UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE UN GPS INERCIAL CON RASPBERRY PI

Juan Francisco Rudilla Pérez

Tutor: Vicent Miquel Rodrigo Peñarrocha

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2017-18

Valencia, 9 de septiembre de 2018

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

Resumen

Este trabajo consiste en el estudio de todos los componentes y necesidades, para la implementación de un GPS Inercial de bajo coste. Estará compuesto de un módulo GPS, una unidad de medición inercial (IMU) y una Raspberry Pi. Para ello, se utilizará para la adquisición de datos de los sensores el lenguaje de programación Python.

Más adelante se detallará el significado en profundidad de lo que es un sistema de medición inercial, pero en un primer acercamiento, podemos decir que está compuesto por un conjunto de sensores empleados para obtener la aceleración en cada uno de los 3 ejes de movimiento. Dichos sensores están incorporados dentro de la IMU, la cual contiene un acelerómetro de 3 ejes, un giroscopio y una brújula también siendo estos de 3 ejes. Con los cuales nos será posible la obtención de aceleración, orientación, giros e incluso con valores de referencia iniciales la velocidad, altitud e incluso ubicación. Dichos valores de referencia serían la ubicación, altitud y velocidad proporcionadas por un GPS.

Gracias a la combinación de estos dos sensores sería posible obtener nuestra ubicación de una forma más exacta minimizando así perdidas de nuestra ubicación por pérdidas momentáneas de señal con los satélites o de pequeños errores puntuales de la ubicación proporcionada por el GPS.

Resum

Aquest treball consisteix en l'estudi de tots els components i necessitats, per a la implementació d'un GPS Inercial de baix cost. Estarà compost d'un mòdul GPS, una unitat de mesura inercial (IMU) i una Raspberry Pi. Per a això, s'utilitzarà per a l'adquisició de dades dels sensors el llenguatge de programació Python.

Més endavant es detallarà el significat en profunditat del que és un sistema de mesurament inercial, però en un primer acostament, podem dir que està compost per un conjunt de sensors emprats per obtenir l'acceleració en cadascun dels 3 eixos de moviment. Aquests sensors estan incorporades dins de la IMU, la qual conté un acceleròmetre de 3 eixos, un giroscopi i una brúixola també sent aquests de 3 eixos. Amb els quals ens serà possible l'obtenció d'acceleració, orientació, girs i fins i tot amb valors de referència inicials la velocitat, altitud i fins i tot ubicació. Aquests valors de referència serien la ubicació, altitud i velocitat proporcionades per un GPS.

Gràcies a la combinació d'aquests dos sensors seria possible obtenir la nostra ubicació d'una manera més exacta minimitzant així perdudes de la nostra ubicació per pèrdues momentànies de senyal amb els satèl·lits o de petits errors puntuals de la ubicació proporcionada pel GPS.

Abstract

This work consists in the study of all the components and needs, for the implementation of a low-cost Inertial GPS. It will be composed of a GPS module, an inertial measurement unit (IMU) and a Raspberry Pi. To do this, the Python programming language will be used to acquire sensor data.

Later we will detail the meaning in depth of what is an inertial measurement system, but in a first approach, we can say that it is composed of a set of sensors used to obtain the acceleration in each of the 3 axes of movement. These sensors are incorporated within the IMU, which contains a 3-axis accelerometer, a gyroscope and a compass, these being also 3-axis. With which we will be able to obtain acceleration, orientation, turns and even with initial reference values the speed,

altitude and even location. These reference values would be the location, altitude and speed provided by a GPS.

Thanks to the combination of these two sensors it would be possible to obtain our location in a more accurate way, thus minimizing losses of our location due to momentary signal losses with the satellites or small point errors in the location provided by the GPS.

Índice

1 INTRODUCCIÓN

1.1 Estado del arte en la geolocalización

Desde el lanzamiento del GPS hace ya más de 30 años y en concreto desde que se abrió al uso civil, el GPS (Global Positioning System) se ha convertido en una herramienta esencial en muchos ámbitos. Su uso se puede dar desde herramientas militares de última generación a juegos para dispositivos móviles.
Pero esta herramienta no es infalible, todos hemos podido comprobar los problemas que puede tener el uso de GPS, desde saltos repentinos de una ubicación a otra cercana, pérdida de conexión con los satélites en túneles o edificios, problemas en días nublados, etc…

Si para los ciudadanos de a pie esto nos resulta molesto, hay que imaginarse en el ámbito militar donde se desarrolló esta herramienta, donde la pérdida de conexión con los satélites o un error en la ubicación puede significar la pérdida de millones de euros o peor aún, vidas de inocentes. El problema, además de la falta de conexión con los satélites, es que las señales GPS pueden ser interferidas bloqueándolas completamente sin posibilidad de recepción de estas, o bien reemplazadas por otras falsas, sin olvidar ataques cibernéticos a los receptores en tierra.

1.2 ¿Qué es un GPS inercial?

Un GPS inercial combina dos sistemas, un GPS clásico, con el que podemos obtener nuestra posición actual, altura y velocidad, además de un sistema de navegación inercial.
Dicho sistema de navegación inercial combina sensores de movimiento (acelerómetros) y sensores de rotación (giroscopios) con los cuales es posible obtener una estima de la posición, su orientación y velocidad.

El sistema de navegación inercial, no necesita durante su funcionamiento referencia externa para su funcionamiento, exceptuando al inicio, cuando es necesario tener su posición inicial, además de que al contrario del GPS es un sistema inmune a interferencias externas. Pero dicho sistema tiene un problema, los acelerómetros son sensores muy ruidosos, además de que al realizar cálculos se van acumulando pequeños errores que finalmente pueden dar un error de ubicación significativo, por lo que una opción es ir actualizando la posición cada cierto tiempo para verificar posibles errores con otro sistema, en este caso con el GPS.

Con la combinación de estos dos sistemas, podremos obtener una ubicación más precisa y además mucho más rápida, esto último se explicará más adelante. Pero para ello es necesario algún tipo de elemento de unión de las mediciones obtenidas por los módulos, dicho elemento será el filtro de Kalman que será explicado posteriormente.

1.2.1 Navegación inercial, ¿Qué es y para qué sirve?

La navegación inercial nació a principios del siglo XX basada en la navegación por estima clásica, la cual se basa en el uso de reloj y brújula. Fue propiciada por la invención de los giróscopos en el siglo XIX por León Foucault, en su experimento para demostrar la rotación de la tierra.
Durante los años 20, se realizaron pruebas para el desarrollo de los primeros pilotos automáticos, y durante la segunda guerra mundial se comenzaron a utilizar combinaciones de acelerómetros y giroscopios para el guiado de los misiles nazis V2.
Pero no fue hasta la travesía del USS Nautilus, el cual cruzó el Polo Norte bajo el casquete polar donde se pudo apreciar realmente el potencial de esta tecnología. Normalmente una embarcación o un submarino, se basaba en la orientación a través de una brújula magnética, la cual funciona comparando la ubicación con el norte magnético.
Posteriormente a esta importante hazaña los sistemas de navegación inerciales redujeron su peso y dimensiones, por lo que se comenzaron a implantar en aviones tanto militares como comerciales.

Dichos sistemas son denominados auto-contenido o autónomos, ya que como se ha mencionado anteriormente no necesitan de elementos ni información exterior para su funcionamiento y mediante sus acelerómetros y giróscopos podemos obtener los parámetros de posición, rumbo y velocidad.
Es muy importante saber, que para el correcto funcionamiento de este sistema debemos conocer inicialmente es su posición inicial, ya que al ser un sistema donde utilizaremos la integración matemática sólo podemos calcular los datos actuales disponiendo de los iniciales.
Para este proyecto se ha utilizado una unidad de medición inercial (IMU) que dispone de acelerómetro, giroscopio y magnetómetro siendo todos de 3 ejes. Más adelante se detallarán las características técnicas de dicho dispositivo, pero antes realizará una breve explicación de los elementos que la componen para entender su funcionamiento.

1.2.1.1 ¿Qué es un acelerómetro?

Para entender el funcionamiento de un acelerómetro podemos imaginarnos un cubo donde en el interior colocaremos una esfera.

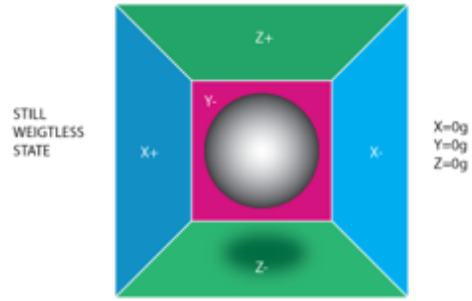

Figura 1.1. Modelo acelerómetro sin fuerzas externas.

Si dentro de esta caja no estuviera afectando ningún campo gravitatorio, la esfera se ubicaría siempre flotando en el centro del cubo.
En la figura 1.1 podemos ver que hemos asociado cada pared a un eje y en la figura siguiente vemos que realizando un movimiento a la izquierda, en dirección al eje x positivo con una aceleración de 1g la esfera golpeara a la pared opuesta con dirección eje -x con una fuerza de -1g, por lo que realmente la fuerza que detecta, es la opuesta a la ejercida también llamada fuerza inercial o ficticia, por lo que deberemos tener en cuenta es que un acelerómetro mide aceleración indirectamente a través de una fuerza que se aplica a uno de sus lados.

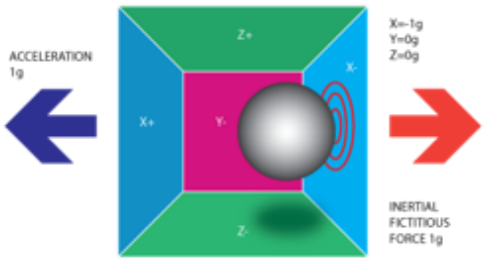

Figura 1.2. Modelo acelerómetro con fuerza ejercida a la izquierda.

A continuación, buscando una simulación real en nuestro planeta, nos encontraríamos que la esfera golpearía la pared -Z con una fuerza de -1g, esto estaría causado por la gravedad terrestre que ejerce una fuerza de 1g hacia todos los objetos situados en ella

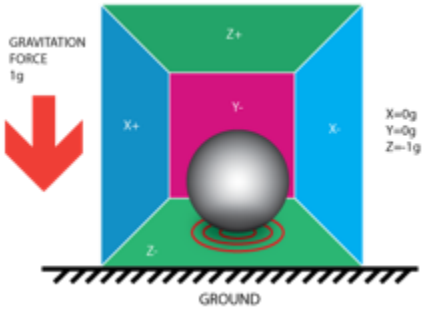

Figura 1.3. Modelo acelerómetro con gravedad terrestre.

Hasta este momento, solo hemos tenido en cuenta la interacción de la esfera con una de las paredes de la caja, o dicho de otra forma con uno de los ejes, pero al ser un acelerómetro triaxial podemos detectar fuerzas varios ejes simultáneamente, por lo que si la imagen anterior de la caja fuera desplazada 45º hacia la derecha observaríamos cómo la esfera impactaría contra dos paredes, las referidas al eje -Z y -X

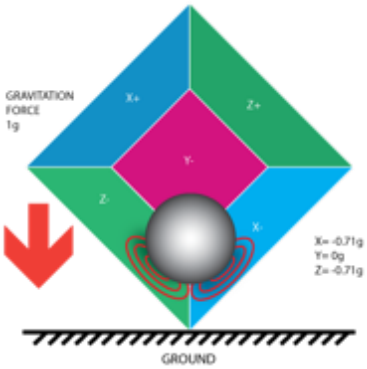

Figura 1.4. Modelo acelerómetro rotado 45º.

En los ejemplos del principio para ilustrar el principio de funcionamiento de un acelerómetro, la fuerza se ha mantenido constante, pero para realizar cálculos será más útil fijar el sistema de coordenadas a los ejes del acelerómetro y tomar como referencia que estos rotan alrededor de nuestro sistema de coordenadas.

A continuación, en la imagen podemos ver como los ejes representados son perpendiculares a las caras de la caja, que hemos utilizado previamente para entender el modelo. También se ha incorporado un vector R que actuara como vector de la fuerza medido por el acelerómetro.

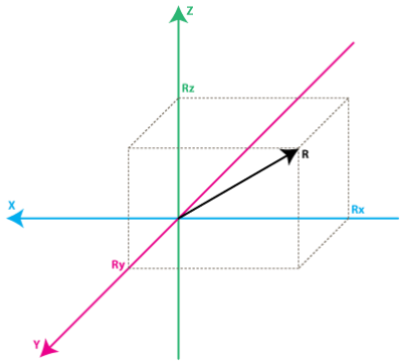

Figura 1.5. Modelo acelerómetro fijado a sistema de coordenadas

R_x, R_y, y R_z serán proyecciones del vector R en los ejes X, Y, Z y podemos expresarlo con la siguiente relación:

$$R^2 = {R_x}^2 + {R_y}^2 + {R_z}^2 \quad (1.1)$$

Dicha fórmula es la ecuación del teorema de Pitágoras en tres dimensiones.

En la actualidad podemos encontrar 2 tipos de acelerómetros, están los digitales que nos presentan la información por medio de protocolos en serie como I2C, SPI o USART o analógicos que nos proporcionarán un nivel de voltaje para posteriormente convertirlos por medio de un convertidor analógico digital (ADC) a un valor digital.

Pongamos un ejemplo, nuestro acelerómetro que incorpora un ADC de 10 bits nos proporciona las siguientes salidas:

$$Rx = 595 \quad (1.2)$$
$$Ry = 685 \quad (1.3)$$
$$Rz = 585 \quad (1.4)$$

Dicho acelerómetro tiene un voltaje de 3.3 V y sabemos que un convertidor analógico digital de 10 bits nos dará un valor entre 0 y 1023 por lo que:

$$VRy = \frac{595*3.3}{1023} = 1.91\ V \quad (1.5)$$

$$VRy = \frac{685*3.3}{1023} = 2.21\ V \quad (1.6)$$
$$VRz = \frac{585*3.3}{1023} = 1.89\ V \quad (1.7)$$

También debemos tener en cuenta que por construcción los acelerómetros tienen un $Voffset$ a 0g, por lo que debemos restarlo a los cálculos previamente efectuados, en este caso tomamos como $Voffset = 1.6$ V.

$$VrealRx = 1.91 - 1.6 = 0.31\ V \quad (1.8)$$
$$VrealRy = 2.21 - 1.6 = 0.61\ V \quad (1.9)$$
$$VrealRz = 1.89 - 1.6 = 0.29\ V \quad (1.10)$$

Ya para terminar necesitaremos convertir nuestra tensión a g, para ello debemos fijarnos en la sensibilidad de nuestro acelerómetro, que encontraremos normalmente en el datasheet de este, en este caso 478.5mV/g

$$Rx = \frac{0.31}{0.4785} = 0.65g \quad (1.11)$$
$$Ry = \frac{0.61}{0.4785} = 1.27g \quad (1.12)$$
$$Rz = \frac{0.29}{0.4785} = 0.61g \quad (1.13)$$

Una vez definidos los componentes que definen nuestro vector de fuerza inercial, si tenemos constancia de que la única fuerza que actúa sobre el acelerómetro es la gravedad, entonces podemos tener en cuenta que esta será la dirección de nuestro vector de fuerza.
También nos sería posible obtener la inclinación del acelerómetro en relación con la tierra, para ello se calculará el ángulo entre este vector y el eje Z. Además, es posible obtener la dirección de inclinación por eje dividiendo éste entre la inclinación en los ejes X e Y que podremos calcular como el ángulo entre el vector gravedad y los ejes X/Y.

Volviendo a la figura 5 y añadiendo los ángulos entre los ejes X, Y, Z y el vector de fuerza R.

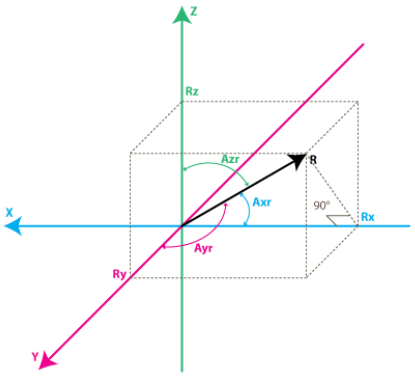

Figura 1.6. Modelo acelerómetro fijado a sistema de coordenadas

Dichos ángulos será posible calcularlos como:

$$\cos(AxR) = \frac{R_x}{R} \rightarrow AxR = arcos\left(\frac{R_x}{R}\right) \quad (1.14)$$

$$\cos(AyR) = \frac{R_y}{R} \rightarrow AyR = arcos\left(\frac{R_y}{R}\right) \quad (1.15)$$

$$\cos(AzR) = \frac{R_z}{R} \rightarrow AzR = arcos\left(\frac{R_z}{R}\right) \quad (1.16)$$

Aunque mediante un acelerómetro podemos obtener la inclinación con ayuda de un giróscopo podremos obtener medidas más inclinación más exactas.

1.2.1.2 ¿Qué es un giróscopo?

Desde nuestra niñez, hemos sido testigos de determinados efectos giroscópicos, desde el lanzamiento de una peonza, viendo como esta se mantenía estable y no caía hasta pasado un tiempo, o como el hacer girar una moneda desde uno de sus bordes.

Figura 1.7. Ejemplo demostrativo de un giroscopio.

Lo peculiar de un giróscopo, es que al aplicar una fuerza que la lógica nos dice que debería tumbar nuestro elemento este al contrario se mantiene estable y rota sin detenerse (definitivamente se detiene y cae debido al rozamiento).

Esto es posible por dos propiedades:
-**Inercia giroscópica**: En la primera ley de Newton se hace referencia a la inercia, en ella se nos dice que un cuerpo solo puede cambiar su estado inicial si le es aplicada una fuerza.
-**Precesión**: Si aplicamos una fuerza que tienda a modificar el plano de rotación del giróscopo la precesión será la inclinación del eje en ángulo recto.

Una vez ya tenemos una idea del funcionamiento físico de un giroscopio volvemos al modelo del acelerómetro que hemos visto en la figura 6.

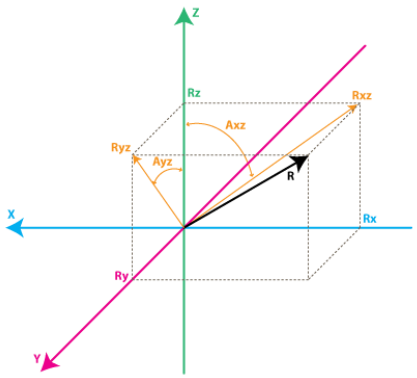

Figura 1.8. Modelo acelerómetro fijado a sistema de coordenadas y añadidos ángulos de giro.

Un giroscopio que dispusiera de 2 ejes dispondría de 2 canales los cuales medirán la rotación de estos ejes.
Definimos como:
R_{xz} → Proyección del vector de fuerza inercial (R) en el plano XZ
R_{yz} → Proyección del vector de fuerza inercial (R) en el plano YZ
A_{xz} → Ángulo entre R_{xz} y el eje Z
A_{yz} → Ángulo entre R_{yz} y el eje Z

Debemos tener claro que lo que medirá un giroscopio es la velocidad de cambio de estos ángulos, o también llamada velocidad angular.
Ahora vamos a pasar a un ejemplo con el que podremos entenderlo muy fácilmente, en dicho ejemplo medimos el ángulo de rotación alrededor del eje Y, dicho ángulo sería lo que hemos llamado previamente como A_{xz}, además dicha medida diremos que ha sido tomada en el instante $t_{inicial}$. Previamente realizaremos otra medida en el instante t_{final}.
Una vez disponemos ya de estas dos medidas podemos calcular dicha velocidad como:

$$VA_{xz}(grados/s) = \frac{A_{xz\,final} - A_{xz\,inicial}}{t_{final} - t_{inicial}} \tag{1.17}$$

Pero como pasa en el caso del acelerómetro no se proporciona la medida deseada si no que se obtiene un valor de ADC por lo que deberemos utilizar una ecuación para obtenerlo, será muy similar a la utilizada por el acelerómetro:

$$VA_{eje\_gyro}\left(\frac{grados}{s}\right) = \frac{\frac{ADC_{plano} * V_{gyro}}{1023} - V_{offset}}{Sensibilidad} \tag{1.18}$$

Tanto la tensión de offset como la sensibilidad podremos obtenerla de nuestro datasheet, en este caso para realizar unos cálculos de prueba utilizaremos una sensibilidad típica de 2 $\frac{mv}{deg/s}$ y como tensión de offset 1.3 V. En cuanto a la tensión de la alimentación del giroscopio la tomaremos como 3.3 V por lo que nuestro V_{gyro} tendrá ese valor.

A continuación, tomaremos como ejemplo los siguientes datos obtenidos por un giroscopio:
$ADC_{XZ=552}$ (1.19)
$ADC_{YZ=346}$ (1.20)

Seguidamente aplicaremos la siguiente ecuación:
$VA_{XZ=}240.3\left(\frac{grados}{s}\right)$ (1.21)
$VA_{YZ} = -91.9\left(\frac{grados}{s}\right)$ (1.22)

Con esto ya tenemos nuestra velocidad de giro alrededor del eje X y del eje Y. En cuanto a la dirección, dentro del datasheet del módulo se puede encontrar cual es la dirección positiva y cual la negativa o como en este caso en la serigrafía del módulo ya aparece indicada cuál es la dirección natural del eje.

Una vez que ya tenemos claro cómo obtener las medidas en las unidades deseadas tanto del acelerómetro como del giroscopio ya podemos pasar al siguiente módulo, en este caso el GPS. Más adelante en la implementación del código se explicará cómo combinar las medidas del acelerómetro.

1.2.2 GPS principios básicos

El programa NAVSTAR GPS (Navigation System Timing And Ranging, Global Positioning System) comenzó en el 1973 por el Departamento de Defensa de los Estados Unidos, y unos años después en 1978 se produjo el lanzamiento del primer satélite.

Previamente en la década de los 60 el ejército americano había iniciado otro programa orientado a la ubicación por medio de satélites de elementos en la tierra, el programa TRANSIT, dicho sistema se basaba en observaciones en observaciones Doppler de dos señales portadoras que se emitían desde un satélite pero dichos satélites tenían una escasa altura de sus órbitas y estas señales eran muy afectadas por las variaciones del campo de gravedad, además de tener un número limitado de satélites (5-7) por lo que había mucho hueco entre observaciones.

El Sistema de Posicionamiento Global o más conocido como sus siglas GPS es un sistema que nos permite obtener las coordenadas espaciales de puntos respecto a un sistema de referencia mundial. Dichos puntos pueden estar ubicados en cualquier parte del planeta, estos puntos pueden ser tanto estáticos como estar en movimientos, además dicho sistema funciona en cualquier hora del día.

Este sistema se basa en el cálculo de distancia desde el receptor a cuatro satélites como mínimo. Estas distancias se obtienen mediante señales emitidas por lo satélites en las cuales podemos encontrar información referente al identificador del satélite, hora del reloj interno a la que fue lanzada la señal, además de información sobre la constelación.
El receptor es el encargado de realizar el cálculo de las coordenadas ya que al disponer de la hora a la que fue emitida la señal de cada satélite y disponiendo de la hora a la que se recibió puede calcular a la distancia de la que se encuentra de cada satélite.
Una vez se dispone de las cuatro distancias por medio de geometría se realiza la ubicación del receptor en la tierra, básicamente al adquirir la primera distancia del satélite el receptor es consciente de que está situado en la superficie de una esfera teniendo este el satélite como centro.

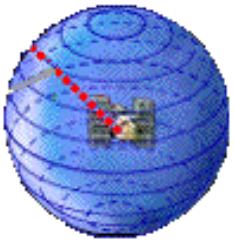

Figura 1.9. Representación distancia desde un satélite.

Al recibir la distancia de un segundo satélite disponemos de una segunda esfera, esta nos proporcionará otra distancia con la cual descubriremos que estamos en un punto dentro de la intersección entre las dos esferas, como la ubicación debe estar en la superficie de las esferas ya sabemos que nuestra ubicación estará ubicada en el borde de una circunferencia.

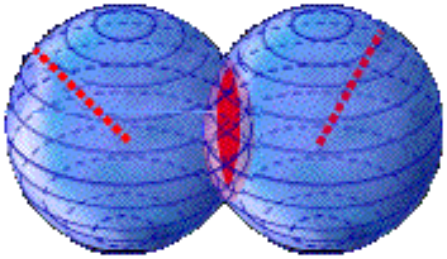

Figura 1.10. Representación distancia desde dos satélites.

Al añadir la distancia de un tercer satélite una tercera esfera entra en contacto con las dos anteriores y nos encontramos que nuestra circunferencia se ha visto reducida a dos puntos, los puntos amarillos de la figura 10.

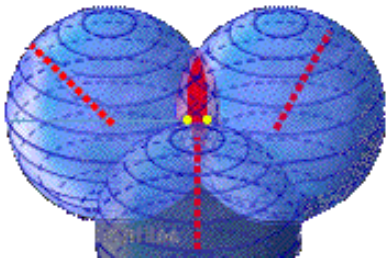

Figura 1.11. Representación distancia desde tres satélites.

Por lo que con 3 satélites podemos obtener nuestra posible ubicación en dos puntos de la tierra, podríamos encontrar nuestra ubicación ya que simplemente observando nuestras coordenadas veríamos que una de las dos está en un punto muy lejano en la tierra por lo que no sería posible, pero con la ayuda de un cuarto satélite evitaríamos ese problema y obtendremos ya nuestra ubicación exacta.
Hemos estado hablando de distancias, pero no se ha mencionado cómo se calcularían dichas distancias, las cuales se calculan sencillamente ya que sabemos a la velocidad que viajan las señales (velocidad de la luz) y sabemos el tiempo que han estado viajando.

$$Distancia = Velocidad * Tiempo$$

Como podemos deducir de está ecuación, es que el tiempo es un parámetro muy importante en este sistema, por lo que los relojes internos deberían ser lo más exactos posibles. Dentro de los satélites prácticamente no hay problema ya que estos disponen de relojes atómicos que son prácticamente perfectos con ínfimas derivas al pasar de los años, pero en el caso de los receptores no es imposible disponer de este tipo de relojes, ya que el precio de estos se encarecería drásticamente.

Para ello se utiliza una sencilla solución, podemos suponer que, si los relojes de nuestros receptores fueran perfectos todas las distancias entre los satélites y nosotros intersectarían en nuestra ubicación, pero como no lo son con 4 medidas en las cuales habrá discrepancias el receptor GPS será capaz de obtener una deriva y aplicarla para mejorar su cálculo de las distancias.

Todo esto es una mera introducción ya que este sistema es bastante más complejo ya que no se puede tomar como literal la velocidad de la luz, y se debe tener en cuenta que las capas de la atmósfera afectan a la medición.
Una vez ya hemos entendido como nuestro GPS nos ubicará en la tierra ahora pasamos a explicar la información que realmente nos proporcionará nuestro módulo GPS.
Dicha información dispone de una estructura siguiendo el estándar NMEA 0183 el cual recibe el nombre “National Marine Electronic Association”, esta asociación se originó en la década de los 80 en estados unidos entre distintos fabricantes del sector náutico en estados unidos ya que cada

fabricante estaba diseñando protocolos diferentes para sus dispositivos sin haber ningún tipo de consenso entre ellos, lo que evitaba cualquier tipo de compatibilidad.

Al ser un estándar orientado a la náutica, pilotos automáticos etc. dispone de diferentes tramas, como se puede suponer sólo una pequeña parte de estas tramas está dedicada al GPS y a continuación vamos a detallar una de ella, en concreto la trama "$GPRMC". Dicha trama nos proporcionará la información suficiente para poder obtener nuestra latitud y longitud.
Para este proyecto utilizaremos la trama $GPRMC. Del por qué escoger esta y no otra de las que existen dentro del estándar NMEA es simplemente porque dicha trama contiene toda la información que necesitamos para nuestra ubicación, coordenadas y velocidad.
A continuación, vamos a tomar una trama obtenida por el GPS y la desglosaremos para explicar todos los campos que la componen.

$GPRMC, 190500.00, A, 3927.87042, N, 00027.73465, W, 0.060, , 300818, , , A*6B
Donde:

190500.00	19:05:00. Hora en formato UTC
A	Nos indica si nuestra conexión GPS está activa o valor S en caso de no tener señal.
3927.87042	Nuestra latitud es 39º y 27.87042'.
N	Ubicados en el hemisferio norte.
00027.73465	Nuestra longitud es 0º y 27.73465'.
W	Hemisferio oeste.
0.060	Es nuestra velocidad en nudos.
	En este campo vacío cuando se cumplen las condiciones aparecerá un ángulo de seguimiento para estimar posiciones siguientes.
300818,,, A*6B	En este campo se nos indica nuestra fecha 30/08/2018 y la variación magnética.

Al principio me extrañaron estas medidas ya que como en la empresa que estoy trabajando realizó la gestión de dispositivos que están geolocalizados asiduamente veo coordenadas de ubicaciones en valencia y no coincidian.
Esto era porque las librerías de visualización de mapas que se utilizaban como puedan ser Google Maps y OpenStreetMaps y prácticamente todas usan las coordenadas en grados decimales y los GPS proporcionan las medidas en grados y minutos.

Para ello se debe realizar unas sencillas operaciones:

$$\frac{27.87042}{60} = 0.4645$$
$$\frac{27.73465}{60} = 0.4622$$

Simplemente hemos dividido los minutos entre 60 ya que 1 grado tiene estos minutos. Sencillamente concatenamos teniendo en cuenta que si estamos ubicados en el oeste debemos agregarle un signo negativo a la longitud, en el caso de que estuviéramos ubicados en el sur se le añadiría un signo negativo a la latitud.
Realizando todo esto ya tenemos nuestra coordenada en grados decimales para poder buscar en google la cual queda como 39. 4645,-0. 4622
A continuación, con una simple búsqueda en google podemos comprobar que ahora la visualizamos correctamente.

Figura 1.12. Ubicación en google obtenida por nuestro GPS.

1.3 Raspberry Pi, breve introducción y porque es idónea para proyectos similares.

Para este proyecto se ha elegido la placa Raspberry Pi, dicha placa es un pequeño ordenador de bajo coste que ha sido desarrollado por la Fundación Raspberry Pi inicialmente con la idea de acercar la informática y la programación a la escuela.

Figura 1.13. Raspberry Pi 3 Modelo B.

Aunque sigue teniendo una fuerte dedicación a esta idea, proyectos en escuelas, orientados a que países del tercer mundo puedan tener ordenadores de muy bajo coste para sus alumnos etc. podríamos decir que actualmente no es la principal utilidad de estos pequeños ordenadores.
La fundación no habría esperado la cabida que ha tenido entre los Geeks, Makers, etc. para sus proyectos personales. Desde videoconsolas retro, centros multimedia, alarmas, estaciones meteorológicas hasta un proyecto que vi hace poco de un comedero automático para perros que el dueño podía controlar desde su Smartphone.
Los bajos costes de productos como Raspberry, Arduino, etc. y su potencia para su escaso tamaño han dado alas a la imaginación de mucha gente.
Para este proyecto se ha elegido el último modelo lanzado por la empresa, la Raspberry Pi3 B dicho modelo tiene unas medidas reducidas de 85.6mm x 56mm x 21mm, además cuenta las siguientes conexiones:

- 40 Pins GPIO
- 4 puertos USB 2
- Salida de Audio Jack 3.5mm
- Salida HDMI
- Alimentación mediante micro USB de hasta 2.5 Amperios
- Slot MicroSd
- Puerto de Cámara
- Puerto LAN 10/100
- Bluetooth 4.1
- Wifi

- Puerto DSI Display

Como podemos comprobar muy completa tiene prácticamente cualquier necesidad aun disponiendo de unas medidas reducidas.
Además, en cuanto a procesamiento y memoria la Raspberry Pi dispone de:

- CPU ARMv8 Quad Core 1.2GHz 64-bits BCM2837
- Memoria Ram 1GB compartida con la GPU

Hablando ahora del sistema operativo que correrá la Rasbberry Pi, está principalmente usa sistemas GNU/Linux. En concreto usualmente se suele utilizar la distribución Raspbian siendo esta una distribución derivada del SO Debian pero optimizada para funcionar correctamente en las placas Raspberry Pi.
Una gran ventaja en cuanto al sistema operativo es que al utilizar como disco duro una tarjeta micro SD en nuestra Raspberry podemos tener diferentes tarjetas con distintos sistemas o distintas aplicaciones, configuraciones implementadas en cada una y cambiar la utilidad de nuestra Raspberry rápidamente simplemente intercambiando las tarjetas.
Todo esto por un relativo bajo coste de aproximadamente 40 euros, bueno añadiendo módulos, sensores, periféricos para nuestra aplicación, pero como podemos comprobar su prestación contra su coste le hacen ser un producto muy demandado.

Este proyecto también podría haber sido abordado con un Arduino, muchos proyectos de este estilo están implementados en el sobre todo los orientados a la navegación de drones ya que además de tener un menor coste tiene una menor complejidad, pero dado que la idea es construir un GPS que en un futuro pudiera ser visualizado en tiempo real en un mapa con la incorporación de una pantalla y debería ser fácilmente trasladable la Raspberry Pi es el candidato perfecto ya que además incorpora una entrada diseñada para incorporar pequeñas pantallas y incluso en caso de necesidad dispone de una salida HDMI.

Figura 1.14. Raspberry Pi más Pantalla integrada

1.4 Lenguaje Python

Figura 1.15. Logo Python.

Los inicios de Python datan de finales de los 80 en los Países Bajos por parte de Guido Van Rossum el creador de este proyecto.
El nombre viene de los famosos humoristas Monty Python ya que Guido era un gran aficionado suyo. Dicho lenguaje fue creado con la intención de que tuviera una sintaxis muy limpia consiguiendo con esto que al leer un código implementado en Python su legibilidad resulta muy sencilla.

Python además es un lenguaje de programación multiparadigma, ya que permite programación imperativa, orientación a objetos y incluso programación funcional. Además, otra gran ventaja es que es multiplataforma, lo que te permite ejecutarlo en Windows, MacOs, páginas web o en un servidor.
También es un lenguaje interpretado, esto hace que no sea necesario compilar un código fuente lo que nos hace ganar en velocidad conforme vamos desarrollando un proyecto. Introduciendo una breve explicación de lo que es un lenguaje interpretado, es que éste no compila nuestro código en lenguaje máquina que pueda comprender un ordenador, para ello un lenguaje interpretado necesita de un intérprete. Esto hace que al ejecutar dicho código en una máquina la ejecución sea un poco más lenta, pero por otra parte ganamos en la portabilidad que hoy en día con tantas diferentes plataformas es un gran aliciente.

Otra de las ventajas de Python es que al tener un intérprete que dispone de línea de comandos donde podemos ir probando pequeñas partes del código, nos permite hacer pruebas y una depuración muy rápida, pero no debemos olvidar otra gran ventaja de Python y es que es un lenguaje orientado a objetos, lo que nos permite poder reutilizar bloques de nuestro código muy fácilmente.

Además, al ser un lenguaje tipado no es necesario tener que declarar las variables si no que ellas mismas establecen su tipo conforme se va ejecutando el programa dependiendo el valor que se le esté asignando. Otra cosa a tener en cuenta en Python es que dicho lenguaje no utiliza separadores, ni llaves en condicionales, funciones etc. Dispone de una notación indentada de obligado cumplimiento lo que quiere decir que estos símbolos serán sustituidos por tabulaciones.
Python como hemos mencionado es un lenguaje muy simple visualmente por los detalles que hemos mencionado, sin necesidad de declarar variables, sin llaves ni palabras clave simplemente tabulaciones, etc., todo esto hace que Python sea uno de los lenguajes más recomendados para comenzar en la programación

Para terminar esta breve introducción sobre el lenguaje Python diré que en la actualidad conviven dos versiones de Python diferentes, Python 2 y Python 3, siendo esta la última versión. Aunque Python 3 sea la versión más actual con bastantes cambios sintácticos todavía Python2 hoy en día sigue teniendo un gran público ya que hay mucho material y aplicaciones desarrolladas en él.
Por su simplicidad y a la vez potencia, además de la ventaja de que sea un código reutilizable en diversas plataformas, se ha elegido este lenguaje, siendo también un lenguaje muy interesante para aprenderlo ya que tiene una gran comunidad creciente donde Python se utiliza cada día más en muchos tipos de aplicaciones como por ejemplo pueda ser hoy en día en Seguridad.

1.5 Especificaciones técnicas de los módulos IMU y GPS

El GPS elegido para este proyecto ha sido el módulo NEO-7M de la empresa suiza UBLOX, en la figura 14 podemos ver una imagen de él.

Figura 1.16. Modulo NEO-7M.

Dicho módulo tiene como principales características:

Receiver type	56 channels, GPS L1(1575.42MHz) C/A code, SBAS:WAAS/EGNOS/MSAS
Horizontal position accuaracy	2.5mCEP (SBAS:2.0mCEP)
Navigation update rate	10Hz máximum (1Hz default)
Capture time	Cool start: 27s (fastest), Hot start:1s
Tracking & Navigation sensitivity	-162dBm
Communication protocol	NMEA(default)/UBX Binary
Serial baud rate	4800, 9600, 19200, 38400, 57600, 115200
Operating temperatura	40-85ºC
Operating voltage	2.7-5V
Operating current	35mA
TXD/RXD impedance	510Ω

Tabla 1. Características GPS.

Más adelante cuando hablemos de las conexiones con la Raspberry Pi hablaremos de los pines que dispone este módulo y donde realizar sus conexiones.
Este módulo ha sido escogido principalmente por varios puntos, uno de ellos tiene que ver con UBLOX, siendo esta de las empresas más importantes en este sector y sus módulos GPS unos de los más utilizado, lo que nos facilita que haya mayor soporte para ellos.

Además, lo bueno es que UBLOX dispone de un programa propio para poder configurar sus módulos y donde poder jugar con ellos. A mí personalmente me ha parecido muy interesante este programa ya que me ha permitido aprender mucho de este módulo, ya que pudiendo acceder a todas las posibilidades del módulo desde una interfaz gráfica es mucho más amigable que desde líneas de código.
Con este programa podemos ver tramas en tiempo real del módulo, ubicar en un mapa, ver derivas de la medida y un sin fin de posibilidades. Más adelante cuando hablemos de la configuración del módulo se mencionará más sobre este programa llamado UBLOX Center.
Tampoco todos los módulos que se ofrecen son capaces de trabajar a una frecuencia de 10Hz pudiendo obtener 10 coordenadas por segundo cosa que nos es muy útil ya que como hemos mencionado anteriormente la IMU realizará muchas medidas por segundo.

En cuanto a la IMU elegida para este proyecto se ha elegido una IMU que incorpora un chip de la serie MPU9250 de la empresa InvenSense, dicha IMU esta comercializada por Waveshare Electronics. En la figura 1.17 podemos ver una imagen del Sensor IMU.

Figura 1.17. Sensor IMU

Dicha IMU incorpora un acelerómetro, giroscopio y una brújula digital, todos estos de 3 ejes cada uno. Además, incorpora un sensor de presión.

A continuación, podemos ver sus principales características:

Driver IC	MPU9255	16-Bit AD converter Gyroscope full-scope range: ±250, ±500, ±1000, ±2000°/sec Accelerometer full-scale range: ±2, ±4, ±8, ±16g Compass full-scale rage：±4800uT
	BMP180	Built-in temperature sensor with temperature measurement compensation Pressure measuring range: 300~1100hpa (+9000m ~ -500m relating to sea level) Accuracy: 0.02hPa (0.17m)
Working voltage	3.3-5V	
Supported interface	I2C	
Dimensions	31.2x17mm	

Tabla 2. Características IMU.

Como se puede ver en la tabla 2 dicho modulo se comunica por el interfaz I2C por el cual accederemos a las medidas de cada componente y eje por separado.
Dicho módulo también incorpora un regulador de voltaje que permite operar entre 3.3 a 5 V.
Ya hemos mencionado anteriormente que este módulo incorpora un sensor de presión con el cual sería posible si fuera necesario la altitud a la que se encuentra la IMU. Si se necesitara calcular dicha altitud sería muy importante calibrarlo inicialmente, ya que como se comenta en la documentación de este en la primera medida diferirá mucho de la real. En el datasheet del módulo podemos encontrar toda la información de cómo calcular dicho error para realizar la corrección.

En dicho datasheet también podemos encontrar la dirección de los registros que debemos acceder, a continuación, podemos ver una captura del datasheet donde se muestran los registros que vamos a utilizar:

3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]
41	65	TEMP_OUT_H	R	TEMP_OUT_H[15:8]
42	66	TEMP_OUT_L	R	TEMP_OUT_L[7:0]
43	67	GYRO_XOUT_H	R	GYRO_XOUT_H[15:8]
44	68	GYRO_XOUT_L	R	GYRO_XOUT_L[7:0]
45	69	GYRO_YOUT_H	R	GYRO_YOUT_H[15:8]
46	70	GYRO_YOUT_L	R	GYRO_YOUT_L[7:0]
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT_H[15:8]
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT_L[7:0]

Figura 1.18. Captura Registros.

Además, podemos encontrar instrucciones para configurar todas las opciones posibles tanto del giroscopio como del acelerómetro, como cambiar su sensibilidad o interpretar los datos recibidos. Esto era necesario ya que al contrario que con el GPS que se disponía del programa UBLOX Center para su configuración este módulo era necesario configurarlo enviando comandos a través de la interface I2C al sensor IMU.

2 CONFIGURACIONES PREVIAS

2.1 Preconfiguración del módulo GPS con ayuda de UBLOX Center

Para la configuración inicial de nuestro módulo se podría realizar mediante comandos incluidos en nuestro código informándonos en el datasheet de dicho módulo cuales son los registros que debemos modificar y con qué comandos.
En este caso esto se puede realizar mucho más sencillo además de poder ver visualmente todas las posibilidades que nos ofrece este módulo con ayuda de un programa suministrado por la empresa Ublox.

Figura 2.1. Logo programa U-Center.

Al no estar este programa disponible para Linux no era posible ejecutarlo en la Raspberry por lo que adquirí un cable conversor que convierte la salida serial del módulo GPS a USB, para su conexión con el ordenador.

GPS	CABLE
Vcc	Rojo
Gnd	Negro
Rx	Verde
Tx	Blanco

Figura 2.2. Cable Serial-USB y conexiones.

Una vez ya conectado al iniciar nuestro programa y conectado al puerto pertinente se comenzaba a visualizar las tramas recibidas por el módulo, además de información varia.
En la figura 2.2 podemos ver las tramas recibidas en ese momento.

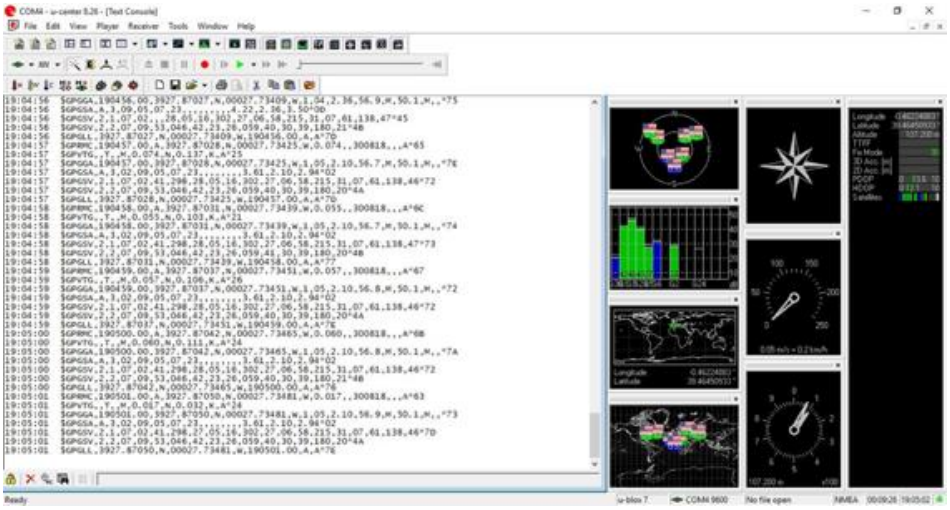

Figura 2.3. Panel principal U-Center con visualización de tramas NMEA.

Además, el programa nos ofrecía más información tales como observar la desviación de las medidas, en la figura 2.4 la podemos ver en un mapa o en un eje.
Cada radio de dicha imagen eran 12.5 metros por lo que se puede observar que la desviación se encontraba en un círculo de radio 6 metros aproximadamente.

Figura 2.4. Desviación de la medida en mapa como en gráfico circular.

También podemos ver información sobre el número de satélites a los que estamos conectados, velocidad, altitud.

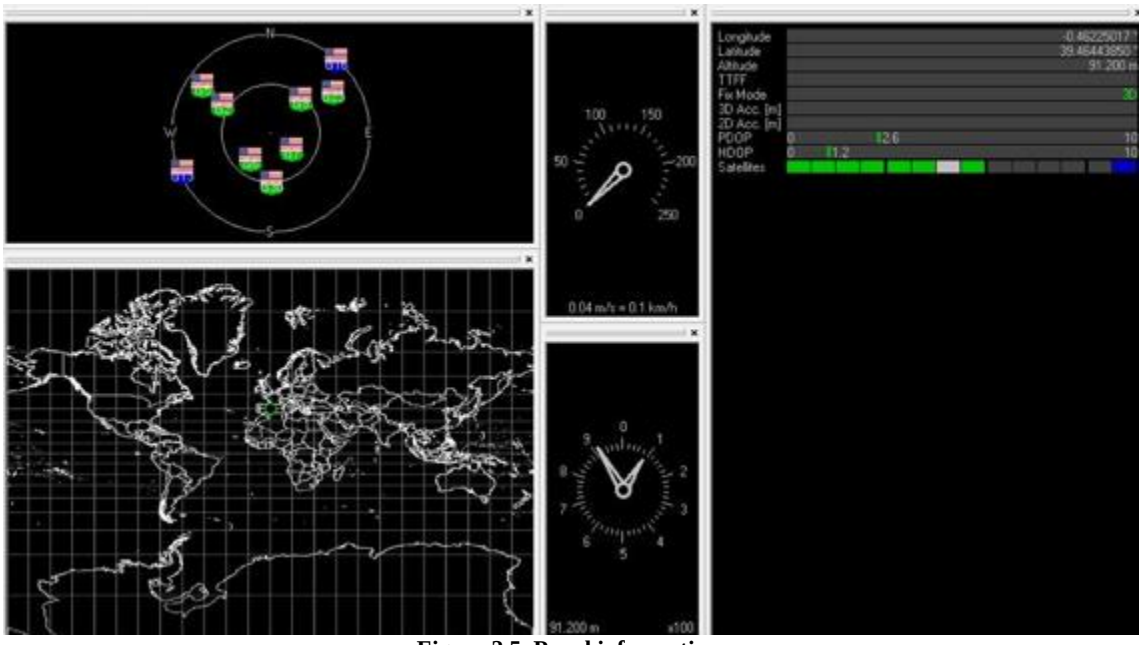

Figura 2.5. Panel informativo.

En cuanto a la configuración del módulo había muchas posibilidades de modificación. Desde omitir tramas, cambios de frecuencia, que tipo de uso íbamos a dar al módulo si navegación a pie o mediante coche y un sinfín de opciones.
Como se comentó en el apartado de especificaciones del GPS, dicho módulo era capaz de trabajar a una frecuencia de 10 Hz como máximo, lo que nos permitía hasta 10 medidas por segundo. Esto se realiza sustituyendo el periodo de 1000 ms a 100 y pulsar el botón enviar ubicado abajo de la ventana.

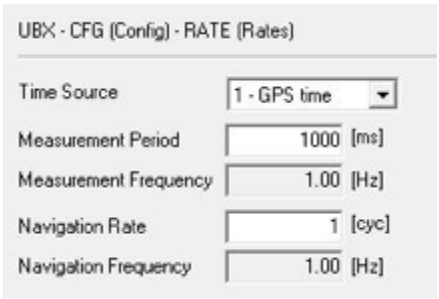

Figura 2.6. Configuración de frecuencia de envió de medidas.

Como se ha comentado también era posible omitir tramas recibidas, además el programa permitía ver en tiempo real las tramas recibidas y un desglose de sus campos con su consiguiente explicación de estos.

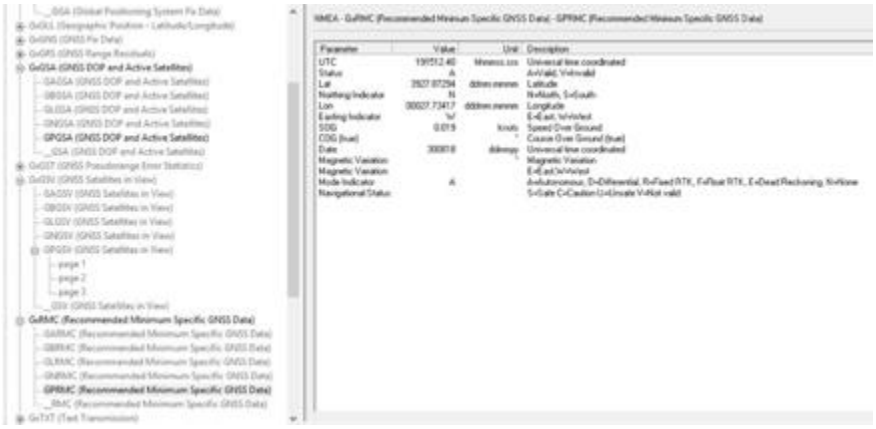

Figura 2.7. Visualización de tramas recibidas y desglose.

Otra característica de este módulo era que no estaba limitado a la utilización de satélites GPS, también era posible la utilización de satélites del sistema Glonass y del sistema Galileo.

Sin entrar mucho en detalle de estos sistemas el sistema Glonass es un sistema controlado por Rusia que fue lanzado en la década de los 80 por parte de la Unión Soviética para competir con el GPS. En cuanto al sistema Galileo, este es el único de los tres controlado por civiles al contrario que GPS y Glonass, ambos controlados tanto por el ejército americano como ruso. Dicho sistema está desarrollado por la Unión Europea y se prevé que sea más exacto en su ubicación y dispondrá de una versión gratuita con una ubicación de alrededor de 1 metro (GPS error de hasta 20 metros) y otra de pago que promete establecer la ubicación con un error de hasta 1 cm. Se prevé que este sistema esté terminado en 2020.
En la figura 2.8 se puede ver cómo era posible la utilización de otros sistemas. Como prueba se trató de utilizar el sistema Galileo únicamente desactivando GPS ya que en teoría ya era posible recibir ya tramas, pero al desactivar el GPS no recibía nada.
En cuanto al formato de trama se prevé que sea el mismo tipo NMEA que hemos tratado anteriormente.

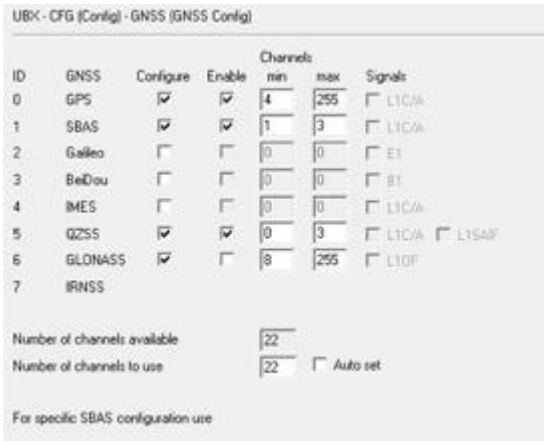

Figura 2.8. Cambio a satélites Galileo y Glonass.

2.2 Conexionado de todos los módulos.

Una vez ya tenemos configurado correctamente el módulo GPS procedemos al conexionado de todas las partes de nuestro proyecto.

En la figura 2.9 podemos ver un mapeado de los pins de la Raspberry. Para la IMU como hemos mencionado anteriormente utilizaremos el bus I2C el cual podemos encontrar en los pines 3 y 5. Y para el GPS utilizaremos los pines donde se encuentra ubicada la UART los cuales son el 8 y el 10.

También deberemos utilizar la alimentación y la tierra de la Raspberry la cual tenemos dos posibilidades ya que los dos módulos nos lo permiten, tanto trabajar a 5V como a 3.3V.

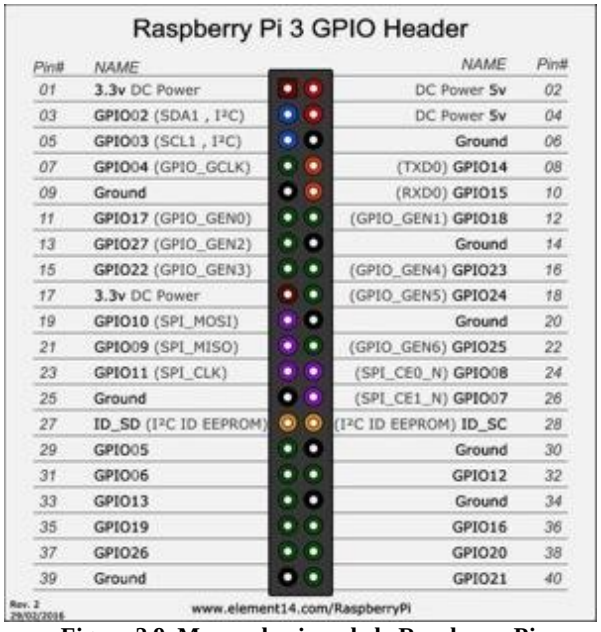

Figura 2.9. Mapeo de pines de la Raspberry Pi.

Para la conexión con los módulos se utilizó una protoboard y un extensor de pines de la Raspberry del que disponía, el cúal facilita mucho la tarea para poder utilizar los pines de la Raspberry. Se realizaron las conexiones pertinentes y se usó la alimentación de 5V.

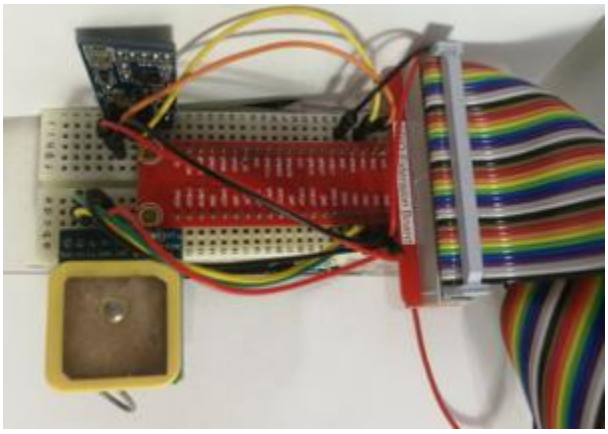

Figura 2.10. Conexión módulos en una Protoboard.

También se le incorporó una pantalla TFT de 7" a la Raspberry la cual fue conectada al conector DSI del que dispone esta. Dicha pantalla ya disponía de agujeros para poder atornillarla a ella y recibía la alimentación de los pines de la Raspberry.

- Tamaño: 194x110x20mm.
- Tamaño de pantalla: 155x86mm.
- Resolución: 800x480 pixeles.
- Alimentación por pin GPIO.
- Cable ribbon para conexión con puerto DSI

Y en la figura 2.11 vemos como debe ir conectada la Raspberry por la parte trasera de la pantalla:

Figura 2.11. Raspberry pi conectada a pantalla TFT.

Otro detalle a tener en cuenta era que debía ser portable ya que para probar el GPS debía ser en exteriores, además que para estar continuamente realizando pruebas de movimientos con la IMU tener un cable de alimentación no resultaba muy cómodo. Por lo que para la alimentación se le conectó una batería portátil que disponía para carga de tablets y móviles. Dicha batería era más que suficiente para las necesidades ya que disponía de 20.000 mA y podía suministrar hasta 2.5A.

Figura 2.12. Batería.

Otro detalle que no tuve en cuenta al principio pero que posteriormente descubrí, fue que al introducir el módulo dentro de la caja le costaba mucho adquirir señal de los satélites por lo que adquirí una antena externa para poder tenerla fuera de la caja y tratar de solucionar este problema. La antena que adquirí tenía las siguientes características:

Frecuencia	800 a 2170 MHz
Conector antena	Sma macho
Conector cable	Sma hembra a ipex
Longitud de antena	11cm
Ganancia de la antena	<2dBi
V.S.W.R	<1.5

Todo esto fue introducido en una caja la cual resultaba muy cómoda para ser transportada y poder realizar movimientos para pruebas con la IMU.

Figura 2.13. Montaje portátil con todo el conjunto.

2.3 Preparación Raspberry Pi

Una vez ya tenemos todo el montaje preparado pasamos a la configuración interna de la Raspberry.

Como ya se mencionó en el apartado de la Raspberry el sistema operativo utilizado para este proyecto será Raspbian. La instalación es muy sencilla, simplemente debemos ir a la página oficial de la distribución:
https://www.raspberrypi.org/downloads/raspbian/

En ella descargamos la imagen de la distribución y a continuación será necesario un programa que realice la instalación de esta imagen en nuestra tarjeta SD. Yo utilice un sencillo programa llamado ApplePi-Baker y fue descargado en el siguiente enlace.

https://www.tweaking4all.com/hardware/raspberry-pi/macosx-apple-pi-baker/

Ahora ya simplemente debemos conectar la MicroSD a nuestro ordenador. Yo he utilizado una tarjeta de 32 Gb ya que ya disponía de esta, pero con una de 8Gb sería suficiente. A continuación, con ayuda del ApplePi cargamos la imagen en la tarjeta y ya podemos introducir está en nuestra Raspberry y ya disponemos de nuestro SO para poder utilizarla.

Una vez ya tenemos nuestro sistema Raspbian corriendo disponemos de varias opciones para poder acceder a ella.

- Conectar monitor por HDMI con ratón y teclado por USB.
- Visualizar todo en pequeña pantalla TFT.
- Conexión por SSH.

La última opción ha sido la elegida ya que es la más cómoda, porque pude utilizar el editor de código que utilizo normalmente (no estaba en Raspbian), dispone de monitor extendido y para búsquedas por internet etc. es más rápido.
Esta conexión resulta muy sencilla en el SO de Apple ya que al estar basado este en UNIX ya dispone de un cliente SSH por lo que simplemente conectándonos por SSH a la IP de la Raspberry ya nos encontrábamos en ella.

```
Juans-MacBook-Pro:~ juan$ ssh pi@192.168.1.110
pi@192.168.1.110's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Aug 28 18:53:22 2018
pi@raspberrypi:~ $
```

Figura 2.14. Captura de conexión SSH.

Una vez ya tenemos conexión con nuestra Raspberry procedemos a la configuración para la utilización del módulo GPS. Lo primero que debemos hacer es actualizar nuestra Raspberry.

```
pi@raspberrypi:~ $ sudo apt-get update
pi@raspberrypi:~ $ sudo apt-get upgrade
```

Y reiniciamos para aplicar las actualizaciones con el comando:

```
pi@raspberrypi:~ $ sudo reboot
```

Ahora debemos configurar la UART a la que será conectada el GPS, para ello debemos editar el archivo de inicio.

```
pi@raspberrypi:~ $ sudo nano /boot/config.txt
```

En este archivo al final debemos añadir lo siguiente:

```
dtparam = spi = en
dtoverlay = pi3-disable-bt
core_freq = 250
enable_uart = 1
force_turbo = 1
```

Una de las cosas por las que debemos añadir esto al inicio del sistema es que la Raspberry utiliza el puerto de la UART para el Bluetooth, por lo que debemos desactivar esto para poder recibir los datos del GPS.
Desactivaremos también una pequeña UART que tiene la Raspberry y lo deshabilitaremos para que no se active cuando la reiniciemos ya que nosotros queremos utilizar la principal ya que esta dispone de más potencia.

```
pi@raspberrypi:~ $ sudo systemctl stop serial-getty@ttyS0.service
pi@raspberrypi:~ $ sudo systemctl disable serial-getty@ttyS0.service
```

Y a continuación debemos activar la UART principal previamente mencionada

```
pi@raspberrypi:~ $ sudo systemctl enable serial-getty@ttyAMA0.service
```

Una vez ya disponemos de nuestra Raspberry ya solo nos quedaría ver que funciona correctamente y podemos recibir las tramas GPS, para ello simplemente podemos ver que recibimos por los puertos GPIO de la Raspberry y comprobar que recibimos las tramas GPS. Para ello debemos utilizar este comando:

```
pi@raspberrypi:~ $ sudo cat /dev/ttyAMA0
```

Y ya podemos observar como estamos recibiendo tramas de nuestro GPS.

```
pi@raspberrypi:~ $ sudo cat /dev/ttyAMA0
?$GPTXT,01,01,01,NMEA unknown msg*58

$GPRMC,095859.00,A,3927.86916,N,00027.73684,W,0.050,,010918,,,A*6A

$GPVTG,,T,,M,0.050,N,0.093,K,A*2C

$GPGGA,095859.00,3927.86916,N,00027.73684,W,1,09,1.00,46.8,M,50.1,M,,*74

$GPGSA,A,3,08,07,26,10,16,18,21,20,27,,,,1.93,1.00,1.65*0A

$GPGSV,3,1,12,04,29,155,23,07,11,312,25,08,34,300,27,10,50,124,41*78
```

Figura 2. 15. Captura de conexión datos recibidos por GPIO de la Raspberry.

Para finalizar la lectura de las medidas lo realizaremos simplemente pulsando ctrl+x.

Una vez ya tenemos configurado nuestro GPS procederemos a configurar el puerto I2C de la Raspberry, el cual será el encargado de recibir la información de la IMU.
Por defecto, no está configurado el puerto I2C por lo que primero debemos hacer es activarlo. Para ello debemos editar el archivo que contiene los módulos.

```
pi@raspberrypi:~ $ sudo nano /etc/modules
```

Y añadir:

```
i2c-bcm2708
i2c-dev
```

También necesitaremos las herramientas del puerto I2C para poder visualizarlo por la línea de comandos. Además, como vamos a querer acceder al puerto a través de código en Python instalaremos un paquete con el que evitaremos conflictos ya que el ya dispone de todas las configuraciones pertinentes.

```
pi@raspberrypi:~ $ sudo apt-get install i2c-tools
pi@raspberrypi:~ $ sudo apt-get install python-smbus
```

Una vez ya tenemos todo configurado ya solo nos queda comprobar que todo está correcto y podemos recibir información por el puerto. Para ello utilizaremos el siguiente comando:

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
```

Donde se nos devolverá un mapeado de las direcciones del bus y como podemos observar se han detectado dos entradas al bus ya que cambian de dos guiones a un valor número con la dirección de nuestro dispositivo.

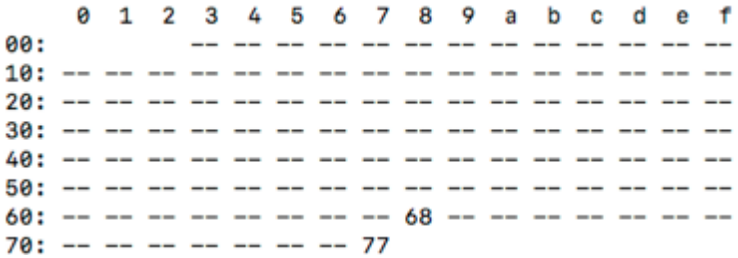

Figura 2.16. Captura de mapeado del bus I2C.

Ahora que ya tenemos todo listo y configurado ya podemos proceder al desarrollo del código de nuestro proyecto.

3 LEYENDO MEDIDAS CON PYTHON

3.1 Lecturas del módulo GPS

El código para realizar la lectura de los datos es muy sencillo, ya que lo único que debemos hacer es escuchar el puerto y ya estaremos recibiendo las tramas.

```
import serial, time, re
import pynmea2

serial = serial.Serial("/dev/ttyAMA0", baudrate=9600)

try:

    while True:

        GPS_NMEA = serial.readline().decode('ISO-8859-1')
        #print (read)
        if GPS_NMEA.startswith('$GPGGA'):
            print ("Trama en bruto", GPS_NMEA)
            msg = pynmea2.parse(GPS_NMEA)
            print("Longitud",msg.latitude, "Latitud", msg.longitude)
except Exception as error:
        print (error)

finally:
        ser.close()
```

Lo único que debemos es informar a que velocidad se realiza la lectura, en este caso a 9600 baudios (esto se obtuvo de la documentación del módulo).
Además, como ya se comentó previamente hay muchos tipos de sentencias dentro de una trama NMEA y en este caso se ha utilizado la $GPGGA que además de latitud y longitud proporcionaba la altitud. En cuanto al parseo de la sentencia en un principio se realizó con código teniendo en cuenta separadores dentro la sentencia como pudieran ser las “,”, pero más tarde encontré una librería llamada PYNMEA2 que facilitaba esto ya que ella ya devolvía toda la información además de ser posible, una muy fácil conversión de grados decimales a grados y minutos además de muchas más opciones.
En el enlace a continuación se puede ver la información de la librería y sus diversas opciones.
https://github.com/Knio/pynmea2

3.1.1 Conversión a coordenadas cartesianas X Y Z

Otro detalle a tener en cuenta que pude observar buscando como realizar la unión de datos de una IMU y las coordenadas proporcionadas por un GPS era que debían estar en el mismo sistema de coordenadas, algo lógico.
Para empezar sin entrar en mucho detalle las coordenadas geográficas son un sistema derivado de sistema de coordenadas esféricas, se utilizan dos coordenadas angulares en este caso latitud (Norte

y Sur) y longitud (Este y Oeste). Dichas coordenadas están alineadas con el eje de rotación y su unidad internacional es el grado sexagesimal.

La latitud nos proporciona la medida del ángulo entre un punto y el ecuador y las líneas de latitud reciben el nombre de paralelos siendo estos paralelos al ecuador.
Además, como se puede decidir al estar situados a igual distancia del ecuador todos los paralelos tienen la misma latitud.
En cuanto a la longitud esta nos da el ángulo a lo largo del ecuador a un punto determinado siendo Greewich la longitud 0. Los círculos que cruzaran siempre los polos son denominados meridianos.

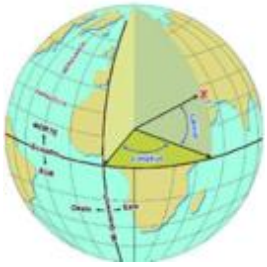

Figura 3.1. Planeta tierra con representación latitud y longitud.

Aunque siendo una versión de las coordenadas esféricas estas tienen otra discrepancia y es que las geográficas sí que pueden estar proyectadas lo que nos permite representar el globo terrestre en una superficie plana como un mapa ya que existe una relación ordenada entre los puntos de la superficie curva y los de la superficie plana.

Ahora que ya hemos entendido brevemente qué es la latitud y longitud ya solo nos queda convertir estas junto a la altura en coordenadas cartesianas X, Y, Z, esto lo haremos ya que como posteriormente se debe unir a las medidas proporcionadas por el acelerómetro el cual trabajara en el eje cartesiano necesario.
Para ello disponemos de las siguientes ecuaciones:

$$\overline{p} = \begin{bmatrix} \overline{p}_x \\ \overline{p}_y \\ \overline{p}_z \end{bmatrix} = \begin{bmatrix} r_s \cos\lambda_s \cos\imath + h\cos\mu\cos\imath \\ r_s \cos\lambda_s \sin\imath + h\cos\mu\sin\imath \\ r_s \sin\lambda_s + h\sin\mu \end{bmatrix} \tag{3.1}$$

$$\lambda_s = \operatorname{atan}((1-f)^2 \tan\mu) \tag{3.2}$$

$$r_s = \sqrt{\frac{R^2}{1+(1/(1-f)^2-1)\sin^2\lambda_s}} \tag{3.3}$$

Dando significado a las variables tenemos:

- Latitud = μ
- Longitud = i
- Altitud = h
- Posición = p
- Nivel del mar = λ_s
- Radio en la superficie definido por el aplanamiento = r_s
- Aplanamiento = f
- Radio ecuatorial = R

A continuación en la siguiente figura podemos ver el código implementado en python que convierte nuestras coordenadas geográficas a cartesianas. En cuanto al factor de aplanamiento se utilizó el estándar WGS84 ya que este es el sistema referencia utilizado por el Sistema de Posicionamiento Global.

```
def geo_to_cart(lat, lon, alt):


  rad = np.float64(6378137.0)
  f = np.float64(1.0/298.257223563)
  cosLat = np.cos(lat)
  sinLat = p.sin(lat)
  FF     = (1.0-f)**2
  C      = 1/np.sqrt(cosLat**2 + FF * sinLat**2)
  S      = C * FF


  x = (rad * C + alt)*cosLat * np.cos(lon)
  y = (rad * C + alt)*cosLat * np.sin(lon)
  z = (rad * S + alt)*sinLat
  return (x, y, z)


  #Las unidades de salida son metros
```

Aprovechando que en el lugar donde trabajo estaba desarrollando una aplicación que debe detectar la cercanía con unos determinados parkings e informar de sus plazas libres comencé a investigar como calcular las distancias entre dos puntos, básicamente cómo un GPS averigua que has llegado a tu destino.
Para ello se utiliza la ecuación de Haversine

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \qquad (3.4)$$

Donde:

- Latitud = Φ
- Longitud = λ
- Radio terrestre = r
-

Aunque dicha ecuación no es del todo exacta ya que toma la tierra como una esfera y eso no es así por lo que han ido apareciendo modelos para mejorar esto incorporando factores de aplanamiento como el WGS84.
Afortunadamente al contrario que para mí aplicación móvil que no encontré ninguna librería de diseñada para esto en JavaScript en Python sí que existía un paquete llamado “Pyproj” que realizaba el cálculo de la distancia pudiendo incorporar tu qué factor de aplanamiento deseabas.
A continuación, en la siguiente figura podemos ver una imagen del código implementado:

```
from math import sin,cos,sqrt,asin,pi
import pyproj, sys
from os import system


lat_casa = 39.464576
```

```
lng_casa = -0.462406

lat_uni = 39.481231
lng_uni = -0.343269

geod = pyproj.Geod(ellps="WGS84") #parametros de forma del elipsoide
angle1,angle2,distance = geod.inv(lng_casa, lat_casa, lng_uni, lat_uni)

#Formula de harvesine básica.
r = 6371000
c = pi/180
d = 2*r*asin(sqrt(sin(c*(lat_uni-lat_casa)/2)**2 + cos(c*lat_casa)*cos(c*lat_uni)*sin(c*(lng_uni - lng_casa)/2)**2))

print ("La distancia entre casa y la universida son %0.2f metros con formula de Harvesine y elipsoide WGS84" % distance)
print ("La distancia entre casa y la universida son %0.2f metros con formula de Harvesine" % d)
```

Se ha realizado la implementación de la ecuación de Harvesine simple para que se pudiera observar el error, a continuación, podemos ver los datos obtenidos:

```
Juan-Mac-2:Desktop juan$ python3 harve.py
La distancia entre casa y la universida son 10416.79 metros con formula de Harvesine y elipsoide WGS84
La distancia entre casa y_la universida son 10392.37 metros con formula de Harvesine
```

Como podemos observar el error es de apenas algo más de 20 metros, si trasladamos este error a grandes distancias puede ser muy significativo.

Y para comprobar que la ecuación funciona he comprobado dicha distancia en google earth.

Figura 3.2. Cálculo de distancia de mi casa a la universidad con Google Earth.

Como podemos ver en la imagen la distancia es prácticamente la misma.

3.2 Lecturas de la IMU

En cuanto a la lectura de los datos procedentes de la IMU tiene un poco más de trabajo ya que como se comentó los datos obtenidos deben ser convertidos.
A continuación, en la siguiente figura se puede ver el código implementado para ello.

```
power1 = 0x6b
power2 = 0x6c

#FUNCIONES
def bus_read(dir):
    data = register_read(dir)
    if (data >= 0x8000):
        return -((65535 - data) + 1)
    else:
        return data
def register_read(dir):
    first = i2c_bus.read_byte_data(imu_dir, dir)
    last = i2c_bus.read_byte_data(imu_dir, dir+1)
    data = (first << 8) + last
    return data

def byte_read(dir):
    return i2c_bus.read_byte_data(imu_dir, dir)

def distance(pos_1,pos_2):
    return math.sqrt((pos_1*pos_1)+(pos_2*pos_2))

def rotation_y(x,y,z):
    rdn = math.atan2(x, distance(y,z))
    return -math.degrees(rdn)

def rotation_x(x,y,z):
    rdn = math.atan2(y, distance(x,z))
    return math.degrees(rdn)

i2c_bus = smbus.SMBus(1)
imu_dir = 0x68
i2c_bus.write_byte_data(imu_dir, power1, 0)

factor_acel = 16384
factor_giro = 131

i=0
try:
```

```
    while True:
        i=i+1

        acel_x = bus_read(0x3b)
        acel_y = bus_read(0x3d)
        acel_z = bus_read(0x3f)

        giro_x = bus_read(0x43)
        giro_y = bus_read(0x45)
        giro_z = bus_read(0x47)

        acel_xout = acel_x / factor_acel
        acel_yout = acel_y / factor_acel
        acel_zout = acel_z / factor_acel

        giro_xout = giro_x / factor_giro
        giro_yout = giro_y / factor_giro
        giro_zout = giro_z / factor_giro

        print ("Giro_X: ", giro_xout)
        print ("Giro_Y: ", giro_yout)
        print ("Giro_Z: ", giro_zout)

        print ("Acel_X: ", acel_xout)
        print ("Acel_Y: ", acel_yout)
        print ("Acel_Z: ", acel_zout)
        time.sleep(0.5)
except Exception as error:
        print (error)
```

En la primera parte del código son declaradas las funciones con las cuales se leerá los registros de la IMU, además para su correcta interpretación estos deben ser convertidos a complemento a dos.

```
def bus_read(dir):
    data = register_read(dir)
    if (data >= 0x8000):
        return -((65535 - data) + 1)
    else:
        return data
```

```
def register_read(dir):
    first = i2c_bus.read_byte_data(imu_dir, dir)
    last = i2c_bus.read_byte_data(imu_dir, dir+1)
    data = (first << 8) + last
    return data
```

Una vez que ya tenemos nuestras funciones para poder leer los registros a continuación procedemos a obtener los datos del acelerómetro y del giroscopio de sus correspondientes registros, dichos registros son obtenidos desde las hojas proporcionada por el fabricante.

```
    acel_x = bus_read(0x3b)
    acel_y = bus_read(0x3d)
    acel_z = bus_read(0x3f)


    giro_x = bus_read(0x43)
    giro_y = bus_read(0x45)
    giro_z = bus_read(0x47)
```

Dichos datos no pueden ser tomados literalmente ya que como se informa en el datasheet es necesario realizar un escalado de estos para poder utilizarlos.
En el caso del acelerómetro se le debe aplicar un factor de $\frac{1}{16384}$ a cada salida y con esto ya tendríamos los valores ejercidos por la gravedad en cada eje. En el caso del giroscopio debemos aplicar un factor de $\frac{1}{131}$ para obtener los grados por segundo.

```
    acel_xout = acel_x / factor_acel
    acel_yout = acel_y / factor_acel
    acel_zout = acel_z / factor_acel


    giro_xout = giro_x / factor_giro
    giro_yout = giro_y / factor_giro
    giro_zout = giro_z / factor_giro
```

Hay que darse cuenta que los valores del acelerómetro por el datasheet sabemos que son proporcionados en g por lo que para poder disponerlos en metros por segundo deberemos multiplicar por la constante de gravedad, 9.8.
Como ya se comentó en el apartado explicativo del acelerómetro, era posible también calcular las rotaciones teniendo como en este caso la gravedad ejercida en cada unos de los ejes. La obtención de estas rotaciones ya se trató en el apartado correspondiente al acelerómetro.

```
def rotation_y(x,y,z):
    rdn = math.atan2(x, distance(y,z))
    return -math.degrees(rdn)


def rotation_x(x,y,z):
```

```
rdn = math.atan2(y, distance(x,z))
return math.degrees(rdn)
```

En una primera toma de medidas ya podemos ver los valores obtenidos por nuestra IMU, a continuación, en la siguiente figura podemos verlos:

```
Rotación eje X:  -0.984083921130012
Rotación eje Y:  -1.8699994968747577
Giro_X:  1.6793893129770991
Giro_Y:  -0.1450381679389313
Giro_Z:  0.21374045801526717
-----
Acel_X:  0.29937744140625
Acel_Y:  -0.16525634765625
Acel_Z:  9.695039062500001
-----
```

En el caso del acelerómetro vemos como en los ejes X e Y la fuerza ejercida por la gravedad es mínima, pero en el caso del eje Z comprobamos cómo obtenemos un valor cercano a 9.8 m/s, la gravedad terrestre.
En el caso de las rotaciones son prácticamente nulas estando en parado como mucho obtenemos variaciones de 1 grado/s pudiendo ser esto tanto por el ruido interno del sensor como por vibraciones existentes en el lugar donde está situada la IMU.

Como ya se ha comentado en el apartado donde se detalla el acelerómetro es un sensor muy ruidoso, como he podido comprobar en las lecturas y posteriormente en una visualización en 3D del movimiento de la IMU.
Como la idea final sería desarrollar un GPS el ruido para el cálculo de distancias sería significativo por lo que busque filtros para poder mejorar dichas respuestas.
Usualmente mayoritariamente se usa el filtro complementario ya que es un filtro que dada aun siendo muy sencilla su implementación da unos resultados aceptables lo que hace que sea uno de los más usados para iniciarse, cuando se trabaja con combinaciones de acelerómetros y giroscopios. Por supuesto existen filtros mejores y más complejos como el filtro del Kalman (explicación más adelante) pero este está bien para una primera aproximación.

Este se comporta como un filtro paso bajo en el caso del acelerómetro y para el giroscopio actúa como filtro paso alto.

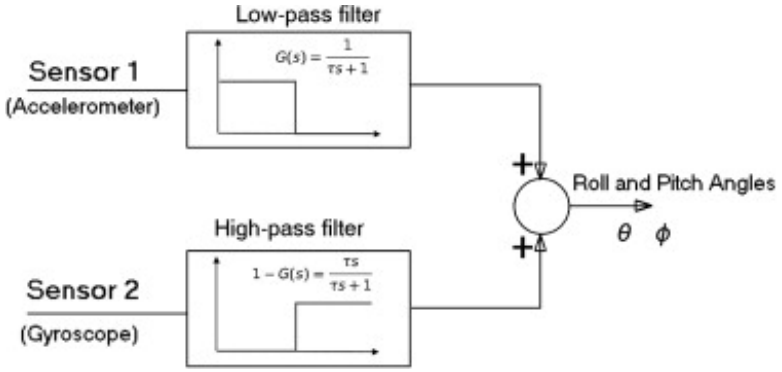

Figura 3.3. Diagrama de bloques filtro complementario

Existen diversas ecuaciones para la realización de este filtro, la que se ha utilizado en concreto para este código ha sido:

$$\theta = k * \theta_{pre} + \theta_{giro} + (1 - C) * \theta_{acel}$$

En cuanto a la constante c debe tener un valor comprendido entre 0 y 1 pudiendo ajustar esta para calibrar el filtro si fuera necesario, pero usualmente suele tener un valor de aproximadamente 0.9. A continuación, podemos ver los resultados una vez implementado el filtro de una rotación alrededor del eje de unos 80-100º.

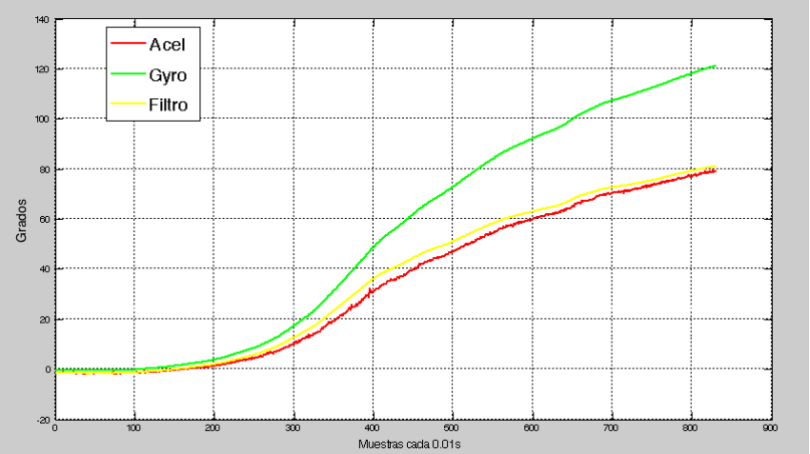

Figura 3.4. Rotación de aproximadamente 90º eje X.

En la imagen anterior no se aprecia la ruidosa que es la medida proporcionada por el acelerómetro, pero a continuación podemos apreciarlo.

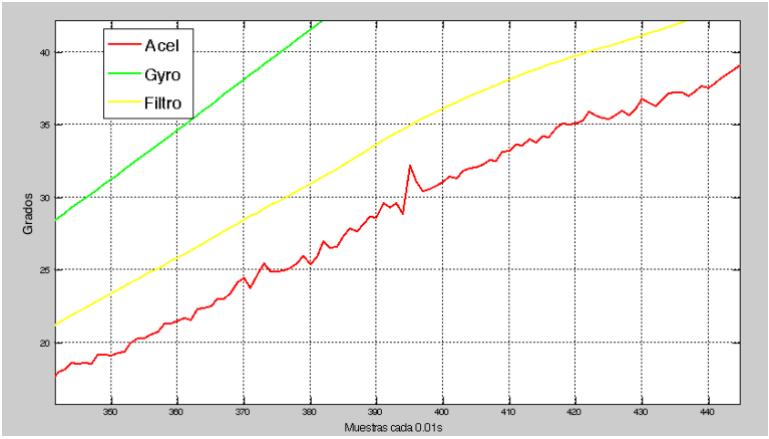

Figura 3.5. Ampliación ruido provocado por el acelerómetro.

Vemos como la línea roja que identifica las medidas del acelerómetro contiene un ruido significativo y encima las medidas del filtro representadas por la línea amarilla con una reducción de ruido notablemente.

A continuación, podemos ver el código implementado en Python para este filtro:

```
t = 0.01
C = 0.98
C_inv = 0.02

( acel_out_x, acel_out_y, acel_out_z, giro_out_x, giro_out_y, giro_out_z,) = read_all()

x_filter_out = rotation_x(acel_out_x, acel_out_y, acel_out_z)
y_filter_out = rotation_y(acel_out_x, acel_out_y, acel_out_z)
```

```
offset_x = giro_out_x
offset_y = giro_out_y

x_filter_out = (x_rot_out) - offset_x
y_filter_out = (y_rot_out) - offset_y

for i in range(0, int(15.0 / t)):
    time.sleep(t - 0.005)

    ( acel_out_x, acel_out_y, acel_out_z, giro_out_x, giro_out_y, giro_out_z,) = read_all()



    gyro_out_x -= offset_x
    gyro_out_y -= offset_y

    delta_x = (giro_scaled_x * t)
    delta_y = (giro_scaled_y * t)

    giro_total_x += delta_x
    giro_total_y += delta_y

    rotacion_x = rotation_x(acel_out_x, acel_out_y, acel_scaled_z)
    rotacion_y = rotation_x(acel_out_x, acel_out_y, acel_scaled_z)

    x_filter_out = C * (x_filter_out + delta_x) + (C_inv * rotation_x)
    y_filter_out = C * (y_filter_out + delta_y) + (C_inv * rotation_y)
```

Ahora procederé a explicar todas las partes del código:

```
offset_x = giro_out_x
offset_y = giro_out_y
```

Procedemos a ajustar los datos del giroscopio por el desplazamiento, el cual es el valor de la lectura del giroscopio en reposo y este ha sido tomado en la primera lectura. Para una aproximación está bien, pero se debería obtener una medida más exacta de calibración.

```
    delta_x = (giro_scaled_x * t)
    delta_y = (giro_scaled_y * t)

    giro_total_x += delta_x
    giro_total_y += delta_y
```

Posteriormente calculamos el delta del giroscopio, simplemente es el valor de cuánto ha girado el giroscopio desde la última lectura y lo agregamos a un total acumulado, con lo que obtendremos un ángulo de rotación únicamente con los valores del giroscopio.

```
rotacion_x = rotation_x(acel_out_x, acel_out_y, acel_scaled_z)
rotacion_y = rotation_x(acel_out_x, acel_out_y, acel_scaled_z)
```

Ahora calculamos la rotación con los valores leídos del acelerómetro y posteriormente procederemos a evaluar el filtro para obtener su valor.

```
x_filter_out = C * (x_filter_out + delta_x) + (C_inv * rotation_x)
y_filter_out = C * (y_filter_out + delta_y) + (C_inv * rotation_y)
```

Para ello tomaremos las últimas medidas y le agregaremos los datos del giroscopio, y lo escalaremos por C, posteriormente le sumaremos la rotación del acelerómetro escalado por C_inv.

3.2.1 Simulación movimientos de la IMU en 3D.

Buscando información sobre combinación de acelerómetro y giroscopio y obtención de datos encontré un programa de visualización en 3D para visualizar datos de acelerómetros, este programa estaba implementado con OpenGl el cual es una API multiplataforma y multilenguaje con la cual podemos escribir programas que se reproduzcan luego en formato gráfico. Además, para seguir lenguaje de este proyecto, el cual está programado en Python y utiliza el módulo Pygame de Python el cual está orientado al diseño de videojuegos.

El funcionamiento era sencillo, se creaba un pequeño servidor en la Raspberry Pi que redirige los datos de salida de nuestra IMU al servidor y este se queda a la espera de recibir petición de la información.
Posteriormente en el ordenador conectamos el código de la aplicación de visionado 3D a la dirección local de nuestra Raspberry.
A continuación, podemos ver algunas imágenes de pruebas con el movimiento viendo cómo coincide la orientación de nuestro proyecto con la simulación.

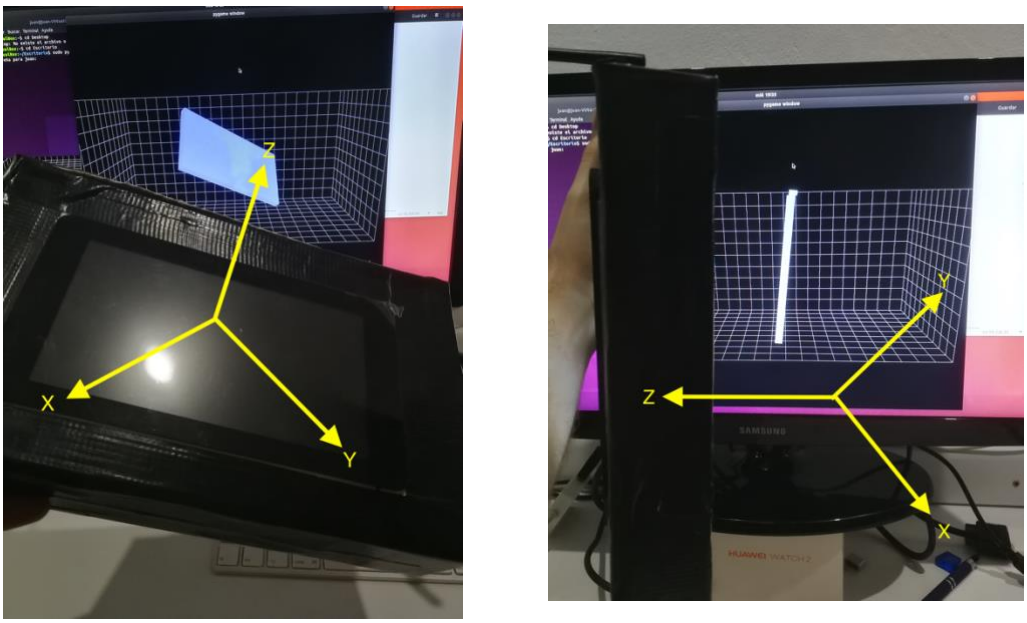

Figura 3.6. Movimientos vistos en físico y simulación 3D.

En estas pruebas fue muy visible el ruido del sensor, sin la aplicación del filtro en la pantalla se notaba una vibración procedente del ruido y aunque posteriormente no fue perfecto si que produjo mejora.

4 COMO UNIR LOS MÓDULOS IMU Y GPS

4.1 Filtro de Kalman, ¿Qué es?

Siendo los dos principales métodos de orientación la navegación inercial y el GPS, y gracias a la evolución y mejora de estas tecnologías durante estos años, nació la idea de integrar estos dos sistemas, habiendo muchas formas de integrar estos sistemas, la más extendida es el filtro de Kalman, aunque hoy en día están surgiendo proyectos de utilización de redes neuronales para dicha tarea.

En concreto el método elegido para este estudio ha sido el filtro de Kalman o también conocido como estimación lineal cuadrática (LQE), el cual nos facilita la estimación del estado de un proceso de forma muy eficiente, además de minimizar la media de los errores al cuadrado, por lo que más que un filtro como se le llama comúnmente, es un estimador.
Con dicho filtro nos es posible obtener información de estados pasados, presentes y futuros.

Rudolf E. Kalman desarrolló el algoritmo para el filtro de Kalman en 1960 y gracias a este filtro, Stanley F. Schmidt mejoró el sistema de guiado del proyecto Apolo de la NASA, el cual tenía problemas de orientación.

Aunque inicialmente su primera utilidad real fue esa, en la actualidad se utiliza desde evaluaciones de tráfico, pasando por economía y un sinfín de utilidades, pero sin olvidar que es el filtro más utilizado en navegación inercial, guiado de misiles, transbordadores espaciales o aplicaciones en las que no es posible depender del GPS, ya sea por ausencia de señal de estos o por miedo a pirateo de la señal como se explicó en el apartado del GPS.

Como la NASA fue la que comenzó a usar generalmente el filtro de kalman pondremos un ejemplo de otro uso que hoy en día le dan ellos al filtro del kalman, calcular la temperatura del interior de la cámara de combustión. En muchos cohetes se utiliza como combustible hidrógeno líquido y éste efectúa una combustión a más de 3000º. Es muy importante controlar la temperatura dentro de la cámara de combustión ya que una temperatura más alta de lo normal podría dañar el motor incluso peores cosas. Pero a temperaturas tan altas no podemos monitorizar el interior de la cámara ya que los componentes electrónicos se deteriorarían o peor se romperían por lo que la solución es tomar la temperatura exterior de la cámara de combustión. Con ayuda del filtro de kalman es posible encontrar la mejor estimación de la temperatura interna a partir de una medición indirecta de la temperatura externa

Aunque se use principalmente para sistemas de navegación inercial sin la existencia de GPS, también es utilizado en la combinación de GPS con navegación inercial incorporada para mejora de la localización y disminución de errores.

Ahora procederemos a realizar una introducción de los elementos que componen el algoritmo del filtro de Kalman y posteriormente se realizará un primer acercamiento a la aplicación del filtro para este proyecto.

4.1.1 ¿Cómo funciona?

Este filtro efectúa estimaciones en cada instante de los diferentes parámetros del sistema, como puedan ser velocidad, altitud o posición los cuales pueden variar en cada momento. Dichas estimaciones se actualizan por las mediciones obtenidas por los distintos sensores, que componen el sistema y no es indispensable que en todas las etapas se contenga información de todos los sensores para obtenerlos, cosa muy importante para este uso ya que hay que recordar que tanto el GPS y la IMU tienen distintas frecuencias de adquisición de datos.

Existe mucha información sobre el filtro de Kalman, pero en la mayoría de casos las explicaciones es pura demostración matemática del filtro, por lo que resulta complicado entender su funcionamiento y las partes que lo componen.

Este filtro se basa en el principio de estimación Bayesiana, la cual se basa en realizar predicciones utilizando la información inicial y mediante un funcionamiento recursivo actualizara dicha información en función de esa información y la actual.

Las dos ecuaciones principales de este filtro son:

$$x_k = Ax_{k-1} + B\mu_k + w_{k-1} \quad (4.1)$$
$$Z_k = H_K x_k + v_k \quad (4.2)$$

Donde:

- x_k = Vector de estados del sistema.
- A = Matriz de transición, es la que relaciona los estados k-1 y k. Solo en ausencia de señales de control.
- B = Matriz de control, es la que Relaciona las señales de control si las hay con el estado actual.
- H = Matriz de salida
- μ_k = Vector de entradas conocidas al sistema o también llamado señal de control (en la mayoría de los casos no existirá).
- w_k= Variable aleatoria que representa el ruido en el proceso.
- v_k = Variable aleatoria que representa el ruido en la medición.
- Z_k= Vector de medidas del sistema

La ecuación 4.1 nos dice que, cualquier x_k, puede evaluarse usando una ecuación estocástica lineal. Recordando lo que es un proceso estocástico, es un proceso el cual sirve para utilizar magnitudes aleatorias que varían con el tiempo. Y será una combinación lineal de su valor anterior más una señal de control y un ruido del proceso. Este ruido es una parte inherente del proceso y no un problema de la medición.

En cuanto a la ecuación 4.2 indica que cualquier valor de medición es una combinación lineal del valor de la señal y el ruido de medición, ambos gaussianos. Otra cosa que debemos considerar es que el ruido del proceso y el de la medición serán estadísticamente independientes, lo que quiere decir que uno no afectara a la probabilidad del otro.

A, B y H idealmente son matrices, pero en muchos procesos dichas entidades son solo valores numéricos y en muchos casos, aunque dichos valores pueden cambiar entre los estados, la mayoría las podemos suponer como constantes.

Debemos estimar la media y la desviación estándar de las funciones de ruido w_{k-1} y v_k. En la vida real ninguna señal es gaussiana pura, pero podemos suponerla así con cierta aproximación.

El algoritmo de kalman intenta converger en estimaciones correctas, aunque los parámetros de ruido de Gauss no están muy bien estimados, pero aun así cuanto mejor calculemos los parámetros de ruido mejores estimaciones obtendremos.

Ahora desarrollaremos el filtro completo. Tendrá dos etapas, una de predicción y otra de corrección. Para ello disponemos de los dos siguientes conjuntos de ecuaciones:

Fase de predicción

$$\hat{x}_k^- = A\hat{x}_{k-1} + B\mu_k + \varepsilon_k \quad (4.3)$$

$$P_k^- = AP_{k-1}A^T + Q \quad (4.4)$$

Fase de predicción

$$K_k = P_k^- H^T (HP_k H^T + R)^{-1} \quad (4.5)$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \quad (4.6)$$

$$P_k = (1 - K_k H)P_k^- \quad (4.7)$$

Como ya se ha comentado las matrices A, B y H muy posiblemente sean constantes, y en muchas ocasiones seguramente sean iguales a la unidad. R y Q son matrices de covarianza para las entradas. Nos informara sobre la desviación de nuestros sensores, pueden ser fáciles de encontrar si tenemos claro las características de los sensores.

Una vez tenemos claras las etapas, deberemos disponer de la estimación para x_0 y P_0 para comenzar a iterar. El valor $\hat{x}_k^-$ será nuestra estimación previa, pero esta será una aproximación ya que todavía no se ha producido la corrección al actualizar la medida. En el mismo caso se encuentra la covarianza de error P_k^-. Estos valores se utilizarán para actualizar la medición con la nueva medida que se ha tomado.

Ya en la etapa de corrección encontraremos $\hat{x}_k$ que serán la estimación en el estado k y la covarianza P_k , las cuales serán necesarias para las estimaciones futuras. En cuanto a ε_k representara las variables aleatorias representativas del proceso y medias independientes, básicamente el error añadido.

En cuanto a la ganancia de Kalman, K_k, que calculamos no se necesita para la siguiente iteración, pero es de lo más importante en este conjunto de ecuaciones, ya que esta ganancia se puede ajustar dependiendo de los resultados que se quieran obtener. Con alta ganancia, el sistema da más peso a las mediciones más recientes, para que el sistema ajuste más rápido y con mayor capacidad de respuesta, en cambio, si la ganancia es menor, el sistema presta más atención a las predicciones.

Dicho de otra forma, una ganancia cercana a la unidad nos dará como resultado una trayectoria estimada de saltos, mientras que una cercana a cero, suavizará el ruido, pero disminuirá la capacidad de respuesta.

4.1.2 Ideas iniciales para la integración de los módulos IMU y GPS con el filtro de Kalman.

Ahora que ya conocemos un poco, como está compuesto el filtro de kalman, llega el momento más complicado y lo que me hubiera gustado obtener un código funcional en Python, para la unión de nuestros dos sensores, se mostrara una primera idea de como debería ser la implementación.

Volviendo a ponernos en antecedentes, los problemas que pueda tener la unión de estos dos módulos son las limitaciones en cuanto al refresco del GPS, de 1 a 10 por segundo como máximo y el acelerómetro que, aunque disponga de captura de medidas bastante mayor acumula muchos errores a lo largo del tiempo.
Por lo que el objetivo será fusionar la precisión del GPS, con la velocidad de refresco de una IMU consiguiendo datos de ubicación a la velocidad de captura de una IMU, 100Hz.

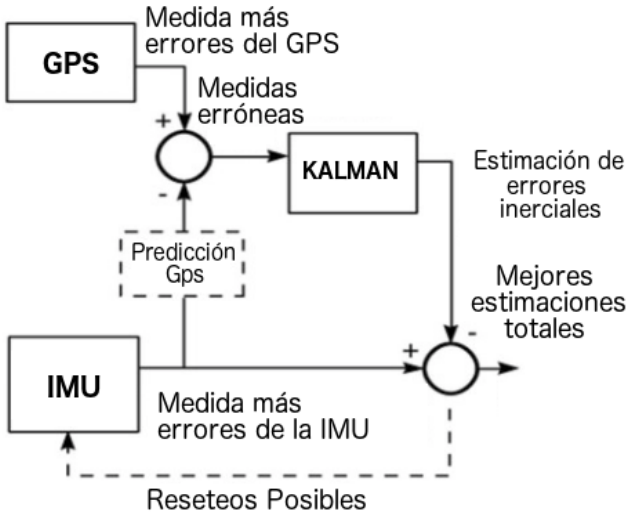

Figura 4.1 Diagrama de bloques de la implementación filtro de Kalman.

La fase de predicción se realizará cada 0.01 segundos con los datos proporcionados por la IMU. Y para ello utilizaremos las ecuaciones 4.3 y 4.4.
En cuanto a la fase de corrección será cada 0.1 segundos con la información del GPS y en esta fase utilizaremos las ecuaciones 4.5, 4.6 y 4.7

Trasladando estas ecuaciones a nuestro sistema los elementos serían sustituidos por:

- La matriz A, sería cómo se vería nuestro sistema ante ninguna entrada, en este caso nuestra posición y velocidad iniciales.
- B definirá las nuevas entradas dadas al sistema.
- μ será nuestra entrada, nuestra aceleración.
- Q será la varianza de la aceleración.
- R será la varianza de la medida del GPS.
- Z será nuestro vector de la medición real del GPS, posición y velocidad.

Esto es una primera idea de cómo se debería trasladar nuestro sistema al filtro de Kalman, las varianzas en el caso de la aceleración podríamos obtenerlas situando el acelerómetro en reposo y calculando su desviación, aunque en el datasheet se incluye un primer valor, por derivas en la fabricación esto puede diferir un poco. En el caso del GPS deberíamos fijarnos en el proporcionado por el fabricante.

5 PRÓXIMOS PASOS

Una vez que tenemos claro todos los elementos que componen el sistema, como obtener los datos capturados por cada uno de los módulos y la realización de la conversión al formato de unidades necesarias, el siguiente paso sería implementar un filtro de Kalman con los dos módulos teniendo claro ya qué medida actúa en cada etapa.

Una vez consiguiéramos tener el filtro en correcto funcionamiento sería hora de pensar en cómo representar gráficamente estas medidas por medio de un mapa para poder ubicarnos. Para esto existirían varias posibilidades. La más rápida y sencilla es proporcionar internet a la Raspberry, por ejemplo, nuestro dispositivo móvil, y utilizar la API proporcionada por google transmitiendo las coordenadas GPS obtenidas por nuestro conjunto. Algo muy importante a tener en cuenta que al igual que habrémos transformado las coordenadas geográficas del GPS a cartesianas para su introducción al filtro, la salida será también en coordenadas cartesianas por lo que para una representación gráfica deberíamos volver a transformarlas, simplemente realizando la inversa de lo que describimos en la sección del GPS.

Lo mas eficiente sería tener incluidos los mapas sin necesidad de estar realizando consumo de internet. Para ello una opción sería utilizar alguna librería de representación gráfica de mapas. Python no dispone de ninguna, pero podríamos utilizar alguna como leaflet escrita en lenguaje Javascript la cual utiliza mapas libres ofrecidos por OpenStreetMaps.

6 CONCLUSIONES

El objetivo de este TFG era el estudio de los elementos que componen un sistema de posicionamiento implementado con un GPS inercial y estudiar su viabilidad de implementación con elementos de bajo coste ya que módulos comerciales que realizan esta función tienen un elevado precio.
Para ello se han seguido los siguientes pasos durante este proyecto:

- Selección de los sensores de coste reducido, en este caso el módulo NEO7 de la empresa Ublox y el módulo inercial MPU-9255 de la empresa InvenSense. A continuación, podemos ver en la siguiente tabla con los costes necesarios para realizar la implementación de este GPS de bajo coste y como podemos comprobar un precio muy inferior a los GPS inerciales en el mercado.

Elemento	Importe
Raspberry PI 3 B+	39,00 €
GPS NEO-7M	21,19 €
MPU-9255	16,48 €
Raspberry Pi Pantalla 7”	71,95 €
TOTAL	**148,62 €**

- Configuración de todos los módulos que lo componen, ya sea por terminal en el caso de las Raspberry, programa Ublox Center para el GPS y mediante envió de comandos por Python a la IMU
- Obtención y análisis de las medidas de cada módulo individualmente, para ello se desarrolló código en el lenguaje de programación Python.
- Investigación y estudio de mejor implementación para la unión de los dos sensores, quedando claro que el filtro de Kalman era la mejor opción y la más extendida en este ámbito.

Cuando comencé este proyecto tenía una idea muy diferente de la implementación que debía tener. Mi idea inicial era simplemente detectar cuando el GPS fallara, dándose el caso de falta de medidas, variación de distancias imposibles etc. Y en ese momento que el acelerómetro tomara el mando calculando este la distancia recorrida y la dirección con ayuda de la última ubicación del GPS como referencia.
Buscando información sobre la navegación inercial fui dándome cuenta que, aunque algunas cosas de mi planteamiento estaban acertadas otras no tanto. Las dos medidas debían combinarse en todo momento ya que una debía rectificar a la otra. En este momento sugerido por mi tutor Vicent Miquel comencé a investigar sobre el filtro de Kalman, el cual ya había ido viendo siempre ligado a la navegación inercial y ahí es cuando me di cuenta que era el camino a elegir.

Pero esto ha resultado una ardua tarea ya que toda la información existente del filtro de Kalman (que hay mucha) suele ser explicación matemática pura y dura la que cuesta mucho de entender y más poder transmitir al mundo real.
Al final he podido comenzar a entender una parte de la funcionalidad filtro, me hubiera gustado poder llegar a implementar el filtro, pero ya no disponía de mas tiempo.

Después de todo esto aunque la navegación por satélite clásica siga siendo la dominante y mejorando cada día sus prestaciones, como por ejemplo con el lanzamiento de los satélites GALILEO los cuales prometen ser mucho más exactos, no podemos olvidar el apoyo que nos brinda la navegación inercial ya que siempre existirán tormentas terrestres, llamaradas solares, o simplemente túneles o reflexiones dentro de ciudades por la altura de sus edificios donde la señal de los satélites pueda no ser la idónea para ubicarnos.

7 BIBLIOGRAFÍA

[1] A.D.KING; Marconi Electronic Systems Ltd, "Inertial Navigation-Forty years of Evolution"

[2] Andrés Marzal Varó, Isabel Gracia Luengo, Pedro García Sevilla,. Universitat Jaume I, Introducción a la programación con Python 3.

[3] Ublox, GPS Essentials of Satellite Navigation.

[4] Escuela Técnica Superior de Ingenieros, Sevilla, "Sistema de Navegación autónomo" http://aero.us.es/na/files1011/T4NA.pdf

[5] Starlino electronics, "A Guide To using IMU (Accelerometer and gyroscope Devices) in Embedded Applications.

[6] Raspberry Pi Foundation, https://www.raspberrypi.org

[7] Python Software Foundation, https://www.python.org

[8] Pedro Gutovnik, Trimble Nagiation Ltd, "Como funciona el Sistema GPS"

[9] National Marine Electronics Association, https://www.nmea.org

[10] Wikipedia, "Coordenadas geográficas," https://es.wikipedia.org/wiki/Coordenadas_geográficas

[11] MathWorks, "LLA (Latitude, longitude and altitude) to ECEF (Earth-centered Earth-fixed)" http://es.mathworks.com/help/aeroblks/llatoecefposition.html

[12] Wikipedia, "World Geodetic System(WGS84)", https://en.wikipedia.org/wiki/World_Geodetic_System

[13] Blog-Bitify, "3D OpenGl visualitation. http://blog.bitify.co.uk/2013/11/3d-opengl-visualisation-of-data-from.html

[14] Boyali, Ali & Hashimoto, N & Matsumoto, Osamu. (2015). A signal pattern recognition approach for mobile devices and it's application to braking state classification on robotic mobility devices. Robotics and Autonomous Systems.

[15] Ferrer, Gonzalo. (2009). Integración Kalman de sensores inerciales INS con GPS en un UAV.

[16] Shane Colton, MIT 1997, The balance filter.

[17] Greg Welch, An Introduction to the Kalman Filter. Gary Bishop University: University of North Carolina at Chapel Hill Department of Computer Science Chapel Hill

8 APENDICE

Código para servidor en Raspberry Pi.

```
import web

urls = (
    '/', 'index'
)

class index:
    def GET(self):
        return "Hello, world!"

if __name__ == "__main__":
    app = web.application(urls, globals())
    app.run()
```

Programa para lectura de datos del servidor ubicado en la Raspberry Pi y simulación 3D de los movimientos.

```
import pygame
import urllib
from OpenGL.GL import *
from OpenGL.GLU import *
from math import radians
from pygame.locals import *

SCREEN_SIZE = (800, 600)
SCALAR = .5
SCALAR2 = 0.2

def resize(width, height):
    glViewport(0, 0, width, height)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(45.0, float(width) / height, 0.001, 10.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    gluLookAt(0.0, 1.0, -5.0,
              0.0, 0.0, 0.0,
              0.0, 1.0, 0.0)
```

```
def init():
    glEnable(GL_DEPTH_TEST)
    glClearColor(0.0, 0.0, 0.0, 0.0)
    glShadeModel(GL_SMOOTH)
    glEnable(GL_BLEND)
    glEnable(GL_POLYGON_SMOOTH)
    glHint(GL_POLYGON_SMOOTH_HINT, GL_NICEST)
    glEnable(GL_COLOR_MATERIAL)
    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)
    glLightfv(GL_LIGHT0, GL_AMBIENT, (0.3, 0.3, 0.3, 1.0));

def read_values():
    link = "http://192.168.1.65:8080" # Change this address to your settings
    f = urllib.urlopen(link)
    myfile = f.read()
    return myfile.split(" ")

def run():
    pygame.init()
    screen = pygame.display.set_mode(SCREEN_SIZE, HWSURFACE | OPENGL | DOUBLEBUF)
    resize(*SCREEN_SIZE)
    init()
    clock = pygame.time.Clock()
    cube = Cube((0.0, 0.0, 0.0), (.5, .5, .7))
    angle = 0

    while True:
        then = pygame.time.get_ticks()
        for event in pygame.event.get():
            if event.type == QUIT:
                return
            if event.type == KEYUP and event.key == K_ESCAPE:
                return

        values = read_values()
        x_angle = values[0]
        y_angle = values[1]
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

glColor((1.,1.,1.))
glLineWidth(1)
glBegin(GL_LINES)

for x in range(-20, 22, 2):
    glVertex3f(x/10.,-1,-1)
    glVertex3f(x/10.,-1,1)

for x in range(-20, 22, 2):
    glVertex3f(x/10.,-1, 1)
    glVertex3f(x/10., 1, 1)

for z in range(-10, 12, 2):
    glVertex3f(-2, -1, z/10.)
    glVertex3f( 2, -1, z/10.)

for z in range(-10, 12, 2):
    glVertex3f(-2, -1, z/10.)
    glVertex3f(-2,  1, z/10.)

for z in range(-10, 12, 2):
    glVertex3f( 2, -1, z/10.)
    glVertex3f( 2,  1, z/10.)

for y in range(-10, 12, 2):
    glVertex3f(-2, y/10., 1)
    glVertex3f( 2, y/10., 1)

for y in range(-10, 12, 2):
    glVertex3f(-2, y/10., 1)
    glVertex3f(-2, y/10., -1)

for y in range(-10, 12, 2):
    glVertex3f(2, y/10., 1)
    glVertex3f(2, y/10., -1)

glEnd()
glPushMatrix()
```

```
        glRotate(float(x_angle), 1, 0, 0)
        glRotate(-float(y_angle), 0, 0, 1)
        cube.render()
        glPopMatrix()
        pygame.display.flip()

class Cube(object):

    def __init__(self, position, color):
        self.position = position
        self.color = color

    # Cube information
    num_faces = 6

    vertices = [ (-1.0, -0.05, 0.5),
                 (1.0, -0.05, 0.5),
                 (1.0, 0.05, 0.5),
                 (-1.0, 0.05, 0.5),
                 (-1.0, -0.05, -0.5),
                 (1.0, -0.05, -0.5),
                 (1.0, 0.05, -0.5),
                 (-1.0, 0.05, -0.5) ]

    normals = [ (0.0, 0.0, +1.0),  # front
                (0.0, 0.0, -1.0),  # back
                (+1.0, 0.0, 0.0),  # right
                (-1.0, 0.0, 0.0),  # left
                (0.0, +1.0, 0.0),  # top
                (0.0, -1.0, 0.0) ]  # bottom

    vertex_indices = [ (0, 1, 2, 3),  # front
                       (4, 5, 6, 7),  # back
                       (1, 5, 6, 2),  # right
                       (0, 4, 7, 3),  # left
                       (3, 2, 6, 7),  # top
                       (0, 1, 5, 4) ]  # bottom

    def render(self):
        then = pygame.time.get_ticks()
```

```
        glColor(self.color)

        vertices = self.vertices

        # Draw all 6 faces of the cube
        glBegin(GL_QUADS)

        for face_no in xrange(self.num_faces):
            glNormal3dv(self.normals[face_no])
            v1, v2, v3, v4 = self.vertex_indices[face_no]
            glVertex(vertices[v1])
            glVertex(vertices[v2])
            glVertex(vertices[v3])
            glVertex(vertices[v4])
        glEnd()

if __name__ == "__main__":
    run()
```