TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE AUTOMATIZACIÓN DE CONTROL DE INVENTARIO BASADO EN MINIDRONES

AUTORA: IRENE MANGAS ROCA

TUTOR: MANUEL CONTERO GONZÁLEZ

COTUTORES: VALERY NARANJO ORNEDO
ADRIÁN COLOMER GRANERO

Curso Académico: 2017-18

Agradecimientos

Quiero aprovechar esta ocasión para agradecer a mis seres
queridos todo el apoyo que he recibido a lo largo de estos años.

Gracias a Valery y Adri,
que me dieron la oportunidad de trabajar en este proyecto.
Gracias por las horas dedicadas
y por depositar vuestra confianza en mí.

Gracias a mis amigos, especialmente a unos cuantos:
Alberto, Carlos, David, Félix, Jacob, Juanito, Rodrigo
y mis amigos de kárate.
La lista es larga, pero sin ellos los días serían más grises.
Gracias por hacer de la universidad
mi segundo hogar.

Gracias a Manolo, Toni, Vicent y otros profesores,
gracias a mis chicos de las cafeterías,
Deportes y Biblioteca.
Ellos también fueron compañeros en la carrera.

Gracias a los que hoy no están.
Me acuerdo de vosotros cada día.

Pero sobre todo,
gracias a mi familia y Adrián,
que han vivido tan intensamente como yo
esta etapa de mi vida.
Todo esto es para vosotros.
Gracias por quererme tanto.

Abstract

The present project aims to provide the first prototype of an automated warehose inventory system, ideated to suit the needs of an industrial plant. This machine-aided process is achieved with help of pilotless controlled drones, with them being monitored by Raspberry Pi. Drones capture data from QR codes attached to different stored products, in a motion planned trajectory for inventory tracking. Raspberry Pi as a control station manages both real-time streaming visualization and QR code decoding, on top of flight instructions. All of these processes run simultaneously, in real-time. Along with these features, is possible to measure accuracy of the official inventory book by comparing the obtained results with the warehouse database. The complete application comes with its own grafical interface for better usability and user experience.

Key words: inventory, warehouse, drone, minidrone, Raspberry Pi, QR code, bash, Python, real time.

Resumen

El presente proyecto desarrolla el primer prototipo de un sistema automatizado para la gestión de inventario en almacenes, diseñado para un entorno industrial. Esta automatización se consigue a través del uso de minidrones monitorizados desde Raspberry Pi. Los minidrones capturan información de códigos QR asignados a distintos productos, en una misión de reconocimiento con una ruta de vuelo previamente trazada. Raspberry Pi, como estación de control, se encarga tanto de la visualización de la imagen en directo como del reconocimiento de códigos QR, además del envío de instrucciones de vuelo. La información obtenida se guarda en un archivo de texto en dos formatos distintos. El programa se ejecuta a tiempo real, es decir, de forma simultanea al vuelo del dron. Posteriormente se puede realizar una comprobación con la base de datos del inventario, desde el mismo programa. El *software* completo cuenta con una interfaz gráfica que facilita al usuario la ejecución de este proceso.

Palabras clave: inventario, almacén, dron, minidron, Raspberry Pi, código QR, bash, Python, tiempo real.

Resum

El present projecte desenvolupa un primer prototip d'un sistema automatitzat per a la gestió d'inventari en magatzems, ideat per a un ecosistema industrial. Aquesta automatització s'aconsigueix mitjançant l'ús de minidrons monitoritzats des de Raspberry Pi. Els minidrons capturen la informació de codis QR assignats a diferents productes, en una missió de reconeixement amb una ruta aèria preprogramada. Raspberry Pi, com a estació de control, s'encarrega tant de la visualització d'imatges en directe com del reconeixement visual de codis QR, a més de les instruccions de vol. La informació obtinguda es guarda en un arxiu de text amb dos formats diferents. El programa es executa en temps real, és a dir, simultani al pilotatje del dron. Posteriorment es pot fer una comprobació amb la base de dades de l'inventari, des del mateix programa. El *software* complet compta amb una interfície gràfica que facilita a l'usuari l'execució d'aquest procés.

Paraules clau: inventari, magatzem, dron, minidron, Raspberry Pi, codi QR, bash, Python, temps real.

Índice general

Índice general IX

Índice de figuras XI

Índice de tablas XIII

Índice de código XV

I Memoria 1

1 Motivación y objetivos 3

1.1 Importancia de un sistema automatizado mediante minidrones . . . 3
1.2 Revisión bibliográfica . . . 5
1.3 Objetivos . . . 7

2 Materiales 9

2.1 Raspberry Pi . . . 9
2.2 Parrot Mambo . . . 13

3 Códigos QR 17

3.1 Descripción general del código QR . . . 17
3.2 Partes de la matriz QR . . . 18
3.3 Obstáculos en la identificación de códigos QR . . . 19
3.4 Codificación de códigos QR . . . 19

4 Diseño y programación del *software* 25

4.1 Requisitos del sistema . . . 25
4.2 Lenguajes . . . 27

4.3 Metodología y diseño del *software* . . . 28
4.4 Implementación del código . . . 32

5 Resultados 63
5.1 Interfaz gráfica . . . 63
5.2 Simulaciones y resultados . . . 66

6 Conclusión 71
6.1 Limitaciones del trabajo . . . 71
6.2 Conclusiones y líneas futuras . . . 72

II Presupuesto 73

7 Presupuesto 75
7.1 Cuadro de precios . . . 75
7.2 Cuadro de precios unitarios . . . 76
7.3 Cuadro de precios descompuestos . . . 76
7.4 Cuadro de presupuestos parciales . . . 78
7.5 Cuadro de presupuesto base de licitación . . . 78

III Anexos 79

A Anexo de código 81
A.1 Bash . . . 81
A.2 Python . . . 83

Bibliografía 95

Índice de figuras

1.1. *Internet of Things* y la industria. 5

2.1. Esquema con los componentes más destacados de Raspberry Pi. 10
2.2. Diagrama de bloques con especificaciones de Raspberry Pi. 11
2.3. Esquema con partes del dron, vista superior e inferior. Fuente: Parrot. 14
2.4. Métodos de carga del dron Mambo. Fuente: Parrot. 14
2.5. Ejes de rotación del dron 16

3.1. Ejemplo de código QR en el que pueden identificarse cada uno de los símbolos . . 18
3.2. Diagrama del proceso de codificación de un código QR 20
3.3. Diagrama extendido del *data encoding* 21
3.4. Patrones con información de localización y alineación. 21
3.5. Distribución de patrones (a) y dirección de escritura (b) en la matriz QR 22
3.6. Recorrido en zig-zag de la lectura. 22

4.1. Diagrama de interacciones entre el espacio usuario, el espacio kernel y el hardware. 28
4.2. Croquis de funcionamiento del programa 29
4.3. Funcionamiento real del programa 30
4.4. Esquema general de funcionamiento, relacionando cada una de las partes implicadas en el sistema. 31
4.5. Diagrama postproceso 42
4.6. Ejemplo de archivo csv, visto desde un visor de texto plano y desde Excel. 42
4.7. CSV y TXT son los formatos escogidos para la escritura de resultados. 43
4.8. Diagrama de detección de QR's a tiempo real 45

4.9. Diagrama de MACRO: obtención de QR's de nueva lista 46
4.10. Diagrama MACRO: creación de lista con elementos nuevos. Primer modelo. . . . 47
4.11. Diagrama MACRO: creación de lista con elementos nuevos. Modelo definitivo. . . 49
4.12. Funcionamiento del proceso de lectura y comparación de la información extraída por el escaner QR 51
4.13. Ejemplos de diferentes contenidos en las bases de datos. 51
4.14. Posibles formas de encontrar el código del producto, en fichero .txt. Ejemplo. . . 53
4.15. Diagramas extendidos de la lectura y comparativa de información en función del archivo de almacenamiento de datos 54
4.16. Funcion `main` de la comprobación de inventario 55
4.17. Funciones `compruebaTXT()` y `compruebaCSV()` de la comprobación de inventario . 56

5.1. Ventana principal de `droneInTheShell`. 64
5.2. Ventana con guía del programa, en `droneInTheShell`. 64
5.3. Ventana de ejecución de programas en `droneInTheShell`. 65
5.4. Ventana de apertura de ficheros en `droneInTheShell`. 65
5.5. Entorno de simulación para misión del dron. 66
5.6. Dron haciendo una misión de reconocimiento en la simulación. 67
5.7. Fotograma del *streaming* de vídeo. 67
5.8. Otro fotograma del *streaming* de vídeo. 68
5.9. Ejemplo poniendo a prueba el algoritmo de comparacion. 69

Índice de tablas

2.1. Especificaciones del minidron Parrot Mambo 13

7.1. Cuadro de mano de obra 75

7.2. Cuadro de materiales 75

7.3. Cuadro de equipos 76

7.4. Cuadro de precios unitarios 76

7.5. Cuadro de precios descompuestos 77

7.6. Cuadro de presupuestos parciales 78

7.7. Cuadro de presupuesto base de licitación 78

Indice de codigo

1. Función `multiproceso` en el programa principal. 32
2. Función `redbaron` en el programa principal. 33
3. Eliminación de programas de forma localizada mediante `redbaron`. 34
4. Definición de funciones de vuelo para el dron. 36
5. Secuencia de vuelo del dron en `vueling.py` 36
6. Función `snapshot` para captura y almacenado de imágenes, escrita en bash. . . . 38
7. Función `leeFicheroCSV` de la aplicación *Comprueba inventario*. 56
8. Función `obtenListaListasCSV` de la aplicación *Comprueba inventario*. 57
9. Función `procesaLineasTXT` de la aplicación *Comprueba inventario*. 58
10. Función `recuperaInfoTXT` de la aplicación *Comprueba inventario*. 59
11. Función `recuperaInfoCSV` de la aplicación *Comprueba inventario*. 60

Parte I

Memoria

Capítulo 1

Motivación y objetivos

En este capítulo se presenta el marco teórico en el que se ubica el proyecto y cómo busca resolverse el problema de gestión de inventario, enumerando sus principales problemas y proponiendo una solución mediante dos de las tecnologías más populares del momento: minidrones y Raspberry Pi.

1.1 Importancia de un sistema automatizado mediante minidrones . . . 3
1.1.1 Necesidad de la gestión de inventario . . . 3
1.1.2 El papel del dron en la industria 4.0 . . . 4
1.1.3 Internet de las Cosas en la industria 4.0 . . . 4
1.2 Revisión bibliográfica . . . 5
1.2.1 Análisis en cuanto a materiales utilizados . . . 5
1.2.2 Análisis en cuanto a lenguajes de desarrollo . . . 6
1.2.3 Conclusión . . . 7
1.3 Objetivos . . . 7

1.1 Importancia de un sistema automatizado mediante minidrones

1.1.1 Necesidad de la gestión de inventario

La gestión de inventarios es una de las problemáticas de mayor importancia en la industria, un sector que debe adaptarse a las fluctuaciones de demanda en un mercado globalizado y competitivo. Cada vez se destinan más esfuerzos a mejorar la gestión de *stocks* y reducir los costes de inventario. Uno de los retos a los que se enfrentan habitualmente es la discrepancia entre la base de datos del inventario y el estado real del mismo: cantidades erróneas, operaciones no registradas, registros con operaciones inexistentes y otros problemas similares. Un mal inventario supone pérdida de poder de planificación, aumento de costes e incluso dar mal servicio a los clientes.

En la actualidad existen distintos métodos que buscan dar solución a este problema. Propuestas como el uso de *software* especializado en gestión de *stocks* o realizar periódicamente recuentos de inventario permitirían corregir posibles inconsistencias. Esta tarea suele requerir un número importante de trabajadores si quiere realizarse en un tiempo razonable, que no corte el flujo

de trabajo habitual. En cualquiera de los casos, existe la necesidad de un sistema que optimice el tiempo dedicado y los recursos que exige este control. Aunque el *software* específico permite llevar un control más ajustado de entradas y salidas, aún se necesitan trabajadores físicos para una inspección, esta vez exhaustiva, del almacén.

1.1.2 El papel del dron en la industria 4.0

El sector industrial está registrando un creciente interés por la integración de drones en almacenes y fábricas. La industria 4.0 busca ir más allá de la automatización y computerizado de sus plantas: quiere implementar el uso de sistemas ciberfísicos e interconectarlos mediante Internet. Los objetivos son mayor adaptabilidad a la demanda del mercado, mejor comunicación entre los distintos elementos de la cadena y más eficiencia en la gestión de recursos.

Por sus características, el dron tiene potencial en campos como el transporte y distribución. Ya se ha expresado la intención de utilizarlo como intermediario en el abastecimiento de medicinas y vacunas en casos de emergencia. Otros ejemplos, algunos de gran impacto económico, son el uso en la agricultura, reforzando el concepto *SmartAgro*; en el sector energético, para mejorar la prevención de riesgos laborales; y para potenciar la eficacia y eficiencia de los equipos de seguridad del estado.

Gracias a los avances tecnológicos en los campos de control y ciencia de materiales, unido al resurgir del movimiento *maker*, los drones están experimentando un fenómeno de democratización. En los últimos cinco años sus ventas y modelos han aumentado notablemente y se han abaratado sus costes, una revolución para una tecnología antes reservada para fines militares.

Las soluciones basadas en drones, que operarían de forma autónoma o controlados por un operario, podrían acelerar el proceso de comprobación e incluso realizarlo con mayor periodicidad, con un impacto económico menor al actual. Además, permitiría que los trabajadores pudieran dedicarse sin interrupciones a sus tareas habituales, a diferencia de lo que ocurre con los métodos convencionales de inspección.

La mayoría de minidrones de espacios interiores o *indoor* tienen características muy útiles para realizar esta labor: videocámaras, controladores automáticos de vuelo, cuerpos robustos contra golpes y sensores ópticos y de geolocalización.

1.1.3 Internet de las Cosas en la industria 4.0

En la última década el mercado de Internet de las Cosas (IoT por sus siglas en inglés) no ha parado de crecer y se pronostica que seguirá creciendo en los próximos años.

Internet of Things engloba aquellas aplicaciones tecnológicas que permiten intercontectar objetos (cosas) entre ellos y con la red (Internet). Estos objetos cuentan con sensores, procesadores y otros dispositivos para el control y monitorización de forma remota. Además, permiten al usuario acceder a la información y llevar un registro de las mediciones.

Paulatinamente tanto industria como servicios están incorporando soluciones basadas en IoT a sus negocios: cámaras de seguridad, automovilística, *wearables*, medidores, teleasistencia, identificadores inteligentes o control logístico, como es el caso de este proyecto.

Figura 1.1: *Internet of Things* y la industria.

Pero para interconectar todos estos dispositivos es necesaria una unidad de control, con acceso a Internet y suficiente potencia como para servir de interfaz de monitorizado. Un microcontrolador puede que solucione tareas puntuales, como gestionar la información de un sensor, pero no tiene la capacidad de gestionar múltiples programas. Un ordenador, sin embargo, puede que se exceda en características. Además, es mucho menos portable que un microcontrolador y consume más energía.

Sin embargo, existe una solución intermedia: los miniordenadores. Raspberry Pi es un ejemplo de ellos y líder en esta categoría.

1.2 Revisión bibliográfica

En este apartado se realizará un análisis documental de los artículos que se han usado como referencia para el proyecto. El análisis se trabajará desde distintos enfoques, introduciéndose los materiales y métodos empleados. Se cierra con la conclusión de cuál es la metodología escogida dadas las condiciones del trabajo.

1.2.1 Análisis en cuanto a materiales utilizados

Para el caso de este proyecto, se desea que la Raspberry Pi retransmita en directo las imágenes obtenidas por el dron a través de una pantalla, de modo que el trabajador pudiera hacer un seguimiento del proceso de reconocimiento.

En trabajos previos como Diachok, Dunets y Klym (2018) se emplea Raspberry Pi 2 para la detección de códigos de barras en un *stream* de vídeo, con la diferencia de que se utilizan dos ordenadores simultáneamente: Raspberry Pi y un segundo equipo al que se le transmiten los datos mediante VCN Viewer. Las imágenes provienen de una cámara USB, como es el accesorio "Raspicam". Esta metodología necesitaría, por tanto, el uso de un segundo ordenador para el trabajo de control y post-procesado de imagen, lo que la convierte en una opción menos compacta y más costosa.

Las soluciones que trabajan conjuntamente con Raspberry Pi y un segundo equipo son recurrentes en la mayoría de fuentes consultadas. Santos y col. (2017) propone la conexión de Raspberry Pi, montada sobre cada uno de los minidrones de una flota, a través de WiFi con la torre de control. Desde esta torre se monitorizan los drones y se consigue la visualización en directo, esta vez sin

necesidad de utilizar VCN. Aunque ya se incorpora el elemento dron en el proyecto, se sigue relegando a Raspberry Pi un papel de microcontrolador.

Con el objetivo de utilizar Raspberry Pi como un ordenador en sí mismo, se opta por un dron que implemente un microprocesador que pueda controlarse de forma remota. Habsi y col. (2015) han trabajado previamente con un dron AR Drone de la empresa Parrot para vuelo en interiores, salvando las diferencias con el trabajo aquí presentado. Mientras que Habsi y col. (2015) han desarrollado una SDK propia para Matlab y Simulink, en este proyecto se usa el kit de desarrollo del propio Parrot mediante la librería de Python `pyparrot`.

Otros autores también han trabajado con AR Drone para vuelos de interiores, como Chakrabarty y col. (2016), en esta ocasión para el seguimiento de objetos en movimiento. En el artículo se utiliza el software de simulación que proporciona Parrot para una comparativa con los resultados del vuelo real.

1.2.2 Análisis en cuanto a lenguajes de desarrollo

Python es un lenguaje habitual en el desarrollo de *scripts* en Raspberry Pi 3, ya que es uno de los lenguajes que incorpora de forma nativa el dispositivo y es sencillo de utilizar. Por ejemplo, Fuster Baggetto (2017) utiliza Python y OpenCV para el desarrollo de un robot autónomo que debe resolver distintas pruebas usando algoritmos para visión artificial. Fuster Baggetto (2017) destaca las virtudes de Python respecto a otros lenguajes tan potentes como C++, precisamente por su sencillez, variedad de estructuras de datos y extensibilidad a través de librerías.

Para la programación en Python se ha utilizado el paquete `pyparrot` para control de minidrones Mambo, tal y como se detalla en la Sección 2.2.2. Cubre tanto el control de instrucciones de vuelo como la visión mediante cámaras frontales y ventrales. Sin embargo, esta última utilidad no estaba completamente operativa en el momento de desarrollo de este proyecto. Al tratarse de una librería que todavía se encuentra en fase de desarrollo, se exploraron otras liberías que brindaran las siguientesf funcionalidades:

- Retransmisión de vídeo en streaming del minidron a la Raspberry Pi.
- Captura de imágenes del *stream*.
- Análisis, preferible en tiempo real, de dichas imágenes.

La aparición de problemas adicionales dificultaron la resolución completa mediante Python: las características de hardware de Raspberry Pi, el problema de Python para manejar múltiples hilos de ejecución o librerías que requerían demasiados recursos (como fffmpeg y VLC). Los problemas de Python con las aplicaciones multihilo los discute Jeff Knupp en *Python's Hardest Problem* (2018), atribuyendo el problema al intérprete GIL[1] de Python.

Dado este inconveniente, se buscó en la documentación de Raspberry Pi aplicaciones de reproducción de vídeo que estuvieran optimizadas para ella. Allí se encontró el reproductor `omxplayer`, que funciona mediante línea de comandos. Es por ello que se decidió optar por *shell scripts* para realizar los procesos de *streaming* de vídeo y de multitarea (vuelo, vídeo, detección de códigos QR) ya que no presentaba problemas, a diferencia de Python. Para completar esta tarea fue

[1] El GIL o *Global Interpreter Lock* es un bloqueo a nivel de intérprete, que impide que varios hilos de ejecución se ejecuten paralelamente en el mismo intérprete.

necesario extraer imágenes mediante otro programa CLI (*Command Line Interface*). `raspi2png` era el más ampliamente recomendado.

1.2.3 Conclusión

Para facilitar la experiencia de vuelo, tanto en la precisión de las instrucciones como en la simplicidad de envío de las mismas, se optó por el modelo Mambo de Parrot. Entre sus ventajas destaca que es ligero, de bajo coste, con conexión WiFi y Bluetooth. Es resistente a los golpes y sigue disponiendo del kit de desarrollo de software que ya han probado, satisfactoriamente, otros investigadores. Una opción razonable para un primer prototipo de coste asequible que permita trabajar con Raspberry Pi.

Tras un análisis detallado de las posibilidades, se opta por un *script* multitarea escrito en `bash` que realizará las tareas de reproducción de vídeo, captura de imágenes y almacenado de las mismas. Desde el mismo *script* se ejecutan las tareas de vuelo y detección de QRs, que son a su vez dos *scripts* escritos en Python 3.5.

1.3 Objetivos

El objetivo de este proyecto es el diseño e implementación de un primer prototipo para un sistema de control de inventario, automatizado mediante la acción conjunta de Raspberry Pi y minidrones.

En primer lugar, se desarrollarán dos aplicaciones básicas y directamente relacionadas con el dron: el control del vuelo y la recepción y visualización del *streaming* de imágenes que graba la cámara FPV. Para ello se especificará una ruta de vuelo en un entorno que simula un almacén, con los embalajes identificados mediante códigos QR con información del tipo de producto[2], fecha de llegada y número de elementos. El *streaming* de imágenes se enviará a Raspberry Pi mediante el protocolo RTSP y se reproducirá en una pantalla LCD perteneciente a la Raspberry.

A continuación, se elaborará una aplicación que permita la captura y almacenado de imágenes, *snapshots* del visionado en directo del dron. Estas capturas deben ser, a medida que son almacenadas en un directorio, analizadas por un escáner QR que se incluye en un *script* escrito en Python. En caso de detección de algún código, se almacenará el resultado en un archivo de texto que permita una posterior comparación con la base de datos del inventario.

Con todo lo anterior dicho, cabe destacar que estos objetivos no deben comprometer el rendimiento de la Raspberry Pi, debiéndose efectuar en un tiempo aceptable y con el mínimo de interacción por parte del usuario. El *software* se ha ideado para que funcione de forma optimizada, asegurando que las distintas tareas se ejecutan sincronizada y concurrentemente.

Por último, teniendo en mente la usabilidades y la compactabilidad, este sistema incluirá una interfaz gráfica que facilite la interacción del usuario con el *software* y le ofrece la posibilidad de efectuar la comparativa automática con la base de datos del inventario.

[2]Mediante un identificador único alfanumérico, obtenido de productos reales en Amazon.

Capítulo 2

Materiales

En el presente capítulo se desarrollan los materiales utilizados en el proyecto: Raspberry Pi y el minidron Parrot Mambo. Se especifican las partes de cada uno de los elementos y el papel que realizan en el trabajo.

2.1 Raspberry Pi 9
2.1.1 Introducción 9
2.1.2 Características físicas 9
2.1.3 Sistema operativo, instalaciones y entornos de desarrollo 11
2.2 Parrot Mambo 13
2.2.1 Descripción y especificaciones 13
2.2.2 SDK y la librería `pyparrot` 14

2.1 Raspberry Pi

2.1.1 Introducción

Raspberry Pi es un proyecto que nace en 2012 de la mano de Raspberry Pi Foundation, con el objetivo de crear un miniordenador de bajo coste y enfoque educacional. Sin embargo, su potencia, tamaño y coste han hecho que esta SBC (*Single Board Computer* u ordenador de placa reducida) sobrepase la barrera académica y se proponga como solución a sistemas de automatización basados en internet en entornos industriales.

2.1.2 Características físicas

Raspberry Pi es un SBC de caracter generalista. A diferencia de los ordenadores tradicionales, cuyas placas bases están conectadas a tarjetas de extensión, Raspberry Pi y otras SBC contienen la mayor parte de los componentes embebidos en una única tarjeta. En la Figura 2.1 se puede observar cómo se incluyen en la pequeña placa los controladores de WiFi y Bluetooth, las conexiones a periféricos mediante USB, RJ45 o HDMI, el chip que contiene la CPU y GPU[1], entre otros dispositivos y utilidades.

[1]Mientras que la CPU se encarga del procesamiento general, la GPU es de carácter específico y está dedicada al procesamiento gráfico.

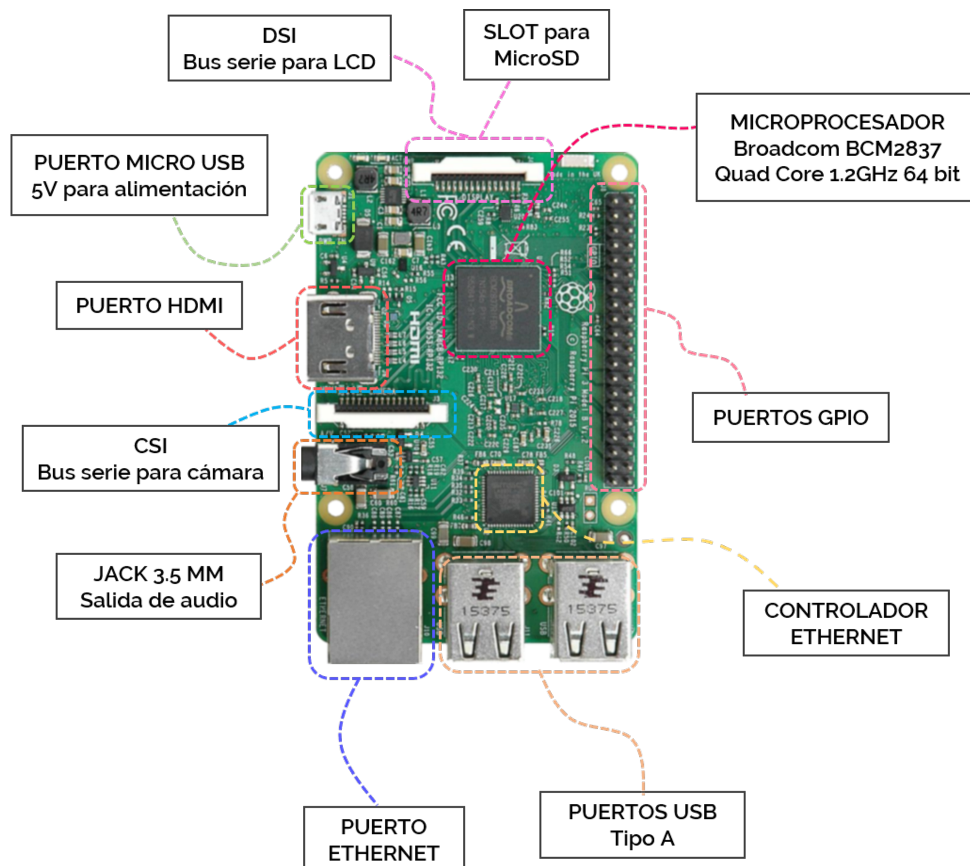


Figura 2.1: Esquema con los componentes más destacados de Raspberry Pi.

Los SBC destacan por sus dimensiones reducidas, bajo consumo energético y coste económico, lo que les confiere gran capacidad de integración y buen control del consumo de potencia eléctrica. Son componentes que se adaptan muy bien a las exigencias de la mayoría de aplicaciones IoT, bien integrados en un sistema embebido o como interfaz de control.

Raspberry Pi es una SBC entre tantas otras opciones que han surgido tras su debut. Los competidores más directos de esta placa son OrangePi, BananaPi, BeagleBone Black o ODroid. Sin embargo, aunque estas puedan tener su segmento de mercado, Raspberry Pi sigue siendo una de las opciones preferidas para los desarrolladores. Las razones son fiabilidad, comunidad y calidad de documentación.

En la Figura 2.2 se añade el diagrama de bloques que resume las especificaciones técnicas del modelo 3 B de Raspberry Pi. El diagrama está dividido según funcionalidades.

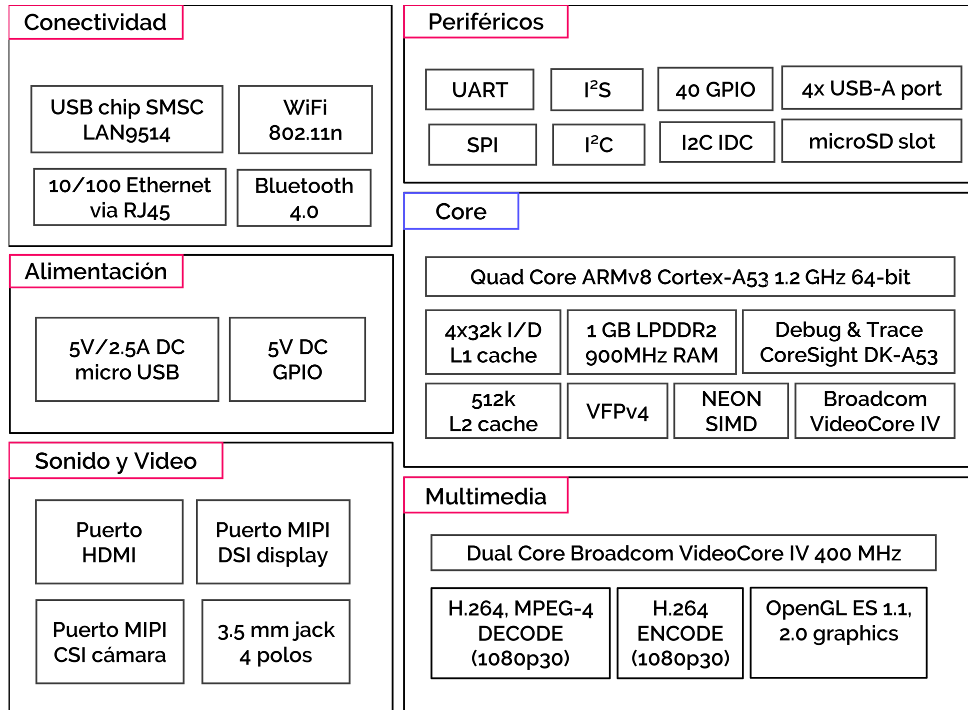


Figura 2.2: Diagrama de bloques con especificaciones de Raspberry Pi.

2.1.3 Sistema operativo, instalaciones y entornos de desarrollo

El sistema operativo escogido es Raspbian Stretch, dado que es el recomendado por la Raspberry Pi Foundation al ser el mantenido por la fundación.

Raspbian es un distribución basada en Debian[2], que en su versión de interfaz gráfica utiliza LXDE como escritorio. Es una distribución ligera y con enfoque educativo, con herramientas de desarrollo como Python, Scratch, Pygame y otros. Su principal ventaja, como ocurre con otros sistemas GNU/Linux, es que dispone de un repositorio de descarga de programas. Este repositorio es un espacio organizado en el que se puede encontrar *software* optimizado para la Raspberry Pi y que se descarga mediante la herramienta APT (*Advanced Packaging Tool*). Para ello, las instrucciones se deben escribir desde el terminal de comandos de Raspberry Pi.

Por ejemplo, para instalar `omxplayer` desde el repositorio:

```
1 pi@raspberry:~$    sudo apt-get install omxplayer
```

Para actualizar programas que ya estén instalados en Raspberry Pi, se utiliza la opción `upgrade`:

```
1 pi@raspberry:~$    sudo apt-get upgrade paquete1 paquete2 paqueteN
```

Después de añadir programas del repositorio, es aconsejable actualizar la lista de paquetes:

```
1 pi@raspberry:~$    sudo apt-get update
```

[2]Debian es una distribución del sistema operativo GNU/Linux, apreciado por su robustez y extensa biblioteca de funcionalidades.

En caso de querer desinstalar alguno, se ejecutaría la siguiente línea:

```
1 pi@raspberry: ~$    sudo apt-get uninstall paquete
```

A veces, los programas que se desean instalar no están disponibles en el repositorio. Un ejemplo claro es OpenCV. Actualmente existen distintos métodos para la instalación de esta macrolibrería de visión artificial: mediante `conda`[3], desde binarios (*pre-built binaries*) o mediante una instalación desde la fuente (*source installation*). Aunque en el desarrollo de este proyecto no se ha utilizado la librería OpenCV, se recomienda la lectura de su instalación para posteriores mejoras del sistema, ya que es uno de los paquetes más extendidos y potentes para el tratamiento de imagen.

Python tiene su propio instalador de paquetes, ya que los paquetes en Python no son exactamente programas, sino librerías de utilidades. Dos de los instaladores más usados son `conda` y `pip`, que son mútuamente excluyentes. Aunque `conda` no es exclusivo de la distribución (Python) Anaconda, es en gran parte conocido por estar incluido en esta.

Por ejemplo, para instalar un paquete en Python mediante `conda`, se ejecutaría la siguiente instrucción:

```
1 pi@raspberry:~$    conda install paquete
```

Para poder usar `conda` en Raspberry Pi hay que instalar previamente Miniconda. Para instalar la última versión disponible para la arquitectura ARM lo recomendable es visitar los repositorios de `conda`. En este ejemplo se instala Miniconda3 para armv7:

```
1 pi@raspberry:~$    sudo /bin/bash Miniconda3-latest-Linux-armv7l.sh
```

Para instalar un paquete mediante `pip` hay dos opciones.

1. `pip` para paquetes en Python 2.
2. `pip3` para paquetes en Python 3.

Este proyecto se ha desarollado con Python 3 y, por tanto, `pip3` sería la opción a utilizar. Por ejemplo, para instalar la librería de funciones matemáticas `numpy`, se escribiría en el terminal:

```
1 pi@raspberry:~$    pip3 install numpy
```

Es probable que haya que preceder esta instrucción con `sudo`, para conceder permisos de administrador o *superuser*.

Los programas CLI (*Command Line Interface*) utilizados en este trabajo, como es el caso de `yad`, `omxplayer` y `raspi2png`, se han instalado satisfactoriamente mediante `sudo apt-get install paquete`. Del mismo modo, para la instalación de paquetes en Python se ha utilizado `sudo pip3 install paquete` sin ningún problema.

[3] `conda` es un gestor de paquetes y de entornos virtuales muy potente, aunque incompatible con algunas herramientas muy extendidas como `pip`.

Raspbian dispone del entorno de desarrollo IDLE para Python 3. Es un entorno de desarrollo sencillo que incluye un terminal para la ejecución de *scripts* o líneas de código, al mismo tiempo que incorpora un editor para escribir programas más extensos. IDLE está disponible también para Windows 10 y Ubuntu 16.04, con lo que se permite trabajar en distintos sistemas operativos indistintamente sin problemas de traducción[4].

Las opciones para desarrollo de *shell scripts* son más variadas. Raspberry Pi Foundation sugiere distintas opciones: el editor de texto Leafpad y los editores CLI Vim y Emacs. Se ha trabajado esencialmente con Leafpad por su comodidad.

2.2 Parrot Mambo

2.2.1 Descripción y especificaciones

Parrot Mambo FPV es un minidron fabricado por la compañía francesa Parrot, orientado a un primer contacto con el mundo de los drones. Es robusto, ligero y fácil de pilotar. Cuenta con varios accesorios, entre los que destacan la cámara frontal FPV y unas gafas de inmersión para dron. También cuenta con un mando *joystick* para pilotaje de Parrot Mambo en tiempo real, con tecnología Bluetooth 4.0 BLE y alcance de 100 m. Sus especificaciones técnicas están resumidas en la tabla 2.1.

Tabla 2.1: Especificaciones del minidron Parrot Mambo

Características	Detalles
Dimensiones	18 x 18 cm
Peso	63 gramos sin cámara 73 g con cámara
Sensores	Giróscopo de tres ejes Acelerómetro de tres ejes Ultrasonidos Presión
Energía	Batería de 660mAh LiPo 10 minutos de autonomía de vuelo 30 minutos de carga (2,1A)
Cámara	Streaming y grabación HD 720p 30fps FOV 120
Conectividad	WiFi Bluetooth 4.0 BLE, 30m
SDK	OS Linux

[4]Se ha observado que algunas herramientas pueden generar problemas, ya que los saltos de línea están codificados de forma distinta en Windows y GNU/Linux

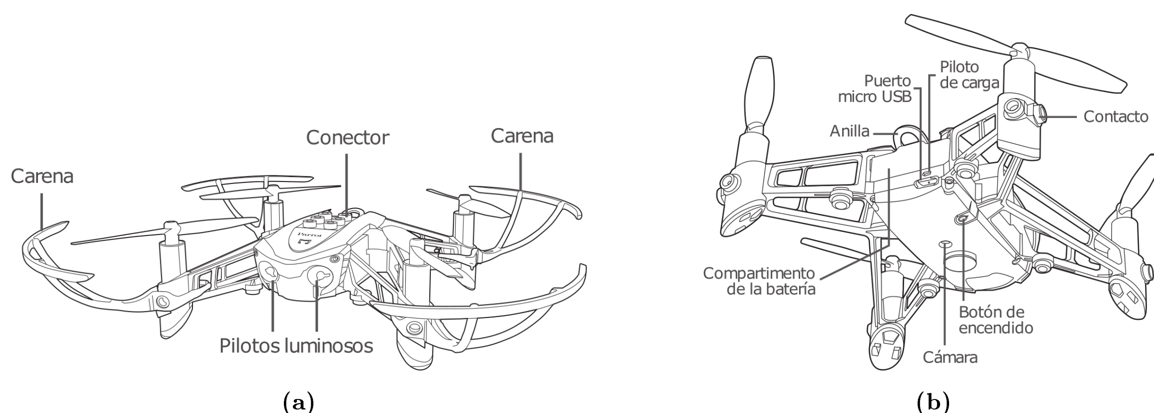


Figura 2.3: Esquema con partes del dron, vista superior e inferior. Fuente: Parrot.

El dron está fabricado esencialmente de plástico, lo que le confiere ligereza y resistencia a los golpes. Incorpora unas carenas o protecciones contra impactos a la hélice y el controlador manda la orden de aterrizaje si sufre uno de ellos. También cuenta con unas almohadillas para proteger los brazos en el aterrizaje. Un esquema con los elementos más relevantes se recoge en la Figura 2.3, mientras que en la Figura 2.4 se muestran las opciones de carga del dispositivo: mediante un cargador de baterías que proporciona el fabricante o conectándolo via USB al ordenador.

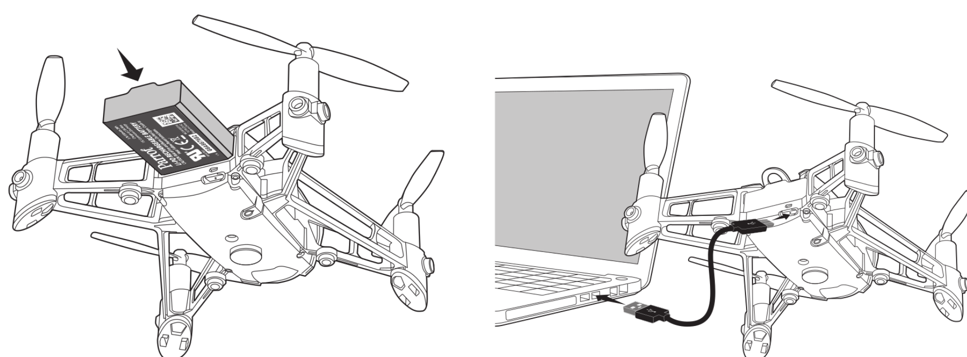


Figura 2.4: Métodos de carga del dron Mambo. Fuente: Parrot.

2.2.2 SDK y la librería pyparrot

La característica más interesante que incorpora Mambo y el resto de drones de la familia Parrot es su SDK (*Software Development Kit*) de código abierto. Esta SDK permite el desarrollo de aplicaciones que interaccionen con el dron, proveyendo de librerías para sistemas operativos basados en Unix, Android e iOS. Las librerías están escritan principalmente en C, aunque existen otros proyectos derivados de esta misma SDK que traducen estas librerías a Python, Javascript o Java.

Las librerías que proporciona la SDK son muy completas, aunque no todas las utilidades están disponibles para cualquier minidron. Para Mambo se dispone de controles referentes al encendido, aterrizaje, vuelo, conexión y desconexión de redes WiFi y Bluetooth, envío de *stream* de imágenes mediante RTSP, control de geoposición (no probado en este proyecto) y control de sensores (no probado en este proyecto). Es posible que más opciones se hayan incluido en momentos posteriores al desarrollo del proyecto.

La librería `pyparrot` es un *binding* o adaptación de esta SDK para el lenguaje de programación Python (concretamente Python 3.6, aunque se puede trabajar con la versión Python 3.5, que es la que viene instalada por defecto en Raspberry Pi 3 B). Es una librería creada por la doctora McGovern, profesora de la Escuela de Informática de la Universidad de Oklahoma.

`pyparrot` organiza sus funciones en varios módulos, entre ellos `Mambo`, que reúne todos los comandos básicos para controlar el vuelo del dron. También cuenta con funciones que piden datos sobre los sensores del dron y que sacan fotos durante el vuelo. Cabe destacar que, a pesar de que la librería extiende sus funcionalidades al campo de la visión del dron, se ha preferido no utilizarla para ese cometido. Se recomienda la lectura de la documentación para un conocimiento más profundo de cada una de las funciones disponibles, no sólo las del módulo Mambo. Ver McGovern 2017b.

Como la autora comenta en el apartado `Vision`, en la documentación de `pyparrot`, existen dos posibilidades para capturar imágenes de la visión del dron: mediante `ffmpeg` y almacenándolas en local, o mediante `vlc` y retransmitiéndolo en directo. Cada alternativa tiene sus desventajas.

- `ffmpeg` almacena cada uno de los fotogramas en local. A treinta fotogramas por segundo, con Raspberry Pi es fácil que el hilo de imágenes se congele y dejen de llegar imágenes correctas. También introduce un retardo importante en el programa.
- `vlc` sólo permite reproducir el *streaming* de imágenes y no permite ejecutar otros hilos simultáneamente. Hay que incrustar las indicaciones de vuelo en el programa principal que maneja la interfaz gráfica. Además, en Raspberry Pi VLC consume muchos recursos.

A continuación, se enumeran algunas de las funciones más utilizadas en el desarrollo de este proyecto.

Nota sobre las versiones:
En este proyecto se utiliza la versión 1.4 de `pyparrot`, en la que se sigue utilizando el módulo Mambo. En la actualización a la versión 1.5.0 el módulo Mambo pasa a llamarse Minidrone y tiene funcionalidades extra para otros drones.

- `connect(num_retries)` y `disconnect(num_retries)` son funciones de conexión y desconexión con el Mambo. En general es una buena práctica utilizarlas para conseguir conexiones y desconexiones limpias. `num_retries` es el número de veces que se debe intentar esta conexión/desconexión.
- `safe_emergency(timeout)` envía una señal de aterrizaje de emergencia al dron, hasta que el dron devuelve la señal de que no está volando. `timeout` indica después de cuánto tiempo debe dejar de enviar dicha señal.
- `is_landed(timeout)` indica que el dron ya aterrizado.
- `safe_land(timeout)` asegura que el dron aterriza.
- `safe_takeoff(timeout)` permite un despegue seguro.

- `smart_sleep(timeout)` mantiene “quieto” al dron en la posición que se encuentre, bien en el aire o bien en el suelo. `timeout` indica el número de segundos que debe permanecer pausado.
- `ask_for_state_update()` actualiza el estado de los sensores del dron y envía la información a Raspberry Pi o el dispositivo de control.
- `fly_direct(roll, pitch, yaw, vertical_movement, duration=None)` permite cambiar el estado de vuelo del dron. Existen cinco parámetros para ajustar que son descritos a continuación. En la Figura 2.5 se observan los ejes de referencia para las rotaciones.

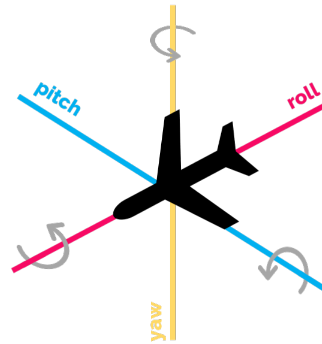


Figura 2.5: Ejes de rotación del dron

1. `roll`: velocidad de alabeo. Valor entre -100 y 100.
2. `pitch`: velocidad de cabeceo. Valor entre -100 y 100.
3. `yaw`: velocidad de guiñada. Valor entre -100 y 100.
4. `vertical_movement`: velocidad de movimiento en vertical. Valor entre -100 y 100.
5. `duration`: duración en segundos de la instrucción de vuelo.

Capítulo 3

Códigos QR

Dada la relevancia y complejidad de los códigos QR, se dedica un capítulo especial a su forma y al análisis de su codificación, que es la base de la librería para detección de códigos de barras `zbar`.

3.1 Descripción general del código QR 17
3.2 Partes de la matriz QR 18
3.3 Obstáculos en la identificación de códigos QR 19
3.4 Codificación de códigos QR 19
3.4.1 Análisis de la información 19
3.4.2 *Data encoding* 20
3.4.3 Codificación de corrección de errores 20
3.4.4 Estructuración final del mensaje 20
3.4.5 Colocación en la matriz QR 21
3.4.6 Enmascaramiento de datos 22
3.4.7 Información de formato o versión 23

3.1 Descripción general del código QR

El código QR, o *Quick Response code*, es un tipo de código de barras de gran difusión en esta última década. Al ser bidimensionales, son más compactos y tienen más capacidad de información que un código de barras convencional.

Los códigos QR presentan ciertas ventajas frente a otros tipos de identificadores.

- Distintos tamaños en función de su versión, desde 21 x 21 hasta 177 x 177.
- Soporta un gran número de caracteres, con un máximo de 7089.
- Pueden ser leídos en cualquier dirección.
- Soportan ideogramas, como son los kanjis japoneses.
- Son resistentes a la suciedad y al daño, gracias a su capacidad de recuperación de información[1].

[1] Para conseguir esta propiedad se implementa el código Reed-Solomon, un método de corrección de errores habitual en CDs y otros soportes ópticos, basado en los códigos de bloque.

En febrero de 2015 entró en vigor un nuevo estándar regulado por ISO, ISO/IEC 18004:2015, que especifica la simbología, los métodos de codificación y decodificación de la información y, adicionalmente, información sobre su identificación automática, cómo capturar la información según con qué metodología y nuevas especificaciones en relación a las nuevas versiones de QR.

3.2 Partes de la matriz QR

Los códigos QR tienen una morfología estipulada, resultado de la combinación de cinco tipos de símbolos: posición, alineamiento, líneas de dimensión, información variable y cuerpo. Estos símbolos están identificados en la Figura 3.1.

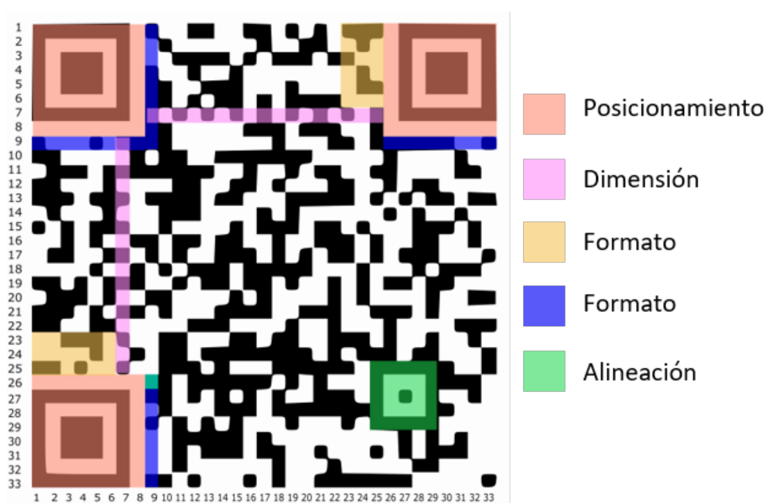


Figura 3.1: Ejemplo de código QR en el que pueden identificarse cada uno de los símbolos

Los símbolos de posicionamientos y alineación ubican al lector de QR y determinan la posición en la que están leyendo la información. Visualmente pueden reconocerse como dos rectángulos concéntricos, siendo los de posicionamiento los dispuestos en las esquinas (en coral) y el de alineamiento el más pequeño (en verde).

Las líneas de dimensión identifican el tamaño en bytes de los símbolos del cuerpo. Visualmente son segmentos (en rosa) de ocho cuadrados entre los símbolos de posicionamiento.

Los datos variables informan del nivel de corrección de errores (L, M, Q o H), la versión de QR, el nivel de indexado de la matriz de datos y otra información que es, como indica su nombre, variable. Suelen rodear o estar próximos a los símbolos de posición (en azul), al igual de las líneas de dimensión.

3.3 Obstáculos en la identificación de códigos QR

Existen variables que pueden dificultar la identificación de los códigos en un entorno real, como es el de un almacén.

Los QR se incluyen para identificar productos en etiquetas, cajas, tarjetas y otros soportes que contienen información irrelevante para su lectura, o dicho con otras palabras: ruido. Desde caracteres alfanuméricos hasta cenefas de un envoltorio, la complejidad de detección del objeto aumenta a mayor sea el ruido de la imagen. Algunos casos, como es el etiquetado de productos alimentarios, suponen un auténtico reto para el desarrollador.

Para una decodificación óptima es necesaria una localización también óptima del código, que reconozca perfectamente los contornos y que extraiga adecuadamente la información. Los símbolos de posicionamiento son buenos puntos de partida para esta tarea.

Otro problema recurrente es la distorsión de perspectiva, o que el objeto (y su entorno) se deforme debido a la proximidad del mismo con respecto al observador. La forma cuadrada característica del código QR pasa a ser un romboide o trapecio. La solución a este problema es sencilla: se detectan los símbolos de posicionamiento y se traza un triángulo cuyos vértices sean dichos símbolos. A continuación, sabiendo los ángulos del triángulo y por paralelismo, se obtiene la cuarta esquina del objeto y se transforma el código a su versión no distorsionada.

Finalmente, la iluminación puede dificultar la detección del código QR. Parámetros como el contraste, la intensidad de la fuente de luz, la reflectividad, las sombras e incluso el color del objeto afectan a la imagen que se obtiene como resultado. A mejores condiciones de iluminación, menor procesado mediante *software* y, por tanto, menor coste.

3.4 Codificación de códigos QR

Una vez introducido qué es un código QR y las partes que lo componen, se procede a describir cómo se construye uno de estos códigos a partir de un *string* de datos. El proceso de codificación se compone de los pasos mostrados en el siguiente diagrama (Figura 3.2) y que se describen a continuación.

3.4.1 Análisis de la información

Este paso, previo a la codificación en sí, consiste en analizar el mensaje que se desea transmitir, para dar con el tipo de codificación que resulta más eficiente en cada caso. La codificación se basa en obtener una cadena de bits (1's y 0's) a partir de la cadena de caracteres que compone el mensaje original. La codificación más eficiente será la que consiga el menor tamaño en la cadena de bits para el mensaje en cuestión. Aunque existen otros tipos/modos de codificación, los principales son: numérico, alfanumérico, byte y kanji.

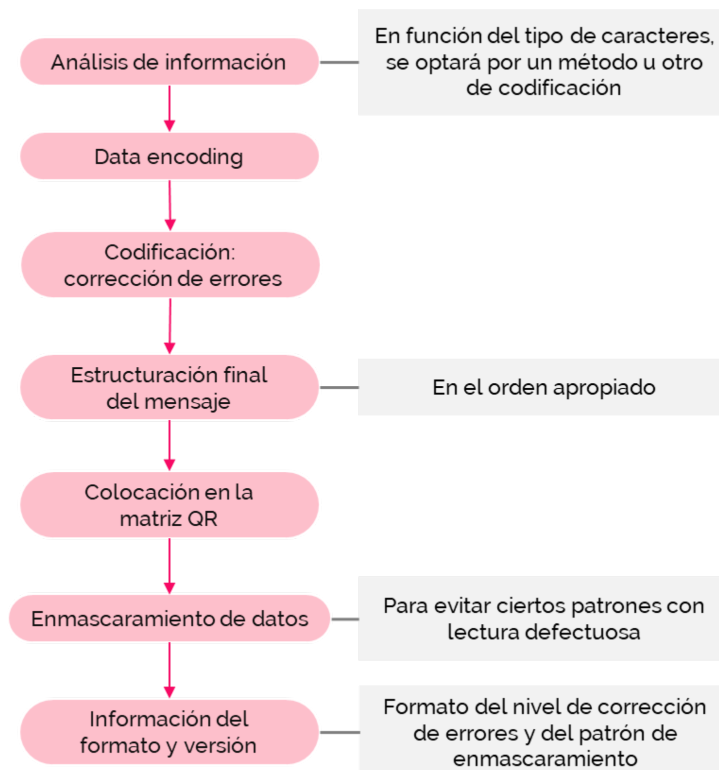


Figura 3.2: Diagrama del proceso de codificación de un código QR

3.4.2 Data encoding

Este proceso es el núcleo de la generación de un código QR, ya que consiste en la transformación del mensaje (formado por una serie cualquiera de caracteres) en una secuencia de bits. La codificación se divide en varios subprocesos. Los primeros son de definición de parámetros y los siguientes, son de generación de la secuencia de bits. Estas etapas se muestran en la Figura 3.3.

3.4.3 Codificación de corrección de errores

Para conseguir la secuencia de corrección de errores, la secuencia primaria es dividida en bloques. Según la versión (tamaño) corresponde cierta cantidad de bloques y, por cada uno de ellos, se generará una secuencia de bytes de corrección.

3.4.4 Estructuración final del mensaje

En la estructura global, primero se van a situar todos los bytes que conforman la secuencia del mensaje y, a continuación, todos los bytes que conforman las secuencias de corrección. En ambas partes, se coloca el primer byte del primer bloque, seguido por el primer byte del segundo bloque, y así hasta el primer byte del último bloque. A éste le sigue el segundo byte del primer bloque y así sucesivamente.

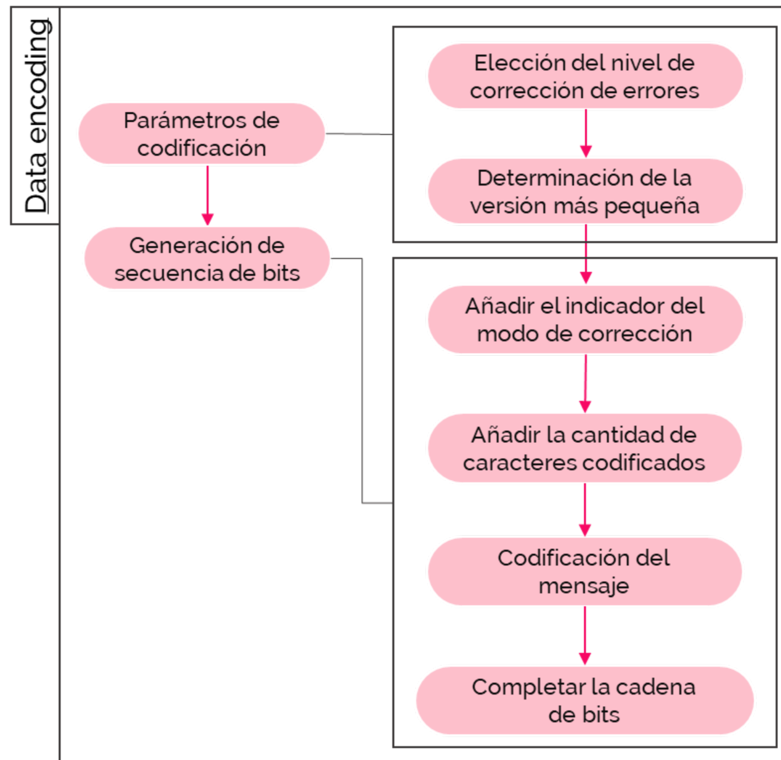
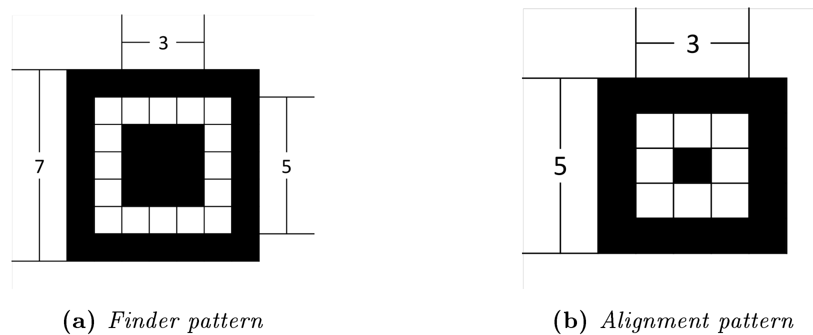


Figura 3.3: Diagrama extendido del *data encoding*

3.4.5 Colocación en la matriz QR

Una vez obtenida la secuencia total de bytes, solo queda saber cómo se distribuye espacialmente. Cada celdilla o módulo representa un bit. Los ceros se corresponden con módulos blancos y los unos, con negros. Sin embargo, existen también una serie de patrones que o bien aportan información o bien son necesarios para que el lector sitúe y oriente la imagen correctamente.

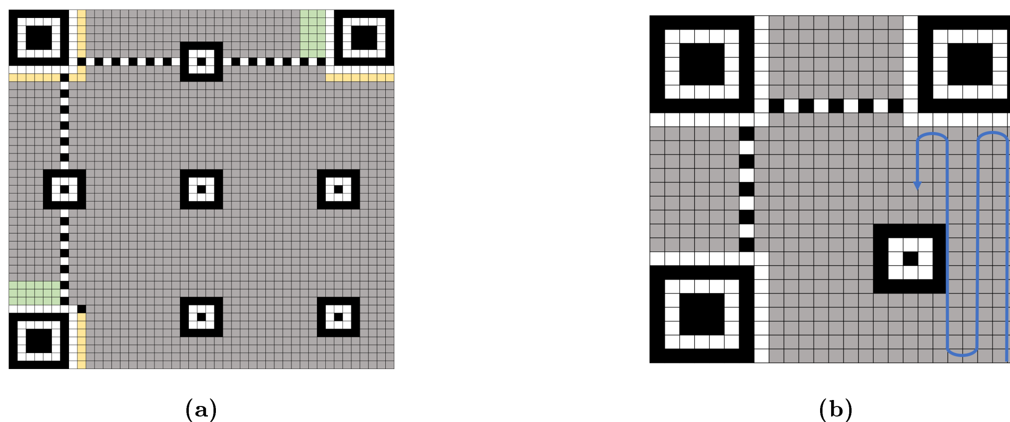


(a) *Finder pattern* **(b)** *Alignment pattern*

Figura 3.4: Patrones con información de localización y alineación.

El primer patrón es el *finder pattern* (Figura 3.4a), situado siempre en las esquinas superior izquierda, superior derecha e inferior izquierda. Alrededor de estos patrones todas las celdillas deben ser blancas. Los siguientes patrones son los de alineamiento: *alignment patterns* (Figura 3.4b). Todas las versiones a partir de la 2ª deben llevarlos. Estos patrones tienen siempre la misma forma, pero se sitúan en sitios diferentes según la versión.

A continuación, están los *timing patterns*. Se trata de dos líneas, horizontal y vertical, discontinuas (comenzando y acabando con celdilla negra). Una se sitúa en la 6ª fila y la otra en la 6ª columna, es decir, entre los *finder patterns*. Por último, queda situar una celdilla negra común para todos los códigos QR y localizar las celdillas reservadas donde se situará la información de versión (para versiones a partir de la 7ª) y formato.



(a) **(b)**

Figura 3.5: Distribución de patrones (a) y dirección de escritura (b) en la matriz QR

El resultado, para un QR versión 7, es el de la Figura 3.5a marcando en color amarillo las celdillas para el formato y en verde las de la versión. La celdilla negra común a todos los códigos es la que está al lado de la esquina del *finder pattern* inferior.

A partir de este momento, todos los patrones están situados en la matriz y se puede pasar a disponer la secuencia de bits de información. La secuencia comienza en la esquina inferior derecha, con dirección hacia arriba (Figura 3.5b). Al llegar al final, comienza la segunda pasada hacia abajo, y así sucesivamente, en zig-zag. Cada pasada tiene dos celdillas de ancho, por lo que es necesario definir un patrón de colocación. Este queda descrito en la Figura 3.6

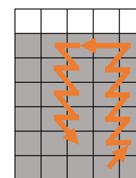


Figura 3.6: Recorrido en zig-zag de la lectura.

3.4.6 Enmascaramiento de datos

Esto consiste en modificar la matriz de manera adecuada para evitar secuencias difíciles de leer para un escáner de QR. Enmascarar una celdilla implica cambiarla de blanca a negra y viceversa. Lógicamente solo se enmascaran celdas que no forman parte de patrones. Existen 8 patrones de enmascaramiento (por ejemplo, enmascarar todas las filas pares o todas las columnas múltiplos de 3, etc.). Para cada mensaje se debe elegir el patrón de enmascaramiento que mejor permite la lectura.

3.4.7 Información de formato o versión

Por último, es necesario añadir cierta información fundamental para la decodificación: el formato y la versión. El formato consistente en una secuencia de 15 bits, se sitúa en las celdillas amarillas de la Figura 3.5. Codifica el nivel de corrección de errores y el patrón de enmascaramiento. La información de la versión solo se añade en la versión 7ª y superiores y se sitúa en las celdillas verdes de Fa figura 3.5.

Capítulo 4

Diseño y programación del *software*

En el presente capítulo se profundizará sobre el diseño del sistema de control de inventario. En primer lugar se introducen los requisitos del sistema y los lenguajes de programación, Python y Bash scripting. Seguidamente se incluye la estrategia de diseño. Por último, se desarrolla la programación de cada uno de los scripts que conforman el programa.

4.1 Requisitos del sistema 25
4.2 Lenguajes 27
4.2.1 Python 27
4.2.2 Shell 27
4.3 Metodología y diseño del *software* 28
4.3.1 Estrategia de diseño 28
4.3.2 Programación multitarea 31
4.4 Implementación del código 32
4.4.1 Gestión multiproceso 32
4.4.2 Control de vuelo del dron 34
4.4.3 Reproducción del *streaming* de vídeo 37
4.4.4 Captura y almacenado de imágenes 37
4.4.5 Escaneado de códigos QR 38
4.4.6 Comparativa de resultados de lectura 50
4.4.7 Interfaz de usuario 60

4.1 Requisitos del sistema

En este punto se desarrollan las fronteras funcionales del sistema, en relación con el comportamiento y propiedades que deben proveerse para la ejecución del software. Se incluyen tanto las interacciones requeridas por el usuario como las restricciones físicas y de datos. Se establece así el comportamiento esperado del sistema expresado en términos cuantitativos y de limitaciones.

1. Requisitos de hardware
    - Procesador: 1 GHz o más rápido.
    - RAM: 1 GB o más.

- Espacio en disco duro: 8 GB o más.
- Pantalla: 800x480 con 60 fotogramas por segundo.
- Otros periféricos: teclado, ratón, tarjeta SD (si se trabaja con Raspberry Pi), cable HDMI.
- Fuente de alimentación: recomendable 5V/2,5A.
- Parrot Mambo FPV.

2. Requisitos de sistema operativo y software
    - Sistema operativo: GNU/Linux. Se ha probado en Raspbian (en Raspberry Pi 3 B) y Ubuntu 16.04.
    - Shell de Unix: Bash. Incluido en la mayoría de sistemas operativos GNU/Linux.
    - Python 3.5
    - Programas CLI:
        - yad: interfaz gráfica del programa.
        - omxplayer: reproductor multimedia optimizado para Raspberry Pi.
        - raspi2png: para captura de imágenes optimizada para Raspberry Pi.
    - Paquetes de Python
        - pyparrot v1.4 : para control de vuelo del dron.
        - zeroconf, untangle: para conectar mediante WiFi con el dron (pyparrot).
        - Pillow: tratamiento de imágenes.
        - numpy: transformación de imágenes a matrices de datos.
        - zbar: detección de códigos QR.
        - csv: escritura de datos en archivo tipo csv.
3. Requisitos de conectividad
    - Conexión WiFi para conectar con el dron.
    - Conexión Bluetooth, en caso de preferir el control de vuelo mediante este sistema.

Se establecen las siguientes interacciones por parte del usuario:

- Encendido y apagado del dron.
- Conexión via WiFi con el dron.
- Trazado de una ruta de vuelo para el dron, mediante Python y la librería `pyparrot`. También puede escogerse una de las rutas que incorpora la aplicación.
- Ejecución de la aplicación de visión, vuelo y detección de códigos QR mediante el programa `droneInTheShell.sh`.
- Ejecución de la aplicación de comprobación de inventario (también en `droneInTheShell.sh`).

4.2 Lenguajes

La programación del software está escrita en dos lenguajes de programación de amplia difusión: Python y Bash *scripting*, una variante de *shell scripting*.

4.2.1 Python

Como bien se describe en su página web, Python es un lenguaje de programación interpretado, de alto nivel y multiparadigma. Nace a finales de 1990 de la mano de Guido van Rossum, pero no es hasta una década más tarde cuando su popularidad empieza a escalar.

La principal razón es su sintaxis sencilla, que hace que programas escritos en Python recuerden al pseudocódigo. Esto le confiere una ventaja con respecto a otros lenguajes: la inteligibilidad, es decir, la facilidad para que un programa pueda entenderse por cualquier miembro de la comunidad.

Como resultado, Python tiene una comunidad de usuarios muy activa, documentación abundante y completa y librerías que facilitan la experiencia de programación. Algunos de los paquetes más importantes en materia científica son Pillow, NumPy, Matplotlib y OpenCV.

4.2.2 Shell

La gestión del stream de imágenes del minidron, así como la programación multitarea, se delegó al intérprete de comandos de la propia Raspberry Pi: el shell. Su nombre se debe a que es la capa más superficial, o cáscara, que recubre al sistema operativo. Pero antes de profundizar en las cualidades de esta herramienta, hay que hablar primero del kernel y su función como intermediario.

El kernel
El kernel es el programa puente que conecta software y hardware y tiene absoluto control del sistema. Es, de hecho, la base de la mayoría de sistemas operativos y ocupa una porción grande del mismo.
Las órdenes del kernel interactúan directamente con componentes físicos como la CPU, la RAM o los periféricos, mediante las denominadas “llamadas al sistema". Estas llamadas al sistema se invocan cuando un usuario requiere de servicios controlados por el sistema operativo, como la lectura de un fichero.
Algunas de las tareas del kernel son la ejecución de procesos, el uso que tienen estos de la memoria y la implementación y control de los periféricos en el sistema. En la Figura 4.1 se puede ver cómo interactúan los distintos espacios (usuario, kernel y hardware) en el sistema operativo.

El kernel es, resumidamente, un gestor de recursos del sistema. El *shell* es el intérprete del lenguaje que lo controla: un programa capaz de comunicarse con el núcleo para mandar instrucciones y recibir respuestas de la máquina. Estas órdenes se transcriben siguiendo una sintaxis específica, que puede incluir comandos, meta-caracteres, comentarios, argumentos, operadores de control y otros.

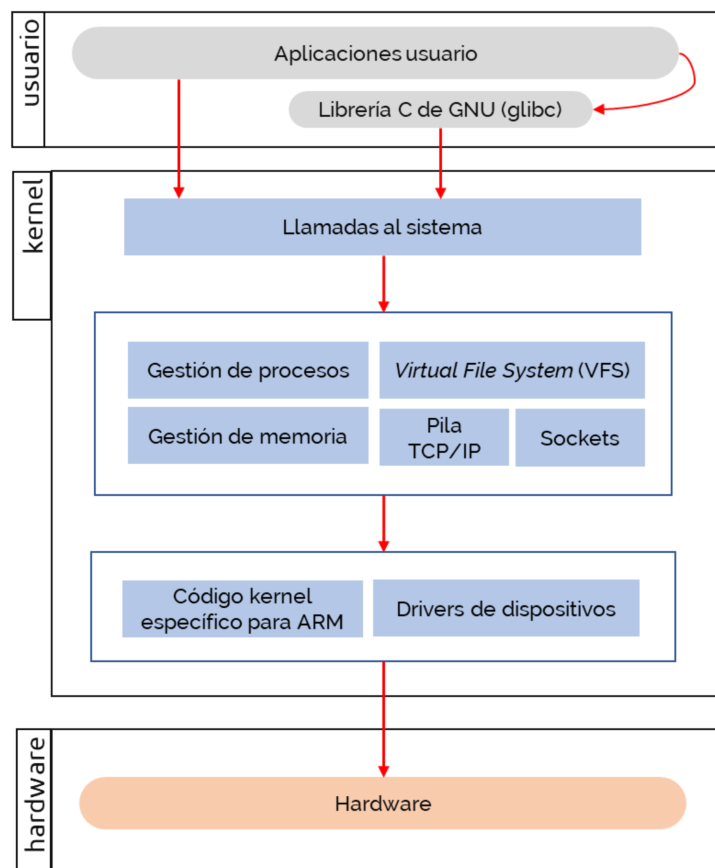


Figura 4.1: Diagrama de interacciones entre el espacio usuario, el espacio kernel y el hardware.

Varios intérpretes están disponibles en el mismo sistema operativo y pueden clasificarse atendiendo a distintos criterios.

Según sea su interfaz de usuario, se dividen entre los de línea de texto (CLI, *command-line interface*) y su contraparte, los de interfaz gráfica (GUI, *grafical user interface*). A costa de un entorno más atractivo, las interfaces gráficas consumen más recursos de tiempo y memoria que las de línea de texto. Dentro de las CLI, los shell pueden subdividirse según familias: tipo Bourne shell, tipo shell de C, no tradicional, histórica.

Entre todas las posibilidades, destaca bash (*Bourne-Again shell*), el intérprete de comandos más extendido en la familia Linux y el utilizado en este proyecto.

4.3 Metodología y diseño del *software*

4.3.1 Estrategia de diseño

El *software* del proyecto permite la interacción de los sistemas dron y Raspberry Pi. Se distinguen dos flujos de trabajo principales: la visión y el vuelo. El esquema de la Figura **??** y el diagrama de la Figura 4.2 ofrecen una perspectiva general de las partes implicadas en el sistema y cómo se subdividen los flujos principales en tareas individuales, respectivamente. Cabe destacar que ambos son de carácter orientativo y su objetivo es acercarse a la comprensión general del *software*.

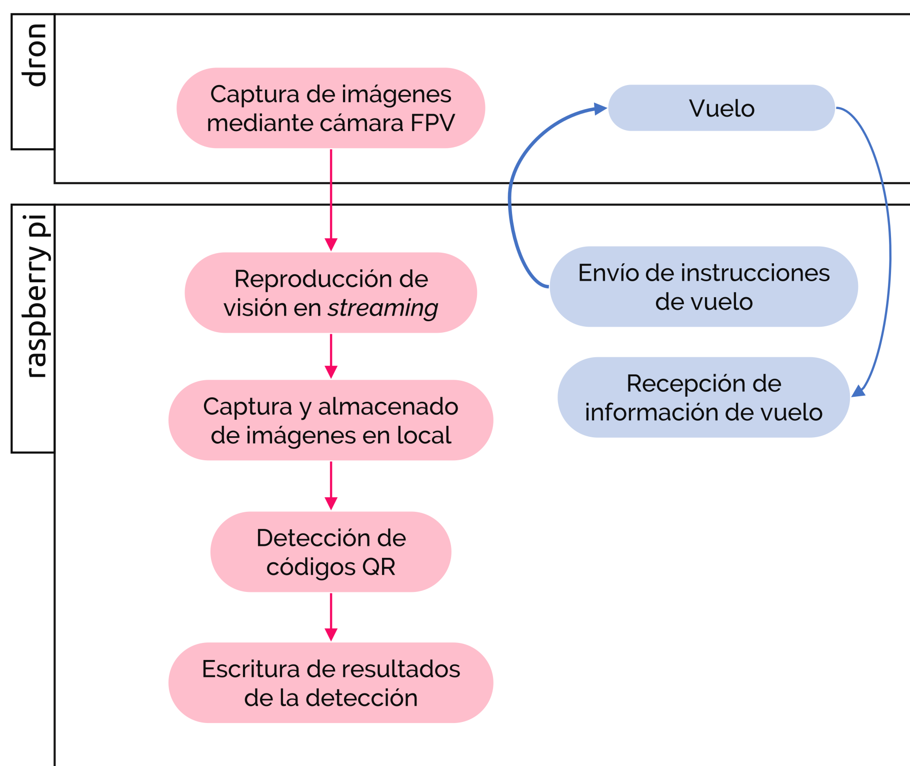


Figura 4.2: Croquis de funcionamiento del programa

En la Figura 4.2 se observa cómo Raspberry Pi recibe y reproduce, en primer lugar, en *streaming* el video generado por la cámara FPV del dron. A continuación realiza capturas de dicho video para, posteriormente, analizarlas en busca de códigos QR. Por último, el contenido de estos códigos detectados debe ser puesto por escrito en un fichero. De esta manera podrá ser comparado con una base de datos del contenido teórico del almacén.

Paralelamente se dan las acciones de control de vuelo del dron. En este sentido, se deben mandar las instrucciones de vuelo para seguir una ruta determinada, así como recibir la información de vuelo.

A pesar de la aparente secuencialidad de tareas, como se podría intuir del esquema anterior, los procesos no ocurren estricamente en serie. Por el contrario, todas las acciones deben ser realizadas simultáneamente: a medida que el *streaming* avanza, nuevas imágenes deben ser capturadas y, al mismo tiempo, el dron debe recibir las siguientes instrucciones de vuelo.

En la Figura 4.3, se ilustra esa ejecución simultánea de las tareas en Raspberry Pi. Se aprecia que cada una de ellas constituye un subproceso propio denominado *child process* y que será ejecutado de manera paralela al resto. Todos los procesos hijos son creados por un proceso principal o padre en el que se enmarcan.

En apartados posteriores se analizará el funcionamiento de cada uno de estos procesos y subprocesos. Sin embargo, es importante destacar que, una vez creados todos los subprocesos, el proceso padre se limita a esperar a que todos ellos terminen. Para evitar que el usuario deba cerrar manualmente el programa, es uno de los procesos hijos el que da la orden de finalización

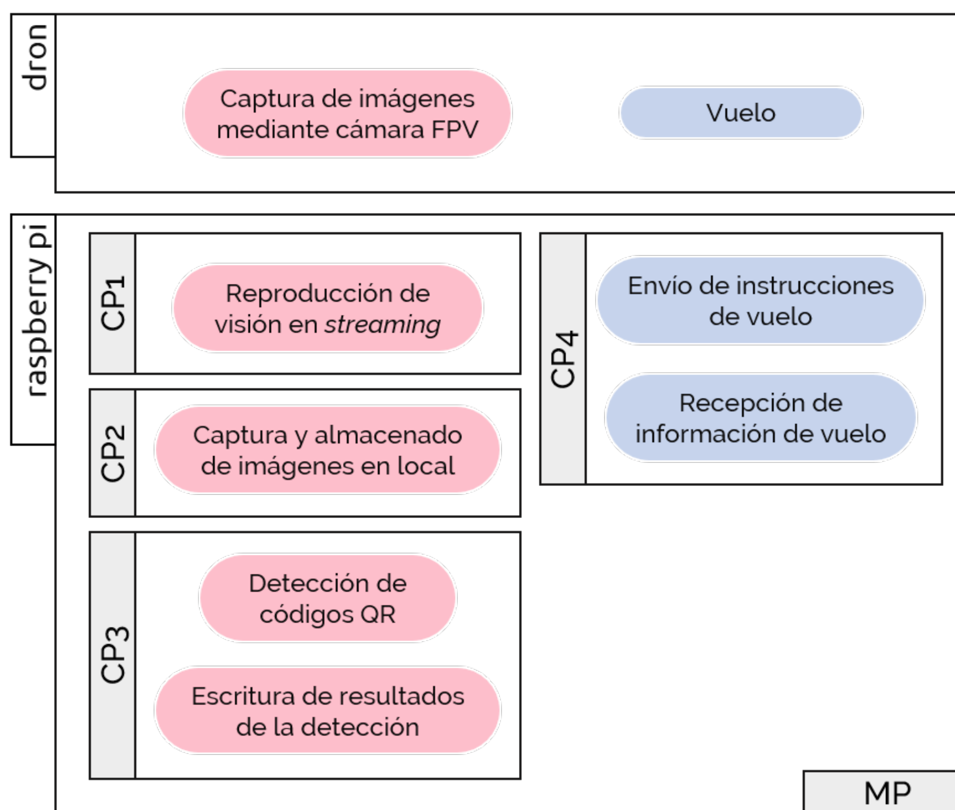


Figura 4.3: Funcionamiento real del programa

del resto (el proceso padre y el resto de hijos). Se trata del proceso que controla el vuelo del dron que, una vez acabada la ruta, finaliza el programa completo.

El programa descrito se ha pensado para ser ejecutado desde una sencilla interfaz gráfica. Esta ha sido diseñada para hacer de intermediaria entre el usuario y el *software* que, dado el sistema de programación, solo podía ser ejecutado mediante el terminal (de comandos) del sistema operativo. De este modo, cuando los procesos finalizan, las ventanas externas se cierran y se vuelve a la interfaz gráfica original. En este momento, se puede comparar el conjunto de mensajes QR detectados con otro fichero, que contiene los códigos que teóricamente se deberían encontrar en el inventario.

Por último comentar un aspecto relevante del funcionamiento multitarea. Este proyecto se ha desarrollado en vistas a crear un sistema automático de control de inventario, en el que la simultaneidad del procesado de imágenes con el vuelo no era necesaria.

Sin embargo, al descender en el nivel de programación, se comprobó que era posible realizar la detección de manera paralela (concurrente). Así, se ha trabajado en la optimización del código de bash para permitir este funcionamiento simultáneo. La razón es que, en un futuro desarrollo de este sistema, se puedan incorporar otras funcionalidades complementarias que requieran que la lectura se realice al mismo tiempo. Un ejemplo podría ser la búsqueda de un producto concreto en el almacén, tanto para comprobar que está, como para averiguar su localización exacta.

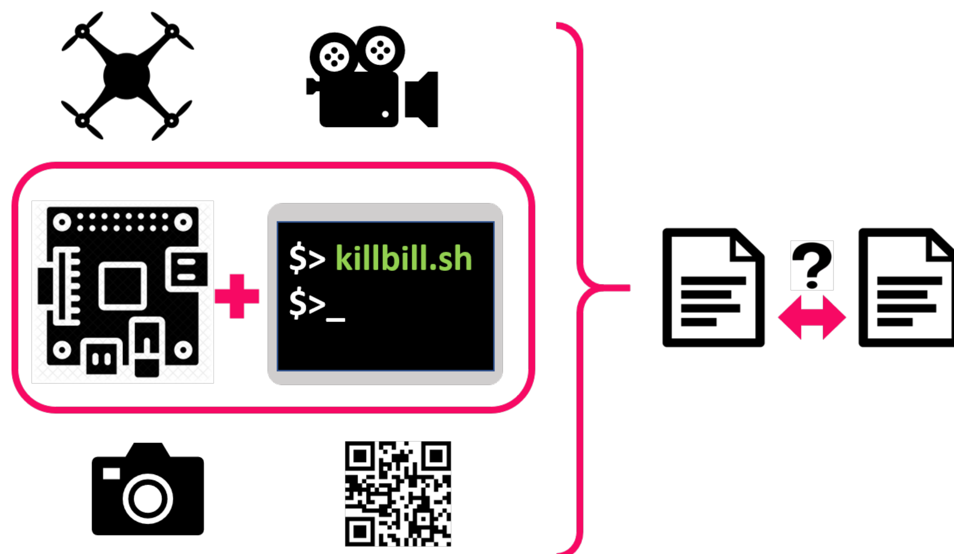


Figura 4.4: Esquema general de funcionamiento, relacionando cada una de las partes implicadas en el sistema.

4.3.2 Programación multitarea

En Linux, los objetos del sistema se clasifican como procesos o como ficheros. Mientras que los ficheros son una colección de datos, los procesos son programas en ejecución identificables mediante un ID único, el PID. En una máquina con una única unidad de procesamiento (CPU), los recursos como la memoria se comparten entre todos los procesos del sistema. Pero en la mayoría de ordenadores actuales, incluida la Raspberry Pi 3 B, ya se habla de sistemas *multicore*, lo que expande las posibilidades de ejecución a uno o más procesadores. Este concepto se conoce bajo el nombre de multiproceso o multitarea, y permite que varios programas se ejecuten de forma paralela o concurrente.

Raspberry Pi 3 B cuenta con un procesador Cortex-A53 con sistema SMP o de multiprocesado simétrico. En este tipo de sistema, cada *core* tiene su propia memoria caché y comparte, junto al resto de cores, la memoria principal (RAM). Los *core* están a su vez interconectados mediante buses. El multiprocesado simétrico permite que cualquier procesador trabaje en cualquier tarea sin importar dónde esté alojada en la memoria, siempre y cuando esta tarea sea procesada por un único *core*.

La programación multitarea no necesariamente favorece la velocidad de ejecución. El núcleo consume recursos extra por la simple gestión de procesos y es de esperar que, al ejecutar varios scripts a la vez, se tarde proporcionalmente más tiempo que si se hiciera de forma secuencial. A pesar de este inconveniente, las condiciones del proyecto piden la simultaneidad de estas acciones (*streaming*, vuelo, captura de imágenes y almacenado de las mismas) y se ha comprobado que su acción conjunta no monopoliza los recursos del sistema.

4.4 Implementación del código

4.4.1 Gestión multiproceso

Ya en los objetivos del proyecto se hace especial hincapié en la simultaneidad de procesos. La pregunta es, ¿cómo se consigue programáticamente una secuencia concurrente de las acciones? La respuesta la tiene el planificador de tareas de la CPU.

La CPU sigue una planificación o *schedule* para la ejecución de procesos, asignando en función de la importancia una prioridad en la ejecución. Programas más importantes se ejecutarán de forma secuencial, en detrimento de otros procesos que, siendo de inferior relevancia, intercalarán sus acciones con otros de su categoría. Al acabar el proceso o tras un debido tiempo, la CPU lanza una interrupción y se pasa a la siguiente tarea. Sin entrar en demasiado detalle, los procesos o rutinas se pueden clasificar siguiendo la siguiente jerarquía:

- Rutinas en primer plano o *foreground*. Son los que se ejecutan interactivamente con el usuario.
- Rutinas en segundo plano o *background*. Son aquellos que se ejecutan "sin molestar", que permiten que el usuario pueda seguir interactuando con el *shell*.

Los procesos en segundo plano consumen menos recursos que los del primer plano, lo que abre la posibilidad de ejecutar más aplicaciones al mismo tiempo. Los procesos que no requieren interacción por parte del usuario son buenos candidatos a ejecutarse en el *background*.

En este proyecto, el programa completo se ejecuta desde dos *scripts* escritos en bash: `droneInTheShell.sh` y `killbill.sh`:

- `droneInTheShell.sh` construye la interfaz gráfica y llama a `killbill.sh`.
- `killbill.sh` ejecuta el grueso del programa. Asigna a cada una de las instrucciones (vuelo, *streaming*, capturas de imágenes y detección de QRs) un subproceso en el *background*.

La asignación de instrucciones a segundo plano se consigue mediante la función `multiproceso`, que está incluída en el programa `killbill.sh`. Para su funcionamiento `multiproceso` recibe como parámetro un *here document*, un tipo de *input* que admite múltiples líneas, emulando un documento de texto. En este caso, cada una de estas líneas equivale a una instrucción del programa. Por ejemplo, la reproducción del *streaming*. En el Listing 1 se resume la función:

```bash
#!/bin/bash
multiproceso () {
    local procesos
    while read procesos; do
        eval "$procesos" &
    done
    wait
}
```

Listing 1: Función `multiproceso` en el programa principal.

En el bucle de tipo *while* se evalúan y mandan al *background* una serie de procesos, escribiendo al final de la instrucción un `&`. Estos procesos son las intrucciones que se leen (`read`) del *here document* y que almacenan en la variable local `$procesos`. Una vez acaba el bucle, se pausa la ejecución del programa mediante `wait`. Como `wait` no va acompañado de ningún parámetro, esta pausa se extenderá hasta que todos y cada uno de los subprocesos acaben. O hasta que se envíe una señal de tipo `kill` (muerte de la ejecución).

No hay que perder de vista que el vuelo marca, en un sentido teórico, el final del programa. Una vez el dron finaliza su misión no tiene sentido seguir reproduciendo el vídeo, ni capturando imágenes ni detectando QRs. Partiendo de este concepto, al fragmento de código anterior se le debería añadir una función que:

- Llamara al *script* `vueling.py`
- Detuviera la ejecución del resto de procesos cuando `vueling.py` acaba.

Con este objetivo se crea la función `redbaron()`. Como puede observarse en el Listing 2, `redbaron()` primero ejecuta en Python el *script* `vueling.py`, y una vez acaba este, "mata" el grupo de procesos que se ejecutan en el `bash` mediante la instrucción `killall -g bash`.

```
1 redbaron (){
2     python3 /home/pi/Documents/Projects/Drone_QR/code/vueling.py
3     killall -g bash
4 }
```

Listing 2: Función `redbaron` en el programa principal.

Tal y como se ejecuta hasta ahora, se cierran todos los procesos abiertos por `multiproceso()`. Existe una opción más conservadora: sustituir `killall -g bash` por una instrucción que acabe con los procesos localizadamente. Para ello, se modifica la función `multiproceso()`, añadiendo la quinta línea al programa original.

```
1 multiproceso () {
2     local procesos
3     while read procesos; do
4         eval "$procesos" &
5         echo $! >/path/a/archivo.pid
6     done
7     wait
```

Esta línea indica que se debe almacenar en un fichero `archivo.pid` el PID del último `$proceso` leído, que se captura mediante la instrucción `$!`. A continuación `redbaron()` se editaría para que quedara de este modo, como aparece en el Listing 3:

```
redbaron (){
    python3 /home/pi/Documents/Projects/Drone_QR/code/vueling.py
    kill $(cat /path/a/archivo.pid)
}
```

Listing 3: Eliminación de programas de forma localizada mediante `redbaron`.

Finalmente, `killbill.sh` pasa el *here document* como argumento a `multiproceso()`, que irá ejecutándolo línea a línea.

```
multiproceso <<STOP
reproductor rtsp://url/del/streaming
snapshot /path/a/carpeta/data
python3 deteccionQRs
redbaron
STOP
```

4.4.2 Control de vuelo del dron

La conexión entre Raspberry Pi y el dron Mambo[1] se realiza mediante conexión WiFi. Para establecer la conexión, se ejecutan los siguientes pasos:

1. Se inserta la cámara FPV en la clavija pertinente del *drone* Parrot MAmbo.
2. Se enciende Mambo con el botón interruptor.
3. Se establece una búsqueda de redes WiFi desde Raspberry Pi.
4. Cuando Mambo aparezca en la lista de redes disponibles, se conecta a ella. Es posible que pida una contraseña.

Para el envío de instrucciones de vuelo se crea un *script* relativo a este control (en el caso del proyecto, `vueling.py`). En la cabecera se incluye la librería `pyparrot`, mediante la cual se importa el módulo `Mambo`, que permite la creación de objetos que modifiquen el estado del dron. A continuación se introduce la dirección MAC de Mambo, mediante la creación de la variable `mamboMac`.

```
from pyparrot.Mambo import Mambo

mamboMac = "e0:17:d9:63:3f:d0"
```

Una vez realizada esta pequeña configuración se pasa a la conexión y secuencia de instrucciones de vuelo. Previamente se crea un objeto de tipo `Mambo` al que se le pasa como argumento la dirección MAC y la opción `use_wifi = True`.

```
mambo = Mambo(mamboMac, use_wifi=True)
```

[1]A partir de este momento se utilizará indistintamente dron o Mambo para referirse a él

Con el objeto Mambo ya creado, se procede a la conexión programática con el dron y su comprobación. La funciones son bastante descriptivas.

```
print("Conectando con el dron")
success = mambo.connect(num_retries=3)
print("Estado: %s" % success)
```

En caso que la conexión con el dron se efectúe, se envía la secuencia de control. Las intrucciones de vuelo deben ajustarse mediante prueba y error hasta encontrar aquella combinación que permita una misión de reconocimiento satisfactoria. En el caso de este trabajo, los parámetros de mambo.fly_direct se han ajustado empíriciamente hasta dar con los valores mostrados en el Listing 4.

Para conseguir un control más ajustado se han diseñado las funciones de vuelaMovimientoDelay, resumidas en el Listing 4. Con ellas se pretende emular el funcionamiento de un motor paso a paso, con desplazamientos discretos en la dirección de uno de los tres ejes principales. Así, las órdenes de avance se indican como número de pasos que tiene que dar en una direccion, siendo estos pasos de una duración y potencia constante. Hay que destacar que sin las debidas pausas y tiempos se producen errores, debidos a ligeros desplazamientos no controlados que acumulados producen un desvío importante.

```
def vuelaVerticalDelay (sentido, numPasos):
    if sentido:
        direccion=1
    else: direccion=-1
    for i in range (0,numPasos):
        print("Vertical")
        mambo.fly_direct(roll=0, pitch=0, yaw=0, vertical_movement=direccion*15
          ↪  , duration=0.3)
        mambo.smart_sleep(2)
def vuelaAdelanteDelay (sentido, numPasos):
    if sentido:
        direccion=1
    else: direccion=-1
    for i in range (0,numPasos):
        print("Frontal")
        mambo.fly_direct(roll=0, pitch=direccion*20, yaw=0, vertical_movement=0
          ↪  , duration=0.3)
        mambo.smart_sleep(2)
def vuelaDeLadoDelay (sentido, numPasos):
    if sentido:
        direccion=1
    else: direccion=-1
    for i in range (0,numPasos):
        print("A un lado")
```

```
23          mambo.fly_direct(roll=direccion*20, pitch=0, yaw=0, vertical_movement=0
             ↪  , duration=0.3)
24          mambo.smart_sleep(2)
```

Listing 4: Definición de funciones de vuelo para el dron.

Las funciones `vuelaMovimientoDelay` admiten dos argumentos: `sentido` y `numPasos`, bastante descriptivos. `sentido` es una variable de tipo boleano, que puede tomar dos valores: True (hacia arriba, hacia adelante o a la derecha) o False (hacia abajo, hacia atrás o hacia la izquierda), acorde al sentido predeterminado de las distintas opciones de `mambo.fly_direct(roll, pitch, yaw, vertical_movement)`.

Una vez definidas las funciones de vuelo y estando conectados al dron, se procede al envío de intrucciones. De forma secuencial se especifican las acciones que deben realizarse: despegar, volar hacia adelante, volar a los lados, aterrizar y desconectar.

Se aconsejan dos buenas prácticas: la primera es que hay que respetar las pausas, pues mejora la experiencia de vuelo y se ajusta mejor a la ruta especificada. La segunda, que se debe desconectar al dron (`mambo.disconect()`) una vez se finalice la ruta de vuelo. El código de ejemplo se observa a continuación, en el Listing 5.

```
1  if (success):
2      # Información de estado
3      print("sleeping")
4      mambo.smart_sleep(2)
5      mambo.ask_for_state_update()
6      mambo.smart_sleep(2)
7
8      print("Despegue")
9      mambo.safe_takeoff(5)
10
11     vuelaAdelanteDelay(1,1)
12     mambo.smart_sleep(2)
13
14     vuelaDeLadoDelay(-1,8)
15     mambo.smart_sleep(2)
16
17     print("Aterrizaje")
18     mambo.safe_land(5)
19     mambo.smart_sleep(3)
20
21     print("Desconectar")
22     mambo.disconnect()
```

Listing 5: Secuencia de vuelo del dron en `vueling.py`

4.4.3 Reproducción del streaming de vídeo

Raspberry Pi utiliza el procesador multimedia de bajo consumo VideoCore, integrado en el SoC de Broadcom. Dado que los recursos de Raspberry Pi son limitados, se utiliza el programa CLI `omxplayer` para la reproducción del *streaming* de imágenes.

`omxplayer` está basado en la API recomendada por Broadcom VideoCore para reproducción de vídeo: OpenMAX IL. Está especificamente diseñada para procesar multimedia en la GPU mediante aceleración de *hardware*. Una de sus ventajas respecto a otros reproductores es que no consume muchos recursos de la CPU, un problema que se había encontrado con VLC o GStreamer.

Para la visualización de las imágenes del dron, hay que conseguir, en primer lugar, la dirección RTSP mediante la que envía el *streaming* de imágenes. Esta se encuentra en la librería `pyparrot`, en el módulo `VisionServer`. A continuación, se invoca a `omxplayer` añandiéndole los siguientes argumentos:

```
1  pi@raspberry:~$    omxplayer --display 4 --aspect-mode fill -o local
   ↪  rtsp://192.168.99.1/media/stream2
```

- `--display 4` hace referencia a que se reproduce el vídeo en la pantalla LCD *touchscreen* de Raspberry Pi.
- `--aspect-mode fill` indica que la ventana de reproducción ocupa toda la pantalla.
- `-o local` indica que permite salida de audio, en caso de que existiera, en local.
- `rtsp://192.168.99.1/media/stream2` es la dirección de *streaming*.

4.4.4 Captura y almacenado de imágenes

Para la captura de imágenes se utiliza `raspi2png`, un programa que realiza *screenshots* de pantalla. La ventaja frente a otras herramientas similares es que, en caso de trabajar desde ssh (*Secure Shell*[2]), captura toda la información mostrada por pantalla y no solo la ventana de comandos.

Para hacer una captura de pantalla, se llama a `raspi2png` desde la ventana de comandos:

```
1  pi@raspberry:~$    raspi2png -p "/path/donde/se/guarda/imagen.png"
```

Sin embargo, queremos realizar esta acción de forma indefinida, con cierto retraso o *delay* entre captura y captura. En el caso de este trabajo, medio segundo es un *delay* aceptable que no sacrifica la integridad de la imágenes[3]. Para ello, se debe ejecutar esta acción desde un bucle `while`, que se escribirá dentro de una función de bash. Esto queda como se muestra en el Listing 6:

```
1  snapshot () {
2  carpetaDestino=$1
3  i=0
```

[2]Secure Shell es un protocolo del nivel de aplicación para ejecutar intrucciones desde/a otro ordenador, de forma segura
[3]Si el retraso es demasiado bajo, por debajo de la décima de segundo, aparecen imágenes corruptas.

```
4  echo "Se inician las capturas de imágenes"
5  cd "$carpetaDestino"
6  while (1); do
7      i=$[$i+1]
8      archivo="$( printf '%04d' "$i" )"
9      raspi2png -p "$carpetaDestino/$archivo.png"
10     sleep 0.5
11 done
12 echo "Capturas finalizadas"
13 }
```

Listing 6: Función `snapshot` para captura y almacenado de imágenes, escrita en bash.

- `echo "Se inician las capturas de imágenes"`, al igual que el otro `echo`, son mensajes que aparecen en el terminal cuando se ejecuta la función. Permite informar al observador qué se está haciendo.
- `carpetaDestino=$1` indica que el primer argumento que se pase a la función `snapshot()` se guardará en la variable `carpetaDestino`. Predeciblemente, este argumento será la ruta a un directorio.
- `i=0` es una variable que se utiliza como contador. Al entrar en el bucle infinito (`while (1); do`) se actualiza incrementándose en una unidad su valor.
- `cd "$carpetaDestino"` cambia de directorio (localmente) a `carpetaDestino`.
- `archivo="$( printf '%04d' "$i" )"` asigna, a la variable `archivo`, el valor de la variable `i` como entero de cuatro cifras.
- `sleep 0.5` pausa o detiene la ejecución del bucle durante 500 ms.

La función `snapshot()` se implementa directamente en el programa `killbill.sh` y se manda al *background* como subproceso mediante la función `multiproceso`.

4.4.5 Escaneado de códigos QR

El último de los programas ejecutados de manera paralela desde la función `multiproceso()` es el encargado de escanear las imágenes del *streaming*: `Qrying.py`.

En apartados anteriores se comentó que, en un primer momento, el proceso de detección de QRs iba a ser ejecutado tras el vuelo del dron. Este era el enfoque inicial, cuando todo el proceso se iba a desarrollar mediante la librería de *pyparrot*. Dicha librería basa la visión del dron en dos herramientas de gestión multimedia: *ffmpg* y *vlc*, las cuales fueron expuestas en la página 15.

Finalmente, el proyecto cambió su rumbo debido a los problemas[4] derivados de las mencionadas herramientas. Así, se pasó a realizar el programa principal en el entorno de *bash*, reservando la librería *pyparrot* únicamente para el control del dron.

[4] No funcionaba

Este hecho permitió que el programa de detección de QRs se hiciera a tiempo real, ya que el nuevo sistema consume muchos menos recursos de la Raspberry. De otro modo, el programa estaba limitado a realizar una comparación de bases de datos *a posteriori*. Sin embargo, esta nueva metodología aumenta las posibilidades de desarrollo posterior del proyecto, permitiendo añadir otras funcionalidades que requieran la detección y procesado de QRs en directo.

De cara a introducir el funcionamiento general, resulta necesario explicar el programa original de análisis de imágenes. Ello facilitará la explicación y entendimiento del programa de detección de QR's a tiempo real. Primero, se explicará el proceso de escaneado de una única imagen.

Escaneado de una imagen

Para el escaneado de códigos QR se ha utilizado el *binding* para Python 3.+ de la librería `zbar`, `zbar-py`.

`zbar-py` utiliza el módulo `ctypes` para utilizar las librerías de `zbar`, que están originalmente escritas en C. Para instalar `zbar-py` es tan sencillo como:

```
1 pi@raspberry:~$    pip install zbar-py
```

La clase que permite analizar los códigos es `Scanner(object)`. En el constructor de esta clase se permiten varias opciones para configurar la lectura de distintos códigos de barras, en el argumento `config`.

```
1 def __init__(self, config)
```

Parámetros de `config`:

- None: predeterminado. Configuración recomendada por el autor y usada en este proyecto.
- Lista de tres valores (`symbol_type, config_type, value`)
  1. `symbol_type`: tipo de código de barras. `ZBAR_NONE` abarca todos los tipos.
  2. `config_type`: tipo de configuración definida en zbar.h. Permita excluir/habilitar ciertos tipos de códigos de barras, la lectura de códigos con una longitud de datos específica y otras opciones. El autor recomienda usar la configuración por defecto.
  3. `value`: 1 para opciones booleanas, o cualquier entero para las otras opciones.

Los objetos de tipo `Scanner` cuentan con el método `scan(self, image)` , que permite las siguientes opciones:

- Escanea una imagen, de tipo array y entero de 8 bits sin signo(uint8).
- Devuelve una lista con tantos elementos como códigos de barras haya encontrado en la imagen. Estos elementos son *namedtuples* de cuatro símbolos:
  1. `type`: tipo de código de barras.

2. `data`: información de producto. Se trata de un objeto con sus métodos y atributos. En el programa se usa junto a su método `decode()` obteniendo así el contenido del código.
3. `quality`: calidad del código.
4. `position`: posición de tipo (x,y) donde se encuentra el código de barras en la imagen.

En este momento, resulta interesante hacer un breve inciso para comentar qué son conceptos como *listas* y *tuplas* en Python.

Listas, tuplas y *namedtuples*
Las *tuplas* y *listas* son estructuras que permiten trabajar con secuencias o colecciones de datos. En general posibilitan el acceso a parte de su información mediante el uso de índices. A diferencia de las *tuplas*, las *listas* son elementos mutables. Esto significa que pueden agregarse o suprimirse elementos posteriormente a la creación de la *lista*.
Existe una variación de las tuplas: las *namedtuples*. Este tipo de estructura de datos asocia cada posición con un nombre. De esta manera, es posible acceder a cierta posición tanto por su índice como por su nombre, como si de un objeto y su atributo se tratase.

Por tanto, para escanear las imágenes que se han almacenado en local, previamente se deben importan las librerías `zbar` y `zbar.misc`. Para la manipulación de imágenes hay que importar `numpy` y `Pillow`, de la que se escogerán los módulos `Image` e `ImageFile`.

Para evitar que las imágenes corruptas, en caso de que existan, detengan el programa principal, se añade la siguiente opción inmediatamente después de importar las librerías:. De este modo, se puede seguir con la ejecución y el escaneado de nuevas imágenes, potencialmente no corruptas:

```
1 Image.File.LOAD_TRUNCATED_IMAGES = True
```

En este momento, se debe cambiar (localmente) el directorio de trabajo a aquel en el que se guardan las imágenes, ya que allí es donde se va a realizar el escaneado. A continuación se crea un objeto a partir de la clase `Scanner`, al que llamaremos `escaner`. Todos los pasos hasta ahora ejecutados quedan resumindos en el siguiente cajetín:

```
1  import os
2  import time
3  import zbar, zbar.misc
4  import numpy
5  from PIL import Image, ImageFile
6  ImageFile.LOAD_TRUNCATED_IMAGES = True
7
8  # Cambiamos el directorio de trabajo y guardamos la ruta en una variable
9  dir = os.chdir("/path/que/lleva/a/data")
10
11 # Crea el objeto Scanner
12 escaner = zbar.Scanner()
```

El siguiente paso es convertir las imágenes a su array equivalente en numpy. Antes que nada, el elemento a convertir se pasa a blanco y negro, ya que no es necesaria la información de color. Una vez se han hecho ambas cosas, se puede escanear `imagen2arr` y almacenar los resultados del escaneo en una variable (en este caso, `resultado`).

```
# Obtención de imágenes mediante Pillow, convertidas a blanco y negro.
imagenInicial = Image.open(elemento).convert('L')
# Convierte imagenInicial en un array, listo para ser escaneado.
imagen2arr = numpy.array(imagenInicial, dtype='uint8')

# Escaneo y resultados
resultado = escaner.scan(imagen2arr)
```

Procesado de imágenes

Con los conceptos claros respecto al escaneado de una única imagen, se puede pasar al siguiente nivel: cómo procesar todas las imágenes de un directorio. Este proceso queda ilustrado mediante el diagrama que muestra la Figura 4.5:

El primer paso, consiste en importar todos los módulos necesarios para el desarrollo del programa. La mayoría de librerías necesarias se encuentran en los fragmentos de código anteriores, a excepción de:

```
import csv
import filetype
```

Es necesario trabajar en el directorio donde la función `snapshot()` ha ido guardando las imágenes capturadas del *streaming* del dron. De esta manera, se podrá acceder a estas capturas. Los archivos donde se escribirán los resultados también se ubicarán en este directorio. Con ello, se puede obtener también el listado con los elementos de ese directorio, es decir, las imágenes a analizar.

```
directorio = os.chdir("/home/pi/Documents/Projects/Drone_QR/data")
# Cambiamos el directorio de trabajo y guarda la ruta en una variable

lista = os.listdir(directorio)
# listado de los elementos de un directorio
```

Este es la única parte del código que el usuario final debería cambiar [5] si desea que las imágenes o los archivos se encuentren en otra ruta.

A continuación, se crean los ficheros de escritura de resultados. En un primer momento los resultados se escribían en ficheros *.txt*. Se decidió modificar el programa y escribir en ficheros *.csv* para dar una mayor comodidad al usuario.

Estos archivos *.csv* (*Comma Separated Values*) son un tipo de documento no estandarizado (de formato abierto) que presenta la información en forma de tablas. En principio, las casillas de

[5] También se debería cambiar en la función `snapshot()`

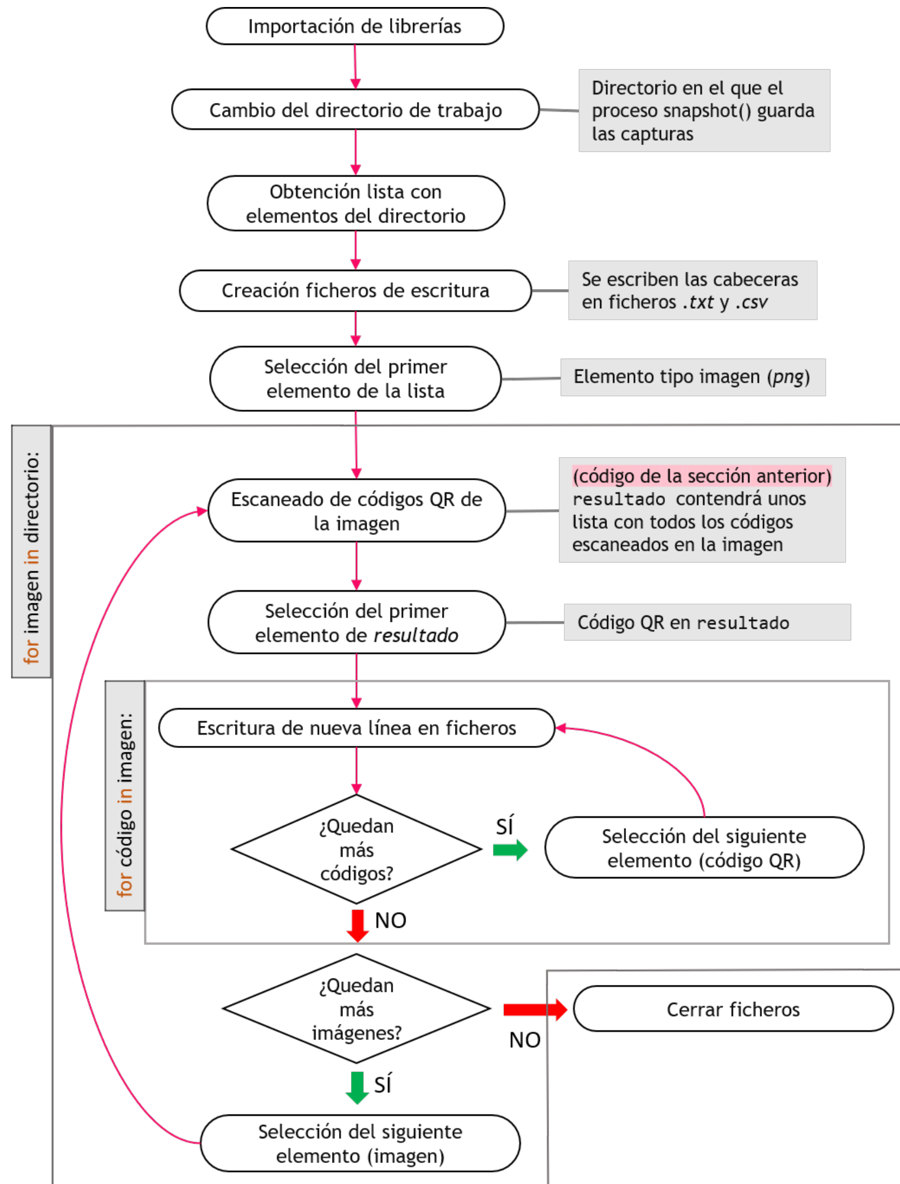


Figura 4.5: Diagrama postproceso

cada fila se separan mediante comas, pero es posible definir otro “delimitador". Este tipo de documentos son útiles para ser abiertos directamente por programas de tipo hoja de cálculo, aunque también desde un bloc de notas, como se puede ver en la Figura 4.6.

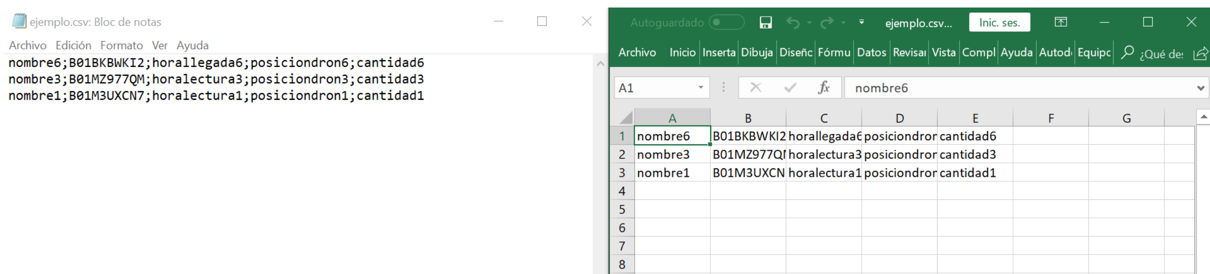


Figura 4.6: Ejemplo de archivo csv, visto desde un visor de texto plano y desde Excel.

También se valoró, en conjunto con el desarrollo de *Comprobación de inventario* (Sección 4.4.6), la posibilidad de utilizar más formatos de escritura típicos de bases de datos como *.sql*. Sin embargo, se decidió dejar únicamente los formatos *.txt* y *.csv* (Figura 4.7) para el presente prototipo, dada la facilidad y transversalidad de ambos.

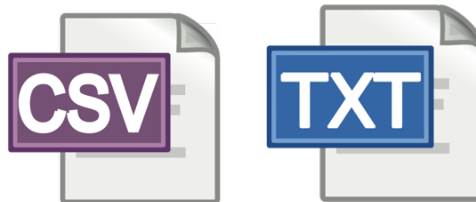


Figura 4.7: CSV y TXT son los formatos escogidos para la escritura de resultados.

Escribir en ficheros *.csv* requiere algo más que la apertura del documento. Es necesario crear un objeto de tipo *writer* al que se le especifican varios parámetros de escritura, como el ya nombrado "delimitador". Una vez abierto, se escriben las cabeceras de fichero, si se considera necesario.

```
1 # Abre ficheros y crea el writer para CSV
2 ficheroTXT = open("DocDron.txt", "a")
3 ficheroCSV=open('DocDron.csv', 'a', newline='')
4 escribir = csv.writer(ficheroCSV, dialect='excel', delimiter=';')
5
6 # Headings
7 escribir.writerow(['ASIN','CANTIDAD','FECHA','CAPTURA'])
```

En este momento, comienza el análisis en bucle de todos los elementos de la lista anterior: `for imagen in directorio:` . Elemento a elemento, se ejecuta el códico desarrollado en el apartado anterior para el escaneado de una imagen:

```
1 for elemento in lista:
2         # Obtener imagenes mediante Pillow
3         imagenInicial=Image.open(elemento).convert('L')
4         imagen2arr=numpy.array(imagenInicial, dtype='uint8')
5         resultado=escaner.scan(imagen2arr)
```

Como se dijo en dicho apartado, el método *scan* devuelve una lista con los códigos QR detectados en la imagen analizada. Con ello, llega el momento de poner por escrito los resultados obtenidos en los ficheros de escritura. Se recorren en bucle los elementos de la lista *resultado*, lo cual se corresponde con `for código in imagen:` en el diagrama de flujo.

La información a incluir en los ficheros es variable según las necesidades del usuario. Podría ser solamente el código QR, podría añadirse también el nombre de la imagen a la que pertenece, podría agregarse el momento en que se procedió a analizar la imagen, etc. En el presente proyecto se ha decidido añadir la información contenida en el proyecto, así como la captura en la que se encontraba.

La información contenida en el código QR puede, a su vez, ser variable según el uso que tenga. Por ello, la escritura en el fichrero estará precedida de un procesado de la información contenida, que diferirá notablemente en función de la aplicación concreta. Para las simulaciones llevadas a cabo se han creado códigos QR, mediante la herramienta ZXing, con un contenido determinado: código ASIN[6], fecha de llegada al almacén y cantidad de productos; en este orden y separados mediante espacios.

Formato de códigos QR
Lógicamente, si en la aplicación final de este prototipo los códigos QR tuvieran otro formato, sería necesario modificar esta parte del código. En el caso de mantener como código de producto el código ASIN, bastaría con añadir un buscador de *Regular Expressions* tal y como se hace en el programa de comprobación de inventario.

De esta manera, el código de escritura de resultados quedará de la siguiente manera:

```
# Extraccion de resultados
for symbol in resultado:
    #Preprocesado
        mensaje=symbol.data.decode('ascii')
        asin= mensaje[:9] # Código ASIN, los primeros 10 caracteres
        fecha= mensaje[11:21] #Fecha de llegada, 10 caracteres: XX-XX-XXXX
        cantidad= mensaje[23:] #Cantidad, la última parte del código
        foto= elemento[:-4] #Captura, el nombre de la imagen eliminando ".png"
        #Fichero TXT
        ficheroTXT.write('data:{}, quantity:{}, fecha:{}, captura:{}'.format(
        ↪  asin, cantidad, fecha, foto))
        ficheroTXT.write('\n')
        #Fichero CSV
        fila = [asin, cantidad, fecha, foto]
        escribir.writerow(fila)
```

Cabe destacar que la escritura de una línea en formato *.txt* es inmediata, simplemente se escribe el formato deseado para la línea y la información a incluir. La escritura en *.csv*, por el contrario, requiere dos instrucciones:

1. En la primera se da forma a la fila que se va a escribir.
2. En la segunda se escribe mediante el objeto *writer*: `escribir`.

La primera instrucción consiste en la creación de una tupla con los elementos a incluir. La escritura de una tupla coloca directamente cada elemento en una casilla diferente de la misma fila.

[6]El código ASIN es una codificación inequívoca de productos de la empresa Amazon. En la Sección 4.4.6 se entra en mayor detalle en este código.

Por último, es importante incidir en que este diseño se compone de dos bucles anidados, como se observa en el diagrama con que comienza este apartado. Esto se debe a que en cada imagen podrían encontrarse más de un código QR. La variable *resultado* contiene los códigos encontrados en la imagen i de la iteración. En general esta variable contendrá normalmente un elemento (o ninguno, si no se detectara ningún código), pero se tiene el for por si algunos QRs se encontraran tan cerca que estuvieran en la misma captura.

Escaneado simultáneo

Por último, es el momento de explicar cómo funciona el programa realmente. Como se ha dicho, la versión definitiva del programa permite que la detección de códigos QR se realice simultáneamente al vuelo del dron y la reproducción y captura de *streaming*.

Para conseguir este objetivo, se debe ejecutar en bucle el algoritmo de detección de QRs. La solución pasa por emular el código anterior, adaptándolo a la ejecución a tiempo real.

Lógicamente, no tiene sentido que se listen y escaneen todos los elementos del directorio en cada iteración del programa. Esto provocaría que las primeras imágenes se escanearan una y otra vez y, conforme el número de imágenes aumentara, aumentaría también el trabajo de procesado.

Así, en cada iteración, se debe obtener una lista solo con los elementos "nuevos", es decir, los elementos que han sido añadidos en el transcurso de la última iteración y que por tanto aún no se han analizado. A continuación, se podrá escanear dicha lista, como se hacía en el algoritmo de `procesado`.

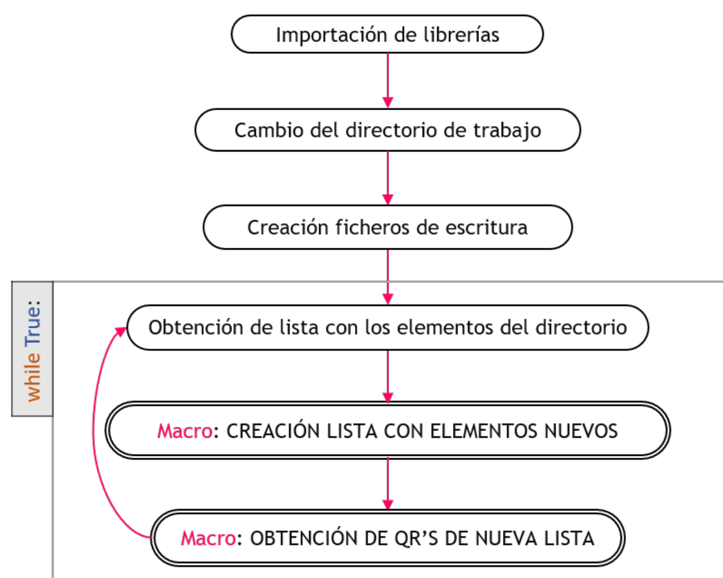


Figura 4.8: Diagrama de detección de QR's a tiempo real

Tal y como se puede observar en el esquema de la Figura 4.8, el desarrollo del algoritmo se ecuentra enmarcado dentro de un bucle de tipo `while True:` . Esto implica que el proceso nunca acabará por sí solo. Tal y como se explica en `multiproceso()` (Sección 4.4.1), es `redbaron()` el que se encarga, una vez el dron aterriza, de "matar" el resto de procesos que se encuentran en bucle infinto.

En el diagrama se observan dos bloques diferentes al resto: las `macros`. Son bloques que agrupan un subconjunto de bloques mayor que, de incluirse en la imagen, dificultarían la comprensión general del programa. El último de los dos es `Macro: OBTENCIÓN DE QR'S DE NUEVA LISTA`, ilustrado en la Figura 4.9. Consiste en el algoritmo de bucles de la sección anterior, encargado de escanear una lista de imágenes y escribir los códigos QR obtenidos en ella.

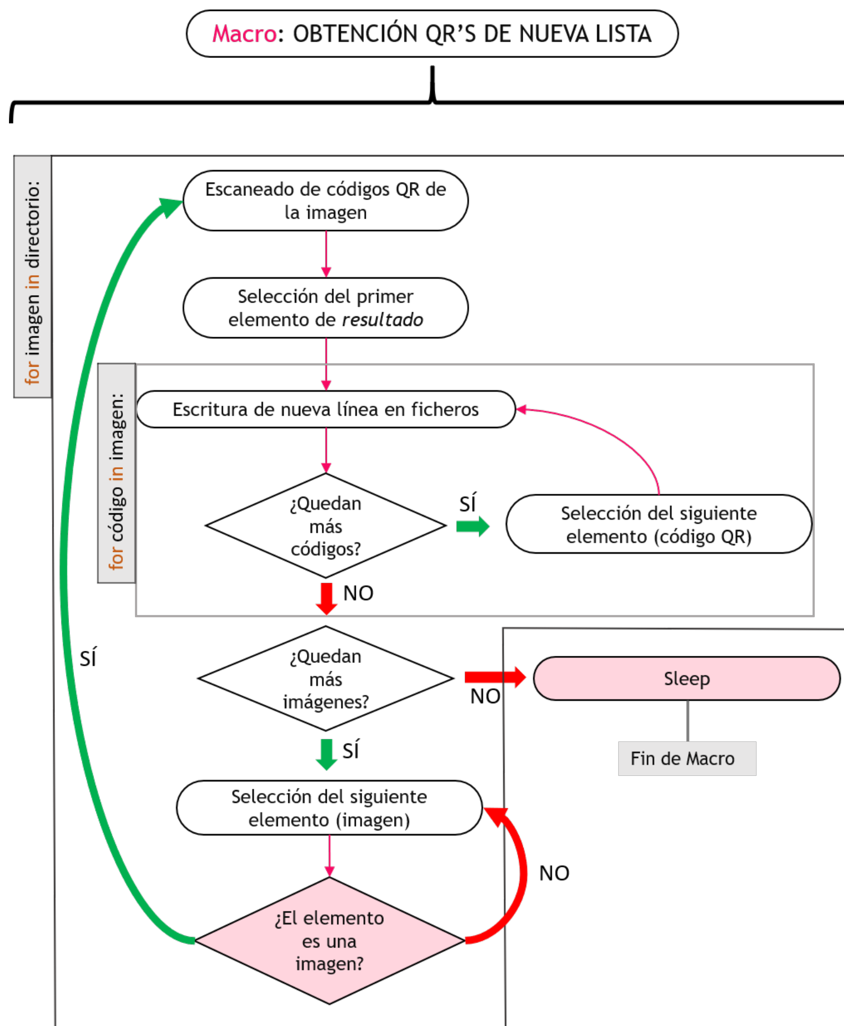


Figura 4.9: Diagrama de MACRO: obtención de QR's de nueva lista

Se puede apreciar que existen ligeras diferencias, marcadas en color rosa, respecto a los bucles explicados anteriormente (Figura 4.5).

La primera de ellas simplemente consiste en eliminar el cierre de ficheros a la salida de los bucles, ya que, al funcionar a tiempo real, el programa no finaliza en ese punto, sino que realiza una nueva iteración. La instrucción es sustituida por un breve `sleep` que pausa el proceso el tiempo suficiente para asegurarse de que en la siguiente iteración la función `snapshot()` habrá introducido nuevas imágenes en el directorio.

La segunda y más importante es un bloque de decisión que se asegura de que cada elemento nuevo es una imagen o, mejor dicho, es un archivo de formato *png*. Si es el caso, se procede al escaneado normal; y si no, se elige el siguiente elemento de la lista.

```
for elemento in lista:
##        Nos aseguramos de que es un elemento imagen
    if not elemento[-4:} == ".png":
        continue

##        Escaneado del elemento
```

Para ello, se analizan las últimas 4 posiciones del *string* correspondiente al elemento de la iteración; si es una imagen, esos 4 caracteres serán exactamente *".png"*. Si no (`not`) se coresponde con un archivo *png*, la instrucción `continue` hace que el programa pase a la siguiente iteración del bucle `for`, sin escanear el elemento.

Es importante destacar por qué esta instrucción no era necesaria en el sistema de procesado. En ese programa, la apertura de los archivos de resultado es posterior al listado de elementos del directorio, con lo que dichos ficheros no estarán en la lista. Sin embargo, en el programa a tiempo real se realiza un listado nuevo en cada iteración y los ficheros se encuentran creados desde el primer momento. Sin estas líneas de código las funciones de escaneado dan errores que hacen saltar el programa repentinamente.

Por último, se encuentra la `macro` encargada de crear una sublista, a partir de la lista total de elementos del directorio, con los elementos "nuevos"de cada iteración: `Macro: CREACIÓN LISTA CON ELEMENTOS NUEVOS`. La solución tomada inicialmente se observa en el diagrama de la Figura 4.10.

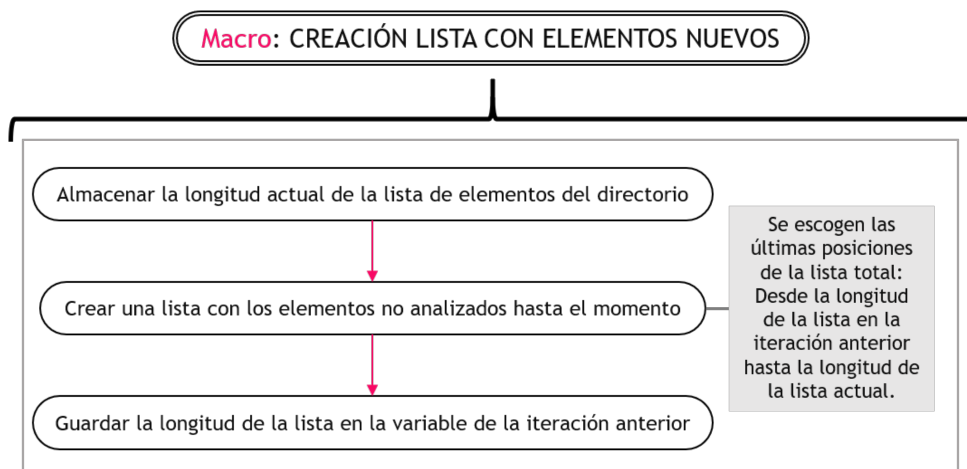


Figura 4.10: Diagrama MACRO: creación de lista con elementos nuevos. Primer modelo.

Por ejemplo, si en una iteración hay 23 elementos y en la siguiente 26, se crearía una sublista con los 3 últimos elementos:

```
while True:
    lista = os.listdir(dir)
    num_elementos = len(lista) #Numero actual en esta iteracion

    # For desde "anterior" hasta el final de la lista actual
    for elemento in lista[anterior:]:
```

```
7           # Macro: OBTENCIÓN DE QR'S DE NUEVA LISTA
8       anterior=num_elementos # Numero actual pasa a ser el numero anterior
```

Este sistema, que se llevó prácticamente hasta el final, presentaba algunos errores que se achacaban a problemas en la visualización de imágenes. Posteriormente se llegó a la conclusión de que había errores de concepto en el programa:

- Los elementos se ordenan en un directorio por orden alfabético y no por orden de entrada. Por ello, si el nombre de las nuevas imágenes es alfabéticamente anterior al nombre de los ficheros, estos ficheros estarían siempre entre los últimos elementos (quitando el puesto a otras imágenes que efectivamente son nuevas)
- Aunque el problema anterior se hubiera solucionado incluyendo dos elementos extra en todas las sublistas, `os.listdir()` devuelve los elementos ordenados aleatoriamente. De esta manera, estadísticamente habría elementos que se analizarían varias veces y elementos que ninguna vez.

Finalmente, se llegó a una solución mucho más fiable para conseguir los elementos nuevos del directorio. Para ello se hace uso de los *conjuntos* o *sets* de Python:

Los sets:
Un *conjunto* es una lista en la que ninguno de sus elementos se encuentra repetido. Mediante la instrucción `set(lista)` se pueden convertir listas en conjuntos, eliminando las repeticiones. Además, estos elementos permiten hacer las denominadas *operaciones de conjuntos*:

1. "–" *diferencia*: conjunto de elementos que están en el primer set, pero no en el segundo (resta unidireccional).
2. "^" *diferencia simétrica*: conjunto de elementos que están en un set y no en el otro, y viceversa (resta bidireccional).
3. "|" *unión*: conjunto de elementos que están en un conjunto o en el otro (suma obviando las repeticiones).
4. "&" *intersección*: conjunto de elementos que están en ambos conjuntos.

Con esta nueva herramienta, la `Macro: CREACIÓN LISTA CON ELEMENTOS NUEVOS` pasa a resolverse de la siguiente manera:

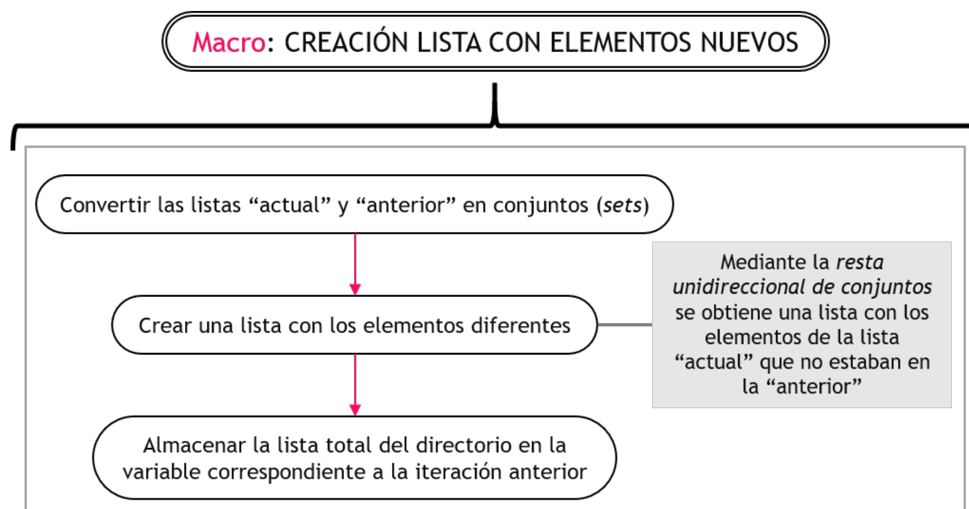


Figura 4.11: Diagrama MACRO: creación de lista con elementos nuevos. Modelo definitivo.

De esta forma, la resta unidireccional del conjunto de elementos actual menos el conjunto anterior da como resultado la lista deseada. Con este sistema, da igual el orden en el que se ordenen los elementos en el directorio o en la lista obtenida con el método `os.listdir()`. El resultado será correcto sin importar la posición que ocupe cada elemento.

```
# Creacion listas
listaAntes=list()
listaAhora=list()

while True:
    listaAhora = os.listdir(dir)

    # Conversión a conjuntos y resta del actual menos el anterior
    lista=set(listaAhora)- set(listaAntes)
    ↪  # obtenemos una lista nueva (pequeña) con elementos nuevos

    # For recorriendo la totalidad de la lista nueva
    for elemento in lista:
       # Macro: OBTENCIÓN DE QR'S DE NUEVA LISTA

    listaAntes=listaAhora # Fin iteración: actualización lista anterior
```

4.4.6 Comparativa de resultados de lectura

El siguiente paso consiste en realizar el control de inventario propiamente dicho, situándolo en el contexto de un almacén de tipo industrial. De manera simplificada, este control consistiría en comparar los códigos QR que Raspberry Pi ha detectado durante el vuelo del dron con el contenido de una base de datos.

En el apartado sobre detección de códigos QR se explicó que los códigos descifrados son puestos por escrito en un fichero de texto para su posterior comprobación. Se puede intuir que este fichero de texto será leído por un operador del almacén, una vez finalizado el vuelo del dron[7].

No obstante, esta manera de proceder puede no ser óptima desde la óptica de un gran almacén industrial. La implementación total del sistema requiere el desarrollo de una herramienta de comprobación informática. En el presente proyecto se ha incluido una herramienta que permite realizar esta función, que cerraría el proyecto cumpliendo la labor de comprobación automática.

Funcionamiento general

Por simplificar, se ha trabajado sobre el mismo tipo de ficheros. Es decir, se ha supuesto que ambas bases de datos son ficheros de la misma clase, en principio, *.txt*. Asimismo, los resultados obtenidos en la comparación se guardan en un fichero con el mismo formato.

Así pues, el primer paso del proceso consiste en la lectura de estos ficheros. El contenido de la lectura debe ser guardado mediante elementos de tipo *array*, para poder almacenar cada fila del fichero en una posición. En el contexto de programación de Python se han utilizado *listas* para tal fin.

Sin embargo, esto ha resultado ser una fuente de error ya que, si la cantidad de capturas realizadas es alta, un mismo código QR podría aparecer en varias de ellas, almacenándose varias veces. Es en este momento cuando entran en juego los conjuntos o *sets* de Python.

Estos elementos de tipo sets ya han sido utilizados en el algoritmo de detección de QRs a tiempo real. Asimismo, ya se detallaron las características en el apartado correspondiente: permiten obtener listas sin repeticiones y realizar operaciones de comparación entre ellas.

De esta manera, se realizan dichas operaciones, obteniendo con ello los resultados del control de inventario. Por último, estos deben ponerse por escrito a modo de informe para que puedan proporcionar la información requerida a los operarios del almacén industrial. En la herramienta desarrollada esta acción consiste en la escritura de un nuevo fichero de texto.

El proceso descrito puede ser ilustrado por medio del esquema que muestra la Fgura 4.12.

[7] De hecho, es así como se ha realizado en las numerosas pruebas que se han llevado a cabo durante el desarrollo del proyecto.

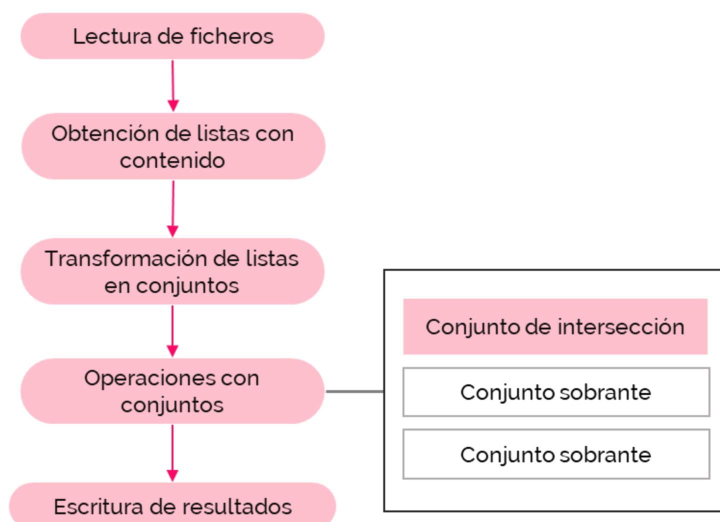


Figura 4.12: Funcionamiento del proceso de lectura y comparación de la información extraída por el escaner QR

Tratamiento de la información complementaria al código QR

A lo largo de toda la explicación del programa, se ha dado por hecho que las líneas de ambos ficheros de texto (la base de datos inicial y la obtenida tras el vuelo del dron) tienen un mismo formato para ser comparados. En caso contrario (Figura 4.13), las operaciones entre conjuntos no funcionarían correctamente, pudiendo tratar el mismo producto como si fuera uno diferente.

Sin embargo, esto podría no ser así. Como se vio en apartados anteriores, el método `scan` proporciona más información aparte del contenido del QR, que podría ser añadida en el fichero. Además, según los requerimientos de la aplicación en concreto, podría ser de utilidad añadir más información relativa al QR detectado. Por ejemplo el instante en el que se ha leído el código o la posición que ocupaba el dron en ese momento.

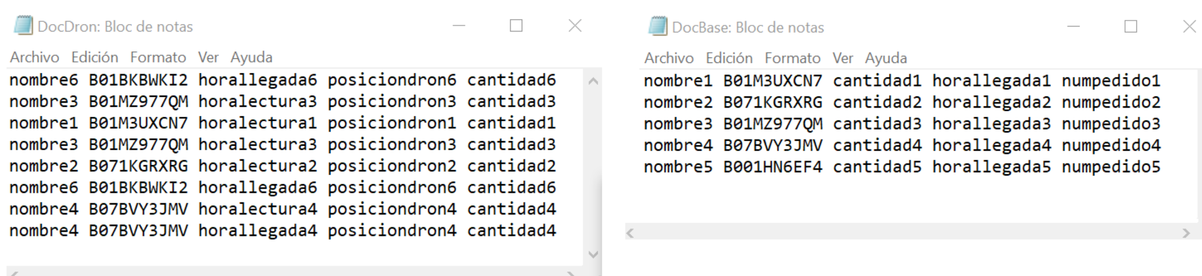


Figura 4.13: Ejemplos de diferentes contenidos en las bases de datos.

En cualquier caso, el contenido de las líneas de este fichero tendría un formato diferente al de la base de datos primigenia. Para lidiar con este problema, la solución pasa por realizar un procesado de las líneas del fichero para buscar el identificador con el que se compararán las bases de datos.

Para ello, se utilizan *Regular Expressions*. Se trata de un modulo de Python que se debe importar (`import re`) y que se encarga de buscar en un mensaje secuencias de símbolos determinadas por un patrón. Con esta funcionalidad, el código se hace robusto frente a cambios en el formato de escritura de cualquiera de las bases de datos.

El primer paso es registrar el patrón deseado en un objeto de tipo *Regular Expressions*:

```
ObjetoREGEX=re.compile(r"Patrón[A]{Buscar}")
```

A continuación, se procede a la búsqueda del patrón mediante alguno de los métodos de búsqueda de los que dispone el objeto:

```
patronREGEX = ObjetoREGEX.search("Cadena de caracteres en la que buscar")
↪  # Devuelve el primer patrón
```

```
patronesREGEX = ObjetoREGEX.findall("Cadena de caracteres en la que buscar")
↪  # Devuelve todos
```

Notar que estos métodos devuelven más información añadida al patrón encontrado, como la situación espacial del mismo dentro de la cadena de caracteres. Para quedarse solo con los patrones, se requiere el método `patrones.group()`.

El módulo de *Regular Expressions* ofrece numerosas posibilidades,pero no se van a comentar todas ellas ya que no es el objeto de este trabajo. Las funcionalidades utilizadas para definir el patrón necesario, se encuentran en el siguiente fragmento de código.

```
  """
    Creamos la Regular Expression para encontrar el identificador:

    * El identificador se compone de 10 caraceres "{10}"
    * Contiene únicamente letras mayúsculas y números "[A-Z0-9]"
    * Admitimos 12 opciones, en función de su posición ".../.../.../..."
      se han dividido en grupos correspondientes a cada linea.

        - Las primeras rodeado de dos espacios, comas o puntos y coma:
                "\s___\s" ",___," ";___;"
        - Las segundas al inicio de la cadena y seguidas de espacio, coma o punto y coma:
                "^___\s" "^___," "^___;"
        - Las terceras rodeado de comas o puntos y comas con espacios:
                ",\s___," ";\s___;"
        - Las cuartas al final de la cadena con espacio, coma o punto y coma:
                "\s___$" ",___$" "___$"
        - La quinta ocupando la totalidad de la cadena (principio y fin):
                "^___$"

    * El parámetro re.VERBOSE permite separar la cadena en varias lineas,
        ignorando los espacios, tabulaciones y caracteres creados para ello.
    """

    identificadorREGEX=re.compile(r"""
                                \s[A-Z0-9]{10}\s|,[A-Z0-9]{10},|;[A-Z0-9]{10};|
```

```
26                                  ^[A-Z0-9]{10}\s|^[A-Z0-9]{10},|^[A-Z0-9]{10};|
27                                  ,\s[A-Z0-9]{10},|;\s[A-Z0-9]{10};|
28                                  \s[A-Z0-9]{10}$|,[A-Z0-9]{10}$|;[A-Z0-9]{10}$|
29                                  ^[A-Z0-9]{10}$""", re.VERBOSE)
```

Se ha de tener en cuenta que se han tomado los códigos ASIN de Amazon como referencia para realizar el proyecto y las pruebas. Estos códigos, únicos e inequívocos para cada producto, se componen de 10 caracteres formados por letras (mayúsculas) y números en cualquier orden. Así, el patrón se ha definido para encontrar estas secuencias en cualquier lugar de la cadena, separados mediante espacios, comas o puntos y coma del resto de información. En la Figura 4.14 se muestra un ejemplo del tipo de ficheros con los que se espera trabajar.

```
B071KGRXRG horalectura2 posiciondron2 cantidad2 nombre2
horalectura3 posiciondron3 cantidad3 nombre3 B01MZ977QM
nombre1 B01M3UXCN7 horalectura1 posiciondron1 cantidad1
B07BVY3JMV
B01BKBWKI2, horallegada6, posiciondron6, cantidad6, nombre6
horallegada6;posiciondron6;B01BKBWKI2;cantidad6;nombre6
```

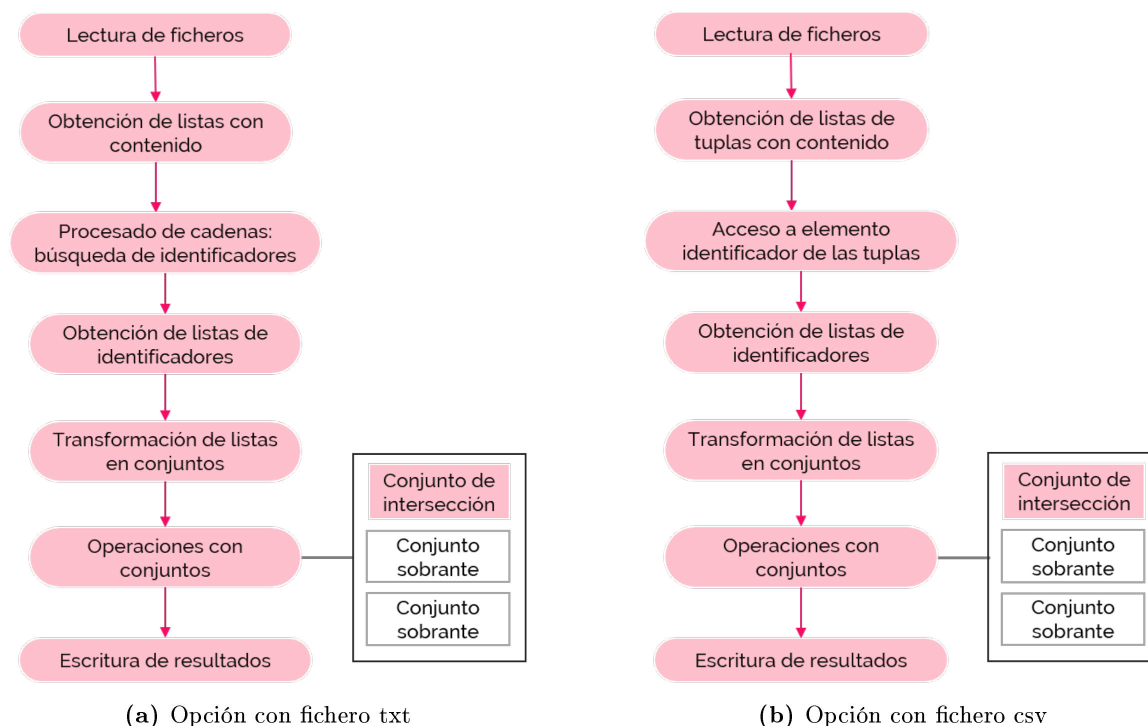
Figura 4.14: Posibles formas de encontrar el código del producto, en fichero .txt. Ejemplo.

Sin embargo, existe otra opción más eficiente: los archivos CSV. En apartados anteriores se habló de estos archivos con formato para ser leídos directamente mediante programas de hoja de cálculo tipo Excel.

Al leer estos archivos cada una de las filas se convierte en una lista. Con ello se obtiene una lista cuyos elementos (filas) son otras listas. Mediante este sistema, no es necesario recorrer la cadena de caracteres en busca del identificador. Basta con acceder a la posición de la lista que, en cada fila, contiene el mensaje del código QR.

En cualquiera de los dos casos, tanto con ficheros de texto y el uso de *Regular Expressions*, como con ficheros CSV y el trabajo con listas, se puede trabajar con bases de datos que contengan diferente información. Mediante estos mecanismos se obtiene, para cada fichero, una lista con el identificador de producto contenido en cada linea. Estas listas ya pueden ser comparadas correctamente entre sí por medio de las operaciones de conjuntos.

Una vez obtenido el resultado, este puede ponerse por escrito en modo texto normal o modo CSV, según convenga. Además, de cara a escribir los resultados, es posible recuperar la información complementaria al identificador deseado. De esta manera, el funcionamiento queda representado en la Figura 4.15.



(a) Opción con fichero txt

(b) Opción con fichero csv

Figura 4.15: Diagramas extendidos de la lectura y comparativa de información en función del archivo de almacenamiento de datos

Código en Python

Una vez se han descrito las acciones a realizar por el software, en esta sección se pretende entrar en el funcionamiento del mismo con mayor detalle. Cabe decir que esta etapa es la más variable en función de las necesidades particulares de la empresa que utilice esta herramienta.

Por ejemplo, la base de datos podría no encontrarse de manera local en la Raspberry; podría darse el caso de que el dron solo cubra una sección del almacén y, por tanto, de la base de datos; o, simplemente, podría ser que se trabaje en otro tipo de archivo, como *.sql*, en vez de *.txt* o *.csv*. Todas las posibles variables se han tratado de simplificar. Así, se asume, entre otras cosas:

- que la base de datos se encuentra almacenada de manera local en la Raspberry
- que la comprobación de inventario es realizada por ella
- que el dron cubre la totalidad del almacén (y de la base de datos)
- que los archivos se encuentran, o bien en formato *.txt*, o bien en *.csv*

Aunque la solución mediante archivos CSV puede parecer más elegante y sencilla, lo cierto es que según la aplicación concreta que el usuario le dé al programa, podría resultar de mayor utilidad tener los resultados en forma de texto escrito en vez de por casillas. Es por ello que se ha dejado la posibilidad de utilizar ambas opciones, con el objetivo de dotar de la mayor transversalidad posible a la aplicación.

Así, se han creado funciones específicas para la comparación entre archivos de tipo *.csv* y *.txt* que ejecutan los pasos descritos en el subapartado anterior. Estas funciones, descritas más adelante, son coordinadas mediante dos funciones principales: `comparacionTXT()` y `comparacionCSV()`.

Estas, a su vez, son ejecutadas desde el programa principal (*main*) tal y como se observa en la Figura 4.16.

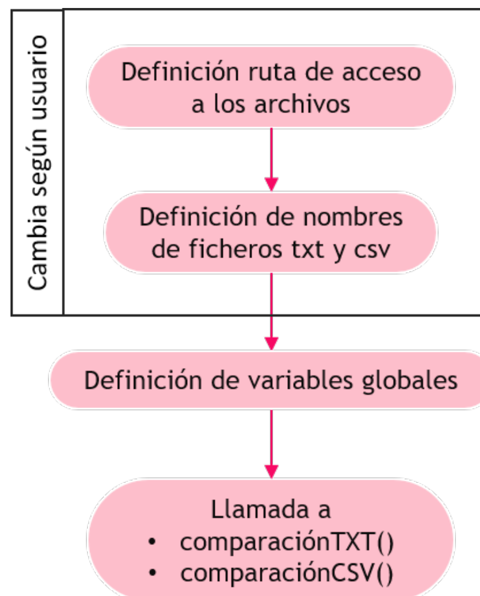


Figura 4.16: Funcion `main` de la comprobación de inventario

En el programa principal se inicializan los nombres de los ficheros que se desea comparar, tanto los de formato *.csv* como los de *.txt*. También se inicializa la ruta o *path* en la que se encuentran dichos archivos. Tal y como se ha planteado el programa, estas variables son las únicas que el usuario debería modificar para adaptarlo a su aplicación en particular; el resto del programa se ejecuta sin necesiadad de ninguna variación.

A continuación, se inicializan algunas variables pertenecientes al *global scope*, esto es, variables globales que podrán ser accedidas por diferentes funciones. Por último, se ejecutan las funciones de comparación, representadas en los dos diagramas de la Figura 4.17. Estas funciones, que gestionan a las demás, llevan a cabo las mismas acciones, solo que adaptadas a los requerimientos de los archivos *.txt* y y*.csv*, respectivamente.

En ambas funciones, el primer paso es llamar a las funciones de lectura de ficheros `leeFicheroTXT()` y `leeFicheroCSV()`, la última transcrita en el Listing 7. Las dos dan como resultado una lista, la cual es retornada a la función de comparación correspondiente. Cada elemento de la lista se corresponde con una linea del fichero.

La lectura en formato *.txt* solo requiere de un par de líneas de código, una vez abierto el fichero:

```
    for linea in fichero:
        listaEscritura.append(linea)
```

En formato *.csv* requiere de la creación de un objeto csv.reader. El delimitador definido en la creación de este objeto es determinante para las siguientes funciones.

```
# Cada fila es un elemento tipo String de la lista.
# Las columnas de CSV se separan mediante el "delimiter"->';'
```

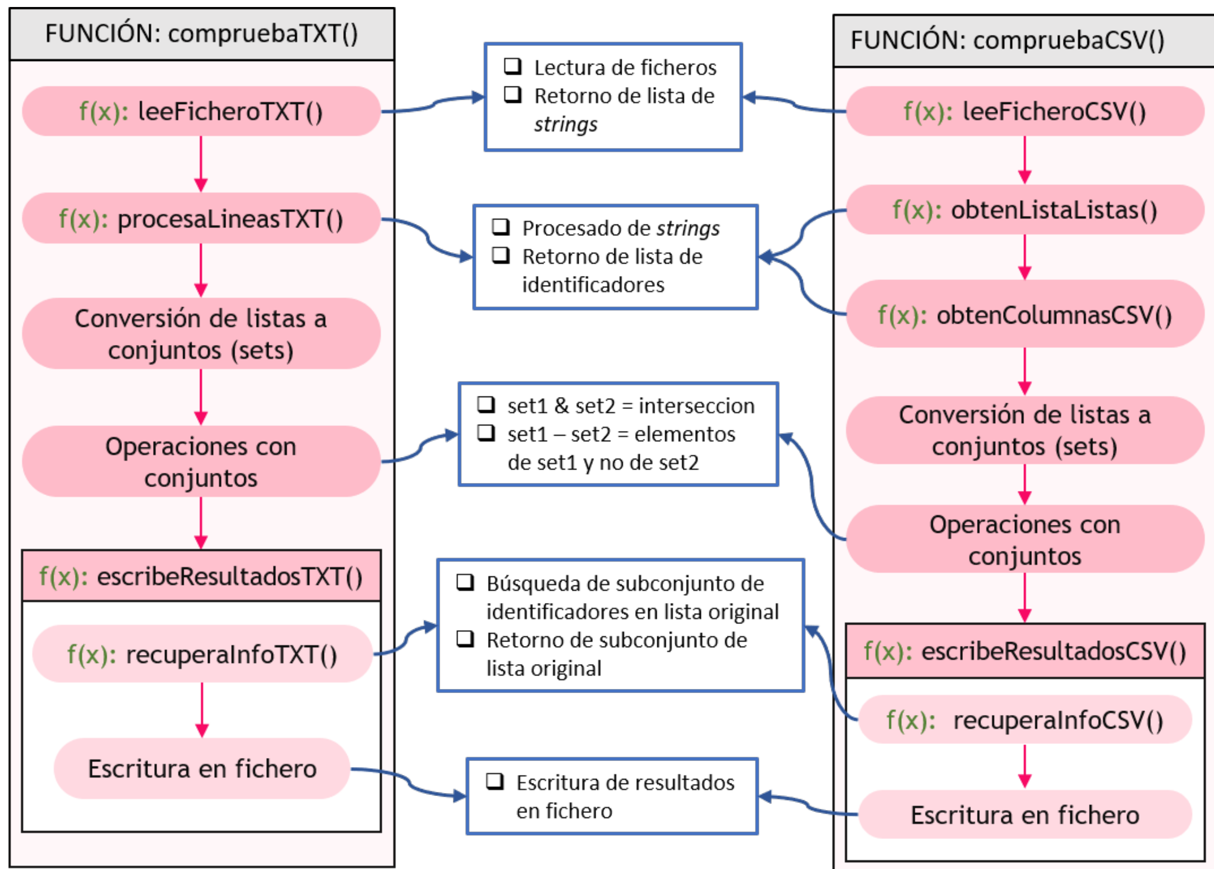


Figura 4.17: Funciones `compruebaTXT()` y `compruebaCSV()` de la comprobación de inventario

```
3  def leeFicheroCSV(nomfichero):
4      lista=list()
5      with open(nomfichero, 'r', newline='') as fichero:
6          lector=csv.reader(fichero,dialect='excel', delimiter=';')
7          for linea in fichero:
8              lista.append(linea)
9      return lista
```

Listing 7: Función `leeFicheroCSV` de la aplicación *Comprueba inventario*.

Anteriormente se dijo que la lectura del fichero tipo *.csv* retornaría una lista cuyos elementos son tuplas con las casillas de cada línea. Sin embargo, esto no es así directamente. Cada fila del fichero se convierte en una cadena de caracteres en la que el contenido de las casillas se separa mediante el *delimiter* definido, en este caso, ";". Así, el siguiente paso es dividir cada cadena de caracteres entre fragmentos entre puntos y comas, obteniendo una lista con casillas de cada línea. Para ello el siguiente paso es llamar a la función `obtenListaListasCSV()` (Listing **??**).

```
1  # Devuelve "listas de listas"
2  # Divide los Strings por casillas, entre ';' y ';'
```

```
3  def obtenListaListasCSV(listaInicial):
4      listaFinal=list()
5      for elemento in listaInicial:
6  
7          elem=elemento[:-2] # Elimina el \r\n del final de cada linea
8          prim=0
9          seg=0
10         contador=0
11 
12         # Cada elemento de la "lista principal" (cada fila),
13         # es una lista a su vez: "listaInterna"
14         listaInterna=list()
15 
16         # Se obtienen todas las casillas (excepto la última)
17         for caracter in elem:
18             if caracter==';':
19                 seg=contador
20                 listaInterna.append(elem[prim:seg])
21                 prim=contador+1
22             contador+=1
23 
24         # Obtención de la última casilla
25         listaInterna.append(elem[prim:len(elem)])
26         # La lista interna (con las casillas de esta fila) esta completa
27 
28         # Se añade a la lista de listas un elemento "fila":
29         # una lista interna con las casillas
30         listaFinal.append(listaInterna)
31 
32     return listaFinal
33 # Fin de obtenListaTuplas
```

Listing 8: Función `obtenListaListasCSV` de la aplicación *Comprueba inventario*.

Como se ha explicado anteriormente, el contenido de los ficheros puede ser variable con la aplicación, por lo que se hace necesario aislar el identificador de cada producto del resto de la información. Con este objetivo, las funciones principales llaman a `procesaLineasTXT()` y `obtencolumnaCSV()`, respectivamente (Listing 9).

En el caso de la función para ficheros *.csv*, esta función simplemente permite obtener una lista reducida, cogiendo, de cada fila, el elemento correspondiente al identificador. Para ello, solo es necesario conocer el numero de la columna que contiene esa información en cada fichero.

Por contra, la función para *.txt* requiere de un procesado algo mayor. Cada linea contiene una cadena de caracteres de longitud indeterminada en la que se debe buscar el identificador de

producto. Como se ha explicado con anterioridad, se hace uso de las *Regular Expressions* para lidiar con ello.

```
def procesaLineasTXT (listaOrigen):

    # Creamos la Regular Expression para encontrar el identificador:

        identificadorREGEX=re.compile(r"""
                                \s[A-Z0-9]{10}\s|,[A-Z0-9]{10},|;[A-Z0-9]{10};|
                                ^[A-Z0-9]{10}\s|^[A-Z0-9]{10},|^[A-Z0-9]{10};|
                                ,\s[A-Z0-9]{10},|;\s[A-Z0-9]{10};|
                                \s[A-Z0-9]{10}$|,[A-Z0-9]{10}$|;[A-Z0-9]{10}$|
                                ^[A-Z0-9]{10}$""", re.VERBOSE)
    listaSimple=list()
    listaFinal=list()

    for elemento in listaOrigen:
        # Buscamos la Regular Expression en cada fila
        # Esto proporciona mas info aparte del mensaje
        # Por eso hace falta el .group()
        encontrado=identificadorREGEX.search(elemento)

        if encontrado==None:
            print("En esta linea  habia ningun identificador")
            continue

        # Nos quedamos con lo que nos interesa
        contenido=encontrado.group()
        listaSimple.append(contenido)

    for ident in listaSimple:
        # Elimina los espacios de antes y despues (si los hay) y el \n
        if ident[0]==" ": listaFinal.append(ident[1:-1])
        else: listaFinal.append(ident[:-1])

    return listaFinal
```

Listing 9: Función `procesaLineasTXT` de la aplicación *Comprueba inventario*.

Como se observa, esta función se encarga de buscar los identificadores y agruparlos en una lista. Además, modifica los elementos encontrados, eliminando los espacios, caracteres de fin de línea, etc. que rodean a los identificadores.

Una vez obtenidas las listas de identificadores, las funciones de comparación realizan las denominadas operaciones de conjuntos, obteniendo los subconjuntos: *interseccion*, *sobrante* y *faltante*, tal y como se puede apreciar a continuación:

```
# Comparación de conjuntos
interseccion=setcolumnaLectura & setcolumnaTeorica
falta=setcolumnaTeorica - setcolumnaLectura
sobra=setcolumnaLectura - setcolumnaTeorica
```

Con ello, solo resta poner los resultados por escrito por medio de `escribeResultadosTXT()` y `escribeResultadosCSV()`. Ambas funciones realizan acciones simétricas, creando y escribiendo los ficheros de manera análoga a la lectura. Previamente, se llama a las funciones `recuperaInfoTXT()` (Listing 10) y `recuperaInfoCSV()` (Listing **??**).

```
def recuperaInfoTXT(lista,dron):
    listaCompleta=list()
    global listaTeorica
    global listaLectura
    if dron:
        listapadre=listaLectura # Lista de listas generada x el dron
    else:
        listapadre=listaTeorica # Lista de listas generada x base datos
    for identificador in lista:
        for fila in listapadre:
            if identificador in fila:
                listaCompleta.append(fila)
                break
    return listaCompleta
```

Listing 10: Función `recuperaInfoTXT` de la aplicación *Comprueba inventario.*

```
def recuperaInfoCSV(lista,dron):
    listaCompleta=list()
    columna=0 #Columna en la que se debe encontrar el identificador
    global listaDeListasLectura
    global listaDeListasTeorica
    if dron:
        listapadre=listaDeListasLectura # Lista de listas generada x el dron
    else:
        listapadre=listaDeListasTeorica # Lista de listas generada x base datos
    for elemento in lista:
        for fila in listapadre:
            if elemento==fila[columna]:
                listaCompleta.append(fila)
    return listaCompleta
```

Listing 11: Función `recuperaInfoCSV` de la aplicación *Comprueba inventario*.

Estas funciones sirven para, obtener la información completa de las listas originales correspondiente a un subconjunto de identificadores. Una vez obtenidos las listas finales, estas son transcritas en los ficheros de resultados.

Simultaneidad de procesos

Cabe destacar que, a diferencia del resto de programas descritos en apartados anteriores, este no se ejecuta de manera simultánea al resto de procesos. Por el contrario, una vez que el subproceso que controla el vuelo del dron finaliza la ruta, este manda la orden de finalización de los subprocesos "hermanos" y, por tanto, del "padre". Ello devuelve al usuario a la interfaz gráfica, desde la que se podría proceder a la comparación con la base de datos.

El motivo es que una supuesta comprobación simultánea no proporciona ningún beneficio. Si el objetivo es obtener un resultado de la comparación de dos bases de datos, no es determinante que esta se realice durante el vuelo o inmediatamente después de la finalización del vuelo.

Sin embargo, dado que la Raspberry es un dispositivo con importantes limitaciones respecto a un ordenador, añadir más subprocesos simultáneos podría ser perjudicial para la ejecución principal.

Teniendo en cuenta lo dicho, lo mismo podría decirse del programa de Python que realiza la detección de códigos QR. Más valdría realizar el procesado de las imágenes *a posteriori* para no ralentizar el resto de la ejecución.

No obstante, se ha intentado ejecutar este programa de manera simultánea para dotar al sistema de funcionalidades complementarias al control de inventario, las cuales podrían ser desarrolladas en versiones posteriores del proyecto. El ejemplo más obvio es la búsqueda de un elemento concreto. En este caso, resulta conveniente analizar los códigos QR al instante y comparar cada uno con el buscado. Por el contrario, comparar en cada iteración el código detectado con la base de datos completa no añade ventajas significativas.

4.4.7 Interfaz de usuario

La interfaz gráfica es una interfaz de usuario sencilla, que pretende facilitar la interacción entre el programa y la persona. Sin ella las aplicaciones se ejecutarían desde el terminal CLI del ordenador, un entorno que puede intimidar a muchos y que implica algunos riesgos para usuarios no experimentados.

La interfaz está desarrollada mediante `yad`, un *fork* de la herramienta de desarrollo de interfaces para CLI `zenity`. Como describen en su wiki, es un programa para la creación de ventanas diálogo de estilo GTK+. Su principal ventaja es que es sencillo de programar y se consigue un estilo consistente y adaptable a distintos sistemas operativos.

La interfaz cuenta con una ventana principal en la que se presenta el programa y pregunta qué tareas se desean realizar. Las variables `TITULO` y `TEXTITO` incluyen la información que se va a mostrar en el título y dicha pregunta. Además, se incluyen tres botones que permiten escoger qué se desea hacer: leer información del producto, ejecutar algún programa o leer los resultados.

```
TITULO="droneIntheShell"
TEXTITO="Bienvendo a droneInTheShell \n
            Por favor, escoge una de las siguientes opciones."
inicio=$(yad \
        --text-align=center \
        --title="$TITULO" \
    --text="$TEXTITO" \
    --borders=10 \
    --width=240 \
    --button="Informacion":2 \
    --button="Ejecutar...":3 \
    --button="Resultados":4)
```

Estos botones, junto al de Salir y Ok, devuelven un número del 0 al 4 que se almacena en la variable `resultado` mediante: `resultado=$?`. Estos números se interpretan de distintas formas:

- 0 : el programa se ha ejecutado correctamente.
- 1 : se desea salir del programa.
- 2 : se desea leer la información.
- 3 : se desea ejecutar un programa.
- 4 : se desea leer los resultados.

En caso de un funcionamiento normal, se saldrá cuando se cierre la ventana de forma habitual, o presionando Salir.

```
[[ $resultado -eq 1 ]] && exit 0
```

Si, por contra, se desea obtener más información del *software*, aparecerá una nueva ventana en la que se leerá la información. Esta ventana cuenta adicionalmente con dos botones: uno cierra el programa completamente y otro permite la reejecución del GUI, sin cerrar la ventana. Esto se hace para facilitar el acercamiento a la interfaz. El código se resume a continuación.

```
INFO=$(<info.txt)
if [[ $resultado -eq 2 ]]; then
yad --title="Info" \
        --text="$INFO" \
    --borders=20 \
    --width=240 \
    --button="Volver:bash -c /path/a/droneInTheShell.sh" \
    --button=gtk-close:1
```

Una vez el usuario presiona el botón `Ejecutar`, se abre un nuevo cuadro de diálogo. En él se ofrecen tres posibilidades de ejecución: la comparación de ficheros, el programa de misión de reconocimiento o cerrar la ventana. El código sería el siguiente:

```
elif [[ $resultado -eq 3 ]]; then
yad --title="Ejecutar..." \
        --text="¿Qué programa quieres ejecutar?" \
        --text-align=center \
    --borders=10 \
    --width=20 \
    --button="comparaFicheros:python3 comparaFicheros.py'' \
    --button="killbill:bash -c killbill.sh" \
    --button=gtk-close:1
```

- `comparaFicheros` ejecuta la comparación de las dos bases de datos: la original y la obtenida en la misión. Al final del programa aparecerá un informe de resultados, explicando qué elementos aparecen en qué base de datos. Se aconseja que se realice este paso si se ha ejecutado completa o satisfactoriamente la misión de reconocimiento.
- `killbill` ejecuta la misión de reconocimiento. Al pulsar, pasan unos segundos hasta que se abre una nueva ventana, esta vez a pantalla completa, con la retransmisión de imágenes del dron. Al salir de esta ventana, las imágenes recogidas hasta el momento estarán almacenadas en el directorio indicado, al igual que el archivo de texto con el resultado de la lectura de códigos QR.

La lectura de ficheros es para el usuario que puede decantarse por una comprobación rápida o, sencillamente, no tiene interés en comparar bases de datos. Para este caso se implementa el código siguiente:

```
else
    FICHERO=$(yad --file --file-filter="*.txt *.csv" \
    --filename=/home/pi/Documents/Projects/Drone_QR/data/mensajes.txt \
      --width=300 --height=200)
    if [ ! -f "$FICHERO" ]; then
        exit 1
    else
    yad --text-info --wrap<"$FICHERO"
    fi
fi
```

En esta opción, se abre un explorador de archivos para escoger el fichero a leer. Sólo se permiten archivos de tipo .txt o .csv, puesto que estos son las posibles opciones de almacenamiento de resultados. Tampoco se permite la edición de los archivos, por la posibilidad de que alguien pretendiera falsearlos. En caso de que no existiera dicho fichero, saldría del programa. En caso de que sí exista, se abrirá en el visualizar de documentos de texto de `yad`.

Capítulo 5

Resultados

A lo largo de los capítulos anteriores, se ha desarrollado un prototipo de sistema automatizado de control de inverntario. En el presente capítulo se expondrán los resultados obtenidos en el proyecto. En primer lugar, se comentará la interfaz gráfica desarrollada. A continuación se describirán las pruebas a las que se ha sometido y los resultados obtenidos.

5.1 Interfaz gráfica 63
5.2 Simulaciones y resultados 66

5.1 Interfaz gráfica

En el apartado 4.4.7 se detalló el código desarrollado para crear la interfaz gráfica que permite al usuario ejecutar los programas y acceder a la carpeta con los resultados en cualquier momento. A continuación se muestran capturas de pantalla exponiendo el producto de dicho código.

El programa, `droneInTheShell.sh`, se presenta como un icono con forma de dron en el escritorio de Raspberry Pi. Al hacer doble *click* sobre el icono, se abre la ventana principal del GUI, como aparece en la Figura 5.1. Se trata de un menú con tres botones: “Información", “Ejecutar” y “Resultados”.

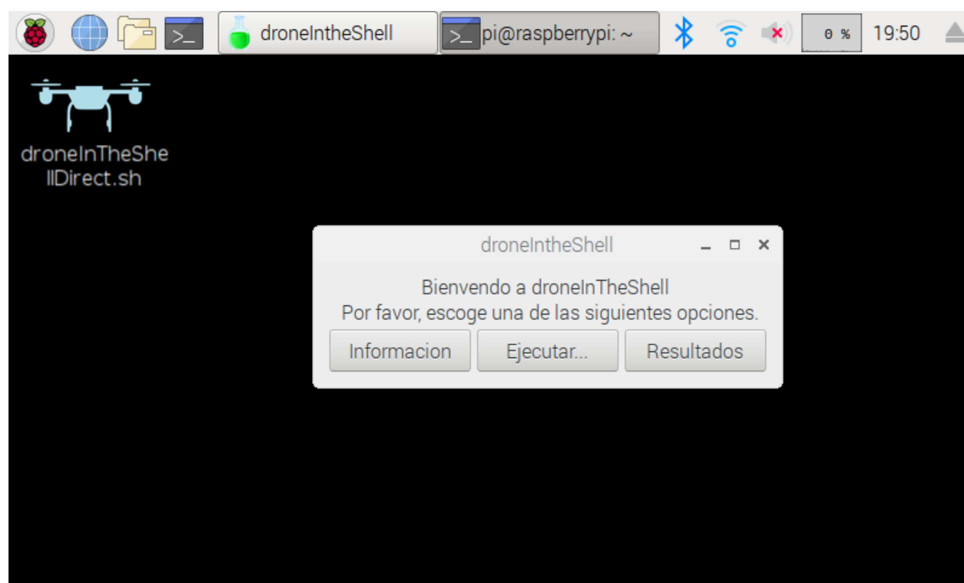


Figura 5.1: Ventana principal de `droneInTheShell`.

El botón de información abre una nueva ventana, que puede apreciarse en la Figura 5.2. Esta contiene las instrucciones necesarias para que un usuario externo utilice la aplicación sin problemas.

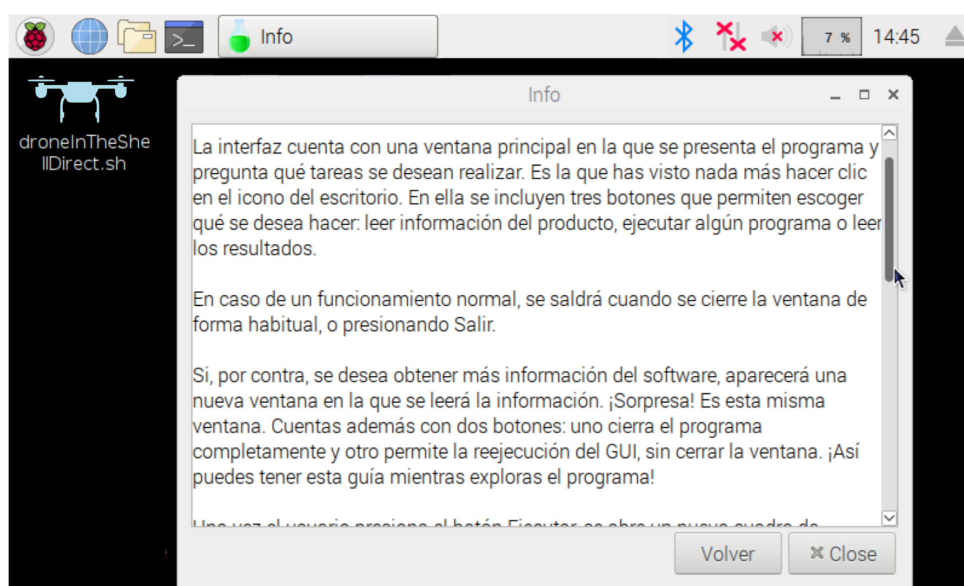


Figura 5.2: Ventana con guía del programa, en `droneInTheShell`.

El botón ejecutar abre un nuevo menú de interacción con el usuario, que ofrece dos opciones principales: ejecutar `killbill.sh` o ejecutar `comprobacion.py`. Lógicamente, cada opción se corresponde con los dos programas principales, desarrollados en apartados anteriores. En la Figura 5.3 se puede observar cómo aparece esta ventana.

Por último, el botón "resultados" abre una ventana para seleccionar el fichero que desea leerse. Tal y como se aprecia en la Figura 5.4, sólo se permiten los archivos que son de tipo .txt y .csv.

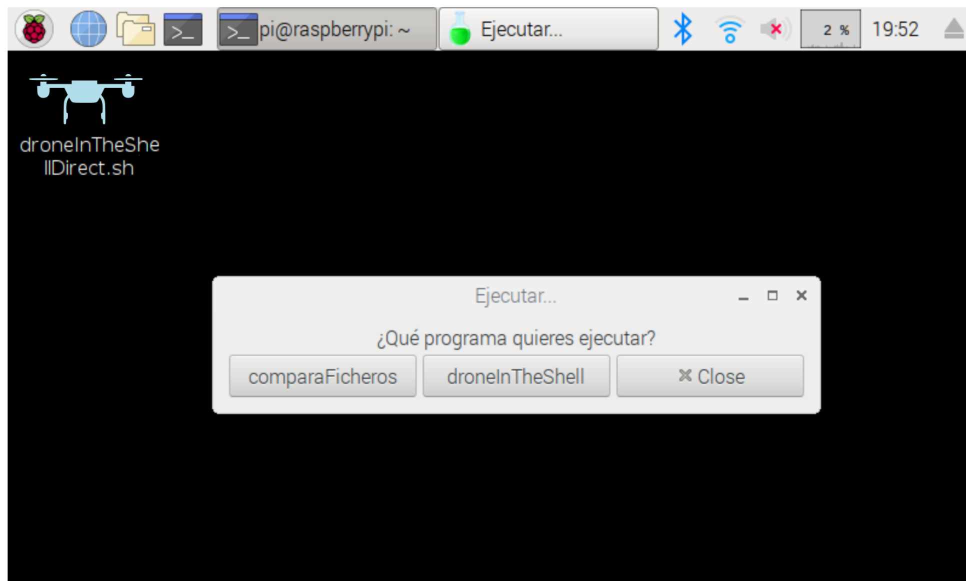


Figura 5.3: Ventana de ejecución de programas en `droneInTheShell`.

Tras seleccionarse el archivo en cuestión, aparecerá un visor de docuentos del mismo tipo que en la guía del programa.

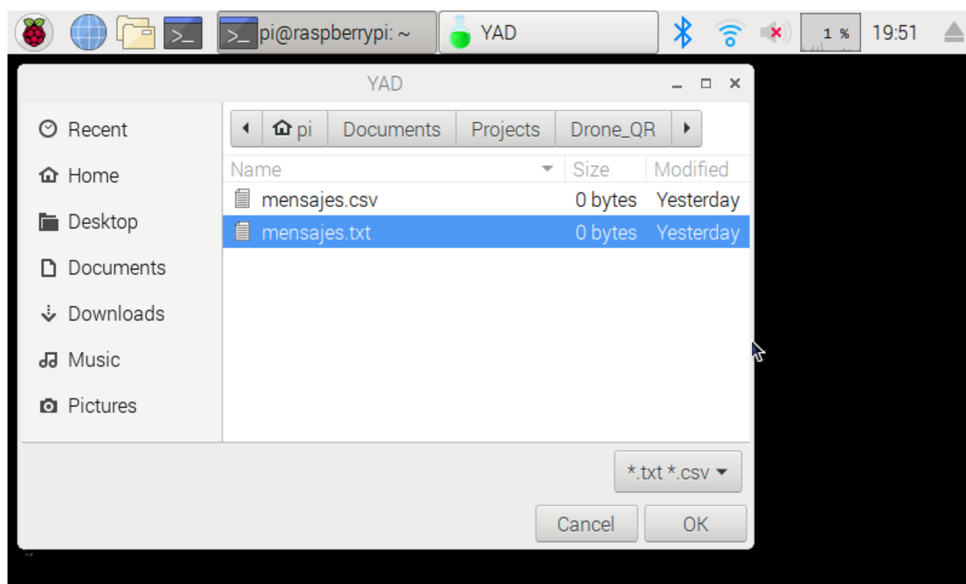


Figura 5.4: Ventana de apertura de ficheros en `droneInTheShell`.

5.2 Simulaciones y resultados

De cara a comprobar el buen funcionamiento de la aplicación, se ha sometido el sistema a una simulación de reconocimiento de códigos QR en un almacén. Esta simulación no se ha podido desarrollar en un entorno profesional ya que no se contaba con dichas instalaciones.

En su lugar se han creado varios escenarios caseros, como el que se observa en las fotografías de las Figuras 5.5 y 5.6. La recreación está hecha a base de cajas de cartón con códigos QR impresos y acoplados. Estas cajas están alineadas, tal y como se esperaría en una estantería propia de un almacén.

Figura 5.5: Entorno de simulación para misión del dron.

Así, en el programa `vueling.py` se han programado las instrucciones necesarias para que el dron siga la ruta marcada por las cajas.

Figura 5.6: Dron haciendo una misión de reconocimiento en la simulación.

El control de vuelo del dron ha sido una de las partes más conflictivas del proyecto. Ya se ha comentado en el Capítulo 1 que la batería del dron puede interferir en la precisión de las instrucciones. Además, estas imprecisiones, en un recorrido largo, pueden ocasionar que el dron se desvíe sigificativamente de su rumbo. Conforme se aumenta la longitud de la ruta y disminuye la batería disponible, los resultados empeoran sensiblemente.

En lo que se refiere a la calidad de la imagen, se nota que la cámara FPV no tiene las mejores prestaciones, es de 720p. A esto se debe sumar las limitaciones de la propia Raspberry Pi, que difícilmente es capaz de procesar 30 fps. A pesar de estos inconvenientes, se puede decir que el resultado de este prototipo es satisfactorio (Figuras 5.7 y 5.8)

Figura 5.7: Fotograma del *streaming* de vídeo.

Reduciendo el número de fps, el programa ha sido capaz de captar imágenes suficientes para que todos ellos queden registrados. Además, una gran mayoría de las imágenes tienen calidad suficiente para que los QR se escaneen adecuadamente.

Figura 5.8: Otro fotograma del *streaming* de vídeo.

Hay que tener en cuenta que no solo se están capturando imágenes: también se está reproduciendo el video en directo, enviando nuevas instrucciones de vuelo y escaneando captura a captura a tiempo real. Esto último era en un principio impensable[1] ya que eran conocidas las limitaciones del trabajo. Sin embargo, con este cambio de enfoque, las lecturas son adecuadas y se consigue un sistema más completo.

Por último, el programa encargado de realizar las comprobaciones de bases de datos ha dado un resultado sobresaliente en todas las pruebas que se han realizado. Gracias al uso de las Regular Expressions, el programa se adapta a distintos formatos y contenidos de los ficheros de texto. Se puede concluir que es una herramienta que complementa acertadamente el sistema de control desarrollado.

[1] la autora de la librería `pyparrot` ya avisaba de que las funciones de visión no estaban pensadas para Raspberry Pi.

```
RESULTADOS COMPARACIÓN BASE DE DATOS

Elementos no encontrados en el almacén:
nombre5 B001HN6EF4 cantidad5 horallegada5 numpedido5

Elementos no registrados en la base de datos:
nombre6 B01BKBWKI2 horallegada6 posiciondron6 cantidad6

Elementos correctos:
nombre2 B071KGRXRG horalectura2 posiciondron2 cantidad2
nombre4 B07BVY3JMV horalectura4 posiciondron4 cantidad4
nombre3 B01MZ977QM horalectura3 posiciondron3 cantidad3
nombre1 B01M3UXCN7 horalectura1 posiciondron1 cantidad1
```

Figura 5.9: Ejemplo poniendo a prueba el algoritmo de comparacion.

Capítulo 6

Conclusión

El presente capítulo expone las conclusiones extraídas de este trabajo académico y sus posibilidades de desarrollo y líneas futuras.

6.1 Limitaciones del trabajo 71
6.2 Conclusiones y líneas futuras 72

6.1 Limitaciones del trabajo

El presente trabajo académico busca el desarrollo de un primer prototipo para la automatización del control de inventario basado en minidrones. El monitorizado de estos minidrones se debe realizar mediante Raspberry Pi, un miniordenador de bajo coste y fácil de utilizar. Se deben cubrir tanto las tareas de vuelo como de detección de códigos QR con la información relativa al producto y la posterior comparación con la base de datos del inventario.

Sin embargo, el trabajo con minidrones, Raspberry Pi y códigos QR presenta ciertas desventajas.

- La batería de los minidrones es limitada, habitualmente máximo 10 minutos de vuelo ininterrumpido.
- La calidad de imagen de las cámaras FPV (*First Person Vision*) o USB puede ser insuficiente (720p). En ocasiones hay pequeños cortes en la transmisión o las imágenes transmitidas están corruptas. Esto puede dificultar la detección de códigos QR en las imágenes extraídas del *streaming*.
- El monitorizado de vuelo del dron *sin joystick* no siempre da los resultados esperados y depende muy estrechamente del nivel de batería, haciendo más difícil mantener las distancias entre el dron y los objetos a examinar.
- Raspberry Pi no tiene la misma potencia que un ordenador convencional y requiere la búsqueda de herramientas optimizadas para este dispositivo, como es el caso de `omxplayer` y `raspi2png`. Algunas librerías que funcionan perfectamente en equipos más potentes, como VLC, fffmpeg u OpenCV dieron problemas en Raspberry Pi 3 B.

6.2 Conclusiones y líneas futuras

Aún con las dificultades previamente expuestas, se extrae la conclusión de que el prototipo desarrollado es adecuado y cumple con las exigencias del proyecto. Además, supera las expectativas que en un principio se habían puesto en él, ya que se ha conseguido la ejecución simultánea de todas las acciones: no es necesario realizar un postprocesado de las imágenes, sino que estas se van analizando a medida que se capturan del streaming.

A pesar de la baja calidad en las imágenes, se reconocen los QRs en un alto porcentaje de ellas. Asimismo, la velocidad de captura de fotogramas es adecuada dada la velocidad de vuelo del dron, impidiendo que haya códigos que pasen desapercibidos.

Se puede decir que se ha conseguido un prototipo eficiente multipropósito y de bajo coste.

Existe todavía mucho margen para añadir nuevas funcionalidades y mejoras del sistema.

De cara al control de vuelo, sería beneficioso complementar las instrucciones con información proporcionada por los sensores del dron. Por ejemplo, mediante la geolocalización o ultrasonidos. Otra posibilidad es que mediante la cámara ventral y gracias a algún tipo de señalítica en el suelo, el dron fuera capaz de ir aprendiendo la ruta por si mismo. Es decir entrenar un modelo predictivo empleando métodos de *Deep Learning* que a partir de imágenes de la cámara ventral sea capaz de ir recorriendo los pasillos de un gran almacén. De esta manera, se dotaría al sistema de una mayor robustez, pudiendo ampliar las rutas de vuelo sin miedo a que las pequeñas desviaciones afecten al correcto desempeño.

Por otro lado, podría ser interesante incluir más opciones de formato en cuanto a archivos de escritura de resultados. Una opción sería utilizar una base de datos almacenada en local o en MySQL, con su programación específica. De cara a entornos industriales con muchos más datos, resulta una mejora importante, si bien los archivos CSV son fácilmente exportables mediante Excel a formatos de base de datos. Esta es una de las principales razones por las que se ha decidido usar este tipo de archivos, combinando la facilidad de lectura y escritura con la portabilidad.

Por último, la interfaz gráfica podría ofrecer opciones más personalizadas. Entre ellas, las más interesantes son la elección o modificación de los nombres de los archivos, las rutas en las que estos se encuentran e incluso importar una nueva ruta de vuelo.

Parte II

Presupuesto

Capítulo 7

Presupuesto

En el presente capítulo se recoge el presupuesto correspondiente al desarrollo de este proyecto. En cada una de las secciones se especifican los distintos cuadros que detallan el coste de la inversión, y que culmina con el presupuesto base de licitación.

7.1 Cuadro de precios 75
7.2 Cuadro de precios unitarios 76
7.3 Cuadro de precios descompuestos 76
7.4 Cuadro de presupuestos parciales 78
7.5 Cuadro de presupuesto base de licitación 78

7.1 Cuadro de precios

1. **Mano de obra**: Se ha considerado que el desarrollador principal es una graduada, recién titulada, en Tecnologías Industriales.

Cuadro de precios: Mano de obra	
Descripción del recurso	**Coste (€/hora)**
Ingeniera en Tecnologías Industriales recién titulado	20,00 €

Tabla 7.1: Cuadro de mano de obra

2. **Materiales**: Se han listado los materiales físicos que componen la totalidad del sistema.

Cuadro de precios: Materiales	
Descripción del recurso	**Coste (€/unidad)**
Raspberry Pi 3 B	37,90 €
Dron: Parrot Mambo FPV	179,90 €
Pantalla LCD TouchScreen 7"	83,90 €
Soporte de pantalla para Raspberry Pi	19,90 €
Tarjeta SD sistema operativo	9,90 €
Set 4 baterías minidron	60,00 €

Tabla 7.2: Cuadro de materiales

3. **Equipos**: En este cuadro de precios se ha utilizado un modelo de amortización lineal, de modo que se calcula la diferencia entre el precio de compra de dicho producto y su valor residual al final de la vida útil, y se divide entre la vida útil.

Cuadro de precios: Equipo					
Descripción del recurso	Coste (€/hora)	P. compra	P. residual	Vida útil	
				Años	Horas
Ordenador ACER Aspire V 13	0,03 €	650,00 €	0,00 €	8	23360
Software programación en Python	- €	0,00 €	0,00 €		
Software programación en Bash	- €	0,00 €	0,00 €		
Software escritura en Latex	- €	0,00 €	0,00 €		
Equipo de desarrollo Raspberry	0,02 €	151,60 €	0,00 €	3	8760

Tabla 7.3: Cuadro de equipos

Cabe destacar que Raspberry Pi aparece por duplicado: primero en Materiales y después en Equipos. Esto se debe a que el miniordenador se ha utilizado para desarrollo del trabajo académico, pero también formaría parte del propio producto. El sistema no es sólo un programa informático, sino que es un paquete de *hardware* y *software* que se proporcionaría al comprador, en el lanzamiento del producto al mercado.

7.2 Cuadro de precios unitarios

Cuadro de Precios Unitarios		
Nº Unidad de obra	Descripción de la unidad de obra	Coste unidad
1	Instalación de sistema operativo y paquetes en Raspberry Pi	767,19 €
2	Desarrollo de código de programación en Bash	1.428,07 €
3	Desarrollo de código de programación en Python	1.326,07 €
4	Simulación y depuración del sistema	1.880,78 €
5	Desarrollo de los documentos del proyecto	1.123,68 €

Tabla 7.4: Cuadro de precios unitarios

7.3 Cuadro de precios descompuestos

Raspberry Pi y sus accesorios aparecen de forma individualizada una única vez, en el apartado de Instalación, ya que es cuando se efectúa la compra del material. En posteriores apartados del cuadro, puesto que no existe una compra sino amortización del material, figura como "Equipo de desarrollo Raspberry ".

Cuadro de Precios Descompuestos					
Nº Unidad	Descripción	Unidades	Rendimiento	Precio	Importe
1	**Instalación de sistema operativo y paquetes en Raspberry Pi**	u			
	Equipos				
	Ordenador ACER Aspire V 13	h	5	0,03 €	0,15 €
	Equipo de desarrollo Raspberry	h	20	0,02 €	0,40 €
	Material				
	Raspberry Pi	u	1	37,90 €	37,90 €
	Pantalla LCD TouchScreen 7"	u	1	83,90 €	83,90 €
	Soporte de pantalla para Raspberry Pi	u	1	19,90 €	19,90 €
	Tarjeta SD sistema operativo	u	1	9,90 €	9,90 €
	Mano de Obra				
	Ingeniera en Tecnologías Industriales recién titulado	h	30	20,00 €	600,00 €
	Costes directos complementarios		0,02	752,15 €	15,04 €
			Coste Total:		767,19 €
2	**Desarrollo de código de programación en Bash**	u			
	Equipos				
	Ordenador ACER Aspire V 13	h	70	0,03 €	2,10 €
	Equipo de desarrollo Raspberry	h	70	0,02 €	1,40 €
	Software programación en Bash	h	70	- €	- €
	Mano de obra				
	Ingeniera en Tecnologías Industriales recién titulado	h	70	20,00 €	1.400,00 €
	Costes directos complementarios		0,02	1.403,50 €	28,07 €
			Coste Total:		1.428,07 €
3	**Desarrollo de código de programación en Python**	u			
	Equipos				
	Ordenador ACER Aspire V 13	h	65	0,03 €	1,95 €
	Equipo de desarrollo Raspberry	h	65	0,02 €	1,30 €
	Software programación en Python	h	65	- €	- €
	Mano de obra				
	Ingeniera en Tecnologías Industriales recién titulado	h	65	20,00 €	1.300,00 €
	Costes directos complementarios		0,02	1.303,25 €	26,07 €
			Coste Total:		1.326,07 €
4	**Simulación y depuración del sistema**	u			
	Equipos				
	Ordenador ACER Aspire V 13	h	80	0,03 €	2,40 €
	Equipo de desarrollo Raspberry	h	80	0,02 €	1,60 €
	Software programación en Bash	h	80	- €	- €
	Software programación en Python	h	80	- €	- €
	Material				
	Dron: Parrot Mambo FPV	u	1	179,90 €	179,90 €
	Set 4 baterías minidron	u	1	60,00 €	60,00 €
	Mano de obra				
	Ingeniera en Tecnologías Industriales recién titulado	h	80	20,00 €	1.600,00 €
	Costes directos complementarios		0,02	1.843,90 €	36,88 €
			Coste Total:		1.880,78 €
5	**Desarrollo de los documentos del proyecto**	u			
	Equipos				
	Ordenador ACER Aspire V 13	h	55	0,03 €	1,65 €
	Software escritura en Latex	h	55	- €	- €
	Mano de obra				
	Ingeniera en Tecnologías Industriales recién titulado	h	55	20,00 €	1.100,00 €
	Costes directos complementarios		0,02	1.101,65 €	22,03 €
			Coste Total:		1.123,68 €

Tabla 7.5: Cuadro de precios descompuestos

7.4 Cuadro de presupuestos parciales

Cuadro de presupuestos parciales					
Nº Capítulo	**Nombre del Capítulo**				**Importe capítulo**
1	Desarrollo de software				3.521,33 €
	Nº Unidad de Obra	Coste Unidad	Medición Unidades	Coste total unidades	
	1	767,19 €	1	767,19 €	
	2	1.428,07 €	1	1.428,07 €	
	3	1.326,07 €	1	1.326,07 €	
2	Simulación del prototipo				1.880,78 €
	Nº Unidad de Obra	Coste total Unidad	Medición Unidades	Coste total unidades	
	4	1.880,78 €	1	1.880,78 €	
3	Documentación				1.123,68 €
	Nº Unidad de Obra	Coste total Unidad	Medición Unidades	Coste total unidades	
	5	1.123,68 €	1	1.123,68 €	

Tabla 7.6: Cuadro de presupuestos parciales

7.5 Cuadro de presupuesto base de licitación

Nº Capítulo	Nombre Capítulo	Importe capítulo	Medición Capítulo	Importe total
1	Desarrollo de software	3.521,33 €	1,00	3.521,33 €
2	Simulación del prototipo	1.880,78 €	1,00	1.880,78 €
3	Documentación	1.123,68 €	1,00	1.123,68 €
	Presupuesto de Ejecución Material (PEM):			6.525,79 €
		Gastos generales (13%):		848,35 €
		Beneficio industrial (6%):		391,55 €
	Presupuesto de ejecución por contrata:			7.765,69 €
			IVA (21%):	1.630,79 €
	Presupuesto de Licitación:			**9.396,48 €**

Tabla 7.7: Cuadro de presupuesto base de licitación

El presupuesto base de licitación asciende a NUEVE MIL TRESCIENTOS NOVENTA Y SEIS EUROS y CUARENTA Y OCHO CÉNTIMOS.

Parte III

Anexos

Apéndice A

Anexo de código

Este anexo recoge el código de los programas implementados para el desarrollo del proyecto. Se divide según el lenguaje de programación y se incluye el script *completo. Las rutas de achivos han sido en algunos casos modificadas o acortadas para facilitar la lectura del documento pero, a excepción de este punto, es idéntico al* software *genuino.*

A.1 Bash . . . 81
A.1.1 Programa principal: `killbill` . . . 81
A.1.2 Interfaz gráfica: `droneInTheShell` . . . 82
A.2 Python . . . 83
A.2.1 Detección de códidos QR: `qrying` . . . 83
A.2.2 Comparación de ficheros: `inventario` . . . 85

A.1 Bash

A.1.1 Programa principal: killbill

```
#!/bin/bash
snapshot () {
    carpetaDestino=$1
    i=0
    SECONDS=0
    echo "Snapshots!"
    cd "$carpetaDestino"
    while (True); do
        i=$[$i+1]
        numero="$( printf "%04d" "$i'' )"
        raspi2png -p "$capetaDestino/$numero.png"
        sleep 0.2
    done
    echo "Snapshots ha acabado"
}
```

```
16
17 multiproceso () {
18     local procesos
19     while read procesos; do
20         eval "$procesos" &
21         echo "$?"< /path/a/multi.pid
22     done
23     wait
24 }
25
26 redbaron () {
27     python3 /home/pi/.../vueling.py
28     pkill -F /path/a/multi.pid
29 }
30
31 multiproceso <<STOP
32 omxplayer --display 4 --aspect-mode fill -o local rtsp://192.168.99.1/media/stream2
33 snapshot /home/pi/Documents/Projects/Drone_QR/data
34 python3 /home/pi/Desktop/qryingiter.py
35 redbaron
36 STOP
```

A.1.2 Interfaz gráfica: droneInTheShell

```
1  #!/bin/bash
2  TITULO="droneIntheShell"
3  TEXTITO="Bienvendo a droneInTheShell \n
4           Por favor, escoge una de las siguientes opciones."
5
6  inicio=$(yad \
7          --text-align=center \
8          --title="$TITULO" \
9      --text="$TEXTITO" \
10     --borders=10 \
11     --width=240 \
12     --button="Informacion":2 \
13     --button="Ejecutar...":3 \
14     --button="Resultados":4)
15
16 resultado=$?
17
18 [[ $resultado -eq 1 ]] && exit 0
19
20 if [[ $resultado -eq 2 ]]; then
21 yad --title="Info" \
```

```
            --text-info --wrap \
        --borders=20 \
        --width=240 \
        --button="Volver:bash -c /home/pi/.../droneInTheShell.sh" \
        --button=gtk-close:1< info.txt

elif [[ $resultado -eq 3 ]]; then
yad --title="Ejecutar..." \
            --text="¿Qué programa quieres ejecutar?" \
           --text-align=center \
        --borders=10 \
        --width=20 \
        --button="comparaFicheros:python3/home/pi/.../inventario.py" \
        --button="droneInTheShell:bash -c /home/pi/.../killbill.sh" \
        --button=gtk-close:1

else
    FICHERO=$(yad --file --file-filter="*.txt *.csv" \
    -filename=/home/pi/.../data/mensajes.txt \
    --width=300 --height=200)
    if [ ! -f "$FICHERO" ]; then
        exit 1
    else
       yad --text-info --wrap<"$FICHERO"
    fi
fi
```

A.2 Python

A.2.1 Detección de códidos QR: qrying

```
import os
import csv
import filetype
import time
import zbar
import zbar.misc
import numpy
from PIL import Image, ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

# Cambiamos el directorio de trabajo y guarda la ruta en una variable
dir = os.chdir("/home/pi/.../data")

```

```
14  # Abre ficheros y crea el writer para CSV
15  ficheroTXT = open("DocDron.txt", "a")
16  ficheroCSV=open('DocDron.csv', 'a', newline='')
17  escribir = csv.writer(ficheroCSV, dialect='excel', delimiter=';')
18
19  # Headings
20  escribir.writerow(['ASIN','CANTIDAD','FECHA','CAPTURA'])
21
22  # Crea el reader
23  escaner=zbar.Scanner()
24  i=0
25  time.sleep(5)
26
27  # Creacion listas
28  listaAntes=list()
29  listaAhora=list()
30
31  while True:
32      listaAhora = os.listdir(dir)
33      lista = set(listaAhora)- set(listaAntes)
34      listaAntes = listaAhora
35
36      for elemento in lista:
37          if not filetype.guess(elemento).extension == "png":
38              continue
39          # Obtener imagenes mediante Pillow
40          imagenInicial = Image.open(elemento).convert('L')
41          imagen2arr = numpy.array(imagenInicial, dtype='uint8')
42
43          resultado=escaner.scan(imagen2arr)
44
45          # Extraccion de resultados
46          for symbol in resultado:
47                  #Preprocesado
48              mensaje=symbol.data.decode('ascii')
49              asin = mensaje[:9] # Código ASIN, los primeros 10 caracteres
50              fecha = mensaje[11:21] #Fecha de llegada, 10 caracteres: XX-XX-XXXX
51              cantidad = mensaje[23:] #Cantidad, la última parte del código
52              foto = elemento[:-4]
                ↪  #Captura, el nombre de la imagen eliminando ".png"
53
54              #Fichero TXT
55              ficheroTXT.write('{}, {}, {}, {}'.format(asin, cantidad, fecha,
                ↪  foto))
56              ficheroTXT.write('\n')
```

```
57
58              #Fichero CSV
59              fila = [asin, cantidad, fecha, foto]
60              escribir.writerow(fila)
61      time.sleep(1)
62
63
64  # Cerramos fichero
65  fichero.close()
```

A.2.2 Comparación de ficheros: inventario

```
1   """
2   Este programa es un complemento a los desarrollados para el dron.
3   No se ejecuta en tiempo real. Una vez procesadas las capturas
4   del dron, se crea un fichero de texto con los QR leídos.
5   Se desea comparar el contenido del txt obtenido con el de una base de datos
6   (que contendría el contenido teórico del inventario)
7
8   """
9
10  import os
11  import csv
12  import re
13
14
15  #FUNCIONES CSV
16
17  def comparacionCSV():
18      LecturaCSV=ruta+contenidoLecturaCSV
19      TeoricoCSV=ruta+contenidoTeoricoCSV
20
21      global listaLectura
22      global listaTeorica
23      global listaDeListasLectura
24      global listaDeListasTeorica
25
26      ##Leemos los ficheros convirtiendolos en listas
27      listaLectura= leeFicheroCSV(LecturaCSV)
28      listaDeListasLectura=obtenListaListasCSV(listaLectura)
29
30      listaTeorica= leeFicheroCSV(TeoricoCSV)
31      listaDeListasTeorica=obtenListaListasCSV(listaTeorica)
32
33      columnaLectura=obtencolumnaCSV(listaDeListasLectura,0)
```

```
34      columnaTeorica=obtencolumnaCSV(listaDeListasTeorica,0)
35
36      setcolumnaLectura=set(columnaLectura)
37      setcolumnaTeorica=set(columnaTeorica)
38
39      ##Comparación de conjuntos
40      #('Elementos en común entre la lista teórica y la leída')
41      interseccion=setcolumnaLectura & setcolumnaTeorica
42      #('Elementos en la lista teórica y NO en la leída')
43      falta=setcolumnaTeorica - setcolumnaLectura
44      #('Elementos NO en la lista teórica pero SÍ en la leída')
45      sobra=setcolumnaLectura - setcolumnaTeorica
46
47      escribeResultadosCSV(resultadoCSV,interseccion,falta,sobra)
48
49  #Funcion para leer todas las filas de un archivo CSV
50  #Cada fila es un elemento tipo String de la lista.
51  #Las columnas de CSV se separan mediante el "delimiter"->';'
52  def leeFicheroCSV(nomfichero):
53      lista=list()
54      with open(nomfichero, 'r', newline='') as fichero:
55          lector=csv.reader(fichero,dialect='excel', delimiter=';')
56          for linea in fichero:
57              lista.append(linea)
58      return lista
59  # FIn de leeFicherosCSV
60
61  #Funcion que transforma las listas de la funcion anterior a "listas de tuplas"
62  #En realidad son "listas de listas"
63  #Divide los Strings por casillas, entre ';' y ';'
64  def obtenListaListasCSV(listaInicial):
65      listaFinal=list()
66      for elemento in listaInicial:
67          elem=elemento[:-2] #Elimina el \r\n del final de cada linea
68          prim=0
69          seg=0
70          contador=0
71
72     # Cada elemento de la "lista principal" (cada fila),
73     # es una lista a su vez: "listaInterna"
74          listaInterna=list()
75          for caracter in elem:
76              if caracter==';':
77                  seg=contador
78                  listaInterna.append(elem[prim:seg])
```

```
79                  prim=contador+1
80              contador+=1
81          listaInterna.append(elem[prim:len(elem)])
82
83      # Se añade a la lista de filas un elemento "fila":
84      # una lista interna con las casillas
85          listaFinal.append(listaInterna)
86      return listaFinal
87  # Fin de obtenListaTuplas
88
89  # A partir de la "lista de listas" anterior, devuelve en una lista
90  # los elementos correspondientes a las casillas de cierta columna del CSV
91  def obtencolumnaCSV(listaDeListas,columna):
92      listaRetorno=list()
93      for elemento in listaDeListas:
94          listaRetorno.append(elemento[columna])
95      return listaRetorno
96  # Fin de obtencolumnaCSV
97
98  # Busca los elementos de una lista simple en la columna correspondiente
99  # de la "lista de listas" inicial.
100 # Devuelve una lista de listas nueva,
101 # con los elementos de la lista simple completados con los correspondientes
102 # de la lista de listas inicial
103 def recuperaInfoCSV(lista,dron):
104     listaCompleta=list()
105     columna=0 #Columna en la que se debe encontrar el identificador
106     global listaDeListasLectura
107     global listaDeListasTeorica
108     if dron:
109         listapadre=listaDeListasLectura #Lista de listas generada x el dron
110     else:
111         listapadre=listaDeListasTeorica #Lista de listas generada x base datos
112     for elemento in lista:
113         for fila in listapadre:
114             if elemento==fila[columna]:
115                 listaCompleta.append(fila)
116     return listaCompleta
117 # Fin de recuperaInfoCSV
118
119 #Funcion de escritura de un nuevo fichero CSV con los resultados de la comparacion
120 def escribeResultadosCSV(nombreFichero, interseccion,falta,sobra):
121     rutaCompleta= ruta+nombreFichero
122     interseccionCompleta=recuperaInfoCSV(interseccion,True)
123     faltaCompleta=recuperaInfoCSV(falta,False)
```

```
124     sobraCompleta=recuperaInfoCSV(sobra,True)
125 
126     with open (rutaCompleta, 'w', newline='') as ficheroCSV:
127         escribir=csv.writer(ficheroCSV, dialect='excel', delimiter=';')
128         escribir.writerow(['Elementos no encontrados en el almacén'])
129         for elemento in faltaCompleta:
130             escribir.writerow(elemento)
131         escribir.writerow('')
132         escribir.writerow(['Elementos no registrados en la base de datos'])
133         for elemento in sobraCompleta:
134             escribir.writerow(elemento)
135         escribir.writerow('')
136         escribir.writerow(['Elementos correctos'])
137         for elemento in interseccionCompleta:
138             escribir.writerow(elemento)
139 #Fin de funcion escribeResultadosCSV
140 
141 
142 ##----------------------------------------------------------------------------
143 """--------------------------------------------------------------------"""
144 ##----------------------------------------------------------------------------
145 """--------------------------------------------------------------------"""
146 
147 
148 #FUNCIONES TXT
149 
150 ##Función principal de funcionamiento mediante ficheros TXT:
151 def comparacionTXT ():
152 
153     ##Leemos los ficheros convirtiendolos en listas
154     global listaTeorica
155     listaTeorica= leeFicheroTXT(contenidoTeoricoTXT)
156     global listaLectura
157     listaLectura= leeFicheroTXT(contenidoLecturaTXT)
158 
159     ## Obtencion de los identificadores de codigo QR
160     listaLecturaIdent=list()
161     listaLecturaIdent=procesaLineasTXT(listaLectura)
162     listaTeoricaIdent=list()
163     listaTeoricaIdent=procesaLineasTXT(listaTeorica)
164 
165     #print(listaTeorica)
166     print(listaTeoricaIdent)
167     #print(listaLectura)
168     print(listaLecturaIdent)
```

```
169
170
171     """
172     Conversion de las listas en conjuntos o sets
173     Los sets son listas en las que ningun elemento se encuentra repetido.
174     Al convertir las listas en sets, eliminamos los posibles QRs que
175     se hayan leido varias veces
176     """
177     setTeorico=set(listaTeoricaIdent)
178     setLectura=set(listaLecturaIdent)
179
180     #print(listaTeorica)
181     print(setTeorico)
182     #print(listaLectura)
183     print(setLectura)
184
185     ##Comparación de conjuntos
186     #('Elementos en común entre la lista teórica y la leída')
187     interseccion=setTeorico & setLectura
188     #('Elementos en la lista teórica y NO en la leída')
189     falta=setTeorico - setLectura
190     #('Elementos NO en la lista teórica pero SÍ en la leída')
191     sobra=setLectura - setTeorico
192
193     escribeResultadosTXT(resultadosTXT, interseccion,falta,sobra)
194 #Fin comparacionTXT
195
196 ##Funcion para leer el contenido de los ficheros de texto
197 ##La función devuelve una lista con el contenido de cada linea del fichero
198 def leeFicheroTXT (nombreFichero):
199     rutaCompleta=ruta+nombreFichero
200     fichero=open(rutaCompleta, 'r')
201     listaEscritura=list()
202     for linea in fichero:
203         listaEscritura.append(linea)
204     fichero.close()
205     return listaEscritura
206 ##Fin leeFicheroTXT
207
208 ##Funcion destinada a encontrar el identificador de Amazon en una cadena
209 ##de caracteres mediante REGEX
210 def procesaLineasTXT (listaOrigen):
211
212     """
213     Creamos la Regular Expression para encontrar el identificador:
```

```
214
215     * El identificador se compone de 10 caraceres "{10}"
216     * Contiene únicamente letras mayúsculas y números "[A-Z0-9]"
217     * Admitimos 12 opciones, en función de su posición "...|...|...|..."
218       se han dividido en grupos correspondientes a cada linea.
219
220         - Las primeras rodeado de dos espacios, comas o puntos y coma:
221                 "\s___\s" ",___," ";___;"
222         - Las segundas al inicio de la cadena y seguidas de espacio, coma o punto y coma:
223                 "^___\s" "^___," "^___;"
224         - Las terceras rodeado de comas o puntos y comas con espacios:
225                 ",\s___," ";\s___;"
226         - Las cuartas al final de la cadena con espacio, coma o punto y coma:
227                 "\s___$" ",___$" "___$"
228         - La quinta ocupando la totalidad de la cadena (principio y fin):
229                 "^___$"
230
231     * El parámetro re.VERBOSE permite separar la cadena en varias lineas,
232         ignorando los espacios, tabulaciones y caracteres creados para ello.
233     """
234
235     identificadorREGEX=re.compile(r"""
236                                 \s[A-Z0-9]{10}\s|,[A-Z0-9]{10},|;[A-Z0-9]{10};|
237                                 ^[A-Z0-9]{10}\s|^[A-Z0-9]{10},|^[A-Z0-9]{10};|
238                                 ,\s[A-Z0-9]{10},|;\s[A-Z0-9]{10};|
239                                 \s[A-Z0-9]{10}$|,[A-Z0-9]{10}$|;[A-Z0-9]{10}$|
240                                 ^[A-Z0-9]{10}$""", re.VERBOSE)
241
242     listaSimple=list()
243     listaFinal=list()
244     for elemento in listaOrigen:
245         # Buscamos la Regular Expression en cada fila
246         encontrado=identificadorREGEX.search(elemento)
247
            ↪  # Esto proporciona mas info aparte del mensaje. Por eso hace falta el .group()
248
249         if encontrado==None:
250             print("En esta linea  habia ningun identificador")
251             print(elemento)
252             continue
253         contenido=encontrado.group()#Nos quedamos con lo que nos interesa
254         identificador=contenido ##Busca el codigo identificador
255         print (identificador)
256         listaSimple.append(identificador)
257
```

```
258     for ident in listaSimple:
259
         ↪  # Elimina los espacios de antes y despues (si los hay) y el fin de linea
260         if ident[0]==" ": listaFinal.append(ident[1:-1])
261         else: listaFinal.append(ident[:-1])
262
263     return listaFinal
264 ## Fin de procesaLineasTXT
265
266 ##Funcion contraria a la inmediatamente anterior:
267 ##Busca, en la lista origen, la linea que contenga el identificador deseado
268 def recuperaInfoTXT(lista,dron):
269     listaCompleta=list()
270     global listaTeorica
271     global listaLectura
272     if dron:
273         listapadre=listaLectura #Lista de listas generada x el dron
274     else:
275         listapadre=listaTeorica #Lista de listas generada x base datos
276     for identificador in lista:
277         for fila in listapadre:
278             if identificador in fila:
279                 listaCompleta.append(fila)
280                 break
281     return listaCompleta
282 # Fin de recuperaInfoTXT
283
284 ##Funcion para escribir los resultados de la comparación
285 def escribeResultadosTXT(nombreFichero, interseccion,falta,sobra):
286     rutaCompleta= ruta+nombreFichero
287     print (rutaCompleta)
288     interseccionCompleta=recuperaInfoTXT(interseccion,True)
289     faltaCompleta=recuperaInfoTXT(falta,False)
290     sobraCompleta=recuperaInfoTXT(sobra,True)
291     print (interseccionCompleta)
292     print (faltaCompleta)
293     print (sobraCompleta)
294
295     fichero=open(rutaCompleta, "w")
296
297     fichero.write("RESULTADOS COMPARACIÓN BASE DE DATOS\n\n")
298
299     fichero.write("Elementos no encontrados en el almacén:\n")
300     for elemento in faltaCompleta:
301         fichero.write(elemento)
```

```python
    fichero.write('\n')
    fichero.write("Elementos no registrados en la base de datos:\n")
    for elemento in sobraCompleta:
        fichero.write(elemento)
    fichero.write('\n')
    fichero.write("Elementos correctos:\n")
    for elemento in interseccionCompleta:
        fichero.write(elemento)
# Fin de escribeResultadosTXT


##-----------------------------------------------------------------------
"""-------------------------------------------------------------------"""
##-----------------------------------------------------------------------
"""-------------------------------------------------------------------"""


#PROGRAMA PRINCIPAL -> MAIN

"""
Para saber cómo escribir la ruta en la apertura de ficheros,
puede ser util escribir estas lineas en el Shell de Python
import os
os.getcwd()
También sirve desde el terminal, escribiendo:
pwd
"""
#Ruta donde se encuentran todos los ficheros
ruta= 'C:\\Users\\irenita\\'

# Nombres de los ficheros de tipo TXT
contenidoTeoricoTXT = 'DocBase.txt'
contenidoLecturaTXT = 'DocDron.txt'
resultadosTXT = 'DocComparacion.txt'

# Variables de las funciones TXT que tienen que ser globales (global scope)
listaTeorica=list()
listaLectura=list()

# Funcion pricipal TXT
comparacionTXT()

# Nombres de los ficheros de tipo CSV
contenidoLecturaCSV = 'DocDron.csv'
contenidoTeoricoCSV = 'DocBase.csv'
```

```
347 resultadoCSV = 'resultadoCSV.csv'
348
349 # Variables de las funciones CSV que tienen que ser globales (global scope)
350 listalista=list()
351 listaLectura=list()
352
353 # Función principal CSV
354 comparacionCSV()
```

Bibliografía

Chakrabarty, A. y col. (jun. de 2016). "Autonomous indoor object tracking with the Parrot AR.Drone". En: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, págs. 25-30. DOI: `10.1109/ICUAS.2016.7502612` (vid. pág. 6).

Diachok, R., R. Dunets y H. Klym (mayo de 2018). "System of detection and scanning bar codes from Raspberry Pi web camera". En: *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, págs. 184-187. DOI: `10.1109/DESSERT.2018.8409124` (vid. pág. 5).

Fuster Baggetto, Alejandro (sep. de 2017). "Navegación autónoma mediante raspberry PI y la plataforma OpenCV". Español. En: (vid. pág. 6).

Habsi, S. Al y col. (dic. de 2015). "Integration of a Vicon camera system for indoor flight of a Parrot AR Drone". En: *2015 10th International Symposium on Mechatronics and its Applications (ISMA)*, págs. 1-6. DOI: `10.1109/ISMA.2015.7373476` (vid. pág. 6).

McGovern, Amy (2017a). https://github.com/amymcgovern/pyparrot (vid. pág. 15).

— (2017b). *pyparrot's documentation*. https://pyparrot.readthedocs.io/en/latest/ (vid. pág. 15).

Python's Hardest Problem (2018). Jeff Knupp (vid. pág. 6).

Santos, N. y col. (jun. de 2017). "Development of a software platform to control squads of unmanned vehicles in real-time". En: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, págs. 1-5. DOI: `10.1109/ICUAS.2017.7991528` (vid. pág. 5).