Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Implementación de un EKF sobre plataformas Android para posicionamiento de dispositivos portables en espacios interiores a partir de medidas de sensores inerciales

Autor: Rubén Cervigón Rey

Tutor: Mª del Carmen Pérez Rubio

Cotutor: David Gualda Gómez

2017

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Implementación de un EKF sobre plataformas Android para posicionamiento de dispositivos portables en espacios interiores a partir de medidas de sensores inerciales

Autor: Rubén Cervigón Rey

Directores: Mª del Carmen Pérez Rubio

David Gualda Gómez

Tribunal:

Presidente: D. Jesús Ureña Ureña

Vocal 1º: Dª. Cristina Losada Gutiérrez

Vocal 2º: Dª. Mª del Carmen Pérez Rubio

CALIFICACIÓN: .....

FECHA: .....

A mis padres y hermana. . .

. . . en especial por ti papá, para que donde quiera que estés, te sientas orgulloso de mi.

"La muerte no existe, la gente sólo muere cuando la olvidan"
Isabel Allende

Agradecimientos

A todos los que la presente vieren y entendieren.

Inicio de las Leyes Orgánicas. Juan Carlos I

Para empezar, me gustaría agradecer a mi tutora, Mª del Carmen Pérez Rubio, por haberme proporcionado la oportunidad de realizar este trabajo dentro de un grupo de investigación magnífico y del que salgo totalmente satisfecho por el trabajo realizado y todo lo aprendido. Gracias a Mª del Carmen y a mi cotutor, David Gualda Gómez, por la forma en la que me han ido guiando, ayudando a resolver todo tipo de dudas y aportarme sugerencias a lo largo de la elaboración de este trabajo.

En segundo lugar, agradecer a todos los integrantes del grupo de investigación GEINTRA y en concreto al grupo US&RF, por estos seis meses increíbles en los que he adquirido un montón de cosas de cada uno de ellos. Quiero hacer una mención especial a Edel Díaz y Sergio Matas, por su ayuda en este trabajo y su inestimable colaboración a lo largo del desarrollo del mismo.

Por último, que no por eso menos importante, dar las gracias tanto a mi familia como a mis amigos que han estado conmigo durante estos intensos cuatro años, compartiendo buenos momentos y sobre todo ayudándome a superar aquellos no tan buenos. Sin vosotros todo lo conseguido no hubiera sido posible.

Resumen

Este Trabajo Fin de Grado desarrolla un algoritmo de posicionamiento relativo (PDR) implementado sobre una plataforma Android usando la unidad de medición inercial (IMU) de un dispositivo Shimmer situado a la altura del bolsillo del pantalón del usuario. Las señales resultantes del algoritmo PDR son usadas para la detección de la posición del individuo. La estimación PDR se ha llevado a cabo mediante un Filtro de Kalman Extendido (EKF) con la intención de fusionar la información de acelerómetros y giróscopos ofrecida por la IMU y obtener los ángulos de orientación del sensor (*Roll*, *Pitch* y *Yaw*). Usando esta información se puede estimar la trayectoria descrita por el usuario en diferentes entornos de trabajo como se demuestra con los resultados experimentales obtenidos.

Palabras Clave: Posicionamiento; PDR; IMU; EKF; Android.

Abstract

This study presents a relative positioning algorithm (PDR) developed on an Android-based platform using the Inertial Measurement Sensor (IMU) of a Shimmer device located on the pocket of a user. The signals obtained by the PDR algorithm are used for detecting the position of a person. PDR estimation has been carried out by an Extended Kalman Filter (EKF) algorithm in order to fuse the gyroscopes and accelerometers information provided by the IMU and obtain the values of Roll, Pitch and Yaw. Using this information, we can estimate the trajectory described by the pedestrian in different environments as it is demonstrated in the experimental results.

Keywords: Positioning; PDR; IMU; EKF; Android.

Índice

Índice de figuras

Índice de tablas

Glosario de acrónimos y símbolos

BT	Bluetooth.
EKF	Filtro de Kalman Extendido.
GNSS	Sistemas de Navegación Global por Satélite.
GPS	Sistema de Posicionamiento Global.
IMU	Unidad de Medición Inercial.
KF	Filtro de Kalman.
LBS	Servicios Basados en Localización.
LPS	Sistemas de Posicionamiento Local.
MEMS	Sensores Micro ElectroMecánicos.
PDR	*Pedestrian Dead Reckoning.*
RF	Radio-Frecuencia.
SHSs	*Step and Heading Systems*.
SL	*Step Length.*
TDOA	*Time Difference Of Arrival.*
U-LPS	Sistemas de Posicionamiento Local Ultrasónico.
US	Señales Ultrasónicas.

$\Delta\odot$	Amplitud de *Pitch*.
ϕ	Ángulo *Roll*.
$\odot$	Ángulo *Pitch*.
Ψ	Ángulo *Yaw*.

$\widehat{\Phi}$	Estimación ángulo *Roll*.
$\widehat{\Theta}$	Estimación ángulo *Pitch*.
$\widehat{\Psi}$	Estimación ángulo *Yaw*.
α	Valores medidos con los acelerómetros de la IMU.
ω	Valores medidos con los giróscopos de la IMU.

1. Introducción

a. Contexto y objetivos del Trabajo Fin de Grado

El enorme potencial de desarrollo de servicios basados en localización (LBS) en dispositivos portables, tipo teléfonos inteligentes o tabletas, ha supuesto una revolución e impulso para la investigación en sistemas de posicionamiento y navegación de personas. En espacios exteriores, los Sistemas de Navegación Global por Satélite (GNSS), como el GPS, proporcionan datos de posición con precisiones del orden de 10m. No obstante, estos sistemas no son adecuados para aplicaciones interiores en las que se demanden precisiones del orden de centímetros, no existiendo una tecnología alternativa desarrollada al mismo nivel para resolver el posicionamiento [Lopes, 2014].

Dado que gran parte de nuestro tiempo lo pasamos precisamente en espacios interiores, resultan comprensibles los esfuerzos actuales en el desarrollo de tecnologías de localización precisa (centimétrica) de dispositivos portables dada la variedad de aplicaciones que se podrían ofrecer a los usuarios de dichos dispositivos: navegación guiada en interiores de edificios (aeropuertos, hospitales, fábricas…), aplicaciones de realidad aumentada para turismo cultural, juegos, estudio de puntos de interés y patrones de movimiento con objeto de mejorar los servicios o inserción de publicidad personalizada [Filonenko, 2013].

En este sentido, el proyecto LOCATE-US, *"Sistema de localización basado en señales ultrasónicas codificadas para posicionamiento de dispositivos móviles en interiores*" ref. CCG2016/EXP-078, en el que se enmarca el presente Trabajo Fin de Grado (TFG), plantea la propuesta y desarrollo de nuevas técnicas de codificación y fusión sensorial como línea de avance para la mejora de Sistemas de Posicionamiento Locales Ultrasónicos (U-LPSs), para conseguir una localización en entornos interiores extensos de dispositivos portables, donde la precisión pueda modularse en función de las necesidades de cada zona o espacio concreto.

Este proyecto tiene dos líneas de trabajo prioritarias. La primera, ya desarrollada, propone la localización centimétrica de dispositivos portables en entornos interiores extensos mediante el procesamiento de señales ultrasónicas (US) emitidas por un conjunto de sistemas U-LPSs ubicados en el entorno [Pérez, 2016]. Con respecto a trabajos anteriores, destacar las ventajas que se consiguen con esta propuesta, como la utilización únicamente de señales ultrasónicas frente a trabajos basados en señales de radio frecuencia (RF), que consiguen menores precisiones. Otros trabajos, como [Aguilera, 2013], también utilizan señales ultrasónicas, pero en la banda de 18-22kHz, lo que implica molestias debidas a artefactos audibles y una menor calidad de la señal recibida por el bajo ancho de banda. En [Pérez, 2016] los U-LPS emiten a 41kHz, y gracias al uso de una tarjeta de adaptación diseñada en un proyecto previo (US-PHONE, CCG2014/EXP-077) se captura la señal ultrasónica emitida, se digitaliza y se envía al dispositivo móvil vía USB para su procesamiento.

Aunque en aplicaciones de robótica móvil se demandan precisiones centimétricas, para la localización de personas no es necesario mantener una precisión tan alta en todo el recorrido sino solo en las zonas de interés; con lo que en determinadas zonas se podrían relajar las exigencias minimizando el número de balizas (y por tanto los costes). En esa idea va dirigida la segunda línea de trabajo del proyecto (en la que se enmarca el presente Trabajo Fin de Grado), que aborda la navegación en las zonas en las que no exista cobertura de los U-LPS mediante sensores inerciales y, mediante algoritmos de fusión sensorial [Gualda, 2014], corregir la trayectoria con la información de algunas balizas simples o resetear el error acumulativo de los sensores inerciales al entrar de nuevo en el área de influencia de un U-LPS (véase la Figura 1).

El resultado esperado final del proyecto LOCATE-US será una aplicación sobre plataforma Android que integre dos procesos en paralelo en el dispositivo móvil (véase la Figura 2). Por un lado, el procesamiento de las señales ultrasónicas (captadas con la tarjeta de adquisición, que está conectada al móvil vía USB), que requiere la correlación con los códigos emitidos por las balizas y el cálculo de diferencias de tiempos de vuelo (TDOA, *Time Difference Of Arrival*); y simultáneamente, por otro lado, el procesamiento de las señales obtenidas con los sensores inerciales de la IMU (enviadas al móvil vía Bluetooth), que requiere un

Filtro de Kalman Extendido (EKF), y un algoritmo de detección de pasos, de estimación de la anchura del paso y de posición relativa.

El posicionamiento se conseguirá mediante la fusión de las TDOA con la posición relativa obtenida a partir de las medidas de los sensores inerciales, exceptuando cada primera vez que se llegue a la zona de cobertura de los U-LPS, donde el posicionamiento se conseguirá a partir de la aplicación del algoritmo Gauss Newton.

A continuación, se muestra en la Figura 1 una ilustración del objetivo global del proyecto LOCATE-US y del objetivo local del TFG (encuadrado en rojo) y en la Figura 2 el diagrama de bloques del proyecto y TFG (encuadrado en rojo).

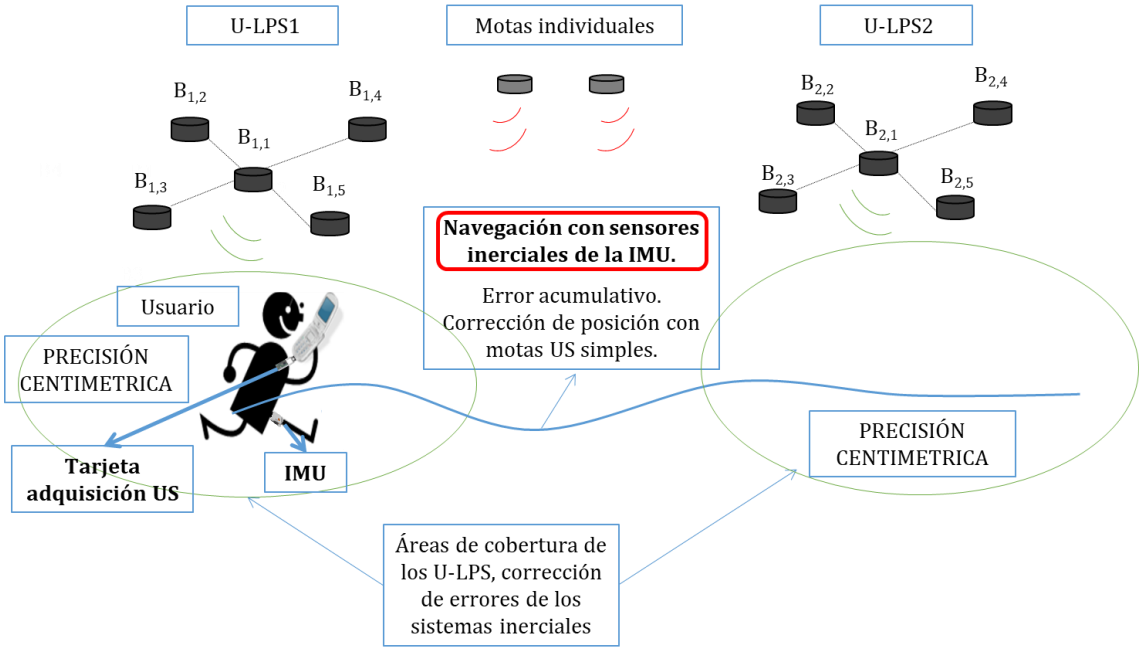

Figura 1: Propuesta general del proyecto LOCATE-US y objetivo del TFG enmarcado en rojo.

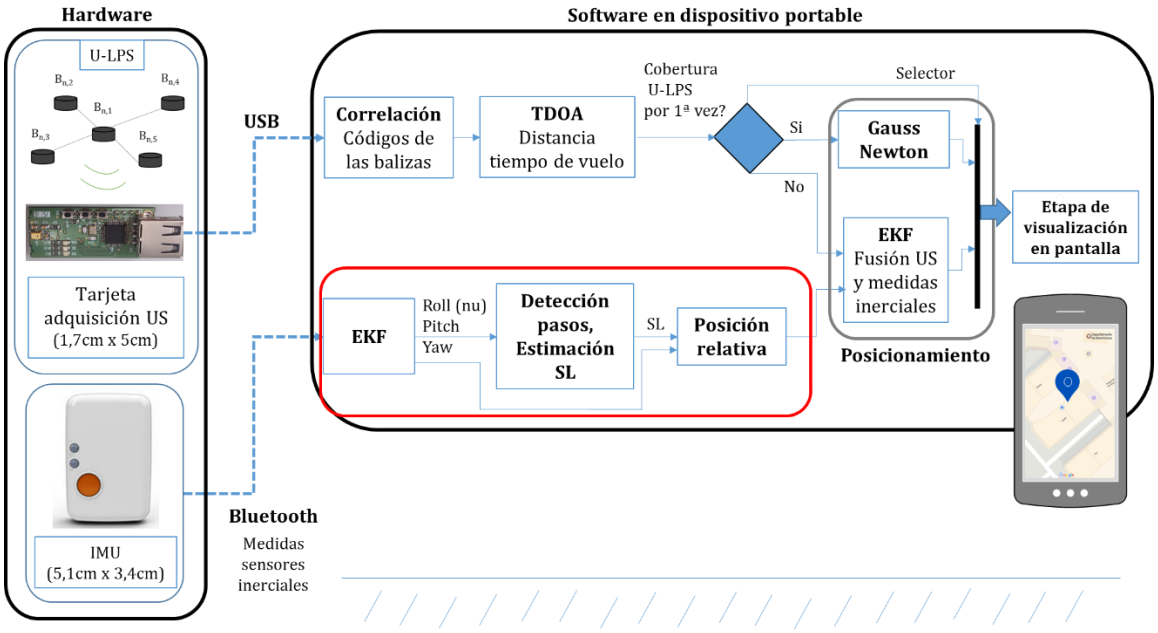

Figura 2: Diagrama de bloques del proyecto LOCATE-US, con recuadro en rojo indicando las tareas del presente TFG.

Como se observa en las figuras expuestas, la realización de este TFG se centra en conseguir un posicionamiento del móvil a partir de las medidas de los sensores inerciales de la IMU (enviadas al dispositivo vía Bluetooth) en las zonas en las que no exista cobertura de los U-LPS.

Para ello se estudia la forma de implementar un Filtro de Kalman Extendido (EKF) sobre plataformas Android, para conseguir un posicionamiento de dispositivos móviles en espacios interiores mediante las medidas de sensores inerciales, en una app que pueda estar incluida en cualquier dispositivo Android y que permita visualizar la trayectoria realizada por el usuario. Será fundamental que los algoritmos desarrollados sean lo más eficientes posibles, para que el procesamiento pueda hacerse en el dispositivo portable en tiempo real. Esta implementación sobre plataforma Android es uno de los incentivos de este trabajo frente a estudios anteriores implementados sobre otras plataformas, como Matlab [Munoz, 2016] debido al amplio abanico de opciones que se abren al tener el algoritmo en funcionamiento en un dispositivo móvil a tiempo real.

Por último, destacar que como se puede observar en la Figura 2, el filtro de Kalman que se va a desarrollar en este TFG se particularizará y usará en trabajos futuros para poder realizar la fusión de las señales ultrasónicas con las medidas de los

sensores inerciales de la IMU; pretendiendo, por un lado, mejorar la posición obtenida con las señales ultrasónicas, y por otro reducir los errores en aquellos puntos de baja cobertura de los U-LPS.

b. Revisión de sistemas de navegación basados en sensores inerciales

Los sistemas de navegación basados en sensores inerciales usan PDRs (*Pedestrian Dead Reckoning*, métodos para la estimación de la posición y orientación) para estimar la trayectoria del usuario. Principalmente destacan dos tipos de PDR, el algoritmo *strap-down* y el basado en los pasos y la orientación (SHSs) [Harle, 2013]. El algoritmo *strap-down* consiste en la doble integración de la aceleración respecto al tiempo para la obtención de la posición. Los acelerómetros proporcionan una medida debida a la suma del movimiento propio de la persona, la gravedad, la fuerza centrípeta, la fuerza de Euler o la fuerza de Coriolis, siendo estas tres últimas indiferentes para un método PDR ya que la velocidad del paso no cambia significativamente la longitud y la latitud. Por tanto, para obtener la aceleración se ha de eliminar la componente proveniente de la gravitación, lo que se consigue al saber la orientación del sensor, obtenida a partir de la información de los giróscopos. El problema de este método es que la estimación errónea de la orientación supone un error acumulativo en la posición debido a la gravitación no eliminada; por lo que este método solo será factible para sensores micro electromecánicos (MEMS) en combinación con correcciones como la actualización a velocidad cero (ZUPT, *zero velocity update*) [Jiménez, 2009], método que solo se puede utilizar con la IMU en el pie, ya que es el único lugar en el que se obtiene velocidad igual a cero durante el paso (al contactar con el suelo). Este TFG utiliza el método basado en la detección de pasos y la orientación (SHSs), el cual estima secuencialmente la posición del individuo basándose en la posición previa de cada paso detectado, la anchura del mismo y los ángulos de orientación del individuo. Esta forma permite que la IMU se pueda colocar en cualquier localización y consiste principalmente en la estimación de la orientación (gracias a los ángulos de Euler, ángulo de alabeo (*Roll* (ϕ)), elevación (*Pitch* ($\odot$)) y dirección o guiñada (*Yaw* (Ψ))) y de la anchura de paso. A partir de ahora se utilizará la nomenclatura anglosajona (*Roll*, *Pitch* y *Yaw*) para

hacer referencia a los ángulos de Euler, ya que es la que se utiliza normalmente en la literatura de estos temas.

La orientación del sensor e individuo es la misma si el sensor se encuentra sujeto al cuerpo del individuo. Hay diferentes enfoques en la literatura para estimar la orientación del sensor; pueden clasificarse en deterministas (asumen todas las condiciones iguales) y probabilísticos (asumen condiciones iniciales, son los de mayor utilización en la actualidad [Munoz, 2015]). En este TFG, se seguirá la alternativa probabilística, introduciendo un filtro de Kalman (KF), para a partir de las medidas de los sensores inerciales (acelerómetros y giróscopos), extraer las señales de los ángulos de Euler.

Para la detección de pasos una posibilidad es identificar cambios en el desplazamiento vertical de la pelvis, siempre y cuando el sensor este colocado en el centro de masa del individuo, por ejemplo, en el cinturón [Goyal, 2011]. Para otras localizaciones del sensor que no están tan cerca del centro de masa como en la pierna o la mano, no es conveniente utilizar ese método. Otros algoritmos de detección de pasos de la literatura utilizan la aceleración, específicamente los picos que se dan en la misma a consecuencia del impacto del pie con el suelo. Se necesita procesar esas señales de aceleración en los tres ejes y establecer umbrales que eviten detecciones de picos de amplitudes pequeñas no debidas a pasos. La desventaja de este enfoque es que en la aceleración el patrón que siguen las medidas está muy influenciado por la velocidad del individuo, lo que puede ser una complicación a la hora de establecer esos umbrales. Otra alternativa es la de la detección de pasos a partir del Pitch, ya sea mediante sus cruces por cero [Lo, 2011] o mediante la detección de sus máximos [Munoz, 2014]. En este TFG, aunque en la literatura el algoritmo más usado para la detección de pasos es con la aceleración, se usarán los máximos de la señal del ángulo Pitch para un sensor situado a la altura del bolsillo lateral del pantalón, debido a que en esta ubicación los movimientos de individuo y sensor están fuertemente relacionados.

Por tanto, para la estimación de la anchura del paso, se estudiarán aquellos algoritmos desarrollados que puedan usarse con el sensor en la posición utilizada.

Para estimar la anchura del paso (*step length* (SL)), el método más simple es utilizar una relación lineal entre la SL y la altura del individuo (h) [Zhou, 2016]:

$$SL = k * h \tag{1.1}$$

Siendo k una constante dependiente del sexo del individuo. Este método tiene el gran inconveniente de no depender de la velocidad de paso del individuo.

Otro método más fiable es usar la relación no lineal entre la SL y la aceleración en el eje z de los acelerómetros, donde destacan los algoritmos propuestos por [Weinberg, 2002] y [Scarlett, 2007], que se muestran a continuación en (1.2) y (1.3) respectivamente.

$$SL = k * \sqrt[4]{a_{max} - a_{min}} \tag{1.2}$$

$$SL = k * \frac{\frac{\sum_{i=1}^{N} |ai|}{N} - a_{min}}{a_{max} - a_{min}} \tag{1.3}$$

Siendo k una constante, N el número de valores de la aceleración en un ciclo y a_{max} y a_{min} el máximo y mínimo valor de la aceleración medida en el eje z del fragmento de tiempos correspondientes entre cada paso.

El algoritmo de [Weinberg, 2002] es el que menos error produce tras diferentes pruebas experimentales [Zhou, 2016]. Dicho algoritmo necesita una calibración previa para determinar la k y ofrece la ventaja de no necesitar la altura de la persona (h), parámetro que puede suponer muchos errores. Para este TFG se usará un algoritmo basado en la relación entre el ángulo *Pitch* y la anchura de paso [Munoz, 2015], que permite dinámicamente estimar la anchura de paso de la persona a partir de la amplitud de pitch ($\odot_{max} - \odot_{min}$) tomada en el periodo entre el paso actual y el anterior.

Para acabar con el estado del arte, destacar que los errores de posicionamiento suelen estar por debajo del 5% de la distancia total recorrida. La mayor parte de ese error se genera con la estimación de la orientación.

En [Jiménez, 2009] se presentan una serie de algoritmos de posicionamiento relativo junto con sus pruebas experimentales, demostrando que los más precisos

son aquellos que usan tanto la anchura de paso como la orientación de la IMU a la hora de posicionar.

En esa línea se centra este TFG, que utilizará un algoritmo de posición relativa en función de la anchura de paso y la señal *Yaw* obtenida a partir de los valores recibidos de la IMU, que representa el movimiento lateral de la pierna en la que se sitúa el sensor.

c. Estructura del documento

Este documento está dividido en 10 capítulos, organizados como se explica a continuación:

El capítulo 2 expone la base teórica sobre la que se ha basado este trabajo (EKF, algoritmo de posicionamiento, etc); Las principales características del material utilizado se describen en el capítulo 3; El capítulo 4 explica todo lo relacionado con la programación de la aplicación Android desarrollada; En el capítulo 5 se evalúa el algoritmo propuesto con diferentes pruebas experimentales; Finalmente para acabar con la parte de la memoria del proyecto se presentan, en el Capítulo 6, las conclusiones obtenidas con la realización del mismo y los trabajos futuros que quedarían por realizar para la finalización del proyecto.

En los últimos capítulos, se describen otras cuestiones del proyecto como la serie de condiciones para el funcionamiento correcto de la aplicación (Capítulo 7: Pliego de condiciones), coste (Capítulo 8: Presupuesto), manual de usuario de la app (Capítulo 9) y la bibliografía (Capítulo 10).

2. Base teórica: Descripción del Filtro de Kalman y del algoritmo de posicionamiento

a. Descripción Filtro de Kalman y Filtro de Kalman Extendido

En 1960, Rudolf Emil Kalman publicó su famoso escrito describiendo una solución recursiva al problema de filtrado lineal de datos discretos [Kalman, 1960]. Desde entonces, el filtro de Kalman ha sido objeto de una extensa investigación y debido en gran parte a los avances en la informática digital, ha tenido gran implantación en el ámbito de las ayudas autónomas o navegación.

Una gran serie de problemas tanto teóricos como prácticos de la comunicación y el control son de naturaleza estadística, como la predicción de señales aleatorias, la separación de las mismas de ruido aleatorio o la detección de señales de forma conocida, como pulsos o sinusoides, en presencia de señales ruidosas. De ahí la importante función del filtro de Kalman, estimador estocástico dinámico, que a diferencia de los deterministas consigue un funcionamiento correcto también cuando hay ruido. Esto es de gran importancia ya que las perturbaciones a las que está sujeto un sistema de control automático provocan que su salida difiera del comportamiento ideal, por ello usar sistemas que permitan contemplar medidas satisfactorias, incluso en la presencia de este tipo de señales de naturaleza aleatoria, es fundamental.

Resumiendo, el filtro de Kalman es un conjunto de ecuaciones matemáticas que proporciona un algoritmo computacional eficiente que asume un sistema dinámico lineal y un observable corrompido con ruido gaussiano y permite estimar el estado de un proceso, la variable desconocida x_k, a partir de las señales conocidas, u_k y z_k, entrada y observable respectivamente, minimizando la matriz de covarianza de estimación del error. En un sistema con ruido, hay incertidumbre en las medidas, pero con el filtro de Kalman se consigue la mejor estimación posible. Cabe destacar que dicho filtro permite las estimaciones tanto de estados presentes como de pasados y futuros.

Como se ha expresado anteriormente, muchas de las aplicaciones más importantes del filtro de Kalman son para sistemas de navegación, donde tanto las mediciones

como los sistemas suelen ser no lineales. Es ahí donde se utiliza el Filtro de Kalman Extendido o también conocido por sus siglas como EKF [Cox, 1964].

El EKF es una ampliación del filtro de Kalman para sistemas no lineales usando una aproximación de Taylor, que linealiza un sistema no lineal respecto a la estimación actual. Es decir, el algoritmo del EKF describe los mismos pasos recursivos que el filtro de Kalman lineal, predicción y corrección, con la particularidad de que se realiza la linealización de Taylor. En cada estimación del estado, se calcula una linealización del sistema, por lo que el modelo del sistema cambia en cada instante. Se predice la estimación del estado actual en el tiempo a partir del estado anterior y las ecuaciones dinámicas, y a continuación se ajusta o corrige la predicción mediante una observación del estado actual y así sucesivamente, como representa la Figura 3 expuesta a continuación.

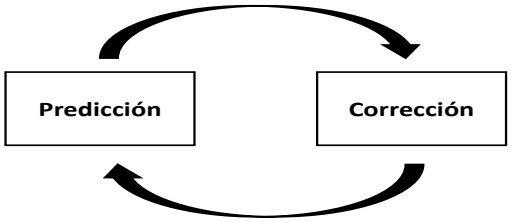

Figura 3: Ciclo del filtro de Kalman.

Una vez hecho un breve resumen introductorio al filtro de Kalman y al EKF, se procede a explicar las ecuaciones del Filtro de Kalman Extendido, las cuales serán las que se implementarán posteriormente, primero en Matlab y a continuación en Android.

Un sistema no lineal generalizado se puede describir de la siguiente forma:

$$\boldsymbol{x_k} = \boldsymbol{f}(\boldsymbol{x_{k-1}}) + \boldsymbol{w_k} \tag{2.1}$$

$$\boldsymbol{z_k} = \boldsymbol{h}(\boldsymbol{x_k}) + \boldsymbol{v_k} \tag{2.2}$$

Siendo $\boldsymbol{x_k}$ el vector de estado; $\boldsymbol{z_k}$ la observación; $f(x_{k-1})$ y $h(x_k)$ funciones dependientes del estado anterior, a particularizar para cada uso concreto del EKF; w_k y v_k son las perturbaciones del proceso y de la medida respectivamente. Los ruidos, w_k y v_k, son blancos (es decir que siguen una distribución gaussiana con media 0 y varianza o matriz de covarianza finita, $\boldsymbol{Q}$ y $\boldsymbol{R}$ respectivamente) e incorrelados (no están vinculados unos con otros).

$$\boldsymbol{w_k} \sim \boldsymbol{N(0, Q_k)} \tag{2.3}$$

$$\boldsymbol{v_k} \sim \boldsymbol{N(0, R_k)} \tag{2.4}$$

La incertidumbre que introducen los ruidos dependerá de la campana gaussiana, es decir de la desviación típica (la raíz cuadrada de la varianza). A mayor varianza, mayor incertidumbre.

El primer paso a la hora de implementar el EKF es realizar la etapa de predicción, que consiste en calcular la estimación a priori del estado y la matriz de covarianza de error a priori.

$$\boldsymbol{x_k}^- = \boldsymbol{f(x_{k-1})} \tag{2.5}$$

$$\boldsymbol{P_k}^- = \boldsymbol{A} * \boldsymbol{P_{k-1}} * \boldsymbol{A^T} + \boldsymbol{Q} \tag{2.6}$$

Siendo:

- $\boldsymbol{x_k}^-$: estado a priori, se calcula a partir de la estimación del estado anterior a posteriori.
- $\boldsymbol{P_k}^-$: matriz covarianza del error a priori, dependiente de la matriz de covarianza del error del estado anterior a posteriori, $\boldsymbol{P_{k-1}}$; la varianza del ruido del proceso, $\boldsymbol{Q}$; y la matriz $\boldsymbol{A}$, que es el jacobiano de la ecuación transición y sirve para linealizar el sistema.

$$\boldsymbol{A} = \frac{\boldsymbol{dx_k}^-}{\boldsymbol{dx_k}} \tag{2.7}$$

El segundo paso es la etapa de corrección, de estimación final, en la cual se corrige la estimación prevista en la etapa de predicción mediante una observación del estado actual.

$$\boldsymbol{S} = \boldsymbol{H} * \boldsymbol{P_k}^- * \boldsymbol{H^T} + \boldsymbol{R} \tag{2.8}$$

$$\boldsymbol{K} = \boldsymbol{P_k}^- * \boldsymbol{H^T} * \boldsymbol{S^{-1}} \tag{2.9}$$

$$\boldsymbol{V} = \boldsymbol{z_k} - \boldsymbol{h(x_k}^-) \tag{2.10}$$

$$\boldsymbol{x_k} = \boldsymbol{x_k}^- + \boldsymbol{K} * \boldsymbol{V} \tag{2.11}$$

$$\boldsymbol{P_k} = \boldsymbol{P_k}^- - \boldsymbol{K} * \boldsymbol{S} * \boldsymbol{K^T} \tag{2.12}$$

Teniendo:

- $\boldsymbol{S}$: matriz intermedia, dependiente de la matriz $\boldsymbol{H}$, el jacobiano de la ecuación de medida; la matriz covarianza del error a priori, $\boldsymbol{P_k}^-$; y la varianza del ruido de la medida, $\boldsymbol{R}$.

$$\boldsymbol{H} = \frac{\boldsymbol{dh}(\boldsymbol{x_k}^-)}{\boldsymbol{dx_k}^-} \tag{2.13}$$

- $\boldsymbol{K}$: ganancia de Kalman, dependiente de la matriz covarianza del error a priori, $\boldsymbol{P_k}^-$; la matriz $\boldsymbol{H}$; y la matriz intermedia $\boldsymbol{S}$.
- $\boldsymbol{h}(\boldsymbol{x_k}^-)$: estimación de las observaciones a partir del estado a priori $\boldsymbol{x_k}$.
- $\boldsymbol{V}$: matriz intermedia de innovación, depende de la observación, $\boldsymbol{z_k}$ y de $\boldsymbol{h}(\boldsymbol{x_k}^-)$.
- $\boldsymbol{x_k}$: estado a posteriori. Depende del estado a priori, de la ganancia de Kalman y la matriz de innovación. La ganancia de Kalman es la que impone la importancia que tiene sobre el sistema la predicción o la actualización.
- $\boldsymbol{P_k}$: matriz covarianza del error a posteriori, calculada a partir de la matriz covarianza del error a priori, $\boldsymbol{P_k}^-$; la ganancia de Kalman, $\boldsymbol{K}$; y la matriz $\boldsymbol{S}$.

Las variables devueltas para procesos posteriores son la estimación final tanto del estado como de la covarianza de error, $\boldsymbol{x_k}$ y $\boldsymbol{P_k}$, respectivamente, según se muestra en la Figura 4.

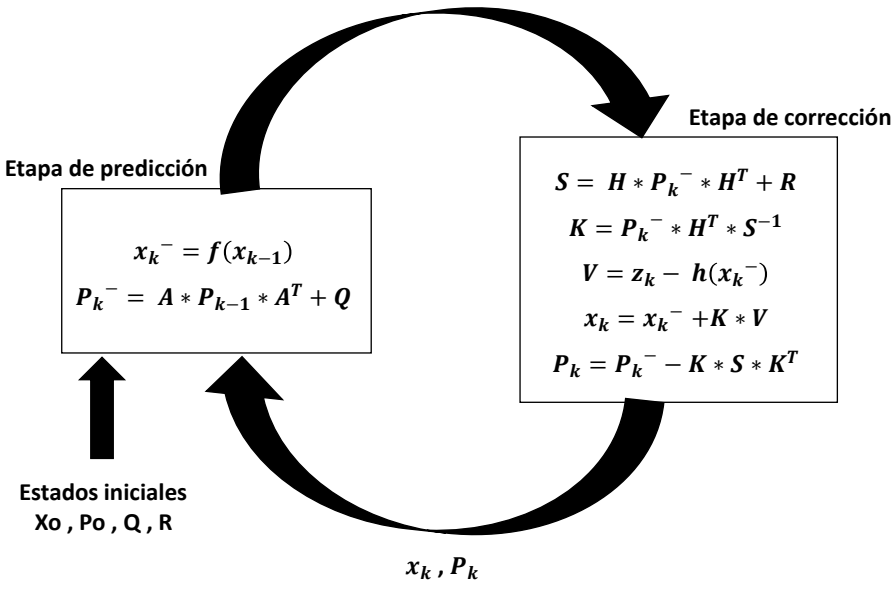

Figura 4: Ciclo y ecuaciones del filtro de Kalman Extendido.

b. Particularización del Filtro de Kalman para la estimación de la orientación

Una vez expuesta una pequeña introducción sobre el EKF, procedemos a particularizarlo para nuestro caso en concreto. El EKF lo usamos para a partir de los valores medidos del Shimmer (IMU usada), acelerómetros (α en unidades de m/s^2) y giróscopos (ω en unidades de º/s) de los tres ejes, obtener una estimación de la orientación del Shimmer (colocado en la parte superior de la pierna, a la altura de los bolsillos del pantalón), de los ángulos de Euler (*Roll* (ϕ), *Pitch* (⊙) y *Yaw* (Ψ)), como se muestra a continuación en la Figura 5.

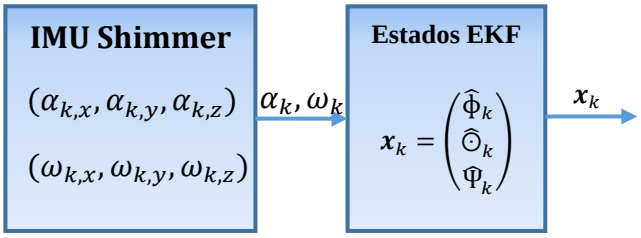

Figura 5: Diagrama de bloques de la estimación de los ángulos de Euler basado en un EKF.

Siendo k el número de iteración del ciclo y $\boldsymbol{x}$ el vector de estados, que será un vector columna de dimensiones 3x1.

A continuación, se muestra en la Figura 6 una imagen de la IMU usada (Shimmer3) junto con la representación de sus ángulos de Euler.

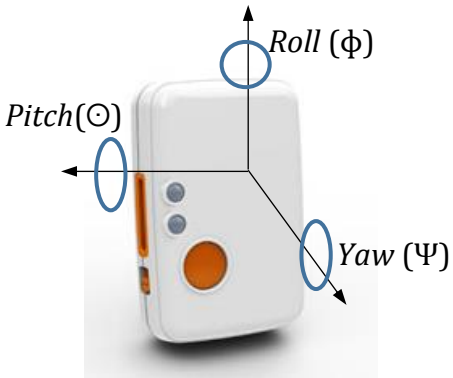

Figura 6: Representación de los ángulos de Euler del Shimmer.

Al calcular la orientación del sensor usamos un método probabilístico por lo que tenemos que introducir unas condiciones iniciales.

La estimación de los ángulos de Euler en la primera iteración ($\widehat{\Phi}_0, \widehat{\Theta}_0, \widehat{\Psi}_0$) se obtiene a partir de las siguientes expresiones:

$$\widehat{\Phi}_0 = \tan\left(\frac{\alpha_{0,y}}{\alpha_{0,z}}\right)^{-1} \tag{2.14}$$

$$\widehat{\Theta}_0 = \tan\left(\frac{-\alpha_{0,x}}{\sqrt{{\alpha_{0,y}}^2 + {\alpha_{0,z}}^2}}\right)^{-1} \tag{2.15}$$

$$\widehat{\Psi}_0 = 0 \tag{2.16}$$

Para el resto de iteraciones el vector de estado se actualizará de acuerdo al EKF expuesto en la Figura 4; es decir, siguiendo una etapa de predicción realizada a partir de la estimación anterior y la información de los giróscopos primero y de corrección después.

La etapa de corrección no se realiza en todas las iteraciones, solo cuando la norma de la aceleración ($|\alpha| = \sqrt{{\alpha_{k,x}}^2 + {\alpha_{k,y}}^2 + {\alpha_{k,z}}^2}$) es aproximadamente 9.8 m/s^2. En esa situación (al estar el pie de la pierna en la que se lleva la IMU en contacto con el suelo), el individuo no está en movimiento por lo que es el momento adecuado para corregir el estado del *Roll* y *Pitch* (el *Yaw* no lo corregimos, del *Yaw* solo interesa observar las variaciones entre iteraciones para calcular la posición relativa, no su valor exacto). Las iteraciones en las cuales no se corrige, se actualiza el vector de estado con el vector de estado a priori calculado en la etapa de predicción.

Por último, resaltar que al estar tratando con multiplicaciones de matrices será fundamental introducir correctamente las dimensiones de las mismas. Es por ello que se expone a continuación el tamaño estándar de las matrices del algoritmo.

$[\boldsymbol{x_k}^-]$ = *mx1.*	$[\boldsymbol{P_k}^-]$ = *mxm.*	$[\boldsymbol{A}]$ = *mxm.*
$[\boldsymbol{Q}]$ = *mxm.*	$[\boldsymbol{S}]$ = *nxn.*	$[\boldsymbol{H}]$ = *nxm.*
$[\boldsymbol{R}]$ = *nxn.*	$[\boldsymbol{K}]$ = *mxn.*	$[\boldsymbol{V}]$ = *nx1.*
$[\boldsymbol{h}(\boldsymbol{x_k}^-)]$ = *nx1.*	$[\boldsymbol{z_k}]$ = *nx1.*	$[\boldsymbol{x_k}]$ = *mx1.*
$[\boldsymbol{P_k}]$ = *mxm.*		

Donde *m* indica el número de estados estimados y *n* el número de estados corregidos. Para el algoritmo desarrollado como tenemos tres estados (*Roll, Pitch y Yaw*) y solo queremos corregir dos de ellos (*Roll y Pitch*), los valores de *m* y *n* son 3 y 2 respectivamente.

Cabe destacar que en el algoritmo desarrollado no todas las matrices son dinámicas y calculadas a partir de las ecuaciones expuestas en la sección anterior, hay algunas matrices fijas, ya sea porque no cambian en cada iteración o por aproximaciones para ganar tiempo de computación. Las matrices fijas del algoritmo y el valor elegido para ellas se expresan a continuación:

- La matriz $\boldsymbol{A}$, jacobiano de la ecuación transición, sigue la ecuación (2.7) y es aproximadamente igual a una matriz identidad de tercer orden.

$$\mathbf{A} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \tag{2.17}$$

- La matriz $\boldsymbol{Q}$, que representa el valor de la varianza del ruido del proceso, de las varianzas de los componentes del giroscopio, cuyo valor obtenido experimentalmente es $\sigma_\omega^2 = 0.001^2$.

$$\mathbf{Q} = \begin{bmatrix} \boldsymbol{\sigma_\omega^2} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\sigma_\omega^2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\sigma_\omega^2} \end{bmatrix} \tag{2.18}$$

- La matriz $\boldsymbol{H}$, jacobiano de la ecuación de medida, sigue la ecuación (2.13) y es aproximadamente:

$$\mathbf{H} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix} \tag{2.19}$$

- La matriz $\boldsymbol{R}$, que representa el valor de la varianza del ruido de la medida, su diagonal principal representa la varianza de la medida, cuyo valor obtenido experimentalmente es como se muestra a continuación.

$$\mathbf{R} = \begin{bmatrix} \mathbf{0.01^2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0.01^2} \end{bmatrix} \tag{2.20}$$

c. Algoritmo de posicionamiento

El algoritmo de posicionamiento que se ha desarrollado está compuesto por una serie de funciones, que reciben la señal *Pitch* y *Yaw* obtenidas a partir del EKF y devuelven la posición relativa del individuo. Este algoritmo de posicionamiento está compuesto de un detector de pasos, un estimador de la anchura del paso (SL) y un algoritmo de posición relativa como se expone en la Figura 7.

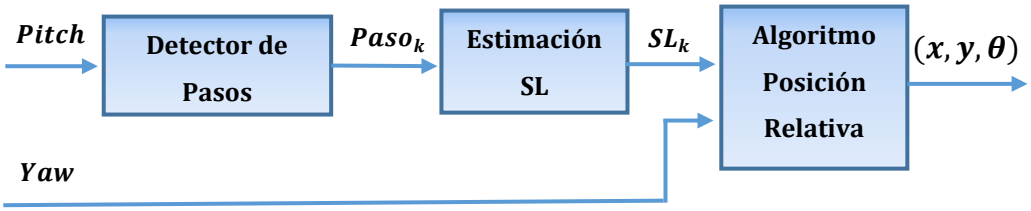

Figura 7: Diagrama de bloques del algoritmo de posicionamiento.

Siendo x e y, las coordenadas sobre el plano, y θ la orientación.

i. Detector de pasos

El algoritmo desarrollado para la detección de pasos se basa en la detección de los máximos y mínimos absolutos de la señal *Pitch*.

Se han probado otras alternativas como los cruces por cero o la derivada de la señal, pero no han dado los resultados esperados debido a que, a partir de los valores medidos de nuestra IMU, la señal *Pitch* resultante no cumple con criterios expuestos en la literatura [Munoz, 2016] de que en superficies horizontales siempre se produce cruce por cero de la señal *Pitch*, además de ser bastante ruidosa con lo que se producen falsos máximos locales que dificultan la obtención del máximo absoluto con la derivada.

A continuación, en la Figura 8, se muestra la señal *Pitch* obtenida en una prueba hecha alrededor de una superficie rectangular de dimensiones 9x4m aproximadamente; en la que como se puede observar, la señal *Pitch* no cruza por cero, pero al aplicarle el algoritmo de detección de máximos ($\odot_{max}$, señalados en verde) y mínimos ($\odot_{min}$, señalados en rojo) desarrollado en este TFG los detecta perfectamente. Siendo el eje de tiempos, el *timestamp*, el contador interno de la IMU.

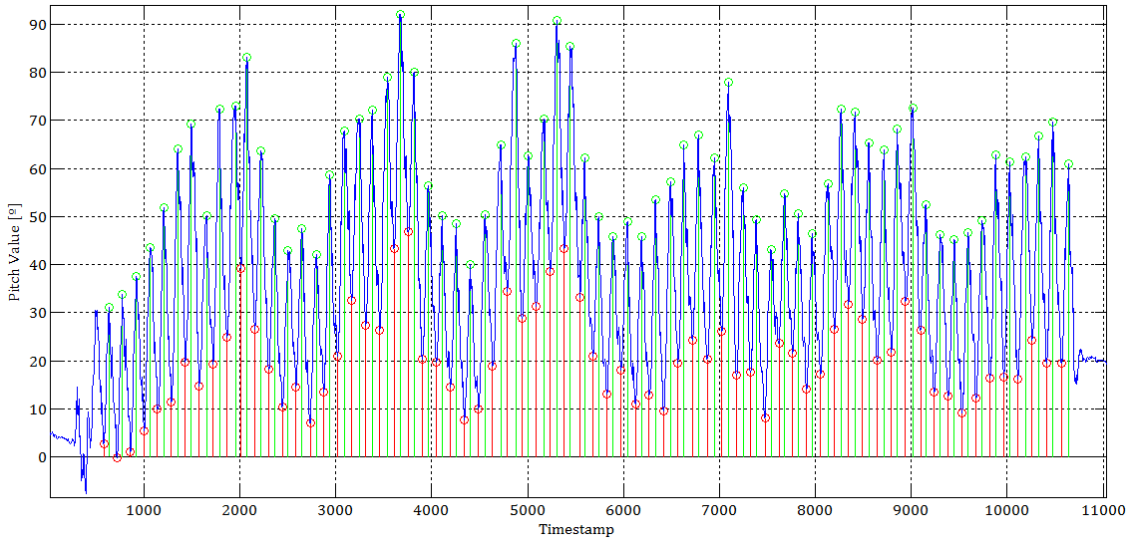

Figura 8: Ángulo Pitch durante una trayectoria rectangular.

Si la IMU está situada en la parte superior de la pierna del individuo, la señal *Pitch* describe el ángulo de apertura de la persona al caminar. Por lo que en este algoritmo al detectar máximos y mínimos de la señal *Pitch*, tendremos el ángulo de apertura máximo y mínimo de la persona al caminar, que a posteriori servirá para realizar una estimación de la anchura del paso dado.

En la Figura 9 se muestra la señal *Pitch* en otro experimento durante 15 pasos del mismo tras aplicarle el algoritmo de detección de pasos (explicación práctica del desarrollo del algoritmo en la sección Desarrollo de la aplicación).

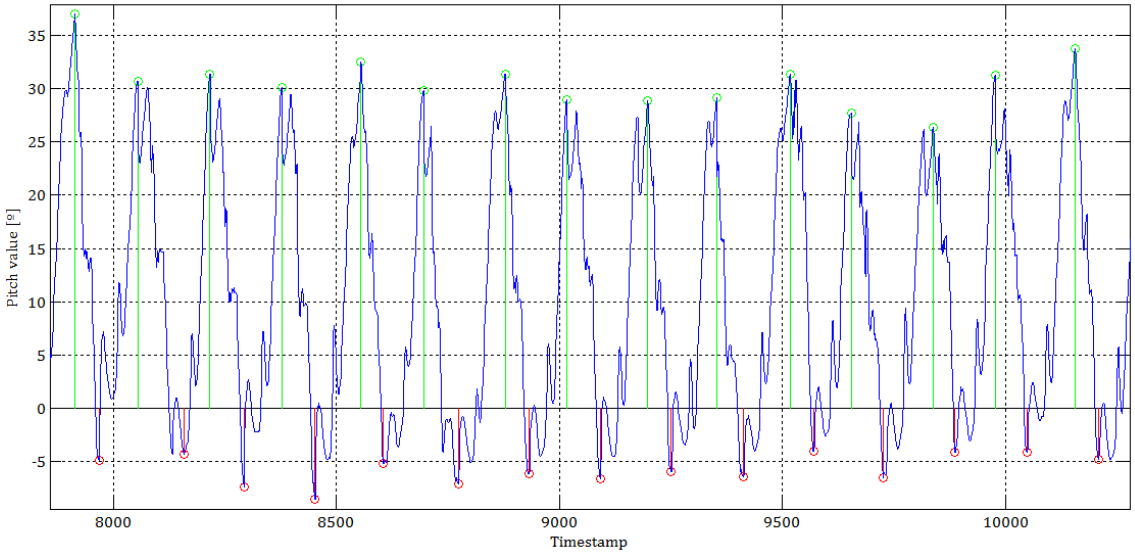

Figura 9: Ángulo Pitch durante un paseo en superficie horizontal, ilustrando 15 pasos del mismo.

ii. Estimación de la anchura de paso

Una vez hecha la detección del paso junto con su máximo y mínimo absoluto asociados al mismo, se puede calcular la amplitud del ángulo de apertura de la pierna en ese paso o también llamada amplitud de *Pitch* ($\Delta\odot$) que se define como la diferencia entre el máximo y el mínimo absoluto.

$$\Delta\odot_k = \odot_{max} - \odot_{min} \tag{2.21}$$

Según la literatura para cada amplitud de la señal *Pitch* corresponde una anchura de paso diferente. En la Figura 10 se representa la posición límite de las piernas de un individuo durante un paso, señalando cuando se produce la elongación máxima y mínima, y la anchura de paso.

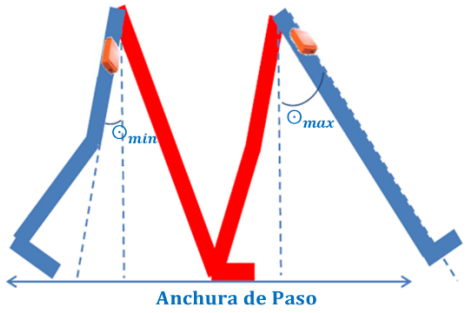

Figura 10: Representación de las elongaciones límite de las piernas del individuo en un paso.

En estudios previos realizados en el Instituto de comunicaciones y navegación del centro aerospacial (DLR) de Munich [Munoz, 2016] con diferentes voluntarios andando a diferentes velocidades en una máquina de correr (para obtener velocidades fijas) sin desnivel (en horizontal), obtienen la siguiente relación entre la anchura de paso (Step Length (*SL*)) y la amplitud de Pitch (*Pitch amplitude* ($\Delta\odot$)):

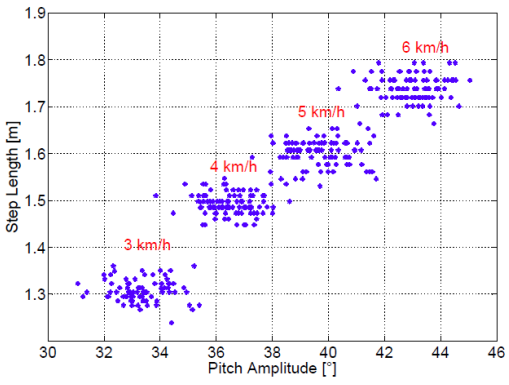

Figura 11: Relación entre SL y $\Delta\odot$ en el experimento realizado por el DLR [Munoz, 2016].

A partir de ese experimento realizan una regresión lineal de mínimos cuadrados de los diferentes voluntarios (rectas verde, azul, roja y morada de la figura 12) con las cuales se obtiene una regresión lineal universal (recta negra) de todos ellos.

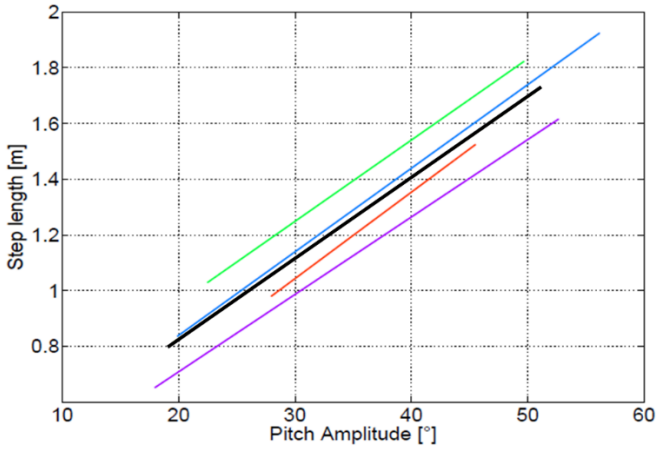

Figura 12: Regresión lineal universal (negra) de la relación entre SL y ΔΘ para diferentes voluntarios (verde, azul, roja y morada) [Munoz, 2016].

Para el algoritmo propuesto, se usará la recta de regresión lineal universal, ya que la aplicación en la que lo vamos a implementar va a funcionar en tiempo real y por lo tanto no se podría realizar un estudio previo a cada individuo y conseguir la regresión lineal específica del mismo, a costa de perder precisión a la hora de la estimación del paso; algo que es perfectamente asumible ya que como se explicó en la introducción, el objetivo de este TFG es la navegación en las zonas que no tienen cobertura de las balizas colocadas en el techo, corrigiendo la posición con los ultrasonidos cuando se llegue a las zonas con cobertura de los U-LPS.

Por tanto, la ecuación del algoritmo utilizado para la estimación de la anchura del paso es la siguiente:

$$\mathrm{SL}_k = a_H * \Delta\Theta_k + b_h \tag{2.22}$$

Siendo a_H y b_H, 0.0294 y 0.232 respectivamente, valores calculados a partir de esa regresión lineal universal de la Figura 12.

La gran ventaja de este algoritmo frente a otros de la literatura es que se obtiene una anchura de paso dinámica en función del ángulo de apertura de la pierna del individuo en el paso; es decir, en función de la velocidad del usuario.

iii. Algoritmo de posición relativa

Esta parte del algoritmo estima la posición relativa sobre el plano del usuario ($\widehat{\mathrm{Y}}_k$) que se compone de la estimación del eje x ($\hat{x}_k$), del eje y ($\hat{y}_k$) y de la orientación ($\hat{\theta}_k$).

La posición relativa se calcula a partir de la estimación previa ($\widehat{\mathrm{Y}}_{k-1}$) y del incremento de distancia ($\Delta d_{\mathrm{PDR}} = \mathrm{SL}_k$, obtenido mediante el algoritmo de anchura de paso) y de orientación ($\Delta\theta_{\mathrm{PDR}}$=$\Delta\Psi$, variación del valor del *Yaw* en la posición del máximo absoluto entre el paso actual (k) y el anterior (k-1)).

$$\widehat{\mathbf{Y}}_k = \begin{bmatrix} \hat{x}_k \\ \hat{y}_k \\ \hat{\theta}_k \end{bmatrix} = \begin{bmatrix} \hat{x}_{k-1} + \mathrm{SL}_k \cdot \cos\left(\hat{\theta}_{k-1} + \Delta\Psi\right) \\ \hat{y}_{k-1} + \mathrm{SL}_k \cdot \sin\left(\hat{\theta}_{k-1} + \Delta\Psi\right) \\ \hat{\theta}_{k-1} + \Delta\Psi \end{bmatrix} \tag{2.23}$$

Destacar de este algoritmo, que se inicializa a 0 ($\widehat{\mathbf{Y}}_0 = [0 \quad 0 \quad 0]^T$) con lo que servirá para visualizar y diferenciar tipos de movimientos como trayectorias en recto o circulares, pero no ofrece información sobre la dirección de movimiento en la que se realiza la prueba. Para poder hallar el sentido/dirección en la que se realizan las pruebas habría que introducir condiciones no nulas, para lo que se necesitaría un conocimiento previo de la posición del individuo a la hora de empezar el test.

Los algoritmos PDR suelen ser poco precisos debido a que, al trabajar con información relativa, no absoluta, sus valores ya contienen un error, el cual con el paso de las iteraciones se va acumulando. Este error acumulativo se podrá corregir cuando se disponga de información absoluta, como puede ser el caso de la información obtenida por los U-LPS o con información de los mapas (el error acumulativo podría llevar a trayectorias como que el usuario atraviese paredes; lo cual se podría solucionar con una implementación de grafos del plano).

Además, el *Yaw* es la señal más sensible con la que se está trabajando, por lo que es fundamental que la IMU esté bien sujeta a la pierna, alineada respecto a la orientación del pie y que no se mueva durante las pruebas, para conseguir la mayor precisión posible y evitar errores a la hora de obtener la posición relativa.

3. Descripción del material utilizado

En esta sección se describen las principales características del material utilizado para la realización del TFG, recogidas en la Tabla 1 expuesta a continuación:

Equipo	Descripción
IMU	Unidad de medición inercial o IMU (del inglés *inertial measurement unit*), es un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales de un aparato, usando una combinación de acelerómetros (miden la aceleración) y giróscopos (miden los cambios en la orientación del dispositivo en los tres ejes de coordenadas). La IMU es el componente principal de los sistemas de navegación inercial usados en aviones, naves espaciales, buques y misiles guiados entre otros. En este uso, los datos recolectados por los sensores de una IMU permiten a un computador seguir la posición del aparato, a partir de la detección de los cambios en los ángulos de alabeo (*Roll*), cabeceo (*Pitch*) y guiñada (*Yaw*). En nuestro caso se ha utilizado el Shimmer3 (http://www.shimmersensing.com/products/shimmer3), sensor diseñado para usos portátiles y de detección remota, además de ser altamente flexible y adaptable; de ahí que se utilice con frecuencia en diversas actividades como la vigilancia de la actividad, la ciencia del deporte, y las aplicaciones de construcción inteligente. Las características y especificaciones principales de esta IMU son las siguientes: • CPU MSP430 de 24MHz. • Detección inercial integrada mediante acelerómetros, giróscopos y magnetómetros (de los tres ejes), cada uno con

	rango seleccionable por el usuario en un software que proporciona el fabricante (Consensys). • Dos tipos de acelerómetros, que dan la opción entre el ruido ultrabajo (*accel Low Noise*) o la gama amplia (*accel Wide Range*). • Consumo de energía muy bajo y peso ligero. • Interruptor deslizante para encender/apagar. • Se conecta fácilmente a través de Bluetooth o almacenamiento local a la tarjeta microSD. • Conectores internos y externos para expansión (los módulos de expansión incluirán entre otros, señales de ECG (electrocardiograma) y EMG (Electromiografía)). • 5 LEDs de color en 2 ubicaciones para indicar el estado del dispositivo y el modo de funcionamiento, así como indicando la funcionalidad de transmisión de Bluetooth. • Dimensiones: 51mm x 34mm x 14mm.
Tablet	Samsung Galaxy Tab S2, procesador de 64 bits y 8 núcleos a 1.9GHz, 3GB de RAM y 32 GB de ROM.
Matlab	Herramienta de software matemático que ofrece un entorno de desarrollo con un lenguaje de programación propio (lenguaje M).
Android Studio	Entorno de desarrollo aplicaciones sobre plataforma Android.
Ordenador	Intel Core i7 2.93Ghz, 3GB de RAM y procesador de 64 bits.

Tabla 1: Equipos y *software* utilizados en el TFG.

4. Desarrollo de la aplicación

a. Introducción

En este TFG se va a realizar una aplicación sobre una plataforma Android, que permita detectar los pasos del usuario, estimar la distancia recorrida y visualizar la posición relativa de la trayectoria realizada, mediante las medidas de los sensores inerciales de la IMU.

Para explicar la aplicación se han diferenciado las siguientes partes, que se explicarán (como se ha implementado, parte teórica ya se expone en el apartado de base teórica) en las secciones posteriores, siendo las que se exponen a continuación: configuración del Shimmer (configuración de la IMU con las características necesarias para la app), programación de la conexión bluetooth entre dispositivos (como se realiza la conexión entre la IMU y el dispositivo portable), programación para la obtención de las señales aceleración y giróscopo (cómo obtener las señales adecuadas a partir de los valores enviados por el Shimmer3), programación de la extracción de los valores del sistema inercial mediante EKF (cómo conseguir los ángulos de Euler a partir de las señales de acelerómetros y giróscopos), programación del algoritmo de posicionamiento (cómo obtener la posición relativa a partir de los ángulos de Euler) y programación de la parte de visualización de la posición (cómo representar la trayectoria realizada por el usuario).

Antes de empezar a explicar esas secciones expuestas con más detalle, se va a hacer una breve introducción a Android y su lenguaje de programación, ya que será útil para entender posteriormente por qué se ha programado así la aplicación.

Lo primero que se hace a la hora de crear una aplicación Android es elegir la mínima versión (*minimun SDK*) en la cual se puede ejecutar la app y cuál es la versión objetivo (*target SDK version*), para la cual está prevista que se use la aplicación. Lo normal, es utilizar la mínima versión posible, para que así la app pueda ser accesible al mayor número de usuarios (se ha utilizado la API 15-Android 4.0.3, IceCreamSandwich-, que a día de hoy es compatible con el 97.4% de dispositivos). De versión objetivo se ha escogido la API 25 (Android 7.1.1, Nougat), que es la última

versión disponible (es recomendable que sea lo mayor posible, para que depure fallos posibles en versiones anteriores).

Cada proyecto Android contiene un descriptor de la aplicación o *manifest*, que es donde se programan los permisos de la aplicación (Bluetooth, SD), el icono de la aplicación, y las diferentes actividades del proyecto (entre ellas se configura cuál es la actividad de arranque al iniciar la app (*Launcher*)); ficheros para construir el módulo/proyecto, *Gradle Scripts*, donde se configura la versión objetivo para compilar el programa, la versión mínima y donde se añaden las librerías externas a Android utilizadas. En este proyecto se han utilizado las siguientes librerías externas: *Apache Commons Math 3.4.1*, para poder realizar operaciones matemáticas de matrices; *Androidplot Core Library 0.9.8*, para poder trabajar con gráficas; y la librería *Jcoord 1.0* que permite convertir entre latitud/longitud (WG S84) y coordenadas UTM, necesaria a la hora de obtener las coordenadas para representar en Google Maps. Por último, y lo más importante de un proyecto Android son las actividades (*activities*), que están conformadas por dos partes: la parte lógica y la parte gráfica. La parte lógica es el código fuente, un archivo .java que es una clase, que al ser una actividad extiende de la clase *Activity*, y en el cuál se programa todo lo relacionado con la funcionalidad para la cual se realiza la app (algoritmos, funcionalidad de los botones, etc.). La parte gráfica es un XML que tiene todos los elementos que se ven por pantalla como iconos, botones, texto, etc.; es decir se encarga del diseño de las diferentes pantallas de la aplicación (*layouts*).

Cabe destacar el ciclo de vida de una actividad ya que las actividades son las que realmente controlan el ciclo de vida de las aplicaciones. Una actividad puede tener 4 estados:

- Activa (*running*), actividad visible, tiene el foco.
- Visible (*paused*), actividad visible pero no tiene el foco, por ejemplo, cuando se pasa a otra actividad que no ocupa toda la pantalla.
- Parada (*stopped*), actividad no visible, tapado por completo.
- Destruida (*destroyed*), cuando la actividad termina o es finalizada por el sistema.

En resumen, el ciclo de vida de una actividad se ilustra en la Figura 13:

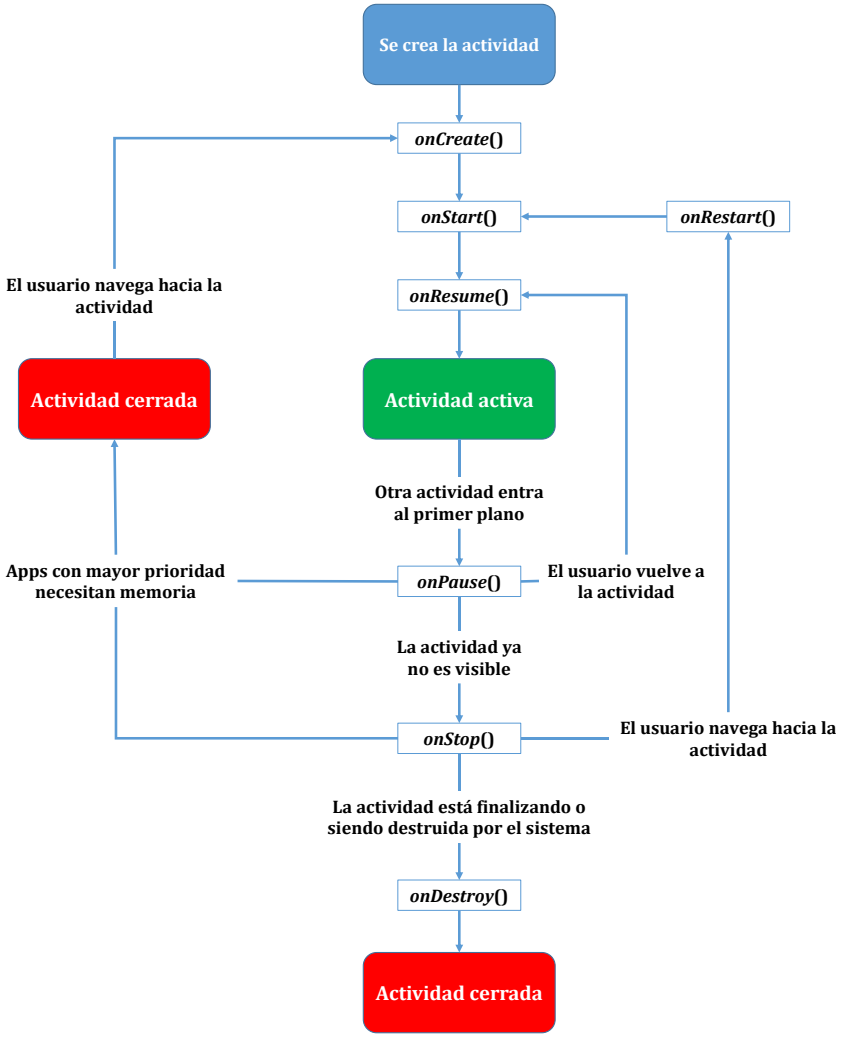

Figura 13: Ciclo de vida de una actividad Android.

Cada vez que una actividad cambia de estado se van a generar eventos que podrán ser capturados por ciertos métodos de la actividad y en los cuales se permitirá programar el código. Los diferentes métodos que capturan estos eventos son:

- *onCreate*(): se utiliza para programar las inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos.
- *onStart*(): indica que la actividad está a punto de ser mostrada.
- *onResume*(): cuando va a comenzar a interactuar la actividad con el usuario. Sirve para lanzar animaciones, música, etc.
- *onPause*(): antes de lanzar la actividad a segundo plano. Se suele utilizar para detener animaciones, música y almacenar datos.

- *onStop*(): la actividad deja de ser visible para el usuario, si hay poca memoria, puede que se destruya la actividad sin pasar por este método.
- *onRestart*(): actividad vuelve a ser representada tras *onStop*().
- *onDestroy*(): se llama antes de que la actividad sea destruida totalmente. Al igual que el método *onStop*, si hay muy poca memoria puede que ni se ejecute.

Para poder manejar variables en varias clases sin tener que usar constructores (función dentro de una clase con su mismo nombre, que permite manipular las variables que se introducen como entradas a la clase), se ha creado una clase (llamada *Link_data*), en la cual almacenar las variables y que puedan ser accesibles desde todas las clases.

Esta aplicación desarrollada implementa una serie de algoritmos, que conllevan un procesado interno que el usuario no observa. En Android, hay un problema a la hora de realizar tareas que se extienden en el tiempo, ya que a partir de los 5s de duración de un proceso en la tarea principal muestra automáticamente un diálogo AMR, de que la app no responde. En la tarea principal se ejecutan todas las operaciones que gestionan la interfaz de usuario de la aplicación, además de las actividades, servicios y *broadcast receivers*. Es por ello que cualquier operación costosa/larga que se realice en el hilo principal, va a bloquear la ejecución del resto de componentes de la aplicación, y por supuesto también la interfaz, produciendo al usuario un efecto evidente de lentitud, bloqueo o mal funcionamiento en general, algo que se debería evitar a toda costa, ya que podría suponer la desinstalación de la aplicación por parte del usuario. La solución para evitar este suceso es utilizar tareas asíncronas (hilos), consiguiendo que las tareas pesadas se ejecuten en un hilo aparte, evitando el bloqueo de la interfaz.

Hay dos formas de trabajar con hilos, mediante *Thread* o mediante *AsyncTask*. Para esta app se ha utilizado el tipo *Thread* que consiste en una clase extendida a hilo que en su interior contiene una función *run*, en la cual se introduce todo el código a ejecutar cuando el hilo una vez creado, se arranque (hilo.start()). Al utilizar este tipo de hilos, hay más opciones a la hora de manipular el hilo (mediante comandos como: hilo.join(), hilo.Run(), hilo.getPriority(), etc.).

El hilo de mayor importancia que se utiliza en esta aplicación es el llamado *ReadingThread*, que es el hilo en el cual se produce desde la lectura de los valores enviados por el Shimmer (vía Bluetooth) hasta el algoritmo de posicionamiento.

Por último, para acabar con esta introducción, decir que a excepción del *main* los hilos no pueden interactuar con la interfaz de usuario, por lo que para visualizar algo por pantalla desde un hilo se deberá utilizar el manejador del hilo (*Handler*).

b. Configuración de la IMU

El Shimmer3 se configura desde el software Consensys, proporcionado por el fabricante de las IMU (Shimmer) y permite programarle el *firmware* al dispositivo, su frecuencia y las medidas que se quieren recibir del mismo.

Para esta aplicación se ha programado al Shimmer el firmware LogAndStream_0.7.1_UAH.txt desarrollado dentro del grupo de investigación en el que se encuentra este trabajo (GEINTRA), que es una modificación del firmware que proporciona el fabricante (LogAndStream_Shimmer3_v0.6.0) ya que se detectaron fallos en el mismo, como que al haber cualquier tipo de desconexión entre el móvil y el Shimmer (debido a distancia entre ambos, que se quede alguno de ellos sin batería, etc.) el buffer de datos no se inicializaba.

Se ha configurado el Shimmer con una frecuencia de 128Hz para que envíe las medidas de la batería del mismo, de los acelerómetros (*wide range accelerometer*) y de los giróscopos. Con esta configuración, se evita la pérdida de paquetes de información a la hora de la comunicación bluetooth, cuestión que se ha comprobado experimentalmente que ocurría al pedir al Shimmer que mandara más información (medidas de otros sensores que contiene la IMU como la aceleración LN (*low noise*)) y a mayor rapidez (frecuencias superiores).

En la literatura se expresa que la frecuencia óptima en aplicaciones PDR es en torno a 100 Hz [Hervás, 2014] y como el reloj del Shimmer se configura en múltiplos de dos, se han elegido esos 128 Hz (frecuencia múltiplo de dos más cercana a 100Hz).

Con esta configuración, en cada iteración, el Shimmer envía 18 Bytes al dispositivo vía Bluetooth, en el orden que se expresa a continuación:

- Byte inicial [0]: de valor 255 la primera vez y el resto usamos ese byte para enviar el byte 18 de la iteración anterior.
- Byte ACK o byte de control [1]: de valor 0 en todas las iteraciones menos en la última (al mandar al Shimmer que pare de enviar información) que tiene valor 255, permitiendo saber si estamos o no en la última iteración.
- *Timestamp,* contador interno del Shimmer [2:4].
- Batería del Shimmer [5:6].
- Giróscopo [7:12] (Giróscopo_x [7:8], Giróscopo_y [9:10] y Giróscopo_z [11:12]).
- Acelerómetro [13:17] (Acelerómetro_x [13:14], Acelerómetro_y [15:16] y Acelerómetro_z [17] (el segundo byte correspondiente al Acelerómetro_z es el enviado en la posición cero de la iteración siguiente)).

Cabe destacar que esta información se envía toda en formato Little Endian (primero byte menos significativo (LSB) y en último lugar el más significativo (MSB)) excepto la de los giróscopos, que se envía en formato Big Endian (por ejemplo, para el giróscopo x, primero se envía byte de mayor peso (MSB) Giróscopo_x [7] y luego el de menor peso (LSB) Giróscopo_x [8]).

c. Programación de la conexión Bluetooth entre dispositivos

La aplicación a desarrollar en este TFG utiliza comunicación bluetooth para recibir en el dispositivo portable los datos de los sensores inerciales de la IMU (Shimmer3). Por tanto, una vez se tenga configurado la IMU con las características deseadas, habrá que realizar la conexión bluetooth entre el dispositivo portable en el que se carga la app y la IMU, para que se puedan mandar información entre ellos.

Para esta parte de la aplicación se ha utilizado una *activity* realizada previamente por Edel Díaz [Díaz, 2017], por lo que no se entrará a fondo a explicar el código ya que no ha sido objeto de estudio en este TFG. No obstante, se explicará a groso modo cómo funciona y qué se ha usado a la hora de crear las conexiones Bluetooth entre el dispositivo portable y los dispositivos con visibilidad bluetooth externos.

Inicialmente la app comprobará que el bluetooth del dispositivo portable utilizado (tablet, móvil, etc.) esté conectado y pedirá permiso al usuario para que permita

conectarlo en caso de que no lo esté (como se observa en la Figura 14). Esto se realizará a partir del adaptador Bluetooth (*BluetoothAdapter*). El *BluetoothAdapter* representa el adaptador local de Bluetooth (radio Bluetooth), siendo el punto de entrada de toda interacción de Bluetooth. Gracias a esto, se pueden ver otros dispositivos Bluetooth, consultar una lista de los dispositivos conectados (sincronizados) o crear una instancia (conexión) con uno de esos dispositivos Bluetooth remotos (*BluetoothDevice*) de dirección MAC conocida, en nuestro caso el Shimmer3. En *BluetoothDevice* se almacena la información sobre el dispositivo conectado, como su nombre, dirección, clase y estado de conexión.

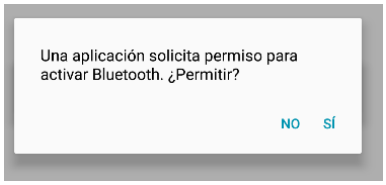

Figura 14: Captura de pantalla del permiso encender Bluetooth.

Esta *activity* permite observar los dispositivos disponibles en una tabla y conectarte a uno de ellos si no estás vinculado a ningún dispositivo o desconectarte de un dispositivo al que ya estás vinculado. A continuación, se muestra en la Figura 15 una captura de la pantalla de la app en el momento que te ilustra la lista de dispositivos Bluetooth visibles. Al tocar uno de esos dispositivos visibles de la lista, se habilita el botón de conectar (por ejemplo, en la captura que se muestra a continuación el botón de desconectar esta deshabilitado), permitiendo pulsarlo y crear la conexión entre el dispositivo portable en el que se ha cargado la app y el dispositivo remoto, en este TFG el Shimmer3-2ABA.

En el caso de conectarse, se crea el *BluetoothSocket,* que representa la interfaz de un socket de Bluetooth, siendo el punto de conexión que permite que una aplicación intercambie datos con otro dispositivo Bluetooth a través de los *buffers* InputStream (permite el envío de información del Shimmer al dispositivo portable en el que se carga la aplicación) y OutputStream (permite el envío de información del dispositivo portable al Shimmer). En el caso de la desconexión entre IMU y dispositivo portable, se cerrarán dichos *buffers*.

La velocidad de transmisión es de 9600 baudios, valor fijado por el Shimmer.

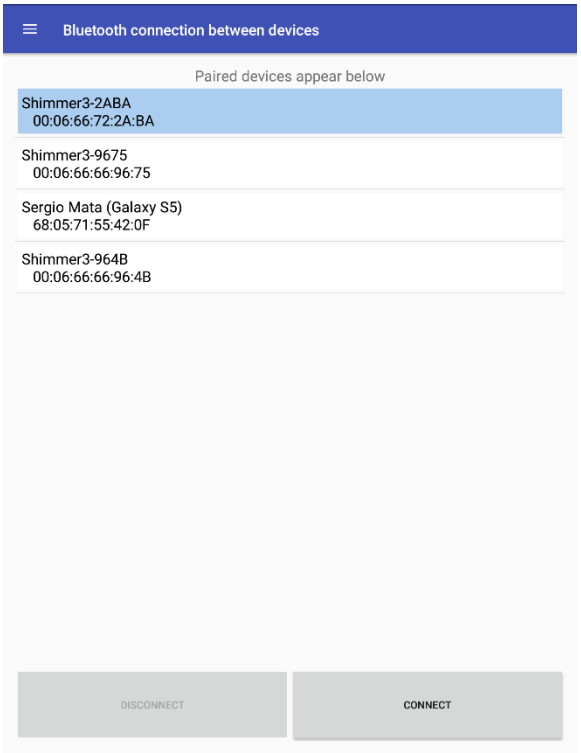

Figura 15: Captura de pantalla del layout vinculado a la activity Bluetooth.

Destacar, que a partir de las hojas de características del Shimmer, se obtienen los siguientes códigos en decimal, que se usan para decirle al Shimmer qué hacer. Nosotros utilizamos la variable *packet_types* para almacenar el valor que escribir en el buffer OutputStream para controlar el Shimmer.

- *packet_types* = 7 -> código a enviar al Shimmer para que empiece a mandar datos vía Bluetooth.
- *packet_types* = 32 -> código a enviar al Shimmer para que pare de mandar datos vía Bluetooth.
- *packet_types* = 112 -> código a enviar al Shimmer para que empiece a mandar datos vía Bluetooth y grabe esos valores enviados en la SD del Shimmer.
- *packet_types* = 151 -> código a enviar al Shimmer para que pare de mandar datos vía Bluetooth y acabe de grabar en la SD del Shimmer.
- *packet_types* = 146 -> código a enviar al Shimmer para que grabe las medidas configuradas en la SD del Shimmer.
- *packet_types* = 147 -> código a enviar al Shimmer para que pare de grabar en la SD del Shimmer.

- *packet_types* = 3 -> código a enviar al Shimmer para mande la frecuencia a la cual se ha configurado el Shimmer.

Dentro de esta *activity*, se pedirá al Shimmer que envíe la frecuencia del Shimmer y la almacene dentro de la variable *frequency* de la clase Link_data, por lo que se podrá acceder a su valor desde cualquier otra clase refiriéndose a la variable Link_data.*frequency*. De esta forma se tiene almacenado el valor al que se ha configurado el Shimmer de manera dinámica, valor que se necesitará posteriormente al realizar el EKF para hallar el periodo de muestreo (la inversa de la frecuencia). Es imprescindible que se lea bien esa frecuencia, ya que es necesaria para el correcto funcionamiento del algoritmo; de ahí que en el caso de que no se lea correctamente esa frecuencia, se mandará un mensaje al usuario por pantalla comentándole el problema en la lectura de la frecuencia y se procederá a eliminar la conexión entre los dispositivos, remitiendo al usuario a que vuelva a intentarlo hasta que se consiga una conexión con lectura de la frecuencia correcta.

En el buffer InputStream se recibirán en bruto, los valores de las señales seleccionadas para extraer del Shimmer. En el siguiente apartado (Programación para la obtención señales aceleración y giróscopo) se explicará cómo se hace para a partir de los valores obtenidos vía Bluetooth de la IMU, adecuar las señales para que sean aptas para realizar el algoritmo EKF.

d. Programación para la obtención de las señales aceleración y giróscopo

El Shimmer manda por el buffer los bytes de las medidas que han sido configuradas. Manda esos bytes en decimal, por lo que lo primero que hay que hacer al recibir los datos es pasar esos valores que representan una medida en varios bytes a un valor único. Nos vamos a centrar específicamente en las señales de aceleración y giróscopo que tienen dos bytes para cada eje, por lo que, para obtener el valor único, se multiplicará el byte de mayor peso (MSB) por 2^8 (256, 1byte tiene 8 bits) y se le sumará al byte de menor peso (LSB), siguiendo la siguiente expresión:

$$\textbf{valor\_no\_calibrado} = -((\sim(\text{short})(\text{msb} * 256 + \text{lsb})) + 1) \qquad (4.1)$$

Cabe destacar que para otras medidas recibidas como el *timestamp* (de tres bytes), se realizará lo mismo multiplicando el byte de mayor peso por dos elevado al bit donde empieza ese byte (2^{16}, para el caso de 3 bytes, 2 bytes ocuparían los bits [0:15]) y que es fundamental tener en cuenta el formato en el que se envía cada medida (Big o Little Endian), para seleccionar que byte es el MSB y cual el LSB y calcular correctamente el valor de la medida recibida.

Las medidas de cada uno de los ejes se introducirán en formato vector columna, obteniendo las matrices *gyro_no_calibrate* y *wraccel_no_calibrate*, que siguen la siguiente expresión:

$$\boldsymbol{medida\_no\_calibrada}[\mathbf{3}][\mathbf{1}] = \begin{bmatrix} medida\_no\_calibrada\_x \\ medida\_no\_calibrada\_y \\ medida\_no\_calibrada\_z \end{bmatrix} \tag{4.2}$$

Una vez calculado el valor no calibrado, se calibran las medidas de los sensores inerciales de la IMU (acelerómetros y giróscopos) a partir de la fórmula de calibración de la IMU proporcionada por el software Consensys, que se muestra a continuación:

$$\mathbf{c} = \boldsymbol{R}^{-1} * \boldsymbol{K}^{-1} * (\boldsymbol{u} - \boldsymbol{b}) \tag{4.3}$$

Siendo **c** la matriz calibrada, $\boldsymbol{R}$ la matriz de alineación, $\boldsymbol{K}$ la matriz de sensibilidad, **u** la matriz no calibrada y $\boldsymbol{b}$ la matriz de offset.

Para calibrar de manera estándar (en un rango de valores del giróscopo entre $\pm 500º/s$) las medidas del giróscopo, se utilizan las siguientes matrices:

$$\mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.4}$$

$$\mathbf{K} = \begin{bmatrix} 65.5 & 0 & 0 \\ 0 & 65.5 & 0 \\ 0 & 0 & 65.5 \end{bmatrix} \tag{4.5}$$

$$\mathbf{R} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{4.6}$$

Para las medidas del acelerómetro WR (*wide range*) con una calibración estándar (en un rango de aceleraciones entre $\pm 2g$ ($g=9.8 m/s^2$)), se utilizan las siguientes matrices:

$$\mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.7}$$

$$\mathbf{K} = \begin{bmatrix} 1631 & 0 & 0 \\ 0 & 1631 & 0 \\ 0 & 0 & 1631 \end{bmatrix} \tag{4.8}$$

$$\mathbf{R} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{4.9}$$

Como las matrices **K** y ***R*** son fijas tanto para aceleración como para giróscopo, se realizará el cálculo de sus inversas y el producto de ambas al inicio del programa, con el fin de ejecutarlo una sola vez y no en cada iteración, ganando así tiempo de ejecución.

Estas señales calibradas ya son aptas para realizar la extracción de los valores del sistema inercial mediante un EKF como se mostrará en el siguiente apartado.

e. Programación de la extracción de los valores del sistema inercial mediante EKF

i. Programación en Matlab y en Android

El objetivo de esta sección es a partir de los valores obtenidos de aceleración y giróscopo obtener una estimación de la orientación del Shimmer (colocado en la pierna del usuario), de los ángulos de Euler (Roll (ϕ), Pitch ($\odot$) y Yaw (Ψ)).

Inicialmente la estimación de los ángulos de Euler ($[x_0 = \hat{\phi}_0, \hat{\odot}_0, \hat{\Psi}_0]$) se obtiene a partir de las ecuaciones (2.14), (2.15) y (2.16) explicadas en la sección de base teórica. Para el resto de iteraciones se tomará como conocimiento previo el estado a posteriori (x_k) y la matriz de covarianza del error a posteriori (P_k) calculados en la iteración anterior ($k-1$). Siendo x_k la estimación del Roll, Pitch y Yaw.

$$\boldsymbol{x}_k = \begin{bmatrix} \widehat{\Phi}_k \\ \widehat{\odot}_k \\ \widehat{\Psi}_k \end{bmatrix} \tag{4.10}$$

A partir de ese conocimiento previo y de las señales del giroscopio y del acelerómetro se realiza la etapa de predicción, según las ecuaciones (2.5) y (2.6).

En concreto, el cálculo de la estimación del estado (2.5) se realiza con la información relativa de los giróscopos (ω), siguiendo la siguiente expresión:

$$\boldsymbol{x_k}^- = \begin{bmatrix} \widehat{\Phi}_{k-1} + \omega_{k,x} * Ts \\ \widehat{\odot}_{k-1} + \omega_{k,y} * Ts \\ \widehat{\Psi}_{k-1} + \omega_{k,z} * Ts \end{bmatrix} = \begin{bmatrix} \widehat{\Phi}_k^- \\ \widehat{\odot}_k^- \\ \widehat{\Psi}_k^- \end{bmatrix} \tag{4.11}$$

Siendo Ts, el periodo de muestreo, inverso de la frecuencia de muestreo, la frecuencia a la que se configura el Shimmer. Ts se puede calcular también con la diferencia entre la marca de tiempos (*timestamp*) actual y la de la iteración anterior proporcionada por el Shimmer, según se muestra a continuación:

$$\boldsymbol{Ts_k} = \frac{1}{f_{shimmer}} = \frac{timestamp_k - timestamp_{k-1}}{1000} \tag{4.12}$$

Como se comentó en la sección teórica, en función de la norma de la aceleración, se realizará (si $|\alpha| = \sqrt{{\alpha_{k,x}}^2 + {\alpha_{k,y}}^2 + {\alpha_{k,z}}^2} \approx 9.8\ \mathrm{m/s^2}$) o no (se actualiza el vector de estado con el vector de estado a priori calculado en la etapa de predicción) la etapa de corrección.

En esa etapa de corrección, se actualizará el valor del observable (z_k) a partir del valor actual del Roll y Pitch calculado con los acelerómetros (4.13) y se pondrá al vector de estimaciones de las observaciones ($h(x_k^-)$) las estimaciones realizadas en la etapa de predicción (4.14).

$$\boldsymbol{z}_k = \begin{bmatrix} \tan\left(\dfrac{\alpha_{k,y}}{\alpha_{k,z}}\right)^{-1} \\ \tan\left(\dfrac{-\alpha_{k,x}}{\sqrt{{\alpha_{k,y}}^2 + {\alpha_{k,z}}^2}}\right)^{-1} \end{bmatrix} \tag{4.13}$$

$$\boldsymbol{h}(\boldsymbol{x_k}^-) = \begin{bmatrix} \widehat{\Phi}_k^- \\ \widehat{\Theta}_k^- \end{bmatrix} \tag{4.14}$$

Una vez obtenidas estas matrices, se realiza la etapa de corrección a partir de las ecuaciones (2.8), (2.9), (2.10), (2.11) y (2.12). Como todas esas matrices se calculan a partir de matrices ya conocidas, para simplificar el código principal, se ha realizado en Matlab una función y en Android una clase, con variables de entrada x_k^-, P_k, A, H, z_k, $h(x_k^-)$, Q y R que realiza en su interior esas ecuaciones, devolviendo las matrices útiles para iteraciones posteriores, x_k y P_k.

Por último, se ilustra en la Figura 16 el diagrama de flujo de la parte del programa que tanto en Matlab como en Android realiza la extracción de los valores del sistema inercial.

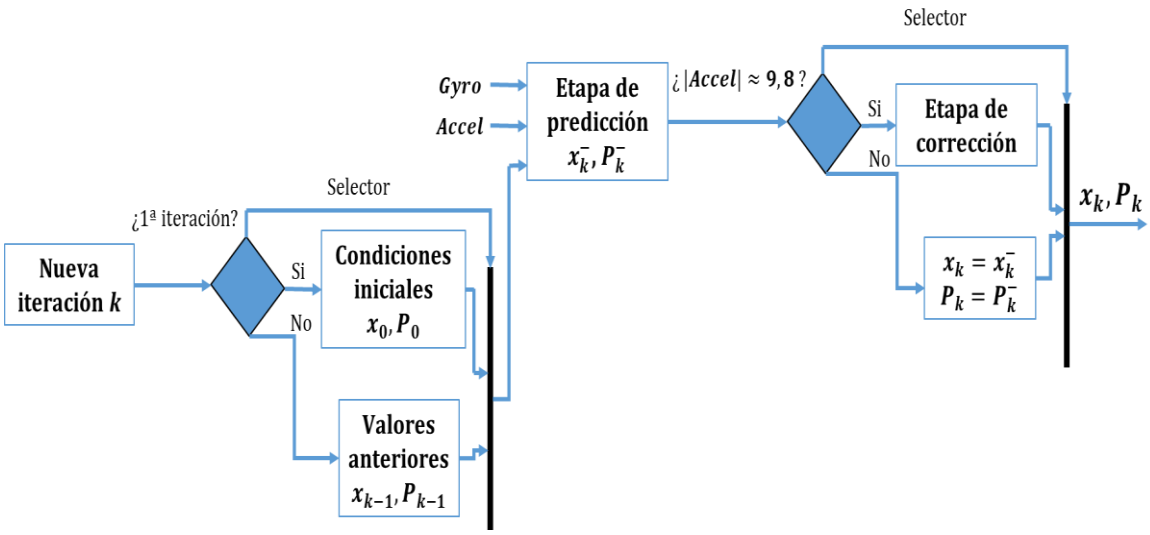

Figura 16: Diagrama de flujo de la extracción de los valores del sistema inercial mediante EKF.

ii. Resultados experimentales, comparación entre los resultados obtenidos en Matlab y los obtenidos en Android

Para observar el correcto funcionamiento del EKF implementado sobre plataforma Android se utilizan los datos obtenidos en una prueba experimental en la cual se almacenan los valores del Shimmer en la SD del mismo (exportados posteriormente con Consensys a default_exp_Session1_id9675_Calibrated_SD.dat), y se compara con el programa Analisis_Shimmer.m (realizado en Matlab por David Gualda).

Este programa extrae a partir del fichero .dat los valores de acelerómetros, giróscopos y *timestamp*, los procesa según el diagrama de flujo expuesto en la Figura 16 y plotea las señales *Roll*, *Pitch* y *Yaw* resultantes.

Para que la comparación entre ambos programas (en Matlab y Android) sea justa y realicen lo mismo, se vuelcan en Matlab las señales extraídas de acelerómetros, giróscopos y *timestamp* a un .txt. En concreto, se graban únicamente 76800 muestras de cada una de las señales de la prueba realizada, para que a la hora de procesar ese archivo en Android no se demore mucho. Se realiza un programa en Android (programa PDR) que lee esas señales del .txt, las procesa siguiendo el diagrama de flujo de la Figura 16 y posteriormente graba un archivo .txt (archivo_salida_PDR) en la Tablet con las señales Roll, Pitch y Yaw calculadas. Se coge ese archivo de la memoria del dispositivo, se exporta a Matlab (Comando_PDR) y se representan las señales.

A continuación, se muestran en la Figura 17 y las gráficas obtenidas para ambos casos (Matlab y Android):

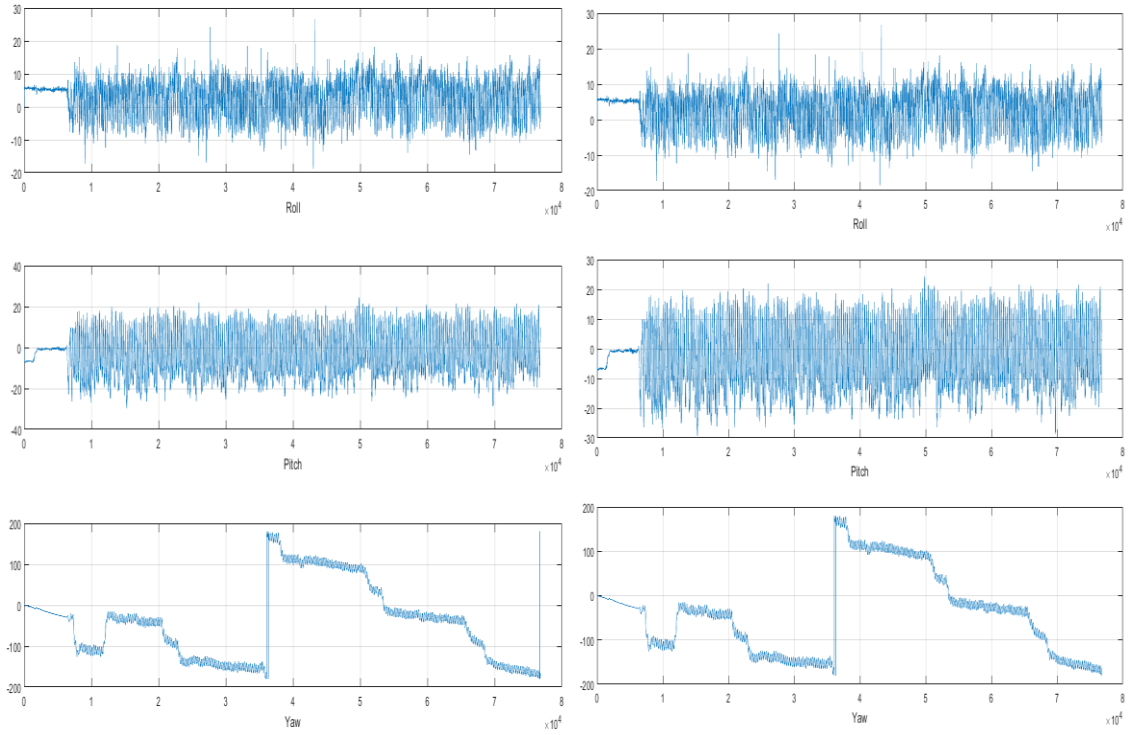

Figura 17: Comparación ángulos de Euler obtenidos en Matlab (izda) y Android (dcha).

Siendo la diferencia que se produce entre la señal Android y la de Matlab:

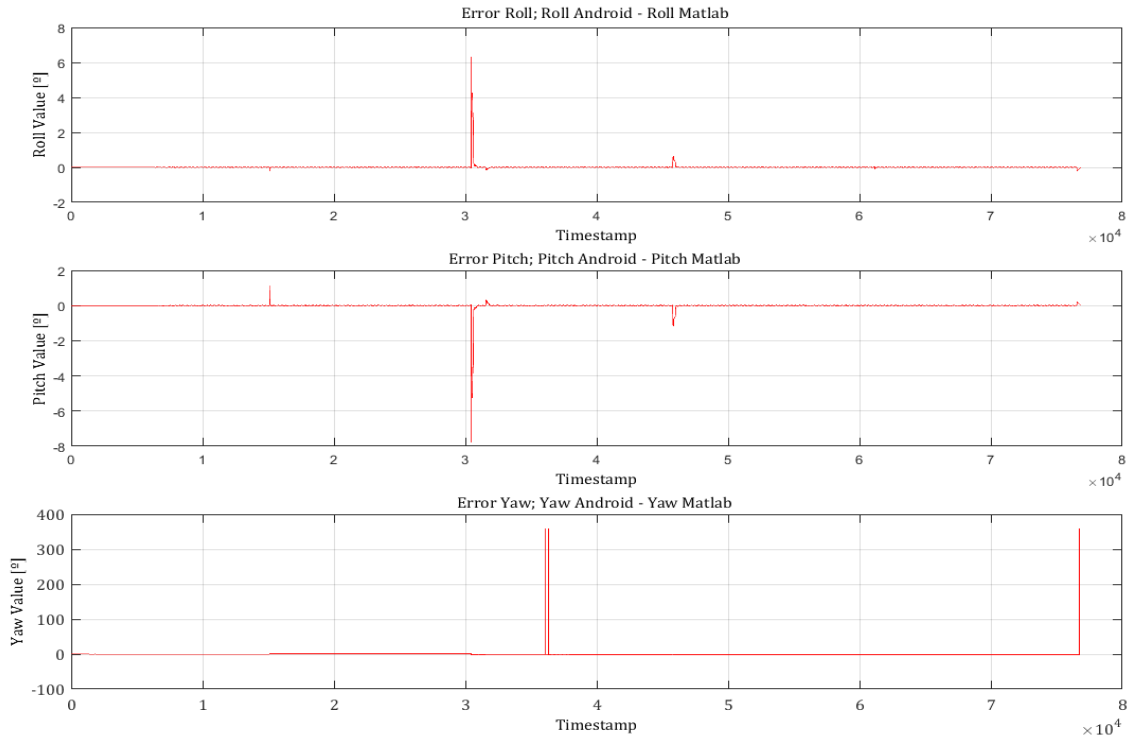

Figura 18: Diferencia ángulos de Euler obtenidos en Matlab y Android.

Como se puede observar, dicha diferencia, es prácticamente cero en toda la prueba, con lo que se verifica que el programa realiza lo mismo en las dos plataformas, siendo válida la implementación del algoritmo EKF en Android. Destacar que esa diferencia entre uno y otro se debe principalmente a la diferencia de criterios de números decimales que usan Matlab y Android (Android maneja en torno a algo más del triple de decimales que Matlab). Ese tema de los decimales ha podido derivar en que en una plataforma pudiera entrar a corregir (si $|Accel| \approx 9.8\, m/s^2$) y en la otra no. Aun así, no es de vital importancia, ya que como los dos siguen la misma tendencia, si no entra a corregir en una iteración por poco, corregirá en la siguiente, como se puede observar en la imagen de la Figura 18, que al momento el error vuelve a regularse en torno a 0.

Por último, se adjunta en la Figura 19 una captura de los tiempos de ejecución del KF en cada una de las iteraciones en Android, en la que se observa que el algoritmo es suficientemente rápido incluso en los peores casos, cuando realiza la etapa de corrección (en torno a 300 µs).

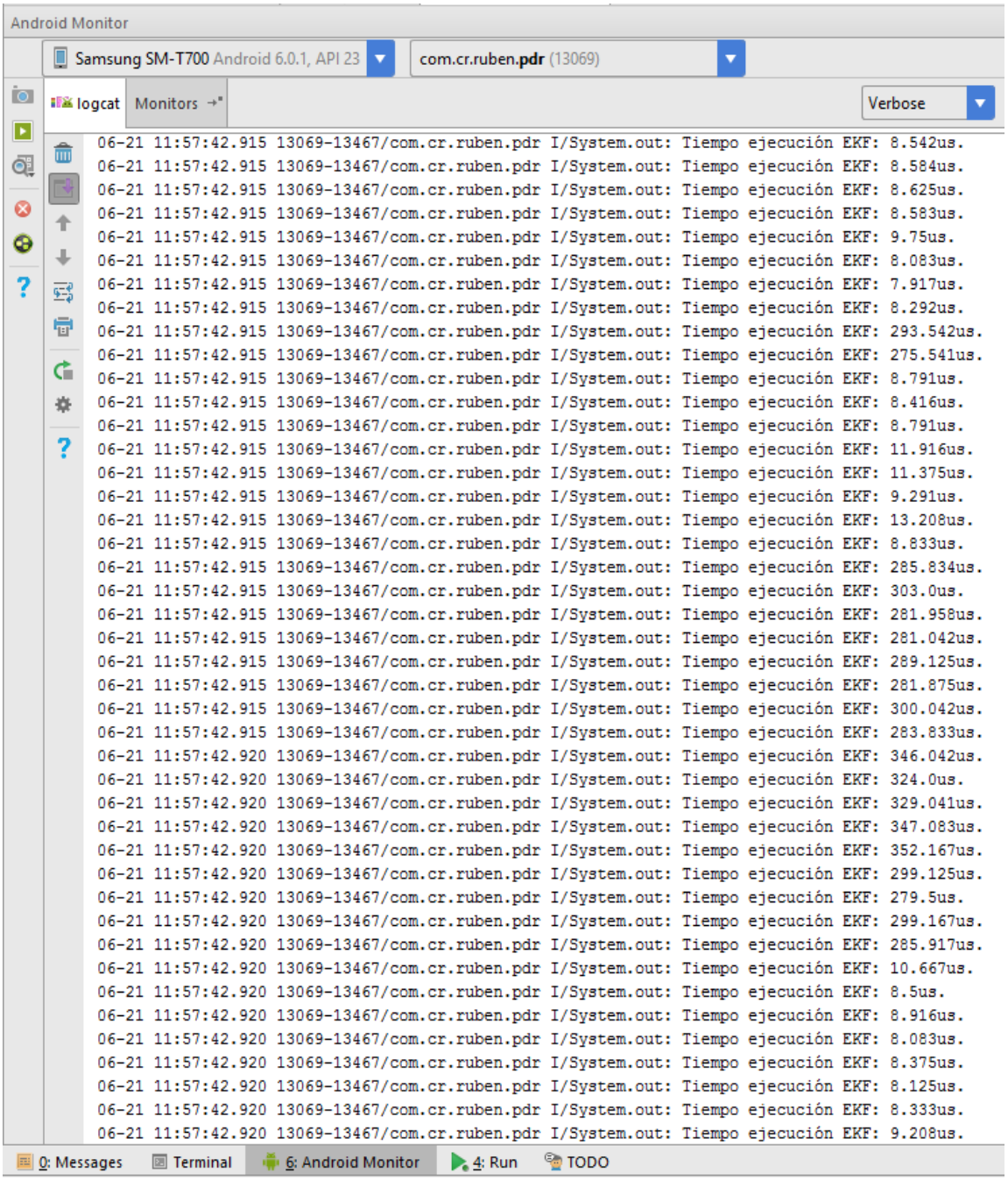

Figura 19: Tiempos de ejecución del KF en Android.

f. Programación del algoritmo de posicionamiento

El objetivo de este algoritmo es a partir de los valores calculados de los ángulos de Euler (*Roll* (ϕ), *Pitch* (⊙) y *Yaw* (Ψ)) obtener dos vectores de coordenadas x e y que contengan la posición relativa del usuario frente al punto inicial durante toda la trayectoria realizada. Para ello se ha desarrollado un algoritmo de posicionamiento basado en tres subsistemas, la detección de los pasos, la estimación de la anchura del paso y el algoritmo de posición relativa.

El algoritmo desarrollado para la detección de pasos se basa en la detección de los máximos y mínimos absolutos de la señal *Pitch*. Para la detección de los mismos, al comprobar que no se pueden usar directamente criterios como los expuestos en la literatura, de que la señal *Pitch* de un usuario cruza en cada paso por cero, se ha tenido que realizar un procesado de la señal para poder cumplir esos requisitos.

El procesado de la señal *Pitch* consiste en calcular el promediado de los últimos 101 puntos de la señal. Se coge dicho valor para que sea un valor próximo a la frecuencia utilizada (128Hz) y con el cual se tenga suficientemente información para hacer un buen promediado de la señal, y no demasiada ya que supondría un esfuerzo computacional innecesario. En cada iteración (a partir de la 101) se calcula la diferencia entre la señal *Pitch* promediado y la señal *Pitch* calculada (a esa señal resultante se le ha llamado *Pitch* procesado). Se aplica el algoritmo de cruces por cero, que consiste en ir almacenando los valores de los máximos y mínimos de la señal (y su posición en el tiempo) y cuando se detecten dos cruces por cero, quiere decir que se ha producido un periodo de la señal, por lo que en ese momento se almacenan los valores del máximo y mínimo obtenidos hasta ese instante, y se inicializan dichas variables (el máximo a menos infinito y el mínimo a más infinito) con ciertos umbrales marcados para evitar falsos máximos y mínimos, a esa señal *Pitch* procesada.

A partir de pruebas experimentales se observa que entre dos máximos/mínimos de la señal *Pitch* procesada (véase la Figura 20), se encuentra un máximo/mínimo de la señal *Pitch* original. Por lo que cuando se detecte un periodo de la señal *Pitch* procesada (dos cruces por cero), se irá recorriendo, entre la posición del máximo/mínimo actual y el máximo/mínimo anterior obtenidos de la señal *Pitch*

procesada, en la señal *Pitch* original y obteniendo el valor máximo/mínimo en ese intervalo de posiciones. Destacar que al necesitar dos máximos/mínimos de la señal *Pitch* procesada y al ir dicha señal retrasada respecto a la señal *Pitch* original debido al promediado realizado, se pierde el primer máximo/mínimo de la señal *Pitch*, lo que conlleva a perder el primer paso realizado, algo totalmente aceptable para un algoritmo PDR.

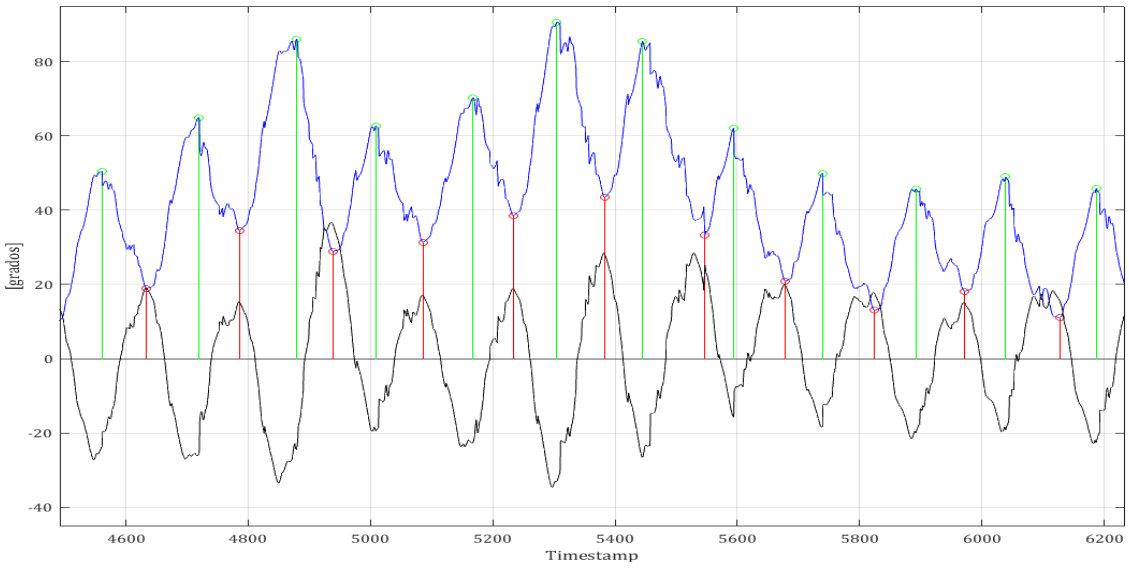

Figura 20: Captura de la señal Pitch original (azul), con máximos (verde) y mínimos (rojo), y la señal Pitch procesada (negra).

Para finalizar con la parte de programación del algoritmo de detección de pasos, se ilustra en la Figura 21 el diagrama de flujo de dicha parte del programa.

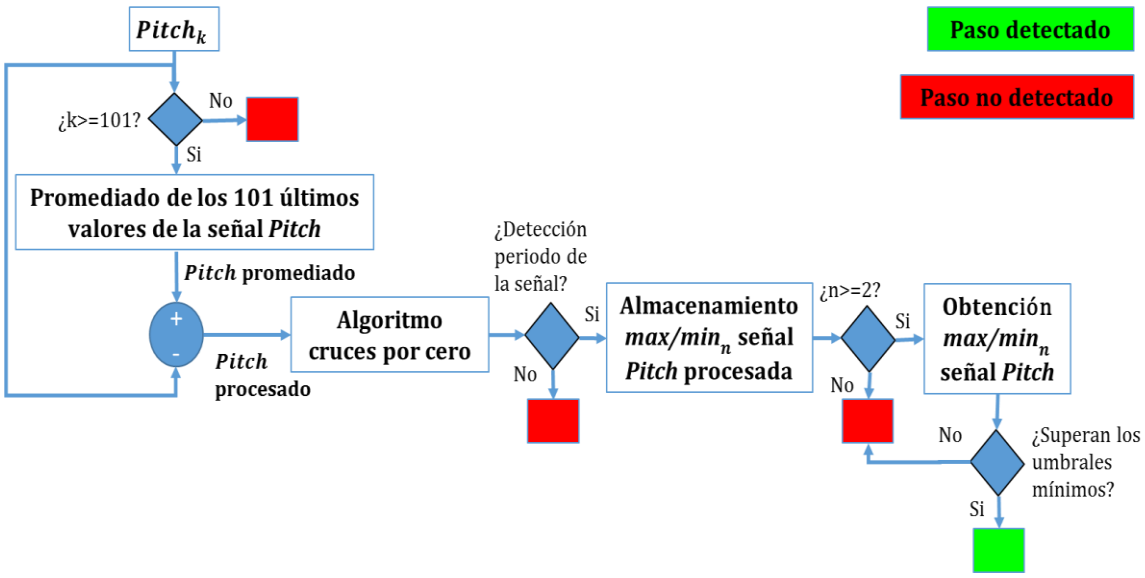

Figura 21: Diagrama de flujo del algoritmo de detección de pasos.

La programación de la parte de la obtención de la anchura de paso (destacar que como paso se están tomando únicamente los de la pierna del sensor) se resuelve implementando la ecuación (2.22), siendo los parámetros universales (a_H y b_H, 0.0294 y 0.232 respectivamente) los obtenidos a partir de la regresión lineal universal de la Figura 12.

Por último, para obtener la posición relativa se ha implementado la ecuación (2.23) explicada en el apartado de base teórica, de la cual se obtiene la estimación tanto de la posición relativa en x ($\hat{x}_k$), como en y ($\hat{y}_k$). Se ha creado un botón en la pantalla principal, para que una vez finalizado el experimento se cree un fichero llamado x_y.txt con la posición relativa del usuario y el usuario pueda observar en dicho fichero de texto los valores x e y obtenidos en su prueba. Para realizar el enlace entre la aplicación y un archivo almacenado en el sistema, se ha utilizado un *intent* (objeto que enlaza la actividad *MainActivity* y la del fichero .txt) de un esquema de datos URI (*Uniform Resource Identifier*), que contiene la URN (dirección) del archivo en cuestión.

Para dispositivos de versiones Android 7.0 o superiores, no se permite utilizar URI, por lo que para solucionar este problema en dispositivos que tengan esta versión lo que se ha hecho es conseguir que el sistema operativo (VM) ignore a los URI, pudiendo así ejecutarlos correctamente sin que provoquen error. Esto se ha realizado con un *Strict Mode VM Policy Builder* en el método *onCreate*().

g. Programación de la visualización de la posición relativa

El objetivo de esta sección es a partir de los valores relativos de la posición tanto del eje x como del y, representar en Google Maps la trayectoria realizada por el usuario en el periodo de tiempo en el que se haya realizado el experimento (desde que pulse el botón *Start Test*, hasta que presione el de *End Test*).

Para ello se ha creado un botón en la pantalla principal, al cual se le ha dado la funcionalidad para poder enlazar con el mapa de Google Maps y representar en él la trayectoria realizada, a partir de un punto de inicio conocido (X0, Y0). Este botón se habilitará cuando el usuario pulse el botón de finalizar la prueba, el resto de tiempo permanecerá bloqueado.

Esta parte se ha realizado junto con un compañero del laboratorio, Sergio Matas, aprovechando un trabajo previo suyo, en el cual transforma las coordenadas x e y obtenidas en la posición relativa, a coordenadas UTM (sistema de coordenadas universal transversal de Mercator).

Una vez se tienen las coordenadas UTM, se convierten a formato latitud longitud mediante la librería Jcoord, ya que este formato es el que usa Google Maps. Cuando tenemos los dos arrays de latitud y longitud, se hace un *intent* para pasar a la *MapsActivity* (para la realización de la cual se han tenido que solicitar las credenciales pertinentes a Google), en la cual se representará con un marcador cada una de las posiciones relativas obtenidas con cada paso.

h. Diagrama de flujo de la aplicación

Para finalizar con el apartado del desarrollo de la aplicación se expone el diagrama de flujo de la app, en el que se entrelazan los diferentes bloques de los que se ha hablado en esta sección.

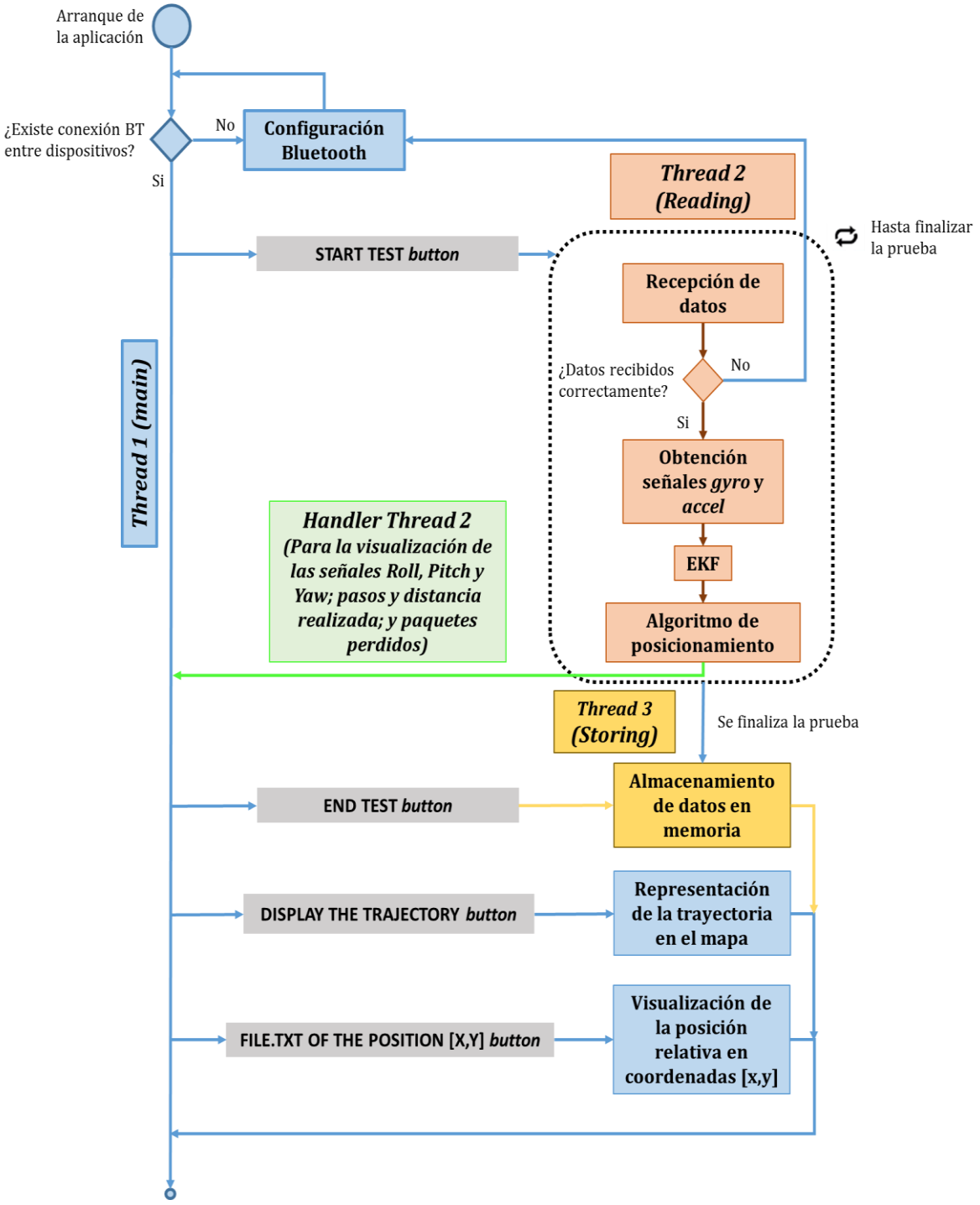

Figura 22: Diagrama de flujo de la aplicación.

En este diagrama se diferencia entre las tareas realizadas por los tres hilos principales del programa (se utiliza un cuarto hilo (*Sending*) para el envío de información al Shimmer). Dichos hilos son los siguientes:

- **Hilo *main***, representado en azul, hilo que más recursos dispone, encargado de atender los eventos de los distintos componentes. Es decir, este hilo ejecuta los métodos onCreate(), etc, es decir controla el interfaz de usuario.
  Todas las actividades y servicios de una aplicación son ejecutados por el hilo principal. Cuando la aplicación ha de realizar trabajo intensivo como respuesta a una interacción de usuario, hay que tener cuidado porque es posible que la aplicación no responda de forma adecuada. Por ejemplo, en nuestro caso, al acabar la prueba (cuando el usuario pulse el botón ***END TEST***) se ha programado que se almacenen en la memoria del teléfono diferentes datos de la prueba realizada en ficheros de texto. Si la creación y escritura de ficheros se hiciera desde el hilo de ejecución principal este quedaría bloqueado a la espera de que termine el proceso, sin poder utilizar el resto de funcionalidades de la aplicación. Aunque pudiera darse el caso de que el proceso sea tan rápido que no dé la sensación al usuario de que la aplicación se ha bloqueado, para asegurarnos se han creado una serie de hilos que realicen ese tipo de trabajos. Además, como se expuso en apartados anteriores, si el hilo principal es bloqueado más de 5 segundos, el sistema mostrará un cuadro de dialogo al usuario "La aplicación no responde" para que el usuario decida si quieres esperar o detener la aplicación, algo que se debe evitar de todos modos en el desarrollo de la aplicación.
- **Hilo *Reading***, representado en marrón, se encarga de los cálculos. Se ha realizado en él toda la algoritmia (desde la recepción de información proporcionada por el Shimmer hasta el cálculo de la posición relativa del usuario en el caso de que se haya detectado un nuevo paso) en un único hilo, ya que además de que nuestro proceso es secuencial (no se tienen cálculos paralelos que se pudieran ir haciendo de manera simultánea). En Android no se tiene la capacidad de administrar los recursos, es el propio sistema el que los otorga, por lo que hacerlo de forma secuencial y en único hilo es la única forma de asegurar que una vez que Android dé los recursos, se ejecuta de

seguido. Haciéndolo de forma segmentada, por ejemplo en dos partes; si a la hora de administrar recursos al acabar la primera parte del algoritmo y el sistema no le da paso a la parte restante del algoritmo y se lo da a cualquier otro proceso del teléfono (como una llamada entrante por ejemplo), retrasaría la ejecución del algoritmo restante y se podrían dar problemas con recibir datos nuevos por parte del Shimmer sin haber finalizado con los anteriores. Destacar que a este hilo se le ha programa un **manejador (handler)**, representado en verde, para poder representar en tiempo real en la pantalla diferentes datos obtenidos en la ejecución del hilo, como las señales *Roll*, *Pitch* y *Yaw*, los pasos y distancia realizada hasta el momento y el número de paquetes de información BT perdidos.

- **Hilo *Storing***, representado en dorado, encargado del almacenamiento de los datos de la prueba realizada en la memoria del dispositivo. Crea dos ficheros de texto; uno en el que almacena los valores procesados de acelerómetros y giróscopos a partir de los valores enviados por el Shimmer (este fichero se ha usado para estudiar a posteriori posibles fallos detectados en las pruebas y detectar el foco de error) y otro que almacena las coordenadas x e y de la posición relativa del usuario en la trayectoria descrita.

Destacar también que ***START TEST***, ***END TEST***, ***DISPLAY THE TRAJECTORY*** y ***FILE.TXT OF THE POSITION [X,Y]*** son los botones de la pantalla principal de la aplicación.

5. Pruebas experimentales finales

En esta sección se van a mostrar varias pruebas con la aplicación desarrollada para observar los resultados obtenidos y comprobar la fiabilidad del algoritmo implementado. La posición de la IMU para el correcto funcionamiento de la aplicación es la que se muestra en la siguiente Figura:

Figura 23: Posición de la IMU durante las pruebas.

El dispositivo Shimmer3 se conecta al dispositivo portable (Samsung Galaxy Tab S2) usando comunicación Bluetooth. La Tablet procesa en tiempo real la información inercial que proporciona el sensor y estima la posición relativa del usuario frente a un punto de inicio conocido (tanto posición como orientación). La aplicación muestra por pantalla en tiempo real las gráficas correspondientes a la orientación del sensor, los pasos realizados y la distancia recorrida (véase la Figura 25). Además si el usuario lo requiere puede observar la trayectoria realizada en google Maps (véase la Figura 26 o la Figura 28) y redireccionarse a un archivo .txt creado por la aplicación en la memoria interna del teléfono.

a. Descripción del entorno de pruebas y resultados

Para mostrar la robustez del algoritmo se han probado varias trayectorias y con diferentes personas. A continuación, se procede a explicar dos trayectorias completamente diferentes realizadas en el interior del edificio politécnico de la Universidad de Alcalá para observar la versatilidad del algoritmo propuesto.

- Trayectoria recta: prueba realizada caminando recto por un pasillo (véase la Figura 24).

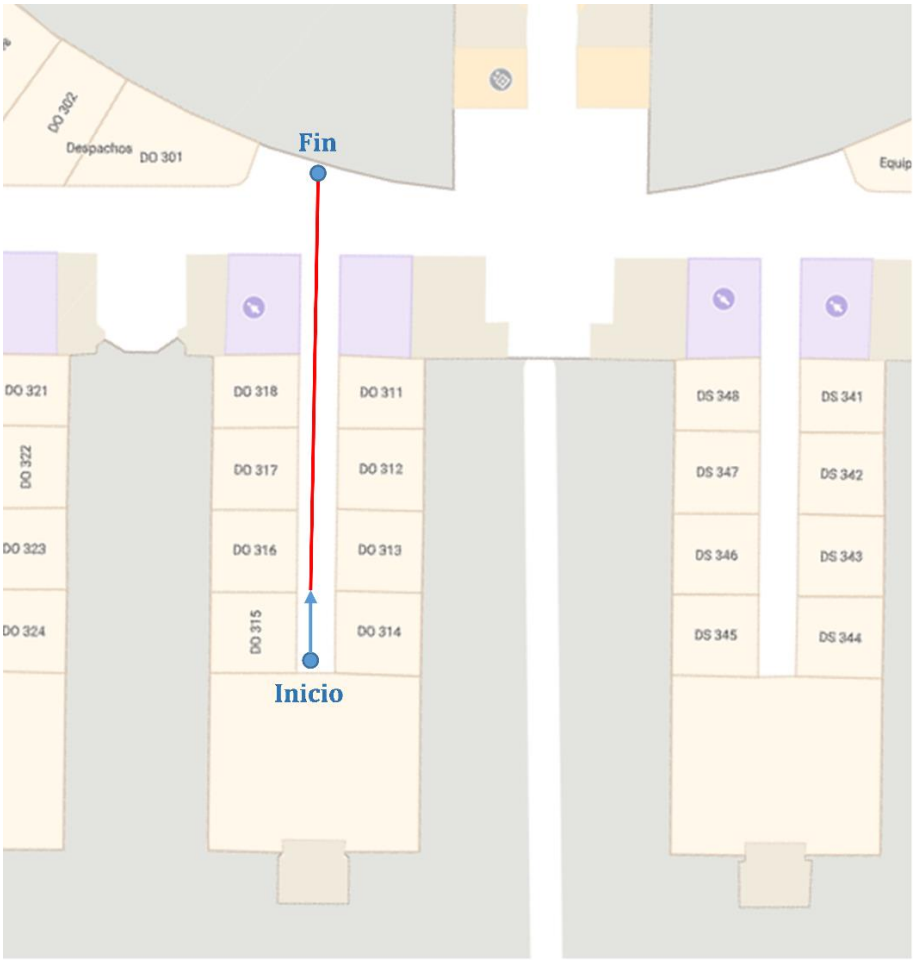

Figura 24: Trayectoria recta real realizada.

A continuación, se muestra la pantalla principal del programa tras la trayectoria realizada, como queda la representación de la posición relativa en Google Maps y la comparativa entre los valores obtenidos con la aplicación y los valores reales de la prueba realizada.

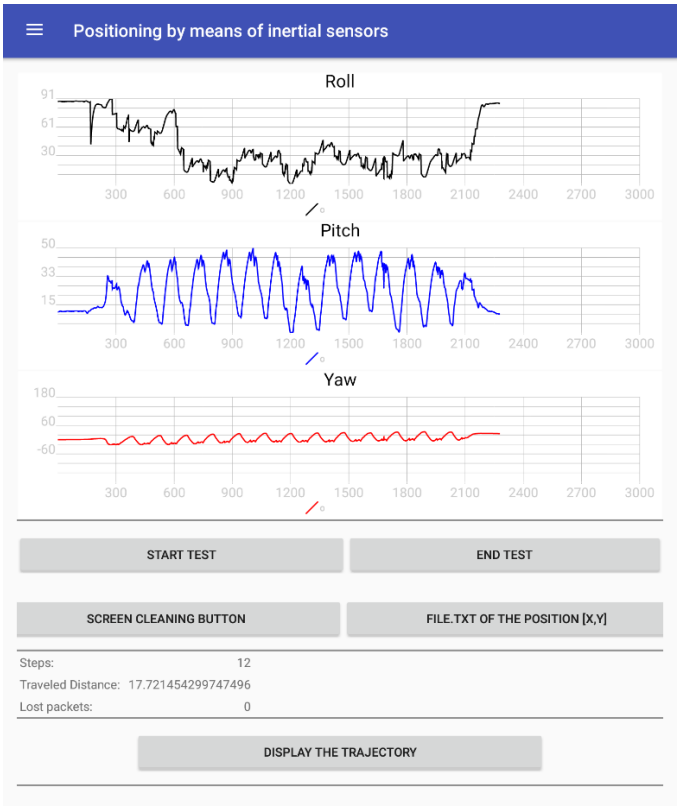

Figura 25: Pantalla principal de la aplicación.

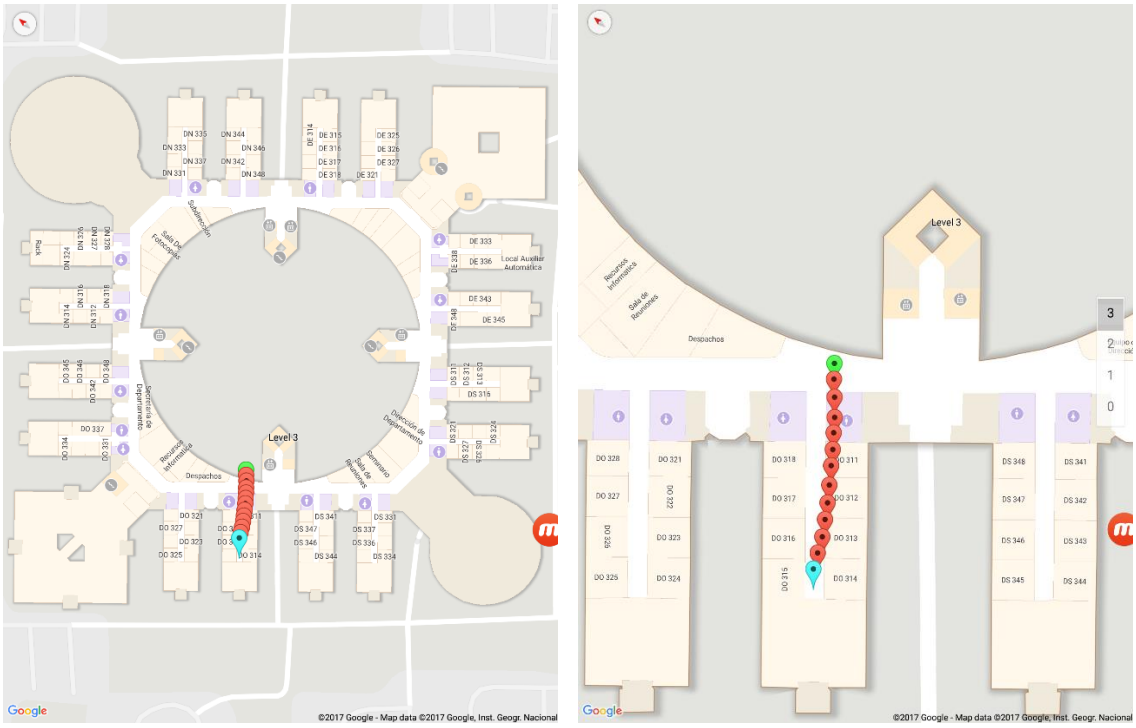

Figura 26: Trayectoria estimada con la aplicación.

Siendo los marcadores rojos los puntos intermedios de la trayectoria, el azul cian el punto de inicio de la prueba y el verde el final.

	Distancia recorrida	Pasos realizados
Valores reales	21.5 m	14
Valores obtenidos en la app	17.72 m	12
Error (%)	17.58%	14.28%
Error sin contar como tal el primer paso (%)	10.71%	7.14%

Tabla 2: Comparativa de la trayectoria recta.

Destacar que como se expresó en la sección del desarrollo de la app, el algoritmo desarrollado de por sí pierde el primer paso, por lo que aunque el error parezca muy grande, se debe principalmente a que la trayectoria realizada es de corta duración, por tanto la pérdida del primer paso tiene mucho peso. Aun así, se observa como la trayectoria estimada es muy similar a la real.

- Trayectoria circular: prueba realizada dando una vuelta por el interior del edificio (véase la Figura 27).

Figura 27: Trayectoria circular real realizada.

A continuación, se muestra la representación de la posición relativa en Google Maps y la comparativa entre los valores obtenidos con la aplicación y los reales.

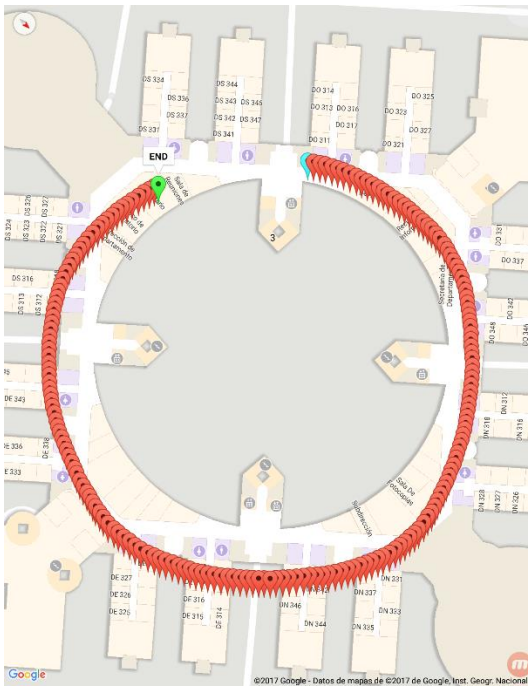

Figura 28: Trayectoria estimada con la aplicación.

Siendo los marcadores rojos los puntos intermedios de la trayectoria, el azul cian el punto de inicio de la prueba y el verde el final.

	Distancia recorrida	**Pasos realizados**
Valores reales	244.73 m	186
Valores obtenidos en la app	243.39 m	183
Error (%)	0.55%	1.61%

Tabla 3: Comparativa de la trayectoria circular.

En comparación con la otra trayectoria se puede observar como los errores son inferiores debido a que es una trayectoria mayor. Destacar también que al ser una trayectoria de mayor duración se obtiene un mayor error acumulativo, debido a que estamos trabajando únicamente con valores relativos.

Como conclusión, expresar que aun trabajando con valores relativos el algoritmo desarrollado presenta un resultado totalmente aceptable en diversos escenarios.

b. Tiempos de ejecución

Además de la comprobación del funcionamiento de los algoritmos desarrollados, es fundamental comprobar que el tiempo de ejecución de los mismos no sea un impedimento a la hora de ejecutar en tiempo real la aplicación. Es por ello, que durante las pruebas experimentales se haya ido midiendo lo que tardan en procesar la información las diferentes partes del programa (principalmente parte de obtención señales gyro y accel, EKF y algoritmo de posicionamiento, las de mayor carga computacional) en cada iteración de envío de datos del Shimmer. A continuación, se muestra una imagen del tiempo de ejecución de cinco iteraciones de una prueba realizada con la aplicación.

Figura 29: Tiempos de ejecución de los diferentes algoritmos en Android.

En esta imagen se observa como incluso en el caso más extremo en el que se tenga que ejecutar la etapa de corrección del EKF y la detección del paso, se requiere un tiempo en torno a los 5ms. Este tiempo es inferior a los 7.8125 ms que disponemos entre el envío de datos sucesivos del Shimmer, límite marcado por la frecuencia a la que se ha configurado el sensor (128Hz).

Por tanto, se cumple también este requerimiento, con lo que acabamos la verificación del código desarrollado.

6. Conclusiones y trabajos futuros

Este trabajo fin de grado tenía varios objetivos a alcanzar. Por un lado, el trabajo propuesto tenía como meta la implementación de un EKF sobre plataforma Android. Se ha realizado una clase en Android, que realiza la etapa de corrección, proceso uniforme en todos los filtros de este tipo, consiguiendo así tener una "función" estándar, que recibe unas matrices de entrada, las procesa y devuelve unas matrices de salida. Al utilizar esta forma de implementación se consigue una clase válida para diferentes aplicaciones, únicamente teniendo que variar las matrices de entrada introducidas a la clase.

Por otro lado, se pretendía la obtención de la posición relativa en tiempo real a partir de las medidas de los sensores inerciales de la IMU. Se ha desarrollado un algoritmo (probado en Matlab y una vez verificado, implementado en Android) que a partir de los ángulos de Euler (obtenidos mediante un EKF a partir de las medidas de los sensores inerciales de la IMU), pueda detectar los pasos realizados por el individuo, estimar dinámicamente la anchura de cada paso realizado y obtener la posición relativa del individuo tras la realización de cada paso.

Por último, se ha buscado la forma de visualizar la trayectoria realizada por el usuario una vez finalizada la prueba, para dar un extra al trabajo desarrollado, ya que todo el algoritmo desarrollado proporcionaba los valores de los pasos y distancia realizada y un fichero .txt en el que se mostraban las posiciones x e y estimadas, pero al no visualizarlo, no era una información de gran utilidad para la observación de la trayectoria realizada. Se ha conseguido la forma de representar dicha trayectoria en los mapas de Google, consiguiendo así una representación en un mapa que ofrece la posibilidad de hacer acercamientos o alejamientos para mostrar en el mapa, además de mucha información adicional sobre el entorno de trabajo. Este logro ha sido de vital importancia ya que aunque la mayor parte de este TFG reside en la algoritmia, disponer de una interfaz gráfica en la que representar los resultados de trayectoria puede resultar de gran utilidad y atractivo para el usuario final. A lo largo de este trabajo se ha intentado cuidar dicha interfaz, ya que de cara a la exposición del producto es un elemento diferenciador.

En definitiva, se ha desarrollado una aplicación llamada LOCATE-US_PDR la cual es capaz de obtener y visualizar la posición relativa del usuario durante un tiempo determinado por el mismo en cualquier entorno de trabajo, ya sea exterior (aunque no tendría mucho sentido utilizar el posicionamiento relativo a partir de medidas inerciales teniendo otros elementos más desarrollados en esa parcela como el GPS) como interior, siempre y cuando las trayectorias se realicen sobre superficies horizontales, cumpliendo con el objetivo principal del proyecto en el que se enmarca este trabajo.

Como trabajo futuro, quedaría fusionar esta parte de posicionamiento a partir de los sensores inerciales de la IMU con el posicionamiento ultrasónico (ya desarrollado) mediante un EKF (se usará la clase implementada en este TFG); la programación para que funcione en modo US-IMU cuando tenga cobertura de las U-LPS y en modo IMU cuando no la tenga; la mejora de la localización gracias al desarrollo de algoritmia de alto nivel para la mejora del posicionamiento, técnica *map matching*, que es una técnica que combina el mapa electrónico con la información de localización para conseguir la posición real del individuo, evitando posibles trayectorias que pudieran llevar a la impresión en el mapa de que por ejemplo se traspasen paredes; y el diseño final de la interfaz de la aplicación.

7. Planos

En esta sección se van a mostrar algunos extractos de las partes más importantes del código desarrollado: la extracción de los valores del sistema inercial mediante EKF y el algoritmo de posicionamiento.

- **Extracción de los valores del sistema inercial mediante EKF**

Esta parte se centrará en la función en Matlab y la clase Android creadas para realizar la etapa de corrección (resto de código del EKF explicado en el Capítulo 4, apartado e) con el objetivo de conseguir una función estándar del EKF en ambas plataformas que se pueda usar para diferentes ámbitos únicamente cambiando las matrices de entrada del filtro.

En Matlab se ha creado una función cuyo código es el siguiente:

```
function [X P] = Filtro_Kalman_Extendido(X_,P,A,H,Z,h_X_,Q,R)
%FILTRO_KALMAN_EXTENDIDO

% X_: estado a priori
% P: Matriz de covarianza del error
% A: Jacobiano ecuación transición
% H: Jacobiano ecuación de medida
% Z: Observaciones
% h_X_: Estimación de las observaciones a partir del estado a priori
% Q: Matriz covarianza error ecuacion de transición
% R: Matriz covarianza error ecuación de medida

% Devuelve el estado actualizado X y la matriz de covarianza del
error P

        P_=A*P*A' + Q;
        S=H*P_*H' + R;
        K=P_*H'*(S^(-1));
        V=Z-h_X_;
        X=X_+K*V;
        P=P_- K*S*K';
end
```

Esta función se podrá utilizar en cualquier programa de dicha plataforma haciendo una llamada de la siguiente forma:

```
%Filtro EKF
[X P]=Filtro_Kalman_Extendido(X_,P,A,H,Z,h_X_,Q,R);
```

Siendo X_, P, A, H, Z, h_X_, Q y R matrices conocidas.

Para la implementación sobre plataforma Android se ha utilizado una clase, llamada KalmanFilter, que contiene en su interior un constructor, método con el mismo nombre de la clase en el cual se inicializan las variables de la clase con las introducidas en la misma; un método, llamado Filtro_Kalman_Extendido, en el cual se realizan las operaciones correspondientes a la etapa de corrección del EKF (para poder realizar las operaciones con matrices se ha tenido que utilizar una librería externa a Android Studio llamada *Apache Commons Math 3.4.1* y más en concreto su interfaz RealMatrix); y dos métodos getter para poder devolver las variables calculadas en dicha clase.

El código desarrollado es el siguiente:

```
package com.cr.ruben.myapplication_pdr;

import org.apache.commons.math3.linear.LUDecomposition;
import org.apache.commons.math3.linear.MatrixUtils;
import org.apache.commons.math3.linear.RealMatrix;

/*
Clase para el Filtro de Kalman
*/

public class KalmanFilter {

    /*Variables externas introducidas.
      No introducimos dimensiones, ya que son matrices dinamicas*/
    public double[][] X_ , P , A , H , Z , h_X_ , Q , R , X ;

    public KalmanFilter(double[][] X_r, double[][] Pr,
double[][] Ar, double[][] Hr, double[][] Zr, double[][] h_X_r,
double[][] Qr, double[][] Rr){

        /*Eliminamos las dimensiones de las matrices, para que no
        haya problemas con posibles valores anteriores*/
        X_ = null;
        P = null;
        A = null;
        H = null;
        Z = null;
        h_X_ = null;
        Q = null;
        R = null;
        X = null;
```

```
        //Asignamos las dimensiones, segun variables introducidas
        X_ = new double[X_r.length][X_r[0].length];
            /*NOMBREMATRIZ.length -> nºfilas;
            NOMBREMATRIZ[0].length ->nºcolumnas*/
        P = new double[Pr.length][Pr[0].length];
        A = new double[Ar.length][Ar[0].length];
        H = new double[Hr.length][Hr[0].length];
        Z = new double[Zr.length][Zr[0].length];
        h_X_ = new double[h_X_r.length][h_X_r[0].length];
        Q = new double[Qr.length][Qr[0].length];
        R = new double[Rr.length][Rr[0].length];
        X = new double[X_r.length][X_r[0].length];

        /*Asignamos los valores de las variables introducidas a las
        matrices de nuestra clase*/
              X_ = X_r;
              P = Pr;
              A = Ar;
              H = Hr;
              Z = Zr;
              h_X_ = h_X_r;
              Q = Qr;
              R = Rr;

        /*
        Siendo:
            X_: estado a priori ; []mx1
            P: Matriz de covarianza del error; []mxm
            A: Jacobiano ecuación transición; []mxm
            H: Jacobiano ecuación de medida; []nxm
            Z: Observaciones; []nx1
            h_X_: Estimación de las observaciones a partir del
            estado a priori; []nx1
            Q: Matriz covarianza error ecuacion de transición; []mxm
            R: Matriz covarianza error ecuación de medida; []nxn
            X : variables de estado ; []mx1
        */

    }

    public void Filtro_Kalman_Extendido(){

        //Asociamos matrices ya creadas a Real Matrix
        RealMatrix mX_ = MatrixUtils.createRealMatrix(X_);
        RealMatrix mP = MatrixUtils.createRealMatrix(P);
        RealMatrix mA = MatrixUtils.createRealMatrix(A);
        RealMatrix mH = MatrixUtils.createRealMatrix(H);
        RealMatrix mZ = MatrixUtils.createRealMatrix(Z);
        RealMatrix mh_X = MatrixUtils.createRealMatrix(h_X_);
        RealMatrix mQ = MatrixUtils.createRealMatrix(Q);
        RealMatrix mR = MatrixUtils.createRealMatrix(R);

        //Matrices intermedias
        RealMatrix mAMatrixT = mA.transpose();
        RealMatrix mHMatrixT = mH.transpose();
```

```
        //P_ = A*P*A' + Q;
        RealMatrix mP_ =mA.multiply(mP).multiply(mAMatrixT).add(mQ);

        //S = H*P_*H' + R;
        RealMatrix mS =mH.multiply(mP_).multiply(mHMatrixT).add(mR);

        RealMatrix mSMatrixInv = new
        LUDecomposition(mS).getSolver().getInverse(); //S^(-1)
        //K = P_*H'*(S^(-1));
        RealMatrix mK =
        mP_.multiply(mHMatrixT).multiply(mSMatrixInv);

        //V = Z-h_X_;
        RealMatrix mV = mZ.subtract(mh_X);

        //X = X_+K*V;
        RealMatrix mX = mX_.add(mK.multiply(mV));

        RealMatrix mKMatrixT = mK.transpose(); //K^(T)
        //P = P_- K*S*K';
        mP = mP_.subtract(mK.multiply(mS).multiply(mKMatrixT));

        /*Pasamos las matrices que queremos devolver, X,P, de tipo
        RealMatrix a double*/
        X = mX.getData();
        P = mP.getData();

    }

    public double[][] getX() {
        return X;
    }

    public double[][] getP() {
        return P;
    }

}
```

Esta clase se podrá utilizar en cualquier activity de un programa de dicha plataforma de la siguiente forma: primero se asocia un objeto nuevo a la clase, definiéndole las variables ya conocidas, X_, P, A, H, Z, h_X_, Q y Rr (matriz R); a posteriori se realiza a ese objeto las operaciones correspondientes al EKF con el método Filtro_Kalman_Extendido; y por último se recuperan de la clase las variables de interés para el resto de código. Los comandos que habría que utilizar se expresan a continuación:

```
//Filtro EKF
KalmanFilter filtro = new KalmanFilter(X_, P, A, H, Z, h_X_, Q, Rr);
filtro.Filtro_Kalman_Extendido();
X = filtro.getX();
P = filtro.getP();
```

- **Algoritmo de posicionamiento**

Se procede a adjuntar el código desarrollado en el presente TFG para resolver el problema de la obtención de la posición a partir de la detección de los pasos en la señal Pitch (véase la Figura 21 de forma que sirva de guía del mismo), calculada a partir de los valores inerciales de la IMU, ampliando lo explicado en el Capítulo 4, apartado f.

Este código se ha programado en Matlab dentro de un bucle en un script y en Android se ha introducido dentro del hilo que lee los valores procedentes del Shimmer, por lo que su forma en ambas plataformas no varía mucho como sí ocurría en el EKF al usar una función aparte. De ahí que solo se adjunte el código correspondiente a Matlab debido a la semejanza de los mismos.

```
%Aqui ya tendríamos los ángulos de Euler del Shimmer en [°].
%Por tanto a partir de aquí aplicamos, algoritmo de posicionamiento:
detección de pasos, estimación anchura de paso y posición relativa

%%%%%%%%%%%%%%Detección De Pasos%%%%%%%%%%%%%%%%%%%%%

if i>=101
      Pitche_promediado(i,1)=mean(Pitche(i-100:i));
      Pitche_procesado(i,1)=Pitche_promediado(i,1)-Pitche(i,1);

      if Pitche_procesado(i-1,1)<0 && Pitche_procesado(i,1)>0
      %cruce por cero, direccion ascendente
            zero=zero+1;
            up=1;
            down=0;
      end

      if  Pitche_procesado(i-1,1)>0 && Pitche_procesado(i,1)<0
      %cruce por cero, direccion descendente
            if semaphore_inicial>=1
                zero=zero+1;
            else
                semaphore_inicial=1;
            % semaphore_inicial=0 para conseguir que inicialmente
            detecte maximo del pitche, minimo del pitche_promediado
            end
            down=1;
            up=0;
      end

      if up==1 && Pitche_procesado(i,1)>maximo_promediado
            maximo_promediado= Pitche_procesado(i,1);
            posmax_promediado=i;
      end
```

```
if down==1 && Pitche_procesado(i,1)<minimo_promediado
      minimo_promediado= Pitche_procesado(i,1);
      posmin_promediado=i;
end

if zero==2 %Dos cruce por cero
       if abs(maximo_promediado-minimo_promediado)>25

          if maximo_promediado>7 && minimo_promediado<-7
              contador=contador+1;
              contador_Array=contador_Array+1;
              Array_posmax(contador_Array)=posmax_promediado;
              Array_posmin(contador_Array)=posmin_promediado;

              if contador>=2
                  for j=Array_posmax(contador_Array-1): 1:
                      Array_posmax(contador_Array)
                      if Pitche(j)>maximo
                          maximo=Pitche(j);
                          posmax=j;
                      end
                  end

                  minimo=maximo;

                  for j=Array_posmin(contador_Array-1):1:
                      Array_posmin(contador_Array)
                      if Pitche(j)<minimo
                          minimo=Pitche(j);
                          posmin=j;
                      end
                  end

                 Array_posmax(1)=Array_posmax(contador_Array)
                 Array_posmin(1)=Array_posmin(contador_Array)
                  contador_Array=1;
                  stem(posmax,maximo,'g');
                  stem(posmin,minimo,'r');
                  step=step+1;
                  Pitch_amplitude(step,1)= maximo-minimo;

                 %%%%%%%%%%%%Anchura De Paso, SL%%%%%%%%%%%%
                  SL(step)=ah*Pitch_amplitude(step,1)+bh;
                  distancia=distancia+SL(step);

                  %%%%%%Posicion relativa%%%%%%%%%%%%%%%%
                  YAW(step+1)= Yawe(posmax,1)*pi/180;
                 %angulos en rad, para usar sin, cos
                  attitude= O(step) +YAW(step+1) -YAW(step);

                  x(step+1)=x(step) + SL(step)*cos(attitude);
                 %posicion x
                  y(step+1)=y(step) + SL(step)*sin(attitude);
                 %posicion y
                  O(step+1)=attitude; %Orientación en radianes

                  maximo=-inf;
              end
```

```
                    minimo_promediado=inf;
                    maximo_promediado=-inf;

                elseif maximo_promediado<7
                    maximo_promediado=-inf;
                elseif minimo_promediado>-7
                    minimo_promediado=inf;
                end

            end

            zero=0;

        end

end
```

8. Pliego de condiciones

El presente pliego de condiciones tiene por objeto definir las condiciones para las que se ha creado la aplicación, no asegurando el correcto funcionamiento de la misma si no se siguen dichas condiciones. Las condiciones para las cuales se ha originado la app se describen a continuación.

- Aplicación diseñada para smartphones Android de versión 4.0.3 o superior.
- Programación de los algoritmos de extracción de valores para las características de envío de la IMU Shimmer3 y para la configuración del Shimmer expuesta en la sección de desarrollo de la aplicación. Si se usara otra IMU habría que modificar esa parte, una vez se tuvieran señales acelerómetros y giróscopos se procesarían igual que lo hecho en este trabajo. De hecho, actualmente se está empezando a trabajar con los algoritmos desarrollados con otra IMU de mayor precisión, funcionando sin ningún problema.
- La IMU debe estar bien sujeta a la pierna, alineada respecto a la orientación del pie y que no se mueva durante las pruebas (como se muestra en la Figura 23). Si se coloca de forma incorrecta, los valores de los acelerómetros y giróscopos no serán los deseados, obteniendo resultados que distan mucho de lo correcto. Esta condición es fundamental, ya que la IMU utilizada es muy sensible (sobre todo es sensible la señal *Yaw* obtenida a partir de sus sensores inerciales, señal que nos da información sobre el desplazamiento lateral de la pierna), por lo que una desviación de la misma produciría un error considerable en el cálculo de la trayectoria realizada.
- Se necesita dar permiso al Bluetooth del dispositivo en el que se ha cargado la app, ya que toda la información del sensor se recibe vía BT. Se solicitará permiso al usuario a través de un diálogo como el mostrado en la Figura 14.
- Se precisa acceso a Internet a la hora de representar la trayectoria en Google Maps, ya que sin Internet no se cargan los mapas. Aunque se podría programar automáticamente a encender el wifi del usuario sin su permiso, para tratar de ser lo más cuidadosos posibles con la intimidad del usuario, se ha creado una alerta que sale al pulsar el botón ***DISPLAY TRAJECTORY*** y no tener Internet activado, que pide permiso al individuo para que autorice la

activación del wifi. Dicho diálogo de alerta es como el mostrado en la Figura X, expuesta a continuación:

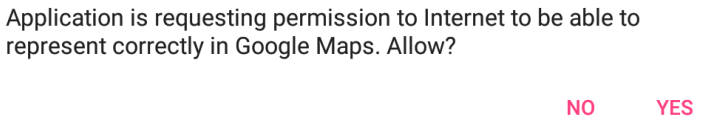

Figura 30: Permiso al Wifi del dispositivo portable.

Cabe destacar, que en el caso de que el usuario no permita la conexión a Internet, no se representa la trayectoria.

- Se requiere permiso a la memoria del teléfono, ya que cada vez que el usuario acaba una trayectoria, se generan un archivo .txt en el directorio externo de la memoria interna del dispositivo (*/storage/emulated/0/*), que es al que luego se direccionará desde la app en el caso de pulsar el botón ***FILE. TXT OF THE POSITION [X, Y]***. Se solicitará permiso al usuario a través de un diálogo como el mostrado a continuación:

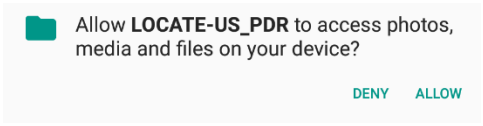

Figura 31: Permiso a la memoria del dispositivo portable.

- Se necesita precisar la localización inicial del usuario al iniciar la prueba (tanto posición como orientación a la que se va a empezar a caminar) para poder representar correctamente la trayectoria en el mapa. Actualmente está posición inicial se introduce en el código del programa Android antes de cargarlo en las variables X0, Y0 en formato UTM. Para obtener las coordenadas UTM a partir de una localización del mapa Google se ha utilizado la siguiente página web: http://boulter.com/gps/#40.51293786431562%2C-3.3499702624976635 La orientación inicial hacia la que se realiza la trayectoria se ha ido variando, cambiando la posición en la que se visualizan los marcadores en el mapa en función de si se utiliza un tipo de trayectoria u otro.

9. Presupuesto

El presupuesto que se va a exponer, hace referencia tanto a los costes directos debidos a las horas de trabajo como programador (el propio alumno) como a los costes del material necesario para su desarrollo. Sin embargo, no se han tenido en cuenta los tiempos de formación e investigación para el desarrollo del mismo, ni el empleado en la redacción del mismo. Dicho presupuesto del proyecto resulta sólo orientativo y no está pensado en vistas a una posible comercialización y por tanto amortización de la herramienta creada. En este caso la aplicación desarrollada, satisface una necesidad planteada por el Departamento de Electrónica de la Universidad de Alcalá de Henares en el seno del proyecto LOCATE-US.

a. Coste del material utilizado

En esta sección se muestran los costes debidos a los elementos físicos utilizados en este proyecto.

Equipo	Precio (€)	Periodo amortización (años)	Usos (meses)	Coste de amortización (€)
Shimmer3 (IMU)	359	3	6	59.83
Tablet Samsung Galaxy Tab S2	425	3	6	70.83
PC Intel i7, 3GB RAM	800	3	6	133.33
Total	**1584**	**-**	**-**	**264**

Tabla 4: Coste del material utilizado.

b. Costes directos de la aplicación

Para obtener los costes de programación (tanto en Android como en Matlab), se ha asumido un coste estándar como "programador" de 50 €/h. El desglose de las horas de programación se ha dividido en los grandes bloques del proyecto, englobando en cada uno de ellos desde su creación hasta su funcionamiento, y se muestra en la siguiente tabla:

Bloque del proyecto	Tiempo empleado (horas)	Coste (€)
Filtro EKF	50	2500
Algoritmo de posicionamiento	200	10000
Visualización de la posición relativa	20	1000
Batería de pruebas	100	5000
Diseño del interfaz	30	1500
	Total	**20000**

Tabla 5: Costes directos de la aplicación.

c. Costes indirectos de la aplicación

En este apartado se reflejan los costes indirectos, entendiendo por estos costes como los que suponen las licencias de los programas necesarios para el desarrollo del proyecto.

Licencia	Coste (€)	Periodo amortización (años)	Usos (meses)	Coste de amortización (€)
Matlab r2016b	2000	2	6	500
Android Studio 2.3	0	2	6	0
Microsoft Office 2013	450	2	6	112.5
Consensys	0	2	6	0
Total	**2450**	-	-	**612.5**

Tabla 6: Costes indirectos de la aplicación

d. Coste del proyecto

Concepto	Coste íntegro del proyecto (€)	Coste de amortización (€)
Coste del material utilizado	1584	264
Costes directos de la aplicación	20000	20000
Costes indirectos de la aplicación	2450	612.5
Total (sin IVA)	**24034**	**20876.5**
Total (con IVA del 21%)	**29081.14**	**25260.57**

Tabla 7: Coste del proyecto.

Por tanto, el importe estimado del proyecto en la duración del mismo asciende a la cantidad de:

Veinticinco mil doscientos sesenta euros y cincuenta y siete céntimos

Guadalajara, a 21 de Julio de 2017.

Firmado: Rubén Cervigón Rey

Ingeniero Industrial.

10. Manual de usuario

a. Descripción de la aplicación

La aplicación **LOCATE-US_PDR** es una aplicación para smartphones de versión 4.0.3 o superior (a día de hoy el 97.4% de los dispositivos utilizados) que permite obtener y visualizar en Google Maps la trayectoria relativa del usuario durante un tiempo determinado por el mismo en cualquier entorno de trabajo.

b. Descarga de la aplicación

Esta aplicación se puede instalar a través de la plataforma de desarrollo de aplicaciones Android (Android Studio) con el programa desarrollado en este TFG, llamado **MyApplication_PDR**. Será tan sencillo como abrir el programa con Android Studio y compilar y ejecutar el programa, a través del botón ***Run* 'app'**, como se aprecia en la imagen adjuntada a continuación.

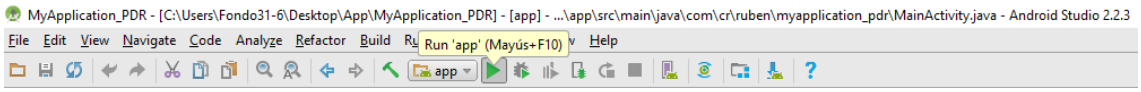

Figura 32: Instalación de la aplicación (I).

Una vez seleccionada esta opción, aparecerá una pantalla para elegir en qué dispositivo de los conectados al ordenador vía USB cargar el programa, como se puede observar en la siguiente imagen:

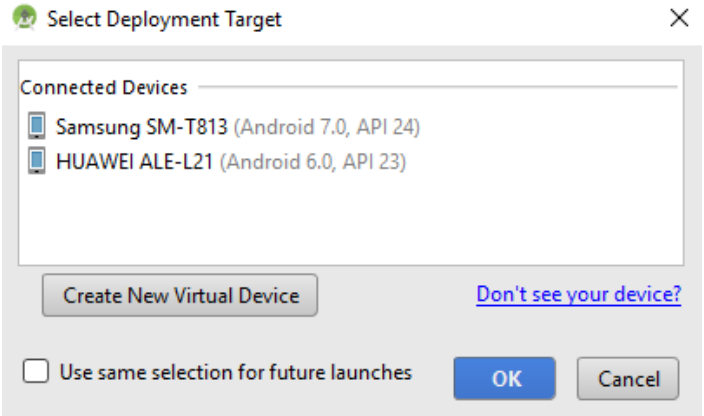

Figura 33: Instalación de la aplicación (II).

Se selecciona el dispositivo deseado, se pulsa **OK** y automáticamente se carga la aplicación en dicho dispositivo.

c. Navegación por la aplicación

i. Interfaz de arranque

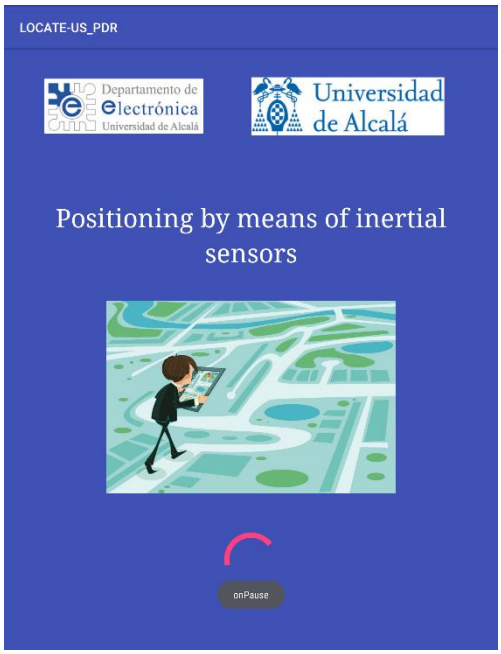

Figura 34: Interfaz de arranque.

Se ha programado una pantalla de arranque a la aplicación, de duración 4 segundos, que sirva como presentación de la aplicación que se va a abrir.

ii. Pantalla de inicio

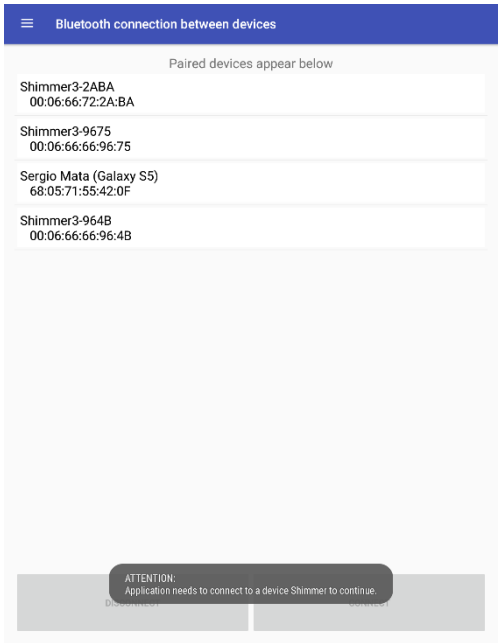

Figura 35: Pantalla de inicio.

La pantalla de inicio será la de conexión Bluetooth entre dispositivos. En esta pantalla se podrá seleccionar con que dispositivo (de una lista que contiene los dispositivos conectados y con BT visible) se vincula el dispositivo portable en el que se ha cargado la aplicación (véase la Figura 35). En nuestro caso se pulsará el Shimmer3 con identificador BT 2ABA, que es el que se ha usado a lo largo de este trabajo. Una vez que se pulse un dispositivo, se habilitará el botón de conectar y se intentará establecer conexión BT entre los dispositivos (véase la Figura 15).

iii. Transición entre pantallas

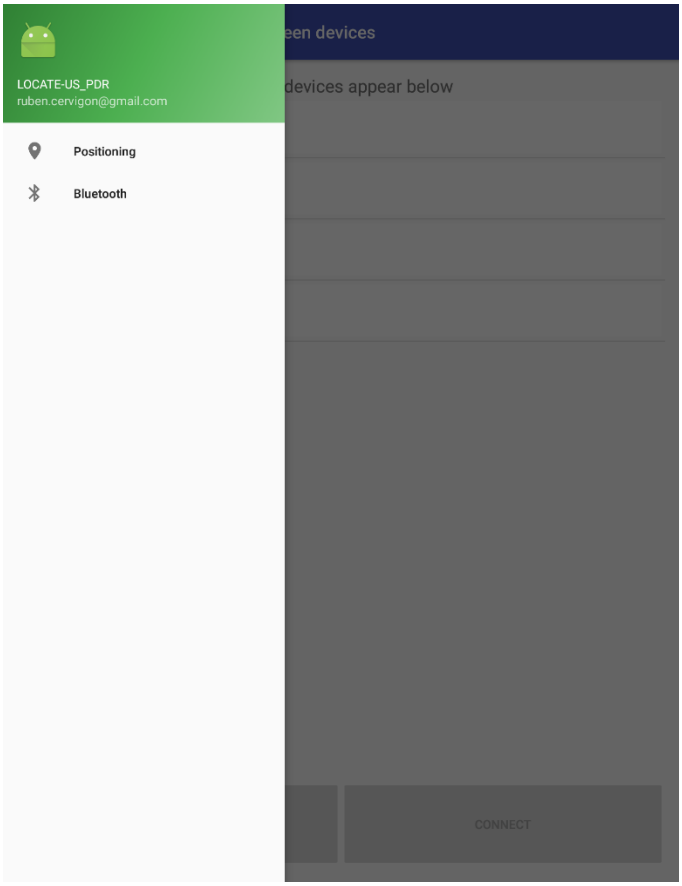

Figura 36: Transición entre pantallas (I).

Se ha programado un menú deslizante (que se despliega al pulsar el icono situado en la parte superior izquierda de las pantallas, como se puede observar enmarcado en las imágenes del final de la página) que permita pasar de la pantalla de inicio a la pantalla principal (pulsando el ítem ***Positioning***) o viceversa (pulsando **Bluetooth**).

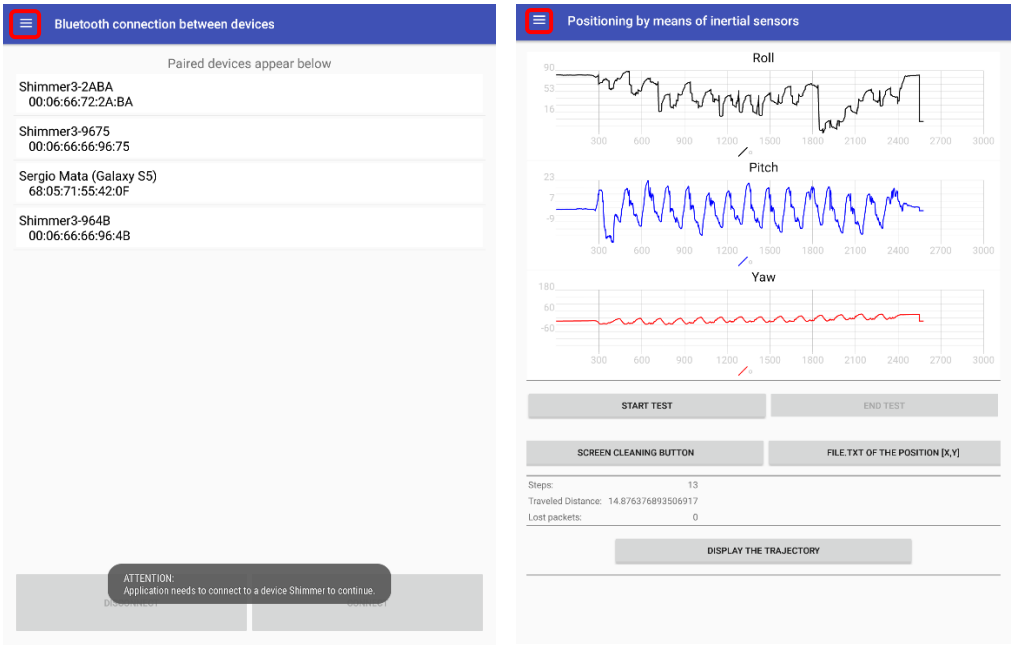

Figura 37: Transición entre pantallas (II).

iv. Pantalla principal

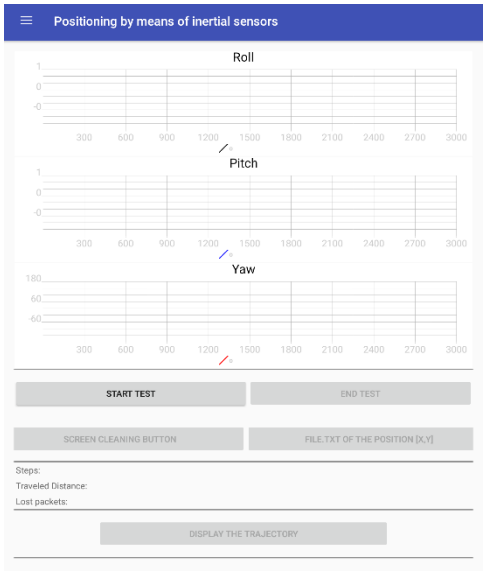

Figura 38: Pantalla principal antes de comenzar una prueba.

En esta pantalla es en la que se permite al usuario que comience la prueba (botón ***START TEST***). Inicialmente el resto de botones están bloqueados. Una vez se pulsa el botón se desbloquean los botones de finalizar prueba (***END TEST***) y de limpieza de la pantalla (***SCREEN CLEANING BUTTON***).

Durante la prueba se visualiza por pantalla las gráficas de los ángulos de orientación de la IMU (*Roll*, *Pitch* y *Yaw*), los pasos realizados, la trayectoria estimada y los paquetes perdidos (para comprobar la fiabilidad de la prueba). Para guiar al usuario a lo que debe hacer y evitar que pueda tocar botones que den problemas durante la prueba, se bloquean los botones que no tienen utilidad en dicho momento.

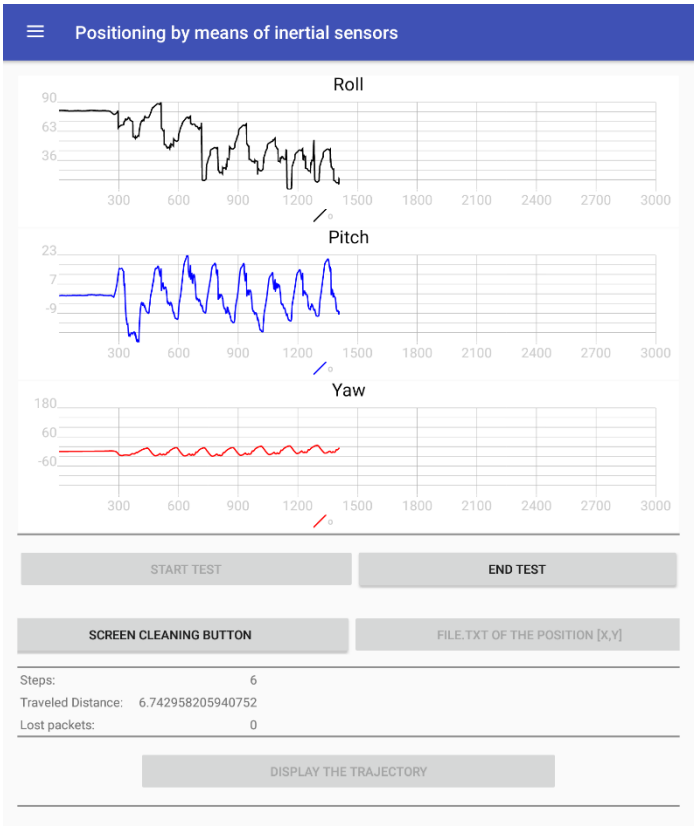

Figura 39: Pantalla principal durante la prueba.

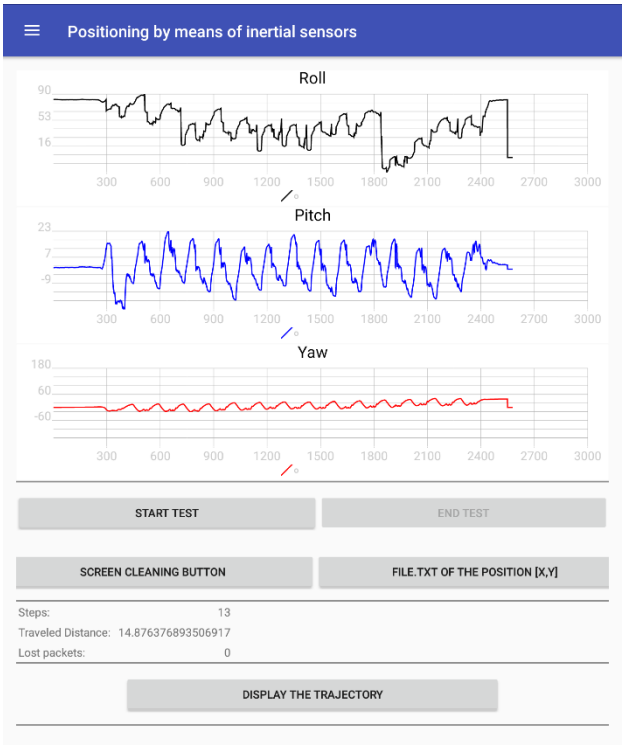

Figura 40: Pantalla principal al finalizar la prueba.

Al acabar la prueba (al pulsar el botón ***END TEST***), se habilitan dos nuevos botones (además del ***START TEST***), que permiten observar las coordenadas relativas de los puntos de la trayectoria realizada por el usuario (***FILE.TXT OF THE POSITION [X, Y]***, como el mostrado en la Figura 41) y visualizar la trayectoria realizada por el usuario en Google Maps (***DISPLAY THE TRAJECTORY***, véase la Figura 42).

X_Y.TXT

0.0	0.0
1.0481838319452739	-0.27232060888788123
2.169599303167857	-0.5788419654558259
3.3095726508546828	-0.9048235984343731
4.612752412486458	-1.1596038179193489
5.609766213726311	-1.292975567850082
6.584880113561208	-1.368215769073485
7.8209909716823836	-1.4334518931791347
9.047908191148334	-1.4990352531321052
10.200346269661827	-1.5324854169283606
11.283450333048517	-1.4589877517358387
12.551297921987288	-1.3458547895030883
13.671814707943188	-1.0998755221034435
14.662676938989488	-0.9130722520394743

Figura 41: Posicion relativa [x,y] de la prueba.

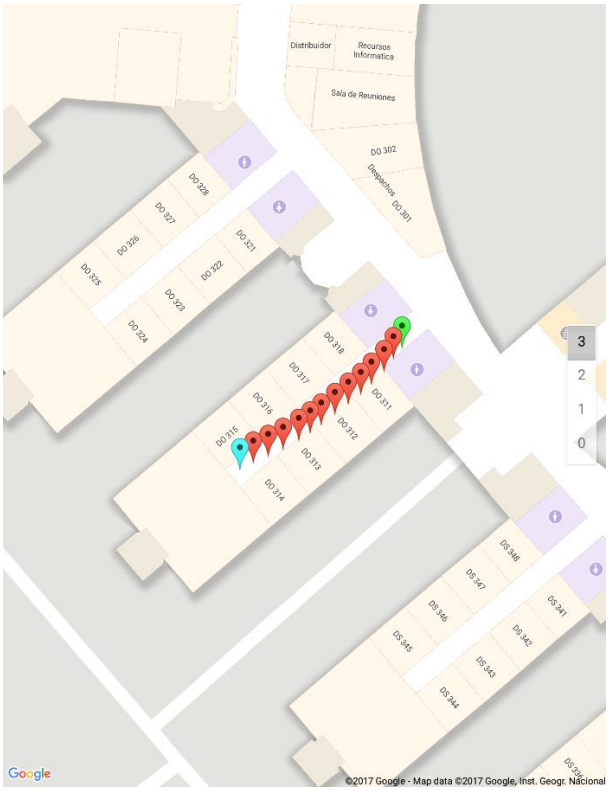

Figura 42: Trayectoria estimada durante la prueba.

d. Diagrama de flujo del interfaz de usuario

Para finalizar con el capítulo del manual de usuario se ilustra en la Figura 43 el diagrama de flujo del interfaz, con los pasos que habría que seguir para un funcionamiento habitual de la aplicación.

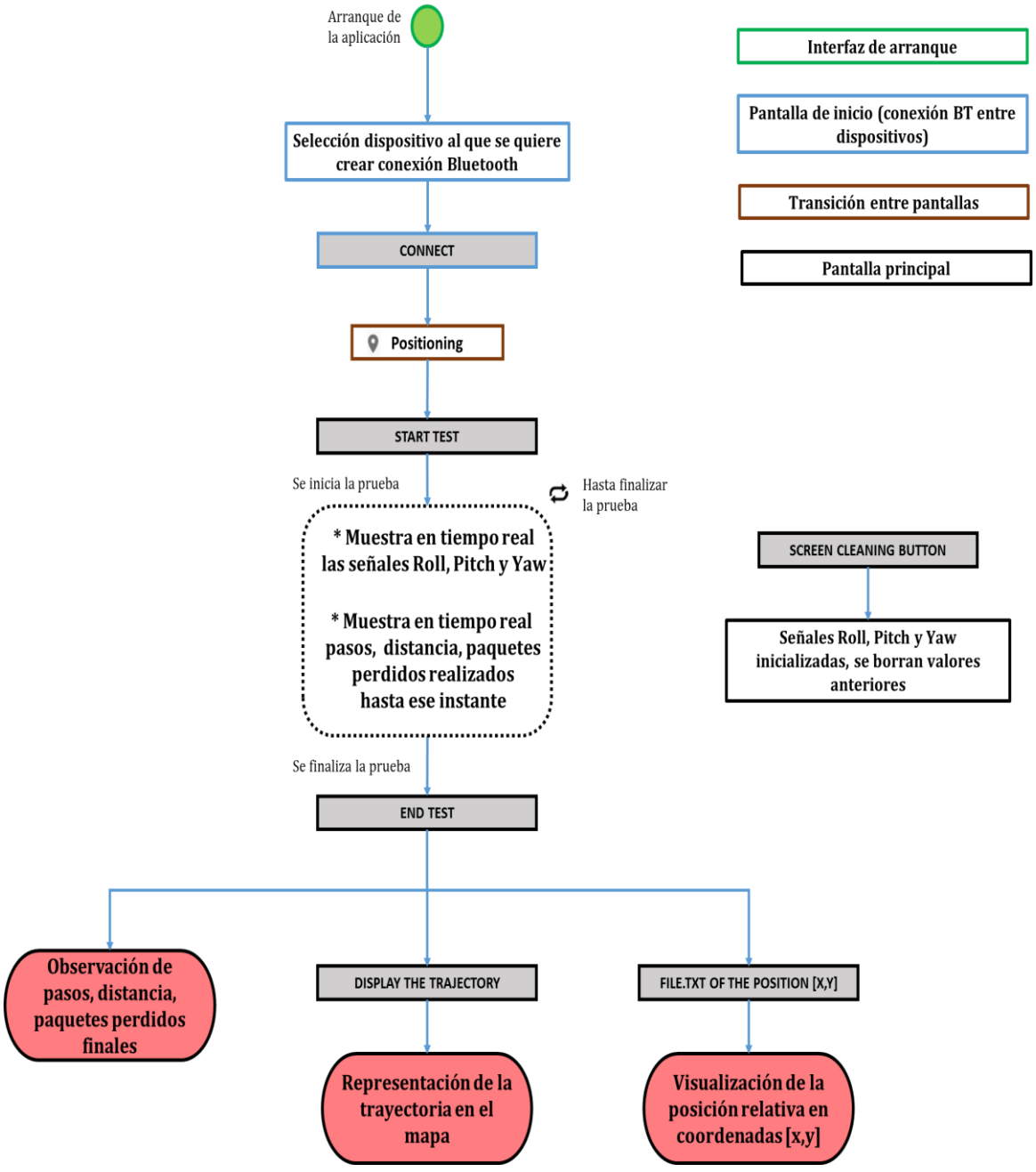

Figura 43: Diagrama de flujo del interfaz de usuario.

11. Bibliografía

[Aguilera, 2013] T. Aguilera, J. A. Paredes, F. J. Álvarez, J. I. Suárez, A. Hernández, "Acoustic Local Positioning System Using an iOS Device", Proceedings of the 2013 International Conference on Indoor Positioning and Indoor Navigation, pp. 1-8, October, 2013.

[Cox, 1964] H. Cox, "On the estimation of state variables and parameters for noisy dynamic systems", *IEEE Transactions on Automatic Control*, vol.9, no.1, pp.5-12, 1964.

[Diaz, 2017] E. Diaz, R. Gutierrez, J. J. Garcia, W. Marnane, A. Jimenez and D. Gualda, "Android based warning system for the early detection of allergic reactions," *2017 IEEE 14th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, Eindhoven, Netherlands, 2017, pp. 141-144.

[Filonenko, 2013] V. Filonenko, C. Cullen, J. D. Carswell, "Indoor positioning for smartphones using asynchronous ultrasound trilateration," ISPRS International Journal of Geo-Information, vol. 2, pp. 598-620, 2013.

[Goyal, 2011] P. Goyal, V. J. Ribeiro, H. Saran, and A. Kumar, "Strap-Down Pedestrian Dead- Reckoning System," *IEEE International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1–7, 2011.

[Gualda, 2014] D. Gualda, J. Ureña, J. García, and A. Lindo, "Locally-Referenced Ultrasonic – LPS for Localization and Navigation," Sensors, vol. 14, no. 11, pp. 21750–21769, November, 2014.

[Harle, 2013] R. Harle, "A Survey of Indoor Inertial Positioning Systems for Pedestrians," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 3, pp. 1281–1293, 2013.

[Hervás, 2014] R. Hervás, S. Lee, C. Nugent, J. Bravo, "Ubiquitous Computing and Ambient Intelligence: Personalisation and User Adapted Services", *8th International Conference UCAml, Belfast, UK, 2014.*

[Jiménez, 2009] A. R. Jiménez, F. Seco, C. Prieto and J. Guevara, "A comparison of Pedestrian Dead-Reckoning algorithms using a low-cost MEMS IMU," *IEEE International Symposium on Intelligent Signal Processing*, pp. 37-42, Budapest, 2009.

[Kalman, 1960] R.E. Kalman, "A new approach to linear filtering and prediction problems", *J.Basic Eng.*, vol.82, no.1, pp.35-45, 1960.

[Lo, 2011] C.-C. Lo, C.-P. Chiu, Y.-C. Tseng, S.-A. Chang, and L.-C. Kuo, "A Walking Velocity Update Technique for Pedestrian Dead-Reckoning Applications," *IEEE 22nd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, pp. 1249–1253, 2011.

[Lopes, 2014] S. I. Lopes, J. M. N. Vieria, J. Reis, D. Alburqueque, "Accurate Smartphone indoor positioning using WSN infrastructure and non-invasive audio for TDoA estimation," Pervasive and Mobile Computing, pp. 1-18, 2014.

[Munoz, 2014] E. Munoz Diaz and A. L. Mendiguchia Gonzalez, "Step Detector and Step Length Estimator for an Inertial Pocket Navigation System," *IEEE International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2014.

[Munoz, 2015] E. Munoz Diaz, A. Jimenez, F. de Ponte Müller, and F. Zampella, "Evaluation of AHRS Algorithms for Inertial Personal Localization in Industrial Environments" *IEEE International Conference on Industrial Technology,* 2015.

[Munoz, 2015] E. Munoz Diaz, "Inertial Pocket Navigation System: Unaided 3D Positioning," *Sensors*, vol. 15, pp. 9156–9178, 2015.

[Munoz, 2016] E. Munoz. Díaz "Inertial Pocket Navigation System for Pedestrians", PhD Thesis, 2016.

[Pérez, 2016] M. C. Pérez, D. Gualda, J. M. Villadangos, J. Ureña, P. Pajuelo, E. Díaz, E. García, "Android application for indoor positioning of mobile devices using ultrasonic signals", Proceedings of the 2016 International Conference on Indoor Positioning and Indoor Navigation, October, 2016.

[Scarlett, 2007] J. Scarlett, "Enhancing the performance of pedometers using a single accelerometer", *Application Notes, American Devices, 2007.*

[Weinberg, 2002] H. Weinberg, “Using the ADXL202 in Pedometer and Personal Navigation Applications”, *Application Notes, American Devices*, 2002.

[Zhou, 2016] R. Zhou, "Pedestrian dead reckoning on smartphones with varying walking speed," *IEEE International Conference on Communications (ICC)*, pp. 1-6, Kuala Lumpur, 2016.

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá