Proyecto Fin de Máster

Máster en Electrónica, Tratamiento de Señal y Comunicaciones

SPADE - Smartphone Platform for Acquisition of Data Experiment

Autor: Ana Haro Pérez

Tutor: Fernando Muñoz Chavero

Dep. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Máster
Máster en Electrónica, Tratamiento de Señal y Comunicaciones

SPADE - Smartphone Platform for Acquisition of Data Experiment

Autor:

Ana Haro Pérez

Tutor:

Fernando Muñoz Chavero

Profesor titular

Dep. de Ingeniería Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Máster: SPADE - Smartphone Platform for Acquisition of Data Experiment

Autor: Ana Haro Pérez

Tutor: Fernando Muñoz Chavero

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

Índice

1. Introducción

1.1 ¿Qué es SPADE?

SPADE es un experimento llevado a cabo por un equipo multidisciplinar de estudiantes de Ingeniería de la Universidad de Sevilla que ha sido seleccionado por el programa *REXUS/BEXUS* en la campaña *BEXUS 20/21*.

El experimento ***SPADE*** *(Smartphone Platform for Acquisition of Data Experiment)* pretende diseñar y construir una plataforma inalámbrica de adquisición de datos basada en hardware comercial. Dicha plataforma consiste en una red de sensores inalámbricos y una unidad de medición basada en smartphones, la cual irá embarcada en un globo estratosférico *(BEXUS 21)* que ascenderá hasta unos 30 kilómetros de altura.

El objetivo del experimento es analizar el rendimiento de la plataforma en condiciones estratosféricas como primera aproximación para futuras plataformas de sensores inalámbricos embarcadas en misiones de exploración espacial. El resultado del experimento será de gran utilidad por extrapolación para la industria aeronáutica, comercial y militar.

1.2 ¿Cómo surgió?

La iniciativa del proyecto ***SPADE*** surgió a través de la asociación *LEEM* de la escuela de Ingenieros de la Universidad de Sevilla.

Profesores que había presentado anteriormente propuestas para el programa *REXUS/BEXUS* contactaron con miembros de ésta asociación para realizar una nueva propuesta.

Se empezó a formar el equipo con miembros de la asociación, y éstos a su vez buscaron, con ayuda de los profesores colaboradores, los perfiles que necesitaban para cubrir las necesidades del proyecto.

1.3 El equipo *SPADE*

-María de la Cruz Sánchez Gómez:

Estudiante de Ingeniería aeroespacial y presidenta de *LEEM-US*. Team Leader en el proyecto ***SPADE*** y al cargo de las tareas de outreach.

-María Teresa Rodríguez Tabares:

Estudiante de Ingeniería aeroespacial. Al cargo del diseño del equipamiento electrónico para el proyecto

-Beatriz Carretero Parra:

Estudiante de Ingeniería de Telecomunicaciones. Al cargo del desarrollo de software para la estación de tierra

-Pedro María Antúnez Herrera:

Estudiante de Ingeniería Industria. Responsable de la adaptación del equipamiento

-Diego Almodóvar López:

Estudiante de Ingeniería Industrial. Al cargo del hardware y las pruebas

-Ana Haro Pérez:

Ingeniero en Informática y estudiante del Máster en Electrónica, Tratamiento de Señal y Comiunicaciones. Al cargo del desarrollo del software para los smartphones y de la conexión con el *E-Link*

1.4 Colaboradores

Para la realización de éste proyecto hemos contado con numerosos colaboradores, tanto los que nos han proporcionado aportación económica, material e instalaciones, como el personal docente que nos ha ayudado y apoyado a lo largo de todo el proyecto

1.4.1 Personal docente:

-Miguel Ángel Aguirre:

Ingeniero Industrial y Doctorado en la Facultad de Ingeniería de la Universidad de Sevilla, en el Departamento de Ingeniería Electrónica, Sistemas y Automatización.

Actualmente imparte clases de Diseño de Microelectrónica Digital y Diseño de Lógica Programable en el Departamento de Ingeniería Electrónica de la Universidad de Sevilla, como Profesor Asistente.

Es autor de más de 40 publicaciones en revistas y conferencias IEEE y varias patentes.

El Dr. Aguirre lidera un grupo de investigación de la Universidad de Sevilla cuya actividad se centra en el diseño fiable de sistemas embebidos sobre sistemas programables en chip y los efectos de radiación en circuitos integrados de dispositivos digitales y FPGAs. Desarrolló el sistema de inyección de fallas mediante la reconfiguración parcial de las SRAM-FPGA denominadas FT-UNSHADES, que actualmente utiliza la Agencia Espacial Europea. También está involucrado en el desarrollo de sistemas integrados para el procesamiento de video en tiempo real aplicados en máquinas de selección de la industria agroalimentaria.

-Hipolito Guzman-Miranda:

Obtuvo su licenciatura en Ingeniería de Telecomunicaciones, con especialización en Electrónica.

Ingresó en el Departamento de Ingeniería Electrónica de la Universidad de Sevilla en el mismo año y obtuvo su doctorado (con honores y mención europea) en la misma universidad en 2010.

Actualmente trabaja como profesor asistente en la Universidad de Sevilla, donde imparte clases de Microelectrónica, Sistemas de comunicaciones electrónicas y diseño de lógica programable, y un curso de maestría en microelectrónica digital.

Está involucrado en el desarrollo, mantenimiento y evolución del sistema de inyección de fallas FT-Unshades2, un proyecto de la Agencia Espacial Europea que se está convirtiendo en la solución preferida para la evaluación temprana de la sensibilidad a la radiación de los diseños digitales. Ha publicado más de 10 artículos de la revista JCR y más de 30 contribuciones a la conferencia. Es miembro del Comité Técnico de Sistemas Electrónicos en Chip de la IEEE.

-Rogelio Palomo Pinto:

Doctor en Física. Universidad de Sevilla

-Fernando Muñoz Chavero:

Doctor en Ingeniería de Telecomunicaciones, coordinador del Máster en Electrónica, Tratamiento de Señal y Comunicaciones. Universidad de Sevilla.

1.4.2 Patrocinadores

REXUS/BEXUS Programme

rexusbexus.net/
www.facebook.com/rexusbexus

Universidad de Sevilla (US)

www.us.es/

Escuela Técnica Superior de Ingenieros (ETSI)

www.etsi.us.es/

Asociación de Investigación y

Cooperación Industrial de Andalucía (AICIA)

aicia.es/

Campus de Excelencia

Internacional Andalucía Tech

www.andaluciatech.org/

German Aerospace Centre DLR

(Deutsches Zentrum fuer Luft- und Raumfahrt)

www.dlr.de/

Swedish National Space Board (SNSB)

www.snsb.se

European Space Agency (ESA)

www.esa.int/ESA
www.esa.int/Education

SSC Group

www.sscspace.com/

Center of Applied Space Technology

and Microgravity (ZARM)

www.zarm.uni-bremen.de/

EuroLaunch

www.eurolaunch.org/

Adevice Solutions

www.adevice.es/
twitter.com/adevice

Alter Technology (TÜV NORD GROUP)

www.altertechnology.com/
www.facebook.com/altertechnology

Cátedra Airbus Group de Estudios Aeronáuticos

www.airbusgroup.com

Saft Baterías (SAFT Group)

www.saftbaterias.es
www.saftbatteries.com
twitter.com/Saft_batteries

Instituto de Microelectrónica de Barcelona IMB-CNM

www.imb-cnm.csic.es/

Consejo Superior de Investigaciones Científicas (CSIC)

www.csic.es/

2. Fases del proceso de selección

En el siguiente mapa se puede ver dónde se sitúa cada una de las instituciones que se visitaron durante las diferentes fases del proyecto

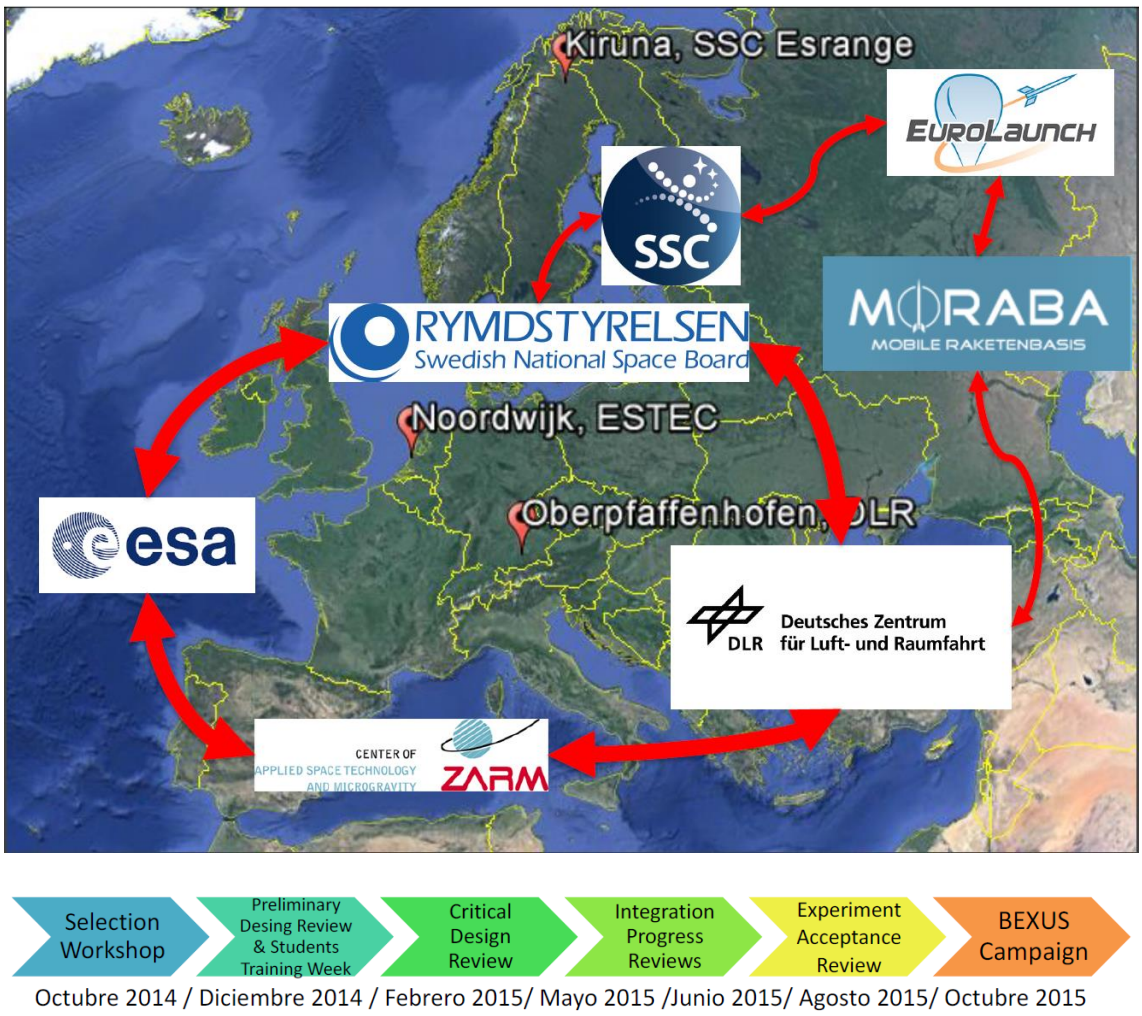


Fig. 2.1: Mapa de instituciones visitadas

2.1 Selection workshop

Fue la primera defensa del proyecto, que se hizo ante el panel de expertos en ESTEC (ESA), en Holanda de 2-4 dic. 2014

Participaban 16 equipos europeos (8 *BEXUS* y 8 *REXUS*) de los cuales sólo 8 equipos serían seleccionados

2.2 Preliminary Design Review (PDR) & Students Training Week (STW)

PDR 1ª revisión del diseño con panel expertos

- STW Entrenamiento intensivo como proyecto real: gestión de proyectos, modo trabajo, competencias, documentación…

  En Alemania: 9-13 febrero 2015

  Único evento con los equipos *RX/BX* (europeos y alemanes): 6 *BEXUS* y 6 *REXUS*

2.3 Critical Design Review (CDR)

ESTEC (ESA): Holanda, 20-21 Mayo 2015

- Última defensa antes de proceder a la fabricación y testeo del experimento

2.4 Integration Process Review (IPR)

Los expertos de la ESA, entre ellos nuestro mentor Hanno, nos visitan para ver los avances de la fabricación del experimento

- Universidad de Sevilla 27 julio 2015

2.5 Experiment Acceptance Review (EAR)

Jianning Li, ingeniero de la Agencia Espacial Sueca, nos visita para comprobar que nuestro experimento es apto para el vuelo

Universidad de Sevilla 4 septiembre 2015

2.6 Campaña de lanzamiento

Octubre de 2015 en Esrange, Kiruna

La fase definitiva, montaje, testeo y lanzamiento

Recuperación del experimento y análisis rápido de los resultados.

Posteriormente, cuando llegamos a Sevilla, tuvimos que realizar el procesamiento y análisis posterior de los resultados obtenidos.

Durante todo el proceso, tuvimos que ir generando un documento de especificaciones, restricciones, presupuesto, etc. En dicho documento se recoge toda la información referente a cada una de las fases descritas.

3. Organización del proyecto

Ésta es la función que tenía asignada nuestra Team Leader.

Para el desarrollo del proyecto se llevó a cabo una distribución por funcionalidad, atendiendo a las competencias de cada miembro, y que puede verse en el siguiente esquema:

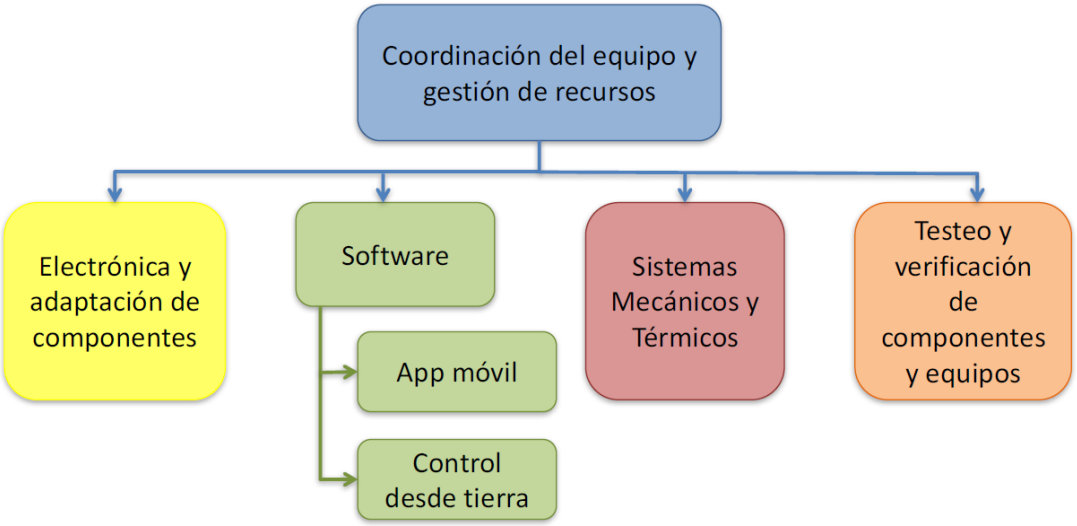


Fig. 3.1: Bloques del proyecto

4. El experimento

4.1 Trasfondo científico y técnico

En el sector espacial, existe un gran interés en la formación de misiones de vuelo en enjambre con pequeños satélites, los cuales se espera que tengan importantes beneficios en cuanto a costes y despliegue del sistema, capacidad y redundancia de sistemas.

A día de hoy, las comunicaciones en el espacio sólo se han podido llevar a cabo mediante enlaces entre satélites (ISL), principalmente basados en los protocolos de los propietarios.

Una idea es adoptar los ya bien conocidos estándares de comunicación terrestres, como IEEE 802.11. [8]

Por ejemplo, ZigBee, que se basa en el estándar IEEE 802.15.4. Ha sido probado para un amplio rango de aplicaciones industriales y ofrece la oportunidad de desarrollar una infraestructura inteligente con éstos protocolos y cambios mínimos.

Wireless sensor networks (WSNs) son redes multi-hop self-organizing, que incluyen un gran número de nodos que integran medida, procesamiento de datos y comunicaciones inalámbricas para recoger y procesar información. El interés en usar space-based wireless sensor networks (SB-WSNs) está aumentando. Esto aplica el concepto de redes inalámbricas de sensores terrestres al espacio.

4.2 Los sensores inalámbricos

WSNs es una tecnología comercial que tiene el potencial de mejorar la recolección de datos y beneficios científicos para misiones espaciales futuras usando los estándares terrestres de comunicación.

Los ingenieros de la ESA se encuentran trabajando también en ésta dirección, ya que planean aplicar el mismo sistema de comunicaciones inalámbricas que conectan los dispositivos comerciales, como teléfonos móviles u ordenadores portátiles para desarrollar una nueva generación de hardware espacial en red.

Finalmente, los smartphones son pequeñas unidades de procesamiento de bajo coste, que incluye una cantidad aceptable de sensores. Éstos dispositivos COTS podrían modificarse como procesadores o unidades de control para naves espaciales.

Son fáciles de usar, y las oportunidades que ofrecen hacen que sea más interesante tenerlos en cuenta para la creación de la nueva generación de dispositivos espaciales.

4.3 Objetivos del experimento

4.3.1 Objetivos principales

El experimento incluye dos objetivos técnicos principales:

Obj. 1.1: Estudiar el comportamiento de una red de sensores inalámbrica que incluya la comunicación desde y hacia una unidad central cuyas condiciones iban a ser externas al *BEXUS* (condiciones estratosféricas, lo que implica bajas presiones y aún más baja temperatura y condiciones que no son fácilmente reproducibles en tierra, con es la radiación)

Obj. 1.2: Testear el comportamiento de componentes comerciales, como son los smarthphones, sensores de los nodos y baterías comerciales como unidad central para éste tipo de plataforma en condiciones difíciles

El experimento incluye una meta científica principal:

Obj. 1.3: Medir el flujo de partículas de rayos cósmicos (partículas de energía) en la estratosfera, ya que afectan a los dispositivos electrónicos. Queremos monitorizar y recoger la cantidad de partículas de rayos cósmicos que atraviesan nuestro experimento como parámetro que afecta a la electrónica del sistema

4.3.2 Objetivos secundarios

A parte del experimento en sí, queremos tener en cuenta los siguientes objetivos secundarios:

Obj. 2.1: Probar en el experimento la aplicación '*Crayfis'*, que puede recoger partículas de rayos cósmicos en tierra. La idea es probar que dicha aplicación obtiene buenos resultados obteniendo ésta medida de rayos cósmico en la estratosfera.

Obj. 2.2: Probar la resistencia de las cajas protectoras que se han montado para albergar los dispositivos.

4.3.3 Éxito de la misión

Nuestro experimento comprobará si es posible reemplazar la tecnología espacial por tecnología comercial, estudiando el comportamiento de la plataforma y la exactitud de los datos obtenidos.

Tanto si el comportamiento es bueno como si es malo, los resultados obtenidos serán de gran importancia.

En caso de fallo, tendremos los datos necesarios para probar cuándo y en qué condiciones se ha producido éste.

Con respecto a las partículas de rayos cósmicos, la aplicación de *CRAYFIS*, cuenta con una base de datos que compara las medidas tomadas con el contenido de ésta, determinando si lo que se ha leído es un rayo cósmico o no, por lo que podremos saber si los resultados son tan buenos como en tierra.

4.4 Concepto del experimento

El experimento es una plataforma de recogida de datos en tiempo real para misiones de exploración estratosférica, y comprobará si es posible sustituir la tecnología espacial por tecnología comercial

El layout de la plataforma será el siguiente:

-Tres nodos de sensores (SN)

-Un nodo concentrador (CN).

-Dos smartphones con sensores internos y cámara (SD1/SD2).

-Un router

-Una estación de tierra (GS): un portátil

Con estos elementos, el experimento tendrá tres bloques funcionales diferenciados:

Bloque 1: red inalámbrica de sensores, que incluye los tres nodos sensores y el nodo concentrador, formando una red en estrella ZigBee.

El concentrador guardará la información de la red y la enviará al router.

Bloque 2: consiste en una unidad central (CU) compuesta por los dos smartphones y el router. Se utilizarán adaptadores para conectar de forma cableada los smartphones al router, ya que una conexión por wifi podría interferir en los protocolos de comunicación internos del *BEXUS*.

SD1, SD2 tomarán medidas de forma independiente. Se han incorporado dos dispositivos para asegurar la redundancia en caso de fallo de alguno de los dispositivos.

El router mandará toda la información al *E-Link*.

Bloque 3: estación de tierra (GS). Recibirá los datos tomados en tiempo real por todos los dispositivos y los almacenará para su posterior análisis. También dispondrá de comandos para poder realizar configuración remota en caso de que fuese necesario.

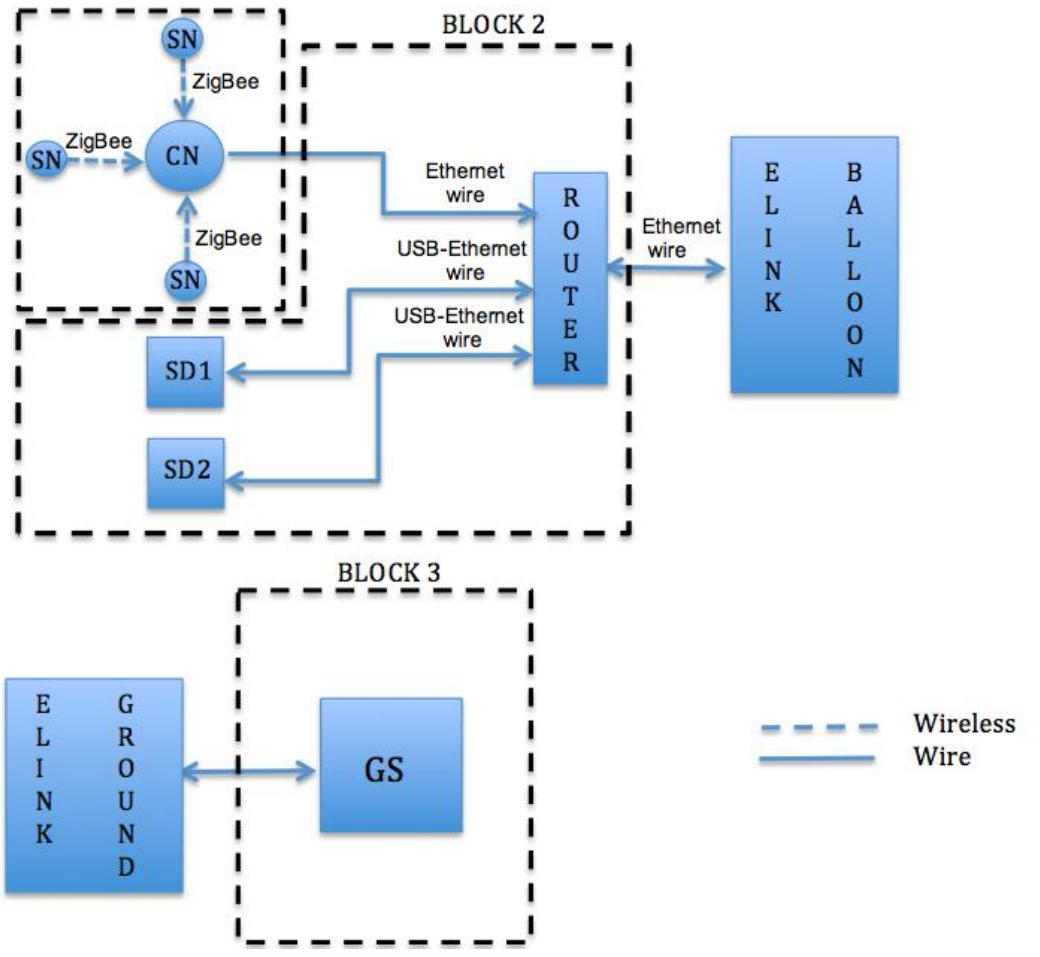


Fig. 4.4-1: Estructuración del proyecto 1

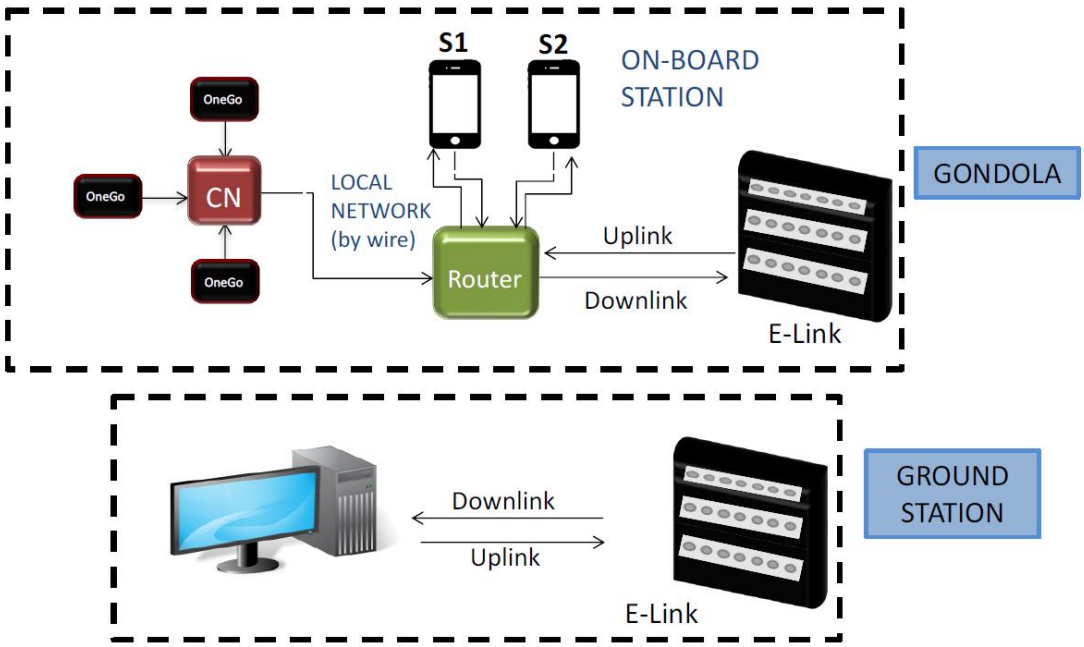


Fig. 4.4-2: Estructuración del proyecto 2

4.5 Requisitos y restricciones

Antes de comenzar la implementación del proyecto, tuvimos que definir requisitos y restricciones para acotar los posibles problemas y confirmar la viabilidad del proyecto.

4.5.1 Requisitos funcionales

Tabla 4.5.1-1: Requisitos funcionales

F.1:	El experimento medirá la temperatura fuera de la góndola durante todo el vuelo del globo usando un nodo sensor.
F.3:	El experimento medirá la concentración de CO2 fuera de la góndola durante todo el vuelo del globo utilizando un nodo sensor.
F.4:	El experimento medirá las partículas de rayos cósmicos durante todo el vuelo del globo.
F.5:	El experimento medirá la aceleración del globo durante todo el vuelo usando dos sensores de acelerómetro de smartphone.
F.6:	El experimento medirá la velocidad angular de rotación del globo durante todo el vuelo.
F.7:	El experimento medirá la posición del globo durante todo el vuelo.

Las medidas de partículas de rayos cósmicos serán tomadas por las cámaras de los smartphones, con la aplicación *CRAYFIS*.

4.5.2 Requisitos de desempeño

Tabla 4.5.2-1: Requisitos de desempeño

Respondiendo a F.1	
P.1:	El rango de medición de la temperatura operacional estará entre -80 y +50 grados Celsius.
P.2:	La medición de la temperatura fuera de la góndola se realizará con una precisión de +/- 1 grado Celsius.
Respondiendo a F.3	
P.5:	El rango de medición de la concentración de CO2 operacional estará entre 400 y 10000 ppm.
P.6:	La concentración de CO2 que se mida fuera de la góndola se realizará con una precisión de +/- 3%.
Respondiendo a F.1 - F.3	
P.7:	La temperatura y la medición de la concentración de CO2 fuera de la Góndola se realizarán a una velocidad de 1 medida cada 60 segundos.
Respondiendo a F.5	
P.9:	El rango de medición de la aceleración operacional debe estar entre 0 y +/- 20 metros por segundo cuadrado.
P.10:	La medición de la aceleración se realizará con una precisión de +/- 2 metros por segundo cuadrado.
Respondiendo a F.6	
P.11:	El rango de medición de la velocidad angular de rotación debe estar entre 0 y 70 grados por segundo
P.12:	La medición de la velocidad de rotación angular se realizará con una precisión de 10 grados por segundo.
Respondiendo a F.7	
P.13:	La medición de posición se realizará con una precisión de +/- 30 metros.
Respondiendo a F.5 - F.7	
P.20:	La medida de los sensores del teléfono inteligente se realizará a una velocidad de 1 medición cada 60 segundos.

Como nuestro objetivo es la adquisición de datos en tiempo real, estimamos que no era necesario mandar información tan frecuentemente, ya que no se iban a producir cambios significativos en un periodo corto de tiempo

4.5.3 Requisitos de diseño

Los requisitos de diseño para el proyecto ***SPADE*** son:

Tabla 4.5.3-1: Requisitos de diseño

D.1:	Los smartphones funcionarán en el perfil de temperatura del globo *BEXUS*.
D.2:	Las baterías de los smartphones deben estar cualificadas para su uso en el globo *BEXUS*.
D.3:	Las baterías de los smartphones deberán ser recargables o tener capacidad suficiente para realizar el experimento durante las pruebas previas al vuelo, la preparación del vuelo y el vuelo.
D.4:	Las baterías de los smartphones deben ser accesibles desde el exterior en 1 minuto, una vez que el experimento esté en la góndola.
D.5:	Los smartphones operarán en el perfil de presión del globo *BEXUS*.
D.6:	Los smartphones irán fijados a la caja de la Central Unit.
D.7:	Los componentes de la Red de Sensores Inalámbricos (WSN) deben operar en el perfil de temperatura del globo *BEXUS*.
D.8:	Las baterías de los componentes WSN deben estar cualificadas para su uso en un globo *BEXUS*.
D.9:	Las baterías de los componentes WSN deberán ser recargables o tener capacidad suficiente para ejecutar el experimento durante las pruebas previas al vuelo, la preparación del vuelo y el vuelo.
D.10:	Las baterías de los componentes WSN serán accesibles desde el exterior en 1 minuto, una vez que el experimento esté en la góndola.
D.11:	Los componentes de WSN funcionarán en el perfil de presión del globo *BEXUS*.
D.12:	Los componentes WSN no transmitirán en 2.4GHz
D.13:	Las cajas de la caja de WSN se fijarán bien a la góndola.
D.14:	Las cajas experimentales deben resistir el perfil de vibración del globo *BEXUS*.
D.15:	Las cajas de experimentación deberán resistir una caída de 3 metros.
D.16:	Las cajas de experimentación no deben dañar o perturbar la góndola.

D.17:	Las cajas de experimentación se fijarán bien a la góndola
D.19:	El router funcionará en el perfil de presión del globo *BEXUS*.
D.20:	El router funcionará en el perfil de temperatura del globo *BEXUS*.
D.21:	El router debe estar bien fijado a la caja del experimento.
D.22	Las cámaras de Smartphone tendrán una visión clara hacia arriba. (Respuesta a F.4)

4.5.4 Requisitos operacionales

Tabla 4.5.4-1: Requisitos operacionales

O.1:	El experimento debe poder cambiar el smartphone a cualquier modo recibiendo una señal de la estación terrestre.
O.2:	El experimento debe ser capaz de almacenar todos los datos en una memoria interna.
O.3:	El experimento podrá encender / apagar la cámara durante el vuelo.
O.4:	El experimento deberá poder reiniciar los dispositivos de la estación terrestre en caso de que alguno de ellos cuelgue.
O.5:	El experimento dejará de tomar medidas después del aterrizaje.
O.6:	El experimento deberá poder seguir enviando mediciones incluso si algún sensor no funciona.

4.5.5 Restricciones

Tabla 4.5.5-1: Restricciones

C.1:	El coste del experimento no excederá de 7000 €
C.3:	Los componentes del experimento COTS usualmente trabajan en un rango de temperatura más alto que en las condiciones *BEXUS*.
C.4:	Los nodos sensores deben realizar mediciones cada 60 segundos para optimizar la red de datos.
C.5:	La cantidad de tiempo que cada miembro puede dedicar al proyecto es limitado debido a que todos los miembros del equipo son estudiantes.
C.6:	Los agujeros para las cámaras de los smartphones deben colocarse en la parte superior de la caja de la unidad central.

5. Componentes software

5.1 Software de tierra

La parte del software de tierra consiste en dos ordenadores, uno de ellos se utilizará como backup

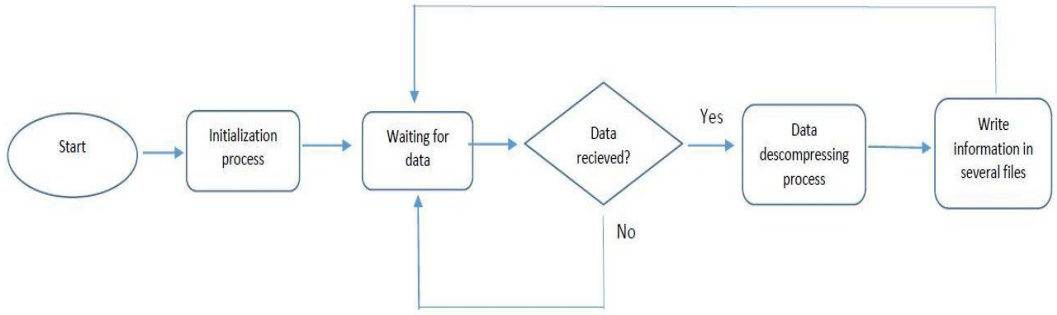


Fig. 5.1-1: Diagrama de la estación de tierra

El software utilizado está escrito en C. Recogerá todos los datos que proporcionen los dos smartphones y las tres motas a bordo de la góndola a través de la conexión con el *E-Link*, que requerirá una conexión Ethernet.

Datos esperados:

- Temperatura, CO_2 y protones obtenidos por los sensores de los nodos a bordo.
- Partículas de rayos cósmicos, obtenidas por los dos smartphones a bordo.
- Posición, aceleración y velocidad angular de rotación obtenidos por los sensores de los dos smartphones a bordo.
- ID del Smartphone que está enviando el dato

El software consistirá en cinco partes:

1. La primera parte será el software del nodo concentrador. Para esta parte del software vamos a montar un servidor FTP, así que vamos a usar el puerto 20. Ya hemos montado y probado ese servidor FTP y su comportamiento con el nodo concentrador porque Adevice nos lo ha proporcionado. Además, Adevice nos ha proporcionado una interfaz gráfica donde podremos ver durante los vuelos todas las medidas y el estado de los componentes (nodos y concentrador) en tiempo real.
2. La segunda parte será el software de los smartphones. Vamos a ejecutar dos archivos ejecutables (a.out) en paralelo. Los programas harán lo mismo, recogerán los datos recibidos de los smartphones a bordo, pero van a utilizar diferentes puertos. Cada programa utilizará un puerto TCP (enlace descendente) y un puerto TCP (enlace ascendente).

3. La tercera parte consistirá en el software para enviar órdenes de emergencia a los smartphones a bordo para cambiar entre modos o reiniciar su sistema.

4. La cuarta parte será el software para separar los diferentes tipos de datos durante el proceso. Este software consistirá en un ejecutable (a.out). Este ejecutable leerá un archivo (el archivo con todos los datos que hemos recibido durante el proceso) y separará cada tipo de datos en diferentes archivos.

5. La quinta parte será el software del estudio estadístico de paquetes perdidos que será suministrado por el Departamento de Ingeniería Electrónica de la Universidad de Sevilla.

Por lo tanto, la estructura del sowtware de la estación de tierra será:

1. Estructura del software del concentrador:

El servidor FTP recibirá los datos recogidos por los nodos sensores. La tabla que esperamos recibir es:

Tabla 5.1-1: Datos esperados del concentrador

Datos recogidos por el sensor de protones	Datos recogidos por el sensor de CO2	Datos recogidos por el sensor de temperatura	Fecha y hora de la lectura

Las muestras recogidas por los sensores a bordo se enviarán cada minuto a través de la conexión *E-Link* con la estación de tierra.

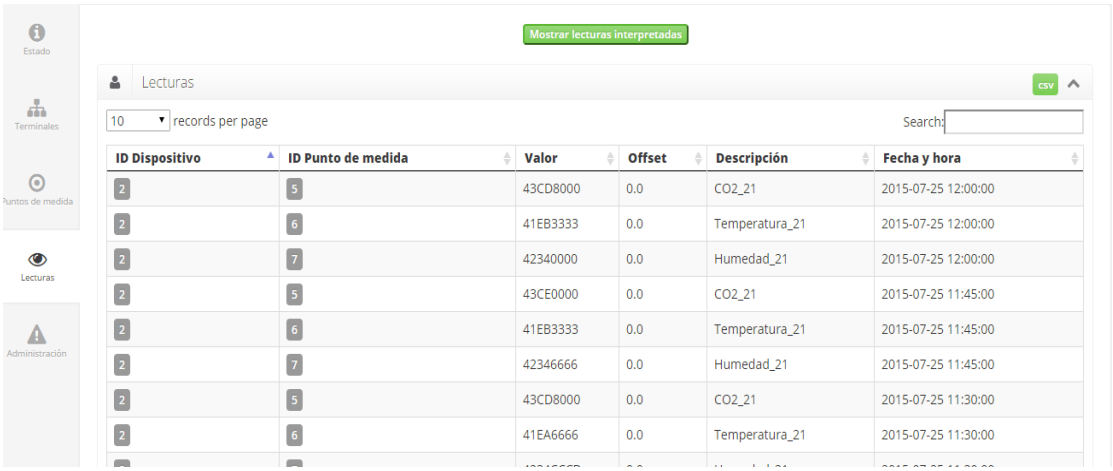


Fig. 5.1-2: Interfaz de Adevice

Esta es una imagen de la interfaz de Adevice. Esta interfaz tiene acceso al software del concentrador y muestra qué medidas ha recibido el concentrador en tiempo real.

En el lado derecho podemos ver la fecha y la hora en que recibimos esa medida. En el centro de la foto podemos ver el valor de la medida (en este caso los valores están en hexadecimal pero podemos convertirlos en decimal haciendo un clic en el botón verde en la parte superior). A la izquierda también podemos ver el ID del dispositivo (mote) y, como podemos ver, tenemos la

descripción del sensor y qué es esa medición del sensor. Todas estas medidas serán recibidas por el servidor FTP.

En esta interfaz conoceremos también el estado del concentrador y las motas.

Los posibles estados de los dispositivos son:

- Conectado: El concentrador ha detectado la mota y la mota está esperando la configuración. En el caso del concentrador significa que está funcionando normalmente.
- Conectado y configurado: Significa que la mota ha recibido la configuración del concentrador y empezará a mandar medidas
- Desconectado: Hemos perdido la conexión con la mota o el concentrador no se ha detectado todavía.

El concentrador envía trazas de configuración cada dos minutos.

Para mantener el control de toda la red de sensores, podemos enviar al concentrador unos comandos que serán muy útiles antes y durante el vuelo.

Estos comandos nos permitirán:

- Pedir al concentrador que envíe la traza a cada nodo en ese momento (no será necesario esperar dos minutos).
- Saber en ese momento qué nodo sensor está conectado a la red(podemos ver también la interfaz gráfica que hemos mostrado antes, pero a veces tarda unos minutos en mostrase, por lo que de ésta forma es más rápido)
- Cambiar la frecuencia con la que el concentrador envía la traza de configuración

Todos éstos comandos se enviarán usan Putty, que es un cliente SSH, por lo que vamos a utilizar una conexión SSH.

2. Estructura del software de los smartphones:

Vamos a ejecutar dos programas iguales en paralelo. Cada programa funcionará como un servidor TCP y recibirá medidas de un smartphone.

Necesitaremos dos sockets TCP (uno escuchando a cada programa) y enlazaremos un puerto diferente a cada uno de ellos. Cada socket TCP recibirá los datos recogidos por los smartphones. Los datos tendrán la siguiente estructura:

Tabla 5.1-3: Traza recibida de los smartphones

ID del smartphone	Datos recogidos por el sensor GPS	Datos recogidos por el giróscopo (coordenada x,y,z)	Datos recogidos por el acelerómetro	Datos recogidos por *Crayfis* de rayos cósmicos	Fecha y hora	Nivel de batería del smartphone	Número de paquete

El ID del smartphone nos permitirá identificar de qué teléfono son los datos recibidos. El resto de la tabla contendrá la información recogida por los sensores y la fecha y horade la medida. Cada vez que recibimos datos, almacenaremos esa tabla en un archivo.

Las muestras adquiridas de los smartphones de a bordo se enviarán cada minuto a través de la conexión *E-Link* a la GS.

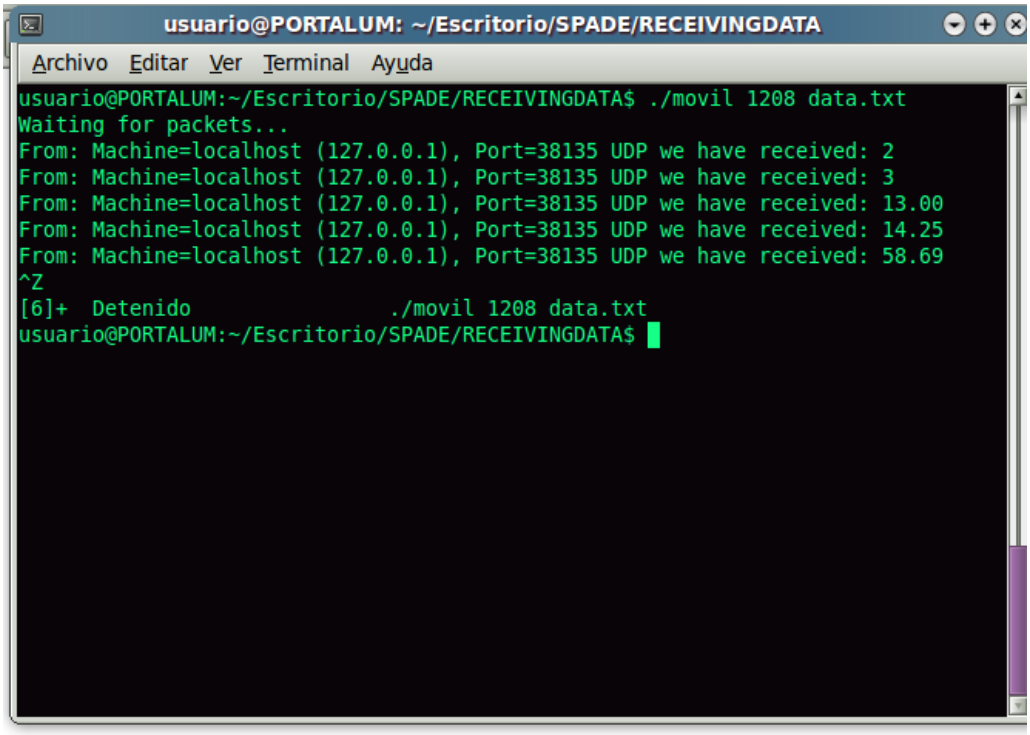


Fig. 5.1-3: Software de la estación de tierra durante el test 1

Esta es una imagen del software en ejecución durante una prueba. Para ejecutar el software tenemos que escribir en la línea de comandos el puerto que queremos escuchar en el servidor y el nombre del archivo donde vamos a escribir las medidas (en este caso "data.txt"). Después de eso, comenzamos a esperar los paquetes. Esperamos recibir paquetes cada minuto, pero después de tres minutos sin recibir ningún paquete el programa dejará de funcionar y enviaremos un comando de emergencia. En la imagen podemos ver qué datos hemos recibido y qué máquina ha enviado los datos.

3. Software del comando de emergencia

El control de información está diseñado para casos en los que o bien se produzca un comportamiento inadecuado o bien errores fatales, y nos permitirá también cambiar entre los modos definidos en el software del smartphone.

En el caso de que se produzca un error en alguno de los smartphones (identificaremos que hay error mediante el software de recepción de datos), podremos reiniciarlos. Si el error persiste, podremos seguir trabajando sin los datos de ese smartphone.

La información de control gestionará:

- Envío de comandos para el cambio de modo (de modo normal a modo vuelo y viceversa) para ello tendremos que hacer modificaciones en el sistema operativo de los smartphones.

La estructura de envío de la información de control será la siguiente:

Tabla 5.1-4: Banderas de comandos

Bandera de reboot (Flag 1)	Bandera de cambio de modo (Flag 2)

Los posibles valores de Flag 1 serán:

- 1, si queremos reiniciar el dispositivo, 0 si no.

Los posibles valores de Flag 2 serán:

- 1 si queremos cambiar a modo vuelo y 0 si queremos cambiar a modo normal

Posteriormente, ambas banderas se unificaron

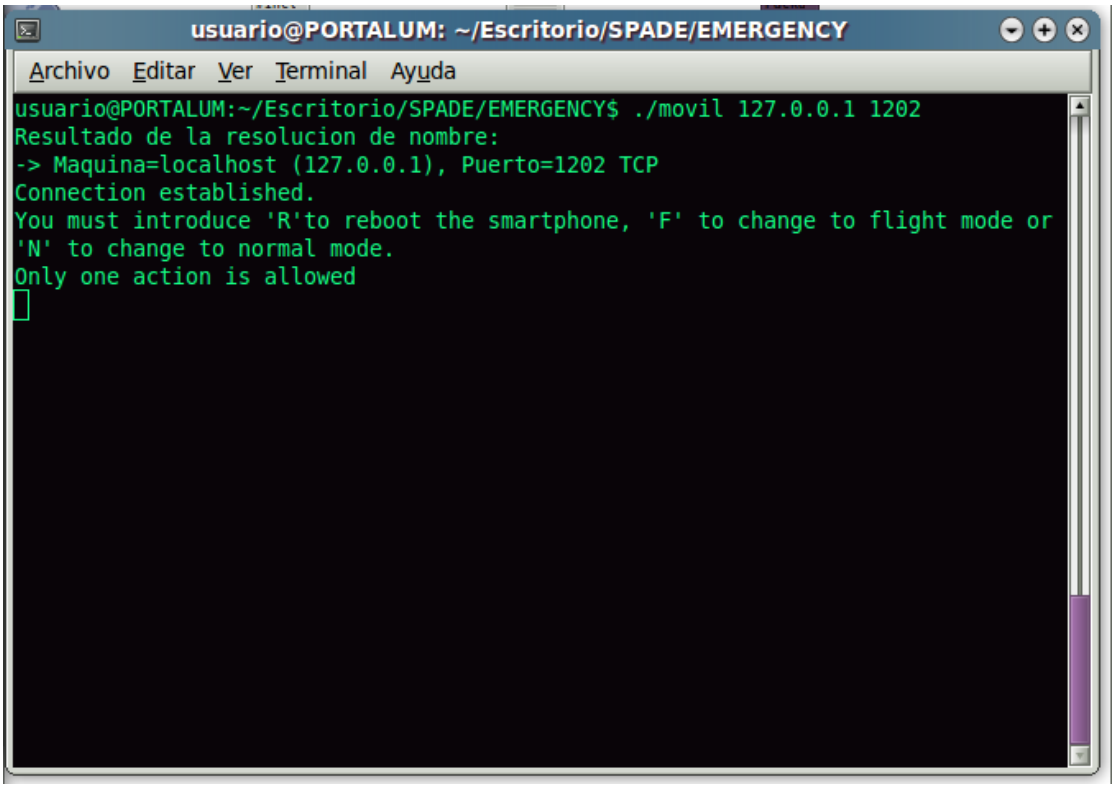


Fig. 5.1-4: Software de la estación de tierra durante el test 2

En ésta imagen se muestra el software en ejecución durante una prueba. Podemos ver que lo primero que debemos hacer es la IP del Smartphone al que queremos mandar la información y su puerto. Una vez que se haya establecido la conexión, aparecerá un menú que nos indica cómo introducir los comandos según la acción que queramos realizar.

5.2 Software a bordo del *BEXUS*

El lenguaje de programación utilizado para el desarrollo del software de a bordo es Java.

La aplicación implementada se encarga de tomar medidas de los nodos sensores de los smartphones y enviarlas a la estación de tierra.

La información recogida por los smartphones se almacenará en la memoria interna del cada teléfono, y se enviará a la estación de tierra a través del router conectado al *E-Link*

El comportamiento del software de los smartphones se puede describir con el siguiente diagrama de flujo:

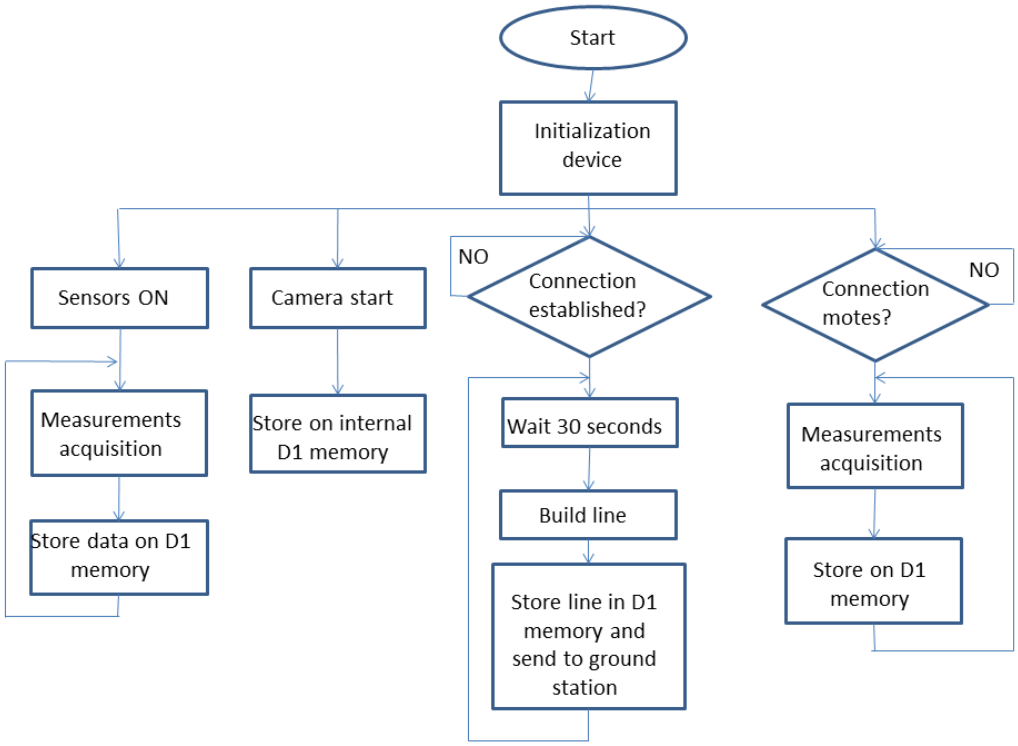


Fig. 5.2-1: Diagrama de flujo de la aplicación para los smartphones

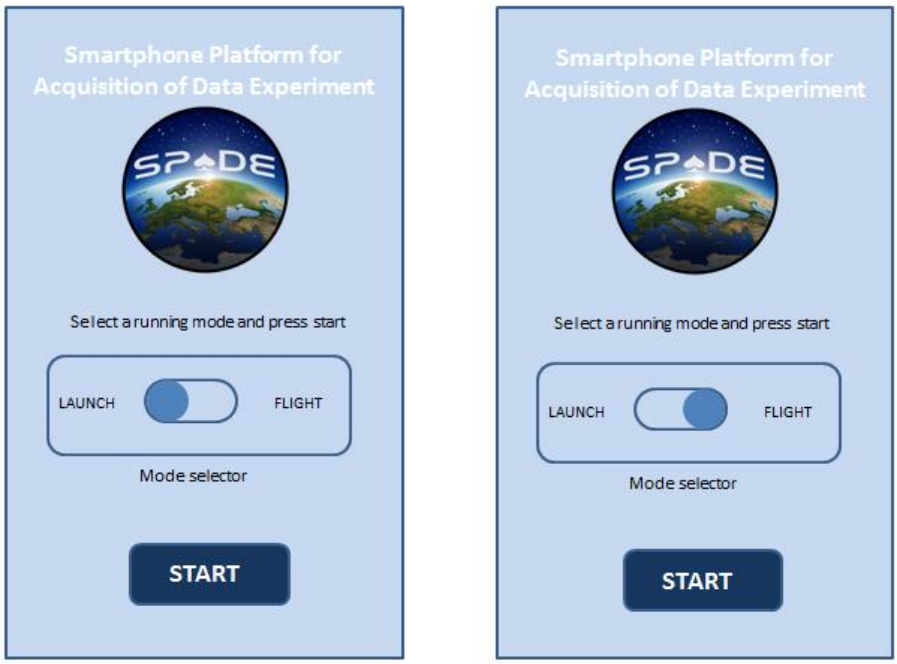
Una de las medidas va a ser tomada por la aplicación *Crayfis*, incluida las de ambos smartphones. Esta aplicación medirá rayos cósmicos utilizando la cámara del móvil para detectarlos.

La aplicación desarrollada para los smartphones se encargará de la toma de medidas de los sensores de éstos, y de realizar una conexión con la aplicación *Crayfis* para añadir la medida de rayos cósmicos a los datos recopilados.

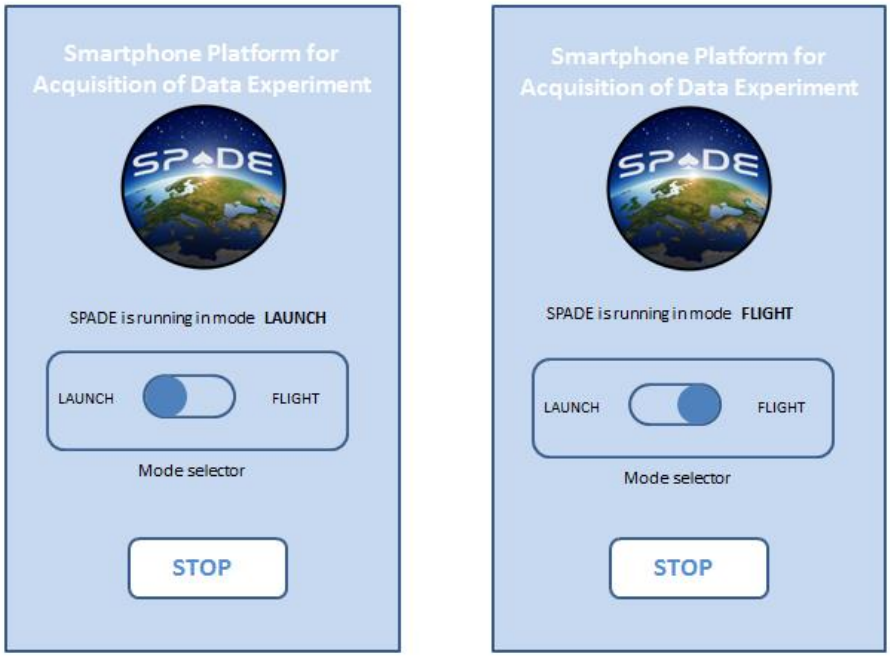
Las trazar generadas en cada lectura se encapsularán y se enviarán al router que se encontrará también a bordo, y el cual se encargará de redirigir a la estación de tierra a través de la conexión *E-Link*, donde se almacenará y se procesará la información

Los smartphones podrán también recibir comandos enviados desde la estación de tierra para efectuar un reinicio del dispositivo o un cambio de modo de la aplicación.

Debido a que la aplicación se va a encargar únicamente de enviar y recibir datos, consideramos que lo mejor era implementar una interfaz sencilla que evitase posibles problemas debido al ciclo de ejecución de las actividades de ésta.



Figuras 5.2-2 y 5.2-3: Vista de la aplicación 1



Figuras 5.2-4 y 5.2-5: Vista de la aplicación 2

Por defecto, la aplicación se iniciará en modo normal y empezará a tomar medidas cuando se pulse el botón *Start* o cuando se reciba la señal de la estación de tierra.

Por lo tanto, la funcionalidad de la aplicación se puede describir como:

1) Inicio en modo normal

2) Después de 60 segundos, si el modo no se ha cambiado, empezaremos a tomar medidas de todos los sensores y se almacenarán en la tarjeta SD.

3) La traza leída se envía a la estación de tierra. Si esto falla debido a la conectividad, las trazas almacenadas se enviarán en un bloque cuando se restablezca la conexión

4) Cuando el modo se cambie a modo vuelo, después de 60 segundos si no sea cambiado el modo, la aplicación comenzará a tomar medidas de nuevo que se almacenarán en la tarjeta SD.

5) La traza leída se envía a la estación de tierra. Si esto falla debido a la conectividad, las trazas almacenadas se enviarán en un bloque cuando se restablezca la conexión

Éste ciclo se ejecuta en un bucle de estados que puede ser terminado cerrando de forma manual la aplicación cuando se recupere el experimento de la góndola pulsando el botón *Stop* o si hay algún problema que nos obligue a reiniciar el dispositivo.

Para reiniciar el dispositivo, se enviará un comando su/reboot en la aplicación.

La comunicación entre la aplicación ***SPADE*** y *Crayfis* se realiza mediante archivos compartidos, ejecutando ésta última en background.

El formato de la traza enviada a la estación de tierra, como ya se ha comentado antes, será el siguiente:

Tabla 5.2-1: Traza enviada a la estación de tierra

ID	GPS	Gyro (x,y,z)	Accelerometer	Time	Crayfis	Battery_level	Num_package

6. Mi aportación al proyecto

6.1 Entornos para el desarrollo

Para el desarrollo del proyecto se han tenido que configurar diferentes entornos de desarrollo.

El primero de ellos ha sido Android Studio, que ha sido el software utilizado en el desarrollo de la aplicación para los smartphones

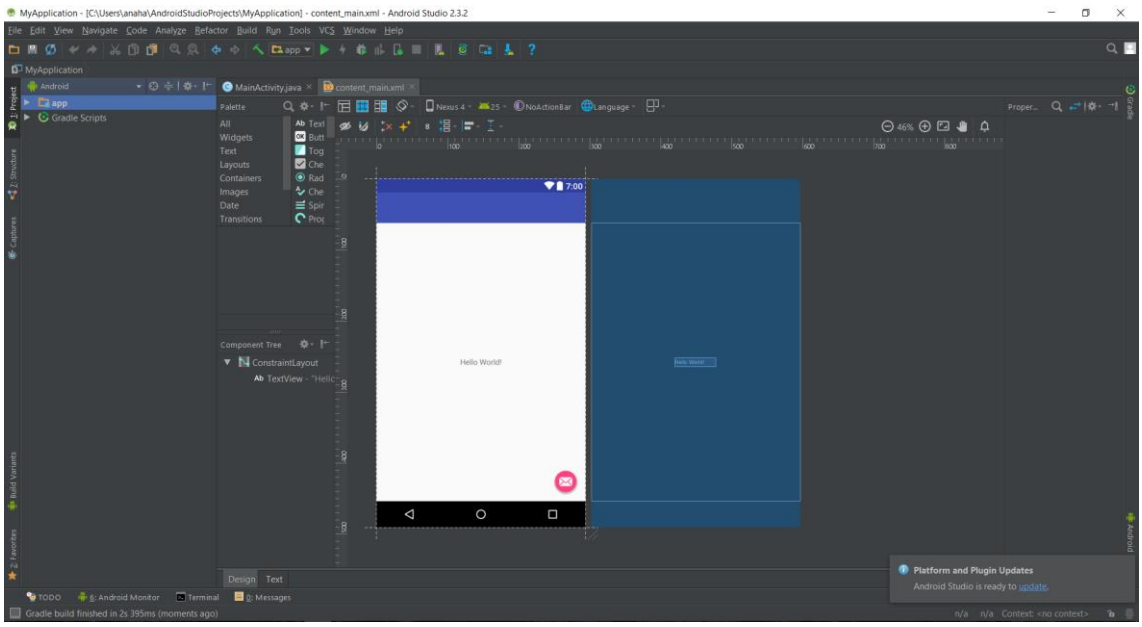


Fig. 6.1-1: Entorno de desarrollo Android Studio

El segundo entorno de desarrollo utilizado ha sido Eclipse Luna, en el que se ha desarrollado el software de procesamiento de datos

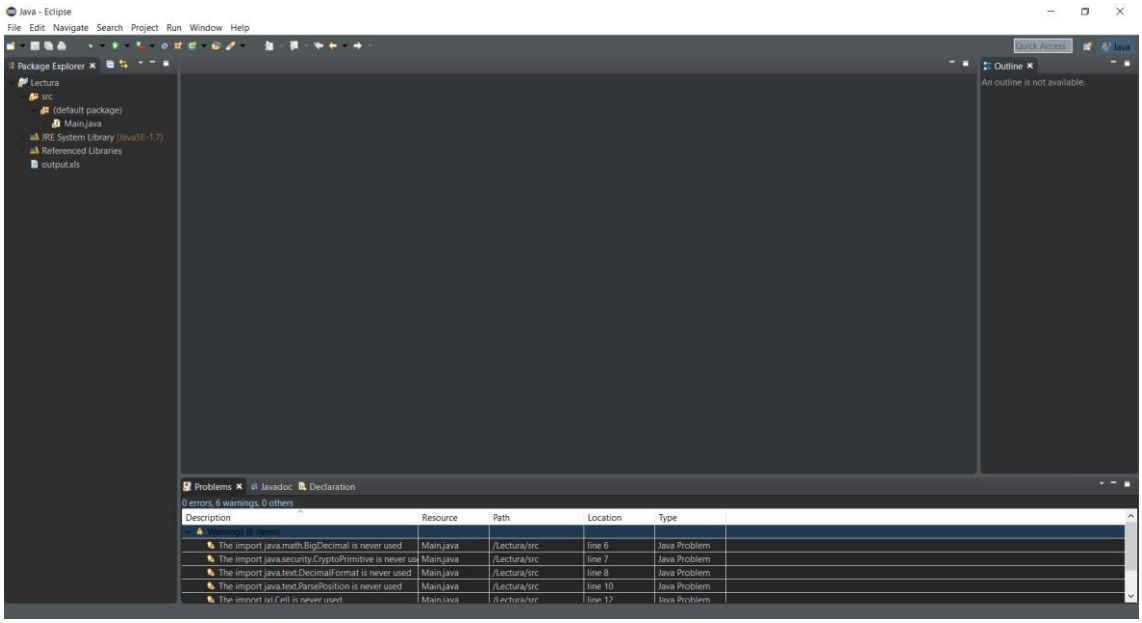


Fig. 6.1-2: Entorno de desarrollo Eclipse Luna

6.2 Configuración y adecuación de los teléfonos

Para la configuración y adecuación de los dispositivos a las necesidades del software desarrollado, se ha utilizado el siguiente programa

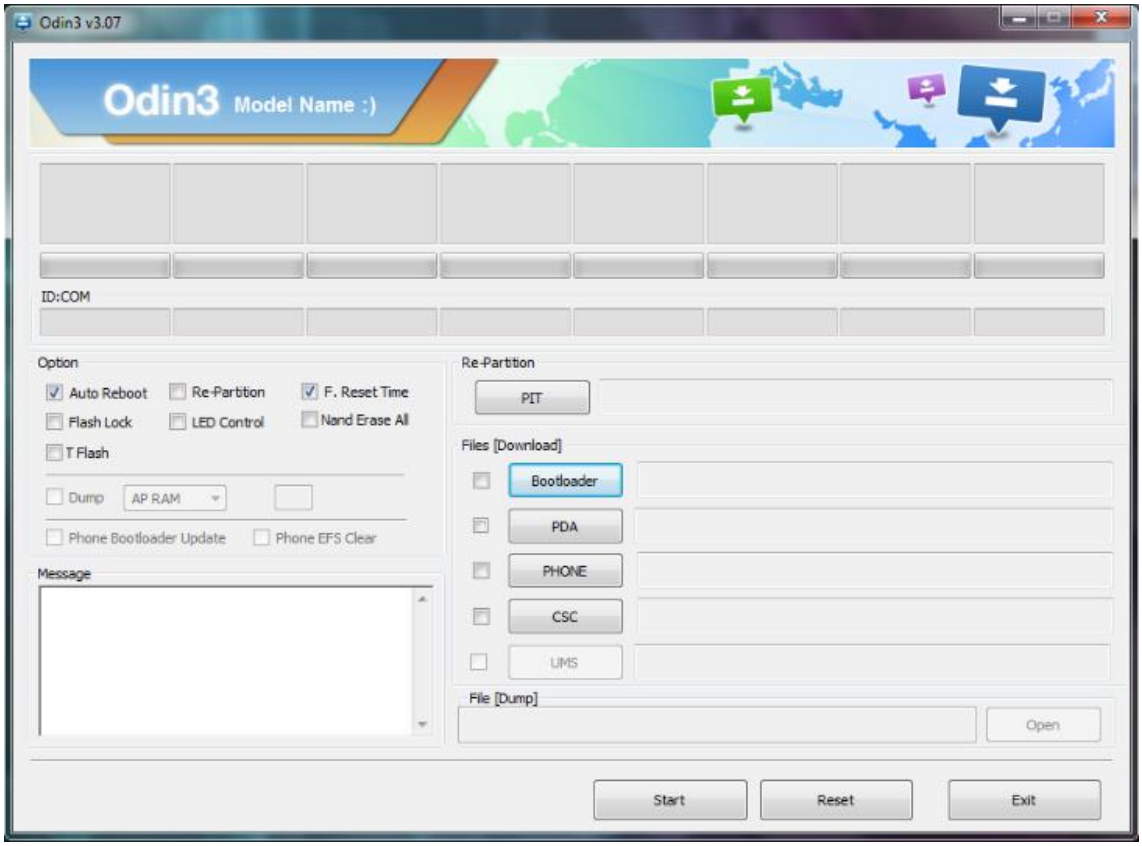


Fig. 6.2-1: Programa para rootear los smartphones Odin3

Éste programa se ha utilizado para rootear los teléfonos. He tenido que realizar ésta operación porque con la configuración del sistema operativo que tenían (Android kitkat y android lollipop), no se permitía la ejecución de comandos de administrador en la aplicación. Actualmente está restringido el acceso a algunas de las funcionalidades que ofrece el sistema operativo de los smartphones. Por ejemplo, en el caso de nuestra aplicación, no se podía realizar un reinicio de forma remota a través de un comando, y no se podía configurar de forma estática la ip en la red.

También fue necesario instalar los drivers de éstos dispositivos en el ordenador en el que se estaba haciendo el desarrollo, ya que Android Studio no lo reconocía de forma automática.

Debido a los recursos limitados del equipo utilizado para el desarrollo, la única forma de testear la aplicación durante todo el proceso de desarrollo ha sido conectando el smartphone al ordenador, ya que era imposible ejecutar un dispositivo virtual.

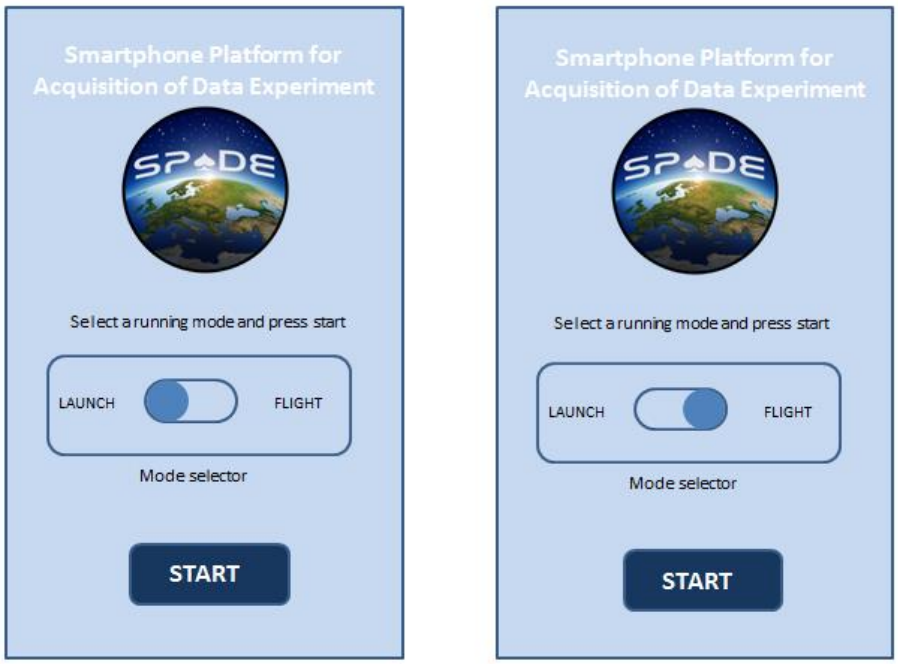
6.3 Desarrollo de software

6.3.1 On board

Estructura de la aplicación

La aplicación, tras su desarrollo inicial, sufrió varias modificaciones.

Estructura inicial



Figuras 6.3.1-1 y 6.3.1-2: Vista de la aplicación

Inicialmente se nos requería disponer y poder manejar el cambio entre el modo 1, para despegue y aterrizaje, y el modo 2, para todo el tiempo de vuelo.

Visualmente se implentó un switch para seleccionar modos, y un botón para iniciar y detener la toma de datos.

Éste diseño incluía también una escucha para la recepción de comandos por parte de la estación de tierra, que permitía cambiar de modo y reiniciar el teléfono en caso de que fuese necesario.

También se mostraba un toast en el que se mostraban los datos recogidos y enviados en la traza a GS y almacenados en el almacenamiento interno.

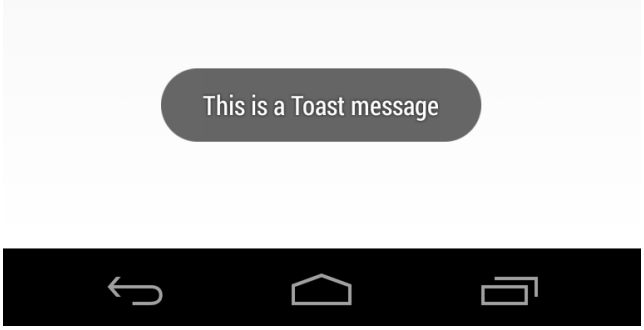


Fig. 6.3.1-3: Mensaje Toast

Inicialmente creíamos que la red de los teléfonos con el router se iba a configurar por wifi, pero debido a las restricciones de comunicaciones de la góndola, ésta idea se tuvo que modificar a una conexión cableada, con todas las dificultades que genera la configuración cableada para un teléfono.

La ip se configuraba de forma dinámica, y no existía ningún problema a la hora de realizar la conexión, ya que si el router perdía la conexión, al restaurarse, se volvía a conectar de forma automática a la red.

El protocolo implementado era TCP/UDP, para recepción y envío de datos respectivamente.

La evolución del software se fue adaptando a los requisitos e inconvenientes que se nos fueron presentando:

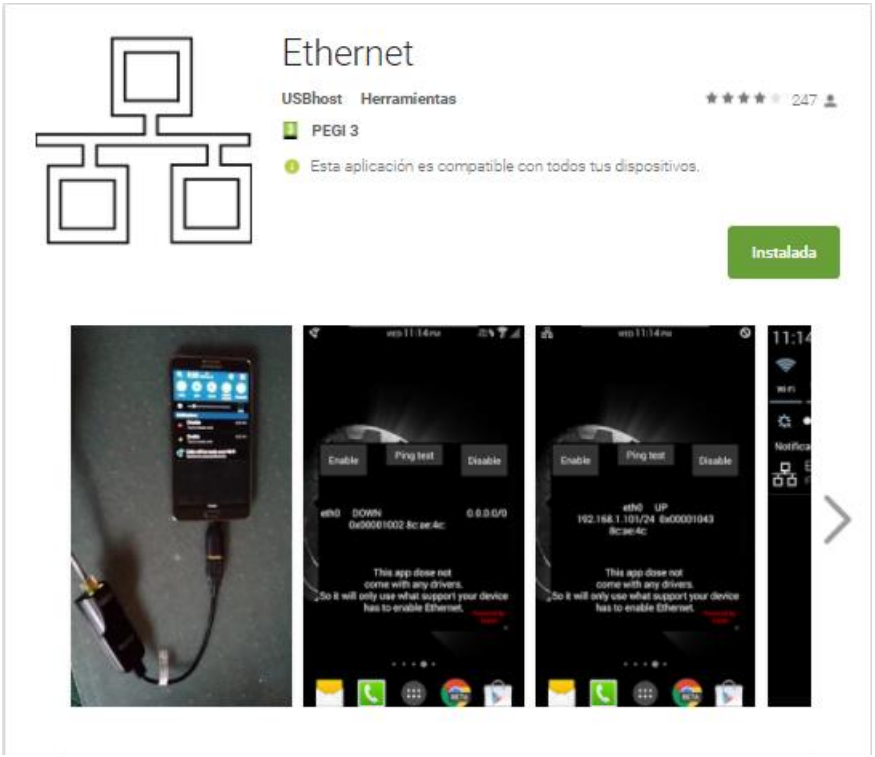
-La conexión wifi se sustituyó por conexión cableada, usando adaptadores de red

-El protocolo de envío de datos se tuvo que modificar debido a que el adaptador para la conexión cableada únicamente permite protocolos de comunicación TCP, no reconoce UDP.

-Al hacer la conexión cableada, tuvimos que establecer un orden en el encendido de dispositivos, siendo el primero el router, para que fuese posible que los teléfonos obtuviesen la ip. En un primer desarrollo de la solución para éste problema, se implementó un proceso de espera activa, en el que se solicitaba una ip hasta que el router la devolviese.

-Se tuvo que establecer una ip estática, definida por los ingenieros de la ESA, ya que cada equipo dispondríamos de unas direcciones concretas para la conexión con el *E-Link*. Tuvimos que configurar el router con acuerdo a éstos requisitos, y las ip de los teléfonos de forma estática dentro de la LAN creada.

Para esto hicimos uso de una aplicación de Google Play, que gestionaba las conexiones Ethernet



Figuras 6.3.1-4: Aplicación para la configuración Ethernet

Era necesario iniciarla manualmente antes de cargar nuestra aplicación, con el problema añadido del reboot. Si se tenía que reiniciar el teléfono por comandos no podríamos volver a realizar la conexión con el router, pero seguiría guardando los datos en local, por lo que se consideró una opción aceptable.

- Nos dijeron que el acceso a los dispositivos iba a ser limitado, ya que antes del lanzamiento, sólo el responsable de pruebas iba a poder modificar y ajustar brevemente el experimento completo. Debido a esto, la interfaz con botones pasó a ser algo inviable, ya que los teléfonos tendrían que estar atornillados, con la pantalla hacia abajo (para poder realizar las medidas de rayos cósmicos con la cámara trasera), y no podríamos acceder a tocar nada.

-El hecho de que los móviles tuvieran que estar atornillados durante bastante tiempo sin que pudiéramos acceder a ellos implicaba que, si se mantenían encendidos durante el tiempo de preparación, la batería se iría drenando. Debido a que tenía que llevar conectado el adaptador Ethernet mientras estaba encendido para no perder la ip asignada, o tendría que reiniciarse, se tuvieron que incluir cargadores inalámbricos (no muy efectivos, ya que la batería se gastaba más rápido que se cargaba)

Tras todos los cambios que se tuvieron que realizar, la versión final de la aplicación tenía las siguientes partes:

-Inicio de la aplicación de forma automática al iniciar el teléfono

-Configuración de red

-Toma de medidas

-Envío de traza a GS

-Receptor de comandos

Inicialmente se creó un escuchador de comandos, para que se pudiese manejar desde GS, que finalmente, a pesar de seguir integrado en ésta y tener posibilidades de usarlo, se decidió dejar en un segundo plano y utilizarlo sólo en caso de emergencia.

Visualmente, la versión definitiva de la aplicación quedó de la siguiente forma:

Fig. 6.3.1-5: Vista de la aplicación definitiva

Inicio de la aplicación de forma automática al iniciar el teléfono

Para poder realizar ésta operación, es necesario declarar el permiso correspondiente

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

La finalidad de esto es solicitar permisos al usuario para poder acceder a la señal de boot completado.

También tenemos que declarar un receptor de eventos

```
<receiver android:name=".MyBroadCastReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

Que será el encargado de redirigirnos a *.MyBroadcastReceiver* cuando detecte la señal BOOT_COMPLETED, ambos en el Manifest.xml

En *MyBroadcastReceiver* tendremos el siguiente código:

```
public class MyBroadCastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

        if(intent.getAction().equals(Intent.ACTION_BOOT_COMPLETED)){
            Intent i = new Intent(context, MainActivity.class);
            i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            context.startActivity(i);
        }
    }
}
```

Ésta clase debe extender necesariamente *BroadcastReceiver* para poder lanzarse en el momento adecuado del ciclo de ejecución de la aplicación.

Se declara un método que sobrescribe el onReceive que por defecto tienen las clases en los proyectos Android.

Éste método comprobará que el action que hay en el Intent que ha recibido como parámetro se corresponde efectivamente con la constante propia de los *Intent*

```
public static final String ACTION_BOOT_COMPLETED =
"android.intent.action.BOOT_COMPLETED";
```

En caso correcto, se crea una llamada a la actividad principal *MainActivity*, la cual contiene toda la funcionalidad de la aplicación.

Configuración de red

Para poder adaptar los terminales a las necesidades de red que se habían configurado en el router de forma estática, se añadió una configuración por comandos al inicio de la aplicación en el método *onCreate*

```
try {

    Runtime.getRuntime().exec("su -c netcfg eth0 up" );
    Runtime.getRuntime().exec("su -c netcfg eth0 dhcp" );



    StringBuilder s4=new StringBuilder();
    s4.append("ifconfig wlan0 down");
    String comando4=s4.toString();
    Process p4=Runtime.getRuntime().exec(comando4);
    p4.waitFor();



    StringBuilder s3=new StringBuilder();
    s3.append("ifconfig eth0 192.168.10.156 netmask 255.255.255.0");
    String comando3=s3.toString();
    Process p3=Runtime.getRuntime().exec(comando3);
    p3.waitFor();



    StringBuilder s1=new StringBuilder();
    s1.append("su -c /system/bin/sh setprop net.dns1 208.67.222.222");
    String comando1=s1.toString();
    Process p1=Runtime.getRuntime().exec(comando1);
    p1.waitFor();

    StringBuilder s2=new StringBuilder();
    s2.append("su -c setprop net.dns2 208.67.222.220");
    String comando2=s2.toString();
    Process p2=Runtime.getRuntime().exec(comando2);
    p2.waitFor();


    StringBuilder s=new StringBuilder();
    s.append("su -c route add default gw 192.168.10.1" +
            " dev eth0");
    String comando=s.toString();
    Process p=Runtime.getRuntime().exec(comando);
    p.waitFor();

    Toast.makeText(this,"terminado",Toast.LENGTH_LONG).show();
} catch (IOException e) {
    e.printStackTrace();
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

Cada uno de los bloques ejecutará un comando

Bloque 1:

Habilita eth0 y dhcp, esto es necesario para que se reconozca la conexión cableada a la red

```
Runtime.getRuntime().exec("su -c netcfg eth0 up" );
Runtime.getRuntime().exec("su -c netcfg eth0 dhcp" );
```

Bloque 2:

Deshabilita el wifi en caso de que se encontrase activado

```
StringBuilder s4=new StringBuilder();
s4.append("ifconfig wlan0 down");
String comando4=s4.toString();
Process p4=Runtime.getRuntime().exec(comando4);
p4.waitFor();
```

Bloque 3:

Configura la ip del dispositivo y la máscara de red. Para cada uno de los móviles era necesario cambiar ésta configuración antes de que se instalara la aplicación

```
StringBuilder s3=new StringBuilder();
s3.append("ifconfig eth0 192.168.10.156 netmask 255.255.255.0");
String comando3=s3.toString();
Process p3=Runtime.getRuntime().exec(comando3);
p3.waitFor();
```

Bloque 4:

Se configuran las DNS

```
StringBuilder s1=new StringBuilder();
s1.append("su -c /system/bin/sh setprop net.dns1 208.67.222.222");
String comando1=s1.toString();
Process p1=Runtime.getRuntime().exec(comando1);
p1.waitFor();

StringBuilder s2=new StringBuilder();
s2.append("su -c setprop net.dns2 208.67.222.220");
String comando2=s2.toString();
Process p2=Runtime.getRuntime().exec(comando2);
p2.waitFor();
```

Bloque 5:

Se configura la default Gateway

```
StringBuilder s=new StringBuilder();
s.append("su -c route add default gw 192.168.10.1" +
        " dev eth0");
String comando=s.toString();
Process p=Runtime.getRuntime().exec(comando);
p.waitFor();
```

Inicialmente se creó una pantalla de configuración de ip, la cual tuvo que ser sustituida por la configuración anterior debido al problema con la accesibilidad a los dispositivos y por el reboot del dispositivo. Por lo que se decidió compilar la aplicación, con los parámetros predefinidos de forma interna, para cada dispositivo

Fig. 6.3.1-6: Pantalla de configuración de ip

Configuración extra

Para evitar que la pantalla se apagase y la aplicación se pusiera en suspensión, se implementó la siguiente configuración, en la que además se establece el brillo de la pantalla al mínimo para disminuir el gasto de batería

```
WindowManager.LayoutParams lp = this.getWindow().getAttributes();
lp.screenBrightness =0.0f;
this.getWindow().setAttributes(lp);
getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
```

A parte de ésta configuración, en las opciones del teléfono se estableció también el mínimo brillo, para optimizar recursos y se fijó la posición de la pantalla en vertical para evitar el pausado y reanudación de la aplicación durante el vuelo.

Toma de medidas

Se configuran las variables que se van a mostrar por pantalla

```
gyro = (TextView) findViewById(R.id.gyro);
hora = (TextView) findViewById(R.id.hora);
accelerometer = (TextView) findViewById(R.id. accelerometer);
latituteField = (TextView) findViewById(R.id.Latitud);
longitudeField = (TextView) findViewById(R.id.Longitud);
battField = (TextView) findViewById(R.id.battery);
```

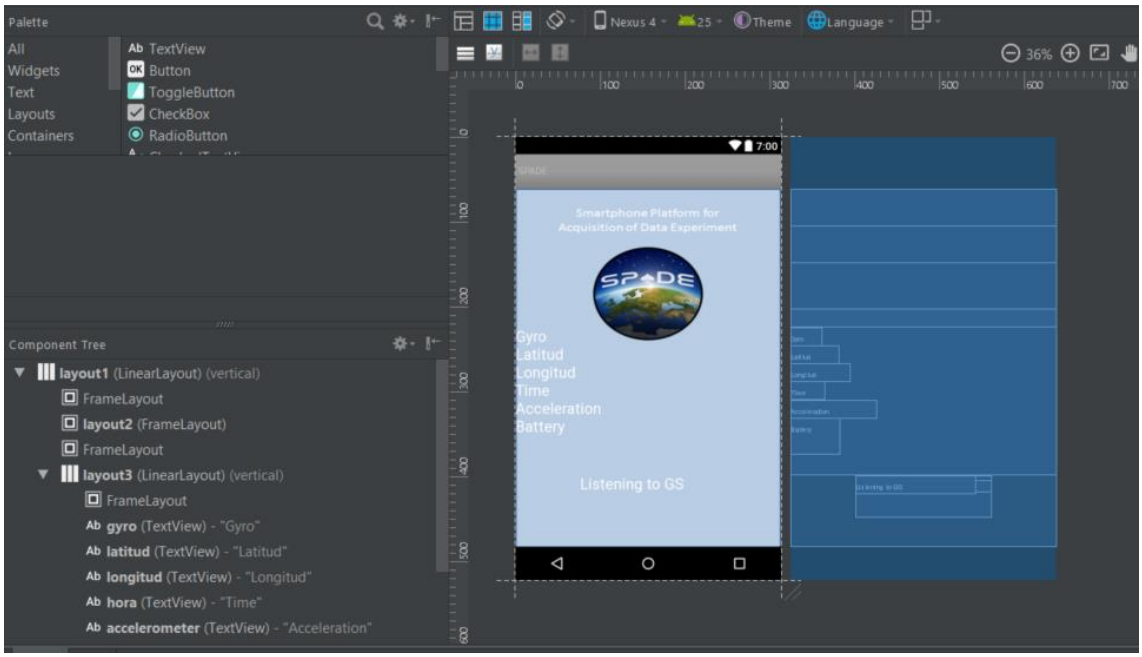


Fig. 6.3.1-7: Vista de la aplicación en la herramienta de desarrollo

COORDENADAS

Para la configuración del lector de coordenadas es necesario configurar un servicio de localización de la siguiente forma

```
// Se obtiene el servicio
locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
// Se define el criterio de cómo se va a seleccionar el rpovider
Criteria criteria = new Criteria();
provider = locationManager.getBestProvider(criteria, false);
Location location = locationManager.getLastKnownLocation(provider);

// Inicialización de los campos
if (location != null) {
    onLocationChanged(location);
} else {
    latituteField.setText("Latitude: Loading");
    longitudeField.setText("Longitude: Loading");
}

LocationManager locationManagerGPS = (LocationManager)
getSystemService(LOCATION_SERVICE);

if (locationManagerGPS.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
    Toast.makeText(this, "GPS is Enabled in your device",
Toast.LENGTH_SHORT).show();
} else {
    showGPSDisabledAlertToUser();
}
```

En la inicialización de campos, si no se han obtenido aún las coordenadas lo que veremos en pantalla será "*Loading"* en el campo correspondiente

Se comprueba también si el GPS se encuentra activado en el dispositivo. En caso de que no lo esté, se avisa por pantalla con un alert, ya que hacerlo de forma programada es un proceso bastante complejo y no merecía la pena implementarlo ya que esa configuración se mantenía incluso en el reboot.

```
public void showGPSDisabledAlertToUser() {
    AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);

    // Título
    alertDialogBuilder.setTitle("GPS alert!");

    // Mensaje
    alertDialogBuilder
            .setMessage("The GPS must be activated!")
            .setCancelable(false)
            .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                }
            });

    // Crear alerta
    AlertDialog alertDialog = alertDialogBuilder.create();

    // Mostrar alerta
    alertDialog.show();
}
```

Fig. 6.3.1-8: Mensaje de alerta GPS

Cuando la localización cambia, los parámetros se actualizan

```
@Override
public void onLocationChanged(Location location) {
    int lat = (int) (location.getLatitude());
    int lng = (int) (location.getLongitude());
    latituteField.setText("Latitude:" + String.valueOf(lat));
    longitudeField.setText("Longitude:" + String.valueOf(lng));

    Double latitud=new Double(location.getLatitude());
    Double longitud=new Double(location.getLongitude());

    gpsStored=latitud.toString()+","+longitud.toString()+" ";
}
```

ACELERÓMETRO

Para obtener los valores del acelerómetro se creó un escuchador de eventos

```
private final SensorEventListener AccelerometerSensorListener
        = new SensorEventListener() {

    @Override
    public void onSensorChanged(SensorEvent event) {

        if(codeMode.equals("01")){
            if (event.accuracy == SensorManager.SENSOR_STATUS_UNRELIABLE) {
                return;
            }

            if(accelerationMeasure==true){
                accelerometer.setText("Acceleration: " + event.values[0]);
                Float acc=new Float(event.values[0]);
                accelerationStored=acc.toString()+" ";
                accelerationMeasure=false;
            }

        }

    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {

    }
};
```

El valor de *codeMode* se corresponde con el comando introducido en ground station para realizar una operación.

El código ‘01’ corresponde con el modo vuelo. Nuestra aplicación sólo comenzará a tomar medidas de GPS cuando se encuentre en modo vuelo

GIRÓSCOPO

Éste sensor se ha utilizado a demás como hilo principal para el envío de trazas, ya que se encuentra en constante cambio

```
private final SensorEventListener GyroSensorListener
        = new SensorEventListener() {

    @Override
    public void onAccuracyChanged(Sensor arg0, int arg1) {
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        if (codeMode.equals("01")) {
            Time time = new Time();
            enviarTraza(event.accuracy, event, time);
            if (event.accuracy == SensorManager.SENSOR_STATUS_UNRELIABLE) {
                return;
            }

        }
    }

    public void enviarTraza(int accuracy,SensorEvent event, Time time){
        if (event.accuracy == SensorManager.SENSOR_STATUS_UNRELIABLE){
            gyroStored = "-1";
        }else{
            gyro.setText("Orientation X:" + Float.toString(event.values[2]) +
"\n" +
                    "Orientation Y:" + Float.toString(event.values[1]) + "\n"
+
                    "Orientation Z:" + Float.toString(event.values[0]));
```

```
gyroStored=Float.toString(event.values[2])+","+Float.toString(event.values[1]
)+","+Float.toString(event.values[0])+" ";

        }

        time.setToNow();
        if (latencia + 10 == time.second) {

            latencia = time.second;
            battery = batteryLevel(context);
            battField.setText("Battery:" + battery);
            accelerationMeasure = true;

            try {

                File myFile = new File("/sdcard/dataSpade.txt");
                if(!myFile.exists()){
                    myFile.createNewFile();
                }

                BufferedWriter bW;
                bW = new BufferedWriter(new FileWriter(myFile,true));
                numPackage+=1;

                String cosmicRays = ReadBtn();
                sb=new StringBuilder();
                sb.append("1 ").append(gpsStored+" ").append(gyroStored+"
").append(accelerationoinStored+" ").append(timeStored+" ").append(cosmicRays+"
").append(battery+" ").append(numPackage);

bW.write(sb.toString());
                bW.newLine();
                bW.flush();
                bW.close();

            } catch (Exception e) {
                Toast.makeText(getBaseContext(), e.getMessage(),
                        Toast.LENGTH_SHORT).show();
            }

            hiloEnvio();

            updateTime(latencia, time);

        }
    }

};
```

Se pueden distinguir varios bloques dentro de éste método

Bloque 1: Cambio de medida en el sensor

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (codeMode.equals("01")) {
        Time time = new Time();
        enviarTraza(event.accuracy, event, time);
        if (event.accuracy == SensorManager.SENSOR_STATUS_UNRELIABLE) {
            return;
        }

    }
}
```

Si el sensor no es fiable, se envía la traza con el parámetro sin informar y se para el evento

Bloque 2: Envío de traza

Cada vez que se modifica el valor del giróscopo, se cambia la hora a la hora actual.

Sólo se va a enviar la traza si ha pasado el tiempo establecido desde el envío de la última traza, que se ha establecido a 10 segundos.

A continuación leemos el nivel de batería y modificamos la bandera para que se actualice el acelerómetro.

Se comprueba si existe el archivo local en el que se están guardando las trazas, si no es así, lo crea. Se crea un buffer que se va a enlazar al archivo al que hemos accedido

Leemos el contenido del archivo en el que se ha guardado la medida de *Crayfis*, en el que aparecerá siempre el último número de rayos cósmicos leídos.

Creamos un stringBuilder para montar la cadena que se va a enviar a ground station con todas las medidas tomadas, añadiendo todas las medidas y una nueva línea al final, y cerramos el buffer.

Por último, llamamos al método que hace el envío de la traza

```
public void enviarTraza(int accuracy,SensorEvent event, Time time){
    if (event.accuracy == SensorManager.SENSOR_STATUS_UNRELIABLE){
        gyroStored = "-1";
    }else{
        gyro.setText("Orientation X:" + Float.toString(event.values[2]) + "\n" +
                "Orientation Y:" + Float.toString(event.values[1]) + "\n" +
                "Orientation Z:" + Float.toString(event.values[0]));

gyroStored=Float.toString(event.values[2])+","+Float.toString(event.values[1])+"
,"+Float.toString(event.values[0])+" ";

    }

    time.setToNow();
    if (latencia + 10 == time.second) {

        latencia = time.second;
        battery = batteryLevel(context);
        battField.setText("Battery:" + battery);
        accelerationMeasure = true;

        try {

            File myFile = new File("/sdcard/dataSpade.txt");
            if(!myFile.exists()){
                myFile.createNewFile();
            }

            BufferedWriter bW;
            bW = new BufferedWriter(new FileWriter(myFile,true));
            numPackage+=1;

            String cosmicRays = ReadBtn();
            sb=new StringBuilder();
            // El primer parámetro es el identificador del smartphone
            sb.append("1 ").append(gpsStored+" ").append(gyroStored+"
").append(acceleratioinStored+" ").append(timeStored+" ").append(cosmicRays+"
").append(battery+" ").append(numPackage);

            bW.write(sb.toString());
            bW.newLine();
            bW.flush();
            bW.close();

        } catch (Exception e) {
            Toast.makeText(getBaseContext(), e.getMessage(),
                    Toast.LENGTH_SHORT).show();
        }

        hiloEnvio();
        updateTime(latencia, time);

    }
}
```

Método para leer del archivo en el que se guardan los datos de *Crayfis*

```
public String ReadBtn() {

    String res="";
    String readstring = new String();
    try {
        FileInputStream fileIn=openFileInput("/sdcard/crayfis.txt");
        InputStreamReader InputRead= new InputStreamReader(fileIn);

        char[] inputBuffer= new char[READ_BLOCK_SIZE];
        String s="";
        int charRead;


        while ((charRead=InputRead.read(inputBuffer))>0) {
            readstring=String.copyValueOf(inputBuffer,0,charRead);
        }
        InputRead.close();

    } catch (Exception e) {
        e.printStackTrace();
    }
    return readstring;
}
```

La aplicación *Crayfis* se ejecuta en paralelo con nuestra aplicación, y se ha modificado para que los datos que va leyendo se almacenen también en un archivo que va a ser compartido.

Lo que hacemos en éste método es abrir éste archivo, leerlo y devolver su contenido, que va a ser la última medida de rayos cósmicos tomada, que se incluirá en la traza que se envíe a ground station

Método de envío de traza

Abrimos un socket con la ip y el puerto en el que va a estar escuchando el router, ya que se ha configurado una ip y puerto diferentes para cada Smartphone.

Se escribe la cadena en el buffer y lo envía, y cuando termina limpia el buffer y el socket.

La ejecución de éste proceso se hace en un nuevo hilo nuevo que se crea y se lanza aquí

```
public void hiloEnvio(){

    new Thread(new Runnable() {
        public void run() {
            try{
                Socket socket = new Socket(ip,port);


                BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                BufferedWriter out = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));

                out.write(sb.toString());
                out.flush();

                out.close();
                socket.close();
            } catch (UnknownHostException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }).start();
}
```

Método de actualización de la hora

```
public void updateTime(int latencia, Time time){

    if (latencia + 10 > 60) {
        latencia = latencia - 60;
    }

    hora.setText("Time:" + (time.hour < 10 ? "0" + time.hour : time.hour) +
":" + (time.minute < 10 ? "0" + time.minute : time.minute)
            + ":" + (time.second < 10 ? "0" + time.second : time.second));


    timeStored=(time.hour < 10 ? "0" + time.hour : time.hour) + ":" +
(time.minute < 10 ? "0" + time.minute : time.minute)
            + ":" + (time.second < 10 ? "0" + time.second : time.second);
}
```

Método de lectura del nivel de batería

```
public static String batteryLevel(Context context)
{
    Intent intent  = context.registerReceiver(null, new
IntentFilter(Intent.ACTION_BATTERY_CHANGED));
    int    level   = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, 0);
    int    scale   = intent.getIntExtra(BatteryManager.EXTRA_SCALE, 100);
    int    percent = (level*100)/scale;
    return String.valueOf(percent);
}
```

Para enlazar tato el sensor del acelerómetro como el del giróscopo declaramos el servicio de sensores y accedemos a cada uno de los sensores.

```
sManager = (SensorManager) getSystemService(SENSOR_SERVICE);

envSense = sManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

gyroSense = sManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

RECEPCIÓN DE COMANDOS

Cuando arranca la aplicación, en el método principal, se crea e inicia un hilo que implementa una espera activa de la recepción de comando. Está configurado con un protocolo TCP

```
text = (TextView) findViewById(R.id.lgsText);
updateConversationHandler = new Handler();
this.serverThread = new Thread(new ServerThread());
this.serverThread.start();
```

Método de espera activa de comando

Se crea un socket en el puerto, diferente para cada Smartphone, y acordado con ground station.

```
class ServerThread implements Runnable {

    public void run() {
        Socket socket = null;
        try {
            serverSocket = new ServerSocket(SERVERPORT);
        } catch (IOException e) {
            e.printStackTrace();
        }
        while (!Thread.currentThread().isInterrupted()) {

            try {
                socket = serverSocket.accept();
                CommunicationThread commThread = new
CommunicationThread(socket);
                new Thread(commThread).start();

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Cuando se reciba un comando, se abrirá un proceso de lectura de dicho comando

```
class CommunicationThread implements Runnable {

    private Socket clientSocket;
    private BufferedReader input;
    public CommunicationThread(Socket clientSocket) {

        this.clientSocket = clientSocket;

        try {

            this.input = new BufferedReader(new
InputStreamReader(this.clientSocket.getInputStream()));

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void run() {

        while (!Thread.currentThread().isInterrupted()) {

            try {

                String read = input.readLine();
                updateConversationHandler.post(new updateUIThread(read));
                input.close();

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

}
```

Una vez leído el comando, se procederá a realizar los cambios necesarios en el estado de la aplicación dependiendo del valor recibido.

```
class updateUIThread implements Runnable {
    private String msg;

    public updateUIThread(String str) {
        this.msg="";
        this.msg = str;
    }

    @Override
    public void run() {
        if (msg != null) {
            String code= (String.valueOf((msg.charAt(msg.length()-
2))))+(String.valueOf((msg.charAt(msg.length()-1))));
            if (code.equals("00") || code.equals("01") | code.equals("10")) {

                if(code.equals("10")){
                    try {
                        Process proc = Runtime.getRuntime().exec(new String[]
{ "su", "-c", "reboot" });
                        proc.waitFor();
                    } catch (Exception ex) {
                        Log.i("Sorry", "Could not reboot", ex);
                    }

                }else{
                    codeMode=code;
                }
            }

        }
    }
}
```

Tabla 6.3.1-1: Comandos

Código	Valor
00	Modo normal
01	Modo vuelo
10	Reboot

Inicialmente estos valores se asignaban a variables diferentes, una para reboot y otra para cambiar entre modo normal y vuelo, pero por clarificar se decidió unificar en una única variable.

Si se recibe el código de reboot, se lanza un comando de consola para reiniciar el teléfono.

En el caso de que se reciba un código de modo, el valor simplemente se almacenará en *codeMode*, que es una variable de la cual se comprueba su valor en la toma de medida de giróscopo y del GPS

Antes de reiniciar el dispositivo, se llama a *onStop* para guardar adecuadamente los datos

ONSTOP

Cuando se cierre la aplicación hay que hacerlo de la forma adecuada, para ello hay desenlazar los listeners de los sensores.

También debemos cerrar el socket, ya que si no lo hacemos, cuando volvamos a cargar la aplicación nos encontraremos con que el puerto está ocupado y no podremos volver a recibir comandos

```
@Override
public void onStop(){

    sManager.unregisterListener(GyroSensorListener);
    sManager.unregisterListener(AccelerometerSensorListener);
    super.onStop();

    try {
        serverSocket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

6.4 Análisis de los resultados

6.4.1 Desarrollo de sw para el procesamiento de archivos de resultados

Para el desarrollo del software de procesamiento de archivos de resultados se ha creado una aplicación java.

Ésta aplicación se encarga de leer un archivo de texto, que se encuentra en una ubicación pasada en los parámetros. Se procesará el contenido, que se ha guardado con un formato determinado, y se exportará a un Excel, el cual se utilizará para generar gráficas de análisis de lo datos.

Establecemos la ruta de la que se va a leer el archivo, y la ruta donde se a a crear el Excel, y creamos las variables en las que se van a informar los datos en cada iteración

```
public class Main {

    public static void main(String[] args) throws FileNotFoundException, Bif-
fException, IOException, WriteException{
        BufferedReader in = new BufferedReader(new File-
Reader("C:/dataSpade2.txt"));
        try {
             String line;
             WritableWorkbook wworkbook;
            wworkbook = Workbook.createWorkbook(new File("C:/Users/Ana
HP/Desktop/output.xls"));
            WritableSheet wsheet = wworkbook.createSheet("First Sheet", 0);
             int i=1;
             while((line = in.readLine()) != null)
             {

                 String[] parts = line.split(" ");
                 String idMovil = parts[0]; // 004
                 String gps = parts[1]; // 034556
                 String gyro = parts[2];

                 String[] partsGyro=gyro.split(",");
                 String part1Gyro=partsGyro[0];
                 String part2Gyro=partsGyro[1];
                 String part3Gyro=partsGyro[2];

                 String acelerometro=parts[4];
                 String hora=parts[6];
                 String crayfis=parts[7];
                 String batery=parts[8];
                 String paquete=parts[9];
```

Definimos la columna y la cabecera asociada, y las variables auxiliares para los campos compuestos por varios datos

```
    Label label = new Label(0, 0, "Id Smartphone");
                wsheet.addCell(label);
                Label label1 = new Label(1, 0, "GPS");
                wsheet.addCell(label1);
                Label label2 = new Label(2, 0, "Gyro X");
                wsheet.addCell(label2);
                Label label3 = new Label(3, 0, "Gyro Y");
                wsheet.addCell(label3);
                Label label4 = new Label(4, 0, "Gyro Z");
                wsheet.addCell(label4);
                Label label5 = new Label(5, 0, "Accelerometer");
                wsheet.addCell(label5);
                Label label6 = new Label(6, 0, "Time");
                wsheet.addCell(label6);
                Label label7 = new Label(7, 0, "Crayfis");
                wsheet.addCell(label7);
                Label label8 = new Label(8, 0, "Battery");
                wsheet.addCell(label8);
                Label label9 = new Label(9, 0, "Package");
                wsheet.addCell(label9);

                Number idN = new Number(0, i, new Double(idMovil));
                Label gpsN = new Label(1, i, gps);



                double d = new Double(part1Gyro);
                NumberFormat nf = NumberFormat.getInstance();
                nf.setMinimumFractionDigits(7);

                double d1 = new Double(part2Gyro);
                NumberFormat nf1 = NumberFormat.getInstance();
                nf1.setMinimumFractionDigits(7);

                double d2 = new Double(part3Gyro);
                NumberFormat nf2 = NumberFormat.getInstance();
                nf2.setMinimumFractionDigits(7);

                double d3 = new Double(acelerometro);
                NumberFormat nf3 = NumberFormat.getInstance();
                nf3.setMinimumFractionDigits(7);
```

Rellenamos cada variable con el dato correspondiente y añadimos la celda.

Una vez recorrido todo el documento de entrada, se escribe y cierra el excel

```
                    Number gyroX = new Number(2, i, d);
                    Number gyroY = new Number(3, i, d1);
                    Number gyroZ = new Number(4, i, d2);
                    Number accN = new Number(5, i, d3);
                    Label horaZ = new Label(6, i, hora);
                    Number cryfisN = new Number(7, i, new Double(crayfis));
                    Number batN = new Number(8, i, new Double(batery));
                    Number packN = new Number(9, i, new Double(paquete));
                    i=i+1;
                    wsheet.addCell(idN);
                    wsheet.addCell(gpsN);
                    wsheet.addCell(gyroX);
                    wsheet.addCell(gyroY);
                    wsheet.addCell(gyroZ);
                    wsheet.addCell(accN);
                    wsheet.addCell(horaZ);
                    wsheet.addCell(cryfisN);
                    wsheet.addCell(batN);
                    wsheet.addCell(packN);









                }
                wworkbook.write();
                wworkbook.close();


                in.close();
            } catch (IOException e) {
            }
        }
```

Con ésta aplicación tendremos a la entrada un archivo de éste tipo

```
1 null 0.005844116,4.7302246E-4,0.0018615723  0.04208374  09:58:44 0 100 1

1 null 0.0077209473,-0.005355835,3.0517578E-5  0.07644653  09:58:54 0 100 2

1 null 0.019851685,-0.027130127,0.010986328  -0.30148315  09:59:04 0 100 3

1 null 0.0048980713,-4.272461E-4,0.0035095215  0.23474121  09:59:14 0 100 4

1 null 0.027328491,-0.0014953613,-0.0018310547  0.14750671  09:59:24 0 100 5
```

Y a la salida

	A	B	C	D	E	F	G	H	I	J
1	Id Smartpl	GPS	Gyro X	Gyro Y	Gyro Z	Accelerom	Time	Cryfis	Battery	Package
2	1	null	0,005844	0,000473	0,001862	0,042084	09:58:44	0	100	1
3	1	null	0,007721	-0,00536	3,05E-05	0,076447	09:58:54	0	100	2
4	1	null	0,019852	-0,02713	0,010986	-0,30148	09:59:04	0	100	3
5	1	null	0,004898	-0,00043	0,00351	0,234741	09:59:14	0	100	4

6.4.2 Análisis de los datos recopilados por los smartphones

Durante la campaña de lanzamiento, tuvimos que realizar algunos cambios:

Tuvimos que eliminar el bucle de sockets que se abría cada 5 minutos, ya que después de 2 horas de ejecución de la aplicación, nos dimos cuenta de que se quedaba bloqueado.

Tuvimos que modificar por lo tanto el software para la conexión a 2 y asegurarnos así la conexión

También fue necesario establecer un protocolo en la conexión de los elementos, ya que los smartphones no eran capaces de obtener una dirección IP del router a menos que éste estuviese conectado y estable antes de lanzar la aplicación.

Los datos que recogimos de los smartphones tras el experimento no fueron todos los que se esperaban.

- La aplicación para la toma de datos de rayos cósmicos (*CRAYFIS*) parece no estar preparada para tomar medidas en las condiciones en las que se desarrolló el experimento, ya que no obtuvimos datos. También es posible que los smartphones estuvieran demasiado aislados con las cajas metálicas en las que se encontraban.
- Lo mismo sucedió con información del GPS. No pudimos obtener datos debido al aislamiento producido por las cajas.
- El resto de medidas fueron tomadas correctamente y almacenadas

Por otra parte, sólo uno de los smartphones funcionó correctamente durante el vuelo, y estuvo tomando medidas durante todo el tiempo que duró la batería. El segundo smartphone envió sólo un par de trazas, y a pesar de poder hacer un reinicio de forma remota, nos aconsejaron no hacerlo mientras el otro teléfono estuviera enviando trazas.

A pesar de que pusimos cargadores inalámbricos para mantener la batería del teléfono al 100%, fue necesario sacar los teléfonos y recargarlos antes de cada test

Éste fue el principal problema de los smartphones, que sólo pudimos obtener medidas durante una hora.

Además, los adaptadores Ethernet que tuvimos que utilizar consumían batería directamente del teléfono, por lo que también se redujo el tiempo de vida de éstos.

GIRÓSCOPO:

Las medidas del giróscopo se pueden interpretar de acuerdo a la siguiente orientación:

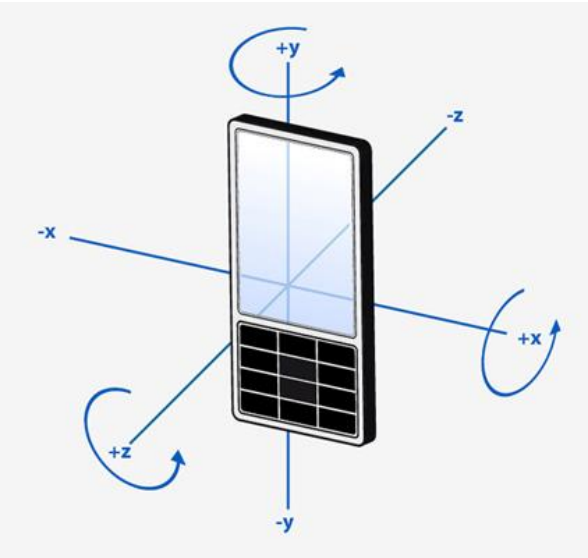


Fig. 6.4.2-1: Orientación del giróscopo

Dado que los smartphones estaban colocados horizontalmente y con la pantalla hacia abajo, la orientación real del experimento es la siguiente:

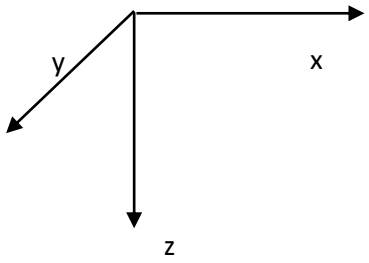
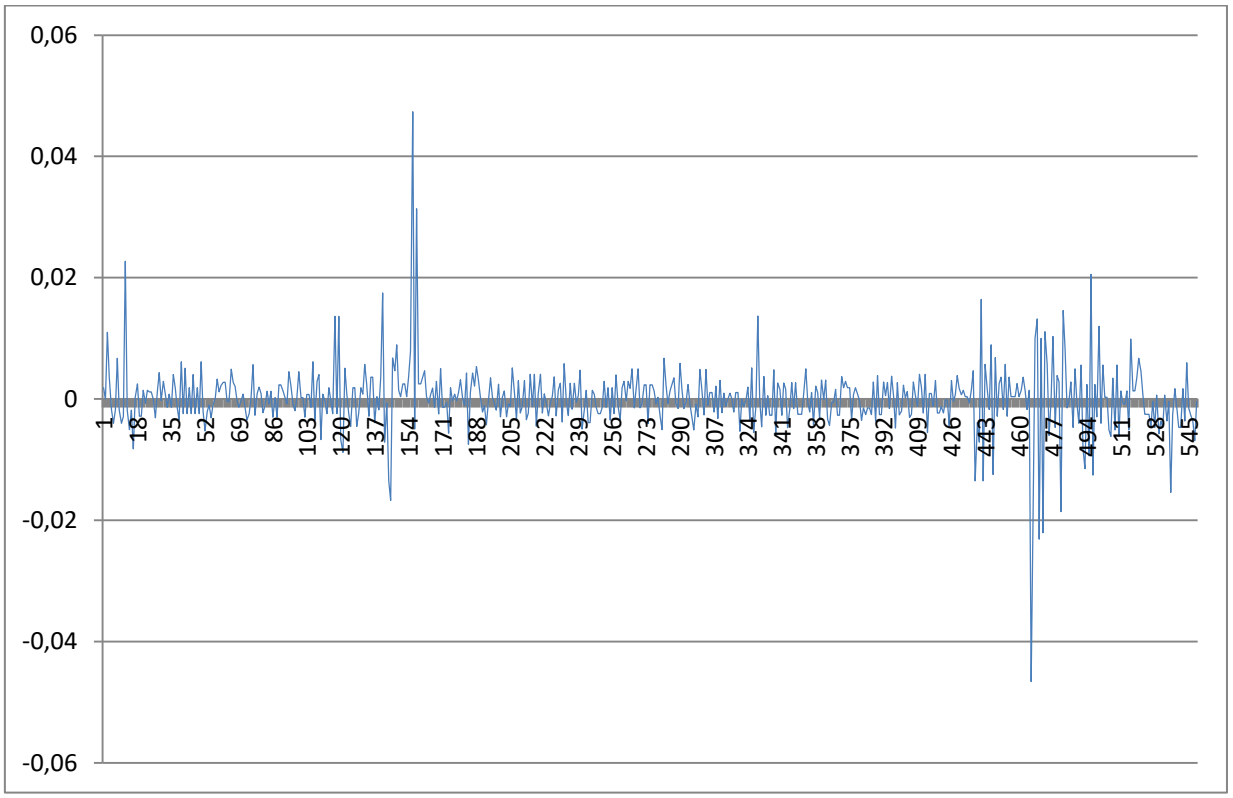


Fig. 6.4.2-2: Ejes de referencia en los smartphones

Los valores mostrados a continuación se corresponden con las medidas tomadas durante el tiempo de preparación y vuelo.

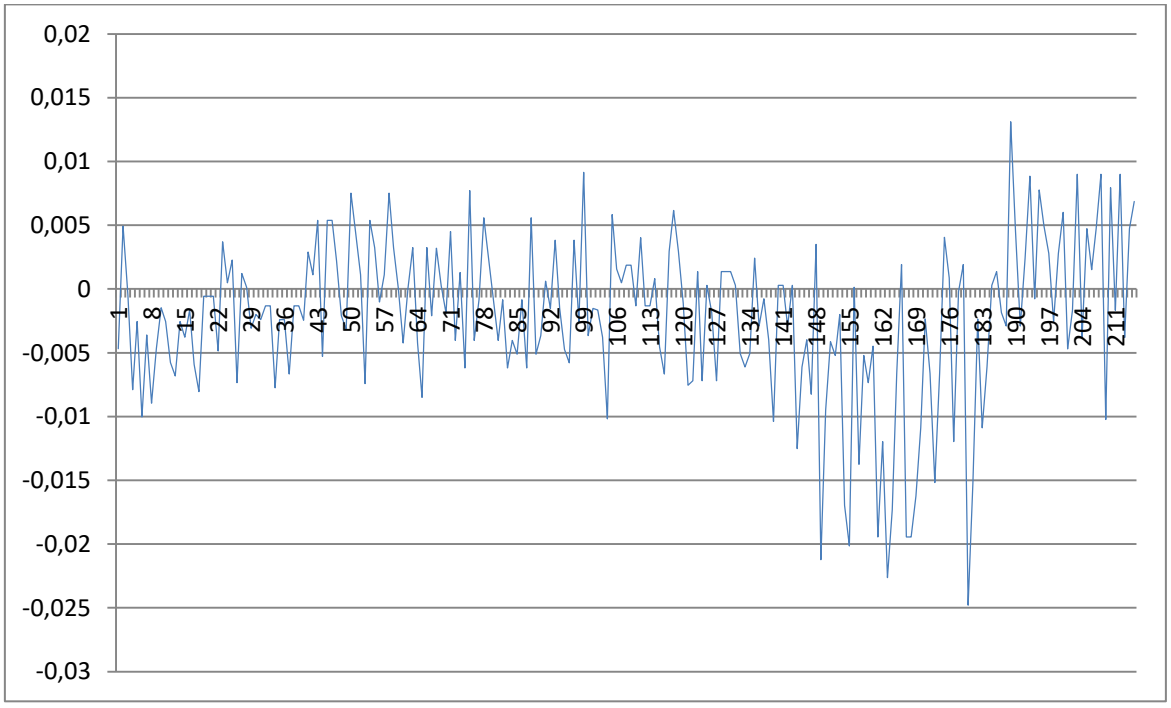
En lugar de utilizar tiempo para mostrar las gráficas, se va a utilizar número de paquetes. Se recibía un paquete cada 10 segundos, y el número de paquete en el momento del lanzamiento es el 549.

Eje-Z



Gráfica 6.4.2-1: variación Z vs número de paquetes

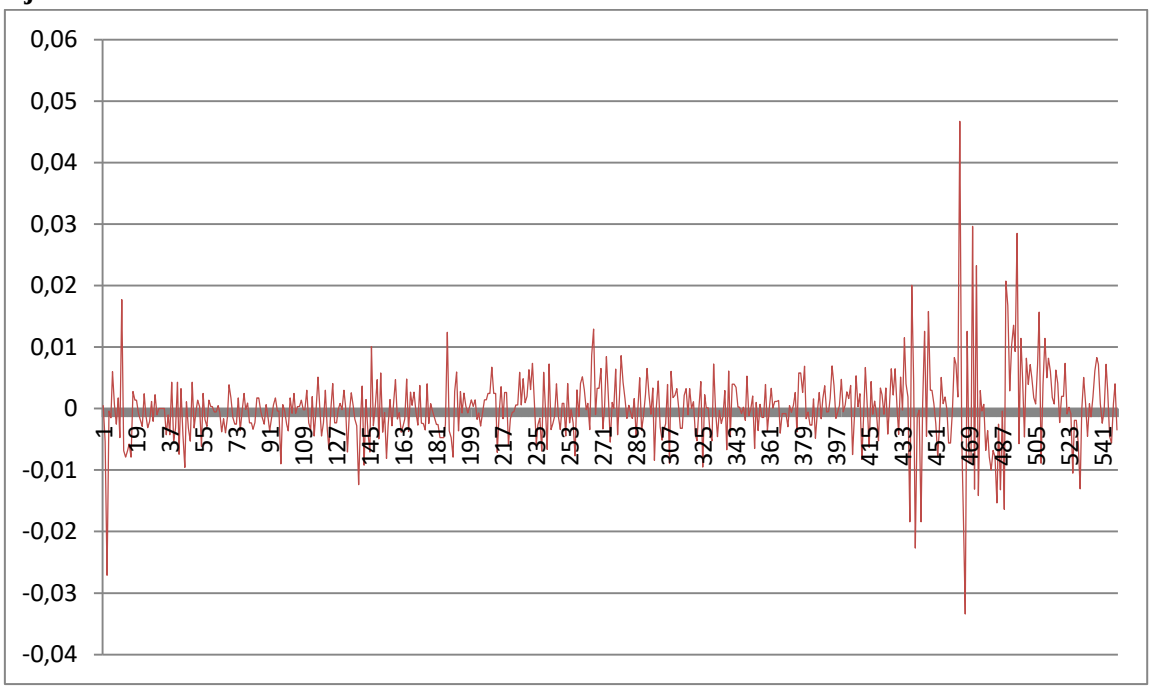
Tiempo de preparación (hasta 11:30:04)



Gráfica 6.4.2-2: variación Z vs número de paquetes

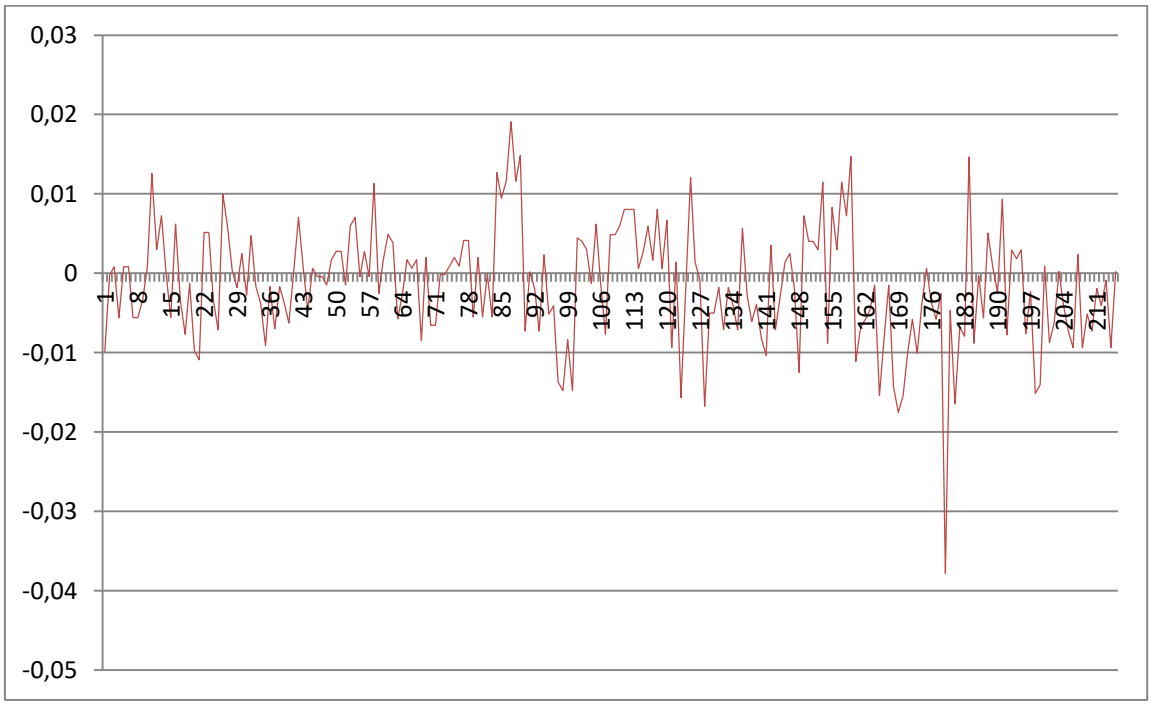
Tiempo de vuelo (desde 11:30:14 hasta 12:05:54)

Eje-Y



Gráfica 6.4.2-3: variación Y vs número de paquetes

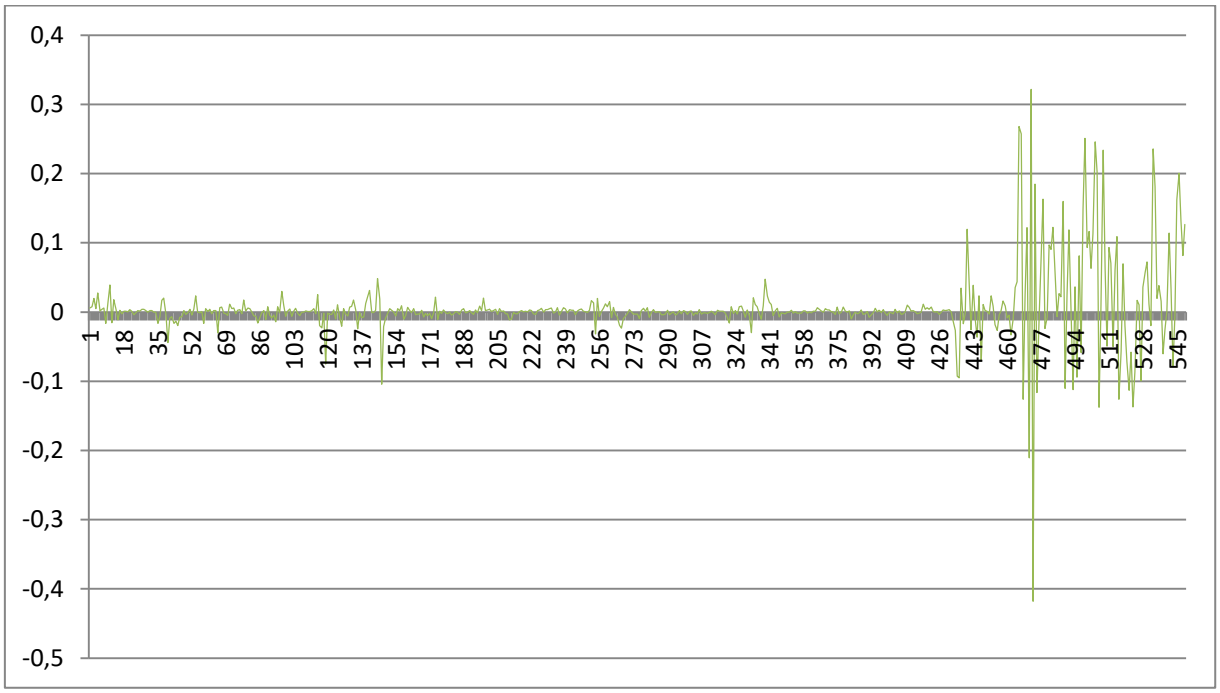
Tiempo de preparación (hasta 11:30:04)



Gráfica 6.4.2-4: variación Y vs número de paquetes

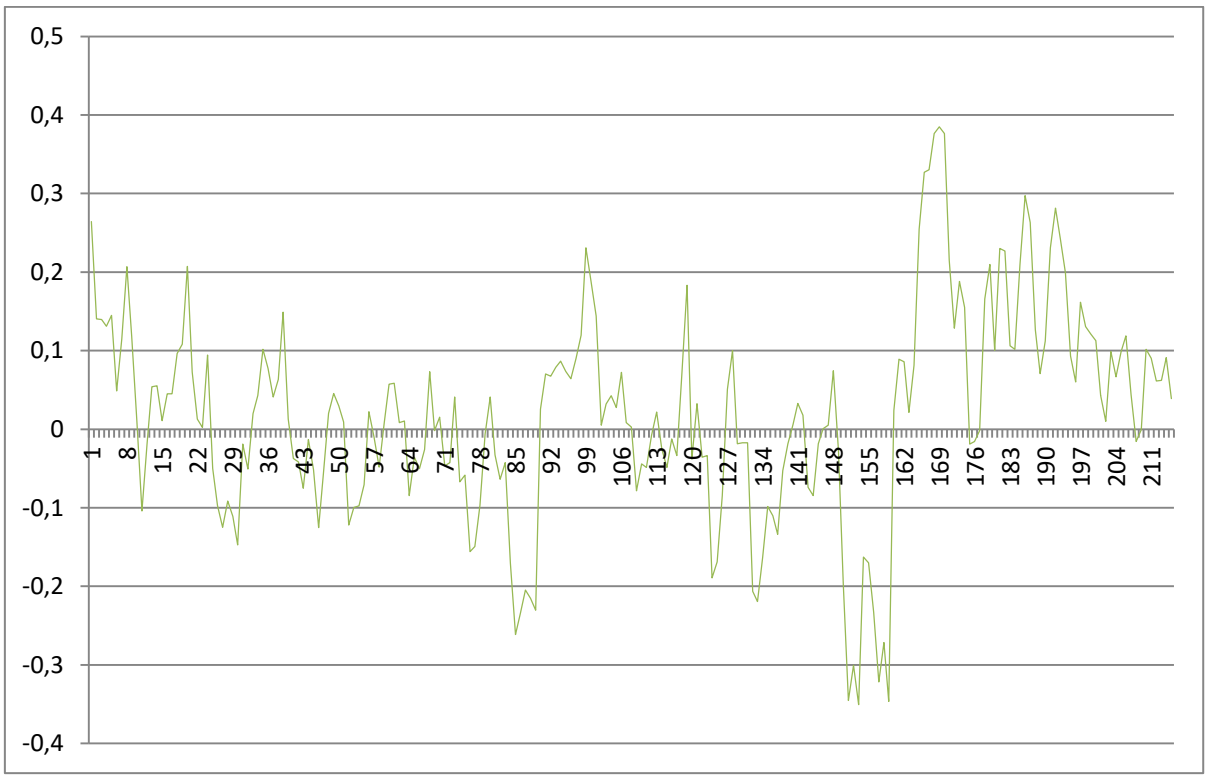
Tiempo de vuelo (desde 11:30:14 hasta 12:05:54)

Eje-X



Gráfica 6.4.2-5: variación X vs número de paquetes

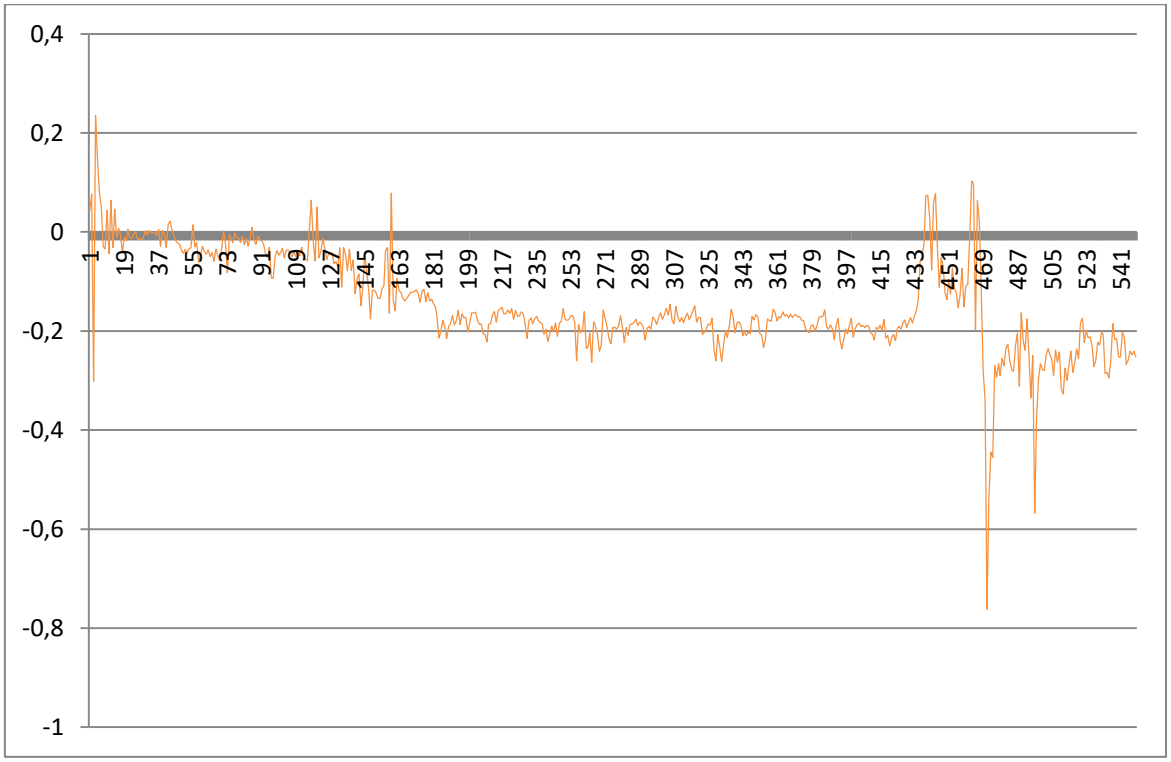
Tiempo de preparación (hasta 11:30:04)



Gráfica 6.4.2-6: variación X vs número de paquetes

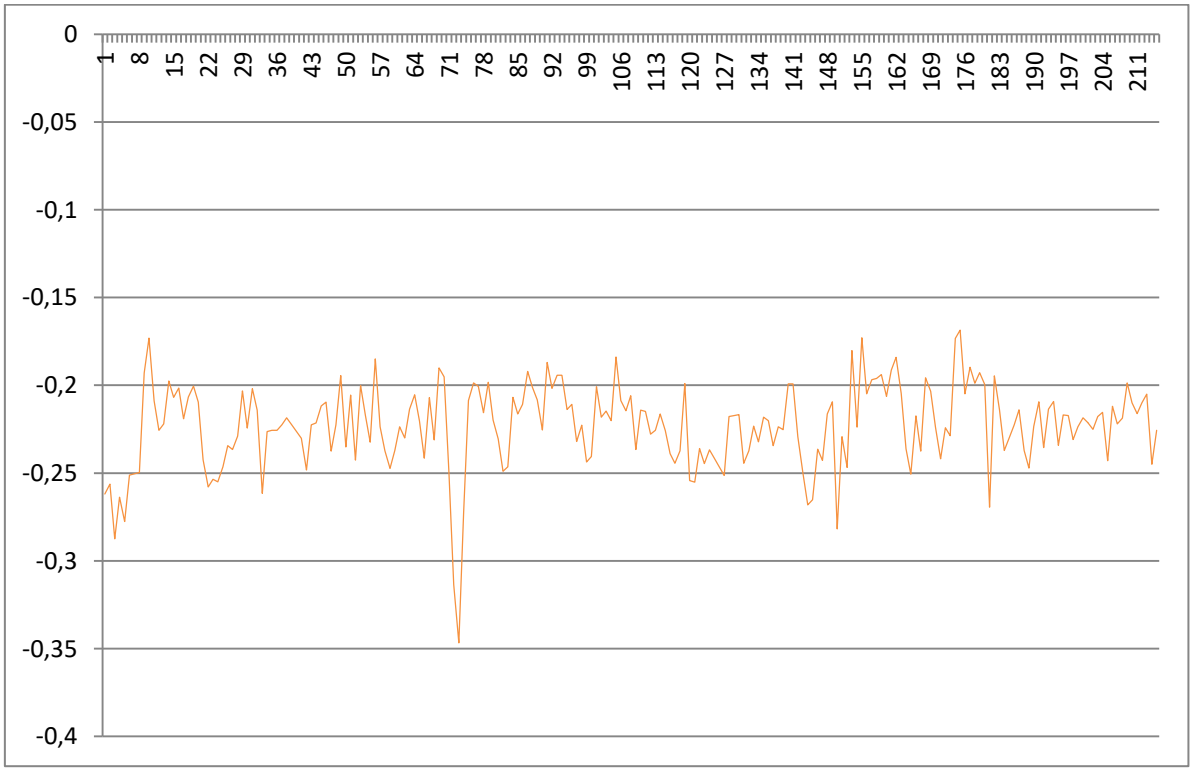
Tiempo de vuelo (desde 11:30:14 hasta 12:05:54)

ACELERACIÓN:



Gráfica 6.4.2-7: Aceleración vs número de paquetes

Tiempo de preparación (hasta 11:30:04)



Gráfica 6.4.2-8: Aceleración vs número de paquetes

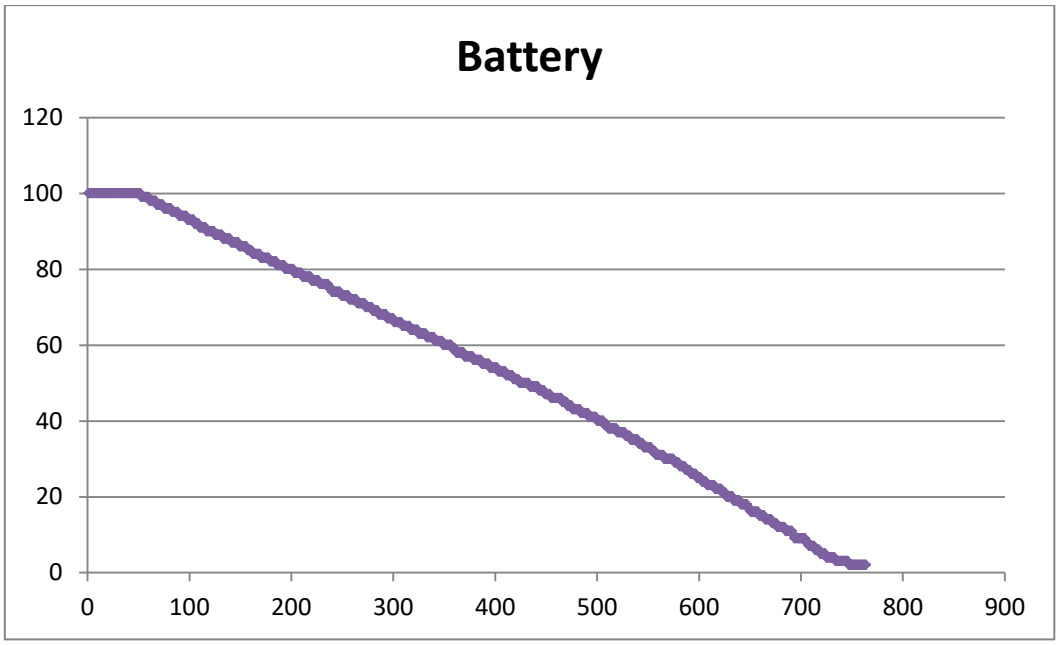
Tiempo de vuelo (desde 11:30:14 hasta 12:05:54)

Como hemos dicho antes, los smartphones se encontraban con la pantalla hacia abajo, por lo que los valores negativos de aceleración en la gráfica representan una aceleración positiva.

Podemos ver que a medida que se acerca el comienzo del tiempo de velo, la aceleración es mayor.

NIVEL DE BATERÍA:

Obviamente, la gráfica para el nivel de batería no tiene ningún tipo de oscilación, ya que no se disponía de ninguna fuente que pudiese alimentar los smartphones en la góndola



Gráfica 6.4.2-9: Nivel de batería vs número de paquetes

7. Tareas de outreach

A parte de las tareas de programación y configuración, también he realizado tareas que estaban destinadas al responsable de outreach, ya que se tuvieron que repartir entre los miembros del grupo cuando la persona encargada dejó el grupo.

Entre las tareas realizadas se incluye, principalmente la configuración y publicación de contenido de la página de Wordpress del proyecto, publicación de contenido en Facebook y Youtube, y toma de fotos, video del lanzamiento y edición de video.

7.1 Configuración y publicación de contenido en Wordpress

La configuración de Wordpress se realizó sobre un host proporcionado por la Universidad de Sevilla.

http://woden.us.es/spade/

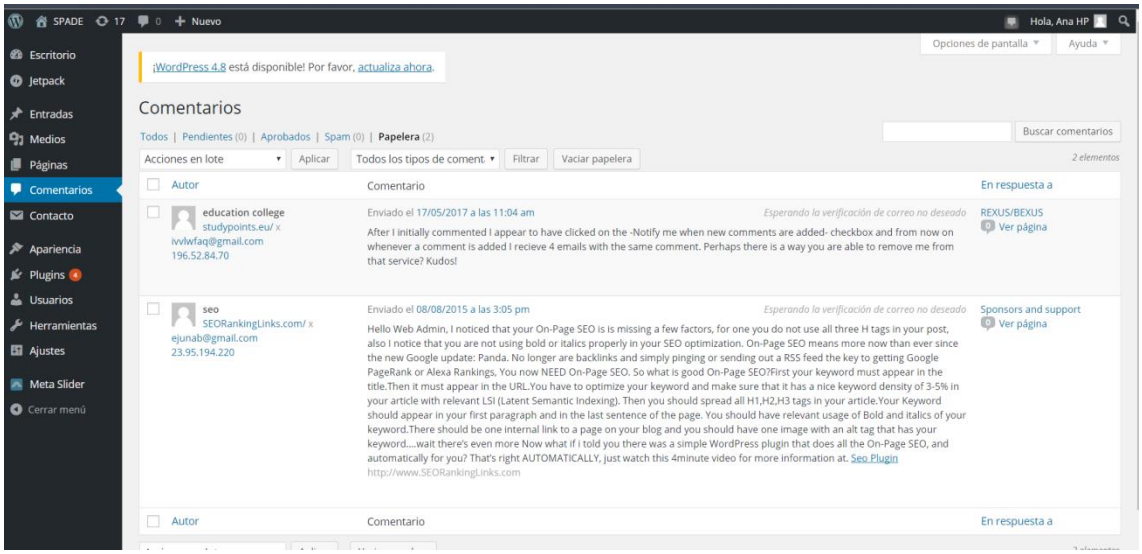


Fig. 7.1-1: Página de Wordpress en modo edición

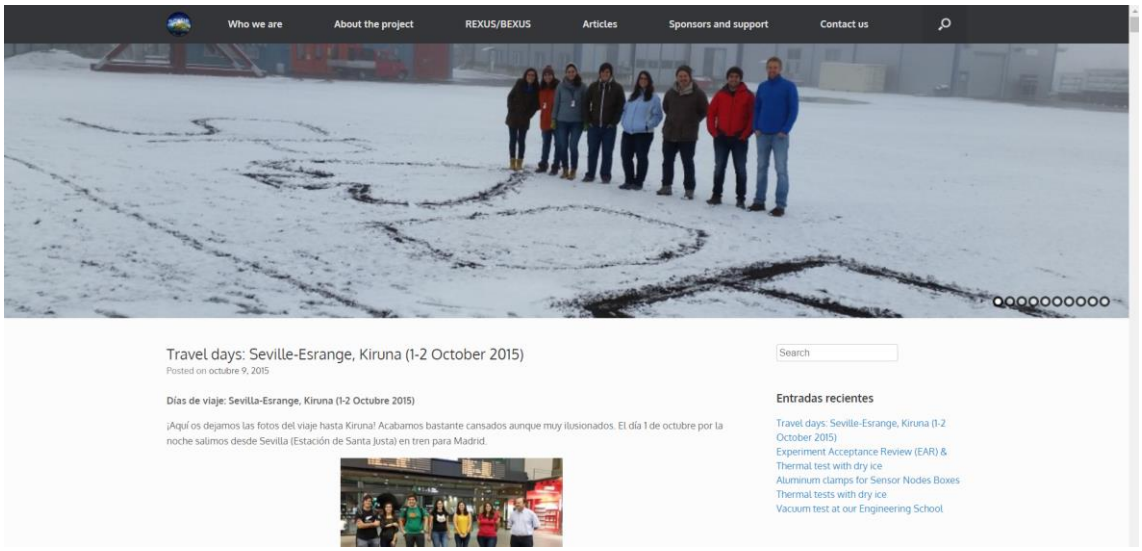


Fig. 7.1-2: Vista pública de la página

7.2 Publicación de contenido en la página de Facebook

https://www.facebook.com/projectspade/

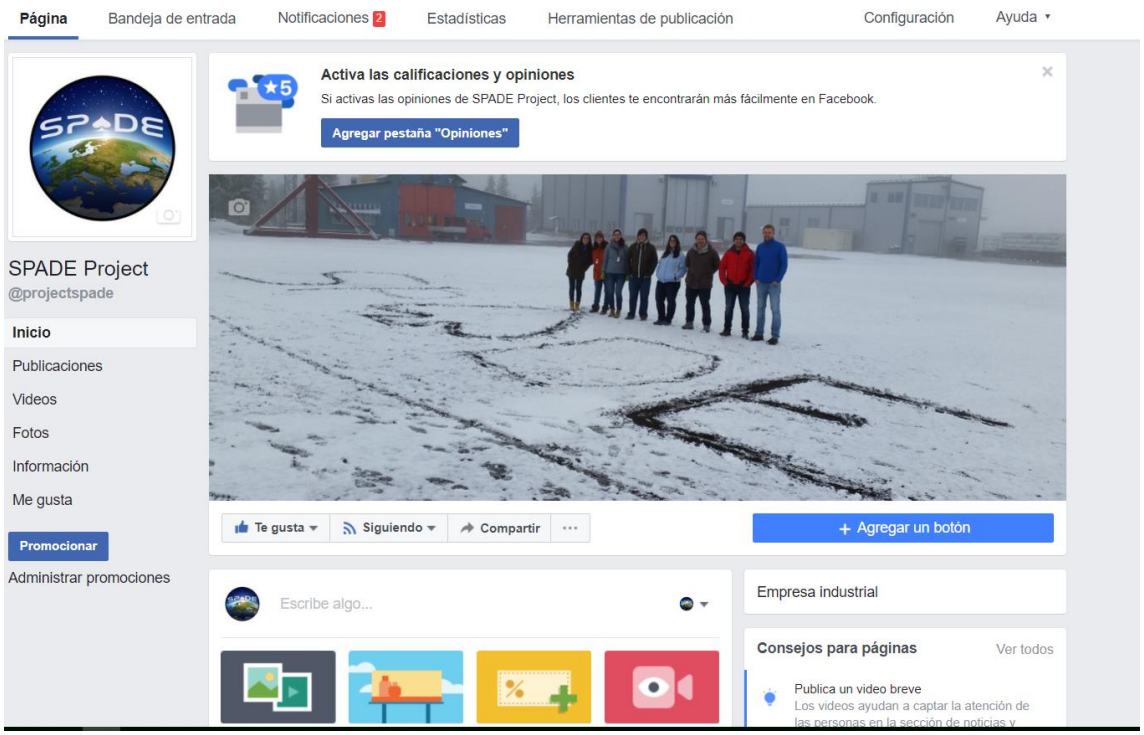


Fig. 7.2-1: Vista de administrador de la página de Facebook

7.3 Vídeos en Youtube

https://www.youtube.com/watch?v=aJhWCBj5HYw

https://www.youtube.com/watch?v=gTj_3ibbr3c

https://www.youtube.com/watch?v=jfcQ48Rvs5k&t=19s

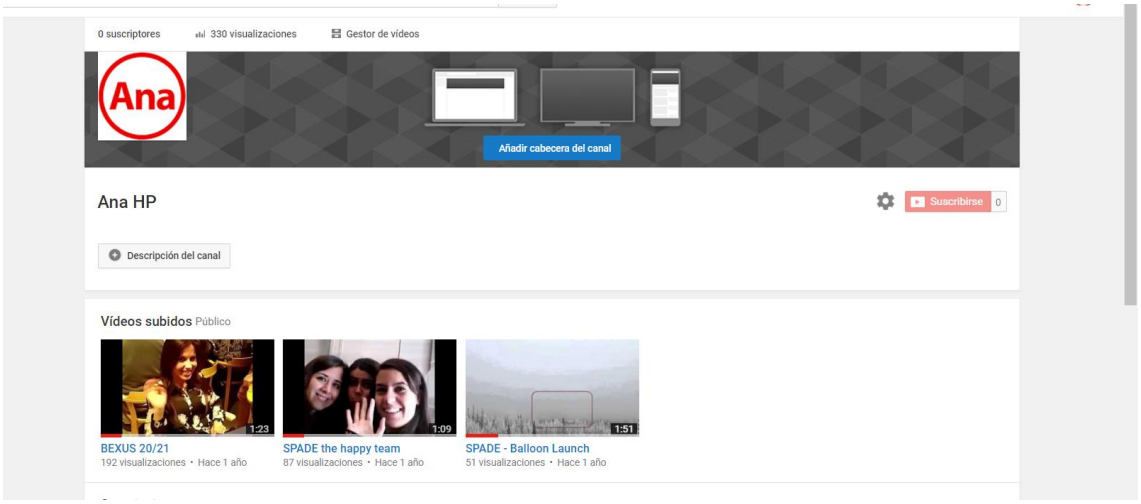


Fig. 7.3-1: Mi canal personal de Youtube en con las publicaciones de la campaña

Canal *SPADE*

https://www.youtube.com/watch?v=DdiPTLUov1k&feature=youtu.be

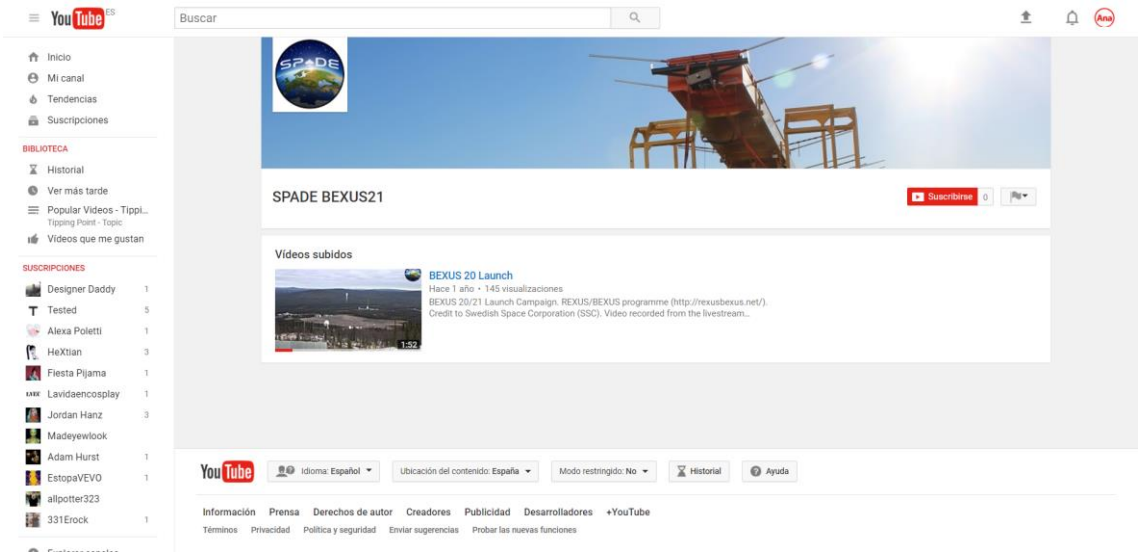


Fig. 7.3-2: Canal de Youtube de *SPADE*

Para la edición de video se ha utilizado Camtasia Studio

Fig. 7.3-3: Vista de edición de Camtasia Studio

8. Conclusiones y lecciones aprendidas

En éste proyecto he aprendido en muchos ámbitos, no se ha tratado de un desarrollo de una aplicación más:

- Desarrollo de una aplicación sujeta a estrictas restricciones
- Trabajo dependiente de otros miembros y necesidad de integración del software con otra plataforma
- Coordinación con un equipo multidisciplinar con formas muy diferentes de plantear problemas y soluciones.
- Creación de un software con una complejidad considerable sin apenas soporte externo.
- Adecuación de los sistemas operativos de los smartphones para el correcto funcionamiento del experimento
- Necesidad de solucionar problemas bajo presión
- Protocolos de pruebas en experimentos críticos.
- Documentación de un proyecto (SED)
- Si algo puede salir mal, saldrá mal, y debemos ser capaces de solucionarlo.

La experiencia que ha supuesto todo el proceso de desarrollo, y la campaña de lanzamiento ha sido una tarea difícil pero muy gratificante. La oportunidad de trabajar en un proyecto que finalmente ha podido volar en el *BEXUS* supone una gran satisfacción.

Merece la pena el tiempo dedicado y el esfuerzo realizado a pesar de que el resultado obtenido no haya sido todo lo bueno que hubiésemos deseado.

Hemos conocido y trabajado con personas de muchos lugares diferentes en un ambiente colaborativo entre equipos y con personal altamente cualificado de la ESA y otras instituciones, que nos han enseñado mucho. Hemos contado con el apoyo de la Universidad de Sevilla y de profesores que se han implicado en éste proyecto con nosotros. A todos ellos, gracias por la experiencia.

Fig. 8.1: Personal y alumnos que participaron en la campaña BEXUS 20/21 con el HERCULES

SPADE - Universidad de Sevilla - BX21
Beatriz Parra, Pedro Herrera, Maria Teresa Tabares, Ana Haro Perez, Maria de la Cruz Sanchez Gomes, Diego Lopez

Fig. 8.2: Integrantes de *SPADE* tras la recogida del experimento