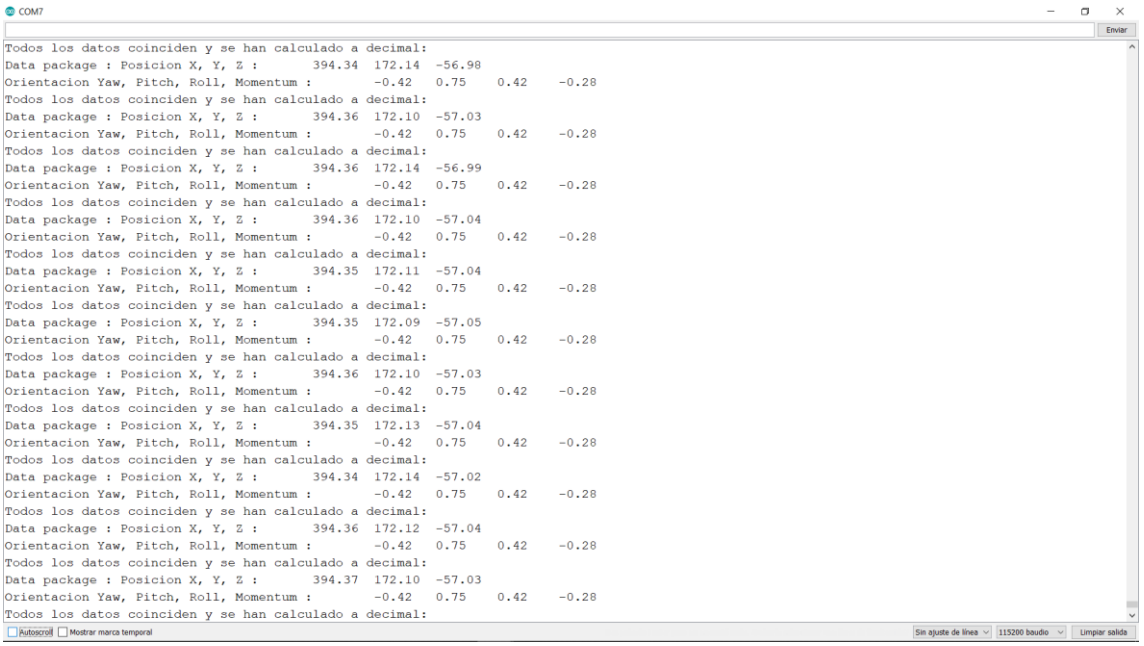



*Figura 31. Medición obtenida por ArduPREMO en y=140mm, x = 400mm y z = 0.*

### c) Medición a 300 milímetros de la fuente en el eje Z.

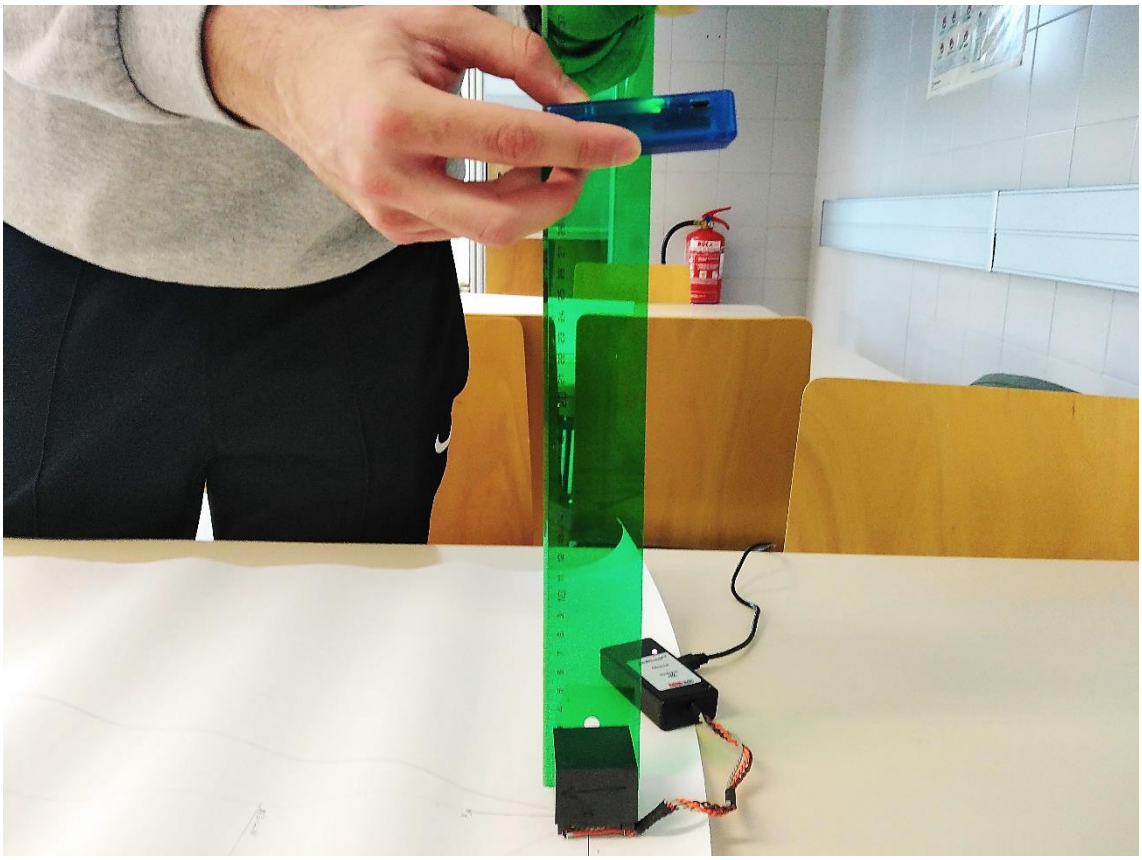


*Figura 32. Medición real en z=300mm, x = 0 e y = 0.*

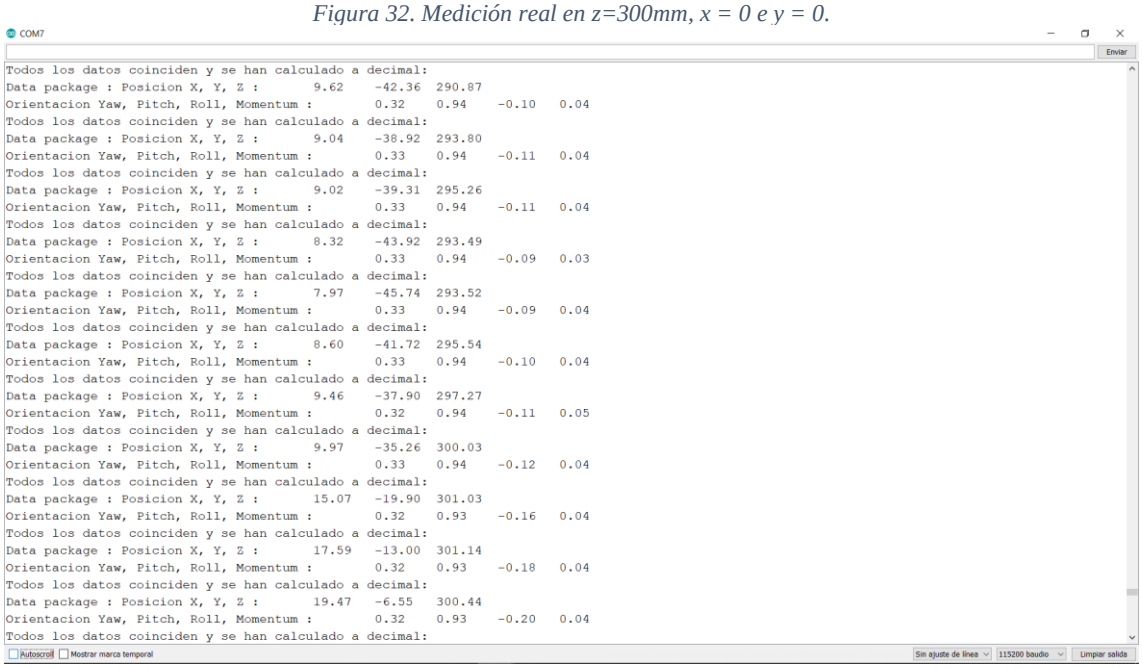



*Figura 33. Medición obtenida por ArduPREMO en z=300mm, x = 0 e y = 0.*

Por otro lado, no fue posible medir con exactitud las orientaciones, pero en una serie de pruebas previas se observó cómo variaban los cuaterniones x, y, z, w en el programa para Microsoft de Amfitech para el kit Amfitrack y lo que variaban esos cuaterniones

utilizando el programa de ejemplo indicado anteriormente. Los resultados eran muy similares, por lo que se determinó que era correcto el cálculo realizado.

# 6. CONCLUSIONES Y PROPUESTA DE FUTURO

Finalmente, en este punto se indican las conclusiones obtenidas tras el desarrollo del software para la adaptación del receptor 3D Amfitech en dispositivos Plug&Play y se añade una propuesta de líneas futuras para la mejora del producto.

## 6.1. Conclusiones sobre el trabajo desarrollado

Se ha alcanzado el objetivo principal del proyecto, el cual era establecer comunicación entre el Hub Amfitrack y el Arduino MEGA2560 para fomentar la utilización del *kit* de Amfitech en la Comunidad Arduino y para proyectos de *hardware* libre de toda índole. Para lograr el objetivo final han tenido que ser superados una serie de retos secundarios como son el estudio del protocolo para comunicaciones USB 2.0, el dominio de la programación en base a objetos y la respectiva comprensión y creación de librerías, además de aprender más sobre el entorno Arduino, tanto en el análisis del hardware, las distintas librerías software y la Comunidad Arduino, participando activamente en los foros.

Por otro lado, se abre la ventana a la posibilidad de participar en proyectos donde se requiera una comunicación USB, comunicación muy utilizada en el IoT (Internet of Things o Internet de las Cosas en español), al llegar a comprender los distintos tipos de clases de dispositivos USB que existen y la configuración de estos para su implementación *software*.


*Figura 34. Logotipo de la iniciativa de código abierto.*

Finalmente, como se ha podido observar, se consiguió el objetivo de crear una librería de *software* libre que pueda utilizar el usuario final para sus propios proyectos que precisen del kit de Amfitech y PREMO. Aun así, el camino trazado para poder llegar a tal propósito no fue exactamente como el descrito en el punto 4.2 de la presente memoria. Hasta llegar a él se han creado bastantes versiones del *software*, utilizando distintas partes de las

librerías de USB Host Shield 2.0. Esto se debió a que la única documentación que existe sobre estas son una serie de árboles de herencia y colaboración, creados con Doxygen. Por lo que, para poder llegar a la conclusión, se tuvo que estudiar los distintos árboles, acompañado siempre de la documentación aportada por el protocolo de usb.org y distintos blogs de desarrolladores que han trabajado en esta materia. Esto ralentizó el proceso de desarrollo del *software*, llegando al punto de que no se podía avanzar sin ponerse en contacto con el equipo de desarrollo del USB Host Shield 2.0. Finalmente, este contacto llegó a ocurrir (enlace de Issues de GitHub para USB Host Shield 2.0 Rev) y la librería ArduPREMO llegó a término

## 6.2. Líneas futuras

A pesar de que la librería ArduPREMO se constituye como un código que facilita el acercamiento de los usuarios de *software* libre a la tecnología de Amfitech y PREMO, este puede mejorarse antes de acercarles el producto. Para ello, se podrían realizar dos proyectos más:

**a) Implementación de comunicaciones Bluetooth.**

Volviendo al esquema establecido en la introducción de la Metodología y Resultados (punto 4.1):

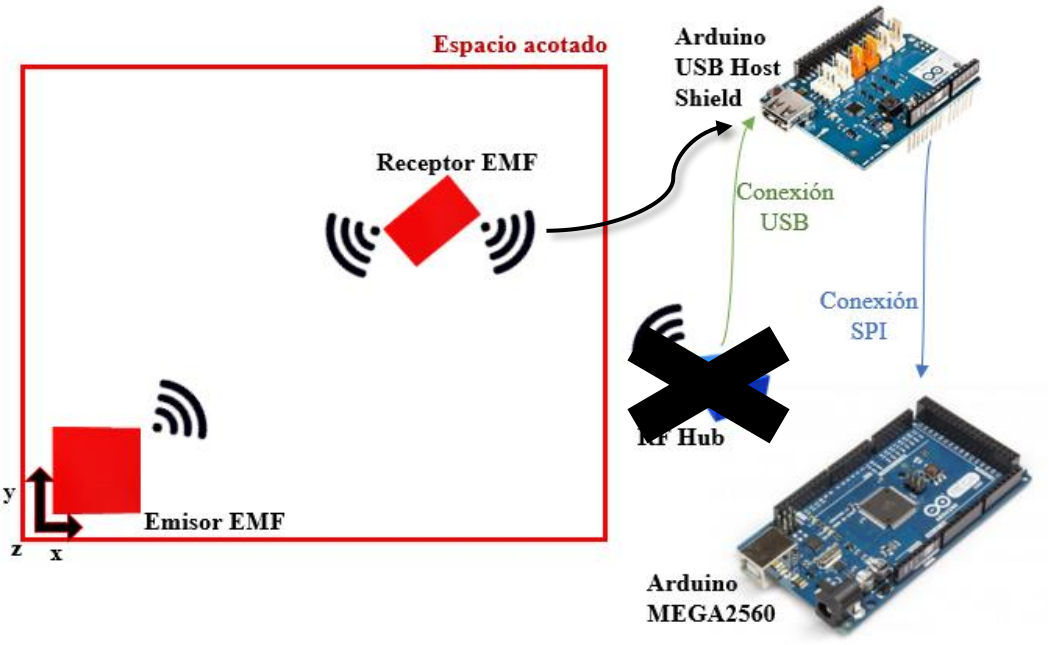



*Figura 35. Modelo del proyecto desarrollado eliminando el RF Hub del kit.*

Se puede observar que el Hub Amfitrack (RF Hub) es solo un instrumento que centraliza las conexiones inalámbricas de los receptores EMF del Amfitrack. Puesto que en el proyecto se está utilizando un microcontrolador Arduino, se podría prescindir del Hub y establecer mediante un Shield Bluetooth Arduino conectado directamente al Arduino MEGA2560 y creando una librería que recoja

los paquetes emitidos vía Bluetooth por el sensor. Igual que se hizo por USB desde el Hub Amfitrack, pero esta vez recogiendo los paquetes de datos que envía el Sensor Receptor Amfitrack (Receptor EMF) por Bluetooth. Más allá, si se desea tener ambas opciones, la de comunicación Bluetooth y la de comunicación USB, ya sea porque el sensor está separado del microcontrolador (por ejemplo, un brazo robótico con distintos receptores) o porque este deba estar donde esté el microcontrolador (por ejemplo, Receptor Amfitrack directamente conectado al microcontrolador de un dron vía USB), podría plantearse la creación de una nueva *shield* que recoja ambos tipos de comunicación (Bluetooth y USB) y que se pueda conectar a la placa Arduino.

Con esto se conseguiría que el kit Amfitrack sea versátil y fácilmente implementable a todo tipo de proyectos Arduino e incluso para otras marcas, como SparkFun Electronics.

**b) Cambiar la plataforma Arduino por la plataforma Raspberry PI.**

Esta segunda línea propone dejar la plataforma Arduino y explorar la opción del uso de un Raspberry PI, el cuál utiliza un derivado del OS de Linux por lo que el desarrollo de cualquier *software* basado en comunicaciones USB o de cualquier otro tipo es muy sencillo. Además, este miniPC se utiliza en proyectos de robótica de todo tipo por lo tanto se conseguiría el objetivo de dar a conocer el kit en la comunidad del *software* libre como una solución eficaz para el seguimiento y trazabilidad de dispositivos en entornos acotados.

*Figura 36. Raspberry PI. (Imagen obtenida de Wikipedia).*

Para esta plataforma pueden utilizarse las ideas de la línea futura anterior, pero en vez de crear un *shield* para Raspberry PI, se puede modificar el diseño del Sensor Receptor Amfitrack para poder conectar este tanto por USB, por patillas de comunicación Serie y por Bluetooth a una Raspberry PI, siendo esta la encargada

de controlar todos los sensores receptores e incluso las fuentes gracias a su mayor capacidad frente a los microcontroladores Arduino.

# 7. BIBLIOGRAFÍA Y REFERENCIAS


[1] Ángel Franco García, "Campo magnético producido por una corriente circular en un punto fuera de su eje." [Online]. Available: http://www.sc.ehu.es/sbweb/fisica3/magnetico/espira/espira_1.html. [Accessed: 16-May-2019].

[2] S. 90 and S. 90, *Energia.* Britannica Digital Learning, 2014.

[3] P. Fazekas and P. Fazekas, *Lecture notes on electron correlation and magnetism*. .

[4] Wikipedia, "Ley de Biot-Savart - Wikipedia, la enciclopedia libre." .

[5] Q. Yin, B. Chen, R. Xiong, Z. Cai, and B. Zhou, "Research on six-degree-of-freedom electromagnetic tracking system," *2010 9th Int. Symp. Antennas Propag. EM Theory, ISAPE 2010*, pp. 1184–1187, 2010.

[6] S. Song, H. Ren, and H. Yu, "An improved magnetic tracking method using rotating uniaxial coil with sparse points and closed form analytic solution," *IEEE Sens. J.*, vol. 14, no. 10, pp. 3585–3592, 2014.

[7] M. B. A. Mccullough, "Electromagnetic Motion Tracking," no. March, 2018.

[8] A. A. Dorrington and R. Künnemeyer, "A simple microcontroller based digital lock-in amplifier for the detection of low level optical signals," 2002.

[9] X. Yun, E. R. Bachmann, R. B. Mcghee, and M. J. Zyda, "Iros_2000," pp. 1–9, 2001.

[10] QuantDare, "El filtro de Kalman." [Online]. Available: https://quantdare.com/filtro-kalman/.

[11] Lukebeech, "Motion Capture | Creative Technology." [Online]. Available: https://lukebeech.wordpress.com/motion-capture/. [Accessed: 23-Oct-2019].

[12] D. J. Sturman, "5/1/12 3:26 PM A Brief History of Motion Capture for Computer Character Animation A Brief History of Motion Capture for Computer Character Animation."

[13] PS-TECH, "3D localization technology | Optical tracking explained | 6DOF | PS-Tech." [Online]. Available: http://www.ps-tech.com/optical-tracking-explained/. [Accessed: 23-Oct-2019].

[14] TERRITORY, "Motion Capture." [Online]. Available: https://www.moviemaker.com/wp-content/uploads/a03d03a0-ab01-4aba-9991-e50b528f7812.jpg. [Accessed: 23-Oct-2019].

[15] A. M. Franz, T. Haidegger, W. Birkfellner, K. Cleary, T. M. Peters, and L. Maier-Hein, "Electromagnetic Tracking in Medicine—A Review of Technology, Validation, and Applications," *IEEE Trans. Med. Imaging*, vol. 33, no. 8, pp. 1702–1725, Aug. 2014.

[16] W. Birkfellner, J. Hummel, and E. Wilson, "Chapter 2 Tracking Devices," *Bus.*

*Media LLC*, vol. 32, no. 2008, pp. 23–45, 2008.

[17] E. Navarro et Al., "Electromagnetic tracking systems explained: behind the scenes of VR/AR." [Online]. Available: https://3dcoil.grupopremo.com/blog/electromagnetic-tracking-systems-virtual-reality/. [Accessed: 22-Oct-2019].

[18] T. Kaiga, T. Yukawa, K. Mitobe, T. Miura, H. Tamamoto, and N. Yoshimura, "Magnetic Motion Capture System Measuring Movements of Hands and a Body Simultaneously."

[19] C. Hung Hsieh, J. Der Lee, and C. Tsai Wu, "A Kinect-based Medical Augmented Reality System for Craniofacial Applications Using Image-to-Patient Registration," *Neuropsychiatry (London).*, vol. 07, no. 06, pp. 927–939, 2017.

[20] "HID," *Wikipedia*. .

[21] Arduino, "What is Arduino?" [Online]. Available: https://www.arduino.cc/en/Guide/Introduction.

[22] Arduino, "Getting Started with Arduino MEGA2560." [Online]. Available: https://www.arduino.cc/en/Guide/ArduinoMega2560.

[23] Arduino, "Arduino USB Host SHield." [Online]. Available: https://www.arduino.cc/en/Main/ArduinoUSBHostShield&lang=.

[24] Maxim Integrated, "MAX3421E USB Peripheral/Host Controller with SPI Interface - Maxim." [Online]. Available: https://www.maximintegrated.com/en/products/interface/controllers-expanders/MAX3421E.html. [Accessed: 23-Oct-2019].

[25] Arduino, "Arduino IDE." [Online]. Available: https://es.wikipedia.org/wiki/Arduino_IDE.

[26] Wikipedia, "PyCharm."

[27] Wikipedia, "Code::Blocks - Wikipedia, la enciclopedia libre." [Online]. Available: https://es.wikipedia.org/wiki/Code::Blocks. [Accessed: 23-Oct-2019].

[28] CoffeeBrain-Wiki, "USART Básico - CoffeeBrain-Wiki." [Online]. Available: http://www.coffeebrain.org/wiki/index.php?title=USART_Básico. [Accessed: 23-Nov-2019].

[29] AprendiendoArduino, "UART y USB en Arduino | Aprendiendo Arduino." [Online]. Available: https://aprendiendoarduino.wordpress.com/2016/11/09/uart-y-usb-en-arduino/. [Accessed: 12-May-2019].

[30] H. Controller, *MAX3421E Programming Guide*, no. December. 2006.

[31] beyondlogic.org, "USB in a NutShell - Chapter 3 - USB Protocols." [Online]. Available: https://www.beyondlogic.org/usbnutshell/usb3.shtml#USBProtocols. [Accessed: 23-Nov-2019].

[32] Microchip, "USB Enumeration - Developer Help." [Online]. Available:

https://microchipdeveloper.com/usb:enumeration. [Accessed: 24-Nov-2019].

[33] Arduino.cc, "Arduino - SPI." [Online]. Available: https://www.arduino.cc/en/Reference/SPI. [Accessed: 20-Nov-2019].

[34] Oleg Mazurov, "USB Host Shield Hardware Manual « Circuits@Home." [Online]. Available: https://chome.nerpa.tech/usb-host-shield-hardware-manual/. [Accessed: 20-Nov-2019].

[35] Juan David Meza González, "Aprende funciones de C++ desde cero, métodos y procedimiento." [Online]. Available: https://www.programarya.com/Cursos/C++/Funciones. [Accessed: 21-Nov-2019].

[36] Felis Oleg, "USB Host Shield 2.0: Class List." [Online]. Available: https://felis.github.io/USB_Host_Shield_2.0/annotated.html. [Accessed: 28-Nov-2019].

# 8. ANEXOS. DOCUMENTACIÓN DEL SOFTWARE

## 8.1. Anexo I: Librería ArduPREMO y Amfitech_Viewer.ino example.

a. ArduPREMO.h

```
#ifndef ARDUPREMO_H
#define ARDUPREMO_H


#include <HIDUniversal.h>
#include <SPI.h>
#include <usbhub.h>


struct Position {
  float X;
  float Y;
  float Z;
  Position()=delete;
  Position(long pX,long pY,long pZ):
        X{static_cast<float>(pX)*0.01},
        Y{static_cast<float>(pY)*0.01},
        Z{static_cast<float>(pZ)*0.01} {}
};



struct Quaternion {
  float X;
  float Y;
  float Z;
  float W;
  Quaternion()=delete;
```

```
  Quaternion(long pX,long pY,long pZ,long pW):
        X{static_cast<float>(pX)/1000000.0},
        Y{static_cast<float>(pY)/1000000.0},
        Z{static_cast<float>(pZ)/1000000.0},
        W{static_cast<float>(pW)/1000000.0} {}
};

class ArduPREMO : public HIDUniversal
{
public:

    USB Usb;

    // CONSTANTS
    #define AmfiHUB_VID       0x0C17
    #define AmfiHUB_PID       0x0D12
    #define MAX_DATA 64 // [bytes]
    #define TRANSMISSION_SPEED 115200 // [bps]
    #define CYCLE_TIME 200 // [ms]

    bool TxUID_bool = false;



private:
    uint8_t TxId = 0; // Message number
    uint8_t TxUID = 0; // Sensor ID
    long int posX = 0;
    long int posY = 0;
    long int posZ = 0;
```

```
    long int quatX = 0;

    long int quatY = 0;

    long int quatZ = 0;

    long int quatW = 0;

    uint8_t* pBUFF; //puntero copia


    void ParseHIDData(USBHID *hid, bool is_rpt_id,
uint8_t len, uint8_t *buf); // Called by the
HIDUniversal library

    void Screening();

    uint8_t OnInitSuccessful(); // Called by the
HIDUniversal library on success (BOOLEAN)


public:

    ArduPREMO(USB *p) : HIDUniversal(p) {};

    bool connected() {

       return (HIDUniversal::isReady() &&
HIDUniversal::VID == AmfiHUB_VID && HIDUniversal::PID
== AmfiHUB_PID);

    };            /*linea93 hiduniversal.h*/


    long intFromBytes(uint8_t b0, uint8_t b1, uint8_t
b2, bool little, bool _signed);



    uint8_t* Getbuf( uint8_t *buffealo){

     return buffealo;

    }


    float getQuaternion(uint8_t n_comp) const {

      Quaternion copia(quatX,quatY,quatZ,quatW);
```

```
      switch(n_comp) {
        case 0 : return copia.X;
        case 1 : return copia.Y;
        case 2 : return copia.Z;
        default : return copia.W;
      }
    }
    Quaternion getQuaternion() const {
      Quaternion copia(quatX,quatY,quatZ,quatW);
      return copia;
    }

    float getPosition(uint8_t n_comp) const {
      Position copia(posX,posY,posZ);
      switch(n_comp) {
        case 0 : return copia.X;
        case 1 : return copia.Y;
        default : return copia.Z;
      }
    }
    Position getPosition() const {
      Position copia(posX,posY,posZ);
      return copia;
    }
};

#endif // ARDUPREMO_H
```

b. ArduPREMO.cpp

```
#include "ArduPREMO.h"



void  ArduPREMO::ParseHIDData(USBHID  *hid,  bool is_rpt_id, uint8_t len, uint8_t *buf) {


   if (len && buf)  {
      pBUFF = Getbuf(buf);
      Screening();


      #if 0
      for (uint8_t i = 0; i < len; i++) {
       Serial.print(pBUFF[i],HEX);
       Serial.print(" ; ");
      }
      Serial.println("");
      #endif
   }



}



uint8_t ArduPREMO::OnInitSuccessful() { // Called by the HIDUniversal library on success (BOOLEAN)
   if  (HIDUniversal::VID  !=  AmfiHUB_VID  || HIDUniversal::PID != AmfiHUB_PID){ // Make sure the right device is actually connected
      return 0;
```

```
    }


    return 1;
}



void ArduPREMO::Screening() {
 uint8_t pkg[64];


 for(uint8_t cont{0}; cont<64; ++cont){
  pkg[cont] = pBUFF[cont];                    //Copia del
buffer para usar en Screening
 }


 if ((pkg[7]!= 0x00 and pkg[0] == 0x02) and (pkg[1] ==
0x0B) and (pkg[2] == 0x83) and
   (pkg[6] == 0x17) and (pkg[33] == 0x00)) {


  TxUID_bool = true;


  TxId = pkg[4];
  TxUID = pkg[7];


  posX = intFromBytes(pkg[10], pkg[11], pkg[12], true,
true);
  posY = intFromBytes(pkg[13], pkg[14], pkg[15], true,
true);
  posZ = intFromBytes(pkg[16], pkg[17], pkg[18], true,
true);
  quatX  =  intFromBytes(pkg[19],  pkg[20],  pkg[21],
true, true);
```

```
 quatY  =  intFromBytes(pkg[22],  pkg[23],  pkg[24],
true, true);

 quatZ  =  intFromBytes(pkg[25],  pkg[26],  pkg[27],
true, true);

 quatW  =  intFromBytes(pkg[28],  pkg[29],  pkg[30],
true, true);


 Serial.println("Todos los datos coinciden y se han
calculado a decimal: ");

} else if(pkg[7] == 0x00) {TxUID_bool = false;}


}//void Screening()




/*

* intFromBytes - makes an long integer from its bytes

*/

long ArduPREMO::intFromBytes(uint8_t b0, uint8_t b1,
uint8_t b2, bool little, bool _signed) {

 long value;


 if (little) { // little-endian

  value = b2;

  value = value << 8;

  value = value + b1;

  value = value << 8;

  value = value + b0;

 }

 else { // big-endian

  value = b0;
```

```
  value = value << 8;
  value = value + b1;
  value = value << 8;
  value = value + b2;
 }

 if (_signed and (value & 0x800000)) { // the 24 bits number is negative
  value = value | 0xFF000000; // Sign extension by padding the left side with ones
 }

 return value;
}
```

c. Amfitech_Viewer.ino

```
#include <hiduniversal.h>

#include <ArduPREMO.h>


/*OBJETOS Y VARIABLES GLOBALES*/

USB Usb;

ArduPREMO premo(&Usb);



/* ARDUINO SETUP&CODE */


void setup()

{

 Serial.begin( TRANSMISSION_SPEED );

#if !defined(__MIPSEL__)

 while (!Serial); // Wait for serial port to connect
- used on Leonardo, Teensy and other boards with
built-in USB CDC serial connection

#endif


 if (Usb.Init() == -1) {

  Serial.print(F("\r\nOSC did not start"));

  while (1); // Halt

 } else Serial.println(F("\r\n Kit Amfitrack for
Arduino program START"));


}



void loop() {
```

```
  Usb.Task();     //Enumeración --- Llega a --> Estado Funcionamiento de USB


  if(Usb.getUsbTaskState() == USB_STATE_RUNNING){


   if(premo.connected()){


    if(premo.TxUID_bool == true){ //Any 3Dcoil sensor ON? YES... PRINT

     //Imprime ID y posición

     Serial.print("Data package : ");

     Serial.print("Posicion X, Y, Z :");

     Serial.print("\t");

     Serial.print(premo.getPosition(0));

     Serial.print("\t");

     Serial.print(premo.getPosition(1));

     Serial.print("\t");

     Serial.println(premo.getPosition(2));

     Serial.print("Orientacion  Yaw,  Pitch,  Roll, Momentum :");

     Serial.print("\t");

     Serial.print(premo.getQuaternion(0));

     Serial.print("\t");

     Serial.print(premo.getQuaternion(1));

     Serial.print("\t");

     Serial.print(premo.getQuaternion(2));

     Serial.print("\t");

     Serial.println(premo.getQuaternion(3));
```

```
    #if 0
     //delays next cycle
     delay(CYCLE_TIME);
    #endif

   } else Serial.println("No 3DCoil connected.");

  }//if(premo.connected)
 } //if(Usb.getUsbTaskState) ... RUNNING

}
```

## 8.2. Anexo II: Árboles de herencia de USB Host Shield principales.

Los principales árboles de herencia y colaboración de las librerías creadas por el equipo de desarrollo de Usb Host Shield 2.0 se encuentran en GitHub [36] y son:

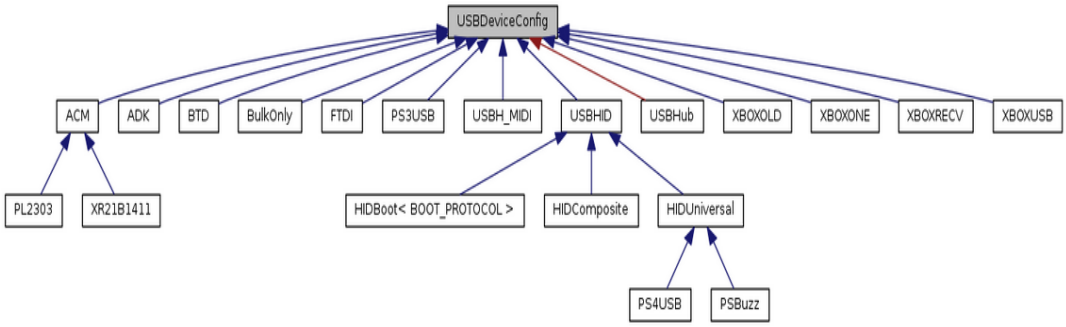



*Figura 37. Árbol de herencia de USBDeviceConfig.*

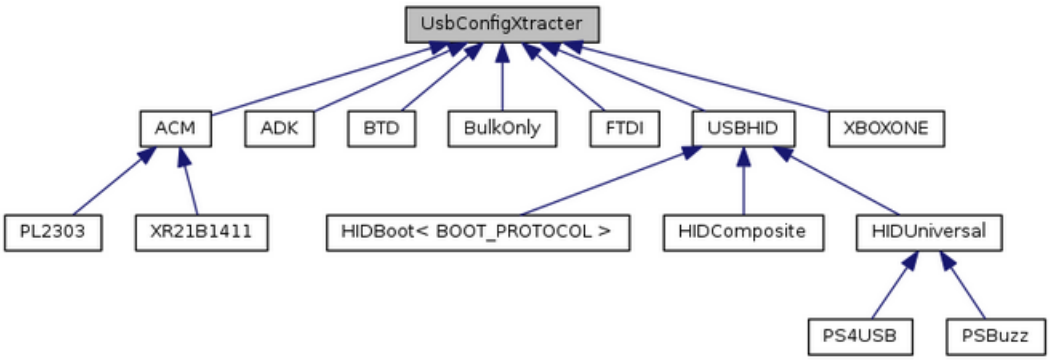



*Figura 38. Árbol de herencia de UsbConfigXtracter.*

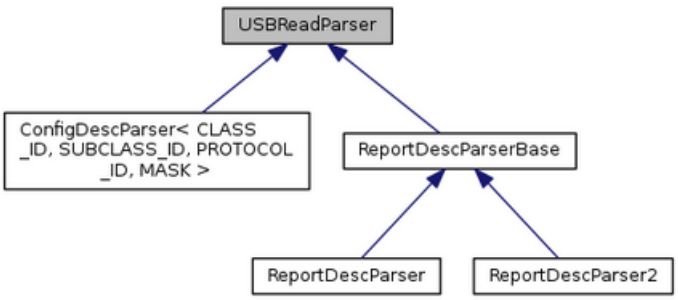



*Figura 39. Árbol de herencia de USBReadParser.*

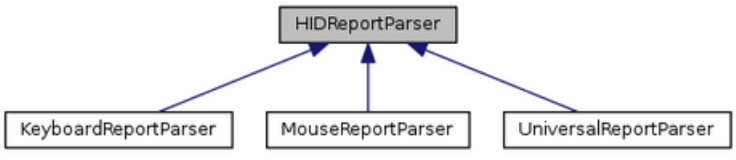



*Figura 40. Árbol de herencia de HIDReportParser.*

## 8.3. Anexo III. Protocolo USB del Hub Amfitrack.

USBReportId [uint8] // Always 0x02

USBSocket [uint8] // Always 0x0B

USBProperty [uint8] // Always 0x83

USBDataLength [uint8]

TxId [uint8] // Increments for each message

MessageType [uint8] // NoACK = 0x09, MustACK = 0x11, ACK = 0x32

PayloadLength [uint8]

TxUID [uint8] // Sensor/Source ID

HeadCRC [uint8]

Payload[???]

PayloadCRC [uint8]

Position Payload (Quarternion)

==========

PayloadType [uint8] // 0x14

XPos [int24LE] // in 0.01 mm resolution

YPos [int24LE] // -||-

ZPos [int24LE] // -||-

QuatX[int24LE] // Divide by 1000000 to get element

QuatY[int24LE] // -||-

QuatZ[int24LE] // -||-

QuatW[int24LE] // -||-

Status [uint8] // Battery, EMF and hardware status bits

CurrentAndFrequency Payload

==========

PayloadType [uint8] // 0x10

CurrentCoilX [float]

CurrentCoilY [float]

CurrentCoilZ [float]

FrequencyX [float]

FrequencyY [float]

FrequencyZ [float]

Sensor Status Payload

==========

PayloadType [uint8] // 0x09

Status [uint8] // Battery, EMF and hardware status bits

Example:

---

Raw USB packet:

02 0B 83 20 17 09 17 00 96 14 10 27 00 10 27 00

10 27 00 00 00 00 00 00 00 00 00 00 40 42 0F 02

CD 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Breakdown:

02 0B 83 USBReportId, USBSocket, USBProperty

20 USBDataLength // 0x20 = 32 bytes useful USB data

17 TxId // Increments for each message

09 MessageType // 0x09 = NoACK

17 PayloadLength // 0x17 = 23 bytes payload

00 TxUID // Sensor/Source ID

96 HeadCRC

14 PayloadType // Position Payload (Quarternion)

10 27 00 XPos [int24LE] // 0x002710 = 10000 and 10000 *0.01mm = 100mm

10 27 00 YPos [int24LE] // 0x002710 = 10000 and 10000 *0.01mm = 100mm

10 27 00 ZPos [int24LE] // 0x002710 = 10000 and 10000 *0.01mm = 100mm

00 00 00 QuatX[int24LE] // 0x000000 = 0 and 0 / 1000000 = 0

00 00 00 QuatY[int24LE] // 0x000000 = 0 and 0 / 1000000 = 0

00 00 00 QuatZ[int24LE] // 0x000000 = 0 and 0 / 1000000 = 0

40 42 0F QuatW[int24LE] // 0x0F4240 = 1000000 and 1000000 / 1000000 = 1

02 Status // TBD

CD PayloadCRC

//
The rest of the USB packet is unused.

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00