UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA MECÁNICA

"DESARROLLO DE UNA APLICACIÓN ANDROID CAPAZ DE GENERAR PLANOS A PARTIR DEL GPS Y EL API DE GOOGLE MAPS"

TRABAJO FIN DE GRADO

Marzo – 2018

AUTOR: Álvaro Bru Serrano

DIRECTOR: César Fernández Peris

ÍNDICE

AGRADECIMIENTOS

A mis padres, por confiar en mí desde siempre y animarme en los momentos difíciles, tanto durante este proyecto como en toda mi etapa estudiantil.

A mi hermano, por su apoyo inestimable y por ser un referente y ejemplo a seguir, en todos los aspectos de la vida.

A mi abuela, por su dedicación y amor desinteresados, y enseñarme lo que significa el valor y la humildad.

A toda mi familia, por el gran cariño y confianza que siempre me han dado y por creer en mí.

A mi compañera de viaje y mejor amiga, por demostrarme que los límites los fijamos nosotros, por creer en mí de la mejor de las formas y por estar siempre ahí.

A mi grupo de amigos, por no haber cambiado durante todos estos años y seguir disfrutando con las mismas ganas que cuando éramos unos críos.

A mi amigo Adrián, por enseñarme lo que significa la amistad desde que tengo memoria y por su gran ayuda desinteresada, tanto para este trabajo como para todo.

A César, director de este trabajo, cuya ayuda ha sido vital para su desarrollo y por darme la oportunidad de aprender sobre el diseño de aplicaciones móviles.

Al profesorado de la Universidad Miguel Hernández de Elche y de la universidad AGH University of Sciences and Technology de Cracovia, por enseñarme sobre esta ciencia tan apasionante como es la ingeniería.

A todas esas personas que, de una forma u otra, me han ayudado durante esta aventura.

A todos vosotros, os estaré siempre agradecido.

1 INTRODUCCIÓN

El presente proyecto *Desarrollo de una aplicación Android capaz de generar planos a partir del GPS y el API de Google MAPS* consiste en la explicación y detallado del proceso de programación, diseño, puesta a punto y depuración[1] de una aplicación destinada al sistema operativo *Android*. Cabe destacar que se trata de un proyecto con finalidad formativa y no lucrativa.

Este trabajo surge a raíz del curso impartido por César Fernández Peris y María Asunción Vicente Ripoll, ambos profesores de la Universidad Miguel Hernández de Elche en el área de Ingeniería de Sistemas y Automática, sobre creación de aplicaciones *Android* en los entornos de desarrollo[2] *MIT App Inventor* y *Android Studio*, siendo este último el utilizado para el desarrollo de la aplicación.

1.1 MOTIVACIÓN Y OBJETIVOS

Vivimos en una sociedad donde el teléfono móvil y las nuevas tecnologías evolucionan constante e imparablemente, mejorando nuestra calidad de vida. Con la aparición de los *Smartphones* o teléfonos inteligentes, nuestra forma de vida y nuestros intereses han dado un giro radical. No solo en la forma de comunicación, sino también en el ocio y el tiempo libre.

Es por ello que, cuando César me propuso el curso anteriormente mencionado y orientado a la realización de Trabajos Fin de Grado de titulaciones de la EPSE (Escuela Politécnica Superior de Elche), no dudé en aceptar la propuesta y poder aprender así a desarrollar aplicaciones.

Una vez finalizado el curso, cuya duración fue de un mes, y con la ayuda de César, nació la idea de desarrollar una aplicación capaz de trabajar con mapas y con intención de medir distancias y superficies. Inicialmente, el objetivo era conseguir que el usuario pudiera recorrer superficies, ya sean edificios, fincas, solares, etc., tomando puntos con la aplicación mediante el GPS. La aplicación almacenaría esos puntos para finalmente calcular la distancia entre ellos y el área de la superficie que encierran.

[1] La depuración de programas es el proceso de identificar y corregir errores de programación. En inglés se conoce como *debugging*, porque se asemeja a la eliminación de *bichos* (*bugs*), manera en que se conoce informalmente a los errores de programación.

[2] "Un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés *Integrated Development Environment* (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software." (Ramos Salavert & Lozano Pérez, 2000)[1]

A medida que me adentro en el mundo del desarrollo de aplicaciones, van apareciendo nuevas posibilidades para la aplicación final, quedando fijados unos objetivos a alcanzar:

- Capacidad para tomar puntos geográficos (latitud y longitud) mediante los sistemas de GPS (Sistema de Posicionamiento Global) y la red móvil.
- Almacenar una cantidad indefinida de puntos, tantos cuantos el usuario necesite.
- Mostrar esos puntos en un mapa.
- Calcular las distancias entre los puntos, al igual que el área y perímetro de la superficie que estos encierran.
- Dibujar la superficie definida por los puntos en el mapa, en forma de polígono irregular[3].
- Posibilidad de tomar puntos directamente sobre el mapa, sin necesidad de estar situado en los mismos.
- Posibilidad de eliminar o editar puntos una vez almacenados.
- Posibilidad de almacenar superficies en la memoria del teléfono.

La intención final de la aplicación es conseguir una estimación de las dimensiones de una superficie lo más aproximada posible, es decir, con el menor error respecto a lo que sería un estudio topográfico completo (véase apartado 3.4).

Por lo tanto, el *target*[4] o destinatario de la aplicación es esa persona, ya sea profesional o no, que necesite un conocimiento aproximado de las dimensiones de una superficie. Por ejemplo, un constructor que desee conocer una estimación en m^2 de un terreno donde desea edificar.

1.2 DESCRIPCIÓN DE LA APLICACIÓN

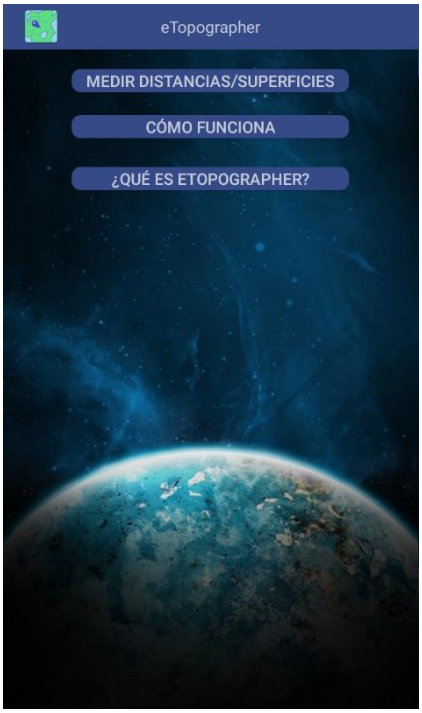
eTopographer (nombre provisional) es una aplicación que permite medir áreas y/o distancias de superficies mediante el uso del GPS, la red móvil y *Google Maps*. El objetivo es obtener la mayor precisión y el menor error en el cálculo de áreas, distancias y perímetros. Pese a que su precisión no es tan exacta como la que obtendría un estudio topográfico, es útil para un primer conocimiento de las medidas de superficies como fincas, edificios, solares, etc.

[3] "En geometría, se le llama polígono irregular a un polígono cuyos lados y ángulos interiores no son iguales entre sí. Los polígonos irregulares no tienen todos sus lados iguales. Sus vértices podrían no estar inscritos en una circunferencia." (Centro para la Innovación y Desarrollo de la Eduacion (CIDEAD))[2]

[4] "La noción se emplea con frecuencia en el marketing y la publicidad. En este contexto, el target es el destinatario al que pretende llegar un servicio o un producto y sus correspondientes campañas de difusión." (Porto & Gardey, 2016)[3]

La aplicación contiene dos a*ctividades*[5]*,* una que contiene el menú inicial y otra con el mapa y la opción de tomar puntos, guardar, etc. A continuación, se muestra la interfaz de las dos actividades:

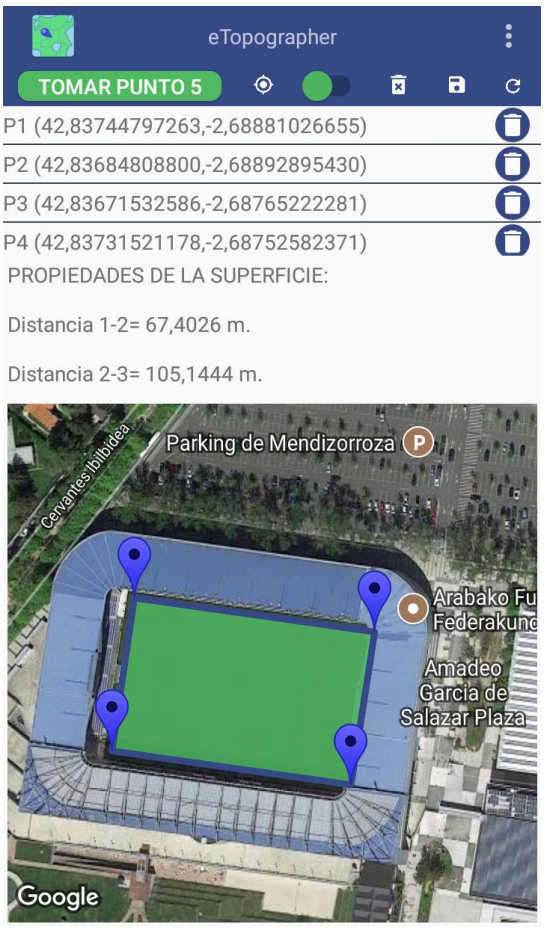
Actividad menú



Actividad principal

El objetivo del usuario es definir la superficie que desea medir mediante puntos. Estos puntos, que se pueden tomar tanto mediante los sistemas GPS y red móvil (para tomar el punto de la localización actual del usuario), como directamente sobre el mapa (mediante clic largo en el punto que queramos del mapa), son las esquinas de la superficie a medir. En el siguiente ejemplo se muestra la medición del campo de fútbol de Mendizorroza, en Vitoria. Los cuatro puntos, tomados mediante clic largo en el mapa, son las esquinas que forman el polígono que describe la superficie a medir:

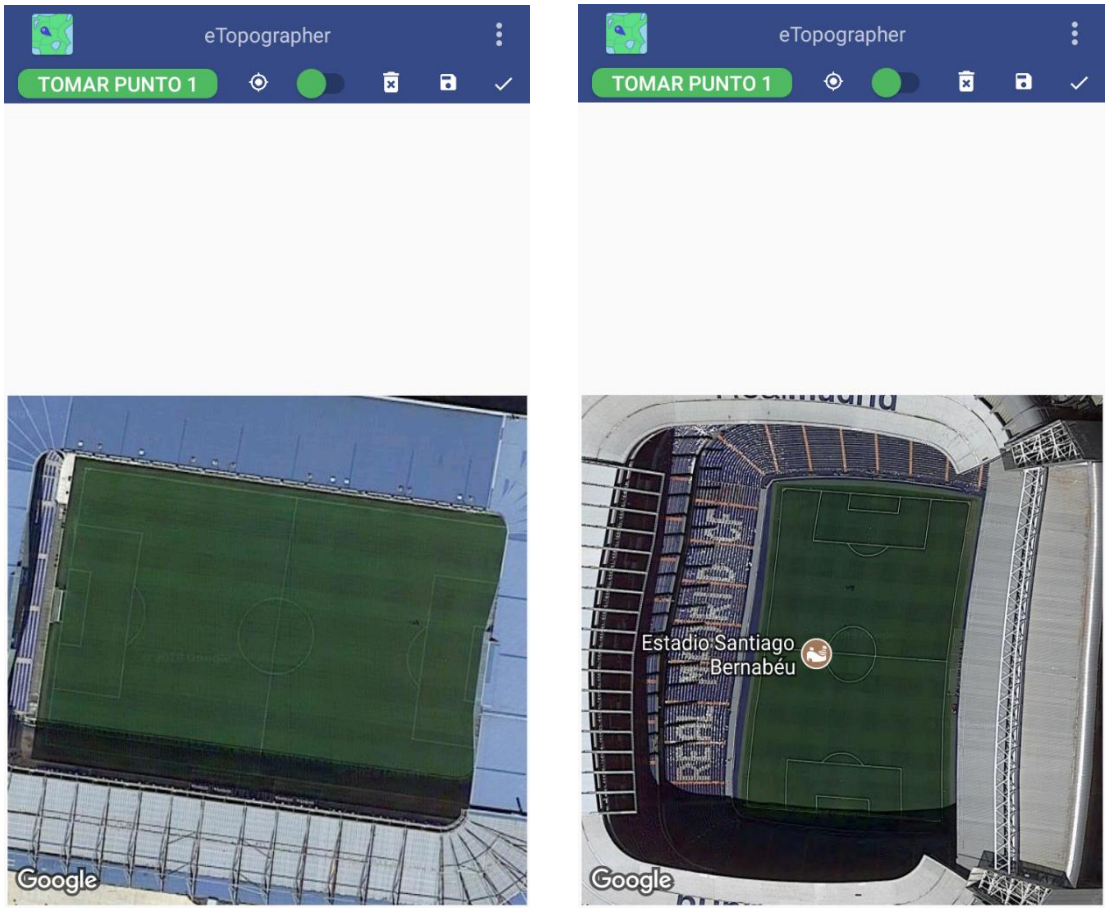
[5] "Las actividades (del inglés a*ctivities*) son uno de los bloques fundamentales de las aplicaciones de la plataforma *Android*. Sirven como punto de entrada para el usuario que navega por la aplicación o entre aplicaciones. La forma en la que las *actividades* de una aplicación están relacionadas es crucial para su desarrollo." (Android Developer, s.f.)[4]



Las dos figuras muestran el resultado final de la medición. La única diferencia está en el apartado de *Propiedades de la superficie*, el cual contiene todas las medidas calculadas. Aquí el usuario debe deslizar hacia abajo para ver toda la información, como muestra la segunda imagen.

Como se puede apreciar en las figuras, los puntos están numerados tanto en el mapa como en la lista, en la cual también aparecen las coordenadas de latitud y longitud de estos. El usuario puede eliminar los puntos que desee haciendo clic sobre ellos en la lista. En el apartado de manual de usuario, se explica con más detalle cómo usar todas las opciones de la aplicación.

En cuanto a la precisión, el error respecto a las medidas reales varía entre el 1 y el 10%. Esto se debe principalmente a la calidad de la foto satelital. Dependiendo de la zona, la imagen se muestra más o menos apta para la medición. Siguiendo con el ejemplo anterior de campos de fútbol, la imagen siguiente muestra la imagen del estadio sin medir, comparada con la de otro estadio distinto:



En la primera figura, las líneas del campo se asemejan más a la realidad porque la foto tiene menos inclinación. Sin embargo, en la segunda la imagen está inclinada y los puntos no se pueden tomar con la misma exactitud.

En el ejemplo de Mendizorroza, la aplicación obtiene unas medidas del terreno de juego de 105,14x67,4 m. Oficialmente son de 105x67 m, obteniendo una precisión muy elevada. En el segundo ejemplo el error es mayor debido a las causas anteriormente mencionadas.

Por último, la aplicación está en inglés y español. Dependiendo del idioma del dispositivo móvil, el texto aparecerá en una lengua u otra. Por defecto, el idioma utilizado es el inglés.

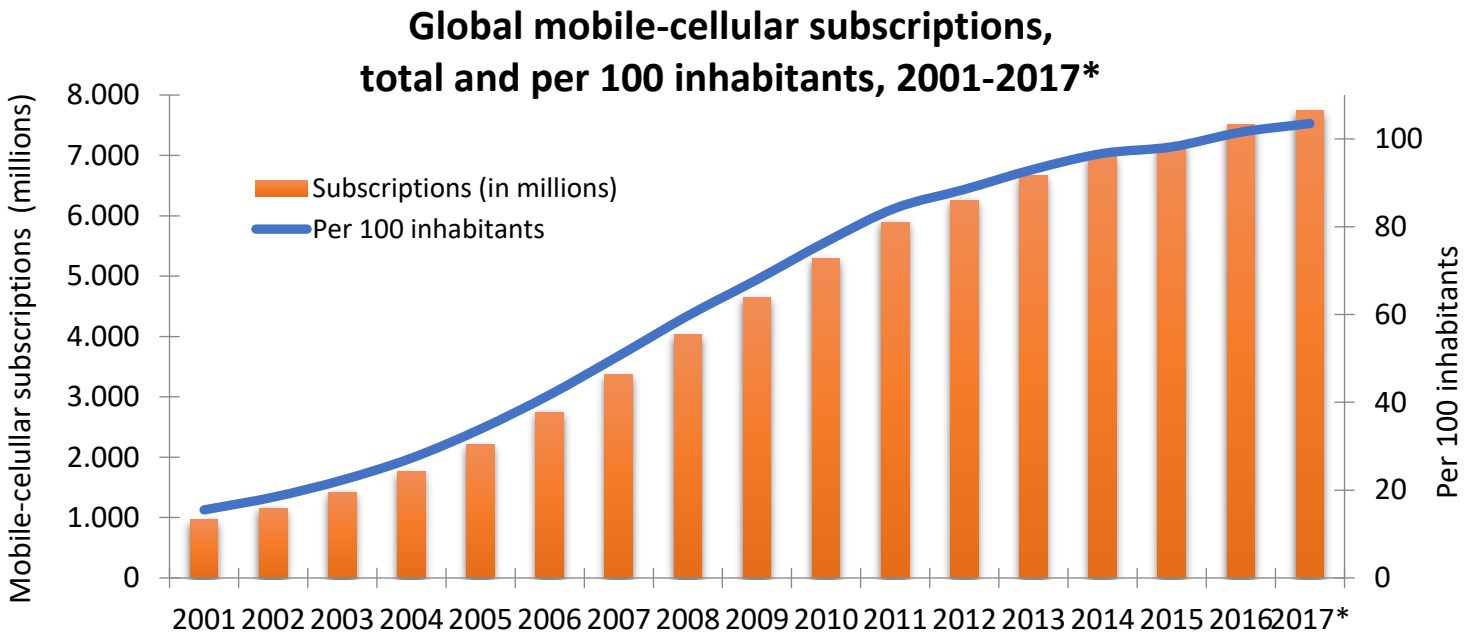
2 ANTECEDENTES

2.1 IMPACTO DEL TELÉFONO MÓVIL EN LA SOCIEDAD

El impacto que ha tenido el teléfono móvil en la sociedad es enorme. Y aún sigue siéndolo. Desde que Martin Cooper, directivo de *Motorola*, realizara la primera llamada mediante un teléfono móvil en abril de 1973, la tecnología móvil ha revolucionado nuestra forma de comunicarnos.

Cada vez es mayor el número de personas en el mundo que usa teléfono móvil. En 1990, había 11 millones de teléfonos móviles en todo el mundo, mientras que en 1999 este número había ascendido a algo más de 400 millones, cuando el número de ordenadores o PCs (Personal Computer) era aproximadamente de 180 millones.

En el siguiente gráfico se muestra el aumento de suscripciones a teléfonos celulares desde 2001 a 2017 por cada 100 habitantes, a nivel mundial [5]:



Note: * Estimate
Source: ITU World Telecommunication /ICT Indicators database

Desde principios del siglo XXI, los teléfonos móviles han adquirido otras funcionalidades a parte de la telecomunicación. A continuación, se muestra una tabla con las diferentes utilidades de las aplicaciones para tabletas o *Smartphones* en la actualidad:

Aplicaciones/Servicios	Descripción
Generales	Calculadoras, despertadores, bloc de notas, agendas.
Geolocalización	Identificación y ubicación de dispositivos (i.e. barcos, astros). Se utilizan mecanismos de GPS (Global Positioning System).
Deporte	Permiten registrar en tiempo real valores como frecuencia cardiaca asociada a la actividad deportiva que se está desarrollando (andar, correr, natación). Aconseja sobre el entrenamiento que se debe seguir; rutas; ejercicios.
Medicina	Mapas mundiales con información de Epidemias Internacionales en tiempo real. Aplicaciones para el control de alcohol en sangre, tele-rehabilitación, estimulación para personas con minusvalías y control de medicamentos.
Ocio	Buscadores de locales de ocio y eventos; reproductores musicales, videos, películas; acceso a televisiones y estaciones de radio; juegos.
Negocios	Presentaciones y videoconferencia; acceso remoto a aplicaciones, estadísticas online y offline, mapas de resultados geográficos, análisis de mercado; acceso a inventarios; presentación de productos a clientes; lanzamiento de campañas de marketing. Pagos a través de la factura de móvil (*direct-to-bill*) para evitar el uso de las tarjetas de crédito en las transacciones por Internet.
Sociedad	Noticias de carácter general, revistas o periódicos convencionales, acceso a redes sociales (*Twitter*, *Facebook*), mensajería (*WhatsApp, Spotbros*).
Nube	Acceso a ficheros depositados en la nube.
Educación	Cursos (idiomas, cocina, instrumentos); traductores; libros (novelas, educación infantil, media, universitaria); universidades virtuales.

2.2 SISTEMA OPERATIVO *ANDROID*

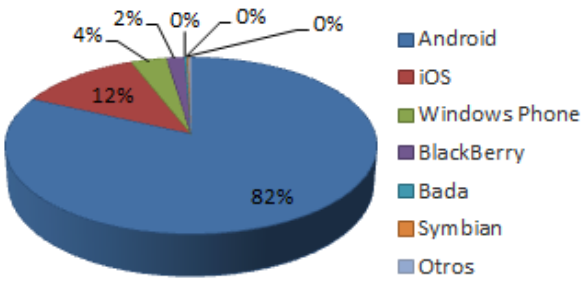
El hecho de que sea el sistema operativo *Android* el elegido para el desarrollo de este trabajo no es casualidad.

Actualmente existen cuatro sistemas operativos importantes a nivel mundial: *Android*, *iOS*, *Windows Phone* y *BlackBerry*. Todos ellos tienen un gran número de clientes en todo el mundo, sin embargo, el más popular de ellos es

Android. En la siguiente imagen se muestra una comparativa del número de *Smartphones* vendidos clasificados por sistema operativo (Gartner, 2013) [6]:

Smartphones vendidos a usuarios finales en el mundo por Sistema Operativo (3er trim. de 2013)

Sistema Operativo	2013/Q3 (Unid)
Android	205,022,000.7
iOS	30,330,000.0
Windows Phone	8,912,000.3
BlackBerry	4,400,000.7
Bada	663,000.3
Symbian	457,000.5
Otros	475,000.2
Total	**250,259,002.7**



Fuente: Gartner (Nov 2013)

Como se puede ver, *Android* es uno de los líderes del mercado de teléfonos móviles. Esto se debe principalmente a su sencillez y a las facilidades para el desarrollo de aplicaciones que este SO ofrece. Además, se trata de un SO libre, lo que hace que los costes sean muy bajos. El hecho de que el desarrollo de aplicaciones sea tan cómodo y barato, hace decantar a muchos desarrolladores a elegir este SO para desarrollar sus aplicaciones.

2.3 HISTORIA DE LA TOPOGRAFÍA

Según la RAE (Real Academia Española), la topografía se define como "arte de escribir y delinear detalladamente la superficie de un terreno" o "conjunto de particularidades que presenta un terreno en su configuración superficial." [7]

También se conoce a la topografía como la disciplina o técnica que se encarga de describir de manera detallada la superficie de un determinado terreno. Esta rama hace foco en el estudio de todos los principios y procesos que brindan la posibilidad de trasladar a un gráfico las particularidades de una superficie terrestre, ya sean naturales o artificiales.

En cuanto a su historia, actualmente se desconoce el origen exacto de la topografía. Se cree que los primeros trabajos topográficos tuvieron lugar en Egipto, debido a la representación de mapas en muros y tablillas. En 1400 a.C, Herodoto pide a Seostris que divida las tierras de Egipto en predios[6] para cobrar impuestos, apareciendo entonces trabajos dedicados a la medición de terrenos.

Cada vez que el Nilo sufría una crecida, ello traía grandes consecuencias. Dado que la crecida variaba mucho de año en año, la superficie que quedaba

[6] Pertenencia inmueble de una cierta extensión superficial.

inundada también variaba considerablemente. Mientras que una crecida insuficiente significaba que quedaban tierras sin irrigar, una inundación excesiva suponía la destrucción de poblados y cosechas. El nivel de inundación entonces determinaba la recaudación fiscal de cada año, haciendo muy importante la labor de estos funcionarios dedicados a la medición de superficies. [8]

2.3.1 AGRIMENSURA

La agrimensura era, antiguamente, una rama de la topografía destinada a la delimitación de superficies. Sin embargo, su propósito es más amplio. También entra dentro de sus competencias la de delimitar derechos.

Actualmente, la comunidad científica internacional la reconoce como una disciplina autónoma, centrada en los estudios territoriales y la fijación de toda clase de límites. Asimismo, produce documentos cartográficos e infraestructura para establecer planos, cartas y mapas. Para cumplir sus objetivos, la agrimensura trabaja junto a la geometría, la ingeniería, la trigonometría, las matemáticas, la física, el derecho, la arquitectura, la geomorfología o la computación, entre otras.

En cuanto a las técnicas de medición, históricamente se utilizaban varias: unión de puntos con cadenas de una longitud conocida para medir distancias; la brújula para medir ángulos horizontales, mejorada posteriormente mediante la adición de discos inscritos con mejor resolución angular, así como el montar telescopios con retículos para ver con más precisión encima del disco; y el altímetro para medir alturas, que consistía básicamente en un barómetro que utilizaba la presión de aire como indicador de alturas.

A partir de finales del siglo XX, se empiezan a utilizar la cinta métrica para distancias cortas, y el teodolito[7] fijado en un trípode para medir los ángulos. [9]

2.4 LA TOPOGRAFÍA HOY EN DÍA

La topografía, como ciencia, ha ido mejorando en función de la evolución de la tecnología de cada época. En la actualidad, el método más utilizado para la toma de datos se basa en el empleo de una estación total, descrita en el siguiente apartado.

[7] Instrumento de medición mecánico-óptico utilizado para medición de ángulos verticales y horizontales con una precisión elevada. Es portátil y manual y está destinado a fines topográficos.

2.4.1 ESTACIÓN TOTAL

La estación total es un aparato electro-óptico consistente en la incorporación de un distanciómetro y un microprocesador a un teodolito electrónico.

Imagen: Estacion total Leica Flexline TS06 Plus [10]

Alguna de las características que incorpora la estación total es una pantalla de cristal líquido (LCD), leds de avisos, iluminación independiente de la luz solar, calculadora, seguidor de trayectoria y distanciómetro, todo en formato electrónico para ser utilizado en ordenadores personales.

2.5 TECNOLOGÍA GPS

GPS (en inglés *Global Positioning System*) es un sistema global de navegación por satélites (GNSS). Para determinar las posiciones en el globo, el sistema se sirve actualmente de 24 satélites y utiliza la trilateración[8].

[8] Método matemático utilizado para determinar las posiciones relativas de objetos usando la geometría de triángulos de forma análoga a la triangulación. A diferencia de ésta, la trilateración utiliza las localizaciones conocidas de dos o más puntos de referencia y las distancias entre los puntos de referencia y el sujeto.

En cuanto a su historia, en la década de 1960 apareció *OMEGA*, el primer sistema mundial de radio de navegación. La armada estadounidense usaba esta tecnología para los sistemas de navegación de flotas. Más tarde se desarrolló TRANSIT, el cual estaba constituido por una constelación de seis satélites en órbita a una altura de 1074 km. Sin embargo, la posibilidad de posicionamiento era intermitente y dependía de las condiciones atmosféricas.

En 1967, la U.S.Navy desarrolló *Timation*, que incorporaba relojes atómicos. Pero no fue hasta 1973 cuando surgió el programa de tecnología de navegación (en inglés *Navigation Technology Program*) conocido también como NAVSTAR GPS.

Tras años de desarrollo, a partir de 1985 el sistema había lanzado once satélites, y ya para 1995 se consiguió una capacidad operacional total y de utilidad civil. [11]

2.6 SISTEMA DE COORDENADAS

Las coordenadas geográficas son un sistema de coordenadas que permite que cada ubicación en la Tierra sea especificada por un conjunto de números, letras o símbolos. Las coordenadas de posición horizontal utilizadas son la latitud y la longitud, que forman un sistema de coordenadas angulares esféricas cuyo centro es el centro de la Tierra y suelen expresarse en grados sexagesimales.

En cuanto a la latitud, es la medida del ángulo entre la línea de un punto sobre la superficie terrestre al centro de la Tierra y el plano del ecuador terrestre:

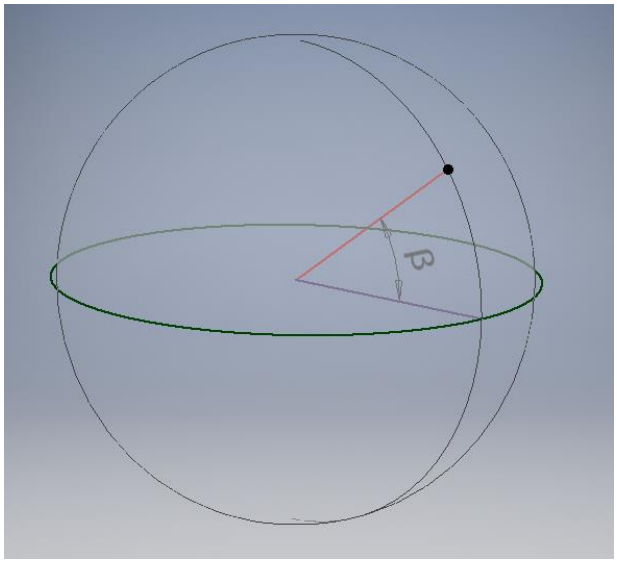


Imagen tomada desde *Inventor Proffesional*

Por otro lado, la longitud es la medida del ángulo que indica la posición este-oeste de un punto sobre la superficie terrestre a partir del meridiano de

Greenwich. Una línea de longitud, también llamada meridiano, es la mitad de un círculo sobre la superficie de la Tierra que pasa a través del polo norte y del polo sur: [12]

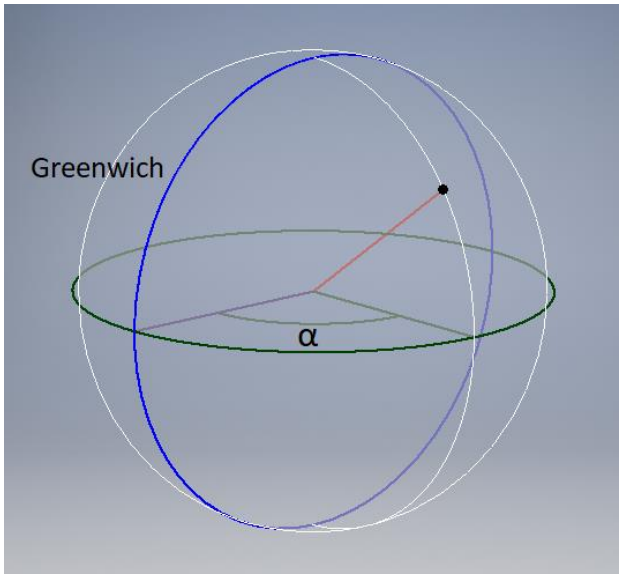


Imagen tomada desde *Inventor Proffesional*

Respecto al sistema de coordenadas utilizado en la aplicación es el WGS84 (del inglés *World Geodetic System 84*, que significa Sistema Geodésico Mundial 1984). Este sistema de coordenadas geográficas mundial permite localizar cualquier punto en la Tierra sin necesitar otro de referencia, con un error de cálculo menor a 2 cm. Es en el que se basa el Sistema de Posicionamiento global (GPS) y está expresado en grados, minutos y segundos.

Para realizar los cálculos de distancias y superficies, es necesario transformar estas coordenadas geodésicas en cartesianas. Para ello se utiliza el sistema UTM.

El sistema de coordenadas universal transversal de Mercator (en inglés *Universal Transverse Mercator*, UTM). Este sistema esta basado en la proyección cartográfica transversa de Gerardus Mercator, un importante geógrafo, matemático y cartógrafo flamenco del siglo XVI.

Este sistema divide la tierra en 60 husos de 6º de longitud. Cada huso se numera con un número entre el 1 y el 60 (1er huso: longitudes 180º-174º, meridiano 177ºW). Cada huso tiene asignado un meridiano central, donde se situa el origen de coordenadas. Los husos se numeran en orden ascendente hacie el este. En cuanto a la latitud, el sistema divide la Tierra en 20 bandas de 8º de latitud, denominadas con letras desde la C hasta la X, excluyendo algunas como la I y la O por evitar confusiones. Las letras menores de N se sitúan en el hemisferio sur, mientras que la N y superiores para el hemisferio norte. [13]

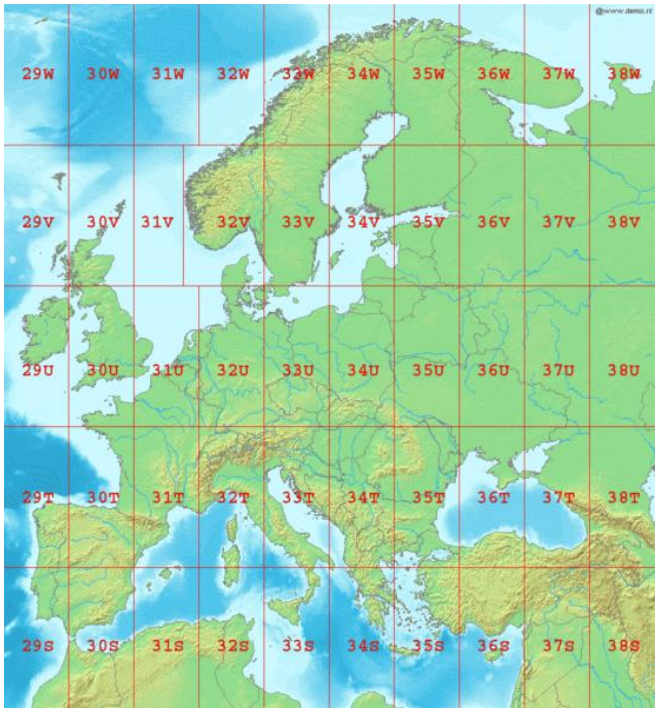


Imagen: (Aronsson, 2006) [14]

La figura anterior muestra un mapa que con las zonas de latitud y longitud del sistema de coordenadas universal transversal de Mercator de Europa. Se puede observar que van desde la zona 29S a la 38W.

3 OBJETIVOS

3.1 OBJETIVOS

A continuación, se presentan los objetivos del proyecto:

Objetivos generales:

- Aprender a programar aplicaciones móviles.
- Realizar un proyecto completo del diseño y desarrollo de una aplicación *Android*.
- Desarrollar una aplicación completa que cumpla unos objetivos fijados.
- Adquirir conocimientos de programación útiles que no se adquieren durante el grado.

Objetivos específicos:

- Investigar sobre el campo de la topografía y conseguir una mejora o innovación en el campo.
- Conseguir una aplicación que trabaje con un mapa integrado.
- Poder dar al usuario la opción de interactuar con el mapa.
- Capacidad para medir distancias, áreas y perímetros de todo tipo de superficies.
- Capacidad de almacenar en la memoria del teléfono.

3.2 QUÉ NO ES OBJETIVO DE ESTE TRABAJO

Pese a la posibilidad de subir la aplicación a la *Store* de *Android*, no ha sido nunca el objetivo de este proyecto. Como ya se ha mencionado con anterioridad, el objetivo principal es obtener unos conocimientos que no se adquieren durante el grado y que sin duda son útiles para cualquier técnico industrial.

Así pues, no se espera obtener ningún beneficio además del meramente formativo. En el apartado de aplicaciones similares, se muestran algunas aplicaciones que se asemejan a la desarrollada en este proyecto, las cuales nunca han sido referencia del mismo ni han servido de guía. Sin embargo, su propósito es parecido y es importante mencionarlas.

4 MATERIALES Y MÉTODOS

4.1 ANDROID STUDIO

Android Studio es el entorno de desarrollo integrado oficial para la plataforma *Android*. Anteriormente, el desarrollo de aplicaciones *Android* se llevaba a cabo mediante el software *Eclipse*, pero a partir de mayo de 2013 la compañía *Google I/O* reemplazó este por *Android Studio*. Sin embargo, no es hasta diciembre de 2014 cuando se lanza la primera versión estable.

Está disponible para las plataformas *Microsoft Windows*, *macOS* y *GNU/Linux*. La última versión es la 3.0, lanzada en octubre de 2017.

Las características de la versión 3.0 son las siguientes:

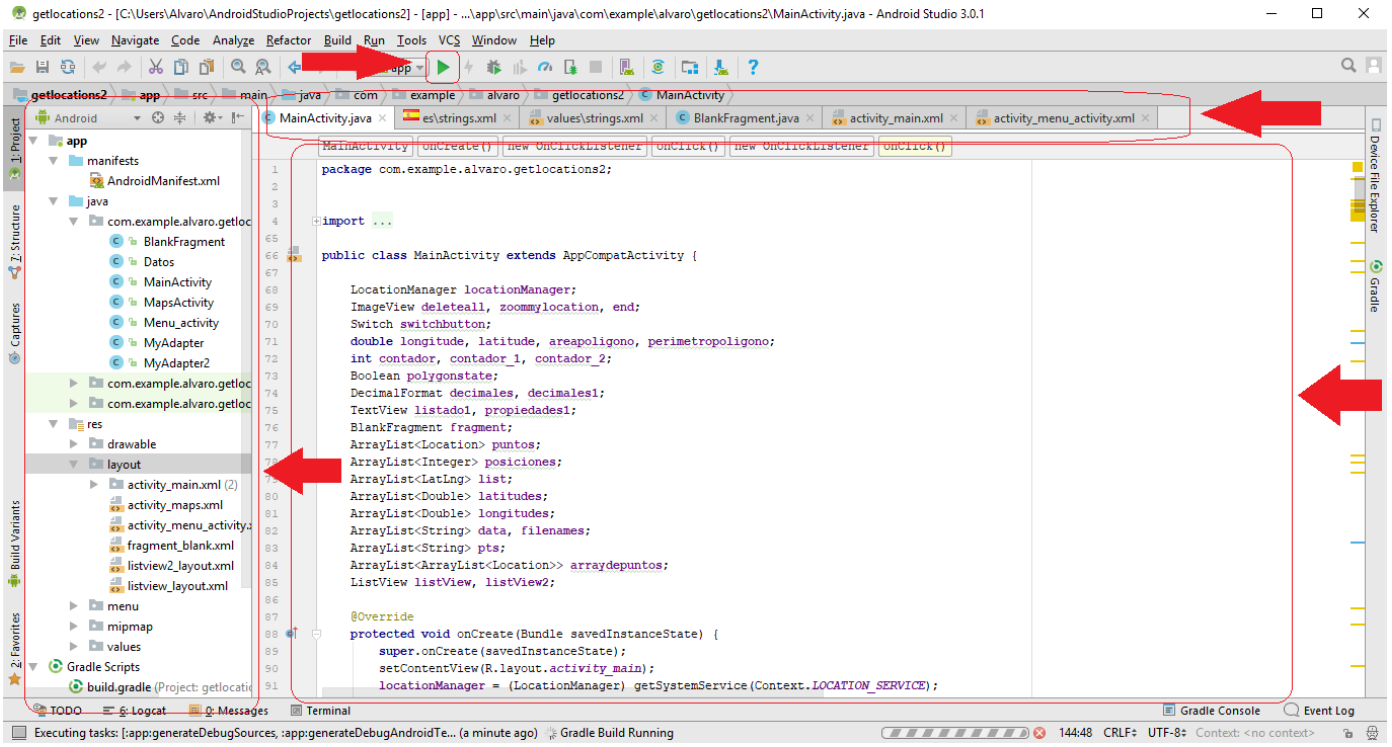
- Integración de *ProGuard* y funciones de firma de aplicaciones.
- Renderizado en tiempo real.
- Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.
- Soporte para construcción basada en *Gradle*.
- Refactorización específica de *Android* y arreglos rápidos.
- Un editor de diseño enriquecido que permite a los usuarios arrastrar y soltar componentes de la interfaz de usuario.
- Herramientas *Lint* para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones y otros problemas.
- Plantillas para crear diseños comunes de *Android* y otros componentes.
- Soporte para programar aplicaciones para Android Wear.
- Soporte integrado para *Google Cloud Platform*, que permite la integración con *Google Cloud Messaging* y *App Engine*.
- Un dispositivo virtual de *Android* que se utiliza para ejecutar y probar aplicaciones.

En cuanto a los requisitos del sistema para su instalación y uso:

	Windows	macOS	Linux
Versión SO	10/8/7 (32 o 64-bit)	Mac OS X 10.10 (Yosemite) o superior, hasta 10.13 (macOS High Sierra)	GNOME o KDE desktop
RAM	3 GB RAM mínimo, 8 GB RAM recomendado y 1 GB más para el emulador de Android		
Espacio en disco	2 GB de espacio en disco para Android Studio, 4 GB recomendados (500 para la IDE y al menos 1,5 GM para Android SDK, imágenes de sistema de emulador y cachés)		
Versión JAVA	Java Development Kit (JDK) 8		

Resolución de pantalla	1280x800 mínimo, 1440x900 recomendado

La interfaz de trabajo de este software es la siguiente:



A la izquierda se encuentra el panel con los diferentes archivos que estructuran el proyecto (Java, xml, etc.). En la parte central se sitúa el cuadro donde se programan los códigos, y en el panel de arriba los nombres de los archivos con los que se está trabajando.

Este entorno de desarrollo integra emuladores para probar las aplicaciones. Sin embargo, también es posible conectar dispositivos móviles al ordenador de trabajo y, tras instalar los respectivos controladores o *drivers*, probar las aplicaciones en nuestros dispositivos. [15]

4.2 GOOGLE MAPS

La plataforma de mapas de *Google*, *Google Maps*, es la integrada en la aplicación. Se trata de un servidor de mapas que pertenece a *Alphabet Inc*. y ofrece imágenes de mapas desplazables y fotografías por satélite del mundo.

Características

Google Maps ofrece la posibilidad de acercar y alejar el mapa. El usuario, tanto si trabaja desde un ordenador personal como desde un dispositivo móvil,

puede controlar el mapa y moverlo a la ubicación que desee. También permite controlar el *zoom* al que se desea ver la imagen.

Una de las características más interesantes que incluye *Google Maps* es la posibilidad de inclinar la cámara. En algunas zonas, esta opción permite visualizar los edificios en 3D, lo que ayuda a conseguir una medición más precisa. En el apartado de manual de uso se explica cómo usar esta opción en la aplicación.

Coordenadas

En cuanto al sistema de coordenadas utilizado en *Google Maps* es el WGS84, explicado en el apartado anterior de sistema de coordenadas. Se trata de coordenadas latitud y longitud siendo positivas para Norte y Este, y negativas para Sur y Oeste.

4.3 LENGUAJE DE PROGRAMACIÓN

Android Studio trabaja principalmente con dos tipos de archivo: XML y JAVA.

XML

Tiene como misión crear las interfaces (o *layouts*) que el usuario visualiza y con las que interactúa. Es el encargado de la parte estética de las aplicaciones, aunque también es posible modificar elementos desde la clase *java*.

JAVA

En estos archivos es donde se programa las instrucciones para el comportamiento de los elementos declarados en los archivos xml. Es necesario identificar los elementos de los archivos xml y declararlos en el java. Esto es posible mediante la instrucción *findViewById*. En el apartado de archivos xml y Java, se enumeran los archivos xml y java que contiene el proyecto de la aplicación.

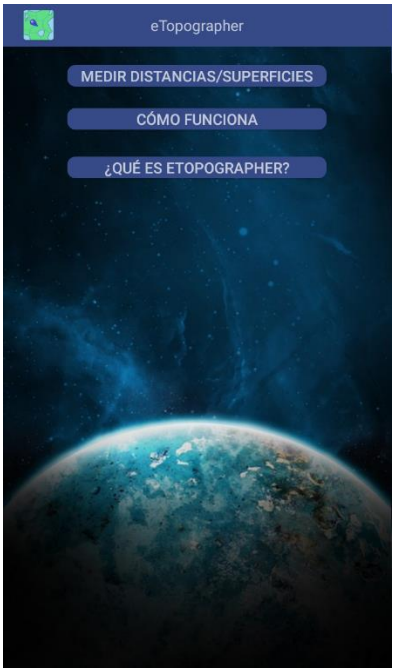
4.4 DISEÑO Y PROGRAMACIÓN

4.4.1 ESTRUCTURA

Como se comenta en el apartado de descripción de la aplicación, el proyecto contiene dos *actividades* principales, cada una de ellas con su respectiva interfaz o *layout*.

***Actividad* menú**

Se trata de la actividad que se lanza al iniciar la aplicación. En ella se da la opción de pasar a la *actividad* principal de medición de superficies, además de otras opciones para que el usuario que inicie por primera vez pueda conocer el propósito de la aplicación y un breve manual de uso.



***Actividad* principal**

Es la actividad que contiene el mapa y todas las instrucciones necesarias para la toma de puntos e interacción con el mapa. El mapa de *Google Maps* se implementa en la interfaz de la actividad mediante la clase *SupportMapFragment*. Además, contiene siete botones (explicados en el apartado manual de uso), así como un *ListView* para mostrar una lista con los puntos tomados y un *ScrollView* para mostrar las medidas de la superficie.

SupportMapFragment

En *Android Studio*, un *fragment* representa un comportamiento o una parte de la interfaz de usuario en una actividad. Puedes combinar múltiples fragmentos en una sola actividad para crear una IU (Interfaz de Usuario) multipanel y volver a usar un fragmento en múltiples actividades. Puedes pensar en un fragmento como una sección modular de una actividad que tiene su ciclo de vida[9] propio, recibe sus propios eventos de entrada y que puedes agregar o quitar mientras la actividad se esté ejecutando (algo así como una "subactividad" que puedes volver a usar en diferentes actividades).

Un fragmento siempre debe estar integrado a una actividad y el ciclo de vida del fragmento se ve directamente afectado por el ciclo de vida de la actividad anfitriona. Por ejemplo, cuando la actividad está pausada, también lo están todos sus fragmentos, y cuando la actividad se destruye, lo mismo ocurre con todos los fragmentos. Sin embargo, mientras una actividad se está ejecutando (está en el estado del ciclo de vida *reanudada*), puedes manipular cada fragmento de forma independiente; por ejemplo, para agregarlos o quitarlos. Cuando realizas una transacción de fragmentos como esta, también puedes agregarlos a una pila

[9] El ciclo de vida de una aplicación *Android* es diferente al de aplicaciones en otros SO. La diferencia es que en *Android* el ciclo es controlado principalmente por el sistema. Una característica importante es que la destrucción de un proceso es controlada por el sistema y no la aplicación, basándose en el conocimiento que tiene de las partes de la aplicación que están corriendo. Si tras eliminar un proceso el usuario vuelve a ella, se crea de nuevo el proceso, perdiendo el estado en el que se encontraba. Los eventos del ciclo de vida en *Android* son: Activa (*Running*): actividad visible y tiene el foco; Visible (*Paused*): la actividad es visible pero no tiene el foco; Parada (*Stopped*): la actividad no es visible, el programador debe guardar el estado de interfaz de usuario, etc.; Destruida (*Destroyed*): la actividad termina al invocarse el método finish(), o es destruida por el sistema. [16]

de actividades administrada por la actividad; cada entrada de la pila de actividades en la actividad es un registro de la transacción de fragmentos realizada. La pila de actividades le permite al usuario invertir una transacción de fragmentos (navegar hacia atrás) al presionar el botón *Atrás*.

Cuando agregas un fragmento como parte del diseño de tu actividad, este se ubica en *ViewGroup*, dentro de la jerarquía de vistas de la actividad y el fragmento define su propio diseño de vista. Puedes insertar un fragmento en el diseño de tu actividad declarando el fragmento en el archivo de diseño de la actividad (XML) como elemento *<fragment>* o agregándolo a un *Viewgroup* existente desde el código de tu aplicación. Sin embargo, no es necesario que un fragmento forme parte del diseño de la actividad; también puedes usar un fragmento con su IU propia, como un trabajador invisible para la actividad.

En nuestro caso, utilizamos *SupportMapFragment*, una clase pública extendida de *fragment* que permite de manera sencilla implementar un mapa en la actividad. Además, esta clase es compatible con versiones de *Android* y niveles de API anteriores. Mediante la clase *getMapAsync(OnMapReadyCallback),* automáticamente se inicializa el sistema de mapas y la vista.

Comunicación entre *fragment* y la *actividad* principal

Un *fragment* puede acceder a la instancia *Activity* mediante *getActivity()* y realizar tareas de manera sencilla. De la misma forma, para llamar a métodos[10] del fragmento usamos *findFragmentById()*.

Un ejemplo en nuestra aplicación es el de la opción de tomar un punto haciendo clic prolongado en el mapa. Para ello, en el *fragment* (BlankFragment.java) utilizamos el siguiente código:

```
mMap.setOnMapLongClickListener(new GoogleMap.OnMapLongClickListener() {
    @Override
    public void onMapLongClick(final LatLng latLng) {
        final AlertDialog.Builder dialog = new AlertDialog.Builder(getContext());
        dialog.setTitle(R.string.addpoint)
                .setPositiveButton(getString(R.string.yes), new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface paramDialogInterface, int paramInt) {
                        ((MainActivity)getActivity()).addpointwithlongclick(latLng);
                    }
                })
                .setNegativeButton(getString(R.string.no), new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface paramDialogInterface, int paramInt) {
                    }
                });
        dialog.show();
    }
});
```

[10] En programación, un método es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase como a un objeto. Existen dos tipos: métodos de instancia (relacionados con un objeto en particular), y métodos estáticos o de clase (asociados a una clase en particular).

Donde *mMap* es un objeto de la clase *googleMap*. El método *onMapLongClickListener* es llamado cuando el usuario hace un clic largo en cualquier parte del mapa. Este método proporciona la variable *LatLng*, compuesta por la latitud y longitud del punto del mapa sobre el cual se ha hecho clic largo. Una vez llamado al método, se abre un dialogo mediante *AlertDialog.Builder* en el que se le da la opción al usuario de si quiere tomar el punto o si no. Si el usuario pulsa el botón positivo, implementado mediante *setPositiveButton*, se llama al método *OnClick* del mismo y se procede a enviar a la actividad principal la variable *LatLng,* recogida anteriormente, mediante el código *((MainActivity)getActivity()).addpointwithlongclick(latLng*), donde *addpointwithlongclick* es el método implementado en la actividad principal (*MainActivity*). El método *addpointwithlongclick* de la actividad principal es el siguiente:

```
public void addpointwithlongclick(LatLng latLng) {
    Location loc = new Location("");
    loc.setLatitude(latLng.latitude);
    loc.setLongitude(latLng.longitude);
    puntos.add(loc);
    fragment.zoom(puntos);
    data.add("P" + (contador - 1) + " " + "(" +
decimales.format(puntos.get(contador - 2).getLatitude())+
","+decimales.format(puntos.get(contador - 2).getLongitude())+ ")");
    Button button = findViewById(R.id.getposition);
    button.setText(getString(R.string.point) + " " + contador);
    contador = contador + 1;
    MyAdapter customAdapter = new MyAdapter(getApplicationContext(), data);
    listView.setAdapter(customAdapter);
}
```

Con este método, se recoge la variable *LatLng* enviada desde el *fragment* y se convierte en variable de clase *Location*. Esta clase esta compuesta por una latitud y una longitud, las cuales se obtienen de la variable *LatLng*, y se añade al *array*[11] denominado "puntos*"* de la clase <*Location*>. Este *array* contiene todos los puntos que el usuario va tomando, ya sea a través de clic largo sobre el mapa o a través de los botones.

Además, este método envía ordenes al mapa mediante la línea *fragment.zoom(puntos).* Esta orden envía el *array "*puntos" al *fragment* y aplica el método *zoom*. Este método se encarga de hacer *zoom* en la zona delimitada por los puntos tomados, de modo que cada vez que el usuario toma un punto, el mapa se aleja o acerca para que todos los puntos tomados aparezcan en el mapa con unos márgenes o *padding* preestablecidos.

Este ejemplo es uno de los varios que contiene el código de la aplicación en los que se comunica el *fragment* con la *actividad* principal. No resulta interesante mostrar todo el código de los archivos, ya que son muchas líneas y

[11] En programación se denomina matriz, vector o formación (en inglés *array*) a una zona de almacenamiento contiguo que contiene una serie de elementos del mismo tipo, los elementos de la matriz. Desde el punto de vista lógico, una matriz se puede ver como un conjunto de elementos ordenados en fila (o filas y columnas si tuviera dos dimensiones).

resultaría pesado para el lector. Además, en la mayoría se repite la estructura del ejemplo mencionado.

ListView

El *ListView* es una subclase pública de la clase *AdapterView*, que muestra una lista vertical de elementos. La ventaja de utilizar esta clase es que puedes añadir, modificar o eliminar elementos de la lista de forma sencilla. En nuestro caso, cada vez que se toma un punto se añade un elemento a la lista. Además, al hacer clic en un punto de la lista se da la opción al usuario de borrar ese punto.

Los *AdapterView* requieren un *adapter*. Los *adapter* son objetos que sirven como puente entre los *AdapterView* y la información que se desea mostrar. El *adapter* da acceso a los archivos de información y es responsable de mostrar una vista (o *view*) para cada ítem de información. En el proyecto, el archivo MyAdapter.java es el *adapter* que relaciona el *ListView* con el *array* "puntos". En el ejemplo anterior se muestra un ejemplo de cómo se añade un elemento al *ListView* mediante la clase *MyAdapter*.

ScrollView

La clase *ScrollView* ofrece un contenedor para una jerarquía vertical de vistas que se puede desplazar por el usuario. Un ejemplo en la aplicación es la vista con la información sobre las medidas de la superficie:

```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="90dp"
    android:layout_weight="5">
    <TextView
        android:id="@+id/listado1"
        android:layout_margin="0dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="" />
</ScrollView>
```

4.4.2 PERMISOS Y MANIFEST

La aplicación requiere permisos del teléfono para acceder a los servicios de ubicación y conexión a internet. Para ello, es necesario incluir estos permisos en el archivo *AndroidManifest.xml* del proyecto de *Android Studio*. Los permisos se solicitan con el siguiente código del *manifest*:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
```

Al abrir la aplicación por primera vez, se solicitan los permisos de ubicación al usuario, el cual debe aceptarlos para permitir a la aplicación acceder a esos servicios.

4.4.3 ARCHIVOS XML Y JAVA

El proyecto de *Android Studio* contiene archivos de tipo XML y JAVA. A continuación, se mencionan todos los archivos editados y creados para la aplicación y su propósito en la misma. Los archivos se pueden dividir en subgrupos según su función en el proyecto:

XML

Función	Archivo
Diseño de la interfaz (*layout*) de las *actividades*.	activity_main.xml
	activity_menu_activity.xml
Diseño de la interfaz de los elementos de la lista (ListView)	listview_layout.xml
Diseño del menú desplegable	toolbar_menu.xml
Diseño de los iconos de los botones	refresh.xml
	save.xml
	mylocation1.xml
	launcher1.xml
	ic_deleteall.xml
	confirmation.xml
	delete.xml
Diseño del logo de la aplicación	launcher4.xml
Diseño de los estilos	styles.xml
Cadenas de texto	strings.xml (inglés)
	strings.xml (castellano)

JAVA

Función	Archivo
Actividad principal	MainActivity.java
Actividad menú	Menu_activity.java
Clase *adapter* para relacionar el *ListView* con el *array* "puntos"	MyAdapter.java
Clase *fragment* para el mapa y sus utilidades	BlankFragment.java

4.4.4 LIBRERÍAS Y APIs

En informática, una biblioteca o librería (del inglés *library*) es una colección o conjunto de subprogramas usados para desarrollar un software. Se trata de recursos externos los cuales el programador puede añadir a su programa para hacer más fácil su desarrollo. Por ejemplo, si el programador necesita para su programa realizar operaciones matemáticas complejas y difíciles de programar. En este caso, es posible que existan librerías que realicen esos cálculos por lo que el desarrollador puede añadirlas a su programa y utilizarlas.

Una *interfaz de programación de aplicaciones*, abreviada como API del inglés (*Application Programming Interface*), es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos[12]) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Solo que la implementación interna de esas funciones está oculta al público y los programadores sólo pueden ingresar datos de entrada y obtener así un resultado, sin saber lo que sucedió en el medio. [17]

En el proyecto de la aplicación se utilizan una serie de bibliotecas y APIs fundamentales para su correcto funcionamiento. La forma de importar librerias, tanto de Java como externas, a nuestro proyecto es mediante *import*. Por ejemplo:

```
import java.util.ArrayList;
```

Librería de *Android* SDK

Un kit de desarrollo de softare (SDK) es un conjunto de herramientas de desarrollo de software que tiene como objetivo ayudar al desarrollador a crear una aplicación informática para un sistema concreto.

[12] En el mundo de programación orientada a objetos (POO), un *objeto* es el resultado de la instanciación de una *clase*. Los conceptos de *clase* y *objeto* son análogos a los de tipo de datos y variable. Es decir, definida una clase podemos crear objetos de esa clase, al igual que de un determinado tipo de dato (por ejemplo, número entero), podemos definir variables de dicho tipo:: (entero a,b). Entero sería el tipo de dato y a y b las variables de dicho tipo.

El SDK de *Android* incluye, entre otras herramientas, un depurador, un simulador y una biblioteca. Para el desarrollo de este trabajo se utilizan clases de esta librería.

Librería de Java

La librería contiene, entre otras, la clase *ArrayList*, la cual es crucial para el desarrollo del proyecto.

API de *Google Maps*

Para poder utilizar los mapas de *Google*, es necesario instalar en el proyecto la biblioteca *Google Maps Android API*. Para ello, es necesario solicitar unas credenciales desde nuestra cuenta de *Google*, indicando el nombre y tipo de proyecto para el que la queremos utilizar.

Para la implementación de este API, en el archivo *build.gradle* del proyecto se añade, en el apartado de *dependencies*:

```
implementation 'com.google.android.gms:play-services-maps:11.6.2'
```

GSON

GSON en una biblioteca en *Java*, desarrollada por *Google*, que se utiliza para convertir objetos *Java* a *JSON*[13] (serialización), y objetos *JSON* a *Java* (deserialización). En el proyecto de la aplicación GSON se utiliza, junto con la clase Java *SharedPreferences*, para dar la posibilidad al usuario de guardar superficies en la memoria del teléfono.

SharedPreferences

SharedPreferences es un API de Android que se usa para guardar una colección relativamente pequeña de pares clave-valor. Un objeto *SharedPreferences* tiene asociado un archivo que contiene pares clave-valor y proporciona métodos simples para leer y escribir dichos valores.

El siguiente ejemplo muestra cómo son utilizados GSON y *SharedPreferences* en el proyecto:

[13] JSON (acrónimo de *JavaScript Object Notation*) es un formato de texto ligero para intercambio de datos. En nuestro caso, lo utilizamos para guardar en la memoria las superficies que el usuario desee.

```
public void save(View view) {
    if (puntos.size()>0) {
        AlertDialog.Builder dialog = new AlertDialog.Builder(MainActivity.this);
        dialog.setTitle(getString(R.string.save));
        //Añadir un EditText
        final EditText filename = new EditText(this);
        dialog.setView(filename);
        dialog.setPositiveButton(R.string.yes, new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface paramDialogInterface, int paramInt) {
                SharedPreferences sharedPreferences = getSharedPreferences("filenames",
MODE_PRIVATE);
                if (sharedPreferences.contains("name list")) {
                    loadData();
                    loadPoints();
                    filenames.add(filename.getText().toString());
                    arraydepuntos.add(puntos);
                    saveData();
                    savePoints();
                }else{
                    filenames.add(filename.getText().toString());
                    arraydepuntos.add(puntos);
                    saveData();
                    savePoints();
                }
            }
        })
                .setNegativeButton(R.string.no, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface paramDialogInterface, int
paramInt) {
                    }
                });
        dialog.show();
    } else{
        Toast.makeText(MainActivity.this, getString(R.string.atleast2),
Toast.LENGTH_SHORT).show();
    }
}

private void savePoints(){
    //Guardar puntos
    SharedPreferences sharedPreferences = getSharedPreferences("points", MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    Gson gson = new Gson();
    String json = gson.toJson(arraydepuntos);
    editor.putString("points list", json);
    editor.apply();
}
private void loadPoints(){
    //Cargar puntos
    SharedPreferences sharedPreferences = getSharedPreferences("points", MODE_PRIVATE);
    Gson gson = new Gson();
    String json = sharedPreferences.getString("points list", null);
    Type type = new TypeToken<ArrayList<ArrayList<Location>>>(){}.getType();
    arraydepuntos = gson.fromJson(json, type);
}
private void saveData(){
    //Guardar nombre del archivo
    SharedPreferences sharedPreferences = getSharedPreferences("filenames",
MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    Gson gson = new Gson();
    String json = gson.toJson(filenames);
    editor.putString("name list", json);
    editor.apply();
}
private void loadData() {
    SharedPreferences sharedPreferences = getSharedPreferences("filenames",
MODE_PRIVATE);
    Gson gson = new Gson();
    String json = sharedPreferences.getString("name list", null);
    Type type = new TypeToken<ArrayList<String>>() {
    }.getType();
    filenames = gson.fromJson(json, type);
}
```

El método *save(View view)* se ejecuta cuando el usuario pulsa el botón de guardar una superficie de puntos. Primero, mediante una sentencia *if* se comprueba si el *array* "puntos" contiene un número de elementos mayor de 0 (sino no habría nada que guardar).

Una vez comprobado, **si cumple** hace lo siguiente: abre un diálogo con un *EditText* (donde el usuario escribe el nombre del archivo a guardar) y dos botones positivo y negativo. Al pulsar el positivo, primero se comprueba si hay algún archivo *SharedPreferences* guardado mediante una sentencia *if*. En caso de **sí haber** archivo *SharedPreferences*: se carga el archivo con el *array* de nombres "filenames" mediante el método *loadData(),* y el archivo con el *array* de puntos "arraydepuntos" (*array* que contiene los *arrays* de "puntos") mediante el método *loadPoints()*. Una vez cargados, se añade al *array* de nombres el nuevo nombre del archivo a guardar (proporcionado por el usuario a través del *EditText*), y al de puntos la nueva lista (*array*) de puntos a guardar. Una vez modificados esos *arrays*, se guardan de nuevo mediante los métodos *saveData()* y *savePoints().* En caso de **no haber** archivo *SharedPreferences* (no se han guardado archivos anteriormente): se añade la cadena de texto escrita por el usuario al *array* de nombres "filenames", y el *array* de puntos "puntos" al *array* "arraydepuntos". Después, se ejecutan los métodos *saveData()* y *savePoints()* para guardar el *array* de nombres y el *array* "arraydepuntos" previamente modificados.

En caso de **no cumplir**, se muestra un mensaje *Toast* con la *string* "atleast2", que muestra el mensaje al usuario de "*Se necesita al menos 1 punto para guardar*." [18]

4.4.5 CLASES IMPORTANTES

En este apartado se mencionan algunas de las clases, pertenecientes a las librerías del apartado anterior, que son importantes en la aplicación:

SphericalUtil de *Google Maps*

Esta clase contiene, entre otros métodos, los de cálculo de distancias, áreas y perímetros:

Método	computeDistanceBetween (LatLng from, LatLng to)
Función	Devuelve la distancia entre dos puntos.
Ejemplo en el proyecto	**double** distance = SphericalUtil.*computeDistanceBetween*(**new** LatLng(**puntos**.get(i).getLatitude(), **puntos**.get(i).getLongitude()), **new** LatLng(**puntos**.get((i + 1)).getLatitude();
Uso	Se deben introducir los dos puntos geográficos en formato *LatLng.* Cuando se pulsa el botón de medir superficies, se utiliza un bucle for para recorrer el array "puntos", calculando la distancia entre los puntos que contenga. La variable i es el contador que recorre el *array*. Por lo tanto, para calcular la distancia entre punto (i) y el (i+1), se crean variables *LatLng* con la latitud y longitud de éstos.

Método	computeArea (java.util.List<LatLng> path)
Función	Devuelve del área de una superficie.
Ejemplo en el proyecto	**double** areapoligono =SphericalUtil.computeArea(**list**);
Uso	Se debe introducir un objeto *List* de la clase Java, el cual contenga el camino (en inglés *path*) de puntos que defina la superficie a medir. Con un bucle hemos creado el objeto *List* "list", que contiene los elementos *LatLng* del *array* "puntos", el cual introducimos y nos devuelve el área.

Método	computeLength (java.util.List<LatLng> path)
Función	Devuelve las medidas de un camino o *path* de puntos.
Ejemplo en el proyecto	**double** perimetropoligono = SphericalUtil.*computeLength*(**list**) + distance1;
Uso	Se debe introducir un objeto *List,* el mismo que el método *computeArea.* Sin embargo, no tiene en cuenta la distancia del último punto al primero. Por lo tanto, se suma esa distancia "distance1".

Más clases del API de *Google Maps*

- *GoogleMap*: para incorporar el mapa.
- *CameraPosition*: para hacer zoom en el mapa.
- *LatLngBounds*: para fijar unos límites o *bounds* a la imagen del mapa.
- *Marker*: para añadir marcadores.
- *MarkerOptions*: para diseñar los marcadores.
- *Polygon*: para dibujar un polígono.
- *PolygonOptions*: para diseñar el polígono.

ArrayList de *Java*

Definida anteriormente. Se trata de una clase que construye una lista de elementos de una clase determinada, como un vector. En la aplicación se usan

ArrayList de tipo <*Location*>, <*Integer*> (número entero), <*String*>, <*ArrayList*> y <*Double*> (número real).

DecimalFormat de Java

Para limitar el número de decimales que aparecen en las cadenas de texto. Por defecto son demasiados y para el objetivo de la aplicación no es necesaria tanta precisión.

4.5 MANUAL DE USO

Cuando el usuario inicia la aplicación por primera vez, se solicita aceptar los permisos de ubicación:

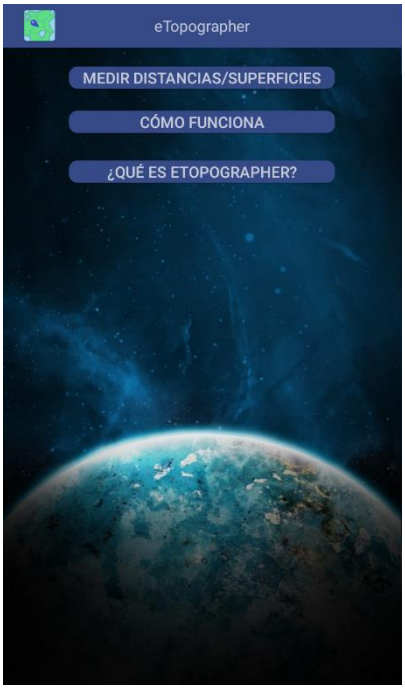
[19]

La aplicación contiene dos a*ctividades,* una que contiene el menú inicial y otra con el mapa y la opción de tomar puntos, guardar, etc. Cuando el usuario inicia la aplicación, se activa la *actividad* menú:

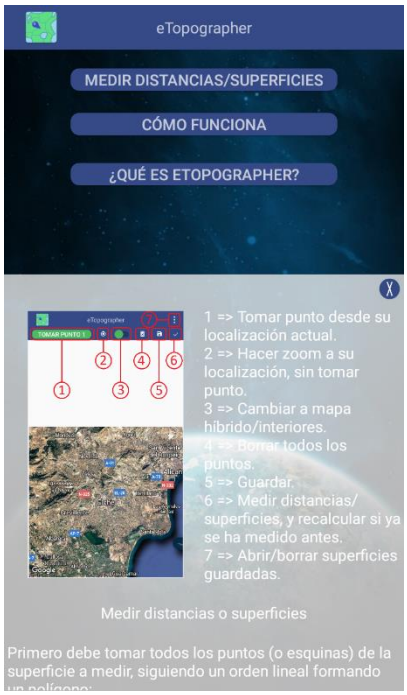
Actividad menú:

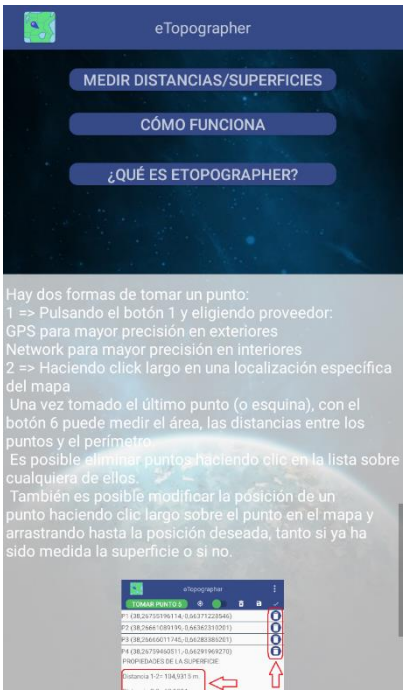
Contiene tres botones:

1. Para acceder a la a*ctividad* principal de medición de superficies.
2. Para una breve explicación sobre el funcionamiento de la aplicación.
3. Para una descripción de la aplicación y los requisitos de software para su correcto uso.

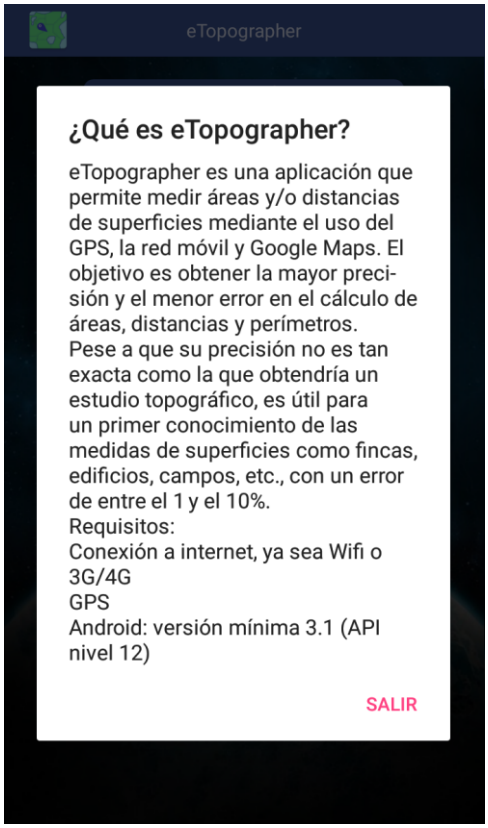


Al pulsar el segundo botón, aparece un breve manual de uso:





Al pulsar el tercer botón, aparece una descripción de la aplicación:



Actividad principal

Contiene siete botones:

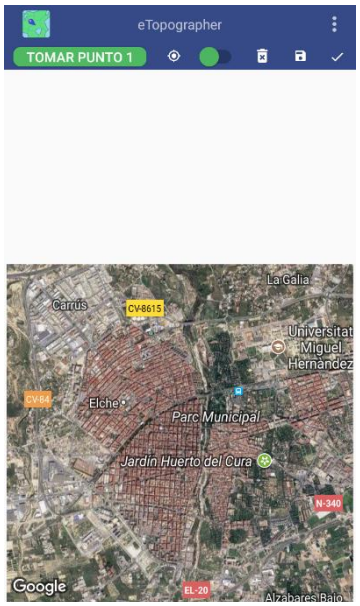
1. Tomar punto desde la posición actual del usuario, eligiendo entre dos proveedores: GPS para exteriores; y Network para interiores.
2. Hacer *zoom* en la localización actual del usuario, sin tomar ningún punto.
3. Cambiar tipo de mapa: híbrido (imagen) o de interiores.
4. Borrar todos los puntos.
5. Guardar superficie.
6. Calcular distancias, área y perímetro, así como dibujar el plano en el mapa.
7. Abrir/cargar y borrar superficies guardas.

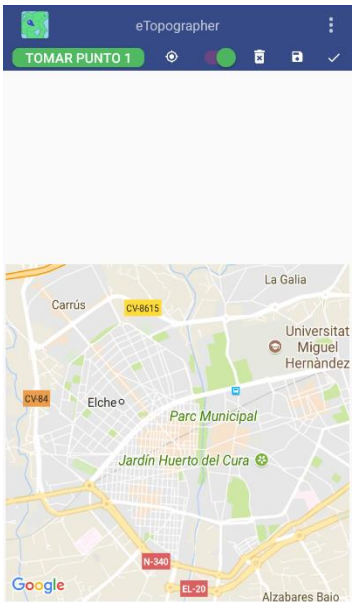
Hay dos opciones para tomar un punto:

- Mediante el botón 1 y seleccionando el proveedor acorde con nuestra situación.
- Haciendo clic prolongado sobre un punto específico del mapa.

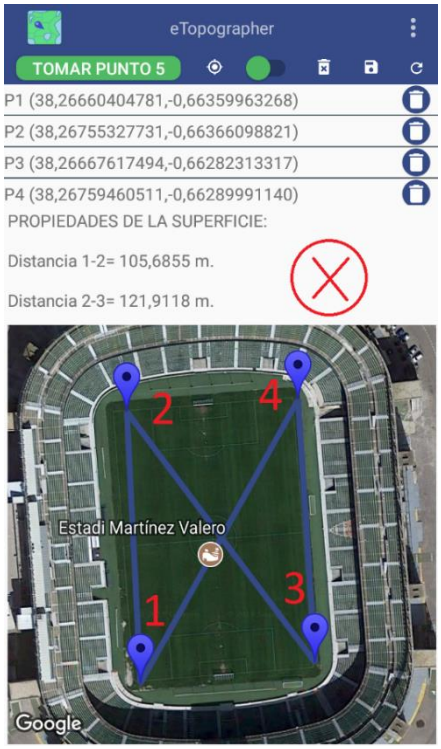
De este modo, el usuario debe tomar cuantos puntos (o esquinas) necesite para describir la superficie a estudiar. Una vez tomados todos los puntos, mediante el botón 6 se procede al cálculo de las propiedades de la superficie: distancia entre puntos, área y perímetro.

Con el botón tipo 3, de tipo *Switch*, se cambia entre mapa híbrido y de interiores:

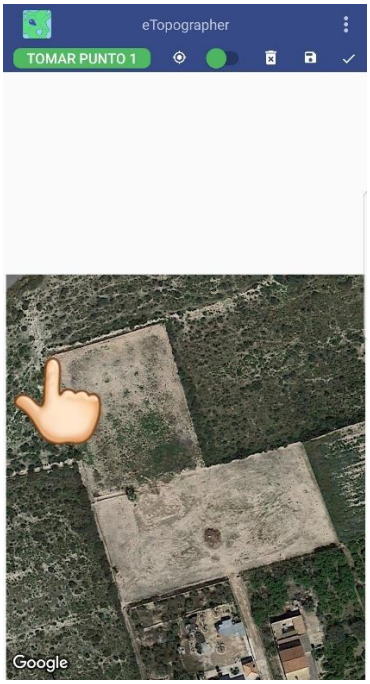


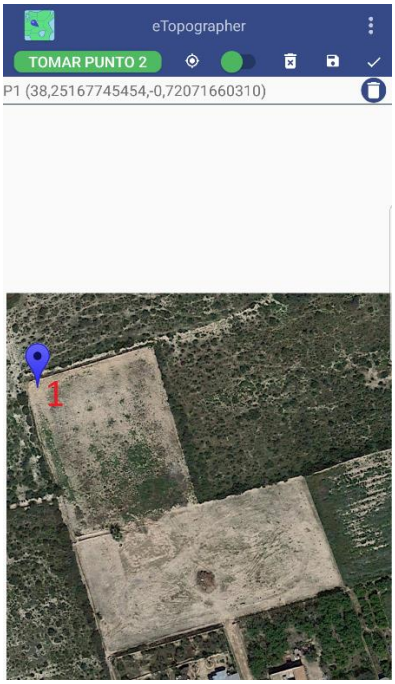


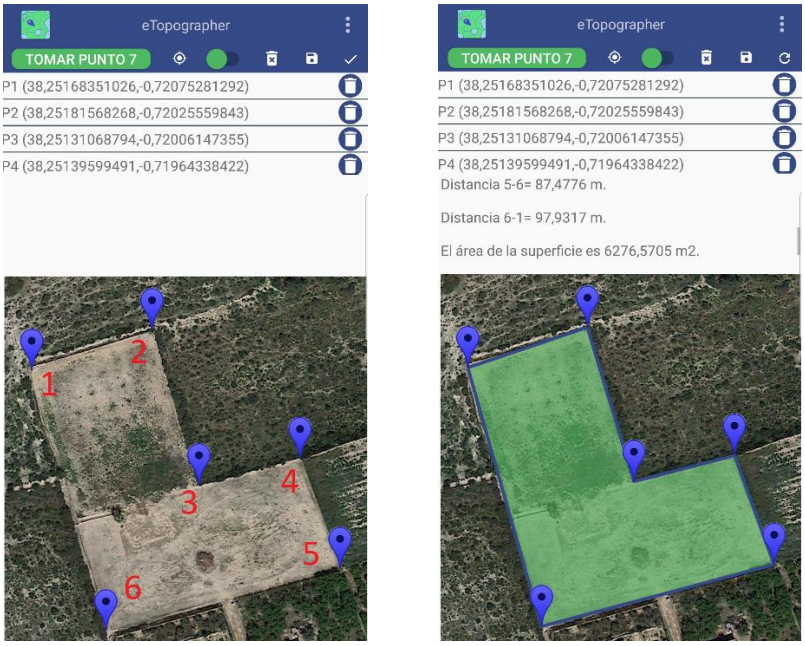
Es importante que, a la hora de tomar puntos, se siga un orden para la formación de la superficie, formando un polígono irregular y evitando el cruce de líneas:



Teniendo en cuenta los consejos anteriores, las siguientes imágenes muestran el proceso de medición de una superficie ejemplo de forma adecuada:





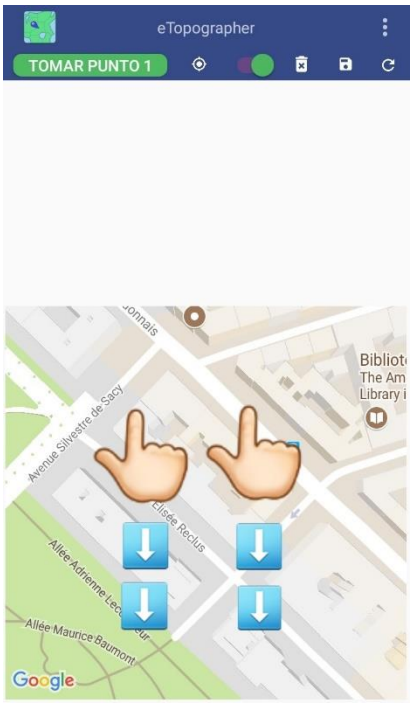


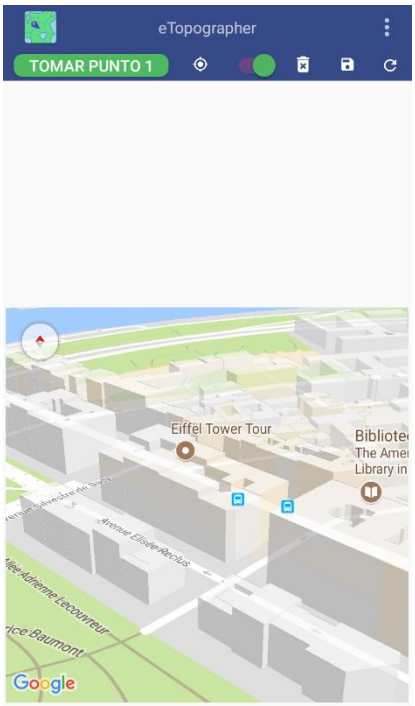
Como se observa, tras marcar todos los puntos y pulsar el botón 6, se dibuja el polígono y se calculan las medidas de la superficie.

Para eliminar un punto, basta con hacer clic sobre éste en la lista y aparecerá un diálogo para confirmar que se desea borrar ese punto. Al pulsar aceptar, el punto desaparecerá de la lista y del mapa, y la lista se reestructurará de forma que, si se ha borrado el punto 3, por ejemplo, el punto 4 pasa a ser el 3, el 5 a ser el 4, etc.

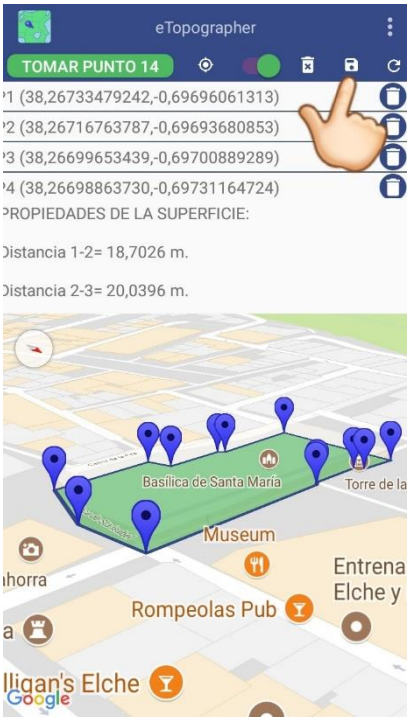
También es posible editar puntos, tanto si la superficie ha sido medida como si no, haciendo clic largo sobre el punto que se desee cambiar:

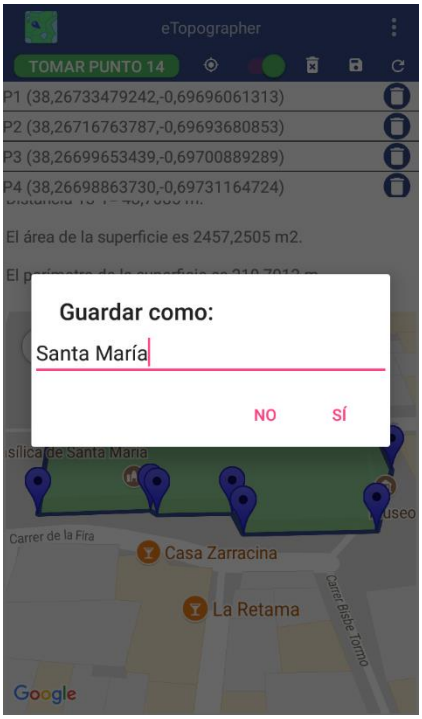
Como se ha comentado anteriormente, una de las características del API de *Google Maps* es la posibilidad de inclinar la cámara. Con esta opción, puede ser más sencillo tomar puntos. Además, en algunas ciudades importantes se ha incorporado la opción de ver edificios en 3D. Para inclinar la cámara basta con tocar con dos dedos el centro del mapa y deslizarlos hacia abajo:



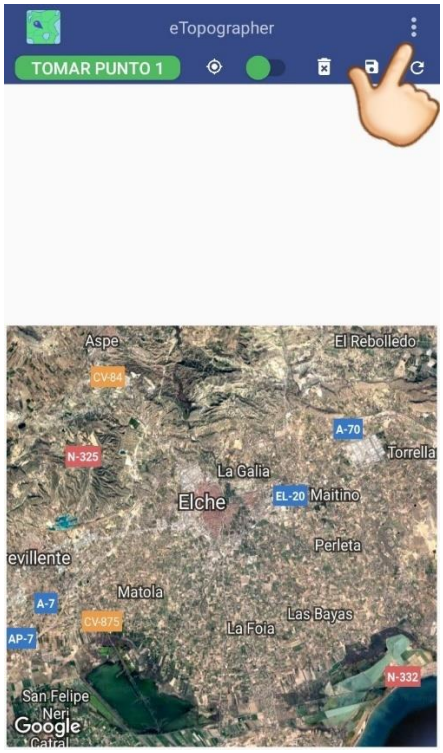


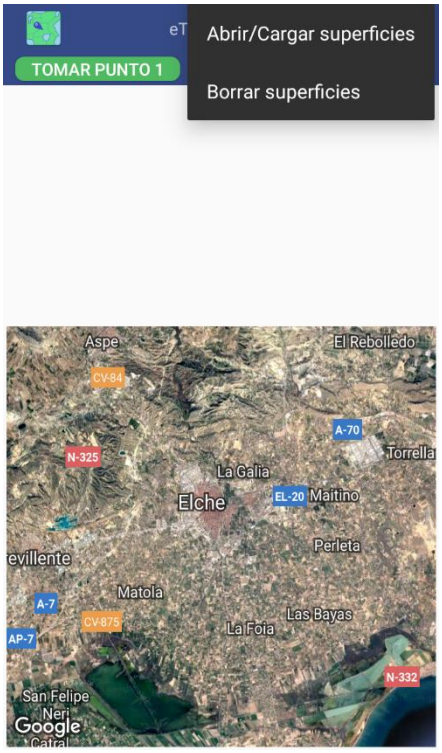
Para guardar una superficie:

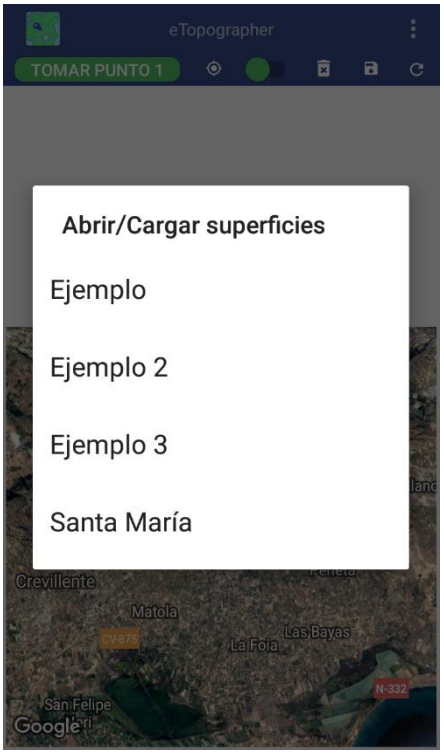




Y para abrir o borrar superficies guardadas:







Al pulsar borrar o abrir superficies, aparece una lista con las superficies guardadas que se desea borrar o abrir. En caso de querer borrar una de ellas, al pulsar sobre ella aparecerá un diálogo para verificar que se desea borrar. En caso de querer abrir una superficie, al hacer clic sobre la misma el mapa automáticamente carga esa superficie y acerca la cámara a la misma.

4.6 REQUISITOS

La aplicación exige unos requisitos mínimos de hardware y software para su correcto funcionamiento:

Requisitos de hardware:

- Dispositivo móvil con el SO *Android*, ya sea tableta o *Smartphone*.
- Conexión a internet, ya sea mediante Wifi o datos móviles (3G/4G).
- Conexión al sistema GPS.
- Memora RAM mínima de 512MB.
- Procesador CPU de 1Ghz mínimo.
- Memoria interna libre de 10MB.

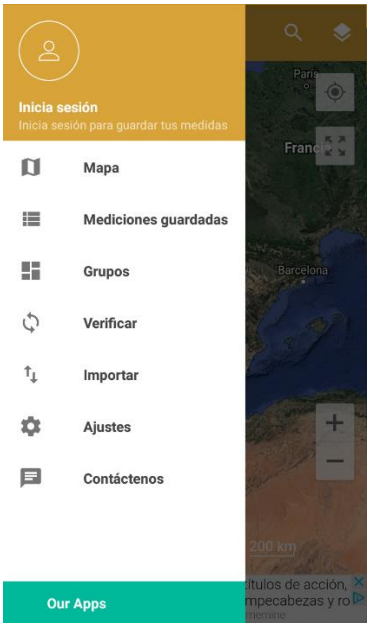
Requisitos de software:

- *Android* versión 3.1 (Nivel de API 12) o superiores.
- *Google Play Service*.
- *Google Maps*.

4.7 APLICACIONES SIMILARES

Fields Area Measure – Studio Noframe

Esta aplicación es una de las más completas para la medición de superficies y distancias. La *actividad* principal contiene el mapa, que ocupa toda la pantalla, y una serie de botones a su alrededor. Además, contiene un menú desplegable a la izquierda donde aparecen las opciones de guardar o abrir superficies.



En cuanto a la forma de medición, se asemeja bastante a la de este trabajo. En la esquina inferior izquierda hay un botón para elegir si se desea medir un área o un recorrido. Tras seleccionar, da a elegir entre tomar puntos mediante el GPS o mediante clic en el mapa. El objetivo de usuario es definir la superficie o el recorrido a medir mediante puntos.

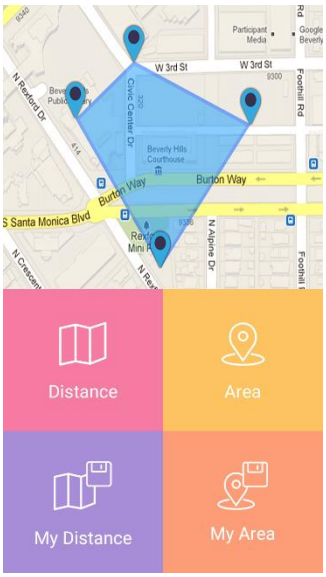
También tiene la opción de cambiar el tipo de mapa, aparte del modo de pantalla completa. Además, incluye la opción de editar puntos seleccionándolos y arrastrándolos hasta la posición deseada. Un ejemplo de una superficie sería:

GeoAreaMap – KBK INFOSOFT

Al igual que la anterior, en esta aplicación el mapa ocupa toda la pantalla. La interfaz es prácticamente igual a la anterior, con un menú desplegable a la izquierda con la opción de guardar superficies y abrirlas y con otro en la esquina superior derecha para cambiar el tipo de mapa.

Area Measurement – Background Changer, Eraser & Booth Photo Editor

A diferencia de las anteriores, esta aplicación tiene un menú y una interfaz más sencilla y amigable. Sin embargo, la forma de medir es la misma. La *actividad* inicial da a elegir al usuario si desea medir distancias o áreas. Abajo esta la opción de cambiar el tipo de mapa y arriba la de guardar.



5 RESULTADOS Y DISCUSIÓN

Este apartado es uno de lo más importantes del trabajo. En él se pretende presentar y discutir los resultados obtenidos, comparándolos con otros trabajos.

Tras investigar sobre las aplicaciones de medición de superficies presentes actualmente en el mercado, mencionadas en el apartado anterior, se pueden extraer varias conclusiones. Respecto a la forma de medición, tanto la aplicación desarrollada en este trabajo como las otras utilizan el mismo sistema: el usuario debe definir la superficie o recorrido mediante puntos, apoyándose en el mapa de *Google Maps*. En cuanto a la interfaz y las opciones del usuario, se puede discutir sobre puntos fuertes y débiles de la aplicación respecto a las otras.

La diferencia más destacable es el diseño de la interfaz. Mientras que en las otras el mapa ocupa prácticamente la totalidad de la pantalla, en esta se sitúa en la parte inferior y ocupa poco más de la mitad de la interfaz. Esta característica trae consigo ventajas y desventajas.

La principal desventaja aparece en la interacción del usuario con el mapa. Este es uno de los puntos débiles de la aplicación, así como el de no tener opción de pantalla completa. Al ocupar el mapa menos espacio en pantalla, el usuario tiene una peor visión de la situación geográfica de la zona a medir y, por consiguiente, necesita esforzarse más y utilizar el *zoom* para tomar el punto de forma más precisa.

Sin embargo, el hecho de tener en pantalla una lista con la posibilidad de eliminar puntos es un punto fuerte de la aplicación respecto a las otras. Gracias al *ListView* integrado en la parte superior, el usuario puede eliminar puntos del mapa de forma sencilla. Además, a diferencia de las otras aplicaciones, los puntos se van numerando para hacer más sencilla su eliminación.

Otro punto a destacar de la aplicación es que proporciona la información de las distancias entre los puntos, mientras que las otras muestran solo el área y el perímetro.

6 CONCLUSIONES

Durante el grado de Ingeniería Mecánica se adquieren conocimientos de distintas ramas de la ingeniería. En cuanto a programación, asignaturas como *Fundamentos de informática*, *Fabricación Asistida por ordenador* o *Teoría de sistemas*, entre otras, ayudan al alumno a aprender a trabajar con distintos lenguajes y softwares de programación. Dicho esto, este trabajo me ha permitido ampliar los conocimientos en esta ciencia tan apasionante y de la cual aún me queda mucho por aprender.

En cuanto al objetivo principal, este proyecto partió con el fin de desarrollar una aplicación relativamente compleja y completa, además de aprender lo máximo posible sobre la programación y el diseño de estas herramientas. Gracias a la investigación en Internet y a la ayuda de César, este objetivo ha sido alcanzado.

Respecto a los objetivos específicos iniciales de la aplicación, algunos se han conseguido como se esperaba mientras que otros han ido cambiando a medida que el proyecto avanzaba. Esto se debe al hecho de investigar sobre un tema tan arduo como es el de la programación, ya que, al inicio de este trabajo, plantear los objetivos que finalmente se establecieron habría resultado una tarea muy difícil. Sin embargo, tanto el entorno de desarrollo como las librerías incorporadas han posibilitado aumentar la lista de objetivos definitiva, recogida en el apartado 3.

La aplicación consigue, ayudándose del mapa de *Google Maps*, que el usuario pueda medir superficies de manera sencilla y rápida. Aun así, se trata de un mundo en el que nunca puede darse por finalizada una aplicación, de modo que, en el apartado siguiente, se proponen algunas sugerencias para dar continuidad a su desarrollo.

6.1 PROPUESTAS DE MEJORA

Posibilidad de compartir superficies con otros usuarios

Una opción que incorporan la mayoría de las aplicaciones y que es crucial para muchas de ellas es la posibilidad de compartir archivos. En este caso, sería interesante que el usuario pudiera compartir superficies con otros que tengan también instalada la aplicación.

Añadir la coordenada de altitud

Esta idea era una de las se barajaron al inicio de este trabajo. Para aquellas superficies en las que la pendiente o inclinación es considerable, es crucial conocer la variación de altitud entre sus puntos. Sin embargo, no se

introdujo finalmente en el proyecto por la complejidad que suponía tanto calcular como dibujar superficies en 3D.

Posibilidad de elegir las unidades de medida

Las unidades son fundamentales para cualquier estudio. Así que sería interesante dar la opción al usuario de elegir en qué unidades desea realizar las mediciones.

Dibujar las superficies “a mano alzada”

Algunas superficies no se pueden trazar de forma precisa mediante puntos porque su forma es algo compleja. Por ejemplo, cuando la superficie contiene curvas. Una buena posibilidad sería la de dibujar sobre el mapa la superficie a medir, sin tener que definirla necesariamente mediante puntos.

Añadir el modo de pantalla completa

Como hemos visto en el apartado de resultados, una opción que incorporan las otras aplicaciones es la del modo pantalla completa. Esto proporcionaría una mejor visión del mapa para el usuario.

Diferenciar entre medir una superficie o un recorrido

En algunas de las aplicaciones estudiadas al usuario se le da a elegir, previamente a la toma de puntos, si desea medir una superficie o un recorrido. Es cierto que la aplicación calcula tanto el área como el recorrido de la superficie, pero no diferencia entre cálculo de recorridos o de áreas. Por lo tanto, añadir la opción de calcular solo el recorrido sería apropiado, ya que, si solo se desea medir eso, dibujar el polígono resulta innecesario.

7 BIBLIOGRAFÍA

[1] Ramos Salavert, I., & Lozano Pérez, M. D. (2000). *Ingeniería del software y bases de datos: tendencias actuales.* Universidad de Castilla La Mancha.

[2] Centro para la Innovación y Desarrollo de la Eduacion (CIDEAD). (s.f.). *Polígono, perímetros y áreas*.

[3] Porto, J. P., & Gardey, A. (2016). *Definición de target*.

URL: https://definicion.de/target/

[4] *Android Developer*. (s.f.). *Actividades*.

URL: https://developer.android.com

[5] International Telecommunications Union (ITU), I. T. (s.f.). Gráfico.

URL: https://www.itu.int/en/ITU-D/Statistics

[6] Gartner. (2013). Gráfico Sistemas Operativos.

URL: https://www.gartner.com

[7] *Real Academia Española*(s.f.). *Definición de topografía*.

[8] National Geographic (s.f.). *El pago de los impuestos en el Antiguo Egipto*.

URL: http://www.nationalgeographic.com.es/historia/grandes-reportajes/el-pago-de-los-impuestos-en-el-antiguo-egipto_7372/3

[9] Wikipedia: la enciclopedia libre. *Agrimensura*.

URL: https://es.wikipedia.org/wiki/Agrimensura

[10] Instop - Estación Total Leica FlexLine TS06 Plus.

URL: https://www.instop.es/flexline_plus/flexline_TS06_plus.php

[11] Wikipedia, la enciclopedia libre. *Sistema de posicionamiento Global.*

URL: https://es.wikipedia.org/wiki/Sistema_de_posicionamiento_global

[12] Mathematics Dictionary (22 de febrero de 2018). *Latitud y longitud*.

URL: http://www.mathematicsdictionary.com/spanish

[13] Deetz, Charles H (1944). Elementos de proyección de mapas y su aplicación a la construcción de mapas y cartas. Washington. Desde Wikipedia. *Sistema de coordenadas universal transversal de Mercator (UTM).*

[14] Aronsson, Lars. (1 de septiembre de 2006). *Wikipedia la enciclopedia libre*. *Imagen del mapa de Europa en coordenadas UTM.*

URL:https://es.wikipedia.org/wiki/Sistema_de_coordenadas_universal_transversal_de_Mercator

[15] *Android Developer*. (s.f.). *Android Studio*.

URL: https://developer.android.com

[16] Universidad Poltécnica de Valencia. Máster en desarrollo de Aplicaciones Android. *Ciclo de vida de una actividad.*

URL: http://www.androidcurso.com/index.php/tutoriales-android/37-unidad-6-multimedia-y-ciclo-de-vida/158-ciclo-de-vida-de-una-actividad

[17] *4rsoluciones*. (2013). *Diferencias entre SDK, API, Biblioteca, Framework.*

URL:http://www.4rsoluciones.com/blog/framework-sdk-biblioteca-api-cuales-son-las-diferencias-2/

[18] Android Developer, StackOverFlow, Hermosa Programación, GitHub, Proyecto Simio, Android Curso. *Soporte para el desarrollo del trabajo.*

URLs: https://developer.android.com

https://es.stackoverflow.com

http://www.hermosaprogramacion.com

http://www.proyectosimio.com

http://www.androidcurso.com

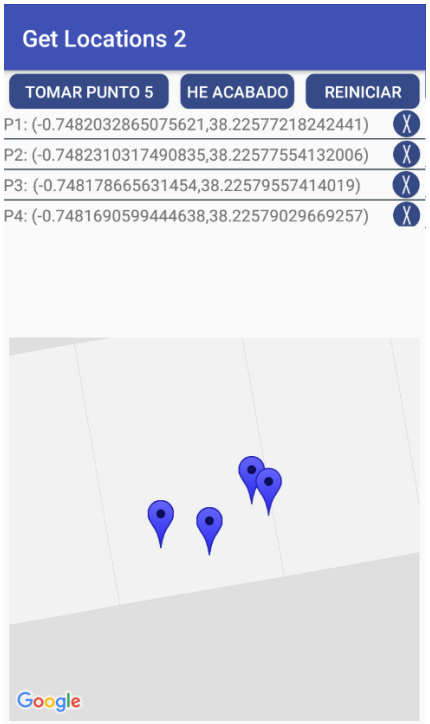
[19] *Pixabay*. (2018). *Imagen para el fondo de pantalla de la aplicación.*

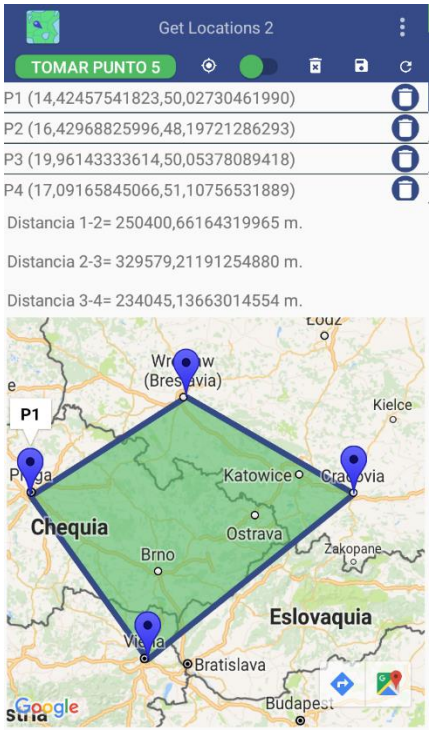
URL: https://pixabay.com/es/espacio-galaxy-planeta-universo-1565986/

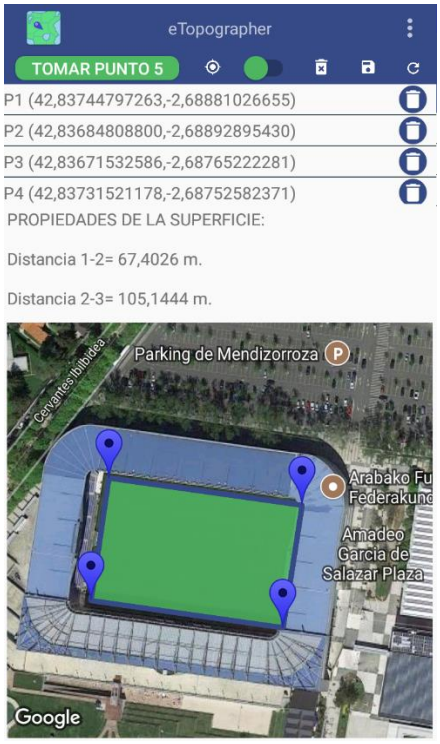
8 ANEXOS

8.1 ANEXO I

Evolución cronológica de la aplicación:







Se muestran seis versiones de la aplicación, en orden cronológico de izquierda a derecha y de arriba a abajo. La primera, tras incorporar el mapa de *Google Maps*. Contiene dos botones: para tomar puntos y para calcular las medidas. Además, ya muestra en el mapa los puntos y el polígono.

La segunda versión varía estéticamente y añade más información en el *ScrollView*. A partir de la tercera, ya se incorpora el listado de puntos a través del *ListView*, y la opción de eliminarlos. Respecto a la cuarta versión, ya contiene el menú desplegable, el icono de la aplicación y más cambios estéticos. La quinta ya es la versión final, a falta de modificar tanto el nombre de la aplicación como detalles como el número de decimales mostrados, cambios que ya se incluyen en la versión final.