UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería de Software

Trabajo de Fin de Grado

Control y monitorización de drones mediante análisis del contexto

Jacobo Conejo Lima

Junio, 2016

UNIVERSIDAD DE EXTREMADURA
Escuela Politécnica
Grado en Ingeniería Informática en Ingeniería de Software

Trabajo de Fin de Grado

Control y monitorización de drones mediante análisis del contexto

Autor: Jacobo Conejo Lima
Fdo:

Tutor: Juan María Hernández Núñez
Fdo:

Tribunal Calificador

Presidente: Adolfo José Lozano Tello
Fdo:

Secretario: Fernando Sánchez Figueroa
Fdo:

Vocal: Juan Carlos Preciado Rodríguez
Fdo:

CALIFICACIÓN:
FECHA:

Resumen

La mayoría de los drones no incorporan por defecto ningún sistema para evitar daños (tanto del propio dron como de otras personas u objetos) ni, tampoco, para garantizar el cumplimiento de la normativa vigente. La falta de este tipo de sistemas hace que no pueda asegurarse la ausencia de incidentes en la realización de tareas peligrosas o que de otra forma supondrían un mayor coste, como pueden ser el apoyo en misiones de salvamento y extinción de incendios o la toma de imágenes aéreas. Un sistema de monitorización del entorno aumentaría la seguridad en el uso de estos dispositivos. Para cumplir los objetivos mencionados, el sistema propuesto en este Trabajo de Fin de Grado permite definir una serie de reglas organizadas por niveles a partir de datos tomados del entorno.

Abstract

Most of drones do not have any system to prevent damages to themselves, people or objects; or even to ensure compliance with current legislation. Lack of such systems makes imposible to ensure the absence of incidents when carrying out dangerous or expensive tasks (e.g., supporting rescue and fire extinction missions or taking aerial images). An environment monitoring system would increase security when using these devices. In order to achieve the aforesaid goals, the system proposed in this Final Degree Project allows the definition of a set of rules ordered by level using data obtained from the environment.

Índice general

Lista de figuras **11**

Lista de tablas **15**

1 Introducción **17**

1.1 Objetivos 18

1.2 Presentación de capítulos 19

2 Motivación **21**

2.1 Grupo de investigación 21

2.2 Presencia de los drones en la sociedad 22

3 Estado del arte **24**

3.1 Drones 24

3.1.1 ¿Qué es un dron? . . . 25
3.1.2 Tipos de drones . . . 25
3.1.3 Partes de un dron . . . 28
3.1.4 Modos de vuelo . . . 34
3.2 Dispositivos utilizados . . . 36
3.2.1 Ordenador de a bordo . . . 36
3.2.2 Placa de conexión de dispositivos de entrada/salida . . . 39
3.2.3 Sensores y actuadores . . . 41
3.3 Procesamiento de eventos complejos . . . 42
3.3.1 Drools Fusion . . . 43
3.3.2 Esper . . . 43
3.4 Enterprise Service Bus . . . 44
3.4.1 Mule ESB . . . 44
3.5 Protocolos de comunicación . . . 45
3.5.1 MAVLink . . . 46
3.5.2 MQTT . . . 47
3.6 Trabajos relacionados . . . 49

3.7 Conclusiones 51

4 Propuesta de solución **53**

4.1 Visión general del sistema 53

4.2 Arquitectura Hardware 55

4.2.1 Dron 55

4.2.2 Raspberry Pi 57

4.2.3 Arduino 59

4.2.4 Sensores 61

4.2.5 Actuadores 67

4.2.6 Batería 70

4.3 Arquitectura Software 71

4.3.1 Sketch Arduino 71

4.3.2 Mule ESB 73

4.4 Reglas definidas 83

4.5 Diagramas de casos de uso y de secuencia 84

4.5.1 Diagrama de casos de uso 84

4.5.2 Diagramas de secuencia 85

5 Validación tecnológica 92

5.1 Prueba de ejecución de una sola acción 93

5.1.1 Explicación y objetivos 93

5.1.2 Recursos utilizados 94

5.1.3 Ejecución de la prueba 94

5.1.4 Resultados obtenidos 95

5.2 Prueba de ejecución de dos acciones de distinto nivel 95

5.2.1 Explicación y objetivos 95

5.2.2 Recursos utilizados 95

5.2.3 Ejecución de la prueba 96

5.2.4 Resultados obtenidos 97

5.3 Comprobación de que tras ejecutar una regla de un nivel, no se ejecutan las de niveles inferiores 97

5.3.1 Explicación y objetivos 97

5.3.2 Recursos utilizados 98

5.3.3 Ejecución de la prueba 98

5.3.4 Resultados obtenidos 99

5.4 Ejecución de acciones del dron en el simulador SITL 99

5.4.1 Explicación y objetivos 99

5.4.2 Recursos utilizados 100

5.4.3 Ejecución de la prueba 100

5.4.4 Recursos utilizados 101

5.4.5 Resultados obtenidos 102

5.5 Prueba en vuelo en un dron real 102

5.5.1 Explicación y objetivos 102

5.5.2 Recursos utilizados 103

5.5.3 Ejecución de la prueba 103

5.5.4 Resultados obtenidos 104

5.6 Prueba en tierra en un dron real 105

5.6.1 Explicación y objetivos 105

5.6.2 Recursos utilizados 105

5.6.3 Ejecución de la prueba 106

5.6.4 Resultados obtenidos 107

6 Trabajos futuros **108**

6.1 Niveles de reglas 108

6.2 DSL que genere el sistema adaptado a distintos tipos de drones, sensores y actuadores 109

6.3 Servicio web para añadir nuevas reglas en tiempo de ejecución 110

6.4 Montar un sistema de reglas de nivel superior, para coordinar varios drones 110

7 Conclusiones **112**

8 Agradecimientos **114**

A Instalación y configuración. **116**

A.1 Instalación de sensores y actuadores. 116

A.1.1 Carga del sketch en el Arduino. 118

A.2 Instalación del sistema a partir de la imagen. 119

A.3 Instalación del sistema paso a paso. 121

A.3.1 Raspbian. 122

A.3.2 Mule ESB. 122

A.3.3 Mosquitto. 128

A.3.4 DroneKit-Python. 129

A.3.5 Aplicación Mule. 129

A.4 Configuración del sistema. 130

B Uso del sistema. 132

B.1 Conexión de dron y Arduino. 132

B.2 Carga de patrones y acciones. 133

B.3 Carga de ruta en el dron. 134

B.4 Puesta en funcionamiento. 135

B.5 Funcionamiento con el simulador. 136

C Índice de acrónimos 139

Referencias 142

Índice de figuras

3.1 Dron militar MQ9-Reaper cargado con misiles. Fuente https://commons.wikimedia.org . . . 25

3.2 DJI Phantom 4. Cuadricóptero de la empresa DJI. Fuente http://www.digitaltrends.com/ . . . 27

3.3 Frame de un cuadricóptero. Fuente http://www.ebay.com/ . . . 29

3.4 Hélice montada en un motor. Fuente http://www.surehobby.com/ . . . 29

3.5 ESC de la marca Simonk. Fuente http://droneschinos.net/ . . . 30

3.6 Batería de 5000 mAh. Fuente http://ardupilot.org/ . . . 30

3.7 Piloto automático Pixhawk. Fuente http://pixhawk.org/ . . . 31

3.8 Módulo GPS de la marca 3D Robotics. Fuente http://ardupilot.org/ 32

3.9 Emisor y receptor R/C de la marca RadioLink. Fuente http://webtoy.net/ 32

3.10 Módulo emisor y módulo receptor de telemetría. Fuente http://ardupilot.org/ . . . 33

3.11 Creación de una misión en Mission Planner. Fuente http://ardupilot.org/ . . . 34

3.12 Ejes de giro de un dron multirrotor. Fuente http://quadcopterproject.wordpress.com/ . . . 35

3.13 ODROID-XU4. Fuente http://www.hardkernel.com/ . . . 37

3.14 Raspberry Pi 3 Model B. Fuente http://www.wired.co.uk . . . 38

3.15 BeagleBoard-X15. Fuente https://beagleboard.org/ . . . 39

3.16 Arduino Uno Rev.3. Fuente http://es.wikipedia.org/ . . . 41

3.17 Esquema del sistema propuesto . . . 52

4.1 Sistema propuesto. Punto de vista físico (izq.) y punto de vista lógico (der.) 53

4.2 Arduino y Raspberry Pi junto con los sensores y actuadores utilizados . . 55

4.3 Controlador de vuelo APM 2.6. Fuente: http://dronecasero.blogspot.com.es/ . . . 56

4.4 Dron utilizado para el trabajo . . . 57

4.5 Raspberry Pi 2 Model B. Fuente: http://wordnice.net/ . . . 59

4.6 Funduino Uno R3. Fuente: http://wordnice.net/ . . . 59

4.7 Grove Shield. Fuente: http://www.seeedstudio.com/ . . . 61

4.8 Sensor de temperatura y humedad. Fuente: http://www.seeedstudio.com/ 62

4.9 Barómetro. Fuente: http://www.seeedstudio.com/ 63

4.10 Sensor de agua. Fuente: http://www.seeedstudio.com/ 63

4.11 Sensor de radiación ultravioleta. Fuente: http://www.seeedstudio.com/ 64

4.12 Sensor de CO. Fuente: http://www.seeedstudio.com/ 65

4.13 Sensor de HCHO. Fuente: http://www.seeedstudio.com/ 65

4.14 Sensor de ultrasonidos. Fuente: http://www.seeedstudio.com/ 66

4.15 Sensor de intensidad luminosa. Fuente: http://www.seeedstudio.com/ . 67

4.16 Circuito que controla la tira de LEDs. 68

4.17 Cámara Pi NoIR. Fuente http://www.gravitech.us/ 69

4.18 Batería USB utilizada para proporcionar energía a la Raspberry Pi. . . . 71

4.19 Flujos que forman la aplicación Mule (I). 74

4.20 Flujos que forman la aplicación Mule (II). 76

4.21 Casos de uso del sistema. 85

4.22 Recogida de datos de sensores. 86

4.23 Atención de un mensaje de acción. 87

4.24 Actualización de posición GPS, altitud y nivel de batería. 88

4.25 Atención de un mensaje con datos de un sensor. 88

4.26 Adición de un nuevo patrón de eventos. 89

4.27 Adición de una acción en respuesta a un patrón. 90

4.28 Detección de un patrón de eventos. 91

5.1 Captura de la prueba en ejecución. 93

5.2 Dron moviéndose al punto <39.479034,-6.342739> en APM Planner. . . . 101

5.3 Dron junto con el sistema de monitorización del entorno y control. 104

A.1 Arduino Uno con los sensores y actuadores conectados a la Grove Shield. 117

A.2 Pi NoIR conectada a la Raspberry Pi. Fuente https://learn.adafruit.com/ 118

A.3 Pantalla mostrada al ejecutar raspi-config. Fuente https://www.raspberrypi.org/ 121

B.1 Orden de comprobación de puertos USB en Raspberry Pi 2 Model B. . . 133

B.2 Parte de la interfaz de Mission Planner que permite la conexión al dron. Fuente http://ardupilot.org/ 134

B.3 Pestaña 'Flight Plan' de Mission Planner. Fuente http://ardupilot.org/ 135

Índice de tablas

3.1 Características de los 3 modelos más modernos de ODROID. Fuente https://en.wikipedia.org/wiki/ODROID#Specifications 37

3.2 Características de 3 modelos de Raspberry Pi. Fuente https://en.wikipedia.org/wiki/Raspberry_Pi#Specifications 38

3.3 Características de 3 versiones de BeagleBoard. Fuente https://en.wikipedia.org/wiki/BeagleBoard#Specifications 39

4.1 Especificaciones de Raspberry Pi 2 Model B. Fuente https://es.wikipedia.org/wiki/Raspberry_Pi 58

4.2 Especificaciones de Funduino Uno R3. Fuente http://www.nexuscyber.com/funduino-uno-r3-atmega328 60

4.3 Especificaciones de Grove Base Shield V1.3. Fuente http://www.seeedstudio.com/wiki/Grove_-_Base_Shield 61

4.4 Especificaciones del sensor de temperatura y humedad. Fuente http://www.seeedstudio.com/wiki/Grove_-_Temperature_and_Humidity_Sensor_Pro 62

4.5 Especificaciones del barómetro. Fuente http://www.seeedstudio.com/wiki/Grove_-_Barometer_Sensor 63

4.6 Especificaciones del sensor de agua. Fuente http://www.seeedstudio.com/wiki/Grove_-_Water_Sensor 64

4.7 Especificaciones del sensor de radiación ultravioleta. Fuente http://www.seeedstudio.com/wiki/Grove_-_UV_Sensor 64

4.8 Especificaciones del sensor de CO. Fuente http://www.seeedstudio.com/wiki/Grove_-_Gas_Sensor(MQ9) 65

4.9 Especificaciones del sensor de HCHO. Fuente http://www.seeedstudio.com/wiki/Grove_-_HCHO_Sensor 66

4.10 Especificaciones del sensor de ultrasonidos. Fuente http://www.seeedstudio.com/wiki/Grove_-_Ultrasonic_Ranger 66

4.11 Especificaciones del sensor de intensidad luminosa. Fuente http://www.seeedstudio.com/wiki/Grove_-_Digital_Light_Sensor 67

4.12 Especificaciones de la tira de LEDs. Fuente https://www.amazon.es/PsmGoods-coches-prueba-flexible-l%C3%A1mpara/dp/B00Y7J3WLO . . 68

4.13 Especificaciones del relé. Fuente https://www.ghielectronics.com/downloads/man/20084141716341001RelayX1.pdf 69

4.14 Especificaciones de la cámara Pi NoIR. Fuente https://www.pollin.de/shop/downloads/D701946D.PDF 70

C.1 Índice de acrónimos 141

Capítulo 1

Introducción

Los drones se han convertido en una tecnología muy presente en la sociedad en los últimos años. Es muy habitual ver **vídeos grabados con estos aparatos**, ya sea de competiciones deportivas, festivales de música, e, incluso, de noticias de actualidad, como el reciente incendio de un vertedero ilegal de neumáticos en Seseña [1]. Por otra parte, también es posible encontrar muchas noticias acerca de los planes de muchas compañías de mensajería para **repartir paquetes haciendo uso de drones** [2] [3] [4].

Sin embargo, a pesar de todos los posibles usos y el gran peso que pueden tener en la economía por los trabajos asociados a estos usos, existen **muy pocos sistemas que garanticen que la seguridad de los drones y el cumplimiento de la legislación vigente**. Algunas compañías han desarrollado sistemas para evitar posibles los obstáculos que puedan presentarse, pero nada más [5].

En este TFG se cubrirán otros escenarios que puedan comprometer la integridad de los drones (como la lluvia), además del cumplimiento de la normativa vigente. En concreto, se desarrollará un **sistema de reglas** en el que tendremos 3 niveles: el nivel más importante contendrá las reglas que se encargan de que el **dron cumpla con la**

normativa vigente, el en siguiente, irán las que se encargaran de asegurar la **auto-preservación** del mismo y, en el último, las que se refieran a **tareas periódicas o puntuales** que el dron deba llevar a cabo. Para conseguir esto, se hará uso de, además del propio dron, un ordenador de a bordo que analizará los datos de un conjunto de sensores que medirán las condiciones del entorno.

1.1 Objetivos

El objetivo general de este TFG es conseguir un sistema que, a partir de las condiciones del entorno en el que se encuentre el dron y, según una serie de reglas que se definan, pueda controlar al dron para asegurar el cumplimiento de la normativa vigente y la integridad del mismo, además de permitir el control de una serie de actuadores para realizar ciertas tareas como, por ejemplo, tomar fotos. Para llegar a conseguir este objetivo, se dividirá en otros más pequeños que sea posible ir cumpliendo poco a poco:

1. Conocer qué es un dron, los tipos existentes y cuál es su funcionamiento.
2. Elegir el dron más adecuado para el trabajo.
3. Identificar los protocolos de comunicación entre el dron y una CGS (*Control Ground Station*), de forma que sea posible utilizarlos para comunicar el dron con el ordenador de a bordo.
4. Elegir un conjunto de sensores para monitorizar el entorno y de actuadores.
5. Estudiar cómo enviar la información producida por los sensores al ordenador de a bordo.
6. Estudiar los motores de procesamiento de eventos complejos existentes y elegir el que más se adecúe al problema.

7. Convertir los datos de los sensores en eventos que puedan ser procesados por el motor CEP (*Complex Event Processing*).

8. Estudiar la forma de crear un conjunto de reglas para definir los patrones de eventos que nos interesan y la respuesta a los mismos.

9. Integrar el procesamiento de eventos, la definición de reglas y la comunicación con el dron y los actuadores en un único sistema.

10. Definir y llevar a cabo una serie de pruebas para corroborar que el sistema se comporta como se espera de él.

11. Explicar los posibles trabajos futuros que pueden desarrollarse partiendo de este sistema.

1.2 Presentación de capítulos

Para cerrar este capítulo, se expondrá qué es lo que puede encontrarse en cada uno de los siguientes capítulos de este TFG:

En el **Capítulo 2** se explicará qué nos ha animado a comenzar este proyecto y cómo se enmarca dentro de las líneas de estudio del Grupo de Investigación.

El **Capítulo 3** se centrará en examinar el entorno en que se enmarca el proyecto, así como las tecnologías que es posible utilizar y los proyectos que se están desarrollando en este campo. A partir de esto, se elegirán las tecnologías y dispositivos que se consideren más apropiados para el sistema.

En el **Capítulo 4** se presentará el sistema desarrollado, detallando su arquitectura hardware y software y las reglas que han sido definidas.

En el **Capítulo 5** se llevarán a cabo las pruebas necesarias para verificar que el sistema puede utilizarse y que proporciona los resultados esperados.

En el **Capítulo 6** se propondrán varios trabajos que pueden llevarse a cabo a partir del trabajo desarrollado en este TFG.

En el **Capítulo 7** se expondrán las conclusiones obtenidas del desarrollo del proyecto.

Además de esto, se han incluido un par de anexos en los que se explica cómo instalar y configurar el sistema (**Apéndice A**) y cómo usarlo (**Apéndice B**).

Capítulo 2

Motivación

En este capítulo se intentará explicar **qué es lo que ha motivado el surgimiento de este proyecto**. Para ello, se comentarán de las principales líneas de investigación existentes en el grupo de investigación, así como de la creciente presencia de los drones en todo tipo de campos en la sociedad actual.

2.1 Grupo de investigación

Este trabajo se desarrolla dentro del **grupo de investigación Quercus** [6], el cual desarrolla varias líneas de investigación: Arquitecturas Orientadas a Servicios, Desarrollo de Software Dirigido por Modelos, Desarrollo de Software Orientado a Aspectos, Ingeniería Web, Accesibilidad Web, Domótica, Visión Artificial...

En el último de estos campos, el de la **Visión Artificial**, se ha estado desarrollando software para detección de personas y de objetos, que permite determinar, por ejemplo, si un objeto está obstruyendo una salida, el número de personas presentes en una sala, el comportamiento de las personas, es decir, por dónde se mueven más

frecuentemente...

Aquí es donde pueden entrar los drones, ya que, hasta ahora, se han estado utilizando **cámaras situadas en un punto fijo**, sin embargo, si fuese posible **montar dichas cámaras en un dron** (o utilizar las que incluyen algunos drones) y procesar las imágenes obtenidas, ya sea en el propio dron o en otro equipo más potente, se abriría un gran abanico de posibilidades.

2.2 Presencia de los drones en la sociedad

En el capítulo anterior se han mencionado sólo dos de los muchos usos que es posible darle a este tipo de aeronaves, que pueden encontrarse en sectores muy diversos, como ingeniería civil e industrial [7], el sector energético y medioambiental [8], el control de explotaciones agropecuarias y forestales [9], la producción audiovisual [10] o el apoyo en misiones de salvamento terrestre y marítimo [11]. El uso de estas aeronaves de control remoto en estos sectores no sólo permite o facilita la realización de determinadas tareas sino que además, suele abaratar considerablemente los costes.

Estos aparatos también se estima que van a tener un importante peso en la economía en los años venideros. Según estudios de la AUVSI [12] (*Association for Unmanned Vehicle Systems International*), para el año 2025 habrá unos **100.000 nuevos empleos relacionados con la industria de los drones**. Otro estudio, esta vez del Comité para la Unión Europea de la Cámara de los Lores británica, estima que en el año 2050 podrían crearse unos **150.000 puestos de trabajo relacionados con la industria de los drones sólo en Europa**. En general, todos los estudios e informes actuales apuntan a que los drones, también conocidos como UAVs (*Unmanned Aerial Vehicle*) o RPAs (*Remotely Piloted Aircraft*), serán uno de los pilares de la economía mundial en un futuro próximo.

La legislación referente al uso de drones de peso inferior a los 150kg al despegue para la realización de trabajos técnicos y científicos se encuentra recogida en la **Ley 18/2014, de 15 de octubre**, en su sección 6ª [13]. Se trata de un reglamento temporal que se espera que sea modificado próximamente [14].

Algunas de las consideraciones más importantes de esta legislación son las siguientes:

- Tener **siempre a la vista** al dron y **no superar los 400 pies** (120m) de altura.
- El dron debe **incorporar una placa** en la que se pueda leer la identificación de la aeronave, el nombre de la empresa operadora y los datos de contacto.
- Disponer del **carné de piloto** de drones.
- Está **prohibido volar de noche**.
- Está **prohibido volar en espacio aéreo controlado o sobre aglomeraciones** de personas sin una autorización.
- Está **prohibido volar a menos de 8km de aeropuertos o aeródromos**.

El incumplimiento de estas normas puede suponer importantes sanciones económicas. Aún así, pueden encontrarse **numerosas noticias sobre incidentes** debidos al incumplimiento de la normativa, tanto en España [15], como en otros países [16] [17].

La **motivación del presente trabajo** es conseguir un sistema que pueda asegurar que las tareas realizadas por el dron (ya sean de procesado de imágenes o de cualquier otro tipo) se realizan de forma que **se cumple con la normativa vigente** y **se asegura la integridad del dron**.

Capítulo 3

Estado del arte

En este capítulo se intentarán explicar todas las tecnologías (tanto hardware como software) relacionadas con el presente trabajo. Además, también se investigará qué otros trabajos se están haciendo en este campo, para buscar ideas que puedan ser de utilidad para este proyecto.

3.1 Drones

En primer lugar, se comenzará **definiendo el concepto de dron**, los **tipos que existen**, las **partes por las que está formado** y los **modos disponibles** para manejarlo. Esto permitirá que el resto de la documentación pueda entenderse más fácilmente.

3.1.1 ¿Qué es un dron?

Un **dron es** *una aeronave que vuela sin tripulación, reutilizable, capaz de mantener de manera autónoma un nivel de vuelo controlado y sostenido, y propulsado por uno o varios motores* [18]. Es precisamente el hecho de que **puedan operar de forma autónoma** lo que los distingue de las Aeronaves Radiocontroladas.

Figura 3.1: Dron militar MQ9-Reaper cargado con misiles. Fuente https://commons.wikimedia.org

Los drones, también llamados **VANT** *vehículo aéreo no tripulado* o **UAV** *Unmanned Aerial Vehicle*, pueden ser tanto de uso civil, como de uso militar. En este último caso, son denominados **UCAV** *Unmanned Combat Aerial Vehicle* y, generalmente van armados.

3.1.2 Tipos de drones

Existen varias clasificaciones de los drones en función de distintos criterios. En este apartado se verán las tres principales clasificaciones, en función de su misión principal, el tipo de alas y el método de control.

En primer lugar, es posible clasificarlos según su misión principal en:

- **Blanco:** Simulan aviones o ataques enemigos en los sistemas de defensa de tierra o aire.
- **Reconocimiento:** Envían información militar sobre posibles objetivos o enemigos.
- **Combate (UCAV):** Proporcionan capacidad de ataque para misiones con alto nivel de riesgo.
- **Logística:** Drones específicamente diseñados para llevar carga y realizar misiones logísticas.
- **Investigación y desarrollo:** Utilizados para probar e investigar los sistemas que se integrarán en los drones existentes.
- **UAV comerciales y civiles:** Son aquellos diseñados específicamente para propósitos civiles y comerciales, como pueden ser: realizar filmaciones, tomar imágenes o apagar fuegos.

Otra clasificación, que en este caso atiende a la forma del dron, concretamente al tipo de alas que éste utiliza [19] es la siguiente:

- **Ala fija:** Se componen de un ala rígida que tiene una determinada superficie de sustentación, lo que hace posible el vuelo gracias a que se genera una fuerza de elevación por la velocidad de avance del UAV. Esta velocidad se genera por el empuje hacia delante que genera una hélice conectada a un motor eléctrico o de combustión interna.
- **Multirrotor:** Está formado por varias hélices que giran alrededor de un mástil, cada una de las cuales se conoce como rotor. Existen muchas configuraciones: un solo rotor (helicóptero), 3 rotores (tricóptero), 4 rotores (cuadricóptero), 6 rotores (hexacóptero), 8 rotores (octocóptero), e incluso configuraciones con 12 y 16

rotores. De todos los anteriores, la configuración más conocida quizá sea la de 4 rotores, ya que son los que más suelen utilizarse. Su principio de funcionamiento es el mismo que el de los drones de ala fija, sin embargo, no es necesario un movimiento de avance continuo para producir un flujo de aire sobre las hélices, sino que son las propias hélices las que están en continuo movimiento para producir el flujo de aire necesario para generar la elevación.

Figura 3.2: DJI Phantom 4. Cuadricóptero de la empresa DJI. Fuente http://www.digitaltrends.com/

La última clasificación que va a presentarse es la que distingue a los drones según el método de control del mismo, lo que permite una mayor o menor autonomía del mismo [20]:

- **Autónomo:** Utiliza sus propios sistemas y sensores, que están integrados en el propio dron para guiarse, lo que le permite evitar posibles obstáculos que puedan presentarse en su camino. De esta forma, no es necesario que un piloto lo controle.
- **Monitorizado:** El dron dirige su propio plan de vuelo, aunque es necesario un técnico humano que le proporcione información. De esta forma, aunque el técnico no controla el dron directamente, sí puede decidir las acciones que llevará a cabo a continuación

- **Supervisado:** El dron realiza algunas tareas automáticamente, aunque es pilotado por un humano.
- **Preprogramado:** Antes de que el dron inicie el vuelo, se diseña la ruta que éste seguirá y se carga en el mismo. Así, el dron simplemente seguirá dicha ruta, sin posibilidad de cambiarla ante posibles obstáculos que se presenten en la misma.
- **Controlado remotamente (R/C):** Son controlados directamente por un piloto con una emisora.

3.1.3 Partes de un dron

Una vez se han definido las distintas clasificaciones que pueden hacerse de un dron, se explicarán qué partes lo componen y para qué sirve cada una. Dicha explicación estará centrada en los componentes de los drones multirrotor, ya que es el tipo de dron que va a utilizarse en este trabajo [21].

- **Frame, motores, hélices y ESCs** . El *frame* es el esqueleto principal de un dron, donde van instalados el resto de componentes; los hay de diferentes formas y tamaños, en función de la configuración que tenga el dron. Los motores son los que se encargan de proporcionar la velocidad de giro suficiente a las hélices, lo que permite el vuelo del dron. Por último, las ESCs *Electronic Speed Control*, entregan a cada uno de los motores la corriente necesaria en cada momento, es decir, se encargan de regular la energía que se envía de la batería a cada motor [22].

Figura 3.3: Frame de un cuadricóptero. Fuente http://www.ebay.com/

Figura 3.4: Hélice montada en un motor. Fuente http://www.surehobby.com/

Figura 3.5: ESC de la marca Simonk. Fuente http://droneschinos.net/

- **Fuente de energía**. En los drones pequeños (como en este caso), suele utilizarse una batería Li-Po como fuente de energía, mientras que los más grandes utilizan motores normales de aviones. Esto es lo que proporcionará la energía necesaria para mover los motores.

Figura 3.6: Batería de 5000 mAh. Fuente http://ardupilot.org/

- **Placa de control de vuelo o piloto automático**. Proporciona al dron la capacidad de vuelo autónomo y consiste en una placa que procesa los datos que obtiene

a través de los distintos sensores para mantener la estabilidad del dron y, opcionalmente, seguir una ruta programada. Uno de los sensores normalmente integrado en la placa es un IMU *Inertial Mesaurement Unit*, formado por una combinación de acelerómetros, giroscopios y magnetómetros, que permite calcular la velocidad y orientación del dron [23]. Otro sensor necesario es el GPS, que proporciona la información sobre la posición y altitud del dispositivo, aunque este sensor no suele ir incluido en la placa, sino que se utiliza un GPS externo se conecta a ella. Los pilotos automáticos más conocidos son *Ardupilot* [24] y *Pixhawk* [25]. El segundo, es una evolución del primero y, debido a su mayor capacidad de cómputo, es el que actualmente se recomienda utilizar.

Figura 3.7: Piloto automático Pixhawk. Fuente http://pixhawk.org/

- **GPS**. Como se explica en el punto anterior, el GPS proporciona la posición en el mundo y la velocidad del dron al piloto automático. Esto permite que el dron siga una ruta predefinida o se mantenga en un punto. Aunque existen modos de vuelo que no necesitan utilizar el GPS, en otros, como el modo AUTO es necesario.

Figura 3.8: Módulo GPS de la marca 3D Robotics. Fuente http://ardupilot.org/

- **Emisor y receptor R/C**. Permite enviar al dron la información para manejarlo de forma manual. El receptor va conectado al piloto automático y le comunica a éste las órdenes que recibe del emisor. Es necesario que el emisor tenga, al menos, 5 canales: 3 para cada uno de los ejes en los que puede moverse el dron, otro para el acelerador (estos 4 se cubren normalmente con 2 joysticks) y un último canal para cambiar el modo de vuelo.

Figura 3.9: Emisor y receptor R/C de la marca RadioLink. Fuente http://webtoy.net/

- **Telemetría**. Este componente permite al dron comunicarse con la CGS utilizando el protocolo MAVLink. Lo cual permite ver los datos sobre velocidad, posición, modo... en tiempo real, además de poder modificar la misión que está llevando a cabo el dron.

Figura 3.10: Módulo emisor y módulo receptor de telemetría. Fuente http://ardupilot.org/

- **CGS *Control Ground Station***. Permite calibrar y configurar el emisor y los sensores del dron, lo cual es necesario antes del primer vuelo. Por otra parte, facilita la creación de la misión que va a llevar a cabo el dron, es decir, la ruta que seguirá, y su carga en el mismo, para que pueda realizarla. Como se indica en el punto anterior, durante la misión, recibirá información sobre la velocidad, altitud, modo, nivel de batería... del dron y proporcionará cierto control sobre el mismo. Como CGS puede utilizarse tanto un ordenador como un *smartphone* con el software necesario para esto. Uno de los programas más conocidos que proporciona la funcionalidad de CGS a un ordenador es **Mission Planner** o **APM Planner**, su versión para Linux y MAC.

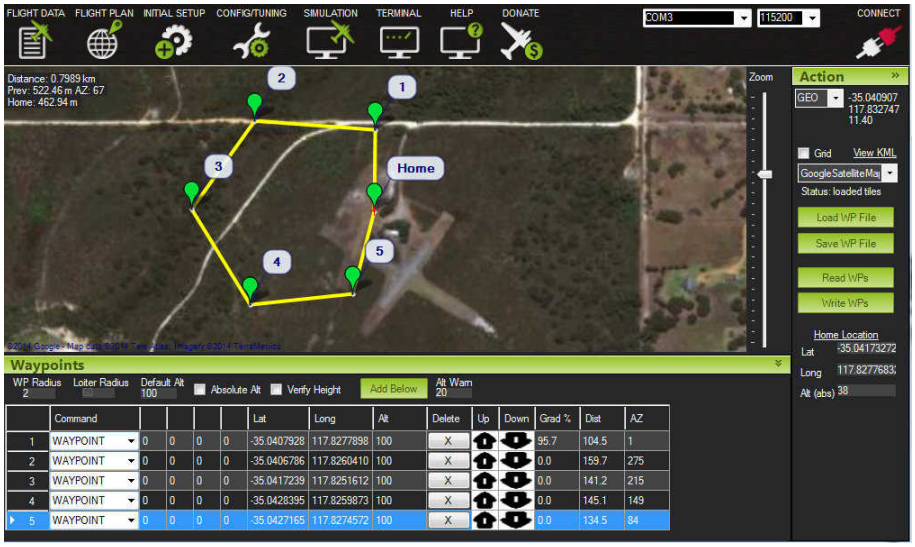

Figura 3.11: Creación de una misión en Mission Planner. Fuente http://ardupilot.org/

3.1.4 Modos de vuelo

Los pilotos automáticos basados en Ardupilot proporcionan **14 modos de vuelo** para los drones multirrotor. De ellos, en este apartado se explicarán los principales, clasificándolos en función de si necesitan GPS o no para funcionar [26].

- Modos de vuelo que **necesitan GPS**:
  - **Loiter**: En este modo, el dron intenta mantener la posición en la que se encuentra. El piloto puede manejar el dron como si estuviese en modo manual, pero cuando suelte los joysticks, el dron mantendrá su posición.
  - **RTL (Return to Launch)**: En el modo RTL, el dron volverá a la posición donde se estableció el punto *home* (normalmente el punto donde despegó). En primer lugar, ascenderá hasta una altura mínima que puede configurarse en Mission Planner y, a continuación, irá en línea recta al punto *home*. Una vez se encuentra sobre dicho punto, aterrizará sobre el mismo y parará los motores.

- **Auto**: Este modo es el utilizado para que el dron lleve a cabo una misión previamente programada. Para ello, es necesario crear una misión (compuesta por puntos por los que tiene que pasar y comandos de acción, como dar vueltas en un punto) en Mission Planner y cargarla en el dron. Con la misión cargada, solo habrá que poner el dron en este modo y comenzará a realizarla.
- **Guided**: El modo Guided permite controlar al dron directamente desde una CGS. En él, se enviarán los comandos que se indiquen en la aplicación de CGS (por ejemplo, Mission Planner) al dron a través de telemetría, lo que permite que manejar el dron interactivamente desde la aplicación enviándole los puntos a los que debe dirigirse en cada momento.

- Modos de vuelo que **no necesitan GPS**:
  - **Stabilize**: En este caso, el dron se maneja manualmente, pero el piloto automático se encarga de nivelar el cabeceo (*pitch*) y el alabeo (*roll*) del dron, para que no se desestabilice.

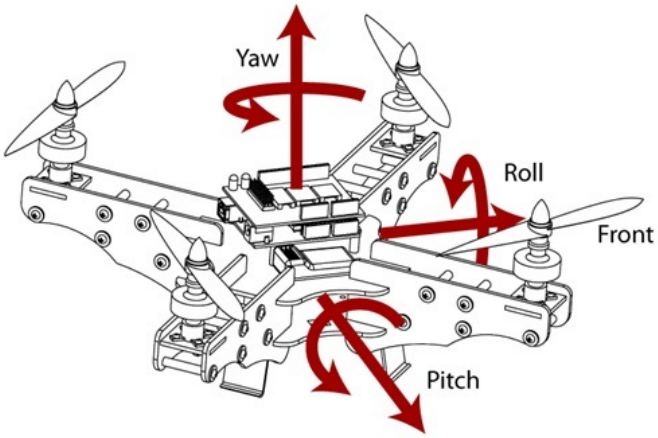

Figura 3.12: Ejes de giro de un dron multirrotor. Fuente http://quadcopterproject.wordpress.com/

  - **Alt Hold (Altitude Hold)**: Este modo es similar al anterior, pero la diferencia radica en que el dron mantiene una altura. Es decir, es posible manejar libremente el dron, pero este siempre está a una altura fija.

– **Land**: El modo Land hace que el dron aterrice sobre el punto en el que se encuentra. Una vez está en el suelo, parará los motores.

3.2 Dispositivos utilizados

En esta sección se hará una introducción de los dispositivos *hardware* de los que se ha hecho uso en este trabajo. Además del propio dron, se hará uso de **4 tipos de dispositivos: un ordenador de a bordo, una placa desarrollo hardware en la que conectar dispositivos de entrada y salida, un conjunto de sensores y un conjunto de actuadores**.

3.2.1 Ordenador de a bordo

El ordenador de a bordo es el componente que se encargará de obtener los datos de los sensores, procesarlos y enviar una orden en caso de que se detecte un patrón de evento relevante. Para llevar a cabo esta misión, es necesario un **dispositivo que tenga una capacidad de computación considerable**, es decir, que no sea simplemente un dispositivo al que es posible conectar sensores y actuadores y añadir cierta lógica (como sería una placa Arduino). Por tanto, es necesario encontrar un dispositivo que ofrezca unas posibilidades similares a las que puede ofrecer un ordenador personal, aunque su capacidad de cómputo será inferior, debido a que su tamaño tendrá que ser menor para que pueda ir montado en el dron y que su consumo de energía no puede ser muy elevado. Es aquí donde aparecen los *single-board computers*, donde pueden enmarcarse dispositivos como ODROID, Raspberry Pi o BeagleBoard [27].

3.2.1.1 ODROID

ODROID es una serie de *single-board computers* creados por **Hardkernel**, una compañía de hardware open-source con sede en Corea del Sur. Aunque el nombre ODROID es una composición de 'open' y 'Android', alguna partes del hardware no han sido liberadas por la compañía. Además, es posible instalar otros sistemas operativos distintos de Android, como distribuciones de Linux [28].

Figura 3.13: ODROID-XU4. Fuente http://www.hardkernel.com/

Modelo	CPU	RAM	Puertos USB
ODROID-XU4	Samsung Exynos5422 Cortex™-A15 2Ghz and Cortex™-A7 Octa core CPUs	2GB LPDDR3	2 x USB 3.0, 1 x USB 2.0
ODROID-C2	Amlogic ARM® Cortex®-A53(ARMv8) 2Ghz quad core CPUs	2GB DDR3 SDRAM	4 x USB 2.0, 1 x MicroUSB
ODROID-C1+	Amlogic ARM® Cortex®-A5(ARMv7) 1.5Ghz quad core CPUs	1GB DDR3 SDRAM	4 x USB 2.0, 1 x MicroUSB

Tabla 3.1: Características de los 3 modelos más modernos de ODROID. Fuente https://en.wikipedia.org/wiki/ODROID#Specifications

3.2.1.2 Raspberry Pi

Raspberry Pi es una serie de *single-board computers* del tamaño de una tarjeta de crédito desarrollados por **Raspberry Pi Foundation** con el interés de promover la enseñanza de la informática en los colegios y los países en desarrollo. Tanto la Raspberry Pi original como la Raspberry Pi 2 son fabricadas con distintas configuraciones de software. En este caso, el hardware no es open-source, sino que es propietario [29].

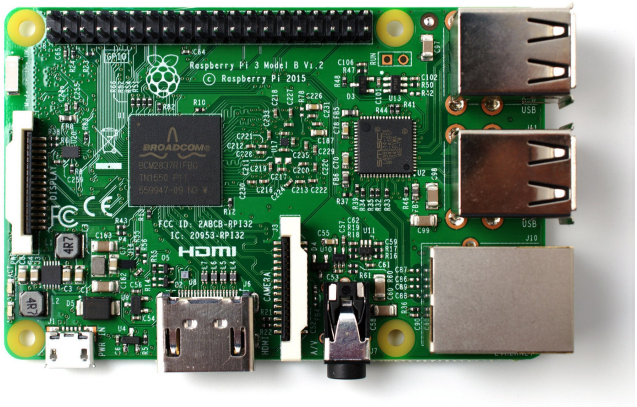

Figura 3.14: Raspberry Pi 3 Model B. Fuente http://www.wired.co.uk

Modelo	CPU	RAM	Puertos USB
Raspberry Pi 3 Model B	1.2 GHz 64-bit quad-core ARM Cortex-A53	1 GB	4 x USB 2.0
Raspberry Pi 2 Model B	900 MHz 32-bit quad-core ARM Cortex-A7	1 GB	4 x USB 2.0
Raspberry Pi 1 Model B	700 MHz single-core ARM1176JZF-S	512 MB	2 x USB 2.0

Tabla 3.2: Características de 3 modelos de Raspberry Pi. Fuente https://en.wikipedia.org/wiki/Raspberry_Pi#Specifications

3.2.1.3 BeagleBoard

BeagleBoard es un *single-board computer* de hardware libre de bajo consumo producido por Texas Instruments junto con Digi-Key y Newark element14. Fue diseñado

pensando en el desarrollo de software libre. La placa fue desarrollada por un pequeño equipo de ingenieros como una placa educacional que pudiese ser utilizada en colegios de todo el mundo, para mostrar las capacidades del *software* y el *hardware* open-source [30].

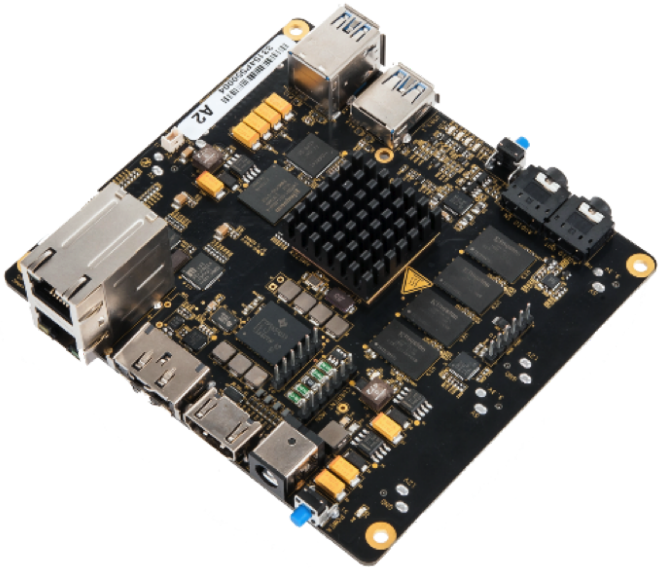

Figura 3.15: BeagleBoard-X15. Fuente https://beagleboard.org/

Modelo	**CPU**	**RAM**	**Puertos USB**
BeagleBoard-X15	1.5 GHz Dual ARM Cortex-A15 + Dual ARM M4	2 GB DDR3L	3 x USB 3.0, 4 x USB 2.0, 1 x MicroUSB
BeagleBone Black	1 GHz Cortex-A8 + 2xPRU	512 MB DDR3	1 x USB 2.0, 1 x MiniUSB
BeagleBone	720 MHz Cortex-A8 + 2xPRU	256 MB DDR2	1 x USB 2.0, 1 x MiniUSB

Tabla 3.3: Características de 3 versiones de BeagleBoard. Fuente https://en.wikipedia.org/wiki/BeagleBoard#Specifications

3.2.2 Placa de conexión de dispositivos de entrada/salida

Éste será el dispositivo al que se conectarán tanto los sensores como los actuadores que formarán el sistema. Para este cometido, una placa Arduino parece el dispositivo

perfecto, ya que tiene una gran comunidad detrás y existen una buena cantidad tanto de sensores como de actuadores pensados para ella.

3.2.2.1 Arduino

Una placa Arduino está formada por un microcontrolador, normalmente Atmel AVR, y puertos digitales y analógicos de entrada/salida, los cuales pueden conectarse a placas de expansión (*shields*) que expanden las características de funcionamiento de la placa. Las placas proporcionan interfaces de comunicación serial, incluyendo USB en algunos modelos, para cargar programas desde un ordenador personal. Para programar los microcontroladores, el proyecto Arduino proporciona un IDE para el lenguaje de programación **Arduino**, el cuál surge a partir de otro llamado *Processing* y que es muy similar a C/C++ [31].

Estas placas son desarrolladas con licencia de hardware libre por la compañía Arduino, la cual también las fabrica y comercializa. Sin embargo, también pueden ser manufacturadas por otros fabricantes, al utilizar una licencia de hardware libre.

Aunque existen varios modelos, todos se basan en la misma idea, diferenciándose, principalmente, en el número de pines (digitales y analógicos) de entrada/salida, la cantidad de memoria flash y el modelo de microcontrolador.

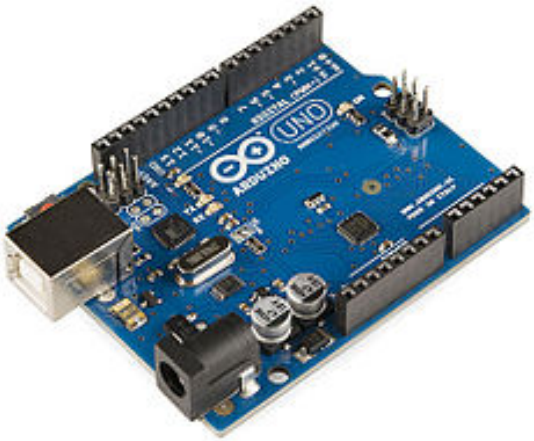

Figura 3.16: Arduino Uno Rev.3. Fuente http://es.wikipedia.org/

3.2.3 Sensores y actuadores

En cuanto a los sensores y actuadores, existe una **amplia gama de los mismos disponibles para Arduino**. Tan solo es necesaria una rápida búsqueda en Internet para comprobar que existen cientos de kits con todo tipo de sensores y actuadores para esta placa.

De entre todos los existentes, se elegirán los más adecuados para medir las condiciones del entorno en el que se encuentra el dron y realizar alguna acción en consecuencia.

Cabe destacar que el piloto automático que gobierna el movimiento del dron también posee sus propios sensores, por lo que podremos utilizar los datos que éste nos proporcione para complementar los obtenidos con el resto de sensores.

Tras analizar los sensores disponibles, se han elegido los siguientes:

- Sensor de temperatura y humedad.

- Sensor barométrico.
- Sensor de agua.
- Sensor de radiación ultravioleta.
- Sensor de monóxido de carbono (CO).
- Sensor de partículas volátiles (HCHO).
- Sensor de ultrasonidos.
- Sensor digital de intensidad luminosa.
- Sensor GPS.
- Sensor de nivel de batería.

Además, se utilizarán dos actuadores:

- Tira de LEDs.
- Cámara.

3.3 Procesamiento de eventos complejos

El procesamiento de eventos es un método de rastreo y análisis (procesamiento) de flujos de información (datos) para derivar una conclusión a partir de ellos. El procesamiento de eventos complejos o **CEP**, por su nombre en inglés *Complex Event Processing*, es **el procesamiento de eventos que combina datos de múltiples fuentes para inferir eventos o patrones que sugieren circunstancias más complicadas**. El objetivo del procesamiento de eventos complejos es identificar eventos significativos (como oportunidades o amenazas) y responder a ellos tan rápido como sera posible [32].

Dicho procesamiento de eventos es el que hará posible identificar las condiciones del entorno en el que se encuentra el dron, de tal forma que, si se detecta un patrón que resulte interesante (por ejemplo, que comience a llover), sea posible actuar en consecuencia.

3.3.1 Drools Fusion

Drools es un sistema de gestión de reglas de negocio, que utiliza un motor de reglas basado en inferencia de reglas hacia delante (*forward chaining*) y hacia detrás (*backward chaining*), conocido como sistema de reglas de producción [33]. Es un proyecto open source de la JBoss Community, que incluye distintos componentes para gestionar las reglas y procesos de negocio. Entre estos componentes, se encuentra Drools Fusion, que es el componente que proporciona el procesamiento de eventos complejos.

3.3.2 Esper

Esper es un software open source para el procesamiento de eventos complejos que analiza series de eventos para derivar conclusiones a partir de ellos. Fue creado en 2006 por EsperTech Inc. y ofrece un lenguaje específico de dominio para el procesamiento de eventos llamado **EPL** (*Event Processing Language*), que es similar a SQL. EPL es un lenguaje de programación declarativa para analizar datos basado en tiempo y detectar las situaciones cuando ocurran [34].

3.4 Enterprise Service Bus

Un ESB *Enterprise Service Bus* es un **middleware que provee interoperabilidad entre diferentes protocolos de comunicación**. Proporciona una capa de abstracción construida sobre una implementación de un sistema de mensajes de empresa que permite explotar el valor del envío de mensajes sin tener que escribir código [35]. Esto permite integrar distintos servicios en una misma aplicación de forma sencilla, creando una arquitectura orientada a servicios (**SOA**).

El uso de la palabra "bus" viene del bus que transporta los bits entre los distintos dispositivos de un ordenador. **El bus de servicio de empresa proporciona una función análoga a un nivel superior de abstracción**. Cuando se utiliza un ESB, las aplicaciones se comunicarán a través del bus, que actúa como intermediario de mensajes entre las aplicaciones.

Idealmente, el ESB debería sustituir todo contacto directo con las aplicaciones, de forma que la comunicación con las mismas tenga lugar a través del bus. Para conseguir esto, el bus debe encapsular la funcionalidad que ofrecen las aplicaciones. Para conseguir dicha encapsulación, se utiliza un modelo de mensajes de empresa, el cual define un conjunto de mensajes normalizado que el ESB recibe y transmite. A veces, ocurre que las aplicaciones no utilizan el mismo modelo de mensajes, por lo que el ESB tendrá que transformar el mensaje a un formato que la aplicación sea capaz de interpretar.

3.4.1 Mule ESB

Mule es un bus de servicio de empresa (ESB) basado en Java liviano, que permite conectar aplicaciones rápida y fácilmente, permitiéndoles intercambiar datos. Está **desarrollado por la empresa MuleSoft y permite una fácil integración de sistemas existentes**, sin importar la tecnología que utilicen (JMS, servicios web, JDBC,

HTTP...). El ESB puede ser desplegado en cualquier sitio y puede integrar y orquestar eventos en tiempo real o por lotes [36].

Este ESB es compatible con múltiples protocolos de cola de mensajes, como pueden ser: AMQP, JMS, WMQ, MQTT..., lo que permitirá escoger el que más se adecúe al proyecto que se esté desarrollando en cada momento. Por otra parte, es multiplataforma, permitiendo ejecutarlo en Windows, OS X, Linux, o plataformas ARM, como Raspberry Pi.

Además del propio motor del ESB, MuleSoft proporciona un IDE que facilita el desarrollo de aplicaciones, llamado **Anypoint Studio** (Mule Studio en versiones anteriores). Dicho IDE dispone de un DSL gráfico con el que es posible crear aplicaciones arrastrando componentes desde una paleta hasta el área de trabajo y configurándolos, lo que reduce en gran medida el tiempo de desarrollo de una aplicación. En Anypoint Studio hay disponibles componentes de todo tipo: conectores, transformadores y filtros de datos, manejadores de excepciones, componentes para ejecutar código en otros lenguajes (Python, Groovy, Ruby...). Además de los componentes que se proporcionan por defecto, también existe una comunidad en la que es posible encontrar más componentes desarrollados por los usuarios [37].

3.5 Protocolos de comunicación

Los protocolos de comunicación son otra de las partes importantes de este trabajo en tanto que hacen posible la comunicación a dos niveles: entre el ordenador de a bordo y el dron, y entre los distintos servicios de la aplicación integrados con el ESB.

Para el primer caso, existen muchas alternativas, aunque la utilización de una u otra dependerá del dron que utilicemos. Los drones de marcas comerciales (como Parrot o DJI) suelen utilizar en los pilotos automáticos sus propios **protocolos propietarios**,

por lo que se dificulta en gran medida el desarrollo de aplicaciones para estos drones. Por otra parte, existen protocolos abiertos, como **MAVLink** [38] utilizados en una gran cantidad de pilotos automáticos, como: ArduPilotMega (APM), pxIMU Autopilot (Pixhawk), SmartAP, Hexo+... Entre los protocolos libres, éste es el mayoritario, estando presente o, al menos, siendo compatible con la mayoría de los pilotos automáticos, por lo que será el que se utilizará.

En cuanto a la comunicación entre los distintos servicios integrados en el ESB, se hará uso del protocolo de cola de mensajes MQTT, ya que se trata de un protocolo muy liviano, por lo que es idóneo en este entorno.

3.5.1 MAVLink

MAVLink o *Micro Air Vehicle Link* es un protocolo abierto (con licencia LGPL) diseñado para la **comunicación con todo tipo de pequeños vehículos no tripulados**. Como se ha comentado, es compatible con la mayor parte de los controladores de vuelo (incluso es compatible con Parrot, aunque éste no lo utiliza como protocolo principal). Su código fuente está disponible en GitHub [39], donde, además, se proporciona un script en Python que permite generar archivos fuente para C, C#, Java, JavaScript, Lua y Python.

Este protocolo está formado por un conjunto de mensajes, que van empaquetados en una estructura fija, que indica el tipo de mensaje, su longitud, la identificación del sistema y el componente que envía el mensaje, el número de secuencia del mismo y un código de comprobación de errores.

De esta forma, el protocolo permite intercambiar todo tipo de información de control (velocidad, posición GPS, modo de vuelo...) y, también, enviar instrucciones al dron (cambio de modo, despegue, aterrizaje...).

3.5.1.1 DroneKit

DroneKit es un proyecto de 3D Robotics, que **busca facilitar la comunicación entre el controlador de vuelo y otro dispositivo**, como puede ser una CGS o un ordenador de a bordo. Dicho controlador de vuelo debe ser compatible con el protocolo MAVLink para que la comunicación pueda llevarse a cabo [40].

Este proyecto ofrece 3 APIs: **DroneKit-Cloud** (un servicio cloud para conectarse al dron y obtener su información), **DroneKit-Python** (orientada a la programación de ordenadores de a bordo) y **DroneKit-Android** (orientada a la creación de aplicaciones para smartphones que funcionen a modo de CGS).

De entre estas tres, la más interesante para este proyecto es la segunda, **DroneKit-Python**, ya que está pensada para la programación de ordenadores de a bordo [41]. Esta API para Python (que, actualmente, está en su versión 2.0) ofrece la posibilidad de conectarse al dron (tanto a través de cable USB como de telemetría), recibir información del mismo, cambiar sus parámetros (por ejemplo, el modo de vuelo), cargar una misión y guiarlo directamente. Con todo esto, se tiene el control necesario sobre el dron para desarrollar el presente trabajo.

3.5.2 MQTT

MQTT, o **Message Queue Telemetry Transport**, es un protocolo de mensajería publish/suscribe, muy simple y liviano, diseñado para dispositivos con una cantidad de recursos limitada y redes no confiables, con poco ancho de banda o elevada latencia. Este protocolo fue creado en 1999 por Andy Stanford-Clark y Arlen Nipper y, actualmente, es un estándar en estado de desarrollo de la ISO [42]. Los principios de diseño del protocolo son: minimizar los requisitos de ancho de banda y recursos a la vez que se asegura la confiabilidad y cierto grado de seguridad de entrega de los mensajes [43]. El

componente que se encargar de enviar los mensajes a los clientes suscritos a un *topic* es el *broker* de mensajes.

Para su funcionamiento, únicamente define 5 tipos de operaciones:

- **Connect**: Establece una conexión con el broker.
- **Disconnect**: Espera a que el cliente MQTT termine cualquier trabajo que tenga pendiente y se desconecta del broker.
- **Subscribe**: Suscribe al cliente a un determinado *topic*.
- **Unsubscribe**: Elimina la suscripción de un cliente a un *topic*.
- **Publish**: Publica un mensaje en un determinado *topic*.

3.5.2.1 Mosquitto

Mosquitto es un **broker de mensajes que implementa el protocolo MQTT** en sus versiones 3.1 y 3.1.1. Fue creado en 2010 por Roger Light como una alternativa open-source a **RSMB** (Really Small Message Broker), otro broker de mensajes para MQTT creado por IBM. Sin embargo, Light decidió ceder el proyecto a Eclipse [44], para incrementar su visibilidad e incrementar un número usuarios y colaboradores del mismo.

Proporciona un servicio de mensajería publish/suscribe muy ligero. Los clientes se conectarán a este servicio para publicar mensajes en uno o varios *topic* y/o para suscribirse a uno o varios *topic* y recibir los mensajes publicados en el mismo. Un *topic* es como una "URL" en la que se publica una determinado mensaje, por ejemplo, el *topic* de un dispositivo que publica la temperatura en el interior de un edificio podría ser **temperatura/interiores/salon**, mientras que un dispositivo que publica datos del tiempo podría hacerlo en el *topic* **tiempo/spain/caceres**.

Este servidor de mensajes está **implementado en C**, ya que esto permite que pueda ser ejecutado en máquinas que ni siquiera tengan capacidad para ejecutar una Máquina Virtual de Java (JVM). La actual implementación de Mosquitto proporciona un ejecutable de unos 120KB que consume aproximadamente 3MB de RAM con 1000 clientes conectados.

3.5.2.2 Eclipse Paho

El proyecto Eclipse Paho proporciona **implementaciones open-source de clientes y utilidades para el protocolo de mensajería MQTT** pensadas para aplicaciones Machine-to-Machine (M2M) y de Internet de las Cosas (IoT). Este proyecto proporciona librerías para trabajar con MQTT en distintos lenguajes: C/C++, Java, JavaScript, Python, Go, .NET y C# [45].

A través de estas librerías, es posible conectarse a un broker, suscribirse a un *topic* y publicar mensajes. Por tanto, proporcionan toda la funcionalidad necesaria para comunicar los servicios que se integrarán con el ESB.

3.6 Trabajos relacionados

Es importante recopilar información acerca de los trabajos que se están desarrollando en este campo antes de comenzar con el presente trabajo, ya que esto permitirá estar al tanto de las últimas novedades. Esto facilitará la búsqueda de una solución a los problemas que puedan surgir durante el desarrollo del mismo, por lo que, a continuación, se presentarán una serie de trabajos relacionados con el ámbito de este TFG.

- W. Liu, Z. Zheng y K.-Y. Cai [46] propusieron en 2013 un **sistema de planificación de rutas en tiempo real** para evitar los obstáculos que se presenten en el

camino de un dron. En este caso, centraron su trabajo en drones de ala fija, teniendo en cuenta únicamente dos dimensiones, aunque éste podría extenderse a drones multirrotor y entornos de tres dimensiones (de hecho, en un apartado se discute esta última posibilidad). Para resolver el problema, toman datos del entorno y, cuando se detecta un posible obstáculo entre el punto actual y el destino, se calcula un punto intermedio para esquivar dicho obstáculo. Para calcular este punto, utilizan programación binivel, estableciendo como objetivo primario (objetivo del líder) la reducción del ángulo entre la dirección del dron en el nuevo punto y la línea que une este punto con el destino, y como objetivo secundario, la reducción de la distancia entre el nuevo punto y el objetivo. Así, apoyándose en una serie de heurísticas, se calcula el punto que mejor satisface ambos objetivos.

- Continuando con la línea del trabajo anterior, existen dos trabajos desarrollados en el MIT que tratan de **evitar los obstáculos utilizando otra aproximación** [47]. El primero de ellos, en vez de calcular el camino teniendo en cuenta el número de obstáculos que hay, se centra en los segmentos de espacio libre que existen, de forma que construye la ruta uniendo dichos espacios libres. El segundo trabajo, utiliza una librería de "túneles" por los que el dron podría moverse, de forma que durante el vuelo se comprueba constantemente dicha librería en busca de un "túnel" que garantice que el dron no colisione con ninguno de los obstáculos que hay en su camino.

- Otro trabajo interesante es el desarrollado por la universidad de KAIST [48], en Corea del Sur, en el que se propone un sistema para **planificar de forma eficiente las misiones de los drones que sirven de apoyo en rescates marítimos**. El objetivo de dicho sistema, es tratar de forma eficiente dos problemas que se presentan en estos sistemas de apoyo: que los drones tienen una autonomía de vuelo definida, por lo que no pueden operar indefinidamente y que la probabilidad de que un superviviente esté en un área determinada cambia con el tiempo debido a la corriente y el viento. De esta forma, se propone un sistema que gestiona varios drones y estaciones de recarga de combustible, y se encarga de crear misiones de

búsqueda para los drones en las que se maximice la probabilidad de encontrar a los supervivientes y que tengan en cuenta el nivel de combustible de los drones, de forma que, si un dron va a quedarse sin combustible, sea reemplazado por otro y vaya a la estación de recarga.

- Por último, y más centrado en la toma de decisiones en función de una serie de reglas, existe un artículo en el que se propone el **uso de una red de Petri difusa para la toma de decisiones en drones** [49]. Se trata de un sistema para drones militares, en el que la decisión a tomar es si disparar o no a un determinado objetivo. Para ello, se toman 3 valores: el valor del objetivo, el nivel de amenaza del mismo y el nivel de combustible del dron. Dichos valores, que en un principio son numéricos, se convierten en valores difusos y se utilizan como entrada a un sistema de reglas para tomar la decisión. Este sistema es similar al que se propone en este TFG en el sentido de que también se define un sistema de reglas para tomar decisiones, sin embargo, en lugar de utilizar procesamiento de eventos complejos, se utiliza un red de Petri, por lo que se ve que, aunque con otro enfoque, también se está trabajando en este campo.

3.7 Conclusiones

Una vez se han dejado claros los conceptos que van a tratarse más adelante, es posible proponer un sistema que resuelva el problema planteado. Se han intentado cubrir todos los aspectos relacionados con el sistema desarrollado, desde el dron hasta los sensores, pasando por todas las librerías y tecnologías que servirán de apoyo para poder procesar los datos y, en caso de detectar un evento relevante, ejecutar una acción en respuesta. Así, se han presentado cada uno de estos aspectos, no entrando demasiado en detalle en cada uno, pero sí lo suficiente como para que puedan entenderse el resto de capítulos de este TFG.

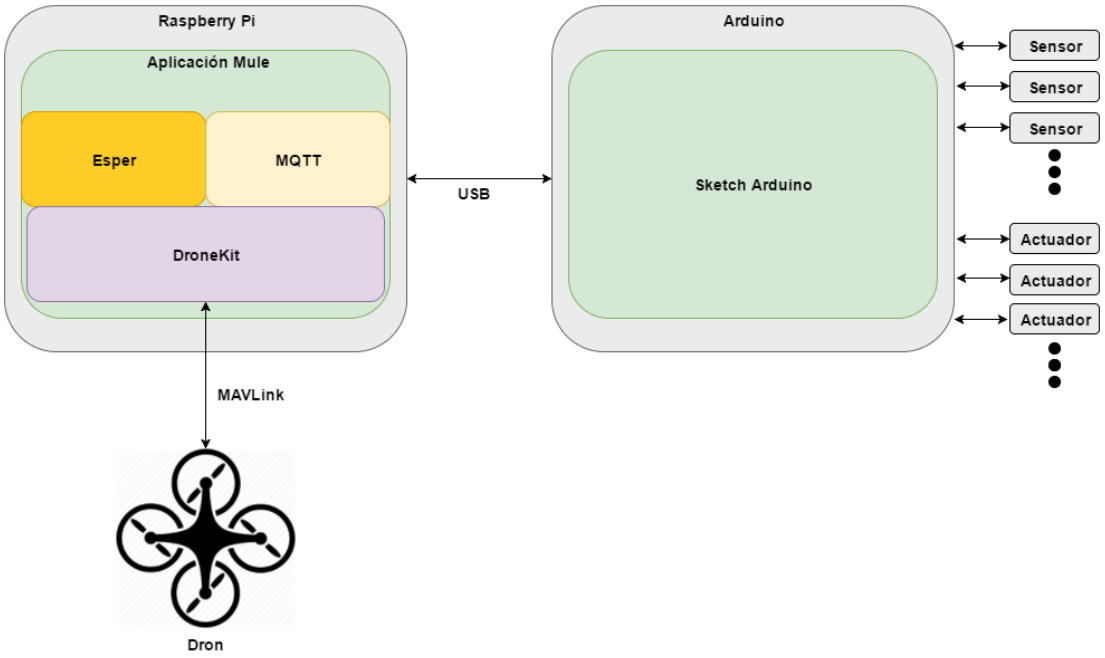

Figura 3.17: Esquema del sistema propuesto

En cuanto a los puntos en los que se han presentado varias alternativas, (esto es, en el Procesamiento de Eventos Complejos y el Ordenador de a bordo), se ha decidido lo siguiente: Como motor de procesamiento de eventos complejos, **se utilizará Esper, ya que parece más orientado al procesamiento de un flujo de datos en el tiempo**, mientras que Drools Fusion está más orientado a reglas y procesos de negocio. En el caso del ordenador de a bordo, prácticamente todas las alternativas que se han presentado podrían ser válidas, ya que permiten instalar una distribución de Linux y tienen una capacidad de computo y memoria suficiente, por lo que se ha decidido utilizar una **Raspberry Pi 2 Model B**, ya que, como se ha dicho, tiene la capacidad suficiente para ejecutar el sistema que se propone y está disponible en el laboratorio, por lo que supone un menor coste tanto en dinero como en tiempo de espera respecto a lo que supondría utilizar un *single-board computer* distinto.

Capítulo 4

Propuesta de solución

4.1 Visión general del sistema

En esta sección se explicará a grandes rasgos cómo es el sistema propuesto, para, en las siguientes, entrar más en detalle en cada una de sus partes. En la **Figura 4.1** se muestra el sistema propuesto, tanto desde el punto de vista físico, como desde el lógico.

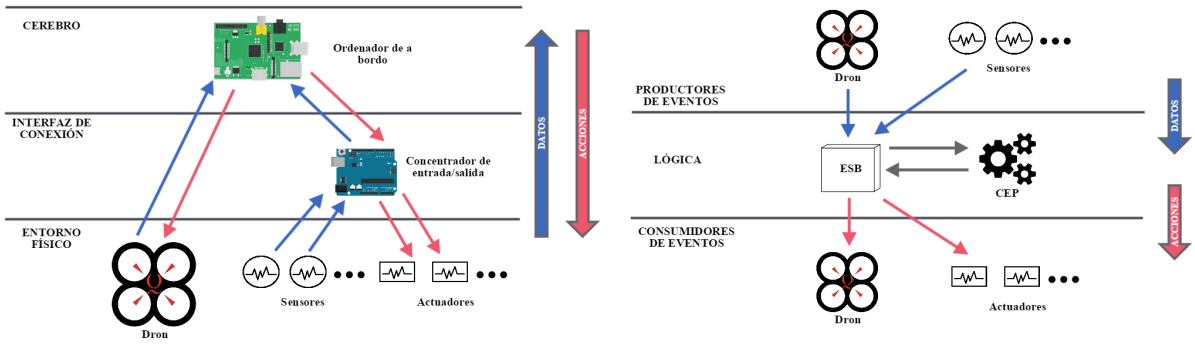

Figura 4.1: Sistema propuesto. Punto de vista físico (izq.) y punto de vista lógico (der.)

Es posible dividir el sistema en 3 grandes partes:

- **Dron:** Es el **elemento que se quiere controlar**. Se comunica con la Raspberry Pi para enviarle datos referentes a su posición GPS, altura, batería... y llevar a cabo las órdenes que ésta le envía. Aquí no se tendrá que llevar a cabo ninguna modificación, pues el control adicional que va a ejercerse sobre el dron lo llevará a cabo la Raspberry Pi, utilizando la librería DroneKit-Python.

- **Raspberry Pi:** Hace la función de **ordenador de a bordo**. Para ello, contiene toda la lógica para el procesado de eventos, la detección de eventos y la ejecución de una determinada acción en respuesta a los mismos. De esto es de lo que se encarga la aplicación Mule a través de los distintos componentes que la forman. Por otra parte, aquí también se ha colocado una cámara, para poder tomar vídeos e imágenes. Esta cámara se comportará como si fuese un actuador más.

- **Arduino:** Sirve como **concentrador de entrada/salida**, esto es, en él se conectarán los sensores con los que se obtendrán los datos del entorno y los actuadores que se activarán en respuesta a algunos de los eventos.

La conexión entre las 3 partes del sistema se lleva a cabo a través de cables USB. Es decir, la Raspberry Pi está conectada tanto al dron como al Arduino mediante un par de cables USB.

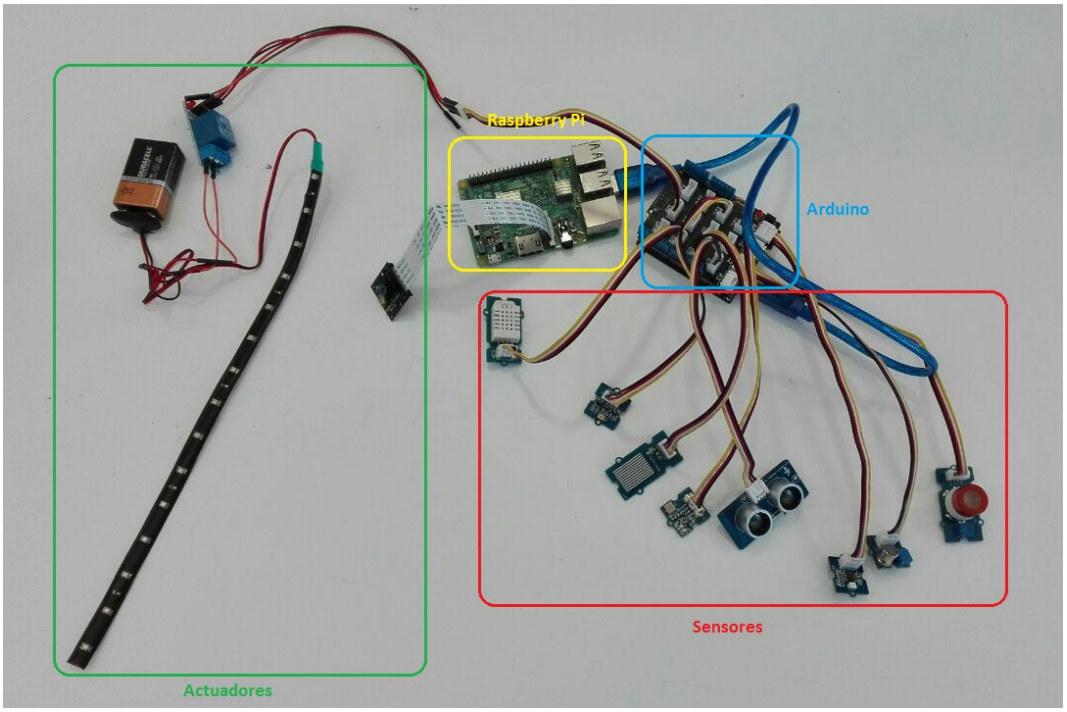

Figura 4.2: Arduino y Raspberry Pi junto con los sensores y actuadores utilizados

4.2 Arquitectura Hardware

Una vez se han descrito qué partes forman el sistema propuesto y de qué se encarga cada una de ellas, se procederá a detallar los componentes hardware del mismo.

4.2.1 Dron

Como se ha comentado anteriormente, es el **componente que se va a controlar con el sistema**. Si bien por sí mismo puede volar, no incorpora ningún sistema para, por ejemplo, evitar choques (aunque ciertas marcas como DJI han desarrollado sistemas para evitar esto [5]) o cumplir con la legalidad vigente (no volar a más de 400 pies, no volar áreas restringidas como aeropuertos...). Esto es lo que se intentará cubrir en este

trabajo y para lo que se añadirán el resto de sistemas, como el ordenador de a bordo y el conjunto de sensores.

El dron que va a emplearse es un **cuadricóptero**, que utiliza una **placa ArduPilot Mega 2.6** (APM 2.6) como piloto automático [50]. Las placas APM ya no están soportadas por las últimas versiones (a partir de la versión 3.3) del firmware que utiliza y éste solo puede utilizarse en otras con CPUs más rápidas, como Pixhawk o VRBrain, sin embargo esto no supone ningún impedimento, ya que es posible usar una versión anterior. Por otra parte, todas las versiones del firmware utilizan el protocolo **MAVLink**, por lo que el sistema será compatible con cualquier controlador de vuelo que lo utilice (sin importar la versión del firmware).

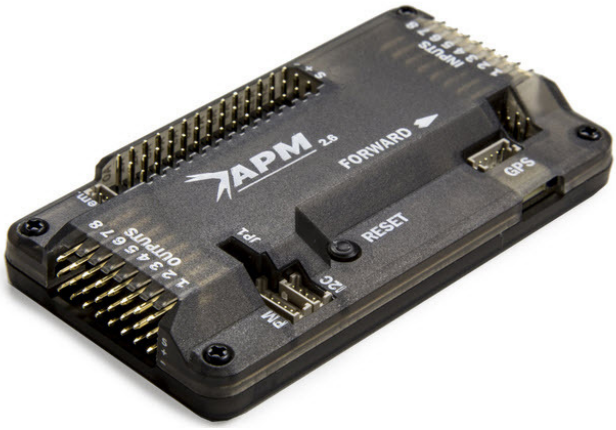

Figura 4.3: Controlador de vuelo APM 2.6. Fuente: http://dronecasero.blogspot.com.es/

En cuanto al resto de componentes del dron, no hay mucho más que comentar, ya que será el controlador de vuelo el que se encargará de comunicarse con ellos. Estos otros componentes son, simplemente, los necesarios para tener un dron funcional, es decir: motores, ESCs, módulo GPS, batería...

Figura 4.4: Dron utilizado para el trabajo

4.2.2 Raspberry Pi

La Raspberry Pi es el componente que **contendrá toda la lógica para la recepción de datos de los sensores, detección de patrones de eventos y ejecución de acciones en respuesta a los mismos**. El modelo que se utilizará será una Raspberry Pi 2 Model B, cuyas especificaciones se muestran en la **Tabla 4.1**.

En este componente, por tanto, será **donde estará en ejecución la aplicación Mule desarrollada**, la cual utilizará el motor de procesamiento de eventos complejos Esper para la detección de patrones de eventos.

SoC	Broadcom BCM2836 (CPU + GPU + DSP + SDRAM + puerto USB)
CPU	ARM 1176JZF-S a 700 MHz (familia ARM11)
Juego de Instrucciones	RISC de 32 bits
GPU	Broadcom VideoCore IV a 250 MHz, OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia), 1080p30 H.264/MPEG-4 AVC
Memoria (SDRAM)	1 GB (compartidos con la GPU)
Puertos USB 2.0	4
Entradas de vídeo	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la RPF
Salidas de vídeo	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), Interfaz DSI para panel LCD
Salidas de audio	Conector de 3.5 mm, HDMI
Almacenamiento integrado	MicroSD
Conectividad de red	10/100 Ethernet (RJ-45) via hub USB
Periféricos de bajo nivel	17 x GPIO y un bus HAT ID
Consumo energético	800 mA, (4.0 W)
Fuente de alimentación	5 V vía Micro USB o GPIO header
Dimensiones	85.60mm x 53.98mm
Sistemas operativos soportados	GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux. No Linux: RISC OS, Windows 10 IoT Core

Tabla 4.1: Especificaciones de Raspberry Pi 2 Model B. Fuente https://es.wikipedia.org/wiki/Raspberry_Pi

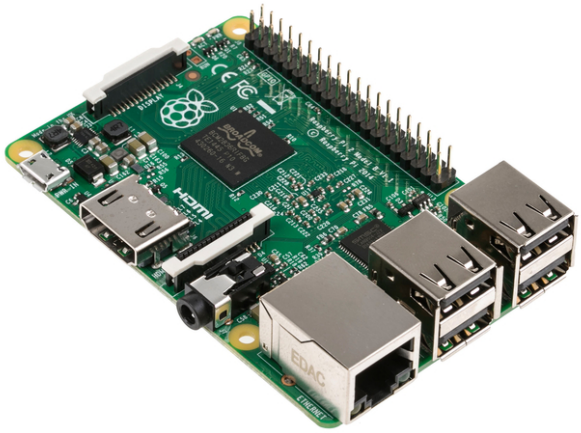

Figura 4.5: Raspberry Pi 2 Model B. Fuente: http://wordnice.net/

4.2.3 Arduino

La placa Arduino que se va a utilizar es una implementación del Arduino Uno hecha por una empresa alemana [51] (esto es posible porque Arduino es open hardware). El modelo concreto es el **Funduino Uno Rev.3**, que tiene las mismas características que un Arduino Uno Rev.3, las cuales se muestran en la **Tabla 4.2**.

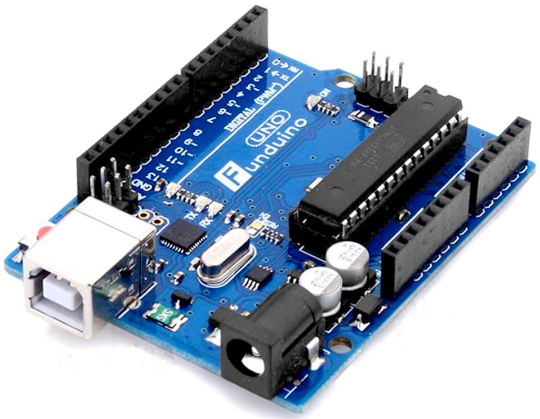

Figura 4.6: Funduino Uno R3. Fuente: http://wordnice.net/

Microcontrolador	ATmega328
Voltaje de operación	5V
Voltaje de entrada recomendado	7-12V
Límites del voltaje de entrada	6-20V
Pines digitales de entrada/salida	14
Pines analógicos de entrada	6
Corriente continua por pin de entrada/salida	40 mA
Corriente continua por pin de 3.3V	50 mA
Memoria flash	32KB, de los cuales 0.5KB son utilizados por el cargador de arranque (bootloader)
SRAM	2KB
EEPROM	1KB
Frecuencia de reloj	16 MHz
Dimensiones	68.6mm x 53.4mm

Tabla 4.2: Especificaciones de Funduino Uno R3. Fuente http://www.nexuscyber.com/funduino-uno-r3-atmega328

Para conectar los sensores al Arduino, hemos utilizado una shield (es decir, una placa que se conecta sobre el Arduino para extender sus funcionalidades) que nos facilita la conexión de los sensores. La shield utilizada es una Grove Base Shield V1.3 [52], que proporciona un conjunto de conectores Grove, lo que nos facilita la utilización de sensores Grove, que utilizan el mismo conector, por lo que solo tendremos que enchufarlos en uno de los conectores y ya podremos usarlos. Las características de esta shield se muestran en la **Tabla 4.3**.

Voltaje de operación	+5V
Conectores Grove	16
Conectores digitales	8 (de los conectores Grove)
Conectores IIC	4 (de los conectores Grove)
Conectores analógicos	4 (de los conectores Grove)
¿Shield estándar?	Sí

Tabla 4.3: Especificaciones de Grove Base Shield V1.3. Fuente http://www.seeedstudio.com/wiki/Grove_-_Base_Shield

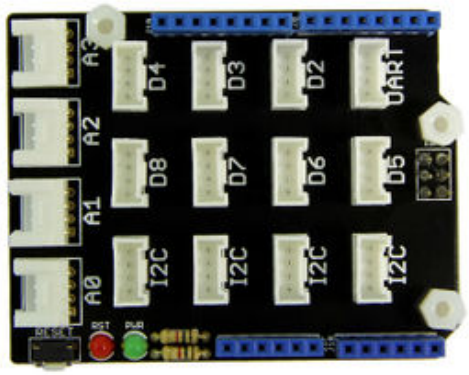

Figura 4.7: Grove Shield. Fuente: http://www.seeedstudio.com/

4.2.4 Sensores

Se hace uso de un conjunto de 8 sensores, que utilizan un conector Grove (para conectarlos al Arduino a través de la Grove Base Shield), para obtener las condiciones del entorno en el que está el dron (temperatura, humedad, nivel de luz...). Además de estos sensores, también se utilizan un par de sensores del dron para obtener las coordenadas GPS donde éste se encuentra y su nivel de batería. A continuación, se enumeran cada uno de los sensores utilizados, junto con sus características:

1. **Sensor de temperatura y humedad** [53]: Este sensor proporciona tanto la temperatura (en grados Celsius), como la humedad relativa (en %) del entorno.

Sus características se muestran en la **Tabla 4.4**. Este sensor se conectará al **pin digital 2 (D2)**.

Figura 4.8: Sensor de temperatura y humedad. Fuente: http://www.seeedstudio.com/

	Mínimo	Normal	Máximo
Vcc	3.3V	-	6V
Medición del suministro de corriente	1mA	-	1.5mA
Suministro de corriente en reposo	40uA	-	50uA
Rango de medida (humedad)	5 %	-	99 %
Rango de medida (temperatura)	-40°C	-	80°C
Precisión (humedad)	-	-	±2 %
Precisión (temperatura)	-	-	±5°C
Resolución (humedad)	-	-	0.1 %
Resolución (temperatura)	-	-	0.1°C
Repetibilidad (humedad)	-	-	±0.3 %
Repetibilidad (temperatura)	-	-	±0.2°C
Estabilidad a largo plazo	-	-	±0.5 %/año
Periodo de medición de señal	-	2s	-
Tiempo de respuesta	6s	-	20s

Tabla 4.4: Especificaciones del sensor de temperatura y humedad. Fuente http://www.seeedstudio.com/wiki/Grove_-_Temperature_and_Humidity_Sensor_Pro

2. **Barómetro** [54]: El sensor barométrico mide la presión atmosférica en Pascales (Pa). Sus características se muestran en la **Tabla 4.5**. Este sensor se conectará a **uno de los pines I2C**.

Figura 4.9: Barómetro. Fuente: http://www.seeedstudio.com/

	Mínimo	Típico	Máximo
Voltaje	3V	5V	5.5V
Corriente	89uA	-	130uA
Rango de presiones	300hPa	-	1100hPa
Mayor tasa de transferencia de datos a través de I2C	-	-	3.4MHz
Dimensiones	20.4mm x 24.3mm x 9.7mm		
Peso	2g		

Tabla 4.5: Especificaciones del barómetro. Fuente http://www.seeedstudio.com/wiki/Grove_-_Barometer_Sensor

3. **Sensor de agua** [55]: Este sensor indicará si ha caído agua en su superficie, por tanto, lo se utilizará para saber si está lloviendo. Sus características se muestran en la **Tabla 4.6**. Este sensor se conectará al **pin digital 3 (D3)**.

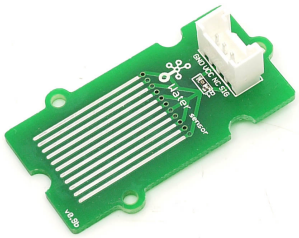

Figura 4.10: Sensor de agua. Fuente: http://www.seeedstudio.com/

	Mínimo	Típico	Máximo
Voltaje de trabajo	4.75V	5V	5.25V
Corriente		<20mA	
Temperatura de trabajo	10ºC	-	30ºC
Humedad de trabajo (sin condensación)	10 %	-	90 %

Tabla 4.6: Especificaciones del sensor de agua. Fuente http://www.seeedstudio.com/wiki/Grove_-_Water_Sensor

4. **Sensor de radiación ultravioleta** [56]: Este sensor proporciona un voltaje de salida que es proporcional al índice de radiación UV, por lo que dicho voltaje se convertirá para obtener el índice UV actual. Sus características se muestran en la **Tabla 4.7**. Este sensor se conectará al **pin analógico 0 (A0)**.

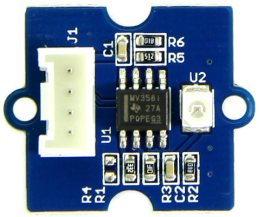

Figura 4.11: Sensor de radiación ultravioleta. Fuente: http://www.seeedstudio.com/

	Mínimo	Típico	Máximo
Voltaje de trabajo	3V	5V	5.1V
Corriente		0.31mA	
Longitud de onda utilizada	240nm	-	370nm
Temperatura de trabajo	-30ºC	-	85ºC

Tabla 4.7: Especificaciones del sensor de radiación ultravioleta. Fuente http://www.seeedstudio.com/wiki/Grove_-_UV_Sensor

5. **Sensor de monóxido de carbono (CO)** [57]: Con este sensor es posible medir la concentración de monóxido de carbono que existe en el aire, en partículas por millón (ppm). Sus características se muestran en la **Tabla 4.8**. Este sensor se conectará al **pin analógico 1 (A1)**.

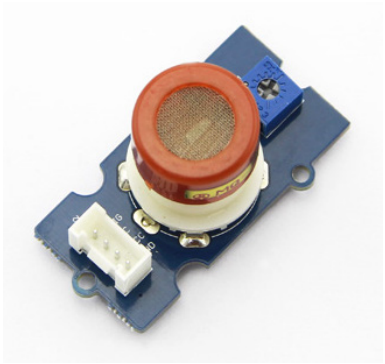

Figura 4.12: Sensor de CO. Fuente: http://www.seeedstudio.com/

	Mínimo	Típico	Máximo
Voltaje de trabajo	4.9V	5V	5.1V
Consumo del calentador	0.5mW	-	340mW
Resistencia de carga	ajustable		
Resistencia del calentador	33Ω±5 %		
Resistencia del sensor	2Ω	-	20000Ω
Detección de concentración	200ppm	-	1000ppm

Tabla 4.8: Especificaciones del sensor de CO. Fuente http://www.seeedstudio.com/wiki/Grove_-_Gas_Sensor(MQ9)

6. **Sensor de partículas volátiles (HCHO)** [58]: Con este sensor es posible medir la concentración de formaldehído (HCHO), tolueno, benceno y alcohol que existe en el aire, en partículas por millón (ppm). Sus características se muestran en la **Tabla 4.9**. Este sensor se conectará al **pin analógico 2 (A2)**.

Figura 4.13: Sensor de HCHO. Fuente: http://www.seeedstudio.com/

	Mínimo	Típico	Máximo
Voltaje de trabajo	4.7V	5V	5.3V
Rango de concentración	1ppm	-	50ppm
Valor de la resistencia del sensor	10KΩ	-	100KΩ
Sensibilidad	Rs(en el aire)/Rs(10ppm HCHO)≥5		

Tabla 4.9: Especificaciones del sensor de HCHO. Fuente http://www.seeedstudio.com/wiki/Grove_-_HCHO_Sensor

7. **Sensor de ultrasonidos** [59]: El sensor de ultrasonidos proporciona la distancia que existe desde el mismo hasta un posible obstáculo. Sus características se muestran en la **Tabla 4.10**. Este sensor se conectará al **pin digital 4 (D4)**.

Figura 4.14: Sensor de ultrasonidos. Fuente: http://www.seeedstudio.com/

	Mínimo	Típico	Máximo
Voltaje de trabajo	3.3V	-	5V
Corriente de trabajo	15mA		
Frecuencia de ultrasonidos	42KHz		
Rango de medición	3cm	-	400cm
Resolución	1cm		

Tabla 4.10: Especificaciones del sensor de ultrasonidos. Fuente http://www.seeedstudio.com/wiki/Grove_-_Ultrasonic_Ranger

8. **Sensor digital de intensidad luminosa** [60]: Este sensor mide la intensidad lumínica en Lux. Permite elegir entre medir todo el espectro o únicamente el espectro infrarrojo. Sus características se muestran en la **Tabla 4.11**. Este sensor se conectará a **uno de los pines I2C**.

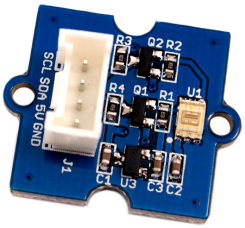

Figura 4.15: Sensor de intensidad luminosa. Fuente: http://www.seeedstudio.com/

	Mínimo	Típico	Máximo
Voltaje de trabajo	3.3V	5V	5.1V
Temperatura de operación	-30ºC	-	70ºC
Rango de medición	0.1lux	-	40000lux

Tabla 4.11: Especificaciones del sensor de intensidad luminosa. Fuente http://www.seeedstudio.com/wiki/Grove_-_Digital_Light_Sensor

9. **Sensor GPS y sensor de nivel de batería**: Los datos de estos dos sensores los se tomarán del dron, haciendo uso de la librería DroneKit-Python. Con el GPS se obtendrán la latitud, longitud y altura relativa al punto de despegue; mientas que con el de nivel de batería se obtendrá el porcentaje de batería que le queda al dron.

4.2.5 Actuadores

Como actuadores, únicamente se utilizan dos elementos: una tira de LEDs y una cámara. Solo se utilizan estos dos actuadores, porque creemos que es suficiente para demostrar que el sistema propuesto funciona correctamente. Estos actuadores se activarán en respuesta a la detección de algunos de los patrones de eventos que se definan (es necesario recordar que, en respuesta a un determinado patrón, también puede realizarse una acción con el dron, como aterrizar o ir a un punto).

1. **Tira de LEDs** [61]: Este actuador simplemente se encenderá o apagará cuando

se quiera. No se encenderá o apagará la tira de LEDs directamente, sino que ésta estará conectada a una pila de 9V a través de un relé (conectado al **pin digital 5, D5**); por tanto, lo que se hará será cerrar o abrir el circuito utilizando el relé, lo que hará que la tira de LEDs se encienda o apague, respectivamente. En la Figura 4.16 puede verse la forma en que se ha montado esto, además del cable con conector Grove utilizado para conectarlo a la shield de Arduino. En cuanto a sus características, en la Tabla 4.12 se muestran las de la tira de LEDs y en la Tabla 4.13, las del relé.

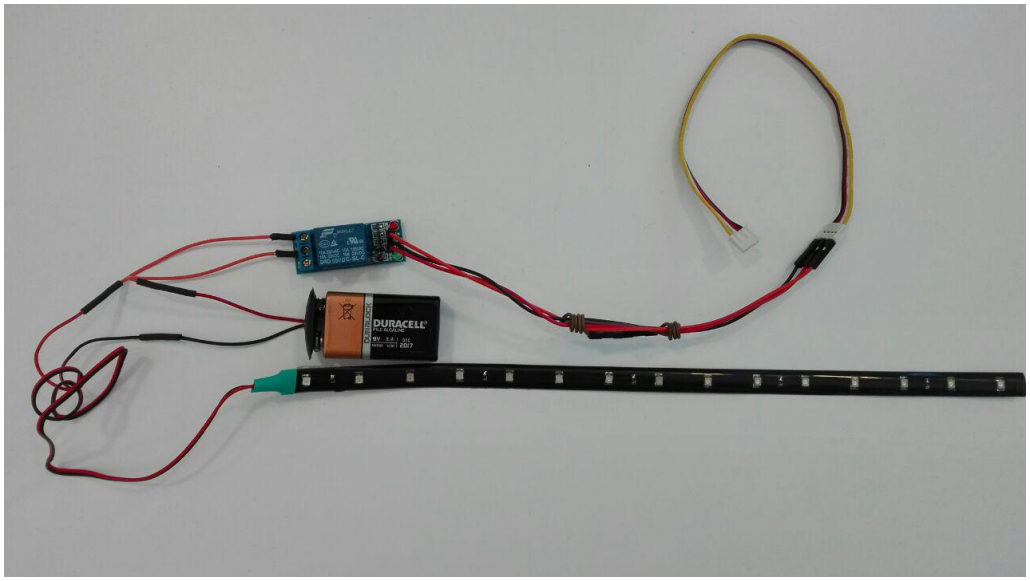

Figura 4.16: Circuito que controla la tira de LEDs.

Tensión de trabajo	12V
Número de LEDs	15
Potencia de salida	0.9W
Temperatura de trabajo	0.9W
Vida útil	0.9W
Resistencia al agua	0.9W
Dimensiones	0.9W

Tabla 4.12: Especificaciones de la tira de LEDs. Fuente https://www.amazon.es/PsmGoods-coches-prueba-flexible-l%C3%A1mpara/dp/B00Y7J3WLO

Voltaje nominal	5V
Corriente nominal	71.4mA
Resistencia de la bobina	10Ω±10 %
Consumo energético	0.36W
Temperatura de trabajo	-25ºC a 70ºC
Humedad de trabajo	45 % a 85 %
Dimensiones	17mm x 43mm x 16.3mm
Peso	11g

Tabla 4.13: Especificaciones del relé. Fuente https://www.ghielectronics.com/downloads/man/20084141716341001RelayX1.pdf

2. **Cámara Pi NoIR** [62]: Se trata de una cámara que se conecta directamente a la Raspberry Pi, a través del **puerto CSI**. Permite tomar tanto fotos como vídeo, haciendo uso de la librería para Python llamada Picamera [63]. La peculiaridad de este modelo es que, además de la luz visible, también capta la luz infrarroja, por lo que haciendo uso de una luz de este tipo, es posible tomar imágenes en la oscuridad. En la Tabla 4.14 se muestran algunas de las características de esta cámara.

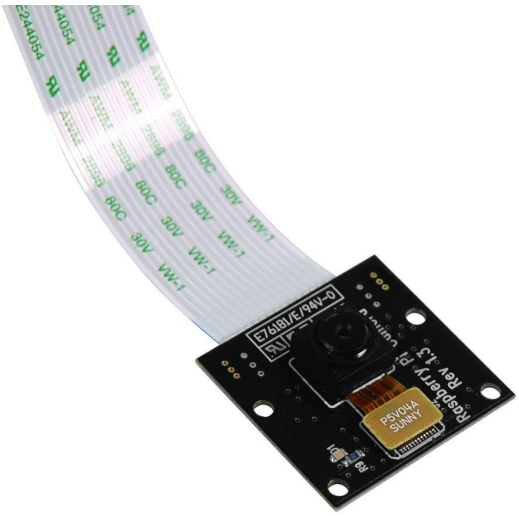

Figura 4.17: Cámara Pi NoIR. Fuente http://www.gravitech.us/

Sensor de imágen	Omnivision 5647 CMOS (sin filtro de infrarrojo)
Resolución	5MP (2592px x 1944px)
Soporte de vídeo	1080p a 30fps, 720p a 60fps
Rango de temperatura (operativo)	-30ºC a 70ºC
Rango de temperatura (imagen estable)	0ºC a 50ºC
Tamaño de la lente	1/4"
Dimensiones	20mm x 25mm x 10mm
Peso	3g

Tabla 4.14: Especificaciones de la cámara Pi NoIR. Fuente https://www.pollin.de/shop/downloads/D701946D.PDF

4.2.6 Batería

Para proporcionar energía al sistema (concretamente a la Raspberry Pi) durante el vuelo, es necesario utilizar una batería auxiliar. Durante las pruebas, simplemente se ha conectado la Raspberry a la corriente con un cargador USB; sin embargo, cuando el sistema esté montado en el dron, esto no será posible, por lo que se tendrá que utilizar una batería USB. La batería en cuestión se muestra en la Figura 4.18.

Figura 4.18: Batería USB utilizada para proporcionar energía a la Raspberry Pi.

4.3 Arquitectura Software

A continuación, se detallarán cada uno de los elementos software que forman el sistema propuesto. Éstos se encuentran en dos partes: un sketch en el Arduino y una aplicación Mule en la Raspberry Pi.

4.3.1 Sketch Arduino

Un sketch de Arduino es un programa similar a cualquier programa C, que **permite controlar el comportamiento de una placa Arduino**. Se compone de dos funciones obligatorias: **setup()** y **loop()**. La primera de ellas, se llama una sola vez,

cuando se inicia la ejecución del sketch y sirve para realizar tareas como inicializar las librerías que vayan a utilizarse o poner el modo en que vayan a utilizarse los pines (cuáles cómo entrada, cuáles como salida...). En cuanto a la función loop(), es un bucle infinito que contendrá el código principal que se quiera ejecutar en el sketch.

En este caso, se utilizará la función loop() para **leer los sensores de cada uno de los valores y escribirlos en el puerto serie** (lo que hará que se envíen por el puerto USB). Para la lectura de los valores de los sensores, en muchos casos, se hará uso de librerías diseñadas para el sensor en cuestión, mientras que en otros casos (como con el sensor de agua) sólo es necesario leer el valor obtenido del pin en el que esté conectado. Una vez se ha obtenido el valor, éste se envía al puerto serie en formato JSON, para que después sea más sencillo tratarlo. Para realizar esta tarea, se utiliza una librería llamada ArduinoJson [64], que facilita la labor de crear el JSON. El formato del JSON enviado por cada lectura de un sensor es el siguiente:

```
{
    "sensorType": "Temperature",
    "sensorId": 3,
    "timestamp": 0,
    "GPSLatitude": 0,
    "GPSLongitude": 0,
    "GPSAltitude": 0,
    "value": 23.56
}
```

En el campo **"sensorType"** se indica el tipo de sensor (temperatura, agua, UV...). En **"sensorId"**, se pone el identificador del sensor, es decir, si se utilizan 3 sensores del mismo tipo, uno de ellos será el 0, otro será el 1 y el último será el 2. En el campo **"timestamp"**, se indica la fecha y hora en formato Unix (es decir, en segundos

desde las 00:00:00 del 01/01/1970); este campo lo se rellenará en la Raspberry Pi, ya que el Arduino no contiene un reloj desde el que coger la fecha y hora, por tanto, lo enviará a 0. Los campos **"GPSLatitude", "GPSLongitude"** y **"GPSAltitude"** contendrán la latitud, longitud y altura (relativa al punto de despegue del dron) donde se tomó la medida del sensor; en principio, estos datos los se tomaban con un sensor GPS conectado al Arduino, pero después se decidió coger dichos valores del GPS del dron, ya que el uso del GPS y su librería asociada complicaba mucho el código del sketch de Arduino, por tanto, estos 3 campos también irán a 0 y se rellenarán en la Raspberry Pi. Por último, el campo **"value"** contiene el valor obtenido del sensor, es decir, la temperatura, humedad, índice UV...

4.3.2 Mule ESB

La aplicación Mule se encargará de varias cosas: recoger los datos generados por los sensores, convertirlos en eventos que pueda procesar Esper y enviárselos, parsear los ficheros que contienen los patrones de eventos que se han definido e insertarlos en Esper, parsear los ficheros que definen las acciones asociadas a cada patrón de eventos e incluirlas en la propia aplicación Mule y ejecutar el script en Python que se encarga de conectarse al dron y ejecutar las acciones asociadas a los patrones de eventos. En la Figura 4.19 y la Figura 4.20 pueden verse cada uno de los flujos que forman la aplicación. Esta aplicación Mule está basada en la desarrollada por el Dr. Juan Boubeta, profesor de la Universidad de Cádiz, en el Capítulo 6 de su tesis doctoral [65], que ha sido adaptada a nuestro sistema.

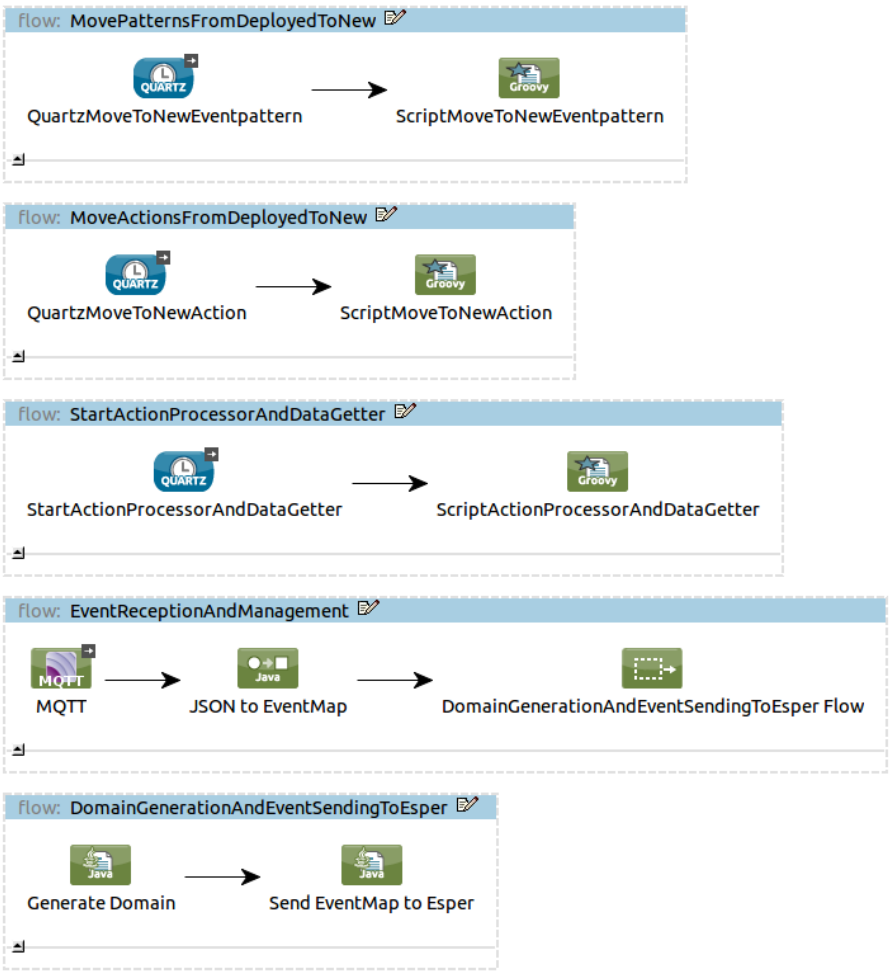

Figura 4.19: Flujos que forman la aplicación Mule (I).

Los dos primeros flujos de la Figura 4.19 (**MovePatternsFromDeplyedToNew** y **MoveActionsFromDeplyedToNew**), simplemente **mueven los archivos donde se definen los patrones de eventos y los que definen las acciones asociadas a los mismos** desde las carpetas **deployed-eventpattern** y **deployed-action** a las carpetas **new-eventpattern** y **new-action**, respectivamente. Esto es necesario porque cuando los patrones y acciones se despliegan, los ficheros donde se definen se mueven a deployed-eventpattern y deployed-action, respectivamente, mientras que la aplicación los lee desde las carpetas new-eventpattern y new-action, respectivamente, por lo que es necesario moverlos cada vez que la aplicación se inicia, para que ésta los encuentre.

El tercer flujo (**StartActionProcessorAndDataGetter**), se encarga de poner en ejecución el script Python que, por una parte, recoge los datos del dron y los que el Arduino envía por USB y los publica en un *topic* de Mosquitto, por otra, ejecuta cada una de las acciones asociadas a la detección de cada patrón. De este script se hablará más detenidamente en la Subsubsección 4.3.2.3.

El cuarto flujo (**EventReceptionAndManagement**), está suscrito al *topic* **"drone/sensors"** de Mosquitto donde cada mensaje es un JSON con cada valor medido por los sensores. De esta forma, va obteniendo cada uno de estos JSON y los convierte al tipo Map de Java. Tras esto, ese Map se pasa al flujo **DomainGenerationAndEventSendingToEsper** que, en primer lugar, añade al motor Esper el tipo de evento representado por ese Map (si no existe) y, a continuación, envía el propio evento al motor Esper, para que éste pueda procesarlo.

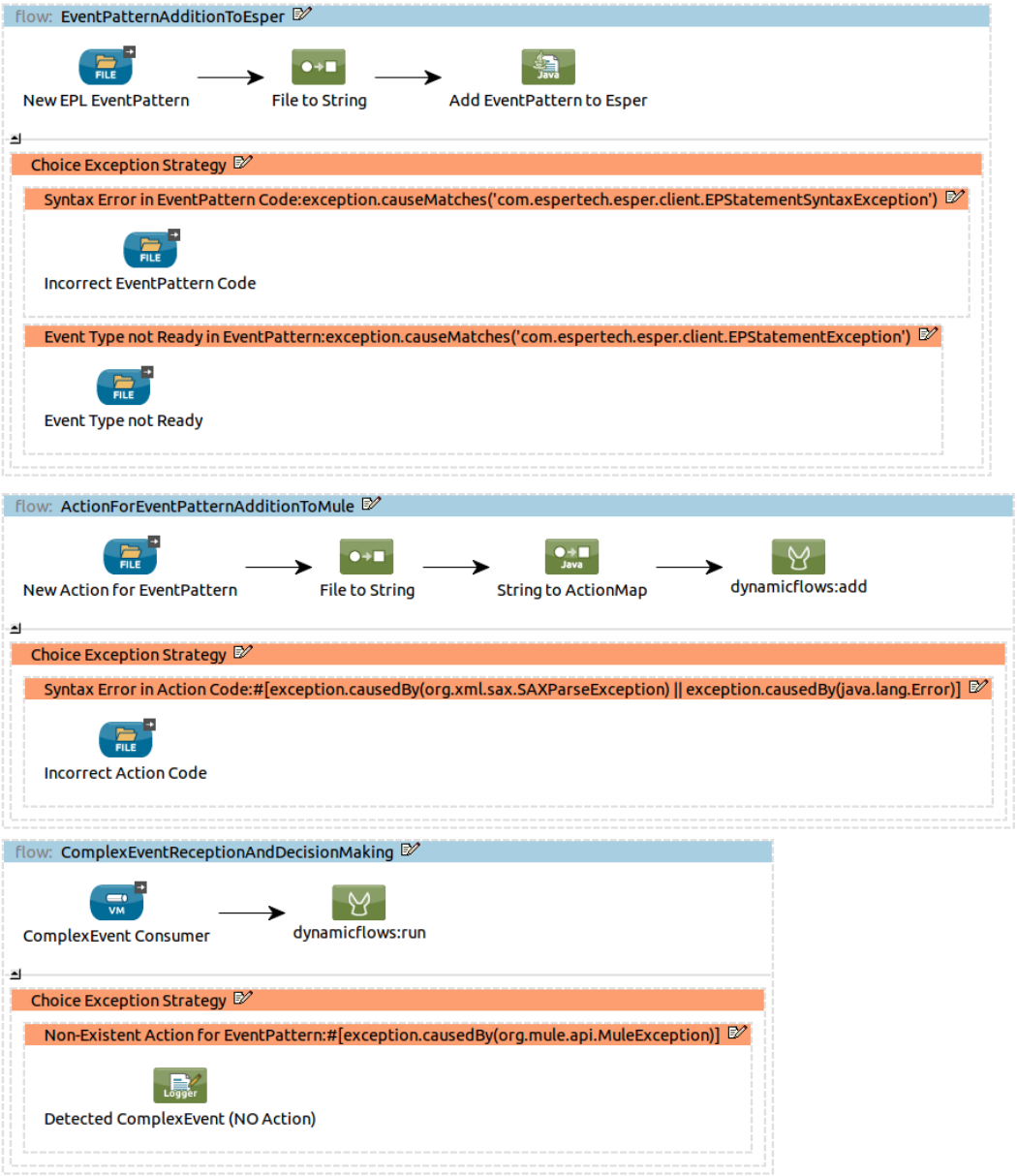

Figura 4.20: Flujos que forman la aplicación Mule (II).

En cuanto a la Figura 4.20, el primer flujo (**EventPatternAdditionToEsper**) coge los archivos de patrones de eventos (aquellos con extensión **.epl**) que existen en la carpeta **new-eventpattern** cada dos segundos e intenta añadir cada uno de esos patrones al motor Esper (además de añadir un listener a dicho patrón para que, cuando se detecte, se éste se envíe a una cola llamada "ComplexEventConsumerGlobalVM"). Tras procesarlos, los ficheros se mueven de la carpeta **new-eventpattern** a la carpeta

deployed-eventpattern. Al intentar añadir cada uno de estos patrones, es posible que salte una excepción y el patrón de evento no se añada; esta excepción puede deberse a uno de los siguientes motivos:

- **La sintaxis del fichero no es correcta**: Si el fichero no sigue correctamente la sintaxis EPL, será movido a la carpeta **incorrect-eventpattern**, para que no vuelva a intentar ser añadido.
- **El tipo de evento al que se refiere el patrón no existe**: Si Mule aún no ha recibido ningún evento del tipo al que se refiere el patrón, dicho evento no estará registrado en Esper, por lo que saltará esta excepción. En este caso, mantenemos el fichero con el patrón en la carpeta **new-eventpattern**, para que el patrón se añada al motor Esper una vez ha llegado algún evento del tipo al que se refiere.

El flujo llamado **ActionForEventPatternAdditionToMule** se encarga de consultar cada 2 segundos la carpeta **new-action**, para obtener los ficheros que definen la acción (aquellos con extensión **.action**) que se ejecutará cuando se detecte un determinado patrón. Dichos ficheros contendrán un XML que define un flujo de Mule que será el que se ejecute cuando se detecte el patrón. Todos los ficheros **.action** definen un flujo que hace lo siguiente: crea un mensaje con la acción concreta que se quiere llevar a cabo y lo envía al *topic* **"drone/actions"** de Mosquitto. Si dicho XML tiene un error de sintaxis, saltará una excepción y el no se añadirá el flujo; además, el fichero se moverá a la carpeta **incorrect-action**.

Por último, el flujo **ComplexEventReceptionAndDecisionMaking**, se encarga de recibir cada uno de los patrones de eventos detectados (para lo cual está suscrito a la cola **"ComplexEventConsumerGlobalVM"**) y poner en ejecución el flujo asociado al mismo (el definido en el fichero **.action**). Para que esto sea posible, el patrón de evento tiene que tener el mismo nombre que el flujo con la acción que se quiera ejecutar cuando se detecte. Si se detecta un evento que no tiene un flujo asociado, saltará una

excepción, que simplemente mostrará un mensaje en consola indicando que no se ejecuta ninguna acción.

4.3.2.1 Ficheros de patrones eventos

Los ficheros .epl definen los patrones de eventos que queremos detectar utilizando el lenguaje EPL [66]. Estos ficheros se situarán dentro de la carpeta **new-eventpattern** y su contenido será similar al siguiente:

```
1 @Name('HighPressure')
2 insert into HighPressure
3 select a1.*
4 from pattern [every a1 = PressureEvent(value > 200)]
```

Como puede verse, en primer lugar se indica el nombre que se le dará al patrón (en este caso, **HighPressure**). Tras esto, se define el patrón en cuestión, utilizando uno o varios eventos.

Se han definido 11 **tipos de eventos** (uno por cada tipo de dato que se lee de los sensores) que son los siguientes:

- **GPSEvent**: Contiene las coordenadas y la altura (respecto al punto de despegue) a la que se encuentra el dron.
- **BatteryEvent**: Indica el porcentaje de batería del dron.
- **TemperatureEvent**: Indica el porcentaje de batería del dron.
- **HumidityEvent**: Indica la humedad relativa en tanto por ciento (%).

- **UltrasonicEvent**: Indica la distancia de un posible obstáculo en centímetros.
- **WaterEvent**: Indica si está lloviendo (1) o no (0).
- **HCHOEvent**: Indica la cantidad de partículas volátiles en el aire en ppm.
- **PressureEvent**: Indica la presión atmosférica en Pascales.
- **LuminosityEvent**: Indica la intensidad luminosa en Lux.
- **COEvent**: Indica cantidad de monóxido de carbono en el aire en ppm.
- **UVindexEvent**: Indica el índice UV.

Todos los eventos anteriores tienen los siguientes **atributos**:

- **sensorId**: Identificador del sensor que ha producido el evento. Este identificador es relativo al tipo de sensor.
- **timestamp**: Fecha y hora a la que se ha tomado el dato, en formato Unix.
- **GPSLatitude**: Latitud a la que se ha tomado el dato.
- **GPSLongitude**: Longitud a la que se ha tomado el dato.
- **GPSAltitude**: Altura (relativa al punto de despegue) a la que se ha tomado el dato.
- **value**: Valor tomado por el sensor. Este atributo no está presente en el tipo de dato **GPSEvent**.

4.3.2.2 Ficheros de acciones

En los ficheros .action se definen los flujos asociados a cada uno de los patrones de eventos. Estos ficheros se situarán dentro de la carpeta **new-action** y su contenido será similar al siguiente:

```
1  <?xml version=1.0" encoding="UTF-8"?>
2  <mule xmlns:quartz="http://www.mulesoft.org/schema/mule/quartz"
   ↪  xmlns:mqtt="http://www.mulesoft.org/schema/mule/mqtt"
   ↪  xmlns="http://www.mulesoft.org/schema/mule/core"
   ↪  xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
   ↪  xmlns:spring="http://www.springframework.org/schema/beans"
   ↪  version="EE-3.4.0"
   ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   ↪  xsi:schemaLocation="http://www.springframework.org/schema/beans
   ↪  http://www.springframework.org/schema/beans/spring-beans-current.xsd
   ↪  http://www.mulesoft.org/schema/mule/core
   ↪  http://www.mulesoft.org/schema/mule/core/current/mule.xsd
   ↪  http://www.mulesoft.org/schema/mule/mqtt
   ↪  http://www.mulesoft.org/schema/mule/mqtt/1.0/mule-mqtt.xsd
   ↪  http://www.mulesoft.org/schema/mule/quartz
   ↪  http://www.mulesoft.org/schema/mule/quartz/current/mule-quartz.xsd">
3          <mqtt:config name="MQTTHighPressure" clientId="HighPressure"
   ↪  doc:name="MQTT">
4          <mqtt:connection-pooling-profile
   ↪  initialisationPolicy="INITIALISE_ONE"
   ↪  exhaustedAction="WHEN_EXHAUSTED_GROW"/>
5      </mqtt:config>
6      <flow name="HighPressure">
7          <set-payload value="LEDON 1" doc:name="Set Payload"/>
8          <mqtt:publish config-ref="MQTTHighPressure"
   ↪  topicName="drone/actions" doc:name="MQTT"/>
9      </flow>
10 </mule>
```

En estos ficheros, en primer lugar, se crea una configuración para el componente de MQTT y, a continuación, se define un flujo en el que se crea un mensaje que el componente de MQTT se encarga de publicar en el *topic* **"drone/actions"**.

Lo que cambiará entre los distintos ficheros **.action** serán los siguientes elementos:

- Atributo **name** de *mqtt:config* y atributo **config-ref** de *mqtt:publish*: Su valor será la cadena "MQTT" concatenada con el nombre que del patrón al que se quiera asociar el flujo.
- Atributo **clientId** de *mqtt:config* y atributo **name** de *flow*: Su valor será el nombre que del patrón al que se quiera asociar el flujo.
- Atributo **value** de *set-payload*: Aquí se pondrá la acción que queramos que se ejecute, junto con el nivel de prioridad de la misma.

En cuanto a las acciones que podemos realizar, hemos definido 7 acciones:

- **LAND** *p*: Hace que el dron aterrice en el punto en el que se encuentre.
- **RTL** *p*: El dron vuelve al punto desde el que despegó y aterriza allí.
- **GOTO** *p lat lon alt*: El dron va al punto <*lat*,*lon*>, situándose a la altura indicada en *alt* (en metros).
- **TAKEPHOTO** *p*: Se toma una foto con la cámara instalada en la Raspberry Pi.
- **TAKEVIDEO** *p t*: Se toma un vídeo de *t* segundos con la cámara instalada en la Raspberry Pi.
- **LEDON** *p*: Enciende la tira de LEDs conectada al Arduino.
- **LEDOFF** *p*: Apaga la tira de LEDs conectada al Arduino.

El valor p que va justo a continuación de cada acción es un número entero que define el nivel de prioridad de la acción.

4.3.2.3 Script Python

El script escrito en Python, lleva a cabo 2 tareas: **obtener los datos de los sensores, completarlos y publicarlos en el *topic* "drone/sensors" de Mosquitto** y **obtener los mensajes publicados en el *topic* "drone/actions" y ejecutar las acciones indicadas por los mismos**.

Para llevar a cabo esto, el script, en primer lugar, se conecta al dron (utilizando la librería DroneKit) y al Arduino. Tras esto, crea 2 hilos: uno que lee los datos de los sensores y otro que ejecuta las acciones.

El hilo que lee los datos de los sensores, está continuamente leyendo del puerto USB y, cada JSON que le llega, lo completa con el timestamp y la latitud, longitud y altura leídas del dron. Tras esto, envía el JSON al *topic* **"drone/sensors"**. Los datos sobre la posición GPS del dron, la altura y el nivel de batería, se actualizan cada segundo. Además, en cada una de estas actualizaciones, se envían dos JSON a **"drone/sensors"**: uno con los datos leídos del GPS y otro con el nivel de batería.

El otro hilo, está suscrito al *topic* **"drone/actions"** y, cada vez que llega un mensaje, crea un nuevo hilo para atender dicha acción. De esta forma, es posible ejecutar concurrentemente acciones en los 3 recursos de los que se dispone: dron, cámara y Arduino. Para evitar problemas, existe una variable por cada uno de estos recursos sobre la que se sincronizan las acciones referentes a cada uno de ellos. Así, sólo será posible ejecutar una acción por cada recurso a la vez.

En cuanto a las acciones, éstas **sólo se ejecutarán si tienen un nivel de prioridad igual o superior a la acción de mayor prioridad que ya se haya**

ejecutado. Lo ideal sería que si llega una acción referida a uno de los recursos y hay otra en ejecución con un nivel de prioridad inferior, ésta se interrumpiese para que se ejecutase la que acaba de llegar, sin embargo, como no ha sido posible conseguir esto, se ha implementado esta primera aproximación.

4.4 Reglas definidas

Se han definido 3 niveles de reglas:

- **Relacionadas con el cumplimiento de la normativa vigente**: Éstas son las reglas con mayor nivel de prioridad (3) y se aseguran que el dron cumple con la ley española. Son las siguientes:
  - Si altura $>$ 400 pies (120m) $\rightarrow$ LAND.
  - Si posición GPS dentro de zona restringida $\rightarrow$ LAND.
  - Si intensidad luminosa $<$ 400lux $\rightarrow$ RTL.
- **Basadas en la auto-preservación**: Con estas reglas se intenta que el dron no sufra daños. Su nivel de prioridad es intermedio (2):
  - Si distancia a obstáculo $<$ 4m $\rightarrow$ LEDON.
  - Si hay lluvia $\rightarrow$ LAND .
  - Si nivel HCHO $>$ 10ppm $\rightarrow$ LEDON.
  - Si presión $<$ 91000Pa && humedad $>$ 70 % $\rightarrow$ RTL.
- **Tareas puntuales o periódicas que deben realizarse a lo largo de la ruta**: Éstas reglas tienen el menor nivel de prioridad (1):
  - Si nivel UV $>$ 5 $\rightarrow$ TAKEPHOTO.
  - Si posición GPS dentro de zona $\rightarrow$ TAKEVIDEO 10.

Estas reglas pueden modificarse o ampliarse, modificando los archivos existentes o añadiendo nuevos archivos.

4.5 Diagramas de casos de uso y de secuencia

En primer lugar, es necesario indicar que no se ha creado ningún diagrama de clases, ya que en este proyecto apenas se utilizan clases como tal (únicamente 6, y son utilizadas para enviar eventos a Esper, añadir nuevos patrones de eventos y añadir nuevos flujos asociados a patrones de eventos). La mayor parte del trabajo se hace a través de componentes de Mule, el script en Python y el sketch de Arduino. Teniendo esto en cuenta, se ha decidido crear un diagrama de casos de uso y, a continuación, explicar cada uno de estos casos mediante diagramas de secuencia.

4.5.1 Diagrama de casos de uso

Como actores en en los casos de uso se han elegido elementos del propio sistema, ya que no existe ninguna persona que interactúe directamente con el sistema en funcionamiento. Concretamente, se han seleccionado como actores al Script en Python, la aplicación Mule y el motor Esper.

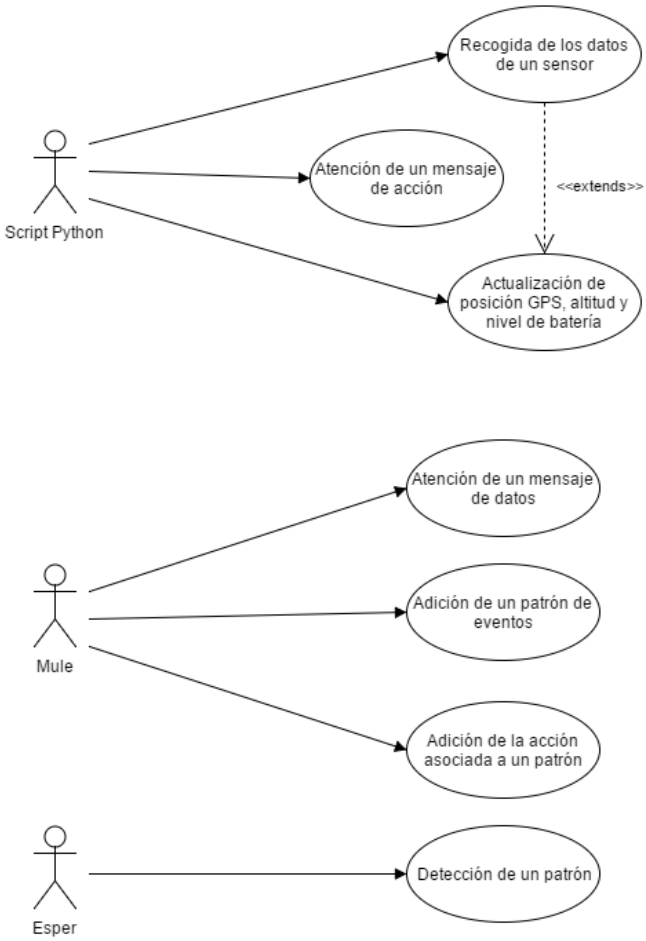

Figura 4.21: Casos de uso del sistema.

4.5.2 Diagramas de secuencia

Aunque el comportamiento de estos casos de uso ya ha sido explicado en las subsecciones anteriores, aquí se verá en forma de diagramas de secuencia, lo que puede ayudar a comprenderlo más fácilmente. No se utilizarán clases, ya que se ha explicado que apenas se utilizan, sino que se representará el **paso de mensajes entre los distintos componentes del sistema**: Script Python, Mule, Esper, Mosquitto, Arduino, Dron y Pi NoIR. Esta forma es la que se ha considerado más adecuada para que quede claro el funcionamiento de cada uno de los casos de uso.

4.5.2.1 Recogida de los datos de un sensor

Este diagrama representa la lectura de valores de cada uno de los sensores y su envío a la Raspberry a través del puerto USB, donde se completará con los datos del GPS y la altitud y se publicará en el *topic* "drone/sensors".

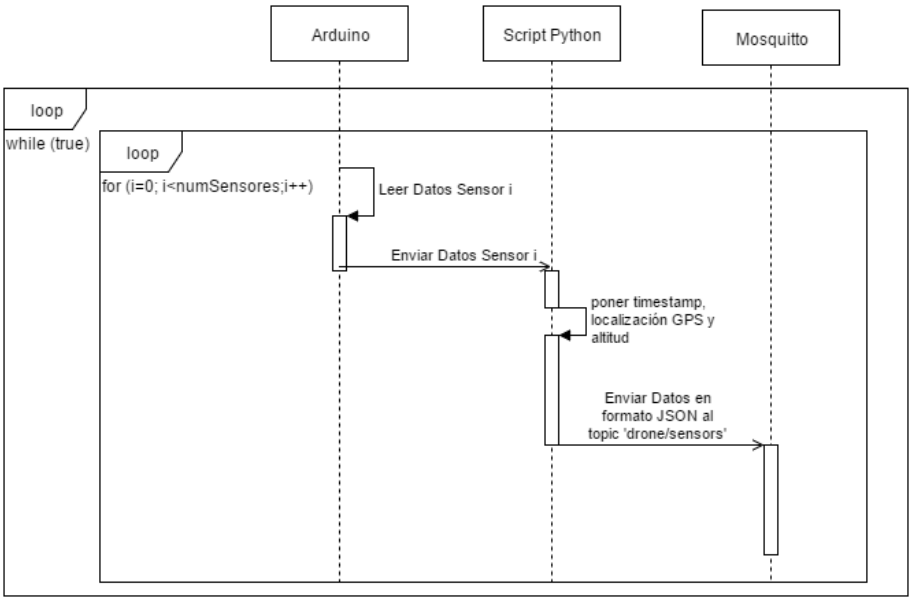

Figura 4.22: Recogida de datos de sensores.

4.5.2.2 Atención de un mensaje de acción

Cuando se recibe un mensaje de acción, en primer lugar se comprueba el nivel de prioridad del mismo. Si es el adecuado, se crea un nuevo hilo para atender dicha acción. Este nuevo hilo se sincronizará sobre la variable correspondiente y, tras ello, se completará la acción.

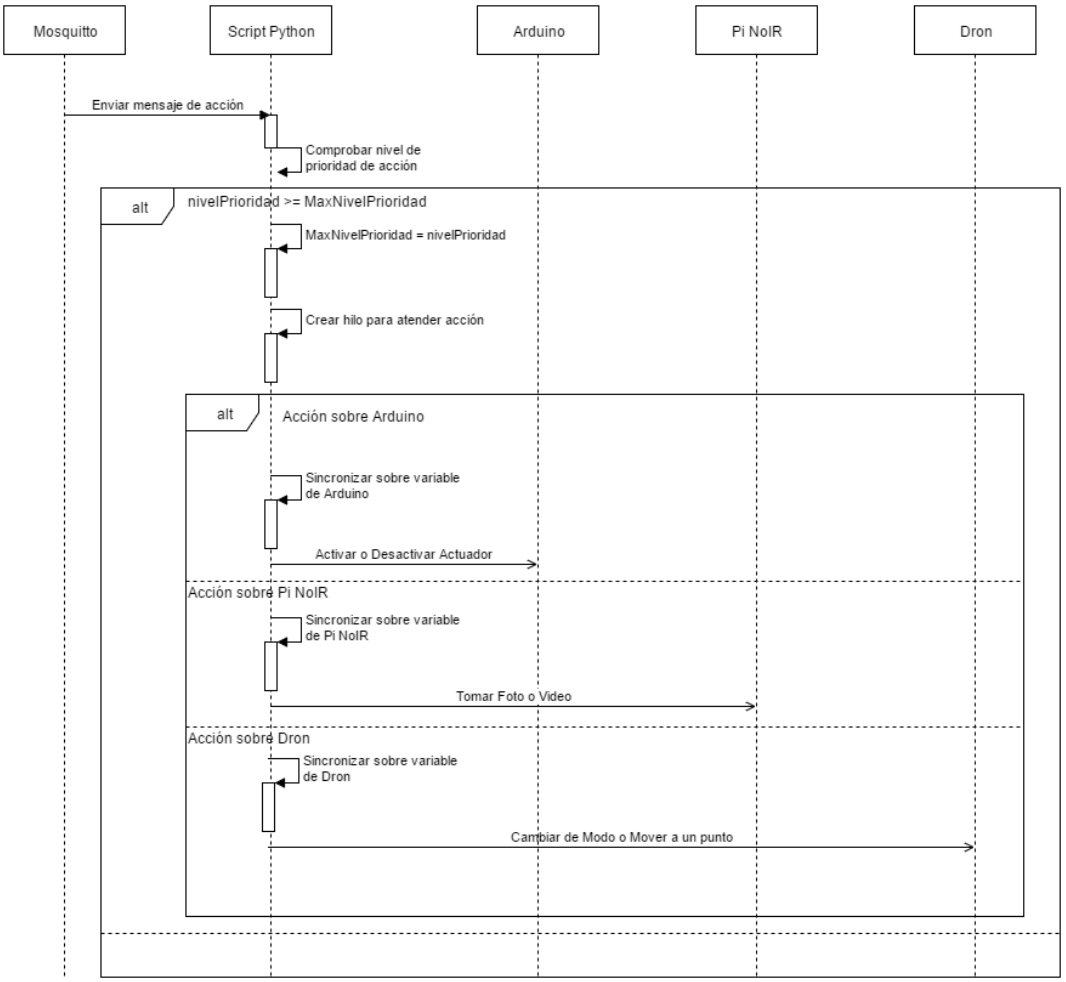

Figura 4.23: Atención de un mensaje de acción.

4.5.2.3 Actualización de posición GPS, altitud y nivel de batería

Cada segundo, se actualiza la posición GPS, altitud y nivel de batería del dron en un hilo del script Python. Cada vez que esto se lleva a cado, se envían dos mensajes al *topic* "drone/sensors", que contienen la posición GPS y el nivel de batería, ambos en formato JSON.

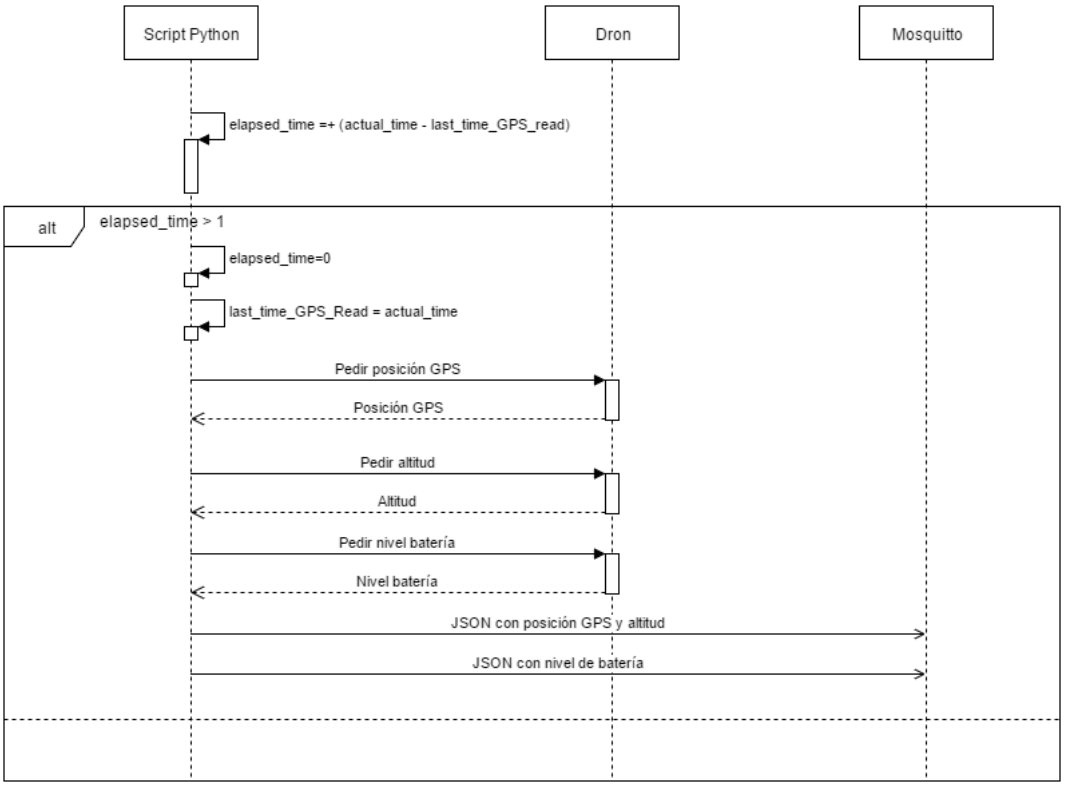

Figura 4.24: Actualización de posición GPS, altitud y nivel de batería.

4.5.2.4 Atención de un mensaje de datos

Cada vez que en el *topic* "drone/sensors" se recibe un mensaje, este se envía a Mule, que se encarga de convertirlo al tipo Map de Java y enviarlo a Esper como un nuevo evento.

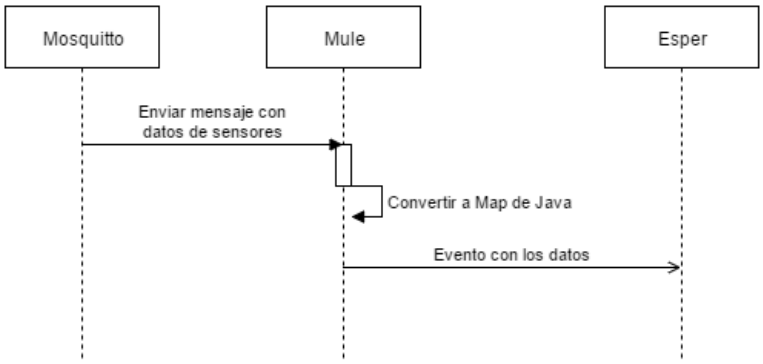

Figura 4.25: Atención de un mensaje con datos de un sensor.

4.5.2.5 Adición de un patrón de eventos

Por cada fichero .epl que exista en la carpeta **new-eventpattern**, Mule se encargará de leerlo, enviarlo a Mule para que lo añada a su motor y, tras esto, moverlo a la carpeta **deployed-eventpattern**.

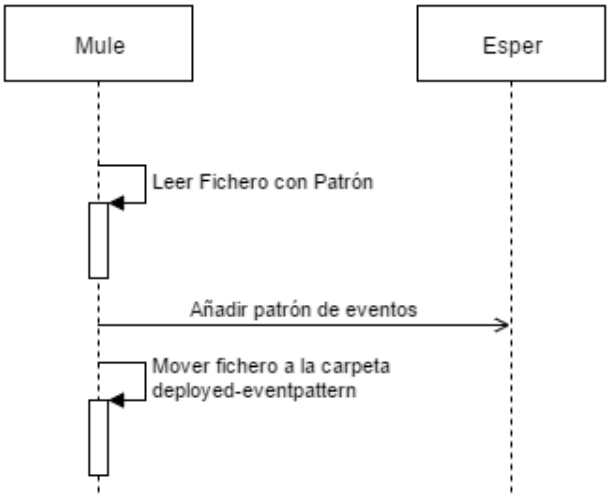

Figura 4.26: Adición de un nuevo patrón de eventos.

4.5.2.6 Adición de la acción asociada a un patrón

Por cada fichero .action que exista en la carpeta **new-action**, Mule se encargará de leerlo, convertirlo al tipo Map, añadir el nuevo flujo y, tras esto, mover el fichero a la carpeta **deployed-action**.

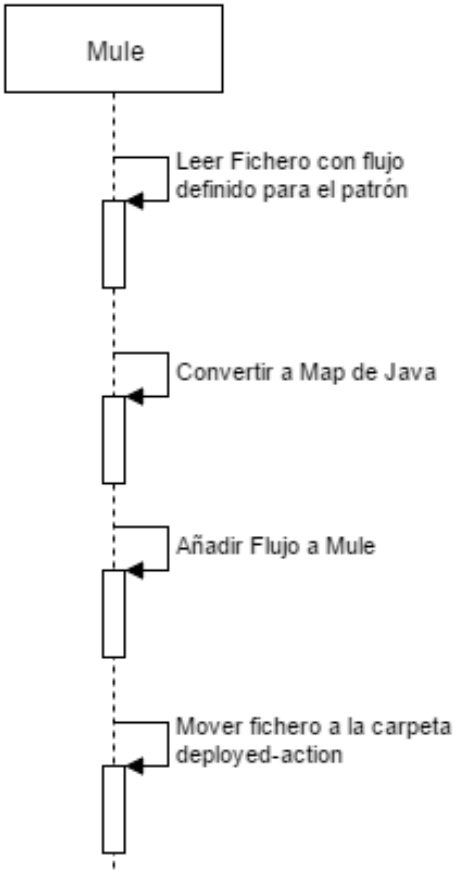

Figura 4.27: Adición de una acción en respuesta a un patrón.

4.5.2.7 Detección de un patrón

Cada vez que Esper detecta un patrón de eventos, envía un evento de patrón a Mule y éste pone en ejecución el flujo asociado a dicho patrón. En este flujo, lo que se hará será enviar un mensaje al *topic* "drone/actions" con la acción que se quiera llevar a cabo.

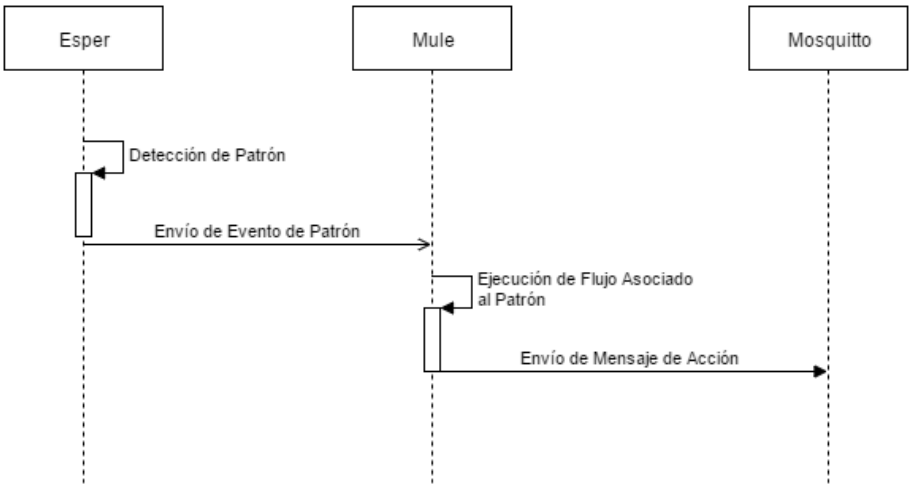

Figura 4.28: Detección de un patrón de eventos.

Capítulo 5

Validación tecnológica

En este capítulo se van a documentar las **pruebas que se han realizado para asegurar que el sistema funciona correctamente**. Para ello, se han utilizado escenarios simples, pero suficientemente válidos para comprobar que se obtienen los resultados deseados. En algunas de las pruebas no se utiliza el dron real, sino un simulador llamado **SITL** (*Software In The Loop*), que proporciona un entorno que emula el comportamiento de un dron real, de forma que es posible conectarse a él utilizando Mission Planner y ver cómo se mueve, el modo en el que está...

Para cada una de las pruebas se explicará qué es lo que se va a hacer, cuál es su objetivo, cómo llevarla a cabo y los resultados obtenidos. Además, existirá un vídeo donde podrá verse la ejecución de la misma. Estos vídeos están en la carpeta **'Validación/Vídeos'** y se recomienda utilizar el reproductor **VLC Media Player** para verlos, ya que otros reproductores es posible que no sean capaces de reproducirlos correctamente.

5.1 Prueba de ejecución de una sola acción

5.1.1 Explicación y objetivos

En esta prueba se hará saltar una regla de nivel 1. Para ello, se modificará un poco la aplicación Mule, para que no ejecute el script Python, el cual se ejecutará directamente mediante consola. Por otra parte, en vez de utilizar un dron real, se usará el simulador **SITL**, por lo que también se modificará dicho script, para que se conecte al simulador. Como en la Raspberry Pi no es posible ejecutar el simulador (ya que solo está disponible para arquitecturas x86 y la Raspberry Pi es del tipo ARM), la prueba se realizará en un portátil con Ubuntu 14.04.

De esta forma, se demostrará que el sistema detecta un patrón de eventos y envía a Mosquitto el mensaje adecuado. El patrón que se quiere detectar es simple: que haya un nivel de humedad superior al 40 %; en este caso, se enviará al *topic* "drone/actions" el mensaje "LEDON 1", es decir, se utilizará una regla del menor nivel (1) que haga encenderse a la tira de LEDs.

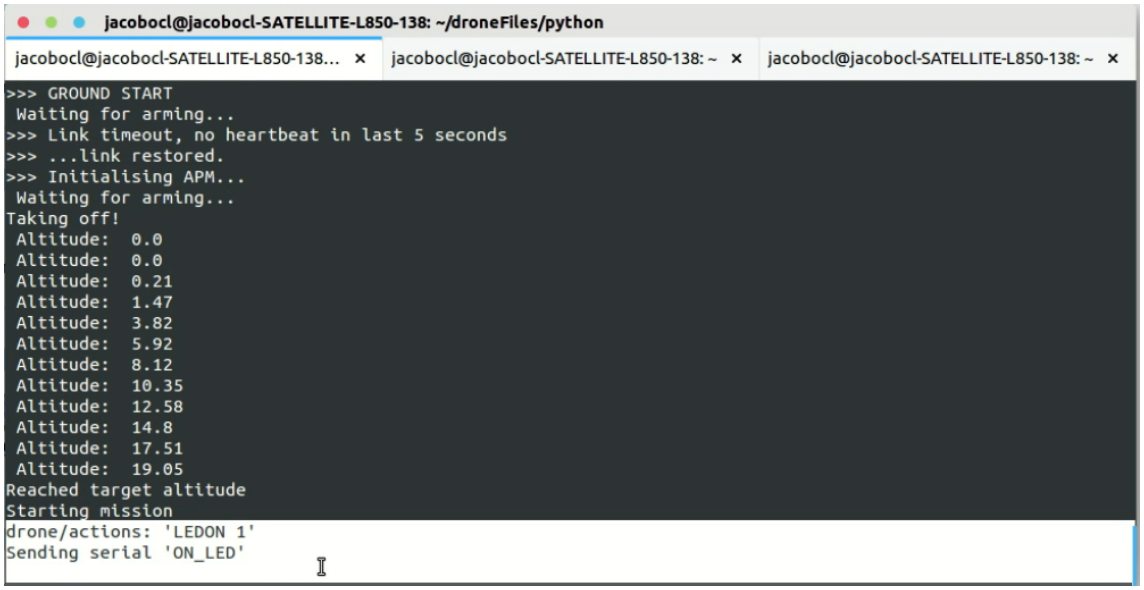

Figura 5.1: Captura de la prueba en ejecución.

5.1.2 Recursos utilizados

- Ordenador con el sistema operativo Ubuntu 14.04, en el que está instalado el IDE Mule Studio, el IDE de Arduino, Mosquitto, SITL, DroneKit y Python.
- Placa Arduino Uno.
- Grove Shield.
- Conjunto de sensores y actuadores conectados al Arduino.

5.1.3 Ejecución de la prueba

Para ejecutar la prueba, en primer lugar, será necesario tener los sensores y actuadores conectados al Arduino utilizando la Grove Shield, como se describe en la Sección 4.2. Tras esto, se conectará el Arduino al portátil y se cargará el sketch utilizado por el sistema (Sección A.1). Por último, se tendrá que incluir el patrón de eventos **HighHumidity.epl** (situado en la carpeta 'Validación/Patrones') y el archivo con la acción asociada **HighHumidity.action** (situado en la carpeta 'Validación/Acciones') en las carpetas **new-eventpattern** y **new-action**, respectivamente.

Una vez está listo todo esto, en primer lugar, será necesario lanzar el simulador SITL; tras esto, se ejecutará el script Python modificado (**droneActionsAndData_-debug.py**, situado en la carpeta 'Validación/Código'. Para ejecutarlo, es necesario tener en la misma carpeta los archivos **QLock.py** y **switch.py**, ya que son usados por el script) y, por último, la aplicación Mule. Ahora, se tendrá que hacer saltar la regla que se ha cargado, para ello, simplemente se cubrirá con la mano el sensor de temperatura y humedad. Una vez se ha hecho esto, se verá cómo salta la regla en la consola en la que se ha ejecutado el script Python. Todo esto se muestra en el vídeo **VTec1Regla.mov**.

5.1.4 Resultados obtenidos

En el vídeo puede verse que, efectivamente, la regla salta cuando se cumplen las condiciones de humedad definidas en el patrón, por tanto, se ha demostrado que el sistema detecta patrones de eventos y ejecuta las acciones asociadas a los mismos.

5.2 Prueba de ejecución de dos acciones de distinto nivel

5.2.1 Explicación y objetivos

En este caso, se demostrará que es posible ejecutar más de una regla. Estas reglas serán de diferente nivel, de forma que se verá cómo, tras ejecutar una regla de un determinado nivel, no hay ningún problema en ejecutar otra de un nivel superior. Al igual que en la prueba anterior, se utilizará la aplicación Mule modificada para que no ejecute el script Python y el simulador del dron (**SITL**).

Con esta prueba se quiere demostrar que el sistema puede funcionar con varias reglas, no únicamente con una. Las dos reglas que se utilizarán son las siguientes: que cuando haya un nivel de humedad superior al 40 % se envíe el mensaje "LEDON 1" y, que cuando haya una cantidad de luz superior a 1000lux, se envíe el mensaje "LEDOFF 2" (regla de nivel 2 que hace apagarse a la tira de LEDs).

5.2.2 Recursos utilizados

- Ordenador con el sistema operativo Ubuntu 14.04, en el que está instalado el IDE Mule Studio, el IDE de Arduino, Mosquitto, SITL, DroneKit y Python.

- Placa Arduino Uno.
- Grove Shield.
- Conjunto de sensores y actuadores conectados al Arduino.
- Flash del móvil, para utilizarlo como linterna.

5.2.3 Ejecución de la prueba

Como en la prueba anterior, se conectarán los sensores y actuadores al Arduino utilizando la Grove Shield y se cargará el sketch en el Arduino (si esto ya se ha hecho, no es necesario volver a hacerlo). Tras esto, se incluirán los patrones **HighHumidity.epl** y **HighLuminosity.epl** (situados en la carpeta 'Validación/Patrones') dentro de la carpeta **new-eventpattern** y las acciones **HighHumidity.action** y **HighLuminosity.action** (situados en la carpeta 'Validación/Acciones') dentro de la carpeta **new-action**.

A continuación, se lanzará el simulador, el script Python modificado (**droneActionsAndData_debug.py**, situado en la carpeta 'Validación/Código'. Para ejecutarlo, es necesario tener en la misma carpeta los archivos **QLock.py** y **switch.py**, ya que son usados por el script) y la aplicación Mule, en ese orden. Para hacer saltar la primera regla, se cubrirá con la mano el sensor de temperatura y humedad. Una vez ha saltado, se iluminará el sensor de luminosidad con el flash del móvil, para que salte la segunda regla. En el vídeo **VTec2Reglas.mov** se muestra la ejecución de esta prueba y puede verse cómo saltan las dos reglas.

5.2.4 Resultados obtenidos

Los resultados obtenidos son positivos, ya que puede verse cómo ambos patrones de eventos se detectan y se lanza la acción asociada a cada uno de ellos. Así, se demuestra que el sistema puede funcionar con más de una regla.

5.3 Comprobación de que tras ejecutar una regla de un nivel, no se ejecutan las de niveles inferiores

5.3.1 Explicación y objetivos

En esta prueba se intentarán ejecutar reglas de tres niveles distintos, de forma que, cuando se ejecute una de un determinado nivel, no sea posible ejecutar las de niveles inferiores que lleguen a continuación, pero sí las del mismo nivel o niveles superiores. Como en las pruebas anteriores, se utilizará la aplicación Mule sin el flujo que lanza el script Python y el simulador **SITL**.

El objetivo de esta prueba es comprobar que el sistema de niveles de reglas funciona adecuadamente. Para llevar a cabo la prueba, se utilizarán estas 3 reglas: Enviar el mensaje "LEDON 1" cuando haya un nivel de humedad superior al 40 %, enviar el mensaje "LEDOFF 2" cuando haya una luminosidad superior a 1000lux y enviar el mensaje "TAKEPHOTO 3" (regla de nivel 3 que hace que la cámara tome una foto) cuando haya un objeto a menos de 20cm del sensor de ultrasonidos. En cuanto a esta última regla, como no se está utilizando la Raspberry Pi, no se tiene acceso a la cámara, por lo que en el script Python modificado el código de esta acción se ha modificado para que simplemente muestre un mensaje.

5.3.2 Recursos utilizados

- Ordenador con el sistema operativo Ubuntu 14.04, en el que está instalado el IDE Mule Studio, el IDE de Arduino, Mosquitto, SITL, DroneKit y Python.
- Placa Arduino Uno.
- Grove Shield.
- Conjunto de sensores y actuadores conectados al Arduino.
- Flash del móvil, para utilizarlo como linterna.

5.3.3 Ejecución de la prueba

Como en la prueba anterior, se conectarán los sensores y actuadores al Arduino utilizando la Grove Shield y se cargará el sketch en el Arduino (si esto ya se ha hecho, no es necesario volver a hacerlo). Tras esto, se incluirán los patrones **HighHumidity.epl**, **HighLuminosity.epl** y **ObstacleComing.epl** (situados en la carpeta 'Validación/Patrones') dentro de la carpeta **new-eventpattern** y las acciones **HighHumidity.action**, **HighLuminosity.action** y **ObstacleComing.action** (situados en la carpeta 'Validación/Acciones') dentro de la carpeta **new-action**.

A continuación, se lanzará el simulador, el script Python modificado (**droneActionsAndData_debug.py**, situado en la carpeta 'Validación/Código'. Para ejecutarlo, es necesario tener en la misma carpeta los archivos **QLock.py** y **switch.py**, ya que son usados por el script) y la aplicación Mule, en ese orden. Para hacer saltar la regla de primer nivel, se cubrirá con la mano el sensor de temperatura y humedad. Una vez ha saltado, se iluminará el sensor de luminosidad con el flash del móvil, para que salte la regla de segundo nivel. Ahora, se intentará volver a ejecutar la regla de nivel 1, pero no será posible porque se ya se ha ejecutado una de nivel 2; para ello, se cubrirá con la mano

el sensor de temperatura y humedad. Por último, se acercará el sensor de ultrasonidos a una superficie para ver cómo salta la regla de mayor nivel. Esta prueba la puede verse en el vídeo **VTec3Reglas.mov**, donde se muestra cómo se ejecutan las reglas de la forma que se espera.

5.3.4 Resultados obtenidos

Se han obtenido los resultados esperados, por tanto, queda demostrado que el sistema de niveles de reglas funciona adecuadamente.

5.4 Ejecución de acciones del dron en el simulador SITL

5.4.1 Explicación y objetivos

En este caso, se ejecutarán una serie de acciones que implican el uso del dron. Para ello, se enviarán directamente desde consola los mensajes que se enviarían cuando se detectase un determinado patrón al *topic* **"drone/actions"**. Con estos mensajes, se simulará al detección de reglas de distinto nivel que, como acción asociada, tienen un cambio de modo o movimiento del dron. Como dron, se utilizará el simulador **SITL**, ya que esto permite ejecutar las pruebas de una forma más cómoda, además de que así puede verse toda la información del mismo en APM Planner.

Con esta prueba se intentará demostrar que las acciones sobre el dron se ejecutan correctamente y que, por lo tanto, el sistema puede manejarlo cuando sea necesario. Las acciones que se enviarán serán las siguientes: **"GOTO 2 39.479034 -6.342739 40"** (regla de nivel 2 que hace que el dron vaya al punto <39.479034,-6.342739> a una altura

de 40m), **"LAND 1"** (regla de nivel 1 que hace que el dron cambie al modo LAND, es decir, que aterrice en el punto que esté) y **"LAND 2"** (regla de nivel 2 que hace que el dron cambie al modo LAND).

5.4.2 Recursos utilizados

- Ordenador con el sistema operativo Ubuntu 14.04, en el que está instalado el IDE de Arduino, Mosquitto, SITL, DroneKit, Python y APM Planner.
- Placa Arduino Uno.
- Grove Shield.
- Conjunto de sensores y actuadores conectados al Arduino.

5.4.3 Ejecución de la prueba

Como en la prueba anterior, se conectarán los sensores y actuadores al Arduino utilizando la Grove Shield y se cargará el sketch en el Arduino (si esto ya se ha hecho, no es necesario volver a hacerlo). En este caso, la aplicación Mule no es necesaria, ya que seremos nosotros los que mandaremos los mensajes de acción.

Una vez se ha hecho esto, se lanzará el simulador, se abrirá APM Planner para cargar una ruta en el dron, se desconectará APM Planner del dron, se lanzará el script Python (**droneActionsAndData_debug.py**) y volverá a conectarse el dron con APM Planner, pero esta vez, utilizando el puerto **5763**, en vez del **5760**. Todo esto se explica en la Sección B.5.

Ahora que está todo listo, se pueden enviar los mensajes a Mosquitto. Para hacer esto, se utilizará el comando **mosquitto_pub**, indicando el equipo donde está Mosquit-

to (localhost) y el topic en el que se quiere publicar (drone/actions). La primera acción que se enviará, será **"GOTO 2 39.479034 -6.342739 40"**, para mover el dron a un punto. A continuación, se enviará una acción que no debe ejecutarse, ya que es de nivel 1: **"LAND 1"**. Por último, se enviará la misma acción que antes, pero cambiando su nivel de prioridad al nivel 2, para que pueda ejecutarse.

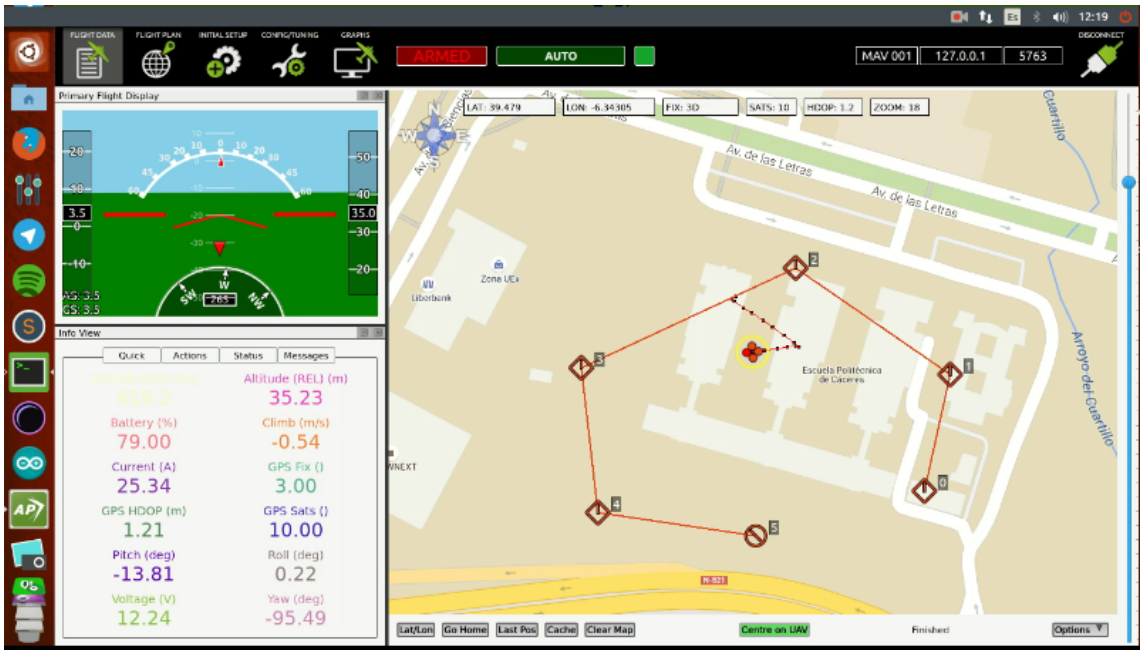

Figura 5.2: Dron moviéndose al punto <39.479034,-6.342739> en APM Planner.

5.4.4 Recursos utilizados

- Ordenador con el sistema operativo Ubuntu 14.04, en el que está instalado el IDE Mule Studio, el IDE de Arduino, Mosquitto, SITL, DroneKit y Python.
- Placa Arduino Uno.
- Grove Shield.
- Conjunto de sensores y actuadores conectados al Arduino.

En el vídeo **VTecSimulador.mov**, puede verse la ejecución de todo lo mencio-

nado anteriormente. Es preciso indicar que APM Planner no funciona demasiado bien con el simulador SITL, por lo que, en ciertas ocasiones, se cierra o no muestra los puntos por los que tiene que pasar el dron.

5.4.5 Resultados obtenidos

En la prueba se ve cómo el dron responde adecuadamente a los comandos que se le envían, ya sean estos de movimiento o de cambio de modo. Además, el script Python maneja sin problema los distintos niveles de reglas. Así, puede afirmarse que esta prueba también ha sido satisfactoria y que el sistema controla correctamente al dron.

5.5 Prueba en vuelo en un dron real

5.5.1 Explicación y objetivos

En esta prueba, se ejecutará el sistema en un entorno real, es decir, utilizando un dron real y con el sistema de control en una Raspberry Pi que estará conectada a dicho dron. Se cargará una regla sencilla, para comprobar que se ejecuta cuando se cumplen las condiciones definidas en la misma.

El objetivo de esta prueba es demostrar que el sistema desarrollado funciona en un entorno real. En concreto, la regla que se utilizará consistirá en que, cuando se alcance una altura superior a 15m, el dron vaya a un punto.

5.5.2 Recursos utilizados

- Ordenador con el sistema operativo Ubuntu 14.04, en el que está instalado APM Planner.
- Raspberry Pi con el sistema desarrollado en este trabajo.
- Batería para alimentar la Raspberry Pi.
- Placa Arduino Uno.
- Grove Shield.
- Conjunto de sensores y actuadores conectados al Arduino.
- Dron que soporte el protocolo MAVLink.

5.5.3 Ejecución de la prueba

Al igual que en el resto de pruebas, se conectarán los sensores y actuadores al Arduino utilizando la Grove Shield y se cargará el sketch en el Arduino (si ya se ha hecho esto, no es necesario volver a hacerlo). Tras esto, se cargará la regla en la Raspberry Pi; para ello, se copiará el patrón **HighAltitude.epl** (situado en la carpeta 'Validación/Patrones') dentro de la carpeta **new-eventpattern** y la acción **HighAltitude.action** (situada en la carpeta 'Validación/Acciones') dentro de la carpeta **new-action**. Estas carpetas se encuentran dentro de 'droneFiles', la cual, normalmente, estará en el directorio /home/pi.

Una vez se ha hecho lo anterior, solo quedaría conectar el dron al ordenador para cargarle una misión con APM Planner (en la que pondremos un punto que esté a una altura superior a los 15m) y, una vez cargada, conectar el dron a la Raspberry y comenzar la misión. Así, cuando el dron vaya subiendo para alcanzar dicho punto, una

vez sobrepase los 15m, se activará la regla e irá hacia el punto definido en **HighAltitude.action**.

Figura 5.3: Dron junto con el sistema de monitorización del entorno y control.

5.5.4 Resultados obtenidos

En este caso, los resultados no son positivos ya que el dron se desestabilizó cuando estaba despegando debido al peso que la caja que contiene la Raspberry Pi, el Arduino, la batería y los sensores ejerce sobre uno de los extremos del dron. Al desestabilizarse, el dron cayó y se rompió una hélice, por lo que la prueba no pudo llevarse a cabo. Debido

al *frame* que utiliza el dron, no es posible situar la caja en otro lugar. Así, aunque según los resultados obtenidos en el simulador, el sistema debería funcionar, no ha sido posible demostrarlo. Para asegurarnos de que realmente funciona haremos otra prueba.

5.6 Prueba en tierra en un dron real

5.6.1 Explicación y objetivos

En esta última prueba se intentará demostrar que el sistema funciona en un entorno real, aunque sin poner el dron en vuelo, ya que, como se ha visto en la prueba anterior, la caja con los dispositivos desestabiliza al dron y no es posible situarla en otro lugar ya que el *frame* del que se dispone no lo permite.

Al igual que en el caso anterior, se ejecutará el sistema utilizando un dron real, con el sistema de control en una Raspberry Pi que estará conectada a dicho dron. Se cargará una regla sencilla, para comprobar que se ejecuta cuando se cumplen las condiciones definidas en la misma.

El objetivo de esta prueba es demostrar que, si se consigue centrar el peso, el sistema podría funciona en un dron en vuelo, ya que funcionaría en un dron real. En concreto, la regla que se utilizará consistirá en que, cuando haya un nivel de humedad superior al 15 %, el dron se ponga en modo LAND.

5.6.2 Recursos utilizados

- Ordenador con el sistema operativo Ubuntu 14.04, en el que está instalado APM Planner.

- Raspberry Pi con el sistema desarrollado en este trabajo.
- Batería para alimentar la Raspberry Pi.
- Placa Arduino Uno.
- Grove Shield.
- Conjunto de sensores y actuadores conectados al Arduino.
- Dron que soporte el protocolo MAVLink sin hélices, para evitar que despegue.

5.6.3 Ejecución de la prueba

Como en el resto de pruebas, se conectarán los sensores y actuadores al Arduino utilizando la Grove Shield y se cargará el sketch en el Arduino (si ya se ha hecho esto, no es necesario volver a hacerlo). Tras esto, se cargará la regla en la Raspberry Pi; para ello, se copiará el patrón **Humidity15.epl** (situado en la carpeta 'Validación/Patrones') dentro de la carpeta **new-eventpattern** y la acción **Humidity15.action** (situada en la carpeta 'Validación/Acciones') dentro de la carpeta **new-action**. Estas carpetas se encuentran dentro de 'droneFiles', la cual, normalmente, estará en el directorio /home/pi.

Una vez se ha hecho lo anterior, solo quedaría conectar el dron al ordenador para cargarle una misión con APM Planner y, una vez cargada, conectar el dron a la Raspberry y comenzar la misión. A continuación, cuando el dron arranque los motores para comenzar la misión, taparemos con la mano el sensor de temperatura y humedad para que el dron se ponga en modo LAND, lo que hará que los motores se paren (ya que el dron estará parado en el punto de despegue).

En el vídeo **VTecRealSinVuelo.mov** se ve cómo, efectivamente, tras llegar la humedad a un valor superior al 15 %, se detecta el evento y los motores se paran.

5.6.4 Resultados obtenidos

Tras la ejecución de esta prueba, se han obtenido los resultados esperados, por tanto, puede afirmarse que el sistema puede trabajar en un entorno real, solo habría que buscar la manera de equilibrar el peso.

Capítulo 6

Trabajos futuros

Se ha llegado a este capítulo habiendo explicado en qué consiste el sistema desarrollado y asegurándose de que funciona como se espera de él. Aun así, hay ciertos objetivos que no se han conseguido o aspectos que podrían mejorarse. En este capítulo se explicará qué es lo que podría mejorarse, así como algunos proyectos que podrían tener este trabajo como base para llevarse a cabo.

6.1 Niveles de reglas

Como se comentó en el Capítulo 4, la forma que se utiliza para manejar los distintos niveles de reglas no es la que idealmente se desearía. Lo que se hace es dejar de ejecutar las acciones con un nivel de prioridad inferior a la acción con mayor nivel de prioridad que se haya ejecutado durante el vuelo. Sin embargo, lo ideal sería que, cuando llegase una acción, se evaluase su nivel de prioridad y el recurso que tiene que utilizar (dron, cámara Pi NoIR o actuador conectado al Arduino) y, en caso de que haya otra acción ejecutándose sobre ese recurso, si esta última es de un nivel inferior, se debería detener y ejecutar la nueva acción.

Por otra parte, la forma en que se hace actualmente también podría mejorarse, ya que, en vez de no ejecutar la acción asociada a un patrón, podría buscarse una forma de, directamente, no detectar dicho patrón, liberando así de trabajo tanto al motor CEP como, en menor medida, al script Python.

6.2 DSL que genere el sistema adaptado a distintos tipos de drones, sensores y actuadores

El sistema que se ha desarrollado está pensado para funcionar en un entorno en el que cada uno de los sensores y actuadores estén conectados en un puerto concreto del Arduino (o de la Raspberry Pi, en caso de la cámara). Además, tanto el dron como el Arduino tienen que conectarse de una determinada manera a la Raspberry para que todo funcione correctamente. Aunque es posible (y relativamente sencillo) cambiar esta configuración, dicho cambio obliga a modificar manualmente el código del script Python o del sketch de Arduino, en función de lo que se quiera modificar.

Ante este panorama, sería mucho más sencillo crear un DSL (*Domain-Specific Language*) gráfico (ya que sería más cómodo que uno textual) en el que se pudiese especificar qué sensores y actuadores se van a utilizar y a qué pin van a conectarse cada uno de ellos, si se va a utilizar la cámara Pi NoIR u otro tipo de cámara (o, incluso, ninguna), cómo va a conectarse el dron y el Arduino a la Raspberry... De esta forma, el código del sistema se generaría automáticamente para la configuración específica que se utilizase.

Por otra parte, este DSL podría extenderse aún más y permitir la definición de las reglas que vayan a utilizarse, es decir, se definirían los patrones de eventos, junto con la acción asocida a cada uno de ellos y su nivel de prioridad, de forma que se generasen los ficheros .epl y .action, que solo tendrían que ponerse en la carpeta correspondiente.

6.3 Servicio web para añadir nuevas reglas en tiempo de ejecución

Otro posible proyecto futuro que podría realizarse a partir del presente trabajo, sería añadir un servicio web que permitiese añadir nuevas reglas mientras el dron está en vuelo. En el estado actual del trabajo, ya es posible añadir nuevas reglas mientras el sistema está en ejecución (recordemos que las carpetas con los patrones y las acciones asociadas a estos se comprueban cada 2 segundos y, si existen nuevos archivos, se añaden al sistema), por lo que la idea sería añadir un módem USB 3G [67] y crear un servicio web al que pudiesen enviarse estos archivos para que fuesen añadidos mientras el dron está en vuelo.

Además, aprovechando la conexión a Internet, podría crearse otro servicio web que publicase los datos de los eventos detectados, los datos obtenidos de cada sensor y el estado del dron (posición, altura, nivel de batería, modo...) en un servidor. Esto sería muy interesante para comprobar el funcionamiento del sistema en tiempo real y, sobre todo, porque podría saberse el estado del dron, ya que, al conectarse la Raspberry Pi al dron, se pierde la posibilidad de obtener esa información en una CGS mediante telemetría, puesto que sólo es posible crear una conexión con el dron a la vez.

6.4 Montar un sistema de reglas de nivel superior, para coordinar varios drones

Aprovechando el sistema que se ha desarrollado, podría desarrollarse otro que utilizase la información que se produce en un dron (tanto de los propios sensores como de los patrones de eventos que se detectan) en un sistema similar, pero de nivel superior. Así, se podría manejar un conjunto de varios drones con este sistema, es decir, con la

información de cada uno de los drones, se podría montar otro sistema de reglas que se encargase de coordinar a todos esos drones.

Así, si, por ejemplo, se tiene un sistema con 7 drones, que se encargan de monitorizar un área con 3 drones a la vez, manteniendo otros 3 a la espera de que a los que están en vuelo se les acabe la batería para relevarlos y otro más en reserva por si alguno queda fuera de servicio, podría coordinarse todo esto con un solo sistema. Esto sería útil en situaciones como las siguientes:

- **Un dron detecta lluvia en la zona en que está** y el sistema le indica a los demás que no se acerquen a dicha zona.
- **Uno de los drones que está volando deja de funcionar** (por ejemplo, porque es derribado). En este caso, el sistema mandaría a uno de los que está en espera a monitorizar esa zona y se pondría en espera el que está en reserva.
- **Varios drones se encuentran en la monitorizando la misma zona**. Si esto ocurriese, el sistema podría detectarlo y enviarlos a otras zonas, para no tener información repetida.

Capítulo 7

Conclusiones

El desarrollo de este TFG ha sido una tarea muy interesante que me ha supuesto enfrentarme a nuevos retos, por lo que estoy bastante satisfecho con el resultado obtenido. Si bien es cierto que la librería DroneKit ha facilitado mucho la tarea de interactuar con el dron, ha sido muy gratificante combinar el uso de dicha librería con el resto del sistema (Esper, Mule, Arduino...) para conseguir un resultado final que cumple en gran medida con los objetivos iniciales. Por supuesto, el sistema podría mejorarse, ya que, por ejemplo, no es muy flexible a la hora de ponerlo en marcha, puesto que hay que seguir unos pasos específicos para que funcione correctamente, además, también sería interesante que cuando se detecta un obstáculo en la ruta, ésta se recalculase para evitarlo. Sin embargo, creo que el sistema que se ha conseguido al final de este proyecto puede ser una buena base para conseguir estos objetivos y los planteados en el Capítulo 6.

Lo que más me atrajo inicialmente para decantarme por este proyecto fue el hecho de que se trata de una tecnología nueva (hará 2-3 años que los drones han aparecido en el entorno civil), por lo que me gustó la idea de investigar en un entorno desconocido. Además, intentar aportar algo a unos dispositivos que actualmente están en boca todo el mundo, también es un gran aliciente.

Como conclusión final, puedo decir que he aprendido mucho durante el desarrollo de este TFG, no sólo sobre drones, sino también en otros aspectos como la planificación de tareas o la búsqueda de distintas soluciones para encontrar la que más se adecúe al problema que hay que resolver. También tengo que destacar que ha sido muy apasionante ver cómo el sistema iba evolucionando, desde simplemente recoger en la Raspberry Pi los datos que producen los sensores conectados al Arduino, hasta utilizar estos y otros datos para definir patrones de eventos que provocan el control del dron o de unos actuadores cuando se detectan. Cada uno de estos pequeños hitos me ha animado a seguir trabajando para conseguir la solución que finalmente se presenta en este TFG, por lo que me siento muy satisfecho con el resultado final y todo lo que me ha aportado este trabajo.

Capítulo 8

Agradecimientos

Para terminar este trabajo, me gustaría darle las gracias a todas las personas que me han acompañado durante estos 4 años y cuyo apoyo ha hecho posible que haya llegado hasta aquí. Gracias a ellos este camino ha sido mucho más sencillo y ha estado lleno de buenos momentos.

En primer lugar, debo dar las gracias al grupo Quercus, por permitirme colaborar con ellos durante estos últimos dos años y por todo lo que he aprendido durante este tiempo. Especialmente, a Juan Hernández y a José Enrique Moguel por ofrecerme realizar este proyecto y su continuo apoyo y dedicación. Igualmente, debo agradecerle a Juan Boubeta toda la ayuda que me ha proporcionado en la parte de procesamiento de eventos complejos durante su estancia en el grupo. También muchas gracias a mis compañeros de laboratorio (Edurne, Alejandro, Guillermo, Fernando, Pery, Julio) por su ayuda y los momentos que hemos compartido tanto dentro como fuera del laboratorio.

Gracias también a la gente que he tenido la suerte de conocer en la Universidad, tanto profesores, por todo lo que me han enseñado, como compañeros de clase, que después de tanto tiempo compartido con ellos han pasado a ser amigos. Sobre todo a

Sergio, por su gran determinación y todos el tiempo que hemos pasado haciendo trabajos en grupo, Enrique, con el que también he tenido el placer de trabajar, Dani, Eneas, Tamara, Edurne, Javi y todos los demás. Lo he pasado muy bien con vosotros dentro y fuera del aula y me siento afortunado de haberos tenido como compañeros.

Muchas gracias a mi familia por todo el cariño y apoyo que siempre me dan y por entenderme cuando no he podido ir a casa tanto como me gustaría porque tenía que quedarme trabajando. También quiero acordarme de mis abuelos, a los cuales, se fuesen hace más o menos tiempo, siempre los tengo en el recuerdo y estoy seguro de que estarían orgullosos de "su nieto".

Finalmente, quiero dar las gracias a mis amigos: Fernando, Álvaro, Fran, Alba, Inés, Fran Rosa, Víctor Cuenca, Manuel, José Alberto. Siempre están ahí cuando los necesito y quiero agradecerles todas las tardes (y noches) que he pasado con ellos, ya hayan sido en el pueblo, en Cáceres o en cualquier otro sitio y también las conversaciones por Whatsapp que me hacían sentirlos mucho más cerca. Gracias por todos esos momentos compartidos y por los que nos quedan, ya que eso ha sido lo que me ha dado ánimos incluso los días en los que tenía mil cosas que hacer y no sabía por dónde empezar.

Muchas gracias a todos.

Anexo A

Instalación y configuración.

En este anexo se tratará la instalación y configuración de todo el sistema, tanto de la parte software como de la parte hardware. Para la parte software del sistema que va en la Raspberry Pi, se indicarán dos formas de instalarla: a partir de la imagen que se proporciona en el disco o, instalando cada uno de los componentes necesarios.

Tanto la imagen del sistema que va en la Raspberry Pi, como el proyecto de Arduino (incluye el sketch y las librerías que éste usa para) se encuentran en la carpeta **'Software'** del DVD.

A.1 Instalación de sensores y actuadores.

Para instalar los sensores en el Arduino, en primer lugar, es necesario conectar la Grove Shield al Arduino. Esto es muy sencillo, ya que simplemente se tienen que encajar los conectores de la parte inferior en los pines de la placa Arduino Uno. Una vez está conectada la placa, se tienen que conectar cada uno de los sensores y actuadores al pin adecuado:

- **Pin A0**: Sensor de radiación ultravioleta.
- **Pin A1**: Sensor de monóxido de carbono (CO).
- **Pin A2**: Sensor de partículas volátiles (HCHO).
- **Pin D2**: Sensor de temperatura y humedad.
- **Pin D3**: Sensor de agua.
- **Pin D4**: Sensor de ultrasonidos.
- **Pin D5**: Relé que enciende la tira de LEDs.
- **Pin I2C**: Sensor barométrico.
- **Pin I2C**: Sensor digital de intensidad luminosa.

Una vez se han conectado todos los sensores, se tendrá algo similar a lo que se muestra en la Figura A.1.

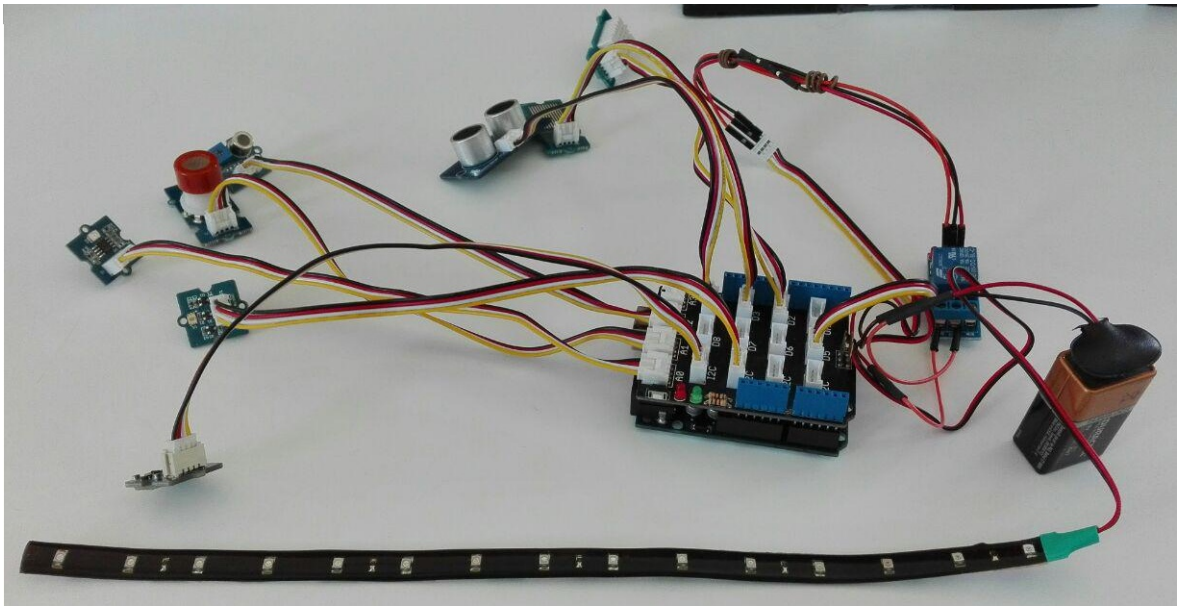

Figura A.1: Arduino Uno con los sensores y actuadores conectados a la Grove Shield.

En cuanto al otro actuador, la cámara Pi NoIR, simplemente es necesario que conectar el bus al puerto CSI de la Raspberry Pi, como vemos en la Figura A.2.

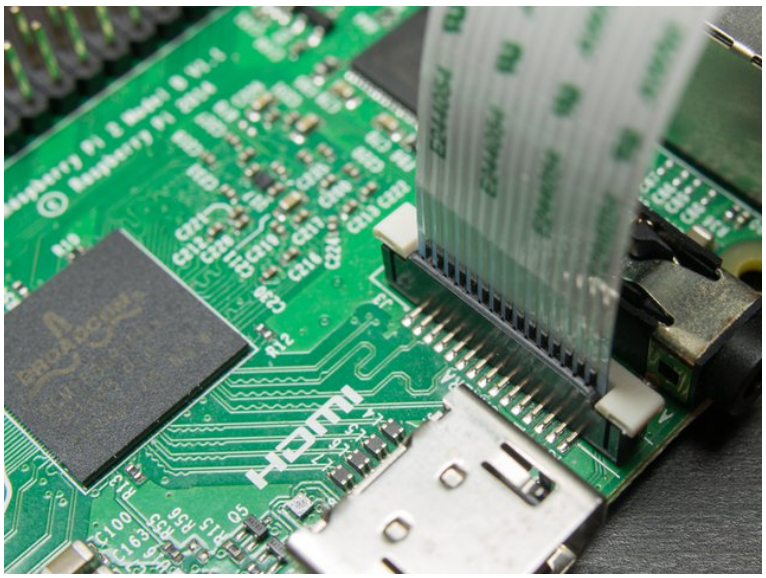

Figura A.2: Pi NoIR conectada a la Raspberry Pi. Fuente https://learn.adafruit.com/

A.1.1 Carga del sketch en el Arduino.

Para terminar, se tendrá que cargar el sketch que lee los valores de los sensores y los envía en formato JSON al puerto serie en el Arduino. Para ello, se hará uso del IDE de Arduino.

La instalación de este IDE en Ubuntu (a partir de la versión 12.04) se hace ejecutando el siguiente comando en la consola:

```
sudo apt-get install arduino arduino-core
```

Con el IDE instalado, solo queda copiar el contenido del archivo **sendSensorsAndReadCommands+Libraries.zip** en la carpeta donde Arduino almacena los proyectos y librerías. Esta carpeta normalmente se encuentra dentro del directorio *home*,

con el nombre **sketchbook**.

Por último, se tendrá que conectar el Arduino al ordenador a través del puerto USB y abrir el archivo **SendSensorsAndReadCommands.ino** con el IDE de Arduino. Cuando esté abierto, se hará click en el icono con la flecha hacia la derecha (o en Sketch > Upload), para que el sketch se compile y se cargue en el Arduino.

A.2 Instalación del sistema a partir de la imagen.

Para instalar el sistema a partir de la imagen que se encuentra en el DVD, en primer lugar, es necesaria una tarjeta MicroSD de, al menos 8 GB, ya que el sistema operativo junto con todos los programas y el sistema desarrollado en este TFG ocupa 4,4 GB. No es necesario que la tarjeta esté formateada, ya que se formateará durante el proceso.

En primer lugar, es necesario extraer el archivo **SO_RaspberryPi+DroneSystem.zip**, que contiene la imagen del sistema. Para ello, se utilizará el comando:

```
unzip SO_RaspberryPi+DroneSystem.zip
```

Una vez se ha extraído la imagen, se ejecutará el siguiente comando para ver qué dispositivos hay conectados:

```
df -h
```

Tras esto, se conectará la tarjeta MicroSD y se volverá a ejecutar el comando anterior:

```
df -h
```

El nuevo dispositivo (o dispositivos, si la tarjeta tiene varias particiones) que aparece es la tarjeta SD. En la columna de la izquierda se muestra el nombre de la tarjeta SD; será algo parecido a */dev/mmcblk0p1* o */dev/sdd1*. La última parte (p1 o 1, respectivamente) es el número de partición. Por tanto, si se quita esa parte del nombre (obteniendo, por ejemplo, */dev/mmcblk0* o */dev/sdd*) se obtendrá el nombre de toda la tarjeta SD. Ahora que se ha determinado cuál es el nombre del dispositivo, habrá que desmontar todas sus particiones para poder escribir en él la imagen del sistema operativo. Suponiendo que el dispositivo tiene una sola partición, denominada */dev/sdd1*, se ejecutará el siguiente comando para desmontarla:

```
umount /dev/sdd1
```

Si tuviese más particiones, se procedería de igual forma con todas ellas. A continuación, se utilizará el comando **dd** para escribir la imagen en la tarjeta. En el argumento *if* se indicará la ruta hacia el **fichero .img** que se ha descargado, mientras que en el argumento *of* se pondrá el nombre de la tarjeta completa. Además, con el argumento *bs* se indicará el tamaño de los bloques, el cual se recomienda que sea de 1 MB. Suponiendo que el nombre de la tarjeta es */dev/sdd*, el comando sería el siguiente:

```
sudo dd if=/home/user/carpetaDeImagen/SO_RaspberryPi+DroneSystem.img
of=/dev/sdd bs=1M
```

El comando **dd** no muestra ninguna información de su progreso, por tanto, es necesario tener paciencia hasta que termine. Una vez que el comando **dd** ha terminado, ya estará Raspbian instalado en la Raspberry Pi, junto con el sistema de control del dron. Sin embargo, solo habrá unos 300 MB de espacio libre en el sistema de ficheros.

Para poder aprovechar todo el espacio que tiene la tarjeta SD, se tendrá que expandir el sistema de ficheros, ya que la imagen que se ha copiado solo tenía unos 300 MB de espacio libre. En primer lugar, habrá que encender la Raspberry Pi y ejecutar el comando de configuración:

```
sudo raspi-config
```

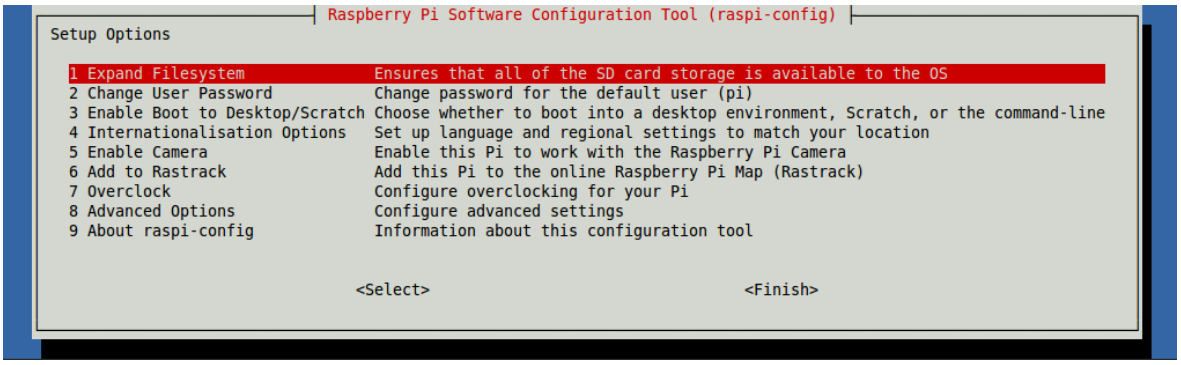

Figura A.3: Pantalla mostrada al ejecutar raspi-config. Fuente https://www.raspberrypi.org/

Esto mostrará una pantalla como la que se puede ver en la Figura A.3, en la que existe una lista con distintas opciones de configuración del sistema. Se elegirá la primera opción, **Expand Filesystem**, y una vez termine, se reiniciará el sistema para que se apliquen los cambios:

```
sudo reboot
```

En este punto, la Raspberry Pi ya está lista, de forma que, cuando se inicia, ejecuta automáticamente el sistema de control del dron.

A.3 Instalación del sistema paso a paso.

Si se prefiere instalar cada uno de los programas en la Raspberry Pi, es necesario instalar los siguientes: Mule ESB, Mosquitto, DroneKit-Python y la aplicación Mule desarrollada en este TFG. Además de esto, se tendrá que instalar el sistema operativo Raspbian.

A.3.1 Raspbian.

Para instalar Raspbian, se actuará de la misma manera que en la Sección A.2, solo que en lugar de utilizar la imagen del CD, se hará uso de la imagen de Raspbian que proporciona la página oficial de Raspberry.

Se entrará en `https://www.raspberrypi.org/downloads/` y se seleccionará **Raspbian**. A continuación, se descargará la imagen haciendo click en **«Download ZIP»**. Esto descargará la última imagen del sistema operativo (Raspbian Jessie), lanzada en Febrero de 2016 y basada en Debian Jessie. Tras completarse la descarga, se descomprimirá el fichero para obtener el archivo .img, que es el que contiene la imagen.

Con el **archivo .img** ya puede continuarse con la instalación como se muestra en la Sección A.2.

A.3.2 Mule ESB.

En este apartado se explica cómo instalar el **servidor Mule ESB** en la Raspberry Pi (en la cual se ha instalado Raspbian previamente), el cuál será utilizado para **desplegar la aplicación Mule** que controla al dron. Es necesario indicar que la versión gratuita de Mule (Mule Community), tal como se proporciona, no puede ser ejecutada en arquitecturas ARM (como la usada por Raspberry Pi), por tanto, se deben realizar cambios en algunos de sus ficheros. Esto es debido a cuestiones relacionadas con las licencias de algunas partes del programa proporcionadas por terceros. En concreto, se trata de *Java Service Wrapper*, un software usado para permitir que programas creados en Java se ejecuten como demonios Unix (o servicios, en caso de Windows). De esta forma, se consigue que Mule pueda ser ejecutado como un demonio que se inicie al arrancar el sistema.

Una vez explicados estos detalles sobre la instalación, se procede a llevarla a cabo siguiendo los pasos que se indican a continuación:

1. En primer lugar, es necesario **descargar el servidor Mule 3.7.0 Standalone Runtime** [68]. Para ello, se abrirá una terminal y se ejecutará:

   wget https://repository-master.mulesoft.org/nexus/content/repositories/releases/org/mule/distributions/mule-standalone/3.7.0/mule-standalone-3.7.0.tar.gz

2. Tras esto, **hay que descargar el software Java Service Wrapper** [69]. Como va a instalarse en una Raspberry Pi 2, se usará la versión **armhf** puesto que soporta operaciones en coma flotante de forma nativa. Si se usase otra versión, puede comprobarse este aspecto ejecutando el siguiente comando:

   ```
   readelf -A /proc/self/exe | grep Tag_ABI_VFP_args
   ```

   Si se si encuentra la etiqueta Tag_ABI_VFP_args, soporta operaciones en coma flotante; si, por contra, el comando no devuelve nada, tendremos que descargar la versión **armel**.

3. Para llevar a cabo la descarga, se hará lo siguiente:

   wget http://wrapper.tanukisoftware.com/download/3.5.27/wrapper-linux-armhf-32-3.5.27.tar.gz

4. **Se descomprimirán los ficheros descargados** en los pasos anteriores usando el comando **tar**. Lo que generará dos carpetas, una por cada fichero descomprimido.

   ```
   tar -zxvf mule-standalone-3.7.0.tar.gz
   tar -zxvf wrapper-linux-armhf-32-3.5.27.tar.gz
   ```

5. Una vez descomprimidos los ficheros, **se renombrarán algunos archivos del *wrapper* para que coincidan con los esperados por Mule ESB**. Se ejecutarán los siguientes comandos para ello:

```
mv wrapper-linux-armhf-32-3.5.27/bin/wrapper
↪ wrapper-linux-armhf-32-3.5.27/bin/wrapper-linux-armhf-32
mv wrapper-linux-armhf-32-3.5.27/lib/libwrapper.so
↪ wrapper-linux-armhf-32-3.5.27/lib/libwrapper-linux-armhf-32.so
```

Si se hubiese usado la versión **armel** en vez de **armhf**, los ficheros deberían haberse renombrado a **wrapper-linux-armel-32** y **libwrapper-linux-armel-32.so**, respectivamente.

6. Tras modificar el nombre a los archivos necesarios, se pasará a **copiar los archivos del *wrapper* a la carpeta del servidor Mule ESB**. En primer lugar, se eliminará el fichero **wrapper-3.2.3.jar** para evitar conflictos, el cuál será sustituido por una versión del *wrapper* compatible con Raspberry Pi. Además, se copiarán los archivos renombrados en el paso anterior:

```
rm mule-standalone-3.7.0/lib/boot/wrapper-3.2.3.jar
cp wrapper-linux-armhf-32-3.5.27/lib/wrapper.jar
↪ mule-standalone-3.7.0/lib/boot
cp wrapper-linux-armhf-32-3.5.27/bin/wrapper-linux-armhf-32
↪ mule-standalone-3.7.0/lib/boot/exec
cp wrapper-linux-armhf-32-3.5.27/lib/libwrapper-linux-armhf-32.so
↪ mule-standalone-3.7.0/lib/boot
```

7. El siguiente paso será **añadir la arquitectura del dispositivo**, ARM, entre las arquitecturas que el servidor Mule ESB soporta. Para ello, habrá que modificar el *script* de ejecución del mismo:

```
nano mule-standalone-3.7.0/bin/mule
```

8. Tras la **línea 274** se añadirá el fragmento de código que se muestra a continuación y se guardará el archivo. Es importante que el código se copie tal como está, sino, no funcionará:

```
armv*)
        if [ -z "`readelf -A /proc/self/exe | grep
↪  Tag_ABI_VFP_args`" ] ;
        then
                DIST_ARCH="armel"
                DIST_BITS="32"
        else
                DIST_ARCH="armhf"
                DIST_BITS="32"
        fi
        break;;
```

Este código se ha adaptado a partir del que aparece en el *script* de ejecución de la versión Mule Enterprise, que es de pago y sí puede ser ejecutada directamente en Raspberry Pi, puesto que incluye la arquitectura ARM como una de sus arquitecturas válidas.

9. A continuación, se tendrá que **modificar un archivo de configuración del *wrapper***, para que la máquina virtual de Java se adapte a la cantidad de memoria disponible en la Raspberry Pi:

   `nano mule-standalone-3.7.0/conf/wrapper.conf`

   En las **líneas 32 y 33** se cambiará el valor de **NewSize** y **MaxNewSize** a 150m:

   ```
   wrapper.java.additional.12=-XX:NewSize=150m
   wrapper.java.additional.13=-XX:MaxNewSize=150m
   ```

   Además, en las **líneas 132 y 135**, se modificará el valor de **initmemory** y **maxmemory** a 200 y 300, respectivamente:

```
# Initial Java Heap Size (in MB)
wrapper.java.initmemory=200

# Maximum Java Heap Size (in MB)
wrapper.java.maxmemory=300
```

En este punto, ya está terminada la preparación para que el servidor Mule ESB pueda ejecutarse en Raspberry Pi.

10. Una vez finalizada la configuración y preparación del servidor, se puede **mover la carpeta de Mule ESB al directorio donde normalmente se almacenan los programas instalados localmente**, modificando además los permisos necesarios para que pueda ser utilizado por cualquier usuario:

    ```
    sudo mv mule-standalone-3.7.0 /usr/local
    sudo chmod -R ugo+rwx /usr/local/mule-standalone-3.7.0
    ```

11. En este momento, debe **crearse un enlace simbólico que permitirá ejecutar Mule ESB directamente desde consola**, puesto que se situará en el directorio /**usr**/**local**/**bin**, que ya está incluido por defecto en la variable $PATH:

    ```
    sudo ln -s /usr/local/mule-standalone-3.7.0/bin/mule /usr/local/bin/mule
    ```

12. Por último, se hará que el **servidor Mule ESB se inicie en segundo plano cada vez que arranque la Raspberry Pi**, de tal forma que no sea necesario preocuparse por iniciar manualmente la aplicación Mule que ponga en él. Para ello:

    ```
    sudo nano /etc/init.d/mule-server
    ```

13. En el fichero que se abre, se insertará el siguiente *script*:

```
#! /bin/bash
### BEGIN INIT INFO
# Provides: mule-server
# Required-Start: $network $syslog
# Required-Stop: $network $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: MULE INITIATION
# Description: #
### END INIT INFO


/usr/local/bin/mule
```

En el *script* creado, se ha incluido la información necesaria para que el programa se inicie cuando arranca el sistema. Es decir, se ha creado un *daemon* o *demonio* del sistema.

14. Se le darán permisos de ejecución al *script* creado:

    ```
    sudo chmod ugo+x /etc/init.d/mule-server
    ```

15. Utilizando el comando **update-rc.d**, se actualizarán automáticamente los enlaces a los *scripts* ejecutados por *init*, es decir, aquellos que se ejecutan durante el arranque del sistema:

    ```
    sudo update-rc.d mule-server defaults
    ```

16. Una vez completado el paso anterior, Mule ESB se iniciará automáticamente. Esto puede comprobarse de la siguiente manera:

    ```
    mule status
    ```

A.3.3 Mosquitto.

Para instalar Mosquitto en una Raspberry Pi, en primer lugar, es necesario **añadir y activar el repositorio de Mosquitto**:

```
curl -O http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
sudo apt-key add mosquitto-repo.gpg.key
rm mosquitto-repo.gpg.key
cd /etc/apt/sources.list.d/
sudo curl -O http://repo.mosquitto.org/debian/mosquitto-$(awk
↪ -F"[)(]+" '/VERSION=/ {print $2}' /etc/os-release).list
sudo apt-get update
```

A continuación, **se llevará a cabo la instalación**. Para ello se instalarán dos paquetes: **el propio broker y un cliente en línea de comandos**, que permitirá hacer algunas pruebas.

```
sudo apt-get install mosquitto mosquitto-clients
```

De esta forma se instalará Mosquitto y, además, se iniciará automáticamente. El siguiente paso será **parar el servicio de Mosquitto para modificar el archivo de configuración** y hacer que los mensajes no se almacenen en disco, ya que no es necesario.

```
sudo /etc/init.d/mosquitto stop
nano /etc/mosquitto/mosquitto.conf
```

En el archivo de configuración hay que buscar la variable **persistence** y le ponerle el valor **false**. Una vez se ha hecho esto, se guardará el fichero y se podrá volver a iniciar el servicio con el comando:

```
sudo /etc/init.d/mosquitto start
```

Ya está instalado Mosquitto en la Raspberry Pi y se iniciará cada vez que arranque el sistema.

A.3.4 DroneKit-Python.

Para llevar a cabo la instalación de la librería DroneKit para Python, en primer lugar, se tendrá que instalar el paquete **python-dev**, ya que, debido a la forma en que viene instalado Python (instalando solo aquello imprescindible), no está disponible el archivo *Python.h*, necesario para compilar la librería DroneKit.

```
sudo apt-get install python-dev
```

Tras esto, se procederá a instalar la librería:

```
sudo pip install dronekit
```

Una vez termina el proceso, la librería de Python estará disponible y podrá utilizarse.

A.3.5 Aplicación Mule.

Por último, para instalar la aplicación Mule, solo habrá que copiar el proyecto y unos archivos. Veamos cómo se hace esto.

En primer lugar, se descomprimirá el archivo que contiene el proyecto y algunos archivos necesarios:

```
unzip proyecto_drone_threads_single_python.zip
```

Cuando se haya extraído, habrá un archivo comprimido llamado **soa-cep-drone-threads-single-python.zip** y una carpeta llamada **'droneFiles'**.

A continuación, se copiará la carpeta **'droneFiles'**, donde están situadas las carpetas que contienen los patrones de eventos, las acciones asociadas a los patrones y el script Python que se lanza desde la aplicación Mule. Esta carpeta puede copiarse donde se quiera, pero como, por defecto, la aplicación está configurada para buscarla en **'/home/pi'**, la copiaremos ahí.

```
cp -r droneFiles /home/pi
```

Finalmente, sólo quedará copiar la aplicación al directorio donde Mule ESB Runtime almacena todas las aplicaciones. Este directorio es una carpeta llamada **apps**, que se encuentra dentro del directorio donde esté Mule. Si se han seguido los pasos indicados en la Subsección A.3.2, el comando será el siguiente:

```
sudo cp -r soa-cep-drone-threads-single-python.zip
↪ /usr/local/mule-standalone-3.7.0/apps
```

A.4 Configuración del sistema.

Únicamente existe un parámetro que configurar, que es la **ruta donde se encuentra la carpeta 'droneFiles'**. Si se ha copiado en un directorio distinto a '/home/pi', habrá que indicar cuál es este directorio en el archivo **'mule-app.properties'**, que se encuentra dentro de la carpeta con el proyecto Mule. Como este archivo se encuentra dentro de un subdirectorio de **'/usr/local'**, es necesario abrirlo como *root* para que sea posible guardar los cambios. Una vez abierto, solamente habrá que modificar el valor de la variable **'filesDir'**, indicando el directorio donde está la carpeta **'droneFiles'** (al

final de la ruta habrá que poner una barra '/' para que todo funcione correctamente). Por ejemplo:

```
filesDir=/home/pi/Desktop/droneFiles/
```

Anexo B

Uso del sistema.

En este otro anexo, se indicará cómo utilizar el sistema instalado en el anterior anexo. Se trata de un sistema sencillo de utilizar, solo hay que saber cómo conectar el dron y el Arduino a la Raspberry, cómo cargar las reglas y cuáles son los pasos a seguir para ponerlo en funcionamiento. Además, se explicará cómo cargar una ruta en el dron, ya que el sistema actuará sobre un dron que esté siguiendo una ruta prefijada.

B.1 Conexión de dron y Arduino.

Al iniciar el sistema operativo Raspbian, en primer lugar se comprueban los dispositivos conectados en los 2 puertos USB que están situados al lado del puerto Ethernet y, a continuación, los otros 2. Dentro de la misma "columna" de puertos USB, se comprueba primero el superior y luego el inferior. Esto permite saber el nombre que el sistema operativo le pondrá al Arduino y al dron (concretamente, al Ardupilot) cuando se inicie el mismo (siempre que ambos estén conectados antes de encender la Raspberry, si no, se nombrarán en el orden en el que se conecten). A ambos se les dará el nombre **ttyACMX** (siendo **X** un número), pero el que esté conectado en el puerto USB que se compruebe

antes será el *ttyAMC0*, mientras que el otro será el *ttyAMC1*. El orden de comprobación se muestra en la Figura B.1.

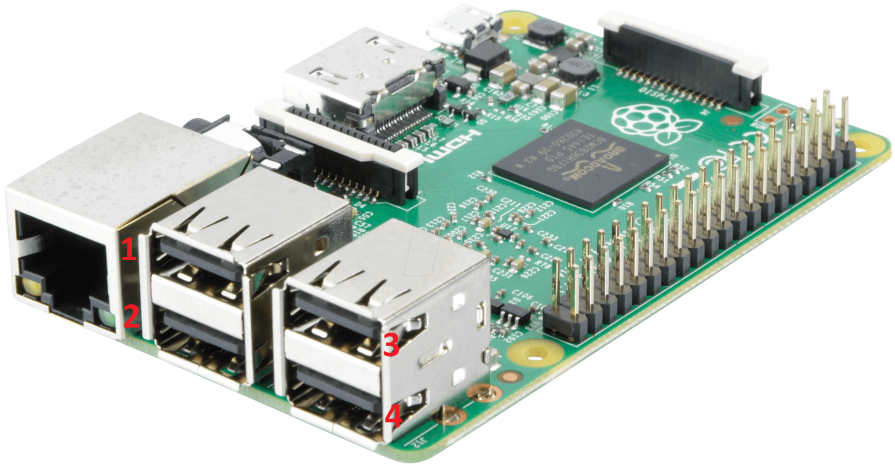

Figura B.1: Orden de comprobación de puertos USB en Raspberry Pi 2 Model B.

Teniendo en cuenta este orden de comprobación, habrá que conectar el dron de forma que se le asigne el nombre **ttyACM0** y al Arduino, **ttyACM1**. Por ejemplo, se puede conectar el dron al USB que se comprueba en primer lugar y, el Arduino, al que se comprueba en tercer lugar.

B.2 Carga de patrones y acciones.

Una vez se ha creado una regla (un par de patrón de eventos y acción), habrá que cargarla en el sistema. Para ello, simplemente será necesario que situar el archivo con la regla (**.epl**) en la carpeta **'new-eventpattern'** y el archivo con la acción (**.action**), en la carpeta **'new-action'**. Esta carpeta está dentro de la carpeta **'droneFiles'**, donde se encuentran todos los archivos utilizados por la aplicación Mule. Tras situar los archivos en sus carpetas, la aplicación los cargará y la regla será utilizada.

Si se quiere **eliminar una regla**, habrá que borrar el archivo **.epl** y el **.action** que

la define. Estos archivos estarán en las carpetas **'new-eventpattern'** y **'new-action'** si aún no han sido cargados, y en las carpetas **'deployed-eventpattern'** y **'deployed-action'** si ya se han cargado.

B.3 Carga de ruta en el dron.

Para cargar una ruta (también llamada misión) en el dron, habrá que conectarlo a nuestro ordenador (a través de USB o de telemetría) y entrar en APM Planner (o Mission Planner, si se utiliza Windows). En la pantalla principal de APM Planner, se elegirá el **puerto** que se le haya asignado al dron, y se pondrá la **velocidad de transmisión** de datos adecuada (115200 baudios si se utiliza USB y 57600 en el caso de la telemetría). Tras esto, se pulsará el botón de conectar y APM Planner se conectará al dron. En [70] se explica este proceso más detalladamente.

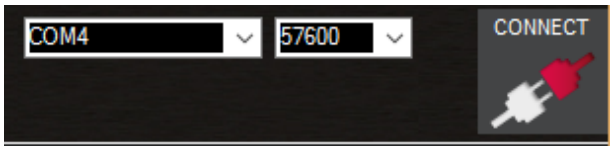

Figura B.2: Parte de la interfaz de Mission Planner que permite la conexión al dron. Fuente http://ardupilot.org/

A continuación, se hará click en la pestaña **'Flight Plan'** y, en el mapa que aparece, habrá un marcador con el punto donde está el dron, que será considerado el punto de despegue, o punto ***home*** (si se va a despegar desde otro punto, es posible moverlo). Para crear una ruta, se irá haciendo doble click sobre el mapa para añadir más marcadores, con los que se conformará la ruta que seguirá el dron. En la parte inferior de esta pestaña, se pueden editar las propiedades de cada uno de estos puntos, de forma que no sean simplemente puntos de paso. Una vez se haya creado la misión, se hará click en **'Write WPs'** y la misión se escribirá en la memoria del dron. Toda la información relativa a la creación de misiones, puede encontrarse en el apartado **'Mission Planning'**

de la web de Ardupilot [71].

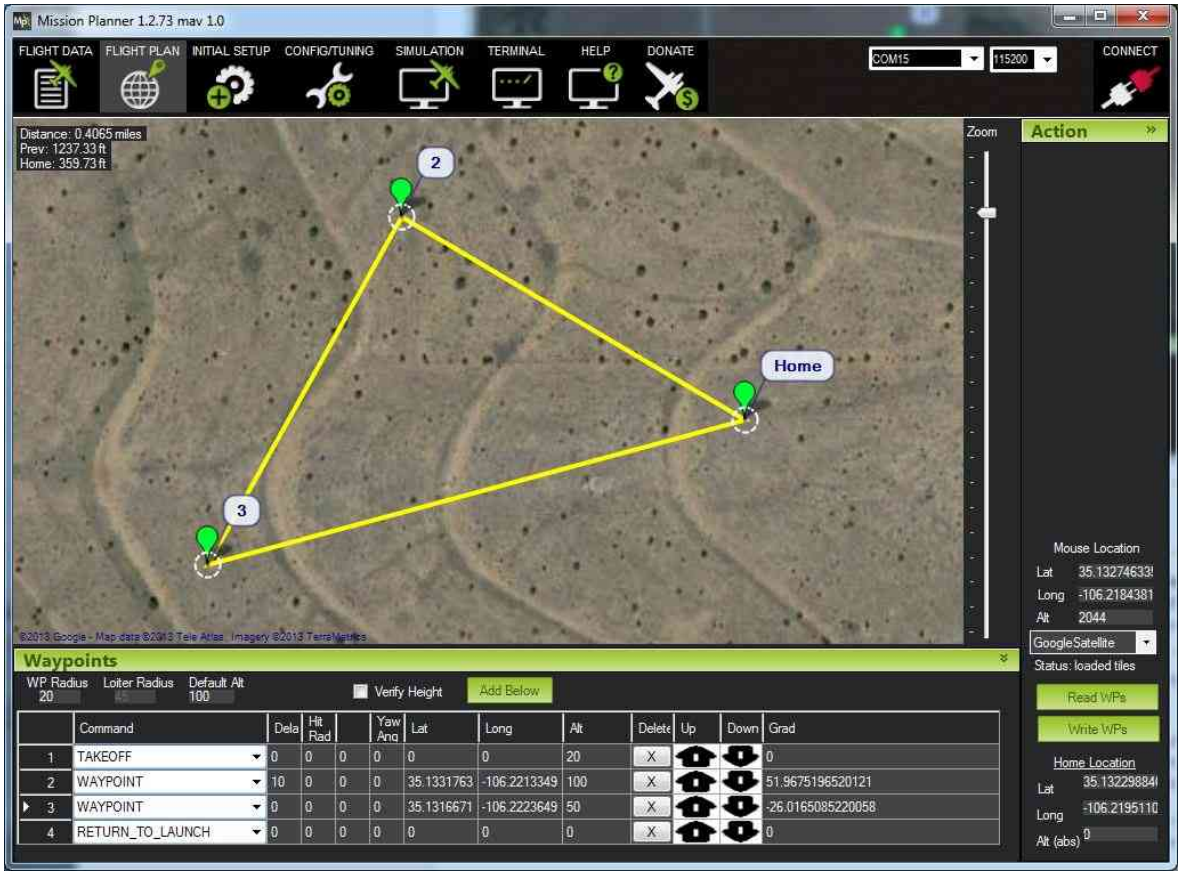

Figura B.3: Pestaña 'Flight Plan' de Mission Planner. Fuente http://ardupilot.org/

B.4 Puesta en funcionamiento.

Para poner el sistema en funcionamiento, habrá que seguir estos pasos:

1. Cargar las reglas en la Raspberry Pi, de modo que estén disponibles para cuando se conecte al dron y al Arduino.
2. Cargar una misión en el dron.
3. Conectar la Raspberry Pi al dron y al Arduino, siguiendo las indicaciones de la Sección B.1.

4. Encender la Raspberry Pi y esperar hasta que la tira de LEDs parpadee (tarda entre 1'30" y 2'), lo que indicará que todo está correcto.

5. Comenzar la misión con el dron.

Siguiendo estos pasos, no debería haber ningún problema y el sistema funcionará como se espera de él.

B.5 Funcionamiento con el simulador.

Para ejecutar **ciertas pruebas** o, con **propósitos de depuración**, es preferible no utilizar el dron real y usar en, su lugar, el **simulador SITL**, ya que esto permite poner en marcha las pruebas de una forma rápida y, además, se podrán observar los movimientos del dron en APM Planner. Dicho simulador no puede utilizarse en la Raspberry Pi, ya que solo soporta arquitecturas x86. Por otra parte, en la Raspberry Pi tampoco se podría instalar APM Planner, por lo que no sería posible ver los movimientos del dron simulado.

En primer lugar, habrá que **instalar el simulador SITL**. La forma más rápida de hacer esto, es instalando un paquete que proporciona la librería DroneKit (para que esto funcione, se debe haber instalado previamente DroneKit, como se indica en la Subsección A.3.4). Para instalar dicho paquete, se ejecutará el siguiente comando en una consola:

```
sudo pip install dronekit-sitl
```

Una vez instalado, se podrá probar directamente (sin hacer ningún tipo de modificación) el ejemplo que se muestra en http://python.dronekit.io/guide/quick_start.html y así garantizar de que funciona correctamente.

Por otra parte, habrá que utilizar el script Python llamado **droneActionsAndData_debug.py**, en vez del que viene por defecto en la aplicación Mule. Para ello, se irá al flujo que se encarga de lanzarlo y en el componente de Groovy, se cambiará el nombre del script, para utilizar la versión *debug*. Si no se quiere ejecutar el sistema completo, sino únicamente el script Python, no es necesario hacer esto, solo habrá que lanzar el archivo **droneActionsAndData_debug.py** cuando sea necesario.

Tras esto, se podrá intentar lanzar el simulador DroneKit-SITL y conectarse a él desde el script el Python (sólo o incluido en la aplicación Mule) y, a la vez desde APM Planner. De esta forma, se cargará una misión en el dron con APM Planner y, una vez hecho esto, se hará que el dron despegue y comience la misión con el script en Python. De esta forma, se podrá ver en APM Planner cómo se va moviendo el dron simulado y toda la información acerca del modo, nivel de batería...

Los pasos que hay que seguir son los siguientes:

1. Para **lanzar el simulador** DroneKit-SITL, se tendrá que ejecutar el siguiente comando en un terminal:

   ```
   dronekit-sitl copter --home=39.478317,-6.341928,584,353 --model=quad
   ```

   Se pasan 3 parámetros: En primer lugar, el tipo de vehículo utilizado (en este caso, *copter*); a continuación, las coordenadas de ***home*** junto con la altura del suelo y los grados de desviación respecto al norte y, por último, el modelo de vehículo (en este caso *quad*).

2. Una vez se ha lanzado el simulador, este aceptará conexiones *TCP* en la dirección *IP* **127.0.0.1:5760**. Esto se usará para **conectarlo con APM Planner**. Para ello, en primer lugar, habrá que activar el modo avanzado; esto se consigue haciendo click en la opción del menú: **File** > **Advanced Mode**. A continuación, se creará una conexión al simulador; para hacer esto, se usará uno de los nuevos menús a los que se tiene acceso. En concreto, se irá a **Communication** > **Add Link** > **TCP**

y, en la ventana que se abre, se introducirá el puerto **5760** en **TCP Port** y la dirección IP **127.0.0.1** en **Host Address**; se pulsará en **Connect** y esto hará que APM Planner se conecte al simulador DroneKit-SITL. Esto, además, provocará que dicho simulador acepte conexiones en otros dos puertos, concretamente en el **5762** y el **5763**, los cuales se utilizarán más tarde. **Una vez conectado, se creará una misión y se escribirá en el dron**. Tras esto, **se desconectará** Mission Planner del dron (ya que el puerto 5760 se utilizará en el script de Python para enviar instrucciones al dron) **y se volverá a conectar**, pero utilizando el puerto **5763**, en vez del 5760.

3. Por último, **se lanzará la aplicación Mule** (o solamente el script Python), que se conectará al dron y hará que comience la misión que se ha cargado en él. En el caso de que se quiera lanzar la aplicación Mule completa, esto se hará desde **Mule Studio** (o Anypoint Studio), mientras que si solo se quiere lanzar el script, puede hacerse **desde consola** utilizando el comando `python droneActionsAndData_debug.py`

4. En caso de que **se haya lanzado solo el script Python**, normalmente querremos **enviarle comandos para que los ejecute**, ya que no estará disponible el sistema de reglas que se encarga de hacerlo cuando detecta un patrón de eventos. Para hacer esto, se utilizará el siguiente comando en una terminal:

   ```
   mosquitto_pub -h localhost -t "drone/actions" -m "<Comando que se quiera enviar>"
   ```

   La cadena <**Comando que se quiera enviar**> se sustituirá por el comando que quiera que ejecutarse, ya sea GOTO *p lat lon alt*, LAND *p*, RTL *p* o, incluso alguno referido al uso de la cámara (que se simulará, ya que ésta solo está disponible en la Raspberry Pi) o al encendido o apagado de la tira de LEDs.

Anexo C

Índice de acrónimos

Acrónimos por orden alfabético:

AMQP — Advanced Message Queuing Protocol.

API — Application Programming Interface.

APM — ArduPilot Mega.

ARM — Advanced RISC Machine.

AUVSI — Association for Unmanned Vehicle Systems International.

CEP — Complex Event Processing.

CGS — Control Ground Station.

CPU — Central Processing Unit.

CSI — Camera Serial Interface.

DSI — Display Serial Interface.

DSL — Domain Specific Language.

DSP — Digital Signal Processing.

DVD — Digital Versatile Disc.

EEPROM — Electrically Erasable Programmable Read-Only Memory.

ESB — Enterprise Service Bus.

ESC — Electronic Speed Control.

fps — frames por segundo.
GPS — Global Positioning System.
GPIO — General-Purpose Input/Output.
GPU — Graphics Processing Unit.
HDMI — High-Definition Multimedia Interface.
HTTP — Hypertext Transfer Protocol.
IBM — International Business Machines.
IDE — Integrated Development Environment.
IoT — Internet of Things.
IP — Internet Protocol.
ISO — International Organization for Standardization.
JDBC — Java Database Connectivity
JMS — Java Message Service.
JVM — Java Virtual Machine.
LCD — Liquid-Crystal Display.
LED — Light-Emitting Diode.
LGPL — Lesser General Public License.
Li-Po — Lithium-Polymer.
M2M — Machine-to-Machine.
mAh — miliamperio-hora.
MAVLink — Micro Air Vehicle Link.
MicroSD — Micro Secure Digital.
MIPI — Mobile Industry Processor Interface.
MIT — Massachusetts Institute of Technology.
MQTT — Message Queue Telemetry Transport.
NTSC — National Television System Committee.
PAL — Phase Alternating Line.
R/C — Radiocontrol.
RAM — Random Access Memory.

RCA	—	Radio Corporation of America.
RJ-45	—	Registered Jack 45.
RPA	—	Remotely Piloted Aircraft.
RPF	—	Raspberry Pi Foundation.
RSMB	—	Really Small Message Broker.
SDRAM	—	Synchronous Dynamic Random Access Memory.
SITL	—	Software In The Loop.
SOA	—	Service-Oriented Architecture.
SoC	—	System on Chip.
SRAM	—	Static Random Access Memory.
TCP	—	Transmission Control Protocol.
TFG	—	Trabajo de Fin de Grado.
UAV	—	Unmanned Aerial Vehicle.
UCAV	—	Unmanned Combat Aerial Vehicle.
UEx	—	Universidad de Extremadura.
USB	—	Universal Serial Bus.
UV	—	Ultravioleta.
VANT	—	Vehículo Aéreo No Tripulado.
WQM	—	Websphere Message Queue.
XML	—	eXtensible Markup Language.

Tabla C.1: Índice de acrónimos

Referencias

[1] Incendio del cementerio de neuméticos de Seseña, a vista de dron | El País. http://elpais.com/elpais/2016/05/13/videos/1463132050_060950.html, consultado el 7 de Junio de 2016.

[2] Correos utiliza drones para repartir paquetes y cartas en Asturias | elPeriodico. http://www.elperiodico.com/es/noticias/economia/correos-drones-repartir-paquetes-cartas-asturias-4727091, consultado el 7 de Junio de 2016.

[3] Google comenzará a repartir sus pedidos con drones a partir de 2017 | El Mundo. http://www.elmundo.es/tecnologia/2015/11/03/5638725a22601da4658b457e.html, consultado el 7 de Junio de 2016.

[4] Amazon podrá repartir paquetes con drones | El Mundo. http://www.elmundo.es/tecnologia/2015/03/20/550c3ecfca474124708b4572.html, consultado el 7 de Junio de 2016.

[5] Guidance: Un sistema de percepción visual revolucionario para plataformas aéreas. http://www.dji.com/es/product/guidance, consultado el 22 de Mayo de 2016.

[6] Quercus Software Engeenering Group | Universidad de Extremadura. http://www.unex.es/investigacion/grupos/quercus, consultado el 8 de Junio de 2016.

[7] H. Yeo D. Hyunchul Shim, J. Han. A development of unmanned helicopters for industrial applications. Journal of Intelligent and Robotic Systems, 2009.

[8] S. Leva M. Mussetta M. Aghaei P.B. Quater, F. Grimaccia. Light unmanned aerial vehicles (uavs) for cooperative inspection of pv plants. IEEE Journal of Photovoltaics, 2014.

[9] A. Restas. Forest fire management supporting by uav based air reconnaissance results of szendro fire department, hungary. 1st International Symposium on Environment Identities and Mediterranean Area (ISEIMA '06), 2006.

[10] R. Suffritti F. Grimaccia P. Sinogas O. Guye C. Papin T. Michalareas L. Gazdag I. Rakkolainen M. Neri, A. Campi. Skymedia - uav-based capturing of hd/3d content with wsn augmentation for immersive media experiences. IEEE International Conference on Multimedia and Expo (ICME), 2011.

[11] J. Capitán J.R. Martínez-De-Dios A. Ollero I. Maza, F. Caballero. Experimental results in multi-uav coordination for disaster management and civil security applications. Journal of Intelligent and Robotic Systems, 2011.

[12] Amazon podrá repartir paquetes con drones | El Mundo. http://higherlogicdownload.s3.amazonaws.com/AUVSI/958c920a-7f9b-4ad2-9807-f9a4e95d1ef1/UploadedImages/New_Economic%20Report%202013%20Full.pdf, consultado el 7 de Junio de 2016.

[13] España. Ley 18/2014, de 15 de octubre, 2014. http://www.seguridadaerea.gob.es/media/4389070/ley_18_2014_de_15_octubre.pdf.

[14] Nueva Ley de Drones | Drone Spain. http://www.dronespain.pro/nueva-ley-de-drones-en-espana-marzo-2016/, consultado el 15 de Junio de 2016.

[15] Un avión que aterrizaba en Bilbao se cruzó con tres drones | elPeriódico. http://www.elperiodico.com/es/noticias/sociedad/tres-drones-ponen-riesgo-aterrizaje-avion-bilbao-5152015, consultado el 15 de Junio de 2016.

[16] EEUU registró casi 600 incidentes con drones en menos de seis meses | El Mundo. http://www.elmundo.es/tecnologia/2016/03/28/56f8e992e2704e7d198b45db.html, consultado el 15 de Junio de 2016.

[17] Un dron, a punto de chocar con un avión en un aeropuerto de París | El País. http://economia.elpais.com/economia/2016/03/04/actualidad/1457078339_462092.html, consultado el 15 de Junio de 2016.

[18] Wikipedia - Vehículo aéreo no tripulado. https://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado, consultado el 10 de Mayo de 2016.

[19] Fixed Wing Versus Rotary Wing For UAV Mapping Applications. http://www.questuav.com/news/fixed-wing-versus-rotary-wing-for-uav-mapping-applications, consultado el 10 de Mayo de 2016.

[20] Tipos de drones en función del método de control. http://www.areatecnologia.com/aparatos-electronicos/drones.html, consultado el 10 de Mayo de 2016.

[21] What you need to build a multiCopter - Copter Documentation. http://ardupilot.org/copter/docs/what-you-need.html#multicopter-frame-including-motors-escs-and-propellers/, consultado el 11 de Mayo de 2016.

[22] DroneSpeak - Electronic Speed Control (ESC). http://dronespeak.com/dronespeak-vocabulary/unmanned-vehicle-system-uvs/unmanned-aerial-vehicle-system-uas/unmanned-aerial-vehicle-uav/propulsion-system/uav-electrochemical-mechanical-propulsion/electronic-speed-control-esc/, consultado el 11 de Mayo de 2016.

[23] Wikipedia - Inertial measurement unit. https://en.wikipedia.org/wiki/Inertial_measurement_unit, consultado el 11 de Mayo de 2016.

[24] ArduPilot Mega Home. http://www.ardupilot.co.uk/, consultado el 11 de Mayo de 2016.

[25] Pixhawk Flight Controller Hardware Project. https://pixhawk.org/start, consultado el 11 de Mayo de 2016.

[26] Flight Modes - Copter Documentation. http://ardupilot.org/copter/docs/flight-modes.html, consultado el 11 de Mayo de 2016.

[27] Wikipedia - Single-board computer. https://en.wikipedia.org/wiki/Single-board_computer, consultado el 17 de Mayo de 2016.

[28] ODROID | Hardkernel. http://www.hardkernel.com/main/main.php, consultado el 17 de Mayo de 2016.

[29] Página oficial de Raspberry Pi. https://www.raspberrypi.org/, consultado el 17 de Mayo de 2016.

[30] Página oficial de BeagleBoard. http://beagleboard.org/, consultado el 17 de Mayo de 2016.

[31] Wikipedia - Arduino. https://en.wikipedia.org/wiki/Arduino, consultado el 17 de Mayo de 2016.

[32] Wikipedia - Vehículo aéreo no tripulado. https://en.wikipedia.org/wiki/Complex_event_processing, consultado el 11 de Mayo de 2016.

[33] Wikipedia - Drools. https://en.wikipedia.org/wiki/Drools, consultado el 12 de Mayo de 2016.

[34] Wikipedia - Esper. https://en.wikipedia.org/wiki/Esper_(software), consultado el 12 de Mayo de 2016.

[35] Wikipedia - Enterprise Service Bus. https://es.wikipedia.org/wiki/Enterprise_service_bus, consultado el 12 de Mayo de 2016.

[36] What is Mule ESB? | MuleSoft. https://www.mulesoft.com/resources/esb/what-mule-esb, consultado el 16 de Mayo de 2016.

[37] Anypoint Exchange | Connectors | MuleSoft. https://www.mulesoft.com/exchange#!/?types=connector&sortBy=name, consultado el 17 de Mayo de 2016.

[38] MAVLink Micro Air Vehicle Communication Protocol - QGroundControl GCS. http://qgroundcontrol.org/mavlink/start, consultado el 17 de Mayo de 2016.

[39] Proyecto MAVLink en GitHub. https://github.com/mavlink/mavlink, consultado el 17 de Mayo de 2016.

[40] DroneKit by 3D Robotics. http://dronekit.io, consultado el 17 de Mayo de 2016.

[41] Documentación de DroneKit-Python. http://python.dronekit.io/, consultado el 17 de Mayo de 2016.

[42] International Organization for Standardization. Message Queuing Telemetry Transport (MQTT) v3.1.1. Technical Report 20922, ISO, 2016.

[43] Página oficial del protocolo MQTT. http://mqtt.org/, consultado el 17 de Mayo de 2016.

[44] Página oficial del protocolo Mosquitto en Eclipse. http://projects.eclipse.org/projects/technology.mosquitto/, consultado el 18 de Mayo de 2016.

[45] Página oficial del proyecto Paho en Eclipse. http://www.eclipse.org/paho/, consultado el 18 de Mayo de 2016.

[46] Z. Zheng W. Liu and K.-Y. Cai. Bi-level programming based real-time path planning for unmanned aerial vehicles. Knowledge-Based Systems, 2013.

[47] Drones Dodge Obstacles. http://news.mit.edu/2016/csail-drones-do-donuts-figure-eights-around-obstacles-0119, consultado el 20 de Mayo de 2016.

[48] S. Lee and J. R. Morrison. Decision support scheduling for maritime search and rescue planning with a system of UAVs and fuel service stations. International Conference on Unmanned Aircraft Systems (ICUAS), 2015.

[49] J. Tang M. Lundell and K. Nygard. Fuzzy Petri net for UAV decision making. Proceedings of the 2005 International Symposium on Collaborative Technologies and Systems, 2005.

[50] APM 2.5 and 2.6 Overview. http://ardupilot.org/copter/docs/common-apm25-and-26-overview.html, consultado el 23 de Mayo de 2016.

[51] Página web de Funduino. http://funduino.de/, consultado el 23 de Mayo de 2016.

[52] Seeed wiki - Grove Base Shield. http://www.seeedstudio.com/wiki/Grove_-_Base_Shield, consultado el 23 de Mayo de 2016.

[53] Seeed wiki - Grove - Temperature and Humidity Sensor Pro. http://www.seeedstudio.com/wiki/Grove_-_Temperature_and_Humidity_Sensor_Pro, consultado el 23 de Mayo de 2016.

[54] Seeed wiki - Grove - Barometer Sensor. http://www.seeedstudio.com/wiki/Grove_-_Barometer_Sensor, consultado el 23 de Mayo de 2016.

[55] Seeed wiki - Grove - Water Sensor. http://www.seeedstudio.com/wiki/Grove_-_Water_Sensor, consultado el 23 de Mayo de 2016.

[56] Seeed wiki - Grove - UV Sensor. http://www.seeedstudio.com/wiki/Grove_-_UV_Sensor, consultado el 23 de Mayo de 2016.

[57] Seeed wiki - Grove - Gas Sensor(MQ9). http://www.seeedstudio.com/wiki/Grove_-_Gas_Sensor(MQ9), consultado el 23 de Mayo de 2016.

[58] Seeed wiki - Grove - HCHO Sensor. http://www.seeedstudio.com/wiki/Grove_-_HCHO_Sensor, consultado el 23 de Mayo de 2016.

[59] Seeed wiki - Grove - Ultrasonic Ranger. http://www.seeedstudio.com/wiki/Grove_-_Ultrasonic_Ranger, consultado el 23 de Mayo de 2016.

[60] Seeed wiki - Grove - Digital Light Sensor. http://www.seeedstudio.com/wiki/Grove_-_Digital_Light_Sensor, consultado el 23 de Mayo de 2016.

[61] Amazon - Tira de LEDs roja. https://www.amazon.es/PsmGoods-coches-prueba-flexible-l%C3%A1mpara/dp/B00Y7J3WLO, consultado el 24 de Mayo de 2016.

[62] Raspberry Pi NoIR Camera - Infrared Camera. https://www.raspberrypi.org/products/pi-noir-camera/, consultado el 24 de Mayo de 2016.

[63] Raspberry Pi NoIR Camera - Infrared Camera. http://picamera.readthedocs.io/en/release-1.10/, consultado el 24 de Mayo de 2016.

[64] GitHub de la librería ArduinoJson. https://github.com/bblanchon/ArduinoJson, consultado el 24 de Mayo de 2016.

[65] Dr. Juan Boubeta Puig. Desarrollo Dirigido por Modelos de Interfaces Específicas de Dominio para el Procesamiento de Eventos Complejos en Arquitecturas Orientadas a Servicios., 2014.

[66] Capítulo 5 del libro de Esper. Referencia de EPL. http://www.espertech.com/esper/release-5.4.0/esper-reference/html/epl_clauses.html, consultado el 25 de Mayo de 2016.

[67] Lista de Módems USB 3G verificados para Raspberry Pi. http://elinux.org/RPi_VerifiedPeripherals#USB_3G_Dongles, consultado el 1 de Junio de 2016.

[68] Página de descarga de Mule ESB Runtime. https://developer.mulesoft.com/download-mule-esb-runtime, consultado el 2 de Junio de 2016.

[69] Página de descarga de Java Service Wrapper. http://wrapper.tanukisoftware.com/doc/spanish/download.jsp, consultado el 2 de Junio de 2016.

[70] Ardupilot - Conexión a Mission Planner. http://ardupilot.org/copter/docs/common-connect-mission-planner-autopilot.html, consultado el 6 de Junio de 2016.

[71] Ardupilot - Creación de misiones en Mission Planner. http://ardupilot.org/copter/docs/common-mission-planning.html, consultado el 6 de Junio de 2016.