
# Trabajo de Fin de Grado

# Sistema de posicionamiento para un drone

*Positioning system for a drone*

Elías Rodríguez Martín

La Laguna, 8 de julio de 2015

D. **Jonay Tomás Toledo Carrillo**, con N.I.F. 78698554Y, Profesor Contratado Doctor del Departamento de Ing. Informatica y de Sistemas de la Universidad de La Laguna, como tutor

**C E R T I F I C A (N)**

Que la presente memoria titulada:

*"Sistema de posicionamiento para un drone"*

ha sido realizada bajo su dirección por D. **Elías Rodríguez Martín**,

con N.I.F. 42.194.367-Q.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de julio de 2015

## Agradecimientos

***Especialmente a***

Aquilino Rodríguez Luis

***Y también, por orden alfabético, a***

Ana Yazmina Galván Hernández

Jonay Tomás Toledo Carrillo

José Miguel Rodríguez Gutiérrez

Nereida Pérez Galán

Pilar Martín Medina

;)

***Sin olvidar a todos los familiares y amigos que me han apoyado durante estos años.***

# Licencia

## Resumen

*El presente trabajo trata sobre el desarrollo de un sistema de posicionamiento basado en odometría visual para un drone que vaya a utilizarse en interiores o en lugares en donde no se dispone señal GPS.*

*Para ello se ha usado ROS, un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo.*

*El trabajo se centra en el desarrollo de un módulo para ROS que se encargue de realizar la captura de una secuencia de imágenes del "suelo" y estimar su posición relativa al mismo a partir de la detección de desplazamiento entre las imágenes obtenidas. Dicho módulo ha de ejecutarse en un sistema empotrado con recursos limitados implícitamente por el consumo, volumen y peso máximo admisibles, por lo que la optimización y eficiencia de los algoritmos implicados adquieren enorme importancia.*

*Finalmente se ha demostrado que es posible desarrollar herramientas efectivas de posicionamiento basadas en odomotría visual y sistemas empotrados de recursos limitados.*



**Palabras clave:** drone, robótica, visión artificial, sistemas empotrados, localización.

**Abstract**

*This paper discusses the development of a positioning system based on visual odometry for drones being used at indoors, or wherever GPS signal is not available.*

*ROS has been selected for this system, because it is a framework for the development of robot software that provides the functionality of an operating system in a heterogeneous cluster.*

*The work focuses on the development of the ROS module, that is responsible for capturing a sequence of images of the "floor" and estimating thereafter the relative position, by detection of displacement between those obtained images. This module is meant to be run in an embedded system, which is itself limited by consumption, volume and maximum allowable weight. Therefore, optimizing resources and efficiency of the algorithms involved are important key factors.*

*At the end, the paper shows that it is possible to develop effective positioning tools based on visual odometry, in conjuction with resource limited embedded systems.*

**Keywords:** drone, robotics, artificial vision, embedded systems, localization.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1
# Antecedentes y estado actual

*En este primer capítulo, se verá qué es un 'drone', su historia y los diferentes hitos que se han logrado dentro de este campo desde sus orígenes y la evolución que ha seguido. También se hablará sobre las últimas tendencias en sistemas de posicionamiento de este tipo de máquinas, en exteriores e interiores, con especial atención a estos últimos. Además, se hará una revisión de los entornos de desarrollo más usados en este tipo de sistemas junto con sus características principales.*

## 1.1 ¿Qué es un 'drone'?

Un *drone* o VANT (Vehículo Aéreo No Tripulado) es una aeronave reutilizable que vuela sin tripulación, capaz de mantener un nivel de vuelo controlado y sostenido, y propulsado por uno o más motores eléctricos, de explosión o reacción.

Existen *drones* con múltiples formas, tamaños, configuraciones y características. Originalmente, los VANT eran simplemente aviones tradicionales en los que reemplazaba al equipo humano por una serie de ordenadores conectados a varios actuadores y sensores que realizaban las tareas de los pilotos por control remoto. Con el tiempo, esos sistemas se han ido mejorando progresivamente buscando dar a este tipo de aeronaves más autonomía, hasta llegar así a lo que actualmente se entiende como *drone* o VANT.

Como resultado del desarrollo de nuevos sistemas que permiten a estas naves volar de forma autónoma, hoy en día se fabrican multitud de modelos diseñados desde el principio con la idea de no ser pilotados por personas.

## 1.2 Clasificación de los 'drones'

Los *drones* se pueden clasificar según múltiples criterios, en este caso se tendrán cuenta tres: según uso (distinguiendo especialmente entre militares y civiles), según sus capacidades de vuelo (en términos de altura máxima y autonomía de vuelo), y por último, según su forma y sistema de sustentación principal.

### 1.2.1 Por su uso

Dependiendo de su uso, se distingue entre:

**Militares**

El uso militar ha sido, como en muchos casos, el gran impulsor del desarrollo de este tipo de vehículos, que se emplean para realizar múltiples tareas, como:

- **Blanco:** usados como objetivo en simulaciones de ataque a naves enemigas para probar sistemas de defensa de tierra o aire.
- **Reconocimiento:** se utilizan principalmente para reconocimientos militares, aunque se empiezan a usar en reconocimiento de zonas catastróficas. Este tipo de naves envían información sobre el terreno o el espacio aéreo por el que vuelan.
- **Combate:** se utilizan normalmente para llevar a cabo misiones de alto riesgo. Este tipo de *drones* se denominan "vehículos aéreos de combate no tripulado" (UCAV, por su acrónimo en inglés).

  No se deben confundir con los misiles de crucero pues, si se tiene en cuenta la definición, este tipo de vehículos no pueden ser considerados como *drones* por no ser reutilizables, a pesar de que también son no tripulados y en algunos casos guiados remotamente.

Actualmente, se usan los VANT militares para realizar misiones de reconocimiento y/o ataque. No se puede hablar de que todos los ataques realizados por estas naves sean satisfactorios, pues existen casos en los que se producen daños colaterales o bien fallan en la selección de su objetivo, tal y como sucede con otro tipo de armas. De ahí que no se pueda en general

considerar que la "precisión" sea la ventaja más relevante del uso de *drones* en misiones militares.

**Civiles**

Hasta hace poco tiempo, el uso civil de *drones* estaba muy poco extendido, pero últimamente se ha ido popularizando gracias a las comunidades DIY (Siglas de 'Do It Yourself', "hazlo tu mismo") y al abaratamiento de muchos de los componentes necesarios para su construcción.

Entre los usos civiles mas extendidos se pueden distinguir los siguientes:

- **Logística:** Naves de transporte de carga. Mencionar por ejemplo el proyecto de Amazon (*Amazon Prime Air*), que solo está a pendiente de utilizarse por cuestiones legales. Dentro de no mucho, al menos en EEUU, es posible que al sonar el timbre nos encontremos en la puerta con un *drone* que viene a entregarnos un nuevo *gadget* que compramos hace media hora.
- **Investigación y desarrollo:** Usados para pruebas e investigaciones de nuevos sistemas en desarrollo.
- **Seguridad:** Diseñados para desempeñar labores de lucha contra incendios o seguridad civil, como la vigilancia de oleoductos. Estos vehículos suelen usarse preferentemente para realizar tareas muy peligrosas y/o monótonas para los aviones tripulados.
- **Ocio:** En este campo, su uso es de lo mas variado, desde “cámaras voladoras” para filmar escenas de películas a menor coste, hasta como juguete para entretener a los niños.

### 1.2.2 Dependiendo de la altura y alcance máximos

También pueden clasificarse según su techo y alcance máximo [1]:

- **CIS Lunar:** viaja entre la Luna y la Tierra.
- **ORBITAL:** en órbitas bajas terrestres (Mach 25+).
- **HYPERSONIC alta velocidad, supersónico (Mach 1-5) o hipersónico (Mach 5+):** unos 15 km de altitud o altitud suborbital, alcance de 200 km.
- **HALE (high altitude, long endurance):** sobre 9 km y alcance indeterminado.
- **MALE (medium altitude, long endurance):** hasta 9 km de altitud y un alcance de unos 200 km.
- **Tactical:** unos 5,5 km de altitud, hasta 160 km de alcance.
- **NATO:** unos 3000 metros de altitud y hasta 50 km de alcance.
- **Close:** unos 1500 metros de altitud y hasta 10 km de alcance.
- **Handheld:** unos 600 metros de altitud y unos 2 km de alcance en vuelo.

### 1.2.3 Por su forma y sistema de sustentación principal

- **Ala fija:** parecidos a un avión o avioneta tradicional pero sin cabina, pues no llevan tripulación. Pueden tener uno o mas motores de hélice o propulsión.
- **Multirotor:** inspirados en los helicópteros, solo que en lugar de tener un solo rotor, lo normal es que tengan al menos tres. Los mas comunes son de cuatro rotores.

Figura 1.1: VANT militar de ala fija.

Figura 1.2: VANT multirotor civil.

## 1.3 Historia

A mediados del siglo diecinueve, Austria envió globos no tripulados llenos de bombas para atacar a Venecia. Los *drones* que hoy conocemos comenzaron a desarrollarse al comienzo del siglo veinte, y fueron utilizados originalmente para prácticas de tiro en el entramiento de personal militar. Su desarrollo avanzó durante la Primera Guerra Mundial, cuando la empresa Dayton-Wright Airplane desarrolló un torpedo aéreo sin piloto capaz de caer y explotar en un tiempo preestablecido. [2]

Nikola Tesla describió una flota de vehículos aéreos de combate no tripulados en 1915. [3] El primer intento de un vehículo aéreo no tripulado fue el de A.M. Low's 'Aerial Target' en 1916. [4] El primer RPV *(Remote Piloted Vehicle)* fue desarrollado por Reginald Denny, estrella de cine y entusiasta del aeromodelismo, en 1935. [4] Varios de estos vehículos fueron fabricados durante la carrera tecnológica de la Segunda Guerra Mundial, utilizados para capacitar a los artilleros antiaéreos y para llevar a cabo misiones de ataque. La Alemania nazi produjo y utilizó varios aviones UAV durante el curso de la Segunda Guerra Mundial. Tras la misma aparecieron los primeros modelos equipados con motores a reacción, tales como el GAF Jindivik australiano y el Teledyne Ryan Firebee I en 1951, mientras que empresas como Beechcraft también entraron en juego con su modelo 1001 para la Marina de los Estados Unidos en 1955. [4] Sin embargo, estas naves eran poco más que aviones a control remoto.

En 1959 la Fuerza Aérea de Estados Unidos, preocupada por la pérdida de

pilotos sobre territorio hostil, comenzó a planificar el uso de aviones no tripulados. [5] Dicha planificación se intensificó después de que un U-2 fuera derribado sobre la Unión Soviética en 1960. En pocos días se lanzó un programa secreto de desarrollo de UAV bajo el nombre en clave 'Red Wagon'. [5] Durante el choque de agosto de 1964 en el Golfo de Tonkin entre unidades navales de los Estados Unidos y la Marina de Vietnam del Norte, Estados Unidos comenzó a usar sus UAVs ultra secretos (Ryan modelo 147, Ryan AQM-91 Firefly, Lockheed D-21) en sus primeras misiones de combate en la guerra de Vietnam. [5] Cuando la *Red Chinese* [5] mostró fotografías de los UAV estadounidenses caídos a través de *Wide World Photos*, [5] la respuesta oficial de Estados Unidos fue "sin comentarios".

El Tadiran Mastiff Israeli, que primero voló en 1973, es considerado como el primer UAV de combate moderno, por su sistema de enlace de datos y la transición de vídeo en tiempo real, entre otras innovaciones. [6]

En 1973 el ejército estadounidense confirmó oficialmente que habían estado utilizando UAVs en el sudeste asiático (Vietnam). [5] Más de 5.000 aviadores estadounidenses habían muerto y más de más 1.000 desaparecieron o fueron capturados. La USAF 100th Strategic Reconnaissance Wing había volado aproximadamente 3.435 misiones UAV durante la guerra [5], con un costo de alrededor de 554 vehículos aéreos no tripulados perdidos por varias causas. En palabras del general de la USAF George S. Brown, comandante de la Fuerza Aérea Sistemas de Mando, en 1972, "La única razón por la que necesitamos (UAVs) es que no queremos perder innecesariamente hombres en la cabina del piloto." [5] Más tarde ese mismo año, el general John C. Meyer, comandante en jefe del Comando aéreo Estratégico, declaró: "dejamos que el avión no tripulado haga los vuelos de alto riesgo ... la tasa de pérdida es alta, pero estamos dispuestos a arriesgar más de ellos ... ¡salvan vidas! " [5]

Durante la guerra del Yom Kippur en 1973, las baterías de misiles soviéticos tierra-aire en Egipto y Siria causaron enormes daños a los jets de combate Israelies. Como resultado, Israel desarrolló el primer UAV con vigilancia en tiempo real. [7] [8] [9] Las imágenes de radar y señuelo proporcionadas por estos UAVs ayudaron a Israel a neutralizar por completo las defensas aéreas sirias en el inicio de la Guerra del Líbano de 1982, dando como resultado ningún piloto derribado. [10]

Con la maduración y la miniaturización de las tecnologías aplicables en los años 1980 y 1990, el interés por los UAV creció dentro de los niveles más altos de los militares estadounidenses. En la década de 1990, el Departamento de Defensa de Estados Unidos concedió un contrato a la AAI Corporation, junto con la compañía israelí Malat. La Armada de Estados Unidos compró el UAV Pioneer AAI que fue desarrollado conjuntamente por la AAI y Malat. Muchos de los recién desarrollados UAVs Pioneer se utilizaron en la Guerra del Golfo de 1991. Los UAV fueron vistos como la posibilidad de ofrecer máquinas voladoras mas baratas que podrían ser usadas sin poner en riesgo tripulaciones aéreas. Las primeras generaciones fueron principalmente aviones de vigilancia, pero algunos estaban armados, como el General Atomics MQ-1 Predator, que utilizaba misiles aire-tierra AGM-114 Hellfire.

A partir de 2012, la Fuerza Aérea de los Estados Unidos empleó 7.494 UAV [11] [12] La CIA también ha operado UAVs. [13]

En 2013 se informó de que los UAV están siendo utilizados por al menos 50 países, varios de los cuales desarrollaron sus propias versiones, por ejemplo, Irán, Israel y China. [14]

## 1.4 Sistemas de posicionamiento

Para que un *drone* pueda realizar un vuelo autónomo, se hace imprescindible que este disponga de algún sistema que le permita conocer, si no su posición global, al menos su posición relativa al punto de despegue, así como la posición de cualquier obstáculo con el que pudiera colisionar mientras vuela.

### 1.4.1 Factores a tener en cuanta para elegir el sistema más apropiado

Para implementar un sistema de posicionamiento se pueden usar multitud métodos basados en diferentes tecnologías. A la hora de elegir la tecnología mas apropiada hay que tener en cuenta varios factores como:

- **Coste:** el punto mas importante a tener en cuenta es el coste económico de la construcción y mantenimiento de la nave. Muchas veces el mejor sistema no es el mas económico, por lo que se hace necesario optar por sistemas mas limitados y baratos que cumplan en la mayor medida

posible con las necesidades prioritarias.

- **Características especificas de la zona de vuelo**: evidentemente no es lo mismo volar en campo abierto, en medio de un bosque lleno de árboles, por las calles de una ciudad o en el interior de un centro comercial. Dependiendo de este factor, unas tecnologías pueden sobreponerse con respecto a otras.

- **Capacidad de carga:** en cualquier máquina voladora, el peso es uno de los factores importantes. Cuanto mayor peso deba transportar la nave, mayor será su consumo energético, mayores tendrán que ser su motores y por lo tanto mayor será su coste. Por lo tanto, los sistemas de abordo deberán ser lo mas ligeros posible.

- **Abastecimiento energético:** aunque puede haber una tecnología que se imponga como la mas precisa para un entorno determinado, es posible que no sea posible usarla debido su consumo energético. El suministro eléctrico en un *drone* está condicionado al tipo de acumuladores y/o generadores que se usen, lo que limita la cantidad de energía disponible para los ordenadores y sensores de abordo. Este factor se relaciona con el límite de peso que pueda trasportar el *drone*, pues unos generadores y/o baterías mas potentes añadirán mas peso a la nave.

  La cantidad de energía disponible impone inmediatamente un límite en la capacidad de cálculo de los ordenadores de abordo, así como en el número y tipo de sensores instalados.

### 1.4.2 Sistemas de posicionamiento en exteriores

Cuando se vuela en exteriores, es posible usar sistemas de posicionamiento satelital como GPS para conocer la posición en la que se encuentra la nave, esa posición global unida a un mapa permite al sistema conocer su localización con respecto al planeta y cualquier obstáculo registrado en el mapa. Para localizar y evitar posibles obstáculos que no estén registrados en el mapa, se pueden usar sistemas de apoyo basados en sensores de láser o visuales.

### 1.4.3 Sistemas de posicionamiento en interiores

En interiores la tarea se hace bastante mas difícil, normalmente no se dispone cobertura GPS, por lo que es imprescindible usar un sistema alternativo. Estos sistemas pueden estar basados una o varios de las siguientes tecnologías:

**Balizas**

Originalmente consistía en una serie de transmisores de radio instalados en posiciones fijas y bien conocidas. De este modo, se puede usar un sistema de tiangulación de señal (similar al usado en el GPS) para determinar la distancia del *drone* con respecto a tres o mas balizas y, por lo tanto, con respecto a cualquier punto conocido en el mapa.

Hoy en día se usan también balizas ópticas que son captadas por cámaras e identificadas para inferir la posición. También han evolucionado los tipos de balizas de radio, adaptándose a nuevas tecnologías como WiFi, Bluetooth, Zigbee, RFID, etc.

**Láseres**

Una serie de medidores de distancia láser, informan al *drone* sobre la distancia con respecto a cualquier posible obstáculo que pueda haber a su alrededor. Esa información se puede usar para determinar su posición relativa al punto de despegue o a un punto de vuelo determinado.

**Visual (cámaras)**

Basado en odometría visual, una técnica que permite inferir la posición del *drone* en base a las variaciones en la imagen captada por una o mas cámaras situadas en puntos estratégicos de la nave.

## 1.5 Entornos de desarrollo más utilizados

### 1.5.1 CARMEN.

El *Carnegie Mellon Robot Navigation Toolkit*, es una colección de software código abierto para el control de robots móviles. Está diseñado como un software modular para proporcionar funcionalidades de navegación básica, que incluyen: control de base y sensores, registro, evasión de obstáculos,

localización, planificación de ruta y cartografía.

Tras la aparición ROS, esta colección ha quedado en desuso, pues este último ha atraído la atención de la mayoría de comunidades.

### 1.5.2 ROS.

Es una colección de frameworks para desarrollo de software para robots que proporciona la funcionalidad de un sistema operativo en un clúster informático heterogéneo. ROS provee servicios estándar del sistema operativo como abstracción de hardware, control de dispositivos de bajo nivel, implementación de las funciones utilizadas normalmente, paso de mensajes entre procesos y gestión de paquetes.

Actualmente es el sistema mas usado para programar robots, sean del tipo que sean.

### 1.5.3 Entornos personalizados.

Esta es la vertiente alternativa, usada principalmente por empresas privadas especializadas en el sector. Uno de los ejemplos mas representativos es PARROT.

En este caso, usan un ordenador de abordo basado en ARM, equipado con todos los sensores necesarios, así como una serie de co-procesadores encargados de realizar muchos de los cálculos por "hardware". Sobre dicho ordenador, se ejecuta un sistema GNU/Linux que tiene instalado todo el software privativo encargado de realizar todas las tareas necesarias.

# Capítulo 2
# Objetivos

*En este capítulo se enumerarán los objetivos del proyecto, así como las herramientas disponibles para su desarrollo, cuales de ellas han sido usadas y de que forma.*

## 2.1 Objetivo general

Este Trabajo de Final de Grado se centra en el diseño de un sistema de posicionamiento basado en odometría visual para su uso por *drones* que vuelen en recintos cerrados, con el objetivo de ser usado para mantener a un cuadricóptero volando lo más "quieto" y estable posible, y corregir en tiempo real cualquier desplazamiento o rotación producida por inestabilidades inherentes al sistema de sustentación.

## 2.2 Premisas

El sistema de posicionamiento ha de basarse en el uso de los siguientes materiales y métodos:

- *drone* cuadricóptero basado en *Arducopter*.
- Cámara orientada hacia abajo, captando imágenes de superficie.
- Sensor ultrasónico de altitud.
- Sensores de inclinación con respecto a los ejes X e Y.
- Ordenador de placa reducida basado en ARM.
- Técnica de odomoetría visual para el posicionamiento.

El ordenador de abordo deberá capturar en tiempo real las imágenes del suelo que le entrega la cámara y usar esa secuencia para inferir si se ha producido algún desplazamiento con respecto la superficie, así como su ángulo de rotación.

La opción de utilizar la odometría visual, se justifica por dos razones principales:

1. Al tratarse de vuelo en recintos cerrados no se dispone de sistemas de localización satelitales, como el GPS.
2. El coste de los sensores necesarios para realizar la odometría visual necesaria es muy bajo. Por lo que es muy interesante estudiar hasta que punto se puede llegar únicamente con esas herramientas.

## 2.3 Objetivos específicos

Por la magnitud del proyecto, los objetivos se plantean como primarios y secundarios, entendiendo que los primarios son aquellos imprescindibles para poder considerar el desarrollo del trabajo como satisfactorio.

En cuanto los objetivos secundarios, están enfocados a dejar abiertas las puertas a futuras líneas de investigación o TFGs que pudieran llevar el presente proyecto a un punto apto para su implementación y posible explotación, o bien, concluir su no factibilidad, ya sea por no ser la solución adecuada al problema, por cuestiones económicas o por la aparición o existencia de otras tecnologías que cubran mejor las necesidades expuestas con un coste similar.

### 2.3.1 Objetivos primarios

Los objetivos primarios son todos los hitos a los que se ha de llegar para que el *drone* sea capaz de conocer su posición actual con respecto a una posición anterior dada. Dichos hitos se reducen a diseñar e implementar algoritmos capaces de realizar las siguientes operaciones:

- Inducir el desplazamiento y la rotación del *drone* con respecto a la superficie, a partir de la secuencia de imágenes de la cámara inferior.
- Capturar la información de inclinación con respecto al horizonte,

proveída por el sistema de navegación integrado en el propio *drone*, y usarla para inferir el desplazamiento real del *drone* con respecto a la superficie y, por lo tanto, con respecto a cualquier posición anterior.

### 2.3.2 Objetivos secundarios

Estos objetivos se centran en el diseño del sistema de piloto automático que permitirá al *drone* mantenerse lo mas estable posible en un lugar fijo del espacio, apoyándose en los datos de posicionamiento obtenidos por el sistema anteriormente mencionado.

Se pretende presentar un modelo teórico los mas detallado posible, para permitir el desarrollo en la práctica del mismo en el futuro, como por ejemplo en alguna línea de investigación derivada de este TFG.

# Capítulo 3
# Materiales y recursos

*En este capítulo se expondrán con detalle los diferentes materiales y herramientas de los que se dispone para la realización del trabajo.*

## 3.1 Ardupilot Mega

El ArduPilot Mega es un sistema de piloto automático de código abierto completo basado en Arduino Mega [15]. Se ha convertido en la tecnología más vendida. Ganó el prestigioso concurso '2012 Outback Challenge UAV'. La versión APM2.5 + viene lista para usar, sin necesidad de ensamblaje. Permite al usuario convertir cualquier tipo de vehículo, volador o no, (incluso coches y barcos) en un vehículo completamente autónomo; capaz de realizar misiones programadas con puntos de referencia GPS.

## 3.2 ArduCopter

El sistema ArduCopter cuenta con capacidad de vuelo totalmente autónomo basado en 'way-point', planificación de la misión y telemetría en tiempo real a través de una estación de control en tierra. Es una herramienta que permite controlar *drones* tipo helicóptero o multirotor con bastante facilidad. Se ejecuta en el piloto automático ArduPilot o Pixhawk [16]. Si se dispone de GPS, este APM/pixhawk proporciona una solución completa UAV que se diferencia de los multirotores tradicionales, que a menudo sólo soportan control remoto.

### 3.2.1 Principales Características

- Control automático de gran calidad del nivel y la altitud y vuelo en

línea recta. También dispone del modo "vuelo simple", que hace que el manejo sea muy cómodo y sencillo. El sistema es capaz de responder a las órdenes que el usuario le da, no de forma directa sino mediante la interpretación de lo que realmente el usuario desea hacer. De este modo, el usuario se limita a indicarle al *drone* hacia donde desea que se desplace, el ArduCopter se encargará de modificar los valores necesarios para lograr ese desplazamiento mantenido la estabilidad.

- No se requiere programación. Se tiene que instalar una utilidad de escritorio fácil de usar para cargar el software con un solo clic, y configurar ArduCopter con indicadores visuales rápidos, planificador de la misión y estación terrestre completa.
- Puntos GPS ilimitados. Al apuntar y hacer clic en los puntos del mapa en el planificador de la misión, el ArduCopter volará por sí mismo hacia ellos. No hay límites de distancia.
- Con simplemente mover un interruptor, el *drone* mantendrá su posición utilizando sus sensores GPS y de altitud.
- Es posible indicarle las coordenadas de la base, de modo que se le puede ordenar en cualquier momento "volver a casa".
- Comunicación completa durante el vuelo, incluso es posible cambiar los puntos de la planificación de vuelo mientras el *drone* está volando.
- Despegue y aterrizaje automático. Al pulsar un interruptor ArduCopter ejecuta su misión de forma totalmente autónoma, volviendo a casa y aterrizando por sí mismo al finalizar.
- Multi-plataforma. Compatible con Windows, Mac y Linux. Se puede usar la utilidad de configuración Misión Planner gráfica en Windows (funciona bajo Parallels en Mac) o utilizar una interfaz de línea de comandos en cualquier otro sistema operativo. Una vez que ArduCopter está configurado, se puede utilizar con una selección de tres estaciones de tierra, incluyendo QGroundcontrol, que se ejecuta de forma nativa en Windows, Mac y Linux.
- Compatibilidad con estándares de la industria robótica líder, tales como el sistema ROS, de Willow Garages, y el protocolo de comunicaciones

MAVLink.

### 3.2.2 El protocolo de comunicaciones MAVLink

Se trata del "Micro Air Vehicle Communication Protocol", una librería muy ligera para micro vehículos aéreos, que nos permite recibir los datos del ArduCopter y enviarle los comandos. [17]

Esta librería puede empaquetar estructuras C a través de los canales serie con mucha eficiencia y enviar dichos paquetes a la estación de control en tierra. Ha sido muy probada en las plataformas PX4, PIXHAWK, APM y Parrot AR, sirviendo como núcleo para las comunicaciones MCU/IMU y también entre procesos Linux y el enlace de comunicaciones con tierra. El código fué liberado por primera vez en 2009 bajo licencia LGPL por Lorenz Meier. [18]

### 3.2.3 ¿Qué puede controlar ArduCopter?

ArduCopter es capaz de controlar todas las principales células de aeronaves de ala móvil, incluyendo helicópteros tradicionales, Tricopter (3 / Y6), Quadrotor (X / +), Hexa (X / +), y Octa (X / + / V). Todo lo que se necesita hacer es cambiar el firmware y algunos parámetros en el tablero Mega ArduPilot.

## 3.3 Raspberry Pi

Raspberry Pi es la denominación de una serie de ordenadores de una sola placa del tamaño de una tarjeta de crédito, desarrollada en Gran Bretaña por la Raspberry Pi Foundation con la intención de promover la enseñanza básica de informática en las escuelas. [19] [20] [21]. Es un dispositivo que permite a personas de todas las edades explorar la computación, y aprender a programar en lenguajes como Scratch y Python. Es capaz de hacer todo lo que se espera que pueda hacer de un ordenador de escritorio, desde navegar por Internet y reproducir vídeo de alta definición, trabajar hojas de cálculo, procesadores de texto, jugar juegos o controlar robots.

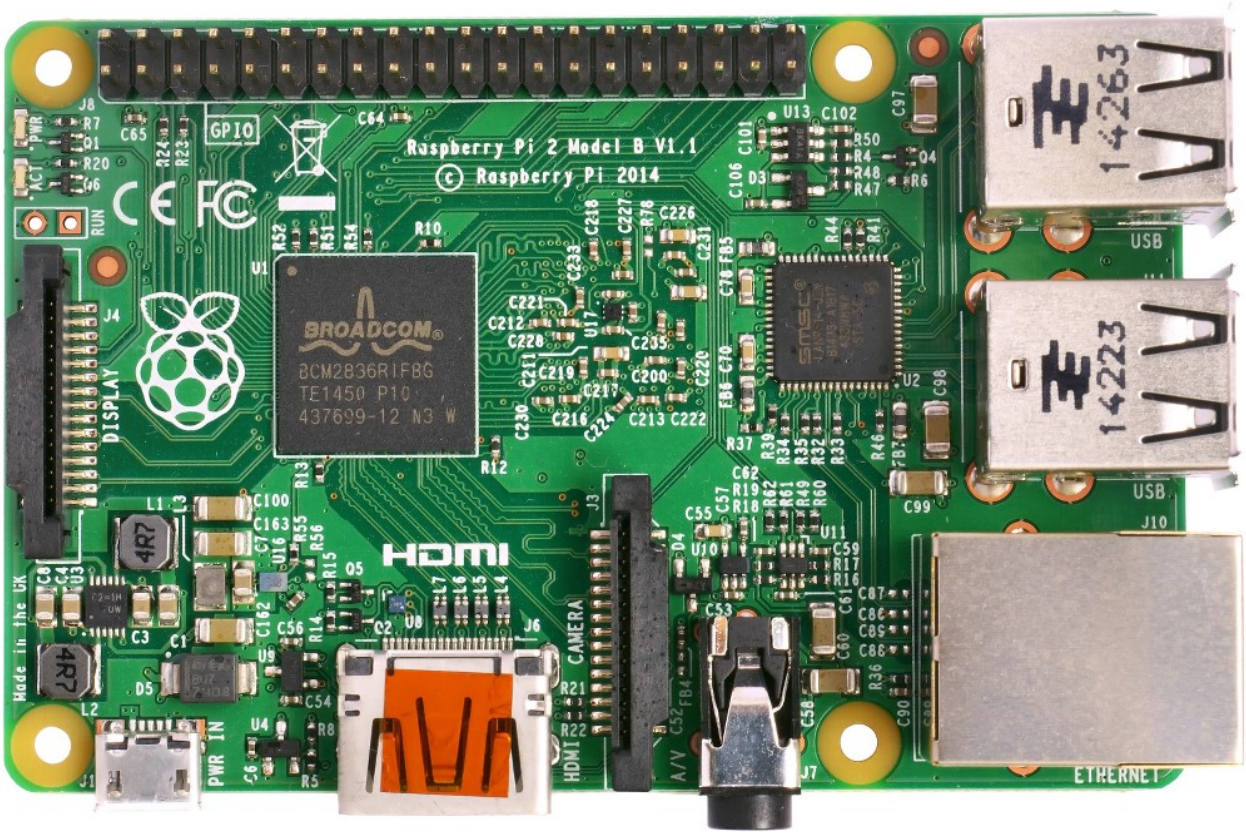


Figura 3.1: Fotografía de la Raspberry Pi 2 B utilizada en este TFG.

### 3.3.1 Historia

En 2006, los primeros prototipos de Raspberry Pi se basaban en el microcontrolador Atmel ATmega644. Sus esquemas y diseños de la placa fueron liberados públicamente [22]. La fundación Eben Upton reunió a un grupo de profesores, académicos y entusiastas de la informática para diseñar una computadora para inspirar a los niños [23]. La placa está inspirada en la Acorn BBC Micro de 1981 [24] [25]. El modelo A, el modelo B y el modelo B+ son referencias a los modelos originales de la BBC Micro, desarrollado por Acorn Computers [26]. La primera versión del prototipo ARM se montó en un paquete del mismo tamaño que un *pendrive* USB [25], tenía un puerto USB en un extremo y un puerto HDMI en el otro.

El objetivo de la fundación era ofrecer dos versiones, a 25 USD y 35 USD respectivamente. Empezaron a aceptar pedidos para el modelo B en febrero de 2012 [27] y para el modelo A en febrero de 2013 [28] y, por último para el A+, de costo aún más bajo (20 USD), en noviembre de 2014 [29].

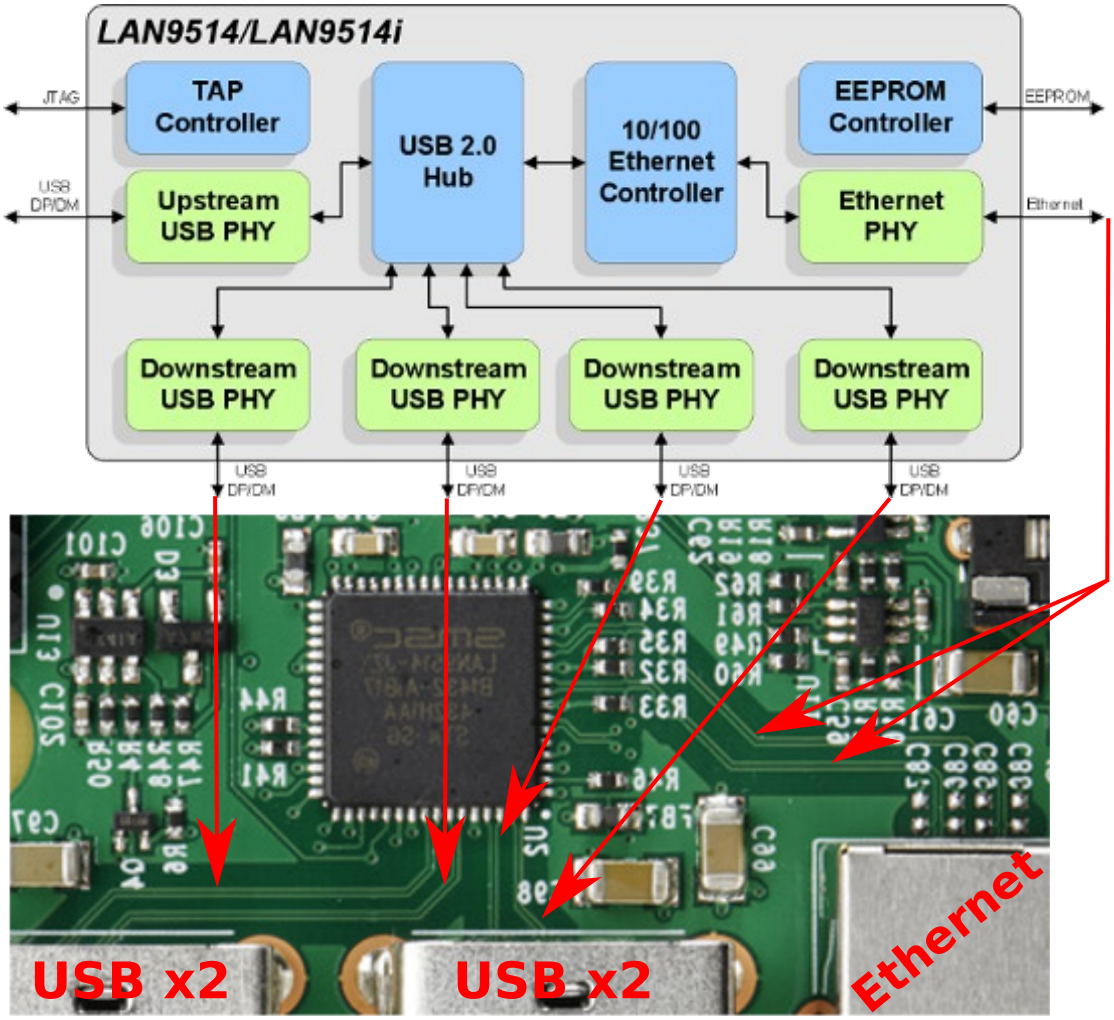


Figura 3.2: Diagrama de la configuración del Shitch en la Raspberry Pi.

### 3.3.2 Hardware

La última versión se basa en el SoC Broadcom BCM2836 [30], incluye un procesador ARM Cortex-A7 de cuatro núcleos a 900 Mhz y una GPU VideoCore IV [31]. Dispone de 16KB de caché de nivel 1 y 128 KB de caché de nivel 2. La caché de nivel 2 es usada principalmente por la GPU. Aunque en las primeras versiones, la memoria RAM estaba integrada en el mismo chip, en la última versión ésta se ha situado en un chip dedicado y separado que le aporta 1GB de capacidad.

El SoC dispone de un único puerto USB 2.0 que está conectado a un *hub* que "lo convierte" en una inerfaz de red 10/100 y cuatro puertos USB 2.0. Como resultado, la velocidad máxima de todo el conjunto de puertos está limitada a la velocidad del puerto USB 2.0 interno. Por lo que si la red está

trabajando a pleno rendimiento, los USB solo dispondrán de unos 300 Mb/s para realizar sus transferencias.

### 3.3.3 Software

El Raspberry Pi usa mayoritariamente sistemas operativos basados en el núcleo Linux. Raspbian, una distribución derivada de Debian que está optimizada para el hardware de Raspberry Pi, se lanzó durante julio de 2012 y es la distribución recomendada por la fundación para iniciarse. [32]

Para sorpresa de muchos, recientemente Microsoft ha decidido dar soporte para poder instalar Windows 10 en la última versión (v. 2) de Raspberry Pi. En la actualidad, estos son los principales sistemas operativos disponibles para la Raspberry Pi:

| **Basados en Linux** | | **Otros S.O.** |
|---|---|---|
| Android [33] | Pidora (versión Fedora Remix optimizada) [37] | AROS |
| Arch Linux ARM | QtonPi (distribución con un framework de aplicaciones multiplataforma basado en Qt framework) | RISC OS [40] |
| Debian Whezzy Soft-Float (versión de Debian sin soporte para coma flotante por hardware) | Raspbian, [32] versión de Debian Wheezy para ARMv6, (con soporte para coma flotante por hardware) | FreeBSD [41] |
| Firefox OS | Slackware ARM, también conocida como ARMedslack | NetBSD107 [42] |
| Gentoo Linux [34] | Plan 9 from Bell Labs [38] [39] | Windows 10 |
| Google Chromium OS | | |
| Kali Linux | | |
| Open webOS [35] | | |
| PiBang Linux (distribución Linux derivada de Raspbian con diferente escritorio y aplicaciones) [36] | | |

**Tabla 3.1: Sistemas Operativos disponibles para Raspberry Pi**

## 3.4 Raspbian

Según Wikipedia [43]:

> "Raspbian es una distribución del sistema operativo GNU/Linux y por lo tanto libre basado en Debian Wheezy (Debian 7.0) para la placa computadora (SBC) Raspberry Pi, orientado a la enseñanza de informática. El lanzamiento inicial fue en junio de 2012. [44]
>
> Técnicamente el sistema operativo es un port no oficial de Debian Wheezy armhf para el procesador (CPU) de Raspberry Pi, con soporte optimizado para cálculos en coma flotante por hardware, lo que permite dar más rendimiento en según que casos. El port fue necesario al no haber versión Debian Wheezy armhf para la CPU ARMv6 que contiene el Raspberry PI. [45]
>
> La distribución usa LXDE como escritorio y Midori como navegador web. Además contiene herramientas de desarrollo como IDLE para el lenguaje de programación Python o Scratch, y diferentes ejemplos de juegos usando los módulos Pygame.
>
> Destaca también el menú 'raspi-config' que permite configurar el sistema operativo sin tener que modificar archivos de configuración manualmente. Entre sus funciones, permite expandir la partición root para que ocupe toda la tarjeta de memoria, configurar el teclado, aplicar *overclock*, etc. [46]
>
> El 17 de diciembre de 2012, junto a la versión *2012-12-16-wheezy-raspbian* de Raspbian, se lanzó la tienda de aplicaciones "Pi Store", que en el momento de salida incluía desde aplicaciones como LibreOffice o Asterisk a juegos como Freeciv o OpenTTD. En esta plataforma se puede poner a disposición de todos los usuarios de Raspbian, mediante moderación y posterior lanzamiento, contenidos gratuitos o de pago, como archivos binarios, código Python,

imágenes, audio o vídeo. Además se quiere incluir documentación acerca del Raspberry Pi como la revista MagPi y tutoriales de proyectos. [47]"

## 3.5 ROS


es una colección de frameworks para desarrollo de software para robots que proporciona la funcionalidad de un sistema operativo en un clúster informático heterogéneo. ROS provee servicios estándar del sistema operativo como abstracción de hardware, control de dispositivos de bajo nivel, implementación de las funciones utilizadas normalmente, paso de mensajes entre procesos y gestión de paquetes. El conjunto de procesos en ejecución basados en ROS son representados en un grafo donde el procesamiento se realiza en los nodos, que pueden recibir, enviar y múltiplexar mensajes de sensores, control, estado, planificación y actuadores, entre otros. A pesar de la importancia de la reactividad y baja latencia en los sistemas de control de un robot, ROS en sí mismo no es un sistema operativo en tiempo real, aunque es posible integrar ROS con código en tiempo real [48]. El ecosistema de software de ROS puede dividirse en tres grupos:

- Lenguajes y herramientas independientes del lenguaje utilizadas para la creación y distribución de software basada en ROS.
- Implementaciones de biblioteca de cliente ROS como *roscpp, rospy* y *roslisp.*
- Paquetes que contienen código relacionados con la aplicación que utiliza una o más bibliotecas de cliente ROS.

Las herramientas independientes del lenguaje y las bibliotecas principales de cliente (C++, Python y LISP) son liberadas bajo los términos de la licencia BSD y como tal es software de código abierto y gratis tanto para usos comerciales como para investigación. La mayoría de los otros paquetes se licencian bajo una variedad de licencias de código abierto. Dichos paquetes implementan funcionalidades comúnmente usadas y aplicaciones tales como controladores de hardware, modelos de robot, tipos de datos, planificación,

percepción, localización simultánea y mapeo, herramientas de simulación y otros algoritmos.

Las principales bibliotecas de cliente ROS (C ++, Python, LISP) están orientadas hacia un sistema Unix, debido principalmente a sus dependencias al software de código abierto. Para estas bibliotecas de cliente, Ubuntu Linux aparece como "compatible", mientras que otras variantes como Fedora Linux, Mac OS X y Microsoft Windows son designados como "experimental", apoyados por la comunidad. [49] Sin embargo, la librería de cliente Java ROS nativa 'rosjava', no comparte estas limitaciones y ha permitido que el software basado en ROS pueda ser escrito para el sistema operativo Android. [50] Con 'rosjava' también se ha permitido que ROS se integre en las herramientas de MATLAB con apoyo oficial, que se pueden utilizar en Linux, Mac OS X y Microsoft Windows. [51] También se ha desarrollado una librería cliente JavaScript 'roslibjs', que permite la integración de software en un sistema ROS a través de cualquier navegador compatible con los estándares.

### 3.5.1 Historia

ROS fue desarrollado originalmente en 2007 bajo el nombre "switchyard by the Stanford Artificial Intelligence Laboratory in support of the Stanford AI Robot STAIR (STanford AI Robot) project." [52] [53]

**Descripción del STAIR en su página web:**

> "Nuestra sencilla plataforma para robots integrará métodos procedentes de todas las áreas de la IA, incluyendo el aprendizaje de la máquina, la visión, la navegación, la planificación, el razonamiento y el habla / procesamiento del lenguaje natural. Esto está en claro contraste con la tendencia de los últimos 30 años de fragementar el trabajo de los sub-campos de la IA, y será un vehículo para conducir la investigación hacia la verdadera integración de la AI."

Desde 2008 hasta 2013, el desarrollo se llevó a cabo principalmente en el Willow Garage, un instituto/incubadora de investigación de la robótica. Durante ese tiempo, los investigadores de más de veinte instituciones colaboraron con ingenieros del Willow Garage en un modelo de desarrollo federado. [54] [55]

En febrero de 2013, la administración de ROS migró a la Open Source Robotics Foundation. [56] En agosto de 2013, un blog [57] anunció que Willow Garage sería absorbida por otra empresa iniciada por su fundador: Suitable Technologies. El soporte para el PR2 (Personal Robot 2) creado por Willow Garage también fué asumido más tarde por Clearpath Robotics. [58]

### 3.5.2 Infraestructura de comunicaciones

En el nivel más bajo, ROS ofrece una interfaz de paso de mensajes que proporciona comunicación entre procesos que comúnmente se conoce como un *middleware.* El *middleware* ROS ofrece las siguientes facilidades:

- **Paso de mensajes** [59]

  Un sistema de comunicación es a menudo una de las primeras necesidades que surgen en la implementación de una nueva aplicación de robot. El sistema de mensajería integrado en ROS ha sido exhaustivamente probado, y ahorra tiempo mediante la gestión de los detalles de la comunicación entre nodos distribuidos a través del mecanismo de publicación/suscripción anónima de mensajes. Otra ventaja de utilizar un mensaje que pasa por el sistema es que obliga a implementar interfaces claras entre los nodos, mejorando así la encapsulación y facilitando la reutilización de código. La estructura de estas interfaces de mensajes se define en el "mensaje IDL" [60] (Interface Description Language).

- **Grabación y reproducción de mensajes** [61]

  Debido a que el sistema de publicación/suscripción es anónimo y asincrónico, los datos pueden ser capturados y reproducidos fácilmente sin cambios en el código. Si, por ejemplo, una tarea A lee datos de un sensor, y una tarea B está procesando los datos producidos por A, ROS hace que sea fácil capturar los datos publicados por la tarea A en un archivo y, a continuación, volver a publicar los datos almacenados en un momento posterior. La abstracción del paso de mensajes permite que la tarea B sea agnóstica con respecto a la fuente de los datos, que podría ser de la tarea A o del archivo de registro. Este es un patrón de diseño de gran alcance, que puede reducir significativamente el esfuerzo de desarrollo y promover la flexibilidad y modularidad en el sistema.

- **Petición/respuesta de llamadas a procedimiento remoto** [62]

  La naturaleza asincrónica de publicación/suscripción de mensajería funciona para muchas necesidades de comunicación en robótica, pero a veces se requieren interacciones sincrónicas de petición/respuesta entre los procesos. El *middleware* ROS proporciona esta capacidad usando los servicios. Al igual que los mensajes, los datos que se envían entre los procesos en una llamada de servicio se definen con el mismo y sencillo "mensaje IDL" [60].

- **Sistema de parámetros distribuidos** [63]

  El *middleware* de ROS también proporciona una forma para que las tareas de compartan información de configuración a través de un almacén global clave-valor. Este sistema le permite modificar fácilmente los parámetros de una tarea, e incluso permite que una tarea pueda cambiar la configuración de otras.

### 3.5.3 Características específicas para robots

Además de los componentes centrales del *middleware*, ROS proporciona librerías y herramientas especificas que permiten poner un robot en funcionamiento rápidamente. Éstas son sólo algunas de las capacidades del robot específico que ROS proporciona:

- **Definiciones de mensajes estándar** [64]

  Años de discusión y desarrollo de la Comunidad han dado lugar a un conjunto de formatos de mensaje estándar que cubren la mayor parte de los casos de uso común en la robótica. Entre otras, hay definiciones de mensaje para conceptos geométricos (poses, transformaciones y vectores), para sensores (cámaras, IMU y láser), y para los datos de navegación (odometría, caminos y mapas). Usándolas, las aplicaciones pueden interoperar sin problemas con el resto del ecosistema ROS.

- **Geometría del robot** [65]

  Un reto común en muchos proyectos de robótica es mantener la pista de donde están las diferentes partes del robot con respecto a las otras. Por ejemplo, si se quieren combinar datos de una cámara con datos de un

láser, se necesita saber donde está cada sensor, con respecto a un punto de referencia común. Este problema es especialmente importante para robots humanoides con muchas partes móviles. ROS aborda este problema con la librería *tf* (transformar) [66].

La librería *tf* [66] se ha utilizado para administrar y trasnformar los datos de coordenadas para robots con más de un centenar de grados de libertad y con tasas de actualización de cientos de hercios. La librería *tf* [66] permite definir transformaciones estáticas, tales como una cámara que está fijada a una base móvil y transformaciones dinámicas, tales como un conjunto en un brazo robótico. Puede transformar datos de los sensores entre cualquier par de sistemas de referencia. La librería *tf* tiene en cuenta que los productores y consumidores de esta información pueden estar distribuidos en toda la red, y el hecho de que la información se actualiza a diferentes velocidades.

- **Lenguaje de descripción del robot** [67]

  Otro problema común que ROS resuelve es cómo describir el robot de forma legible por la máquina. ROS proporciona un conjunto de herramientas para describir y modelar el robot, de forma que pueda ser entendido por el resto de su sistema, incluyendo *tf* [66], *robot_state_publisher* [68] y *rviz* [69]. El formato para la descripción del robot en ROS es URDF (Unified Robot Descripción Format)[70], que consiste en un documento XML en el que se describen las propiedades físicas del robot, la longitud de las extremidades y tamaños de ruedas para las ubicaciones de los sensores y la apariencia visual de cada parte del robot.

  Una vez definido de esta manera, el robot se puede utilizar fácilmente con la biblioteca *tf* [66], y se puede renderizar en tres dimensiones para obtener las visualizaciones que se usan con los simuladores y los planificadores de movimiento.

- **Llamadas a procedimiento remoto apropiable** [71]

  Mientras que los *topics* (publicación anónima/suscripción) y los *services* (llamadas a procedimiento remoto) cubren la mayoría de los casos de uso de la comunicación en la robótica, a veces se necesita para iniciar

un comportamiento de búsqueda de metas, monitorizar su progreso y recibir una notificación cuando se ha completado. ROS proporciona *actions* para este propósito. Las *actions* son como los *services*, excepto que pueden reportar los avances antes de devolver la respuesta final, y se pueden interrumpir por el emisor de la llamada. Así, por ejemplo, se puede instruir al robot para desplazarse a algún lugar, monitorizar su progreso en su intento de llegar allí, detenerlo o redirigirlo a lo largo del camino, y ser informado de cuando ha tenido éxito (o fracaso).

- **Diagnóstico** [72]

  ROS proporciona una forma estándar para producir, recopilar y agregar diagnósticos [72] acerca del robot de forma que, a simple vista, se puede ver rápidamente el estado del robot y determinar la forma de abordar los problemas que se presenten.

- **Estimación de Pose** [73]**, Localización** [74] **y Navegación** [75]

  Hay paquetes de ROS que resuelven problemas básicos de robótica como plantear la estimación de la posición, la localización en un mapa, la construcción de un mapa, e incluso la navegación móvil.

### 3.5.4 Herramientas

Uno de los rasgos más característicos de ROS es el potente conjunto de herramientas de desarrollo. Estas herramientas apoyan la introspección, la depuración, el trazado, y la visualización del estado del sistema que se está desarrollando. El mecanismo subyacente de publicación/suscripción permite la introspección espontánea de los datos que fluyen a través del sistema, por lo que es fácil comprender y depurar los problemas que se produzcan. Las herramientas de ROS se aprovechan de esta capacidad de introspección a través de una extensa colección de utilidades de línea de comandos y gráficas que simplifican el desarrollo y la depuración.

#### Herramientas de línea de comandos

ROS se puede utilizar al 100% sin una GUI. Todas las herramientas de funcionalidad del núcleo y de introspección, son accesibles por alguna de las más de 45 herramientas de línea de comandos. Hay comandos para lanzar grupos de nodos, *topics* de instrospección, *services* y *actions*, datos de

grabación y reproducción, y una serie de otras situaciones. Si se prefiere utilizar las herramientas gráficas, 'rviz' [69] y 'rqt' [76] proporcionan una funcionalidad similar, y ampliada.

- **rviz:** Esta es tal vez la herramienta de propósito general más conocida en ROS, facilita la visualización tridimensional de muchos tipos de datos de sensor y de cualquier robot descrito por URDF [70]. Puede visualizar muchos de los tipos de mensajes comunes previstos en ROS, como escáneres láser, nubes de puntos tridimensionales, y las imágenes de la cámara. También utiliza la información de la biblioteca *tf* para mostrar todos los datos de los sensores en un sistema de coordenadas común a elegir, junto con una representación tridimensional del robot. La visualización de todos los datos en la misma aplicación no sólo queda gráficamente impresionante, sino que también permite ver rápidamente lo que "ve" el robot, e identificar problemas como desajustes de sensores o inexactitudes en el modelo de robot.

- **rqt:** es un marco basado en Qt para el desarrollo de interfaces gráficas para robots [76]. Permite crear interfaces personalizadas componiendo y configurando la extensa biblioteca de una función de plugins RQT en pestañas, pantalla dividida y otros diseños. También permite introducir nuevos componentes de la interfaz escribiendo nuevos plugins RQT. Algunos de los plugins disponibles mas destacables son:
  - **'rqt_graph'** proporciona la introspección y la visualización de un sistema de ROS en directo, que muestran los nodos y las conexiones entre ellos, lo que permite fácilmente depurar y entender su sistema de funcionamiento y cómo se estructura.
  - **'rqt_plot'** permite monitorear codificadores, tensiones, o cualquier cosa que se puede representar como un número que varía con el tiempo. El plugin 'rqt_plot' permite elegir el backend de trazado (por ejemplo, 'matplotlib', 'Qwt', 'pyqtgraph') que mejor se adapte a sus necesidades.
  - **'rqt_topic' y 'rqt_publisher'** se pueden usar para el seguimiento y el uso de *topics*. El primero permite supervisar cualquier número de *topics* que se publiquen dentro del sistema. El último le permite

publicar sus propios mensajes a cualquier *topic*, lo que facilita la experimentación ad hoc con el sistema.

- **'rqt_bag':** Para el registro y reproducción de datos, ROS utiliza unos ficheros que llama *bags*. Las *bags* se pueden crear y acceder de forma gráfica. Este plugin puede puede grabar los datos a las *bags*, reproducirlos, y visualizar el contenido de una *bag*, incluyendo la visualización de imágenes y de trazado de los valores numéricos en el tiempo.

### 3.5.5 Versiones publicadas

Las diferentes versiones de ROS puenden no ser compatibles con otros lanzamientos, y son referidas a menudo por su nombre en clave en lugar de por su número de versión. Los grandes lanzamientos hasta el momento son:

| | |
|---|---|
| 22/01/2010 - **ROS 1.0** | 23/04/2012 - **Fuerte** |
| 01/03/2010 - **Box Turtle** | 31/12/2012 - **Groovy Galapagos** |
| 03/08/2010 - **C Turtle** | 04/09/2013 - **Hydro Medusa** |
| 02/03/2011 - **Diamondback** | 22/07/2014 - **Indigo Igloo** (estable) |
| 30/08/2011 - **Electric Emys** | 23/05/2015 - **Jade Turtle** (en desarrollo) |

**Tabla 3.2: Versiones de ROS liberadas hasta la fecha**

## 3.6 Cámara

Para capturar las imágenes de superficie, se ha pensado en dos opciones, por un lado se puede usar la cámara de Play Station 3 (Play Station Eye), o una cámara específica para Raspberry Pi, puesto que esta cuenta con un puerto especifico para conectarle una cámara.

### 3.6.1 Especificaciones de la Play Station Eye

La PlayStation Eye es capaz de capturar vídeo estándar con velocidades de de 60 fps a una resolución de 640×480 píxeles y 120 fps a 320×240 px, [77] llegando a poder trabajar a mayor frecuencia de refresco, hasta 320x240 a 187 fps o 640x480 a 75 fps, con aplicaciones específicas (Freetrack y Linuxtrack).

Gracias a la colaboración de Sony con OmniVision Technologies, esta cámara dispone de un sensor especialmente diseñado para operar con muy poca cantidad de luz. [78] Sony establece que el PlayStation Eye puede producir "video de calidad razonable" bajo la iluminación proporcionada por un aparato de televisión. [79]

La cámara cuenta con una lente que permite cambiar su campo de visión entre dos ajustes fijos. Girando el cuerpo del objetivo, la PlayStation Eye se puede establecer un campo de visión (punto rojo) de 56° [79], óptimo para su uso en aplicaciones de chat, o un campo de 75° (punto azul), pensado para su uso con juegos y aplicaciones interactivas. [77]

La PlayStation Eye es capaz de dar salida de vídeo a la consola sin comprimir, [77] por lo que "no hay artefactos de compresión"; [79] o con la opción de compresión JPEG. [77] Con respecto a la profundidad de color, alcanza los 8 bits por píxel.

### 3.6.2 Especificaciones de la cámara de Raspberry Pi

El módulo de la cámara Raspberry Pi es un *add-on* de diseño personalizado para Raspberry Pi. Se conecta a esta a través de uno de los dos pequeños buses de la parte superior de la placa mediante un cable plano corto (la otra toma es para conectar un LCD). Esta interfaz utiliza la interfaz dedicada CSI, que fue diseñada especialmente para la conexión a las cámaras. El bus CSI se comunica directamente con el procesador BCM2835/BCM2836 y es capaz trabajar con velocidades de datos muy altas, llevando exclusivamente datos de píxeles.

El sensor tiene una resolución nativa de 5 megapíxeles, y cuenta con una lente de foco fijo. En cuanto a la captura de fotografías, la cámara es capaz de entregar imágenes de hasta 2592x1944 píxeles y cuando se trata de vídeo puede trabajar en 1080p a 30 fps, 720p a 60 fps y 640x480p 60/90 fps vídeo.

### 3.6.3 Cámara elegida

De las dos opciones, se ha decidido usar la cámara de la "Play Station 3" debido a que sus caracteristicas son más homogéneas y mejor conocidas. Aparte de que tiene una tasa de refresco elevada, muy útil para el propósito que se le desea dar, y que no requiere del uso de ninguna librería ni *driver* extra para su funcionamiento.

## 3.7 Librerías

Aparte de todas las funcionalidades proveídas directamente por ROS, es necesario el uso de algunas librerias alternativas como OpenCV y GSL para realizar la captura y el procesamiento de las imágenes.

### 3.7.1 OpenCV

OpenCV (Open Source Computer Vision) es una libreria de funciones de programación orientadas principalmente a la visión por ordenador en tiempo real, desarrollada por el centro de investigación de Intel Rusia en Nizhny Novgorod, ahora apoyada por Willow Garage y Itseez. [80] Es gratuita para uso bajo la licencia BSD de código abierto. La librería es multiplataforma. Se centra principalmente en el procesamiento de imágenes en tiempo real. Además es capaz de aprovechar optimizaciones nativas de procesadores como Intel para incrementar su rendimiento al máximo.

Esta librería venia "integrada" en ROS hasta la versión Hydro Medusa, es decir, que no era necesaria su instalación por separado. A partir de Indigo Igloo esto ha cambiado y si que es necesario instalarla en el sistema para poder ser usada, lo cual es una ventaja más que un inconveniente.

### 3.7.2 GNU Scientific Library

GNU Scientific Library (GSL) es una biblioteca numérica para C y C ++ que ofrece una amplia gama de rutinas matemáticas tales como generadores de números aleatorios, funciones especiales y mínimos cuadrados montaje. Hay más de 1.000 funciones en total con una extensa serie de pruebas. De todas ellas, en el desarrollo de este trabajo se usan las relativas a la minimización por mínimos cuadrados.

GSL es software libre bajo la Licencia Pública General de GNU.

# Capítulo 4
# Metodología

*En este capítulo se describe la arquitectura física y lógica del sistema diseñado, los algoritmos en que se basa y como se combinan para obtener los resultados deseados.*

## 4.1 Conceptos

### 4.1.1 ¿Que es la odometría?

La palabra odometría se compone de las palabras griegas *odos* (que significa "camino") y *metron* (que significa "medida").

La odometría consiste en el uso de datos de sensores de movimiento para estimar cambios de posición a lo largo del tiempo. Esta técnica la utilizan algunos robots, tanto voladores como terrestres, para estimar (no determinar) su posición relativa a un lugar de referencia. Este método no es preciso, pues no se basa en medias fiables, por eso se usa como sistema de estimación. A corto plazo, los errores de medida suelen ser despreciables, pero si espera el tiempo suficiente, la acumulación de todos los errores de medida llega a ser muy considerable, por lo que se hace necesaria la intervención de algún otro sistema corrector que (atendido o desatendido) restablezca el sistema y especifique de algún modo cual es su posición real con respecto al punto de referencia inicial.

Por otro lado, esta técnica se basa en ecuaciones matemáticas no muy complejas, por lo que su coste computacional y económico, es mucho mas bajo en comparación con otros sistemas mas sofisticados. Si se requiere gran grado de precisión, el uso de la odometría no es recomendable, pero para casos como el que ocupa este trabajo puede ser suficiente, pues no se trata un sistema

crítico. Si el *drone* se desvía lentamente de su posición inicial, esta puede ser corregida ocasionalmente por control remoto, pero gracias al sistema implementado, el número de veces que se debe intervenir para mantener al *drone* en su posición se ve reducido considerablemente.

### 4.1.2 ¿En que consiste la odometría visual?

En robótica y visión artificial, la odometría visual es el proceso de determinar la posición y orientación de un robot mediante el análisis de las imágenes de las cámaras asociadas. Se ha utilizado en una amplia variedad de aplicaciones robóticas, como en los *Mars Exploration Rovers*. [81]

Los sitemas de odometría tradicionales se basan en el uso de los datos del movimiento de los actuadores para estimar el cambio en la posición en el tiempo a través de dispositivos, tales como codificadores giratorios para medir rotaciones de la rueda. Si bien es útil para muchos vehículos de ruedas o tipo oruga, las técnicas de odometría tradicionales no se pueden aplicar a los robots móviles con métodos de locomoción no estándar, tales como robots con patas. Además, como ya se ha comentado, la odometría sufre universalmente de problemas de precisión, puesto que, por ejemplo, las ruedas tienden a deslizarse sobre el suelo creando una distancia de viaje no uniforme en comparación con las rotaciones de la rueda. El error se agrava cuando el vehículo opera en superficies no lisas.

Sin embargo, la odometría visual mejora considerablemente la precisión de navegación en cualquier tipo de robots o vehículos, independientemente del sistema de locomoción que usen, así como de las características de la superficie sobre la que se desplacen.

### 4.1.3 ¿En que consiste la homografía?

Dadas dos imágenes del mismo plano tomadas desde dos puntos de vista diferentes, es posible relacionar los puntos en común de ambas imágenes mediante la matriz de transformación "H". La homografía es el proceso de obtención dicha matriz a partir de las nubes de puntos obtenidas de cada una de las imágenes de modo que cumple las siguientes condiciones:

- Considerando un punto $x=(u,v,1)$ en una imagen y otro punto $x'=(u',v',1)$ en la otra, la matriz de homografía es una matriz de 3x3

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

Figura 4.1: Matriz de homografía

- La homografía relaciona las coordenadas de cada píxel en las dos imágenes si. Cuando se aplica a todos los píxeles, la segunda imagen es una versión deformada de la primera.

Esto tiene muchas aplicaciones prácticas, tales como la rectificación de imagen, registro de la imagen o cálculo de movimiento de rotación y traslación entre dos imágenes de la cámara. Una vez que la rotación y la traslación de la cámara han sido extraídas de una matriz de homografía, esta información puede ser utilizada para la navegación, o para insertar modelos 3D de objetos en una imagen o un vídeo, de modo que se presenten con la perspectiva correcta y parezcan haber formado parte de la escena original.

## 4.2 Arquitectura física

En la siguiente figura se puede ver un diagrama que representa la forma en la que las diferentes partes del *drone* están interconectadas.

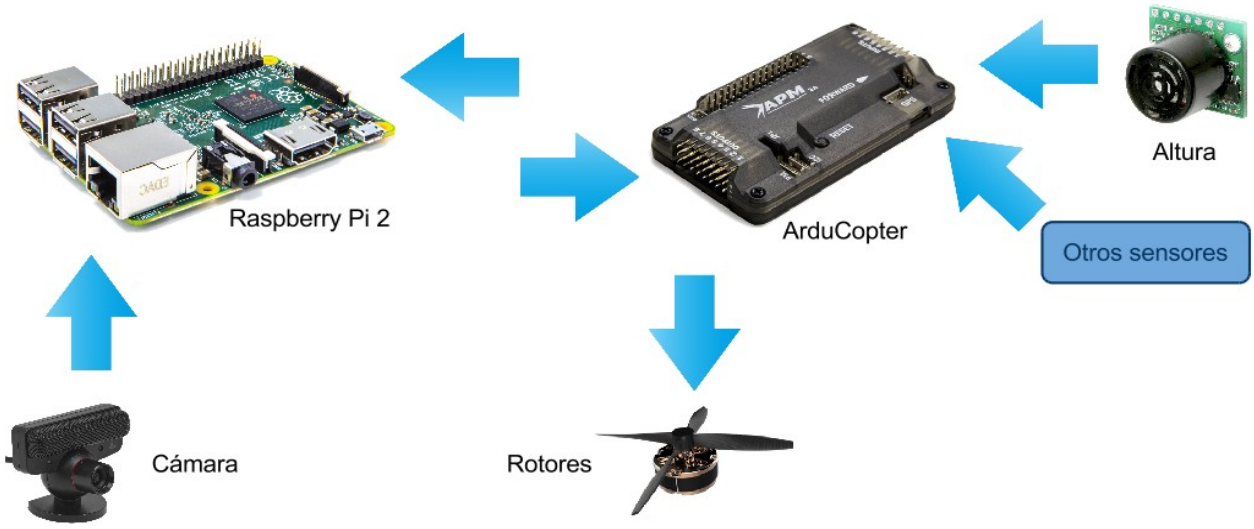


Figura 4.2: Diagrama de interconexión de los diferentes componentes del sistema

La figura se puede dividir en dos partes. En el lado derecho se encuentran

todos los componenetes pre-existentes en el *drone*: placa central (Arducopter), sensor ultrasónico de altitud, otros sensores y los rotores. A la izquierda se encuentran los componentes añadidos: ordenador de abordo (Raspberry Pi) y cámara.

La placa central del piloto automático dispone de un puerto de comunicaciones serie que está conectado a través de un conversor a uno de los puertos USB del ordenador de abordo. Esa interfaz de comunicaciones permite a la Raspberry Pi tanto enviar órdenes al *drone* como recibir datos de los sensores conectados al mismo.

## 4.3 Arquitectura lógica

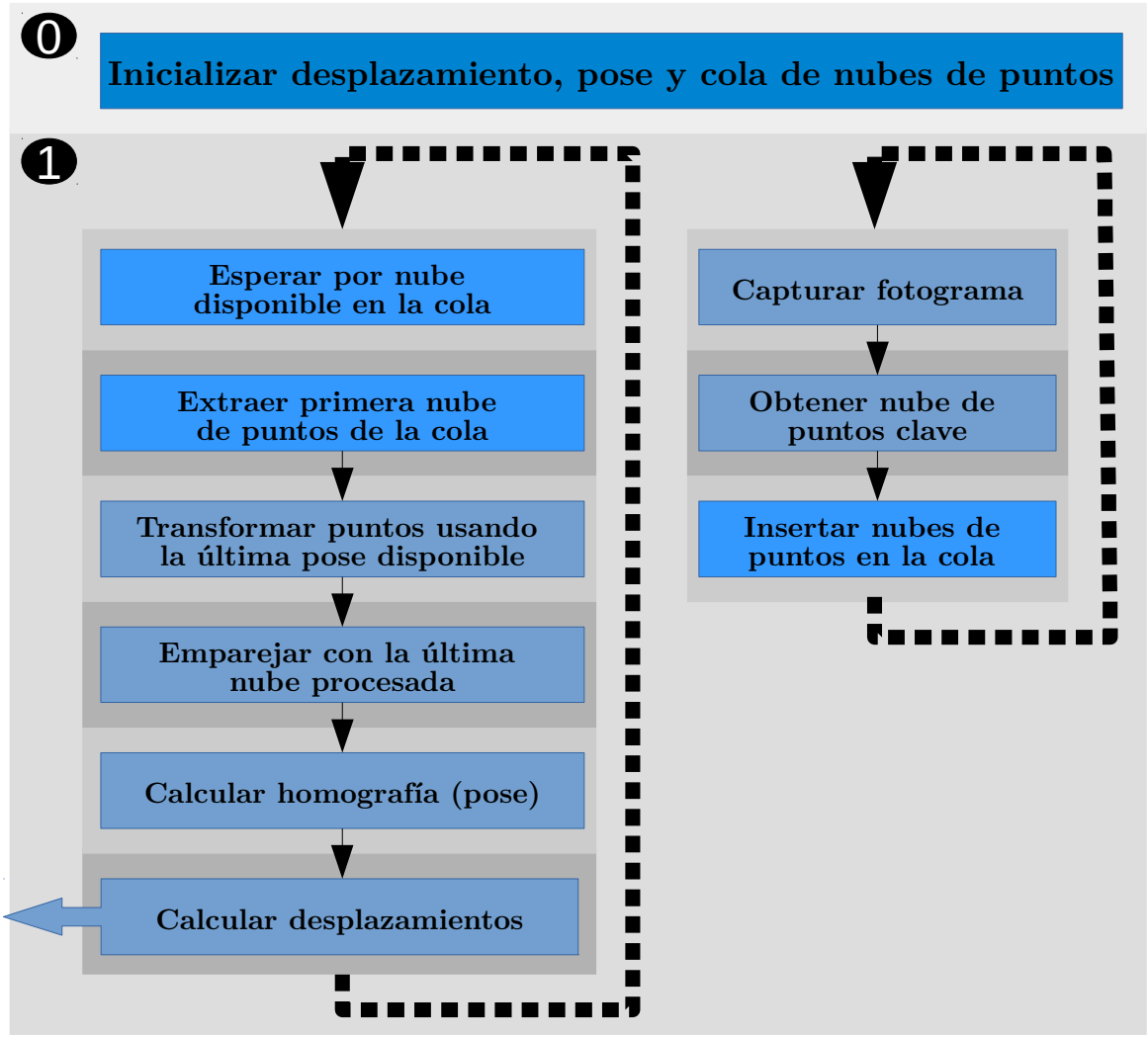


Figura 4.3: Diagrama de funcionamiento del módulo de ROS encargado de inferir odometría a partir de la secuencia de imágenes.

El desarrollo realizado en este trabajo se centra en la implementar el módulo para ROS que se ejecuta en la Raspberry Pi y obtiene los datos de posicionamiento a partir de las imágenes de la cámara y las medidas de los sensores de altura e inclinación.

En el diagrama superior se pueden ver los diferentes procesos implicados en la obtención de la odomotría, así como el orden en que se ejecutan, cuáles lo hacen de forma simultánea y cuáles no. En general, todo el trabajo se divide en dos etapas numeradas como 0 y 1.

En cada etapa se pueden ver los procesos que se ejecutan simultáneamente, por ejemplo, en la primera etapa solo se ejecuta una tarea encargada de la incialización de los valores de las diferentes variables necesarias para el proceso, y en la segunda etapa se pueden ver dos procesos que se ejecutan al mismo tiempo, cada uno en un hilo, y que a su vez se dividen en varias etapas. El paso de una etapa a la siguiente exige que todas las tareas implicadas en la misma hayan finalizado.

La etapa 1 es especial, pues ejecuta dos conjuntos de procesos de forma simultánea. Mientras se calcula la homografía a partir de la última pareja de puntos y se obtienen los desplazamientos (odometría, a la izquierda), se está capturando el siguiente fotograma y obteniendo sus puntos clave (bloque de la derecha). Esta etapa es la que genera los datos de salida del proceso, tal y como se puede deducir del cuadro con flecha a la izquierda. Dichos datos pasarán al proceso principal para ser procesados y lograr el objetivo final.

A continuación se resumen cada uno de los procesos implicados en el diagrama superior, así como la descripción de cada uno:

- **Captura de un fotograma:** como su propio nombre indica, captura un fotograma de la cámara inferior. Para que este proceso sea los mas rápido posible se implementa usando directamente las capacidades de OpenCV.
- **Obtención de nube de puntos clave:** a partir de cada fotograma se realiza un búsqueda de puntos potencialmente localizables en el siguiente fotograma. Esta búsqueda se basa en localizar zonas de alto contraste, como bordes y esquinas en los diferentes patrones y texturas de la imagen.

- **Emparejamiento de dos nubes de puntos:** este proceso empareja los puntos de dos imágenes sucesivas, descartando los que no sean comunes. A partir de este momento, se dispone de dos nubes de puntos relacionadas entre si.
- **Cálculo de homografía:** a partir de las dos nubes de puntos emparejadas, se obtiene la matriz de homografia, que especifica la perspectiva de la segunda nube de puntos con respecto a la primera.
- **Transformación de puntos en base a pose:** este proceso mejora considerablemente la precisión mediante la transformación de última nube de puntos a la perspectiva usando la última pose. De este modo, el emparejado es mucho mas preciso. El inconveniente es que para poder realizar este proceso se requiere disponer de los resultados de la homografía de los dos fotogramas anteriores, por lo que se limita la simultaneidad.
- **Cálculo de desplazamientos:** a partir de la matriz de homografía, calcula los desplazamientos y el giro que probablemente haya causado el cambio de perspectiva. Esos son los datos de salida del módulo descrito.

Los datos resultantes son publicados como mensajes estándar de odometría de ROS y capturados por el módulo o proceso principal. El modo en que dichas comunicaciones y flujos de datos se realizan queda definido en el siguiente diagrama:

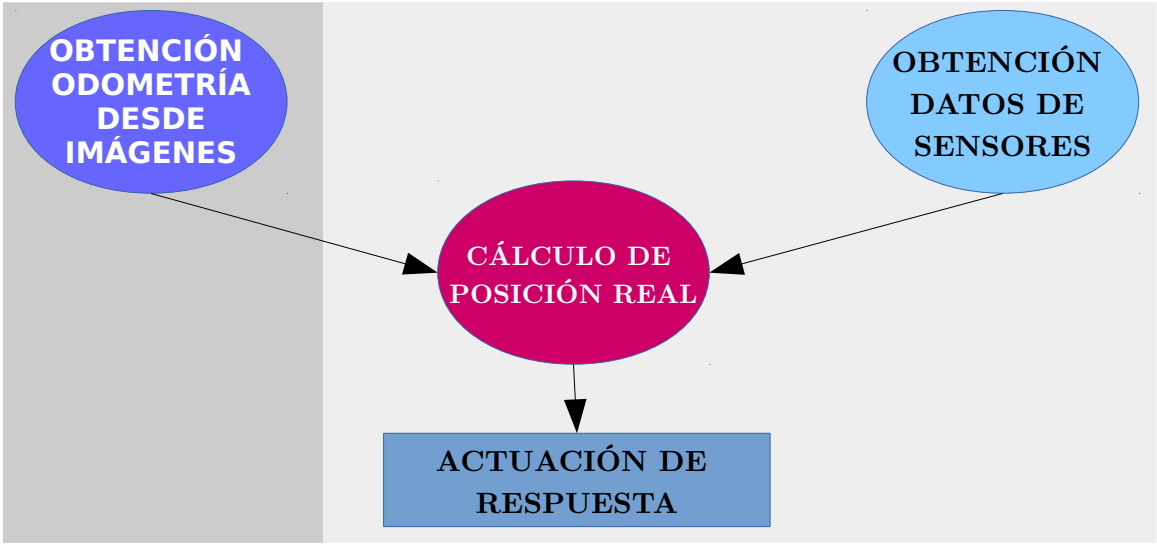


Figura 4.4: Diagrama de funcionamiento general.

La etapa principal, marcada con fondo gris oscuro, es la explicada anteriormente. Los procesos sobre fondo gris más claro son los realizados por el hilo principal y pertenecen a la última etapa del desarrollo.

En esta última etapa se recibe la información de la odometría del módulo principal y se une a la información de altura e inclinación de los sensores de abordo. Toda esa información se combina para lograr dos objetivos:

1. corregir errores de medida en el desplazamiento que puedan estar ocasionados por la inclinación actual del *drone*, situación que ocurre cada vez que este esté realizando una maniobra de rectificación de posición, usando los sensores de inclinación se pueden;
2. convertir los desplazamientos, hasta ahora dados en píxeles, a metros o centímetros, mediante las medidas de altitud.

Por último, una vez normalizados todos los datos se procede a calcular las posibles maniobras de corrección de posición que pudieran ser necesarias y en caso de serlo, a ejecutar dichas maniobras.

## 4.4 Obtención de la odometría a partir de la homografía

### 4.4.1 Cálculo de la odometría a partir de la matríz de homografía

Como ya se ha comentado anteriormente, la matriz de homografía relaciona los puntos de un fotograma con los del siguiente. Por lo tanto, es posible calcular los desplazamientos y las rotaciones que han generado dicha transformación.

Como las transformaciones afectan a un plano 2D, la matriz de homografía se limita del siguiente modo, pues no hay tercera dimensión:

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

Figura 4.5: Matriz de homografía para 2D

Por lo tanto, las transformaciones se pueden describir como sigue:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Figura 4.6: Transformación de puntos mediante la matriz de homografía

La nueva matriz de homografía se puede factorizar de modo que quedan separados los desplazamientos de la rotación, escala y 'shear', lo que da como resultado:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & h_{13} \\ 0 & 1 & h_{23} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} h_{11} & h_{12} & 0 \\ h_{21} & h_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Figura 4.7: Transformación de puntos mediante la matriz de homografía factorizada

Donde los elementos $h_{13}$ y $h_{23}$ se refieren directamente a los desplazamientos en los ejes X e Y respectivamente y la submatriz $\begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix}$ puede ser descompuesta en el resto de factores como sigue,

$$A = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} = \begin{pmatrix} p & 0 \\ 0 & r \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ q & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\Theta) & sen(\Theta) \\ -sen(\Theta) & -\cos(\Theta) \end{pmatrix}$$

Figura 4.8: Descomposición de la submatriz de transformación sin incluir desplazamiento

dónde:

Escala en X: $p = \sqrt{h_{11}^2 + h_{12}^2}$

Escala en Y: $r = \frac{det\, A}{p} = \frac{h_{11}h_{22} - h_{12}h_{21}}{\sqrt{h_{11}^2 + h_{12}^2}}$

'Shear': $q = \frac{h_{11}h_{21} - h_{12}h_{22}}{det\, A} = \frac{h_{11}h_{22} - h_{12}h_{21}}{h_{11}h_{21} - h_{12}h_{22}}$

Ángulo de giro respecto al punto 0,0: $\Theta = atan\,2(h_{12}, h_{11})$

[82]

Por último solo queda solucionar un inconveniente. Como la cámara se

encuentra situada en el centro inferior del drone, cuando este gira sobre si mismo, la imagen gira sobre su centro, mientras que el giro y el desplazamiento que la homografía entrega está inferido con respecto a la esquina superior izquerda (punto 0,0).

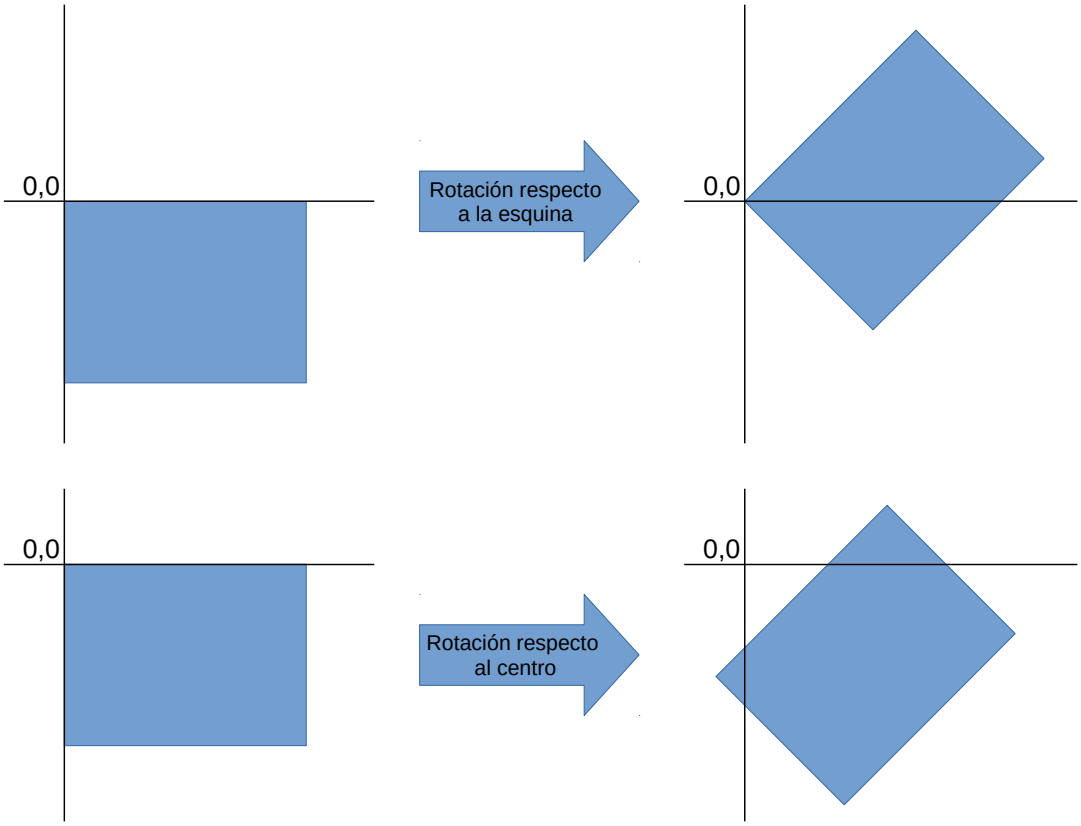


Figura 4.9: Diferencia de desplazamiento provocada por el cambio de referencia

Independientemente del punto de referencia, el ángulo de giro siempre es el mismo, por lo que el ángulo obtenido del procedimiento anterior es válido sin más.

Visto desde el punto (0,0), un giro con respecto al centro de la imagen es equivalente al mismo giro con respecto a la esquina superior izquierda más el desplazamiento en los ejes X e Y resultante del cambio de referencia, tal y como se puede ver en la figura superior.

Tras un giro y un desplazamiento con respecto al centro de la imagen. El desplazamiento observado con respecto al punto (0,0) es la suma del provocado por el cambio de referencia y el efectivamente realizado.

Para obtener el desplazamiento con respecto al centro, tan solo es necesario

calcular qué parte del mismo que es resultado del cambio de referencia y restárselo al obtenido con respecto al punto (0,0), obteniendo así el desplazamiento real. La forma de hacerlo matemáticamente se ilustra a continuación:

- Se define la posición de la esquina superior izquierda con respecto al centro de la imagen como:

  $$pos_{0,0} = \begin{pmatrix} -W/2 \\ -H/2 \\ 1 \end{pmatrix}$$

  en donde:

  - W es el ancho de la imagen en píxeles
  - H es el alto de la imagen en píxeles

- Se calcula error en el desplazamiento causado por el cambio de referencia

  $$desp_{err} = \begin{pmatrix} \cos(\Theta) & sen(\Theta) & 160 \\ -sen(\Theta) & \cos(\Theta) & 120 \\ 0 & 0 & 1 \end{pmatrix} \cdot pos_{0,0}$$

- Se le resta al desplazamiento obtenido el error provocado por el cambio de referencia

  $$desp = \begin{pmatrix} d_x \\ d_y \\ 1 \end{pmatrix} - desp_{err}$$

Figura 4.10: Procedimiento de reajuste de desplazamientos por el cambio de punto de referencia

En donde el elemento (0,0) de 'desp' representa desplazamiento sobre el eje X (dx) y el (1,0) el desplazamiento sobre el eje Y (dy)

### 4.4.2 Conversión de unidades

Como los desplazamientos inferidos por el ordenador vienen dados en píxeles, éstos deben ser convertidos a metros, para ello se usará la información de altura con respecto al suelo, que entrega uno de los sensores del *drone*, en combinación con la distancia focal de la cámara usada. Utilizamos para ello la

siguiente expresión:

$$d_X = F \cdot \frac{d_x}{Z} \qquad d_Y = F \cdot \frac{d_y}{Z}$$

Figura 4.11: Conversión de desplazamiento en píxeles a metros con el uso de la altura y la distancia focal.

en dónde:

$d_X$: Desplazamiento sobre el eje X en metros

$d_Y$: Desplazamiento sobre el eje Y en píxeles

F: Distancia focal de la cámara en metros

## 4.5 Combinación de la posición de inferida por la cámara y los datos de los sensores

Cualquier cambio del ángulo de inclinación del *drone* con respecto a la superficie provocará que el sistema de odometría visual detecte un falso desplazamiento, que se puede ver marcado en rojo en la siguiente figura:

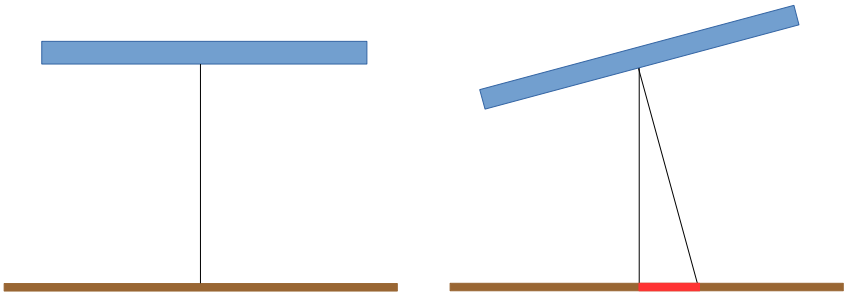

Figura 4.12: Falso desplazamiento provocado por los cambios de inclinación

Por lo tanto, es necesario tener en cuenta la inclinación del *drone* con respecto a sus ejes X e Y para compensar el desplazamiento relativo provocado por dichos ángulos.

Como ya se ha comentado anteriormente, el *drone* dispone de dicha información, por lo que solo es necesario lanzar el módulo 'mavros' de ROS y "escuchar" los valores de altura e inclinación. Usando esos valores, se puede aplicar el teorema de Pitágoras para calcular los desvíos provocados por los cambios de inclinación, en el que la altura medida por el sensor es la hipotenusa del triángulo y los catetos son la altura real y el falso

desplazamiento. Por lo que el cálculo se reduce a:

$$d_x = d_x - sen(\Theta_x)/Z$$

$$d_y = d_y - sen(\Theta_y)/Z \quad ,$$

en donde Z es la altura medida por el sensor ultrasónico del *drone*.

Figura 4.13: Compensación del desplazamiento usando los ángulos de inclinación

## 4.6 Cálculo de maniobras correctivas

En este punto, ya se dispone de toda la información necesaria para calcular las maniobras necesarias para la corrección de la posición del *drone*.

Para realizar este paso hay que generar mensajes que cambien la inclinación del *drone* sobre el eje en que se desee realizar el desplazamiento, e ir reduciéndola de forma progresiva hasta llegar a la posición deseada. Repitiendo este procedimiento cada vez que la posición del *drone* salga de un margen establecido se logrará mantenerlo aproximadamente en el mismo lugar.

Con respecto a la rectificación del giro, seria posible maniobrar de modo análogo para recuperar así también la orientación, pero como el objetivo es simplemente mantener la posición, se deja para posibles futuras líneas de investigación.

# Capítulo 5
# Desarrollo

*Este capítulo está dedicado a exponer las diferentes tareas que se han llevado a cabo durante todo el proceso de desarrollo, describiendo los pasos seguidos para configurar los diferentes sistemas, las partes del código escrito y su propósito, así como algunas de las pruebas realizadas y sus resultados.*

## 5.1 Introducción

El desarrollo del trabajo ha sido dividido en dos partes, primero se realizó la investigación y el desarrollo de un software que sirva de prueba de concepto, luego se procedió a llevar a cabo las pruebas oportunas para verificar la validez del mismo, su precisión y su eficiencia, teniendo siempre en mente que dicho código debería ser capaz de ejecutarse con la máxima fluidez en una Raspberry Pi.

## 5.2 Instalación y configuración Raspbian y ROS en el Raspberry Pi

A continuación se procederá a explicar cuáles han sido los pasos seguidos para instalar y configurar ROS Indigo con los componentes necesarios para este desarrollo en una Raspberry Pi. Estas instrucciones están basadas en la instalación desde los códigos fuente de Indigo, teniendo en cuenta algunas dependencias específicas de Raspbian.

### 5.2.1 Requisitos

Se asume que ya se dispone de un Raspbian (Wheezy) instalado en la Raspberry Pi. Dicho sistema puede ser descargado desde:

http://www.raspberrypi.org/downloads

### 5.2.2 Configuración de los repositorios de ROS

Para agregar los repositorios necesarios se ejecutan los siguientes comandos:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu wheezy main"
> /etc/apt/sources.list.d/ros-latest.list'
$ wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O
- | sudo apt-key add -
```

A continuación, se actualiza el índice del gestor de paquetes de Debian y se realiza una actualización del sistema:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Lo siguiente es instalar las dependencias del 'bootstrap' de ROS:

```
$ sudo apt-get install python-setuptools python-pip python-yaml python-
argparse python-distribute python-docutils python-dateutil python-
setuptools python-six
$ sudo pip install rosdep rosinstall_generator wstool rosinstall
```

Luego se procede a inicializar 'rosdep':

```
$ sudo rosdep init
$ rosdep update
```

### 5.2.3 Instalación

A partir de este momento se puede comenzar con la descarga y compilación de ROS Indigo.

Para poder compilar los paquetes del núcleo, es necesario crear un espacio de trabajo para 'catkin', la herramienta encargada de automatizar la compilación:

```
$ mkdir ~/ros_catkin_ws
$ cd ~/ros_catkin_ws
```

A continuación hay que descargar los paquetes del núcleo junto a todos los necesarios para el proyecto. Para ello se usará la herramienta 'wstool'. En este caso no se descargará ningún paquete relacionado con el entorno gráfico, pues

no serán necesarios y solo consumirían parte de los limitados recursos de la Raspberry Pi . Para ello se ejecutan los siguieres comandos:

```
$ rosinstall_generator ros_comm vision_opencv cv_bridge common_msgs tf
mavros --rosdistro indigo --deps --wet-only --exclude roslisp --tar >
indigo-custom_ros.rosinstall
$ wstool init src indigo-ros_comm-wet.rosinstall
```

Con esto se añadirán todos los paquetes de código y se guardarán en el directorio ~/ros_catkin_ws/src directory. El comando puede tardar varios minutos en descargar todos los paquetes de ROS. Para acelerar la descarga se puede usar el parámetro -j8, provocando que se hagan 8 descargas simultáneas.

**Nota 1:** Se ha excluido la librería 'roslisp' en el generador debido a que la dependencia 'sbcl' no está disponible en los repositorios de Raspbian. En teoría es posible compilarla desde el código fuente, pero no se ha probado.

**Nota 2:** Si 'wstool init' falla o es interrumpido, se puede reanudar con el siguiente comando:

```
wstool update -t src
```

### 5.2.4 Resolver dependencias

Antes de poder comenzar hay que asegurarse de que se dispone de todas las dependencias requeridas para la compilación. Para ello se puede usar la herramienta 'rsodep', pero algunos paquetes no están disponibles en los repositorios de Raspbian, por lo que habrá que compilarlos manualmente.

En este caso concreto, los paquetes necesarios son 'libconsole-bridge-dev' y 'liblz4-dev'.

Primero se inicaliza el entorno en el que se compilarán los paquetes y se instalan algunas dependencias:

```
$ mkdir ~/ros_catkin_ws/external_src
$ sudo apt-get install checkinstall cmake
$ sudo sh -c 'echo "deb-src http://mirrordirector.raspbian.org/raspbian/
testing main contrib non-free rpi" >> /etc/apt/sources.list'
$ sudo apt-get update
```

A continuación se procede a descargar, compilar e instalar las librerias necesarias. Primero 'libconsole-bridge-dev':

```
$ cd ~/ros_catkin_ws/external_src
$ sudo apt-get build-dep console-bridge
$ apt-get source -b console-bridge
$ sudo dpkg -i libconsole-bridge0.2_*.deb libconsole-bridge-dev_*.deb
```

Y a continuación 'liblz4-dev':

```
$ cd ~/ros_catkin_ws/external_src
$ apt-get source -b lz4
$ sudo dpkg -i liblz4-*.deb
```

Ahora que se sabe que no puede haber inguna dependencia incumplida, se puede usar 'rosdep' para que resuelva automáticamente el resto de dependencias:

```
$ cd ~/ros_catkin_ws
$ rosdep install --from-paths src --ignore-src --rosdistro indigo -y -r
—os=debian:wheezy
```

Este comando busca en todos los paquetes del directorio 'src' y rellena todas las dependencias que tenga. Luego las instala de forma recursiva.

La opcción '--from-paths' indica que se desean instalar las fuentes para el directorio de paquetes completo, en este caso 'src'. La opción '--ignore-src' indica a 'rosdep' que no debe intentar instalar ningún paquete de ROS en el el directorio 'src' desde el gestor de paquetes, puesto que no es necesario porque serán compilados posteriormente. La opción '--rosdistro' es necesaria para indicar qué versión de ROS se desea instalar, pues no ha sido especificado hasta ahora. Finalmente, la opción '-y' indica a 'rosdep' que debe ejecutar el gestor de paquetes en modo automático, de forma que no haga preguntas de confirmación.

Después de un tiempo, 'rosdep' terminará con la instalación de las dependencias y se podrá continuar el resto del proceso.

**Nota:** 'rosdep' puede reportar que 'python-rosdep', 'python-catkin-pkg', 'python-rospkg', y 'python-rosdistro' fallaron al instalarse; esos avisos pueden ser ignorados porque ya han sido instalados por 'pip'.

### 5.2.5 Compilar

Una vez que se han resuleto todas las dependencias se puede ejecutar el siguiente comando para comenzar con las compilación:

```
$ sudo ./src/catkin/bin/catkin_make_isolated --install
-DCMAKE_BUILD_TYPE=Release --install-space /opt/ros/indigo
```

A partir de este momento, ya se dispone de ROS compilado e instalado, por lo que se puede ejecutar el siguiente comando para empezar a usarlo:

```
$ source /opt/ros/indigo/setup.bash
```

### 5.2.6 Añadir nuevos paquetes

Es posible que a lo largo del desarrollo sea necesario añadir algún nuevo paquete al sistema ROS. Para ello hay que volver a crear el fichero 'rosinstall' con la nueva lista de paquetes, incluidos los que ya se encuentran instalados. Por ejemplo, si se desa añadir soporte para joystick, se ejecutaría el siguiente comando:

```
$ cd ~/ros_catkin_ws
$ rosinstall_generator ros_comm vision_opencv cv_bridge common_msgs tf
mavros joystick_drivers --rosdistro indigo --deps --wet-only --exclude
roslisp --tar > indigo-custom_ros.rosinstall
```

A continuación se actualiza el entorno de trabajo con 'wstool':

```
$ wstool merge -t src indigo-custom_ros.rosinstall
$ wstool update -t src
```

Después de la actualización, se puede ejecutar 'rosdep' para que busque e instale cualquier nueva dependencia requerida por los paquetes añadidos:

```
$ rosdep install --from-paths src --ignore-src --rosdistro indigo -y -r
—os=debian:wheezy
```

Y finalmente se puede proceder a recompilar y reinstalar el nuevo sistema:

```
$ sudo ./src/catkin/bin/catkin_make_isolated --install
-DCMAKE_BUILD_TYPE=Release --install-space /opt/ros/indigo
```

## 5.3 Desarrollo del nodo de ROS para obtención de la odometría

### 5.3.1 Diseño del sistema

Antes de comenzar, lo primero que se hizo fue diseñar a grandes rasgos cómo debería funcionar todo el sistema y sobre qué tecnologías se iba a implementar.

Desde el punto de vista del hardware, algunos aspectos ya estaban definidos desde el principio, como las especificaciones del *drone* a usar, mientras que en otros como el tipo de cámara y el ordenador de abordo, había varias opciones.

Después de estudiar las diferentes posibilidades se decidió usar la "Play Station Eye" conectada a una Raspberry Pi por las razones ya expuestas en capítulos anteriores.

Con respecto al software, lo más evidente es que la mejor opción es usar un lenguaje compilado como C++ para realizar un software que aproveche al máximo los recursos del sistema, por lo que se decidió implementar un módulo para ROS en C++ que se encargase de obtener y emitir la odmoetría a partir de las imágenes y los datos de los diferentes sensores.

Dado que se iba a usar OpenCV para el procesamiento de las imágenes, se decidió aprovechar su capacidad de acceder directamente a la cámara, evitando así tener que lanzar un proceso de ROS que emita mensajes de imagen, con la consecuente saturación de la memoria y la CPU.

Una vez decidida la forma en que las imágenes van a ser capturadas, solo quedaba plantear cómo serian procesadas. La forma más evidente de hacerlo es usar las herramientas de homografía de las que dispone OpenCV para buscar puntos clave que se encuentren en dos imágenes sucesivas y usar dichos puntos para calcular el desplazamiento y la rotación que habría provocado la transformación observada.

### 5.3.2 Aproximación inicial

Usando como referencia el código de ejemplo de OpenCV que muestra como realizar una homografía entre las imágenes provenientes de una cámara, se

comenzó con el desarrollo del módulo de ROS, el código para la captura de las imágenes ya estaba escrito y la forma en que combinar los diferentes procedimientos para llevar a cabo la homografía también. El problema se limitaba tan solo a extraer los puntos obtenidos en ambas imágenes y procesarlos para inferir el desplazamiento y la rotación.

Extraer las parejas de puntos en ambas imágenes, usado el código de ejemplo, es una tarea trivial, pues tan solo hay que almacenarlos en un vector para procesarlos luego. A continuación se puede ver el fragmento del código con el que se pueden obtener dichas parejas:

```
01 void drawMatchesRelative(const vector<KeyPoint>& train, const
   vector<KeyPoint>& query,
02 std::vector<cv::DMatch>& matches, Mat& img, const vector<unsigned
   char>& mask = vector<unsigned char> ())
03 {
04     for (int i = 0; i < (int)matches.size(); i++)
05     {
06         if (mask.empty() || mask[i])
07         {
08             Point2f pt_new = query[matches[i].queryIdx].pt;
09             Point2f pt_old = train[matches[i].trainIdx].pt;
10
11             point_pairs_.pushBack(Pair<Point2f, Point2f>(pt_new,
   pt_old));
12
13             cv::line(img, pt_new, pt_old, Scalar(125, 255, 125), 1);
14             cv::circle(img, pt_new, 2, Scalar(255, 0, 125), 1);
15
16         }
17     }
18 }
```

En las líneas marcadas 8 y 9, se puede ver cómo se obtiene una pareja de puntos y en la 11 cómo son almacenados en el vector. Las líneas que se ven a

continuación serían eliminadas posteriormente, pues tan solo sirven para dibujar los puntos sobre la imagen, una tarea que no es necesaria para el propósito perseguido y que consume parte de los recursos necesarios para otros trabajos. La versión completa del código de ejemplo está disponible en Internet. [83]

Una vez obtenidas las parejas de puntos, se procedió a calcular los centroides de cada una de las nubes de puntos, lo que da como resultado dos puntos que representan el centro de la nube de la primera imagen y el de la segunda. Simplemente restando esas dos coordenadas se pueden obtener los desplazamientos, pero con este sistema no se puede inferir la rotación, por lo que solo sirve como prueba de concepto y para comprobar que la homografía se está calculando correctamente.

### 5.3.3 Segunda aproximación

La aproximación inicial, aunque era muy capaz de inferir los desplazamientos, no lo era de inferir la rotación, por lo que se optó por cambiar de método en busca de uno que sí pudiera solucionar todas las variables.

La forma de solventarlo fue mediante la resolución de un sistema de ecuaciones que permite calcular cual seria la posición de un punto después de sufrir un desplazamiento y una rotación, lo que limita el problema a rellenar un sistema de '2n' ecuaciones con tres incógnitas, donde 'n' coincide con el número de parejas de puntos obtenidos mediante la homografía, quedando un sistema como el siguiente:

$$x_n' = x_n \cos(\Theta) + y_n sen(\Theta) + dx$$
$$y_n' = -x_n sen(\Theta) + y_n \cos(\Theta) + dy$$

Figura 5.1: Sistema de ecuaciones que define el giro y desplazamiento de la imagen

El sistema de ecuaciones es compatible desde el momento en que se disponga de al menos tres parejas de puntos, pero no se puede solucionar de forma tradicional. Los puntos obtenidos y emparejados no tienen por qué estar situados con total precisión, incluso se puede dar el caso de parejas de puntos incorrectamente asociados, por lo que la solución de la ecuación no sería correcta. Por lo tanto, lo que hay que hacer es, primero construir el

sistema de ecuaciones completo y luego “ir probando” diferentes valores de las incógnitas dx, dy y Θ hasta encontrar la combinación de valores que produzca que los resultados de todas las ecuaciones presenten la menor desviación posible unos con respecto a otros.

En lugar de afrontar este problema desde cero, se decidió usar una librería matemática que permite solucionar problemas como este por el método de mínimos cuadrados. Después de probar con una denominada 'dlib', se pasa a usar otra llamada 'gsl' por tener un rendimiento muy superior a la anterior.

Para poder solucionar el sistema de ecuaciones por este método fue necesario redefinir la función de modo que se convierta en un sistema de una sola ecuación, de modo que un cambio en cualquiera de las varibles implique un cambio en el resultado, por lo que el sistema pasa a tener 'n' ecuaciones como la siguiente:

$$A=(x_n\cos(\Theta)+y_n sen(\Theta)+dx)^2+(-x_n sen(\Theta)+y_n\cos(\Theta)+dy)^2$$

Figura 5.2: Ecuación cuadrática usada para el ajuste por mínimos cuadrados

en donde 'A' es el valor que la librería debe ajustar.

Para realizar el ajuste de las incógnitas, este sistema de resolución usa la derivada de la función con respecto a cada una de sus incógnitas, que le indica la pendiente y le permite decidir en qué sentido incrementar o disminuir las diferentes variables para ajustarse al máximo.

Tanto 'gsl' como 'dlib', permiten el uso de una función que hace la derivada numérica de la función, de modo que no es necesario implementar la derivada, pero si se usa este método el rendimiento se ve enormemente deteriorado debido a su elevado coste computacional. Por lo tanto, se decidió implementar la derivada para mejorar así la eficiencia del sistema.

$$d/dx=2.0\cdot(x_1[i]\cdot\cos(\Theta)-y_1[i]\cdot sen(\Theta)+d_x-x_2[i])$$
$$d/dy=2.0\cdot(x_1[i]\cdot sen(\Theta)+y_1[i]\cdot\cos(\Theta)+d_y-y_2[i])$$
$$d/d\Theta=(x_1[i]\cdot(-sen(\Theta))-y_1[i]\cdot\cos(\Theta))\cdot dx+(x_1[i]\cdot\cos(\Theta)-y_1[i]\cdot sen(\Theta))\cdot dy$$

Figura 5.3: Derivadas de la ecuación cuadrática

Para más detalles acerca de la implementación, se pueden encontrar las partes más representativas del código en el apartado 9.1.

Para que este método funcione correctamente, es necesario desplazar los puntos de modo que el centro de la imagen quede en la posición 0,0, pues la rotación es inferida con respecto al origen de coordenadas mientras que en realidad éstas se producen con respecto al centro de la imagen, debido a que la cámara se encuentra en el centro de la parte inferior del *drone*.

Los resultados de esta implementación fueron muy satisfactorios, obteniendo hasta 8 fps sin ninguna optimización del código, pero tuvo que ser destacado al descubrir que se podía eliminar y sustituir por un cálculo menos costoso sobre la matriz de homografía, que hasta este momento solo se usaba para reajustar los puntos del siguiente *frame*.

### 5.3.4 Aproximación final

Tras estudiar en mayor profundidad el funcionamiento del código implementado y los fundamentos matemáticos en los que se basa se descubrió que, aunque la segunda aproximación era muy precisa y eficaz, implicaba realizar cálculos redundantes, pues esas operaciones ya habían sido realizadas por el proceso de homografía que se usaba como origen de las nubes de puntos, por lo que se empezó a estudiar como extraer dicha información y se llegó al método explicado en el apartado 4.4.

La adaptación del código se redujo a eliminar los algoritmos de resolución por mínimos cuadrados y su sustitución por una serie de cálculos realizados sobre la matriz de homografía resultante del proceso de cada par de imágenes. De este modo, se pudo llegar una implementación muy precisa, que es capaz emparejar eficientemente los puntos de las imágenes, pudiendo incluso descartar aquellos que queden fuera como consecuencia del desplazamiento.

También se optimizó el código para ser capaz de usar varios hilos para resolver de forma simultánea varias partes del problema, lo que incrementó aun más el rendimiento.

## 5.4 Resultados obtenidos

En las pocas pruebas que se pudieron realizar sobre el código final, se obtuvieron resultados muy satisfactorios que indican que el sistema diseñado puede cumplir con su cometido, siempre que se den las condiciones ambientales adecuadas.

### 5.4.1 Rendimiento

En cuanto al rendimiento, destacar que la mejora de la implementación para que fuera "multihilo" implicó pasar de una media de 8 imágenes por segundo a 20. Teniendo en cuenta que la implementación actual sigue siendo muy susceptible a ser optimizada, se estima que sería posible pasar de las 30 imágenes por segundo con relativa facilidad.

### 5.4.2 Precisión

Como era de esperar, debido a que este sistema necesita poder identificar puntos comunes entre dos fotogramas sucesivos, es imprescindible contar con unas condiciones de luz apropiadas y sobretodo con que la superficie sobre la que vuele el *drone* aporte texturas y patrones reconocibles. En el caso de volar sobre superficies lisas, la eficacia del sistema se reduce a cero, quedando completamente inútil, pero cuando hay texturas y patrones, como objetos, manchas y demás los resultados han sido bastante precisos, lo que lleva a pensar que para esos casos este sistema es factible.

Para comprobar el correcto funcionamiento del prototipo implementado se hizo una prueba de "vuelo simulado" donde, en lugar de hacer volar al *drone,* se sujetó con las manos y se desplazó a lo largo de un tramo rectangular de unos 9 metros mientras este registraba todo el movimiento detectado.

Teniendo en cuenta que el sistema está pensado para detectar desplazamientos muy cortos, los resultados obtenidos se pueden considerar muy satisfactorios.

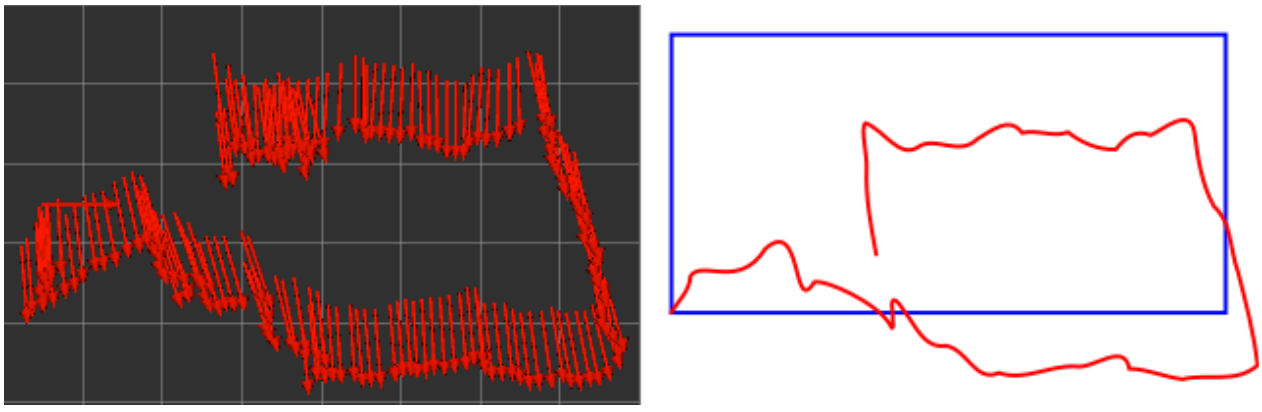

Figura 5.4: Resultados en un recorrido largo andando con el *drone* en las manos

# Capítulo 6
# Conclusiones y líneas futuras

## 6.1 Conclusiones

Los resultados obtenidos se han situado por encima de las expectativas, pues no se esperaba inicialmente poder llegar a los niveles de precisión obtenidos con el tiempo y los recursos disponibles.

El estudio e investigación de muchos detalles sobre el campo de la visión artificial, consumió la mayor parte del tiempo, y provocó situaciones en las que hubo que volver sobre el camino ya andado para prácticamente comenzar de nuevo. El tiempo restante no fue suficiente para profundizar todo lo que habría sido deseable en el desarrollo y optimización de los algoritmos, y mucho menos para realizar más pruebas que las imprescindibles sobre el prototipo final.

## 6.2 Líneas futuras

Incluso con el factor tiempo en contra, se ha logrado sentar una base sólida sobre la que se puede trabajar para llegar a una solución completa, donde se cubran con más detalle aspectos como la optimización de los algoritmos de odometría y el diseño detallado de los de reposicionamiento, que no han podido ser completamente definidos a lo largo de este trabajo. Resulta por tanto factible crear una nueva línea que continúe y finalice el trabajo realizado hasta ahora.

# Capítulo 7
# Conclusions & Future work lines

## 7.1 Conclusions

Results obtained have exceeded expectations with regard to time and available resources which, at first, did not seem enough to reach the precision finally attained.

Most of the worktime was invested on research and implementation of artificial vision concepts, due to situations in which the already walked path had to be undone and restarted all over again .

The remaining time was not enough neither to deepen in development and algorithms' optimization -as would be ideal, nor to perform the necessary tests on the finished prototype.

## 7.2 Future work lines

Despite the lack of time, it has been layed out a solid foundation towards a comprehensive solution. However, some aspects still need additional development, such as the optimization of odometry algorithms and the detailed design of the repositioning ones, which have not been completely defined. Therefore, it is possible to create a new line of research that furthers and completes the work done.

# Capítulo 8
# Presupuesto

| Descripción | Cant. | Precio | Subtotal |
|---|---|---|---|
| Desarrollo e investigación (mano de obra) | 240 h. | 45,00 € | 10.800,00 € |
| VANT (*drone*) | 1 ud. | 500,00 € | 500,00 € |
| Raspberry Pi y componentes auxiliares | 1 ud. | 100,00 € | 100,00 € |
| Cámara | 1 ud. | 15,00 € | 15,00 € |
| Amortización(1) de equipos informáticos | 240 h. | 0,12 € | 28,80 € |
| | | **Total presupuesto:** | **11.443,80 €** |

**Tabla 8.1: Presupuesto**

*Impuestos no incluidos

(1) La amortización se obtuvo a partir del valor de reposición del equipamiento(1,576,80 €) y de una vida útil estimada de 13140 horas.

# Capítulo 9
# Fragmentos de código fuente

## 9.1 Obtención de desplazamiento y rotación por mínimos cuadrados

[...]

```
Point3d RotationMatrix::solveLeastSquaresGSL(const
std::vector<std::pair<Point3d, Point3d> > &pairs) {
    Point3f result;

    const gsl_multifit_fdfsolver_type *T;
    gsl_multifit_fdfsolver *s;
    int status;
    unsigned int i, iter = 0;
    const size_t n = pairs.size();
    const size_t p = 3;

    if (n < p) {
        result.x = 0;
        result.y = 0;
        result.z = 0;
        return result;
    }

    gsl_matrix *covar = gsl_matrix_alloc (p, p);
    double x_1[n], y_1[n], x_2[n], y_2[n], sigma[n];
    struct data d = { n, x_1, y_1, x_2, y_2, sigma};
    gsl_multifit_function_fdf f;
    double x_init[3] = { 0.0, 0.0, 0.0 };
    gsl_vector_view x = gsl_vector_view_array (x_init, p);
```

```
const gsl_rng_type * type;
gsl_rng * r;

gsl_rng_env_setup();

type = gsl_rng_default;
r = gsl_rng_alloc (type);

f.f = &gsl_f;
f.df = &gsl_df;
f.fdf = &gsl_fdf;
f.n = n;
f.p = p;
f.params = &d;

// Datos que van a ser ajustados
for (i = 0; i < n; i++) {
    x_1[i] = pairs[i].second.x;
    y_1[i] = pairs[i].second.y;
    x_2[i] = pairs[i].first.x;
    y_2[i] = pairs[i].first.y;

    sigma[i] = 0.1;
};

T = gsl_multifit_fdfsolver_lmsder;
s = gsl_multifit_fdfsolver_alloc (T, n, p);
gsl_multifit_fdfsolver_set (s, &f, &x.vector);

do {
    iter++;
    status = gsl_multifit_fdfsolver_iterate (s);

    if (status)
        break;

    status = gsl_multifit_test_delta (s->dx, s->x,
                                      1e-4, 1e-4);
}
while (status == GSL_CONTINUE && iter < 500);

result.x = gsl_vector_get(s->x, 0);
```

```
    result.y = gsl_vector_get(s->x, 1);
    result.z = fmod(gsl_vector_get(s->x, 2),(2*M_PI));

    gsl_multifit_fdfsolver_free (s);
    gsl_matrix_free (covar);
    gsl_rng_free (r);

    return result;
}

int RotationMatrix::gsl_f (const gsl_vector * x, void *data,
        gsl_vector * f) {
    size_t n = ((struct data *)data)->n;      //Número de funciones
    double *x_1 = ((struct data *)data)->x_1; //Datos observados
    double *y_1 = ((struct data *)data)->y_1; //Datos observados
    double *x_2 = ((struct data *)data)->x_2; //Datos observados
    double *y_2 = ((struct data *)data)->y_2; //Datos observados
    double *sigma = ((struct data *) data)->sigma;

    double d_x = gsl_vector_get (x, 0);
    double d_y = gsl_vector_get (x, 1);
    double tita = gsl_vector_get (x, 2);

    double costita = cos(tita);
    double sintita = sin(tita);

    size_t i;

    for (i = 0; i < n; i++) {
        double Xr = x_1[i] * costita - y_1[i] * sintita + d_x - x_2[i];
        double Yr = x_1[i] * sintita + y_1[i] * costita + d_y - y_2[i];
        gsl_vector_set (f, i, (Xr*Xr + Yr*Yr)/sigma[i]);
    }

    return GSL_SUCCESS;
}

int RotationMatrix::gsl_df (const gsl_vector * x, void *data,
         gsl_matrix * J) {
    size_t n = ((struct data *)data)->n;      //Número de funciones
    double *x_1 = ((struct data *)data)->x_1; //Datos observados
    double *y_1 = ((struct data *)data)->y_1; //Datos observados
```

```
    double *x_2 = ((struct data *)data)->x_2; //Datos observados
    double *y_2 = ((struct data *)data)->y_2; //Datos observados
    double *sigma = ((struct data *) data)->sigma;

    double d_x = gsl_vector_get (x, 0);
    double d_y = gsl_vector_get (x, 1);
    double tita = gsl_vector_get (x, 2);

    double costita = cos(tita);
    double sintita = sin(tita);

    size_t i;

    for (i = 0; i < n; i++) {
        // d/dt( (x_1 * cos(t) - y_1 * sin(t) + d_x - x_2)^2 + (x_1 *
sin(t) + y_1 * cos(t) + d_y - y_2)^2 )
        double dx = 2.0 * (x_1[i] * costita - y_1[i] * sintita + d_x -
x_2[i]);
        double dy = 2.0 * (x_1[i] * sintita + y_1[i] * costita + d_y -
y_2[i]);
        double dt = (x_1[i] * (-sintita)-y_1[i] * costita) * dx+
                    (x_1[i] *   costita -y_1[i] * sintita) * dy;
        gsl_matrix_set (J, i, 0, dx/sigma[i]);
        gsl_matrix_set (J, i, 1, dy/sigma[i]);
        gsl_matrix_set (J, i, 2, dt/sigma[i]);
    }
    return GSL_SUCCESS;
}

int RotationMatrix::gsl_fdf (const gsl_vector * x, void *data,
          gsl_vector * f, gsl_matrix * J) {
    gsl_f (x, data, f);
    gsl_df (x, data, J);

    return GSL_SUCCESS;
}
```

# Bibliografía

1: Wikipedia, Clasificación de los VANT, https://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado#Clasificaci.C3.B3n_de_los_VANT

2: Says, Robert Kanyike, History of U.S. Drones, http://understandingempire.wordpress.com/2-0-a-brief-history-of-u-s-drones/

3: Dempsey, Martin E., Eyes of the Army – U.S. Army Roadmap for Unmanned Aircraft Systems 2010–2035, http://www-rucker.army.mil/usaace/uas/US%20Army%20UAS%20RoadMap%202010%202035.pdf

4: Taylor, A. J. P., Jane's Book of Remotely Piloted Vehicles,

5: Wagner, William, Lightning Bugs and other Reconnaissance Drones; The can-do story of Ryan's unmanned spy planes,

6: The Encyclopedia of the Arab-Israeli Conflict, A Political, Social, and Military History,

7: owstuffworks.com, A Brief History of UAVs, http://www.howstuffworks.com/reaper1.htm

8: Strategypage.com, Russia Buys A Bunch Of Israeli UAVs, http://www.strategypage.com/htmw/htairfo/articles/20090409.aspx

9: Azoulai, Yuval, Unmanned combat vehicles shaping future war, http://www.globes.co.il/serveen/globes/docview.asp?did=1000691790

10: Levinson, Charles, Israeli Robots Remake Battlefield, http://online.wsj.com/article/SB126325146524725387.html

11: , Almost 1 In 3 U.S. Warplanes Is a Robot, http://www.wired.com/2012/01/drone-report/

12: Singer, Peter W, A Revolution Once More: Unmanned Systems and the Middle East, http://www.brookings.edu/articles/2009/11_robotic_revolution_singer.aspx

13: Radsan, AJ; Murphy, Measure Twice, Shoot Once: Higher Care for Cia-

Targeted Killing,

14: Horgen, John, Unmanned Flight,
http://ngm.nationalgeographic.com/2013/03/unmanned-flight/horgan-text

15: Arduino, Arduino Mega, http://www.arduino.cc/

16: Pixhawk, Pixhawk, https://pixhawk.org

17: , Mavlink, http://qgroundcontrol.org/mavlink/start

18: , ,

19: Cellan-Jones, Rory, A £15 computer to inspire young programmers,
http://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_young.html

20: Price, Peter, Can a £15 computer solve the programming gap?,
http://news.bbc.co.uk/2/hi/programmes/click_online/9504208.stm

21: Bush, Steve, Dongle computer lets kids discover programming on a TV,
http://www.electronicsweekly.com/Articles/2011/05/25/51129/Dongle-computer-lets-kids-discover-programming-on-a.htm

22: Wong, George, Build your own prototype Raspberry Pi minicomputer,
http://www.ubergizmo.com/2011/10/build-raspberry-pi-minicomputer/

23: Moorhead, Joanna, Raspberry Pi device will 'reboot computing in schools', http://www.guardian.co.uk/education/2012/jan/09/raspberry-pi-computer-revolutionise-computing-schools?newsfeed=true

24: raspberrypi.org, Raspberry Pi • View topic - Raspberry Pi as the successor of BBC Micro,
http://www.raspberrypi.org/phpBB3/viewtopic.php?f=2&t=5118

25: Quested, Tony, Raspberry blown at Cambridge software detractors,
http://www.businessweekly.co.uk/blog/cambridge-today-tony-quested/13664-raspberry-blown-at-cambridge-software-detractors

26: Williams, Chris, Psst, kid... Wanna learn how to hack?,
http://www.theregister.co.uk/2011/11/28/raspberry_pi/page3.html

27: Richard Lawler, Raspberry Pi credit-card sized Linux PCs are on sale now, $25 Model A gets a RAM bump,

http://www.engadget.com/2012/02/29/raspberry-pi-credit-card-sized-linux-pcs-are-on-sale-now-25-mo/

28: , launch of the model A announced,
http://www.raspberrypi.org/archives/3215

29: Raspberry Pi Foundation, Introducing Raspberry Pi Model A+,
http://www.raspberrypi.org/blog/#raspberry-pi-model-a-plus-on-sale

30: Broadcom.com, BCM2835 Media Processor; Broadcom,
http://www.broadcom.com/products/BCM2835

31: Brose, Moses, Broadcom BCM2835 SoC has the most powerful mobile GPU in the world?,
https://web.archive.org/web/20120413184701/http://www.grandmax.net/2012/01/broadcom-bcm2835-soc-has-powerful.html

32: raspbian.org, Raspbian – Debian optimized for the Raspberry Pi hardware, http://www.raspbian.org/

33: Millman, Rene, Raspberry Pi gets memory upgrade,
http://www.itpro.co.uk/643555/raspberry-pi-gets-memory-upgrade

34: intelminer.com, Gentoo for the Raspberry Pi!,
http://intelminer.com/blog/?p=18

35: , Open WebOS on a Raspberry Pi!,
http://www.raspberrypi.org/archives/tag/open-webos

36: , Inspired by CrunchBang Linux, and based on Raspbian. Features the Openbox desktop environment,

37: , Pidora, http://pidora.ca/

38: , 9pi, http://9fans.net/archive/2012/08/129

39: , Plan 9 announcement, http://www.raspberrypi.org/archives/2665

40: Shead, Sam, Raspberry Pi delivery delays leave buyers hungry (and angry), http://www.zdnet.com/raspberry-pi-delivery-delays-leave-buyers-hungry-and-angry-7000005919/

41: , FreeBSD – Raspberry PI, http://kernelnomicon.org/?p=164

42: , NetBSD 6.0 released with initial Raspberry Pi support, http://www.h-

online.com/open/news/item/NetBSD-6-0-released-with-initial-Raspberry-Pi-support-1731897.html

43: Wikipedia, Raspbian, https://es.wikipedia.org/wiki/Raspbian

44: Raspbian, Welcome to Raspbian, http://www.raspbian.org/

45: Raspbian, Raspbian FAQ, http://www.raspbian.org/RaspbianFAQ

46: elinux, RPi_raspi-config, http://elinux.org/RPi_raspi-config

47: Eben Upton, Introducing the Pi Store, http://www.raspberrypi.org/archives/2768

48: , ROS-Introduction, http://wiki.ros.org/ROS/Introduction

49: ROS Wiki, ROS Installation, 2014-07-12

50: ROS Wiki, ,

51: Mathworks.com, Robot Operating System (ROS) Support from MATLAB - Hardware Support, http://www.mathworks.com/hardware-support/robot-operating-system.html

52: , STanford Artificial Intelligence Robot, http://stair.stanford.edu/

53: Morgan Quigley, Eric Berger, Andrew Y. Ng, STAIR: Hardware and Software Architecture, http://www.aaai.org/Papers/Workshops/2007/WS-07-15/WS07-15-008.pdf

54: ROS.org, Repositories, http://www.ros.org/wiki/Repositories

55: Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew Ng, ROS: an open-source Robot Operating System, http://www.robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf

56: Osrfoundation.org, Osrf - Ros @ Osrf, http://osrfoundation.org/blog/ros-at-osrf.html

57: Willow Garage, employees join Suitable Technologies, http://www.willowgarage.com/blog/2013/08/21/willow-garage-employees-join-suitable-technologies

58: Robotics Corner, Clearpath Welcomes PR2 to the Family, http://www.clearpathrobotics.com/blog/clearpath-welcomes-pr2/

59: wiki.ros.org, publish/subscribe anonymous message passing, http://wiki.ros.org/Topics

60: wiki.ros.org, mensaje IDL, http://wiki.ros.org/Messages

61: wiki.ros.org, recording and playback of messages, http://wiki.ros.org/rosbag

62: wiki.ros.org, request/response remote procedure calls, http://wiki.ros.org/Services

63: wiki.ros.org, distributed parameter system, http://wiki.ros.org/Parameter%20Server

64: wiki.ros.org, Standard Message Definitions for Robots, http://wiki.ros.org/common_msgs

65: wiki.ros.org, Robot Geometry Library , http://wiki.ros.org/tf

66: wiki.ros.org, ROS TF, http://wiki.ros.org/tf

67: wiki.ros.org, Robot Description Language, http://wiki.ros.org/robot_model

68: wiki.ros.org, robot_state_publisher, http://wiki.ros.org/robot_state_publisher

69: wiki.ros.org, rviz, http://wiki.ros.org/rviz

70: wiki.ros.org, URDF, http://wiki.ros.org/urdf

71: wiki.ros.org, Preemptable Remote Procedure Calls, http://wiki.ros.org/actionlib

72: wiki.ros.org, Diagnostics , http://wiki.ros.org/diagnostics

73: wiki.ros.org, Pose Estimation, http://wiki.ros.org/robot_pose_ekf

74: wiki.ros.org, Localization, http://wiki.ros.org/amcl

75: wiki.ros.org, Navigation, http://wiki.ros.org/navigation

76: wiki.ros.org, rqt, http://wiki.ros.org/rqt

77: Sony Computer Entertainment America, PLAYSTATIONEye Brings Next-Generation Communication to PLAYSTATION 3, http://www.us.playstation.com/News/PressReleases/396

78: Croal, N'Gai, The Playstation Eye is Nearly Upon Us. Dr. Richard Marks

Takes Us Behind the Scenes of its Birth.,
http://blog.newsweek.com/blogs/levelup/archive/2007/04/30/geek-out-the-playstation-eye-is-nearly-upon-us-dr-richard-marks-takes-us-behind-the-scenes-of-its-birth.aspx

79: , PlayStation Eye - Q+A, http://threespeech.com/blog/?p=394

80: Itseez leads, the development of the renowned computer vision library OpenCV, http://itseez.com/

81: Maimone, M.; Cheng, Y.; Matthies, L., Two years of Visual Odometry on the Mars Exploration Rovers, http://www-robotics.jpl.nasa.gov/publications/Mark_Maimone/rob-06-0081.R4.pdf

82: http://math.stackexchange.com/users/6622/joriki, Decomposition of a nonsquare affine matrix, http://math.stackexchange.com/a/78165

83: , video_homography.cpp,
https://openlab.ncl.ac.uk/gitlab/john.shearer/clappertracker/blob/d788483f5609302a8602027d1037f7c496d745f1/src/3rdparty/opencv/samples/cpp/video_homography.cpp