Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DE COMPUTADORES

Mejora de la calibración de sistemas de posicionamiento en interiores con teléfonos móviles

Estudiante: Jose Luis Álvarez Quiroga
Dirección: Carlos Vázquez Regueiro

A Coruña, febreiro de 2020.

”Yeah, science!”
-Jesse Pinkman, Breaking Bad

Agradecimientos

A mi tutor por guiarme y ayudarme constantemente durante todo este proceso.
A mis compañeros por todas las horas hemos compartido.
A mis amigos por estar siempre ahí.
A mis padres por apoyarme siempre y confiar en mí.
A Noelia.

Resumen

En este proyecto se aborda el estudio de una serie de mejoras aplicadas al proceso de calibración de edificios para posicionamiento en interiores. Este proceso de calibración, con las herramientas proporcionadas por los proveedores de servicios de posicionamiento en interiores, tiende a ser complejo, lento y muy propenso a errores humanos. Para tratar de abordar estos problemas, se ha desarrollado una aplicación Android, con una arquitectura modular, que permite automatizar el proceso, y prevenir los errores más comunes.

El usuario de esta aplicación puede crear un edificio, seleccionar su planta y posición actual, y comenzar a calibrar de forma automática. Para ello, se hace uso de la librería ARCore como sistema odométrico, obteniendo los desplazamientos del usuario en los 3 ejes.

Se aplican también mejoras en lo relacionado con los cambios de planta, creando un módulo de detección automática, para tratar de aliviar aún más la carga de trabajo en el usuario a la hora de llevar a cabo una calibración.

Abstract

This project addresses the study of a series of improvements applied to the process of calibrating buildings for indoor positioning. This calibration process, with the tools provided by indoor positioning service providers, tends to be complex, slow and very prone to human error. To try and address these problems, an Android application has been developed, with a modular architecture, which allows the process to be automated, and prevents the most common errors.

The user of this application can create a building, select its floor and current position, and start calibrating automatically. To do this, the ARCore library is used as an odometric system, obtaining the user's movements on the 3 axes.

Improvements have also been made in relation to floor changes, creating an automatic detection module, to try and further alleviate the user's workload when carrying out a calibration.

Palabras clave:

- Posicionamiento en interiores
- Calibración de edificios
- ArCore
- Aplicaciones Android

Keywords:

- Indoor positioning
- Buildings calibration
- ArCore
- Android Applications

Índice general

1 Introducción **1**

1.1 Motivación . . . 1

1.1.1 Propuesta . . . 1

1.2 Objetivos . . . 2

1.3 Estructura de la memoria . . . 3

2 Estado del arte **5**

2.1 Situm: Situm Mapping Tool . . . 5

2.2 Indoor Here: HERE Indoor Radio Mapper . . . 6

2.3 Indoor Atlas: Indoor Atlas Map Creator 2 . . . 7

3 Conceptos teóricos **9**

3.1 Localización . . . 9

3.1.1 Localización en exteriores (GPS) . . . 9

3.1.2 Sistema de posicionamiento en interiores (IPS) . . . 10

3.2 Proceso de calibración de un IPS y principales problemas . . . 11

3.3 Cálculo del desplazamiento: odometría . . . 12

3.3.1 Podómetros . . . 12

3.3.2 Odometrías visuales . . . 13

3.3.3 Otras odometrías e integración . . . 13

3.4 ArCore . . . 13

3.4.1 ArCore como odometría visual . . . 14

4 Análisis y requisitos **17**

4.1 Análisis de tecnologías . . . 17

4.1.1 Odometría: *ArCore* . . . 17

4.1.2 Almacenamiento: Base de datos *SQLite* . . . 17

4.1.3 Proveedor IPS: *Situm* . . . 18

4.1.4 Herramientas de desarrollo . . . 19
4.2 Requisitos . . . 20
4.2.1 Requisitos funcionales . . . 20
4.2.2 Requisitos no funcionales . . . 21
4.3 Casos de uso . . . 21

5 Metodología **23**
5.1 SCRUM . . . 23
5.2 Roles en SCRUM . . . 24
5.3 Implementación en este proyecto . . . 24

6 Planificación y costes **27**
6.1 Recursos . . . 27
6.1.1 Recursos materiales . . . 27
6.1.2 Recursos software . . . 27
6.1.3 Recursos humanos . . . 28
6.2 Riesgos y planes de contingencia . . . 28
6.3 Planificación inicial . . . 29
6.3.1 Análisis de requisitos y herramientas . . . 29
6.3.2 Sprint 1: Odometría . . . 30
6.3.3 Sprint 2: Módulo de calibración . . . 30
6.3.4 Sprint 3: Visualización . . . 31
6.3.5 Sprint 4: Interacciones de usuario . . . 32
6.3.6 Sprint 5: Gestión de edificios . . . 32
6.3.7 Sprint 6: Generación de modelos de posicionamiento . . . 33
6.3.8 Sprint 7: Cambio de planta automático . . . 33
6.3.9 Sprint 8: Estudio de la generación automática de mapas . . . 34
6.3.10 Pruebas, documentación y memoria . . . 34
6.4 Seguimiento del proyecto . . . 34
6.5 Costes . . . 36
6.5.1 Costes del material . . . 36
6.5.2 Costes del software . . . 38
6.5.3 Costes del personal . . . 38
6.5.4 Coste final del proyecto . . . 38

7 Arquitectura del sistema **39**
7.1 Estructura general . . . 39
7.2 Modelo de datos . . . 40

7.2.1 Edificios . . . 40
7.2.2 Escaneos y datos sensoriales . . . 40
7.2.3 Odometría . . . 41
7.2.4 Exportación a fichero . . . 42
7.3 Lógica de la aplicación (almacenamiento) . . . 42
7.3.1 Almacenamiento de datos de edificios . . . 42
7.3.2 Almacenamiento de ficheros . . . 43
7.3.3 Almacenamiento de datos simples . . . 43
7.4 Lógica de la aplicación (calibración) . . . 44
7.4.1 Escaneo de señales . . . 44
7.4.2 Módulo de detección de cambio de planta . . . 45
7.4.3 Módulo de calibración . . . 46
7.5 Vista . . . 48
7.5.1 Actividades . . . 48
7.5.2 Módulo de visualización . . . 49
7.5.3 Tutorial . . . 49

8 Implementación **53**
8.1 Módulo de calibración . . . 53
8.1.1 Cálculo de correcciones de usuario . . . 53
8.1.2 Transformación de odometría . . . 53
8.2 ArCore . . . 55
8.2.1 Obtención de datos . . . 55
8.2.2 Tracking state . . . 55
8.2.3 Previsualización . . . 56
8.2.4 Dispositivos soportados . . . 56
8.3 Módulo de visualización . . . 57
8.3.1 Eventos de interacción . . . 58
8.3.2 Gestos . . . 59
8.3.3 Pintado y cálculo de dimensiones . . . 60
8.4 Experiencia de usuario . . . 60
8.4.1 Ilustraciones . . . 61
8.4.2 Respuesta del sistema . . . 61
8.4.3 Vistas personalizadas . . . 61

9 Pruebas **63**
9.1 Pruebas de final de sprint . . . 63
9.2 Pruebas de funcionamiento de ArCore . . . 63

9.3 Pruebas en Situm . . . 64
9.4 Pruebas finales . . . 66
9.4.1 Tiempo de calibración . . . 66
9.4.2 Densidad de GTs . . . 67
9.4.3 Errores cometidos . . . 68
9.4.4 Cambios de planta . . . 69
9.5 Pruebas de generación de mapas de ocupación . . . 69

10 Conclusiones y trabajo futuro **71**
10.1 Conclusiones . . . 71
10.1.1 Requisitos funcionales . . . 72
10.1.2 Requisitos no funcionales . . . 72
10.2 Trabajo futuro . . . 73
10.2.1 Integración en herramienta de calibración de IPS . . . 73
10.2.2 Mejoras de la propia aplicación . . . 73

A Despliegue automático **77**

B Contenido del DVD **81**

C Manual de uso **83**
C.1 Requisitos . . . 83
C.2 Manual de usuario . . . 83
C.2.1 Tutorial de inicio . . . 83
C.2.2 Creación de edificio . . . 84
C.2.3 Pantalla detalles de edificio y creación de plantas . . . 84
C.2.4 Calibración del edificio . . . 85

Lista de acrónimos **87**

Glosario **89**

Bibliografía **91**

Índice de figuras

2.1 Prueba de calibración con *Situm Mapping Tool*. 6
2.2 Prueba de calibración con *HERE Indoor Radio Mapper*. 7
2.3 Prueba de calibración con *Indoor Atlas Map Creator 2*. 8

3.1 Constelación de satélites GPS alrededor de la tierra. 10
3.2 Diferencias entre los tipos de IPS. 11
3.3 Aplicación Ikea Place con funcionalidades de ArCore. 14
3.4 Ejemplo de detección de planos y de puntos de interés con la librería ArCore. . 15

4.1 Dashboard del IPS de Situm. 18
4.2 Aplicación *Situm Mapping Tool*. 19

5.1 Flujo de trabajo en SCRUM. 24

6.1 Diagrama de Gantt con la planificación inicial del proyecto. 35
6.2 Diagrama de Gantt con la ejecución final del proyecto. 37

7.1 Esquema con la arquitectura general del proyecto. 40
7.2 Estructura de la base de datos de edificios. 41
7.3 Clase «BuildingsManager.java». 43
7.4 Contenidos del paquete «scanners» de nuestra aplicación. 44
7.5 Ilustración de la zona de detección del cambio de planta en un edificio. 46
7.6 Interfaz «FloorDetectionListener». 46
7.7 Interfaz «CalibrationModuleListener». 47
7.8 Ejemplos de indicaciones visuales en el módulo de visualización. 50
7.9 Tutorial sobre el funcionamiento de la aplicación. 50

8.1 Método «rotatePoint». 54
8.2 Método «handleFrame». 55

8.3 La mínima versión del sdk Android soportada por la aplicación es la 24. 56
8.4 Código para comprobar si la librería ArCore está instalada en el dispositivo. . 57
8.5 Mensaje que aparece en un dispositivo sin ArCore. 58
8.6 Parte del método *onDraw*() del módulo de visualización. 61
8.7 Ejemplo de ilustraciones de la librería *undraw.co*. 62
8.8 Ejemplo de la visualización de las dimensiones de un edificio. 62

9.1 Primer prototipo de aplicación con integración de ARCore. 64
9.2 Ejemplo de visualización del modelo wifi en el *Dashboard* de Situm, generado con nuestra aplicación. 65
9.3 Ejemplo de visualización del modelo wifi en el *Dashboard* de Situm, generado con la aplicación de calibración de Situm. 66
9.4 Ejemplo de dispositivo posicionado en un edificio calibrado con nuestra aplicación. 67
9.5 Resultados de la calibración del edificio Citius. A la izquierda, con Situm Mapping Tool; a la derecha, con nuestra aplicación. 68
9.6 Prototipo de aplicación ArCore para detectar de planos verticales y horizontales. 70

A.1 Captura de pantalla del canal de Telegram donde se han publicado y compartido las nuevas versiones de la aplicación. 79

C.1 Menú inicio aplicación y creación de nuevo edificio. 84
C.2 Creación de planta. 85

Índice de tablas

6.1 Riesgos contemplados en el proyecto y su impacto estimado. 28
6.2 Costes de los recursos materiales del proyecto. 36
6.3 Coste de los recursos humanos del proyecto. 38
6.4 Coste de los recursos humanos del proyecto. 38

Capítulo 1

Introducción

En este proyecto se tratará de programar una aplicación que permita mejorar el proceso de calibración de edificios para sistemas de posicionamiento en interiores.

A continuación se hablará de los objetivos y motivación del proyecto, y se explicará por encima los contenidos de este documento.

1.1 Motivación

Para conseguir un sistema de posicionamiento en interiores robusto y de calidad, es necesario un proceso previo de calibración en el entorno de trabajo (p.e. un edificio). La calidad del proceso de calibración condiciona el rendimiento posterior de todo el sistema.

En la mayoría de los casos, este proceso de calibración lo realiza el usuario mediante una aplicación específica. Una vez cargado el mapa del edificio, el usuario debe recorrer el edificio indicando repetidamente sobre el mapa la posición que ocupa en cada momento (lo que en la jerga de la calibración se denomina "Ground Truth" o GT). La aplicación va escaneando periódicamente las señales bluetooth y wifi así como otros sensores de interés (sensores inerciales, etc.). Como los datos no están sincronizados, la posición del usuario se interpola linealmente a partir de los GTs, asumiendo que la velocidad del usuario es constante.

La calibración es un proceso tedioso y muy repetitivo y, por lo tanto, muy propenso a despistes y errores humanos. Los más habituales son marcar el GT con cierto error (falta de precisión al establecer la posición real del usuario), desorientación puntual del usuario, mala interpretación del mapa, movimientos no rectilíneos y cambios en la velocidad.

1.1.1 Propuesta

Para los problemas mencionados en el apartado anterior, en este proyecto se propone emplear varias medidas que permiten simplificar el proceso de calibración. El objetivo final

es conseguir realizar una calibración más robusta, rápida y que permita generar un modelo de datos más denso y con menos errores.

Puesto que la facilidad de uso es uno de los principales requisitos del proyecto, se planteará la utilización de algún dispositivo móvil para acometer esta tarea, lo más compacto posible, y a poder ser sin accesorios adicionales. Este requisito nos ha hecho descartar posibles ideas como la utilización de robots con control remoto o microcontroladores con sensores odométricos muy precisos, pero aparatosos y no altamente disponibles. Por tanto, se ha tomado la decisión final de basar el proyecto en teléfonos móviles. En concreto, nos centraremos en el sistema operativo Android, ya que es el sistema operativo de teléfonos móviles más usado en la actualidad [1].

La principal mejora propuesta será incluir una odometría capaz de calcular desplazamientos relativos del usuario en tiempo real, mostrando en la propia aplicación de calibración su trayectoria sobre el mapa. El usuario podrá en todo momento corregir al sistema e indicar su posición correcta en caso de discrepancia.

Otra mejora a estudiar será identificar automáticamente los cambios de planta realizados por el usuario durante su trayectoria, bien por medio del uso de odometrías tridimensionales, bien por medio de otros sensores (p.e. barómetro).

1.2 Objetivos

El objetivo principal del proyecto es simplificar y mejorar el proceso de calibración de un sistema de posicionamiento en interiores basado en teléfonos móviles. Para ello, se tratarán de cumplir los siguientes objetivos en cada uno de los hitos del proyecto.

1. Incluir un sistema odométrico para calcular y mostrar en la aplicación de calibración los desplazamientos del usuario sobre el mapa del edificio, permitiendo al usuario corregir al sistema en caso de ser necesario.

2. Diseñar y desarrollar una arquitectura general de la aplicación. Que en el futuro se pudiese integrar en cualquier sistema de localización de interiores e, incluso, integrar diferentes “fuentes de odometría”.

3. Estudiar un mecanismo para detectar los cambios de planta y facilitar la calibración.

4. Estudiar la posible automatización en la generación de los mapas del edificio (espacio libre).

1.3 Estructura de la memoria

En los siguientes capítulos de la memoria se explicará en más detalle el desarrollo llevado a cabo en este proyecto. A continuación se explican de forma breve cada uno de dichos capítulos:

- Estado del arte: Se analizarán las alternativas disponibles en el mercado que traten de solventar objetivos similares a los de este proyecto.
- Conceptos teóricos: En este capítulo se explicará de forma teórica los principales conceptos y términos abarcados en este proyecto.
- Análisis y requisitos: Este capítulo hablará de los requisitos y casos de uso de la aplicación.
- Metodología: Se explicarán las decisiones en cuánto a metodología de desarrollo, y sus aplicaciones en este proyecto.
- Planificación y costes: Planificación de tareas, estimación de costes y análisis de riesgos.
- Arquitectura: En este capítulo se explicará el diseño de la arquitectura del proyecto.
- Implementación: Detalles de implementación del proyecto basados en la arquitectura definida en el capítulo anterior.
- Pruebas: Listado de las pruebas llevadas a cabo.
- Conclusiones y trabajo futuro: En este último capítulo de la memoria se hablará de las conclusiones obtenidas a lo largo del desarrollo, así como de futuras mejoras y cambios en el proyecto

Capítulo 2

Estado del arte

En esta capítulo se analizarán proyectos de índole similar al nuestro, centrándonos en qué características ofrecen, y en qué aspectos pueden ser mejoradas. Ya que el concepto de este proyecto es tan innovador, no se han encontrado sistemas que permitan realizar procesos de calibración automática, al menos expuestos de forma pública. Por tanto, en esta sección se hará un pequeño análisis de las herramientas de calibración de las tres principales empresas de posicionamiento en interiores a modo de comparativa.

2.1 Situm: Situm Mapping Tool

Situm Mapping Tool [2] es la herramienta utilizada para llevar a cabo la calibración de edificios de la empresa de posicionamiento en interiores *Situm* [3]. Para hacer uso de esta herramienta, el manual de usuario [4] indica que se debe andar a ritmo constante, en movimiento siempre rectilíneos, pulsando sobre el mapa para indicar la posición. Se nos indica que el usuario debe prestar atención a la hora de indicar correctamente su posición sobre el mapa, y que nunca debería dejar de andar durante el proceso de calibración. En caso de hacerlo, deberá parar la calibración, e iniciar una nueva cuándo se ponga en marcha de nuevo.

Basándonos en pruebas realizadas, esta aplicación permite calibrar plantas enteras en una sola calibración (como se puede apreciar en la figura 2.1), por lo que el proceso es relativamente rápido, pero implica estar constantemente pendiente de la aplicación, y aún así, se cometen errores muy frecuentemente a la hora de indicar la posición sobre el mapa, sobre todo si dicho mapa no está correctamente actualizado o representado.

Además de todo esto, debemos estar pendientes de mantener un ritmo constante, prestar especial atención a las trayectorias, y no parar de movernos hasta terminar la calibración. De lo contrario la calibración final obtenida podría resentirse.

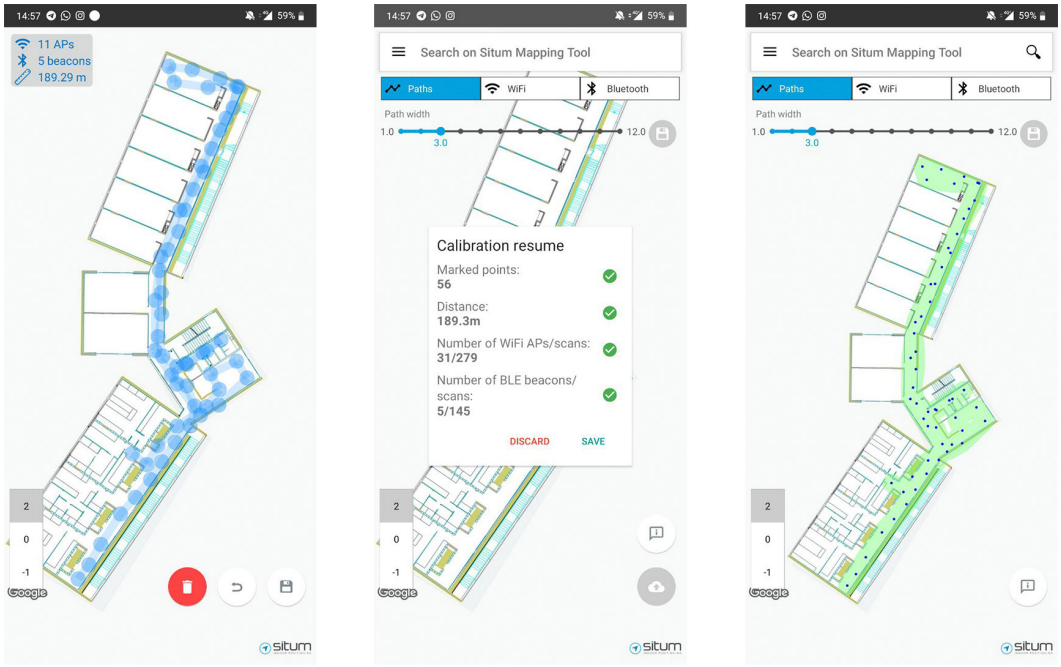

Figura 2.1: Prueba de calibración con *Situm Mapping Tool*.

2.2 Indoor Here: HERE Indoor Radio Mapper

La aplicación *HERE Indoor Radio Mapper* [5] es la herramienta de la empresa *HERE Indoor Positioning* [6] para llevar a cabo la recolección de datos del entorno. Siguiendo la documentación disponible en su web y tutoriales en vídeo sobre cómo usar la aplicación [7][8], se nos indica que el proceso es similar a *Situm*. El usuario debe andar siempre en movimientos rectilíneos, marcando su posición exacta cada poca distancia y manteniendo un ritmo constante.

Haciendo pruebas con la aplicación, descubrimos que el proceso es mucho más lento que con Situm, pero manteniendo las mismas restricciones. El debe hacer movimientos sólo rectilíneos, manteniendo velocidad constante, y pulsando la pantalla para indicar su posición. Sin embargo, lo que hace que este proceso sea mucho más lento que el de Situm, es que la aplicación obliga a terminar la calibración al final de cada movimiento en línea recta, y comenzar otra calibración nueva (figura 2.2). De esta manera, el usuario avanza en línea recta unos metros, se para, interactúa con el dispositivo móvil, y vuelve a avanzar.

Por tanto, presenta los mismos problemas que la aplicación de *Situm*, hay que prestar atención al ritmo con el que se camina, la trayectoria y marcar con precisión nuestra posición en el mapa. Todo esto con la inconveniencia añadida de tener que parar cada pocos segundos para inciar una nueva calibración, y la pérdida de tiempo que esto implica.

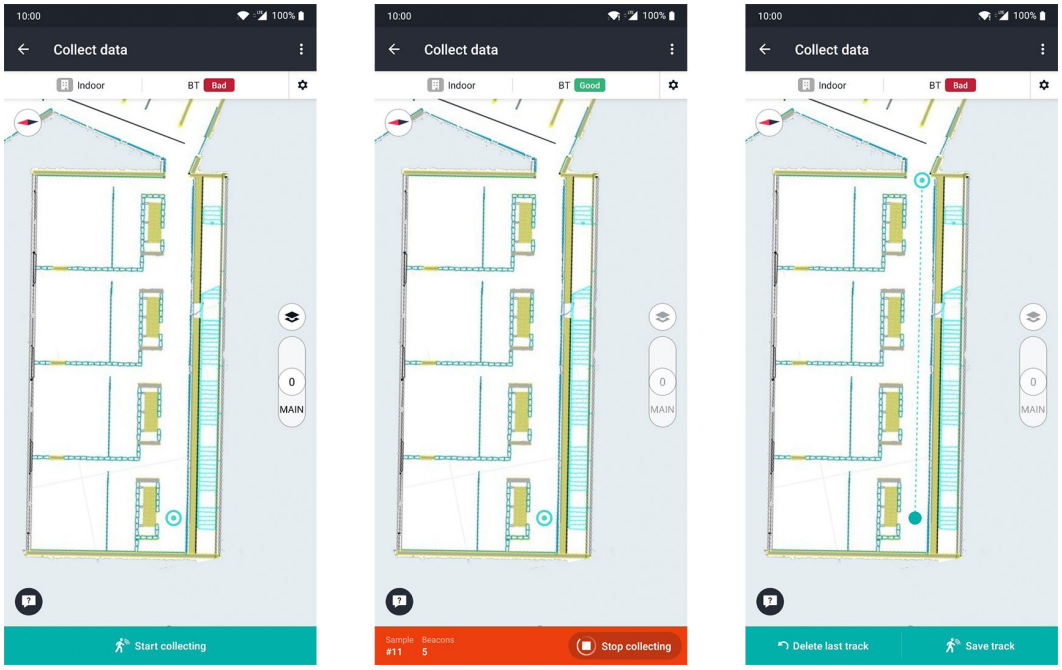

Figura 2.2: Prueba de calibración con *HERE Indoor Radio Mapper*.

2.3 Indoor Atlas: Indoor Atlas Map Creator 2

El proveedor de posicionamiento en interiores *IndoorAtlas* proporciona una aplicación para llevar a cabo la calibración de edificios ligeramente diferente a las dos mencionadas anteriormente [9].

Antes de comenzar a andar, se deberán marcar sobre el mapa los denominados *waypoints*. Estos *waypoints* serán puntos marcados por el usuario que indicarán la trayectoria a seguir a la hora de empezar a calibrar [10]. Por tanto, el proceso es el siguiente:

- El usuario genera la ruta de antemano marcando los *waypoints* por los que deberá pasar
- Una vez generada la ruta, comienza a andar, y cada vez que alcance un waypoint, deberá indicárselo a la aplicación.

A mayores, utiliza un sistema inercial para ayudar al usuario. De esta forma, permitiría al usuario indicar en todo momento la distancia a la que se encuentra del siguiente *waypoint*, proporcionando, teóricamente, una ayuda visual muy importante.

Una vez más, se realizaron pruebas con esta aplicación para evaluar el proceso de calibración con este sistema. Marcar los *waypoints* antes de empezar la ruta, aunque a veces complique la visión del propio mapa durante la calibración (figura 2.3), parece una ayuda considerable a la hora de no perderse cuándo se está avanzando por el edificio, puesto que

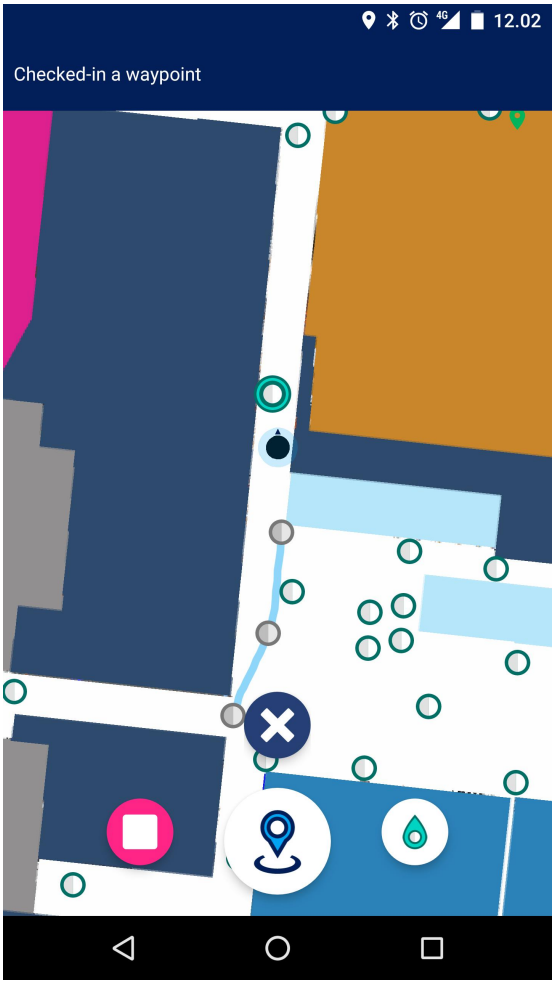

Figura 2.3: Prueba de calibración con *Indoor Atlas Map Creator 2*.

implica haber realizado un previo análisis del mismo para marcar la trayectoria deseada. A pesar de esto, el tiempo de calibración se amplía mucho, ya que el usuario debe preparar la trayectoria antes de comenzar la calibración, y debe pulsar la pantalla para indicar su posición igual que con el resto de sistemas. Además, el sistema odométrico parece no proporcionar resultados fiables, por lo que al final el usuario se ve obligado a hacer movimientos rectilíneos y con velocidad constante, igual que en el resto de sistemas.

Capítulo 3

Conceptos teóricos

En este capítulo se hará una introducción a los principales conceptos teóricos que se van a manejar durante la memoria. En primer lugar se explicarán conceptos básicos, funcionamiento, y tipos de localización, principalmente basándose en dispositivos móviles. A continuación, se expondrá de forma teórica el concepto de odometría, centrándonos en aquellas odometrías usadas o contempladas en nuestro proyecto.

3.1 Localización

La localización de describe como el proceso mediante el cual se obtiene la posición de un sujeto en un sistema de referencia. En el ámbito de los dispositivos móviles, se contemplan dos tipos de sistemas de referencia.

La localización en exteriores (como la que pueda proporcionar Google Maps u otros tantos proveedores de servicios de localización global) tiene como sistema referencia el globo terráqueo, presentando posiciones basadas en latitud longitud. La localización en interiores, por la contra, permite a un usuario obtener su posición dentro de un contexto más reducido, como pueda ser la planta de un edificio. Las posiciones generalmente vienen expresadas en un sistema de referencia cartesiano, en unidades métricas. Conociendo parámetros de dicho sistema de referencia, como las dimensiones en metros, o su posición geográfica, se puede llegar a hacer la conversión de un sistema de referencia a otro.

3.1.1 Localización en exteriores (GPS)

GPS o Global Positioning System es un sistema de posicionamiento basado en satélites, que permite determinar la posición de un usuario en cualquier parte del planeta, con una precisión que puede llegar hasta la orden de centímetros.

Este sistema de posicionamiento se basa en una red de al menos 24 satélites que se mantienen en órbita alrededor de la tierra. Cada uno de dichos satélites completará una órbita precisa

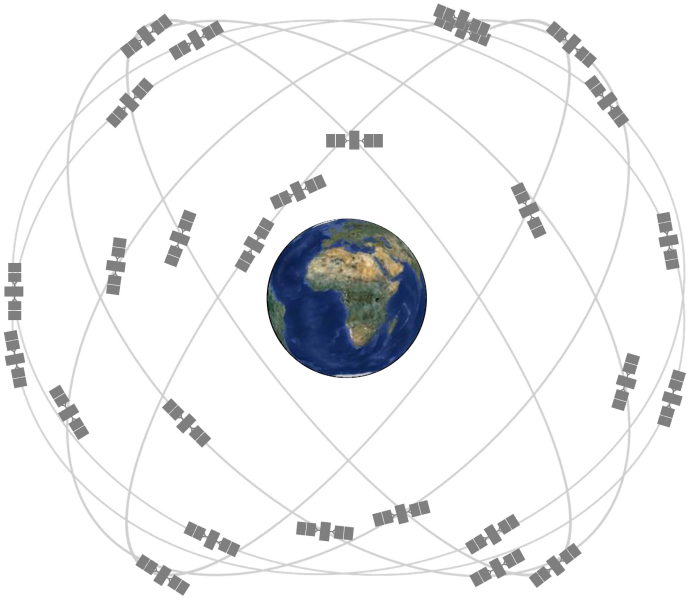

Figura 3.1: Constelación de satélites GPS alrededor de la tierra.

alrededor de la tierra dos veces al día, conociéndose su ubicación exacta en cada momento.

A la hora de posicionar un dispositivo, recibirá la posición exacta y el tiempo de envío de señal de cada uno de los satélites que escuche en su ubicación actual. A partir de dicha información, podrá calcula la distancia hasta el satélite. Una vez se conoce la distancia de al menos 4 satélites, se usará geometría para calcular la posición final del usuario. [11]

Debido a la naturaleza de la señal GPS, en entornos cerrados como puedan ser edificios, la señal se degrada de forma considerable, llegando incluso a no escucharse en absoluto. Por este motivo surgen los sistemas de posicionamiento en interiores.

3.1.2 Sistema de posicionamiento en interiores (IPS)

Un IPS o Indoor Positioning System es un sistema que permite localizar a uno o varios usuarios dentro de un entorno cerrado, haciendo uso de distintos sensores. Los sistemas de posicionamiento orientados a dispositivos móviles, se basan principalmente en escaneos de señales electromagnéticas, y datos de sensores inerciales, dado que la señal GPS no aporta información fiable en dichos entornos.

Dentro de los IPS, hay dos tipos de servicios de posicionamiento. En la figura 3.2 se ilustra la principal diferencia entre ambas.

- ***Navigation***. El servicio de navegación en IPS es la funcionalidad más extendida cuándo se habla de posicionamiento en interiores. Permite a un dispositivo obtener su posición dentro de un entorno basándose en sensores inerciales y escaneos de señales. Se necesita

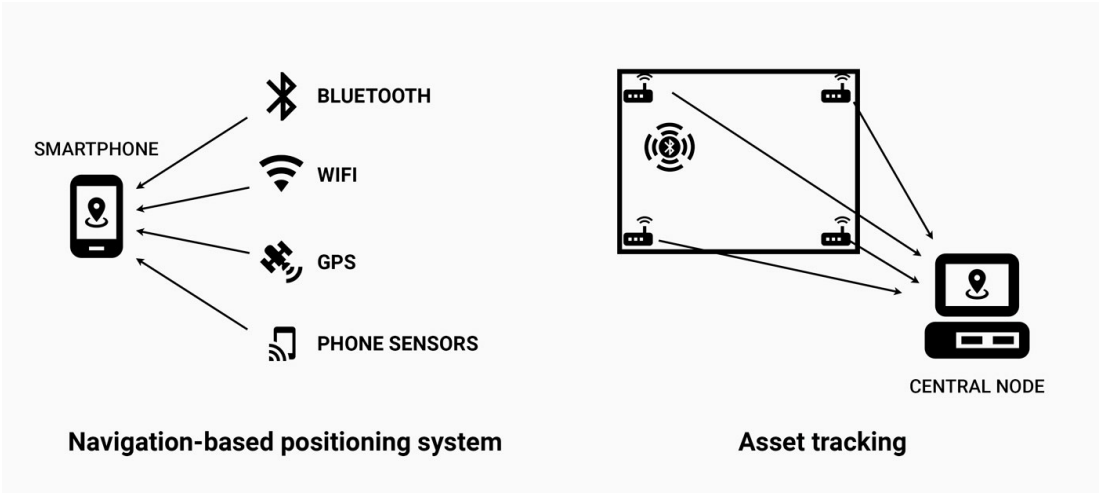

Figura 3.2: Diferencias entre los tipos de IPS.

de un proceso de calibración del entorno previo, además de, en la mayoría de casos, instalación de balizas bluetooth en el edificio para mejorar la cobertura de señal. Todo el cómputo de posición se realiza en el propio dispositivo, por lo que es muy escalable a la hora de número de usuarios. A lo largo de este documento, nos referiremos a este tipo de servicio cuándo hablemos de posicionamiento en interiores.

- ***Asset Tracking* o Localización de activos**. Este servicio, a diferencia del anterior, es el objeto a localizar el que emite señal. Para poder calcular su posición, se colocan rastreadores de señal a lo largo del edificio, que enviarán los datos de lecturas a un nodo de cómputo central, dónde se procesará la información y se devolverá la posición de cada uno de los dispositivos. Este mecanismo permite al usuario posicionarse con objetos tan simples como balizas bluetooth, o incluso pulseras, y no requiere de un proceso de calibración previa. La desventaja de este sistema, es que la precisión cae de manera considerable con respecto al anterior, y es mucho menos escalable. Por tanto, el fin de este servicio es más bien detectar la zona del edificio estimada dónde se encuentra un usuario, más que la localización exacta.

3.2 Proceso de calibración de un IPS y principales problemas

Los sistemas de posicionamiento, generalmente, obtienen la información para el cálculo de posición a partir de odometrías (podómetro, odometría visual...) y a partir de un modelo de señal previamente generado. Dicho modelo se obtiene mediante la calibración del entorno, idealmente, asociando cada punto del espacio con una medida de potencia para cada una de las señales presentes en el edificio.

Este proceso se lleva a cabo generalmente mediante el uso de dispositivos móviles, con una aplicación propietaria del proveedor de posicionamiento en interiores.

Los pasos a reproducir son relativamente sencillos, pero tediosos y muy propensos a errores. El usuario debe abrir la aplicación, con el plano del edificio precargado, y seleccionar la planta en la que se encuentra actualmente. A continuación, deberá ir pulsando el mapa cada pocos metros para indicar su posición en cada momento dentro del edificio. Durante todo este proceso, la aplicación se encargará de escanear señales wifi y ble a la mayor frecuencia posible

3.3 Cálculo del desplazamiento: odometría

La odometría se define como el estudio de la estimación de la posición de un sujeto, basándose en la información obtenida de su desplazamiento. Es un término generalmente asociado a la robótica, donde la naturaleza del movimiento de los robots permite obtener resultados muy precisos y robustos, ya que las rotaciones generalmente están restringidas a un eje, y la traslación, a dos. Además de esto, con el uso de *encoders* se puede medir de forma muy precisa la distancia recorrida [12].

Dado que las odometrías se basan en el cálculo del desplazamiento, no podemos conocer la posición absoluta del mismo solamente con una odometría, pero sí proporciona una información muy valiosa para los IPS.

En dispositivos móviles, los tipos de odometría disponibles están limitados por el número de sensores del mismo. Principalmente se tratará de podómetros y odometrías visuales.

3.3.1 Podómetros

Un podómetro permite obtener el desplazamiento de un usuario basándose en las medidas obtenidas de sensores como el acelerómetro, el giróscopo o el magnetómetro del teléfono. Con algoritmos tales como filtros de Kalman se pueden lograr resultados bastante fiables a partir este tipo de odometrías en condiciones ideales.

Esta tecnología presenta una serie de dificultades a la hora de trabajar en dispositivos móviles. La primera de ellas, es el ruido que puedan tener cada uno de los sensores del teléfono, que se traducirán en un deterioro de la calidad del algoritmo. Otra desventaja con respecto a trabajar, por ejemplo, con robots, es la naturaleza del movimiento de un teléfono móvil. A diferencia de los robots, el teléfono no está restringido ni en rotación, ni en desplazamiento, en ninguno de sus ejes. Esto provoca que los cálculos de orientación sean altamente complejos, generalmente llevando a pequeñas desviaciones, que a medida que se continua el desplazamiento, se irán acumulando, llegando a provocar resultados con un alto grado de error.

3.3.2 Odometrías visuales

Este tipo de odometrías permiten obtener la posición de un sujeto a partir del procesado de fotogramas secuenciales capturados con la ayuda de una cámara. De cada una de las imágenes, se extraen una serie de puntos de interés, que permiten llevar a cabo los cálculos de desplazamiento del usuario [13].

La odometría visual tiene la desventaja de que, al requerir el uso continuo de la cámara, el consumo de batería suele dispararse comparado con un podómetro, y la luz es un factor determinante a la hora del cálculo de posición.

Los resultados sin embargo tienden a ser más precisos que los de un podómetro, y proporcionan una ventaja importante con respecto a los mismos; el cierre de lazos. Como ya se ha mencionado anteriormente, la odometría visual identifica puntos de interés en cada uno de los fotogramas que procesa. En caso de que el usuario vuelva a transitar una zona previamente analizada por el sistema odométrico, podría reconocer dichos puntos de interés, y automáticamente corregir los posibles errores de rotación o traslación que haya cometido durante la trayectoria.

3.3.3 Otras odometrías e integración

Además de las odometrías comentadas, se podrían emplear otras fuentes. Por ejemplo, basadas en sensores lidar, en sensores inerciales o incluso sensores específicamente diseñados para tal fin (tipo la familia Intel Real Sense). Pero como ya se ha comentado, en ese caso se necesitaría hardware adicional lo que haría el sistema final mucho menos portable y también más caro. En este prototipo se ha primado la sencillez y la facilidad de despliegue.

Cuando existen diversas odometrías (bien del mismo o diferente tipo) se hace necesaria integrarlas para conseguir un único desplazamiento del usuario. Existen diversas técnicas para realizar esta fusión. Sin embargo, es necesario disponer de alguna métrica para evaluar la precisión y fiabilidad de cada fuente.

Como veremos en el capítulo de implementación (8), en la documentación de la librería elegida para la odometría visual no hemos encontrado ningún mecanismo para obtener esta información. Sólo disponemos del estado global del sistema.

3.4 ArCore

ARCore es un kit de desarrollo software creado por Google enfocado a la programación de aplicaciones en realidad aumentada. Es una librería relativamente nueva, lanzada el 1 de marzo de 2018, que requiere tener una versión actualizada de Android, y dispone de una lista reducida de dispositivos soportados [14]. A pesar de ello, es la más robusta y utilizada entre los dispositivos Android.

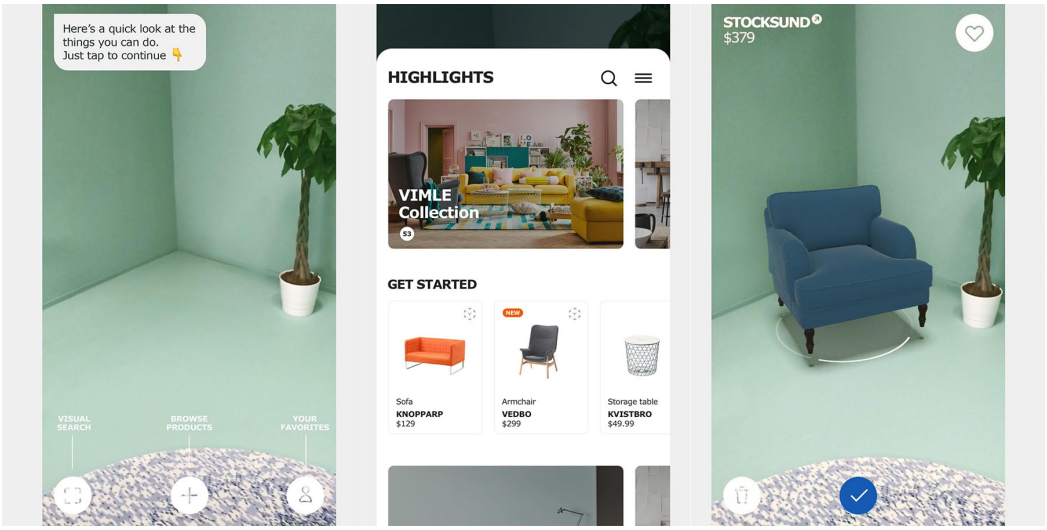

Figura 3.3: Aplicación Ikea Place con funcionalidades de ArCore.

Empresas como *Ikea* [15] y *Amazon* [16], entre otras, utilizan esta librería para añadir funcionalidades de realidad aumentada a sus aplicaciones propietarias (figura 3.3).

Para poder trabajar con realidad aumentada, esta librería necesita reconocer el entorno y los movimientos del teléfono. Esto lo consigue mediante un algoritmo de procesado de imágenes fusionado con los datos obtenidos del sistema inercial del dispositivo. Dicho de otra manera, ArCore trabaja combinando un podómetro y una odometría visual para conseguir un sistema de cálculo de desplazamientos muy fiable.

3.4.1 ArCore como odometría visual

Como ya mencionamos anteriormente, para la parte de odometría visual de esta librería, ArCore, como el resto de odometrías visuales, obtiene una serie de puntos de interés para reconocer el movimiento del dispositivo. Pero adicionalmente, ArCore agrupa estos puntos de interés y puede de esta forma llevar a cabo un reconocimiento de planos, como puede observarse en la figura 3.4.

La librería ArCore también genera información específica sobre los planos detectados, aunque sea parcialmente. Con esta información sería posible plantearse crear una especie de mapa con los "lugares transibles" del entorno. Es decir, el plano del suelo visible y, por lo tanto, no ocupado con obstáculos. Incluso sería posible emplear los planos verticales detectados (posibles paredes) para mejorar la información del mapa propuesto.

De todos modos, en las pruebas preliminares de la librería se ha comprobado que la detección de los planos no es demasiado fiable, ya que depende en gran medida de la textura de los objetos, de la iluminación del entorno y de la distancia relativa de la cámara. Así, muchas

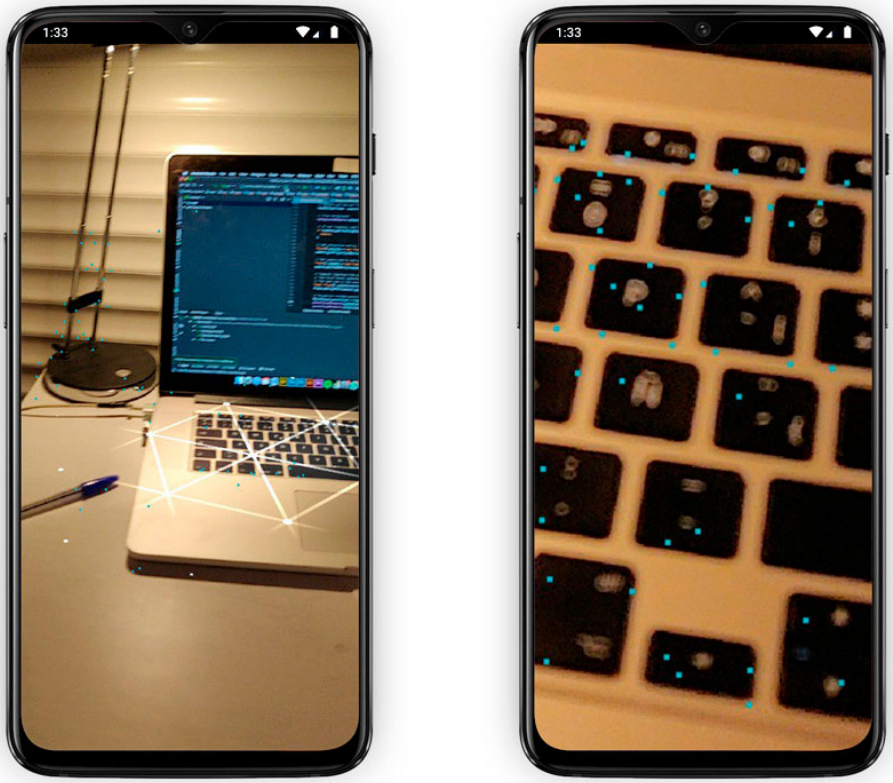

Figura 3.4: Ejemplo de detección de planos y de puntos de interés con la librería ArCore.

de las paredes blancas sólo son detectadas por ArCore cuando la cámara del móvil está muy cerca. Por lo tanto, es muy posible que la detección de los obstáculos no sea muy robusta. Otra cosa sería la detección del plano del suelo, fundamental para el seguimiento.

Capítulo 4

Análisis y requisitos

En este capítulo se entrará más en detalle en la fase de análisis del desarrollo. Se analizarán los principales requisitos de la aplicación, tanto funcionales como no funcionales, y los casos de uso de la misma.

4.1 Análisis de tecnologías

En esta sección se hará un análisis de las herramientas o recursos tecnológicos más relevantes para nuestro proyecto.

4.1.1 Odometría: *ArCore*

Se han barajado distintos tipos de odometrías y sensores para obtener los desplazamientos relativos del usuario. Al desarrollarse una aplicación móvil, la mayoría de sensores que no sean propios del dispositivo, implicarán una gran carga de desarrollo e investigación para permitir la conexión entre ambos. Por ese mismo motivo, podómetros u odometrías visuales son las dos opciones más fácilmente integrables en este proyecto. Tanto *ArCore* como *Arkit*, las dos librerías de realidad aumentada explicadas anteriormente, cumplen ambos requisitos. Debido a la facilidad de desarrollo, despliegue y distribución de aplicaciones Android con respecto a aplicaciones para *iOS*, se ha elegido ArCore como odometría para este proyecto. Se llevará a cabo un desarrollo para permitir obtener el desplazamiento relativo del usuario a través de esta librería.

4.1.2 Almacenamiento: Base de datos *SQLite*

SQLite es una librería de gestión de bases de datos relacionales, que viene por defecto integrada en todos los dispositivos móviles, y gran parte de ordenadores [17]. Trabaja con sintaxis SQL, y cómo ya se ha mencionado, viene preinstalada en el sistema operativo Android,

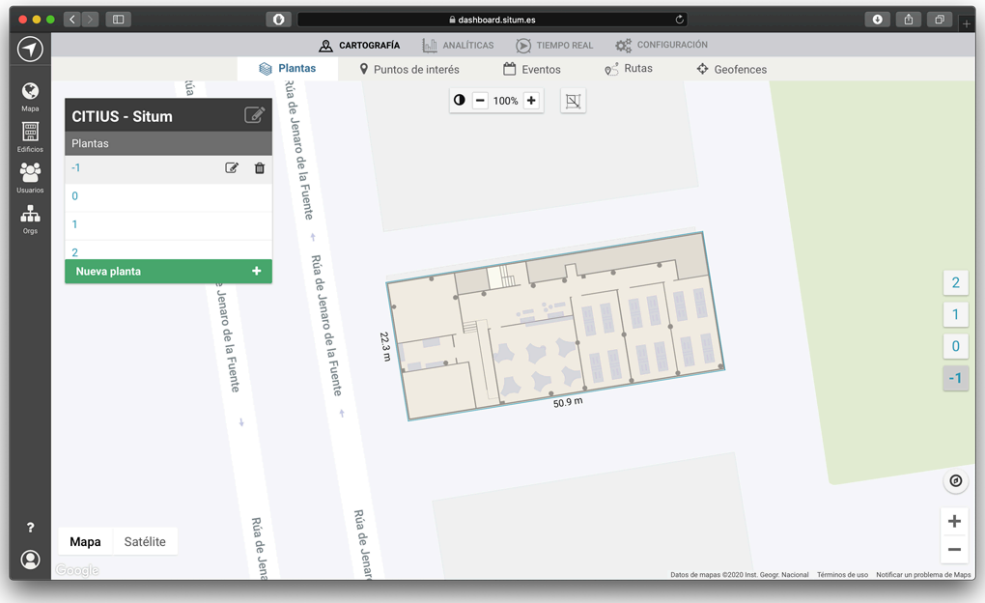

Figura 4.1: Dashboard del IPS de Situm.

por lo que simplifica la integración en nuestra aplicación. Se utilizará esta tecnología para almacenar la información sobre los edificios creados por el usuario.

4.1.3 Proveedor IPS: *Situm*

Una vez generadas las calibraciones con nuestra herramienta, tendremos que hacer pruebas de posicionamiento para comprobar el correcto funcionamiento de nuestra aplicación. Para ello se hará uso de distintas herramientas del proveedor de posicionamiento en interiores *Situm*.

Dashboard

La herramienta *Dashboard* de *Situm* [18] es una plataforma web que nos permite trabajar con los edificios que tenemos creados, gestionar las calibraciones subidas de cada edificio, y ver los dispositivos posicionando en tiempo real, entre otras muchas funcionalidades. Usaremos este *Dashboard* para crear edificios en una cuenta de *Situm*, subir nuestras calibraciones y generar modelos de posicionamiento.

Situm Mapping Tool

Situm Mapping Tool [2] es una aplicación móvil que ofrece Situm. Como ya mencionamos en el apartado de estado del arte (2), esta aplicación nos permite tanto calibrar como posicio-

Figura 4.2: Aplicación *Situm Mapping Tool*.

nar. En nuestro caso, estaremos interesados sólo en la parte de posicionar, para poder llevar a cabo pruebas de localización usando el modelo de posicionamiento que hayamos generado previamente.

4.1.4 Herramientas de desarrollo

Para el desarrollo del proyecto se ha hecho uso de las siguientes herramientas:

- *Android Studio*: Para todo el desarrollo de la aplicación Android se ha hecho uso del *IDE* gratuito ofrecido por Google, Android Studio.
- *Latex*: Para la edición de archivos *latex* se ha usado el editor online gratuito Overleaf
- *Gitlab*: Se ha utilizado Gitlab para el control de versiones y para hacer uso de las ven-

tajas de integración continua que ofrecen de forma gratuita, pudiendo de esta forma configurar tests automatizados, y compilaciones y despliegues automáticos.

- *Telegram bots*: A la hora de hacer despliegues automáticos disponibles para el equipo de desarrollo de la forma más rápida, se ha hecho uso de bots de *Telegram*, creando un canal dónde se publica cada nueva versión de la aplicación compilada y subida directamente desde los procesos de integración continua de *Gitlab* al hacer cambios en el repositorio, mostrando un *changelog* con los cambios incluidos en dicha nueva versión.

- Figuras y diagramas: Para la creación de figuras y diagramas se ha hecho uso de herramientas de edición de imágenes como *Adobe Photoshop* y herramientas de creación de diagramas online como draw.io

4.2 Requisitos

A continuación se detallan los requisitos, tanto funcionales como no funcionales, que debe cumplir nuestra aplicación.

4.2.1 Requisitos funcionales

En esta sección se detallarán los requisitos funcionales surgidos de reuniones con el director del proyecto y del análisis de trabajo previo realizado en capítulos anteriores.

- Gestión de edificios y persistencia: El usuario de nuestra aplicación debe poder crear distintos edificios, con sus respectivas plantas e imágenes de planta. Esta información se debe persistir, preferentemente, de forma local en el propio dispositivo. No se hará uso de base de datos en la nube, ya que el fin de esta herramienta es la futura integración en algún IPS, cuya infraestructura ya tendrá contemplada la creación y gestión de edificios, por lo que sería replicar trabajo.

- Poder cambiar de planta de forma sencilla y lo más transparente para el usuario posible. La aplicación debería permitir al usuario cambiar de planta en medio de una calibración, sin tener que manualmente detener el proceso de calibración, cambiar de planta, y reanudar de nuevo. Se estudiará también la posibilidad de permitir calibraciones continuas, aplicando el cambio de planta automáticamente de forma transparente para el usuario.

- Corregir al sistema en caso de error de precisión. Durante el proceso de calibración, el usuario debe poder en todo momento corregir al sistema e indicar su posición real.

- Ejecución en tiempo real. Entre otros requisitos implica minimizar escrituras a disco. Una calibración trabaja con una gran cantidad de datos que crecen de manera lineal a medida que aumenta el tiempo de la propia calibración. También se de debe mantener limitado el tiempo de procesamiento de los datos. Éste no debe crecer con el número de datos de calibración.

- Guardar ficheros localmente en el teléfono móvil en una zona de fácil acceso para el usuario. Los ficheros finales con los datos de calibración deberán almacenarse en una ruta que permita al usuario acceder de forma sencilla.

- Visualización en tiempo real. Durante una calibración, el usuario deberá poder ver en tiempo real todos los datos con los que trabaja la aplicación. Es decir, deberá poder ver sobre el mapa, su trayectoria, las correcciones que haya añadido el usuario, y los escaneos de wifi y bluetooth con sus respectivas posiciones.

4.2.2 Requisitos no funcionales

Además de lo requisitos funcionales ya mencionados, el proyecto debe también contemplar algunos no requisitos no funcionales. Los más importantes son los siguientes:

- Interfaz intuitiva. Dado que el proceso de calibración de un edificio es un proceso tedioso y propenso a errores, la interfaz de la aplicación debe ser lo más clara y concisa, ayudando al usuario en la medida de lo posible, para minimizar problemas.

- Dispositivo compacto. El dispositivo escogido debe ser lo más compacto posible, tanto por facilidad de uso para el operario, como por comodidad a la hora de transportarlo y calibrar el edificio.

- Independencia de proveedor. Los datos de salida de esta aplicación, deberán tener un formato que no esté atado a un proveedor de posicionamiento concreto, pudiendo adaptarse en un futuro en función de las necesidades.

- Fecha de entrega predefinida y no modificable.

4.3 Casos de uso

A continuación se muestra una listado de los casos de uso de la aplicación para un usuario de la misma:

- **CU-01** Ver edificios: Cuándo el usuario abre la aplicación, se le mostrará un listado de los edificios que tenga creados.

- **CU-02** Crear edificio. El usuario podrá crear un nuevo edificio pulsando el botón flotante de añadir edificio.
- **CU-03** Eliminar edificio. El usuario podrá eliminar edificios manteniendo pulsado un ítem de la lista.
- **CU-04** Visualizar detalles de edificio. Al hacer click en un ítem de la lista, se abrirá una pantalla de visualización que mostrará los detalles del edificio, incluido un listado de sus plantas.
- **CU-05** Editar detalles de edificio. Una vez el usuario esté en la pantalla de visualización de detalles del edificio, podrá pulsar en el botón de editar, arriba a la derecha, para modificar las características del edificio.
- **CU-06** Crear planta. En la pantalla de visualización de edificio, el usuario podrá añadir nuevas plantas pulsando el botón de ”Añadir planta”.
- **CU-07** Eliminar planta. El usuario podrá eliminar plantas de un edificio desde la pantalla de visualización de edificio.
- **CU-08** Visualizar planta del edificio. Dentro de la pantalla de visualización de edificio, al pulsar el botón ”Calibrar”, se mostrará la imagen de planta del edificio.
- **CU-09** Cambiar planta. Al visualizar la imagen de planta del edificio, se mostrará un selector a mano derecha, dónde el usuario podrá cambiar de planta.
- **CU-10** Iniciar calibración. Con una imagen de planta seleccionada, el usuario podrá pulsar el botón con el icono de ”play” para iniciar una calibración.
- **CU-11** Parar calibración. Una vez iniciada una calibración, el usuario podrá pulsar el botón con el icono de ”stop” para parar la calibración.
- **CU-12** Añadir GT. Una vez iniciada una calibración, el usuario podrá pulsar el mapa para añadir un GT.
- **CU-13** Abrir tutorial. En la pantalla de inicio de la aplicación, el usuario podrá abrir un tutorial desde el menú de arriba a la derecha.

Capítulo 5

Metodología

En este capítulo se describirá la metodología de desarrollo escogida para nuestro proyecto, se explicará el porqué de estas elecciones, y se hablará de la adaptación de dicha metodología al caso concreto de este proyecto.

5.1 SCRUM

SCRUM es una metodología para el desarrollo software, creada para permitir la colaboración efectiva de los miembros de un equipo en la implementación de tareas complejas [19]. Se caracteriza por el desarrollo iterativo, y las entregas a cliente de manera incremental y regular del producto [20]. Estas características hacen que esta metodología sea acertada para proyectos en entornos complejos, donde los requisitos son cambiantes y se necesita obtener resultados de la manera más rápida posible.

El proyecto se desarrolla en períodos cortos, llamados sprints, marcados por una duración de un mes o menos. Al final de cada sprint, se debe obtener un producto funcional, usable, y potencialmente listo para despliegue.

El proceso generalmente comienza con una reunión con cliente, dónde se establecen los requisitos priorizados en función de beneficio y coste.

Una vez se dispone de dicha lista, para cada sprint, se comenzaría por agrupar los requisitos en bloques funcionales que se desarrollarán durante dicho período de tiempo. Se organiza una lista de tareas concretas para cumplir con dichos requisitos y se organiza una planificación.

Durante todo este proceso, se realizan reuniones diarias entre el equipo de desarrollo, dónde los miembros se ponen al día de el progreso de cada tarea.

Al final de cada sprint, se realiza una reunión de revisión, dónde se analizan los problemas y resultados del sprint, y se muestra a cliente el funcionamiento de esta iteración. De esta reunión podrían salir nuevos requisitos.

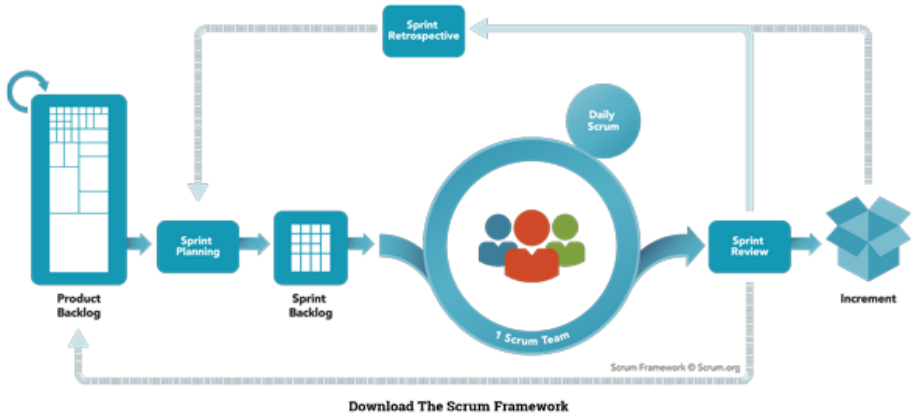

Figura 5.1: Flujo de trabajo en SCRUM.

5.2 Roles en SCRUM

- ***Product Owner***. Es el responsable de maximizar el valor resultante del producto obtenido del trabajo del equipo de desarrollo. Es el encargado de añadir, modificar, eliminar y priorizar requisitos del producto final.
- **Equipo de desarrollo**. Es un grupo de profesionales, encargados del análisis, implementación y pruebas de cada una de las iteraciones del producto. Debería ser un equipo autogestionado, dónde no debería haber subgrupos ni coordinadores.
- ***Scrum master***. El Scrum master es el encargado de promover el proceso de SCRUM, orientando al equipo de desarrollo y al cliente, ayudando a comprender las teorías, reglas y ventajas del propio SCRUM.

5.3 Implementación en este proyecto

Se ha escogido SCRUM como metodología de desarrollo principalmente debido a la naturaleza del proyecto, donde la modificación de requisitos es un riesgo con una probabilidad potencialmente alta.

A la hora de aplicar esta metodología de desarrollo en este proyecto, hay que tener en cuenta que el número de participantes en el mismo, serán solo dos personas, y el equipo de desarrollo estará compuesto sencillamente por el alumno, por lo que los roles de SCRUM estarán repartidos entre ambos.

La repartición de roles será por tanto la siguiente:

- ***Product Owner***: El rol de product owner será asumido por el tutor del trabajo de fin de grado.

- **Equipo de desarrollo**: como ya comentamos anteriormente, el equipo de desarrollo en este proyecto estará compuesto enteramente por el alumno.
- ***SCRUM master***: A falta de miembros en el proyecto, este rol estará desempeñado también por el alumno.

En cuánto a las características concretas de la aplicación de SCRUM en nuestro proyecto, podemos decir que:

- La duración de los *sprints* será variable, entre 1 y 4 semanas, plazos temporales que se ajustarán a las limitaciones de cada uno de los participantes.
- Las reuniones diarias del equipo de desarrollo serán inherentes al propio proyecto, ya que todo el equipo está compuesto por un sólo integrante, el alumno.
- Se realizarán reuniones a final de sprint con el profesor de la universidad, a modo de *Product owner*, dónde se valorará y analizará cada una de las iteraciones del producto, surgiendo nuevos requisitos si se considera necesario.

Capítulo 6

Planificación y costes

En este capítulo se hablará de la planificación inicial del proyecto, su descomposición en hitos y sprints, los recursos utilizados para el desarrollo, y el análisis de riesgos.

6.1 Recursos

A continuación se expone la lista de recursos disponibles para la realización del proyecto, divididos en recursos humanos, materiales y software.

6.1.1 Recursos materiales

Los materiales utilizados para el desarrollo del proyecto son los siguientes:

- Ordenador portátil Apple, utilizado por el desarrollador para llevar a cabo todo el trabajo de implementación y diseño.
- Dispositivos móviles, para desarrollo, pruebas de funcionamiento de la aplicación, y pruebas de campo con sistemas odométricos. En concreto, se ha hecho uso de un OnePlus 6 para desarrollo, y un OnePlus 6T y un LG Q6 para pruebas.

6.1.2 Recursos software

A continuación se listan todos los recursos software empleados para este proyecto:

- Entorno de desarrollo oficial de Android, *Android Studio*, ofrecido públicamente de forma gratuita por Google. Versión 3.3.2.
- Herramientas de gestión de posicionamiento en interiores. Se ha hecho uso del Dashboard de Situm para crear un edificio, subir las calibraciones creadas con nuestra aplicación, y generar un modelo de posicionamiento. Para probar dicho modelo de posi-

Tabla 6.1: Riesgos contemplados en el proyecto y su impacto estimado.

Riesgo	Probabilidad	Impacto en el proyecto
Modificación de requisitos	ALTA	ALTO
Interfaz de visualización compleja	MEDIA	BAJO
Librerías nuevas o desconocidas	ALTA	MEDIO
Dedicación equipo desarrollo	MEDIA	ALTO

cionamiento, se ha empleado la aplicación Situm Mapping Tool, disponible de forma pública en el Google Play Store.

6.1.3 Recursos humanos

La lista de recursos humanos para este proyecto es muy escueta. No obstante, se compone de los siguientes integrantes:

- Jefe de proyecto, es el alumno. Ejerce tareas como supervisor y *scrum master.*
- Analista/programador, es el papel desempeñado por el alumno, realizando las tareas de análisis, diseño e implementación.
- Asesor externo, es el tutor del TFG. Ejerce también de *Product Owner.*

6.2 Riesgos y planes de contingencia

En esta sección se hablará de los posibles riesgos previstos para el proyecto. Se ha evaluado la probabilidad de que sucedan (clasificando en alta, media, baja) y su repercusión total en el proyecto en el caso de darse (alta, media, baja). Se puede observar dicha evaluación en la tabla 6.1. A continuación se explica cada uno en más detalle:

- **Modificación de requisitos**. Debido a la novedad del proyecto, debe contemplarse en todo momento que la lista de requisitos inicial podría llegar a ser modificado, teniendo que hacer una re-evaluación de costes y tiempos.
- **Interfaz de visualización compleja**. La visualización de tantos datos, y en tiempo real, puede llegar a dar problemas en cuánto a rendimiento, o en cuánto a la claridad con la que aporta información.
- **Uso de librerías nuevas y desconocidas**. ARCore es una librería reciente, lo que nos coloca en un posible entorno cambiante. Además, la documentación de dicha librería está en pleno desarrollo, y todavía hay poca comunidad que haya hecho pruebas con la misma.

- **Dedicación del equipo de desarrollo**. El equipo de desarrollo de este proyecto se compone de un sólo integrante, el alumno. Dicho estudiante debe compaginar este proyecto, con el trabajo, y clases y exámenes. Por ese mismo motivo la planificación inicial podría no cumplirse en determinados momentos.

A continuación se muestra una serie de medidas que se han planteado adoptar en caso de surgir alguno de los riesgos mencionados anteriormente.

- **Modificación de requisitos**. Gracias a la metodología de desarrollo escogida (SCRUM), la modificación de requisitos no debería suponer grandes cambios en el desarrollo del proyecto, ya que es una característica contemplada dentro de la propia organización de SCRUM.
- **Interfaz de visualización compleja**. En caso de que aparezcan problemas a la hora de crear la parte de visualización, se podrá simplificar cierta funcionalidad, sobre todo a la hora de permitir interacciones de usuario. Esto afectará a nivel experiencia de usuario, pero no en cuánto funcionalidad final de la aplicación.
- **Uso de librerías nuevas y desconocidas**. Se ha establecido en los dos primeros sprints un tiempo estimado más relajado para permitir al equipo de desarrollo familiarizarse con dichas librerías, e intentar prevenir en la medida de lo posible retrasos temporales a la hora de finalizar las tareas.
- **Dedicación del equipo de desarrollo**. De nuevo, gracias a SCRUM, la organización del trabajo contempla la posibilidad de que haya fluctuaciones en cuánto a la dedicación temporal disponible para cada integrante del proyecto, permitiendo reorganizar la carga de trabajo para poder cumplir los objetivos propuestos.

6.3 Planificación inicial

En esta sección se hablará de la planificación inicial del proyecto, basándose en los recursos destinados al desarrollo del mismo, tanto materiales como humanos. Se dividirá la planificación en sprints, descompuestos así mismo en tareas concretas. Se mencionan además, tanto la fase inicial de estudio y análisis, como la fase de finalización de proyecto, que no se incluyen dentro de la fase de implementación.

6.3.1 Análisis de requisitos y herramientas

Fase preliminar al desarrollo propiamente dicho. Se dedicarán dos semanas al análisis del proyecto. Dentro de esta fase estaría la selección de dispositivo, selección de odometría y reuniones del equipo de desarrollo para organizar el transcurso del proyecto.

6.3.2 Sprint 1: Odometría

En este sprint se creará el primer prototipo de la aplicación. Este prints tendrá una duración estimada de 3 semanas. El objetivo es llevar a cabo la integración de la odometría que más adelante se usará para la calibración automática. El producto de este sprint será una aplicación simple, con una sola actividad, que mostrará por pantalla los valores de posición del usuario, en los ejes x, y, z. Las tareas concretas para este sprint serán:

- **Estructura básica de la aplicación**. Comenzar con la estructura básica de la aplicación teniendo en cuenta la descomposición en módulos acordada en la parte de arquitectura del sistema.
- **Obtención de desplazamientos relativos de ArCore**. Se tiene que conseguir extraer las deltas de movimiento del usuario a partir de la librería ARCore, definiendo un intervalo de obtención de dichos datos.

Hito 1, Reunión con el jefe de proyecto. Al finalizar este sprint, habrá una reunión con el director de proyecto para analizar los resultados de la librería ARCore, y su integración en la aplicación.

6.3.3 Sprint 2: Módulo de calibración

El objetivo de este sprint será comenzar con la implementación del módulo de calibración. Tras finalizar el anterior sprint, tendremos un módulo de obtención de desplazamiento basado en odometría, cuyos valores deberán valer de entrada para nuestro módulo de calibración. Este sprint tiene una duración de tres semanas , y la aplicación resultante permitirá llevar a cabo escaneos wifi y bluetooth, interpolando sus posiciones entre los GTs obtenidos de la odometría. Este sprint se descompone en las siguientes tareas:

- **Creación del módulo de calibración**. Deberá poder recibir como entrada los datos de odometría, calculados mediante la implementación de la iteración anterior. Esta entrada de posiciones debe ser lo más flexible posible en cuánto a tipo de datos, ya que en un futuro podría interesar cambiar la librería de odometría utilizada.
- **Escaneos de señal**. Para esta tarea, se creará dentro del módulo de calibración una interfaz común que permita escanear datos de señales como wifi o bluetooth. Estos datos deben llegar al módulo de calibración, que obtendrá una posición estimada para cada una de las lecturas interpolando entre los dos GTs más cercanos basándose en el tiempo.
- **Gestionar precisión de ArCore**. Se deberá también gestionar la parada automática de la calibración cuándo se detecte que las medidas de ARCore no son precisas.

- **Exportación a fichero**. El módulo de calibración deberá exportar los datos obtenidos durante el proceso de calibración a fichero, de formato libre, y en una ruta a la que el usuario tenga fácil acceso.

Hito 2, Reunión con el jefe de proyecto. Al finalizar este sprint, se hará una reunión con el director de proyecto para analizar resultados del sprint.

6.3.4 Sprint 3: Visualización

El objetivo de este sprint será permitir al usuario de la aplicación visualizar en el propio dispositivo los datos de la calibración. Tendrá una duración de 1 mes y el resultado será una aplicación en la que se muestre en tiempo real la trayectoria del usuario, así como las posiciones interpoladas de cada uno de los escaneos. Este sprint se descompone en las siguientes tareas:

- **ArCore en segundo plano**. Deberá permitirse usar ArCore sin que se muestre en la pantalla la previsualización de la cámara.

- **Creación de módulo de visualización**. Se creará un módulo de visualización personalizado. Se conoce de antemano que la API de Google Maps no permite hacer suficiente zoom como para trabajar con la precisión necesaria en nuestra aplicación. Por ese motivo se creará una vista personalizada con operaciones básicas como desplazar y hacer zoom.

- **Gestionar permisos**. Se deberán gestionar los permisos Android necesarios para la aplicación en tiempo de ejecución y bajo demanda.

- **Experiencia de usuario**. Se implementarán medidas que mejoren la experiencia de usuario, como gestionar la activación/desactivación automática de sensores a la hora de iniciar la calibración, bloquear el giro de pantalla para evitar rotaciones innecesarias de la vista…

- **Tutorial**. Crear un tutorial para la primera vez que se inicia la aplicación, explicando a grandes rasgos qué hace la aplicación.

Hito 3, Pruebas de funcionamiento Al finalizar este sprint, se hará una reunión con el director de proyecto para realizar pruebas en diferentes entornos con varios dispositivos. Se hará además una reunión con el *product owner* para hacer una pequeña demostración del funcionamiento.

6.3.5 Sprint 4: Interacciones de usuario

En este sprint se comenzará con la integración de interacciones del usuario con el módulo de calibración. Tendrá una duración de 3 semanas, y el producto obtenido de este sprint será una aplicación, en la que se le muestre al usuario el plano de un edificio, pueda indicar su posición inicial y orientación, y pueda corregir al sistema odométrico pulsando en la pantalla para indicar su posición real. La descomposición de tareas de este sprint es la siguiente:

- **Pintar mapa del edificio**. En el módulo de visualización se deberán poder pintar planos de edificios. Para ello, habrá que preparar nuestra vista personalizada para que trabaje en metros internamente, y la escala de los datos obtenidos de la odometría coincida con las dimensiones del edificio.
- **Posición inicial sobre mapa**. El usuario deberá poder indicar su posición inicial y orientación pulsando sobre el mapa.
- **Correcciones de usuario**. Una vez iniciada la calibración, el usuario podrá corregir al sistema odométrico pulsando la pantalla para indicar su posición real.

Hito 4, Reunión con el jefe de proyecto. Este sprint finalizará con una reunión con el director de proyecto para analizar las interacciones del usuario, y pulir o corregir pequeños errores en caso de ser necesario.

6.3.6 Sprint 5: Gestión de edificios

Este sprint tiene como objetivo que el usuario pueda trabajar con varios edificios o plantas. Este sprint tendrá una duración de 2 semanas, y el resultado obtenido será una aplicación que permita crear, editar y eliminar edificios, con persistencia en base de datos SQLite, y pudiendo añadir múltiples plantas para cada edificio. Las tareas a implementar durante este sprint son:

- **Gestión de edificios**. Se deberá crear una base de datos SQLite en el propio dispositivo, dónde se almacenará información de los edificios que el usuario haya creado. Se deberán poder añadir, editar o eliminar edificios.
- **Selector de planta**. Una vez creados un edificio, suponiendo que tenga varias plantas añadidas, el usuario deberá poder cambiar de planta en el módulo de visualización. Al cambiar de planta se deberá, de manera transparente para el usuario, finalizar la calibración actual y comenzar una nueva.

Hito 5, Reunión con el jefe de proyecto. Al finalizar este sprint se llevará a cabo una reunión con el jefe de proyecto para analizar los cambios llevados a cabo en este sprint, con el fin de mejorar la interacción del usuario y corregir pequeños errores.

6.3.7 Sprint 6: Generación de modelos de posicionamiento

El objetivo de este sprint será utilizar la aplicación desarrollada para obtener datos de calibración, que se subirán al servidor de Situm, generando modelos de posicionamiento, y probando posteriormente el correcto funcionamiento de nuestro sistema. Para ello, habrá que modificar la librería de escaneos wifi y ble para adaptarse el tipo de datos que utiliza Situm en sus calibraciones. Como resultado se tendrá una serie de edificios correctamente calibrados creados en la cuenta de Situm. Este sprint tendrá como duración 1 semana. Las tareas concretas son las siguientes:

- Creación de cuenta Situm. Crear una cuenta en el Dashboard de Situm, dando de alta los edificios que se crean convenientes para llevar a cabo las pruebas.

- Subir datos a servidor Situm. Utilizar nuestra aplicación para calibrar los edificios creados en el Dashboard de Situm. Posteriomente subir los ficheros generados al servidor de Situm, para poder hacer pruebas de posicionamiento.

Hito 6, Pruebas de campo. Al finalizar este sprint se llevará a cabo una reunión con el jefe de proyecto, con el fin de hacer pruebas de posicionamiento en los edificios calibrados durante este sprint. Se hará también una reunión con el *product owner* para poner al día de los nuevos cambios.

6.3.8 Sprint 7: Cambio de planta automático

En este sprint se creará un módulo de detección de cambio de planta automático, integrado dentro del propio módulo de calibración. La duración de este sprint será 1 semana, y el producto obtenido del mismo será una aplicación que permita al usuario cambiar de planta durante una calibración y que el sistema lo detecte automáticamente. Las tareas concretas son:

- Añadir altura de planta. Añadir altura entre plantas como un parámetro configurable para cada edificio.

- Módulo detección. Crear el módulo de detección dentro del módulo de calibración, que se encargará de analizar los desplazamientos del usuario en el eje Z para poder cuándo cree que se está realizando un cambio de planta.

Hito 7, Reunión con jefe de proyecto. Al finalizar este sprint se llevará a cabo una reunión con el jefe de proyecto, para analizar el funcionamiento del cambio de planta automático, y hacer pruebas.

6.3.9 Sprint 8: Estudio de la generación automática de mapas

En este sprint, se realizará un análisis de la viabilidad de la detección de planos de ARCore como un mecanismo de generación de mapas automático. Este sprint tendrá una duración de 2 semanas. Las tareas concretas de este sprint serán:

- Hacer una prueba de concepto que permita detectar planos y paredes con ARCore.
- Obtener de ARCore las posiciones de cada uno de los planos detectados.
- Pintar sobre una vista los planos detectados con ARCore.

Hito 7, Reunión con jefe de proyecto. Al finalizar este sprint se llevará a cabo una reunión con el jefe de proyecto, en la que se estudiará la viabilidad de la prueba de concepto llevada a cabo en este sprint.

6.3.10 Pruebas, documentación y memoria

Al finalizar el último sprint, se prevé un tiempo adicional de 3 semanas para completar la parte de conclusiones y trabajo futuro de la memoria del proyecto. También se realizarán pruebas de funcionamiento con distintos dispositivos para detectar posibles errores, así cómo hacer pequeños cambios de UX.

6.4 Seguimiento del proyecto

En la figura 6.1 se muestra la planificación original del proyecto mediante un diagrama de Gantt. A continuación se muestra un seguimiento del proyecto a lo largo de los sprints mencionados anteriormente, con los cambios e incidencias más relevantes.

- **Sprint 1**: Este sprint se desarrolló en el marco de tiempo previsto por la planificación inicial.
- **Sprint 2**: Este sprint se desarrolló en el marco de tiempo previsto por la planificación inicial.
- **Sprint 3**: En este sprint, se produjeron problemas de integración a la hora de utilizar la librearía de ARCore en segundo plano. Este riesgo estaba contemplado inicialmente, por lo que solamente produjo pequeñas desviaciones temporales.
- **Sprint 4**: A la hora de gestionar las pulsaciones del usuario en la pantalla, para corregir la posición de la odometría, surgieron bastantes problemas por culpa de la cantidad de transformaciones que se llevaban a cabo en el módulo de visualización. Finalmente se consiguió implementar pero a costa de retrasos temporales.

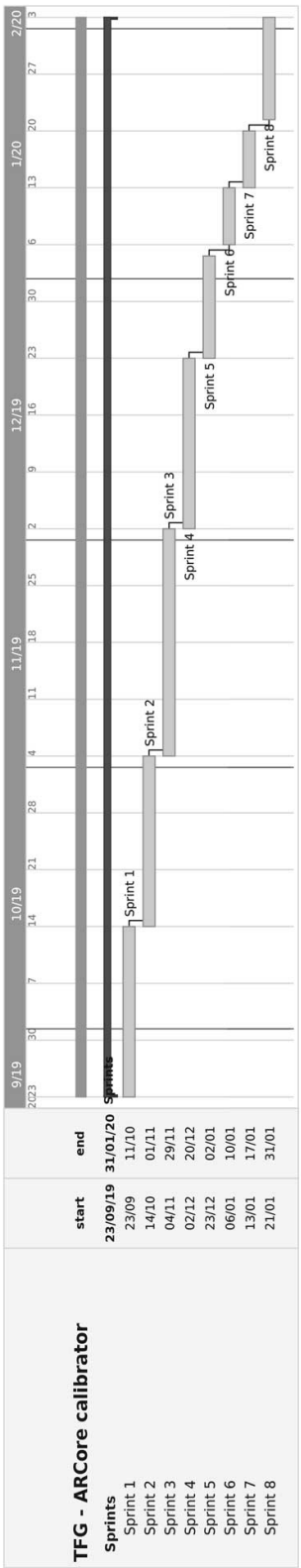

Figura 6.1: Diagrama de Gantt con la planificación inicial del proyecto.

Tabla 6.2: Costes de los recursos materiales del proyecto.

Recurso	Coste real	Coste en el proyecto
Ordenador portátil Apple	2.000€	0€
Teléfono móvil OnePlus 6	330€	0€
Teléfono móvil OnePlus 6T	400€	0€
Teléfono móvil LG Q6	150€	0€
Total		0€

- **Sprint 5**: De nuevo, este sprint siguió la planificación esperada.
- **Sprint 6**: El desarrollo de este sprint transcurrió según lo previsto.
- **Sprint 7:** La implementación del módulo de cambio de planta, requirió muchas pruebas de campo para poder seleccionar el algoritmo idóneo. Dichas pruebas prolongaron el tiempo de desarrollo, llevando a desviaciones en la planificación inicial.
- **Sprint 8**: Debido a los retrasos temporales en anteriores sprints por motivos laborales del equipo de desarrollo, y puesto que el proyecto tenía un *deadline* prefijado, este sprint no se ha podido completar a tiempo. Aunque sí se ha llegado a crear una prueba de concepto de la detección de planos, como indicaba la primera tarea de este sprint, que se explicará más adelante en la sección de pruebas.

Para mostrar el seguimiento de una forma más clara y concisa, se ha creado un diagrama de Gantt que refleja el seguimiento final de la planificación, los sprints llevados a cabo, y el desarrollo temporal del proyecto. Dicho diagrama se puede observar en la figura 6.2.

6.5 Costes

En esta sección se hablará de los costes estimados basándose en la planificación inicial del proyecto, y los materiales y recursos empleados.

6.5.1 Costes del material

En los costes de material se indica el coste inicial estimado para cada uno de los recursos utilizados en el proyecto. Cabe destacar, que todos los recursos mencionados anteriormente estaban disponibles y amortizados a la hora de iniciar el proyecto, por lo que el coste real será 0, como se puede apreciar en la tabla 6.2.

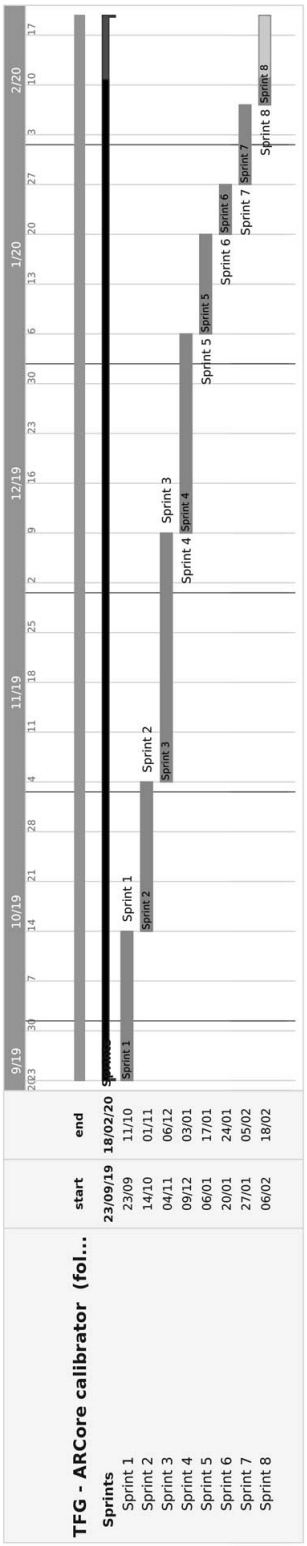

Figura 6.2: Diagrama de Gantt con la ejecución final del proyecto.

Tabla 6.3: Coste de los recursos humanos del proyecto.

Recurso	Salario por hora	Horas estimadas	Coste
Jefe de proyecto	50€	50	2.500€
Asesor externo	50€	10	5.000€
Programador	25€	400	10.000€
Analista	40€	80	3.200€
Formación	0€	80	0€
Total			20.200€

Tabla 6.4: Coste de los recursos humanos del proyecto.

Recursos	Coste (€)
Materiales	0
Software	0
Humanos	20.200
Total	20.200

6.5.2 Costes del software

Para llevar a cabo este proyecto, todo el software utilizado ha sido de licencia libre. Por ese mismo motivo, tampoco se imputarán costes en cuánto a software.

6.5.3 Costes del personal

A continuación se muestran los costes estimados en cuánto a personal para este proyecto. Se propone una jornada de 4 horas diarias, 5 días a la semana para el programador. Se considerarán las horas de reuniones con el director de proyecto como jornadas de analista, que serán 4 horas por reunión, una vez a la semana, jornada que también deberá cumplir el propio director de proyecto. Se incluyen también en esta planificación las horas de formación, que se asociarán a los dos primeros *sprints*.

El desglose total de horas y costes del personal se muestra en la tabla 6.3.

6.5.4 Coste final del proyecto

El resumen con el coste final del proyecto se puede ver en la tabla 6.4. Como se puede apreciar fácilmente, sólo se han computado los costes de los recursos humanos.

Capítulo 7

Arquitectura del sistema

En este capítulo se describe la arquitectura general del sistema y sus componentes principales. Es decir, el modelo de datos, las diferentes lógicas de la aplicación (almacenamiento y calibración) y, por último, la vista de la aplicación Android.

7.1 Estructura general

Nuestra aplicación está formada por 3 componentes claramente definidos (figura 7.1), tratando de seguir el modelo de arquitectura Modelo-Vista-Controlador (*MVC*).

La capa de Modelo permite representar los datos de entrada de nuestro sistema, que variarán desde datos de edificios, a escaneos de sensores del propio teléfono.

En la capa de controladores, dónde estará la lógica de la aplicación, nos encontraremos con la arquitectura que permite comunicar los datos del modelo, con la vista. Dentro de esta parte de la aplicación, estarán el gestor de edificios, el módulo de calibración, detección de planta y el encargado de calcular desplazamientos basándose en ARCore. Por su importancia para el proyecto, explicaremos en más detalle la lógica de almacenamiento y la de calibración, cada una por separado.

Por último, la parte de vista de la aplicación, será la representación visual de todos los datos obtenidos anteriormente. Permitirá además interacciones del usuario, que se comunicarán directamente con la capa de controladores.

A continuación, se hablará en más detalle de cada una de estas capas, desde el modelo de datos, hasta la interfaz de usuario, pasando por los elementos que componen la parte de la lógica del proyecto.

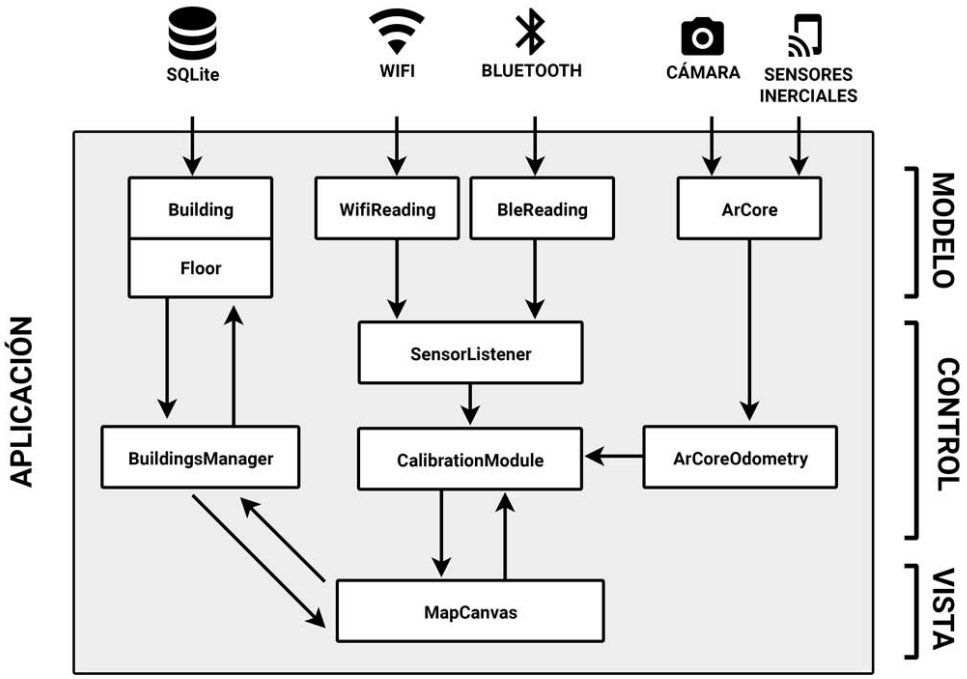

Figura 7.1: Esquema con la arquitectura general del proyecto.

7.2 Modelo de datos

En esta sección se hablará del tipo de datos con el que se trabajará en las distintas partes de la aplicación. Empezaremos explicando la información que se guarda sobre los edificios a calibrar, de los datos sensoriales obtenidos en los escaneos, de cómo se modela la odometría del sistema y, por último, el modelo de la información que se exporta a fichero.

7.2.1 Edificios

Para almacenar la información de cada edificio, se hace uso de una base de datos SQLite. Se tienen dos tablas, una dónde se almacena la información inherente del edificio, como el tamaño y el nombre, y otra tabla dónde se almacena la información de cada planta. Se muestra un diagrama con dicha configuración en la figura 7.2

7.2.2 Escaneos y datos sensoriales

A medida que el usuario avanza en la calibración, la aplicación se encargará de ir recibiendo señales de *wifi* y *bluetooth*. Cada uno de estos escaneos (*scans*) se almacenará en memoria, para ser exportado al final de la calibración.

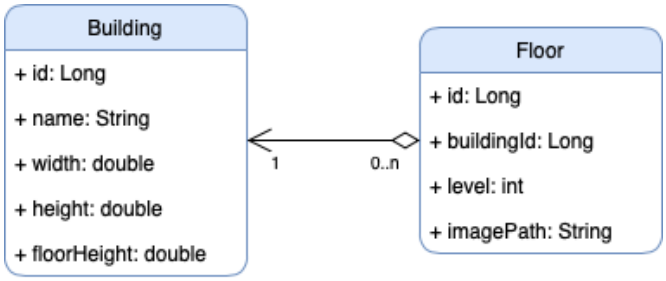

Figura 7.2: Estructura de la base de datos de edificios.

La información que se almacena de cada uno de estos escaneos será la siguiente:

- *timestamp*: Tiempo de llegada en milisegundos de la lectura del punto de acceso.
- *type*: Tipo de punto de acceso. Es un campo de tipo *string* que nos indica si se trata de una lectura de *wifi* o de *bluetooth*.
- *mac*: Representación hexadecimal de la *MAC* del punto de acceso.
- *identifier*: Representación decimal de la *MAC* del punto de acceso. Será el valor que se use para identificar de manera única a cada punto de acceso.
- *name/UUID*: En este campo, en caso de ser una lectura wifi, se guardará el nombre de *broadcast* del punto de acceso, y en caso de ser una lectura bluetooth, se guardará el UUID.
- *rssi*: Representa el valor de potencia de la lectura. Sus valores oscilan, de manera aproximada, entre -100dBm y -30dDm. Cuánto más cerca de 0 dBm, mayor será la potencia recibida.
- *frequency*: Este valor se usará sólo para las lecturas wifi. Nos indica la frecuencia de la señal, permitiéndonos discernir entre señales wifi de 2 o de 5GHz.

7.2.3 Odometría

Los datos que nuestra aplicación va a aceptar de la odometría seleccionada, que en este caso es ARCore, pero podría modificarse en un futuro, serán los siguientes:

- *timestamp*: Tiempo de llegada en milisegundos de la lectura de odometría.
- *x*: Desplazamiento en el eje x, en metros, calculado por la odometría.
- *y*: Desplazamiento en el eje y, en metros, calculado por la odometría.

- *z*: Desplazamiento en el eje z, en metros, calculado por la odometría. En caso de usarse una odometría en 2 dimensiones, este valor debería marcarse siempre como 0.
- *confidence*: Valor de confianza de los valores devueltos por la odometría. Como se comentará más adelante en el capítulo de implementación, ARCore sólo nos devuelve esta información de forma binaria, por tanto, los valores serán 0 o 1 en nuestro caso concreto.
- *eulerAngles*: Ángulos calculados para cada uno de los ejes del desplazamiento, representados en radianes.
- *quaternion*: Valores del cuaternión obtenidos del cálculo de odometría.

7.2.4 Exportación a fichero

Al finalizar cada calibración, se exportarán dos ficheros al espacio de almacenamiento del sistema. Estos ficheros estarán en formato CSV, ya que es fácil de manejar y convertir a otros formatos en caso de ser necesario para el usuario, a la hora de integrar con distintos proveedores de sistemas de posicionamiento.

Los dos tipos de ficheros que se exportan actualmente son:

- **Fichero de *GTs***: En este fichero, se almacenarán los datos de odometría con los mismos campos mencionados en la sección anterior. A mayores, se añadirán otras 3 columnas, con los valores de desplazamiento en las 3 dimensiones, pero corregidos. Es decir con las correcciones introducidas por el usuario y con las transformaciones aplicadas para que coincida con la posición y orientación inicial indicadas al inicio de la calibración.
- **Fichero de *scans***: De nuevo, se almacenarán los mismos datos mencionados en la sección de escaneos, pero añadiendo 3 columnas más. Estas columnas indicarán la posición dónde se realizó el escaneo, que surgirá de interpolar linealmente dicha lectura entre los dos *GTs* más cercanos, basándonos en el *timestamp* de cada uno.

7.3 Lógica de la aplicación (almacenamiento)

En esta sección se explicará la parte de lógica de la aplicación que gestionará la parte de almacenamiento y persistencia de los diferentes datos del sistema: edificios, ficheros (escaneos, posiciones, imágenes, etc.) y otro tipo de datos más simples.

7.3.1 Almacenamiento de datos de edificios

Como ya se comentó en la sección de requisitos, el usuario debe poder crear, editar y eliminar edificios. Además, dichos datos deben tener persistencia, por ese motivo se ha escogido una base de datos SQLite.

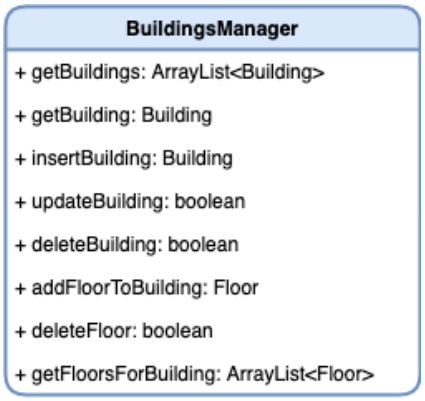

Figura 7.3: Clase «BuildingsManager.java».

Las acciones que el usuario pueda hacer sobre esa base de datos, estarán marcadas por las funciones definidas en nuestra clase `BuildingsManager.java`, ilustrada en la figura 7.3.

7.3.2 Almacenamiento de ficheros

Para poder interactuar con el almacenamiento de ficheros propio del sistema, se ha creado la clase `StorageManager.java`, que gestionará todas las lecturas y escrituras a fichero de nuestra aplicación.

Como ya se ha comentado en la sección de modelo de datos, esta aplicación exporta a ficheros csv la información obtenida durante el proceso de calibración. Dichos ficheros, ya que van a ser consultados por el usuario final de la aplicación, se deben almacenar en un lugar conocido y de fácil acceso para dicho usuario. Por ese motivo, se ha elegido la carpeta de descargas para este fin.

Hay otro tipo de ficheros que la aplicación va a manejar, y son las imágenes de planta de cada edificio. A la hora de crear una nueva planta, como ya vimos en la sección anterior, se almacena tanto el id, como el nivel, como la ruta al fichero de imagen. Dicha imagen se guarda en el almacenamiento privado que Android reserva para cada aplicación [21]. Se ha escogido almacenar en esta localización, ya que las imágenes de planta representan ficheros que nuestra aplicación usará internamente, y que el usuario no debería poder modificar a mano.

7.3.3 Almacenamiento de datos simples

Además de los dos tipos de almacenamiento mencionados anteriormente, habrá un tercero que nos permita guardar tipos de datos simples. Se trata de las *SharedPreferences* de *Android* [22]. En nuestro caso concreto, se ha creado una capa que encapsula el acceso de lectura y escritura a dicho almacenamiento, en nuestra clase `SharedPrefsManager.java`. Actual-

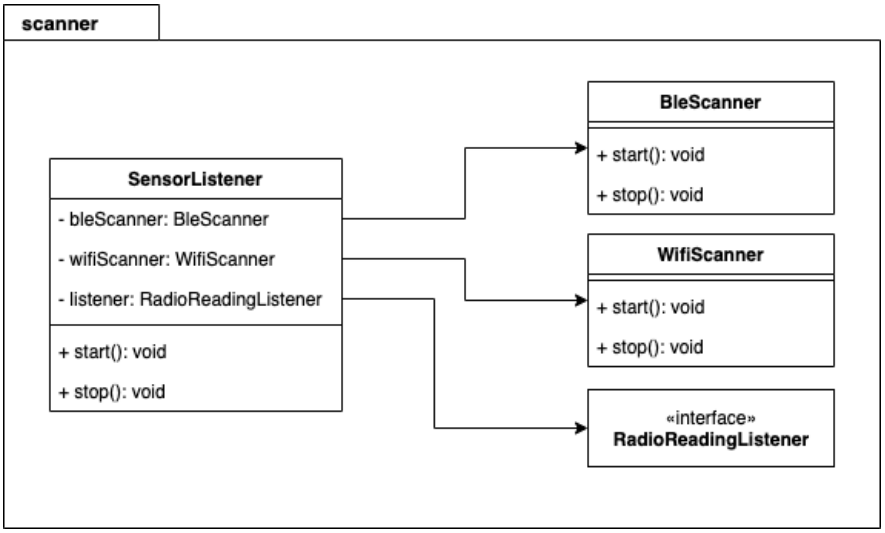

Figura 7.4: Contenidos del paquete «scanners» de nuestra aplicación.

mente sólo se almacena la información de si el tutorial ha sido visitado ya o no, para poder mostrarlo cuándo se inicie la aplicación por primera vez. En caso de crearse una pestaña de ajustes para la aplicación, dichos valores deberían almacenarse a través de esta clase.

7.4 Lógica de la aplicación (calibración)

A continuación se explicarán los módulos que se encargarán de la lógica de la aplicación, basándonos en el apartado de calibración. Empezaremos explicando cómo se escanean los datos sensoriales (señales de radiofrecuencia), cómo se puede detectar un cambio de planta y, por último, el módulo de calibración propiamente dicho.

7.4.1 Escaneo de señales

Está funcionalidad será gestionada por nuestra clase `SensorListener.java`, que se encargará de definir un *listener* para cada uno de los sensores que la aplicación necesite escanear. De esta manera, el resto de la aplicación se abstrae de los datos con los que la calibración funcione por detrás. Añadir nuevos sensores implicaría sencillamente dar de alta un nuevo listener en esta clase. Un diagrama ilustrativo de este módulo de puede encontrar en la figura 7.4

La frecuencia de escaneo en milisegundos de cada una de estas señales, vendrá proporcionada por el módulo de calibración, dónde a su vez, será configurada por el usuario a la hora de inicializar dicha clase.

7.4.2 Módulo de detección de cambio de planta

La gran mayoría de los sistemas de localización generan modelos de posicionamiento para cada planta del edificio de forma individual, por lo tanto, un punto muy importante para automatizar las tareas de calibración será detectar cuando el usuario ha cambiado de planta y realizar las acciones pertinentes (generar nuevos ficheros, etc.)

Cabe destacar que la detección de cambio de planta está limitada por las odometrías utilizadas y los sensores del teléfono móvil. Por ejemplo, empleando únicamente odometría visual, sólo podemos detectar cambios de altura al subir/bajar escaleras o rampas, dónde la cámara del móvil es capaz de detectar desplazamientos en el eje z en función del entorno, pero no subiendo o bajando en ascensor. Esto último sí se podría llegar a hacer con un sensor barométrico bien calibrado.

Para llevar a cabo esta detección, se ha implementado un módulo que recibe los desplazamientos en el eje Z de la odometría, para posteriormente calcular la probabilidad de cambio de planta. Se han probado distintos métodos para calcular de manera automática dichos cambios de planta, pero usando sistemas odométricos tan precisos como ArCore, se concluyó que los mejores resultados se obtenían basándose en la altura absoluta del sistema con respecto a la posición inicial.

El usuario debe ser el encargado de definir la altura entre plantas desde la aplicación, que por defecto se asumirá un valor de 3m. Se considerará que el usuario ha cambiado de planta cuándo su desplazamiento en el eje z con respecto a la posición inicial sea de +-3m menos un margen de altura. Dicho margen permite que el cambio de planta se complete ligeramente antes de que el usuario llegue a pisar la nueva planta.

Cómo margen de altura podría usarse el 50% de la altura total. De esta manera, todas las plantas quedarían calibradas en altura hasta la mitad de la siguiente planta. Sin embargo, se ha decidido ampliar ese valor, permitiendo de esta manera que haya solape entre las calibraciones de ambas plantas en zonas de transición (escaleras, rampas, etc). Asumiendo un ejemplo teórico en el que un usuario quiera cambiar de la planta 0 a la 1 y viceversa, (suponiendo que *h* es la altura actual del usuario, *h1* es la altura de la planta 1, y *h0* la altura de la planta 0) el cambio de planta al subir, se efectuará cuándo se cumpla la siguiente condición:

$$h >= (h1 - h0) - ((h1 - h0) * 0.2) \tag{7.1}$$

y al bajar cuándo se cumpla:

$$h <= (h1 - h0) + ((h1 - h0) * 0.2) \tag{7.2}$$

Un ejemplo ilustrativo de este margen de altura se puede observar en la figura 7.5.

Asumiendo que dicho sistema puede cometer errores, la decisión de cambio de planta no

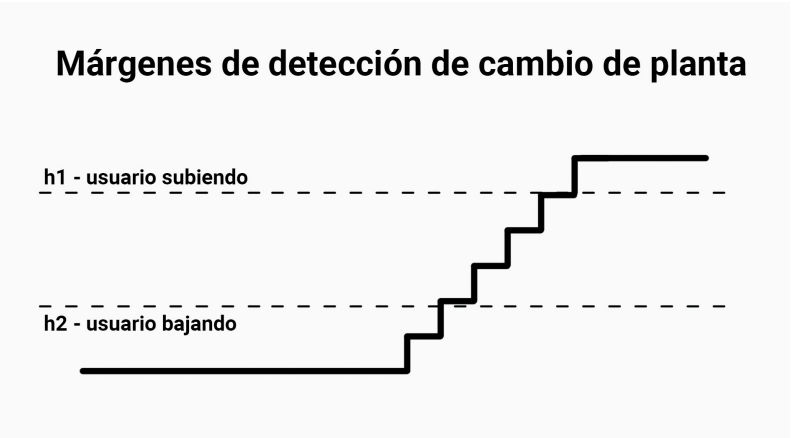

Figura 7.5: Ilustración de la zona de detección del cambio de planta en un edificio.

```
1 public enum FloorChangeDirection {UP, DOWN}
2
3 public interface FloorDetectionListener {
4     void onFloorChangeDetected(FloorChangeDirection flChDirection);
5 }
```

Figura 7.6: Interfaz «FloorDetectionListener».

se toma dentro del propio módulo de calibración. Se informa, a través de una interfaz, que se ha detectado un cambio de planta, y la dirección de dicho cambio (figura 7.6). Será entonces la aplicación la que gestione esa toma de decisión, en este caso, mostrando una alerta al usuario buscando confirmación.

7.4.3 Módulo de calibración

Este módulo será el responsable de calcular las posiciones de los escaneos wifi y bluetooth, y aplicar las correcciones del usuario recibidas a los desplazamientos de la odometría. Además, será el módulo central de la calibración, que gestionará el ciclo de vida de los demás módulos de cómputo, para así poder generalizar el ciclo de vida de cada calibración.

Entrada de datos

Este módulo permite entrada de dos tipos de GTs. GTs del usuario, y GTs de odometría. Internamente, este módulo sólo trabajará con una serie de GTs, y para ello tendrá que integrar ambas fuentes de información.

Para llevar a cabo este proceso, cada vez que llegue un GT de usuario, calculará la diferencia de traslación en ambos ejes entre dicho GT, y la última posición proporcionada por la

```
public interface CalibrationModuleListener {
    void onNewWifiScans(List<WifiReading> readings);

    void onNewBleScans(List<BleReading> readings);

    void onNewUserGt(GtPosition gt);

    void onNewOdometryGt(GtPosition gt);

    void onFloorChangeDetected(FloorChangeDirection flChDirection);
}
```

Figura 7.7: Interfaz «CalibrationModuleListener».

odometría. Aplicará a partir de entonces esa diferencia de posición a las futuras entradas de odometría.

Dado que la odometría proporciona desplazamientos relativos, necesitaremos saber de antemano la posición y orientación del usuario. Por ese mismo motivo, será necesario enviar dicha información para que el módulo de calibración pueda comenzar a funcionar. Tendrá un método público que permita mandar ambos valores:

`public void setInitialParams(PointF point, float direction);`

Cómo mencionamos anteriormente, este módulo será también el encargado de calcular la posición de cada uno de los escaneos de señales. Para obtener la posición en el espacio de cada uno de los escaneos, se interpolará linealmente entre los dos GTs de odometría corregida más cercanos, basándose en el tiempo de cada uno de ellos.

Salida de datos

Este módulo proporcionará una serie de datos mientras esté en funcionamiento. Para ello, se expondrá una interfaz pública que actuará a modo de listener.

A través de esta interfaz se enviarán datos como la posición actual del usuario (calculada a partir de los datos de odometría con las transformaciones iniciales aplicadas, así como las correcciones del usuario) o los escaneos de señales con la posición interpolada, cómo se puede observar en la figura 7.7.

A continuación se explican en más detalle los valores devueltos por esta interfaz:

- **Escaneos Wifi y Bluetooth**. Una vez realizada la interpolación de posiciones, el módulo enviará los datos a través de la interfaz pública, incluyendo el nombre del punto de acceso que se ha escaneado, la potencia, y la posición, entre otros valores.

- **GTs corregidos**. Cada vez que el módulo de calibración reciba un desplazamiento de odometría, se devolverá a través de la interfaz pública la posición absoluta del usua-

rio, calculada a partir de las correcciones que el usuario haya indicado a lo largo de la calibración.

- **Alerta de cambio de planta**. Cuándo se detecte un cambio de planta, se enviará un mensaje informando de ello, acompañado de la dirección del cambio.

Gestión de errores

A la hora de devolver datos de error, el módulo de calibración expondrá un método en la interfaz pública llamado `onError(Error error)`, dónde se enviará el error que se haya producido internamente, bien sea a la hora de realizar escaneos, o a la hora de procesar los datos de entrada.

A mayores, cuándo se intenten enviar datos de odometría o correcciones de usuario sin estar el módulo inicializado (es decir, sin haber indicado ni la posición inicial ni la orientación), se devolverá una `NotInitializedException`.

7.5 Vista

En esta sección se hablará de las distintas partes que componen la vista de la aplicación. Empezaremos por las diferentes actividades de la aplicación, el módulos específico para visualizar el proceso de calibración y, por último, el tutorial.

7.5.1 Actividades

La parte de la vista de la aplicación, estará compuesta por *activities* [23], que permitirán al usuario interactuar con el resto de módulos del sistema. Ya que cada *activity* trata una funcionalidad independiente de las demás, se ha considerado innecesario el uso de *fragments*.

La aplicación se descompone en cinco actividades, de las cuales tres suponen el flujo primario de uso de la aplicación. A continuación se explica en más detalle cada una de ellas.

- **MenuActivity**: Esta es la actividad principal, la primera que se abre cuándo el usuario inicia la aplicación. En ella se mostrará la lista de edificios creados. Al pulsar en cada uno de ellos, llevará a la actividad *BuildingDetailsActivity*, mandando el ID de edificio como campo adicional.

- **BuildingDetailsActivity**: Esta actividad, como su nombre indica, muestra los detalles del edificio seleccionado. El usuario podrá añadir o eliminar plantas, y editar los valores del edificio, tanto nombre como dimensiones, como altura entre plantas. Habrá también un botón que permita al usuario iniciar la actividad de calibración.

- **CalibrationActivity**: En esta actividad estará incluido el módulo de visualización que se explicará en la siguiente sección. Será la encargada de iniciar la odometría y el módulo de calibración cuándo el usuario lo desee, y actuar como puente entre dichos módulos y el componente de visualización.
- **TutorialActivity**: Esta actividad es la que engloba la funcionalidad del tutorial que se muestra cuándo se abre la aplicación por primera vez. También está accesible desde el menú superior derecho en la actividad *MenuActivity*.
- **AboutActivity**: Muestra una vista dónde se incluye el logo, el nombre y la versión de la aplicación, además del nombre del desarrollador. De nuevo, es accesible desde el menú superior derecho de *MenuActivity*.

7.5.2 Módulo de visualización

Ya que el proceso de calibración es un proceso tedioso y complejo, la interacción del usuario con la aplicación debe ser lo más simple e intuitiva posible. Por ese mismo motivo, se ha empleado tiempo en crear un módulo de visualización personalizado (*MapCanvas*).

Este módulo permite al usuario ver su trayectoria completa, pintada en tiempo real, con las correcciones que hayan sido aplicadas. Dicha trayectoria se pintará sobre el plano del edificio seleccionado, en la planta que el usuario indique en cada momento. Sobre la trayectoria también se pintarán las posiciones de cada una de las lecturas de los sensores wifi y bluetooth, con un color diferente para poder distinguirlas entre sí.

A mayores, centrándose en experiencia de usuario, se permitirá usar gestos globalmente conocidos, como utilizar un dedo para trasladar, o dos dedos para escalar la imagen. Las indicaciones visuales se mostrarán con alertas de tipo *Toast* [24]. Dichas indicaciones incluirán gestión de errores, o mensajes de éxito cómo cuándo un fichero de calibración ha sido guardado en el sistema de almacenamiento correctamente.

Adicionalmente, se indicará con un icono y una alerta personalizada en la parte superior izquierda cuándo se considere que el sistema odométrico esté devolviendo valores poco fiables. Los cambios de planta se indicarán a través de diálogos, buscando confirmación del usuario.

Un ejemplo visual de todo lo mencionado anteriormente se muestra en la figura 7.8.

7.5.3 Tutorial

Siguiendo en la línea de facilidad de uso, y como se ha comentado en la sección 7.5.1 se ha incluido un pequeño tutorial al inicio de la aplicación, para poner en contexto al usuario sobre qué va a encontrar cuándo abra nuestra aplicación. Se han utilizado ilustraciones llamativas, y animaciones sutiles, pero que permitan captar la atención del usuario cuándo abre nuestra aplicación por primera vez (figura 7.9).

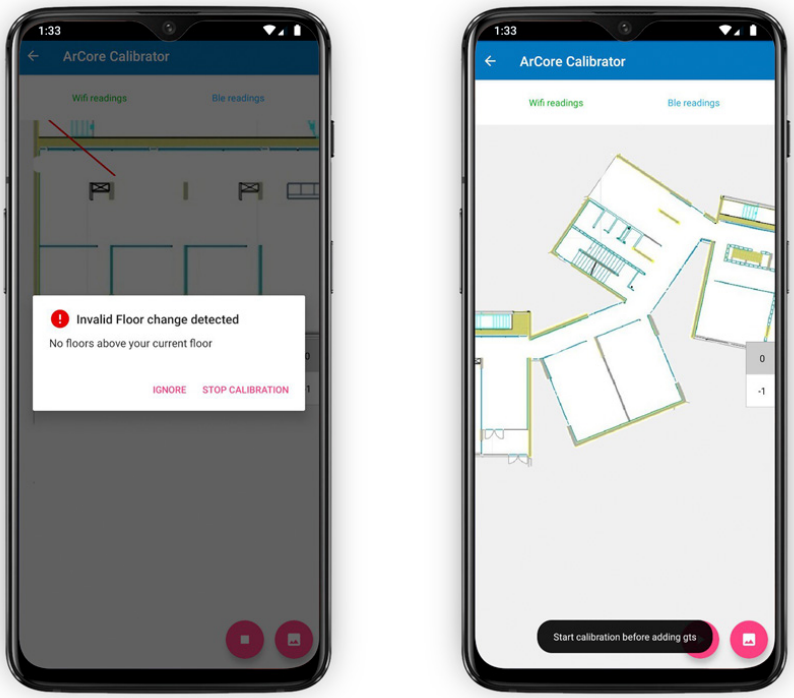

Figura 7.8: Ejemplos de indicaciones visuales en el módulo de visualización.

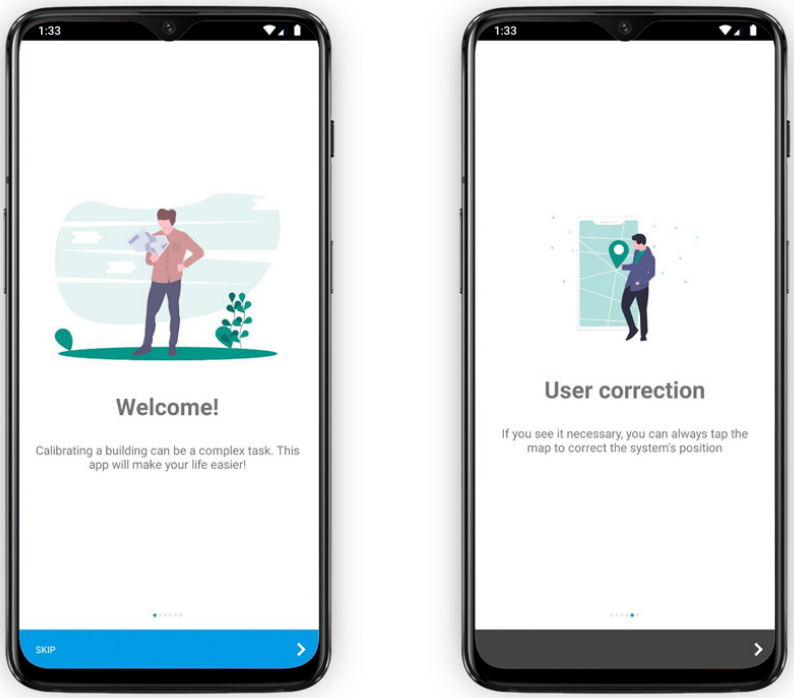

Figura 7.9: Tutorial sobre el funcionamiento de la aplicación.

Este tutorial consiste en un *ViewPager* [25] personalizado, dónde cada ventana es un elemento del mismo. Se ha implementado de tal manera que añadir una nueva página al tutorial implicaría sencillamente añadir un nuevo archivo XML con la vista de dicha página, en previsión de que se puedan añadir nuevas funcionalidades a la aplicación en un futuro.

Capítulo 8

Implementación

En este capítulo se recogen los detalles de la implementación llevada a cabo a lo largo del desarrollo de la aplicación. Nos centraremos en los aspectos más relevantes para nuestra aplicación: el proceso de calibración, la integración de la librería ArCore, el módulo de visualización y la experiencia global del usuario con la aplicación.

8.1 Módulo de calibración

En esa sección se explicará cómo se han llevado a cabo ciertos aspectos del módulo de calibración. Como ya dijimos en la parte de arquitectura, este módulo es el encargado de calcular la posición de las lecturas wifi y bluetooth, de aplicar correcciones del usuario a los datos de odometría (GTs), y de aplicar la rotación y posición inicial introducidas por el usuario.

8.1.1 Cálculo de correcciones de usuario

Ya que las correcciones de usuario simplemente se permiten en cuánto a traslación, el valor de dichas correcciones será la diferencia entre el GT introducido por el usuario, y la última posición de la odometría.

8.1.2 Transformación de odometría

Para transformar los datos de odometría en posiciones absolutas del usuario, hay que hacer una serie de transformaciones. Asumiendo que:

- p = posición marcada por odometría,
- $p0$ = posición inicial marcada por el usuario,
- $initialYaw$ = orientación inicial marcada por el usuario,

```java
public static PointF rotatePoint(PointF point, PointF origin, float
    rotation) {
    rotation = toRadians(rotation);
    double sin = Math.sin(rotation);
    double cos = Math.cos(rotation);

    // Translate to origin
    float translatedX = point.x - origin.x;
    float translatedY = point.y - origin.y;

    // Rotate around 0,0
    float newX = (float) (translatedX * cos - translatedY * sin);
    float newY = (float) (translatedX * sin + translatedY * cos);

    // Translate back to original position
    newX += origin.x;
    newY += origin.y;

    return new PointF(newX, newY);
}
```

Figura 8.1: Método «rotatePoint».

- userCorrection = corrección aplicada por el usuario al añadir nuevos GTs. El valor por defecto será 0 en caso de que no haya habido correcciones manuales.

El proceso que se ejecuta cada vez que llega un nuevo valor de odometría es el siguiente:

- En primer lugar, sumar a *p* la posición inicial marcada por el usuario como desplazamiento en ambos ejes.

```java
PointF t = new PointF(p.x + p0.x, p.y + p0.y);
```

- En segundo lugar, aplicar el valor de orientación inicial marcado por el usuario, para rotar la posición trasladada, usando la *p0* como punto de ancla. Para este fin, se hace uso de la función *rotatePoint()*, que está definida dentro de nuestro módulo de funciones útiles aritméticas (figura 8.1).

```java
PointF r = rotatePoint(t, p0, initialYaw);
```

- Por último, se deberá aplicar la corrección aplicada por el usuario.

```java
PointF c = new PointF(r.x + userCorrection.x, r.y +
    userCorrection.y);
```

```
private void handleFrame(Frame frame) {
    // Tracking state is our only valid source of quality
    TrackingState trackingState =
    frame.getCamera().getTrackingState();

    // Get translation data from ArCore
    Pose pose = frame.getAndroidSensorPose();

    // Swap y and z axis (ArCore uses Y as its vertical axis)
    Quaternion quaternion = new Quaternion(
            pose.qw(),
            pose.qx(),
            pose.qz(),
            pose.qy()
    );

    // Get euler angles from quaternion
    float[] eulerAngle = ArCoreUtils.toEulerAngles(quaternion);
    ...
}
```

Figura 8.2: Método «handleFrame».

8.2 ArCore

Como ya se ha mencionado anteriormente, ArCore es un kit de desarrollo ofrecido por Google. En general está orientado a desarrollar aplicaciones de realidad aumentada, pero también exporta métodos que permiten obtener desplazamientos relativos usando la cámara y los sensores inerciales de un teléfono móvil.

8.2.1 Obtención de datos

Para calcular los desplazamientos necesitamos hacer uso de un *handler*, con un intervalo periódico, que se encargará de obtener el último *frame* procesado por ArCore, y extraer la pose y otros valores de interés para nuestra aplicación. La única información sobre la rotación del teléfono obtenida de ArCore será el cuaternión, que deberemos transformar a ángulos de Euler si queremos trabajar con dichas medidas (figura 8.2).

8.2.2 Tracking state

La odometría que nos proporciona ArCore no nos devuelve un valor de confianza para cada uno de sus campos. Por lo tanto, no podremos estimar de forma cuantitativa la validez de los resultados ni, de llegar el caso, integrar adecuadamente esta odometrías con otras fuentes.

Por ese mismo motivo debemos basarnos en el *TrackingState* de ArCore para obtener la

```
defaultConfig {
    minSdkVersion 24
    ...
}
```

Figura 8.3: La mínima versión del sdk Android soportada por la aplicación es la 24.

calidad de los datos obtenidos del mismo. Según la documentación oficial de ArCore [26], disponemos de tres estados que nos ayudarán a determinar la fiabilidad del sistema:

- `TrackingState.PAUSED` - ArCore ha dejado de *trackear* correctamente, pero puede volver a retomar la sesión en un futuro. Cuándo estemos en este estado, no cortaremos la calibración, pero se ignorarán las posiciones recibidas, ya que podrían no ser fiables. Según la documentación de *Google*:

  > "When in this state the properties of the instance may be wildly inaccurate and should generally not be used."

- `TrackingState.STOPPED` - En este estado ArCore se ha perdido definitivamente y no reanudará la sesión.

- `TrackingState.TRACKING` - ArCore está funcionando de manera normal.

8.2.3 Previsualización

Otra dificultad a la hora de trabajar con ArCore, es que está pensado para aplicaciones de realidad aumentada, dónde la aplicación siempre va a mostrar en la pantalla del dispositivo lo que ve la cámara en todo momento. ArCore, y Google en general, pone muchas restricciones a la hora de usar las cámaras en segundo plano. En nuestro caso concreto, con una aplicación de calibración, no nos interesa demasiado que el usuario vea la salida de la cámara, ya que no aporta ningún tipo de información relevante y ocupa un espacio muy valioso para mostrar otros datos de interés.

ArCore necesita un context de OpenGl para el renderizado [27]. Usar ArCore sin mostrar la previsualización por pantalla implica crear un contexto de OpenGl oculto. Para ello se crearon las clases OpenGl y SessionCreator dentro del paquete de gestión ArCore.

8.2.4 Dispositivos soportados

Dado que la lista de dispositivos soportados por ArCore es relativamente pequeña [14], hay que gestionar todos los casos en los que un dispositivo no pueda usar nuestra aplicación.

```
private void checkArCoreCompatibility() {
    ArCoreApk.Availability availability =
        ArCoreApk.getInstance().checkAvailability(this);

    if (availability.isTransient()) {
        // Re-query while compatibility is checked in the background
        new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {
                checkArCoreCompatibility();
            }
        }, 200);
    }

    if (availability.isSupported()) {
        try {
            if (ArCoreApk.getInstance().requestInstall(this, true)
    == INSTALL_REQUESTED) {
                // The user has been requested to install ArCore
            } else {
                // ArCore is installed
            }
        } catch (UnavailableDeviceNotCompatibleException |
    UnavailableUserDeclinedInstallationException e) {
            // ArCore is not installed
        }
    } else {  // ArCore is not supported
    }
}
```

Figura 8.4: Código para comprobar si la librería ArCore está instalada en el dispositivo.

En primer lugar, debemos comprobar que la versión de Android sea superior a Android Nougat [28]. Por ese mismo motivo, definimos en la configuración del build.gradle la versión 24 del sdk de Android como la mínima soportada (figura 8.3). ArCore proporciona además un método (figura 8.4) para comprobar si el dispositivo está dentro de la lista de soportados, o si la aplicación propia de ArCore está instalada en el teléfono.

En caso de que efectivamente el dispositivo no esté dentro de la lista de soportados, o que la aplicación de ArCore no esté instalada, se le mostrará por pantalla al usuario indicando que la aplicación no podrá funcionar, como se puede ver en la figura 8.5.

8.3 Módulo de visualización

El módulo visualización se encarga de mostrar en tiempo real toda la información relevante durante el proceso de calibración, así como de gestionar toda la interacción del usuario. Inicialmente se estudió la posibilidad de usar librerías como Google Maps y así acelerar el

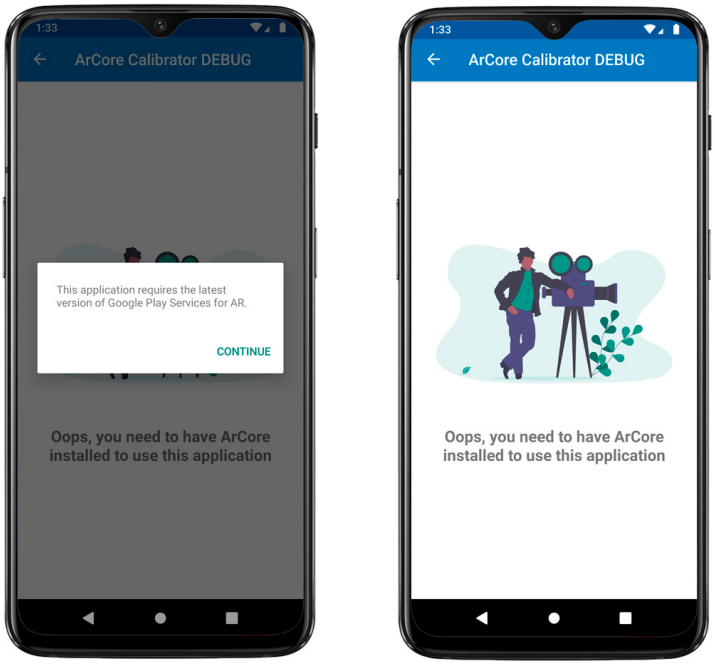

Figura 8.5: Mensaje que aparece en un dispositivo sin ArCore.

desarrollo del proyecto. Sin embargo, actualmente el nivel de zoom máximo para la API de Android de Google Maps no permite trabajar con una precisión suficiente para poder marcar puntos de referencia (GTs) sobre el mapa.

Por otra parte, si se usase Google Maps, antes de empezar el usuario debería indicar la localización y rotación del edificio con respecto al sistema de referencia terráqueo. Obviamente, ello implicaría bastantes pasos adicionales (muy tediosos) antes de comenzar la calibración de un nuevo edificio. Por todas estas razones se descartó emplear esta librería y se optó por crear un vista propia basado en un Canvas de Android, a pesar del coste que esta decisión conlleva.

Crear una vista personalizada extendiendo de la clase `View` de Android, implica que debemos gestionar a mano el pintado de nuestro componente, calculando todas las rotaciones, traslaciones, escalados, e interacciones del usuario con la vista.

8.3.1 Eventos de interacción

Para las interacciones del usuario con el mapa mediante el uso de gestos, se sobrescribe el método `onTouchEvent(MotionEvent event)` de la clase `View` de Android. Android registra las pulsaciones y gestos del usuario sobre la pantalla y hace una llamada a la función *onTouchEvent()* contra la *View* que considere que ha sido pulsada. La clase *MotionEvent* nos proporciona valores tan útiles como el tipo evento que ha sido lanzado, el índice del dedo que ha generado el evento, o la posición del mismo. Los tipos de eventos que vamos a gestionar

en esta vista personalizada son:

- `MotionEvent.ACTION_DOWN` - Este evento indica que el usuario ha pulsado la pantalla sin haber un puntero previamente pulsando. Este es el primer evento que se lanza cuándo el usuario inicia interacción con la pantalla. El puntero que ha iniciado este evento se considera el puntero principal.
- `MotionEvent.ACTION_POINTER_DOWN` - De nuevo indica que el usuario ha pulsado la pantalla, pero este evento se lanza cuándo ya hay otro puntero pulsando la pantalla. Con este evento comenzaría un evento multitouch.
- `MotionEvent.ACTION_MOVE` - Se lanza cuándo se detecta un evento de movimiento entre que un puntero pulsa y posteriormente se levanta de la pantalla.
- `MotionEvent.ACTION_POINTER_UP` - Este evento nos informa de que uno de los punteros ha sido levantado de la pantalla.
- `MotionEvent.ACTION_UP` - Este evento nos indica que la interacción del usuario con la pantalla ha finalizado.

8.3.2 Gestos

Nuestro módulo de visualización, utilizando la información de los eventos mencionados anteriormente, diferencia cuatro tipos de gestos:

- Pulsación prolongada + traslación - Cuándo el usuario pulsa la pantalla y mantiene el dedo en un mismo punto durante más de 500ms, se considera que está intentando lanzar un evento de larga pulsación. Este evento se utilizará para indicar la posición inicial del usuario al comenzar la calibración. Se marcará el inicio de la pulsación prolongada como la posición inicial del usuario.

  Posteriormente, sin levantar el dedo de la pantalla, el usuario deberá deslizar para marcar su orientación. Se dispararán múltiples eventos `MotionEvent.ACTION_MOVE`, que serán usados para actualizar el pintado de una línea sobre el mapa, facilitando al usuario la tarea de indicar su orientación.

  Cuándo se registre un evento `MotionEvent.ACTION_UP`, se calculará el ángulo entre la posición actual y la inicial de todo la acción. Se considerará que el gesto ha terminado, y se mandará la posición y orientación al módulo de calibración para poder inicializar.

  Para llevar a cabo la detección de la larga pulsación, se hará uso de un *Runnable* [29] de Android, se indica un delay de 500ms, y cuándo se ejecute el *runnable*, se considerará que el usuario ha mantenido pulsada la pantalla.

En caso de que haya cualquier otro evento en esa ventana de tiempo, se cancelará el *runnable*, y por tanto, el evento *long press* quedará descartado.

- Pulsación simple - Se considerará un gesto como pulsación simple, cuándo el usuario provoca un evento `MotionEvent.ACTION_UP` en un tiempo de menos de 500ms después haber pulsado la pantalla, sin haber eventos `MotionEvent.ACTION_MOVE` de por medio. Este tipo de gestos se usarán para que el usuario indique su posición real sobre el mapa durante el proceso de calibración, a fin de corregir al sistema odométrico si lo considera necesario.
- Traslación - Suponiendo que no se cumplan los casos anteriores, cada vez que se recibe un evento de tipo `MotionEvent.ACTION_MOVE`, se calcula la distancia entre la posición actual y la posición inicial del gesto, para poder aplicar dichas medidas de desplazamiento a todos los elementos de la vista.
- Escalado - El usuario podrá también escalar la vista mediante gestos. Para hacer esta acción intuitiva para el usuario, se usará el `GestureDetectorListener` [30] proporcionado por Android. De esta manera podremos almacenar el valor de salida del detector de escala, y aplicarlo cada vez que se recalcule la posición de los elementos en la pantalla.

8.3.3 Pintado y cálculo de dimensiones

Para llevar a cabo el pintado de datos sobre nuestro canvas, hay que tener claro en primer lugar en que unidades están cada uno de los datos. Al trabajar con una aplicación de calibraciones, tanto la salida de la odometría como las dimensiones de los edificios van a estar en metros. La vista sin embargo, trabajará en píxeles relativos a la pantalla del dispositivo, al igual que las interacciones del ususario con el teléfono. A la hora de realizar el pintado sobre nuestra vista personalizada, habrá que mantener una relación de escala que nos permita trabajar con metros y reflejarlos adecuadamente haciendo el cambio de unidades a píxeles.

La función que se encarga del pintado de la vista, `onDraw(Canvas canvas)` nos viene proporcionada por extender de la clase View de Android. Para poder trabajar con unidades en metros, deberemos primero aplicar las transformaciones necesarias al canvas, como se indica en 8.6. Estas mismas transformaciones tendrán que ser aplicadas, en orden inverso, a la hora de transformar las pulsaciones del usuario sobre la pantalla a metros.

8.4 Experiencia de usuario

Para mejorar la experiencia de usuario en la aplicación, se ha dedicado una parte del desarrollo a hacer una interfaz intuitiva y fácil de usar. Para ello, se han tomado distintas medidas.

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    // Paint all image grey
    canvas.drawARGB(255, 240, 240, 240);

    // Apply translation in both axis
    canvas.translate(totalTranslationX, totalTranslationY);

    // Scale canvas
    canvas.scale(mScaleFactor, mScaleFactor);

    // Now you can draw paths, scans, gt positions, and
    // building images with dimensions in meters
    ...
}
```

Figura 8.6: Parte del método *onDraw*() del módulo de visualización.

8.4.1 Ilustraciones

En diversas zonas de la aplicación, para mostrar mensajes de alerta, o para el tutorial mencionado anteriormente, se ha hecho uso de ilustraciones de una librería open-source [31]. Dichas ilustraciones permiten transmitir mensajes de forma más concisa y clara.

8.4.2 Respuesta del sistema

Basándose en la tecnología de HapticFeedback, se ha creado una clase que permite controlar la vibración del dispositivo. Se utilizarán pequeñas vibraciones en ciertas acciones del usuario como confirmación, por ejemplo al iniciar o parar calibraciones.

8.4.3 Vistas personalizadas

En varios lugares de la aplicación, se han creado distintas vistas personalizadas. Una de ellas, como ya se comentó anteriormente, es el tutorial de inicio de la aplicación, que extiende del componente ViewPager [25] original de Android.

El botón de crear edificios, es una vista personalizada que extiende de AlertDialog [32], que infla un diálogo con una animación de traslación y escalado.

Por último, la actividad de visualización de un edificio, contiene una pequeña vista, que muestra las dimensiones del mismo en metros, y se pinta en la pantalla con las proporciones adecuadas. Esta vista permite al usuario comprobar rápidamente si las dimensiones introducidas del edificio son realmente las deseadas.

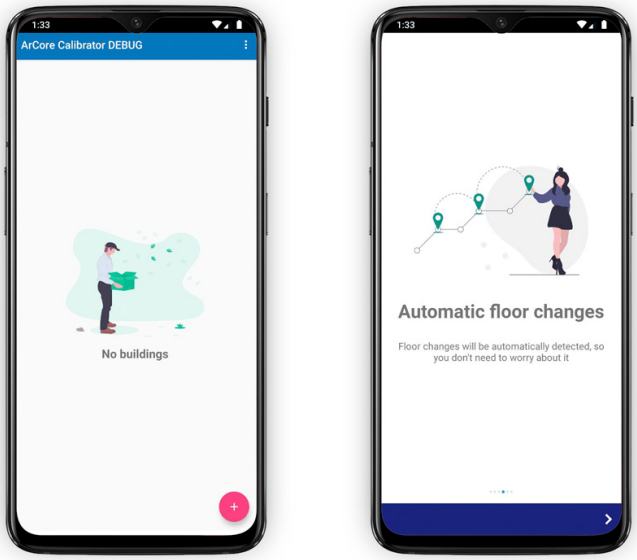

Figura 8.7: Ejemplo de ilustraciones de la librería *undraw.co*.

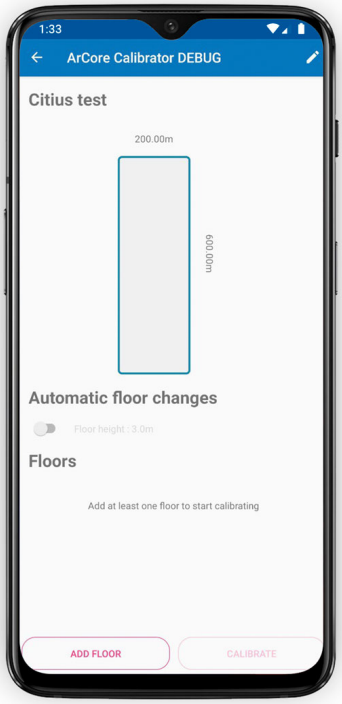

Figura 8.8: Ejemplo de la visualización de las dimensiones de un edificio.

Capítulo 9

Pruebas

En este capítulo se explicarán las pruebas llevadas a cabo para analizar la idoneidad de la odometría seleccionada, ArCore, así como las pruebas realizadas para generar modelos de posicionamiento en Situm.

9.1 Pruebas de final de sprint

Al finalizar cada sprint del proceso de desarrollo, se han hecho pruebas funcionales para corregir errores, mejorar aspectos de interfaz, o detectar mejoras en el funcionamiento de la aplicación.

Adicionalmente, con cada uno de los prototipos funcionales obtenidos de los sprints, se hicieron reuniones con el *product owner* para hacer pequeñas demostraciones.

9.2 Pruebas de funcionamiento de ArCore

Para valorar la idoneidad de *ARCore* en nuestro proyecto, se realizaron diversas pruebas al inicio del desarrollo. El primer prototipo de la aplicación consistía en una integración simple de la librería. Permitía visualizar con realidad aumentada, puntos sobre el espacio 3d marcando la trayectoria del usuario, como se muestra en la figura 9.1.

Sobre este prototipo de aplicación, se hicieron pruebas de rendimiento de la librería, comprobando la precisión del sistema, así como sus puntos fuertes y débiles. Se pudo comprobar que ARCore proporciona una odometría muy robusta en condiciones ideales, llegando a dar errores de tan sólo medio metro en recorridos largos, en edificios grandes como la facultad de informática.

Sin embargo, en entornos dónde las paredes y suelos tienen una textura muy homogénea, ARCore no consigue detectar correctamente el movimiento del usuario, resultando en valores inconsistentes de odometría.

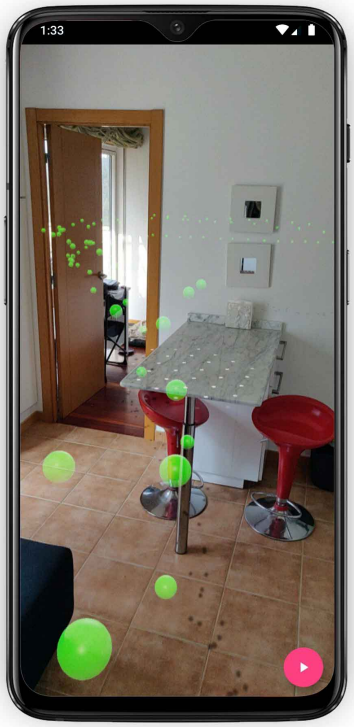

Figura 9.1: Primer prototipo de aplicación con integración de ARCore.

En ambientes de luz muy reducida, debido al sensor de cámara de los dispositivos móviles, ARCore tampoco consigue devolver resultados fiables. Cabe destacar, que en los entornos en los que haya poca luz en una zona concreta (por ejemplo, las escaleras de la facultad de informática), si se pasa a través de la zona poca iluminada en un intervalo corto de tiempo, el cálculo inercial de ARCore permite mantener la trayectoria de una manera relativamente fiable.

9.3 Pruebas en Situm

Para realizar las pruebas de posicionamiento, se ha hecho uso del *Dashboard* y de la aplicación *Mapping Tool* de *Situm*. Como primer paso se eligieron varios edificios para llevar a cabo las sesiones de calibración y posicionamiento. Se consiguieron las dimensiones y las imágenes de planta de los mismos. Con toda esta información pudimos incluir esos edificios en nuestra aplicación de calibración y proceder a realizar una serie de pruebas.

Para poder realizar las comparaciones, se creó una cuenta y se dieron de alta los edificios en el *Dashboard* de Situm. Se utilizaron las mismas imágenes y dimensiones que en nuestra aplicación de calibración.

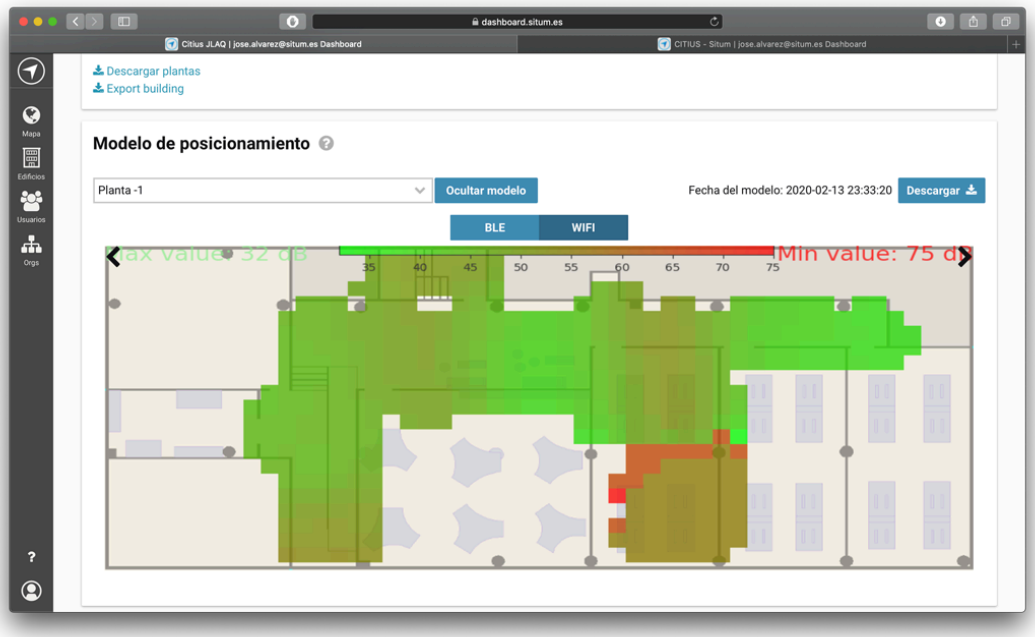

Figura 9.2: Ejemplo de visualización del modelo wifi en el *Dashboard* de Situm, generado con nuestra aplicación.

Al finalizar el proceso de calibración con nuestra aplicación, se subieron a la plataforma de Situm las calibraciones generadas. La primera comprobación fue observar si el modelo de posicionamiento se generaba de forma correcta. Para ello, accediendo a la pestaña de Configuración del *Dashboard* de Situm, visualizamos los modelos generados por Situm y también los GTs de la calibración. Como se puede apreciar en la figura 9.2, el modelo generado con nuestra aplicación parece correcto. Se hicieron las mismas calibraciones con la aplicación de calibración de Situm, para poder comparar modelos (figura 9.3). Se puede observar que el modelo generado con nuestra aplicación aparenta tener una mayor cobertura wifi en toda la planta. Esto se debe sencillamente a la escala de color en la visualización del Dashboard de Situm. Si nos fijamos en la figura dónde se muestra nuestro modelo (figura 9.2), el valor mínimo de señal, que aparecerá representado en rojo, son -75dB. En el caso de la calibración hecha con la aplicación de Situm (figura 9.2), la lectura de menor potencia, marcada con color rojo de nuevo, equivale a -55dB.

Una vez creados y calibrados los edificios, se descargó la aplicación *Situm Mapping Tool* en los teléfonos. Iniciando sesión con la misma cuenta que en el *Dashboard*, tenemos acceso a los edificios previamente calibrados. La siguiente prueba consistió en comprobar que la herramienta de posicionamiento "entrenada" con nuestras calibraciones, funcionaba correctamente. (9.4)

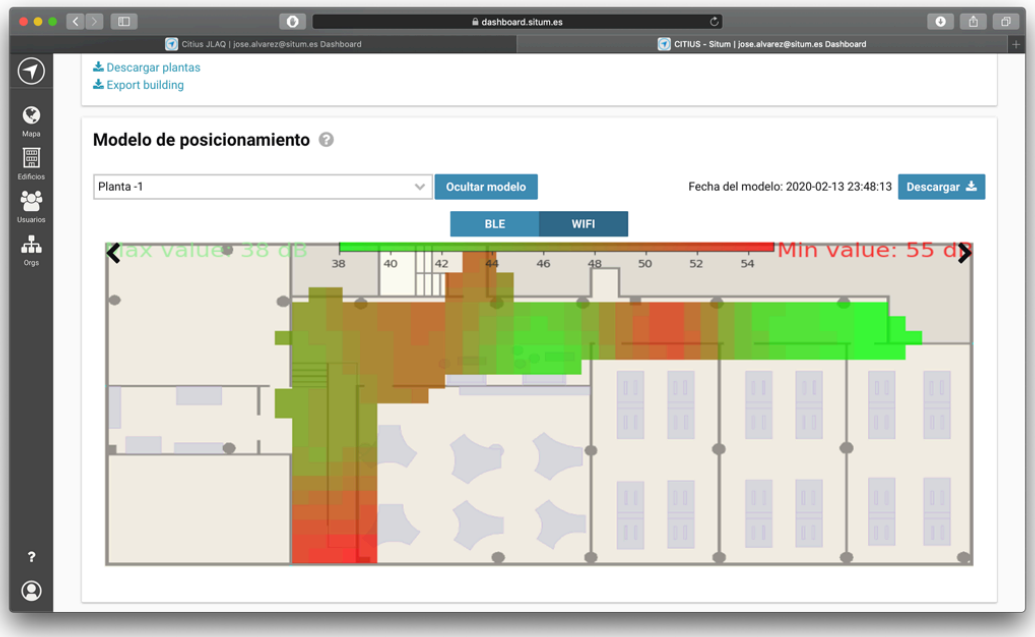

Figura 9.3: Ejemplo de visualización del modelo wifi en el *Dashboard* de Situm, generado con la aplicación de calibración de Situm.

9.4 Pruebas finales

Basándonos en distintas pruebas de funcionamiento llevadas a cabo para nuestra aplicación, con el mismo procedimiento que el descrito en "Pruebas en Situm", se han obtenido una serie de impresiones de nuestro sistema. Debido a que no podemos analizar el contenido de las calibraciones de Situm, la mayoría de estos criterios no tendrán una base puramente objetiva a la hora de comparar.

A continuación se comentan las conclusiones obtenidas tras calibrar el edificio del Citius de la Universidad de Santiago de Compostela. Veremos como varía el tiempo de calibración, la densidad de las medidas y los errores cometidos.

9.4.1 Tiempo de calibración

Para calibrar el Citius con la aplicación de Situm, hizo falta realizar varias calibraciones por planta, y parar e iniciar una nueva calibración en cada cambio de planta. También hay que resaltar que esta aplicación es necesario ir fijando constantemente la posición del usuario en el mapa para ir indicando el GT del dispositivo. Lo cual exige un considerable esfuerzo de concentración y, como ya se ha comentado, es una constante fuente de problemas y errores.

Con nuestra aplicación, sólo fue necesario un único recorrido por las cuatro plantas, sin

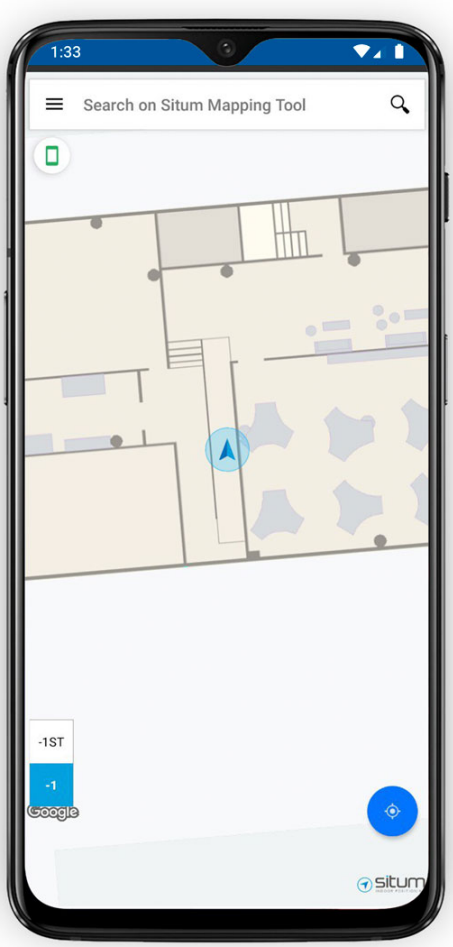

Figura 9.4: Ejemplo de dispositivo posicionado en un edificio calibrado con nuestra aplicación.

parar la calibración en cada cambio de planta y, sobre todo, sin tener que preocuparnos de ir corrigiendo constantemente nuestra posición en el mapa.

El tiempo de calibración del edificio entero con *Situm Mapping Tool* fue de aproximadamente **25 min**, mientras que con nuestra aplicación sólo hemos necesitado **9 min**. Es decir, una reducción muy considerable, cercana al 64%. Es de esperar que esta mejora sea aún mayor cuanto más grande o complejo (p.e. número de plantas) sea el edificio a calibrar.

9.4.2 Densidad de GTs

Para poder medir la densidad de las medidas de calibración, nos hemos centrado en la pasillo de la planta 0 del edificio (figura 9.5), descartando el laboratorio.

De media, en ese pasillo con la aplicación de Situm se añadieron unos **10 GTs** aproximadamente. Con nuestra aplicación se consiguió una densidad de alrededor de **200 GTs** en la

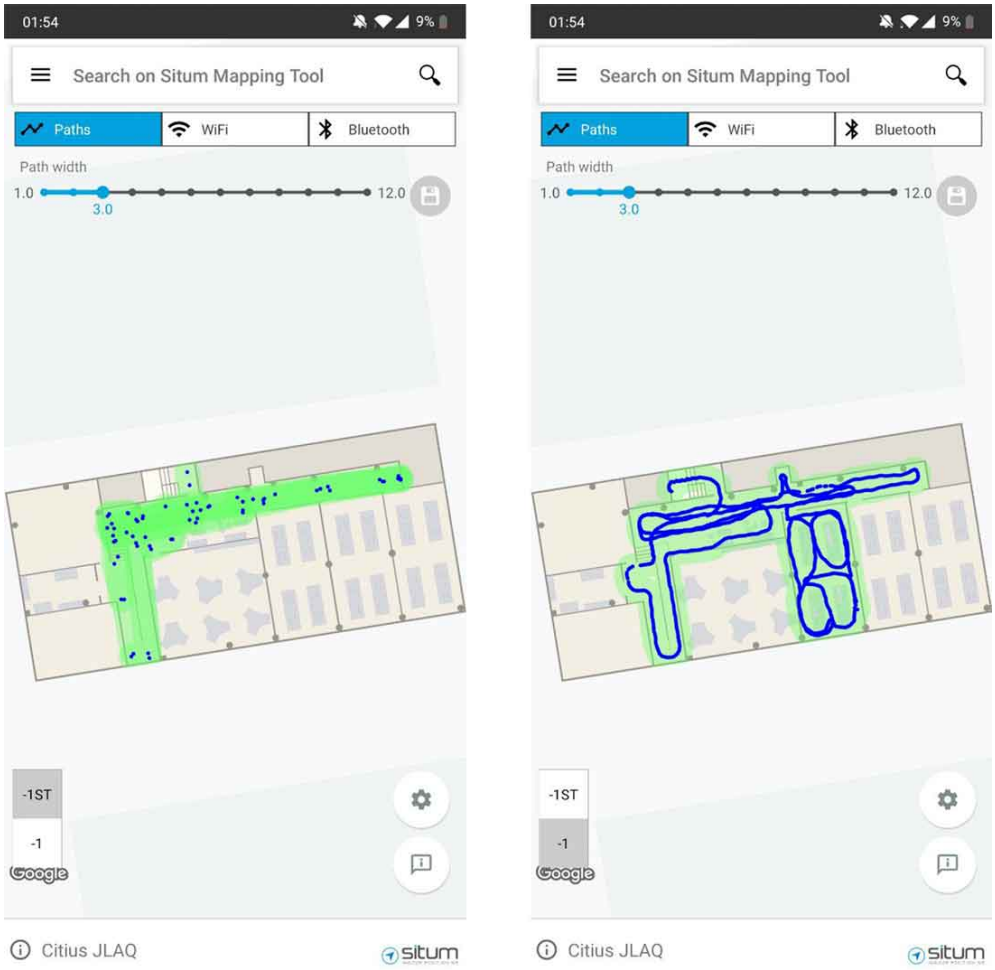

Figura 9.5: Resultados de la calibración del edificio Citius. A la izquierda, con Situm Mapping Tool; a la derecha, con nuestra aplicación.

misma distancia. Es decir, una mejora de un 1000%. Se puede apreciar visualmente la diferencia observando los GTs (puntos azules) de la figura 9.5.

Obviamente, con la aplicación de Situm sería posible aumentar en número de GTs, pero a costa de un esfuerzo mucho mayor y, seguramente, de introducir un mayor número de errores (p.e. por falta de precisión al tocar la pantalla).

9.4.3 Errores cometidos

Esta prueba se llevó a cabo en un entorno conocido, y bien iluminado, por lo que con ninguna de las dos aplicaciones se produjeron grandes errores. De todos modos, se tiene la sospecha de que en múltiples puntos de la trayectoria, se cometieron errores de varios metros con la aplicación de Situm al confundir columnas o puertas a la hora de interpretar el mapa

del edificio. Como no podemos acceder a los datos de las calibraciones de Situm, no podemos comprobar estas hipótesis.

Otro aspecto reseñable es la gran precisión de ArCore a la hora de determinar el movimiento del dispositivo en el entorno. Como se puede observar fácilmente en la figura 9.5, se siguen perfectamente todos los desplazamientos que realiza el usuario, sin tener que ajustar ni usa sola vez la posición estimada. Este resultado en particular nos hace pensar que sería posible crear directamente un mapa de un edificio sin necesidad de un plano inicial, muchas veces difícil de obtener.

9.4.4 Cambios de planta

Cabe destacar, que las pruebas de calibración realizadas en el Citius, explicadas en las anteriores secciones, se llevaron a cabo con la detección de planta automática.

En la figura 9.5, dónde se comparaba la densidad de GTs entre las dos aplicaciones, podemos observar en la imagen de la derecha, que durante toda la trayectoria realizada en la planta, no hubo falsos positivos de cambio de planta. En caso de haberlos, se habría cortado la calibración, y acabaríamos con dos ficheros en vez de uno, pudiendo detectar fácilmente dicho comportamiento.

Además, se puede observar cómo en la zona de las escaleras, se realiza el cambio de planta a poco más de la mitad de altura, confirmando el correcto funcionamiento de nuestro módulo de detección automática.

9.5 Pruebas de generación de mapas de ocupación

A pesar de no haber llegado a tiempo para llevar a cabo el sprint de generación de mapas automáticos, sí se llegó a realizar una prueba de concepto de la detección de planos, tanto verticales como horizontales. No se llegó a conseguir obtener las posiciones de dichos planos, y por tanto, medir la distancia del espacio libre, pero sí se consiguió renderizarlos sobre una vista de realidad aumentada de ARCore (figura 9.6).

Obviamente, hemos comprobado que la detección de planos funciona correctamente cuando hay grandes contrastes y texturas muy marcadas. Sin embargo y para nuestra sorpresa, ArCore detecta mucho mejor de lo que preveíamos superficies planas, aunque aparentemente no tengan mucha textura o un color muy marcado (p.e. paredes blancas).

De todos modos, hemos apreciado que en estos casos, la detección depende en gran medida de la distancia y orientación relativa entre el dispositivo y el objeto. Por otra parte, la velocidad del desplazamiento del usuario también influye negativamente. Por todo ello, será necesario realizar más estudios para ver si la detección de las paredes y, por tanto, la generación automática de mapas con obstáculos de los edificios durante el proceso de calibración es

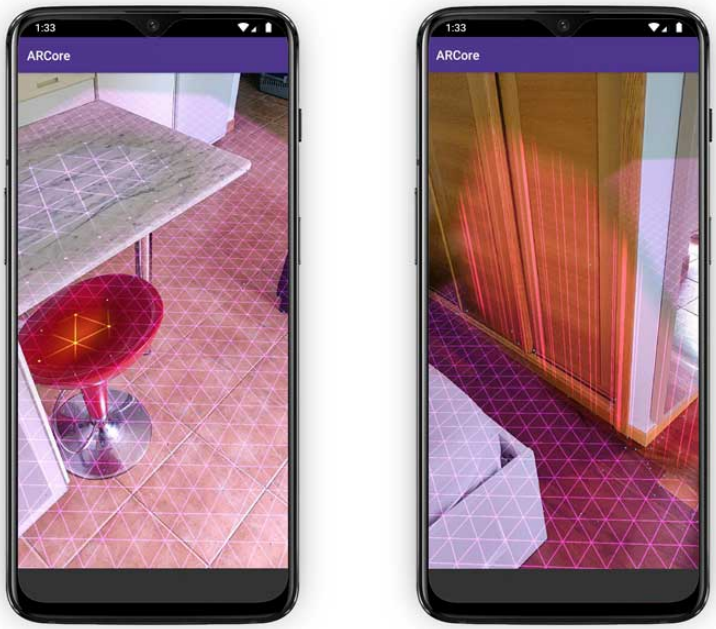

Figura 9.6: Prototipo de aplicación ArCore para detectar de planos verticales y horizontales.

viable.

Lo que si parece viable es “determinar” de forma aproximada cual es el espacio libre por donde se mueve el usuario durante la calibración y crear así versiones simplificadas de los mapas.

Capítulo 10

Conclusiones y trabajo futuro

En este último capítulo de la memoria, hablaremos sobre las conclusiones obtenidas a lo largo del desarrollo del proyecto. También se hablará de medidas a corto y largo plazo en cuánto a trabajo futuro.

10.1 Conclusiones

Este proyecto tenía como objetivo principal la mejora del proceso de calibración para sistemas de posicionamiento en interiores. A lo largo de esta memoria se ha explicado el proceso y el desarrollo llevado a cabo para cumplir con los objetivos definidos.

Como se ha podido observar en el apartado de pruebas, las mejoras con respecto a otras aplicaciones de calibración, son muy significativas. Especialmente en los apartados de tiempo de calibración, facilidad de uso, densidad de la medidas y precisión en el posicionamiento.

Concretamente hemos reducido el tiempo de calibración a más de un tercio del que se necesita con la herramienta de Situm. Al mismo tiempo también se ha aumentado la densidad de escaneos correctamente posicionados en un orden de magnitud, reduciendo enormemente la necesidad de interacción del usuario con la aplicación.

En la misma línea, se ha automatizado el proceso del cambio de planta durante la calibración. También se ha desarrollado y probado un módulo para detectar de forma automática los cambios de planta. Las pruebas realizadas han demostrado que con las herramientas existentes se pierde mucho tiempo en esta parte del proceso, provoca un gran estrés en el usuario y, además, es fuente de muchos errores. A veces tan graves que es necesario repetir toda la calibración de una planta.

Dentro de la propia aplicación, a lo largo de las distintas iteraciones, se han ido solucionando pequeños problemas y mejorando ciertas funcionalidades, para llegar al producto final con el que hemos finalizado esta fase del desarrollo y obtenido un primer prototipo totalmente funcional.

A continuación se muestra la lista de los requisitos inicialmente propuestos, y cómo se ha abordado cada uno de ellos de forma resumida.

10.1.1 Requisitos funcionales

- **Gestión de edificios y persistencia**. Los usuarios de la aplicación pueden, efectivamente, crear, editar y eliminar edificios, persistiendo dichos cambios en el dispositivo. Se han creado interfaces de usuario que permitan hacer todo este proceso más llevadero.

- **Cambio de planta sencillo**. Se ha implementado un módulo de detección de cambio automático, que el usuario puede usar de forma opcional. Con dicho módulo, el cambio de planta se hace de manera completamente transparente para el usuario.

  Aún con esta funcionalidad desactivada, la aplicación se encarga de gestionar los cambios de fichero de calibración cuándo el usuario indica manualmente los cambios de planta. Lo que reduce el esfuerzo y el tiempo en comparación con el resto de herramientas actualmente disponibles.

- **Corrección del sistema en caso de error**. Como ya se ha mencionado en diversas ocasiones a lo largo de la memoria, durante el proceso de calibración, el usuario, puede corregir la posición de la odometría siempre que lo crea conveniente.

- **Guardar ficheros en zona de fácil acceso**. Tal y como indicaba el requisito, los ficheros de calibración se guardan en la carpeta de descargas del dispositivo. Una carpeta de fácil acceso que no requiere permisos adicionales de lectura y escritura.

- **Visualización en tiempo real**. Como ya se comentó en los capítulos 7 y 8, se ha creado un módulo de visualización personalizado que permite cumplir con todos los requisitos establecidos al inicio del proyecto: ejecución en tiempo real, agilidad de respuesta al usuario, gestión de los gestos e interacciones y visualización de toda la información de interés durante la calibración.

10.1.2 Requisitos no funcionales

- **Interfaz intuitiva**. A lo largo del desarrollo, se ha hecho un gran esfuerzo por intentar buscar los componentes, ilustraciones, flujos de actividad e interacciones del usuario que permitiesen crear una interfaz más fácil de usar y más amigable.

- **Dispositivo compacto**. Como ya se planteó desde un primer momento, nuestra decisión de dispositivo compacto y portátil fue un teléfono móvil, dada la extensión de este tipo de tecnología entre la gran mayoría de personas, además de cumplir perfectamente con los requisitos establecidos.

- **Independencia de proveedor**. De nuevo, como ya estaba establecido desde un primer momento, la independencia de un proveedor de posicionamiento concreto primó a la hora de tomar una gran mayoría de las decisiones de diseño. Los ficheros de salida de esta aplicación son de fácil manejo, los datos de edificio están gestionados de forma interna, y no se hace subida automática a ninguna plataforma de ningún proveedor concreto.

  Sin embargo, también se ha testeado la correcta integración con algún proveedor de referencia, concretamente Situm. Demostrando que el prototipo desarrollado es viable y perfectamente integrable de ser requerido.

- **Fecha de entrega**. El proyecto se ha entregado en el plazo previsto, gracias a una correcta gestión del mismo y al empleo de metodologías ágiles de desarrollo.

10.2 Trabajo futuro

En esta sección se hablará de posibles ideas o líneas generales que se plantearían como desarrollo futuro en caso de disponer de tiempo adicional. Estas nuevas funcionalidades deberían seguir la misma metodología de desarrollo llevada a cabo a lo largo del proyecto, con iteraciones incrementales, y reuniones con los integrantes del proyecto para priorizar y analizar requisitos.

Las mejoras futuras planteadas se pueden agrupo en dos grandes líneas principales: la integración en sistemas IPS comerciales y las mejoras específicas de la aplicación.

10.2.1 Integración en herramienta de calibración de IPS

Uno de los caminos más prometedores para este proyecto, sería su integración dentro de la herramienta propietaria de algún proveedor de servicios de IPS, en vez de seguir siendo una aplicación independiente. Esto nos permitiría tener acceso a toda la infraestructura de gestión de edificios, autenticación de usuarios y subida automática de los ficheros de calibración a los servidores *backend*.

A la hora de integrarlo, tal y cómo se ha diseñado la arquitectura de este proyecto, no implicaría grandes cargas en cuánto a desarrollo, ya que se ha llevado a cabo manteniendo la escalabilidad e independencia entre módulos como un factor importante a tener en cuenta, previendo que el futuro del proyecto apuntase hacia esta vertiente.

10.2.2 Mejoras de la propia aplicación

La aplicación presentada tiene margen de mejora en varios aspectos. También se podrían estudiar y añadir nuevas funcionalidades.

- **Generación de mapas automáticos**. Retomar el estudio de la generación automática de mapas usando la detección de planos de ARCore. Esto permitiría al usuario comenzar a calibrar un edificio sin tener ningún tipo de conocimiento del entorno. Sencillamente abrir nuestra aplicación, apuntar con la cámara hacia delante y andar. Como ya se ha comentado, los estudios preliminares apuntan a importantes limitaciones que habría que testear con mayor profundidad o esperar a que mejore la librería ARCore.

- **Mejoras en el módulo de visualización**. Este módulo es una parte muy importante de la aplicación como una herramienta independiente. Sin embargo, no se ha invertido mucho esfuerzo en su desarrollo por si el prototipo es finalmente integrado en otras aplicaciones, con su propias interfaces. De todos modos, algunas mejoras que se podrían acometer podrían serían: pulir la fluidez de los gestos del usuario, añadir la opción de rotación mediante gestos, centrar la vista en la posición del usuario, etc.

- **Almacenamiento de edificios**. Para mejorar la experiencia de usuario y la compartición de información, se plantearía el uso de una base de datos remota para almacenar los datos de edificio. De esta forma, varios usuarios podrían calibrar un mismo edificio sin tener que crearlo cada uno desde cero en su dispositivo. Esto implicaría creación y gestión tanto de usuarios como de permisos de acceso a cada edificio. Herramientas como Firebase facilitarían estos desarrollos.

- **Corrección de odometría**. A la hora de corregir las posiciones de la odometría, el usuario puede pulsar la pantalla para indicar su posición real. El sistema entonces ajusta la odometría basándose en traslación. Una posible mejora sería añadir la opción de ajustar también la orientación inicial al añadir GTs al mapa.

  En caso de implicar un desarrollo largo, otra opción sería añadir una pequeña herramienta dentro de la propia aplicación, que permitiera modificar calibraciones a "posteriori". El objetivo sería aplicar transformaciones geométricas básicas (escalado, rotación o traslación) para ajustar la trayectoria obtenida al plano del edificio.

- **Integración de otras fuentes de odometría**. En este proyecto ha primado la portabilidad del dispositivo final. Sin embargo, es sistema se ha diseñado para que se puedan añadir y/o fusionar otras fuentes de odometría.

Apéndices

Apéndice A

Despliegue automático

A la hora de compilar y distribuir la aplicación entre los miembros del equipo de desarrollo, o entre distintos dispositivos, se ha tratado de buscar un mecanismo que reduzca tiempos y facilite el proceso.

Para ello, ya que el código del proyecto se encuentra alojado en *Gitlab*, se ha hecho uso de la integración continua, servicio que *Gitlab* ofrece de manera gratuita a sus usuarios. Se ha creado un archivo *yaml* de configuración, que permita detectar cambios en la rama *master* del repositorio, compile la aplicación en versión de *release*, lea un archivo de *changelog* disponible en el repositorio, y utilice esa información para publicar en el medio de comunicación escogido por el equipo de desarrollo. Dicho medio de comunicación, en este caso, fue *Telegram*. Se hizo uso de la librería de *bots* públicos, lo que nos permitió crear un canal dónde se publican todas las nuevas compilaciones de la aplicación, con la versión *release*, código ofuscado, y con un pequeño resumen de los cambios que incluye. Se incluye una captura de pantalla de dicho canal en la figura A.1

Dicho archivo de configuración tiene los siguientes campos:

- *image* : Indica la imagen de docker que queremos usar para la compilación de nuestro proyecto

```
image: openjdk:8-jdk
```

- *variables* : Define las variables de entorno que se usarán a lo largo de la compilación

```
variables:
  ANDROID_COMPILE_SDK: "27"
  ANDROID_BUILD_TOOLS: "28.0.3"
  ANDROID_SDK_TOOLS:   "4333796"
```

- *before_script*: define las acciones que se realizarán antes de ejecutar el script

```
before_script:
  - cd Android/ArCoreApplication/
  ... Descarga de Android SDK tools, platform tools, y demás
herramientas necesarias
```

- *stages*: define cada una de las etapas a ejecutar. En nuestro caso tendremos dos, una etapa de tests, y una etapa de compilación y despliegue.

```
stages:
  - test
  - build
```

- *debugTests*: Esta es la tarea que queremos que se ejecute en el stage *tests*. Sencillamente ejecutará los tests definidos en la aplicación Android, y fallará el pipeline si no se superan los tests correctamente.

```
debugTests:
  stage: test
  script:
    - ./gradlew -Pci --console=plain :app:testDebug
```

- *assembleRelease*: Esta será la tarea del stage *build* encargada de compilar la aplicación, y desplegarla en nuestro bot de telegram. El campo *only:* nos permite seleccionar sólo la rama master para que se ejecute este stage.

```
assembleRelease:
  stage: build
  script:
  - ./gradlew assembleRelease
  - cd ../..
  - mkdir dist
  - mv
Android/ArCoreApplication/app/build/outputs/apk/release/* dist/
  - mv
Android/ArCoreApplication/app/src/main/res/raw/changelog.txt
dist/changelog.txt
  # Send this file to telegram bot
  - sh uploadTelegram.sh
  artifacts:
    paths:
    - dist
  only:
    refs:
      - master
```

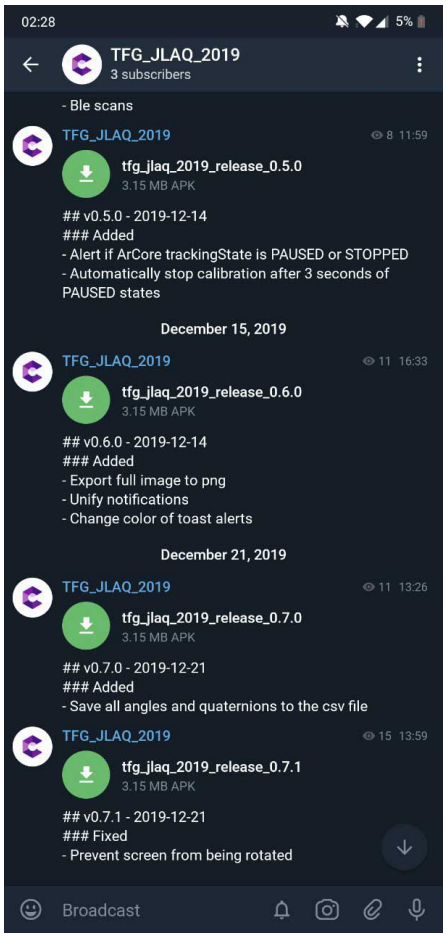

Figura A.1: Captura de pantalla del canal de Telegram donde se han publicado y compartido las nuevas versiones de la aplicación.

Apéndice B

Contenido del DVD

En el DVD adjunto en la entrega final de este proyecto, se incluyen los siguientes ficheros:

- Una copia de esta memoria en formato PDF.
- La última versión de la aplicación, en versión release, firmada, y con el código ofuscado.
- El código fuente del proyecto de la aplicación, comprimido en un archivo *zip*.
- Código fuente de esta memoria.
- Archivo zip con imágenes de planta y medidas de varios edificios, para poder llevar a cabo pruebas con nuestra aplicación.

Apéndice C

Manual de uso

En este apéndice se explicarán los pasos a reproducir para poner en funcionamiento la aplicación tras la primera instalación.

C.1 Requisitos

El uso de esta aplicación requiere de un dispositivo móvil que cumpla con las siguientes características:

- Sistema operativo Android 7.0 o superior.
- Debe estar entre la lista de dispositivos soportados de ARCore [14].
- *Google Play AR services* instalado. Disponible en *Google Play Store* en el siguiente enlace: `https://play.google.com/store/apps/details?id=com.google.ar.core`
- Al menos 10 MB de almacenamiento disponible.

C.2 Manual de usuario

C.2.1 Tutorial de inicio

Al iniciar la aplicación por primera vez, se nos mostrará por pantalla un pequeño tutorial para poner en contexto sobre qué trata nuestra aplicación. Deslizando hacia la izquierda, o pulsando la flecha en la esquina inferior derecha, podremos avanzar hacia el siguiente item del tutorial. Al llegar al último, el botón de avanzar cambiará a un botón de finalizar.

Una vez cerrado el tutorial, no volverá a aparecer la siguiente vez que abramos la aplicación. Sin embargo, se podrá volver a consultar manualmente en cualquier momento, desde el menú superior derecho en la página principal de nuestra aplicación.

C.2.2 Creación de edificio

Una vez terminado el tutorial, se nos presentará una pantalla con una indicación de que no hay edificios creados, y un botón con un símbolo + en la esquina inferior derecha. Para añadir un nuevo edificio, pulsar en dicho botón, e introducir los datos deseados. (Ver figura C.1

Se ha incluido en el DVD adjunto a esta memoria, un archivo *zip* con las imágenes de planta y las dimensiones de edificios como las Facultades de Informática y Caminos de la Universidade de A Coruña, así como del edificio Citius de la Universidade de Santiago de Compostela.

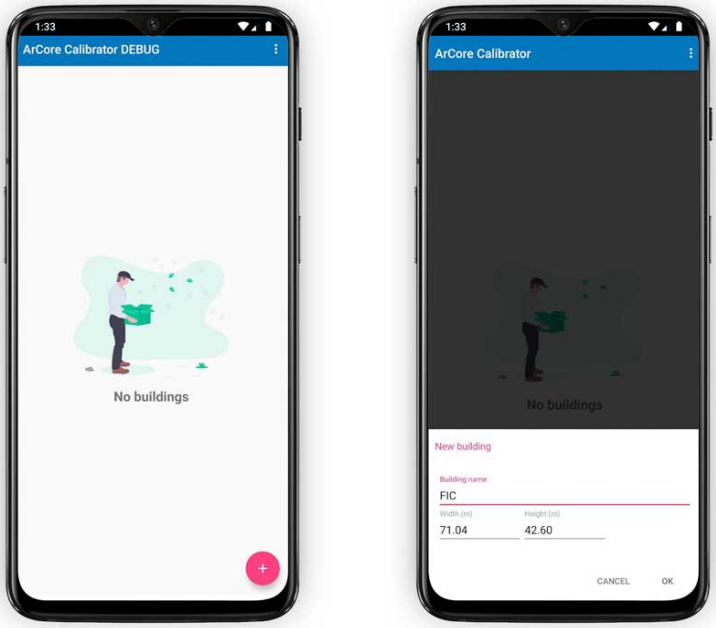

Figura C.1: Menú inicio aplicación y creación de nuevo edificio.

C.2.3 Pantalla detalles de edificio y creación de plantas

Una vez creado el edificio, se nos mostrará una pantalla con todos los detalles del mismo. Se muestran una serie de botones y acciones:

- Para editar nombre, dimensiones o altura entre plantas, se deberá pulsar el icono de editar, arriba a la derecha.
- Para añadir una nueva planta, se pulsará el botón *"Add Floor"*, abajo a la izquierda. Nos pedirá que seleccionemos el nombre de la planta, y que seleccionemos un archivo de imagen para la misma. Este proceso se ilustra en la figura C.2

- Botón de calibrar, abajo a la derecha. Este botón llevará a la actividad de calibración para este edificio. Como acción previa a pulsar este botón, en caso de que queramos activar el cambio de planta automático, habrá que activar dicha funcionalidad en la pantalla actual, en el apartado de *"Automatic Floor Changes"*. Este botón estará deshabilitado en caso de que el edificio seleccionado no tenga plantas creadas.

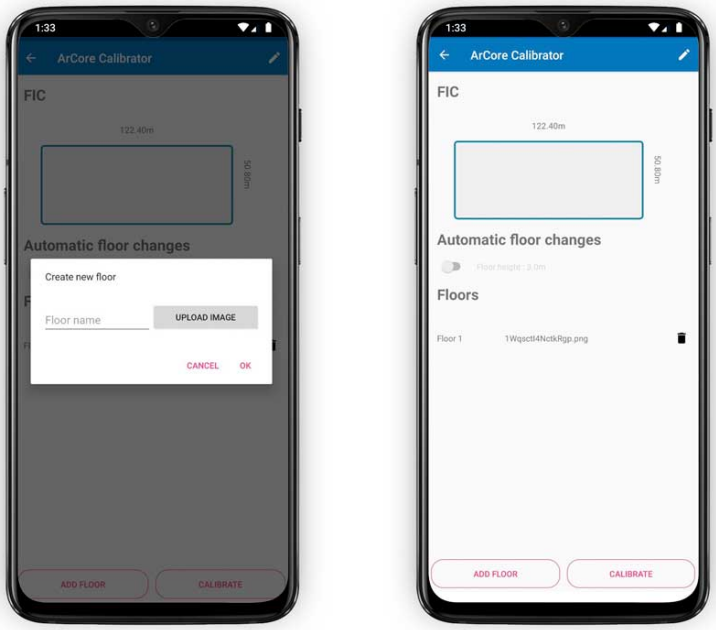

Figura C.2: Creación de planta.

C.2.4 Calibración del edificio

Una vez en la pantalla de calibración, se nos mostrará dibujada la imagen de planta seleccionada, dos botones abajo a la derecha, un selector de planta en el lateral derecho, y un texto en la parte superior que indica en qué color se representarán cada una de las lecturas.

A partir de Android 9, Android aplica limitaciones a los escaneos wifi que cada aplicación puede llevar a cabo en un período de tiempo. Si el dispositivo con el que se realizan las pruebas, tiene Android 9, hay que asumir que sólo se podrán obtener lecturas wifi cada 30 segundos o más. En caso de ser Android 10, se puede desactivar esta limitación desde los ajustes de desarrollador [33].

Para comenzar a calibrar, el flujo de acción es el siguiente:

1. Buscar nuestra posición actual sobre el mapa, seleccionando la planta adecuada.
2. Mantener pulsada la pantalla en nuestra posición actual, y deslizar para indicar nuestra orientación.

3. Asegurarse de no tapar la cámara del dispositivo, y pulsar el botón con el icono de *play*.

4. Comenzar a andar.

A medida que avanzamos sobre el mapa, veremos cómo se pinta nuestra trayectoria en color rojo. Cada una de las lecturas wifi durante dicho recorrido, se pintarán como puntos de color verde, indicando la posición exacta dónde se llevaron a cabo, y lo mismo con los datos de bluetooth, pero representados en color azul.

En caso de discrepancias con la posición indicada en la aplicación, el usuario puede pulsar el mapa para corregir al sistema, indicando su posición real sobre el mapa. Una vez hecho esto, el sistema almacenará la corrección introducida y la aplicará a las futuras lecturas de odometría.

Al finalizar la trayectoria, se pulsará el botón con el icono de *stop*. En ese momento, ARCore dejará de funcionar, y se guardarán en la carpeta de descargas del dispositivo dos ficheros, uno con las posiciones del usuario de la trayectoria completa, y otro con todos los escaneos realizados, ordenados por tipo de sensor.

Lista de acrónimos

AR *Augmented Reality.*

BLE *Bluetooth Low Energy.*

CSV *Comma-Separated Values.*

GT *Ground Truth.*

IPS *Indoor Positioning System.*

MAC *Media Access Control address*

MVC *Model View Controller.*

RSSI *Received Signal Strength Indicator.*

UUID *Universally Unique IDentifier.*

UX *User Experience.*

Glosario

Ground Truth (GT) Posición fiable normalmente introducida por el usuario a mano.

Cuaternión Los cuaterniones son extensiones de los números reales, similares a los números imaginarios, pero añadiendo los componentes i, j y k.

Backend Parte de un sistema informático, normalmente no accesible al usuario final, que se encarga de la gestión de datos.

Espacio libre En lo relacionado a sistemas de posicionamiento en interiores, se consideran espacio libre aquellas zonas del edificio que son transitables para un usuario, obtenidas a partir de las calibraciones llevadas a cabo. El algoritmo de cálculo de posición excluirá las áreas que no se consideren espacio libre, por tanto, nunca debería devolver una posición fuera de las zonas calibradas de un edificio.

Bibliografía

[1] "Mobile operating system market share worldwide," Último acceso: 12 de febrero de 2020. [En línea]. Disponible en: https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] "Herramienta de calibración, localización y navegación de *Situm*," Último acceso: 28 de diciembre de 2019. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=es.situm.maps&hl=es

[3] "*Situm Indoor Positioning*," Último acceso: 21 de diciembre de 2019. [En línea]. Disponible en: https://situm.es/es

[4] "Manual usuario *Situm*," Último acceso: 28 de diciembre de 2019. [En línea]. Disponible en: https://dashboard.situm.es/manuals/Esquema_Manual_App.pdf

[5] "*HERE Indoor Radio Mapper*," Último acceso: 9 de febrero de 2020. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.here.hirmu&hl=en

[6] "*Indoor Here*," Último acceso: 27 de diciembre de 2019. [En línea]. Disponible en: https://indoor.here.com/

[7] "*Indoor Here*: Collecting and managing radio data," Último acceso: 9 de febrero de 2020. [En línea]. Disponible en: https://indoor.here.com/#/tools/help-text-3

[8] "*HERE Indoor Radio Mapper* tutorial," Último acceso: 9 de febrero de 2020. [En línea]. Disponible en: https://www.youtube.com/watch?v=Ysw63DPCYQM

[9] "*IndoorAtlas MapCreator 2*," Último acceso: 22 de diciembre de 2019. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.indooratlas.android.apps.jaywalker

[10] "*Indoor Atlas*: Mapping with *Map Creator 2*," Último acceso: 22 de diciembre de 2019. [En línea]. Disponible en: https://www.youtube.com/watch?v=kTFxvTrcYcQ

[11] "*GPS*: The global positioning system," Último acceso: 9 de febrero de 2020. [En línea]. Disponible en: https://www.gps.gov

[12] "Robot odometry," Último acceso: 11 de febrero de 2019. [En línea]. Disponible en: https://groups.csail.mit.edu/drl/courses/cs54-2001s/odometry.html

[13] D. Scaramuzza and F. Fraundorfer, "Tutorial: Visual odometry," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, 12 2011.

[14] "List of *ArCore* supported devices," Último acceso: 30 de enero de 2020. [En línea]. Disponible en: https://developers.google.com/ar/discover/supported-devices

[15] "Ikea place," Último acceso: 30 de enero de 2020. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.inter_ikea.place&hl=en

[16] "Amazon ar view," Último acceso: 30 de enero de 2020. [En línea]. Disponible en: https://www.amazon.com/adlp/arview

[17] "Sqlite," Último acceso: 9 de febrero de 2020. [En línea]. Disponible en: https://www.sqlite.org/index.html

[18] "*Situm Dashboard*," Último acceso: 21 de diciembre de 2019. [En línea]. Disponible en: https://dashboard.situm.es

[19] "What is scrum?" Último acceso: 30 de diciembre de 2019. [En línea]. Disponible en: https://www.scrum.org/resources/what-is-scrum

[20] R. d. l. H. d. D. Alonso Álvarez García, *Métodos Ágiles y Scrum*, 1st ed. Anaya Multimedia, 2012.

[21] "Access app-specific files on *Android*," Último acceso: 12 de febrero de 2020. [En línea]. Disponible en: https://developer.android.com/training/data-storage/app-specific

[22] "*Android shared preferences*," Último acceso: 12 de febrero de 2020. [En línea]. Disponible en: https://developer.android.com/reference/android/content/SharedPreferences

[23] "Introducción a las *actitivies* en android," Último acceso: 11 de febrero de 2020. [En línea]. Disponible en: https://developer.android.com/guide/components/activities/intro-activities

[24] "*Toast* alerts on *Android*," Último acceso: 9 de febrero de 2020. [En línea]. Disponible en: https://developer.android.com/guide/topics/ui/notifiers/toasts

[25] "*Android ViewPager* component," Último acceso: 9 de febrero de 2020. [En línea]. Disponible en: https://developer.android.com/reference/android/support/v4/view/ViewPager

[26] "*ArCore* tracking state," Último acceso: 14 de enero de 2020. [En línea]. Disponible en: https://developers.google.com/ar/reference/java/arcore/reference/com/google/ar/core/TrackingState

[27] "Use *ArCore* with no previsualization," Último acceso: 3 de febrero de 2020. [En línea]. Disponible en: https://github.com/google-ar/arcore-android-sdk/issues/259

[28] "Min supported *Android* version for *Arcore*," Último acceso: 2 de febrero de 2020. [En línea]. Disponible en: https://developers.google.com/ar/discover#supported_devices

[29] "*Android runnable*," Último acceso: 12 de febrero de 2020. [En línea]. Disponible en: https://developer.android.com/training/multiple-threads/define-runnable

[30] "Scale gesture detector," Último acceso: 8 de febrero de 2020. [En línea]. Disponible en: https://developer.android.com/reference/android/view/ScaleGestureDetector.SimpleOnScaleGestureListener

[31] "Librería de ilustracions open-source," Último acceso: 9 de febrero de 2020. [En línea]. Disponible en: https://undraw.co

[32] "Diálogos en *Android*," Último acceso: 9 de febrero de 2020. [En línea]. Disponible en: https://developer.android.com/guide/topics/ui/dialogs

[33] "*Android wifi Scan throttling*," Último acceso: 12 de febrero de 2020. [En línea]. Disponible en: https://developer.android.com/guide/topics/connectivity/wifi-scan#wifi-scan-throttling