PROYECTO FIN DE GRADO:

DISEÑO DE UN SISTEMA DE CONTROL PARA UN CUADRICÓPTERO

Autor:

Juan Carmona Fernández

INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Tutor: José María Armingol Moreno

Departamento de Ingeniería de Sistemas y Automática

Universidad Carlos III de Madrid

Febrero 2013

PROYECTO FIN DE GRADO:

DISEÑO DE UN SISTEMA DE CONTROL PARA UN CUADRICÓPTERO

Autor:

Juan Carmona Fernández

INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA.

Tutor: José María Armingol Moreno

Departamento de Ingeniería de Sistemas y Automática

Universidad Carlos III de Madrid

Febrero 2013

AGRADECIMIENTOS:

A mi familia,

a los integrantes del laboratorio LSI por el apoyo recibido.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ABREVIATURAS

SO: Operating System, sistema operativo.

ROS: Robot Operating System, sistema operativo para robots.

UAV: Unmanned Aerial Vehicle, vehículo aéreo no tripulado.

UAS: Unmanned Aircraft System, vehículo aéreo autónomo.

UCAV: Unmmaned Combat Air Vehicle, vehículo aéreo no tripulado de combate.

MAV: Micro Aerial Vehicle, micro vehículo aéreo.

VFR: Visual Flight Rules, normas emitidas en el Reglamento de Circulación Aérea que establecen las condiciones necesarias para realizar un vuelo cuya principal herramienta, es la observación visual.

HALE: High Altitude Long Endurance, rango de vuelo en aeronaves, el vuelo se realiza por encima de los 30,000 ft (9,100 m), alta altitud, y con infinito rango, por tanto, pueden utilizarse en vuelos de larga distancia.

Capítulo 1 : INTRODUCCIÓN

En este proyecto partimos de un ardrone diseñado para una navegación manual, a través de iOS o Android con una sencilla aplicación, que en algunos casos es capaz incluso de tomar foto o video.

La idea principal de este proyecto es aprovechar el cuadricóptero ardrone 2.0, diseñado por la empresa francesa Parrot, y usar su cámara de alta definición integrada en el frontal, para la navegación autónoma.

Otra característica importante, por la cuál ha sido elegido este modelo de cuadricóptero, es una característica de conexión, la cual hace posible una integración con otros sistemas operativos para los que, en un principio, no tiene soporte. Su conexión Wi-Fi, es la conexión que utiliza para conectarse con el propio Smartphone, y recibir los datos de navegación a través de la misma.

Aunque existen multitud de aplicaciones, como juegos o grabaciones de videos, a través de los SO iOS y Android ya mencionados, todos tienen como denominador común la necesidad de un piloto para realizar el vuelo, que maneja el aparato desde el Smartphone.

La idea principal es la de dar compatibilidad a través de otro sistema operativo. En este caso, elegimos Linux en concreto Ubuntu, distribución basada en Debian, por su capacidad de desarrollo de aplicaciones, y la compatibilidad con ROS.

Después de tener una compatibilidad con Ubuntu, la dirección del proyecto ha seguido por la vía de tratamiento de imagen, para poder realizar una navegación autónoma del vehículo aéreo.

1.1 Objetivos

El presente proyecto tiene como fin, el diseño de un sistema basado en ROS, para poder realizar una navegación autónoma del vehículo aéreo ardrone 2.0.

La idea del proyecto surge al realizar pruebas de estabilidad a este modelo. El cuadricóptero posee un sistema de estabilización que hace una herramienta útil para la visión aérea a corta distancia de zonas de difícil, o peligroso acceso. Las cámaras integradas pueden realizar una tarea de vigilancia, o como medio para el análisis sobre el terreno desde una perspectiva más amplia, al ser aérea.

Las capacidades de un sistema aéreo de pequeño tamaño y gran estabilidad, como es un cuadricóptero, son múltiples. Desde una vigilancia de un terreno delimitado, o la búsqueda de personas en un terreno escarpado o tras un desastre natural. Incluso, a nivel militar se puede utilizar de avanzadilla para reducir el peligro gracias al conocimiento del terreno a través de la visión del drone.

Analizadas todas las posibles líneas de desarrollo, el proyecto se centra en la navegación autónoma. Esta navegación se basa en la recepción y tratamiento de las imágenes, aplicando los métodos de la visión por computador, vistos en [1]. La navegación se realiza con el envío de comandos de navegación, en tiempo real, emitidos en base al tratamiento de imágenes. Esto convierte al ardrone en un vehículo autónomo.

1.2 Elección del hardware: ardrone 2.0

El ardrone 1.0 fue la primera versión de este vehículo aéreo con fin recreativo. Como primera versión, no tiene gran diferencia visualmente, tampoco en el sistema de estabilización, en cambio, sí en el número de sensores y la capacidad de precisión de los mismos, incluidas las cámaras que en el proyecto las llegamos a utilizar como un sensor más.

La cámara principal utilizada para este proyecto, es una cámara frontal capaz de recoger imágenes en alta definición a 720p a 30 fps. Esto hace disponer de una calidad de imagen suficientemente buena como para recoger la información y procesarla para el vuelo autónomo.

Además otra de las ventajas, respecto a otros modelos de cuadricópteros, es que se distribuye en cualquier comercio especializado o gran superficie, haciendo fácil el acceso a este modelo de vehículo aéreo.

Ilustración 1. Cuadricóptero en vuelo

Capítulo 2 : ESTADO DEL ARTE

2.1 Vehículo aéreo no tripulado, UAV

Las siglas UAV corresponde en inglés a Unmanned Aerial Vehicle, es decir, vehículo aéreo no tripulado. El origen del desarrollo de estos vehículos pertenece a fines militares, ya que dan la posibilidad de realizar operaciones de alto riesgo, o incluso la de sobrevolar un zona en conflicto para la vigilancia o recogida de información.

Vista la configuración aerodinámica, de apariencia novedosa, que se lleva trabajando en estos particulares sistemas desde hace años, sin embargo, en estos últimos años es cuando su pendiente desarrollo ha sido más exponencial, llegando incluso algunos de ellos a utilizarse de forma recreativa.

Existen una gran variedad de UAVs, desde vehículos de grandes dimensiones para altos vuelos y/o grandes distancias de vuelos, hasta pequeñas dimensiones llegando incluso a pocos centímetros.

El desarrollo de un UAV se basa en el tipo de acción que va a llevar a cabo, y la distancia que tendrá que recorrer. Además de dimensiones específicas, necesita de una tecnología de transmisión y vuelo necesaria para realizar la tarea.

En este punto es necesario aclarar que vehículo no tripulado, UAV, no tiene un significado autónomo, sino que estará comunicado desde un operador de tierra, sean pilotos, controladores o cualquier otro tipo de operario relacionado con la monitorización de la aeronave. Una vez superado el reto de la creación de vehículos no tripulados, se investigó en otro nivel los llamados UAS, Unmanned Aircraft System. Un UAS se trata de la evolución directa de un UAV.

Los UAVs pueden estar controlados remotamente desde una estación de tierra por un operador, en cambio, los UAS son autónomos y seguirán una trayectoria ya predefinida, o un vuelo con los recursos de los propios sensores. Existen dos estaciones que pueden manejar información del UAV, la estación de tierra y la estación a bordo del UAV. Dependiendo de cuan autónomo sea el UAV, la estación de tierra realizará más o menos funciones de forma habitual.

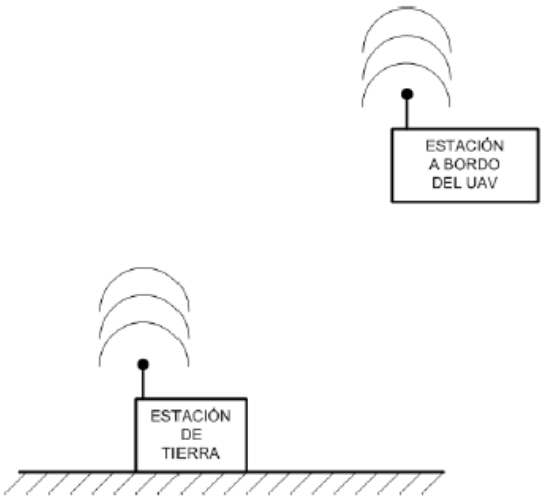

Ilustración 2. Estaciones UAV

Además convendría añadir que no todo vehículo no tripulado que se encuentre en el aire se considera un UAS, los misiles no son considerados como tales, incluso aunque posea un data link o utilice una tecnología parecida a los UAS. Esto es debido a que los misiles no realizan un vuelo de retorno, son desechables.

2.2 Modelos de vehículos UAV.

Tanto en la empresa privada, como a nivel público más con fines militares, véase [2], se han desarrollado multitud de modelos de vehículos no tripulados para diferentes tareas los más actuales incluso de pequeñas dimensiones para un uso recreativo. Los tipos de aeronaves, son desde aviones, helicópteros o incluso dirigibles.

Más que por su uso, porque en algunos casos puede tener varias áreas de uso, podemos clasificar por su configuración aerodinámica, aviones, helicópteros o dirigibles. Y dentro del sector de los aviones, por su tipo de motor que nos da pistas de cuál va a ser su uso, en el aspecto de corto o largo alcance.

2.2.1 Aviones UAV con motor a reacción.

En uso militar predominan los UAV de tipo avión, con motores de propulsión a reacción o de pistón. Dentro de los UAV, existe otra denominación para un uso en combate, que es UCAV, Unmmaned Combat Air Vehicle, denominados vehículos aéreos no tripulados de combate.

Un ejemplo de UCAV propulsado con reactores, es el Barracuda proyectado conjuntamente entre España y Alemania, a través de EADS.

Ilustración 3. Barracuda

Con el mismo fin de desarrollo se encuentra el X-45, creado por la empresa Americana Boeing fue parte del proyecto J-UCAS de DARPA.

Ilustración 4.X-45.

2.2.2 Vehículos UAV con motor de pistón.

UAV con propulsión de tipo pistón se encuentra Predator, sirve principalmente en misiones de reconocimiento, pero además, tiene capacidad ofensiva con la posibilidad de incorporarle dos misiles. En la siguiente imagen además del propio avión podemos ver parte de la cabina de control de tierra, compuesta en su totalidad por una plantilla de 55 personas.

Ilustración 5. Predator.

Otro modelo de un avión propulsado por hélices es el Herti de Bae Systems, este tiene como fin un ámbito militar pero también civil. La principal misión de este vehículo es la de reconocimiento.

Ilustración 6. Herti.

2.2.3 UAV con configuraciones distintas.

UAV de tipo dirigible operado por la empresa Blackwater, empresa de ámbito privada, innovo con un vehículo dirigible no tripulado, como es el de la siguiente imagen.

Ilustración 7. Blackwater.

UAV tipo helicóptero convencional, válido tanto para misiones de uso civil o de uso militar, es el Boenig A160Hummingbird, de origen Americano, se diseñó con tecnologías avanzadas que dotan a este helicóptero para vuelos en altitudes altas.

Ilustración 8. Hummingbird.

2.3 Vehículos no tripulados de peso reducido.

El más novedoso es el Honeywell RQ-16A T-Hawk, desarrollado por Honeywell**,** [3]**,** es un pequeño UAV con un motor de gasolina propulsado por un único ventilador colocado en su centro. Su fin es la vigilancia y reconocimiento, este modelo no dispone de armas de defensa u ofensiva, pero tiene un largo alcanzo en proporción a su peso de solo 8.3 Kg. Su reducido peso y tamaño hace que entre dentro de la denominación MAV, del inglés, Micro Aerial Vehicle.

Su creación surgió gracias a la agencia DARPA, responsable de la investigación en el ámbito militar, y actualmente se sigue utilizando en Afganistán por el ejército de los EEUU. Al ser un vehículo no tripulado, no supone riesgo en labores de reconocimiento.

Ilustración 9. T-Hawk

En el contexto del desarrollo de vehículos no tripulados, en su mayoría para un fin militar, tanto de grandes dimensiones, o de pequeñas dimensiones como el RQ-16 T-Hawk, aparecen numerosos modelos para un uso recreativo. Se distribuyen multitud de modelos que se pueden calificar, al igual que los anteriores, en su propulsión o configuración aerodinámica.

Pero el más innovador, fue el Ardrone 1.0 de Parrot, empresa francesa que lo desarrolló y distribuye, con una configuración de cuadricóptero con motores eléctricos.

Ilustración 10. Ardrone 1.0

Se diferencia de otros modelos ya que posee un microprocesador competente para el procesamiento de las señales recibidas a través de una serie de sensores. Además incluye dos cámaras que le permiten captar lo que ocurre a su alrededor. Incorpora un cambió en la estación de control, ya que necesita de un Smartphone para el control del mismo.

Posee una conexión Wi-Fi, y tiene un techo de altitud limitado por la propia conexión Wi-Fi. Debido a la necesidad de una estación de control, también pertenece, en su versión sin modificaciones, al sector de los UAV.

De nuevo, se debe aclarar que un UAV no tiene una connotación de vehículo autónomo, sino que es solo un vehículo tripulado. Tras esto debido a la integración de dos cámaras y otros sensores podemos hacer de este UAV forme parte del grupo de UAS, autónomos.

Existe un gran avance en el tratamiento de imagen. Recibida la imagen por las cámaras del cuadricóptero hacen posible, gracias también al SDK que distribuye gratuitamente Parrot, véase [4], una integración de un sistema de control para el vehículo autónomo.

Varias universidades, en su mayoría de origen Estadounidense, han realizado avances en este sector.

La Universidad con mayor avance en la versión 1.0 del Ardrone, es la Brown University, [5], situada en Providence, EEUU. Han desarrollado un software con las cámaras que albergaba este modelo la utilización para el seguimiento de una imagen, o incluso la utilización de este robot como vehículo aéreo de reconocimiento para el movimiento de robots terrestres.

Pero con la llegada de un nuevo modelo de este cuadricóptero, el Ardrone 2.0, con diferentes cámaras y sensores, ha sido necesario un nuevo modelo de sistema de control para este novedoso modelo.

Ilustración 11. Ardrone 2.0

La versión 2.0 es más potente en todas sus características, pero que mantiene la esencia de su predecesor con la conexión Wi-Fi, la distribución de sus dos cámaras y las pequeñas dimensiones.

El reto de este proyecto, era el de realizar a través de un software en código abierto, en este caso ROS, un sistema de control con el uso de tratamiento de imagen para hacer de esta versión 2.0, más reciente, un vehículo no tripulado, UAV, pero además sin depender de una estación de control. El núcleo del proyecto es la evolución de un UAV comercial, en un UAS, capaz de realizar un vuelo y ser autónomo añadiendo control visual.

2.4 Control visual

Actualmente, el uso de cámaras no solo es válido para la toma de imágenes o realizar vídeos, sino que el campo de la visión ha avanzado mucho, y hoy en día es usado como un sensor más a la hora de obtener información del entorno.

La colocación de las cámaras es un importante punto de estudio, a causa de que condicionará desde que punto recibirá el vehículo la información de visión. También es importante el tipo de cámara elegida, monocromática, a color, de infrarrojos, son algunos tipos de cámaras.

Las cámaras pueden estar colocadas sobre el propio vehículo, cámara a bordo, o estar distribuidas por la zona de operación del mismo. Esta última opción será viable únicamente en entornos para realizar pruebas, entornos académicos, donde se vayan ensayos en lugares cerrados, ya que si el UAV tiene que realizar una misión de reconocimiento la posición inicial del vehículo es conocida.

En Suiza, se ha realizado un trabajo uniendo el ámbito del vuelo con el control por visión en el IDSC, Institute for Dynamic Systems and Control, [6], donde a través de la visión desde el techo de 8 cámaras se obtiene una precisión milimétrica para el vuelo. Gracias a la integración del control por visión de estas 8 cámaras, un cuadricóptero es capaz de pasar por el marco de una ventana colocada en mitad de la sala a una alta velocidad. Existen otros trabajos de utilización de cuadricópteros con un control visual, algoritmos para navegar en formación usando técnicas de trayectorias libres de colisión.

Otra de las áreas de investigación es la utilización de sistemas como Kinect, de Microsoft para el manejo de vehículos aéreos, estos no convierten al vehículo en un sistema autónomo, pero realizan una mejora en la navegación al llevar un control más intuitivo para la estación de control.

Gracias a las librerías OpenCV, podemos realizar un reconocimiento de patrones capaz de evolucionar un vehículo no tripulado, UAV, a un vehículo no tripulado autónomo. Para el desarrollo en OpenCV véase [7] y [8].

2.5 Normativa UAV

Sistemas aéreos no tripulados como son cometas, pequeños cohetes, aviones r/c, y globos meteorológicos poseen una regulación propia, aunque está no pertenece a los UAV. Actualmente los UAV están operando en espacio aéreo segregado, para operaciones militares.

Todas las vías de desarrollo que se han investigado en el presente proyecto en el apartado *Trabajos futuros*, necesitan de una normativa para integrar de manera segura los UAV en el espacio aéreo civil. Estas aplicaciones pueden interesar incluso a los propios gobiernos ya que podrían ser misiones de vigilancia de fronteras, recogida datos o patrulla marítima.

Los pioneros en la regulación sobre el espacio aéreo son Suiza, Reino Unido o Australia, legislan las operaciones aéreas mediante normas nacionales. En otros países las operaciones de cualquier UAV requiere de una autorización específica por parte de las autoridades aeronáuticas.

Los principales organismos de gestión y control del tráfico aéreo son, FAA (Estados Unidos) y EASA (Europa) están realizando estudios sobre cómo integrar las aeronaves no tripuladas en sus respectivos espacios aéreos, y por tanto establecer unos mínimos sobre los que certificar estas aeronaves.

La FAA, en conjunción con la Universidad del MIT, estudia dirigir la regulación de las aeronaves no tripuladas realizando estudios de tamaño, masa, probabilidad de fallo o probabilidad de impacto contra aeronaves tripuladas.

Según estos estudios, las aeronaves inferiores en masa 14kg, donde estaría el ardrone 2.0, serían reguladas mediante una serie de normas para el uso civil. Estas normas, serían similares a las reglas seguidas por los aficionados de R/C, no registradas como leyes, y sí como normas de clubes de R/C, basadas en consejos de la FAA. Estas normas, establecen cotas de operación muy bajas,

mantener siempre el contacto visual con la aeronave y se restringe su uso fuera del espacio aéreo controlado, lejos de zonas “problemáticas” como sendas de aproximación y despegue de aeropuertos.

Para los UAV de peso superior establecen unas normas homogéneas a las ya existentes para el control del espacio aéreo, aplicando las mismas normas que para aeronaves tripuladas que operan en VFR, como son los ultraligeros. Otra opción sería la de operaciones de UAS del tipo HALE, que vuelan a una cota superior al espacio aéreo que usan los vuelos comerciales.

Por otro lado en Europa, se está trabajando para que UAS de gran tamaño puedan utilizar el espacio civil en 2016. La Unión Europea trabaja para el desarrollo de las aplicaciones civiles de los Sistemas Aéreos Pilotados Remotamente.

Las única regulación existente son las normas de conductas emitidas por las asociación AUVSI, Asociación Internacional de Vehículos no Tripulados, su sitio web [9]. Estas normas buscan realizar unas directrices para concebir seguridad, y acelerará la confianza pública en estos sistemas.

Las normas emitidas por la AUVSI son las siguientes:

Seguridad

- No operaremos UAS de forma que represente riesgo para las personas o propiedades en la superficie o en el aire.
- Aseguraremos que los UAS serán pilotados por personas que están debidamente entrenadas y tienen competencias para operar vehículos o sus sistemas.
- Aseguraremos que los vuelos de los UAS serán realizados solo después de una rigurosa valoración de los riesgos asociados con la actividad. Esta valoración de riesgos, incluirá, pero no limitará.
- Condiciones climáticas en relación con la capacidad de los sistemas.

- Identificación de normalidad de modos de fallo anticipado (perdida de enlace, fallos de potencia, pérdida de control, etc.) y las consecuencias de los fallos.
- Condiciones físicas de la tripulación para operar el vuelo.
- Cumplimiento de las regulaciones de aviación y de la apropiada operación en el espacio aéreo y procedimientos no nominales.
- Comunicación, comando, control y requisitos de espectro de frecuencia del palead (carga útil).
- Fiabilidad, rendimiento y aeronavegabilidad mediante estándares establecidos.

Profesionalidad

- Cumpliremos con las leyes nacionales, estatales y locales, ordenanzas, acuerdos y restricciones relativos a las operaciones de UAS.
- Operaremos nuestros sistemas como miembros responsables de la comunidad aeronáutica.
- Seremos sensibles a las necesidades de las personas.
- Cooperaremos totalmente con las autoridades nacionales, regionales y locales en el despliegue en respuesta a emergencias, investigación de accidentes y relaciones con los medios.
- Estableceremos planes de contingencia para todos los eventos anticipados y los compartiremos abiertamente con todas las autoridades pertinentes.

Respeto

- Respetaremos los derechos de otros usuarios en el espacio aéreo.
- Respetaremos la privacidad de las personas.
- Respetaremos los asuntos del público así como los relativos a las operaciones de los UAS.
- Apoyaremos la mejora del conocimiento y educación pública sobre las operaciones de los UAS.

Capítulo 3 : HADWARE Y SOFTWARE DEL SISTEMA

3.1 Ardrone 2.0

Ilustración 12. Ardrone 2.0 outdoor.

La principal herramienta utilizada para el desarrollo de este proyecto es el ardrone 2.0. Desarrollado y distribuido por Parrot, se trata de un vehículo aéreo no tripulado propulsado por cuatro motores eléctricos en configuración de cuadricóptero. Fue lanzado al mercado en Agosto de 2012, tras una revisión y actualización de la primera versión de este vehículo.

Creado para uno uso recreativo civil, es similar a otros vehículos radiocontrol de la categoría de cuadricópteros, pero la principal diferencia es que incluye una serie de sensores, y un microprocesador avanzado. Entre los sensores, cuenta con acelerómetros y dos cámaras integradas, una vertical, y la frontal renovada cuenta con una recepción de video en alta definición.

Otra de sus características, es que no dispone de un mando físico por radiocontrol para su manejo, si no que se vincula a través de su conexión Wi-Fi a un dispositivo móvil con los sistemas operativos iOS de Apple, o Android de Google. Esta conexión le permite ser controlado por los dispositivos móviles e incluso ser controlado a través de la recepción de las imágenes y datos de telemetría obtenidos por el propio ardrone.

Para el uso regular, se necesita tener instalado una aplicación en el Smartphone para el manejo del cuadricóptero. Estas aplicaciones oficialmente solo están desarrolladas para iOS o Android, ya que, en la actualidad, son los principales sistemas operativos, a nivel de cuota de mercado, en dispositivos móviles. Pero debido a que la empresa Parrot público el control a nivel bajo han aparecido aplicaciones no oficiales en otros sistemas operativos.

Ilustración 13. Ardrone comercial

3.2 Especificaciones técnicas del hardware

El núcleo de este ardrone, versión 2.0, es un microprocesador 1GHz de 32 bit ARM Cortex A8 con 800MHz. También dispone de una memoria DDR2 RAM de 1Gb. El sistema operativo introducido es Linux en su versión 2.6.32. Dispone de una conexión USB 2.0 de alta velocidad, para extensión con una memoria extraíble. El microprocesador dispone de un conector Wi-Fi b, g, n.

Ilustración 14. Placa madre ardrone 2.0

Gracias a estas altas características de procesamiento para un vehículo aéreo recreativo, es una buena herramienta para un desarrollo más allá de un pilotaje básico. Además, disponemos de varios sensores que abren el camino de desarrollo para un pilotaje autónomo.

3.3 Sistema de propulsión

El ardrone 2.0 es un vehículo aéreo con un movimiento físico propulsado por cuatro motores eléctricos basado en una estructura aerodinámica en configuración de cuadricóptero.

Ilustración 15. Motor ardrone 2.0

Los cuatro motores de rotor interno sin escobillas, hacen que el dispositivo tenga una alta velocidad de reacción en el aire y pueda, dentro de su categoría de vuelo, ascender a alta distancia. La fuente de alimentación de la que dispone es una batería de tres celdas de Litio-Polímero con capacidad de 1000mAh a 11.1V y capacidad de descarga de 10C, cumple con la normativa de seguridad UL2054, [10]. El cumplimiento con la norma UL2054 hace viable una evolución del aparato hacia un vuelo de mayor longitud y tiempo.

Estos cuatros motores son los responsables, a nivel de hardware, para que se mantenga en el aire, con la ayuda de los comandos a nivel de software emitidos por el sistema de estabilización.

3.4 Sensores

En el desarrollo de este proyecto para un pilotaje autónomo, se ha basado en la recepción de la señal de video y los datos de telemetría. Por tanto, la elección de este dispositivo se ha basado en los datos de sensores disponibles para la navegación.
Centrándonos en las características técnicas de los sensores, dispone de dos cámaras integradas:

- La principal, una cámara frontal en alta definición de 720p a 30 fps.

Ilustración 16. Cámara frontal HD

- La segunda cámara, menos potente, es QVGA de 60 fps y con una colocada de forma vertical con orientación descendente.

Esta última además de poder ser utilizada para la recepción de imagen, su característica principal es la medición de la velocidad respecto del suelo. A parte de las cámaras, disponemos de una serie de sensores útiles para el pilotaje del aparato y las principales herramientas para el sistema automático de estabilización.

- Para el control de altitud integra un sensor de ultrasonidos, este genera datos de telemetría en función de los cambios de altitud. Sensor de presión con una precisión de ± 10 Pa.

- Acelerómetro digital de 3 ejes con precisión de ±50mg para monitorizar los movimientos de posición.

- Giróscopo de 2 ejes y giróscopo piezoeléctrico de precisión de 2000º/s para los movimientos de cabeceo, giro y viraje.

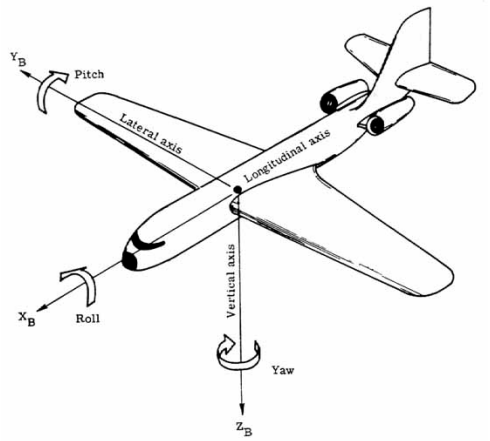

Ilustración 17. Cabeceo, giro y viraje

- Magnetómetro de 3 ejes con 6º de precisión con fines de orientación

El conjunto de estos sensores proporcionan toda la información al ardrone 2.0 para la navegación. La fusión de los datos de los sensores proporciona los ángulos de Euler, utilizados para la estabilización.

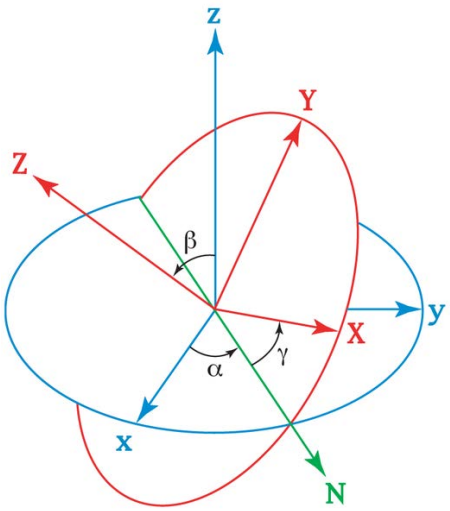

Ilustración 18. Ángulos de Euler.

3.5 Introducción al software

Elegido el desarrollo a través del software ROS, de sus siglas en inglés, Robot Operating System, las cuales, nos dejan ver que ROS es un sistema operativo, debemos añadir que no es solo un sistema operativo, es un meta-sistema operativo.

Permite introducir librerías y herramientas para ayudar al desarrollo del software de aplicaciones en el campo de la robótica. Gracias, entre otras cosas, a la abstracción de hardware, los controladores de dispositivo, bibliotecas, visualizadores, paso de mensajes, o gestión de paquetes son algunas de las funciones que nos permite.

Debido a la falta de normativa para el vuelo de estos aparatos, micro UAV, al cual pertenece el ardrone, véase *Normativa UAV*. Hemos centrado la elección de la vía de desarrollo del sistema, en base a la licencia libre del software.

Actualmente ROS está bajo licencia de código abierto, la licencia BSD, cuyos términos se encuentran en [11]. La licencia BSD tiene menos restricciones en comparación con otras como GPL, estando muy cercana al dominio público. Permite el uso del código fuente en software no libre.

3.6 ¿Qué es ROS?

ROS es un meta-sistema operativo de código abierto para el desarrollo en el sector de la Robótica. Provee diferentes servicios que se esperarían de un sistema operativo, como abstracción de hardware, control de dispositivos en bajo nivel, funciones que se usan comúnmente, comunicación de procesos mediante mensajes y administración por paquetes.

Pero decimos que ROS es un meta-sistema operativo porque además, también comparte características con algunos sistemas middleware (acoplamiento clasificación y envío por el paso de mensajes) y frameworks (callbacks).

Comparando ROS con la mayoría de sistemas middleware y frameworks, ROS impone una leve política restrictiva en el desarrollador, tanto en términos de API, como en problemas de licencia.

La comunidad de este meta-sistema operativo aumenta dado que la curva de aprendizaje no es demasiado alta, y la mayoría del código puede ser fácilmente trasladado, una vez que se entiende lo básico ROS. Para ello la web oficial, dispone de tutoriales, véase [9].

Ilustración 19. Relación nodos y topics

ROS nos proporciona herramientas y librerías que permiten obtener, construir, escribir y ejecutar programas entre varios computadores, como se ha definido en el apartado anterior, es similar en algunas características con otros frameworks usados en robótica, tales como Player, Orocos, Yarp, Orca, Microsoft Robotics Studio, entre otros. Sin embargo, ROS no es un framework en tiempo real, aunque sí es posible integrarlo con código en tiempo real, por ejemplo el robot PR2 de Willow Garage, véase [12], usa un sistema que se encarga de transportar mensajes en tiempo real.

3.7 Objetivo de ROS

El objetivo de ROS es el de implementar todas las conexiones a nivel bajo, para acelerar el proceso de desarrollo, e incluso poder portar las aplicaciones a otros robots gracias a su sistema general de mensajes y nodos. La posibilidad de portar

las aplicaciones es una gran ventaja, ya que avances que se realicen para el ardrone, pueden funcionar con otros dispositivos.

Entre otras ventajas tenemos:

Independencia de idiomas: es fácil de implementar en cualquier lenguaje de programación moderno, el principal es C++. Aunque, ya se ha implementado en Python, C + +, y Lisp, y existen bibliotecas experimentales en Java y Lua.

Test fácil: ROS posee un test, llamado rostest que hace que sea fácil de someter a pruebas.

Escala: ROS es adecuado para sistemas pequeños, pero también, para sistemas grandes y ejecución de los procesos de desarrollo de gran tamaño.

3.7.1 Áreas de aplicación de ROS.

Algunas áreas de uso de ROS son:

- Publicación o subscripción de flujos de datos: imágenes, estéreo, láser, control, actuador, contacto, etc.
- Multiplexación de la información.
- Testeo de sistemas.
- Percepción
- Identificación de Objetos.
- Segmentación y reconocimiento.
- Reconocimiento facial.
- Reconocimiento de gestos
- Seguimiento de objetos.
- Comprensión de movimiento
- Visión estéreo: percepción de profundidad.
- Robots móviles
- Agarre de objetos

3.8 Entorno de ROS

ROS ofrece una infraestructura de publicación-suscripción de mensajes diseñado para apoyar la construcción rápida y fácil de los sistemas de computación distribuida.

Herramientas: proporciona un amplio conjunto de herramientas para configurar, iniciar, introspección, depuración, visualización, registro, control, y detener los sistemas de computación distribuida.

Capacidades: ofrece una amplia colección de bibliotecas que implementan la funcionalidad de robot útil, con especial atención a la movilidad, la manipulación y la percepción.

Ecosistema: ROS es apoyado y mejorado por una comunidad grande, con un fuerte enfoque en la integración y la documentación. El sitio web lo encontramos en [13].

3.9 Conceptos generales de ROS

Para entender el sistema ROS debemos conocer unos sencillos conceptos propios de este sistema, documentados en [14]. Aunque ROS actualiza anualmente la versión del software estos conceptos no cambian, manteniendo su estructura.

ROS posee tres niveles de conceptos:

- Nivel del sistema de archivos.
- Nivel de Computación Gráfica.
- Nivel comunitario

3.9.1 ROS nivel del sistema de archivos

En este nivel nos referimos a la totalidad de archivos necesarios para el funcionamiento de ROS, no nos referimos al sistema de conexiones o envío de mensajes a nivel interno.

Paquetes: unidad principal en ROS, puede contener procesos de ejecución (nodos), una biblioteca ROS-dependiente, conjuntos de datos, archivos de configuración, o cualquier otro archivo que se útil para la organización de ROS.

Manifiestos: residentes en manifest.xml proporcionan datos sobre un paquete, incluyendo su información de licencia y dependencias, así como información específica del idioma.

Pilas: Pilas o stacks son colecciones de paquetes que proporcionan funcionalidad agregada, como una "stack de navegación".

Manifiestos de pila o stack: se trata del archivo stack.xml, proporcionan datos sobre una pila, incluyendo su información de licencia y sus dependencias en otras pilas.

3.9.2 Computación en ROS, nivel gráfico

El gráfico de la computación es la red peer-to-peer de los procesos de ROS que están procesando los datos conjuntamente. Se trata de un gráfico bastante útil en el desarrollo, ya que de una forma sencilla, podemos observar las conexiones internas en ROS.

Los conceptos básicos de computación gráfica en ROS, nos proporcionan los datos de la gráfica de diferentes maneras. Son los siguientes:

Nodos: Los nodos son procesos que llevan a cabo cálculos. Por ejemplo, en el caso de un sistema de control de un robot, en nuestro para el ardrone 2.0, un nodo se encargara del movimiento de navegación, otro de la búsqueda de patrones, y además dispondremos de varios nodos para la navegación autónoma. Los nodos publican o se suscriben a topics.

Master: El master ROS proporciona registro de nombres y la búsqueda para los nodos. Sin los nodos no sería capaz de encontrar mensajes entre sí, intercambio, o invocar los servicios.

Parámetro servidor: El servidor de parámetros permite que los datos se almacenarán con llave en un lugar central. En la actualidad es parte del Master.

Mensajes: los nodos se comunican entre sí por el paso de mensajes. Un mensaje es simplemente una estructura de datos, que comprende los campos con tipo estándar (entero, flotante, booleano, etc.).

Topics o temas: Los mensajes toman rutas a través de un sistema de transporte de publicación o suscripción semántica. Un nodo envía un mensaje mediante su publicación a un tema determinado. El tema es un nombre que se utiliza para identificar el contenido del mensaje. Un nodo que está interesado en un determinado tipo de datos se suscribirá al tema correspondiente. Puede haber varios editores y suscriptores concurrentes a un mismo tema, y un único nodo puede publicar y / o suscribirse a múltiples temas.

Lógicamente, se puede pensar en un tema como un Bus de mensajes. Cada Bus tiene un nombre, y cualquier persona puede conectarse al Bus para enviar o recibir mensajes, siempre y cuando sean del tipo correcto. En general, los editores y suscriptores no son conscientes de la existencia de los demás. La idea es disociar la producción de información a partir de su consumo.

Esta arquitectura es común a cualquiera programa desarrollado en ROS, esto hace posible que, con algunos cambios, sea portable a diferentes modelos de robot, o incluso la ejecución de varios robots bajo ROS, en el mismo momento.

Un ejemplo de portabilidad, es el paquete básico camera_calibration, podemos encontrarlo en la red en [15], diseñado para una cámara USB, con unos determinados cambios de configuración se ha podido calibrar la cámara de este proyecto, en el ardrone vía Wi-Fi.

3.9.3 ROS a nivel comunitario

ROS permite el intercambio de recursos o información, entre grupos de investigación o universidades, proporcionando vías para el intercambio, algunas fuentes de intercambio son:

- Repositorios [16]: ROS se basa en una red de repositorios de código, donde diferentes instituciones, como programadores o laboratorios, pueden desarrollar y lanzar sus propios componentes de software del robot.
- El Wiki ROS [17]: ROS comunidad Wiki es el foro principal para documentar la información sobre ROS. Cualquier persona puede inscribirse para una cuenta y contribuir con su propia documentación, facilitar las correcciones o actualizaciones, escribir tutoriales véase [18].
- ROS answers [13]: respuestas a preguntas y respuestas lugar de responder a sus preguntas relacionadas con ROS. No se trata de un foro, sino que es un sistema sencillo de pregunta y respuesta.
- Blog: multitud del blogs destinados al desarrollo de aplicaciones y tutorial de ROS el más importante el blog de Willow Garage, [12], ofrece actualizaciones periódicas..

Capítulo 4 : DISEÑO DE SISTEMA DE CONTROL PARA UN CUADRICÓPTERO.

4.1 Software de desarrollo del sistema de control: GostaiLab

La primera meta a tratar en este proyecto, fue la de encontrar un sistema de recepción y envío de datos basado en Linux, también debía estar en código abierto, además de una licencia de restricción de investigación baja, como puede ser la licencia LSB.

A través de la red se han encontrado algunos programas o entornos con un desarrollo más rápido, ya que no es necesario centrarse en el nivel bajo para el desarrollo del mismo. Como es Gostai Lab, basado en el entorno de LabView nos permite comunicarnos y desarrollar herramientas para el ardrone, al ser una herramienta de LabView permite un uso en distintos sistemas operativos como Windows, Mac OS o Linux.

Se puede comprar el la propia página de Gostai, [19] , el principal problema es que para empezar a desarrollar es necesaria la compra de una licencia, que para esta herramienta es superior a los 1000€.

En pocos minutos, es posible manejar el ardrone sin necesidad de la aplicación oficial para el control a través de un Smartphone, sino que se realiza con un ordenador, y es capaz de recoger datos de los distintos sensores.

En la siguiente figura podemos observar como recoge la señal de la cámara, los datos de batería y su nivel de altura gracias a la información de los acelerómetros. Además están incluidos un botón de despegue, take off, y otro de aterrizaje, landing

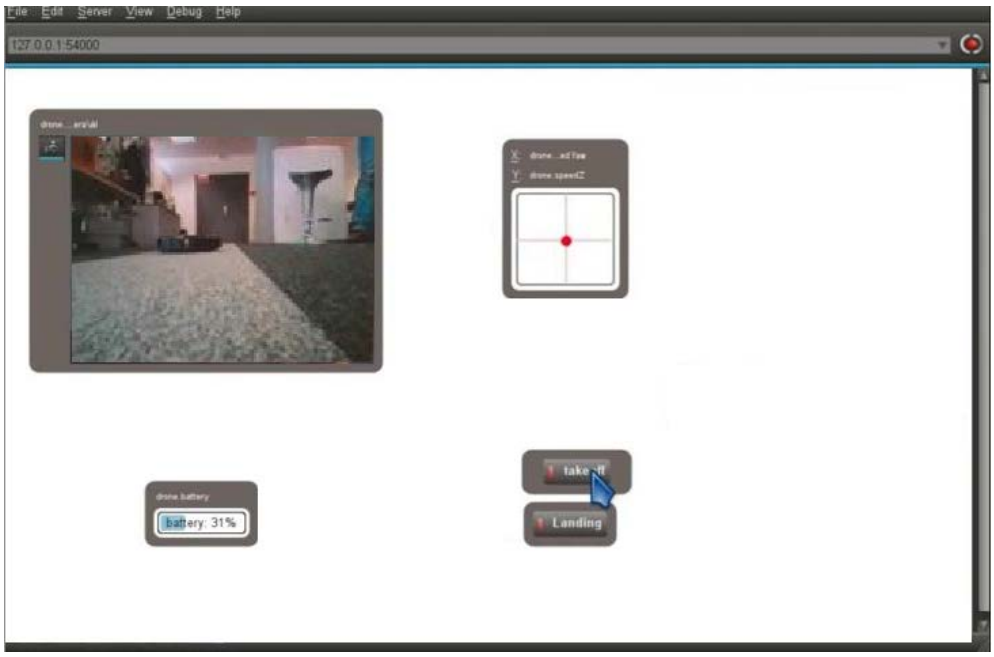

Ilustración 20. Gostai Lab

4.2 Software de desarrollo del sistema de control: ROS.

Finalmente, tras una toma de contacto con GostaiLab, se pensó en realizar en un entorno más libre de restricciones para la investigación.

Se aplicó ROS, y es el entorno en el que se ha finalizado el proyecto. ROS se trata de un meta-sistema operativo, que nos permite introducir librerías y herramientas para ayudar al desarrollo del software de aplicaciones en el campo robótico. Configuración realizada en el anexo I.

Se ha utilizado varios paquetes el principal se trata del paquete ardrone_autonomy, que es el encargado del envío y recepción de mensajes del cuadricóptero con el ordenador.

4.2.1 ROS: paquete ardrone_autonomy

El paquete ardrone_autonomy, distribuido en [20], se trata de un driver basado en ROS para el Parrot ardrone 2.0. Este driver está basado en el oficial SDK versión 2.0, [4], y es una actualización del driver anterior para hacer posible la comunicación de con el nuevo modelo 2.0 de cuadricóptero.

En el siguiente diagrama podemos observar la función de recepción de datos y envío de datos que tiene ardrone_autonomy. Además, en el anexo II podemos conocer cómo proceder en la configuración del paquete en ROS.

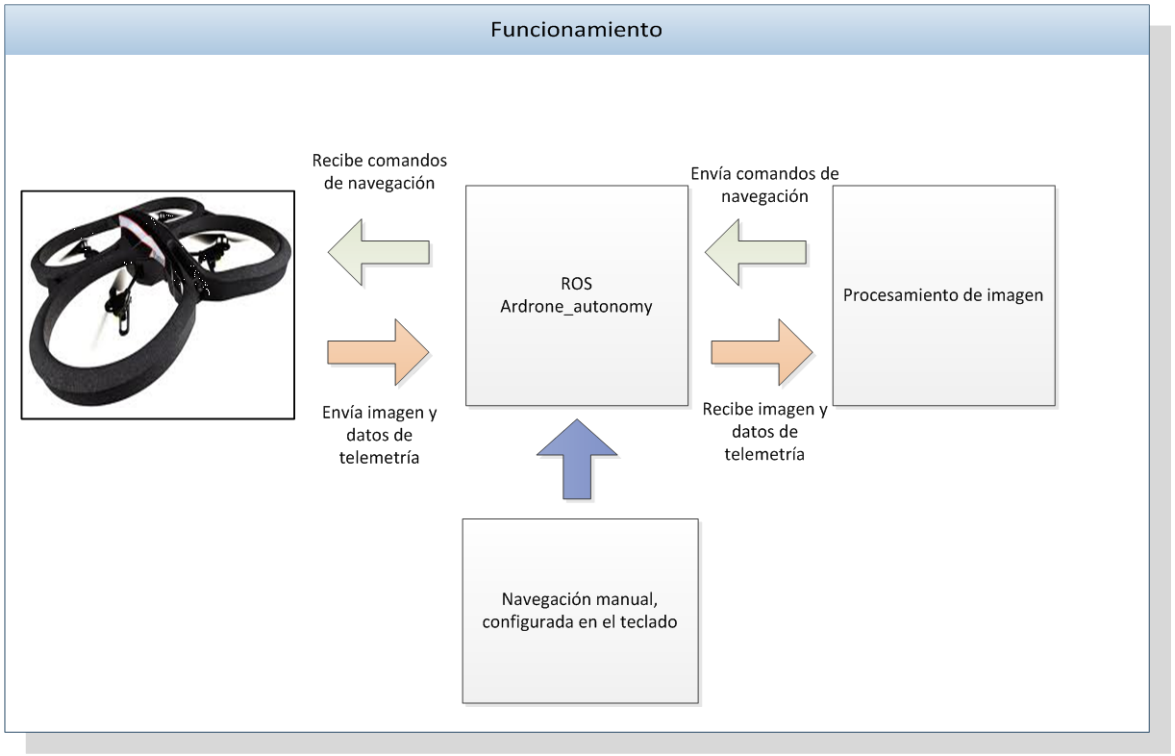

Ilustración 21. Funcionamiento ardrone_autonomy

4.2.1.1 Datos de navegación y telemetría.

El principal tema o topic, utilizado en este paquete es el navdata topic. Se trata de la información transmitida por los distintos sensores, exceptuando las cámaras. La información recibida por el ardrone es publicada a través de navdata, y hace posible la lectura de los distintos datos y telemetría emitidos por el ardrone.

Los principales tipos de mensaje son los siguientes:

- header: cabecera ROS mensaje
- batteryPercent: Carga restante del drone (%)
- state: Estado actual del cuadricóptero:

  0: Desconocido *1: Iniciado *2: Despegue *3,7: Volando *4: Suspendido en el aire *5: Test *6 (?): Lanzamiento *8: Aterrizaje *9: Looping

- rotX: inclinación izquierda/derecha en grados (rotación alrededor de eje X).
- rotY: adelante atrás de inclinación en grados (rotación alrededor del eje Y).
- rotZ: orientación en grados (rotación alrededor del eje Y).
- magX,magY,magZ: Magnetómetro lecturas.
- pressure: presión detectada por el barómetro del drone.
- temp: temperatura detectada por el sensor del vehículo aéreo.
- win_speed: velocidad del viento aproximada.
- win_angle: ángulo del viento aproximado.
- wind_comp_angle compensación estimada ángulo del viento.
- altd: estimación altura en mm.
- vx,vy,vz: velocidad lineal (mm/s).
- ax,ay,az: aceleración lineal (g)
- tm: marca el tiempo en microsegundos de los datos devueltos por el drone desde el arranque.

4.2.1.2 Publicación de mensajes para proceder a un movimiento físico.

El drone realizara los distintos movimientos de despegue, aterrizaje o parada de emergencia cuando los siguientes topics se encuentren en fase Empty, es decir vacíos.

- ardrone/takeoff el ardrone despega y se mantiene a una distancia limitada del suelo.
- ardrone/land el cuadricóptero comienza el descenso para finalizar en un aterrizaje.
- ardrone/reset se activa la parada de emergencia.

Tras el despegue se pueden proceder al movimiento del ardrone, publicando un mensaje de tipo geometry_msgs::Twist a cmd_vel topic. Este topic o tema, es el encargado del movimiento, con la configuración de parámetros basados en la velocidad angular y lineal del aparato.

El topic /cmd_vel tiene la siguiente estructura:
-linear.x : retroceso
+linear.x : avance
-linear.y : movimiento hacia la derecha.
+linear.y movimiento hacia la izquierda.
-linear.z: movimiento descendente sobre el eje z.
+linear.z: movimiento ascendente. Sobre el eje z.
-angular.z: rotación izquierda.
+angular.z: rotación derecha.

Se pueden publicar de forma manual mensajes en este topic, pero se ha pensado en el uso de un teclado para un manejo del robot más intuitivo.

4.2.1.3 Cámaras ardrone 2.0

Equipado con dos cámaras, una con visión frontal con visión delantera y otra vertical en dirección descendente.

El driver crea dos topics, cada una de ellos con el estándar de ROS en cameras, publican un mensaje de tipo imagen, son los siguientes:

- ardrone/front/image_raw
- ardrone/bottom/image_raw

La información de la calibración se encuentra en los archivos *ardrone_front.yaml* y *ardrone_bottom.yaml*, y se publica en el topic camera_info. Para una correcta calibración se debe usar el paquete camera_calibration, en el anexo III se añade cómo realizarla en el ardrone.

El ardrone 2.0 no permite la característica PIP por tanto en el topic *ardrone/** solo contendrá la imagen y video de la cámara seleccionada, la vertical o la frontal.
ardrone/front, cámara frontal.
ardrone/bottom, cámara vertical.

Un rápido acceso a todos los topics que contiene este driver, se usa el comando *rostopic list*, muestra por pantalla una lista de topics activos.

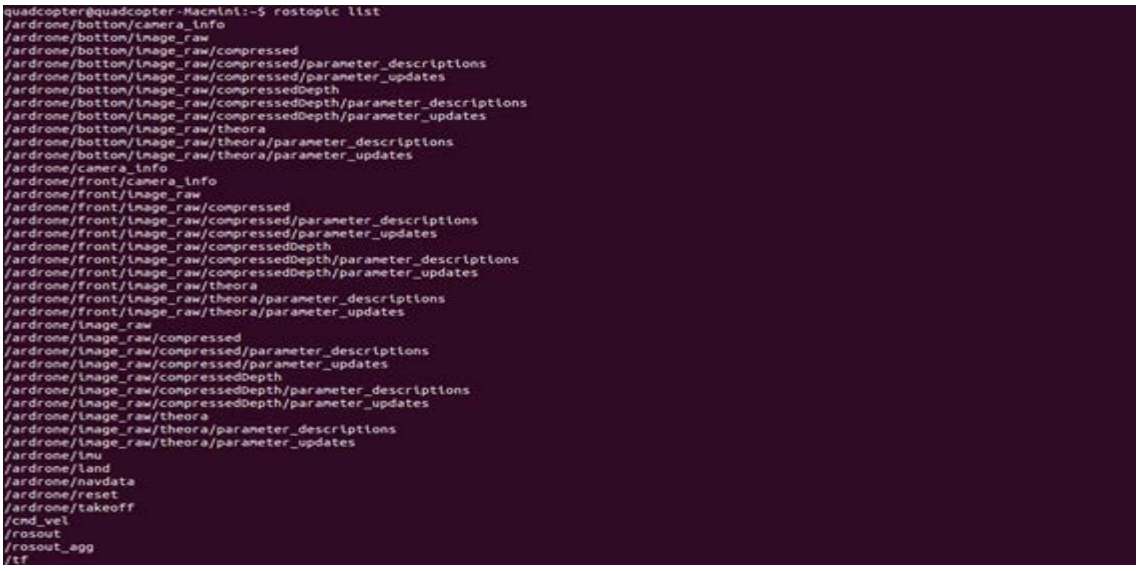

Ilustración 22. rostopic list

El comando *rostopic list* pertenece al paquete *rostopic*, este paquete también soporta otros comandos que muestran en pantalla los topics publicadores y suscriptores de un tema específico, la tasa de publicación de un tema, el ancho de banda de un tema y mensajes publicados para un tema. La pantalla de mensajes se puede configurar para una salida en un formato más fácil de trazar.

Los distintos comandos de *rostopic* son:

- rostopic bw muestra el ancho de banda usado por el topic
- rostopic echo imprime la publicación de un topic que emite información.
- rostopic find encuentra topics por tipo.
- rostopic hz publica el ritmo de publicación de un topic
- rostopic info publica información acerca un topic activo
- rostopic list publica una lista de todos los topics activos.
- rostopic pub publica información en el topic.
- rostopic type imprime el tipo de topic.

Para el uso de los comandos que van asociados a un topic en concreto, debe usarse una forma como específica como la siguiente:

$ rostopic <comando_rostopic> /<nombre_topic>

Un ejemplo es:

$ rostopic echo /ardrone/front/image_raw

Al insertar por comando la herramienta *rxgraph,* para la visualización de un gráfico de computación en ROS, en la siguiente figura podemos ver el gráfico de ROS en modo activo del driver ardrone_autonomy.

Ilustración 23. rxgraph ardrone_autonomy

4.2.2 Control manual del ardrone configurado para una estación remota basada en Linux.

Una vez superado el camino de la comunicación de nuestro vehículo aéreo no tripulado con Linux, se debe avanzar hacia el movimiento a través de una estación remota, en este caso también basada en Linux, y aplicando el sistema de publicación de mensajes de ROS.
Para ello he desarrollado un paquete nuevo compatible con la nueva versión 2.0 del ardrone, modificando varios paquetes necesarios para el cuadricóptero anterior.

Los movimientos de navegación pueden tener un rango de -1.0 a 1.0, en esta ocasión se ha utilizado ±0.1, ya que se ha diseñado para un entorno interior, donde las distancias son cortas, y es preferible tener un dominio más exacto del control. Aumentando este valor en el código aumentamos la velocidad pero disminuimos la precisión.

Para distancias medias se puede elevar este número, llegando incluso a ±1.0 para una necesidad de movimiento más veloz como se ha utilizado en velocidades angulares.

En código los datos de navegación son los siguientes:

```
move_bindings = {
        68:('linear', 'y', 0.1), #izquierda
        67:('linear', 'y', -0.1), #derecha
        65:('linear', 'x', 0.1), #adelante
        66:('linear', 'x', -0.1), #atrás
        'w':('linear', 'z', 0.1), #arriba
        's':('linear', 'z', -0.1),#abajo
        'a':('angular', 'z', 1),#rotación hacia la izquierda
        'd':('angular', 'z', -1),#rotación hacia la derecha}
```

Pulsando la tecla correspondiente, activamos el envío de datos de navegación, y drone_teleop es el nodo que se encarga de comunicar y publicar en los diferentes topics para que se produzca el movimiento.

El menú desde el que podemos ver mover el cuadricóptero es el siguiente:

```
quadcopter@quadcopter-Macmini:~$ rosrun drone_teleop drone_teleop.py

arDrone 2.0
---------------------------
up/down:       Mover delante/detras
left/right:    Mover izquierda/derecha
w/s:           Incremetentar/ reducir altitud
a/d:           Girar izquierda/derecha
t/l:           Despegar/aterrizar
r:             Reset (Luces rojas)
Otra:          Parar

No use el bloqueo de mayusculas
CTRL+C para salir

Dpto. de Ingenieria de Sistemas y Automatica. LSI.
Universidad Carlos III de Madrid.
```

Ilustración 24. Menú control manual

Las teclas up, down, left o right se refiere a las teclas flecha encontradas en nuestro teclado.

Se puede ver claramente en el siguiente gráfico, como tras el uso del paquete drone_teleop, el nodo drone_teleop publica mensajes en los topics referentes al movimiento.

Ilustración 25. rxgraph drone_teleop

Es posible ver la velocidad con la que ardrone se está moviendo, leyendo el *topic /cmd_vel* , donde se guardan los parámetros de velocidad lineal y angular. Este topic tendrá el formato anterior explicado en el punto “Publicación de mensajes para proceder a un movimiento físico”.

4.2.3 Reconocimiento del entorno a través de una cámara monocular.

Desarrollada la comunicación y configurado un sistema intuitivo para mover el ardrone 2.0, sin usar la aplicación oficial para iOS o Android, sino que se establece en una plataforma Linux. Un nuevo punto, es el del reconocimiento del entorno a través de su cámara monocular frontal, este modelo integra una cámara de alta resolución de 720p.

Esta alta calidad produce un reconocimiento más preciso del entorno, pero la velocidad de procesamiento disminuye al tratar imágenes de mayor tamaño.

Gracias al reconocimiento del entorno, podemos procesar esas imágenes y en base a los datos recibidos, enviar comandos de navegación. En esta ocasión se busca un patrón en concreto, el vehículo aéreo no tripulado gracias a este sistema de control por percepción del entorno, se convierte en un vehículo aéreo autónomo.

Para esta misión se necesita una percepción del entorno, se ha elegido la búsqueda de un patrón, que es el siguiente:

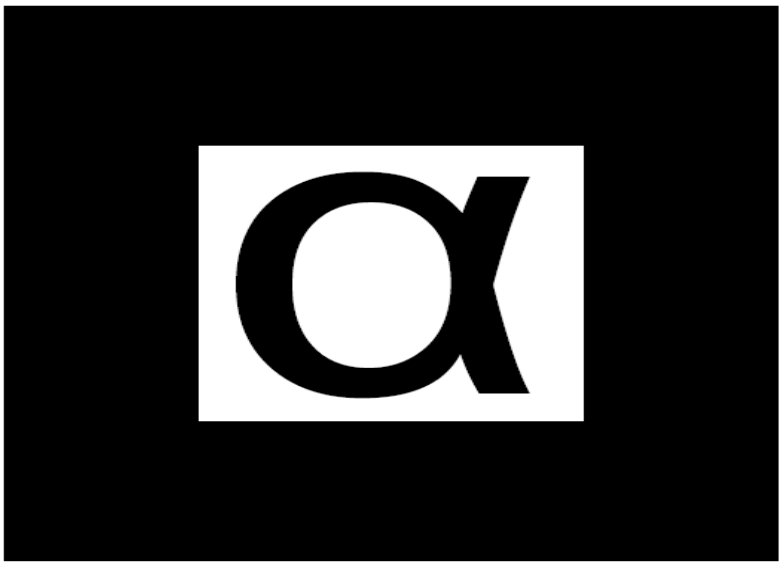

Ilustración 26. Patrón alfa

El reconocimiento se basa en la búsqueda de cuadriláteros, y en la comprobación de que si en el interior se encuentra una alfa como en la imagen anterior, si esto es así, recuadra el patrón en color verde de manera fija. Lo podemos ver en la imagen siguiente:

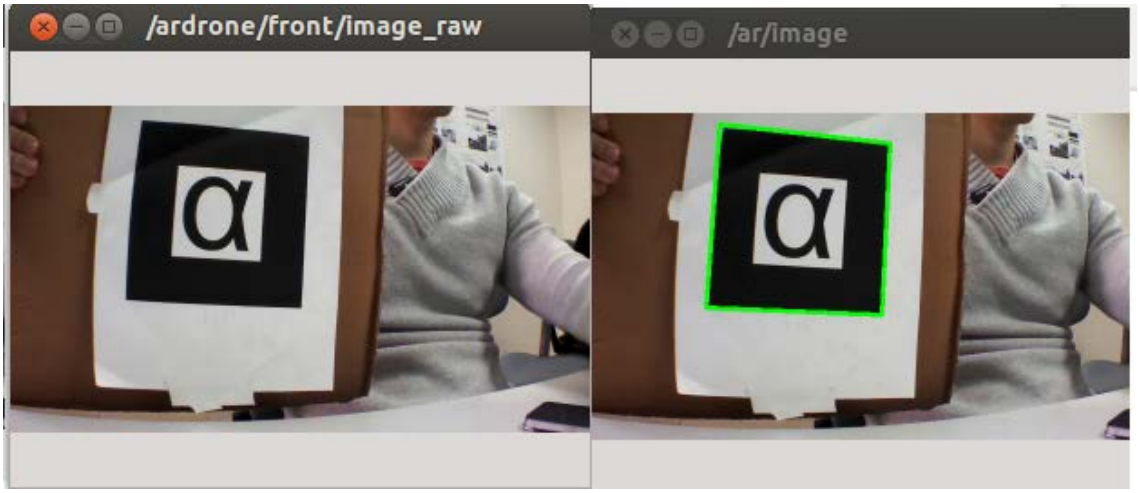

Ilustración 27. Tag patrón

Gracias a una buena calibración de la cámara puede llegar a reconocer hasta el patrón algo ladeado, como muestra la siguiente imagen:

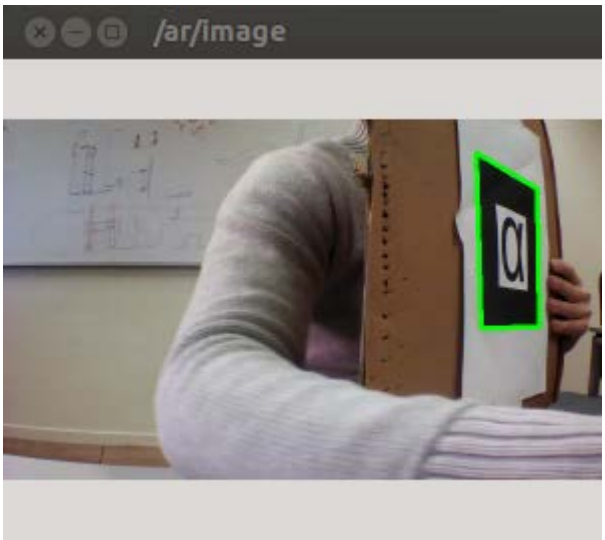

Ilustración 28. Tag patrón ladeado.

Podemos observar el gráfico de computación de ROS emitido con el comando *rxgraph*, en este ejemplo vemos que está activado tanto el manejo manual del aparato como el reconocimiento de imagen.

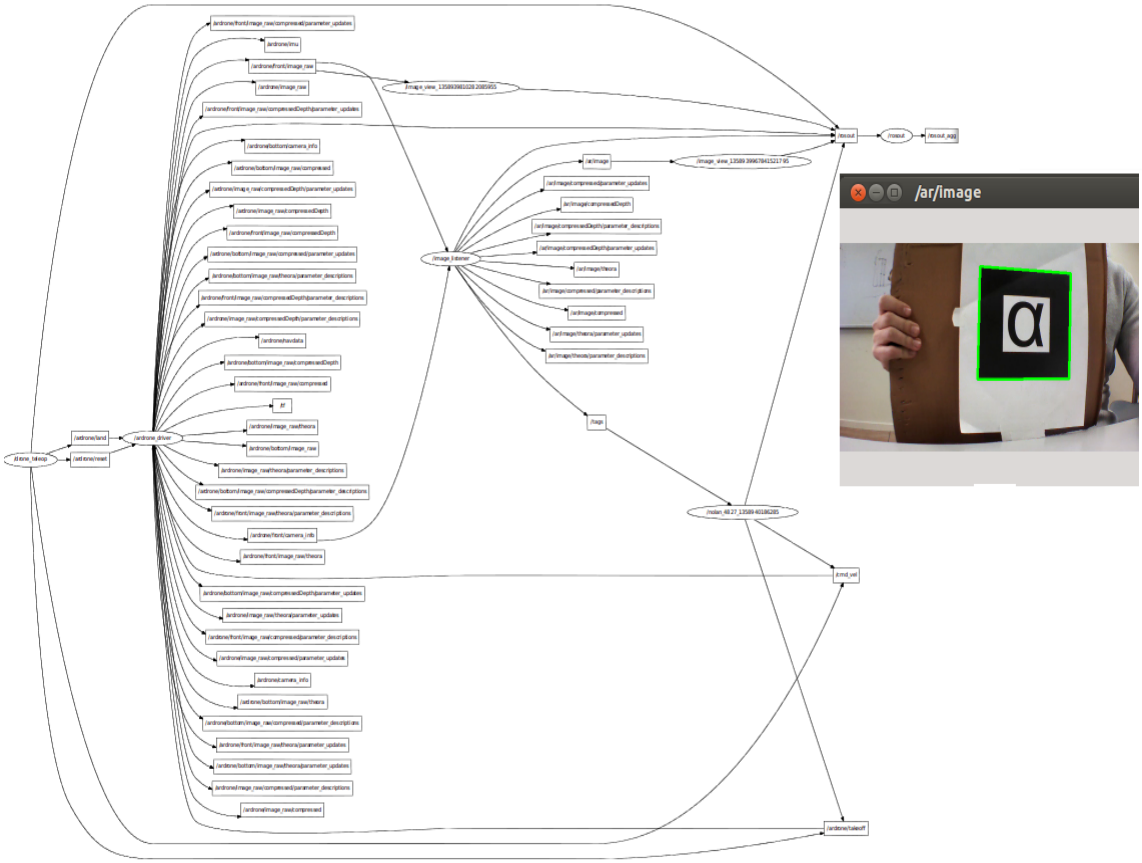

Ilustración 29. Rxgraph ar_recog

Para iniciar el reconocimiento debemos cargar el archivo que queremos encontrar,este se encuentra en un archivo en escala de grises. Además, debemos seleccionar el ángulo de visión que determina la parte de la escena que es captado por la cámara. Debido a las características del contexto, donde se desarrolla este proyecto, se ha elegido un ángulo de visión de 67º.

Después de los test realizados para observar un correcto reconocimiento del patrón, se ha pasado al procesamiento de envío de mensajes de navegación en base a la imagen recibida.

4.2.4 Navegación autónoma.

La navegación aérea autónoma se trata de la navegación que no necesita ningún envío de información, o control exterior para poder completar el vuelo con éxito. En este punto se trata el objetivo principal de este proyecto, que el ardrone con la información recibido por sus sensores incluido las cámaras, pueda completar un vuelo.

Se ha aplicado un filtro realimentado que desarrolla en el ardrone un vuelo autónomo. El vehículo aéreo busca el patrón como se ha explicado en el anterior punto, “Reconocimiento del entorno con una cámara monocular”, al encontrar el patrón se mantiene a una distancia, rectificando su velocidad y altura para mantener una visión directa con el patrón.

Además, pensando en un futuro desarrollo de otras herramientas, el controlador emite por mensajes en pantalla la distancia y ubicación del patrón al ser visualizado por la cámara del ardrone, incluso la velocidad angular y linear que tiene el vehículo.

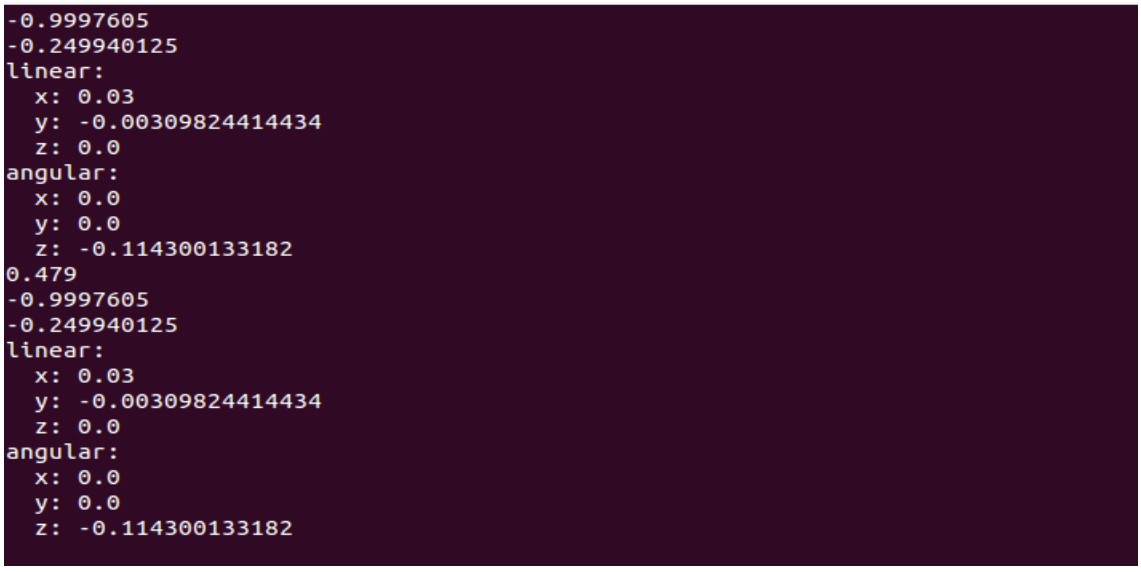

Ilustración 30. Datos nolan3d.

Gracias al nivel grafico de ROS, y la herramienta rxgraph, podemos ver a nivel de conexiones cómo ROS está ejecutando el programa. En la siguiente ilustración vemos parte del gráfico de ROS.

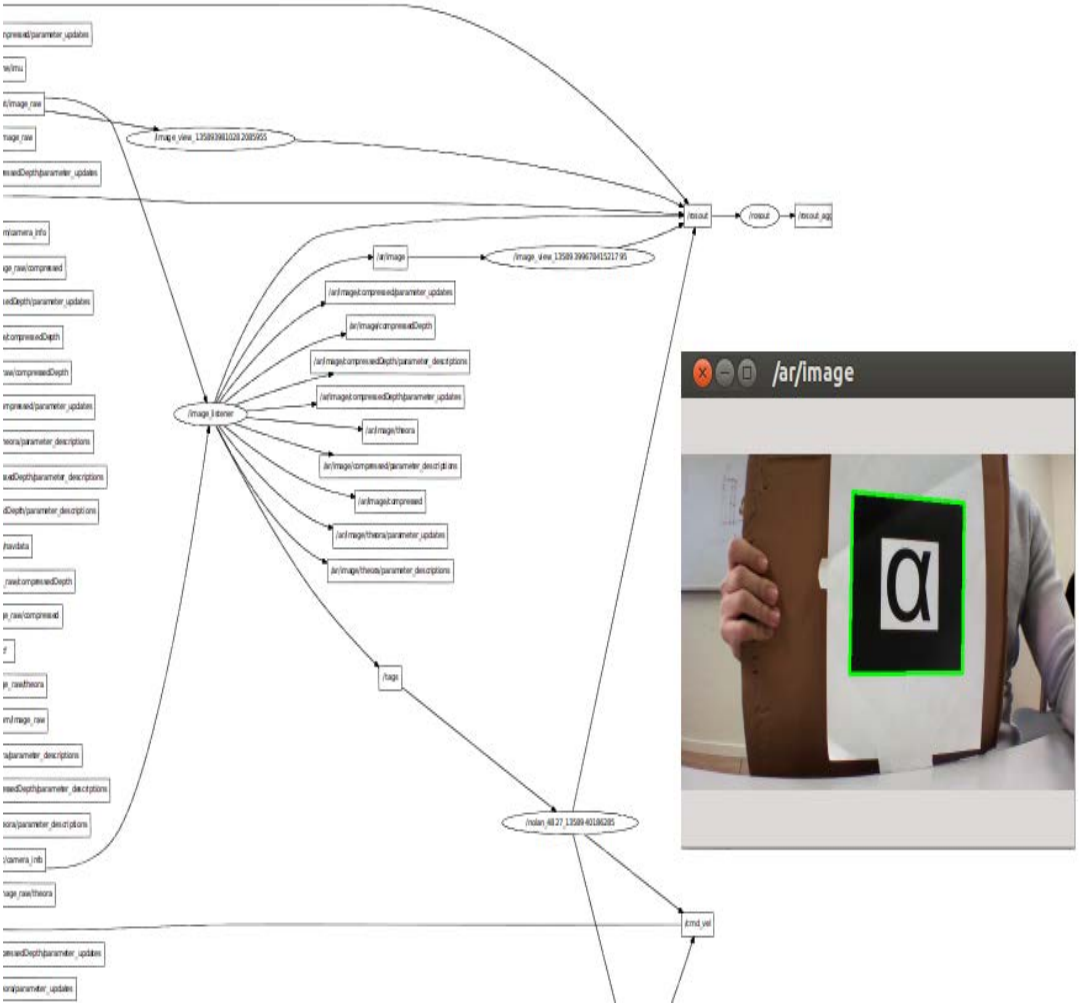

Ilustración 31. Rxgraph nolan3d

Finalmente, obtenemos el objetivo de la navegación autónoma de un vehículo no tripulado. En la imagen 31, podemos observar el ardrone 2.0 en plena navegación y reconociendo el patrón configurado. Volando totalmente autónomo utilizando la cámara a bordo frontal como principal sensor para el vuelo.

Ilustración 32. Ardrone vuelo autónomo

Durante el vuelo, el cuadricóptero reconoce el patrón y mantiene una distancia estable, se puede observar el instante de reconocimiento en su cámara a bordo, en la siguiente imagen 33.

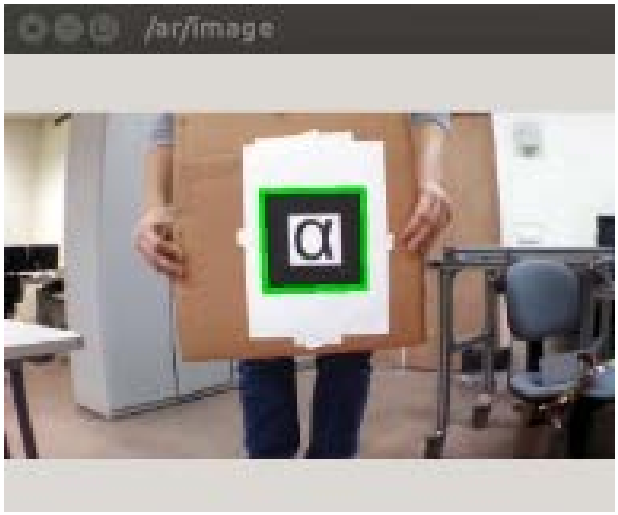

Ilustración 33. Vuelo autónomo, a bordo.

A través del siguiente diagrama de flujo, podemos observar el funcionamiento del proyecto de navegación del ardrone 2.0

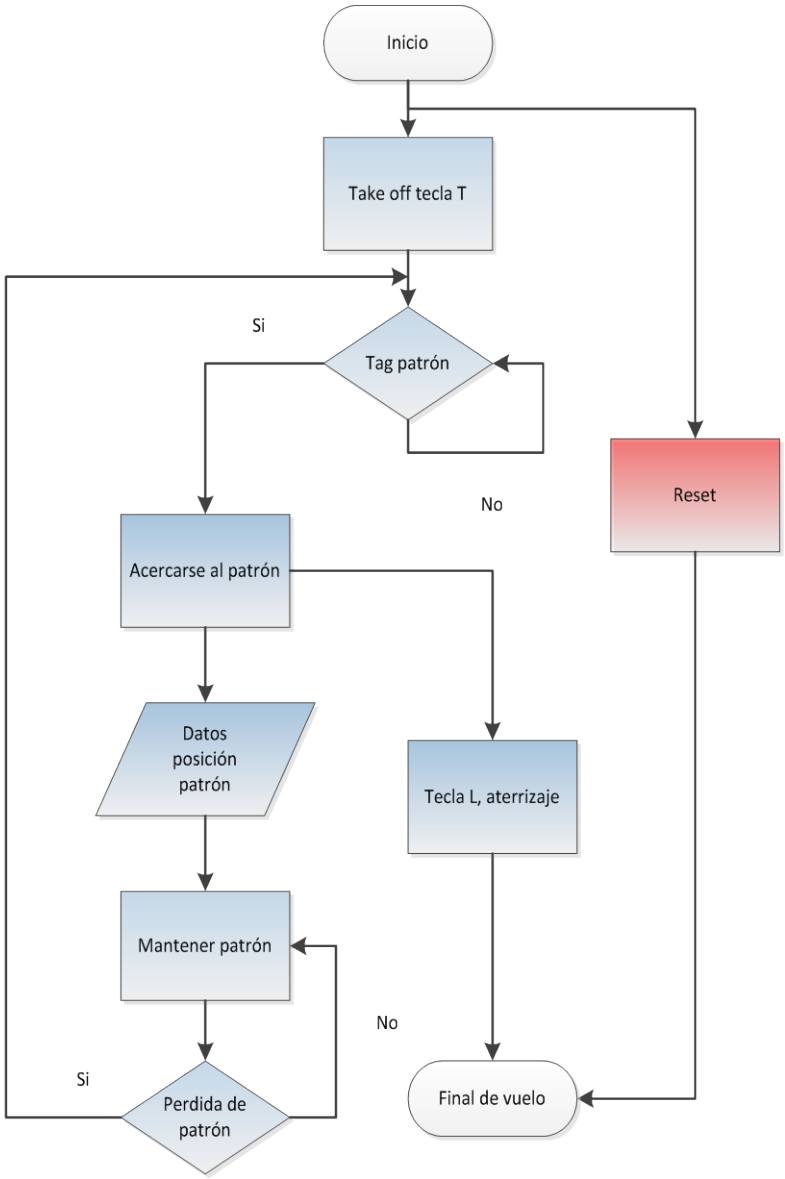

Ilustración 34. Diagrama de flujo.

4.3 Detección de líneas utilizando la transformada de Hough.

Concluida el desarrollo de una aplicación para de conseguir la navegación autónoma mediante el guiado por patrón del Ardrone se ha investigado más a fondo en el uso de las OpenCV en ROS, principalmente en estos libros [7] y [8].

Se ha creado un paquete dependiente de varios paquetes entre ellos roscv, paquete de OpenCV en ROS, para la detención de líneas en una imagen a través de la transformada de Hough. Además de las líneas, también es posible encontrar todo tipo de figuras que puedan ser expresadas matemáticamente, tales como circunferencias o elipses.

La transformada de Hough, [21], es uno de los algoritmos más básicos de la tecnología de visión por computador. Su uso reside en múltiples aplicaciones ya que tiene variedad de reconocimiento como el reconocimiento de patrones, comparación de objetos, detección Lane o SLAM son algunos de ellos. La detección de líneas se utiliza en la base en todas las variedades nombradas.

OpenCV nos ofrece dos tipos de detección de línea:

1) Transformada estándar de Hough. (SHT)

2) Método probabilístico de la transformada de Hough (HPP)

La transformada de línea de Hough parte en que toda imagen binaria puede ser parte de un conjunto de líneas posibles. El caso más simple para la transformada de Hough es la transformación lineal para detectar rectas. En el espacio de la imagen, la recta se puede representar con la ecuación $y = m * x + n$ y se puede graficar para cada par (x, y) de la imagen. En este caso al contener también líneas verticales debemos usar parámetros de coordenada polar (ρ, θ).

El parámetro ρ representa la distancia entre el origen de coordenadas y la recta, mientras que θ es el ángulo del vector director de la recta perpendicular a la recta original y que pasa por el origen de coordenadas.

Si se convierte cada píxel distinto de cero en la imagen de entrada en un conjunto de puntos en la imagen de salida y la suma sobre todas las contribuciones, a continuación, las líneas que aparecen en la imagen de entrada aparecerán como máximos locales en la imagen de salida. El plano donde reside cada punto se resumen el plano (a, b) se llama plano acumulador.

En las siguientes imágenes podemos ver el método estándar de la transformada de Hough, utilizado en la visión de la cámara del ardrone.

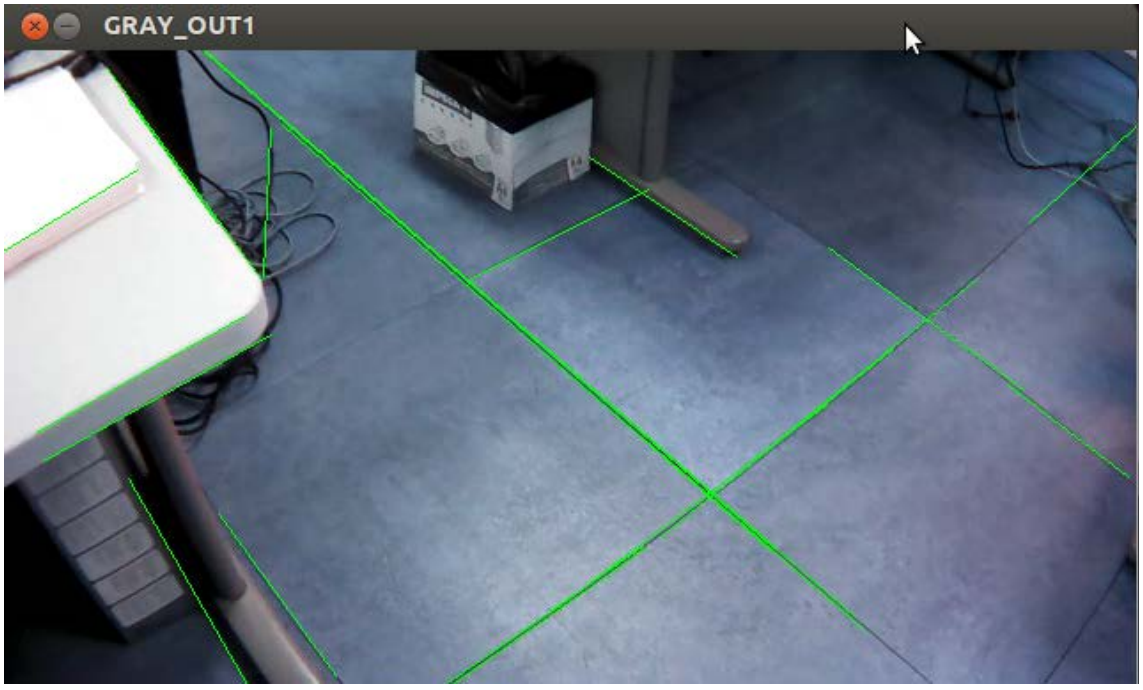

Ilustración 35. Transformada de Hough estándar

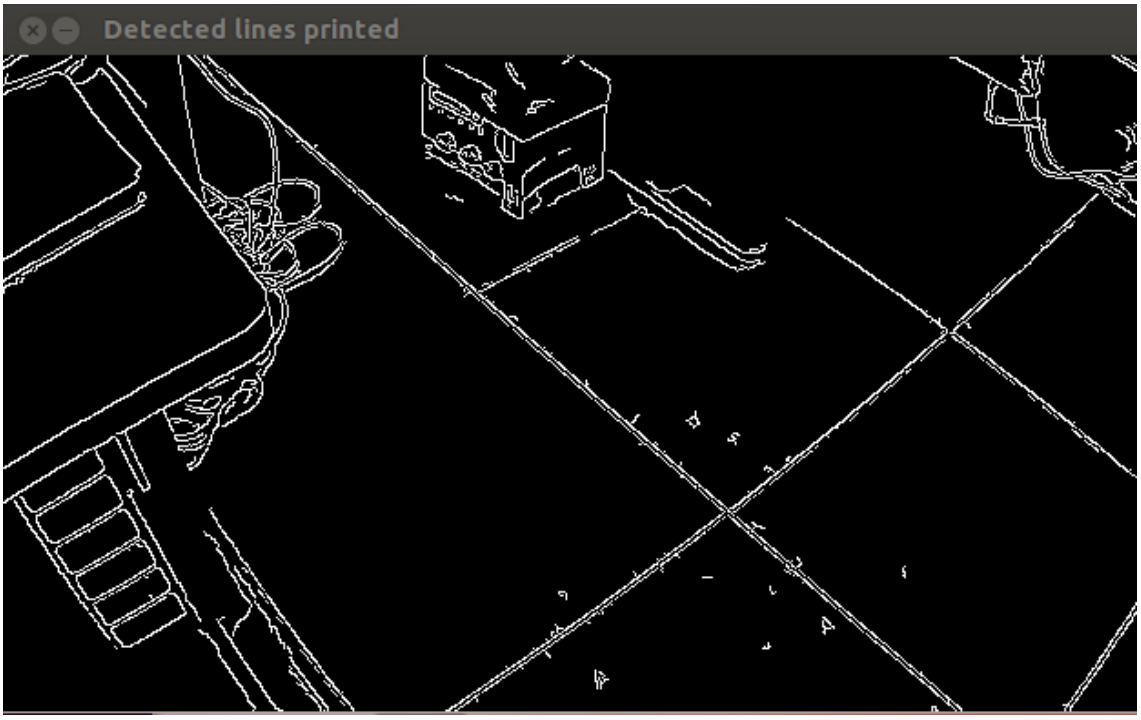

Ilustración 36. Detección de líneas (1)

Ilustración 37. Escala de grises (1)

La diferencia del método probabilístico con respecto a la transformada estándar de Hough, es que el primero, solo acumula una fracción de los puntos en el plano acumulador y no todos ellos.

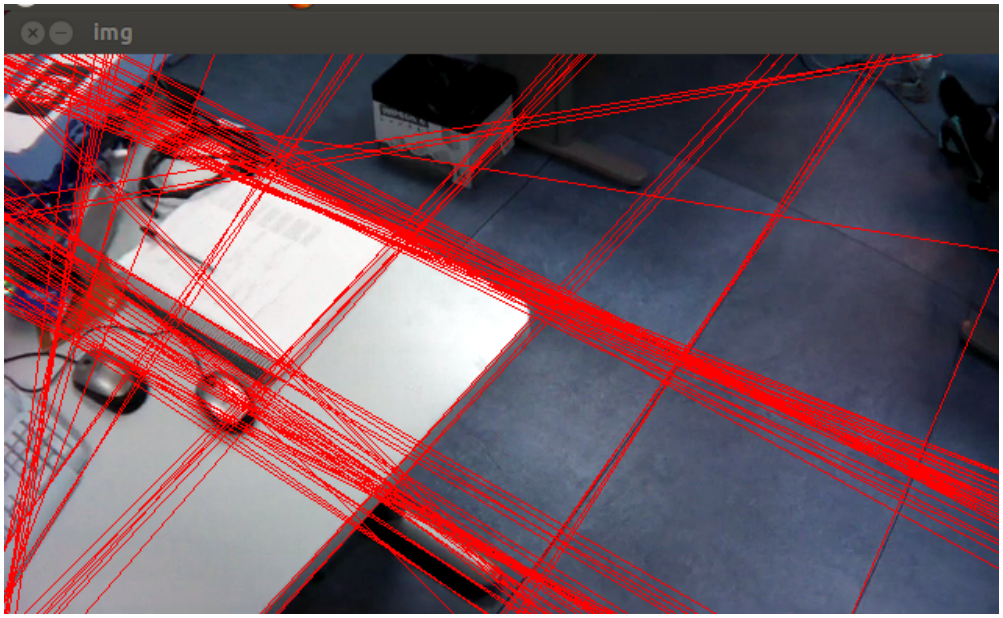

Ilustración 38. Transformada probabilística de Hough.

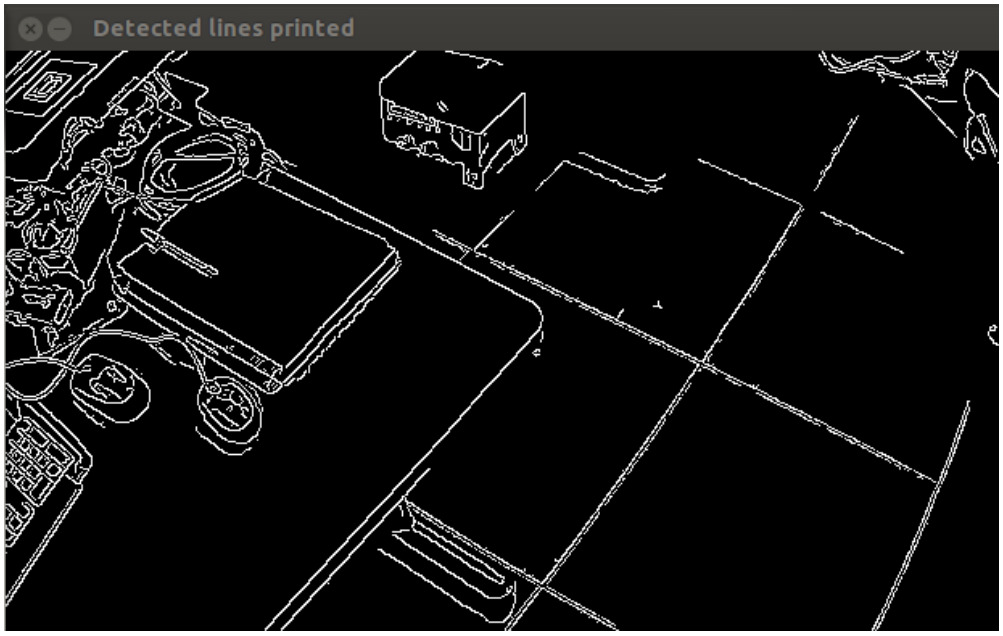

Ilustración 39. Detección líneas (2)

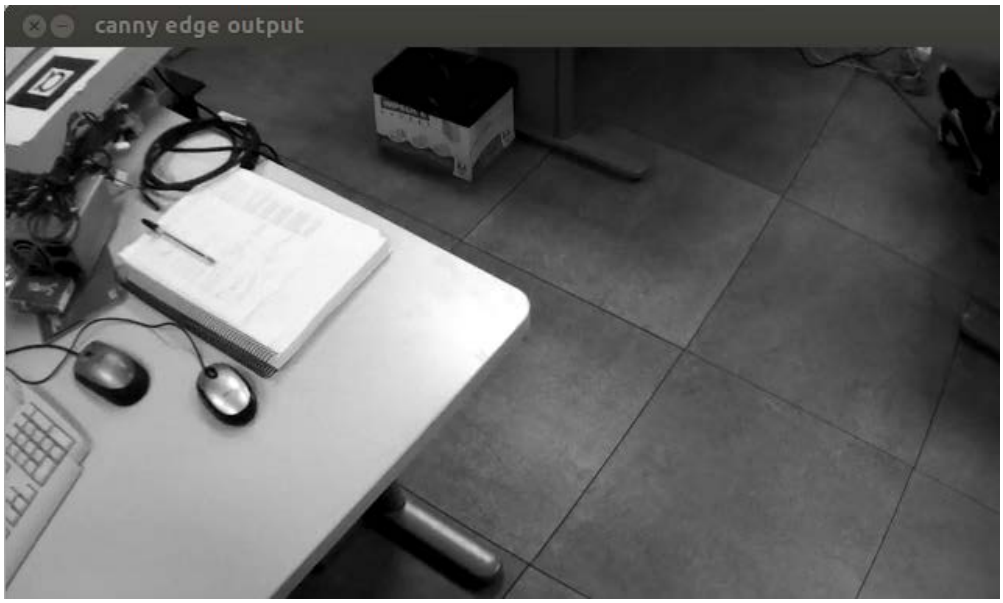

Ilustración 40. Escala de grises (2)

Capítulo 5 : RESULTADOS Y DISCUSIÓN.

Partiendo de un vehículo no tripulado recreativo llamado ardrone 2.0, comercializado para un pilotaje manual a través de un Smartphone, se ha conseguido que realice vuelo autónomo a partir de los datos obtenidos por los sensores integrados.

El vuelo autónomo se basa en el reconocimiento del entorno, principalmente con la información obtenida por su cámara frontal. Una vez iniciado el vuelo el ardrone busca el patrón cargado y entonces inicia el movimiento de acercamiento manteniéndose a una distancia configurada por el usuario.

En el caso de que el patrón sea dinámico, el ardrone seguirá su movimiento intentando mantener el patrón en el centro de su visión y a la distancia configurada.

Incluso en el caso de pérdida del patrón, el ardrone comenzará a buscar de nuevo, y se activará de nuevo el procedimiento de vuelo a distancia determinada sobre el patrón.

Si en el momento del vuelo, activamos cualquier tecla de movimiento manual configurada sobre el teclado, el ardrone pasará a vuelo manual, y tendremos total decisión sobre el mismo como función de control de seguridad, en caso necesario. Los datos de ubicación del patrón, una vez reconocido, y velocidad del cuadricóptero, son mostrados por pantalla junto al menú de vuelo manual.

Se ha desarrollado también, visión de líneas a través de OpenCV y usando la Transformada de Hough para futuras ampliaciones.

Debido al uso de la arquitectura ROS, el trabajo realizado en el proyecto para el ardrone es portable a otros dispositivos con algunas pequeñas modificaciones.

Capítulo 6 : TRABAJOS FUTUROS Y CONCLUSIONES

6.1 Trabajos futuros

En el desarrollo del proyecto se ha encontrado la manera de publicar y recibir información referida a la navegación, o la telemetría del cuadricóptero. Se obtuvo un punto de inflexión, ya que al conocer el funcionamiento del ardrone, y de ROS con el ardrone, obtenemos multitud de vías de investigación.

Una vez resuelto el principal objetivo del proyecto, de dotar al ardrone de un software capaz de realizar un vuelo autónomo, se ha investigado en los diferentes caminos de ampliación.

La principal ampliación se centra en la navegación, podríamos dotar al ardrone de una navegación autónoma siguiendo una línea recta, al traer integrada una cámara vertical podemos observar el suelo, y con ella la línea guía para realizar ese vuelo. En el presente proyecto se investiga en la percepción del entorno con el seguimiento por patrón, pero también se establece un visionado de líneas con las transformadas de Hough.

Además, descubiertas las ventajas de este pequeño vehículo no tripulado con visión por cámara, podemos realizar mapeado 3D a través de una única cámara al dotar la cámara de una de un movimiento físico, como posee el ardrone, podemos llegar a conocer las coordenadas de posición del ardrone en el momento de la toma de imagen.

Cercana a la aplicación obtenida en este proyecto podemos realizar una detección de caras, que puede ser muy útil en tareas de vigilancia o desastres naturales. Sería posible encontrar personas en zonas de difícil acceso, o a modo de vigilante para el reconocimiento de intrusos.

6.2 Conclusiones

Demostrada la capacidad de vuelo autónomo en un pequeño vehículo de cuatro motores para el ocio, se encuentra disponible de manera absoluta la investigación en estos pequeños aparatos.

Visto el nivel de reacción y la estabilidad gracias a su configuración de cuatro motores, se pueden sugerir multitud de campos de avance, además de los ya citados.

Los cuadricópteros son vehículos aéreos de corta historia, pero con una curva de alto desarrollo en estos momentos. En el *Capítulo 1* se nombraron algunos, pero son innumerables los modelos utilizados en el ámbito militar, mayoritariamente como elemento de vigilancia, pero la mayoría no disponen de vuelo autónomo.

Dotando a estos aparatos de un pilotaje autónomo se podrían ganar en seguridad, y en rapidez de reacción, por ejemplo, en la búsqueda de artefactos como minas, e incluso, aparatos autónomos suficientemente potentes podrían rescatar a personas una vez encontradas sin esperar a la llegada de un equipo humano.

No se puede estimar la multitud de campos donde estos aparatos pueden avanzar. Sin embargo, el desarrollo de los cuadricópteros es una apuesta segura para el avance en numerosos campos, dando máxima importancia a su característica principal de estabilidad, y posibilidad de reconocimiento del entorno gracias a sensores, entre ellos, las cámaras con el tratamiento de imagen.

Capítulo 7 :PRESUPUESTO

7.1 Coste de material.

La siguiente tabla muestra el coste obtenido de la compra de material y equipos para el desarrollo del proyecto.

Tabla 1. Coste de material.

Código	Unidad	Descripción	Cantidad	Coste unidad	Coste total
110	ArDrone 2.0	Vehículo aéreo de 4 motores eléctricos con distribución de cuadricóptero. Desarrollado y distribuido por Parrot.	1	310,00 €	310,00 €
120	PC	Ordenador con instalación de Ubuntu 12.04 LTS.	1	799,00 €	799,00 €
130	ROS	ROS (Robot Operating System) sistema operativo en código abierto con licencia BSD. Incluye sistema operativo ROS y paquetes necesarios introducidos en ROS.	1	0,00 €	0,00 €
			TOTAL PARTIDA		1.109 €

7.2 Costes de personal.

A continuación se presupuesta el coste del personal necesario para la finalización del proyecto.

Tabla 2. Coste de personal.

Código	Unidad	Descripción	Cantidad	Coste unidad	Coste total
210	Ingeniero industrial electrónico	Ingeniero industrial para investigación y desarrollo del proyecto: DISEÑO DEL SISTEMA DE CONTROL PARA UN CUADRICÓPTERO	3 meses	1600€/mes	4.800,00€
			TOTAL PARTIDA		4.800 €

7.3 Resumen del presupuesto.

Con esta tabla se presenta el coste aproximado total para el desarrollo del proyecto.

Tabla 3. Resumen del presupuesto.

Código	Unidad	Cantidad	Coste
100	Coste material y equipos	1	1.109,00 €
200	Coste de personal	1	4.800,00 €
	TOTAL PARTIDA		5.909 €

Capítulo 8 : BIBLIOGRAFÍA

[1] A. de la Escalera, Visión por computador. Fundamentos y métodos., 2001.

[2] «Military Factory» [En línea]. Available: http://www.militaryfactory.com/aircraft/unmanned-aerial-vehicle-uav.asp. [Último acceso: Enero 2013].

[3] «Honeywell» [En línea]. Available: http://www.thawkmav.com/. [Último acceso: Diciembre 2012].

[4] «Developter guide SDK 2.0 Ardrone» 2012.

[5] «Sitio Web Brown University» [En línea]. Available: http://brown-robotics.org/wp/. [Último acceso: Octubre 2012].

[6] «Institure for Dynamic Systems and Control,» [En línea]. Available: http://www.idsc.ethz.ch/. [Último acceso: Enero 2013].

[7] G. Bradski y A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, 2008.

[8] S. E. Daniel Lélis Baggio, Mastering OpenCV with Practical Computer Vision Projects, 2012.

[9] «AUVSI» [En línea]. Available: http://www.auvsi.org/Home/. [Último acceso: Febrero 2013].

[10] «Normativa UL,» [En línea]. Available: http://www.ul.com/global/spa/pages/solutions/standards/accessstandards/catalogofstandards/standard/index.jsp?id=2054_2. [Último acceso: Enero 2013].

[11] «BSD license» [En línea]. Available: http://opensource.org/licenses/bsd-license.php. [Último acceso: Enero 2013].

[12] «Willow Garage» [En línea]. Available: http://www.willowgarage.com/. [Último acceso: Enero 2013].

[13] «ROS Answers» [En línea]. Available: http://answers.ros.org/questions/. [Último acceso: Enero 2013].

[14] «ROS-Conceptos» [En línea]. Available: http://www.ros.org/wiki/ROS/Concepts. [Último acceso: Diciembre 2012].

[15] « ROS paquete camera_calibration,» [En línea]. Available: http://www.ros.org/wiki/camera_calibration. [Último acceso: Enero 2013].

[16] «Repositorio ROS» [En línea]. Available: http://www.ros.org/browse/list.php. [Último acceso: Enero 2013].

[17] «WikiROS» [En línea]. Available: http://www.ros.org/wiki/Robots. [Último acceso: Enero 2013].

[18] «ROS-Tutoriales» [En línea]. Available: http://www.ros.org/wiki/ROS/Tutorials. [Último acceso: Diciembre 2012].

[19] «Gostai Lab,» [En línea]. Available: http://www.gostai.com/. [Último acceso: Septiembre 2012].

[20] «Paquete ardrone_autonomy» [En línea]. Available: https://github.com/AutonomyLab/ardrone_autonomy. [Último acceso: Diciembre 2012].

[21] "Tranformada de Hough" [En línea]. Available: http://en.wikipedia.org/wiki/Hough_transform. [Último acceso: Enero 2013].

ANEXO I

- Instalación de ROS

Anexo realizado para Ubuntu 12.04 (Precise)

La instalación de ROS la iniciamos abriendo un terminal con la combinación (Ctrl+Alt+T) y tecleamos las siguientes líneas.

$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.list.d/ros-latest.list'

$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

$ sudo apt-get update

$ sudo apt-get install ros-fuerte-desktop-full

$ sudo apt-get install python-rosinstall python-rosdep

echo "source /opt/ros/fuerte/setup.bash" >> ~/.bashrc

. ~/.bashrc

- Creación workspace o espacio de trabajo.

1. Creamos la carpeta workspace en el directorio deseado en este caso la llamaremos fuerte_workspace.
   $ cd
   $ mkdir fuerte_workspace

2. Abrimos un terminal y vamos a la carpeta home. Ejecutar el siguiente comando:

   $ gedit .bashrc

3. Tras la ejecución anterior se abrirá un documento donde debemos ver que la última línea es la siguiente:

   "source /opt/ros/fuerte/setup.bash"

4. A continuación añadimos justo debajo el siguiente comando. Aclarar de nuevo que mi carpeta se llama fuerte_workspace.

   export ROS_PACKAGE_PATH=~/fuerte_workspace:$ROS_PACKAGE_PATH

   export ROS_WORKSPACE=~/fuerte_workspace

5. Guardar el archivo y cerrar. Cerrar también el terminal
6. Para comprobar si hemos creado bien nuestro workspace ejecutamos en un nuevo terminal:
   $ echo $ROS_WORKSPACE

   Si todo es correcto tendremos un texto parecido al siguiente:
   Aparecerá un texto parecido a este:
   /home/quadcopter/fuerte_workspace

ANEXO II

- Instalación y compilación ardrone_autonomy

1. Debemos acceder a nuestro workspace:
   $ cd ~/fuerte_workspace/
2. Una vez accedemos copiamos el contenido, tecleando los siguiente comandos:

   $ git clone https://github.com/AutonomyLab/ardrone_autonomy.git
   $ rosstack profile && rospack profile

3. Accedemos a la carpeta ardrone_autonomy
   $ roscd ardrone_autonomy

4. Construimos el ejecutable:
   $ ./build_sdk.sh

5. Después de unos minutos escribimos:
   $ ls ./lib
   Debemos ver al menos estas librerías:
   libavcodec.a libavformat.a libpc_ardrone_notool.a libvlib.a
   libavdevice.a libavutil.a libsdk.a
   libavfilter.a libpc_ardrone.a libswscale.a

6. Compilamos el driver:
   $ rosmake ardrone_autonomy

Importante observar que se ha construido sin errores los 21 paquetes. Los diferentes paquetes utilizados en este proyecto dependerán de este driver.

NOTAS:

- Algunos paquetes depende a su vez de otros paquetes deberán ser instalados, las dependencias las encontramos en el archivo manifest.xml.
- Debemos calibrar la camera para un óptimo funcionamiento, una vez hecho ROS creará unos archivos .yaml. Para calibrar podemos seguir los pasos del tutorial de calibración.

ANEXO III

- Calibrar cámara ardrone utilizando ROS.

Para un óptimo uso de las cámaras del ardrone debemos calibrarlas. Ya que usamos ROS, para el desarrollo del proyecto podemos utilizar un paquete llamado "calibration_camera" también basado en ROS.

Para calibrar podemos seguir estos pasos:
Debemos tener en cuenta que para que haya una conexión con el cuadricóptero para ello usaremos el paquete ardrone_autonomy antes de iniciar la calibración, se encuentra en el anexo I.

Instalación del paquete camera calibration:

Abrimos una terminal y accedemos a nuestro workspace de ROS, para ello tecleamos en la terminal:
$ roscd

Descargamos el paquete añadiendo otra línea a la terminal:
$ svn checkout https://code.ros.org/svn/rospkg/stacks/image_pipeline/branches/image_pipeline-1.8

Instalamos el paquete camera_calibration

$ rosdep install camera_calibration
$ rosmake camera_calibration

Cámara frontal.

Empezamos calibrando la cámara frontal en nuestro ardrone para ello escribimos en una terminal:
$ rosrun camera_calibration cameracalibrator.py --size [SIZE] --square [SQUARESIZE] image:=/ardrone/front/image_raw camera:=/ardrone/front

- En [SIZE] escribimos el número de cuadrados que tenemos en nuestro checkerboard.
- También debemos en [SQUARESIZE] escribir el lado de los cuadrados en metros.

Un ejemplo de calibración con todos los datos sería este:

$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.108 image:=/ardrone/front/image_raw camera:=/ardrone/front

Para calibrar la cámara correcta debe reconocer bien los cuadrados y tener un movimiento del checkerboard para realizar la completa calibración. Impreso el checkerboard, en este caso se imprimió en un A4 tenemos 8x6 cuadrados y las medidas son 2,95 cm.

Iniciamos ROS:
$ roscore

Iniciamos el driver de ardrone:
$ rosrun ardrone_autonomy ardrone_driver

Abrimos otra terminal e iniciamos calibration_camera con el número y medida correcta de nuestros cuadrados.

$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.0295 image:=/ardrone/front/image_raw camera:=/ardrone/front

Se abrirá una ventana e inicia camera calibration. Es momento de coger el checkerboard y empezar con la calibración.

Debemos mantenernos a una distancia que permita poder ver todos los cuadrados.

En este momento empezaremos moviendo el checkerboard hasta la izquierda y luego hacia la derecha.

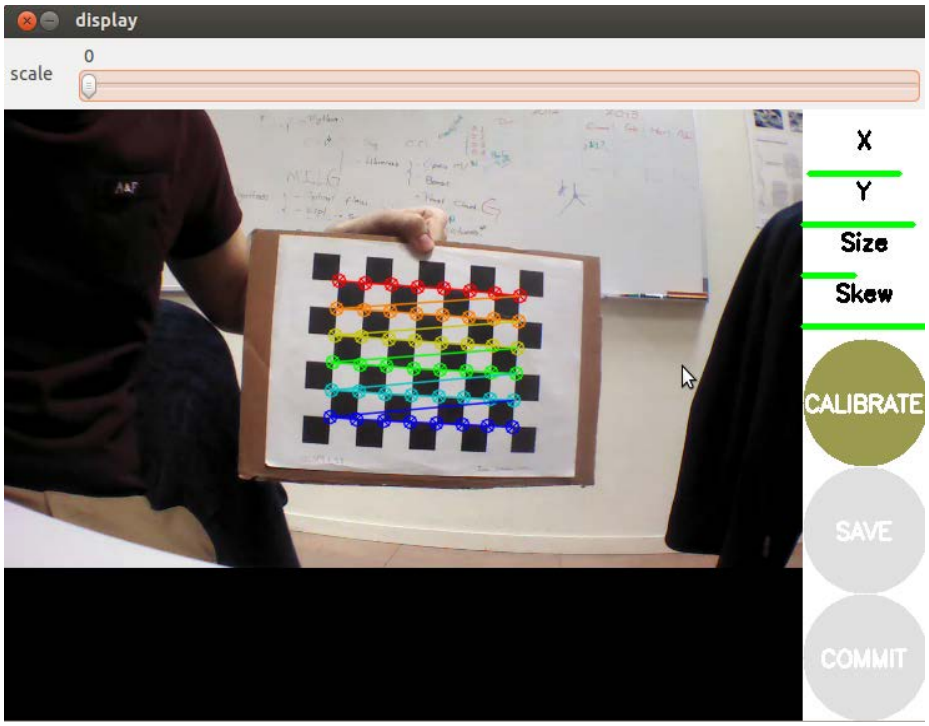

Ilustración 41. Camera_calibration (1)

Una vez se cambie a verde la barra del eje X, podemos calibrar el eje Y. Esto lo hacemos moviendo la hoja de checkerboard desde arriba hacia abajo.

Una buena calibración se debe hacer en los 3 ejes, para hacerlo en el eje Z demos mover hacia la cámara y luego alejar nuestra plantilla de calibración, en camera_calibration el eje Z está representado en la barra Size.

Finalmente inclinaremos un poco la plantilla repitiendo los anteriores movimientos para finalizarla calibración.

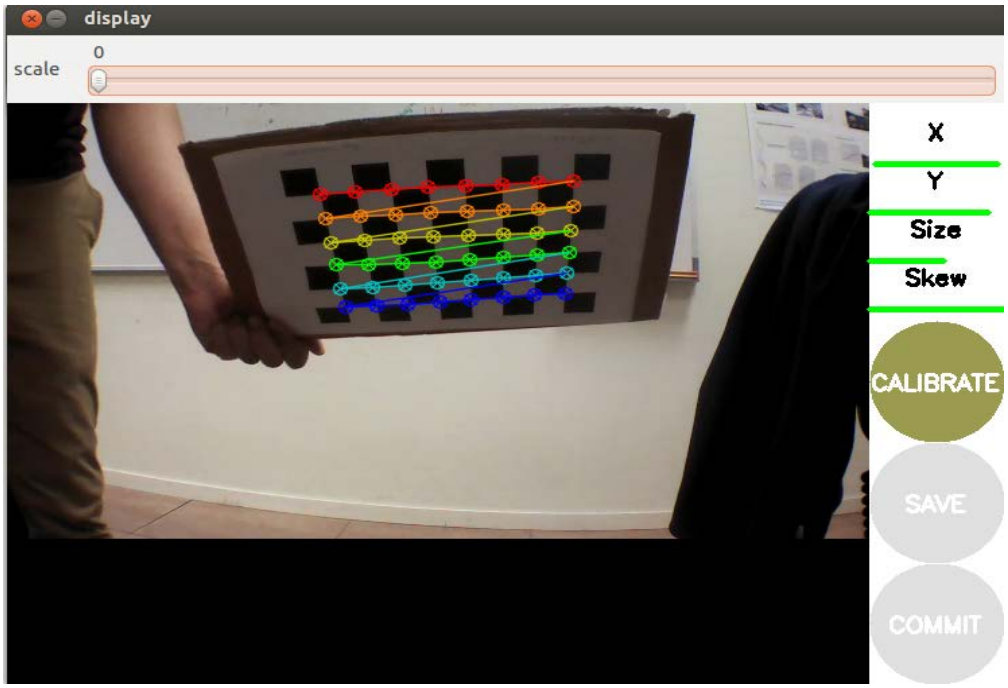

Ilustración 42. Camera_ calibration (2)

Si todo es correcto nos dejara pulsar el botón calibrate, y tendremos resuelto la calibración. Si se ha realizado una calibración correcta aceptamos dando al botón "Commit" y creara el archivo .yaml de calibración.

Cámara vertical.

El sistema de calibración es el mismo pero debemos cambiar en la línea de código front por bottom.

Un ejemplo:

$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.295 image:=/ardrone/bottom/image_raw camera:=/ardrone/bottom

Debemos seguir los mismos pasos que con la cámara anterior, moviendo nuestro checkerboard como en el caso anterior, y aceptando la calibración con el botón calibrate. Nos creara otro archivo *.yaml* para esta cámara vertical.