UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA GEODÉSICA, CARTOGRÁFICA Y TOPOGRÁFICA.

TRABAJO FIN DE CARRERA

DESARROLLO DE UNA APLICACIÓN MÓVIL EN ANDROID PARA CARTOGRAFIAR Y PROCESAR RUTAS EN FORMATO GPX

AUTOR: DAVID FRÍAS GARRIDO

TUTORA: LAURA SEBASTIÁ TARÍN

VALENCIA, FEBRERO 2014

Contenido

Índice de Imágenes y Figuras:

Índice de Tablas:

1. Introducción

El presente documento tiene como finalidad la obtención del título de Ingeniero en Geodesia y Cartografía, en la Escuela Técnica Superior de Ingeniería Geodésica, Cartográfica y Topográfica de la Universidad Politécnica de Valencia.

El objetivo de este trabajo final de carrera es aplicar los conocimientos adquiridos durante los años de estudio, para desarrollar una aplicación móvil para Android, basada en cartografiar la ruta de una actividad, tomando las diferentes localizaciones del usuario.

Se han realizado dos experimentaciones previas mediante los programas SL4A y Processing (junto con el kit de desarrollo para Android), antes de enfrentarse a la aplicación objeto. Después de los dos ensayos, se realiza la principal aplicación objeto de estudio. Con el programa Eclipse y Android Software Development Kit (Android SDK), se desarrolla una completa aplicación móvil. Y para finalizar se verán unas comparativas y conclusiones entre los programas empleados para realizar la aplicación.

La aplicación desarrollada en SL4A se ha implementado mediante el lenguaje de programación Python, mientras que para las aplicaciones desarrolladas en Processing y en Eclipse, se ha utilizado el lenguaje de programación Java. Hay que mencionar que las tres aplicaciones del trabajo final de carrera, se han desarrollado bajo el sistema operativo Android en su versión 4.1.2, también llamada Jelly Bean y se han realizado con el *Smartphone* Samsung Galaxy SII.

Este documento comienza describiendo las dos aplicaciones en SL4A y Processing, realizadas con dos programas que se vieron en diferentes asignaturas de la titulación de Ingeniero en Geodesia y Cartografía, donde se han aplicado los conocimientos aprendidos para el mundo Android. Con ello, se pretende adquirir un bagaje inicial, para poder enfrentarse a la aplicación objeto (realizada con Eclipse) con la mayor solvencia posible.

En primer lugar, se describe la aplicación realizada con SL4A, DFTracker. Dicha aplicación toma la ubicación de la persona cada 30 segundos y una vez acabada la actividad, muestra en un mapa la ruta realizada. El capítulo concluye con la otra aplicación creada con el programa Processing, DFTrail. Esta aplicación está diseñada para orientarse en la actividad, de forma que el usuario se puede ubicar mediante la brújula y las coordenadas de su posición y ver en el mapa donde está localizado. El siguiente capítulo describe el formato GPX y su estructura.

A continuación, se lleva a cabo la explicación detallada de la aplicación objeto de este estudio, TraLoc. Destaca por crear una ruta a partir de la localización de la actividad realizada por el usuario, cargar archivos de ruta de otras personas en la aplicación, visualizar cualquier ruta en un mapa y enviar rutas por email. TraLoc tiene la capacidad de manejar los ficheros GPX, en sus versiones 1.0 y 1.1 para visualizar la

ruta y en la versión 1.1 para crear el archivo de la ruta GPX. Además tiene compatibilidad con otros programas y páginas web para utilizar los ficheros GPX.

Posteriormente, se exponen las conclusiones realizando un análisis comparativo de las aplicaciones, detallando los puntos fuertes y débiles del uso de estas tres aproximaciones para el desarrollo de aplicaciones móviles.

Por último, en el apartado "6. Instalación de los programas", se detallará los pasos a seguir para realizar la instalación de las tres herramientas utilizadas, SL4A, Processing y Eclipse.

1.1. Localización de las pruebas realizadas

Cada aplicación se ha ido comprobando durante su desarrollo, con lo que se han llevado a cabo diferentes pruebas en diferentes lugares.

Para las tres aplicaciones se han realizado pruebas en el municipio de Sagunto, para ser más exactos en la localidad de Puerto de Sagunto. Su emplazamiento respecto a la localidad ha sido en la zona Sur, al lado de la carretera "Acceso IV Planta" y la "Avenida Tres de Abril".

Imagen 1 - Puerto de Sagunto

Además, para la aplicación de Processing y Eclipse, se hicieron otras pruebas en el campus de la Universidad Politécnica de Valencia. Su localización respecto al campus es la zona Nord-Este. Para ser más concretos, junto a la Escuela Técnica Superior de Ingeniería Geodésica, Cartográfica y Topográfica.

Imagen 2 - Plano UPV

Para Processing:

Imagen 3 - ETSIGCT Oeste

Para Eclipse:

Imagen 4 - ETSIGCT Este

Otras de las pruebas realizadas para la aplicación TraLoc ha sido un recorrido que se ha realizado en coche y que ha servido para realizar los ajustes pertinentes, que se explicarán posteriormente con más detalle. El trayecto realizado ha sido del Puerto de Sagunto hasta Almenara, pasando por la Estación de los Valles.

Imagen 5 - Pto. Sagunto - Almenara

La última zona donde se han realizado pruebas para Eclipse ha sido en el municipio de Gilet, hacia Los Monasterios, por la zona de Santo Espíritu, ascendiendo al pico del Águila (representado en rojo) y al monte Picayo (representado en azul), comprobando las pruebas para montaña.

Imagen 6 - Gilet

Resumiendo, para Eclipse se han realizado tres tipos de pruebas:

- Andando por zona urbana (UPV, Puerto de Sagunto)
- Andando por zona de montaña (Gilet)
- En vehículo por zona inter-núcleos (Puerto de Sagunto – Almenara)

1.2. Por qué de la elección de Android

Existen cinco plataformas móviles diferentes entre sí, con características distintas. A continuación, se puede ver un cuadro resumen[1], comparando las especificaciones más destacables de cada una de ellas:

	Apple iOS 7	Android 4.3	Windows Phone 8	BlackBerry OS 7	Symbian 9.5
Compañía	Apple	Open Handset Alliance	Microsoft	RIM	Symbian Foundation
Núcleo del SO	Mac OS X	Linux	Windows NT	Mobile OS	Mobile OS
Licencia de *software*	Propietaria	Software libre y abierto	Propietaria	Propietaria	Software libre
Año de lanzamiento	2007	2008	2010	2003	1997
Fabricante único	Sí	No	No	Sí	No
Variedad de dispositivos	modelo único	muy alta	media	baja	muy alta
Soporte memoria externa	No	Sí	Sí	Sí	Sí
Motor del navegador web	WebKit	WebKit	Pocket Internet Explorer	WebKit	WebKit
Soporte Flash	No	Sí	No	Si	Sí
HTML5	Sí	Sí	Sí	Sí	No
Tienda de aplicaciones	App Store	Google Play	Windows Marketplace	BlackBerry App World	Ovi Store
Número de aplicaciones	825.000	850.000	160.000	100.000	70.000
Coste publicar	$99 / año	$25 una vez	$99 / año	sin coste	$1 una vez
Actualizaciones automáticas del S.O.	Sí	depende del fabricante	depende del fabricante	Sí	Sí
Familia CPU soportada	ARM	ARM, MIPS, Power, x86	ARM	ARM	ARM
Máquina virtual	No	Dalvik	.net	Java	No
Aplicaciones nativas	Siempre	Sí	Sí	No	Siempre
Lenguaje de programación	Objective-C, C++	Java, C++	C#, muchos	Java	C++
Plataforma de desarrollo	Mac	Windows, Mac, Linux	Windows	Windows, Mac	Windows, Mac, Linux

Tabla 1 – Comparativa plataformas móviles

La plataforma Android destaca sobre el resto de plataformas móviles por los siguientes aspectos:

- **Plataforma abierta**. Desarrollo libre basado en Linux y de código abierto.
- **Portabilidad asegurada**. Al desarrollarse en Java, se asegura que se pueden ejecutar en cualquier CPU (máquina virtual).
- **Adaptable a cualquier tipo de hardware**. Arquitectura basada en componentes inspirados en internet.

[1] Extraído del libro "El gran libro de Android"

- **Filosofía de dispositivo siempre conectado a internet**.
- **Gran cantidad de servicios incorporados**. GPS, bases de datos SQL, navegador…
- **Seguridad**. Cada aplicación limita su actuación mediante los permisos
- **Optimización para baja potencia y poca memoria**. La Máquina Virtual Dalvik es una implementación de la máquina virtual Java optimizada para dispositivos móviles.
- **Alta calidad de gráficos y sonidos**. Incorpora códec estándar de audio y video.

Además de estas características, según un estudio de la empresa Gartner[2], consultora y de investigación de las tecnologías de la información, afirmó que "Android dominará los *Smartphone* para el 2014"[3]. Y según una publicación reciente[4], se corrobora, "ya que en el tercer cuatrimestre de 2013 (Noviembre), Android superó la cuota de mercado del 80%, lo que ayudo a extender su liderazgo. De las ventas realizadas por Android, el 41% se sitúan en China continental, donde hace un año eran de un 34%".

Operating System	3Q13 Units	3Q13 Market Share (%)	3Q12 Units	3Q12 Market Share (%)
Android	205,022.7	81.9	124,552.3	72.6
iOS	30,330.0	12.1	24,620.3	14.3
Microsoft	8,912.3	3.6	3,993.6	2.3
BlackBerry	4,400.7	1.8	8,946.8	5.2
Bada	633.3	0.3	4,454.7	2.6
Symbian	457.5	0.2	4,401.3	2.6
Others	475.2	0.2	683.7	0.4
Total	**250,231.7**	**100.0**	**171,652.7**	**100.0**

Tabla 2 - Venta de los *Smartphone* en todo el mundo a usuarios, finales del 3Q2013 (miles de unidades)

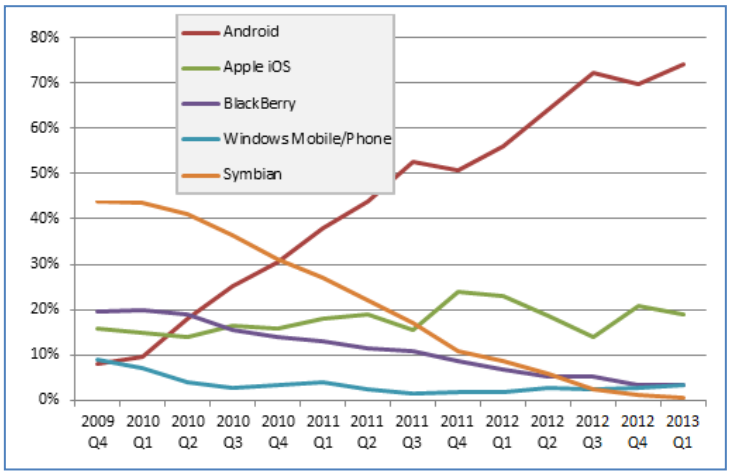

Figura 1 - Porcentaje de *Smartphone* vendidos en el mundo según su sistema operativo hasta el tercer cuarto del 2013

[2] http://www.gartner.com/technology/home.jsp
[3] http://www.gartner.com/newsroom/id/1434613
[4] http://www.gartner.com/newsroom/id/2623415

2. Experimentación previa

Antes de empezar a realizar la aplicación objeto del trabajo final de carrera con el programa Eclipse, se han realizado dos aplicaciones con otras dos herramientas de trabajo, SL4A y Processing.

El objetivo principal del desarrollo de estas aplicaciones ha sido tener una primera toma de contacto con el mundo Android y ver su funcionamiento. Por otro lado, también se ha querido mostrar que pueden desarrollarse aplicaciones móviles con otros programas no tan extendidos, dejando patente que se pueden realizar aplicaciones tan válidas como con Eclipse, aunque con un funcionamiento más rudimentario.

La temática principal de cada una de las tres aplicaciones desarrolladas es la misma: geolocalizar al usuario en su actividad.

2.1. SL4A

Esta primera aplicación, llamada DFTracker, se desarrolla con la herramienta SL4A[5] (Scripting Layer for Android), en su versión r6 y con el Script de Python.

El procedimiento de instalación se describe en el apartado "6.1. SL4A" de este documento.

La aplicación desarrollada mediante SL4A está diseñada para corredores, práctica de senderismo y en general, para la práctica de cualquier deporte o actividad donde la persona que utilice la aplicación pretenda cartografiar la actividad y situarla sobre el mapa.

A continuación, se procede a la explicación del funcionamiento de la aplicación. El primer paso que se requiere es acceder a los servicios de ubicación del teléfono móvil y conectar la localización por la red del operador móvil y GPS.

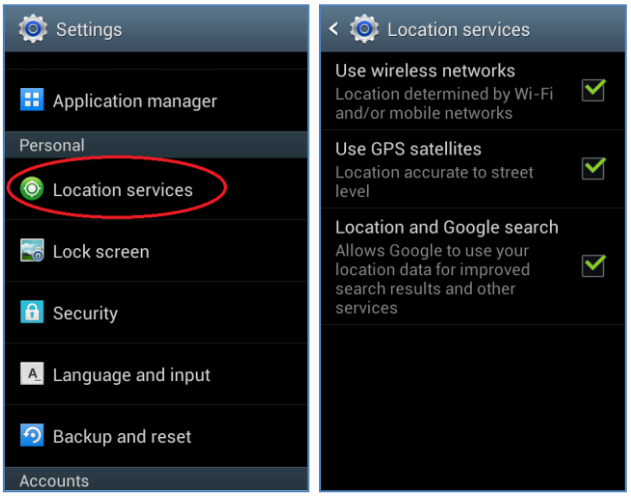

Imagen 7 - Servicios de localización SL4A

[5] http://code.google.com/p/android-scripting/

Seleccionados los servicios de localización en el móvil, para poder ejecutar correctamente la aplicación diseñada, se accede a la aplicación SL4A. Dentro del menú de SL4A, se selecciona el archivo "*DFTracker.py*" y se pulsa la opción *RUN* (primera opción), y a partir de ese momento se ejecuta la aplicación desarrollada.

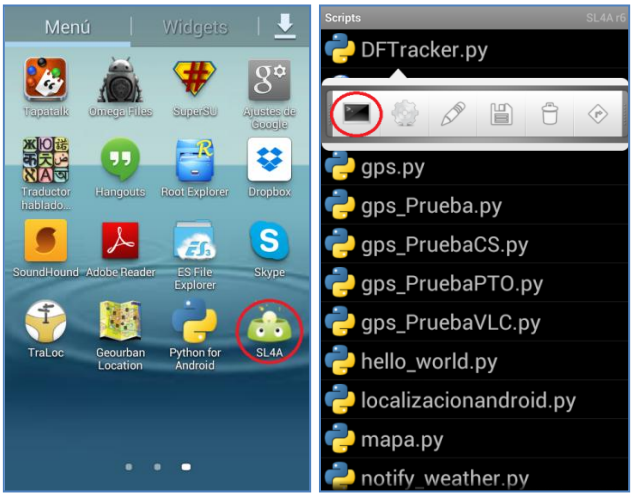

Imagen 8 - SL4A

Al iniciar la ejecución de DFTracker, aparece una cabecera con el nombre del programa. A continuación, sale un mensaje (*toast*), en el que se pregunta al usuario, si quiere seguir ejecutando el programa. Si el usuario decide no seguir con el programa, se muestra por pantalla un mensaje y la aplicación se dará por terminada.

En caso contrario, si el usuario decidiera proseguir, vuelve a salir otro mensaje, donde avisa si se quiere fotografiar la posición de salida de la actividad que se vaya a realizar. Este *toast* no es restrictivo como el anterior, se acepte o no, el programa sigue ejecutándose y la imagen se guardará en la ruta "*/sdcard/sl4a/scripts/Captura.png*". Se ha de tener en cuenta que, aunque en la ruta se especifique *sdcard*, la imagen se guarda en la memoria interna del teléfono. En el desarrollo de las versiones de Android, va cambiando la nomenclatura de las memorias internas y externas, por lo que hay que tener especial cuidado en este aspecto. Por el momento (Jelly Bean, 4.1.2) *sdcard* hace referencia a la memoria interna, mientras que *extSdCard* hace referencia a la memoria externa.

Acto seguido, vuelve a salir otro *toast*, preguntando cuánto tiempo va a durar la actividad a desarrollar. Esto implica que el usuario ha de tener prevista una estimación del tiempo que durará la actividad. Lo más importante en este mensaje, es que se ha de introducir el tiempo en segundos, como así se especifica en el *toast*. Por defecto, muestra la duración de la actividad en 60 segundos.

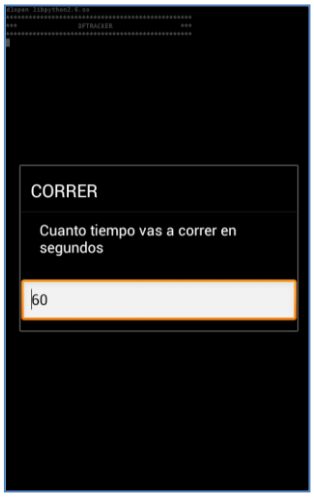

Imagen 9 - Tiempo de la actividad

Una vez seleccionado el tiempo, en el momento que se acepta, el *Smartphone* vibra, para avisar al usuario que se empiezan a captar datos de posicionamiento. A partir de este momento, la aplicación empezará a tomar la posición del usuario cada 30 segundos. Esto se llega a conseguir gracias a la opción *sleep(30)*. A continuación se muestra el código para obtener la localización:

```
#Para que tome los datos GPS en intervalos de 30 segundos
droid = android.Android()
droid.dialogCreateInput("CORRER","Cuanto tiempo vas a correr en
segundos","60","number")
droid.dialogShow()
Y=droid.dialogGetResponse().result["value"]
droid.dialogDismiss()
X=0 #Variable para contar el tiempo
print "Reading GPS ..."
#Hace que vibre el telefono
droid.vibrate(500)
while X <= float(Y):
      datosGPS()
      #Para que no espera 30 seg antes de la ultima peticion de GPS
      if X >= float(Y):
            break
      sleep(30) #Paramos 30 seg para que vuelva a tomar la
      localizacion cada 30 seg
      X=X+30
```

Cuando el usuario ha introducido el tiempo de la actividad, se ha guardado en la variable Y, que se compara con X (variable para contar el tiempo) y mientras X sea menor o igual que la actividad a realizar, se irá pidiendo la ubicación del usuario. Una vez obtenida, se llama al método *sleep*, donde se duerme el proceso el tiempo estipulado, en este caso 30 segundos, y después de ello, se dispone a obtener una nueva

ubicación, incrementado antes X en 30. Este bucle finalizará cuando el tiempo que se va contando en X, sea mayor o igual que el tiempo de la actividad a realizar.

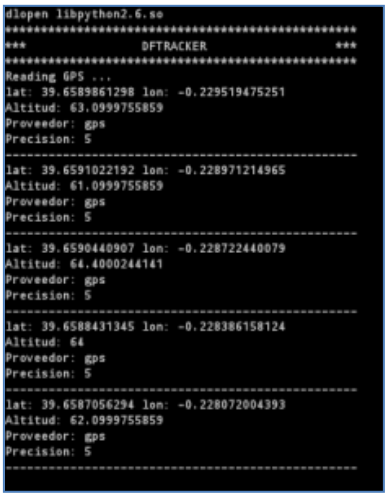

Imagen 10 - Obtención de las coordenadas

Una vez terminado el tiempo previsto para la actividad, la aplicación para de tomar ubicaciones y muestra un mensaje, donde el usuario tiene 10 segundos para tomar un *screenshot* de todas las posiciones que se han ido obteniendo por pantalla. Pasado ese tiempo, a través de la pantalla DFTracker avisa de qué se va a mostrar el recorrido realizado en mapa.

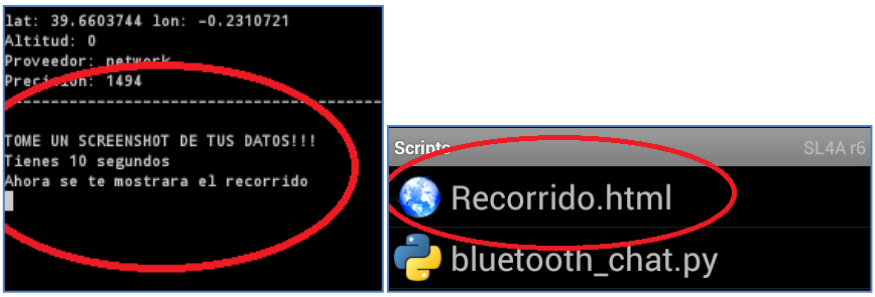

Imagen 11 - Aviso para mostrar el recorrido

En este momento, se genera un archivo .html guardado en la misma ruta que la captura de pantalla, "*/sdcard/sl4a/scripts/Recorrido.html*". Ese archivo se auto lanza, mostrándose el mapa, después de haber avisado por pantalla. El archivo generado, "*Recorrido.html*", se abrirá mediante el navegador predeterminado que se tenga en el *Smartphone*. Este archivo se puede pasar al PC y visualizarse mediante cualquier navegador que se tenga instalado en el ordenador.

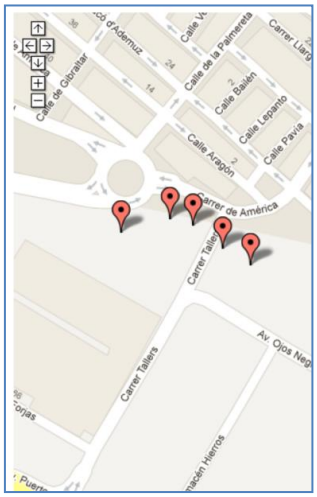

Imagen 12 - Mapa recorrido

La aplicación funcionará de forma diferente dependiendo de la versión de Android de la que se disponga en el móvil. Por ejemplo, con la versión 4.1.2 (Jelly Bean) de Android, el archivo "*Recorrido.html*" no se abrirá automáticamente, pone el aviso de que se va a mostrar el mapa, pero en vez de mostrarse el mapa, se sale de la ejecución, termina la aplicación. Al terminar de ejecutarse DFTracker, se vuelve al menú SL4A y ahí, se puede observar que se crea correctamente el archivo html y se puede ejecutar viendo el recorrido realizado a través del mapa.

Por el contrario, si la aplicación se ejecuta en la versión 2.3.7 (Gingerbread) de Android, sí se ejecuta automáticamente el archivo "*Recorrido.html*", se auto lanza. Por lo que se ha podido verificar, es debido a SL4A y a su compatibilidad con el nuevo sistema de Android, Jelly Bean. Se ha de mencionar que en el momento de dichas pruebas, Jelly Bean llevaba escaso tiempo en el mercado, por lo que muchos desarrolladores no habían podido adaptar sus aplicaciones a la nueva versión.

Volviendo al archivo de la ruta que se crea, visualizándolo, se puede pinchar sobre cualquier *Marker* (marcador) situado en el mapa. Estos muestran un bocadillo con la información del punto de recorrido de la ruta.

Imagen 13 - Bocadillo *Marker*

2.1.1. API Mapas y librería PYMAPS

Gracias a la utilización de la librería pymaps[6] para Python, se ha podido obtener de una manera sencilla el mapa y su posterior visualización a través del navegador.

Lo que se ha realizado en la aplicación, ha sido llamar a la librería, asignándole unas coordenadas para centrar el mapa, un zoom para su mejor visualización y se han añadido los *Marker* de las ubicaciones tomadas en la actividad, devolviendo el mapa creado.

```
#Se llama a la libreria pymaps para que saque el mapa
    mapa = PyMap()
    mapa.maps[0].center = Coord #str(LatLon[0])   #Metemos las
coordenadas para centrar el mapa

    #Clave de la api de Google Maps
    mapa.key = "ABQIAAAAQQRAsOk3uqvy3Hwwo4CclBTrVPfEE8Ms0qPwyRfPn-
DOTlpaLBTvTHRCdf2V6KbzW7PZFYLT8wFD0A"

    #Zoom para el mapa
    mapa.maps[0].zoom = 17

    i=0
    for locat in LatLon:
        Placemark=[locat[0],locat[1] ,'Punto de Recorrido '+str(i+1)]
        mapa.maps[0].setpoint(Placemark)
        i=i+1
          return mapa
```

El mapa creado se pasa al método *writeToHTML*, para escribirlo en html y posteriormente visualizarlo en el navegador del *Smartphone*.

```
def writeToHTML(htmlString):
#Metodo para escribir a un archivo, por defecto, al Recorrido.html
    try:
        file = open('///sdcard/sl4a/scripts/Recorrido.html','w')
        file.write(htmlString)
        file.close()
    except IOError:
        print "Error al escribir archivo"


mapafinal=MapaRecorrido(LatLon)
writeToHTML(mapafinal.showhtml())
      droid.webViewShow('file:///sdcard/sl4a/scripts/Recorrido.html')
```

Aparte de utilizar la librería pymaps, se ha de incluir la clave del API[7] de Google Maps, para que se puedan utilizar los mapas en la aplicación. Se ha de configurar la

[6] http://code.google.com/p/pymaps/source/browse/trunk/pymaps.py

[7] Application Programming Interface, conjunto de procedimiento o métodos que ofrece una biblioteca para ser utilizado por otro software

API, asignándosela al mapa (objeto). Sin asignar la clave de Google Maps, la aplicación no tendría los permisos suficientes y no se podrían utilizar los mapas.

```
    #Clave de la api de Google Maps
          mapa.key =
"ABQIAAAAQQRAsOk3uqvy3Hwwo4CclBTrVPfEE8Ms0qPwyRfPn-
DOTlpaLBTvTHRCdf2V6KbzW7PZFYLT8wFD0A"
```

2.1.2. Localización

La aplicación está diseñada para poder obtener la ubicación a través del *Smartphone* del usuario, mediante el GPS o por la red de internet del operador.

En primer lugar, se intenta acceder al GPS y si no se consigue, se obtiene la ubicación mediante el proveedor de internet. Además de obtenerse las coordenadas de latitud y longitud, se obtienen otros parámetros (altitud, proveedor y precisión).

En la etiqueta de proveedor, se mostrará si los datos proceden del GPS o de la *Network* (red del operador de internet) del teléfono móvil.

```
#Coge los datos GPS de latitud, longitud....
droid = android.Android()
droid.startLocating()
event = droid.eventWaitFor('location',10000).result
if event['name'] == "location":
      try:
            lat = str(event['data']['gps']['latitude'])
            lon = str(event['data']['gps']['longitude'])
            alt = str(event['data']['gps']['altitude'])
            proveedor = str(event['data']['gps']['provider'])
            precision = str(event['data']['gps']['accuracy'])

      except KeyError:
            lat = str(event['data']['network']['latitude'])
            lon = str(event['data']['network']['longitude'])
            alt = str(event['data']['network']['altitude'])
            proveedor = str(event['data']['network']['provider'])
            precision = str(event['data']['network']['accuracy'])
```

Se han realizado diferentes pruebas, tanto por la red del operador de internet como por el GPS integrado en el teléfono. Con el GPS, se alcanza una precisión de unas 5 unidades, mientras que con la *Network* los mejores resultados obtenidos son de 1494 unidades, es decir, como era de esperar, el GPS obtiene una mejor precisión. Según las experiencias desarrolladas en este proyecto y también considerando la experiencia de otros usuarios de *Smartphone*, se presupone que la precisión viene dada en metros, pero otros desarrolladores de aplicaciones, discrepan por los resultados obtenidos en sus pruebas. Por tanto no se sabe a ciencia cierta en que unidades se obtiene la precisión, y tampoco se especifica las unidades en ningún sitio oficial. Se tendría que experimentar

sobre este apartado, comparando las coordenadas obtenidas mediante DFTracker con coordenadas corregidas.

Como se ha mencionado, accediendo a la ubicación, a través de la red del proveedor, la precisión es deficiente, por lo tanto, los puntos no se localizan correctamente en su posición. Se puede observar en la parte izquierda de la imagen 14, como los dos primeros puntos están colocados en una posición errónea, ya que su correcta posición se sitúa como se muestra en la parte derecha de la imagen 14. Se puede observar claramente el desplazamiento existente entre ambas posiciones.

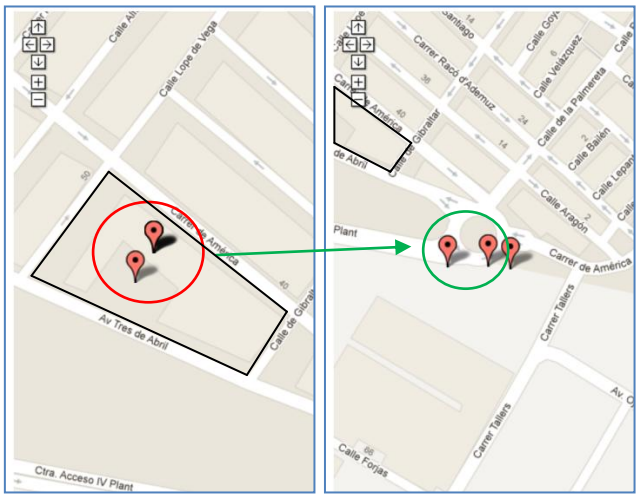

Imagen 14 - Posiciones erróneas

Por la razón detallada anteriormente, se recomienda utilizar la aplicación DFTracker mediante el GPS, para poder llegar a obtener una ubicación correcta de la actividad desempeñada.

Además, otra diferencia entre obtener la ubicación por GPS o por *Network*, es que mediante la primera, se obtiene la altitud, mientras que con la segunda no se obtiene ningún dato adicional. Se han de valorar las altitudes obtenidas con GPS, ya que los datos obtenidos difieren con los de la realidad, pero aún así, las altitudes obtenidas pueden llegar a servir para obtener un perfil de la actividad, ya que las diferencias son en todas las coordenadas más o menos iguales.

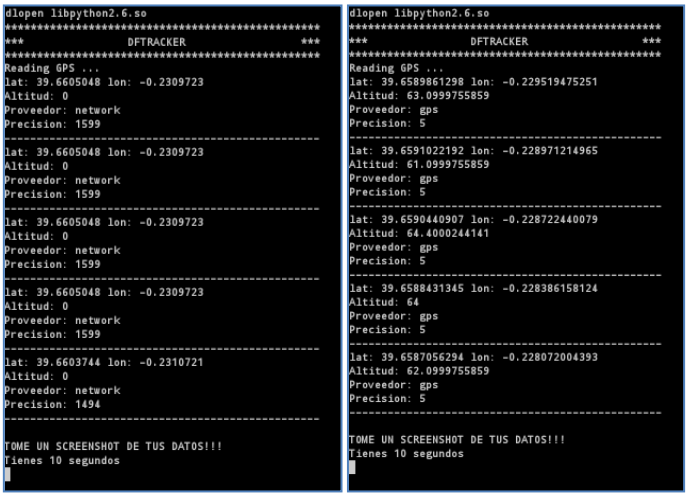

Imagen 15 - Altitudes y precisión GPS

2.2. Processing

Esta segunda aplicación, DFTrail Running, se desarrolla mediante Processing[8], en su versión 2.0.3 fase beta. Hay que destacar que es muy importante trabajar con esta versión, ya que en la 1.6, aún siendo una versión final y teniendo el módulo de Android, a la hora de ejecutar la aplicación, tanto en el móvil, como en el AVD (Android Virtual Device[9]), da error. Por dicha razón, se recomienda trabajar con la versión 2.0.3 beta.

El procedimiento de instalación se describe en el apartado "6.2. Processing" de este documento.

Esta aplicación desarrollada con Processing, se ha diseñado para usuarios que realizan *trail* por montaña, donde pueden orientarse mediante la brújula, ubicarse a través de las coordenadas de su localización y el mapa marcando su posición. DFTrail Running también se puede destinar a otras actividades donde la persona haga uso de las características de la aplicación, como, por ejemplo, ir a recoger setas y orientarse para encontrar el camino de vuelta al punto de partida.

Se ha de recordar que para que el *Smartphone* pueda obtener cualquier localización, se ha de acceder a los servicios de ubicación del teléfono móvil y conectar la localización por la red del operador móvil y GPS.

[8] http://www.processing.org/
[9] Permite emular en un ordenador los dispositivos móviles

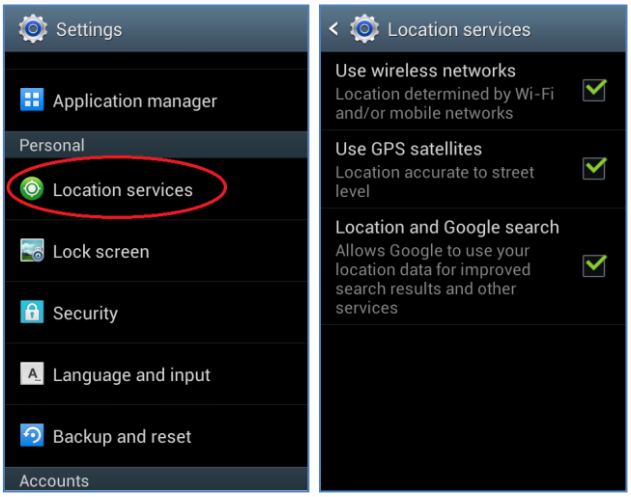

Imagen 16 - Servicios de localización

A continuación, se procede a explicar el funcionamiento de la aplicación. Una vez instalada, se pulsa sobre el icono de la APP y comienza a ejecutarse. Para entender DFTrail Running con claridad, se va a dividir la pantalla principal de la aplicación en tres partes, tal y como se indica en la parte derecha de la imagen 17:

1) Título o encabezado
2) Brújula u orientación
3) Localización y visualización.

Imagen 17 - DFTRAIL RUNNING

En el primer apartado se visualiza la configuración de la pantalla con el nombre de la aplicación. Además de crear el encabezado, se realizan otras tareas, como dibujar los botones de la aplicación y posicionarlos.

El segundo bloque sirve para la orientación en tiempo real, donde se muestra *azimut* (X), *pitch* (y), *roll* (z) y una brújula. Si por algún motivo la aplicación no tiene

señal recibida del sensor del teléfono móvil, aparece un mensaje por pantalla, avisando de que no se ha encontrado el sensor.

Imagen 18 - Título y brújula

Para que siempre se esté actualizando la brújula y la información del sensor en la configuración de la pantalla, sin que se sobrescriba la información, se ha de realizar la tarea de dibujar un cuadrado para borrar la información anterior, y posteriormente al llamar al sensor y a la brújula aparecerán los datos de la nueva información en la configuración de la pantalla. Ésto se realiza siempre de esta manera, es un bucle infinito, siempre dibujando el cuadrado (borrando la información) y dibujando luego la nueva información.

```
void draw() {
	textSize(17);
	dibujaRect();
	fill(255, 255, 255);
	//Sensor
	displaySensorReadings();
	//Brujula
	showCompass();
}
```

El último apartado se compone de cuatro botones y dos cuadrados. Cuando se pulsa el botón de "*Localízame*", el *Smartphone* produce una vibración y obtiene la ubicación del usuario. La información de las coordenadas se sitúa sobre el cuadrado de la derecha. Si no se ha podido recibir la información del GPS o de la red *Network*, se mostrará un mensaje por pantalla, avisando que no se ha podido obtener la ubicación.

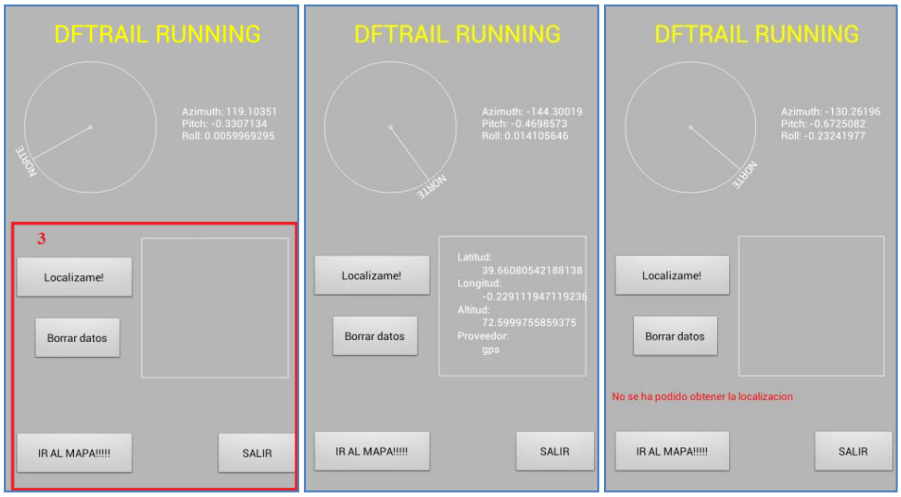

Imagen 19 - Localización

Con el botón de "*Borrar datos*", como indica su nombre, se borran los datos de la localización situados en el cuadrado y además de éste, realiza otro cuadrado para borrar la información de un posible error, en el que no se haya podido obtener la localización. Realmente lo que hace es pintar los dos cuadrados del mismo color que el fondo (gris) para borrar la información, con la instrucción *fill* se le pone el color y con *rect* se realiza el cuadrado.

```
}else if(widget == borrar){
//Para borrar las localizacion
      fill(180, 180, 180);
      rect(220, 380, 240, 230);
      noStroke();
      rect(10, 630, 300, 30);
      stroke(255);
}
```

El tercer botón, "*IR AL MAPA*", muestra desde el navegador del teléfono móvil, la posición donde se encuentra el usuario en un mapa a través de un *Marker*. Desde aquí, se puede acceder al navegador de Google Maps y utilizar todas sus funcionalidades, como por ejemplo ir a otro lugar mediante sus indicaciones o compartir la ubicación del usuario a través de mensajería instantánea.

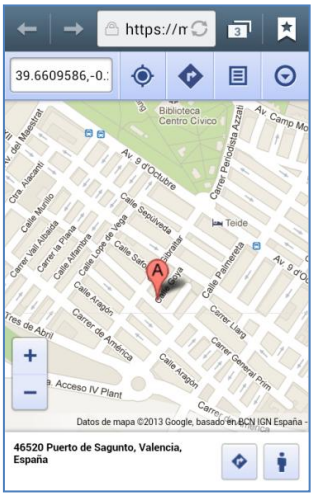

Imagen 20 - Mapa

El último botón, "*SALIR*", sirve para salir de la aplicación.

2.2.1. Permisos teléfono

Para que se puedan realizar una serie de eventos (vibración,...), han de activarse unos permisos, para dar acceso a la aplicación a ciertas funciones del teléfono. Sin ellos, DFTrail Running no podría acceder a la localización, abrir el navegador para mostrar el mapa o vibrar. Para seleccionar los permisos hay que ir a la pestaña de "*Android / Sketch Permissions"*, y se abre una ventana para seleccionar los distintos permisos.

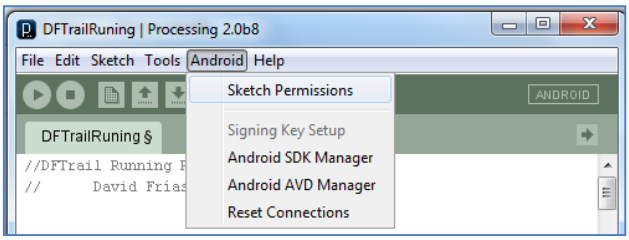

Imagen 21 - Acceder a los permisos

Los permisos seleccionados para DFTrail Running son seis:

- ACCESS_COARSE_LOCATION (Redes móviles, WIFI)
- ACCESS_FINE_LOCATION (GPS)
- ACCESS_LOCATION_EXTRA_COMMANDS (Comandos adicionales del proveedor de ubicación)
- ACCESS_MOCK_LOCATION (Permite crear proveedores de ubicación de prueba)
- INTERNET
- VIBRATE

Los cuatro primeros se han de seleccionar para poder obtener la ubicación, el siguiente permiso para poder abrir el mapa mediante el navegador haciendo petición a internet y el último se ha utilizado para que la aplicación pueda vibrar.

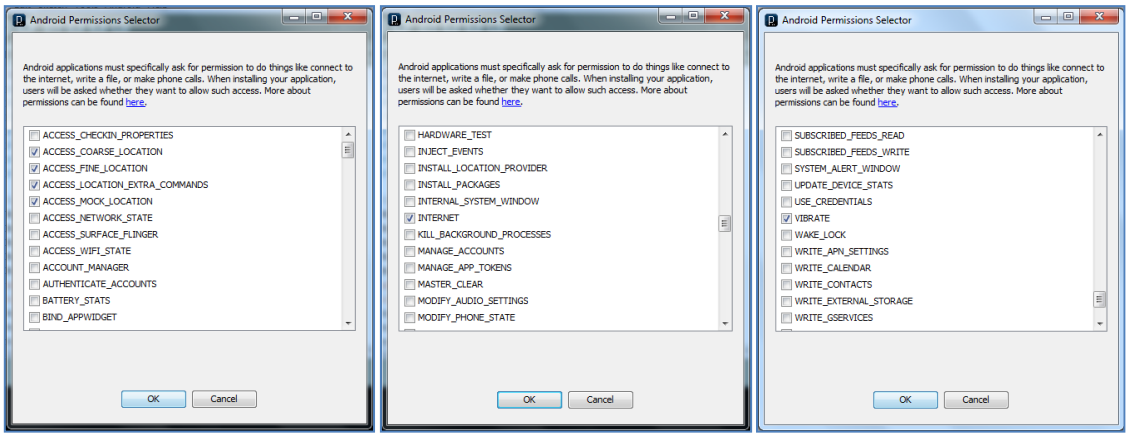

Imagen 22 - Permisos

2.2.2. Librerías

Una característica muy importante de Processing, es la cantidad de información existente a nivel de librerías. Hay de muchos tipos y características, y de fácil acceso, además todas las librerías vienen con ejemplos para simplificar o entender su uso.

Para la creación de la aplicación, se han utilizado dos librerías:

- APWidgets[10]: Para la creación de botones. Se ha usado la versión r3.
- Ketai[11]: Para obtener la localización y los sensores del *Smartphone*. Se ha usado la versión 9, en el momento de la creación de la aplicación, la librería era una versión *beta tester*.

APWidgets

Es una librería, donde no se explican con detalle las clases y los métodos, sólo se plasma en los ejemplos comentando el código de programación.

Para utilizar esta librería, lo primero que se ha de hacer es declarar las variables de cada botón que se vayan a utilizar y del contenedor de los botones.

[10] http://code.google.com/p/apwidgets/
[11] http://code.google.com/p/ketai/

```
//Creacion de variables
APWidgetContainer widgetContainer;
APButton loc;
APButton mapa;
APButton salir;
APButton borrar;
```

A continuación, se crea el contenedor y todos los botones, indicándoles a cada uno de ellos, su tamaño y posición mediante las coordenadas. Después, se añaden al contenedor de botones.

```
widgetContainer = new APWidgetContainer(this); //Se crea el contenedor
widgets
loc = new APButton(10, 410, 200, 75, "Localizame!"); //Boton
Localizacion
mapa = new APButton(10, 700, 200, 75, "IR AL MAPA!!!!!"); //Boton mapa
salir = new APButton(340, 700, 200, 75, "SALIR"); //Boton salir
borrar = new APButton(40, 510, 150, 75, "Borrar datos"); //Boton
borrar
//Se añaden los botones al contenedor
widgetContainer.addWidget(loc);
widgetContainer.addWidget(mapa);
widgetContainer.addWidget(salir);
widgetContainer.addWidget(borrar);
```

Como último paso, se ha de implementar un método, *onClickWidget*, para indicar qué código se debe ejecutar cuando se pulsa uno de estos botones.

```
void onClickWidget(APWidget widget){
//Llama al evento al hacer click en el botón
      if(widget == loc){
      vibe.vibrate(300); //Vibra el teléfono
      obtenerCuaLatLon(); //Obtiene la localizacion
      }else if(widget == mapa){
      link("http://maps.google.com/maps?q=" + latitude + "," +
longitude);
      //Nos saca el mapa centrado donde nos situamos
      }else if(widget == salir){
            exit(); //Para salir del programa
      }else if(widget == borrar){
            //Para borrar las localizacion
            fill(180, 180, 180);
            rect(220, 380, 240, 230);
            noStroke();
            rect(10, 630, 300, 30);
            stroke(255);
      }
}
```

Ketai

A diferencia de la librería anterior, ésta tiene mucha más información para su desarrollo. En su página, se explican todas las clases y métodos implementados dentro de la librería, junto con multitud de ejemplos.

Localización

Para obtener las coordenadas de la localización, es tan sencillo como guardar cada uno de los métodos de la librería, en una variable, lo que indirectamente llama al método para obtener el valor de la variable correspondiente. En el caso del proveedor de las coordenadas se obtiene mediante "*location.getProvider*".

```
void onLocationEvent(double _latitude, double _longitude, double
_altitude){
      //Llama al evento para obtener coordenadas
      longitude = _longitude;
      latitude = _latitude;
      altitude = _altitude;
}
```

```
void obtenerCuaLatLon(){
//Obtiene el proveedor de ubicación
      if (location.getProvider() == "none"){
            fill(255, 0, 0);
            text("No se ha podido obtener la localizacion",  10, 650);
            fill(255);
      }else{
            fill(180, 180, 180);
            fill(255, 255, 255);
            text("Latitud:" + "\n" + "          " + latitude + "\n"
                  + "Longitud:" + "\n" + "          " + longitude +
"\n"
                  + "Altitud:" + "\n" + "          " + altitude +
"\n"
                  + "Proveedor:" + "\n" + "          "  +
location.getProvider(),  250, 420);
      }
}
```

Sensor

Esto se complica un poco más que la localización, ya que se ha de crear una clase para obtener las coordenadas del sensor. Se llama al método así:

```
//Sensor
displaySensorReadings();
```

Este método se utiliza para visualizar las coordenadas en grados y en el caso de que diera un error, mostraría un mensaje avisando.

```
void displaySensorReadings(){
//Lee el sensor
if (sensorAvailable){
            text("Azimuth: " + degrees(azimuth), 290, 180);
            text("Pitch: " + pitch, 290, 200);
            text("Roll: " + roll, 290, 220);
      }else{
            fill(255, 0, 0);
            text("Brujula: No se ha encontrado el sensor", 10, 360);
            fill(255);
      }
}
```

Por último, hay que escribir la clase *SensorEventListener*, donde se obtendrán los valores de las variables del sensor: *azimuth*, *pitch* y *roll*.

```
class mSensorEventListener implements SensorEventListener{
    //Clase sensor
    float[] mGravity;
    float[] mGeomagnetic;
    //Variables de orientacion
    float orientation[] = new float[3];
    //Define todos los metodos del sensor
    public void onSensorChanged(SensorEvent event){
            if (event.accuracy ==
SensorManager.SENSOR_STATUS_ACCURACY_LOW) return;
            switch (event.sensor.getType()){
            case Sensor.TYPE_MAGNETIC_FIELD:
                        mGeomagnetic = event.values.clone();
                        break;
            case Sensor.TYPE_ACCELEROMETER:
                        mGravity = event.values.clone();
                        break;
            }

            if (mGravity != null && mGeomagnetic != null) {
                        float I[] = new float[16];
                        float R[] = new float[16];
                        if (SensorManager.getRotationMatrix(R, I,
mGravity, mGeomagnetic)){
                  //Matriz de rotacion
                        SensorManager.getOrientation(R, orientation);
                        azimuth = orientation[0];
                        pitch   = orientation[1];
                        roll    = orientation[2];
                        }
            }
  }
```

Brújula

Para obtener los datos de la brújula, se llama al método *showCompass*. Desde ese método, se dibuja una circunferencia, dentro de ella se traza la línea que marca el norte, y se especifica con un texto que la flecha indica el norte.

```
//Brujula
showCompass();
```

```
void showCompass(){
      //Brujula
      int cx = 140;
      int cy = 200;
      float radius = 0.9 * 120;

      stroke(255);
      noFill();
      ellipse(cx, cy, radius*2, radius*2);
      if (!firstdraw){
            pushMatrix();
            translate(cx, cy);
            rotate(-azimuth);
            line(0, 0, 0, -radius);
            text("NORTE", -25, -radius-5);
            //Elipse de dentro
            ellipse(0, 0, 5, 5);
            popMatrix();
      }
      firstdraw = false;
}
```

Aquí finaliza la descripción de las dos aplicaciones previas. En el apartado 5 de este documento se realiza una comparativa entre ellas, junto con Eclipse. Se recuerda también que en los apartados 6.1 y 6.2 se describen los procesos de instalación de ambas herramientas de desarrollo.

3. GPX

GPX (GPS eXchange Format) es un formato de poco peso de datos XML[12] para el intercambio de datos GPS (*waypoint*, rutas y *track*) entre las aplicaciones y servicios web en internet.

El formato GPX se está convirtiendo en un estándar entre las marcas de navegadores GPS de mano. Algunas de ellas no crean la ruta en este formato, pero permiten exportarlo a él; en cambio, otras lo crean directamente en este formato. Además, hay varios programas y páginas web que permiten realizar intercambios entre los distintos formatos.

Los beneficios de la utilización de fichero GPX son:

- Permite el intercambio entre programas para Windows, Mac, Linux, Palm y Pocket.
- Se puede convertir en otro formato, a través de conversores.
- Basado en el estándar XML.
- Permite desarrollar nuevas características para utilizar los datos de los receptores GPS

A continuación, se detalla la estructura del formato GPX. Esta información está extraída de la página de http://www.topografix.com/gpx.asp. El fichero GPX es el elemento raíz del archivo XML. Existen dos versiones, la versión 1.0 y la versión 1.1. El esquema general de la versión 1.1 es:

```
<gpx
version="1.1 [1] ?"
creator="xsd:string [1] ?">
<metadata> metadataType </metadata> [0..1] ?
<wpt> wptType </wpt> [0..*] ?
<rte> rteType </rte> [0..*] ?
<trk> trkType </trk> [0..*] ?
<extensions> extensionsType </extensions> [0..1] ?
</gpx>
```

El GPX contiene un encabezado de metadatos, seguido de *waypoints*, rutas y *tracks*. Además se puede agregar información de los propios elementos en la sección de extensiones. Ahora se detallan los distintos tipos de datos de la estructura GPX:

metadataType: Los tipos de metadatos se refieren a la información del archivo GPX, su autor y las restricciones de derechos de autor. Proporciona información significativa sobre sus archivos GPX permitiendo que otros busquen y utilicen los datos GPS. Estructura del *metadaType*:

[12] eXtensible Markup Languaje, lenguaje de marcas para almacenar datos de forma legible

```
<metadata>
<name> xsd:string </name> [0..1] ?
<desc> xsd:string </desc> [0..1] ?
<author> personType </author> [0..1] ?
<copyright> copyrightType </copyright> [0..1] ?
<link> linkType </link> [0..*] ?
<time> xsd:dateTime </time> [0..1] ?
<keywords> xsd:string </keywords> [0..1] ?
<bounds> boundsType </bounds> [0..1] ?
<extensions> extensionsType </extensions> [0..1] ?
</metadata>
```

wptType : Representa un punto de referencia, punto de interés, o una función con nombre en un mapa (wpt). Estructura del *wptType*:

```
<wpt>
lat="latitudeType [1] ?"
lon="longitudeType [1] ?">
<ele> xsd:decimal </ele> [0..1] ?
<time> xsd:dateTime </time> [0..1] ?
<magvar> degreesType </magvar> [0..1] ?
<geoidheight> xsd:decimal </geoidheight> [0..1] ?
<name> xsd:string </name> [0..1] ?
<cmt> xsd:string </cmt> [0..1] ?
<desc> xsd:string </desc> [0..1] ?
<src> xsd:string </src> [0..1] ?
<link> linkType </link> [0..*] ?
<sym> xsd:string </sym> [0..1] ?
<type> xsd:string </type> [0..1] ?
<fix> fixType </fix> [0..1] ?
<sat> xsd:nonNegativeInteger </sat> [0..1] ?
<hdop> xsd:decimal </hdop> [0..1] ?
<vdop> xsd:decimal </vdop> [0..1] ?
<pdop> xsd:decimal </pdop> [0..1] ?
<ageofdgpsdata> xsd:decimal </ageofdgpsdata> [0..1] ?
<dgpsid> dgpsStationType </dgpsid> [0..1] ?
<extensions> extensionsType </extensions> [0..1] ?
</wpt>
```

rteType: Representa una ruta, una lista ordenada de puntos de interés que representan una serie de puntos de giro que llevan a un destino (rte). Estructura del *rteType*:

```
<rte>
<name> xsd:string </name> [0..1] ?
<cmt> xsd:string </cmt> [0..1] ?
<desc> xsd:string </desc> [0..1] ?
<src> xsd:string </src> [0..1] ?
<link> linkType </link> [0..*] ?
<number> xsd:nonNegativeInteger </number> [0..1] ?
<type> xsd:string </type> [0..1] ?
<extensions> extensionsType </extensions> [0..1] ?
<rtept> wptType </rtept> [0..*] ?
</rte>
```

trkType: Representa una pista, una lista ordenada de puntos que describen una trayectoria (trk). Estructura del *trkType*:

```
<trk>
<name> xsd:string </name> [0..1] ?
<cmt> xsd:string </cmt> [0..1] ?
<desc> xsd:string </desc> [0..1] ?
<src> xsd:string </src> [0..1] ?
<link> linkType </link> [0..*] ?
<number> xsd:nonNegativeInteger </number> [0..1] ?
<type> xsd:string </type> [0..1] ?
<extensions> extensionsType </extensions> [0..1] ?
<trkseg> trksegType </trkseg> [0..*] ?
</trk>
```

trksegType: Un segmento de pista tiene una lista de puntos de seguimiento que están conectados lógicamente en orden. Representar un único *track* GPS, si la recepción GPS se ha perdido, o el receptor GPS se ha apagado, se ha de iniciar un nuevo segmento de pista para cada tramo continuo de datos. Estructura del *trksegType*:

```
<trkseg>
<trkpt> wptType </trkpt> [0..*] ?
<extensions> extensionsType </extensions> [0..1] ?
</trkseg >
```

La diferencia entre el rte y el trk es que el rte es el camino a seguir en línea recta desde un *waypoint* a otro, y el trk es la sucesión de puntos de *track*, representan el camino que hay entre el punto de partida y el de llegada. Normalmente estos dos conceptos se suelen confundir y al *track* se le suele denominar ruta.

extensionsType: Puede ampliar el archivo GPX agregando sus propios elementos de otro esquema. Estructura de *extensionsType*:

```
<extensions>
Permitir todos los elementos de un espacio de nombres que no sean
espacio de nombres de este esquema de validación (LAX validation). [0
.. *]
</extensions>
```

latitudeType: La latitud del punto. Grados decimales, datum WGS84.

longitudeType: La longitud del punto. Grados decimales, datum WGS84.

4. Aplicación objeto

Después de realizar las dos experimentaciones previas, mediante SL4A y Processing, se adquieren unos conocimientos mínimos para abordar la aplicación objeto de este trabajo y enfrentarse a ella. La aplicación objeto se denomina TraLoc y se ha implementado con el lenguaje de programación Java. Para ello, se ha utilizado el entorno de desarrollo Eclipse, en su versión 4.3.1 (Kepler).

El procedimiento de instalación se describe en el apartado "6.3. Eclipse" de este documento.

Esta aplicación está diseñada para cartografiar la ruta de una actividad, cargar en ella otras rutas (creadas con otros dispositivos o usuarios), visualizar las rutas ya existentes en TraLoc. Además tiene la capacidad de enviar los ficheros GPX a través de correo electrónico, para compartir las rutas con otros usuarios, y existe la opción de borrar una ruta que no se quiera tener o sea errónea.

La actividad a la que está destinada TraLoc para cartografiar la ruta es la práctica de senderismo. Pero también se puede utilizar para otras actividades, como obtener la ruta en un *trail*, en una carrera o en un desplazamiento con vehículo; en definitiva, cualquier actividad donde se localice al usuario en su desarrollo y posteriormente, a su finalización, se desee ver el resultado a través del mapa, mostrándose el *track* realizado y los puntos de interés existentes por el camino.

TraLoc se ha diseñado para crear las rutas en ficheros GPX, en versión 1.1, pero la aplicación, en su opción de visualizar las rutas, utiliza los archivos GPX en versión 1.0 y 1.1 y a su vez, se envían ambas versiones por correo electrónico.

TraLoc al poder utilizar indistintamente ambas versiones del fichero GPX, tiene la compatibilidad de poder descargarse las rutas de cualquier página web, por ejemplo:

- Senderos de Valencia (http://www.senderosvalencianos.es/),
- WikiLoc (http://es.wikiloc.com/wikiloc/home.do) ,
- Federación Aragonesa de montañismo (http://www.fam.es/web/),
- Senderos del Sistema Central (http://www.senderosdelsistemacentral.com/)
- …

Además, se pueden cargar rutas creadas con GPS de mano y utilizar la ruta creada con TraLoc en los dispositivos GPS de mano. A modo de ejemplo, se ha utilizado el GPS Garmin Etrex 20, para realizar pruebas de compatibilidad de archivos GPX.

TraLoc se ha diseñado para ejecutarse en dispositivos móviles con Android y ha sido desarrollada directamente a través del *Smartphone* Samsung Galaxy SII, sin

utilizarse[13] el emulador AVD (Android Virtual Devices). La versión de Android del *Smartphone* es la 4.1.2 (Jelly Bean), pero la aplicación se ha implementado para una versión mínima de 4.0 (Ice Cream Sandwich, API 14) hasta la versión actual 4.4 (KitKat, API 19).

Con lo expuesto y según un estudio recientemente publicado por Android[14], teniendo en cuenta el número relativo de los dispositivos que ejecutan una versión determinada de la plataforma de Android (toda la información es recogida por la tienda Google Play Store), TraLoc podría llegar a captar el 77.4% de los dispositivos Android, por lo que se puede decir que la aplicación podría abarcar un número importante de usuarios.

Version	Codename	API	Distribution
2.2	Froyo	8	1.3%
2.3.3 - 2.3.7	Gingerbread	10	21.2%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	16.9%
4.1.x	Jelly Bean	16	35.9%
4.2.x		17	15.4%
4.3		18	7.8%
4.4	KitKat	19	1.4%

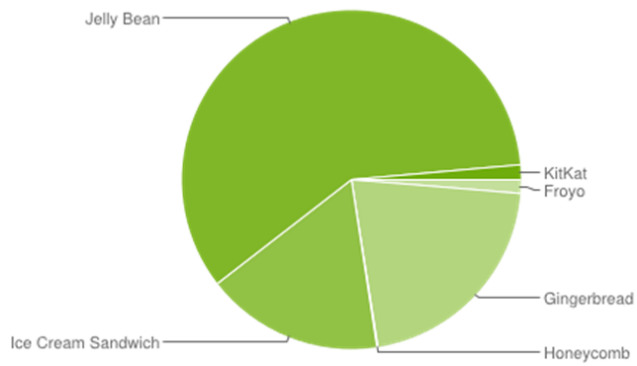

Tabla 3 - Datos correspondientes a la primera semana del año (2014) - Distribución de versiones del sistema operativo

4.1. Estructura del proyecto

En Eclipse, todo proyecto de Android, se subdivide en diferentes carpetas, librerías y ficheros, que componen la estructura de la aplicación. Ahora se detallan las diferentes partes de las que se compone la estructura del proyecto.

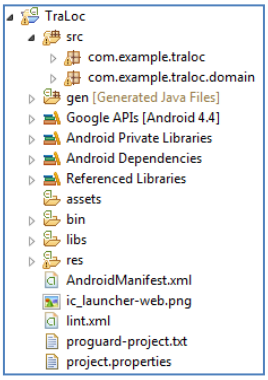

Imagen 23 - Estructura proyecto Eclipse

[13] Más adelante se explicará
[14] http://developer.android.com/about/dashboards/index.html

En primer lugar, se encuentra la carpeta "*src*", en la que se localiza el código fuente de la aplicación. Los ficheros Java se almacenan en un espacio de nombres; éstos, a su vez, se encuentran almacenados en paquetes que contienen clases que desempeñan tareas similares. Todo esto sirve para estructurar de forma ordenada el código de la aplicación. Los paquetes que se han utilizado en TraLoc son dos:

- com.example.traloc: Contiene el código de las actividades que conforma la aplicación
- com.example.traloc.domain: Contiene las diferentes clases implementadas por el autor de la aplicación. Dentro de ella, se han utilizado dos clases: *LocationTrack*, para almacenar la información referida a la ruta (coordenadas, distancia, elevación, nombre y subnombre) y *Route*, donde se implementan métodos que se utilizan en las diferentes actividades para la ruta.

Seguidamente, la carpeta "*gen*" contiene el código generado de forma automática por la aplicación. Nunca se han de modificar de forma manual estos archivos.

Ahora se encuentran una serie de paquetes de librerías. La primera "*Google APIs [Android 4.4]*" es la versión de Android para la que se va a desarrollar la aplicación. Después, "*Android Private Libraries*" son librerías privadas externas, que se usan de forma privada por otras librerías de la aplicación. "*Android Dependencies*" son las librerías dependientes o subordinadas, contienen las librerías necesarias para el funcionamiento de la aplicación. Por último, "*Referenced Libraries*" son las librerías referenciadas (en el apartado 4.4 de este documento se detalla cómo se ha referenciado la librería utilizada).

La siguiente carpeta, "*assets*", contiene una serie de ficheros de datos, ficheros Jar externos, fuentes… utilizados por la aplicación; nunca se ha de modificar el contenido de los ficheros de esta carpeta.

La carpeta "*bin*" es la carpeta en la que se compila el código, y se genera el archivo .apk (archivo que permite instalar la aplicación en el dispositivo móvil Android).

La última carpeta, "*res*", contiene los recursos usados por la aplicación. Se compone de una serie de subcarpetas, que a su vez, contienen diferentes ficheros. Ahora se explican las diferentes subcarpetas:

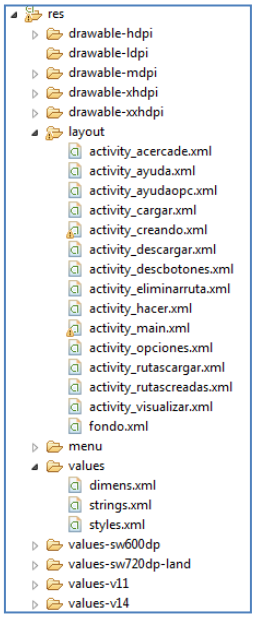

Imagen 24 - Carpeta res

- *drawable*: En esta carpeta, se almacenan los ficheros de las imágenes e iconos utilizados en la aplicación. Android escoge el tamaño de la imagen e icono adecuado al dispositivo, adaptándolo a las diferentes resoluciones de pantalla, por lo que es muy importante poner los diferentes tipos de tamaño de los iconos e imágenes, en sus respectivas carpetas *drawable*.
- *layout*: Contiene ficheros XML con el diseño de las vistas de cada una de las actividades, son la interfaz de usuario. Por lo tanto, deben existir tantos *layouts* como actividades existan.
- *menú*: Contiene ficheros XML con los menús de la aplicación. Son los menús que se visualizarán cuando se pulse el botón físico menú o sobre los botones que se visualicen en la *ActionBar*[15].
- *values*: Contiene ficheros XML donde se indica el valor del tipo *string*, color o estilo.

El fichero "*AndroidManifest.xml*" describe la aplicación Android. En él se indican las actividades, *intents*, servicios y proveedores de contenido de la aplicación. También se declaran los permisos que se requerirán en ella, se indica la versión mínima y máxima de Android para poder ejecutarla. Cada vez que se crea una actividad o se requiere un permiso se ha de declarar en este fichero o se lanzará un error.

El siguiente fichero es una imagen, "*ic_laucher-web.png*", que es el icono de la aplicación. Esta imagen es de gran tamaño y se utiliza en páginas web.

El archivo "*proguard-project.txt*" es un fichero de configuración de la herramienta ProGuard. Permite optimizar y ofuscar el código generado, es decir, obtiene un código .apk más pequeño donde resulta más difícil hacer ingeniería inversa.

[15] Es la barra de título y herramientas que aparece en la parte superior de las aplicaciones

El último fichero, "*project.propierties*", es generado automáticamente por el SDK; nunca se ha de modificar, ya que se utiliza para comprobar la versión del API y otras características cuando se instala la aplicación en el terminal.

4.2. Diseño de la aplicación

Como se ha descrito anteriormente, el diseño de la aplicación se realiza a través de XML. El diseño de la interfaz de usuario se puede crear mediante código Java, pero resulta más conveniente realizarla en XML, ya que este lenguaje por etiquetas permite un diseño limpio y eficiente. Al diseño de cada una de las actividades se le conoce como *layout*, que puede contener una o varias vistas.

TraLoc está formado por una serie de actividades interrelacionadas entre sí. A continuación, se muestra un esquema de las posibilidades de navegación existentes, a través de la aplicación. A veces el acceso a algunas actividades, se puede realizar de varias formas.

Las flechas verdes, indican el avance en la aplicación, y los círculos rojos, el retroceso de las actividades.

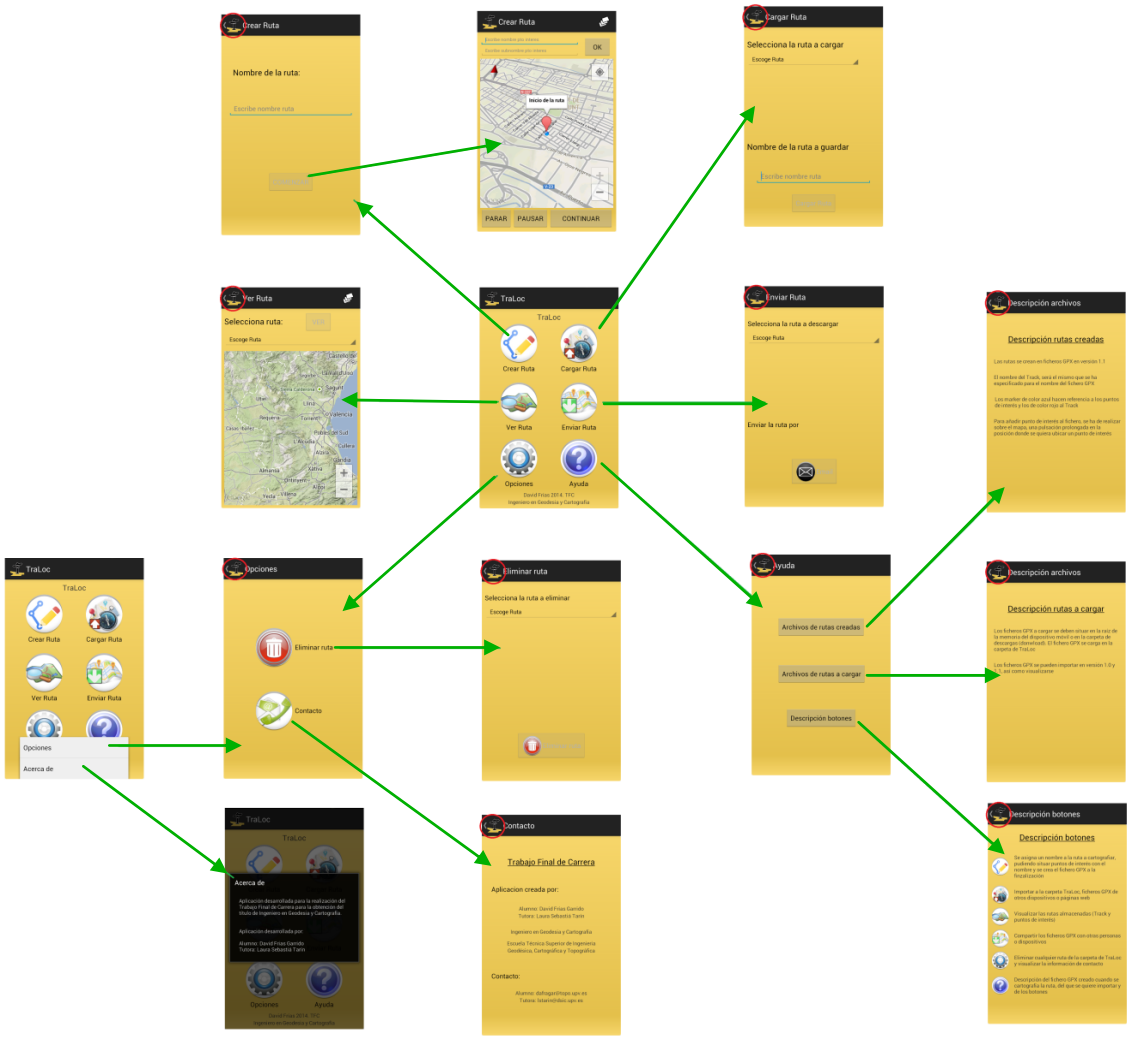

Imagen 25 - Esquema navegación entre *layouts*

4.3. Layout (Interfaz de usuario)

Se describirá por partes el diseño de cada *layout* junto con la actividad que desempeña.

En primer lugar, se ha de pulsar sobre el icono de la aplicación. A partir de este momento, se ejecuta TraLoc.

Imagen 26 - Icono TraLoc

4.3.1. Main

La primera actividad es el *Main*. Es un *layout* que resulta muy sencillo y visual, para que el usuario acceda a las diferentes secciones de la aplicación sin problemas, ya que la imagen del icono, como el texto que lo compone, describe la tarea que desempeña cada botón.

Imagen 27 - *Main*

Esta interfaz se compone de 6 botones. A través de ellos, se puede acceder a otras actividades. Dependiendo del dispositivo móvil y de la versión de Android que se tenga instalada, el *Main* variará. La diferencia es la existencia o no, de un icono formado por tres puntos verticales en la *ActionBar*. Si este icono no aparece, dicho submenú se puede obtener pulsando sobre el botón Menú (botón físico del *Smartphone*) y se mostrará en la parte de debajo de la pantalla. Si, por el contrario, aparece el icono, al pulsar sobre él o sobre el botón físico menú, aparecerá el submenú debajo del icono. Esta posibilidad de ejecución del submenú, se da en todas las *layouts* de la aplicación.

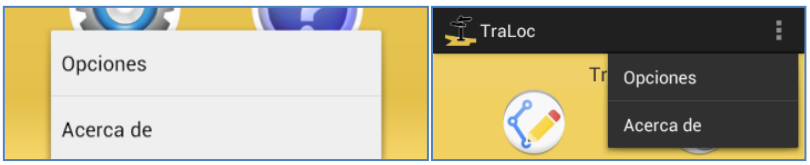

Imagen 28 - Submenú

4.3.2. Crear Ruta

Pulsando sobre el primer icono del *Main*, "*Crear Ruta*", se accede a esta *layout*. Con esta opción del menú se pretende crear un fichero GPX con la nueva ruta y tomando los puntos de interés. Concretamente, se le da un nombre y permite visualizar el cartografiado de la ruta.

La primera acción que realiza esta actividad, es comprobar si se encuentran conectados los servicios de ubicación (red móvil y GPS). Si alguna de las dos opciones se encuentra desactivada, aparece un mensaje, avisando al usuario de que se han de configurar estos servicios y le ofrece la posibilidad de llevarle a los servicios de ubicación del dispositivo móvil, haciendo clic sobre el botón "*Configuración*" o la opción de darle al otro botón, para salir.

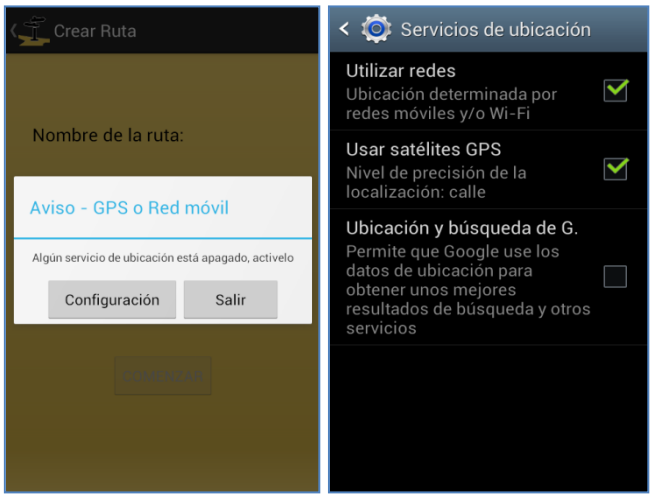

Imagen 29 - Servicios de ubicación

Una vez seleccionados los servicios de ubicación, se observa la interfaz sin ningún elemento sobrepuesto. Ahora, se comienza a describir esta *layout* desde arriba. En la *ActionBar*, en la parte izquierda, aparece un nuevo botón; pulsando sobre él, se vuelve a la actividad anterior, es decir, en este caso, se volvería al *Main*. Este botón aparecerá en todas las actividades siguientes, a excepción de cuando se esté cartografiando la ruta.

Imagen 30 - *ActionBar* Crear Ruta

Como ya se ha indicado, cuando comienza la ejecución de este *layout*, aparece un texto, indicándole al usuario que introduzca el nombre de la ruta. En el cuadro de texto (*EditText*) "*Escribe nombre ruta*", se pone el nombre que se dará a la ruta. A través de este *EditText*, se controla que exista un nombre de ruta y que ese nombre no esté utilizado en otra ruta guardada en la aplicación. Si estas dos premisas se cumplen, el botón "*COMENZAR*" se habilita para hacer clic.

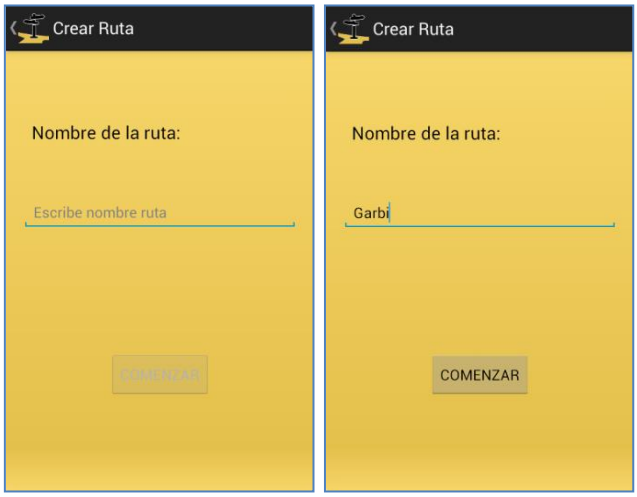

Imagen 31 - Introducción del texto para la ruta

Si el usuario escribe un nombre de ruta ya utilizado, TraLoc avisa que ya existe ese nombre, le permite que introduzca otro y no habilitará el botón "*COMENZAR*".

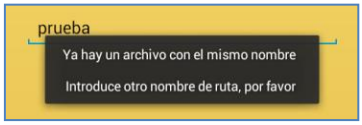

Imagen 32 - Nombre de la ruta repetido

Además de utilizar la escritura por teclado, se puede hacer uso de la entrada de texto mediante la voz, en toda la aplicación.

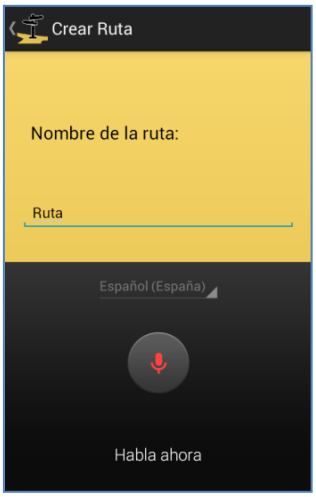

Imagen 33 - Entrada de texto por voz

Al pulsar el botón "*COMENZAR*", se pasa a otra actividad enviándole el nombre que el usuario ha puesto para la ruta.

En un primer momento, aparece un indicador de progreso (*ProgressDialog*), avisando al usuario de que se está obteniendo la ubicación (en el apartado 4.4.1 de este documento, se entrará en detalle sobre la localización), cuando se realiza la tarea del *ProgressDialog*, se obtiene la ubicación del usuario, y este se cierra.

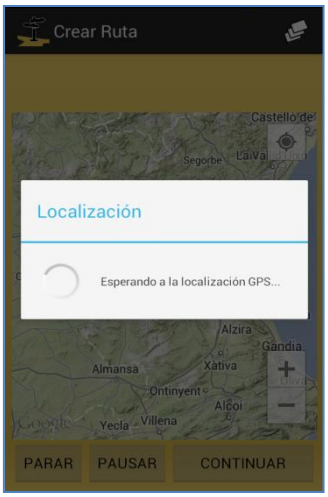

Imagen 34 - *ProgressDialog* GPS

Ahora se observa que el *layout* está compuesto por la *ActionBar*, un espacio vacío (más adelante se ve como es una opción oculta), un *fragment* y una línea compuesta por tres botones. El *fragment* es un tipo de vista que contiene grandes

cantidades de información, como mapas o páginas web; en este caso, se utiliza para alojar un mapa de localización.

Una vez cerrado el *ProgressDialog*, TraLoc mueve y orienta el mapa (tipo terreno) hasta donde está localizado el usuario, realizándose un zoom y añadiendo un marcador (*Marker*) a su posición, de color rojo. Los *Marker* de color rojo, indicarán el inicio y el final del *track*; si se presiona sobre ellos aparece un mensaje indicándolo.

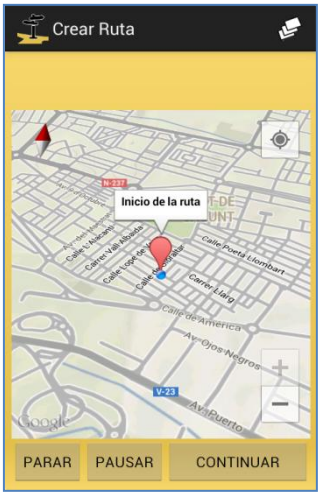

Imagen 35 - *Marker* del inicio de la ruta

A partir de este momento, a medida que el usuario realice su actividad, se irá dibujando la trayectoria mediante líneas en el mapa. La cámara irá cambiándose, siguiendo al usuario en la actividad. Al ser un mapa dinámico se pueden realizar acciones[16] sobre él: zoom, moverlo, rotarlo e inclinarlo (visualizarlo en 3D), pero en el momento que se capture otra posición la cámara vuelve a su configuración inicial.

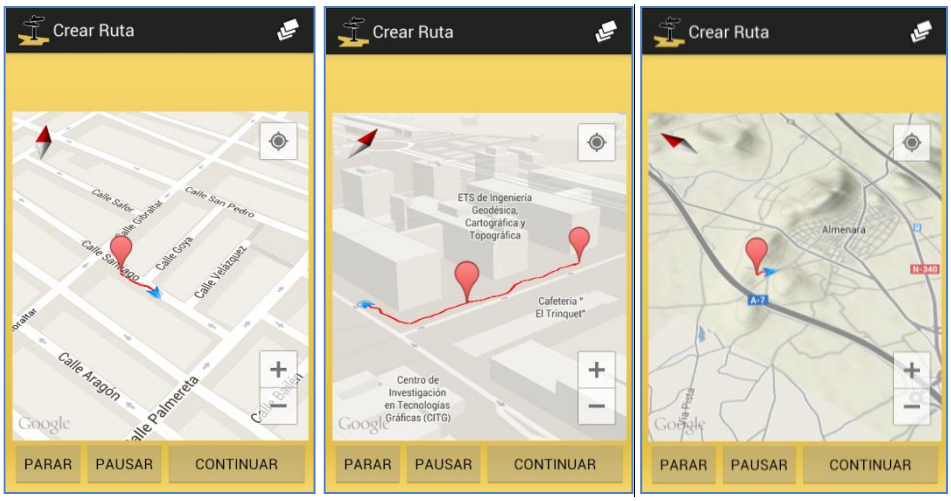

Imagen 36 - Creando una ruta

[16] En el apartado 4.3.4 se explicarán

Para marcar un punto de interés (wpt) sobre el mapa, hay que realizar una pulsación larga en el punto donde se quiere ubicar. En ese momento, en el espacio vacío de la parte superior de la interfaz, aparecen unas opciones, y en la parte inferior se muestra un mensaje, indicando que se escriba el nombre del punto de interés.

En las opciones, se puede poner un título y un subnombre, y pulsando sobre el botón "*OK*", quedará añadido al mapa. Aunque no se haya introducido ningún título o subnombre al wpt, pulsando "*OK*", el punto quedará reflejado en el mapa, aunque no tendrá descripción.

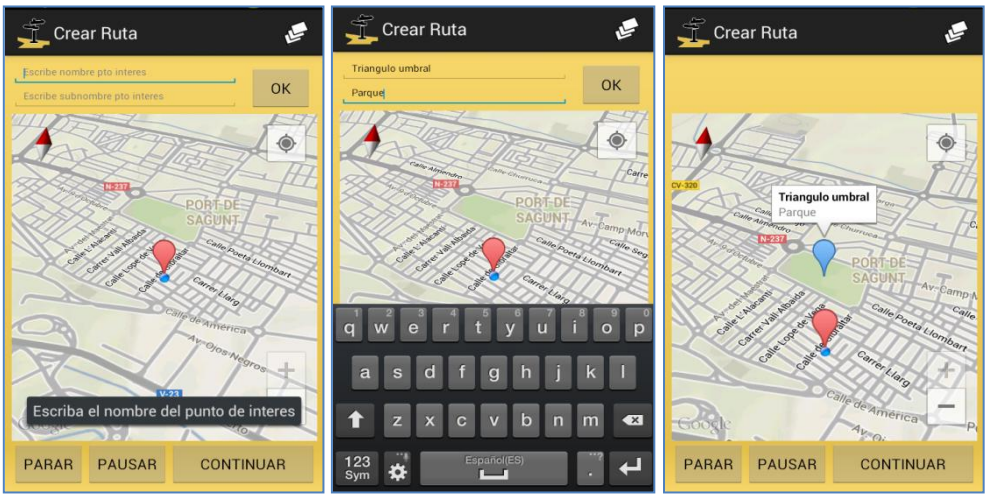

Imagen 37 - Introducir un punto de interés

En el momento de pulsar sobre el botón "*OK*", las opciones se vuelven a ocultar y se añade al *fragment* un *Marker*. Si se presiona sobre él, aparece la información escrita anteriormente. Todos los puntos de interés añadidos al mapa, se colocarán con marcador de color azul, así se diferenciará con un golpe de vista, respecto del *track* que se está realizando.

En la parte inferior hay tres botones:

- *PARAR*, para la toma de datos y guarda la ruta realizada.
- *PAUSAR*, pausa el cartografiado.
- *CONTINUAR*, reanuda la captura de datos.

Presionando sobre cualquiera de los tres, aparece un mensaje avisando de la acción que se ha solicitado.

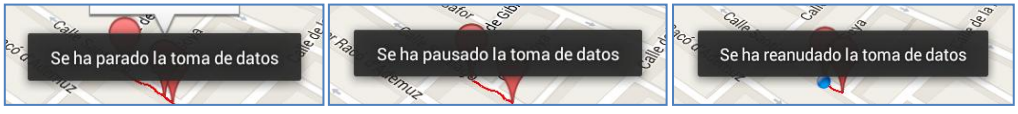

Imagen 38 - Mensajes al pulsar los botones (PARAR, PAUSAR, CONTINUAR)

Cuando se pulsa sobre el botón "*PARAR*", se añade al final del *track* un *Marker* indicando el último punto de posición y se guarda la ruta en un archivo GPX versión 1.1. A los tres segundos de pulsar el botón, la actividad se cierra y retorna al *Main*.

Al pulsar el botón "*PAUSAR*", se añade un nuevo *Marker* al mapa que, si se presiona, informa que es un punto intermedio.

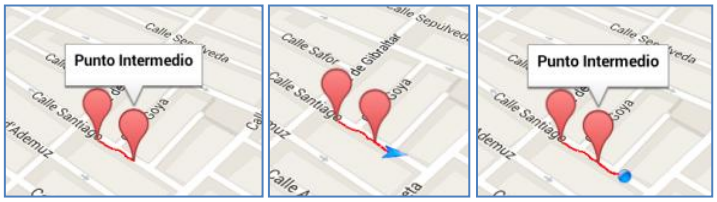

Imagen 39 - *Marker* del punto intermedio

Pulsando el botón "*CONTINUAR*", se reanuda la toma de datos y se sigue cartografiando la ruta desde el punto en que se pausó.

En la *ActionBar* de esta actividad, se ha eliminado el botón de la izquierda, para volver a la actividad anterior, y que no se pueda parar el cartografiado de la ruta. En la parte derecha de la barra aparece un icono, con tres mapas superpuestos. Si se pulsa, surge un submenú con cuatro opciones a seleccionar, que permite cambiar los tipos de mapas a visualizar (en el apartado 4.3.4 se explicará con más detalle).

Imagen 40 - Tipos de mapas

Otra de las medidas que se han tomado para no interrumpir el cartografiado, es la eliminación de las acciones de los botones físicos (menú y atrás). En esta actividad, si se pulsa alguno de ellos, no se realiza ninguna acción. Como se dijo antes, en algunos dispositivos o versiones de Android, aparecía un icono con tres puntos verticales, en el que surgía un submenú con las mismas opciones que pulsando menú; este icono de la *ActionBar* también se ha eliminado.

Una funcionalidad que se ha incorporado es que mientras la toma de la ruta está en pausa, el usuario puede salir de TraLoc, dejándola en segundo plano y posteriormente volver a ella. Pero hasta que no se vuelva a dar al botón "*CONTINUAR*", la toma de datos no continúa.

Otra característica de esta actividad es que nunca se apagará la pantalla del dispositivo móvil.

4.3.3. Cargar Ruta

Esta actividad está pensada para introducir archivos de rutas GPX en versión 1.0 y 1.1 a la aplicación para poder visualizarlas. Se pueden haber creado ficheros con diferentes dispositivos o incluso descargadas de internet.

Pulsando sobre el segundo icono del *Main*, "*Cargar Ruta*", se accede a esta interfaz, compuesta por cinco elementos más la *ActionBar*.

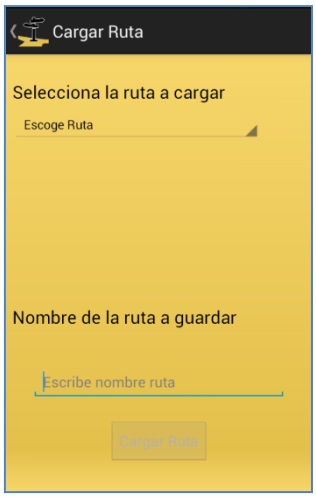

Imagen 41 - Cargar Ruta

En primer lugar, en la parte superior aparece un texto indicando que se ha de seleccionar la ruta que se quiere cargar en TraLoc. Debajo, si se pulsa sobre "*Escoge Ruta*", aparece una lista (*Spinner*), donde se selecciona la ruta que se quiere cargar. Estos archivos están alojados en la raíz de la memoria interna del dispositivo móvil y en la carpeta de descargas (*download*).

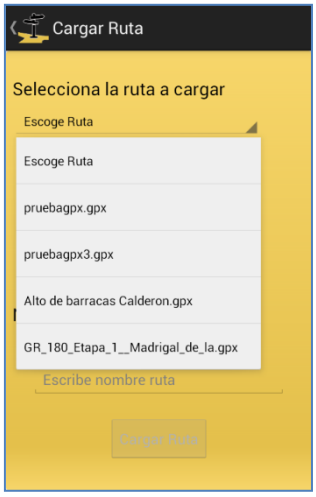

Imagen 42 - Escoger una ruta para importarla

Abajo, se indica al usuario que escriba el nombre que quiere dar a la ruta que va a importar y en el *EditText* se introduce el nuevo nombre.

El último elemento es el botón de "*Cargar Ruta*". Se encuentra deshabilitado, hasta que no se cumplan unas condiciones. Para que quede habilitado, en la lista *Spinner* se ha de seleccionar una ruta (excepto el primer elemento "*Escoge Ruta*") y el nuevo nombre de la ruta a importar ha de ser, como mínimo, de un carácter.

Si este nuevo nombre ya existe en otro fichero (en la carpeta TraLoc), aparece un mensaje avisando al usuario que cambie el texto y no habilita el botón "*Cargar Ruta*".

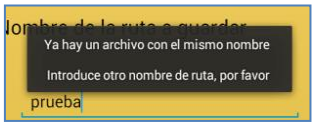

Imagen 43 - El nuevo nombre de la ruta a cargar existe

En el momento que se pulsa sobre el botón, el archivo GPX se copia dentro de la carpeta de la aplicación TraLoc, con el nuevo nombre. Se avisa al usuario de qué se ha realizado la carga correctamente y además el dispositivo móvil vibra. Una vez cargado el fichero, al mantenerse en la misma actividad, aparece un mensaje que el nombre de la ruta ya existe, porque se acaba de importar, por lo que el botón "*Cargar Ruta*" se deshabilita.

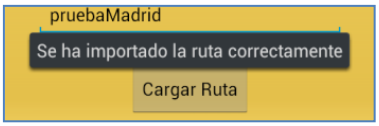

Imagen 44 - Mensaje que se ha importado correctamente la ruta

4.3.4. Visualizar Ruta

Pulsando sobre el tercer botón del *Main*, o primer icono de la segunda fila, "*Visualizar Ruta*", se entra a una interfaz compuesta por cuatro elementos más la *ActionBar*, cuya finalidad es poder visualizar cualquier ruta que tenga TraLoc.

Aparece un texto para indicar que se seleccione la ruta; al elegirla en la lista *Spinner*, el botón "*VER*" se habilita, pero si se opta por el primer elemento ("*Escoge* Ruta"), el botón "*VER*" se deshabilitará. Se pulsa sobre el botón y el mapa se mueve para ubicarse en la zona de la ruta, visualizando todos sus *Marker* (recordar que los *Marker* en color rojo son el *track* y en azul los puntos de interés).

Imagen 45 - Ver Ruta

Si alguna de las rutas que se quiere visualizar no tiene ningún punto de interés, se avisa a través de un mensaje por pantalla, y el mapa se centrará a través del *track*, en lugar de en los puntos de interés.

Imagen 46 - Mensaje que la ruta no tiene puntos de interés

Como en la opción de "*Crear Ruta*", si se selecciona algún *Marker*, aparece un bocadillo con la información que representa y el mapa centra la imagen al marcador.

Imagen 47 - Bocadillos de los *Marker*

El mapa que se visualiza es dinámico, con lo que se puede interactuar con él:

- Desplazar: Deslizar por la pantalla el dedo arrastrando.
- Zoom:
  - Doble toque sobre la pantalla, para aumentar (acercar x1).
  - Dos dedos sobre la pantalla, manteniendo siempre uno apoyado y el otro realiza doble toque (alejar x1).
  - Dos dedos sobre la pantalla y estirarlos (acercar) o encogerlos (reducir).
  - Se pulsa dos veces la pantalla pero la segunda vez no se levanta el dedo, sino que se mantiene apretando, deslizando hacia arriba (acercar) o hacia abajo (alejar).
  - Con los dos iconos del mapa + (acercar) y – (alejar).
- Giro: Pulsando con dos dedos sobre la pantalla, uno se queda fijo (eje) y el otro rota sobre él.
- Inclinación: Pulsando con dos dedos la pantalla, se realiza un movimiento simultáneo hacia arriba o abajo, para aumentar o reducir el ángulo de inclinación (visualización en 3D).

Imagen 48 - Visualización en 3D

En la *ActionBar*, aparte del icono de volver a la actividad anterior, se ha añadido el icono donde se encuentran tres mapas superpuestos para mostrar los distintos tipos de mapas. Este es el mismo icono que en la *ActionBar* de la opción de crear la ruta.

Imagen 49 - Tipos de mapa

Pulsando sobre el icono, se despliega un submenú en el que aparecen los cuatro tipos de mapas. Seleccionando cualquiera de ellos, se visualizan en el *fragment* mediante imágenes de satélite, vista de terreno, de forma hibrida y vectorial o normal. Así dependiendo de la actividad que se esté realizando o visualizando, se puede utilizar el mapa que más interese a las características del momento.

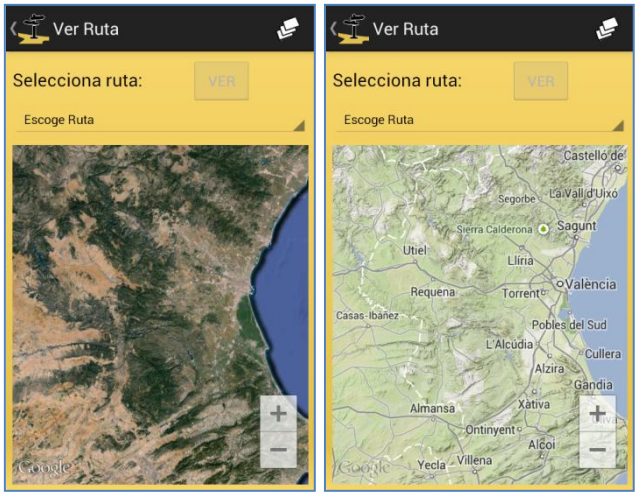

Imagen 50 - Satélite y Terreno

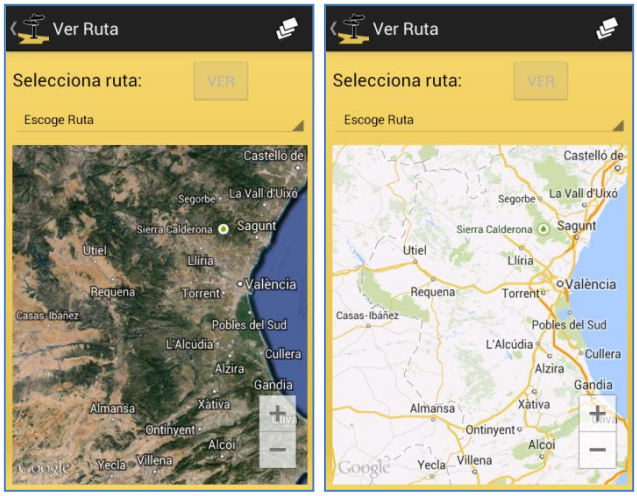

Imagen 51 - Híbrido y Normal

4.3.5. Enviar Ruta

Pulsando en el *Main* sobre el botón "*Enviar Ruta*", aparece esta *layout*, cuya finalidad es compartir las rutas con otros usuarios, enviándolas por correo electrónico.

Se indica a través de un texto "*Selecciona la ruta a descargar*", se elige de la lista *Spinner* y una vez seleccionada, se habilita el botón "*Email*" para poder pulsarlo, siempre y cuando no se haya elegido el primer elemento "*Escoge Ruta*".

Imagen 52 - Enviar Ruta

Dependiendo de la configuración del dispositivo móvil, puede salir un menú contextual para pedirle al usuario que elija la aplicación con la que quiere enviar el email, o la asigna automáticamente.

En el caso de la imagen 53, se le pide al usuario con qué aplicación quiere realizar el envió de la ruta. Si se selecciona por ejemplo, la opción "*Correo electrónico*" se lanza a la actividad del sistema de enviar correo electrónico; si se hubiera pulsado el otro botón, "Gmail", se lanzaría esta aplicación para enviarlo.

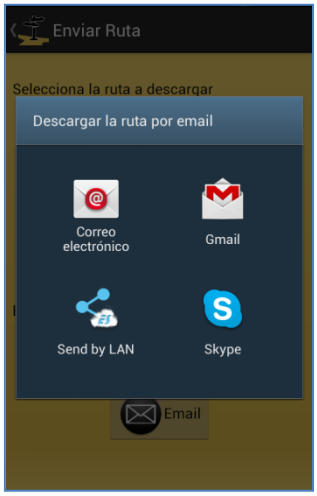

Imagen 53 - Elección de la aplicación para enviar el email

En la aplicación para enviar email se adjunta automáticamente la ruta que se seleccionó. Este email siempre lleva como asunto "TraLoc – Descarga de ruta" y un mensaje de texto, firmado por la aplicación. Lo único que se tendría que especificar sería el correo electrónico del destinatario. Una vez se ha enviado, la actividad retorna a TraLoc, al *layout* de "*Enviar Ruta*".

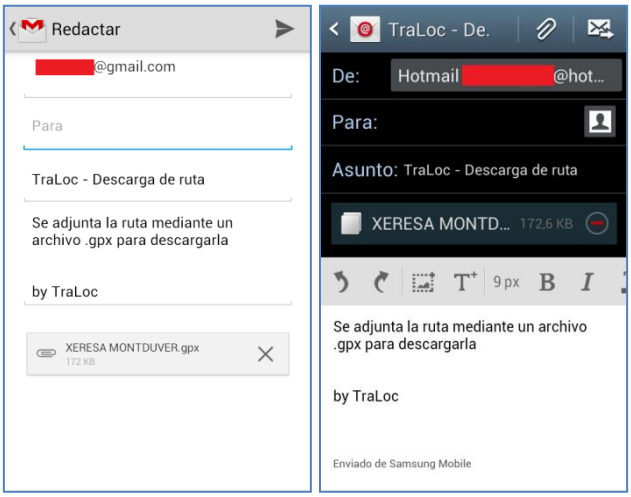

Imagen 54 - Gmail y Hotmail

4.3.6. Opciones

Si se presiona sobre el penúltimo botón del *Main*, "*Opciones*", aparece una interfaz compuesta por dos botones, con imagen y texto, que describen la acción de cada uno. Si se opta por el superior, se puede eliminar cualquier ruta que este dentro de TraLoc, y si se elige el inferior, muestra al usuario la información de contacto.

Imagen 55 - Opciones

Además de poder acceder a esta *layout* a través del *Main*, se puede acceder en cualquier interfaz si se presiona sobre el botón físico menú (o el icono con tres puntos verticales, en la *ActionBar*) y dentro del submenú se selecciona "*Opciones*".

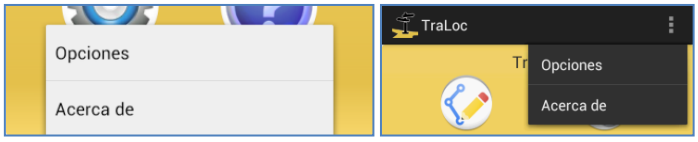

Imagen 56 - Submenú

Seleccionando el botón "*Eliminar Ruta*", aparece otro *layout* indicando que se seleccione la ruta que se quiere eliminar. Se elige la ruta en la lista *Spinner* y si no se ha optado por "*Escoge Ruta*", el botón "*Eliminar Ruta*" se activa.

Al presionar el botón "*Eliminar Ruta*", el dispositivo móvil realiza una vibración y muestra un mensaje, para avisar de que se ha eliminado. Posteriormente vuelve al *Main*.

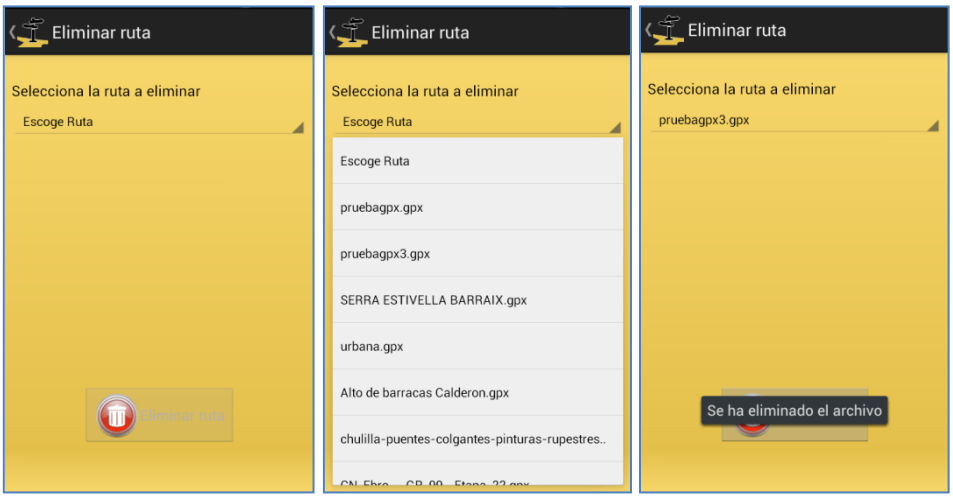

Imagen 57 - Eliminar Ruta

Por el contrario, si se selecciona el botón "*Contacto*", aparece un *layout* donde se describe el autor de la aplicación, su finalidad, y los correos electrónicos.

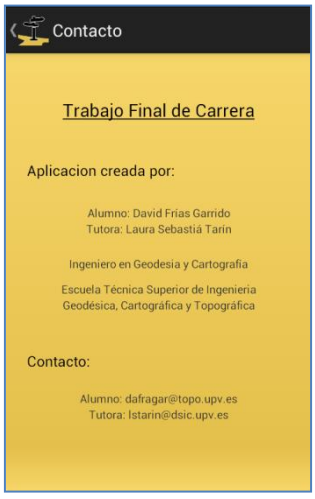

Imagen 58 – Contacto

4.3.7. Ayuda

Si se presiona en el último botón del *Main*, "*Ayuda*", aparece una interfaz con tres botones, cada uno de ellos lleva a la descripción de ciertas partes de la aplicación.

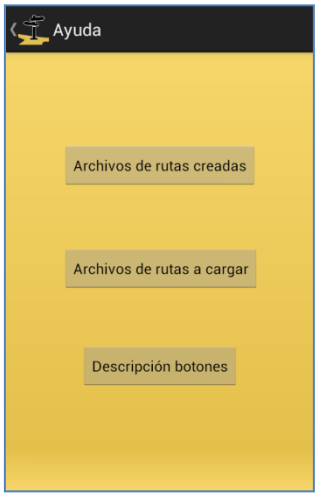

Imagen 59 - Ayuda

Pulsando el primer botón, "*Archivos de rutas creadas*", aparece una *layout* describiendo unas consideraciones del archivo de las rutas creadas.

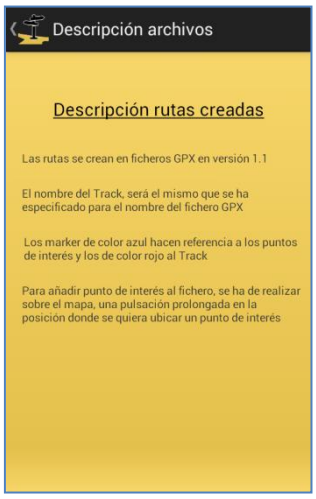

Imagen 60 - Descripción de las rutas creadas

Seleccionando la segunda opción, "*Archivo de rutas a cargar*", volverá a salir otra interfaz describiendo unas pautas para el archivo GPX a cargar.

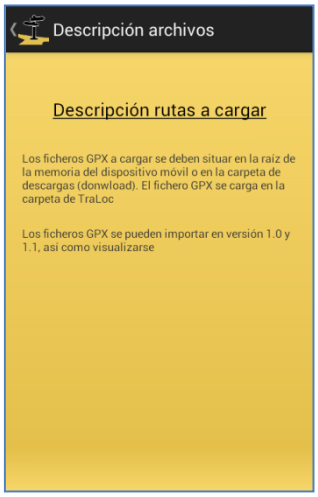

Imagen 61 - Descripción de las rutas a cargar

El último botón hace referencia a la descripción de cada uno de los botones del *Main*.

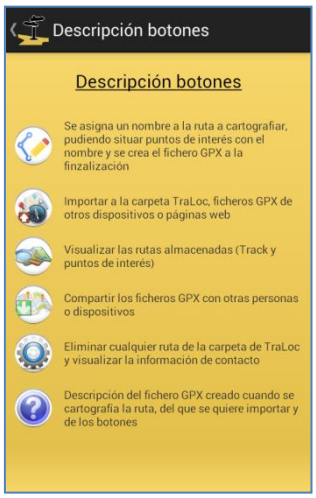

Imagen 62 - Descripción de los botones

4.3.8. Acerca de…

Si en cualquier *layout* se presiona sobre el botón físico menú (o el icono con tres puntos verticales, en la *ActionBar*), se obtiene un submenú con dos opciones.

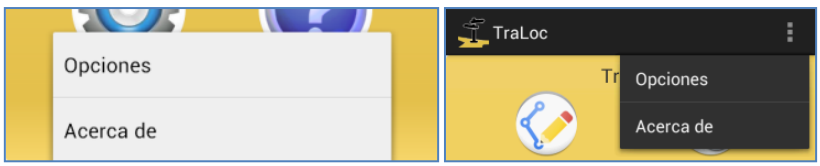

Imagen 63 - Submenú

Al escoger la segunda opción del submenú, "*Acerca de...*", aparece un *layout* totalmente diferente a los vistos hasta ahora. Al iniciarse la actividad, ésta funciona como una ventana *pop-up*. Ofrece información del autor de la aplicación y su finalidad.

Imagen 64 - Acerca de

4.4. Google Play Services

Antes de hacer uso de los mapas de Google Maps y de obtener la localización, hay que hacer unos preparativos previos al proyecto para que funcione correctamente la aplicación, ya que la API v2 de los mapas y de localización se proporciona como parte del SDK de Google Play Services, con lo que se ha de preparar el entorno de desarrollo con el paquete.

Lo primero que se ha de realizar para hacer uso de cualquiera de las APIs incluidas en Google Play Services, es importar en Eclipse el proyecto de la librería donde se implementan. Para ello se abre Android SDK Manager, se accede al apartado de Extras y se selecciona "Google Play Services".

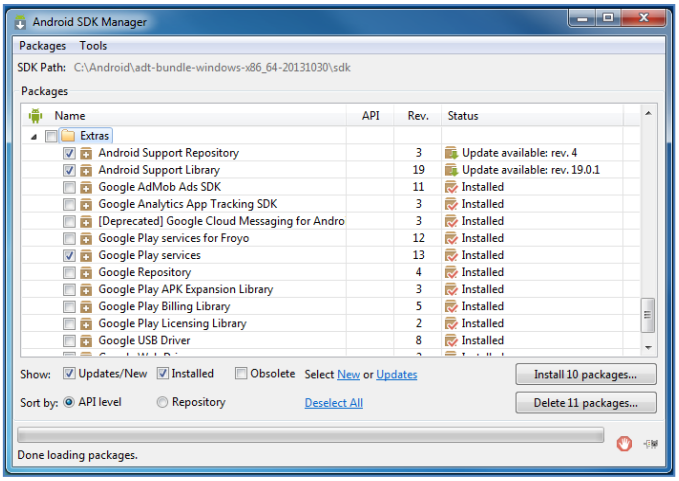

Imagen 65 - Android SDK Manager

Una vez descargado e instalado, falta importarlo. Para ello, en Eclipse, "*File / Import…*" se selecciona "*Android / Existing Android Code Into Workspace*", se pulsa en *Next* y en la siguiente ventana en "*Root Directory*", se selecciona la ruta de la librería[17]. Hay que asegurarse de que el proyecto queda marcado en la lista de proyectos importados y se selecciona la opción "*Copy projects into workspace*".

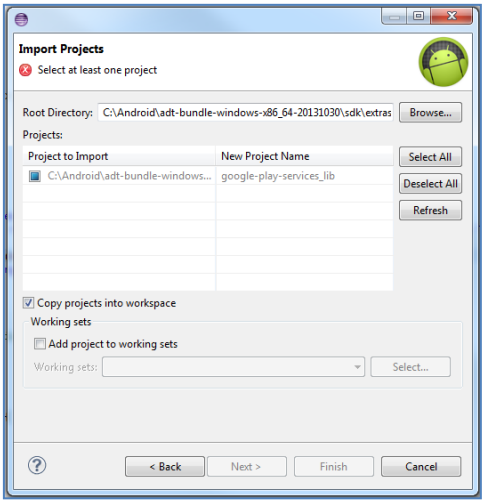

Imagen 66 - Importar proyecto

Se pulsa sobre el botón "*Finish*" y queda importado en el explorador de paquetes de Eclipse. Ahora, se ha de entrar a las propiedades del proyecto importado, con el "*botón derecho / Properties*". Se marca "*Java Build Path*" y hay que asegurarse que la opción de "*Android Private Libraries*" está marcada en la ventana "*Order and Export*".

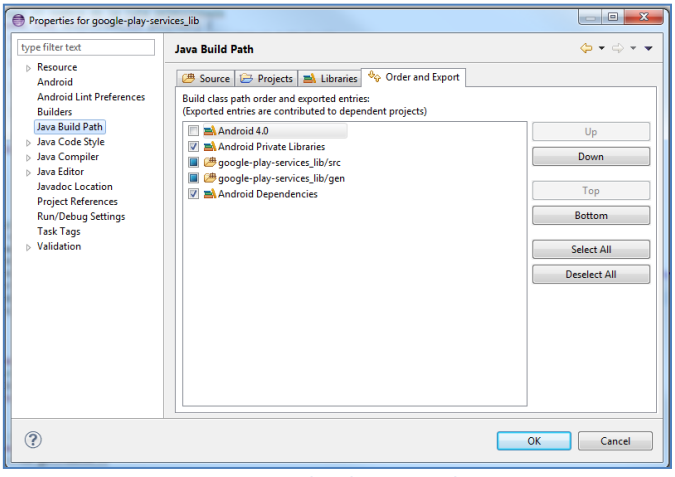

Imagen 67 - Android Private Libraries

Se acepta y con ello se tiene el proyecto de librería preparado. Ahora queda configurar el proyecto donde se usará esta librería, por tanto se accede a las propiedades

[17] <ruta-sdk>\extras\google\google_play_services\libproject\google-play-services_lib

del proyecto TraLoc, "*botón derecho / Properties*", se marca Android y ahí se debe añadir la referencia al proyecto anterior donde se ha importado Google Play Services. Se pulsa sobre "*Add…*", se selecciona el proyecto importado y se acepta.

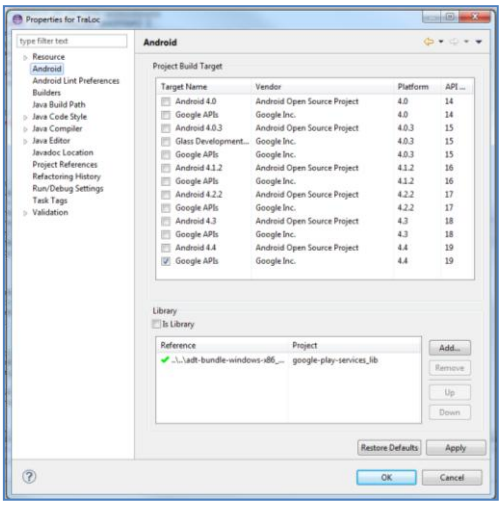

Imagen 68 - Añadir proyecto importado

Se accede al AndroidManifest.xml del proyecto y se han de añadir las siguientes líneas:

```
<meta-data android:name="com.google.android.gms.version"
           android:value="@integer/google_play_services_version" />
```

El siguiente paso es obtener la API *Key* para poder hacer uso de los mapas de Google. Se accede a la Consola de APIs de Google[18], se crea un nuevo proyecto (parte superior izquierda). Se rellena el nombre del proyecto y el ID del proyecto. Después, se accede a la opción "*APIs & Auth*" y se activan las APIs de Google. Para TraLoc, se selecciona "*Google Maps Android API v2*".

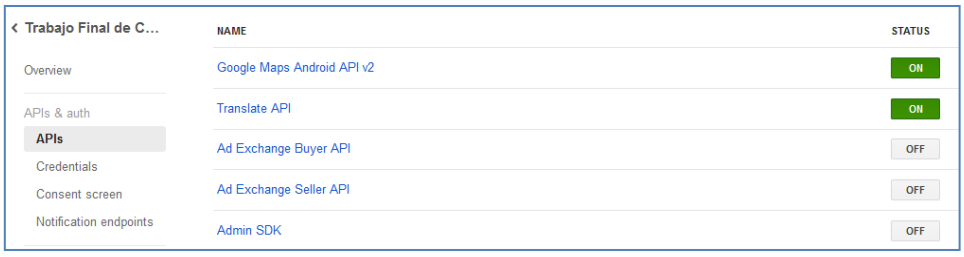

Imagen 69 - Selección de las APIs

Posteriormente a la activación de las APIs necesarias se accede a la opción "*Registered Apps*" y se pulsa sobre el botón "*Register apps*", para registrar la aplicación

[18] https://code.google.com/apis/console/

TraLoc. Ahora se ha de indicar el nombre de la aplicación, el tipo (Android), el tipo de acceso ("*Accessing APIs directly from Android*") y el paquete Java utilizado en la aplicación (*com.example.traloc*) y la huella digital SHA1 del certificado con el que se firma la aplicación.

Para encontrar la huella digital en Eclipse, se accede a "*Windows / Preferences*" y se marca "*Android / Build*". A modo de explicación, toda aplicación ha de ir firmada para poder ejecutarse en cualquier teléfono móvil o emulador. Por ello, se ha de firmar antes de distribuir la aplicación públicamente. Durante el desarrollo de la misma, se realizan pruebas, con lo que la aplicación está firmada, pero con un certificado de pruebas.

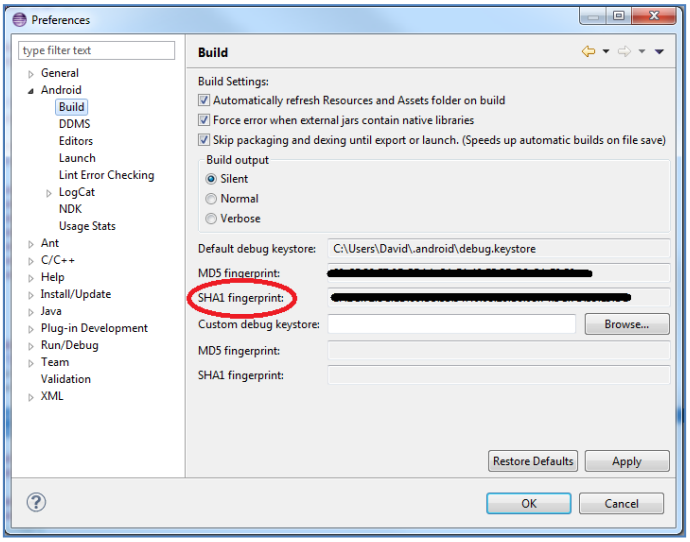

Imagen 70 - Huella digital SHA1

Después de rellenar los datos, se pulsa sobre el botón "*Register*" y se muestra por pantalla la clave generada para la aplicación.

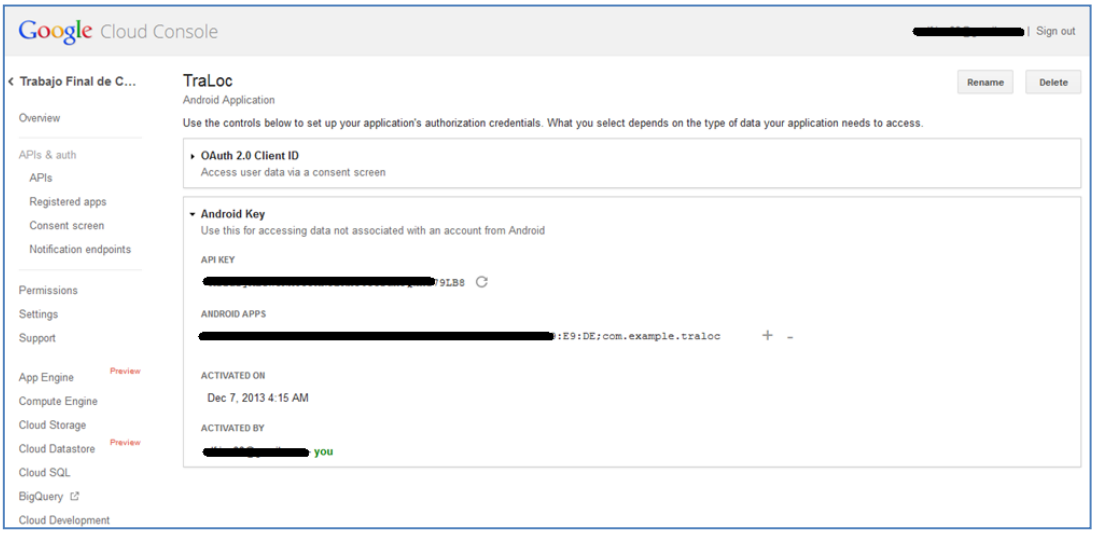

Imagen 71 - API *Key*

Como se hizo anteriormente, se ha de modificar el fichero AndroidManifest.xml. Se introducen las siguientes líneas (donde se inserta la API *key* generada en "*API_key*"):

```
<meta-data android:name="com.google.android.maps.v2.API_KEY"
           android:value="API_key"/>
```

Como la API v2 de Google Maps utiliza OpenGL ES versión 2, también se ha de especificar:

```
    <uses-feature android:glEsVersion="0x00020000"
                  android:required="true"/>
```

Y por último se ha de incluir una serie de permisos para que permitan a TraLoc recibir mapas (*MAPS_RECEIVE*), acceder a internet (*INTERNET*), conocer el estado de la red (*ACCESS_NETWORK_STATE*), acceder al almacenamiento externo del dispositivo para la cache de mapas (*WRITE_EXTERNAL_STORAGE*), hacer uso de los servicios web de Google (*READ_GSERVICES*) y los permisos para obtener la localización aproximada a través de la red móvil o WIFI y más precisa a través del GPS (*ACCES_COARSE_LOCATION*, *ACCES_FINE_LOCATION*).

```
<permission
            android:name="com.example.traloc.permission.MAPS_RECEIVE"
            android:protectionLevel="signature"/>

   <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
   <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
   <uses-permission android:name="android.permission.INTERNET"/>
   <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
   <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
   <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVIC
ES"/>
   <uses-permission
android:name="com.example.traloc.permission.MAPS_RECEIVE"/>
```

Además de los permisos especificados más arriba, TraLoc utiliza otro más, para poder hacer uso de la vibración:

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

4.4.1. Localización Android

Para poder realizar el cartografiado de las rutas, la primera tarea consiste en obtener la ubicación del usuario a través del *Smartphone* y conforme vaya avanzando en su actividad, ir obteniendo nuevas ubicaciones que irán representándose en el mapa.

En un principio, la localización se obtuvo a través de la clase *LocationManager* junto con la clase *Location*. *LocationManager* es la que se encarga de gestionar los datos de localización, capturados por alguno de los proveedores de coordenadas y a través de los métodos de esta clase, se obtiene el objeto *Location*.

Para elegir entre los distintos tipos de localización y actualizarla, se utiliza *requestLocationUpdates*, donde se elige el proveedor (GPS o *Network*) que servirá las coordenadas y el intervalo de tiempo entre actualizaciones, midiéndolo respecto al tiempo (segundos) y al desplazamiento (metros). Con el siguiente ejemplo se actualizaría la localización cada 5 segundos y siempre que se haya desplazado más de 10 metros de la posición anterior.

```
mLocationManager
=(LocationManager)getSystemService(Context.LOCATION_SERVICE);
mLocationListener = new MyLocationListener();
mLocationManager.requestLocationUpdates(LocationManager.NETWORK_PROVID
ER,5000,10, mLocationListener);
```

Según experiencias de algunos usuarios y las obtenidas en las pruebas de TraLoc, dichas variables de actualización no se emplean correctamente, porque aunque no se cumplieran las dos condiciones (tiempo/espacio), *requestLocationUpdates* actualizaría la posición del usuario.

Para hacer uso de los dos objetos (*Location* y *LocationManager*), es necesario implementar la clase *LocationListener*, incluyendo obligatoriamente cuatro métodos de tipo *listener* (escuchadores) de eventos. Estos se encargan de escuchar los eventos cuando:

- Se produce una actualización de la localización (*onLocationChanged*)
- El proveedor está desactivado (*onProviderDisabled*)
- El proveedor se activa tras haber sido desactivado (*onProviderEnabled*)
- El proveedor cambia de estado (*onStatusChanged*)

A través de este primer *listener* (*onLocationChanged*) se puede crear el objeto *Location* y a partir de él obtener las coordenadas latitud y longitud.

Después de estudiar el objeto *GoogleMaps*, se observó que existía un método más apropiado, *OnMyLocationChangeListener*. A través de él, se obtiene el objeto *Location* y a partir de este objeto, toda la información necesaria para obtener la localización. Por tanto, se decidió optar por este método y reestructurar la actividad.

Este método suministra un mejor hallazgo de ubicación y reduce el uso de energía. Además tiene la capacidad de poder ir cambiando entre los distintos tipos de proveedor, conforme estén listos para usarse, eligiendo siempre, como criterio, el que mejor precisión obtenga.

OnMyLocationChangeListener escoge el mejor proveedor, *FUSED* (A-GPS), que utiliza siempre la mejor localización a través del *Smartphone*. En un principio el A-GPS, GPS Asistido, escoge la red, para hacer un posicionamiento grosero (obteniéndose al instante la localización), hasta que obtiene datos GPS y realiza la localización fina (tarda más en conectar). Una vez el *Smartphone* obtiene la ubicación a través del GPS, siempre obtendrá la localización a través de él, exceptuando cuando no la encuentre y pasará a elegir la localización por red.

Al utilizar este método surgen dos problemas:

1. Cada vez que la posición cambia, se registra una ubicación, lo que provoca que se obtenga un fichero GPX de un tamaño considerable, junto a su peso (estos archivos suelen ocupar pocos KB).
2. La mejor precisión que se ha ido obteniendo en las diferentes pruebas ha sido de unos 5 metros, con lo que al obtener coordenadas no habiendo transcurrido esa distancia, podrían no ser correctas.

Como solución a lo anteriormente dicho, se implementa una nueva opción. Si la distancia entre el punto anterior y el nuevo punto de ubicación, no es mayor de 10 metros, no se introduce este punto dentro del fichero GPX. Pero además, no sólo tiene que ser una distancia mínima, sino que no ha de sobrepasar una distancia máxima. Si el *Smartphone* pierde la señal, y ubica el punto muy lejos, se considera que ha tomado mal la ubicación, se desestima y al volver a captar la localización se captaría la posición correcta.

Esto se realiza para introducir los puntos de ubicación en el fichero GPX (figura 2, línea de color morado). Trabajando con el mapa, se han de ir dibujando las localizaciones conforme va cambiando la posición, para que el trazado del *track* guarde una estética visual correcta (en la figura 2 línea de color verde).

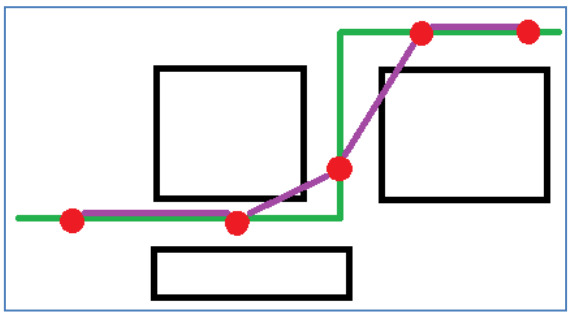

Figura 2 - Visualización de la localización

La utilización del A-GPS realiza una localización aproximada de la ubicación y conforme conecta con el GPS, realiza una localización más fina. Al empezar la ruta, puede que el *Marker* que indica su inicio, se coloque en un sitio inadecuado y al tomar las nuevas ubicaciones se realicen líneas dibujadas en el mapa, que realmente no tendrían que aparecer. Hay que realizar unas comprobaciones respecto a las distancias de los primeros puntos.

Imagen 72 - Mala ubicación del primer punto

Se supone que, cuando un usuario va iniciar la ruta, está quieto, inicia TraLoc y posteriormente, inicia la actividad. Por lo tanto, si el usuario no se mueve, se han de obtener las mismas coordenadas, siempre teniendo en cuenta la precisión con la que se pueden obtener. Se muestra un *ProgressDialog*, para indicar al usuario que se está obteniendo su ubicación y esta ventana de aviso, se cierra cuando ha encontrado la ubicación correctamente.

Para adquirir la ubicación correctamente, se van obteniendo las localizaciones del usuario para el primer punto del *track*. Como en un principio son aproximadas, con el tiempo de exposición, la toma de esas coordenadas va mejorando, reduciéndose la precisión, con lo que las coordenadas de los primeros puntos, no han de variar muchos metros. Por dicha razón, se realiza un bucle, comprobando la distancia entre los puntos de localización; mientras esta distancia no sea mayor de 10 metros, se cuenta en una variable las veces que esa distancia sea inferior y cuando la variable sea igual a 3, se obtendrá la localización correcta del usuario y se insertará en el mapa el *Marker*. A partir de aquí comenzará a cartografiarse la ruta.

Cuando se está cartografiando la ruta, se puede pulsar sobre cualquier botón existente en la *layout* para realizar una acción. Si se pulsa en el botón "*PAUSAR*", se detiene la toma de datos y no se vuelve a reanudar hasta que no se presiona "*CONTINUAR*". Aquí puede surgir un problema, ya que a veces, el dispositivo móvil al reanudar la localización empieza a tomar una ubicación de forma errónea realizando un trazado incorrecto. En cambio otras veces no sucede y se reanuda de forma correcta la toma de datos.

Lo explicado anteriormente indica lo que se observa en la siguiente imagen. En la imagen 73 el punto negro es donde se ha realizado el pausado y la reanudación de la toma de datos, con lo que el triángulo que aparece, es erróneo, solo tendría que aparecer una línea del punto negro al *Marker*.

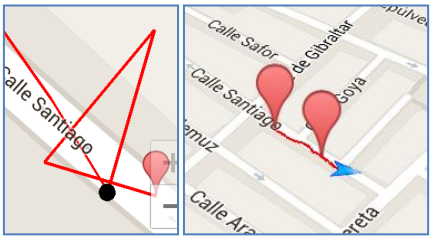

Imagen 73 - Pausado y continuado

Este problema no se ha podido solucionar, ya que es debido a la precisión del dispositivo móvil. Si las precisiones fueran muy pequeñas o no se hiciera el pausado y continuación, el *track* se realizaría de forma correcta.

Se ha de tener en cuenta la actividad a la que se destina TraLoc, porque no es lo mismo realizar una actividad en la montaña o en la ciudad. Los puntos de localización se toman a partir de 10 metros, con lo que se pueden encontrar casos como el siguiente, donde las esquinas de los edificios las corte el *track*:

Imagen 74 - El *track* corta esquina edificio

Si, por el contrario, la distancia entre puntos de localización fuera muy pequeña, se podría cartografiar la ruta con muchos más detalle, por ejemplo, se describiría el trazado realizado a través de una rotonda y cambios de orientación, aun yendo en un vehículo.

Imagen 75 - Cambio de orientación y rotondas

Las precisiones obtenidas en las coordenadas (latitud y longitud) de localización a través del GPS del dispositivo móvil se han llegado a obtener con una mínima de 5 metros. Por norma general es la precisión que se suele dar, siempre y cuando, las condiciones sean favorables, pero en las zonas donde no hay muy buena cobertura para captar satélites y no es una zona despejada, sube esa precisión hasta los 20m.

La precisión de las coordenadas a través de la red de internet suele tener una media de 1400m, con lo que no es nada aconsejable obtener la ubicación a través de ella. A modo de apreciación, si se conecta el dispositivo móvil a una red WIFI de internet, la precisión que se obtiene es casi exacta, ya que Google, con su coche *Street View*, recogió las *macs* de los *routers* y los tiene asociados a unas latitudes y longitudes. Si un usuario se conecta a una de esas redes, inmediatamente se le asigna la latitud y longitud que asignaron a esa *mac* (*router*). Se suele obtener muy buena precisión, por no decir casi exacta, pero en este caso, no es posible utilizarlo, ya que en la montaña no existen redes WIFI.

Respecto a las altitudes dadas por GPS son poco fiables, más o menos se suele desviar unos 20 metros, en el mejor de los casos. La altitud por el proveedor *Network* no se puede obtener, ya que resulta siempre 0. Como en SL4A, obtener la altitud a través del GPS puede servir para ver un perfil sin escala.

Los dispositivos móviles utilizan diferentes componentes del GPS, con lo que no todos tienen la misma precisión, de ahí a veces la diferencia de precio de unos modelos a otros. Un teléfono móvil de gama media, no debería suponer ningún problema para obtener buenas precisiones. Este es uno de los componentes de un *Smartphone* en el que los fabricantes suelen utilizar materiales de menor calidad, para reducir el precio final.

4.4.2. Mapas Android

A continuación se van a describir todos los métodos y objetos utilizados para la creación y modificación de los mapas.

La primera tarea que se ha de realizar es añadir, en el *layout* donde se quiere mostrar un mapa, un objeto *fragment*.

```
<fragment
      android:id="@+id/mapa"
      android:name="com.google.android.gms.maps.SupportMapFragment"
      android:layout_width="match_parent"
      android:layout_height="277dp"
      android:layout_weight="1.17" />
```

Posteriormente, en la actividad, se han de implementar en el método *onCreate*, las opciones del objeto *GoogleMapOptions* y *FragmentTransaction*, todo ello iniciando el mapa en el método *onResume*.

En el método *onCreate*, se han de especificar las opciones generales e iniciales del mapa a través del objeto *GoogleMapOptions*; los parámetros que se pueden establecer son:

- *mapType*: El tipo de mapa que se quiere visualizar.
  - Satélite: Mapa con imágenes satélite.
  - Terreno: Mapa con superfície del terreno.

- Híbrido: Mapa que emplea imágenes satélites superponiendo cartografía de carreteras.
- Normal: Mapa vectorial con elementos de cartografía base.

- *camera*: Posicionamiento de la cámara. Se maneja a través del objeto *CameraPosition*, en este objeto los parámetros que se pueden modificar son:
  - *target*: Requiere un objeto *LatLng*, que a su vez, contenga latitud y longitud. Sirve para centrar el mapa
  - *zoom*: Nivel de zoom de la cámara, el valor 0 corresponde con el valor más alejado
  - *bearing*: Determina la orientación del mapa.
    - 0º - Establece la orientación al Norte
    - 90º - Establece la orientación al Este
    - 180º - Establece la orientación al Sur
    - 270º - Establece la orientación al Oeste
  - *tilt*: Inclinación sobre el mapa, es decir, representa el ángulo de visión. La posición inicial, 0º, es perpendicular al plano y conforme se va inclinando la cámara, va dando valores cercanos a 90º. Con este valor, se permite visualizar la cartografía en 3D y dependiendo de la zona en la que se esté y del mapa seleccionado, se puede llegar a ver los edificios en 3D.

Para insertar el mapa en la interfaz del usuario, se debe hacer uso del objeto *FragmentTransaction*, con él, se consigue crear y visualizar el mapa en la pantalla del dispositivo.

Imagen 76 - Mapa inicial

Se ha de forzar la inicialización del objeto *GoogleMap* (mapa). Para ello, se obtiene la vista del mapa cuando resulta nulo y posteriormente cuando el objeto es diferente de nulo, se realizan las tareas correspondientes. En el método *onResume*, a través de este objeto se pueden ir realizando modificaciones sobre el mapa y cambiar las

opciones iniciales establecidas en el objeto *GoogleMapOptions*, por ejemplo realizar animaciones de cámara (*animateCamera*).

```
private void iniciar() {
    if (mMap == null) {
        mMap = mMapFragment.getMap();
    }
    if (mMap != null) {
        // Hacer lo que se quiera realizar con el objeto mapa
    }
}
```

Con todo lo explicado, se puede visualizar el mapa. Se irán añadiendo unas características al objeto mapa, para que represente la información que se desea; en este caso, se mostrará el *track* con sus puntos de inicio/final, y los puntos de interés representándose mediante líneas y *Marker*.

Para representar los puntos inicio/final del *track* y los puntos de interés, hay que utilizar el objeto *Marker*. Para crear un marcador, se especifica la posición y se añade al mapa, además se pueden establecer sus propiedades mediante *MarkerOption*. Ejemplo básico:

```
mMap.addMarker(new MarkerOptions()
        .position(new LatLng(Latitude, Longitude))
        .title("Inicio Ruta"));
```

Las posibles opciones a configurar:

- *Position*: Requiere un objeto *LatLng*, que a su vez éste contenga latitud y longitud, para posicionar el *Marker*.
- *Title*: Texto principal que se muestra al pulsar el marcador.
- *Snippet*: Texto secundario que se muestra al pulsar el marcador.
- *Draggable*: Permite el movimiento del *Marker*. Valores posibles True/False, por defecto false.
- *Visible*: Permite ver el *Marker*. Valores posibles True/False, por defecto true.
- *Icon*: Icono o imagen a visualizar en el *Marker*. Sobre éste se puede modificar el color, de esta forma:

```
.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE))
```

El título y el resto de las opciones se visualizan en una ventana al pulsar el *Marker*. Esta ventana de información es un objeto de tipo *InfoWindows*; para TraLoc no se ha modificado dicho objeto.

Además de utilizar la opción de *Marker* para añadir puntos de interés, se ha utilizado otro método en el mapa: si se realiza una pulsación mantenida sobre el mapa, se llama al método *onMapLongClick*; a través de él, se añaden los diferentes puntos de interés, mediante un *Marker* de color azul. En este *Marker* se coloca el nombre en *Title* y el subnombre en *Snippet*.

Para trazar las rutas sobre el mapa, se han de dibujar líneas sobre éste. Gracias al objeto *PolylineOptions*, se puede realizar dicha acción. En el objeto se añade el nuevo punto de localización y el anterior punto de localización, para crear una línea entre ellos. Posteriormente se establece el grosor y el color de la línea y, por último, se ha de añadir el objeto creado al mapa. Ejemplo:

```
PolylineOptions lineas = new PolylineOptions()
.add(new LatLng(PrimerPuntoLat, PrimerPuntoLon))
.add(new LatLng(SegundoPuntoLat, SegundoPuntoLon));
lineas.width(3);
lineas.color(Color.RED);
mMap.addPolyline(lineas);
```

El último método utilizado es *OnMyLocationChangeListener*, al que se llama cada vez que cambia la ubicación del punto de localización y tiene como parámetro un objeto *Location*.

4.5. Compatibilidad

Debido a la gran diversidad de dispositivos móviles y *Tablets* que hay en el mercado, resulta un problema desarrollar una aplicación que pueda ser utilizada por todos. Si a esto, se añade que cada fabricante crea sus componentes y en ocasiones adapta la configuración de Android para sus dispositivos, resulta obvio que un código que funcione en un terminal, no significa que funcione en otro.

Aunque existan dificultades a la hora de desarrollar aplicaciones móviles, TraLoc se ha desarrollado de forma que funciona en una gran variedad de dispositivos móviles, adaptándose a las diferentes pantallas existentes en el mercado. En los *Smartphone* y *Tablet* en los que se ha probado, funciona, como se observa en los siguientes ejemplos:

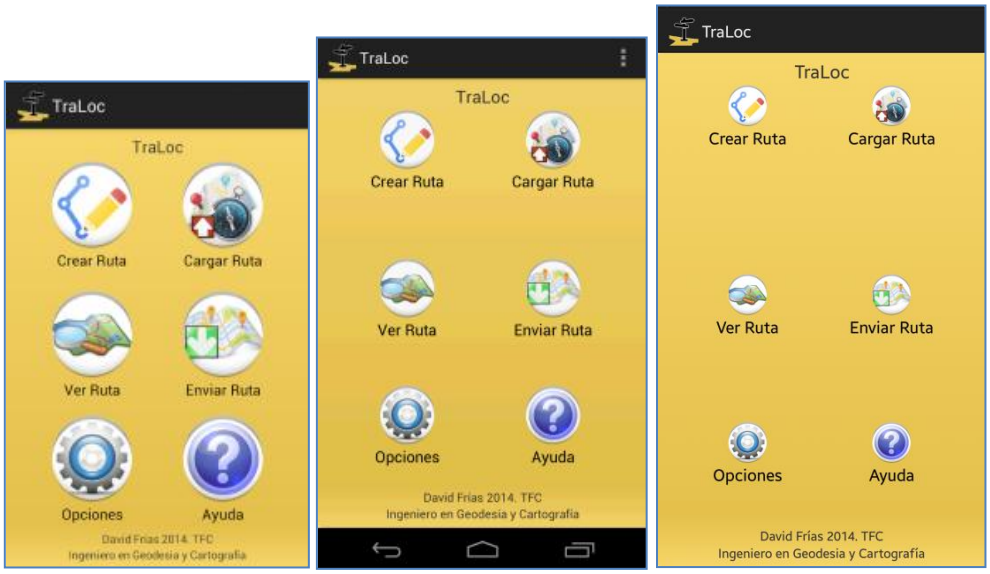

Imagen 77 - Samsung Galaxy SII (Pantalla 4,3"), Nexus 4 (Pantalla 4,7") y Samsung Galaxy S4 (Pantalla 5")

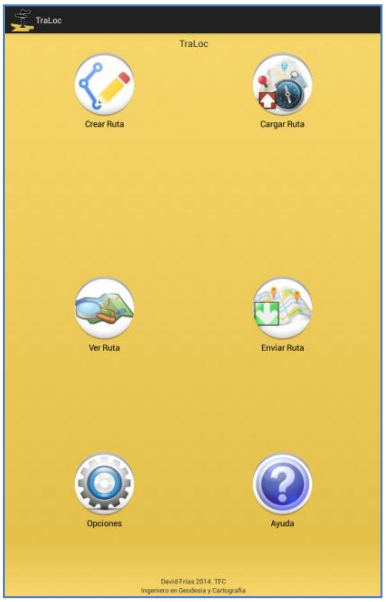

Imagen 78 - Samsung Galaxy Tab 3 (Pantalla 10.1")

En la pantalla principal (*Main*) se aprecia más la diferencia por el tamaño de los iconos, pero la estructura se mantiene. En las demás *layout*, se observa cómo se redimensionan perfectamente todos los elementos y estéticamente son iguales.

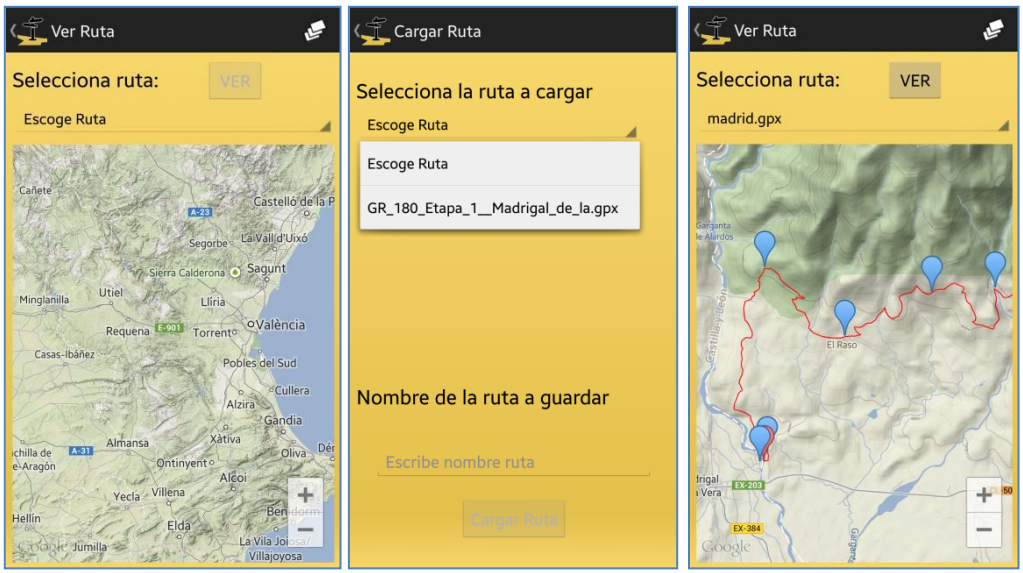

Imagen 79 - TraLoc en Samsung Galaxy S4 - I

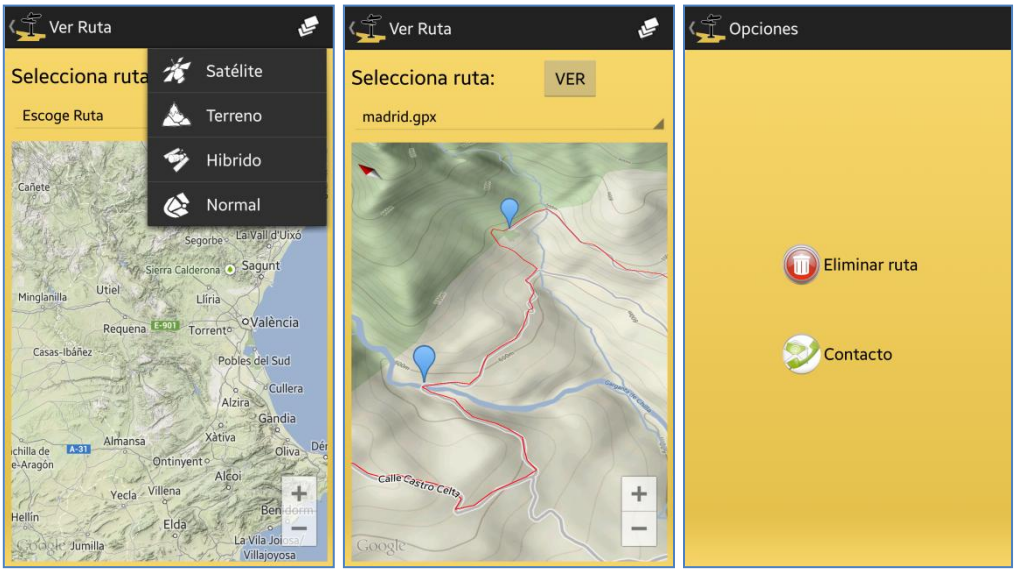

Imagen 80 - TraLoc en Samsung Galaxy S4 - II

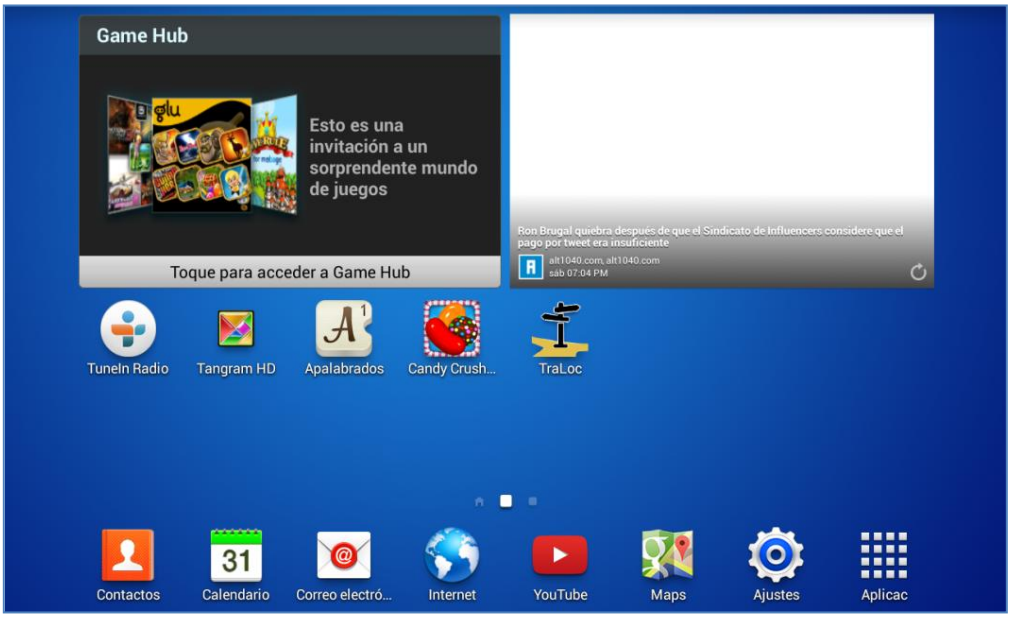

Imagen 81 - Icono TraLoc en Samsung Galaxy Tab 3

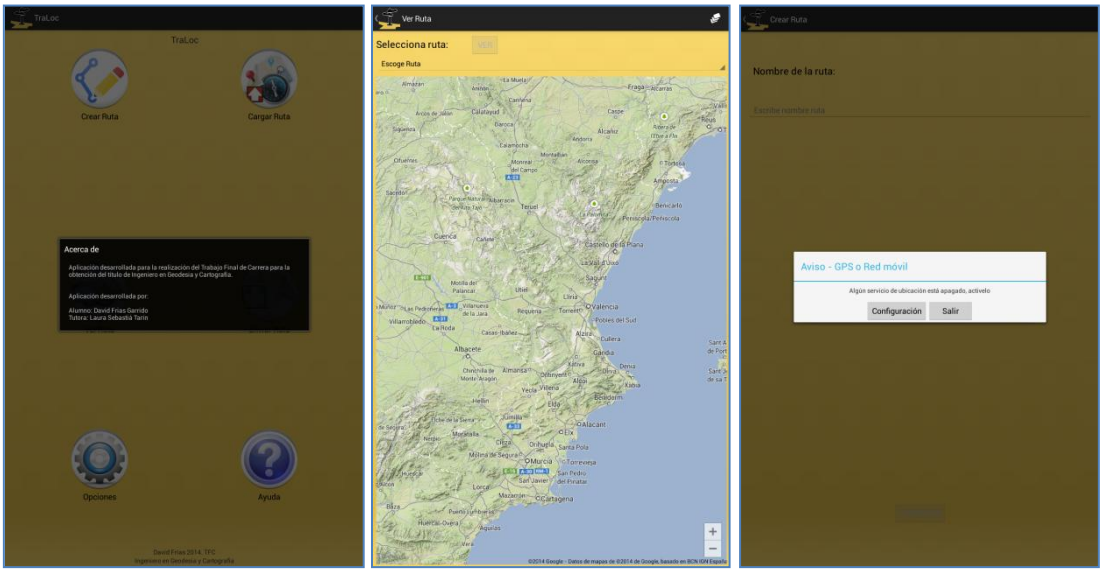

Imagen 82 - TraLoc en Samsung Galaxy Tab 3

4.6. Futuras mejoras y actualizaciones

La aplicación TraLoc, como objeto de este estudio, ya está desarrollada. Sin embargo, ésto no quiere decir que no se pueda mejorar. A lo largo del proceso de las pruebas realizadas y a través de las experiencias de otras personas con sus dispositivos móviles, se han ido observando las diferentes vías de progreso o posibilidades que se pueden implantar para mejorarla.

Una de las mejoras que podría hacerse a TraLoc es la implantación de un servidor para alojar los ficheros GPX. La aplicación iría haciéndole peticiones a través

de internet, para mostrar la información solicitada. El acceso a la información del servidor, en este caso ficheros GPX, no sólo lo tendría un usuario, sino que estaría al alcance de muchas personas, sería una aplicación multiusuario.

La opción de "*Cargar Rutas*" debería poder seleccionar las rutas existentes en todo el terminal, no sólo en la raíz de la memoria interna o en la carpeta descargas (*download*). Cambiaría un poco su finalidad, ya que a la hora de cargar los datos, en vez de enviar las rutas a la carpeta de TraLoc, las enviaría al servidor.

Junto a estas mejoras, habría que insertar un botón en "*Opciones*", para enviar al administrador de la aplicación informes de posibles fallos en las rutas, para que pueda borrarla o modificarla del servidor, ya que el botón de "*Eliminar ruta*" de "*Opciones*", seguiría siendo para las rutas creadas por cada usuario, sólo existentes en la carpeta TraLoc.

Además en "*Opciones*", se crearía un botón de navegación, para que éste guíe al usuario mientras esté realizando la ruta, o enlazar TraLoc a Navigation de Google Maps para que realice esa función.

A la hora de cartografiar una ruta, se debería especificar el tipo de actividad que se vaya a realizar. Así, TraLoc podría seleccionar la distancia con la que tomaría la localización, ya que no es lo mismo estar haciendo una ruta por la montaña, por la ciudad, estar corriendo o ir andando. Se necesita obtener puntos de localización con más o menos distancia, dependiendo de la actividad a realizar.

Habría que efectuar una exhaustiva investigación respecto a la toma de datos en segundo plano, para no gastar tanta batería y poder apagar la pantalla cuando se están capturando datos o poder utilizar otra aplicación diferente. Mientras TraLoc, quedaría en segundo plano cartografiando la ruta. También habría que investigar sobre los botones del sistema, ya que algunos botones (*HOME* y bloqueo móvil) no se pueden desactivar de su acción.

También en la actividad de "*Crear Ruta*", se podría añadir la opción de introducir imágenes a los puntos de interés y alojarlas en el servidor, al pulsar sobre el *Marker* poder visualizar la imagen tomada. Además se podría ir cambiando la orientación de la cámara en el cartografiado de la ruta, conforme ésta va avanzando, la cámara cambiaría la orientación respecto a los puntos de localización, del punto anterior al nuevo, de forma que siempre estaría rotando y moviendo el mapa centrándose en las coordenadas de localización y teniendo una vista vertical.

En "*Ver Ruta*", se podría añadir otro botón, para mostrar el perfil de la ruta seleccionada, aunque los datos no sean los reales, se puede hacer una ligera impresión del perfil que tiene la ruta. A la hora de visualizar la ruta, se ha de reajustar la vista del mapa, ya que como se reajusta sobre el *Marker* en rutas cerradas (inicio y fin el mismo punto), no se visualiza la totalidad del *track*, sino que se corta.

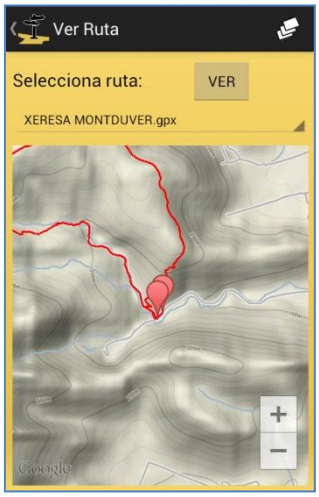

Imagen 83 - Visualización ruta cuadrada

Se podría experimentar con las clases *Geocoder* y *GpsSatellite*, donde *Geocoder* podría utilizarse para insertar los nombres y subnombres de los puntos de interés, así el usuario, solo tendría que modificar el texto, si quisiera introducir uno diferente. Pero habría que investigar si esta clase realiza bien su función fuera de las ciudades.

La clase *GpsSatellite* podría mostrar el número de satélites al que se conecta la aplicación, en cada momento de la toma de datos de localización. En general, investigar con esta clase o con otras, si se podría generar un algoritmo, para reducir la precisión con la que se obtienen los datos de localización y así TraLoc, mejoraría la opción de reanudación de datos, ya que a veces realiza localizaciones erróneas y no obtiene bien el *track* en su reanudación.

Un importante cambio estético, sería realizar una ampliación de los iconos del *layout* principal (*Main*) y del de "*Opciones*", ya que como se ha visto en el apartado de compatibilidad de este documento, los iconos aparecen con un tamaño reducido y no ocupan la totalidad de la pantalla, aunque se redimensiona correctamente a ella. Habría que ampliar los iconos, mediante un software de retoque fotográfico y gráfico, para que se adapte mejor a todos los tamaños de pantalla de los dispositivos móviles.

Con todo lo expuesto, se deja una amplia línea de investigación y desarrollo, para posibles actuaciones sobre aplicaciones móviles para cartografiar rutas.

4.7. Dificultades

Conforme se iba avanzando en el desarrollo de la aplicación TraLoc, se han ido encontrado diferentes dificultades, solventándose de forma desigual. A continuación se describen los problemas encontrados y en cada caso, las soluciones adoptadas o las que se podrían adoptar para salvar dicha dificultad.

Los botones, tanto táctiles como físicos del dispositivo móvil, podían parar el cartografiado durante la creación de la ruta. La forma de resolver esta dificultad fue

eliminar la acción de los botones táctiles en la creación de la ruta, es decir, que aunque se presione alguno no realiza ninguna acción. Pero los botones físicos, como *HOME* o bloquear/desbloquear el terminal, nunca se pueden deshabilitar sus acciones, porque son botones del sistema. Respecto a este problema, en los nuevos dispositivos, existen menos trabas en el desarrollo de aplicaciones, porque todos los botones son táctiles y se pueden desactivar. Para TraLoc el botón *HOME* se ha podido desactivar para los botones táctiles (Nexus 4), pero no para los físicos (Samsung Galaxy SII). En general, hay dificultades de adaptación a todos los dispositivos Android, porque los fabricantes introducen materiales de calidades y componentes diferentes, además de poder modificar Android a su gusto.

Otro problema relacionado con el cartografiado de la ruta, era al girar el dispositivo. Cuando se gira el terminal, la pantalla se orienta según el giro y entonces en la programación se llama de nuevo al método *onCreate*, y por ejemplo se perdían los marcadores que ya estaban tomados, por lo que se decidió realizar toda la aplicación en *portrait* (vertical), además de mantener una mejor visualización en toda la aplicación y salvar dicho error. Después de ir adquiriendo conocimientos en esta materia, se ha comprendido que el error al rotar automáticamente el dispositivo, se podría corregir guardando los objetos (*Marker*) en el método *onSaveInstanceState* y recuperarlos en el método *onCreate*.

Uno de los importantes quebraderos de cabeza, fue la visualización del mapa. Se crea el mapa a través del objeto *GoogleMap*. Se ha de cumplir que el mapa esté disponible y que se inicialice. Ahí está el quid de la cuestión, el mapa no se inicializaba por sí solo, con lo que hay que forzarlo a iniciarlo, para que no resulte nulo.

Uno de los problemas a la hora de cartografiar la toma de datos, es respecto a la primera ubicación. A veces, se obtenía un punto de localización a una distancia considerable de la real, y el siguiente punto podía que ya lo tomara bien o aún no, con lo cual la ruta empezaba con uno, dos o incluso tres puntos de localización erróneos y en la visualización del mapa se observaba una línea considerablemente equivocada.

Imagen 84 - Mala localización del primer punto del *track*

Esto se corrigió controlando la distancia entre los primeros puntos. Teóricamente al iniciar TraLoc, el usuario está parado y después de iniciarlo, comienza la actividad, con lo que la distancia entre el primer punto de localización y el segundo, no puede ser excesivamente grande. Cada vez que se va obteniendo la ubicación para estos primeros

puntos (primer punto de la ruta), se ha de obtener una distancia pequeña entre ellos; cuando esto se produce tres veces, se considera que el *Smartphone* (GPS) ya ha podido ubicar al usuario de forma correcta, con lo que empieza a cartografiar la ruta, TraLoc va un paso por detrás del usuario.

Ha existido dificultad a la hora de trabajar con los ficheros GPX, tanto para crearlos como para visualizarlos. Para crearlos, en un principio cada vez que se obtenía un cambio en la ubicación, se obtenía un punto de localización y se guardaba en el Track. El problema de ésto, es que a la hora de visualizar el fichero GPX o de compartirlo, se ve como es de gran tamaño, tanto en extensión como de peso, además no es factible tener coordenadas que discreparan pocos metros, ya que, la mejor precisión obtenida era de unos 5 metros y dentro de este rango existe la posibilidad de no obtener coordenadas correctas. Por ello, se decidió obtener la distancia entre los puntos de localización y cuando fuera de más de 10 metros, captar esa posición, además no tendría que sobrepasar una distancia límite, por si toma mal la localización, como ocurría en la toma de la primera ubicación.

Respecto a la hora de visualizarlos, el mapa no se centraba bien y se perdía información. Se solucionó centrando el mapa, tomando como referencia los puntos de interés (*Marker*), siempre y cuando existan. Éstos, siempre están más alejados del *track*, y por tanto marcarán la vista de visualización mínima. Si no hay puntos de interés, la vista se reajustará dependiendo de los *Marker* del *track*.

Las pruebas realizadas siempre han sido en sitios despejados con buena visibilidad para capturar el mayor número de satélites. El problema surge cuando estas condiciones favorables no se dan, la precisión aumentaría y existiría mayor error a la hora de obtener la ubicación del usuario.

En relación a la creación de los iconos, sus imágenes no se pueden poner sobre fondo blanco, porque se representaría este fondo. Por tanto se han de dejar sin fondo con la imagen que se desee, lo que implica trabajar con un software de edición de imágenes para poder tratarlas. Además se ha de tener en cuenta, si se va a situar en un fondo oscuro o claro, para que el icono tenga unos colores u otros de contraste.

A la hora de habilitar o deshabilitar los botones para que se cumplan o no las restricciones, se ha de comprobar todas las posibles opciones que se pueden dar, para que esté todo bien realizado. Esto es engorroso de comprobar, ya que influyen varios métodos conectados para habilitarlos o deshabilitarlos.

Como última dificultad encontrada es respecto a la información existente en la página de Android Developer. Es una página donde existe muchísima información, está explicada correctamente, pero como la documentación de desarrollo de Android es muy amplia y densa, se ha de conocer bien todas las herramientas (métodos y clases) para poder desarrollar aplicaciones. Siempre se descubren nuevas clases o métodos con diferentes aplicaciones a ámbitos para poder introducirlos a las necesidades. Todo esto

es debido a que la página no se estructura muy bien, ya que tiene muchos enlaces a todos los elementos y resulta difícil navegar entre tanta información y sus vínculos.

5. Análisis y conclusiones

Hoy en día, todo el mundo lleva un móvil o una *tablet* en el bolsillo, en el bolso o bolsito, la humanidad vive rodeada por la tecnología y depende mucho de ella. Ahora mismo, se vive una época marcada por la crisis económica, donde los puestos de trabajo escasean y las personas jóvenes se tienen que ir abriendo un hueco dentro de este mundo de leones para poder seguir su vida.

La sociedad depende de la tecnología y la evolución de ésta. Cada día, todo está más informatizado, ofreciendo facilidades en el trabajo y en la vida personal. Este ámbito, está continuamente en desarrollo, es un campo que ofrece amplias posibilidades con las que experimentar.

Con el trabajo final de carrera, se han puesto en práctica los conocimientos adquiridos durante los años estudiados, en la titulación de Ingeniero en Geodesia y Cartografía y se han aplicado a este mundo en continuo desarrollo, para poder llegar a tener una nueva e innovadora vía de trabajo, de la que aún queda mucho por experimentar una vez se acceda al mundo laboral.

Por dichos motivos, se realiza la aplicación móvil, basada en la localización del usuario. Así, se encuentra una nueva vía laboral, distinta a la convencional, también con la intención de difundir como un Ingeniero en Geodesia y Cartografía puede llegar a desempeñar trabajos tan útiles como éste.

Con todo ello, se decide investigar esta línea de trabajo, donde se experimenta, analiza y valora las diferentes aplicaciones realizadas con los distintos programas y de ellos, se tendrá en cuenta tanto la facilidad de programación, como la estética de la aplicación.

Comenzando, SL4A. Cualquier aplicación desarrollada a través de este programa, ha de ejecutarse a través de SL4A y nunca se podrá instalar en el teléfono (tener su propio icono), ejecutarla sin SL4A. Las APPs creadas, estéticamente no son muy llamativas, son rudimentarias, y la salida de datos no es muy visual. Además de ello, no se pueden crear aplicaciones con diversas pantallas, se ha de realizar en una única *layout* y a través de ella, se ejecuta el código de arriba abajo, sin poder intercalar otras actividades entre ellas.

Sin embargo, SL4A tiene una gran facilidad a la hora de programación, ya que hay mucha información por la red, y sobre todo existe una página web[19] para Android, donde se explican los métodos implementados y lo que necesitan. La dificultad de ello, podría residir en que la mayoría de documentación está en inglés. Con SL4A, se pueden crear pequeñas aplicaciones a nivel usuario (no comercial), con gran facilidad, bastante rapidez y con una interfaz sencilla, a un golpe de vista. Una de las cosas que hay que

[19] http://www.mithril.com.au/android/doc/index.html

destacar de SL4A es que existe la posibilidad de modificar el código de programación a través del *Smartphone*.

Pasando al segundo programa utilizado, Processing. Al ser un software libre y de código abierto, existe una multitud de librerías que se pueden aplicar a muchos campos para su investigación y además de ello, la gran mayoría están documentadas. La utilización de las librerías facilita mucho el trabajo, es posible aprovechar la labor de otras personas, adaptándola a tus necesidades. Mediante el uso de algunas librerías, se puede mejorar la opción visual de las aplicaciones desarrolladas mediante Processing, por ejemplo, la creación de botones.

Otra característica diferente a SL4A, es la ejecución de la aplicación en el *Smartphone*, ya que se puede acceder a ella, sin ejecutarse a través de otro programa o aplicación.

Mediante Processing se deben especificar los permisos utilizados por la aplicación, sin ellos no funcionaría, ya que estos dan acceso a las actividades que realizan.

Processing está diseñado para creaciones artísticas visuales, multimedia e interactivas, por lo que al crear cualquier aplicación, hay que tener especial cuidado con los fondos que se ponen en cada apartado, ya que se sobrescriben, para que no se pinten los colores de fondo entre ellos y sobre la información que se quiere mostrar, con lo que la parte de esta programación suele ser más laboriosa, al estar pendiente de muchas cosas puntuales. Como en SL4A, existe el inconveniente de que en el diseño de la aplicación solo se puede hacer mediante un *layout*.

Los dos programas más utilizados para la creación de aplicaciones móviles son Eclipse y el nuevo programa Android Studio[20]. De ellos, Eclipse, es el programa más extendido y se ha utilizado para desarrollar la aplicación objeto del trabajo final de carrera, TraLoc.

Eclipse es un programa basado en código abierto, pero a diferencia de los otros, tiene una multitud de opciones, para poder configurar y realizar una aplicación más completa, tanto a nivel visual, como las opciones a utilizar o para manejar la aplicación y el programa. Se puede trabajar en diferentes *layouts* a la vez, ir adelante o atrás en la APP, sin perder datos e intercambiándose actividades y procesos. La creación de aplicaciones móviles a través de Eclipse, se facilita, gracias a la información existente en la red de otros usuarios, ya que hay mucha información que se puede adaptar a las necesidades que se requieran.

Como en Processing, se han de especificar los permisos que se van a usar a través del fichero AndroidManifest.xml, sin éstos, no se podrían utilizar los recursos en

[20] http://developer.android.com/sdk/installing/studio.html

la aplicación. Cuando un usuario se descarga la aplicación desde el Play Store, se produce un aviso, para informarle de los permisos que utilizará dicha aplicación.

El mayor inconveniente de la utilización de Eclipse es que es un programa que consume muchos recursos del ordenador, a la hora de encenderse y cargar el proyecto de la aplicación, necesita un tiempo elevado para poder manejarlo. Además, a la hora de utilizar el AVD, suele dar problemas y en el momento de la ejecución del emulador, se ralentiza el ordenador y el AVD. Por tanto, resulta una mejor opción probar las aplicaciones desarrolladas directamente a través del teléfono móvil.

A la hora de realizar la aplicación, se ha de especificar la versión mínima y máxima, para la que está desarrollada la aplicación en Android, es decir, entre que versiones funcionará dicha APP. Además de esta especificación, a la hora de programar, existe diferente sintaxis de código para un mismo resultado, ya que para versiones inferiores a X, se hace de una forma y para versiones superiores a X, se realiza de otra. La visualización de la aplicación variará, dependiendo de la versión de Android que se tenga, a su vez, la aplicación se adapta a los distintos tipos de pantalla de los dispositivos móviles o tabletas.

Cuadro resumen-comparativo entre los tres programas utilizados:

	SL4A	Processing	Eclipse
Año de lanzamiento	2009	2008	2001
Plataforma de desarrollo	Android	Windows Mac Linux	Windows Mac Linux
Licencia de software	Código abierto	Código abierto	Código abierto
Lenguaje de programación	Python	Java	Java (se pueden utilizar otros)
Multilenguaje	English	English	Si
Dependencia programas (ejecución aplicación)	SL4A	No	No
Utilización librerías	Si	Si	Si
Aplicación con multipantalla	No	No	Si
Modificación código *Smartphone*	Si	No	No
Edición permisos	No	Si	Si

Tabla 4 - Comparativa SL4A - Processing - Eclipse

6. Instalación de los programas

6.1. SL4A

Hay que tener en cuenta que casi todo el proceso de instalación de SL4A, es a través del *Smartphone*, con lo que es recomendable realizar el proceso de instalación, cuando se disponga de una red inalámbrica, para poder conectarse a ella, ya que se van a descargar archivos de internet a través del terminal.

Antes de ver cómo se instala el programa SL4A, hay que realizar una pequeña modificación en el teléfono móvil con sistema Android, para poder instalar aplicaciones cuyas fuentes no están verificadas por Google (que no se encuentra en el Play Store para su descarga). Para ello se ha de ir a "*Ajustes / Seguridad*" y se selecciona "Fuentes *Desconocidas*" (dependiendo de la versión de Android, puede variar la ruta de fuentes desconocidas).

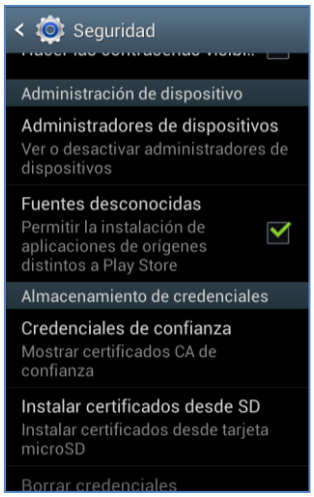

Imagen 85 - Fuentes desconocidas

Una vez realizado ese paso, se abre el navegador, a través del *Smartphone* o mediante el ordenador y se escribe la siguiente url: http://code.google.com/p/android-scripting/. Se procede a la descarga de los archivos, SL4A_r6[21] y PythonforAndroid_r4[22], que se sitúan en la parte izquierda de la página web en el apartado de "*Download*", al lado del código QR.

[21] http://code.google.com/p/android-scripting/downloads/detail?name=sl4a_r6.apk&can=2&q=sl4a_r6
[22] http://code.google.com/p/android-scripting/downloads/detail?name=PythonForAndroid_r4.apk&can=1&q=android

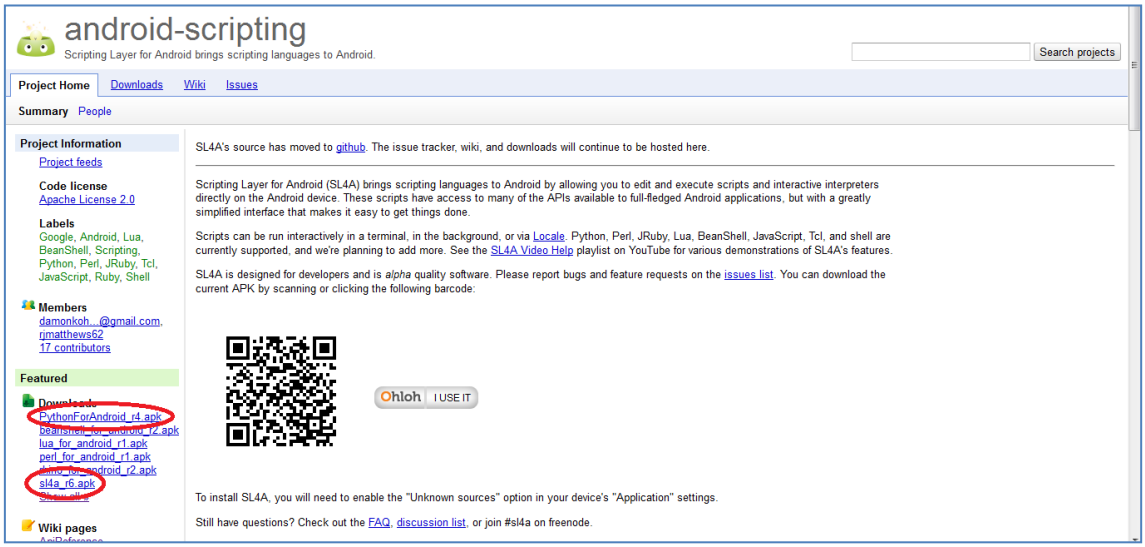

Imagen 86 - Página web para descargarse SL4A y Python for Android

Una vez descargados, si se ha realizado el proceso de descarga mediante el ordenador, se han de pasar los dos archivos, .apk, al *Smartphone*, y se busca la ruta donde se han copiado los archivos. Si por el contrario, se han descargado a través del terminal, se accede a la carpeta de descargas.

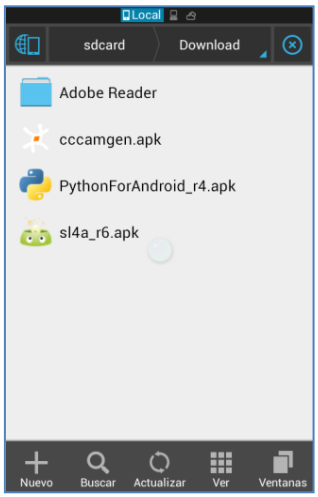

Imagen 87 - Carpeta descargas *Smartphone*

A continuación, se ha de instalar el archivo de Python. Se presiona sobre él para instalar. Atención, en este paso, es donde se descargaran los datos extra.

Se ejecuta y se selecciona "*Instalar*", en la siguiente pantalla, se presiona sobre el botón "*install*" y comienzan a descargarse datos de la aplicación. Cuando termina la descarga, avisa que se ha instalado correctamente la aplicación y se sale de ella. Con esto, queda instalado Python for Android en el *Smartphone*.

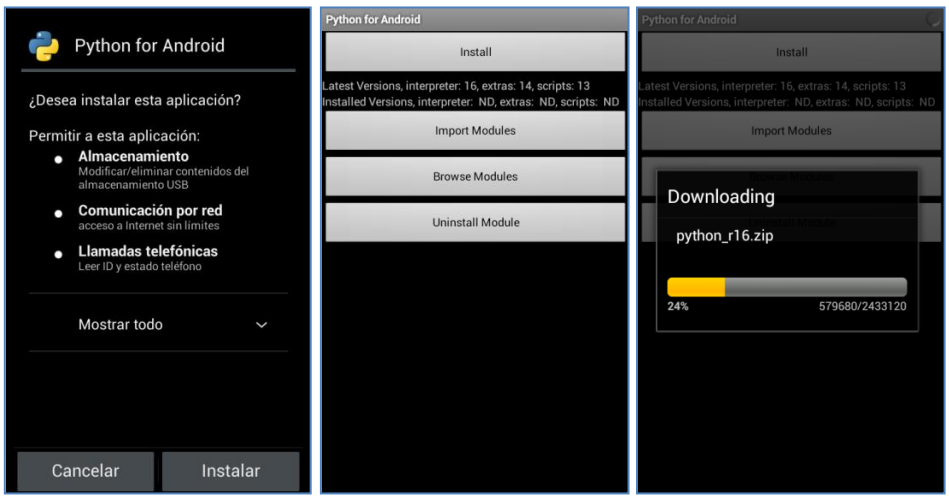

Imagen 88 - Instalación Python for Android y descarga de archivos

Ahora, se va a proceder a instalar el SL4A. Se vuelve donde están los archivos descargados y ahora se presiona sobre el otro .apk, SL4A_r6. Se ejecuta y se selecciona "Instalar".

Imagen 89 - Instalación SL4A

Cuando se termina el proceso de instalación de este .apk, se tendrá instalado todo lo necesario, para realizar la aplicación sobre SL4A. En la pantalla del dispositivo móvil aparecerán los dos iconos de las aplicaciones instaladas. Solo se ha de utilizar la aplicación SL4A para ejecutar las aplicaciones desarrolladas, ya que Python for Android es un intérprete de Python para ejecutar SL4A.

Imagen 90 - Iconos SL4A y Python for Android

Para poder desarrollar con comodidad cualquier aplicación, se puede programar desde el ordenador, en cualquier bloc de notas y luego cambiarle la extensión por ".py". Después se conecta el dispositivo móvil al ordenador y se copia el archivo programado en la url "*D:\sl4a\scripts*"[23]. O también se puede programar directamente desde el teléfono.

Imagen 91 - Edición código *Smartphone*

Para DFTracker, el procedimiento seguido ha sido programar la aplicación mediante IDLE (Python GUI)[24], transferir el archivo al teléfono y luego se realizan algunas pequeñas modificaciones sobre SL4A, editando el fichero.

A modo de explicación, IDLE (Python GUI), se tiene instalado gracias a tener una versión de ArcGIS de estudiante, y éste se instale a través de ArcGIS.

[23] Dependiendo de los dispositivos conectados en el PC, la letra del disco puede variar
[24] http://en.wikipedia.org/wiki/IDLE_%28Python%29

6.2. Processing

Para instalar Processing[25], hay que acceder a su página y en la parte izquierda se hace clic donde pone "*Download*". A continuación, se accede a una página donde se puede realizar una donación a la Fundación de Processing, y posteriormente se descarga el programa. Para la creación de DFTrail Running se usa la versión 2.0.3 para Windows 64-bit.

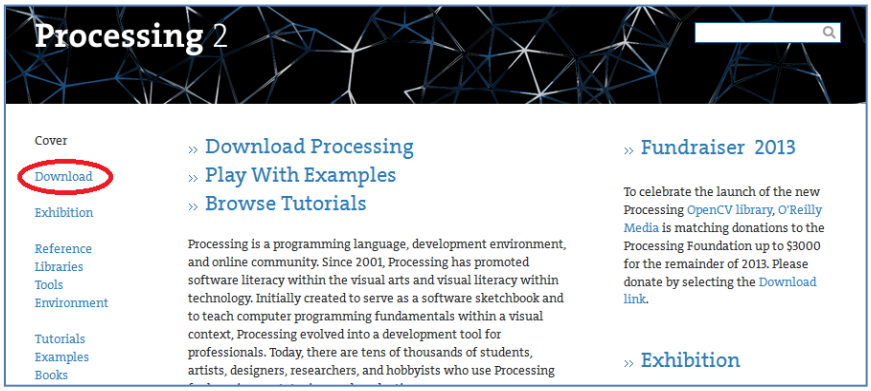

Imagen 92 - Página web Processing

Una vez ha terminada la descarga, el archivo se descomprime en Documentos (*C:\Users\David\Documents\Processing*) y se crea un acceso directo al escritorio del .exe para ejecutar Processing.

Ahora se procede a instalar las herramientas de Android, se descarga el Android SDK[26], se desplaza hacia abajo la página web y se pincha "*USE AN EXISTING IDE*", se presiona sobre el botón "*Download the SDK Tools for Windows*", se aceptan los términos y condiciones y comienza la descarga.

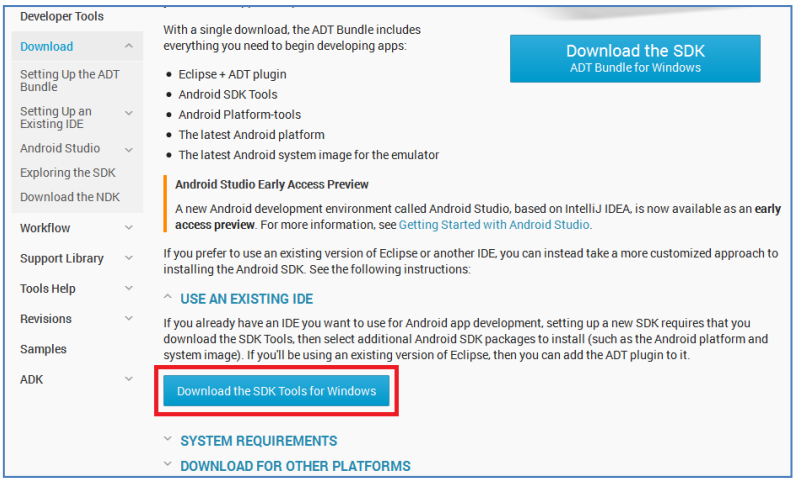

Imagen 93 - Descarga SDK Tools

[25] http://processing.org/
[26] http://developer.android.com/sdk/index.html

Una vez descargado, se hace doble clic sobre el ejecutable para iniciarlo, el instalador verificará la máquina y las herramientas necesarias, como Java SE Development Kit (JDK)[27] y las instalará si fuera necesario. Android SDK Tools se ubica dentro de archivos de programa ("*D:\Archivos de programa\Android\android-sdk*").

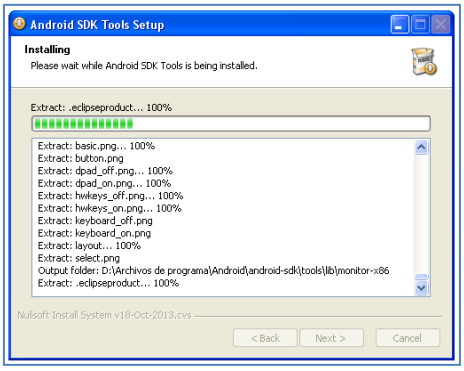

Imagen 94 - Instalación SDK Tools

Cuando se termina la instalación, se abre Android SDK Manager, y se configura. Se marca la casilla, si no lo estuviera, "*Android SDK Plataform-tools*", ya que por defecto viene seleccionada. Ahora se despliega "Android 2.3.3 (API 10)" y se selecciona "*SDK Plaform" y "Google APIs*[28]". Más abajo en el apartado de "*Extras*", se selecciona si no lo estuviera "*Google USB Driver*", que también viene seleccionado por defecto. Realizado todos los pasos, se presiona sobre el botón "*Install 4 packages…*"

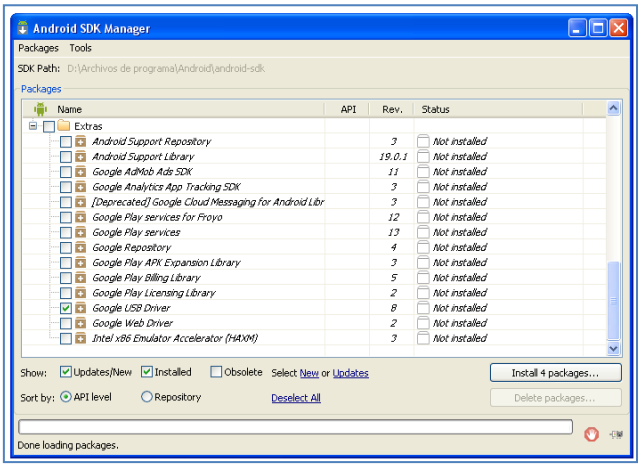

Imagen 95 - Android SDK Manager

Para finalizar, se ha de enlazar Processing con Android SDK. Cuando se ejecuta Processing a la parte derecha, se ha de activar el módulo Android. La primera vez que

[27] http://www.oracle.com/technetwork/es/java/javase/downloads/jdk7-downloads-1880260.html
[28] A partir de la versión de Processing 2.0 ya no haría falta incluirlo

se ejecute, pregunta si se tiene instalado Android SDK, se selecciona en la opción "*YES*" y se elige el directorio donde se instaló Android SDK.

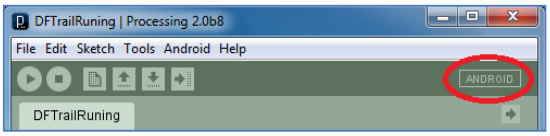

Imagen 96 - Módulo Android

Si se quiere ejecutar los programas desarrollados en el *Smartphone*, sin utilizar el emulador, hay que configurar un apartado en el dispositivo móvil, para que se puedan instalar las aplicaciones y poder probarlas. Para ello, se accede "*Ajustes / Opciones de desarrollador*"[29] y se selecciona "*Depuración de USB*".

Imagen 97 - Depuración USB

6.3. Eclipse

Instalar Eclipse para el desarrollo de aplicaciones móviles basadas en Android, consta de cuatro componentes:

- Java Runtime Environment (JRE)
- Eclipse IDE for Java Developers
- Android SDK
- Android Development Toolkit-ADT

Se van a descargar los archivos necesarios para realizar la instalación. Primero se accede a la página de java y se descarga Java Runtime Environment (JRE)[30], apretando el botón rojo. Una vez descargado el archivo, se va a la carpeta de descargas y se instala.

[29] Dependiendo de la versión de Android la ruta a seguir puede variar
[30] http://java.com/es/download/windows_xpi.jsp?locale=es

Imagen 98 - Descarga Java

Después accediendo a su página, se descarga Eclipse IDE for Java Developers[31], se selecciona el tipo de sistema operativo y en la siguiente pagina se descarga el archivo desde el botón verde.

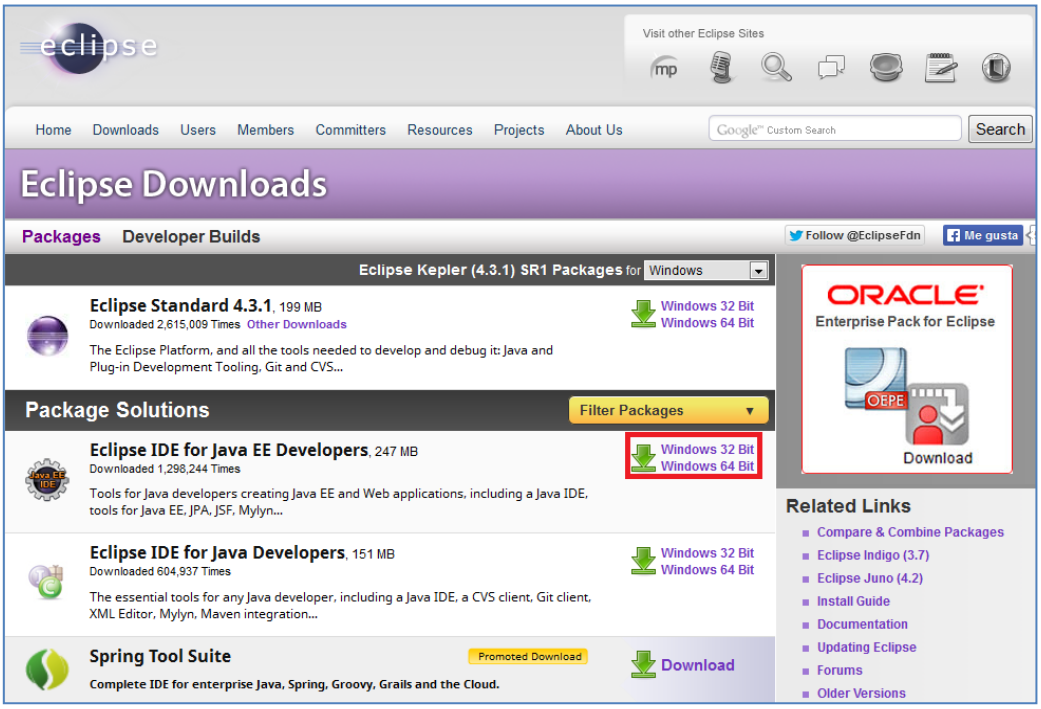

Imagen 99 - Descarga Eclipse IDE for Java Developers – I

[31] http://www.eclipse.org/downloads/

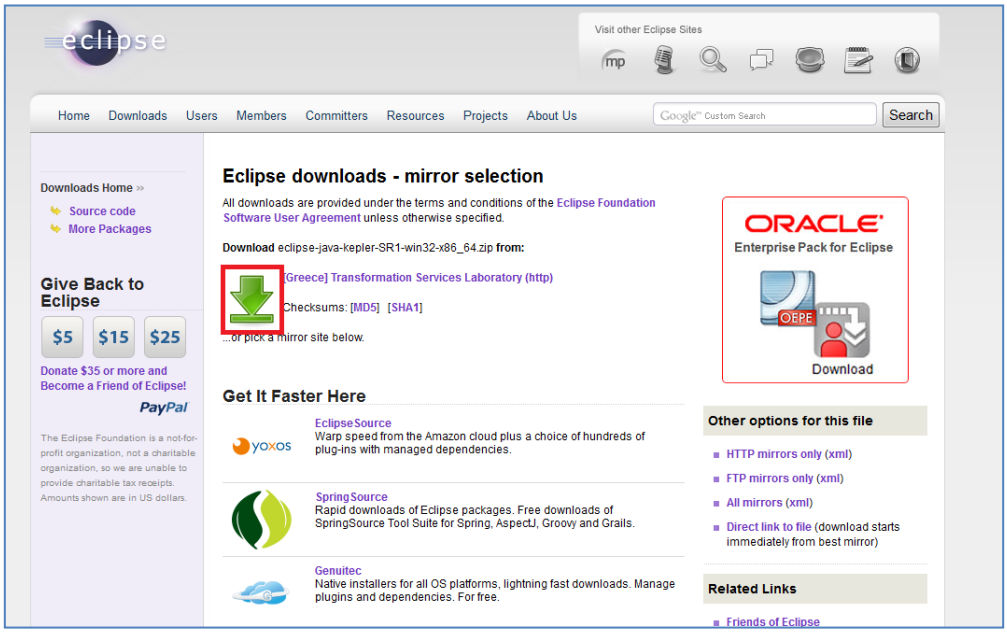

Imagen 100 - Descarga Eclipse IDE for Java Developers - II

Cuando haya terminado, se crea una carpeta en C, con el nombre Android (*C:\Android*) y se descomprime el archivo descargado en dicha carpeta.

A continuación se descarga Android SDK[32], se presiona sobre "*Download the SDK*", se aceptan los términos y condiciones, se selecciona el tipo de sistema y comienza la descarga. Una vez terminada, se descomprimirá en la carpeta creada anteriormente de Android.

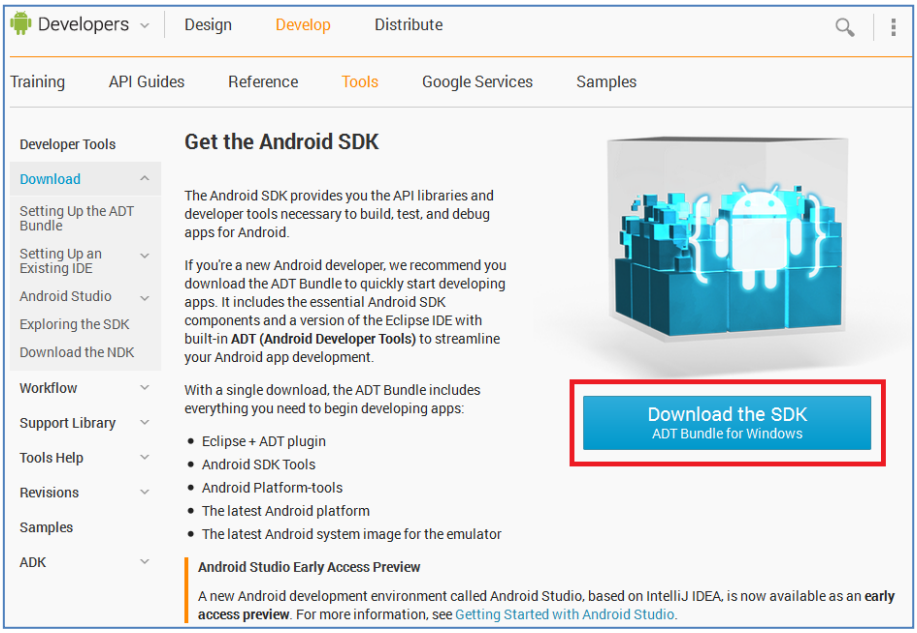

Imagen 101 - Descarga Android SDK

[32] https://developer.android.com/sdk/index.html

Ahora dentro de la carpeta "*C:\Android*", se crea otra carpeta con el nombre *workspace*, se inicia el SDK y se establece el *workspace*, carpeta de trabajo donde se guardaran todos los proyectos. Una vez iniciado, se accede al SDK Manager, a través de él, se descargan los ficheros necesarios. Para el desarrollo de TraLoc, se han descargado los ficheros desde la versión 4.0 hasta la actual (4.4). Después de descargar e instalar los ficheros, se cierra el SDK.

Por último, en el proceso de instalación, hay que instalar el Android Development Toolkit. Para ello, se abre Eclipse (kepler), dentro de él, se accede a la parte de arriba, "*Help / Install New Software*".

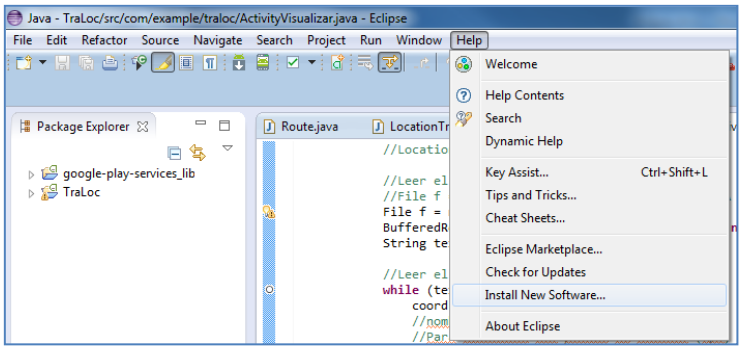

Imagen 102 - Install new software

A continuación, se pulsa el botón "*ADD*" y en *Name* se pone un nombre, por ejemplo Android y en *Location* se coloca "*http://dl-ssl.google.com/android/eclipse/*".

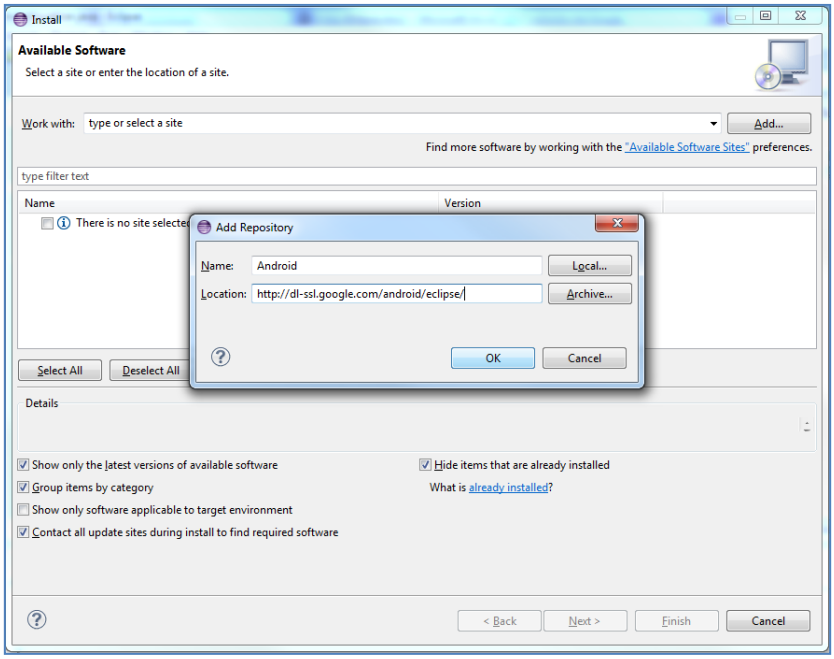

Imagen 103 - Añadir repositorio Android

Después de añadirse, aparecen las herramientas de desarrollo, se seleccionan y se instalan, para finalizar se reinicia Eclipse.

Para comprobar que Eclipse haya localizado el SDK de Android, se ejecuta Eclipse, se abren las preferencias generales y en la pestaña Android, se ha de observar que la ruta del SDK, sea igual a la carpeta descomprimida, sí no es así, habría que configurar la ruta.

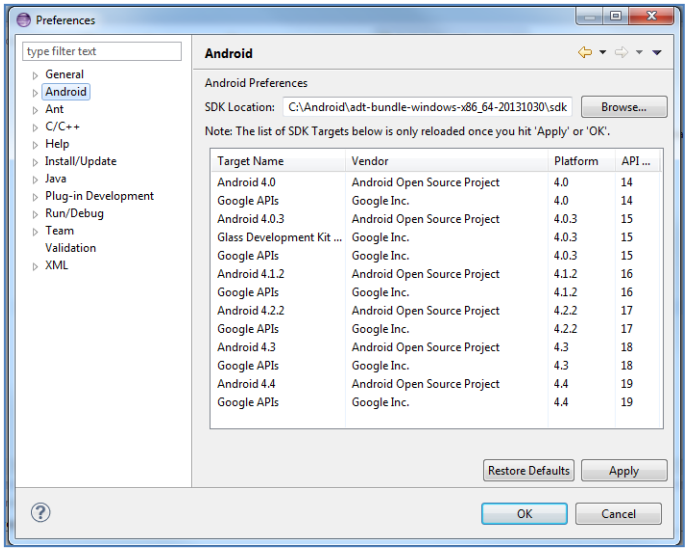

Imagen 104 - Ruta SDK

Si se quiere ejecutar los programas desarrollados en el *Smartphone*, sin utilizar el emulador, hay que configurar un apartado en el dispositivo móvil, para que se puedan instalar las aplicaciones y poder probarlas. Para ello, se accede "*Ajustes / Opciones de desarrollador*"[33] y se selecciona "*Depuración de USB*".

Imagen 105 - Opciones de desarrollador

[33] Dependiendo de la versión de Android la ruta a seguir puede variar

7. Bibliografía

LIBROS & TRABAJOS FINAL DE CARRERA:

FERRIL, P. *Pro Android Python with SL4A*, 2011, Apress. ISBN: 978-1-4302-3570-5

TOMÁS GIRONÉS, J. *El gran libro de Android*, 1ª Edición, 2011, Valencia, Marcombo S.A. ISBN: 978-84-267-1732-0

CARDONA CABALLERO, F. *Diseño de una aplicación para la captura de datos geográficos en dispositivos Android usando la API de Google, 2013*, Valencia, Universidad Politécnica de Valencia, Escuela Técnica Superior de Ingeniería Geodésica, Cartográfica y Topográfica.
Director: JOAQUÍN GASPAR MORA NAVARRO

SITIOS WEB:

SL4A

Tutorial 1 – Getting started in Python.
https://github.com/mapnik/mapnik/wiki/GettingStartedInPython

Android-scripting. http://code.google.com/p/android-scripting/

Android-scripting, API Reference. http://code.google.com/p/android-scripting/wiki/ApiReference

SL4A API Help. http://www.mithril.com.au/android/doc/index.html

ERNESTO CRESPO, *Widgets en Android con python. Parte 9 (selección multiple)*, 2011. http://blog.crespo.org.ve/2011/04/widgets-en-android-con-python-parte-9.html

DIEGO OCAMPO, *PyMaps – Mostrando un google maps for dummies*, 2011. http://diegodevelop.blogspot.com.es/2011/06/pymaps-mostrando-un-google-map-for.html

pymaps.http://code.google.com/p/pymaps/source/browse/trunk/pymaps.py

GOOGLE DEVELOPERS, *Guía para desarrolladores de la versión 2 del API de Google Static Maps*.
https://developers.google.com/maps/documentation/staticmaps/?hl=es#Usage

ALEXANDER OLIVARES, *Python en Android + script de regalo SMS/GPS/SMS*, 2010.http://www.web-aox.com/archives/200

TATIANA AL-CHUEYR, *Introducción a SL4A*.
http://revista.python.org.ar/5/es/html/desarrollo-de-aplicaciones-moviles-para-android-con-python.html

ERNESTO CRESPO, *Aplicación que muestra las coordenadas del Celular en google maps en el escritorio*, 2011.http://blog.crespo.org.ve/2011/01/aplicacion-que-muestra-las-coordenadas.html

ERNESTO CRESPO, Tomar *una foto desde Android con un temporizador desde python*, 2011.http://blog.crespo.org.ve/2011/04/tomar-una-foto-desde-android-con-un.html

tracker.py. https://github.com/njdevil/SL4A-gps-tracker/blob/master/tracker.py

googlemaps 1.0.2. https://pypi.python.org/pypi/googlemaps/

JOHN KLEINT, *googlemaps – Google Maps and Local Search APIs in Python*. http://py-googlemaps.sourceforge.net/

livemap.py. https://gist.github.com/dbr/3304597

Making maps with Python, 2012. http://pyvideo.org/video/672/making-maps-with-python

PROCESSING

Getting Started in Processing, unfolding. http://unfoldingmaps.org/tutorials/getting-started-in-processing.html

Using Processing to read GPS data and displaying a map, 2012. http://www.dfrobot.com/community/using-processing-to-read-gps-data-and-displaying-a-map.html

Processing for Android: Usando el GPS, 2010. http://www.marlonj.com/blog/2010/12/processing-for-android-usando-el-gps/

GPS Data. http://wiki.processing.org/w/GPS_Data

Android. http://wiki.processing.org/w/Android#What_about_Multi-touch.2C_GPS.2C_SMS.2C_Compass.3F

GPS!!. http://forum.processing.org/topic/gps

DANIEL SAUTER, *KETAI Sensor Library for Processing*, 2012. http://danielsauter.com/display.php?project_id=113

GABY GARCÍA CÁRDENAS, *Processing*, 2012. http://computointegradoits.blogspot.com.es/2012/05/processing.html

Ketai. http://code.google.com/p/ketai/

Reference. http://processingjs.org/reference/

apwidgets. apwidgets. http://code.google.com/p/apwidgets/

OSCAR GONZÁLEZ, *Como programar para Android con Processing*, 2010. http://blog.bricogeek.com/noticias/tutoriales/como-programar-para-android-con-processing/

ECLIPSE

Android Asset Studio. http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html

Android Developer. http://developer.android.com/index.html

Android – Botón imagen con fondo transparente, 2013. http://janmi.com/android-boton-imagen-con-fondo-transparente/

GABRIEL KOUYOUMDJIAN, *Android faqs (II): Agregar una imagen a un botón*, 2012. http://gkcodelog.blogspot.com.es/2012/07/android-tips-ii-agregar-una-imagen-un.html

IGNACIO IGLESIAS, *Android (IV): Diseño de layouts o interfaces*, 2011. http://www.ignacionario.com/2011/03/android-iv-diseno-de-layouts-o.html

SERGIO MARTÍNEZ, *Aprendiendo Android: Creando Layouts*, Parte I, 2010. http://www.elandroidelibre.com/2010/11/aprendiendo-android-creando-layouts-parte-1.html

SALVADOR GÓMEZ OLIVER, *Curso Programación Android – Índice de Contenidos*, 2011. http://www.sgoliver.net/blog/?page_id=3011

Subrayado de un textview, 2012. http://www.sakidroid.com/index.php/es/com-phocadownload-info/android/pildoras/91-subrtext

Códigos de colores HTML. http://html-color-codes.info/codigos-de-colores-hexadecimales/

FRANCISCO, *Truco Android: crear fondos basados en xml*, 2010. http://francho.org/2010/04/27/truco-android-crear-fondos-basados-en-xml/

Enviar un archivo adjunto a un email, 2009. http://www.android-spa.com/viewtopic.php?t=1845

Evitar que la pantalla del dispositivo se apague. http://ultra-lab.net/code/wiki:evitar_que_la_pantalla_se_apague

GONZALO DE CÓRDOBA, *Como realizar acción tras esperar unos segundos con postDelayed*, 2011. http://www.tutorialandroid.com/basico/como-realizar-accion-tras-esperar-unos-segundos-con-postdelayed/

KATE RIVAS, *Ejemplo de Paso de Actividades en una aplicación para Android*, 2013. http://proyectosbeta.net/2013/04/ejemplo-de-paso-de-actividades-en-una-aplicacion-para-android/

Cerrar todas las Activity, 2012. https://groups.google.com/forum/#!topic/desarrolladores-android/ai8342uPxQg

Enable and disable Button according to the text in EditText in Android, 2011. http://stackoverflow.com/questions/8225245/enable-and-disable-button-according-to-the-text-in-edittext-in-android

VÍCTOR CUERVO, *Saber si existe un fichero con Java*, 2007. http://lineadecodigo.com/java/saber-si-existe-un-fichero-con-java/

Centrar mapa y ajustar zoom Google Maps, 2009. http://codeofworld.blogspot.com.es/2009/03/centrar-mapa-y-ajustar-zoom-google-maps.html

¿Cómo obtener la selección de un Spinner?, 2010. http://www.android-spa.com/viewtopic.php?t=6856

DIANNE HACKBORN, *Back and other hard keys: three stories*, 2009. http://android-developers.blogspot.com.es/2009/12/back-and-other-hard-keys-three-stories.html

Google Maps Android API v2 detect long click on map and add Marker not working, 2013. http://stackoverflow.com/questions/16097143/google-maps-android-api-v2-detect-long-click-on-map-and-add-*Marker*-not-working

How to make a copy of a file in Android?, 2012. http://stackoverflow.com/questions/9292954/how-to-make-a-copy-of-a-file-in-android

VÍCTOR CUERVO, *Borrar un fichero con Java*, 2008. http://lineadecodigo.com/java/borrar-un-fichero-con-java/

GPX: the GPS Exchange Format. http://www.topografix.com/gpx.asp

Google Developers Console. https://code.google.com/apis/console/

Calling system settings from an Android app – GPS example, 2010. http://www.helloandroid.com/tutorials/calling-system-settings-android-app-gps-example

Hacer vibrar Android (SDK), 2010. http://www.goltratec.com/wp/2010/08/03/hacer-vibrar-android-sdk/