UNIVERSITAT POLITÈCNICA DE VALÈNCIA

INSTITUTO DE DISEÑO Y FABRICACIÓN

Escuela Técnica Superior de Ingeniería del Diseño

Escuela Técnica Superior de Ingeniería del Diseño

Grado en Ingeniería Aeroespacial

Entrega y Curso académico: Septiembre de 2017 (2016-2017)

TRABAJO DE FIN DE GRADO:

# Desarrollo de una aplicación industrial de generación de misiones autónomas con drones para el mantenimiento de grandes aerogeneradores

Proyecto realizado por: Omar Navarro Ivorra

---

Supervisado por:
Juan Antonio García Manrique

**Resumen**

A lo largo de la siguiente memoria, se va a observar el trabajo que conlleva desarrollar una aplicación para un dispositivo móvil , pasando por todos los puntos necesarios, como el análisis del lenguaje con el que se está trabajando, sus precedentes en el apartado "Estado del arte", y todo el potencial que se puede extraer de dicha herramienta.

No se debe olvidar que el trabajo desarrollado tiene por nombre, "Desarrollo de una aplicación industrial de generación de misiones autónomas con drones para el mantenimiento de grandes aerogeneradores", por lo que nunca se van a perder de vista aspectos como la normativa vigente de los drones, según el Ministerio de Fomento y AESA (Agencia Estatal de Seguridad Aérea), y otros procesos que se deben realizar a la hora de publicar nuestra aplicación en la tienda oficial de *Google* para *Android*, los cuales se deben seguir para no incurrir en ilegalidades.

Seguir estos procesos correctamente ayudará a tener una visión totalmente comercial e industrial para la aplicación.

Además, como no podía ser de otro modo, también se hará hincapié en el dispositivo en concreto, el dron, mostrando información general del mismo y más específica como el funcionamiento del GPS o estabilización, así como las conexiones y procesos necesarios que han de desarrollarse para poder aunar el dron y aplicación de móvil en un solo conjunto, que funcione de forma óptima.

Por último, no se podría iniciar este trabajo de análisis de aerogeneradores, sin dar la información necesaria sobre las ventajas que esta aplicación va a proporcionar, respecto al actual método de mantenimiento con operarios y maquinaria pesada.

Se concluirá al final con un apartado de futuras mejoras sobre nuestro trabajo, puesto que no se va a tratar de una aplicación de código finalizado, sino más bien, una fase beta abierta a incluirle todas las funcionalidades posibles, con la motivación de que en un futuro sea totalmente funcional y útil.

**Resum**

Al llarg d'aquesta memòria, s'observarà el treball que comporta desenvolupar una aplicació per un dispositiu mòbil, passant per tots els punts necessaris, com l'anàlisi del llenguatge amb el qual s'està treballant, els seus precedents en l'apartat "Estat de l'art", i tot el potencial que es pot extraure d'aquesta ferramenta.

No s'ha d'oblidar que el treball desenvolupat té per nom, "Desenvolupament d'una aplicació industrial de generació de missions autònomes amb drons pel manteniment de grans aerogeneradors", pel que mai es perdrà de vista aspectes com la normativa vigent dels drons del Ministeri de Foment i AESA (Agència Estatal de Seguretat Aèrea), i altres processos que s'hauran de realitzar a l'hora de publicar la nostra aplicació en la tenda oficial de *Google* per *Android*, els quals s'han de seguir per no caure en il·legalitats.

Seguir aquestos processos correctament ajudarà a tindre una visió totalment comercial i industrial per l'aplicació.

A més a més, com no pot ser de cap altra manera, també es posarà èmfasi en el dispositiu en concret, el dron, mostrant informació general del mateix i més concreta com el funcionament del GPS o estabilització, així com les connexions i processos necessaris, que s'han de desenvolupar per poder juntar el dron i l'aplicació de mòbil en un sol conjunt que funcione de forma òptima.

Finalment, no podrem començar aquest treball d'anàlisi d'aerogeneradors , sense proporcionar la informació necessaria sobre els avantatges que aquesta aplicació ens va a donar, respecte a l'actual mètode de manteniment amb operaris i maquinària pesada.

S'acabarà amb un apartat de futures millores sobre el nostre treball, donat que no es tracta d'una aplicació de codi tancat, sino millor dit , una fase beta oberta a incloure-li totes les funcionalitats possibles, amb la motivació de que en un futur siga totalment funcional i útil.

**Summary**

Throughout this memoir, we are going to see the required effort to develop an application for a mobile device, focusing our attention in all the important topics, as it is the analysis of the language we are working with, its precedents located in the section "State of Art", and the whole potential that can be extracted from this tool.

We mustn't forget that this developed work is named, "Develop an industrial auto maker mission application with drones for big wind turbine maintenance", so that we must never forget things like the actual drone's normative by the "Ministerio de Fomento" and AESA ( Estatal Aviation Security Agency, what accomplishes EASA's (European Aviation Security Agency) rules in Spain), and other processes that must be done in order to post our application in the official store of *Google* for *Android* , which must be followed to avoid any illegality.

Following these processes successfully will help us to have a completely industial and commercial vision for our application.

Moreover, as it couldn't happen in another way , we shall focus in the specific device too, the drone, showing general information related to it, and showing more specific needed information about GPS and stabilization, as well as the connections and required processes, that must be developed to fuse drone and mobile application in a single set which will work optimally.

At last, the work of wind turbine analysis couldn't be strated, without giving enough information about the advantages that this application is going to provide, with respect to the current maintenance method with operators and heavy machinery.

To sum up, it will be shown a section about future improvements about our job, due to that this is not a closed source code, but rather, it is a beta application ready to include it all the possible functionalities, with the purpose of achieveving in a future a completely functional and useful application.

**Agradecimientos**

Este trabajo no se habría podido realizar sin los útiles de gran calidad e instalaciones proporcionadas por la *Universidad Politécnica de Valencia*, así como sin la ayuda del *Instituto de Diseño y Fabricación*, y mi tutor Juan Antonio García Manrique, el cual me trajo la posibilidad de crecer en otros ámbitos no desarrollados en profundidad en esta carrera, como es la programación en lenguajes más genéricos, fuera de los programas comunes orientados a resolución de problemas de cálculo matemático y/o matricial.

# Índice

# Índice de figuras

# 1 Introducción

Como se ha comprendido durante el resumen, el trabajo consta de una gran carga de programación, por lo que se tendrá que realizar una amplia introducción al lenguaje en los subapartados posteriores.

Además, la motivación inicial como se ha comentado, es inspeccionar aerogeneradores, por lo que se darán solo breves tintes de información sobre la cantidad de problemas de deterioro que puede tener un aerogenerador, puesto que se desarrollará más en profundidad en el apartado de "Mantenimiento de Aerogeneradores".

Como comentario adicional , no realizaremos hasta el apartado final de "Ventajas del mantenimiento con drones", la comparativa del mantenimiento con drones o operarios, puesto que se pretende dar primero una visión global tanto de los drones, sus sistemas y la aplicación, como de los resultados que se han obtenido utilizando este innovador método.

## 1.1 Introducción al proyecto

### 1.1.1 Motivación del proyecto

En esta sección se va a tratar la problemática principal del deterioro de aerogeneradores y su mantenimiento.

Generalmente, en las palas, el principal problema de deterioro es la abrasión por gota de agua en pala, además del de impacto de rayo entre otros que también albergan importancia.

Los aerogeneradores pueden estar operando largos periodos de trabajo sin parar, y en punta de pala se pueden alcanzar velocidades cercanas a los 300km/h , por lo que la fibra de vidrio con matriz de poliéster (o epoxi) puede ser dañada gravemente.

Es posible pensar que estos defectos, como el de un impacto de rayo, puedan verse a simple vista por su gravedad, pero la realidad es que un aerogenerador tiene un tamaño de hasta 100 m en vertical y más de 100 m de diámetro del rotor fácilmente, por lo que en muchas ocasiones no se apreciarán a simple vista.

Por tanto, estamos ante un coloso con una gran superficie a cubrir para su análisis.
Es evidente que, por dificultad que pueda acarrear, será necesario establecer un mantenimiento para evitar problemas mayores, y se nos presentan los siguientes métodos:

- Inspección en tierra:

  Este método tiene la ventaja de no estar sujeto a condiciones climatológicas por realizarse desde tierra , no obstante tendremos que valorar la posibilidad de tomar imágenes más genéricas de la pala , con lo que no tendremos información suficientemente precisa de la misma, o tomar capturas (todo esto con una cámara, foto por foto) de gran precisión mediante el análisis de zonas pequeñas.

  No obstante, respecto al segundo procedimiento, es bastante probable que se necesite una cantidad de tiempo inmensa.
  Podemos asegurar que esta idea parte de una base fiable, intentando imaginar cómo sería analizar foto por foto un gran aerogenerador , que

es el nombre y motivación de nuestro proyecto, como el que se instalará en Dinamarca [31]:

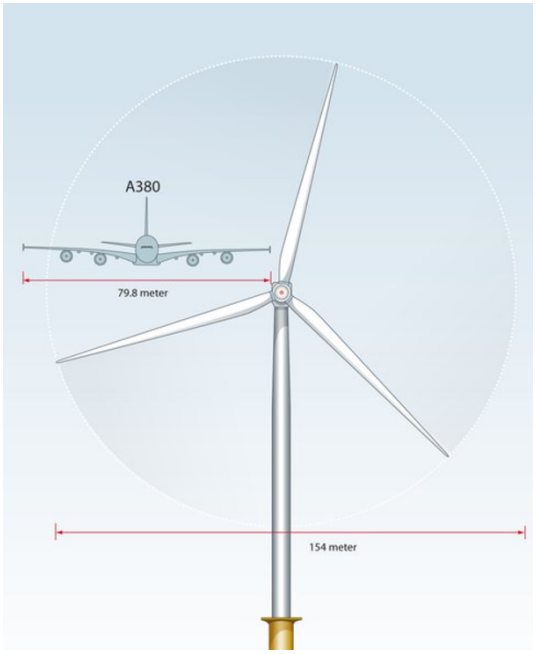



Figura 2: Aerogenerador de 154m de diámetro de rotor

En este momento, ya se podría empezar a valorar otras opciones, puesto que el operario tendría que pasar posiblemente más de una jornada inspeccionando un solo aerogenerador.

- Plataforma de servicio:

Requiere del montaje de una estructura sobre el mismo aerogenerador, la cual contará con numerosas medidas de seguridad y anclaje, por lo que la velocidad de análisis no será su fuerte, y la inversión en la misma tampoco será despreciable:

Figura 3: Plataforma de servicio

- Grúa hidráulica:

  Para este método se tiene una problemática muy similar por no decir igual , es una máquina de gran tamaño que operará a baja velocidad, con protocolos de seguridad concretos , y no a un bajo coste.
  Además, en el supuesto de que el aerogenerador esté en una colina, hecho que se puede observar frecuentemente cuando se recorren las carreteras de nuestro país, este aparato tendría grandes impedimentos en su acceso.

Figura 4: Grúa hidráulica

- Descenso en cuerda:

  Posiblemente este sea el método entre todos los anteriores en el cual vamos a obtener un análisis más "personalizado", por el hecho de que el operario está encima de la pala literalmente:

Figura 5: Descenso en cuerda alrededor de la pala

  No obstante y a pesar de las medidas de seguridad, tendremos siempre presente el factor de riesgo, y en otros aspectos menos importantes que la seguridad del operario, pero también de gran importancia, el hecho de que la remuneración del operario será alta por los incentivos de peligrosidad.

Como se ha observado, no se podría apostar por ninguno de los métodos como veloz y preciso a la vez, y esto sin incluir los temas económicos y de seguridad que también son muy importantes, sino los que más.

De nuevo, recordamos, que la motivación de este proyecto es poder analizar aerogeneradores de gran tamaño como el visto anteriormente, de un **modo rápido, barato, preciso y seguro, y esto se conseguirá mediante los drones.**

## 1.2 Introducción al lenguaje *Java*

En la siguiente sección, se va a tratar de forma breve conceptos de *Android* que, no son esenciales para poder programar, pero sí, si lo que se busca es entender correctamente el entorno de programación, para poder exprimir su uso al máximo.

### 1.2.1 ¿Qué es *Android*?

*Android* [41] es un sistema operativo basado en el núcleo Linux.


Figura 6: Icono representativo del sistema *Android*

Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tablets y también para relojes inteligentes, televisores y automóviles.

Es un sistema escrito en base a los sistemas operativos *Java* , C y C++ (pero puesto que Java está influido por estos dos últimos, nos centraremos solo en él).

### 1.2.2 ¿Qué es *Android Studio*?

*Android Studio* es el entorno oficial integrado de desarrollo de aplicaciones para terminales *Android*, **programado nativamente en *Java***.

Ofrece funciones que aumentan la productividad durante la compilación de aplicaciones para *Android*, como las siguientes [34], entre muchas otras:

- Sistema de compilación con *Gradle* flexible (Se explicará en el apartado correspondiente).
- Contiene un emulador rápido con grandes funcionalidades, lo que en muchos casos y para probar cosas sencillas, evita tener que estar en contacto con el terminal el 100 % del tiempo.
- Entorno unificado para abarcar todos los dispositivos a la vez.
- Ejemplos de código a importar y relación con páginas de programación destacadas como *GitHub* [35]. Aunque no es una funcionalidad de *Android*, otra gran página de apoyo será *Stack Overflow* [36].
- Herramientas *Lint*: No es necesario su uso, pero son una especie de "debuggers" en tiempo real, es decir, nos facilitan información sobre qué errores estamos comentiendo y como solucionarlos.
- Compatibilidad con otros lenguajes como C++.

Además de las funcionalidades, también merece la pena explicar la estructura y archivos genéricos para comprender y saber por donde debemos movernos.

*Android* tiene una alta integración con otros lenguajes, e infinidad de herramientas para programadores de cierto nivel, no obstante, no se podrá abarcar dichos conceptos en este escrito.

El programa consta de una barra superior común a otros programas, la cual contiene edición de las vistas, crear o cargar archivos, editar el código, apartado de herramientas , pestaña de ayuda...

Debajo de la misma, contiene una barra de funcionalidades rápidas (guardar o limpiar proyecto, ver versiones disponibles, cargar la aplicación) , donde la más evidente es el botón "Play" de color verde, para cargar la aplicación, ya sea en el dispositivo o en el emulador.


Figura 7: Barra de proceso de *Android Studio* y ruta de aplicación

Debajo de esta barra encontramos la ruta donde se encuentra el archivo que estemos tratando.

En el margen izquierdo, en cambio, vamos a encontrar una estructuración distinta según la opción que tengamos activada en el desplegable:

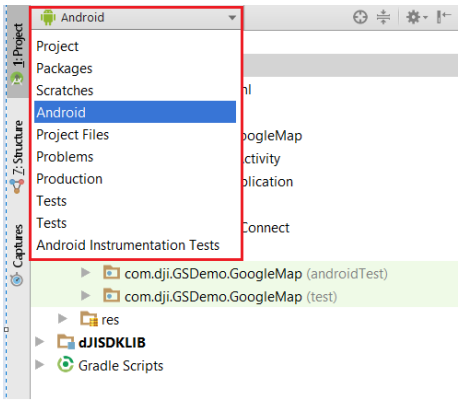



Figura 8: Desplegable de elección de modo de visualización

Para este proyecto, solo se ha requerido activar el modo proyecto una vez, para realizar un análisis sobre los archivos de la librería e intentar comprender mejor aquello que contienen, pero en general y para aplicaciones de un nivel usuario medio, solo utilizaremos la estructuración por defecto, *Android*.

Para finalizar con la estructura de una aplicación en modo de desplegable *Android*, comentaremos los tipos de archivos que se van a encontrar:

- *Android Manifest* [10]
  Cada aplicación de *Android* tiene un archivo denominado *AndroidManifest.xml*. Este archivo se encuentra en la carpeta "manifests" según la estructura *Android*.

El mismo es utilizado por *Android* para verificar que cada componente existe antes de inicializarlo, y es por ello que todas las actividades deben definirse explícitamente en este fichero e indicar cuál será la actividad principal.

En este archivo, los desarrolladores también deben enumerar todos los permisos requeridos necesarios para que la aplicación funcione correctamente, así como contraseñas, en el caso de que la aplicación use funciones especiales externas como es el caso (Servicios de mapa de *Google* y servicios de *DJI*).

En resumen, el archivo *manifest* proporciona información esencial sobre la aplicación al sistema *Android*, que debe ser conocida por el sistema para poder realizar una correcta ejecución del código.

- Archivos de código (*Java*):
  Se caracterizan por el icono azul que veremos a continuación , y se incluyen en la carpeta "java" (Los volveremos a tratar en el apartado de "Conocimientos Previos" con más detalle).

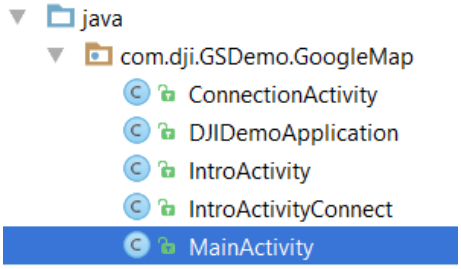



Figura 9: Ubicación de archivos de código

- Archivos gráficos e idiomas:
  Los archivos *layout.xml* son los archivos donde se reflejará nuestro código asociado, mientras que los *string.xml* son los encargados de contener los textos y traducciones (Se tratará en buena medida también en el apartado de "Conocimientos Previos"). Se encuentran en la carpeta "res", los archivos gráficos en "layout", y los de texto en "strings".

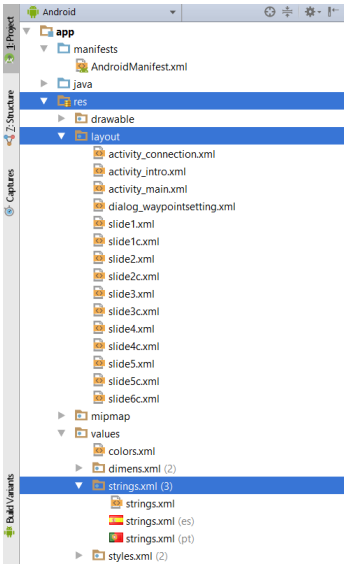



Figura 10: Ubicación de archivos gráficos y cadenas de texto

- Imágenes:
  Ubicaremos las imágenes que se vayan a utilizar en nuestro código en la carpeta "drawable", y las imágenes que vayan a formar el icono de la aplicación en la carpeta "mipmap".

- *Gradle*:
  Aunque todavía no se está en el apartado en que se tratará con más detalle, el *Gradle* hace referencia a las dependencias , versiones, y operativa interna en general de la aplicación.

  No se trata de un apartado en el que se deban realizar cambios significativos frecuentemente , y en esta aplicación se ha accedido al "module:app" para habilitar y declarar la librería y los servicios de *Google*.

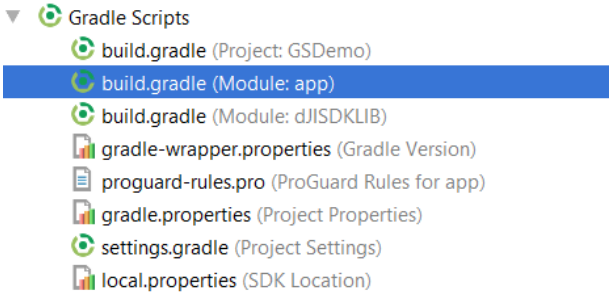



Figura 11: Ubicación del apartado *module:app* en el desplegable del *Gradle*

### 1.2.3 ¿Qué es *Java*?

*Java* [37] es un lenguaje de programación que apareció en 1995, con el propósito de unificar la creación de aplicaciones (no necesariamente de móvil, entre otras cosas porque en aquel entonces era imposible) y no tener que recompilarlas en cada cambio de dispositivo.

El hecho de que este lenguaje sea uno de los más populares de programación desde 2012, da a entender tanto las funcionalidades como la inmensidad de conceptos que contiene, por lo que se va a dar una visión general de cuáles son los aspectos más curiosos e importantes del mismo:

- Se trata de un lenguaje especialmente "sencillo" en la creación de aplicaciones cliente-servidor.

  Un ejemplo sería una aplicación de cálculo numérico, que por potencia del terminal , no nos interesase realizar las operaciones en el móvil. Simplemente enviaríamos los datos mediante la estructura y comandos pertinentes al servidor *Java* (Pc) desde el cliente *Android* (Móvil), y luego devolveríamos las soluciones al terminal.

- Está influenciado en gran medida por el lenguaje C y C++, pero no tiene acceso a niveles de programación tan bajos (Puesto que es un lenguaje de programación de "alto" nivel).

  En la jerga de programación , "bajo" se refiere a nivel de dificultad por ausencia de infraestructuras o macros para realizar ciertas acciones, puesto que se programa en contacto directo con el hardware , específico para cada caso.

  Un ejemplo muy acusado, sería por ejemplo, si se estuviese en un nivel tan bajo que se tuviese que enseñar al dispositivo a sumar programándolo, acción que incorporan todos los lenguajes de programación "altos" (Como los de desarrollo de aplicaciones, están en el nivel más alto, son los que más herramientas prediseñadas y simples ofrecen) con el más que conocido signo +.

  Este tipo de lenguaje es utilizado por ejemplo para los manejadores de dispositivo o *drivers* [38], los cuales conectan con partes específicas de software periférico.

- Es un sistema multiplataforma [39].
  Una plataforma es una combinación de software y hardware donde pueden operar aplicaciones de software.

  Para ejemplificar este complejo concepto, *Java* se puede ejecutar en plataformas de software como *Microsoft Windows, Linux, Eclipse,* ***Android***... Esto es debido a que este lenguaje contiene su propia máquina virtual donde compila la información, hecho que lo convierte en multiplataforma.

- Además de perseguir la filosofía de ser un sistema robusto y fiable en sistemas de red, y multiplataforma, una de sus características más peculiares es la programación orientada a objetos.

  Con la finalidad de no ocupar gran cantidad de este escrito en la explicación del lenguaje *Java*, y poder meternos ya de lleno en *Android Studio* y otros temas, lo definiremos del siguiente modo:

  Los objetos son clases predefinidas que jerarquizan el sistema de programación, es decir, son entes genéricos de software con ciertas propiedades y funcionalidades, que pueden ser reutilizados, a su vez pueden extenderse en subclases, y proveen de una base más estable al lenguaje a la hora de crear grandes proyectos.

  Se puede aportar un ejemplo clarificador para intentar comprender lo que se ha leído:

  Si la clase "vehículo" fuese un aspecto muy importante en la programación, se crearía de ella un objeto. Este objeto, tendría **solo** las características más genéricas (cuatro ruedas, carrocería, motor ).

  A su vez, se extendería en subclases menos genéricas (que particularizarían nuestro código a partir de la clase genérica), como marca del vehículo, cilindrada de dicho motor, tipos de faros que contiene, color de carrocería etc.

De este modo, podría crear una aplicación de coches de montaña , o coches de Fórmula 1, utilizando la misma clase "vehículo", puesto que se adapta perfectamente por ser tan genérica, y ya la particularizaría para cada caso en concreto (para coches de montaña, la clase vehículo se extendería en subclases llamadas, por ejemplo, todoterreno , que a su vez tendrían sus propias características internas, y para los de Fórmula 1, en la subclase llamada competición, por ejemplo).

### 1.2.4 ¿Qué es el *Gradle*?

*Gradle* es un sistema de construcción y automatización de compilación de código abierto. Esta herramienta sirve para declarar la configuración del proyecto y para determinar el orden en que se pueden ejecutar las tareas.

*Gradle* fue diseñado para las construcciones de proyectos múltiples que puedan llegar a ser bastante grandes.
Soporta construcciones incrementales, mediante la determinación inteligente de qué partes del árbol de construcción están actualizadas, de modo que cualquier tarea dependiente de esas partes no tenga que ser reejecutada.

A nivel usuario y para esta aplicación, no se verá un alcance mayor que un apartado donde se define la configuración más genérica de la aplicación, como permisos y versiones de herramientas, y se habilitan e incorporan herramientas externas y librerías.

Las ventajas más importantes de esta potente herramienta son [40] la fácil y completa compilación que podemos realizar con ella, permitida desde la consola incluso, y también lo fácil que resulta crear diferentes versiones para las aplicaciones (para móvil, tableta, televisió, de pago, gratis...).

### 1.2.5 ¿Qué es una API?

La interfaz de programación de aplicaciones (API) es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Con el tema de abstracción, nos referimos a que proporcionan un lenguaje de alto nivel (este concepto se debe haber comprendido en el apartado "¿Qué es *Java*?"), por lo que las API, son macros internos (No se accede a ellos para modificarlos, solo se usan sus funcionalidades) con ciertas herramientas y funcionalidades predefinidas que no tendremos que reprogramar cada vez.

Por otra parte, se va a comentar la API mínima con la que se podría utilizar esta aplicación.
Nuestra aplicación esta diseñada con un sistema operativo requerido mínimo KitKat 4.4, y nivel de API 19 (Que según *Android Studio* cubre el 80 % del mercado, reforzando la visión de difusión industrial), la cual incluye respecto a todas las anteriores las funciones que se muestran a continuación (entre otras que no se incluyen por su complejidad, como mejoras en la máquina virtual para proporcionar mayor velocidad):

- El principal objetivo de la versión 4.4 (con API 19) es hacer que *Android* esté disponible en una gama aún más amplia de dispositivos, incluyendo aquellos con tamaños de memoria RAM de solo 512 MB. Para ello, todos los componentes principales de *Android* han sido recortados para reducir sus requerimientos de memoria.

- El modo de inmersión en pantalla completa oculta todas las interfaces del sistema (barras de navegación y de estado), de tal manera que una aplicación puede aprovechar el tamaño de la pantalla completa más eficientemente.

- Incluye varios protocolos sobre *Bluetooth* y soporte para mandos infrarrojos.

- Se mejoran los sensores para disminuir su consumo y se incorpora un sensor contador de pasos.

- Se facilita el acceso de las aplicaciones a la nube con un nuevo marco de almacenamiento.

- Se añade un *Content Provider* para gestionar los SMS (veremos qué es un "content" en el subapartado "conocimientos previos" ).

### 1.2.6 ¿Qué es el SDK?

El SDK [45] o *Software Development Kit* , es un conjunto de herramientas de desarrollo que permiten a los usuarios elaborar aplicaciones. Estos kits de desarrollo contienen bibliotecas que ayudan a unificar los programas (multiplataforma) , además de ser muy útil para el desarrollador puesto que permite al programa establecer procedimientos con el sistema operativo para realizar ciertas funciones, sin tener que programarlas manualmente.

El SDK de *Android* incluye proyectos a modo de ejemplo con código fuente incluido, herramientas de desarrollo, emulador, depurador y bibliotecas necesarias para crear aplicaciones, las cuales se programan como ya se ha visto, utilizando el lenguaje de programación *Java*.

A la hora de compilar nuestra aplicación en su formato final estándar, de *Android .apk* (*Android Package*, paquete de *Android*), estas herramientas se compilan junto a la misma.

Por otra parte, puesto que estamos en un apartado en el cual se habla de herramientas, se cree de interés comentar que el sistema *Android* implementa el principio del mínimo privilegio [44], es decir, cada aplicación por defecto, tiene acceso sólo a los componentes que requiere para hacer su trabajo y no más, con lo que se crea un entorno seguro en el que una aplicación no puede acceder a partes del sistema para las que no se le habilite el permiso.

En consecuencia, la aplicación debe solicitar permiso para acceder a los datos del dispositivo, como los contactos del usuario, los mensajes SMS, el almacenamiento que contenga (tarjeta SD), la cámara, el *Bluetooth*, etc.

Por último, en nuestro caso particular, se ha utilizado el SDK versión 23 con versión de herramientas 23.0.2, es decir, se puede ejecutar como se ha comentado anteriormente con un sistema KitKat 4.4 de API 19, pero está orientado a un sistema Marshmallow 6.0, de API 23.

### 1.2.7 Ciclo de las actividades

Para poder comprender qué es un ciclo de actividad, es evidente que se ha de definir primero el concepto de Actividad.
Una actividad simboliza el código correspondiente a una pantalla, por ejemplo, para nuestra aplicación, una actividad será la de conexión, otra el tutorial, y otra la gestión de mapa y cámara.
Estas son el contacto directo entre aplicación y desarrollador a partir de los archivos *class*, archivos *.java*.

Normalmente, para aplicaciones que proporcionen alguna función útil, suele haber varias actividades que interaccionan entre sí a través de *Intents* , y todas ellas están declaradas en el *Manifest* (Conceptos que se verán, de nuevo, en el apartado de "Conocimientos Previos"). No obstante, son independientes las unas de las otras.

El concepto de actividades, diferencia a *Android* del resto de sistemas en que no se basa en un archivo Main, sino que empieza por una actividad (que no necesariamente tiene que ser la más importante, puede ser un tutorial como será en nuestro caso) y va interaccionando y cambiando entre las diferentes que conforman la aplicación, según el ciclo de vida de las actividades.

El ciclo de vida de una actividad define el proceso que esta sigue desde que se crea hasta que se destruye, según el siguiente esquema:

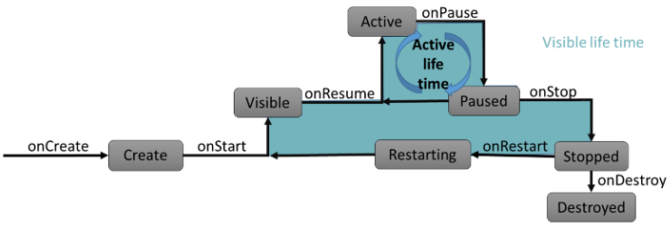



Figura 12: Ciclo de las actividades

El ciclo de las actividades [46], por tanto, tiene las siguientes etapas:

- Creación:
  Comienza con la llamada del onCreate(), donde se construye y conecta con la interfaz de usuario (UI). Una vez hecho esto, la actividad se ha creado, pero aún no es visible.
  Cuando el usuario pulsa el icono de la aplicación en su dispositivo, el método onCreate() se ejecuta inmediatamente, para cargar el diseño de la actividad principal en la memoria.

- Parte de actividad activa:
  Después de cargarse, la actividad se ejecuta en una secuencia de onStart() y onResume(). Aunque onStart() determina el momento en que la actividad está a punto de ser visible para el usuario, previamente cargada en memoria por onCreate(), es onResume () la que confiere la posibilidad de interactuar con el usuario.

- Pausa:
  La actividad está en pausa cuando está parcialmente visible en la pantalla, lo que significa que ha perdido la atención de primer plano. Se realiza mediante el método onPause().

- Parar (*Stop*):
  La actividad se detiene cuando ya no es visible, pero todavía se mantiene en la memoria.
  Cuando una aplicación es enviada al fondo, se ejecuta el método onStop(), que indica que la aplicación ya no es visible, pero sí recuperable con onRestart(), la cual devuelve la actividad al apartado onStart().

- Destrucción:
  El método onDestroy() indica el final del ciclo de vida de la aplicación (Y de todas las activadades en consecuencia), y es el método que se ejecuta antes de finalizar todo el proceso.

# 2 Estado del arte

El estado del arte es el apartado donde se realiza como concepto generalmente aceptado, la recopilación de todos los acontecimientos pertinentes de importancia sobre el tema tratado, es decir, es donde se debe hacer trascender todo el conocimiento existente anterior al tema.

En el estado del arte, en cambio, no se trata de exponer una historia hasta el día actual en el que nos encontramos, sino que se trata de un compromiso entre la exposición de datos anteriores, pero a su vez, tratar qué problemas surgieron, quiénes los investigaron, y cómo se resolvieron, o si en cambio todavía son un aspecto a resolver hoy en día.

Puesto que lo que despierta nuestro interés ahora mismo es abordar los temas de aerogeneradores y drones, así como de aplicaciones, si se desea información más concreta de cómo realizar el Estado del Arte, podemos recurrir al artículo "Guía para construir Estados del Arte" [47].

## 2.1 Evolución y estado de mercado de los drones (Quadcopters)

En este apartado se va a tratar la evolución a lo largo de la historia de los drones, no obstante, esta va a ser de gran extensión y muy centrada en drones bélicos de ala fija, mientras que en este trabajo se focaliza más el interés en drones de ala giratoria.

Este motivo es el que propicia que se pudiese llamar al apartado, "Evolución y estado de mercado de los cuadrirrotores", puesto que sobre los drones en general solo se proporcionará la información necesaria.

El concepto de dron o aeronave no tripulada surge en el 1849, cuando los austriacos atacaron Venecia con Globos aerostáticos cargados de bombas.
Aunque este concepto cumple con las siglas UAV, Unmanned Aircraft System (Aeronave no tripulada), no cumple con lo que se espera de un RPAS, Remotely Piloted Aircraft System (Aeronave tripulada remotamente) , de la cual sí se puede tener un control después del despegue.

Por otra parte, los hermanos Breguet desarrollaron el primer cuadrirrotor en el año 1908, no obstante esta aeronave era tripulada y no voló a más de unos pies de altura.

La evolución de los drones avanzaría, evidentemente, durante la I y II Guerra Mundial.
Durante la Guerra Mundial I, surgió la aeronave no tripulada "Hewitt-Sperry Automatic Airplane", capaz de cargar bombas hasta su objetivo [48], pero continuaba siendo de ala fija.

Sobre el año 1925, se crearon más aeronaves que seguían respondiendo al concepto UAV, como el "Royal Aircraft Establishment Larynx", la cual era una aeronave antibarco que se orientaba a otros barcos, y volaba hacia ellos, teniendo mucho mayor alcance que un proyectil común.

En la Guerra Mundial II, los nazis pusieron en marcha pequeños drones de ala fija remotamente pilotados, pero es en 1957 , cuando EEUU desarrolla drones conocidos como el "Lockheed U-2" [49], drones robustos de ala fija y de gran altitud de vuelo, remotamente pilotados.

Figura 13: Lockheed U-2

A partir de este momento, ya estamos en el concepto de RPAS de drones de ala fija, pero por esta misma época , los cuadrirrotores, aunque ya volaban efectivamente y siendo controlados, no lo conseguían remotamente, como el "Convertawings Model A quadcopter." o el "Curtiss-Wright VZ-7" [50](1958).

Figura 14: Curtiss-Wright VZ-7

La dificultad adicional que conlleva controlar el campo de velocidades de una ala giratoria respecto de una ala fija, en la cual es mucho más sencillo, y el poco avance de la electrónica y microelectrónica en la época, son los causantes de dicha ralentización en el proceso.

Aún así, aunque no de tamaño tan pequeño ni con cuadrirrotores, ya había empresas como WestLand que en 1976 desarrollaron helicópteros remotamente pilotados:

Figura 15: WestLand Wisp

En la década de los 80, se movían en la misma línea, solo que drones de ala giratoria, como el creado por Canadair (CL-227), ya constaban de un *feedback* por parte de un sensor acelerómetro y un barómetro para establecer el control de la altitud.

Además, cada vez se está más cerca de conseguir como en la actualidad (en el ámbito civil), un dron con suficientes sistemas como para proporcionar un vuelo de pilotaje sencillo, y un sistema de control preciso. En este caso, contaba con los sistemas:

Inertial Measurement Unit (Unidad de medida inercial, IMU), the Airborne Computer (Ordenador de abordo , ABC), the Fuel Controller Unit (Unidad de control de combustible, FCU), the Servo Controller Unit (Unidad de control de servos, SCU) y Payload Interface Unit (Interfaz de control de carga de pago, PIU).

Figura 16: Canadair CL-227 Sentinel

En la década de los 90, se tienen avances muy significativos en la evolución de los drones por la liberación o disposición de uso no militar del GPS,

sistema de posicionamiento global, del cual se explicará su funcionamiento en el apartado "Geolocalización".

A partir del conflicto de la Guerra Fría (Finales del siglo XX), se registra un crecimiento exponencial en este sector, y una vez en el siglo XXI, en el año 2006, aparece la empresa china DJI [51], de la cual vamos a utilizar sus drones en este proyecto.

Figura 17: Icono de la empresa *Dà-Jiāng Innovations*

Por esta época, DJI ya fabricaba vehículos aéreos no tripulados, plataformas de vuelo, controladores de vuelo para multi-rotores, accesorios para helicópteros, gimbals portátiles y estaciones terrestres.

Con el desarrollo constante y creciente de la industria de la electrónica, y de los lenguajes de programación, el desarrollo de los RPAS civiles se ha centrado en la obtención de un control más robusto , pudiendo obtener a la vez una optimización del tamaño.

Ahora además, también se busca estar en contacto con el ciudadano y programador, proporcionando herramientas de software desarrolladas por la propia empresa, para que se pueda realizar a nivel usuario las aplicaciones que se requieran para nuestro dron.

Gran parte de la información de todo el periodo histórico de los drones se ha obtenido del artículo de Cristina Cuerno Rejado, de la E.T.S.I. Aeronáuticos de la Universidad Politécnica de Madrid [52].

Por último, como bien es sabido hoy en día, los drones abarcan una inmensidad de sectores como el entretenimiento (Simplemente para jugar, reportajes fotográficos...), competición, militar, reconocimiento en el sector agrónomo, servicios de mensajería , servicios sanitarios y muchos otros más, entre el que está el sector emergente que estamos buscando , análisis de ae-

rogeneradores.

La gran diferencia es que , con todas las herramientas que tenemos al alcance hoy en día como hemos visto, se buscará que dicho análisis, en un futuro, sea totalmente autónomo.

Como comentario adicional, se puede comprobar la importancia del sector, simplemente atendiendo a datos económicos de la empresa DJI, la cual facturó 1500 millones de dólares en 2016, y sobretodo el crecimiento exponencial del mismo, como podemos comprender a través de la evolución de ingresos de solamente esta empresa desde 2009 [53]:

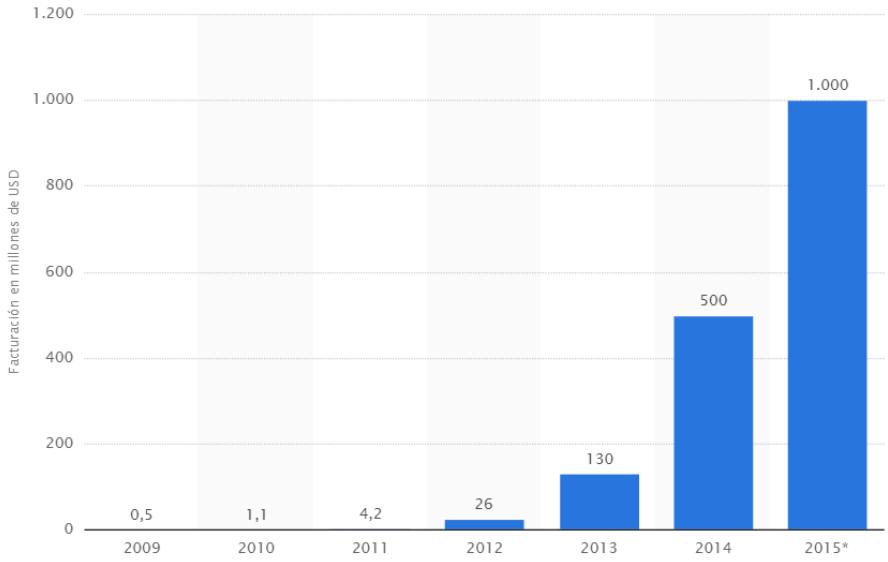



Figura 18: Evolución de ingresos *DJI*, portal de estadísticas *statista*

## 2.2 Evolución y estado de mercado de los aerogeneradores

La energía eólica, a diferencia del desarrollo de drones, se remonta milenios atrás, puesto que el hecho de utilizar la energía del viento para propósitos del ser humano, no es tan complejo como crear un aparato de proporciones reducidas, autocontrolado y pilotado remotamente.

Los inicios de la energía eólica, aunque no se refieran a la conversión en energía eléctrica, se remontan al año 3000 a.C., donde los primeros veleros egipcios utilizaban la fuerza del viento para convertirla en movimiento.

Cuatro milenios después (siglo VII en Afganistán, siglo XII en Europa, Francia), se encuentran los primeros molinos de viento, los cuales se asemejan relativamente a los aerogeneradores, por intercambiar la energía del viento en movimiento, a través de un eje, ya sea para moler grano o bombear agua.

Pero no alcanzaremos un concepto más cercano al aerogenerador moderno hasta el siglo XIX, donde Lord Kelvin, en 1802, tuvo la idea de acoplar un generador eléctrico a una máquina movida por el viento.

Finalmente, en 1850 y con la invención de la dinamo, se abre el camino para alcanzar lo que es un aerogenerador moderno, que primeramente pasó por manos de Charles F. Brush, primer inventor en crear una turbina eólica que produjese electricidad en 1888, y tan solo dos años después, Pour le Cour pondría en marcha el concepto de aerogenerador moderno, una máquina diseñada específicamente para intercambiar energía eólica en eléctrica.

Figura 19: Aerogenerador de *Pour le Cour*

Durante el siglo XX, surgirán muchos inventores e ingenieros que avanzarán en el sector con los siguientes contenidos:

- La instalación de los primeros aerogeneradores de Pour le Cour después de la Guerra Mundial I.

- La constatación de la ley Betz [54] (que se basa en el cálculo del máximo cociente de potencias entre la del flujo perturbado y el flujo sin perturbar) , la cual regula que el límite teórico máximo de energía que se puede intercambiar de eólica a eléctrica es del 59.26 % (16/27) , máximo que se da cuando la velocidad de salida del flujo a través del campo del rotor es un tercio de la velocidad de entrada.

  No obstante este máximo es teórico, y el aprovechamiento real está entorno al 40 % o un poco superior, por efectos de compresibilidad en punta de pala, estela...

- El aerogenerador SmithPutman de 1.25MW, desarrollado en 1941 por Palmer Cosslett Putnam, funcionó sin descanso hasta el 1945 por un problema de materiales, puesto que no existían materiales adecuados para el sector.

Figura 20: Aerogenerador *SmithPutman*

- Johannes Juul desarrolla el primer aerogenerador de corriente alterna de 200kW en Dinamarca en 1957, y este es el predecesor de los aerogeneradores actuales.

Una vez en los 70/80, la crisis del petróleo fomentará más la apuesta por las energías renovables en concreto la eólica (Podemos encontrar toda esta información siguiendo la documentación [55]).

Como se ha podido ver, por esta época, todavía no se tiene indicios claros de los materiales ni como va a afectar el entorno a los aerogeneradores, y su alcance, principal problema y motivación por la cual necesitamos análisis constantes para todos los ejemplares.

No obstante y a pesar de estos problemas, el hecho de ser una energía renovable, limpia, que alcanza la paridad de red en la década de los 2000 (Su desarrollo cuesta lo mismo que cualquier método tradicional de obtención de energía, ya no es más caro), que reduce su coste de inversión en un 80 % en tan solo dos décadas, y que después de instalada, alberga un coste marginal de a penas un céntimo de dolar por kWh en sus mínimos históricos, la hacen una de las más aceptadas hoy en día [56].

De estos costes , se estima que seguirán bajando (además de tener todo el océano como extensión para nuevos parques eólicos, y no estar restringidos a tierra) , por lo que el sector eólico no cesa en su crecimiento, con cifras actuales de producción mundial de 587.7 GW, y en España (31 de diciembre de 2016) de 23026 MW , con una producción energética del 19.3 % del total, siendo la segunda tecnología en producción [57].

Por su gran valor de mercado actual, y por el hecho de respetar nuestro planeta, esta energía merece un interés especial, y en este trabajo se va a intentar aportar mejoras sobre la misma, para contribuir a que continue con su desarrollo.

## 2.3 Importancia del hilo conductor de las aplicaciones

En la última década, se ha experimentado un crecimiento asombroso en el sector móvil y microelectrónica, pasando de grandes dispositivos con poca movilidad que servían para su propósito inicial, llamar, a dispositivos de unos pocos cientos de gramos con funcionalidades tan dispares, como desde una calculadora simple a una aplicación para controlar un dron.

Además de esto, no nos interesa solo este gran desarrollo, sino la gran aceptación mundial, para la que según datos del 2016 [58], se alcanzó la cifra de 7900 millones de dispositivos en La Tierra, más que los habitantes de la misma.

De este modo, y en nuestro territorio, Europa, 78 de cada 100 habitantes tiene un smartphone (80 % en España).

La motivación del desarrollo de este trabajo, en consecuencia, será de nuevo hacer la aplicación lo más industrial posible y que abarque todos los sectores, y lo cierto es que el sector de las aplicaciones nos lo pone muy fácil con estas cifras.

Por otra parte , es cierto que esta no es una aplicación destinada a todos lo públicos y edades, pero los smartphones realizarán fácilmente el trabajo de un portátil en este ámbito de control del aparato, serán fácilmente portables por un operario ,**no teniendo ni que proporcionárselo la empresa incluso**, por su extendido uso y posesión generalizada, y todas estas prestaciones orientadas a dispositivos de base entorno a los 150€ e incluso inferiores.
De este modo, se estará en el mercado sin ninguna restricción por software costoso, de dificultad elevada o de dificultades en el trámite de conseguir un dispositivo.

# 3 Mantenimiento y daños en Aerogeneradores

En apartados anteriores hemos visto el gran tamaño que pueden abarcar los aerogeneradores, y métodos concretos de mantenimiento.

Por otra parte, este apartado proporcionará datos más concretos de cómo se producen dichos defectos y cuáles son los procesos a seguir para repararlos.

## 3.1 Estructura de pala, ensayos y problemas de deterioro por erosión

Las palas de un aerogenerador son probablemente la parte más importante, puesto que recogen la energía del viento y la transforman en eléctrica.

Dichas palas, están construidas de forma muy similar a una ala de avión, es decir, son perfiles aerodinámicos, pero en este caso en vez de intentar maximizar el *Lift* y reducir la resistencia, maximizarán la velocidad de giro con su orientación respecto al viento (Hasta cierto límite de rpm, considerando el no alcanzar velocidades excesivamente cercanas a la velocidad del sonido en punta de pala, para aerogeneradores grandes un máximo de unas 20 rpm o inferior).

Las palas, están construidas con el cajón de torsión habitual y refuerzos internos (costillas , largueros y larguerillos), y con una piel la cual para aerogeneradores grandes, por el peso, suelen ser de fibra de vidrio (Con matriz de poliéster o epoxi):

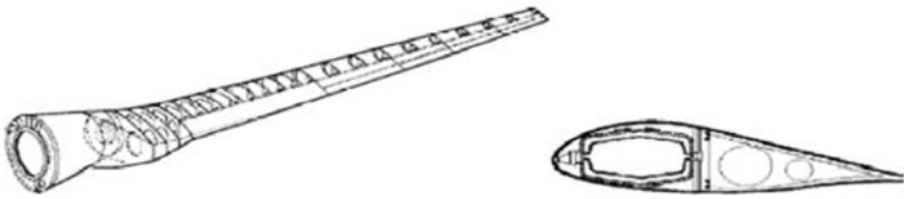


Figura 21: Estructura de una pala y su sección

Como cabe esperar, estas palas son sometidas a:

- Tests dinámicos:

  Se somete a la pala a oscilaciones con su frecuencia natural: cinco millones de ciclos respecto de los dos ejes principales.
  Durante las pruebas, una cámara de infrarrojos de alta resolución se usa para comprobar si hay pequeñas roturas en el laminado del material compuesto de la pala.

- Tests estáticos:

  Las palas son sometidas a cargas extremas durante un tiempo predeterminado (10-15s), para probar su resistencia a la rotura, y son flexionadas en dos direcciones.

- Test a rotura:

  Sigue la misma operativa de un test con carga estática, pero se incrementa la misma hasta la rotura de la pala.
  Esto solo se realiza cuando la composición o forma de la pala se ha cambiado de forma considerable, porque como cabe esperar, romper una pala supondrá un ensayo destructivo, y por tanto, dinero perdido.

Con todos estos análisis que se realizan, además de los de vibraciones y termográficos, tenemos cubiertos todos los problemas estructurales que pueda sufrir el aerogenerador a lo largo de su vida con un mantenimiento moderado (Los impactos de aves o rayos son inevitables a lo largo del tiempo).

Pero la realidad, es que cuando se fabricaron estos aparatos, nadie consideró que el tiempo de operación (a veces ininterrumpido) bajo el ambiente corrosivo de la atmósfera no solo afectaba a la fatiga de la pala , tema para el que están bien preparados, sino que aparecería otro agente mucho más imperceptible e inesperado : el agua.

El impacto directo y constante de las partículas de agua, bajo este ambiente y tiempo de operación, es el mayor problema de desgaste de pala.

Es un problema de tal gravedad, que aunque la vida útil de un aerogenerador puede oscilar entre los 15 años bajo este ambiente, puede aparecer desgaste severo en las palas tan solo en dos años si se instala en el mar, y a

lo largo de cinco años si se instala en tierra.

Resulta tan importante, que la información alcanza incluso periódicos locales de la Comunidad Valenciana, donde se comenta que expertos de la Universidad Cardenal Herrera y otros países, están investigando una solución respecto a los materiales para solventarlo [32].

## 3.2 Alcance del deterioro por erosión

Como hemos visto, los aerogeneradores están en un entorno hostil, en el cual están sometidos a polvo, lluvia, aguanieve y particulas abrasivas en general.

En un primer momento, en el diseño de todos los aerogeneradores diseñados y puestos en funcionamiento, no se tuvo en cuenta este aspecto.

Como se puede comprender, no es viable el hecho de desmantelar todos los campos eólicos del mundo y rehacer los aparatos, por lo que actualmente la solución más aceptada son recubrimientos con pinturas antiabrasión.

Este recubrimiento sobre el borde de ataque de las palas será una opción (y una obligación si pretendemos alargar la vida del aerogenerador al máximo) de bajo coste, en comparación con la sustitución de palas antiguas por otras nuevas, y en comparación con un mantenimiento correctivo continuo.

No obstante, debe realizarse de un modo riguroso, porque cualquier arañazo o muesca sobre esta pala, puede servir de incipiente a una abrasión continuada, que como podemos imaginar por todo lo comentado, va a disminuir la vida útil del aerogenerador así como su eficiencia.

Esta eficiencia se ve menguada no tanto por la eficiencia aerodinámica del perfil, aunque si afectará notablemente por el gran tiempo de uso, sino más bien por el coste de oportunidad de parar uno de estos gigantes constantemente.

Además, como podemos ver en el artículo que presenta *Lisa Rempel* [33], debemos ser capaces de escoger cuál es la protección necesaria para nuestro entorno, puesto que no todas son iguales, y aún así, la protección no es infalible , por lo que tenemos que realizar revisiones periódicas sobre las palas.

Nos cercioramos, una vez más, de lo importante que va a ser el aspecto económico si tenemos que realizar muchos análisis a lo largo del tiempo.

Por otra parte, también tenemos que considerar problemas insospechados localizados en el montaje:
Si no se realiza un recubrimiento riguroso, como se apuntó anteriormente, pueden quedar burbujas de aire atrapadas en el recubrimiento, las cuales, cuando la pala flecte continuadamente o incluso aumente de temperatura, se expandirán, y destruirán parcialmente la pala desde dentro hacia fuera, independientemente del recubrimiento protector.

Figura 22: Destrucción desde el interior de una pala por recubrimiento mal procesado

Por último , y sin cambiar de temática, sino focalizándonos más en el rigor necesario, si no se utilizan los materiales adecuados en el revestimiento, puede que se confíe en que durará un cierto tiempo establecido hasta la siguiente revisión, pero lo cierto es que el revestimiento de la pala se puede quebrar, por lo que el agua del ambiente llegaría hasta lo más interno de la estructura de laminado tipo "sandwich".
Esto provocaría **problemas de tal gravedad**, como que se separase el laminado del núcleo de la estructura de la pala, e incluso desequilibrios en el rotor, que produjesen vibraciones adicionales no deseadas.

En resumen, las inspecciones con drones y recubrimientos correctos de pala , serán notablemente preferibles a desmontar la pala cuando esté gravemente dañada para repararla, acción que parte de un gasto mínimo de unos 75000$ (Precio orientativo de la región de norteamérica) sin contar el gasto de la grúa hidráulica vista en el apartado de introducción.

## 3.3 Tipos de matenimiento

Aunque nuestra principal motivación será la detección de errores en palas por revisiones con drones, y no el trato y reparación de estos defectos, comentaremos los tipos de mantenimientos y alcance de costes, para realzar la importancia de dichas revisiones antes de que ocurra un problema de gravedad como hemos comentado en el subapartado anterior.

- Mantenimiento Preventivo:

  Es el mantenimiento en el cual nos estamos centrando durante todo el proyecto, puesto que, utilizando el método de los drones (posiblemente no en otros casos), va a ser rentable, ya que además de barato, vamos a conseguir identificar pequeños errores si los hay e impedir que prosigan con su crecimiento, y en consecuencia repararlos.

  Además, cuando consigamos crear una base de datos completa, sobre el análisis de un mismo tipo de pala en unas mismas condiciones, podremos crear un mantenimiento predictivo de gran fiabilidad.

- Mantenimiento Predictivo:

  Como se puede prever del punto de mantenimiento preventivo, el mantenimiento predictivo es aquel que se realiza en periodos cercanos a los que se cree que va a fallar o estar deteriorado el producto, optimizando así los recursos disponibles y reduciendo el coste de oportunidad (costes por tiempo de parada de la máquina) al mínimo.

  Es el más óptimo de todos, puesto que la acción correctiva se va a realizar en el momento adecuado, pero el inconveniente es tener que crear la base de datos a base de mantenimientos preventivos.

  Esto propiciará que una vez la tengamos, se obtenga el mejor sistema, pero mientras tanto quizás realicemos los mantenimientos antes (movilizar recursos para nada, todavía no hay daños) o después (se debería haber actuado antes, ahora el daño es ligeramente superior al que debería) de tiempo.

- Mantenimiento Correctivo:

  El mantenimiento correctivo se refiere al mantenimiento no programado que requiere la sustitución o reparación del objeto.

  Esta reparación puede darse si nos demoramos demasiado en las revisiones preventivas, con lo que el daño crece más de la cuenta, y puede incrementarse hasta el punto de tener que bajar la pala y gastar varias decenas de miles de dólares en el transporte y reparación.

# 4 Normativa sobre el pilotaje de drones

En el sector tecnológico de los drones, un sector puntero y emergente tal como se apunta en la página de AESA [18] (Agencia Estatal de Seguridad Aérea), Ministerio de Fomento, es necesario establecer un marco regulatorio que deje claros todos los aspectos legales, y evite al consumidor incurrir en actividades ilegales o en situaciones de riesgo no definidas, vacíos legales.

En el apartado de marco regulatorio para drones [19], se expone que el marco legal de los drones se aprobó y constató según el *Real Decreto-ley 8/2014* el 4 de Julio, para pasar a ser tramitado como ley a partir del 17 de octubre de 2014 con la publicación en el BOE (Boletín Oficial del Estado) de la Ley 18/2014.

Este reglamento temporal (que será el que está en vigencia hasta que entre en vigor la ley final) contempla los distintos escenarios en los que se podrán realizar los trabajos aéreos y en función del peso de la aeronave, además de complementar la anterior Ley 48/1960 de Navegación Aérea.

En el BOE anteriormente mencionado, se incluyen los artículos 50 y 51 [20], de los cuales se van a resumir los puntos más importantes a continuación.

- *Artículo 50*: Operación de aeronaves civiles pilotadas por control remoto.

    - 1: El piloto es el responsable tanto de la aeronave como de la operación que se realice, incluyendo los perjuicios que pueda causar con la misma y las imágenes que tome.

    - 2: Todas las aeronaves pilotadas por control remoto tienen que tener a la vista el equivalente a una matricula que permita identificarlas, y sobre las de peso mayor de 25kg, no podrá aplicarse la normativa de aeronaves menores, además de tener que estar inscritas en el Registro de matrícula de aeronaves y disponer de certificado de aeronavegabilidad

    - Podrán realizarse actividades técnicas o científicas, de día y con meteorología favorable, sujeto a las siguientes condiciones:

Volar la aeronave en zonas donde no habiten personas al aire libre (pueblos, ciudades...) , a menos de 500m y con alcance visual del piloto, y a menos de 120m de altura (aeronaves de menos de 25kg).

Para aeronaves entre 25 y 150kg, y de más de 150kg pero destinadas a servicios de extinción, se requerirá una documentación mucho más estricta y certificado de aeronavegabilidad, además de estudios aeronáuticos favorables de seguridad en la operativa, y tener un seguro o póliza que cubra los daños civiles en caso de haberlos.
También se debe tener un sistema ante interferencias ilícitas.

Para realizar las operaciones de vuelo, el piloto debe acreditar la licencia de piloto, y para hacerlo fuera del alcance visual la licencia de piloto avanzado. Además, las operaciones que se vayan a realizar, deben comunicarse a AESA con una antelación mínima de 5 días (siempre que sean actividades que puedan suponer algún riesgo, como se verá en el artículo 151 que se comenta más adelante), con información que concierne al piloto y la aeronave.

- *Artículo 51*: Modificación de la Ley 48/1960, de 21 de julio, sobre Navegación Aérea

  - 1 : Artículo 11 :
    Se entiende por aeronave (una de las dos definiciones disponibles en este artículo, la que define los drones):
    Cualquier máquina pilotada por control remoto que pueda sustentarse en la atmósfera por reacciones del aire que no sean las reacciones del mismo contra la superficie de la tierra.
  - 2 : Artículo 150:
    Las aeronaves de control remoto no podrán transportar carga, con o sin remuneración. Por otra parte, no necesitarán operar en infraestructuras aeroportuarias autorizadas a menos que se requiera expresamente en su normativa específica.

  - 3: Artículo 151 (párrafo primero):
    A excepción de las actividades turísticas o deportivas , se deberá comunicar a la Agencia Estatal de Seguridad Aérea y esperar su aprobación, cualquier actividad que pueda suponer un riesgo.

# 5 Dispositivos utilizados

En relación al análisis de grandes aerogeneradores mediante drones, han sido utilizadas una gran cantidad de herramientas tanto físicas como de software.
Las más destacables y genéricas son, el dispositivo móvil , y el dron.

En los sucesivos subapartados, se mostrarán los requisitos o versiones mínimas que el dispositivo ha de tener para el correcto funcionamiento, así como la explicación de partes esenciales del dron.

## 5.1 Requisitos para el desarrollo

Para la correcta realización de la aplicación, debemos utilizar productos soportados [23], no tan solo por el ordenador o aplicación, sino incluso por el propio simulador DJI, ya que sino no seremos capaces de poder probarlo con seguridad.
A continuación, se explican las versiones de todos los dispositivos utilizados.

### 5.1.1 Versiones utilizadas y versiones mínimas

Se van a repartir las versiones a utilizar en ordenador, dron y dispositivo móvil de modo que se pueda entender todo con la máxima claridad posible:

- Ordenador:
  Aunque no está oficialmente tratado, se recomienda un ordenador con un nivel de procesadores y memoria RAM media o elevada, puesto que durante el proceso de desarrollo se ha observado la gran cantidad de recursos que consume *Android Studio*.
  Además, se ha de tener un sistema operativo *Windows* (Puesto que desarrollamos en *Android, Mac* tiene sus respectivos soportados) 7, 8 , 8.1 o incluso 10, el cual generalmente funciona (como es nuestro caso) aunque no está admitido como soportado oficialmente.

- Dron:
  En lo referido a los drones, es cierto que podremos ejecutar la aplicación con un *Phantom 3 professional* o un *Phantom 3 advanced*, no obstante según la web de *DJI* de productos soportados, [23] no están soportados por el simulador.
  Por tanto, para tener una experiencia completa, se deberá usar un *Phantom Series 4* (Como ha sido nuestro caso), *Mavic Pro* (Al cual se refiere la web en la mayoría de sus tutoriales) o la serie *Inspire*.

(a) *Phantom 4*

(b) *Mavic Pro*

Figura 23: Drones hacia los que se orienta el desarrollo

- Móvil:
  Para el correcto desarrollo de aplicaciones en móvil, necesitamos como prerequisitos [26] tener una versión de *Android Studio* superior a 1.5, y una API superior a 22.

  Esta aplicación se ha desarrollado en un entorno de *Android Studio* 2.2 y versión de desarrollo 23 (Aunque la mínima API está fijada en 19, para una experiencia completa interesará tener una superior a la 22, bastante usual en nuevos dispositivos), por lo que cumple las expectativas para no propiciar ningún error.

En resumen, es posible que se tuviese algún pequeño problema como desarrollador particular no profesional, puesto que se requiere dispositivos relativamente potentes , y sobretodo en los drones , los cuales pueden suponer una carga económica.

Sin embargo, y tal como se apunta en el trabajo, esta aplicación está destinada y enfocada a la industria, por lo que unos pocos terminales de media potencia y los drones necesarios para el desarrollo (que evidentemente servirán también para la inspección) no supondrán un gran problema.

## 5.2 Datos técnicos del dron

Como se ha comentado anteriormente, hemos trabajado para el desarrollo de la aplicación con el dron *Phantom 4* [24], y se van a mencionar sus datos técnicos.

Algunos de los posteriores apartados, en cambio, serán simplemente explicativos puesto que , por ejemplo, no será posible obtener el código ni información de cómo establece la geolocalización o estabilidad exactamente el dron, pero sí se conocen métodos genéricos en los cuales pueden estar basados.

Antes de empezar con los apartados siguientes, las especificaciones [25] más notables son: Peso de 1380 g, máxima velocidad de 20 m/s , resistencia máxima al viento de 10 m/s, techo de vuelo de 6000 m, tiempo de vuelo máximo de 28 minutos y grabación de vídeos en formato 4K.

### 5.2.1 Geolocalización

En esta aplicación, se dan por sabidos conceptos sobre lo que es una geolocalización (en este caso GPS) puesto que, al igual que con el posterior apartado de estabilidad, se realizan automáticamente con métodos internos.

No obstante y aunque puede ser sabido, se puede recordar de nuevo su operativa para comprender su funcionamiento.

El sistema GPS, Global Positioning System, es un sistema de localización de objetos sobre la tierra mediante la triangulación de señales, de mínimo 4 satélites GPS.

El sistema consiste en la intersección de 3 esferoides , los cuales darán la posición del objeto. Debido al retardo en la señal, trabajando a través de la velocidad de la luz, se calculan pseudodistancias que nos permiten saber la distancia emisor-receptor. No obstante, para cumplir con las distancias exactas, necesitaríamos hacer uso de relojes de precisión perfecta, con coordinación entre los relojes internos.

Este hecho es el que propicia que, puesto que aunque precisos, no van a tener una coordinación 100 % perfecta, necesitemos un cuarto satélite de apoyo que coordine todas las medidas y sea capaz de resolver las coordenadas X, Y, Z a resolver para el objeto a localizar, a través de una cuarta variable

a resolver, la demora entre relojes.

El ajuste de las ecuaciones por mínimos cuadrados acaba presentando un aspecto como:

$$(x_i - U_x)^2 + (y_i - U_y)^2 + (z_i - U_z)^2 = (R_i^{'} - ct)^2 \quad (1)$$

Una vez explicado, el siguiente paso sería, sobre la misma ecuación y para un contador $i = 1..,4$ , resolver a partir de los datos de posición de los satélites $(x_i)$ y las pseudodistancias $(R_i^{'})$ medidas por los mismos (La pseudodistancia es la distancia real que sería medida más el desfase temporal por la velocidad de la luz, distancia real $R_i^{'} = R_i + ct$ ).

Las ecuaciones las podríamos resolver por el método adecuado que se nos ocurra , como por ejemplo desarrollo en series de Taylor para su posterior resolución más sencilla, pero esto ya no es representativo en cuanto a lo que se quería explicar, es decir, entender el funcionamiento del GPS.

Se puede encontrar información útil y bien sintetizada en el artículo [27]: *GPS: EL SISTEMA DE POSICIONAMIENTO GLOBAL*, editado desde la *Escuela de Topografía, Catastro y Geodesia* , Universidad Nacional de Heredia, Costa Rica.

### 5.2.2 Sensores

Se van a comentar los sensores de los cuales dispone el *Phantom 4*, los cuales influyen de forma muy directa en su gran estabilidad.

Este dron contiene 2 sensores en su parte frontal, los cuales son pequeñas cámaras que captan cualquier cosa delante del dron en un angular de 60 grados, e impide que pueda colisionar contra las mismas con su sistema para evitarlo, *Collision Avoidance*.

Figura 24: Sensores frontales del *Phantom 4*

A su vez, este está complementado también con sensores en la parte inferior del dron, en la cual se ubican otras dos cámaras y dos sensores acústicos:

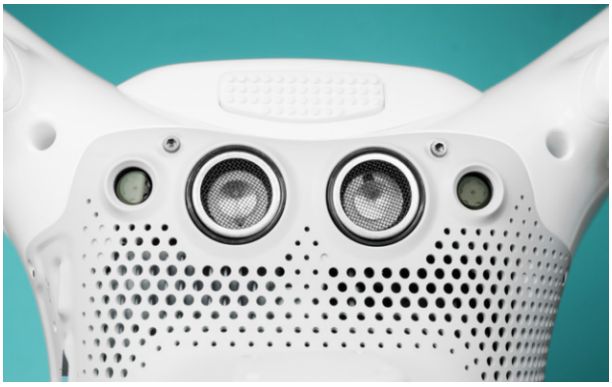


Figura 25: Sensores de la parte inferior del *Phantom 4*

El sistema de evitar colisiones y posicionamiento vertical es bastante robusto, debido a que se complementa la ubicación GPS comentada anteriormente, y su doble IMU interna (Inertial Measurement Unit, calculador interno de la actitud del dron).

Sumado a todo esto, tenemos el VPS (Vision Positioning System), el cual ubica el dron en el espacio mediante sus sensores acústicos a modo de sonar , y las cámaras ya comentadas.

El único punto débil que se ha de tener en cuenta a la hora de volarlo es el vuelo lateral y hacia atrás, puesto que en esta posición, no incluirá sensores que puedan predecir y detener la trayectoria en caso de peligro, por lo que se estará volando totalmente bajo la habilidad del piloto.

### 5.2.3 Motores

En cuanto a los motores, como podíamos ver en las especificaciones, van a ser capaces de mantener este dron de 1.38 Kg de peso y llevarlo a una velocidad ascensional máxima de 6 m/s, de descenso controlado a 4 m/s , y de máxima velocidad de 20 m/s.

Figura 26: Motor del *Phantom 4*

Para ello, contamos con unos motores de unos 55g de peso que desarrollan entorno a (No se ha podido encontrar información específica del motor, pero si de otros muy similares [28] como el Motor 2312/960KV CCW) 960 vueltas por minuto por voltio de entrada en vacío, es decir, 960 rpm/V en vacío, las cuales serán inferiores evidentemente con la hélice acoplada.

Buscando un poco más allá en motores de la misma serie 2312, pero ligeramente más potentes, encontramos que los motores del DJI oscilan entorno a valores de potencia de consumo máxima de 230W, entre otras características de interés que podremos encontrar siguiendo la referencia [29].

### 5.2.4 Estabilidad

La estabilidad del dron es, otro aspecto gestionado automáticamente de forma interna sin que tengamos que preocuparnos por nada, pero la realidad es que en el futuro puede que tengamos que desarrollar nuestro propio dron, para no depender de las empresas o simplemente a modo de investigación en cualquier aspecto que nos interese.

Este motivo da validez al apartado, en el cual se va a desarrollar cómo se gestiona la estabilidad de un cuadrirrotor a partir del tratado de las ecuaciones de movimiento, y posteriormente su gestión con un sistema de control automático.

Primero se empezará definiendo el entorno dinámico en el cual se opera, es decir, en el aire , estando caracterizada su posición y actitud mediante 6 grados de libertad:

- Los 3 correspondientes a su localización del centro de masas (x, y, z) como veremos a continuación en la imagen con un vector verde.

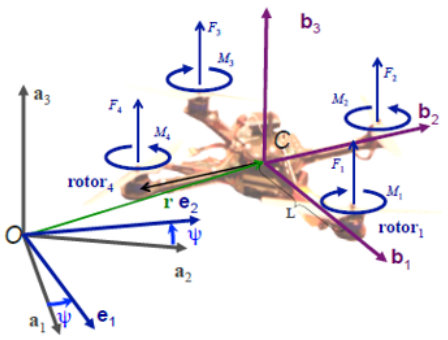



Figura 27: Entorno dinámico del dron

- Los otros 3 g.d.l. correspondientes a la actitud del dron, conocidos como alabeo (Roll, $\phi$) , cabeceo (Pitch, $\theta$) y giro (Yaw, $\varphi$).

A partir de esto, podremos establecer el vector de 6 g.d.l. que lo sitúa en el espacio del modo $q = [x\ y\ z\ \phi\ \theta\ \varphi]^T$, y definir el estado de control del dron como este vector y sus variaciones, $x = [q\ \dot{q}]^T$ .

Con esto se puede plantear el abanico de ecuaciones a resolver, es decir, las de los ángulos de Euler y relación con su ratio de giro, momentos de inercia, y las que caracterizan el empuje del cuadrirrotor.

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 + M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2}$$

*Ecuación del momento de inercia basado en los ratios de giro.*

$$m\ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ \vec{F_1} + \vec{F_2} + \vec{F_3} + \vec{F_4} \end{bmatrix} \tag{3}$$

*Ecuación de Fuerzas (caracteriza el empuje) .*

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\cos\phi\sin\theta \\ 0 & 1 & \sin\phi \\ \sin\theta & 0 & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\varphi} \end{bmatrix} \tag{4}$$

*Ecuación de relación de velocidades angulares con ángulos de Euler.*

Una vez expuestas las ecuaciones, tanto fuerzas como momentos se caracterizarán mediante una constante experimental por la velocidad angular de giro al cuadrado del modo:

$$\begin{aligned} F_i &= k_F \omega_i^2 \\ M_i &= k_M \omega_i^2 \end{aligned} \tag{5}$$

Podemos encontrar información sobre esta aproximación y otros datos de interés sobre trayectorias en el artículo [30] "Trajectory Generation and Control for Quadrotors", por la Universidad de Pensilvania en 2012.

Con todas las ecuaciones que definen el movimiento expuestas, podríamos obtener el estado del dron a través del sistema de control automático.

El método que se va a plantear es un controlador interno el cual calcula la actitud del dron, y uno exterior que calcula su posicionamiento.

Se va a explicar sobre la posición "Hover" (En estacionario), puesto que es el más rápido y sencillo de explicar , y este trabajo no requiere de momento habilidades especiales sobre el control automático.

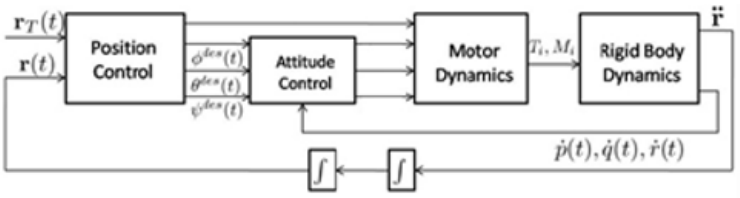



Figura 28: Bucles de control

- Control de Actitud

  Para empezar, se asume que se trabaja sobre un sólido rígido , con ángulos pequeños que nos permiten simplificar muchos senos y cosenos (Esta aproximación no será del todo incierta, puesto que si queremos tomar fotos, necesitaremos desplazamientos lentos y cortos) , y con hipótesis obvias que anulan directamente ciertos ratios de giro y sus variaciones, también en la variación de la posición, porque el dron está quieto en el aire:

  - $r = r_0$ , $\dot{r} = 0$ .
  - $\phi = \theta = 0$, $\varphi = \varphi_0$ .
  - $\dot{\phi} = \dot{\theta} = \dot{\varphi} = 0$ .

  Es sencillo deducir que el vector general de fuerzas se igualará a la masa por la gravedad, y suponiendo un control perfecto, cada motor soportará una fuerza $F_i = \frac{mg}{4}$ .

  Con esta relación y sabiendo que a su vez la fuerza se relacionaba con la velocidad angular por medio de una constante, también podemos obtener dicha velocidad angular.

  Por tanto con esto se ha conseguido tratar la actitud , en cuanto al cálculo de la fuerza neta y momentos necesarios para mantenerse.

$$\begin{bmatrix} \sum \vec{F} \\ \sum \vec{M_{\phi\theta\varphi}} \\ \sum \vec{M_{dx,dy,dz}} \\ \sum \vec{M_{d\phi d\theta d\varphi}} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{6}$$

- Control de posición

En este apartado, podremos comprobar el control de la posición usando las mismas herramientas que en movimiento, pero con la sencillez de poder simplificar con las mismas hipótesis anteriores.

Con dichas hipótesis, se define el error relacionado con la posición como:

$$e_i = (r_{i,deseada} - r_{i,medida}) \tag{7}$$

Y para tratar de minimizar dicho error a cero según un PD, tendremos la ecuación:

$$(\ddot{r}_{i,deseada} - \ddot{r}_{i,medida}) + k_{d,i}(\dot{r}_{i,deseada} - \dot{r}_{i,medida}) + k_{p,i}(r_{i,deseada} - r_{i,medida}) = 0 \tag{8}$$

Donde $\ddot{r}_{i,medida}$ y $\dot{r}_{i,medida}$ son cero.

También cabe mencionar que, aunque supuestamente solo vamos a tener componente de fuerzas y no de momentos por estar en maniobra "hover", esto en la realidad, siendo un sistema no ideal, no va a ocurrir así. La solución pasará por, también mediante constantes de proporcionalidad del PD, reducir dicha matriz a cero:

$$\begin{bmatrix} k_{p,\phi}(\phi_C - \phi) + k_{d,\phi}(p_c - p) \\ k_{p,\theta}(\theta_C - \theta) + k_{d,\theta}(q_c - q) \\ k_{p,\varphi}(\varphi_C - \varphi) + k_{d,\varphi}(r_c - r) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{9}$$

Una vez expuesta la forma de minimizar el error, lo siguiente sería linealizar las ecuaciones y separar la aceleración en ejes $x$ e $y$, que quedarán del modo:

$$\ddot{r}_1 = \ddot{x} = g(\theta \cos\varphi + \phi \sin\varphi) \tag{10}$$

$$\ddot{r}_2 = \ddot{y} = g(\theta \sin\varphi - \phi \cos\varphi) \tag{11}$$

Por último, obtenemos a partir de estas aceleraciones los ángulos comandados necesarios:

$$\phi_C = \frac{1}{g}(\ddot{r}_{1,control} \sin\varphi - \ddot{r}_{2,control} \cos\varphi) \tag{12}$$

$$\theta_C = \frac{1}{g}(\ddot{r}_{1,control} \cos\varphi + \ddot{r}_{2,control} \sin\varphi) \tag{13}$$

Como se puede intuir, se han saltado una gran cantidad de pasos intermedios y se ha calculado el caso particular del dron estacionado en el aire, por lo que es un buen indicativo para comprobar, que gracias a la funcionalidad de autoestabilización del dron, se ha ahorrado una inmensa cantidad de trabajo.

# 6 Aplicación móvil: *Drone Mission Maker*


Figura 29: Icono de la aplicación

En este apartado se va a tratar la aplicación, que como se ha podido observar previamente en el documento, tiene una importancia esencial en la unión del trabajo de los drones y aerogeneradores, además de ser un negocio emergente con grandes posibilidades industriales.

A diferencia de otros ámbitos, este parte de un bajo coste en desarrollo hasta el punto de poder desarrollarse por particulares con no más dispositivos que un teléfono móvil y un ordenador.

A continuación, se van a conocer tanto los permisos y normas que debemos cumplir para poder continuar con la visión de expandir la aplicación en el mercado , y por otra parte, los aspectos técnicos de la aplicación que se han tenido que conocer para hacer posible su desarrollo, así como las funcionalidades que nos ofrece actualmente.

Otro aspecto a destacar es lo fácilmente mejorable y versátil que puede ser en un futuro, apartado que trataremos en el capítulo de "Futuras mejoras".

## 6.1 Permisos previos al desarrollo

Para la realización de este proyecto, se han tenido que reclamar dos permisos: el correspondiente a la empresa *Google* y el de la empresa *DJI*.

El primero radica en el uso directo de su sistema de geolocalización (Para nuestro dispositivo móvil, el dron incluye el suyo propio) , y será necesario dirigirse a la página de desarrolladores de *Google* [1] y poseer una cuenta para obtenerlo.
No debemos olvidar a su vez consultar el catálogo de tarifas [2] que nos brinda *Google*, puesto que si nuestra aplicación resulta ser exitosa, y con esta finalidad se desarrolla, a partir de 25,000 usos por dispositivo y día tendremos que afrontar ciertos pagos.

El segundo ha sido el de la empresa de los drones que se están utilizando , *DJI*, para el cual tendremos que registrarnos como desarrollador, [3] y a su vez, registrar nuestra aplicación y el paquete que la contiene para obtener una clave específica para dicha aplicación.
Esta clave será totalmente gratuita, pero cabe considerar el hecho de hacerse desarrollador *premium*, puesto que tanto las herramientas como tutoriales que las definen serán mucho más amplias y clarificadoras.
A pesar de incurrir en gastos, obtendremos un avance significativo ahorrando horas de trabajo y estando al día en las últimas novedades del mundo laboral.

Otro punto destacable más allá de los permisos será el hecho de que nuestra aplicación esté en fase *beta*, por lo que a la misma se le ha dotado de todos los permisos necesarios para que funcione.
Sin embargo, esto no puede seguir esta dinámica en el momento que se pretenda lanzar esta herramienta a los usuarios a través de *Google Play Store*, tienda oficial de aplicaciones de los sistemas operativos *Android*, puesto que no podemos dar por supuesto que el cliente va a aceptar ciertos permisos en su dispositivo.

Para ello, tendremos que asegurarnos correctamente de cuales son los requisitos [4] que impone la empresa *Google*, con el motivo de que nuestra aplicación no sea declinada.

## 6.2 Conocimientos previos

Para la correcta realización de esta aplicación se requiere poseer una serie de conocimientos en el lenguaje de programación *JAVA*, del cual se ha dado unas breves pinceladas en los primeros apartados.
De este modo, necesitaremos conocer ciertas estructuras básicas y otras más complejas, de las cuales se va a tratar explicar su significado de forma sintética.
No obstante, es cierto que para un trabajo eficiente, se requiere de un personal medianamente acostumbrado a utilizar este tipo de herramientas , puesto que se parte del punto de dar por sabido una inmensa cantidad de conceptos, necesaria para entender cómo funciona la jerarquía de las funciones y métodos dentro de un mismo código, y los operadores y estructuras matriciales o de cualquier otro tipo, necesarias para poder establecer una coherencia y cohesión entre todas las subfunciones que se van incluyendo en dicho código.

Una vez aclarado este punto, los conceptos enfocados a cuando ya se ha tenido una previa iniciación en el lenguaje, destacando algunos de interés entre muchos otros, son:

- Archivos de código:
  Son los ficheros de texto que incluyen el sentido y órdenes que llevará a cabo la aplicación, y los podemos localizar en la parte izquierda de nuestro árbol de opciones, con un icono circular azul y una *C* al centro, de significado *Class*.

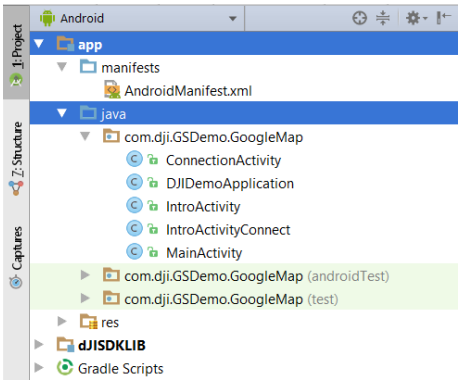
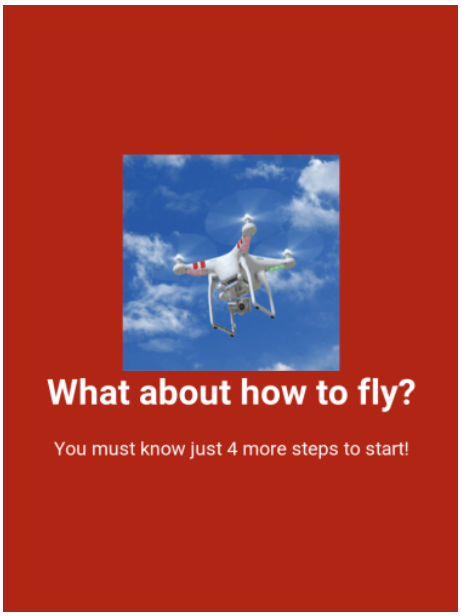



Figura 30: Ubicación de los archivos de código *Java*

Generalmente, si se ha tenido una iniciación correcta en el entorno de programación *Android Studio*, esto se da por sabido, pero será útil para diferenciarlo de los *layouts*.

- *Layouts*:
  Los *layouts* son archivos de texto que contienen código, pero en un formato de programación distinta al de los archivos *Class*, puesto que el código de los *layouts* no incide directamente en la operativa de la aplicación, pero se corresponde con la parte gráfica.
  Esta parte gráfica, luego será llamada mediante el código principal de *Java* para ser posteriormente visualizada en pantalla, de acuerdo con el código que la define dentro de los archivos *layout*.




(a) Parte gráfica

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/bg_screen1">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:gravity="center_horizontal"
        android:orientation="vertical"
        android:weightSum="1">

        <pl.droidsonroids.gif.GifTextView
            android:layout_width="181dp"
            android:layout_height="181dp"
            android:background="@drawable/flyintro" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="What about how to fly?"
            android:textColor="@android:color/white"
            android:textSize="30sp"
            android:textStyle="bold" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:paddingLeft="40dp"
            android:paddingRight="40dp"
            android:text="You must know just 4 more steps to start!"
            android:textAlignment="center"
            android:textColor="@android:color/white"
            android:textSize="16sp" />

    </LinearLayout>

</RelativeLayout>
```

(b) Parte de código

Figura 31: Correspondencia código vs interfaz

La comparativa nos ayuda a comprender (comparativa para ver la carga de código que requiere, ya que en la imagen de la derecha no se puede apreciar su lectura con total claridad) que para realizar esta pantalla tan sencilla de la interfaz, se necesita incluir a la vez diferentes conceptos como la localización del *.gif* y de los textos, e incluso mezclar varios tipos de *layouts*.

Los tipos más comunes de layouts y utilizados en esta aplicación son:

- *Linear Layout* [5], el cual proporciona una ordenación automática de los elementos gráficos, basándose en el criterio del orden en que van siendo introducidos. Es el más sencillo de utilizar pero no se recomienda si necesitamos ubicar objetos en sitios concretos deseados.
- *Relative Layout* [6], el cual permite localizar y ubicar los elementos en la pantalla a través de sencillos comandos que marcan las distancias a los bordes de pantalla.
- *Frame Layout* [7], es el más complejo de los tres mencionados y no se utilizará a no ser que se necesite aislar una parte de la pantalla para realizar una actividad en concreto a parte del resto de la interfaz, como es nuestro caso para aislar la parte del mapa.

Por este motivo, para crear pantallas completas donde podamos visualizar a la vez un gran número de botones, y a su vez una parte de la pantalla para el mapa (además de tener una interfaz doble que cambia con la acción de un botón de mapa a modo cámara y viceversa), necesitaremos establecer un gran número de combinaciones así como mezclar los tipos de *layout* dentro de un archivo *.xml*, que son los que contienen este tipo de código.

También tendremos que utilizar las opciones de visibilidad que se explicarán en el siguiente subapartado.

Para el tutorial del inicio habrá que separar el *layout* en dos:

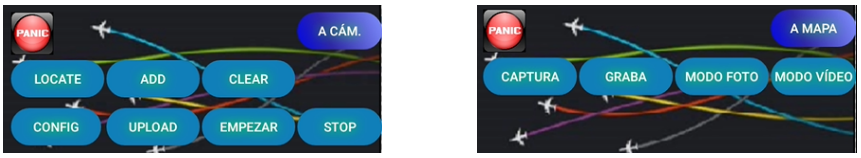
Una será la parte superior en la cual se irán pasando las pantallas, que nos dan la información en un formato deslizadera (*slider*) arrastrando con el dedo, y la inferior es la que contendrá el estado del avance que llevamos mediante unos puntos indicadores.

Este tutorial introductorio ha requerido otro archivo de código específico para él, por la necesidad de relacionar varios *layouts* en movimiento (los de la parte superior) junto a la parte fija de la base de la pantalla.


Figura 32: Interfaz separada del tutorial

- Opción de visibilidad:

  Esta opción no se caracteriza por tener en absoluto un nivel de dificultad elevado, pero es necesario conocerla porque es de gran utilidad.
  En este caso, se ha utilizado para compactar la interfaz de cámara y mapa en una, como se ha comentado anteriormente , a través de un botón.
  Este, fijará o eliminará la visibilidad de los botones en función de la parte de la interfaz en la que nos encontremos, mediante el comando $ObjetoCualquiera.setVisibility(View.VISIBLE/INVISIBLE)$ según se quiera activar o desactivar.




(a) Parte visible: Mapa (b) Parte visible: Cámara

Figura 33: Intercambio de interfaces y visibilidad

Por tanto, con esta opción hemos podido aunar las interfaces de un modo muy sencillo y evitar tener que ir lanzando *intents* (cambiar de un archivo a otro internamente, se explicará en el siguiente apartado) entre interfaces y métodos.

- Intercambio entre archivos de código *Class*: *Intent* [8]

  El comando *Intent*, que se basa en una estructura sencilla como se puede comprobar en la siguiente imagen, tiene la finalidad de establecer el intercambio de información con tareas inactivas hasta el momento, es decir, parte desde el archivo que se está ejecutando hacia el archivo que se quiere pasar a ejecutar.

```
private void launchHomeScreen() {
    startActivity(new Intent(this, ConnectionActivity.class));
    finish();
}
```

Figura 34: Estructura del *Intent*

No debe confundirse con la acción que se recibe cuando se actúa sobre un botón por ejemplo. Este ejemplo en particular, devolverá la respuesta que nosotros le hayamos asignado al botón en cuestión dentro del código, pero la información accedida estará toda dentro del mismo archivo *Class*.
En cambio con un *Intent*, somos capaces de acceder a otros archivos *Class* entre todos los que tengamos disponibles dentro de la carpeta java (Se puede localizar dicha carpeta en la figura (30), y observar como para un proyecto de tamaño medio-grande, se van a tener varios archivos *Class*).

- Icono de la aplicación:

  El icono de la aplicación es un elemento que se suele pasar por alto una vez nos metemos de lleno en el código, no obstante es uno de los elementos más importantes dejando las funcionalidades a parte.
  El icono es el elemento que sin conocer la aplicación nos proporcionará la primera impresión y posiblemente será lo que nos decante a descargarla o no, aunque posiblemente no sería el caso, porque esto está destinado a una aplicación industrial que el cliente examinaría al detalle, sin quedarse solo en la portada.
  Aún así merece la pena asegurarse de ello y ofrecer al cliente una buena impresión desde el primer momento.
  Como primera aproximación, se ha realizado el icono en los diferentes tamaños según el dispositivo de uso, con una página web [9] que los genera automáticamente a partir de la foto que le hemos proporcionado.
  Recordamos el icono personalizado:


Figura 35: Icono de la aplicación

Los tamaños que se nos generarán serán los siguientes:

- Icono de densidad baja (ldpi): 36 x 36 píxeles (no proporcionado por la página, no tiene un uso común).
- Icono de densidad media (mdpi): 48 x 48 píxeles
- Icono de densidad alta (hdpi): 72 x 72 píxeles
- Icono de densidad extra alta (xhdpi): 96 x 96 píxeles
- Icono de densidad extra extra alta (xxhdpi): 144 x 144 píxeles
- Icono de densidad extra extra extra alta (xxxhdpi): 192 x 192 píxeles

- Opción *Launcher* (Archivo *Manifest*):

Como hemos visto en el apartado de introducción al lenguaje *Java*, el archivo *manifest* contendrá información esencial de la aplicación necesaria antes de iniciarla, como permisos, nombre del paquete identificador, actividades incluidas, contraseñas (aquí se ubicarán las claves comentadas anteriormente)...
Pero en este caso en concreto vamos a prestar atención al apartado de las actividades. Entre todas las actividades que tenga nuestra aplicación , las cuales estarán reflejadas en el archivo *manifest*, habrá una que tenga el siguiente aspecto:

```
<activity
    android:name=".IntroActivityConnect"
    android:configChanges="orientation|screenSize|keyboardHidden|keyboard"
    android:label="Drone Mission Maker"
    android:screenOrientation="portrait">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figura 36: Actividad lanzada al iniciar la *app*

La declaración de una actividad como *Launcher* puede parecer un comando obvio, que siempre debe ubicarse en la actividad que va a ser lanzada en primer lugar, no obstante es más versátil que solamente esto.
En la aplicación, por requerimientos del fabricante de drones y los pequeños macros que nos proporciona, no podemos abrirla sin tener primero un producto conectado. Pues bien, el problema se solventará de un modo tan sencillo como abrir el código con *Android Studio*, y declarar como *Launcher* la actividad que queramos abrir e inspeccionar con nuestro dispositivo móvil.
En resumen, nos proporciona una forma rápida de poder tener acceso a todas las partes de nuestro código que estén restringidas por algún motivo en la fase de desarrollo, además de su función usual.

- Eventos de pulsación (*click*):

  El método *onClick*() [11] es un método destinado a las funcionalidades de los botones. Se ejecuta directamente en el hilo principal para evitar demoras, puesto que cuando pulsemos un botón, se va a querer una respuesta inmediata.

  Observamos el método *onClick*() a continuación , con el comando *Switch-Case* y unos botones dentro como ejemplo, se explicará en el siguiente apartado.

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        ////
        case R.id.btn_capture:{
            captureAction();
            break;
        }
        case R.id.btn_shoot_photo_mode:{
            switchCameraMode(SettingsDefinitions.CameraMode.SHOOT_PHOTO);
            break;
        }
```

Figura 37: Ejemplo método *onClick*()

Por otra parte, este método solo define la operativa de los botones, qué harán una vez hayan sido pulsados, pero necesitamos definirlos y habilitar el *listener* de los botones, para que cuando sean pulsados, guarden y transfieran dicha acción al método *onClick*().

La definición de botones es un paso muy sencillo en el cual no se va a hacer mención especial, pero sí mostramos como ejemplo, cómo se habilita el *listener* de un modo tan sencillo como el siguiente:

```
mCaptureBtn.setOnClickListener(this);
```

Figura 38: Declaración del *listener* para un botón

(El botón *mCaptureBtn* se corresponde con el identificador *btncapture*).

Además de este sencillo método que sirve para botones normales, también tenemos otros botones más complejos como lo que es un *Toggle Button*, un botón de dos estados, activado o desactivado, que tiene su propio apartado de código fuera de este método, a pesar de que también es un evento *click*.

No obstante, el botón se ha desarrollado con relativa sencillez, y se basa de nuevo en definir otro tipo de *listeners*, aunque guarda bastante relación con el método *onClick*().
No se pretende explicar de nuevo un método similar como es el *onCheckedChanged*() [12], así que se va a proporcionar la estructura en la siguiente imagen para observar cómo se organiza:

```
SwapButton.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {

    }
}) ;////
SwapButton.setOnCheckedChangeListener((buttonView, isChecked) -> {
        if (isChecked) {
```

Figura 39: Estructura *Toggle Button*

Es obvio que en este método falta por incluir un *else* (además de los corchetes finales para cerrarlo), en el cual se incluye qué hace el botón cuando está en el estado desactivado, pero no se muestra en pantalla porque el código que contiene nuestro botón es excesivamente largo.

Por último, necesitaremos explicar el evento más complejo que contiene nuestro código en cuanto a eventos *click* se refiere, el *onMapClick*() [13].
Este método no contiene dificultad en cuanto a carga de código se refiere, pero por otra parte necesitamos saber una serie de comandos específicos.

Para empezar, necesitamos definir el mapa como un archivo "*private GoogleMap* NuestroMapa", que como cabe esperar no es un formato soportado a priori por *Android Studio*, por lo que habrá que realizar una serie de importación de herramientas a priori (como puede ser "*import com.google.android.gms.maps.GoogleMap*;" entre muchas otras, aunque *Google* nos lo pone fácil con su método autocompletar (Alt+Intro sobre la herramienta que no funcione por no estar importada).

A continuación, crearemos un método de espera que se ejecute cuando el mapa se haya cargado y esté listo , habilitará el *listener* del mapa:

```
private void setUpMap() {
    gMap.setOnMapClickListener(this);// add the listener for click for amap object

}
```

Figura 40: *Listener* del mapa

Por último, con la escucha en modo activado, ya podemos cargar el método *onMapClick*(), en el cual, cuando se pulse el mapa, añadiremos dichas pulsaciones a una lista de *waypoints* que se caracterizarán por la latitud y longitud que se haya pulsado sobre el mapa.

Se puede observar en la imagen que la altitud no se obtiene directamente como *point.altitude* al igual que latitud o longitud, porque no podemos conseguirla en un entorno 2D. Esta se define manualmente en el *Setting Dialog*, punto por punto, herramienta implementada por nosotros.

```
public void onMapClick(LatLng point) {
    if (isAdd == true){
        markWaypoint(point);
        Waypoint mWaypoint = new Waypoint(point.latitude, point.longitude, altitude);
        //Add Waypoints to Waypoint arraylist;
```

Figura 41: Obtención directa de latitud y longitud a través del mapa, altitud fijada manualmente

- Comando *Switch-Case*:

  Este comando es bastante conocido también fuera de este lenguaje, puesto que es una herramienta básica. Tiene una explicación bastante sencilla, pero dada su importancia en la elección de las acciones de nuestros botones, se va a explicar brevemente, y de un modo más sencillo y visual apoyándonos en la siguiente figura:

```
switch (v.getId()) {
    ////
    case R.id.btn_capture:{
        captureAction();
        break;
    }
    case R.id.btn_shoot_photo_mode:{
        switchCameraMode(SettingsDefinitions.CameraMode.SHOOT_PHOTO);
        break;
    }
```

Figura 42: Estructura del *Switch-Case*

Como podemos ver, al lado del *switch*, tenemos el comando $v.getId()$, el cual va a elegir entre los identificadores de todos los botones que tengamos en el archivo *Class*.

Una vez se identifica, pasamos a la parte de los *case*, en los que para cada identificador hay una acción determinada (por ejemplo para el identificador de botón captura, tiene la acción de tomar foto), seguida del comando break que se asegura de que una vez se ha realizado la acción, se corta de raíz el método $onClick()$ para que no haya confusiones internas.

- *Setting Dialog*:

  Este apartado está dedicado a la parte de introducción de datos sobre los *waypoints*. No contiene un código en especial, pero si una combinación de conceptos conocidos que nos ayudarán a entender mejor como funciona este entorno de programación, y la aplicación en general.

  Aunque todavía no se han comentado las funcionalidades de nuestra aplicación, una vez introducidos los *waypoints* mediante el evento *onMapClick*() anteriormente explicado , debemos darle las características con las que el dron sobrevolará dichos puntos. Esto se llevará a cabo con el *Setting Dialog*.

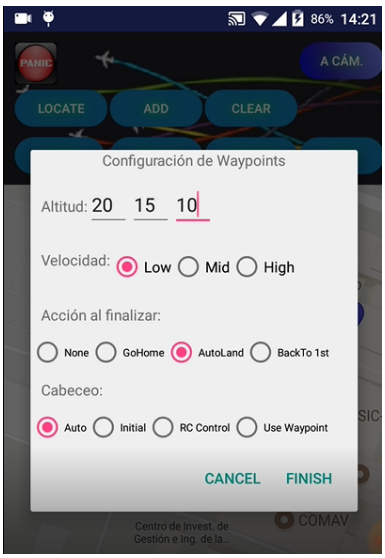



Figura 43: Aspecto del menú de introducción de datos

Las funcionalidades que incluye como se observa directamente, son ajustar la altitud punto a punto, velocidad del trayecto, acción al finalizar la misión y modo de cabeceo. Se explicarán más al detalle en el apartado de funcionalidades, ahora nos centraremos en cómo se ha hecho esto.

Para empezar, creamos un método que se active al pulsar el botón de configuración "Config":

```
case R.id.config:{
    showSettingDialog();
    break;
}
```

Figura 44: Acción del botón configuración

Y el código correspondiente al método *showSettingDialog*() se inicia del siguiente modo (el resto del código se explicará puesto que su longitud es acusada):

```
private void showSettingDialog(){
    LinearLayout wayPointSettings = (LinearLayout)getLayoutInflater().inflate(R.layout.dialog_waypointsetting, null);
```

Figura 45: Inicio de la operativa del *Setting Dialog*

Para empezar, tenemos el comando *inflate* [14] que nos va a mostrar por encima del *layout* principal de la aplicación el *layout* propio correspondiente al menú de introducción de datos.

A continuación, solo vamos a definir de un modo sencillo, como en apartados anteriores, todos los *listeners* y *EditText* (Textos que pueden ser modificados con el teclado por el usuario) necesarios para almacenar dichos datos, y proporcionar algún tipo de respuesta una vez introducidos.

El siguiente paso será tratar estos datos, según las variables analizadas, la mayoría de ellas de un modo directo, como por ejemplo, si le dijésemos al programa que la última acción a realizar es quedarse quieto sin realizar acciones, tendríamos que programarlo como:

```
mFinishedAction = WaypointMissionFinishedAction.NO_ACTION;
```

Figura 46: Ejemplo de opciones sobre *waypoints*

Estas grandes librerías que nos proporciona *DJI*, donde se incluyen métodos como el visto, $WaypointMissionFinishedAction$ , y todos los otros que podamos imaginar equivalentes al *heading* , velocidad, etcétera, nos van a ahorrar un gran tiempo de trabajo y facilitar mucho las cosas a la hora de transmitir la información al dron, aunque debemos ser precavidos porque a su vez, es una información bastante cerrada sin mucha explicación, y muchas veces será necesario el ingenio de un operario cualificado para poder aprovechar dichas ventajas y que no se tornen en nuestra contra.

Por último, toda esta información almacenada y procesada con estos métodos externos (se comentarán algunos más en el apartado siguiente) , se gestiona en el bucle $configWayPointMission()$.

En este lugar, se llama a la lista de *waypoints* anteriormente almacenada en el evento *onMapClick*(), y se fijan sobre la misma los datos mediante un *waypointMissionBuilder*.

Será más clarificador observar su funcionamiento directamente:

```
if (waypointMissionBuilder == null){

    waypointMissionBuilder = new WaypointMission.Builder().finishedAction(mFinishedAction)
            .headingMode(mHeadingMode)
            .autoFlightSpeed(mSpeed)
            .maxFlightSpeed(mSpeed)
            .flightPathMode(WaypointMissionFlightPathMode.NORMAL);

}else
{
    waypointMissionBuilder.finishedAction(mFinishedAction)
            .headingMode(mHeadingMode)
            .autoFlightSpeed(mSpeed)
            .maxFlightSpeed(mSpeed)
            .flightPathMode(WaypointMissionFlightPathMode.NORMAL);

}

if (waypointMissionBuilder.getWaypointList().size() > 0){

    for (int i=0; i< waypointMissionBuilder.getWaypointList().size(); i++){
        //waypointMissionBuilder.getWaypointList().get(i).altitude = altitude;
        final float [] setalt =new float[]{altitude,altitude2,altitude3,altitude4,altitude5,altitude6};
        waypointMissionBuilder.getWaypointList().get(i).altitude = setalt[i];
    }
```

Figura 47: Paso final de fijación de datos

Observamos cómo se fijan todos los datos a partir de las variables definidas anteriormente ( $mFinishedAction$, $mHeadingMode$, $mSpeed$, $altitude$) sobre el *builder* que será el encargado de aplicarlos sobre la lista de *waypoints* a su vez.

No obstante, la parte de la altitud es ligeramente diferente , y esto es debido a que esta también estaba fijada como constante en el macro proporcionado por *DJI* [15] , pero la hemos modificado para que sea variable puesto que en el análisis de una pala de aerogenerador, necesitaremos variar la altitud durante la misión.

- Métodos externos (product instance, waypoints, camera...):

  La aplicación que se está desarrollando, tiene amplias funcionalidades implementadas por nosotros, pero merece la pena destacar el trabajo de los desarrolladores de *DJI* por sus completos macros proporcionados.

  De estos vamos a extraer mucha información que difícilmente se puede comprender con una rápida lectura , pero vamos a tratar de exponer algunos de interés o mayor uso.

  - *Flight Controller*:
    Nos permite acceder mediante comandos sencillos a las funcionalidades de localización de nuestra aeronave, es decir, gestiona "por nosotros" la localización y parámetros de vuelo del dron.
  - Clase *Waypoint* (Mission, *Builder*):
    Son las opciones que nos permiten hacerle entender de un modo directo al dron, que aquello que hemos introducido como una pulsación en la pantalla, es en realidad un lugar al que debe dirigirse y aplicar ciertas propiedades sobre él.
  - Cámara:
    Mediante el método *getInstanceCamera*() y el *callback* correspondiente (llamada al método que se ejecuta ante una cierta acción), entre muchos otros métodos internos como por ejemplo *startShootPhoto*() , se encarga de establecer los procesos necesarios (internamente) para llamar y activar la cámara del dron y permitir que esta haga fotos, además de enviarla al directorio de la tarjeta SD que lleve el dron insertada.
    De un modo muy similar con otros comandos sobre la cámara, conseguimos que realice el mismo proceso pero en vídeo.

    Sobre funcionalidad de vídeo y foto, más allá de como se gestiona internamente el proceso móvil-dron, podemos y debemos actuar implementando el código necesario.
    No se caracteriza por su sencillez puesto que debemos aunar las propiedades de varios macros y solucionar los problemas que van surgiendo. No obstante, para este apartado en concreto, sí tenemos una buena información en la página de desarrolladores [16].

- Conexión:

  Una vez visto el tutorial que se ha creado previamente a la aplicación, esta se lanzará hacia el archivo *Class ConnectionActivity*. Este nos bloqueará el paso hacia el contenido de la aplicación, pero tiene un motivo esencial por el cual nos impide el paso: asegurarse mediante métodos internos (de nuevo) de que lo que está conectado a la aplicación, es un dron oficial y cumple los requisitos (versiones, si es un dron suficientemente nuevo para que funcione bien de acuerdo con la aplicación) , es decir, asegurarse de que el producto está correctamente conectado para poder funcionar.

  No debemos preocuparnos por este aspecto en exceso, puesto que aparece explicado en cualquiera de los tutoriales incluidos en la página, como por ejemplo, el de la cámara anteriormente mencionado [16].

- *Broadcast Receiver* [43]:

  Un *Broadcast Receiver* es el componente que está destinado a recibir y responder ante eventos globales generados por el sistema, como un aviso de batería baja, un SMS recibido, una llamada...

  Para su mejor comprensión, vemos el ejemplo de nuestra aplicación , donde está siendo usado para detectar el evento de conexión del dron a la aplicación, mediante la actualización del método *OnProductConnectionChange()* dentro del *OnReceive*(), cuando se detecta un cambio de estado en el producto.

  El método *OnProductConnectionChange()* a su vez, inicializará el controlador de vuelo y actualizará la posición de nuestro dron.

```
protected BroadcastReceiver mReceiver = new BroadcastReceiver() {

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

private void onProductConnectionChange() { initFlightController(); }
private void initFlightController() {

    BaseProduct product = DJIDemoApplication.getProductInstance();
    if (product != null && product.isConnected()) {
        if (product instanceof Aircraft) {
            mFlightController = ((Aircraft) product).getFlightController();
        }
    }

    if (mFlightController != null) {
        mFlightController.setStateCallback(new FlightControllerState.Callback() {

            @Override
            public void onUpdate(FlightControllerState djiFlightControllerCurrentState) {
                droneLocationLat = djiFlightControllerCurrentState.getAircraftLocation().getLatitude();
                droneLocationLng = djiFlightControllerCurrentState.getAircraftLocation().getLongitude();
                updateDroneLocation();
            }
        });
    }
}
```

Figura 48: *Broadcast Receiver* a la espera de la conexión del producto

- Proveedores de contenido (*Content Providers*) [42]:

  Los proveedores de contenido administran el acceso a un conjunto estructurado de datos, encapsulándolos , es decir, son la interfaz estándar que conecta datos en un proceso con código que se ejecuta en otro proceso.

  *Android* incluye proveedores de contenido que administran datos como vídeo, audio, imágenes e información de contacto personal.

## 6.3 Funcionalidades

La aplicación desarrollada está destinada al análisis de palas de aerogeneradores, por lo cual tiene que tener amplias funcionalidades que proporcionar al cliente.

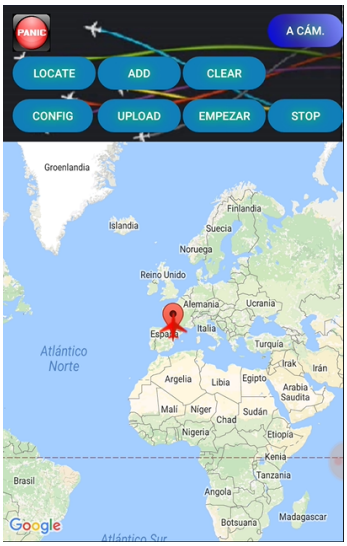
En la fase beta de esta aplicación, en la cual nos encontramos, se proporcionan los servicios de la cámara para tomar imágenes o vídeo, los cuales puedan ser post-procesados, y un sistema de movimiento del dron en el cual se puede actuar sobre el mapa (latitud , longitud) y altitud, para poder definir trayectorias.

Además, incluye un sencillo sistema que bloqueará todas las acciones y nos permitirá retornar el dron a casa a salvo, en el caso de que algo falle.

### 6.3.1 Accesibilidad

Nuestra aplicación tiene un sistema de uso sencillo, que parte de dos pequeños tutoriales cortos , para no perder la atención del cliente, pero a su vez necesarios, para saber cómo conectar la aplicación al dron y cómo volarlo.

Una vez dentro de la misma, siempre y cuando se haya conectado correctamente (como hemos visto, el archivo *ConnectionActivity*se va a ocupar de asegurarlo), nos vamos a encontrar con la interfaz del mapa, la cual contiene unos botones bastante sencillos que indican las acciones que realizan.




(a) Interfaz del mapa


(b) Interfaz de la cámara

Figura 49: Interfaz disponible

Como podemos ver, se trata de una interfaz sencilla (en la cual los botones del mapa no han sido modificados de idioma, para que encajen de forma óptima, pero sus órdenes son sencillas y se pueden comprender perfectamente con el tutorial previo) que cambia según el *Toggle Button* que tenemos arriba a la derecha (ya comentado en apartados anteriores), habilitando la visión del mapa o de la cámara según convenga y sus respectivos botones.
El botón "Panic" nunca desaparece, y descubriremos en detalle su significado en el apartado de seguridad.

Esta interfaz también nos devolverá *Toasts* informativos (Mensajes de corta duración que aparecen en la base de la pantalla) acerca de si la acción que hemos realizado, se ha completado o no con éxito.

Por último y no menos importante, cabe destacar que puesto que se trata de una aplicación con visión industrial, no podemos olvidar que no todos los clientes serán españoles, de hecho es bastante improbable.
Con este motivo, se ha traducido la aplicación dentro de nuestras posibilidades a Inglés, Castellano y Portugués, y se activarán según el idioma del dispositivo [17].

Esto se consigue dirigiéndonos a la carpeta *strings.xml*, con la siguiente ruta:

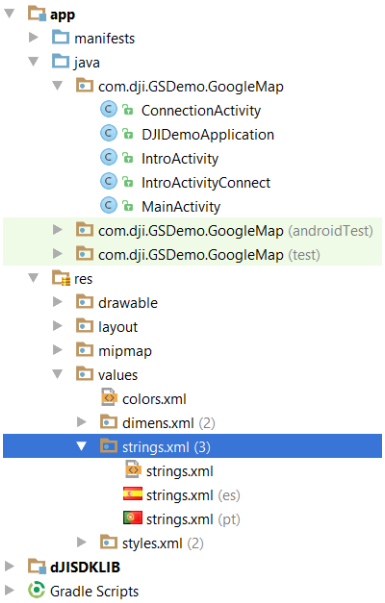



Figura 50: Carpeta contenedora de los textos de la aplicación

Una vez aquí, nos dirigiremos al archivo de mismo nombre, *strings.xml*, y al abrirlo, pulsaremos la opción *Open editor*:

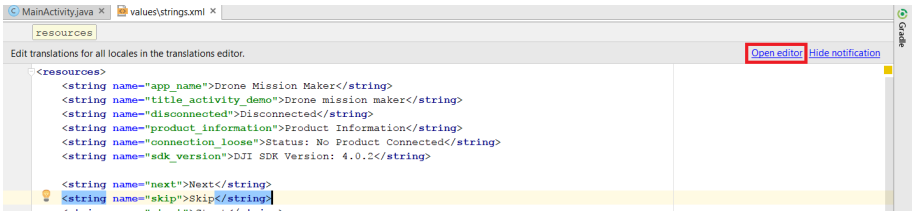



Figura 51: Opción que habilita el modo editor para idiomas

Una vez dentro de editor, tendremos que escoger el idioma para el cual vamos a hacer la aplicación multiidioma , y reescribir todos los textos que tenga en el idioma deseado.

Es **muy importante** ubicar todos los textos que vayamos a utilizar en el archivo *strings.xml*, y llamarlos en el código por su identificador (ya sean archivos *Class* o *layouts* (xml)), puesto que si no se realiza así, es evidente que dicha cadena no aparecerá en el archivo *strings.xml* y no podrá ser traducida.

### 6.3.2 Uso de la cámara

Al inicio de la sección de "Funcionalidades", se ha introducido la funcionalidad de la cámara y en la explicación del código en "Conocimientos previos", su operativa.
De este modo, queda por explicar tan solo su funcionamiento.

Como hemos visto en el apartado accesibilidad , la interfaz incluye los siguientes botones:

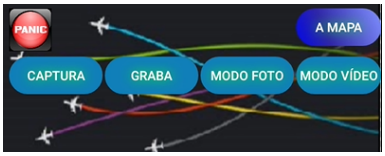



Figura 52: Parte visible: Cámara

Los botones de interés son: modo foto, modo vídeo, graba y captura.
Como se puede deducir, se debe hacer entender a la aplicación , además de que estamos usando la cámara, en qué modo se está haciendo, cámara o vídeo.
En consecuencia, pulsaremos primero el modo que queremos utilizar, y cuando aparezca el *Toast* (mensaje informativo) de que se ha realizado con éxito, procederemos a pulsar el botón "captura" o "graba" , según el modo que estemos usando.

Además, se ha introducido una funcionalidad sobre el modo vídeo, que nos muestra debajo del botón "graba" un temporizador con el tiempo que llevamos grabado.
Finalmente, todo esto se guardará en la tarjeta SD que lleve el dron.

### 6.3.3 Uso del creador de misiones mediante *GPS*

En esta sección se explicará como funciona el creador de misiones, o dicho de otro modo, como lo haremos funcionar como usuarios, puesto que el camino que sigue el código ya ha sido explicado anteriormente.

Toda esta explicación se incluirá de forma sintetizada en el tutorial de inicio, para que el usuario se sienta en todo momento guiado y cómodo con el uso de la aplicación.
A continuación, se muestran los botones de la interfaz de la parte del mapa:

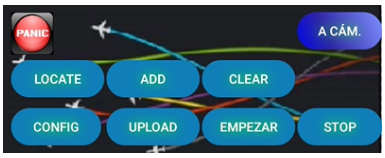



Figura 53: Parte visible: Mapa

El procedimiento a seguir será el siguiente:
Primero, una vez esté el dron conectado, y en un lugar con una buena recepción de la señal GPS (imprescindible, todo el guiado se lleva a cabo por GPS en esta aplicación, aunque no hay que preocuparse por la integridad del dron: si no hay suficiente señal, nos devolverá un error generado automáticamente por su librería interna y no nos dejará despegar), pulsaremos el botón *Locate*, el cual nos devolverá la posición del dron sobre el mapa.

A continuación, pulsaremos el botón *Add*, que nos habilita el evento *onMapClick*(), y añadiremos los waypoints pulsando sobre el mapa.
No debemos olvidar volver a pulsar el botón (que se llamará *Exit* cuando estemos en el modo *onMapClick*() activado) una vez terminada la adición de waypoints para continuar con éxito.

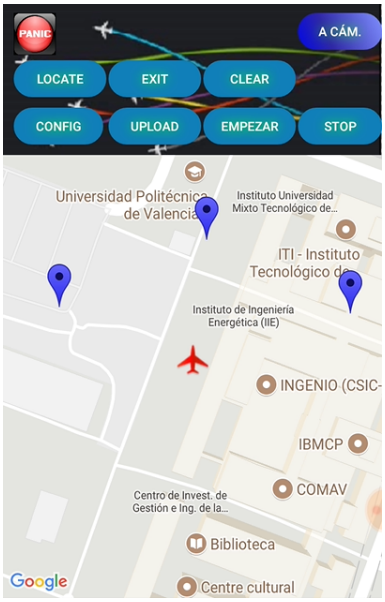



Figura 54: Adición de *waypoints* manualmente

El siguiente paso será pulsar el botón *Config*, que abrirá el menú de introducción de datos *Setting Dialog* , en el cual introduciremos la altitud de *waypoints*, velocidad, cabeceo y acción al finalizar.
Este proceso ha sido explicado con mayor detalle en el apartado de "Conocimientos Previos" , por lo que no se van a volver a adjuntar las imágenes, podemos recurrir a dicho apartado si se requiere ver de nuevo con mayor detalle.

Los últimos dos pasos serán pulsar el botón de *Upload*, que realiza la simple acción de cargar los datos en la lista de *waypoints*, y por último el botón "Empezar", que será el encargado de dar comienzo a la misión.

A lo largo de la duración de la misma, tendremos el botón *Stop* para detener la misión, y podemos utilizar el modo cámara a la vez, mientras el modo mapa se ejecuta en segundo plano sin ningún problema.

También observamos el botón *Clear* , que sirve para borrar la misión antes de iniciarla, por si nos hemos equivocado en la introducción de datos.

## 6.4 Seguridad ante situación desfavorable de la misión

Durante la misión automática de nuestro dron, este realizará todo lo que se le haya pedido.
No obstante, puede que hayamos incurrido en errores, y el hecho de pulsar el botón *Stop* no será suficiente, puesto que luego tendremos que crear otra misión para volver a casa, y lamentablemente la acción *Go home* no registrará la partida del dron desde donde estamos, sino desde el punto donde se pulsó el botón "Stop".

Estos problemas que genera el macro proporcionado por *DJI* [15], hacen peligrar la integridad del dron ante un error humano.

Con la finalidad de evitar este conflicto, se ha desarrollado el botón *Panic*, que luce del siguiente modo en la parte superior izquierda de la interfaz (tanto en modo cámara como en modo mapa):


Figura 55: Botón de seguridad *Panic*

A continuación, se va a explicar el procedimiento que se sigue para llevar el dron de vuelta a casa con seguridad.

En primer lugar, se ha añadido una funcionalidad sobre el botón *Locate*, que además de hacer la función para la cual está destinado, permite almacenar la latitud y longitud de la cual se parte, pero solo la primera vez que se pulsa, y a su vez se guarda como variable global.
Es decir, esta posición inicial se guarda siempre e invariante hasta que no cerremos la aplicación del todo.

Con esto, se va a conseguir , aún pulsando el botón *Stop* o *Clear*, necesarios para definir la misión de vuelta a casa, guardar la posición de *Home* pase lo que pase.

Una vez cumplido este importante punto, seguirá el camino de una misión normal:
Primero, se pulsa el botón *Stop* (estas pulsaciones son ficticias, es decir, se llama el método internamente, así como todos los siguientes botones que vamos a decir que se pulsan) de forma inmediata y se incluyen unos *Delays* (retrasos), necesarios para que el sistema no colapse y pueda gestionar todas las órdenes.

Esto es necesario porque cuando se introducen los datos o pulsaciones a mano, se da un tiempo de descanso a la aplicación entre dato y dato, pero cuando se hace todo automatizado, si se carga todos los datos de golpe, se colapsará.

El siguiente comando aplicado será limpiar toda la lista de *waypoints* y opciones anteriores, y localizar donde se encuentra el dron actualmente (mediante parte de los comandos que incluyen los botones *Clear* y *Locate*).

Lo siguiente, será realizar de nuevo una misión, que tiene como punto incial la localización actual y como punto final, la latitud y longitud que se había guardado imperturbable anteriormente como variable global.

Puesto que se borraron todos los datos, se actúa fijando como acción final *AutoLand* (aterrizaje automático) cuando lleguemos a casa, porque lo común será que el usuario quiera aterrizar si algo ha ido mal.
Además se fijará una velocidad de retorno lenta, para la tranquilidad del usuario, un modo de cabeceo automático, para que no haya que hacer nada, y todo este trayecto se realizará a una altitud de 30 pies , evitando así los posibles obstáculos que pueda haber en el camino de retorno a casa.

En resumen, este botón proporciona la seguridad al cliente de que en caso de fallo inminente, no va a perder el costoso dron y lo recuperará con total seguridad.

# 7 Resultados

Después de todo el trabajo expuesto, queda exponer los datos obtenidos mediante el simulador, y también la prueba de exteriores.

## 7.1 Test en simulador

En este apartado se van a exponer los datos obtenidos de una misión procesada en simulador, la cual tenía como finalidad probar y comprobar el correcto funcionamiento de todas las funcionalidades de la aplicación.
A partir de aquí, con el correcto funcionamiento que se verá a continuación, se puede adquirir la confianza necesaria para lanzar el dron al vuelo, evitando cualquier problema relacionado con nuestra aplicación.

Para empezar, seguiremos el tutorial que expone la página de *DJI* sobre el simulador [21], el cual no conlleva una mayor dificultad que instalar el programa, conectar el dron para que este lo detecte , cargar nuestra localización GPS (tendremos que obtener las coordenadas en las que nos encontramos) y empezar.

Tan solo en el inicio se nos presentan dos problemas:

El primero, pasa por conseguir aproximar las coordenadas sobre el simulador de una forma aceptable, puesto que tendrá que tener coherencia con la señal GPS que detecta directamente el móvil para la aplicación (Puesto que sino, difícilmente funcionará, ya que encontrará discrepancias y se incurrirá en errores que no nos permitirán despegar).
No obstante, se podrá solventar el problema, ya que existen páginas web como la propia que se usa en la aplicación, *Google Maps* [22], que nos proporcionan la ubicación con suficientes dígitos de precisión.

El siguiente problema que se encontrará, es, como a lo largo de todo el desarrollo de la aplicación, que a pesar de que *DJI* proporciona ayudas como el código libre para desarrollar, por otra parte impide cualquier tipo de *feedback* en lo que estamos desarrollando sin tener un dron en posesión.

Por lo tanto, se requiere el dron enchufado para poder realizar esta prueba.

A continuación se muestra el entorno virtual en el cual trabaja el simulador.
Se adjuntarán los vídeos realizados en la aplicación , en inglés y castellano, en los cuales sí se puede observar la correcta acción y respuesta de la aplicación.

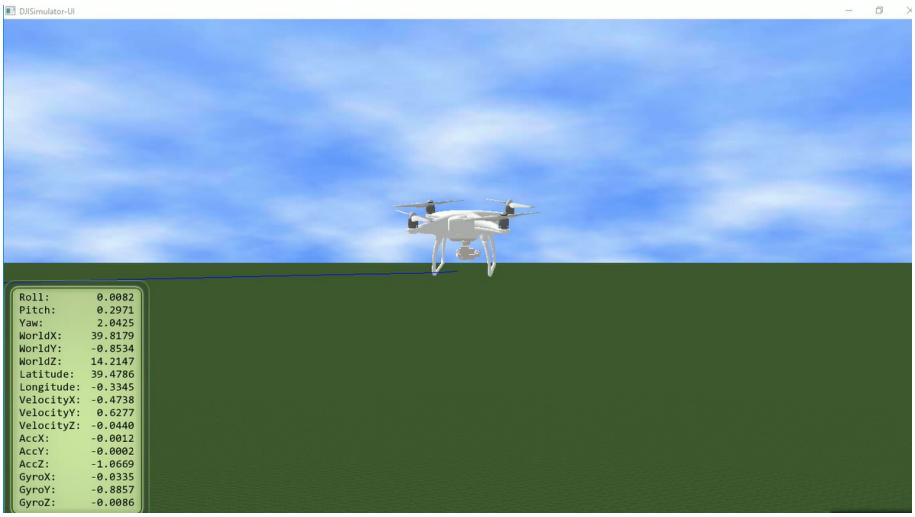



Figura 56: Simulador *DJI*

En el recuadro de la izquierda, encontraremos las útiles coordenadas *WorldX*, *WorldY* y *WorldZ*, en las cuales podremos comprobar longitud y latitud , y sobretodo la altitud con *WorldZ* para ver si realiza los cambios correctamente.

Como apunte, cabe destacar que en este apartado, se observó un comportamiento erróneo en la cámara de vídeo, pero se acabó comprobando que corresponde a la velocidad de lectura del móvil (insuficientemente alta), es decir, el vídeo grabado y capturas se observan correctamente en la tarjeta SD del dron , a pesar de que no siempre en pantalla, por lo que podríamos seguir analizando nuestra hipotética pala, y obtener datos correctos de esta sin ningún problema.

## 7.2 Prueba de campo

La prueba de campo será necesaria para constatar que la aplicación funciona definitivamente de un modo correcto.

Para la realización de la misma y según la normativa, se ha tenido que realizar el desplazamiento a los exteriores de la ciudad en anteriores ocasiones.

Por restricciones del departamento, aunque la aplicación incluye los sistemas de seguridad vistos anteriormente, no nos es permitido volar muy cerca de las inmediaciones de un aerogenerador sin tener un piloto listo, por si hubiese algún problema de gravedad.

Aunque las tomas no están efectuadas con la aplicación, siempre que fijemos la altitud correcta entre raíz y punta de pala (la transición entre ambas es lineal) obtendremos las imágenes deseadas.

De todos modos, la aplicación ha sido probada satisfactoriamente en simulador, así como en vuelo real, con la salvedad de que se ha realizado sin hélices (sosteniéndolo con nuestras manos) , simplemente para comprobar que realizaba cambios en la potencia de los motores en momentos como el ascenso del botón de seguridad o giros, y que el código leía y transfería los movimientos y acciones pertinentes al dron sin ningún problema, exactamente igual que en el simulador.

Por otra parte, tratando el tema de que la aplicación se sigue encontrando en fase beta, sin predicción de trayectoria, si el objetivo es analizar aerogeneradores, se puede asegurar que el principal problema no será disponer de varias tarjetas SD para grabar muchas tomas, y una vez en el aire, en el caso de que nuestra aplicación no diese la precisión o tomas necesarias a la primera, no existiría ningún inconveniente en realizar sucesivas misiones e ir guardando los datos para su post-proceso.

Por lo tanto, no es atrevido el hecho de asegurar que con la aplicación que se ha desarrollado, se podría obtener tomas como las siguientes (obtenidas con este mismo dron del departamento, cuando se tenía la disponibilidad de un piloto):

Figura 57: Pala con desgaste por abrasión de gota de agua

O una imagen como la siguiente, la cual ha pasado por un programa de edición gráfica para eliminar el fondo y apreciar con mayor claridad cualquier defecto:

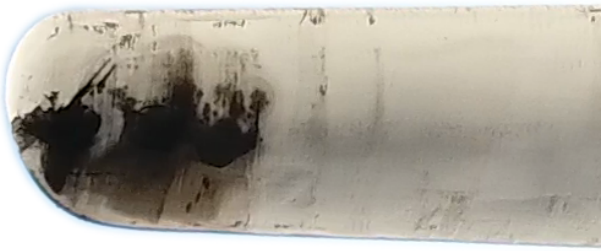


Figura 58: Punta de pala con rotura por rayo

De esta imagen, se puede ver los defectos que tenga la pala, y analizarla en las mismas condiciones o incluso mejores que con un operario colgado, por la posibilidad de revisar la información.

A simple vista sin post-proceso, los resultados ya prometen, y corroboran que el trabajo que se ha realizado no ha sido en vano, al contrario, ha tenido el enfoque puntero deseado (por ser un sector emergente) además de la utilidad que se le atribuía de ahorrar tiempo y recursos económicos.

Por otra parte, no se debe detener aquí la operación del análisis, porque como vamos a ver a continuación, con un post proceso completo, podemos aislar del fondo y reconstruir la pala con varias tomas para su posterior almacenamiento, e incluso en un futuro próximo, un sistema de software autónomo de análisis de imágenes de palas, puesto que los resultados que devuelve el dron con su cámara 4K tienen la suficiente nitidez como para propiciarlo:

Figura 59: Pala completa dañada post-procesada (Parte delantera)

Figura 60: Pala completa dañada post-procesada (Borde de ataque)

Figura 61: Pala completa dañada post-procesada (Parte trasera)

# 8 Ventajas del mantenimiento con drones

A lo largo de este trabajo, se ha extraído punto por punto las ventajas que proporcionaban los drones, por lo que se va a recapitular las más importantes de cara a nuestros intereses, el análisis con drones.

Empezaremos por los aspectos económicos:
El trabajo con drones puede parecer caro a nivel usuario, por el hecho de tener que disponer de uno. No obstante, a nivel empresarial, industrial, es un método muy barato comparado con el análisis de plataforma o de grúa hidráulica entre todos los vistos.

Además del capital invertido en revisiones, el apartado de inteligencia será también muy asequible, puesto que lo que se va a necesitar va a ser, simplemente un dron, un ordenador de potencia media y un operario con cierta experiencia en programación.

De nuevo, volviendo a los análisis a pie de campo, como se ha comentado, son baratos y fáciles ya que se realizan directamente con el terminal móvil del operario, pudiendo ser su mismo terminal personal, ahorrando de nuevo dinero.
Este hecho propicia que se puedan realizar más revisiones a menudo (hasta no tener la certeza de un mantenimiento predictivo , consultar su significado en "Tipos de Mantenimiento" si se requiere), con lo que vamos a evitar que cualquier posible desperfecto crezca.

Continuamos con los aspectos de funcionalidades:

El análisis con la cámara y mapa al alcance inmediato a través de un botón, permiten al operario tener la visión de la pala directamente en su terminal, sin la preocupación de estar seguro por el cordaje en caso de estar colgado en la pala, por ejemplo.

Sumado a todo esto, será un método rápido y seguro, con precisión y gran resolución en las imágenes, gracias a la gran estabilidad de estos dispositivos y calidad de cámaras, y con la posibilidad de tener la aplicación multiidioma, que no será necesario puesto que la mayoría de personas cualificadas entenderán de forma sencilla el poco inglés que contiene la aplicación, pero el hecho de observar que se ha hecho el esfuerzo de incorporar su idioma, les proporcionará el plus definitivo de motivación que necesitan para continuar utilizando esta aplicación.

# 9 Futuras mejoras

Puesto que se está alcanzando el final del escrito, merece la pena volver la vista hacia atrás e inspeccionar cuales son los puntos débiles y tratar de mejorarlos, o por otra parte, pulir ciertos detalles.

A continuación se van a tratar cinco puntos, los dos primeros de suma importancia en el momento que consigamos desarrollarlos, y los tres últimos enfocados a mejorar la experiencia con la interfaz:

- **Predicción de trayectoria:**

  La predicción de trayectoria actual de nuestra aplicación, pese a que se realiza con estabilización automática y sin necesidad de intervención del usuario durante la acción, sí requiere de una acción inicial del operario para fijar exactamente la altitud inicial y final de la pala.

  Puesto que el operario que vaya a analizar dicha pala, estará cualificado y sabrá sobre qué modelo de aerogenerador funciona, además de suponer la pala horizontal, podrá establecer dicha altitud con cierto rigor. No obstante, es posible que necesite realizar dos o más pasadas alrededor de la pala, porque por cualquier imprevisto, no sería extraño pensar que haya podido haber un error de altitud de medio metro, suficiente para no tener imágenes exactas a la primera.

  Pues bien, una notable mejora sobre este trabajo, sería conseguir habilitar la cámara , pero no como está hecho hasta el momento, sino una cámara que sirviese de control:
  La cámara, detectaría la forma del aerogenerador y al alcanzar la parte circular central del mismo, se quedaría en modo *Hover* (Estacionario), y seguiría a la pala que se le indicase, también mediante el seguimiento de la cámara (De un modo bastante similar al funcionamiento de un siguelíneas).

  Esta funcionalidad, junto a la del sonar, permitirían un análisis completamente autónomo, con la única necesidad del operario de dejar el dron en la base del aerogenerador y recogerlo al fin de su misión (Además de supervisar que la misión se desarrolle correctamente y pulsar el botón de seguridad en caso de que no sea así).

Esta gran mejora no se ha podido desarrollar, puesto que presenta una elevada dificultad, no tanto en la programación aunque se prevé costosa, sino mucho más en la disponibilidad de la información.

La intención inicial fue desarrollar este apartado, no obstante, no encontramos ninguna información al respecto en la página de desarrolladores ni externas, teniendo esta información lo más seguro, al hacernos usuarios *premium* de *DJI*, por lo que no se disponía ni tan solo de un pequeño hilo del cual tirar sin acceder a métodos de desarrolladores de pago.

- **Análisis autónomo de imágenes:**

  Este punto complementaría nuestro trabajo, no siendo realizado por nosotros necesariamente, no obstante se cree de interés comentarlo.

  Hoy en día , no es un proyecto inabarcable para expertos en tratado de fotos, incluir unas ciertas propiedades a un software de análisis , que determinen según el tamaño de la rotura, color, profundidad, etc, cuál es el tipo de daño al que ha estado sometida la pala.

  De esta forma, se daría un paso más en la automatización del proceso, sin necesitar horas desperdiciadas de operarios en el análisis manual de las imágenes, cuando este software podría analizarlas y almacenarlas perfectamente.

  De todos modos, aunque parezca un hecho aislado respecto a nuestro trabajo, esta automatización se podrá dar gracias a que el dron del que se dispone toma fotografías en calidad 4K [59], una resolución unas cuatro veces más potente que el *Full HD 1080p* de los televisores de unos pocos años atrás incluso actuales.

- Optimización del canal de transmisión de datos:

  Cuando se utiliza la cámara, en algunas ocasiones, observamos que la cámara pierde la resolución adecuada en pantalla e incluso no se puede ver el contenido, con errores como:

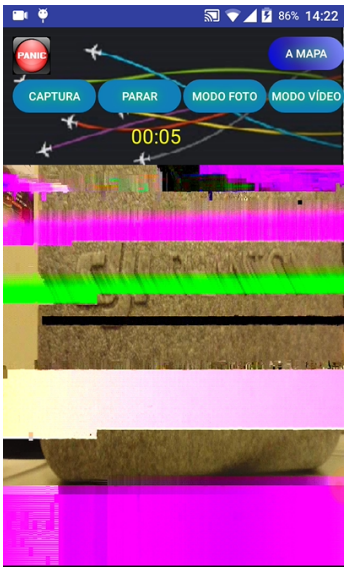



Figura 62: Error de visualización de la cámara

Con las sucesivas pruebas que se realizó sobre la aplicación, se pudo comprobar que esto no afecta al vídeo final, el cual se ve perfectamente a pesar de que en pantalla no se vea.
Además, la misión es autónoma una vez lanzada, por lo que no se requiere del servicio de la cámara durante la misma, más allá de un aspecto de ocio.

Este error puede ser solventado fácilmente incluyendo el método de la cámara en el hilo principal, no obstante, si cargamos demasiado el hilo principal , las funciones de control y tiempo de respuesta pueden verse ralentizadas, y como este no es un problema grave, se optó por dejar funcionar al creador de misiones de forma óptima, al precio de que la cámara pueda tener ciertos errores.

- Barra de estados:

  En muchas aplicaciones, será crítico incluir iconos que nos indiquen el estado de cómo están funcionando las cosas, como en esta sería , la batería y fuerza de la señal GPS.

  No obstante, de nuevo nos encontramos con métodos cerrados sin información, y tras la reflexión de comparar el tiempo de desarrollo contra la funcionalidad que nos iba a proporcionar, llegamos a la siguiente conclusión:

  El mismo dron, si la señal es débil, no nos deja despegar devolviéndonos errores internos, por lo que el icono de la señal GPS es redundante.
  Respecto al de la batería , el mando será capaz de avisarnos mediante pitidos cuando esta esté baja, y por si no es suficiente, como podemos ver en el Manual del *Phantom 4* [60], el mismo dron también nos avisa mediante su secuencia de luces:

| | |
|---|---|
| Parpadeo lento en rojo | Advertencia de batería baja |
| Parpadeo rápido en rojo | Advertencia de batería crítica |

Figura 63: Información del manual sobre batería baja

Figura 64: Ejemplo de secuencia de luces

- Interfaz de doble pantalla:

  Esta funcionalidad se observó en la aplicación de *IOS* (Desarrollo para *Apple*) , no obstante no se incluía en la de *Android*.

  No es un punto importante, pero siempre proporciona una sensación agradable observar interfaces de este tipo.

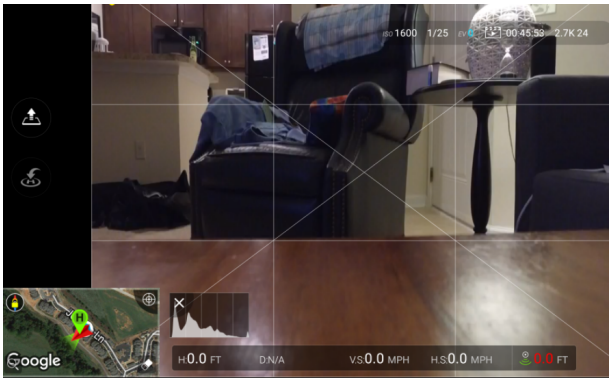



Figura 65: Interfaz de pantalla doble

Del mismo modo, *DJI* no proporciona información sobre estos temas, y aunque se buscó macros de código para poder partir de alguna base, no se encontró información satisfactoria por el momento.

Dado que es un hecho puramente visual, puesto que la doble interfaz con botón que tenemos realiza prácticamente la misma acción, no se profundizó más en el tema.

# 10 Conclusiones

En este punto se alcanza el final del escrito, y debido a que ya se han comentado las ventajas que proporciona la aplicación, y también los puntos a reforzar, se van a tratar las impresiones generales del trabajo de forma breve, para que se puedan retener las ideas esenciales en mente.

Como se ha presentado durante todo el trabajo, se trata de una aplicación industrial, por lo que tenemos análisis baratos y rápidos, además de seguros, aspectos esenciales para no violar la normativa vigente ni incurrir en riesgos innecesarios, además de estar encabezando el mercado por sus precios muy competitivos, en contraposición a los otros métodos actuales.

Se trata de una aplicación de uso sencillo y con una guía inicial, apta para una gran cantidad de dispositivos, no necesariamente de gama alta, con grandes funcionalidades al alcance de la mano y vistas a un futuro desarrollo muy prometedor.

Respecto a la pericia necesaria en el ámbito de los programadores, se tiene la necesidad de conocer una cantidad considerable de contenido de programación, no obstante, es un compromiso entre el conocimiento y funcionalidades, ya que este lenguaje, por el hecho de ser de alto nivel, recompensa gratamente al programador cuando este comprende la dinámica del mismo.

Por último, puesto que el sector eólico crece constantemente, cada vez se van a necesitar más análisis y revisiones, y evidentemente para ser competitivos y evolucionar, se debe estar en la "cresta de la ola".
La mejor opción para conseguirlo, es aprovecharse de un sector puntero en constante desarrollo, y barato, como es el sector de la programación y automatización de procesos con drones.

# 11 Documentación

# Referencias


[1] Información sobre la obtención de clave de google maps : *https://developers.google.com/maps/android/*

[2] Información sobre las tarifas según requerimiento industrial:
*https://developers.google.com/maps/pricing-and-plans/*

[3] Información sobre la obtención de la clave para desarrollar la aplicación:
*https://developer.dji.com/mobile-sdk/documentation/quick-start/index.html*

[4] Información sobre la publicación de la aplicación:
*https://support.google.com/googleplay/android-developer/answer/113469?hl=es*

[5] Linear Layout:
*https://developer.android.com/reference/android/widget/LinearLayout.html*

[6] Relative Layout:
*https://developer.android.com/reference/android/widget/RelativeLayout.html*

[7] Frame Layout:
*https://developer.android.com/reference/android/widget/FrameLayout.html*

[8] Intent:
*https://developer.android.com/reference/android/content/Intent.html*

[9] Generador de Imágenes de Icono:
*https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html*

[10] Información del archivo Manifest:
*https://developer.android.com/guide/topics/manifest/manifest-intro.html*

[11] Información de los botones y método onClick:
*https://developer.android.com/reference/android/widget/Button.html*

[12] Información sobre un botón de dos estados:
*https://developer.android.com/reference/android/widget/CompoundButton.OnCheckedChangeListener.html*

[13] Método click sobre el mapa:
*https://developers.google.com/android/reference/com/google/android/gms/maps/GoogleMap.OnMapClickListener*

[14] Información sobre el Layout Inflater:
*https://developer.android.com/reference/android/view/LayoutInflater.html*

[15] Macro de DJI para misiones de waypoints:
*https://developer.dji.com/mobile-sdk/documentation/android-tutorials/GSDemo-Google-Map.html*

[16] Macro de DJI para la cámara:
*https://developer.dji.com/mobile-sdk/documentation/android-tutorials/index.html*

[17] Información sobre aplicación multilenguaje:
*https://developer.android.com/training/basics/supporting-devices/languages.html*

[18] Información de seguridad aérea, AESA:
*http://www.seguridadaerea.gob.es/l*

[19] Marco regulatorio de drones, AESA:
http://www.seguridadaerea.gob.es/lang_castellano/cias_empresas/trabajos/rpas/marco/resolucion_directora/default.aspx

[20] Artículos 50 y 51, marco regulatorio de drones:
http://www.seguridadaerea.gob.es/media/4389070/ley_18_2014_de_15_octubre.pdf

[21] Instalación y utilización del simulador de DJI:
*https://developer.dji.com/mobile-sdk/documentation/application-development-workflow/workflow-testing.html*

[22] Página Google Maps:
*https://www.google.es/maps*

[23] Productos soportados por el simulador:
*https://developer.dji.com/mobile-sdk/documentation/application-development-workflow/workflow-testing.html#dji-assistant-2-simulator*

[24] Dron utilizado, DJI Phantom 4:
*http://www.dji.com/es/phantom-4?site=developer*

[25] Especificaciones DJI Phantom 4:
*http://www.dji.com/phantom-4/info#specs*

[26] Requisitos de desarrollo Android:
*https://developer.dji.com/mobile-sdk/documentation/application-development-workflow/workflow-prerequisits.html*

[27] Artículo funcionamiento GPS:
*https://dialnet.unirioja.es/descarga/articulo/5381247.pdf*

[28] Motor 2312/960KV CCW:
*http://www.multicopterwarehouse.com/dji-p3-motor-ccw*

[29] Motor 2312/980KV CCW:
*https://hobbyking.com/en_us/multistar-elite-2312-980kv-motor-set-cw-ccw-ezo-bearings-4mm-main-shaft-n45sh-magnets-2-motors.html*

[30] Artículo sobre el entorno dinámico del dron y su resolución:
*http://repository.upenn.edu/cgi/viewcontent.cgi?article=1705&context=edissertations*

[31] Aerogeneradores de gran tamaño:
*http://www.aryse.org/el-aerogenerador-con-las-palas-mas-largas-del-mundo/*

[32] Problema de erosión de pala, periódico Las Provincias:
*http://www.lasprovincias.es/economia/201611/03/expertos-buscan-valencia-soluciones-20161103110030.html*

[33] Artículo sobre problemas de erosión en borde de pala:
*http://www.windsystemsmag.com/media//pdfs/Articles/2012_October/1012_BladeFeature.pdf*

[34] Funcionalidades básicas de Android Studio:
*https://developer.android.com/studio/intro/index.html?hl=es-419*

[35] Página de apoyo para programadores GitHub:
*https://github.com*

[36] Página de apoyo para programadores Stack Overflow:
*https://es.stackoverflow.com/*

[37] Datos genéricos del lenguaje JAVA:
*https://es.wikipedia.org/wiki/Java_(lenguaje_de_programación)*

[38] Datos genéricos del lenguaje JAVA:
*https://es.wikipedia.org/wiki/Manejador_de_dispositivo*

[39] Información sobre qué es lenguaje multiplataforma:
*https://es.wikipedia.org/wiki/Multiplataforma*

[40] Información sobre las ventajas del Gradle:
*https://androidstudiofaqs.com/conceptos/que-es-gradle-en-android-studio*

[41] Información general sobre Android:
*https://es.m.wikipedia.org/wiki/Android*

[42] Información sobre los content:
*https://developer.android.com/guide/topics/providers/content-providers.html?hl=es-419*

[43] Información sobre los Broadcast Receiver:
*https://developer.android.com/reference/android/content/BroadcastReceiver.html*

[44] Concepto de mínimo privilegio:
*https://es.wikipedia.org/wiki/Principio_de_mínimo_privilegio*

[45] Conceptos genéricos del SDK:
*https://en.wikipedia.org/wiki/Software_development_kit*

[46] Ciclo de vida de las actividades:
*http://www.androidcurso.com/index.php/tutoriales-android/37-unidad-6-multimedia-y-ciclo-de-vida/158-ciclo-de-vida-de-una-actividad*

[47] Artículo y guía para la realización del estado del arte:
*http://www.colombiaaprende.edu.co/html/investigadores/1609/articles-322806_recurso_1.pdf*

[48] Hewitt-Sperry Automatic Airplane:
*https://en.wikipedia.org/wiki/Hewitt-Sperry_Automatic_Airplane*

[49] Lockheed U-2:
*https://es.wikipedia.org/wiki/Lockheed_U-2*

[50] Curtiss Wright VZ-7:
*https://en.wikipedia.org/wiki/Curtiss-Wright_VZ-7*

[51] Empresa DJI:
*https://en.wikipedia.org/wiki/DJI_(company)*

[52] Historia de los drones:
*http://drones.uv.es/origen-y-desarrollo-de-los-drones/*

[53] Estadísticas de ingresos de la empresa DJI:
*https://es.statista.com/estadisticas/669636/facturacion-de-dji-en-el-mundo/*

[54] Demostración de la ley de Betz:
*http://rabfis15.uco.es/lvct/tutorial/41/manual/manual3.htm*

[55] Historia de la energía eólica:
*http://www.ekidom.com/historia-de-la-energia-eolica*

[56] Historia de la energía eólica:
*https://es.wikipedia.org/wiki/Energía_eólica*

[57] Datos energía eólica, España:
*https://www.aeeolica.org/es/sobre-la-eolica/la-eolica-en-espana/*

[58] Datos estadísticos de los smartphones:
*http://www.amic.media/media/files/file_352_1050.pdf*

[59] Resolución 4K:
*https://es.wikipedia.org/wiki/Resolución_4K*

[60] Manual del *Phantom 4*:
*https://dl.djicdn.com/downloads/phantom_4/es/Phantom_4_User_Manual_v1.2_es_20160624.pdf*