ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

CONTROL POR VISIÓN DE UN CUADRICÓPTERO UTILIZANDO ROS

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y CONTROL

MARÍA TERESA CALDERÓN JÁCOME
materesa28@hotmail.es

DIEGO JAVIER MALDONADO ANDRADE
djma_18@hotmail.com

DIRECTOR: Ph.D. GEOVANNY DANILO CHÁVEZ GARCÍA
danilo.chavez@epn.edu.ec

Quito, Octubre 2014

DECLARACIÓN

Nosotros, María Teresa Calderón Jácome y Diego Javier Maldonado Andrade, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____	_____
María Teresa Calderón Jácome	Diego Javier Maldonado Andrade

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por María Teresa Calderón Jácome y Diego Javier Maldonado Andrade, bajo mi supervisión.

_____

Dr. Ing. Danilo Chávez

DIRECTOR DEL PROYECTO

AGRADECIMIENTO

Quiero agradecerles a mis papás porque sin su apoyo nada de esto hubiera sido posible, porque sin su cariño no habría llegado tan lejos, agradecerles por todas sus enseñanzas y todos sus consejos, porque más que mis papás son mis amigos. A mis hermanos por toda la paciencia que me han tenido este tiempo y por siempre darme un motivo para reír.

A Diego, por su amistad de todos estos años y por el apoyo que siempre me brindó. Porque gracias a su trabajo y su dedicación podemos hoy cumplir una meta muy importante.

Un agradecimiento especial para el Dr. Danilo Chávez, por su guía y su apoyo a lo largo del desarrollo de este proyecto.

A todos mis profesores, porque gracias a ellos estoy culminando esta etapa de mi formación profesional. Y a la Escuela Politécnica Nacional porque ha sido realmente un honor formar parte de ella.

A Telmo, mi mejor amigo, que siempre ha estado conmigo desde el primer día que nos conocimos y Luismi, mi hermano postizo, el que nunca dudo en darme un abrazo o una palabra cuando lo necesité.

Mis demás amigos que siempre me brindaron su cariño y su apoyo, les agradezco por el tiempo y por todos los momentos increíbles que pasamos juntos.

A mis tíos y mis primos, gracias, por estar ahí y por compartir este momento tan especial conmigo.

María Teresa Calderón

DEDICATORIA

Este proyecto se lo quiero dedicar en primer lugar a mi familia, a mis papás por ser los pilares de mi vida, porque gracias a ellos soy lo que soy y estoy en este punto tan importante, porque me enseñaron el valor del trabajo, del esfuerzo y de la dedicación. A mis hermanos por brindarme su apoyo incondicional, por sus palabras y por todo su cariño.

También quiero dedicar este trabajo a una persona muy especial que quiso compartir este momento conmigo pero que no pudo hacerlo, a mi tío Hermensul, que estoy segura desde algún lugar me está viendo y se siente orgulloso de mí.

A mis tíos y mis primos que siempre han confiado en mí y que jamás dudaron que este día llegaría. Y especialmente a Mamita Soledad por todo el amor que siempre me ha dado.

María Teresa Calderón

AGRADECIMIENTO

A mis padres por ser quienes me dieron su apoyo incondicional para alcanzar las metas que me he propuesto, gracias por sus consejos y enseñanzas, y gracias por ser el ejemplo de esfuerzo y esmero para salir siempre a adelante a pesar de las circunstancias, y lo más importante por su amor y cariño con la que me han cuidado.

A mi hermano Edison, gracias por apoyarme desde el primer día en que llegue a quito, por tu valiosa compañía, porque de ti he aprendido la importancia de esforzarse para alcanzar cada vez metas más altas, a mi cuñada Edith, gracias por tu apoyo y solidaridad permitiéndome vivir durante todo este tiempo con ustedes.

Al Ph.D. Danilo Chávez por su guía y ayuda en la consecución de los objetivos de este proyecto.

A María Teresa, gracias principalmente por tu amistad, por ser una gran amiga y compañera a lo largo de nuestra formación profesional, gracias por todo tu esfuerzo y dedicación plasmada en este proyecto.

Especialmente quiero agradecer Sandro Jua, por la oportunidad de trabajar inicialmente este proyecto contigo y encaminarme de la mejor manera para cumplir con los objetivos y metas propuestas, de seguro este proyecto sin tu colaboración no tendría el mismo alcance. Estoy eternamente agradecido.

A mi familia, por ser siempre ese apoyo fundamental y estar siempre pendiente de mí y mi bienestar. Gracias por sus palabras de aliento y consejos.

A mis amigos y amigas, gracias por todos los momentos vividos con ustedes, por la valiosa amistad compartida.

Diego Maldonado

DEDICATORIA

Este trabajo principalmente quiero dedicarlo a DIOS, por el cambio que solo Él pudo hacer en mí, al depositar su palabra y su amor en mi corazón, porque todos y cada uno de los logros que he alcanzado han sido por la manifestación de su mano poderosa en mi vida. “En el temor a Dios está la sabiduría, y en apartarse del mal la inteligencia”

De igual forma lo dedico muy orgullosamente a mis Padres Victoria Andrade y Antonio Maldonado, gracias por ser el ejemplo a seguir en mi vida, por el amor, el cariño y el apoyo que siempre me han sabido brindan. Quiero decirles que de una u otra forma han sabido ser el pilar fundamental de mi educación, corrigiendo día a día mi comportamiento con lecciones y enseñanzas oportunas que guardo en mi corazón, enseñándome a valerme por mí mismo y dar lo mejor de mí a cada momento. Gracias por ser mis hermosos Padres los Amo con todo el corazón.

A mi hermosa hermanita Viky, y a mis hermanos Edison y Darío, siempre han sabido ser el ejemplo y apoyo incondicional en cada momento.

A Estefy Narváez por tu apoyo y cariño incondicional.

Diego Maldonado

CONTENIDO

RESUMEN

Los avances tecnológicos que se han producido en las últimas décadas, han creado nuevas ramas científicas y campos de investigación. Sin duda, uno de los más interesantes es la Robótica y su estrecha relación con la inteligencia artificial.

Al igual que en los humanos, la entrada de datos más importante es la visión. En un sistema inteligente, se trata de una cámara posiblemente con una serie de sensores; esto se conoce como *visión artificial*, que permite a un sistema percibir objetos y detalles de forma que pueda reaccionar y actuar frente a ellos. La visión artificial comprende una gran variedad de procesos ópticos mediante los cuales un sistema inteligente pueda ser capaz de encontrar información de un entorno para su interpretación mediante el uso del computador.

El comienzo de cualquier proceso de reconocimiento, es la cuantificación digital del entorno [9]. El procesamiento de una imagen captada con visión artificial, es elemental para hacer un reconocimiento preciso. Al procesar una imagen se elimina la información poco necesaria para identificar el objeto, extrayendo solo parámetros que permiten optimizar la identificación del mismo.

Hoy en día existen softwares para desarrollar este tipo de aplicaciones, pero en este trabajo se utiliza un framework relativamente nuevo diseñado exclusivamente para robótica; ROS (Robot Operating System), un marco de código abierto que proporciona toda una serie de servicios y librerías que simplifican considerablemente la creación de aplicaciones complejas para robots. A propósito del concepto de visión artificial tratado en el proyecto, ROS cuenta con el apoyo de una mega biblioteca de código abierto y libre de visón artificial, OpenCV (Open Source Computer Vision Library), cuya publicación se da bajo licencia BSD, lo que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

Estas plataformas de desarrollo de software más la versatilidad del cuadricóptero, juntos en una aplicación de visión artificial, abren la ventana para un gran avance en distintos campos como el de la seguridad.

PRESENTACIÓN

Este proyecto implementa el concepto de visón artificial; específicamente reconocimiento y seguimiento de objetos, con un hardware en particular: un cuadricóptero. Los cuadricópteros comercialmente conocidos como AR.Drones, son equipos que están tomando fuerza en diversidad de aplicaciones; los AR.Drones son vehículos aéreos controlados por sistemas remotos de control, de forma autónoma siguiendo una misión pre programada o desde tierra. Pueden llevar a cabo tareas que los vuelos tripulados consideran arriesgadas, gracias a su "precisión" y características de “sigilo y cautela”. Este trabajo se divide en 5 capítulos que detallan los procesos que se siguieron para implementar el control por visión artificial de un cuadricóptero.

En el primer capítulo se presenta el marco teórico acerca de los elementos claves que se manejan en este proyecto tanto en hardware como en software, dando una visión global de las herramientas empleadas en el desarrollo del mismo.

En el segundo capítulo se presenta el diseño de los algoritmos empleados para el reconocimiento de objetivos específicos dentro de un entorno. En este proyecto los algoritmos se desarrollan en función de dos características del objetivo: su color y su forma básica.

En el tercer capítulo se detalla la implementación del algoritmo de seguimiento después de identificado el objetivo y además se presenta el diseño de los controladores necesarios para un óptimo funcionamiento de la aplicación.

En el cuarto capítulo, se muestran las diferentes pruebas realizadas para verificar el correcto funcionamiento del proyecto; pruebas que involucran errores de iluminación, medición de tiempos de respuesta, estabilidad y precisión del seguimiento.

En el quinto capítulo, se muestran las conclusiones y recomendaciones formuladas después de analizar los problemas que se presentaron y las soluciones que se encontraron para los mismos.

CAPÍTULO 1

MARCO TEÓRICO

1.1 INTRODUCCIÓN

En este capítulo se van a tratar los principales conceptos, información y características de todos los elementos que forman parte de este proyecto. Incluyendo las bases y características del *ROBOT OPERATING SYSTEM (ROS)*; la plataforma que se manejará para el desarrollo del software del proyecto y que constituye el punto más trascendental del mismo.

1.2 CONCEPTOS GENERALES

1.2.1 VANT: VEHÍCULO AÉREO NO TRIPULADO

Un vehículo aéreo no tripulado, UAV, por sus siglas en inglés de Unnamed Aerial Vehicle, es una aeronave que vuela sin tripulación. Hay una amplia variedad de formas, tamaños, configuraciones y características de vehículos aéreos no tripulados conocidos como VANT. En la actualidad se emplea el control autónomo de los VANT; en este sentido se han creado dos variantes: algunos son controlados desde una ubicación remota y otros vuelan de forma autónoma sobre la base de planes de vuelo pre-programados usando sistemas complejos de automatización dinámica. [1]

Figura.1.1 VANT Pioneer en misión de vigilancia sobre Irak [1]

Los vehículos aéreos no tripulados militares realizan tanto misiones de reconocimiento como de ataque; además se usan en aplicaciones civiles, contra incendios o seguridad civil.

Los VANT han demostrado en diferentes escenarios el gran potencial que tienen, en cuanto a obtención, manejo y transmisión de información, y gracias a la aplicación de nuevas técnicas de protección de información resulta posible conseguir comunicaciones más seguras, más difíciles de detectar e interferir. [1]

Un avión no tripulado típico está hecho de materiales compuestos ligeros para reducir el peso, aumentar la maniobrabilidad y superar grandes alturas. Están equipados con cámaras infrarrojas, GPS, misiles guiados por láser y otros sistemas secretos. Un sistema VANT tiene dos partes, el propio avión no tripulado y el sistema de control; el avión es controlado desde tierra por pilotos o programadores de combate entrenados. La nariz del AR.Drone es donde todos los sensores y los sistemas de navegación están presentes. Los materiales de ingeniería utilizados para construir el quadrotor son componentes muy complejos que pueden absorber las vibraciones lo que disminuye el ruido producido. [2]

Figura.1.2 VANT que puede ser lanzado con la mano [2]

1.2.1.1 Ventajas de los VANT

- Posibilidad de uso en áreas de alto riesgo o de difícil acceso.
- No requiere la actuación de pilotos en la zona de combate.

1.2.1.2 Desventajas de los VANT

1.2.1.2.1 Desventajas técnicas

- El enlace vía satélite podría eventualmente ser hackeado en tiempo de guerra y romperse así el canal de comunicaciones entre el operador en tierra y el VANT.
- Tiempo de retardo entre emisión de instrucciones y su recepción, para proceso y ejecución, en condiciones críticas sería fatal para la aeronave.
- Influencia en su funcionamiento por los fenómenos físicos, como sol, mal clima, tormentas de rayos, etc.
- Capacidad de vuelo limitada por el tipo de combustible, fuente de energía, tamaño, alcance y su sistema de navegación.

1.2.1.2.2 Desventajas éticas

- Posibilidad de que la inteligencia artificial del VANT pudiera determinar por sí misma los objetivos a atacar.
- Insensibilidad sobre las consecuencias de la guerra.
- Comercialización no controlada, pudiendo ser adquiridos por personas o grupos de dudosa ética.

1.2.1.2.3 Desventajas económicas

- El alto coste de su adquisición y mantenimiento dificulta su uso civil. [1]

Figura.1.3 S4 Ehécatl mexicano en despegue [1]

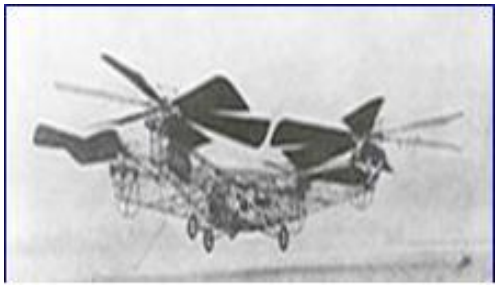


Figura.1.4 Quadrirotor de Bothezat, 1923 [3]

1.2.1.3 Aplicaciones

Se pueden aplicar en ambientes de alta toxicidad química y radiológica, en los que sea necesario tomar muestras con alto peligro para vidas humanas y realizar tareas de control de ambiente.
Además, pueden cooperar en misiones de control del narcotráfico y contra el terrorismo eventualmente. En el ámbito de la observación de la tierra, los VANT tienen múltiples aplicaciones y posibilidades en el mercado civil:

- Cartografía: realización de modelos, elevaciones de terreno de alta resolución.
- Agricultura: gestión de cultivos.
- Servicios forestales: seguimiento de las áreas boscosas, control de incendios.
- Geología.
- Hidrología.
- Medio ambiente: estado de la atmósfera.
- Control de obras y evaluación de su impacto.
- Seguimiento de la planificación urbanística.
- Gestión del patrimonio.
- Seguridad y control fronterizo. [4]

Se debe saber que la duración máxima volando del cuadricóptero, solo es limitada por su combustible y por su sistema de vuelo.

1.2.2 CUADRICÓPTERO - QUADCOPTER

Figura.1.5 Maker Faire quadcopter en Garden City, Idaho [3]

Un cuadricóptero, también conocido como helicóptero quadrotor, es un helicóptero que es levantado y propulsado por cuatro rotores. Los cuadricópteros son clasificados como helicópteros, contrarios a las aeronaves de ala fija, porque su elevación se genera por un set de perfiles aerodinámicos giratorios de acordes estrechos. Los cuatro rotores están generalmente colocados en la extremidad de una cruz, a fin de evitar que el aparato se tumbe respecto a su eje de orientación, y utilizan 2 juegos de hélices inclinadas fijas idénticas; 2 en sentido horario (CW) y 2 en sentido antihorario (CCW), usando la variación de RPM para controlar la elevación y el torque. El control del movimiento del vehículo se consigue mediante la alteración de la velocidad de rotación de uno o más discos de rotor. [3]

Los diseños recientes de cuadricópteros se han hecho populares en el área de *vehículos aéreos no tripulados (VANT)* de investigación. Estos vehículos usan un sistema de control electrónico y sensores electrónicos para estabilizarse; con su pequeño tamaño y maniobrabilidad ágil, pueden volar tanto en interiores como en exteriores [3]. Hay muchas ventajas de los cuadricópteros sobre helicópteros de tamaño comparable. En primer lugar, los cuadricópteros no requieren conexiones mecánicas para variar el ángulo de paso de las palas del rotor a medida que giran, esto simplifica el diseño y mantenimiento del vehículo [3][5]. En segundo lugar, el uso de cuatro rotores permite que cada rotor tenga un diámetro menor que el rotor del helicóptero equivalente, permitiendo tener menor energía cinética durante el vuelo, esto reduce el daño causado si los rotores golpearan algo.

Algunos cuadricópteros de pequeña escala tienen marcos que encierran a los rotores, permitiendo vuelos a través de entornos más difíciles. Debido a su facilidad de construcción y de control, los cuadricópteros son usados frecuentemente como proyectos de modelos de aeronaves de aficionados.

1.2.2.1 Avances Recientes

En las últimas décadas, los VANT a pequeña escala se han vuelto más comunes, y la necesidad de aeronaves con mayor maniobrabilidad y capacidad de estar suspendidas en el aire de forma estable e inmóvil ha dado lugar a un aumento en la intensidad de la investigación sobre cuadricópteros. La investigación de

vanguardia sigue aumentado la viabilidad de cuadricópteros haciendo avances en la comunicación multi – nave, exportación del medio ambiente y maniobrabilidad, cualidades que combinadas permiten la ejecución de misiones autónomas avanzadas que en la actualidad no son posibles con ningún otro vehículo. [6]

1.2.2.2 Usos

1.2.2.2.1 Plataforma de investigación

Los cuadricópteros son una herramienta útil para los investigadores universitarios para poder probar y evaluar nuevas ideas en diferentes campos, incluyendo la teoría de control de vuelo, navegación, sistemas de tiempo real, y la robótica. Los proyectos sobre cuadricópteros comúnmente combinan ciencias de la computación, ingeniería eléctrica y especialistas en ingeniería mecánica [7]. Hay algunos laboratorios de investigación de ingeniería de clase mundial hoy en día desarrollando técnicas avanzadas de control y aplicaciones para cuadricópteros.

1.2.2.2.2 Militar y Policial

Los VANT's son usados para vigilancia y reconocimiento, por organismos militares y policiales, así como en misiones de búsqueda y rescate en entornos urbanos. Los cuadricópteros más populares de aplicaciones militares pertenecen a las Fuerzas Armadas de Estados Unidos, el *MQ – 9 Reaper* y el *MQ – 1B Predator*. AR.Drones de vigilancia y recopilación de información, o hechos para la caza pudiendo transportar hasta cuatro misiles [2].

Figura.1.6 Predator lanzamiento de un misil Hellfire [2]

1.2.2.2.3 Comercial

El mayor uso de los cuadricópteros ha estado en el campo de las imágenes aéreas, aunque, en los EE.UU., es ilegal el uso de vehículos controlados a distancia con fines comerciales. Hoy por hoy se propone que los cuadricópteros sean maquetas teledirigidas no tripuladas en aeromodelismo o por ejemplo para luchar contra las heladas en la agricultura. [6]

Figura.1.7 Quadcopter comercial, con cámara controlado por radio. [6]

Si nos fijamos en los *usos principales de los cuadricópteros* que se han detallado en los párrafos anteriores se pueden listar a manera de resumen los siguientes:

- Proporcionar apoyo táctico.
- Comprobar si hay dispositivos peligrosos en carreteras o áreas de aterrizaje.
- Observar el tráfico y el comportamiento público.
- Proporcionar apoyo aéreo.
- Seguir o atacar blancos sospechosos. [2]

1.2.3 CONTROL DE VUELO

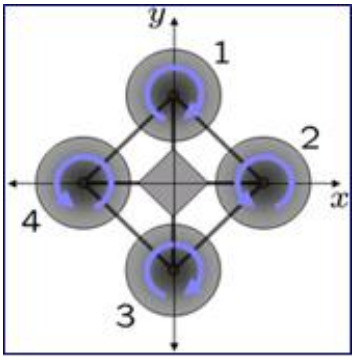
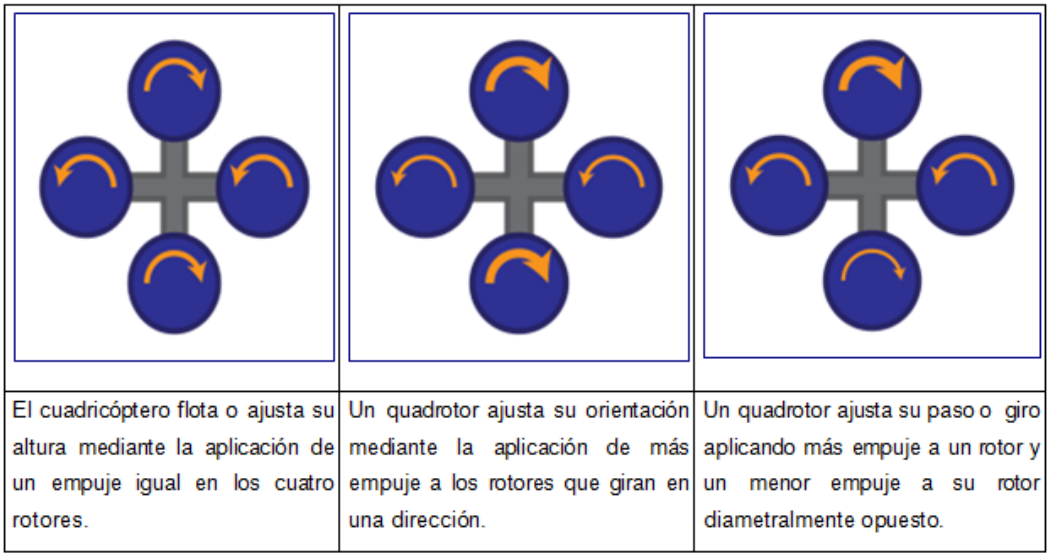


Figura.1.8 Esquema de los torques de reacción en cada motor de un cuadricóptero. [8]

Cada rotor produce tanto un empuje como un par alrededor de su centro de rotación, así como una fuerza de arrastre opuesta a la dirección de vuelo. Si todos los rotores están girando a la misma velocidad angular, con los rotores 1 y 3 girando en sentido horario y los rotores 2 y 4 en sentido antihorario; el par aerodinámico de red y por lo tanto la aceleración angular alrededor del eje de orientación son exactamente cero, lo que implica que el rotor de orientación estabilizador de los helicópteros convencionales no es necesario.

El derrape es inducido por una mala adaptación del equilibrio en pares aerodinámicos. [8]



El cuadricóptero flota o ajusta su altura mediante la aplicación de un empuje igual en los cuatro rotores.	Un quadrotor ajusta su orientación mediante la aplicación de más empuje a los rotores que giran en una dirección.	Un quadrotor ajusta su paso o giro aplicando más empuje a un rotor y un menor empuje a su rotor diametralmente opuesto.

Figura.1.9 Movimientos del cuadricóptero [8]

1.2.4 VUELO AUTÓNOMO

A menudo los cuadricópteros y otros multicópteros pueden volar de forma autónoma. Muchos controladores de vuelo modernos utilizan un software que permite al usuario marcar waypoints o puntos de ruta, que son las trayectorias que el cuadricóptero volará realizando tareas como aterrizaje o despegue. [3]

Figura.1.10 El X-UFO es un juguete quadrotor teledirigido. [3]

1.2.5 FUTURO DE LOS CUADRICÓPTEROS

Una de las mayores preocupaciones sobre el uso de aviones no tripulados es en cuanto a servicios militares; puesto que no se ha explicado aún por parte de los países que cuentan con flotas de VANT armados, cómo se justifica el uso de éstos para atacar y matar a personas conforme al derecho internacional.

Los fabricantes de VANT están buscando usos civiles de los cuadricópteros en campos como la teledetección con el fin de ampliar sus mercados y esto incluye el uso de aviones no tripulados para vigilancia doméstica. Hoy es posible el reconocimiento de caras, conductas y el monitoreo de conversaciones individuales; lo que significa una dramática expansión del área de la vigilancia.

Figura.1.11 Cámara FLIR montado en un VANT [2]

1.3 HARDWARE

Los dispositivos a usarse en este proyecto incluyen un cuadricóptero junto con una laptop de planta en tierra. El cuadricóptero es el dispositivo principal y todos

los programas son ejecutados en la laptop, la cual enviará todas las órdenes al quadcopter vía Wi – Fi.

1.3.1 EL CUADRICÓPTERO: PARROT AR.DRONE 2.0

Figura.1.12 AR.Drone Parrot 2.0 [11]

El Parrot AR.Drone es un vehículo aéreo no tripulado radiocontrolado de uso recreativo civil. Funciona propulsado por cuatro motores eléctricos en configuración *cuadricóptero* y es similar en su estructura básica y aerodinámica a otros modelos radiocontrolados, pero se diferencia de todos ellos en que cuenta con un microprocesador y una serie de sensores entre los cuales se incluyen dos cámaras que le permiten captar lo que ocurre a su alrededor, más un conector Wi–Fi integrado que da la posibilidad de vincularse a dispositivos móviles personales con sistemas operativos iOS, Android o Linux. Es decir, es posible un control directo del cuadricóptero desde un dispositivo móvil, mientras se reciben a la vez imágenes y datos de telemetría que los sensores del AR.Drone receptan.

Figura.1.13 AR.Drone 2.0 comandado por un Smartphone [11]

Este cuadricóptero se usa en universidades y centros de investigación para precisamente probar prototipos o algoritmos en proyectos de robótica, inteligencia

artificial y visión por computador [13]. La razón para usar este dispositivo en este tipo de proyectos es porque no es costoso en comparación con otros módulos y además es fácil de manejar pues existen drivers para conectarse con el AR.Drone directamente y enviarle comandos de velocidad o posicionamiento. [11]

Hay que mencionar que el cuadricóptero que se está manejando no es el más adecuado para aplicaciones reales o prácticas en áreas de investigación y estudio porque es categorizado como juguete y sus motores no resultan ser potentes y no ha sido diseñado para levantar peso extra.

1.3.1.1 Características Generales

Su estructura es muy sencilla y totalmente modular permitiendo un recambio simple de todas sus piezas. Cuenta con una estructura de soporte en forma de cruz de tubos de fibra de carbono sobre el que se montan los motores, la placa madre y la placa de navegación. El cuerpo del AR.Drone es de espuma de polipropileno lo que brinda protección contra golpes a todo el sistema electrónico.

Para el control del cuadricóptero, este se tiene que conectar a un iOS, a un dispositivo Android o a una PC con Wi –Fi y una aplicación de control, como se mencionó en párrafos anteriores. El AR.Drone tiene dos cámaras que pueden proyectarse en la pantalla del dispositivo, compatibilidad con juegos de vuelo de realidad aumentada y un altímetro con sensores de ultrasonido que entre otras cosas permite que el AR.Drone *levite* completamente estabilizado o aterrice de forma automática en caso de que pierda la señal. [13]

El cuadricóptero AR.Drone 2.0 tiene 2 carcasas diferentes hechas de espuma de polipropileno flexible, que ofrecen una cubierta rígida resistente a los impactos. Uno de los cascos es para usar el cuadricóptero en interiores con espuma alrededor de las hélices para protegerlas, y el otro casco es para vuelos en exteriores sin protección para las aspas. Los cascos están sujetos al AR.Drone mediante 2 imanes, uno ubicado en el casco y otro en el cuerpo. Lleva el portapilas montado en el cuerpo principal con espuma que absorbe las

vibraciones de los motores y la batería se localiza hacia la parte trasera del dispositivo, para mantener correcto el centro de gravedad del AR.Drone. [11]

Figura.1.14 AR.Drone con casco exterior [11]

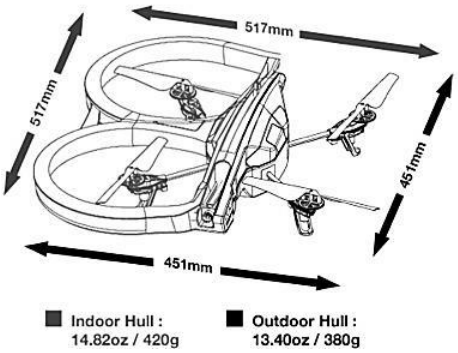


Figura.1.15 Dimensiones AR.Drone [11]

En la figura.1.16 puede verse al AR.Drone con su carcasa para interiores que sirve para proteger a las hélices. Pueden verse los leds indicadores encendidos en verde, señalando que los motores han pasado el control de inicialización y se encuentra listo para volar. [13]

Figura.1.16 AR.Drone con casco para interiores [13]

Su cámara frontal es de alta definición (720p 30FPS), mientras que la cámara ventral es de menor definición (QVGA de 60FPS). Además cuenta con un girómetro, acelerómetro y magnetómetro de 3 ejes, sensor de presión, sensor ultrasónico para medir la altitud desde el suelo hasta el AR.Drone; una batería de litio y su respectivo cargador. [11]

Es controlado por 4 rotores y su velocidad se ajusta de forma individual. Todos los motores contribuyen para elevar y mantener fijo al cuadricóptero en el aire; dos de

ellos giran en sentido horario y los otros dos en sentido antihorario para cancelar así sus torques respectivos.

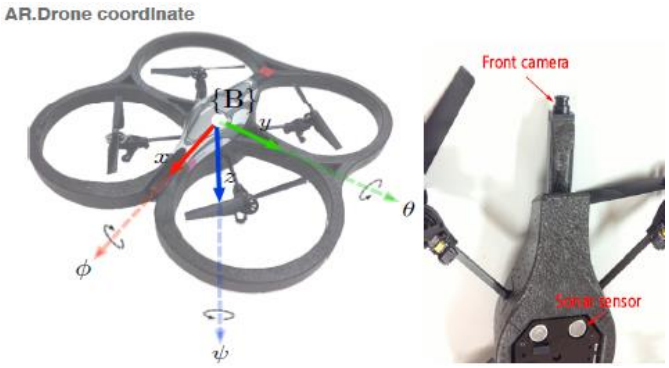


Figura.1.17 Plataforma AR.Drone, cámara frontal, sensor ultrasónico. Ejes de rotación del AR.Drone [14]

1.3.1.2 Especificaciones Técnicas:

Especificaciones Físicas:

- Dimensiones:
  - Con casco 52.5 x 51.5 cm
  - Sin casco 45 x 29 cm
- Peso:
  - 380g con la cubierta del casco para exteriores
  - 420g con la cubierta del casco para interiores
- Velocidad de marcha 5 m/s (18 km/h).
- Techo máximo: limitado por el alcance de la conexión Wi – Fi (entre 50 y 120 metros dependiendo de las condiciones climáticas)
- Alcance máximo: limitado por el alcance de la conexión Wi – Fi
- Autonomía de vuelo promedio: 12 minutos

Sistema Informático Integrado:

La tecnología incorporada en el AR.Drone 2.0 ofrece funciones de control de extrema precisión y estabilización automática.

- Procesador de 1 GHz y 32 bit ARM Cortex A8 con vídeo DSP TMS320DMC64X de 800 MHz.

- Sistema operativo con núcleo Linux 2.6.32
- Memoria RAM DDR2 de 1GB a 200 MHz.
- USB 2.0 de alta velocidad para extensiones
- Módem Wi – Fi b g n. [12]
- **IMU** (Inertial Measurement Unit): registra los datos de velocidad y orientación, usando una combinación de acelerómetros y girómetros permitiendo un vuelo y "estado de levitación" mucho más estables [15].
  La IMU puede ayudar a reconstruir por ejemplo los movimientos reales que hace el AR.Drone para un ambiente virtual previamente descrito.

Sistema de Seguridad y Estructura Aerodinámica:

- Bloque automático de las hélices en caso de contacto.
- Interfaz de control con botón de emergencia para detener los motores.
- Cuatro hélices de alta eficiencia
- Estructura modular de fibra de carbono.

Asistencia Electrónica:

- Girómetro de 3 ejes con una precisión de 2000°/seg.
- Acelerómetro de 3 ejes con una precisión de +/- 50 mg.
- Magnetómetro de 3 ejes con una precisión de 6°
- Sensor de presión con una precisión de +/- 10 Pa
- Sensores de ultrasonido para medir la altitud de avance.

Cámara Ventral:

- Cámara ventral QVGA de 60 FPS para medir la velocidad de avance. [12]
- Cámara de sensor CMOS, alta velocidad de lente diagonal. 64º de amplitud.
- Resolución 176 x 144 pixeles.

Grabación de vídeo HD_ Cámara Frontal:

- Cámara HD, 720p 30 FPS.
- Lente gran angular: Diagonal 92°
- Perfil base de codificación H264.

- Transmisión de baja latencia.
- Almacenamiento de vídeos durante el vuelo con el dispositivo remoto.
- Foto JPEG.
- Almacenamiento instantáneo de vídeos con Wi – Fi, directamente en un dispositivo remoto o en una memoria USB. [12]

Motores y Energía:

Vuela alto, vuela rápido y bastante lejos del suelo.

- 4 motores “inrunner” sin escobillas. 14,5 W 28500 RPM
- Engranajes de bajo ruido para 1 reductor de propulsión 8.75
- Eje de transmisión de acero templado.
- Cojinete de bronce auto – lubricante.
- Resistencia aerodinámica específica de alta propulsión para ofrecer una excelente maniobrabilidad.
- 1 CPU AVR de 8 MIPS por controlador de motor.
- Controlador de motor totalmente reprogramable.
- Batería de ión litio de 3 celdas, con una capacidad de 1000mA/h con un voltaje nominal de 11.1 V, y una capacidad de descarga de 10 Coulombios.
- Batería incluye un módulo de circuito de protección que evita que la batería se sobrecargue, sobredescargue o corto circuito. [12]
- La batería cumple con los estándares de seguridad UL2054. Tiene un tiempo de carga de 90 minutos y permite una autonomía de vuelo de entre 12 y 15 minutos. [13]

Figura.1.18 Batería del AR.Drone 2.0 [11]

Estructura Robusta:

- Fibra de alta calidad (30%) cargada con piezas plásticas de nylon.
- Espuma para aislar el centro inercial de las vibraciones del motor.

- Casco EPP inyectado por un molde de metal.
- Nanorevestimiento que repele los líquidos en los sensores de ultrasonidos.
- Completamente reparable: todas las piezas e instrucciones para la reparación están disponibles en internet. [12]

Compatibilidad:

Mac

- Aplicaciones IOS

Windows

- Ardrone_net

Linux

Sistema UNIX (Ubuntu / Linux)

- ROS (Robot Operating System)
- OPEN CV
- AR.Drone_autonomy: driver para manipulación del AR.Drone

1.3.1.3 Pilotaje

Gracias a las cámaras frontal y ventral y al sistema operativo basado en Linux del AR.Drone, éste es capaz de transmitir la señal de video y los datos de telemetría al dispositivo móvil que lo controla. El “piloto” puede ver en el dispositivo, lo que ven las cámaras del AR.Drone y recibir datos como aceleración, altitud, ángulos, etc.

En el sistema de control para iOS o Android el pilotaje se realiza por medio de uno o dos joystick virtuales sobreimpuestos en la pantalla de control de la aplicación. También se puede efectuar el control en base a los acelerómetros del dispositivo móvil, dirigiendo el AR.Drone de manera intuitiva inclinando a un lado u otro el dispositivo móvil. Al controlar el AR.Drone desde un computador portátil, el software permite hacerlo por medio de un joystick o de un joypad.

El despegue y aterrizaje es vertical y totalmente automático controlado por completo por el sistema operativo del AR.Drone. El driver de control asume el

mando del vehículo en caso de que se pierda la señal Wi – Fi, manteniéndolo suspendido a la altura en la que se encontraba al momento de perder la señal o aterrizándolo dependiendo de la configuración previa dada por el usuario. [13]

1.3.1.4 Diseño de Red

Para correr todo el sistema se utiliza una laptop base conectada vía Wi – Fi con el cuadricóptero. El AR.Drone tiene su “computador a bordo” para estabilización, control de movimiento y adquisición de imágenes con la cámara. Hay que tener en cuenta que no se pueden procesar las imágenes captadas por las 2 cámaras del AR.Drone al mismo tiempo. Se tiene que trabajar con la cámara ventral o bien con la frontal, y durante el vuelo la cámara que esté trabajando puede ser cambiada.

Cada dispositivo AR.Drone 2.0 contiene su propio *router inalámbrico*. El AR.Drone actúa como un servidor inalámbrico y asigna a sí mismo (a través de su servidor DHCP) una dirección IP fija de 192.168.1.1. Todos los nodos (programas) se están ejecutando en la laptop, así como el software del controlador AR.Drone para convertir los mensajes de los programas en los mensajes del AR.Drone. [11]

Todos los datos de procesamiento adquiridos desde los sensores y cámaras del AR.Drone y todos los programas de control que se ejecuten en el equipo portátil y solo los comandos de velocidad se envían al AR.Drone vía Wi – Fi. Así que casi todo el cálculo se lleva a cabo en el ordenador portátil, y no hay cálculos que se realicen en el cuadricóptero, excepto para la estabilización y adquisición de imágenes.

1.4 SOFTWARE

1.4.1 UBUNTU LINUX

El sistema operativo usado para el desarrollo de este proyecto es Ubuntu 12.04; ya que es la última actualización estable disponible. Ubuntu está basado en una distribución Devian – Linux y además es software libre y de código abierto. Una de las razones por las que se usa este sistema operativo es porque es capaz de

soportar los diferentes softwares usados en el proyecto de manera eficiente. Ubuntu, es un sistema operativo comercial como Windows o Mac OS, que tiene una buena interfaz gráfica.

Entre otras cosas, tiene un escritorio, organización de carpetas, aplicaciones de oficina, navegador web y un terminal. Hay muchas aplicaciones disponibles para descargar de manera gratuita como editores de imagen, juegos, herramientas de dibujo, sonido y reproductores de vídeo, entre otras cosas.

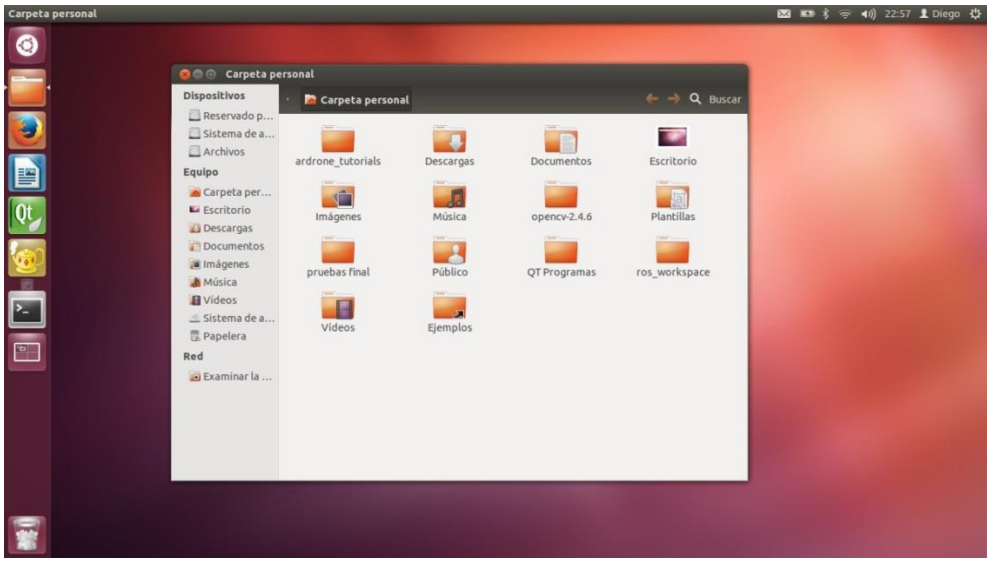


Figura.1.19 Captura de pantalla del escritorio y carpeta del sistema de Ubuntu

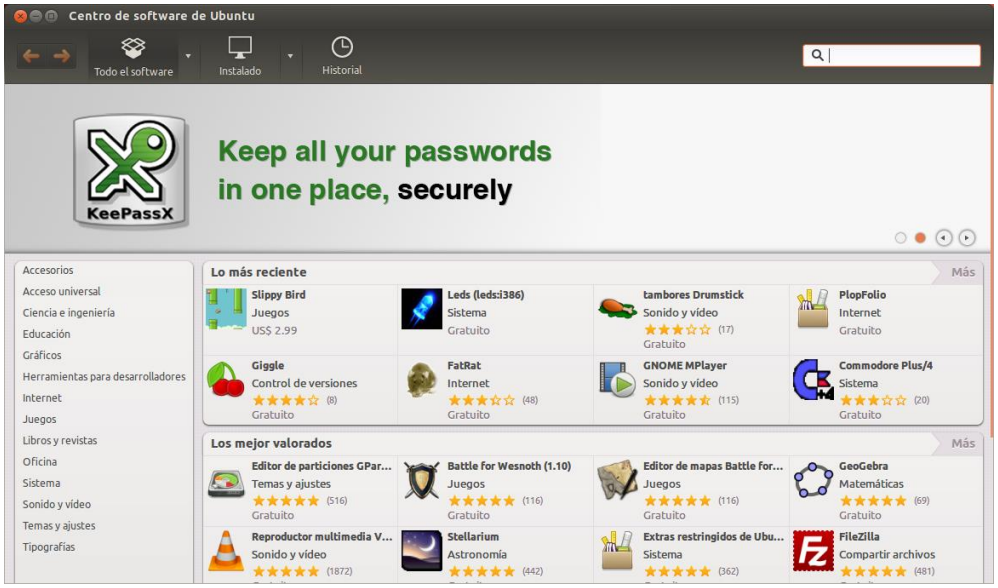


Figura.1.20 Captura de pantalla del Centro de software de Ubuntu

El terminal es una herramienta esencial en el desarrollo de este trabajo.

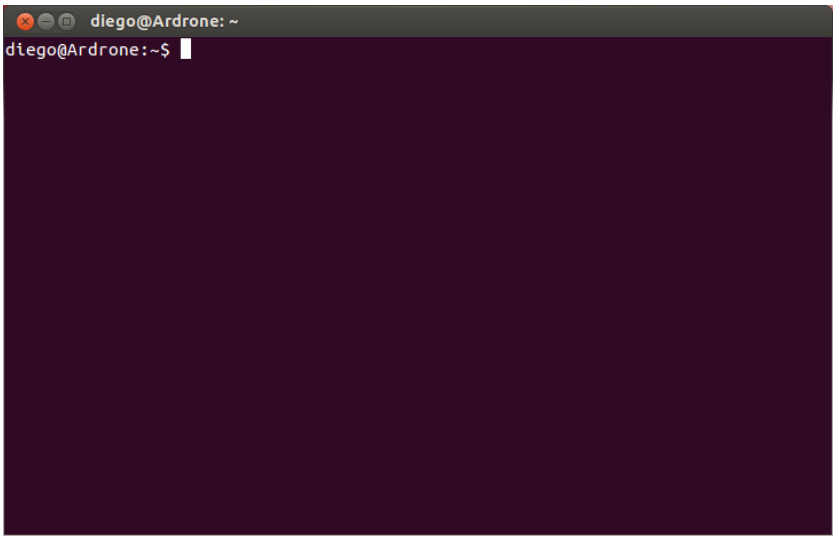


Figura.1.21 Captura de pantalla del Terminal de Ubuntu

1.4.2 ROS (ROBOTIC OPERATING SYSTEM)

Figura.1.22 Robotic Operating System [16]

1.4.2.1 ¿Qué es ROS?

ROS es un sistema operativo de código abierto para robots. Provee los servicios que se esperaría de un sistema operativo, incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de la funcionalidad de uso común, paso de mensajes entre procesos y gestión de paquetes [11]. Proporciona también la funcionalidad de sistema operativo en un clúster informático heterogéneo. Provee además de herramientas y bibliotecas para la obtención, construcción, escritura y ejecución de código en varios equipos. ROS es similar en algunos aspectos a los “frameworks (marcos) de robots”, tales como Jugador, YARP, Orocos, CARMEN, Orca, MOOS y Microsoft Robotics Studio. [16]

La estructura de ejecución de ROS es una red punto a punto de los procesos que están débilmente acoplados utilizando la infraestructura de comunicación de ROS.

ROS implementa varios estilos diferentes de comunicación, incluida la comunicación sincrónica de estilo RPC a través de los servicios, transmisión asincrónica de datos a través de los temas, y el almacenamiento de datos en un Servidor de Parámetros. [16]

ROS no es un framework en tiempo real, aunque es posible integrar ROS con código de tiempo real; también tiene una perfecta integración con el toolkit en tiempo real Orocos. Actualmente sólo funciona en plataformas basadas en Unix (Ubuntu Linux considerado sistema de *soporte*) [16]. El software de ROS se prueba principalmente en sistemas Ubuntu y Mac OS, aunque la comunidad ROS ha contribuido con soporte para Fedora, Gentoo, Arch Linux y otras plataformas Linux (consideradas *experimentales*) [17]. Aunque un puerto de Microsoft Windows para ROS es posible, aún no se ha explorado a fondo.

El sistema ROS básico, junto con útiles herramientas y librerías se lanza regularmente como una *distribución de ROS,* similar a una distribución de Linux y que proporciona un conjunto de software compatible para que otros lo utilicen y aprovechen.

ROS fue desarrollado originalmente en 2007 bajo el nombre de *switchyard* por el laboratorio de Inteligencia Artificial de Stanford en apoyo al proyecto aI Robot Stair de Stanford. Se basa en una arquitectura gráfica donde el procesamiento se realiza en nodos que pueden recibir, enviar y multiplexar información del sensor, control, estado, planificación, actuador y otros mensajes. [17]

ROS tiene dos “lados” básicos: el lado del sistema operativo *ROS* como se describe anteriormente y *ROS-PKG*, una suite de paquetes contribuidos por el usuario (organizados en *metapaquetes*) que implementan funciones como localización simultánea y mapping (cartografía), planificación, percepción, simulación, etc. [18]. ROS es liberado bajo los términos de la *licencia BSD* y es un software de código abierto. Es gratis para su uso comercial y de investigación. Los paquetes de *ROS-PKG* se licencian bajo una variedad de licencias de código abierto. [18][11]

1.4.2.2 Objetivos

Una de las principales dudas que se plantean los usuarios es: ¿Qué tiene ROS que sea diferente de otra plataforma de software robótico? Es una pregunta difícil de responder, puesto que el objetivo de ROS no es ser un framework con más características; sino apoyar la reutilización de código en la investigación y el desarrollo de la robótica.

ROS es un marco de aplicación distribuido de procesos (también conocido como **NODOS**) que permite a los ejecutables ser diseñados individualmente y tener un acoplamiento flexible en el tiempo de ejecución. Estos procesos se pueden agrupar en *paquetes* y *pilas de paquetes*, que pueden ser fácilmente compartidos y distribuidos. Este diseño, permite decisiones independientes sobre desarrollo e implementación, pero todos pueden ser llevados juntos con herramientas de infraestructura ROS. [16][17]

Para apoyar esta meta fundamental de intercambio y colaboración, hay muchos otros objetivos del framework ROS:

- *Delgado*: ROS está diseñado para ser lo más delgado (fino) posible –no se envuelve el main() – para que el código escrito por ROS puede utilizarse con otros marcos de software robot. ROS ya se ha integrado con OpenRAVE, Orocos y Player.
- *Las bibliotecas independientes de ROS*: el modelo de desarrollo preferido es escribir bibliotecas ROS-agnostic con interfaces funcionales limpias.
- *Independencia de Lenguaje*: el framework ROS es fácil de implementar en cualquier lenguaje de programación moderno. Ya se ha implementado en *Python, C*++ y *Lisp*, y se tienen bibliotecas experimentales en *Java y Lua*.
- *Prueba fácil*: ROS ha construido en un marco de prueba de unidad / integración llamado ROSTEST que hace que sea fácil de llevar y derribar accesorios de la prueba.
- *Escala*: ROS es apropiado para sistemas con tiempos de ejecución grandes y para grandes procesos de desarrollo. [16]

1.4.2.3 Aplicaciones

ROS incluye las siguientes áreas:

- Nodo de coordinación maestro.
- Edición o suscripción a flujos de datos: imágenes estéreo, láser, control, actuador, etc.
- Información de multiplexación.
- Nodo de creación y destrucción.
- Los nodos están perfectamente distribuidos, permitiendo del control distribuido multi – core, multi – procesador, GPUs y clusters.
- Registro.
- Servidor de Parámetros.
- Sistemas de Prueba. [18]

Las áreas de Aplicación del ROS Paquete incluyen:

- Percepción.
- Identificación de objetos.
- Segmentación y Reconocimiento.
- Reconocimiento facial.
- Reconocimiento de gestos.
- Seguimiento de movimiento.
- Egomotion
- Comprensión de movimiento.
- Estructura de Movimiento (SFM)
- Visión Estéreo (stereo vision): percepción de profundidad mediante dos cámaras.
- Movimiento.
- Robótica móvil.
- Control.
- Planificación.
- Grasping (capacidad para tomar o agarrar un objeto). [18]

1.4.2.4 Funcionamiento del Sistema

ROS tiene tres niveles de conceptos: el nivel del sistema de archivos, el nivel de computación gráfica, y el nivel comunitario. Estos niveles y conceptos se resumen a continuación:

1.4.2.4.1 Sistema de Archivos:

Son recursos que se encuentran en el propio programa:

- *Paquetes*: Los paquetes son la unidad principal para organizar software en ROS. Un paquete puede contener procesos ejecutables (nodos), una biblioteca dependiente, conjuntos de datos, archivos de configuración, o cualquier otra cosa que sea útil para una organización conjunta.
- *Manifiestos:* proporcionan meta – datos sobre un paquete, incluyendo su información de licencia y dependencias, así como información específica del compilador.
- *Pilas (stacks):* Es una colección de paquetes que tienen una misma función.
- *Manifiestos de pilas:* proporcionan datos sobre una pila, incluyendo su información de licencia y sus dependencias en otras pilas.
- *Mensajes*: definen las estructuras de datos para los mensajes enviados en ROS.
- *Servicios*: definen la solicitud y estructuras de datos de respuesta de los servicios requeridos por ROS. [16][17]

1.4.2.4.2 Computación a Nivel Gráfico

La computación gráfica es la red ROS que se encarga de procesar todos los datos. Los conceptos básicos son nodos, maestro, mensajes, servidor de parámetros, servicios, bolsas y temas, los cuales proporcionan los datos de diferentes maneras:

- *Nodos:* los nodos son procesos que llevan a cabo cálculos. ROS está diseñado para ser modular en una escala básica; un sistema de control de robots comprende por lo general muchos nodos. Por ejemplo, un nodo controla un telémetro de láser, controla los motores de las ruedas, realiza la tarea de localización, un nodo realiza la planificación de la trayectoria,

proporciona una vista gráfica del sistema, y así sucesivamente. Un nodo de ROS se escribe con el uso de una biblioteca de cliente, como roscpp o Rospy.

- *Maestro:* el ROS Master proporciona el registro de nombres y la búsqueda para el resto de la Computación Gráfica. Sin el Maestro, los nodos no serían capaces de encontrar mensajes de cambio entre sí, o invocar servicios.
- *Servidor de Parámetros:* permite que los datos sean almacenados por clave en una ubicación central. Actualmente forma parte del Maestro.
- *Mensajes:* los nodos se comunican entre sí mediante el paso de mensajes. Un mensaje es simplemente una estructura de datos, que comprende los campos de diferentes tipos. Se admiten los tipos primitivos estándar (entero, coma flotante, booleanos, etc.), así como los arreglos de estos tipos primitivos. Los mensajes pueden incluir estructuras y arrays (arreglos) anidados arbitrariamente (similar a estructuras en C).
- *Temas:* los mensajes se enrutan a través de un sistema de transporte con la publicación/suscripción semántica. Un nodo envía un mensaje mediante la "publicación" a un determinado tema. El tema es un nombre que se utiliza para identificar el contenido del mensaje. Un nodo que esté interesado en un determinado tipo de datos será suscrito al tema correspondiente. Puede haber múltiples editores y suscriptores concurrentes para un solo tema, y un único nodo puede publicar y/o suscribirse a múltiples temas. En general, los editores y suscriptores no son conscientes de la existencia de los demás. Lógicamente, se puede pensar en un tema como un bus de mensajes inflexible; cada bus tiene un nombre y se puede conectar a él para enviar o recibir mensajes mientras sean del tipo correcto.
- *Servicios:* el modelo de publicación/suscripción, es un paradigma de comunicación muy flexible. El proceso petición/respuesta se realiza a través de *servicios,* que se definen por un par de estructuras de mensajes: uno para la solicitud y uno para la respuesta.
- *Bags:* Bags son un formato para guardar y reproducir datos de mensajes de ROS. Los Bags son un mecanismo importante para el almacenamiento de datos, como datos de sensores que pueden ser difíciles de recoger, pero son necesarios para desarrollar y probar algoritmos. [16]

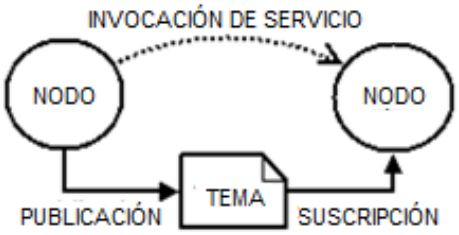


Figura.1.23 Conceptos Básicos de ROS [16]

El ROS Master almacena los temas y servicios de información de registros de los nodos de ROS. Los nodos se comunican con el Maestro para comunicar la información de registro. A medida que estos nodos se comunican con el Maestro, pueden recibir información sobre otros nodos inscritos y hacer las conexiones según corresponda. El Maestro también hará devoluciones de llamadas a estos nodos cuando se ejecuten cambios en la información de registro, lo que permite a los nodos crear dinámicamente las conexiones con nuevos nodos.

Los nodos se conectan a otros nodos indirectamente; el Maestro solo proporciona información de búsqueda, al igual que un servidor DNS. Los nodos que se suscriben a un tema solicitarán las conexiones desde los nodos que publican ese tema, y se establecerá que la conexión se haga a través de un acuerdo sobre un protocolo de conexión. El protocolo más común que se utiliza en ROS se llama *TCPROS,* que utiliza sockets TCP/IP estándar. [16]
Los nombres tienen un papel muy importante en ROS: nodos, temas, servicios y todos los parámetros tienen nombre.

1.4.2.4.3 ROS: Nivel Comunidad

Los conceptos comunitarios de ROS son recursos que permiten a las comunidades separadas intercambiar software y conocimiento. Estos recursos incluyen:

- *Distribuciones:* ROS distribuciones son colecciones de pilas de paquetes versionadas que se pueden instalar. Las distribuciones juegan un papel similar al de las distribuciones de Linux: hacen más fácil la instalación de una colección de software.

- *Repositorios:* ROS se basa en una red federada de repositorios de código, en los que diferentes instituciones pueden desarrollar y lanzar sus propios componentes de software para robots.
- *ROS Wiki:* la comunidad Wiki es el principal foro para la documentación de información sobre ROS. Cualquier persona puede inscribirse para una cuenta y contribuir con su propia documentación, proporcionar correcciones o actualizaciones, escribir tutoriales y más.
- *Sistema de Tickets Bug*.
- *Listas de correo:* la lista de distribución de ROS es el principal canal de comunicación sobre los nuevos cambios de ROS, así como un foro para hacer preguntas sobre el software. [17]

1.4.2.5 Otras herramientas básicas y conceptos

- *Roscore*: es al mismo tiempo el master, el rosout y el servidor de parámetros. En ROS, roscore tiene que ejecutarse en un terminal de Linux.

- *Roscpp*: es una librería de ROS para C++. Permite al programador comunicarse y relacionarse con todas las herramientas de ROS.

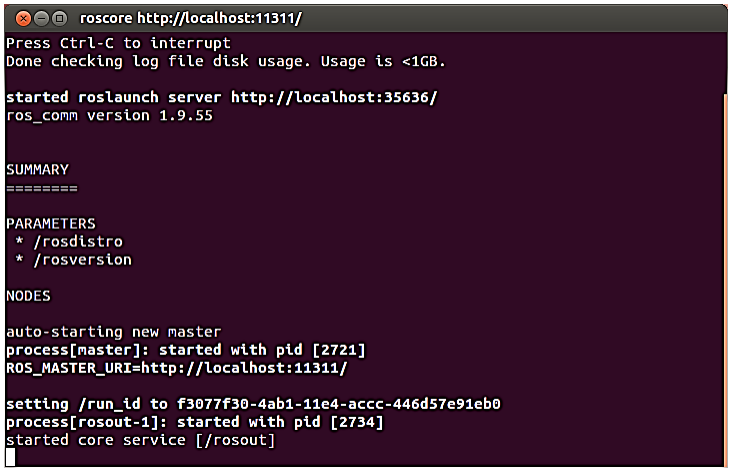


Figura.1.24 Roscore corriendo en un terminal

- *Parámetro*: los parámetros pueden ser ajustados o se puede acceder a ellos desde un nodo o desde un terminal por el usuario. Esto es una forma útil para transferir información puntual. Un ejemplo podría ser la ganancia de un PID

implementado en un nodo, si este valor se toma desde un parámetro, el usuario puede cambiar dicho valor durante el funcionamiento del programa y actualizarlo.

- ***Rxgraph***: es una herramienta que crea un gráfico para mostrar todos los nodos que se están ejecutando y los temas que los une. Una flecha, que sale desde el "publicador" hacia el "suscriptor", representa el tema. Cada nodo puede ser al mismo tiempo publicador (editor) y suscriptor a diferentes nodos. [11][16]

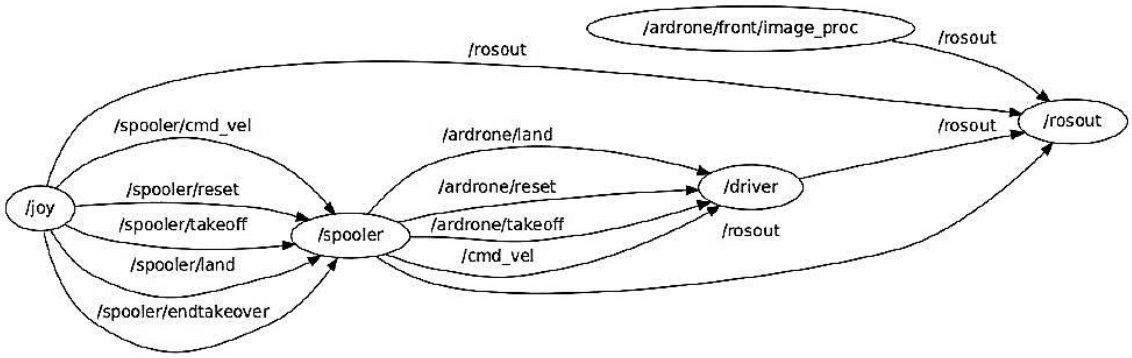


Figura.1.25 Rxgraph mostrando los nodos y temas de ROS [11]

- ***Image_proc package***: este paquete remueve la distorsión de la cámara de la secuencia de imágenes sin procesar (en bruto) o convierte esta secuencia de imágenes en una imagen monocromática, o los dos.

  Dado que la cámara frontal tiene un lente amplio de ángulo 92º (ojo de pez diagonal), la imagen aparecerá en primera instancia redondeada, y para ciertas aplicaciones es necesario rectificar esta distorsión.

  En la parte izquierda de la figura hay una captura de video sin rectificar, donde se puede ver que la tabla no es plana, mientras que en el cuadro de la derecha, donde la misma captura se rectifica con el paquete image_proc, la "tabla" se muestra plana. [11]

Figura.1.26 Image_proc calibración de la cámara

1.4.2.6 ROS build tool: Cmake

La herramienta que ROS utiliza para compilar un paquete es CMake. CMake es un sistema de compilación de código abierto. Controla la compilación de software usando archivos simples. Estos archivos son los *CmakeList.txt* y los *Manifest.xml* y todos los paquetes deben contenerlos para ser compilados.

El archivo CmakeList.txt es en donde se tiene que especificar cuál archivo .cpp tiene que ser compilado, y si son o no mensajes o servidores los que se construyen en el paquete. Cada archivo para ser compilado tiene que estar en la carpeta correcta, la carpeta para los archivos .cpp para crear un nodo se llama **src**, la carpeta para los mensajes es **msg** y para los servicios es **srv**. [11].
En el archivo Manifest.xml tienen que escribirse las dependencias del paquete. Esto es, los otros paquetes con los que éste se comunica o de los que toma información.

1.4.3 PAQUETE AR.DRONE AUTONOMY

El driver para el cuadricóptero Parrot AR.Drone está disponible en la página web de ROS. Este driver es un paquete de ROS llamado *ardrone_autonomy* que contiene el nodo *ardrone_driver*, los mensajes personalizados creados por este nodo y los servidores para los servicios ofrecidos por el driver [11]. El driver ardrone_autonomy está basado en la versión oficial 2.0 del AR.Drone SDK y funciona correctamente tanto para el AR.Drone 1.0 como para el 2.0. Este paquete ha sido desarrollado por el Autonomy Lab of Simon Fraser University. [19]

1.4.3.1 Estructura del Driver

En la figura 1.30, se pueden observar los diferentes temas que interactúan con este driver. Aquellos cuyas flechas apuntan hacia ardrone_autonomy son los que están suscritos al nodo. Esto significa que el nodo está a la escucha de los mensajes que se envían a través de estos temas. Para comandar el AR.Drone un nodo tiene que publicar en estos temas.

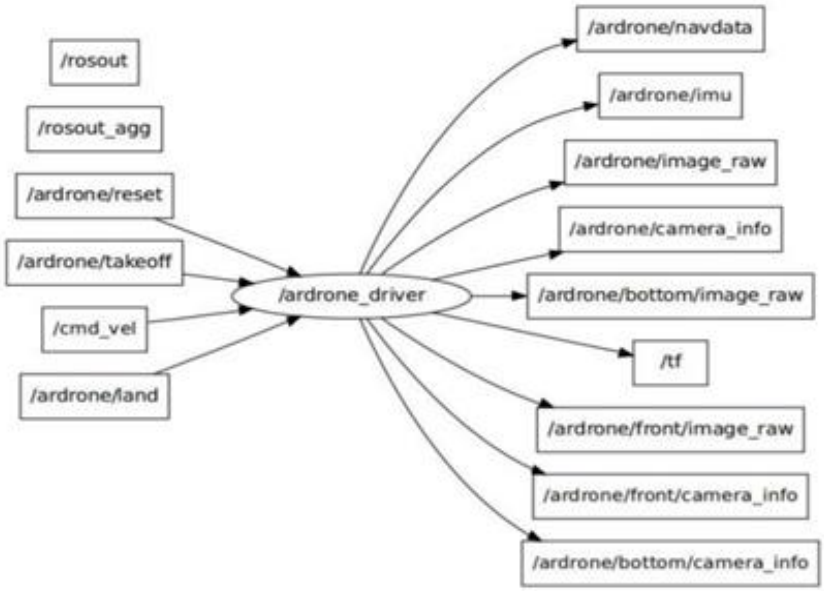


Figura.1.27 Temas en los que el /ardrone_driver publica o está suscrito [11]

Para comandar el AR.Drone y que despegue, aterrice o reinicialice un mensaje en blanco, tiene que enviarse al tema correspondiente. El tipo de estos mensajes tiene que ser std_msgs::Empty. Una vez que el cuadricóptero ha hecho el despegue y está flotando se puede controlar mediante el envío de comandos de velocidad al tema cmd_vel. El tipo de estos mensajes debe ser geometry_msgs::Twist por ejemplo [11]. El nodo publicará información sobre los temas señalados por las flechas y puede enviar mensajes a través de cualquiera de estos temas; y otros nodos que están suscritos a ese tema podrán escucharlos. Si un nodo quiere controlar o comandar el AR.Drone debe escuchar algunos de los temas que el driver publica con el fin de conocer qué está haciendo el cuadricóptero y en qué estado se encuentra.

El tema ***NAVDATA*** contiene información general sobre la navegación del cuadricóptero recibida desde los diferentes sensores [11]. Los más importantes que se utilizan en este proyecto son: porcentaje de batería, estado del AR.Drone, orientación, velocidad y aceleración lineal en los tres ejes. El estado del AR.Drone puede ser: 0: Desconocido, 1: Iniciar, 2: Aterrizado, 3 y 7: Volar, 4: Suspendido en el aire, 5: Test/Prueba, 6: Despegar y 8: Aterrizar.

Los temas */ardrone/bottom/image_raw* y */ardrone/front/image_raw* contienen imágenes de las cámaras ventral y frontal respectivamente. El AR.Drone solamente permite enviar imágenes de una cámara a la vez por lo que mientras uno de estos temas esté transmitiendo el otro no va a enviar nada. El tema /ardrone/image_raw contiene las imágenes que la cámara está transmitiendo en ese momento. Los mensajes en este tema son del tipo sensor_msgs/Image. Cada cámara tiene un tema donde la información de ésta, así como las dimensiones de la imagen es emitida pero como solo una cámara transmite la vez, para escoger cuál lo hará, existen dos servicios:

- /ardrone/togglecam no requiere ningún parámetro o cambio de la alimentación actual de la cámara a otra.
- /ardrone/setcamchannel con parámetro 0 de cambios para el frente y con parámetro 1 para la cámara ventral.

La aceleración lineal, velocidad angular y orientación (Navdata) son publicadas en el tema *ardrone/imu* como se vio anteriormente. El tipo de mensaje es sensor_msgs/Imu y las unidades son métricas. [11][15]

La frecuencia de actualización del Navdata se puede elegir mediante los parámetros *navdata_demo* y *realtime_navdata*. El primero determina la frecuencia de transmisión de datos del AR.Drone: 15Hz cuando el parámetro se establece en 1 o 200Hz cuando se establece en 0. El segundo parámetro afecta a la frecuencia de actualización del controlador, si se establece en **true**, el controlador publicará la información recibida al instante; de lo contrario el driver almacenará en la memoria caché los datos recibidos más recientes, configurados por otro

parámetro llamado **looprate**. La configuración predeterminada realtime_navdata se establece en **false** y looprate se establece en 50. [11]

1.4.3.2 Mensajes Personalizados

Para el tema ardrone/navdata los desarrolladores del código crearon un tipo específico de mensaje llamado ardrone/Navdata, sus campos se explican a continuación:

- header: mensaje de encabezado de ROS.
- BatteryPercent: la carga restante de la batería del AR.Drone (%).
- state: el estado actual del AR.Drone: 0:desconocido, 1: inited, 2: aterrizado, 3 y 7: volando, 4: flotando en el aire, 5: prueba, 6: despegar y 8: aterrizar
- rotX: izquierda/derecha inclinación en grados (rotación sobre el eje x)
- rotY: adelante/atrás inclinación en grados (rotación sobre el eje y)
- rotZ: orientación en grados (rotación sobre el eje z)
- magX, magY, magZ: lecturas del magnetómetro
- pressure: presión percibida por el barómetro del AR.Drone
- temp: temperatura percibida por el sensor del AR.Drone
- wind_speed: velocidad estimada del viento
- wind_angle: ángulo estimado del viento
- wind_comp_angle: compensación estimada del ángulo del viento
- altd: altura estimada (mm)
- vx, vy, vz: velocidad lineal (mm/s)
- ax, ay, az: aceleración lineal (g)
- tm: marca de tiempo (fecha y hora) de los datos devueltos por el AR.Drone como número en micro – segundos transcurridos desde el arranque del AR.Drone. [19]

Todos los otros mensajes utilizan un tipo estándar de mensajes.

1.4.3.3 Servicios de Comunicaciones entre el AR.Drone y un cliente

- El control del AR.Drone se realiza a través de 3 servicios de comunicación principales. Toda la información de datos como nivel de batería, estado,

rotación de motores, velocidad o datos de la IMU, son enviados por el AR.Drone al cliente a través del puerto UDP 5554 [15]. Esta información llamada ***NAVDATA,*** como se detalla en párrafos anteriores, es enviada por el cuadricóptero aproximadamente 30 veces por segundo. [11]

- El control y la configuración del UAV se realiza mediante el envío de comandos AT (generalmente 30 veces por segundo) en el puerto UDP 5556. La latencia de transmisión de los comandos de control es fundamental para la experiencia del usuario. Los comandos AT componen el conjunto de comandos Hayes, un lenguaje de comandos específico, desarrollado por Hayes Smartmodem en 1981.

- La secuencia de vídeo es enviado por el AR.Drone al dispositivo remoto de control por el puerto 5555. Este es codificado por el algoritmo P264. [15]

- Un cuarto canal de comunicación, llamado puerto de control, se puede establecer en el puerto TCP 5559 para transferir datos críticos, por oposición a los otros datos que pueden perderse sin ningún efecto peligroso. Se utiliza para recuperar los datos de configuración, y reconocer la información importante.[11]

1.4.4 OPENCV

Figura.1.28 OpenCV [20]

OpenCV (Open Source Computer Vision Library) es una librería de visión por computador de código abierto [22], originalmente desarrollada por INTEL. La

librería está escrita en C y C++ y se corre bajo sistema Linux, Windows y Mac OS. Hay un desarrollo activo e interfaces para Python, Ruby, Matlab y otros lenguajes.

OpenCV se ha utilizado en infinidad de aplicaciones; desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos, debido a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas. [20]

OpenCV fue diseñado para la eficiencia computacional y con un fuerte enfoque en aplicaciones en tiempo real. OpenCV está escrito en lenguaje C optimizado y puede tomar ventaja de los procesadores multinúcleo; además utiliza automáticamente la librería IPP apropiada, la cual consiste en rutina de bajo nivel en diferentes áreas algorítmicas, en un determinado tiempo de ejecución.

Uno de los objetivos de OpenCV es proporcionar una infraestructura de visón por computador fácil de utilizar, que ayude a construir aplicaciones de visión sofisticadas de forma rápida [20]. Esta mega biblioteca contiene más de 500 funciones que abarcan muchas áreas en visión, incluyendo inspección de productos en fábricas, imágenes médicas, seguridad, interfaz de usuario, calibración de cámaras, visón estéreo y robótica. Debido a que la visión artificial y el aprendizaje automático a menudo van de la mano, OpenCV también contiene una librería *Machine Learning* de uso general completa (MLL). Esta sub librería se centra en el reconocimiento de patrones estadísticos y clustering. La MLL es de gran utilidad para tareas de visión que son el núcleo de OpenCV, pero es lo bastante general para usarse en cualquier problema de aprendizaje automático. [22]

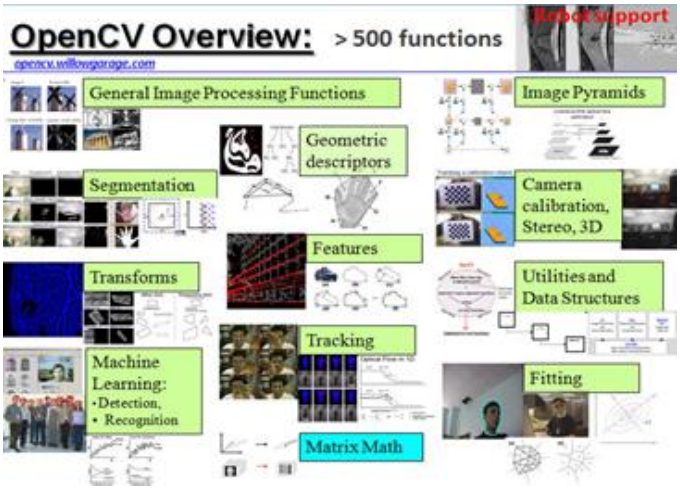


Figura.1.29 Vista general de OpenCV [21]

OpenCV tiene una estructura modular, lo que significa que el paquete incluye varias bibliotecas compartidas o estáticas. Los siguientes módulos son los que están disponibles:

- *core*: módulo compacto que define las estructuras de datos básicas, incluyendo la matriz multi – dimensional **Mat** y las funciones básicas utilizadas por todos los demás módulos.
- *Imgproc*: módulo de procesamiento de imágenes que incluye filtrado de imágenes lineal y no lineal, tranformaciones de imágenes geométricas (redimensionar, afín, punto de vista de las deformaciones, mapeo basado en tablas genéricas), conversión de espacio de color, histogramas, etc.
- *video*: módulo de análisis de video que incluye estimación de movimiento, sustracción de fondo y algoritmos de seguimiento de objetos.
- *Calib3d*: algoritmos básicos de geometría de visión múltiple, calibración de la cámara individual y estéreo, estimación de la posición del objeto, algoritmos de correspondencia estéreo y reconstrucción 3D de elementos.
- *Features2d*: detectores de características sobresalientes, descriptores y descriptor de coincidencias.
- *Objdetect*: detección de objetos e instancias de clases predefinidas (por ejemplo: rostros, ojos, personas, autos, bocas, etc.).
- *highgui*: una interfaz fácil de usar para la captura de video, imagen y codecs de video, tales como las capacidades simples de la interfaz de usuario.

- *Gpu*: algoritmos acelerados por GPU de diferentes módulos OpenCV. [22]

Esta librería proporciona un alto nivel funciones para el procesamiento de imágenes y permite a los programadores crear aplicaciones poderosas en el dominio de la visión digital. OpenCV ofrece muchos tipos de datos de alto-nivel como juegos, árboles, gráficos, matrices, etc.

1.4.4.1 Aplicaciones Conocidas

- OpenCV ha sido usada en el sistema de visión del vehículo no tripulado Stanley de la Universidad de Stanford.
- OpenCV se usa en sistemas de vigilancia de vídeo.
- OpenCV es la clave en el programa Swistrack, una herramienta de seguimiento distribuida. [20]

OpenCV permite además:

- Operaciones básicas,
- Procesamiento de imágenes y análisis,
- Análisis estructural,
- Análisis de movimiento,
- Reconocimiento del modelo,
- Reconstrucción 3d y calibración de cámaras,
- Interfaz gráfica y adquisición. [21]

1.4.4.2 Rasgos de OpenCV

OpenCV implementa una gran variedad de herramientas para la interpretación de imágenes. Es compatible con Intel Image Processing Library (IPL) que provee algunas operaciones digitales en imágenes. OpenCV proporciona ciertas funciones que podrían considerarse primitivas pero que son importantes en el procesamiento de imágenes: binarización, filtrado, estadísticas de la imagen, pirámides, etc. Esta es una librería que implementa algoritmos para técnicas de calibración (Calibración de la Cámara), detección de rasgos, rastreo (Flujo Óptico), análisis de la forma (Geometría, Contorno que Procesa), análisis del movimiento

(Plantillas del Movimiento, Estimadores), reconstrucción 3D (Transformación de vistas), segmentación de objetos y reconocimiento (Histograma).

El rasgo esencial de la librería es junto con funcionalidad y calidad su desempeño. Los algoritmos están basados en estructuras de datos muy flexibles. OpenCV usa la estructura **Iplimage** para crear y manejar imágenes así como la estructura **Mat** que opera basándose en matrices.

En cuanto a análisis de movimiento y seguimiento de objetos, OpenCV ofrece una funcionalidad interesante; incorpora funciones básicas para modelar el fondo y sustraerlo posteriormente, genera imágenes de movimiento MHI (Motion History Images) para determinar dónde hubo movimiento y en qué dirección, entre otros. También es importante conocer que OpenCV viene con una interface gráfica llamada **highGUI**, necesaria para visualizar imágenes. [21]

1.4.4.3 Inconvenientes de OpenCV

Dadas las grandes posibilidades que ofrece OpenCV para el tratamiento de imágenes, calibración de cámaras, y otras muchas aplicaciones como por ejemplo: simular una prótesis ocular basada en un implante cortical y estudiar el funcionamiento de las retinas artificiales, etc.; quizá de los pocos inconvenientes que se pueden encontrar en este software sea en el caso del seguimiento de objetos, en el cual, el principal problema es que no ofrece un producto completo, tan sólo algunas piezas que sirven como base para montar sobre ellas un producto final. Otro de los inconvenientes que tiene es la necesidad de utilizar la librería IPL para tener acceso a funciones de bajo nivel.

Sin embargo, la presencia de funciones muy interesantes, y las posibilidades ya comentadas que ofrece la librería OpenCV hacen que estos contratiempos no sean realmente significantes [21]. Otros datos adicionales que deben mencionarse son que, los microprocesadores más veloces, la caída en el precio de las cámaras y el ancho de banda para la captura de video diez veces mayor que ofrecen tecnologías como USB2 están haciendo posible que los algoritmos de visión para computadoras en tiempo real puedan funcionar en PCs

convencionales [20][21]. Los desarrolladores están utilizando el código de OpenCV en aplicaciones que van desde juguetes hasta la fabricación industrial.

1.4.5 SEGUIMIENTO DE OBJETOS

El seguimiento de objetos es el proceso de estimar en el tiempo la ubicación de uno o más objetos móviles mediante el uso de una cámara. La rápida mejora en cuanto a calidad y resolución de sensores de imagen, junto con el incremento en cuanto a potencia de cálculo, ha favorecido la creación de nuevos algoritmos y aplicaciones mediante el seguimiento de objetos.

El seguimiento de objetos puede ser un proceso lento debido a la gran cantidad de datos que contiene un video. Además, la posible necesidad de utilizar técnicas de reconocimiento de objetos para realizar el seguimiento incrementa su complejidad. [24]

1.4.5.1 Diseño de un seguidor de objetos

Las cámaras de video capturan información sobre los objetos de interés en forma de conjunto de píxeles. Al modelar la relación entre el aspecto del objeto de interés y el valor de los píxeles correspondientes, un seguidor de objetos valora la ubicación de este objeto en el tiempo. Los principales retos a tener en cuenta se relacionan con la similitud de aspecto entre el objeto de interés y el resto de objetos en la escena, así como la variación de aspecto del propio objeto [23][24]. El aspecto del resto de objetos como del fondo pueden confundirse con el objeto de interés, en ese caso, las características extraídas de esas áreas no deseadas pueden ser difíciles de diferenciar de las que se espera que el objeto de interés genere. Este fenómeno se conoce con el nombre de **clutter**. [24]

1.4.5.2 Dificultades del seguimiento

Además del clutter, los cambios de aspecto del objeto en el plano de la imagen dificultan el seguimiento causado por uno o más de los siguientes factores:

- *Cambios de posición.* El objeto móvil de interés varía su aspecto cuando se proyecta sobre el plano de la imagen, por ejemplo, al girar.

- *Iluminación ambiente.* La dirección, intensidad y color de la luz de ambiente influyen en el aspecto del objeto de interés. Asimismo, los cambios en la iluminación son un reto en las escenas al aire libre.
- *Ruido.* El proceso de adquisición de imágenes introduce en la señal de la imagen un cierto grado de ruido que depende de la calidad del sensor.
- *Oclusiones.* Puede ser que un objeto de interés no se observe bien cuando sea parcial o totalmente tapado por otros objetos en la escena. Las oclusiones son generalmente debidas a:
  - Un objeto de interés que se mueve detrás de un objeto estático.
  - Objetos que se mueven en la escena de manera que entorpecen la visión de un objeto de interés. [23]

1.4.5.3 Representación del objeto

Un objeto se puede definir como cualquier cosa que se pueda representar mediante su forma y apariencia y sea de interés. A continuación se describen las representaciones de forma del objeto utilizadas generalmente:

- *Puntos.* El objeto se representa por un punto, es decir, por un centroide o por un conjunto de puntos. Este tipo de representación es adecuada para el seguimiento de objetos que ocupan pequeñas regiones en una imagen.
- *Formas geométricas primitivas.* La forma del objeto se representa con un rectángulo, elipse, etc. El movimiento de estas representaciones es modelada por la translación, afinidad o transformación proyectiva (homografía). Aunque las formas geométricas primitivas son más adecuadas para la representación de objetos rígidos, también se usa para el seguimiento de objetos no rígidos.
- *Silueta del objeto y contorno.* La representación del contorno define el límite de un objeto. La región dentro del contorno se conoce como la silueta del objeto. Esta representación se usa para el seguimiento de formas complejas no rígidas.
- *Modelos articulados de forma.* Los objetos articulados están formados por partes del cuerpo que unidas por articulaciones. La relación entre estas partes se rige por modelos del movimiento cinemático. Para representar un objeto articulado, se puede modelar los componentes con cilindros o elipses.

- *Modelos esqueléticos.* El esqueleto del objeto se puede extraer mediante la transformación del eje medio de la silueta del objeto. La representación del esqueleto se puede utilizar para modelar objetos articulados y rígidos. [24]

También hay varias maneras de representar las características de aspecto de los objetos. Se debe tener en cuenta que las representaciones de forma también se pueden combinar con las de aspecto para llevar a cabo el seguimiento. Algunas de las representaciones de aspecto más comunes son:

- *La densidad de probabilidad del aspecto de los objetos.* Esta densidad de probabilidad (color, textura) se puede calcular a partir de las regiones de la imagen especificada por los modelos de forma (región interior de una elipse o un contorno) y puede ser paramétrica o no paramétrica.
- *Plantillas.* Están formadas con formas geométricas simples o siluetas y una de sus ventajas es que aporta información espacial como de aspecto. [23]
- *Modelos activos de aspecto.* Se generan mediante el modelado simultáneo de la forma del objeto y su aspecto. La forma del objeto se define por un conjunto de puntos, por cada punto se guarda un vector de aspecto en forma de color o textura. Estos modelos requieren una fase de entrenamiento donde forma y aspecto se conocen a partir de un conjunto de muestras.
- *Modelos de aspecto multivista.* Codifican diferentes puntos de vista de un objeto. [23]

1.4.5.4 Selección de características

Seleccionar las características adecuadas tiene un papel fundamental en el seguimiento. En general, la característica visual más deseada es la singularidad porque los objetos se pueden distinguir fácilmente en el espacio de características. Los detalles de las características más comunes son los siguientes:

- *Color***:** En el procesamiento de imágenes se utiliza normalmente el *espacio de color RGB* (rojo, verde y azul) para representar esta característica. Pero, el espacio RGB no es porcentualmente uniforme, y por tanto se han utilizado

variedad de espacios de color en el seguimiento. El color aparente de un objeto se ve influenciado principalmente por dos factores físicos:

- La distribución de energía espectral de la fuente.
- Las propiedades de reflectancia de la superficie del objeto.

- *Márgenes.* Los límites de los objetos suelen generar fuertes cambios en la intensidad de la imagen, la detección de márgenes se utiliza para identificar dichos cambios. Una de sus propiedades es que son menos sensibles a los cambios de iluminación en comparación con las características de color.
- *Flujo óptico.* Es un campo denso de desplazamiento de vectores que define la translación de cada píxel en una región. Se calcula mediante la restricción de brillantez constante y se usa como característica de segmentación basada en movimiento.
- *Textura.* Es una medida de la variación de intensidad de una superficie que cuantifica propiedades como suavidad y regularidad. La textura requiere una etapa de procesamiento para generar los descriptores siendo menos sensibles a los cambios de iluminación. [23]

1.4.6 DETECCIÓN DE LA IMAGEN

Un método común para la detección de objetos es el uso de la información de un solo fotograma. Aunque, algunos otros métodos hacen uso de la información temporal calculada a partir de una secuencia de imágenes; esta información temporal se calcula con la técnica **frame differencing**, que pone de manifiesto las regiones cambiantes en tramos consecutivos. Los métodos más populares para el seguimiento de objetos son:

- Los detectores de puntos.
- La sustracción del fondo.
- La segmentación.

1.4.6.1 Detectores de puntos

Los detectores de puntos se utilizan para encontrar los puntos de interés en imágenes que tienen una textura expresiva en sus respectivas localidades. Los

puntos de interés se usan en el contexto del movimiento y en problemas de seguimiento. Una característica en cuanto a puntos de interés es su invariación en los cambios de iluminación y en el punto de vista de la cámara.

1.4.6.2 Sustracción del fondo

La detección de objetos se consigue con la construcción de una representación de la escena llamada modelo de fondo y después encontrando las desviaciones del modelo para cada fotograma entrante. Cualquier cambio significativo en una región de la imagen del modelo de fondo representa un objeto en movimiento. Los píxeles que constituyen las regiones en proceso de cambio se marcan para su posterior procesamiento, esto se conoce como sustracción de fondo.

1.4.6.3 Segmentación

Figura.1.30 Clasificación de las técnicas de segmentación de un objeto [24]

El objetivo de los algoritmos de segmentación de la imagen es dividir la imagen en regiones perceptualmente similares. Cada algoritmo de segmentación abarca dos problemas, los criterios para una buena partición y el método para conseguir la partición eficiente. Existen diferentes técnicas de segmentación de objetos en movimiento que se pueden separar en dos grandes grupos: ***las basadas en movimientos y las basadas en características espacio – temporales***. [23][24]

1.4.7 TÉCNICAS DE SEGUIMIENTO DE OBJETOS

El objetivo principal de las técnicas de seguimiento de objetos es generar la trayectoria de un objeto a través del tiempo, posicionando éste dentro de la imagen. Una clasificación de técnicas sería: seguimiento de puntos, seguimiento de núcleo (kernel) y seguimiento de siluetas. [23]

Figura.1.31 Esquema de las diversas técnicas utilizadas para realizar el seguimiento de objetos [24]

1.4.7.1 Técnicas de seguimiento de puntos

Los objetos detectados en imágenes consecutivas están representados cada uno por uno o varios puntos y la asociación de éstos está basada en el estado del objeto en la imagen anterior, que puede incluir posición y movimiento. Esta técnica puede presentar problemas en escenarios donde el objeto presenta oclusiones. [23]

1.4.7.2 Técnicas de seguimiento del núcleo (kernel)

Las técnicas de seguimiento del núcleo realizan un cálculo del movimiento del objeto, representado por una región inicial, de una imagen a la siguiente. El movimiento del objeto se expresa en forma de movimiento paramétrico o mediante el campo de flujo calculado en los siguientes fotogramas. Podemos distinguir dos categorías:

- Seguimiento con plantillas y modelos de apariencia basados en densidad de probabilidad.
- Seguimiento basado en modelos multivista.

1.4.7.3 Técnicas de seguimiento de siluetas

Estas técnicas se realizan mediante la valoración de la región del objeto en cada imagen utilizando la información que contiene. Esta información puede ser en forma de densidad de aspecto o de modelos de forma que son generalmente presentados con mapas de márgenes. [23]

1.4.8 APLICACIONES DE SEGUIMIENTO DE OBJETOS

El seguimiento de objetos es la base de aplicaciones que van desde la producción de vídeo hasta la vigilancia remota, y desde la robótica hasta los juegos interactivos.

- *Medios de comunicación y realidad aumentada*. El seguimiento de objetos es importante en la captura de movimiento en el cine y la televisión.
- *Aplicaciones médicas e investigación biológica*. El seguimiento de objetos ha sido cada vez más utilizado por sistemas médicos para ayuda en el diagnóstico y acelerar la tarea del cirujano.
- *Vigilancia e inteligencia de negocios*. El seguimiento de objetos se utiliza en la vigilancia automática para seguridad, vida asistida y aplicaciones de inteligencia de negocio.
- *Tele-colaboración y juegos interactivos*. Las cámaras web estándar incluyen el software de seguimiento que localiza y sigue la cara de un usuario para videoconferencias desde el escritorio. El seguimiento de objetos también está cambiando la manera de enviar el control a las máquinas.
- *Instalaciones de arte y espectáculos*. El seguimiento de objetos se usa en instalaciones de arte y en actuaciones donde la interacción es posible gracias al uso de cámaras de vídeo y a menudo por los sistemas de proyección. [23][24]

CAPÍTULO 2

DESARROLLO DE ALGORITMOS PARA RECONOCIMIENTO DE IMAGEN

2.1 INTRODUCCIÓN

El procesamiento de imagen se usa para encontrar básicamente las características en el cuadro o imagen que será utilizado para reconocer un objeto o varios puntos de interés. Esta información relevante extraída de la imagen (características/rasgos) varía de estructuras simples, como puntos o bordes, a estructuras más complejas como objetos completos. Todas estas características se emplearán como referencia para cualquier tarea de control visual o sistema de control.

En las regiones de la imagen, la “intensidad espacial” también puede ser considerada como una característica útil para el seguimiento. En este contexto, la intensidad de la región puede ser considerada como una característica única que se puede comparar usado correlaciones métricas de patrones de intensidad de la imagen.

La mayor parte de las características utilizadas como referencia son puntos de interés; puntos en una imagen que tienen una posición bien definida, y que se pueden detectar con cierta certeza, y por lo general se encuentran en cualquier tipo de imagen. Algunos de estos puntos son *esquinas* formadas por la intersección de dos *bordes*, y otros son puntos en la imagen que tienen información importante basada en la intensidad de los pixeles.

Las características convenientes para hacer un seguimiento preciso tienen que ser seleccionadas con el fin de asegurar la estabilidad del proceso. Si no se desea caracterizar a un objeto por sus vértices por ejemplo, se pueden considerar sus bordes, existen varios algoritmos de pre procesamiento de imágenes utilizados para encontrar las características de los bordes como el *detector de*

bordes de Canny y otros algoritmos, utilizados en aplicaciones como encontrar las líneas principales y aisladores en una inspección con un VANT. [25]
Para entender cómo se procesan imágenes y se hace un control o seguimiento en base a ellas, en diferentes aplicaciones como la que se presenta en este proyecto, es necesario hacer referencia a un concepto en particular conocido como *VISUAL SERVOING O VISUAL SERVO CONTROL*, que permite entender más claramente lo que se detalla en este capítulo.

2.2 VISUAL SERVOING

El concepto de visual servoing se refiere al uso de los datos de visión por computador para el control del movimiento de un robot, en este caso, el control de vuelo del AR.Drone. Los datos de visión se pueden adquirir desde una cámara que eventualmente podría estar montada en el robot manipulador o en el robot móvil, en cuyo caso el movimiento del robot ocasiona movimiento en la cámara; otra opción es fijar la cámara en el espacio de trabajo de manera que se observe el movimiento del robot desde una configuración estacionaria. [26]

El Visual Servo Control depende de técnicas de procesamiento de imagen, visión por computador y teoría de control. Es una técnica que utiliza la información de realimentación visual extraída de una cámara (o más de una) para controlar el movimiento de un robot; es decir, se basa en encontrar el error entre la característica actual y la referencia, y será ese error el que se realimente.

Las técnicas de control visual se pueden clasificar en tres tipos principales: IBVS (Imagen - control visual base), PBVS (Posición - A base de control visual) y un enfoque híbrido.

IBVS: está basado en el error entre las características deseadas y actuales del frame de la cámara. Las mediciones de las imágenes en 2D son utilizadas para estimar directamente el movimiento deseado del robot.

PBVS: se basa en la estimación de la posición respecto de la cámara, entonces el comando respectivo se emite al controlador del robot. Estos sistemas recuperan

la información en 3D de la escena en la que utiliza el modelo conocido de la cámara (por lo general combinado con un modelo geométrico del objetivo) para estimar la pose (posición y orientación) del objetivo.

Enfoque Híbrido: es un 2-1/2-D (2 dimensiones y media) visual servoing. Esta es una combinación de las dos propuestas anteriores. No necesita ningún modelo geométrico 3D del objeto. Consiste en la combinación de las carcaterísticas visuales obtenidas directamente de la imagen y las características basadas en a posición. [11]

En este proyecto se ha desarrollado el método de visual servoing conocido como *IBVS*.

2.2.1 IMAGE BASED VISUAL SERVO (IBVS)

El Visual Servoing en dos dimensiones (2D) es un enfoque basado en un “modelo de libre control” ya que no necesita el conocimiento de un modelo en el espacio 2D. Así, el 2D visual servoing también se denomina “image-based” visual servoing. [11]

En general, el control visual basado en imágenes es más robusto, no sólo con respecto a la cámara sino también a los errores de calibración del robot. Sin embargo, su convergencia está asegurada sólo en una región (difícil de determinar analíticamente) alrededor de la posición deseada. Excepto en casos muy simples, el análisis de la estabilidad con respecto a los errores de calibración parecen imposibles ya que el sistema es acoplado y no lineal. [27]

La ley de control que rige esta técnica, como se mencionó anteriormente, se basa en el error entre las funciones actuales y deseadas en el plano de la imagen, y no implica ninguna estimación de la pose del objetivo. Las características pueden ser las coordenadas de características visuales, las líneas o los momentos de las regiones. IBVS tiene ciertas dificultades cuando se trabaja con movimientos con rotaciones muy grandes, que son conocidos como *retiro cámara*. [26][11]

A partir de esto, se entiende que primero es importante hacer un pre procesamiento de la imagen que contiene el objetivo que se va a seguir, para después aplicar los algoritmos de interpretación y seguimiento que se detallan en el siguiente capítulo.

A continuación, se consideran las siguientes etapas para conseguir el reconocimiento preciso del objetivo que será sometido a un proceso de seguimiento, con la manipulación de un cuadricóptero:

- Adquisición de las imágenes (captura).
- Pre procesamiento de la imagen; para mejorar la calidad o resaltar detalles que interesan.
    - Suavizar
    - Eliminar ruido.
    - Realzar bordes.
    - Detectar bordes.
- Segmentación: dividir la imagen en segmentos manipulables.
- Extracción de características: tamaño, forma, características geométricas, etc.
- Reconocimiento e interpretación.

2.3 ADQUISICIÓN DE LA IMAGEN Y CALIBRACIÓN DE CÁMARAS

Es sustancial considerar en primer lugar el problema que implica la forma de la lente de la cámara de video del AR.Drone. Las lentes de la cámara normalmente distorsionan la escena haciendo que los puntos lejos del centro parezcan estar aún más lejos. Por lo tanto, las rayas verticales cerca de los bordes de la imagen aparecen ligeramente dobladas; como se menciona en el capítulo 1 de este trabajo. En consecuencia, si se quiere saber la proyección de un pixel, entonces se debe tener en cuenta los componentes de distorsión, hay dos tipos de distorsiones radial y tangencial.

Es necesario entonces calibrar la cámara mediante un proceso de estimación de parámetros intrínsecos y extrínsecos. Parámetros intrínsecos se refieren a las características internas de la cámara, tales como, su distancia focal, inclinación, distorsión, y el centro de la imagen y los parámetros extrínsecos describen su posición y orientación en el mundo. Conocer los parámetros intrínsecos es un paso esencial para la visión por ordenador, ya que permite calcular la estructura de la escena en el espacio euclidiano y eliminar la distorsión de lentes. [11][31]

Todas las herramientas que se requieren para hacer la calibración de la cámara de manera automática están disponibles en la librería OpenCV, específicamente en el paquete *Imagen_proc* de ROS.

Figura.2.1 Distorsión de la imagen por la lente de la cámara [31]

2.3.1 CALIBRACIÓN DE CÁMARAS CON OPENCV

La autocalibración es la base de cualquier sistema autónomo equipado con una cámara, ya que sin ella, las acciones para la navegación deben ser preprogramadas.

A partir de una secuencia de imágenes es posible realizar la calibración. Y a partir de dicha calibración y siempre que no cambien los parámetros intrínsecos de la cámara, es posible navegar por un entorno controlado, es decir, conociendo la distancia a objetos cuyo tamaño real es conocido. Si se cambia por ejemplo el zoom durante el desplazamiento, los parámetros intrínsecos cambiarán, por esta

razón es necesaria la autocalibración, ya que la distancia focal cambia con el zoom.

2.3.1.1 Metodología

Las librerías de OpenCV dan la posibilidad de utilizar ciertas funciones para la calibración de cámaras. Las funciones de calibración de cámaras se emplean para calcular los parámetros intrínsecos e extrínsecos de la cámara con la que se pretende trabajar. Los parámetros de la cámara son una serie de números que describen una configuración particular de la misma.

El procedimiento de calibración se puede resumir en los siguientes pasos:

- Seleccionar un patrón, en este caso un *tablero de ajedrez*.
- Montar el patrón sobre una superficie plana fija.
- Tomar muchas fotografías del patrón en diferentes orientaciones y distancias (Figura.2.2).
- Utilizar ejemplos proporcionados para detectar automáticamente los parámetros de calibración y cálculo.
- Mover el archivo de calibración a un lugar seguro.

Actualmente se recomienda utilizar el patrón de *tablero de ajedrez*, porque parece producir resultados más precisos. Otros tipos de rejillas de calibración se pueden utilizar si se proporciona la ubicación de los puntos de calibración. Es importante especificar el tamaño de cada patrón y el ancho de cada uno de los cuadrados del tablero; el patrón tiene que ser montado en una superficie plana puesto que cualquier deformación disminuirá la precisión de calibración. Antes de empezar a tomar fotografías, hay que asegurarse de que la cámara tenga una longitud focal fija. El procedimiento de calibración asume que todas las fotografías son tomadas con la misma distancia focal.

Hay que tomar una serie de fotos, en diferentes orientaciones, distancias y ubicaciones. El objetivo de calibración debe aparecer a lo largo del borde de la imagen y del centro; el tablero en su totalidad debe ser visible en la imagen y en

algunos casos las fronteras de la imagen también tienen que ser visibles [29]. Se debe mirar los errores de las imágenes individuales y ver si hay valores atípicos o imágenes en las que no se detectó claramente el patrón, porque las fotos con errores inusualmente grandes deben ser retiradas o sustituidas por una mejor imagen. Después solo basta correr el código que ofrece OpenCV para calibración de cámaras, dependiendo del método que se desee usar según las necesidades. [22] [29]

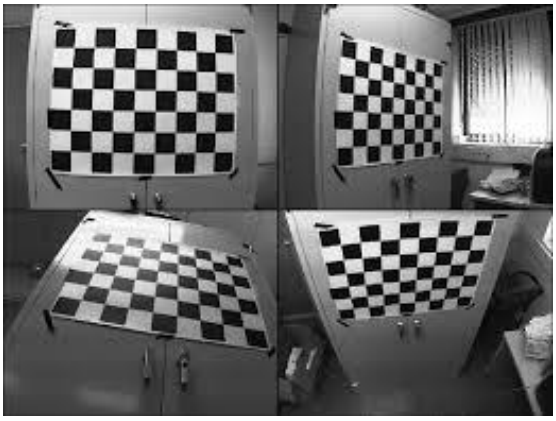


Figura.2.2 Autocalibración cámara tablero de ajedrez [31]

2.3.1.2 Funciones disponibles para calibración

- *cvCalibrateCamera*: Calibra la cámara con una precisión normal. La función no devuelve nada, modifica los parámetros que pasan como argumentos.
- *cvCalibrateCamera 64d:* Calibración de la cámara con doble precisión.
- *cvFindExtrinsicCameraParams:* Encuentra los parámetros extrínsecos para un patrón.
- *cvfindExtrinsicCameraParams 64d:* Igual que el anterior pero con doble precisión para calcular los parámetros extrínsecos.
- *cvRodrigues:* Convierte la matriz de rotación a vector de rotación y viceversa con precisión normal.
- *cvRodrigues 64d:* Calcula exactamente lo mismo que la función anterior pero con doble precisión. [22]

2.3.2 OBTENCIÓN DE LA IMAGEN DESDE UN ARCHIVO O CÁMARA

CV_BRIDGE PACKAGE

Existen ciertos comandos en OpenCV, que permiten leer imágenes y/o videos desde un archivo o bien desde una cámara. Pero en este caso el procedimiento para capturar el video es diferente, ya que en primer lugar la cámara de video del AR.Drone se maneja a través de ROS; así que el video obtenido está en formato de mensajes de ROS y para poder hacer el procesamiento de imagen es necesario cambiarlo a un formato con el que pueda trabajar la librería de OpenCV, con este fin se utiliza la función *cv_bridge* que opera como se explica a continuación:

ROS transporta imágenes en su propio formato de mensaje: sensor_msgs/Image, pero muchas aplicaciones podrían requerir que se usen las imágenes en conjunto con OpenCV [30]. *Cv_bridge es una biblioteca de ROS que proporciona una interfaz entre ROS y OpenCV*, como su nombre lo indica este paquete actúa como un puente entre OpenCV y otros formatos. [11]

Cv_bridge se puede encontrar en el paquete de cv_bridge en el stack de visión_opencv.

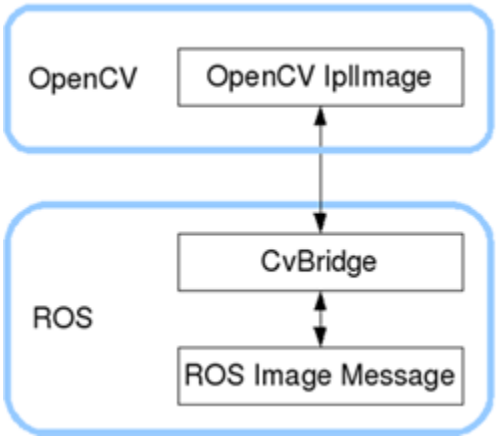


Figura.2.3 Conversión de imagen ROS a imagen OpenCV [11]

A continuación será el comando *imshow()* el que permita desplegar la imagen capturada ya manipulable con OpenCV en una ventana; y como se explica en el capítulo anterior al existir dos tipos de estructuras de datos, en este trabajo se maneja la estructura *Mat*, que almacena las imágenes en una matriz. [28]

2.4 RECONOCIMIENTO DE IMÁGENES POR COLOR

En el siguiente diagrama de flujo se resume la lógica que se sigue para obtener una imagen procesada de tal suerte que sea posible el reconocimiento de objetos y su posterior seguimiento según sus características de color.

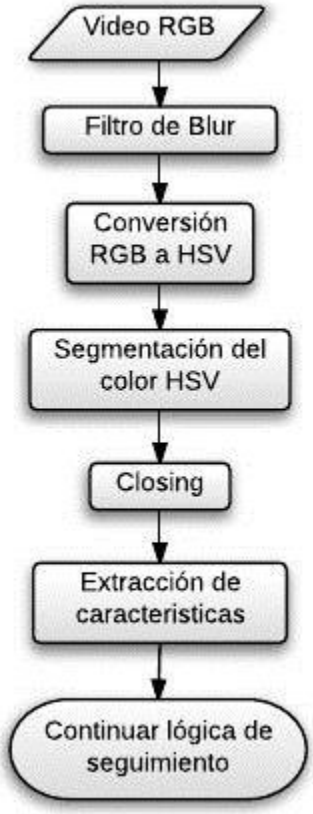


Figura.2.4 Diagrama de Flujo Reconocimiento por Color

En los párrafos siguientes se describe de manera más detallada los algoritmos empleados en el pre procesamiento de imágenes considerando características de color.

2.4.1 PRE PROCESAMIENTO DE IMÁGENES

2.4.1.1 Tratamiento Digital de Imágenes

Aunque los equipos sean cada vez más sofisticados y complejos, su manejo a nivel de usuario se ha adaptado a una ejecución casi automática y de lo más simple. Esto es debido a que un entendimiento y estudio de la tecnología de la imagen digital, solo es accesible mediante un amplio conocimiento de fundamentos básicos.

El tratamiento digital de imágenes estudia el conjunto de *procesos y técnicas que permiten manipular la imagen* de tal manera que con el resultado se pueda descubrir o resaltar cierta información contenida en ella. Como herramienta básica para llevar a cabo todos estos procesos se necesita de un ordenador capaz de maniobrar con imágenes de dos o más dimensiones.

2.4.1.2 Tipos de imagen digital

La clasificación de las imágenes se puede estudiar mediante numerosas formas y varios criterios. Las imágenes digitales pueden ser: imágenes vectoriales e imágenes basadas en mapas de bits. Una imagen vectorial se compone de contornos y rellenos descritos por fórmulas matemáticas. Las imágenes de mapas de bits, en cambio están descritas mediante una concatenación de pequeños cuadritos de igual tamaño, llamados pixeles. En cada pixel se guarda la información de color que presenta la imagen en este punto, dando como resultado una malla que ocupa toda la imagen. Las regiones curvas de un mapa de bits son estructuras dentadas debido a esta topología. [9]

Figura.2.5 Comparación de una región curva descrita mediante una imagen vectorial con una descrita por un mapa de bits [9]

Los mapas de bits se orientan a imágenes con gran diversidad en su gama de color y tonalidad, pero que pierden nitidez al realizar escalados sobre sí mismas. Debido a que la aplicación implementada con el AR.Drone en este proyecto, pretende hacer reconocimiento y seguimiento de objetos mediante el uso de una *WebCam* en donde el escalado digital no es importante, se usarán imágenes en *mapas de bits*.

2.4.1.3 El color de un objeto

Para poder percibir el color de un cuerpo, se necesita luz. El espectro visible se clasifica en 6 colores espectrales como franjas de color, incluyendo en sus límites todas las demás tonalidades.

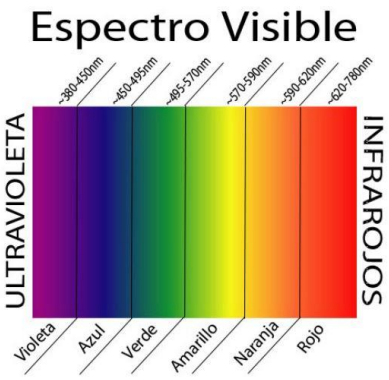


Figura.2.6 Espectro de Color [9]

El color en estos sistemas es cuantitativo, es decir, una combinación de varios números que permiten un análisis objetivo y estandarizado.

2.4.1.4 El espacio RGB y su almacenamiento

La imagen obtenida con ROS por la cámara, sin ningún tipo de procesamiento sino solo habiéndola cambiando a formato OpenCV, se encuentra en uno de los espacios de color más comunes, RGB. Un sistema basado en mapa de bits en el que se distinguen 3 colores principales: rojo, verde y azul. Es un sistema aditivo, en el que la suma de todos los colores generará el color blanco.

Figura.2.7 Generación de colores en el espacio RGB [9]

Este sistema funciona como una matriz tridimensional, compuesta por 3 planos correspondientes a los 3 colores principales: ROJO, VERDE Y AZUL. Cada plano de color es como una imagen en blanco y negro, el valor máximo de 255 es el color primario puro y los valores comprendidos entre 0 y 255 son tonalidades de este color. Cada uno de los tres planos de color trabaja con 8 bits, es decir que por plano de color se tienen 256 diferentes valores. [32]

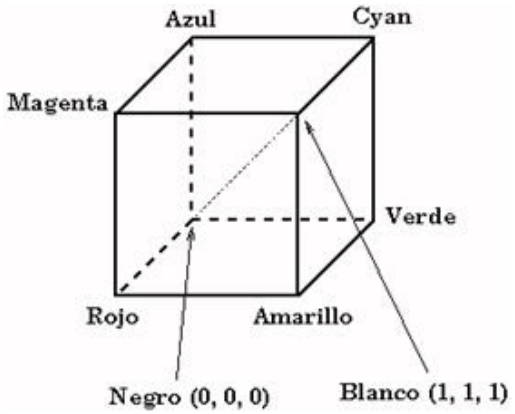


Figura.2.8 Cubo unitario del modelo de color RGB [9]

Este proyecto se enfoca en el reconocimiento y seguimiento de diferentes objetivos según sus características de forma y de color, en un escenario que presente condiciones controladas. Esto hace referencia a un medio en el que la iluminación, la cantidad de objetos presentes en el entorno, el brillo o bien las sombras de estas cosas no representen perturbaciones que dificulten el desarrollo de la aplicación. Este tipo de perturbaciones interfieren de forma significativa en cualquiera de los casos de reconocimiento de objetos que se presenta, porque provocan confusiones especialmente entre el objetivo y el medio.

Tomando en cuenta estas consideraciones, para probar y demostrar cómo van trabajando los algoritmos aplicados en el proyecto paso a paso, se tienen dos escenarios distintos; uno correspondiente a un cartel con diferentes colores (Figura.2.9) que en un principio permitió comprobar la eficiencia de la diferenciación de los mismos.

Pero en situaciones posteriores se puede observar que con el segundo objetivo, un globo (Figura. 2.10), que es más susceptible de las perturbaciones del medio, se puede notar más claramente el grado de impacto o de influencia que tiene cada uno de los filtros y algoritmos que se van implementando en este pre procesamiento de imágenes, primero considerando características de color y posteriormente extrayendo características para reconocimiento por forma. Las imágenes que se presentan a continuación, corresponden a los dos objetivos manejados en formato de imagen para OpenCV y en espacio de color RGB.

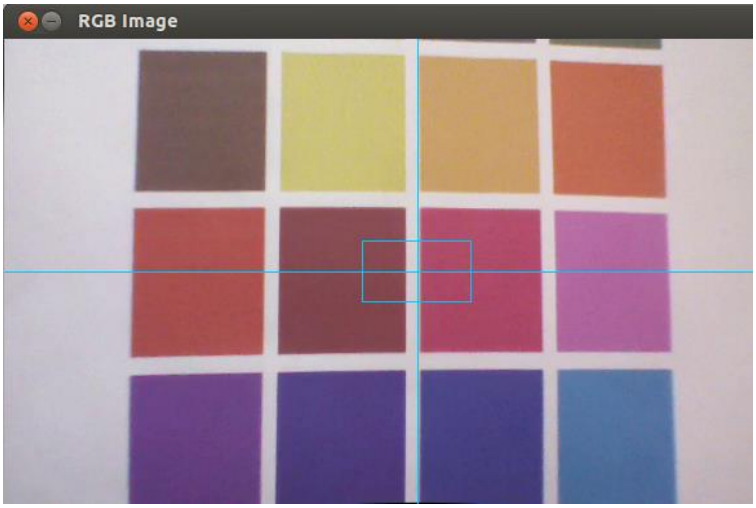


Figura.2.9 Cartel de colores en espacio rgb captado por el AR.Drone

Figura.2.10 Objeto (globo) en espacio rgb captado por la cámara del AR.Drone

A la imagen en espacio de color RGB recibida, se le somete a una serie de filtros, transformaciones y algoritmos con el fin de que sea posible el reconocimiento de un objeto específico de la “escena” captada y su posterior seguimiento.

2.4.1.5 SMOOTH / BLUR IMAGES

El propósito de este paso es “*suavizar o difuminar*” la imagen para reducir el ruido; a través de diferentes filtros lineales, ya que son fáciles de conseguir y tienen una respuesta rápida; los más utilizados con OpenCV son: Homogeneous Filter, Gaussian Filter, Median Filter y Bilateral Filter.

Una imagen se suaviza deslizando una “ventana” a través de la imagen y calculando para cada pixel un valor basado en el núcleo de kernel y el valor de la superposición de pixeles de la imagen original. El núcleo de kernel es la única diferencia entre todos los métodos de suavizado (blurring/visión borrosa). [28]

En este proyecto se usa el *Homogeneous Filter* cuyo código está disponible en la librería de OpenCV para procesamiento de imagen. El efecto de este filtro puede notarse claramente en las siguientes imágenes:

Figura.2.11 Imagen original [33]

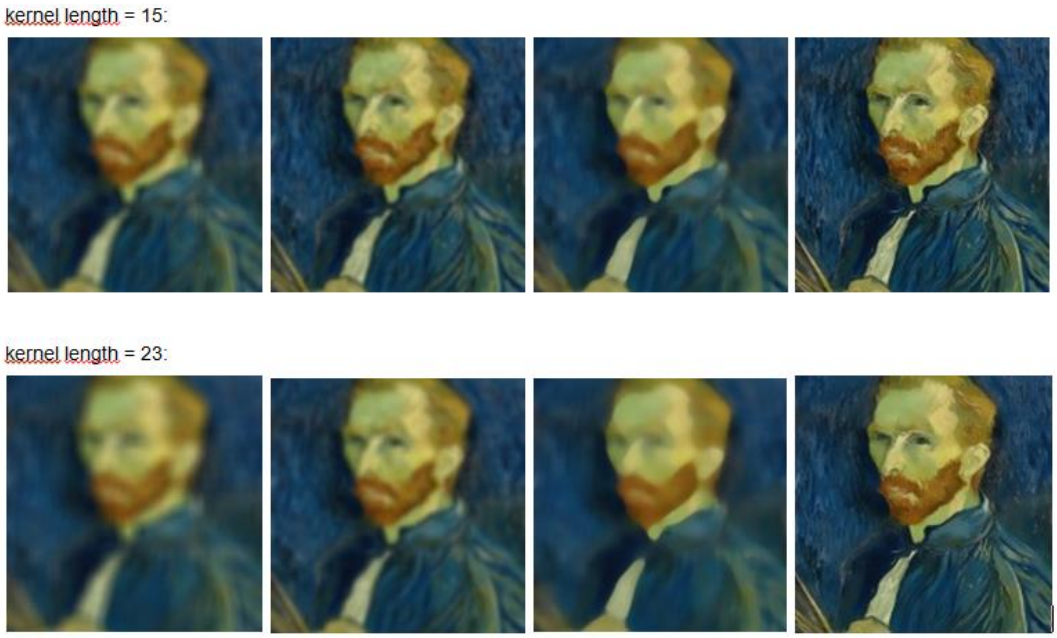


Figura.2.12 De izquierda a derecha: Homogeneous blur, Gaussian blur, Median blur, Bilateral blur. [33]

Líneas de código para aplicar el filtro de *blur* a una imagen cualquiera:

```
1 //Homogeneous blur:
2 blur(image, dstHomo, Size(kernel_length, kernel_length), Point(-1,-1));
```

Figura.2.13 Filtro de Blur

El siguiente paso es transformar la imagen a un formato, o espacio de color más simple, de manera que resulte menos complejo el proceso de identificación y reconocimiento de los colores.

Como se mencionó en párrafos anteriores el video capturado por el cuadricóptero está en espacio de color RGB, ahora se lo transforma a espacio HSV.

2.4.1.6 MODELO CIRCULAR DE COLOR HSV

El espacio de color HSV es más cercano a la idea humana de color. El modelo HSV de sus siglas *Hue, Saturation, Value* es una transformación no lineal del espacio RGB. En el espacio RGB se piensa en el color por sus propiedades sustractivas y aditivas, en el modelo HSV se lo hace en términos más simples como matiz y saturación.

El matiz se representa gráficamente en una región circular, cuyo centro es el blanco. La distancia del centro hacia el color en la región circular define la saturación. Cada color primario está separado del otro 120º, una tercera dimensión es añadida para definir el negro, generando un cono, cuya altura viene dada por el brillo o valor. [9]

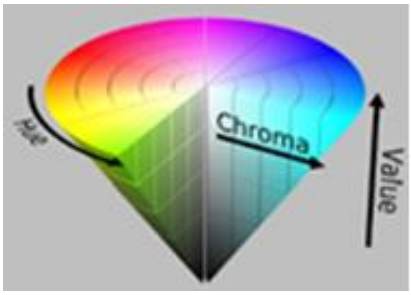


Figura.2.14 Espacio de color HSV [9]

La idea de trasformar la imagen a espacio HSV, es que este espacio de color proporciona información de cromaticidad más intuitiva y sencilla de segmentar, que la que proporciona el espacio RGB; especialmente en regiones de sombras.

La siguiente imagen muestra cómo se ve la captura del objetivo seleccionado, hecha por el AR.Drone, después de su transformación a espacio HSV:

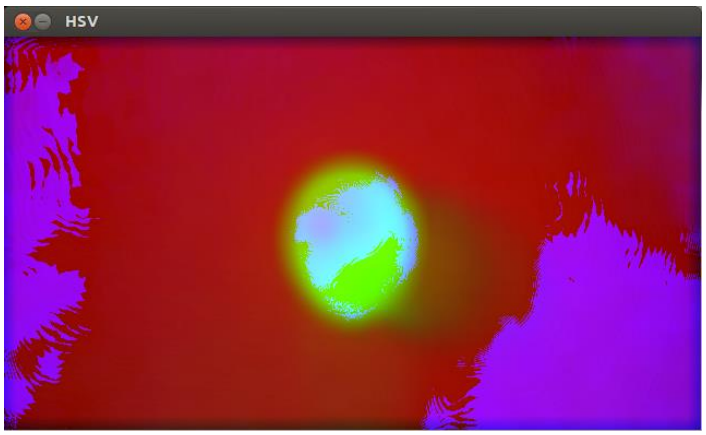


Figura.2.15 Espacio de color HSV

A partir de esto se "*limita*" la tonalidad de los colores de la imagen, específicamente del objeto que se desea seguir, una vez que éste ha sido seleccionado con un CLICK en la ventana que muestra el video capturado por la cámara del AR.Drone.

2.4.2 SEGMENTACIÓN

La segmentación consiste en extraer los objetos en movimiento o regiones de interés de la secuencia de video analizada [34]. La segmentación tanto con RGB

como con HSV se fundamenta en la detección de umbrales del mapa de componentes de estos espacios de color. [35]

Existen ciertos inconvenientes, como que no se tiene un conocimiento a priori del número de objetos que tendrán similares características al objetivo que se desea reconocer; o que las áreas detectadas si bien pueden ser realmente los objetos podrían también ser ruido causado por un mal enfoque, mala iluminación, sombras, brillo o la misma complejidad de colores, además de las texturas, el tamaño de los objetos, así como las variaciones en el fondo [35]. El segmentador deberá elegir aquellas zonas que con mayor probabilidad sean objetos y descartar las zonas que parezcan ruido o zonas pequeñas inconexas que no forman conjuntos compactos.

En este proyecto se maneja un tipo de segmentador conocido como *sustracción de fondo*, que lo que hace es restar o sustraer el fondo de la imagen dejando al descubierto las zonas de interés, pintadas en color blanco. [34]

2.4.2.1 Segmentación en HSV

Considerando la simplicidad del espacio de color HSV, inicialmente se transforma la imagen RGB a HSV como se observa en los párrafos anteriores, luego la idea es separar las tres componentes de color: matiz, saturación y luminancia, para analizarlos y trabajar con ellos.

Los 3 componentes de color deben estar previamente suavizados para conseguir mejoras que permitan obtener los 3 conjuntos de umbrales con mayor éxito. De esta forma se tiene una segmentación del espacio de color. Haciendo una primera aproximación, para detectar objetos de color dentro de una escena, se parte de la idea que la falta de color obedece a dos situaciones: zonas de sombras y zonas de brillo en la imagen. La ventaja es que el espacio de color HSV proporciona una información de falta de cromaticidad más intuitiva y más sencilla de segmentar, de la que proporciona el espacio RGB. [35]

En esta etapa de segmentación, se hace uso del comando de OpenCV, inRange, que se explica a continuación.

2.4.2.2 InRange ( )

La función InRange() de OpenCV es muy similar a la función threshold(), que se detallará más adelante. Se toma una imagen en escala de grises preferentemente y se la convierte en una imagen “binaria”; esto se consigue determinando si un pixel en la imagen está dentro de un rango particular de valores; es decir, se verifica si el pixel está todo blanco o todo negro según su posición dentro de la gama de valores dada.

Se tiene que fijar o comprobar si los pixeles de la imagen son más claros o más oscuros que el umbral dado. El inRange() permite darle al pixel un valor máximo y uno mínimo, y de esta forma segmentar a la imagen para conseguir un objetivo claramente diferenciado del fondo de la escena. [22]

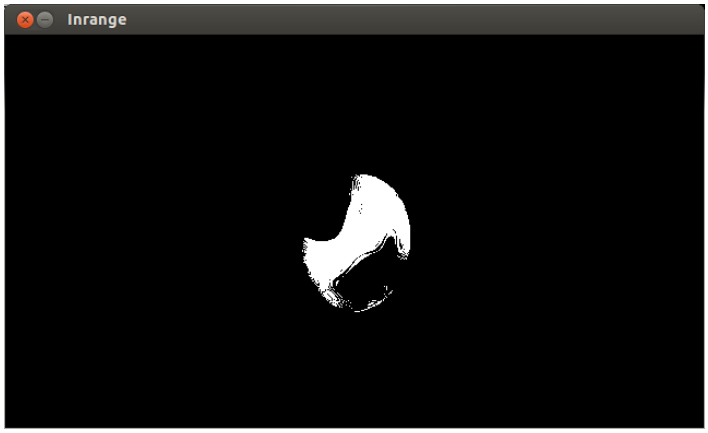


Figura.2.16 Segmentación_inRange( )

2.4.3 EXTRACCIÓN DE CARACTERÍSTICAS

2.4.3.1 Procesamiento morfológico

Lo que corresponde en este punto es conocido como “*PROCESAMIENTO MORFOLÓGICO*”. Esta etapa de procesamiento es necesaria debido a que en la mayoría de los casos la imagen resultante hasta el paso anterior, no da resultados

exactos acerca de los contornos de los objetos seleccionados, siendo en este caso específico algo crítico para el reconocimiento y mucho más para el seguimiento. Suelen aparecer zonas con pixeles mal clasificados que generan bordes imprecisos o discontinuos; es así que se precisa un post – procesamiento de la imagen que realce la forma y la geometría de los objetos. [9]

Para extraer de los objetos sus estructuras geométricas es necesario otro conjunto de forma y tamaño conocido, denominado elemento estructurante.

Figura.2.17 Algunos elementos estructurantes típicos [9]

2.4.3.1.1 Dilatación y Erosión

La erosión es el resultado de comprobar si el *elemento estructurante* está completamente contenido en un conjunto de pixeles. Si algún pixel no corresponde con la limitación impuesta por el elemento estructurante, se pone a cero el pixel central. La consecuencia es una *disminución* o *desaparición* de los elementos más pequeños que el elemento estructurante.

La transformación dual de la erosión es la dilatación. La dilatación en cambio es el resultado de comprobar que el elemento estructurante este al menos contenido en un pixel de un conjunto de pixeles. Si se cumple lo anterior se pone en 1 el pixel central. Consiguiéndose una expansión del objeto en su región fronteriza y un cierre de todas las discontinuidades menores al elemento estructurante. [9]

Teóricamente el resultado de aplicar estos cambios morfológicos en una imagen sería similar a la figura 2.18:

Figura.2.18 (a) Imagen original, (b) Imagen dilatada, (c) Imagen erosionada [9]

Se puede decir que la dilatación y erosión son operaciones duales, porque la dilatación de los pixeles de interés es equivalente a la erosión de los pixeles del fondo. Es importante comprender que estas operaciones no son inversas, y que el resultado que se obtenga al final dependerá totalmente del orden en el que se hayan aplicado a la imagen. Para esta aplicación específica con el AR.Drone, se ha realizado primero una dilatación seguida de una erosión; esta operación se conoce como *CLOSING.* La dilatación permite cerrar agujeros y grietas, y la expansión de los pixeles es parcialmente invertida tras aplicar una erosión; se altera la geometría de los objetos y se suaviza sus bordes, pero se consigue rellenar las fisuras [9]. De esta manera, el resultado obtenido es el siguiente:

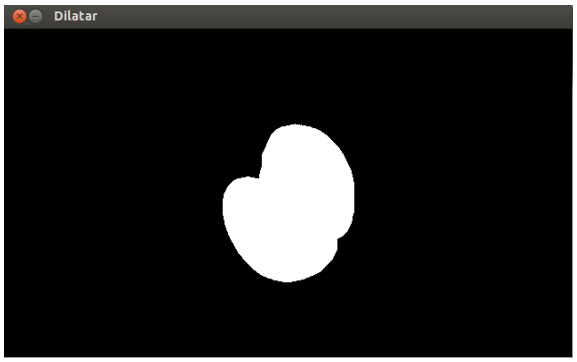


Figura.2.19 Dilatar objetivo

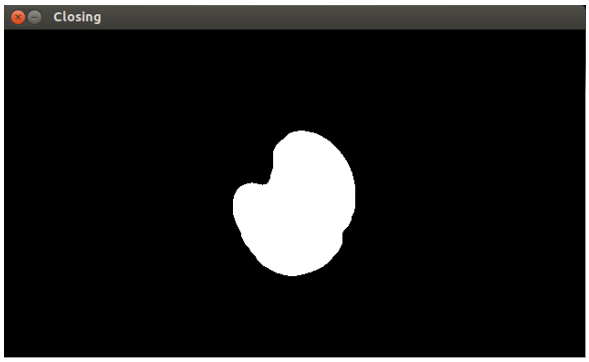


Figura.2.20 Resultado del Closing a la imagen

Después de aplicar esta serie de filtros es necesario tener entendido que la imagen en color blanco es el objetivo identificado, pero puede suceder que haya más de un objeto con las características de color escogidas, de manera que lo que siguiente es determinar la cantidad de objetos que lograron identificarse

según el color seleccionado; y para esto se usa la función *findcontours()* que se explica a continuación:

2.4.3.2 FindContours ( )

Los contornos se definen como una curva que une todos los puntos continuos, a lo largo del límite de la imagen, que tienen el mismo color o intensidad. Los contornos son una herramienta útil para el análisis de forma y la detección y reconocimiento de objetos. Antes de encontrar los contornos, se debe aplicar un filtro threshold o de canny para que el procedimiento sea más eficiente; en este proyecto en lugar de un threshold se utilizó la función InRange() que básicamente provoca el mismo resultado.

En OpenCV, finding contours es similar a encontrar un objeto blanco en un fondo negro, por eso es necesario recordar que el objeto a encontrarse debe ser de color blanco y el fondo debería ser negro. [22]

La función FindContours() calcula contornos de imágenes binarias. Puede tomar imágenes creadas por cvCanny(), que tiene pixeles de borde, o imágenes creadas por funciones como cvThreshold() o cvAdaptiveThreshold(), en el cual los bordes son implícitos como fronteras entre regiones positivas y negativas. Esta función devuelve no solo la identificación de los contornos sino también información sobre estos, características y propiedades tales como: cantidad de objetos, áreas, posiciones, etc. [22]

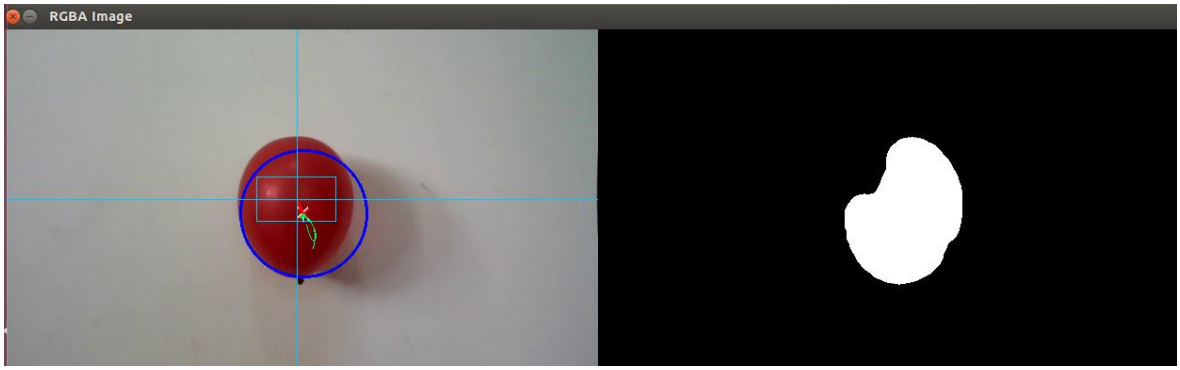


Figura.2.21 Resultado FindContours ()

Para terminar con el reconocimiento por color, es necesario limitar la cantidad de objetos por imagen capturada. Se admiten máximo 4 figuras con la misma característica (color) caso contrario, no se sigue con el reconocimiento porque podría tenerse una situación de error consecuencia de ruido o iluminación. De entre los objetos identificados con el color seleccionado, se escogerá al de mayor tamaño porque se asume que los objetos más pequeños pueden ser defectos de la imagen procesada. Esta situación en particular se trata a detalle en el Capítulo 3 de este trabajo.

2.5 RECONOCIMIENTO POR FORMA

El diagrama de flujo que se muestra en la parte de abajo expone la lógica de programación manejada para conseguir una imagen procesada con la que sea posible el reconocimiento de objetos y su seguimiento considerando su forma geométrica.

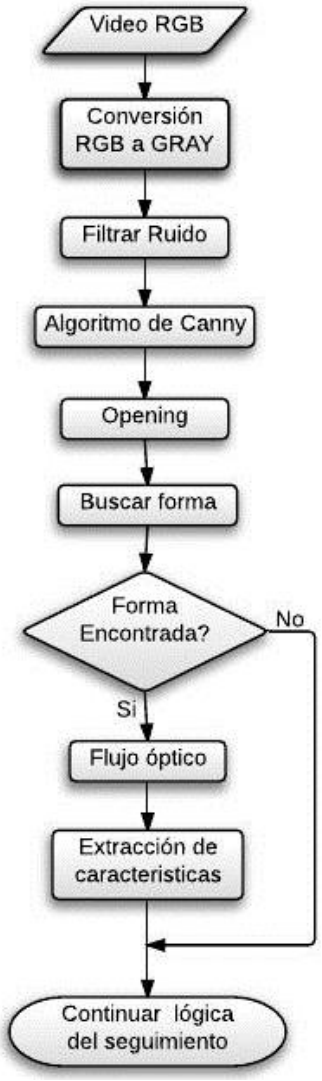


Figura.2.22 Diagrama de Flujo Reconocimiento por Forma

En los párrafos siguientes se puntualizan los filtros y algoritmos más importantes que se utilizan en este tipo de procesamiento de imágenes.

Usando *contornos* con OpenCV, se puede obtener una secuencia de puntos de vértices de cada “mancha blanca” (si se considera que cada mancha blanca es un objeto, un polígono como en el reconocimiento por color), en primera instancia. Además de identificar figuras según el número de vértices se lo puede hacer también por el número de lados calculando la distancia entre estos puntos. Todo lo que se necesita, es una imagen binaria en la que los objetos deben ser blancos y el fondo debe ser de color negro, como se explicó en la aplicación anterior.

De igual forma el reconocimiento y control por visión utilizando ahora las características de formas geométricas necesita pasos o etapas semejantes a las establecidas anteriormente; etapas como son: pre procesamiento, segmentación, extracción de características. Todos estos pasos en general tienen el mismo objetivo y se desarrollan de forma similar que en el reconocimiento por color.

Antes de exponer en detalle los algoritmos que se manejaron en este caso; es preciso saber que la imagen que se va a procesar tiene que estar en *escala de grises* porque este método solo trabaja con imágenes de este tipo [22]. Lo primero que se tiene que hacer es entonces transformar la imagen, es decir el video que se está receptando a través de la cámara del AR.Drone, a formato de escala de grises. OpenCV a través de la función *cvtColor*, permite cambiar una imagen de un espacio de color a otro.

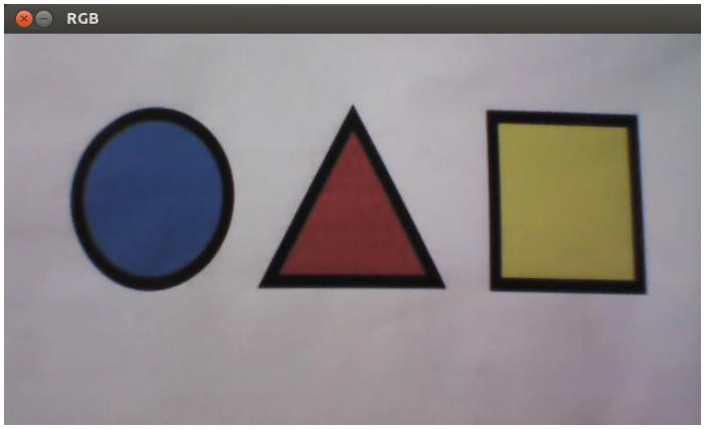


Figura.2.23 Imagen RGB sin procesar

2.5.1 PRE PROCESAMIENTO DE IMÁGENES

2.5.1.1 RGB TO GRAY SCALE

Con la función cvtColor se pueden hacer una serie de transformaciones, pero para esta aplicación se utiliza la RGB – Gray, una transformación en el espacio RGB equivalente a añadir /quitar el canal alfa, invirtiendo el orden de los canales [22]. La conversión de una imagen RGB a escala de grises se consigue con el comando:

```
cvtColor ( src ,  bwsrc ,  CV_RGB2GRAY );
```

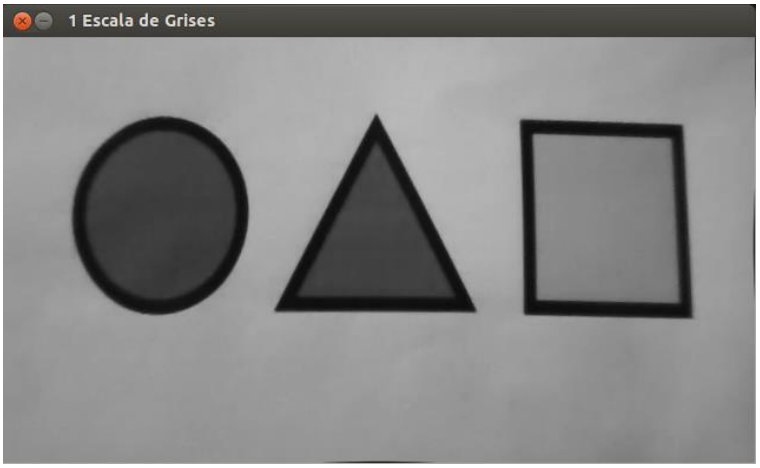


Figura.2.24 Escala de Grises OpenCV

Ahora la imagen debe filtrarse, con el objetivo de eliminar ruido o perturbaciones que pueda tener. Todos los filtros que se aplican a la imagen buscan hacer que el reconocimiento sea más sencillo y en consecuencia el seguimiento también lo sea.

El primer algoritmo que se utiliza juega con la imagen capturada de tal manera que disminuye y aumenta la resolución de la misma redimensionándola. Hay que considerar que cuando se reduce el tamaño de la imagen se está perdiendo información de la misma.

Con las funciones *PyrDown* y *PyrUp* de OpenCV, se puede disminuir y aumentar el tamaño de una imagen respectivamente. En este trabajo, primero se reduce el tamaño de la captura y posteriormente se lo aumenta, obteniendo como resultado una imagen de la que se ha perdido algo de resolución por el hecho de redimensionarla; pero la ventaja es que se logra suavizar los bordes de los objetos siendo más sencillo identificarlos después con algoritmos semejantes. [22]

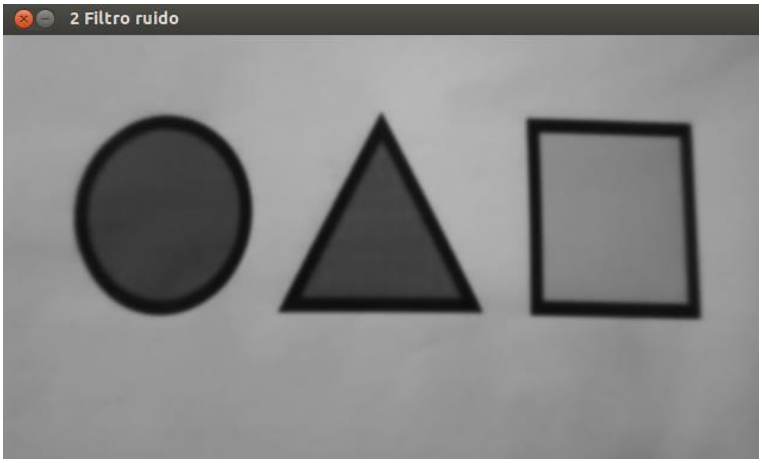


Figura.2.25 Redimensionamiento de la imagen

Lo que sigue es “definir” a los objetos que puedan encontrarse en la imagen para avanzar en la simplificación del reconocimiento del objetivo. De entre los algoritmos para detección de bordes, probablemente el más famoso y utilizado sea el *algoritmo de Canny*.

2.5.1.2 ALGORITMO DE CANNY

Para entender en qué cosiste este algoritmo, se lo resume en 4 pasos principales:

1) *Suavizado:* como se ve en la imagen (Figura.2.26), aplicar el algoritmo de Canny directamente sobre una imagen produce gran cantidad de falsos bordes por el ruido de la imagen. Por ello, es necesario un paso previo de *suavizado* de la imagen, para reducir ese ruido sin alterar la localización de los bordes.

2) *Cálculo del Gradiente:* sobre la imagen suavizada, se calcula el gradiente como las derivadas parciales en *x* y *y*, resultando las imágenes que se muestran en Figura.2.27

3) *Supresión de no máximos:* en este paso se asegura que los bordes no tienen más de un pixel de ancho, es decir que se eliminan los pixeles que no se considere que pertenecen al borde.

Figura.2.26 Resultado del algoritmo de Canny con diferentes suavizados [36]

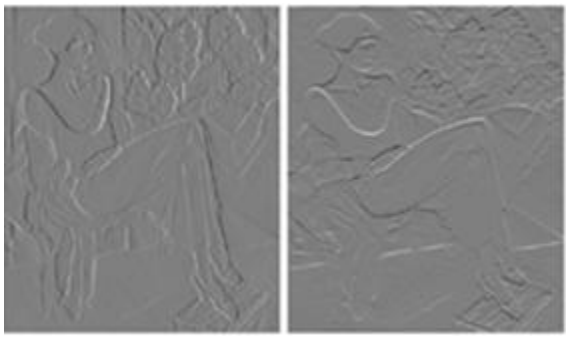


Figura.2.27 Derivadas parciales calculadas en el algoritmo de Canny [36]

4) *Umbralización con histéresis:* finalmente, se binariza la imagen resultante, en la que cada pixel indica si es un borde o no, con diferentes valores de umbrales. [36]

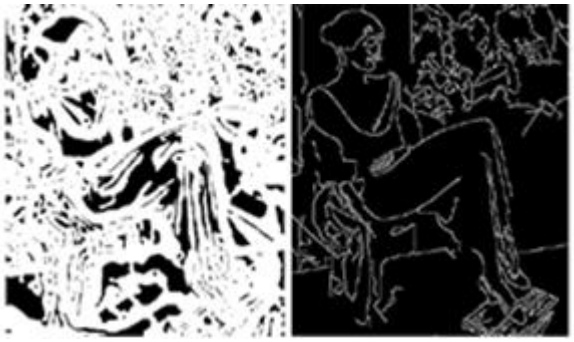


Figura.2.28 Antes y después de la supresión de no máximos [36]

Figura.2.29 Resultado algoritmo de Cany para diferentes valores de umbrales [36]

El objetivo de este proyecto al aplicar el algoritmo de Canny es:

- *Menor proporción de error*: determinando solamente los bordes existentes.

- *Buena localización*: la distancia entre los pixeles de los bordes encontrados y los bordes reales tienen que ser mínima.
- *Respuesta mínima*: sólo una respuesta de detector por borde.

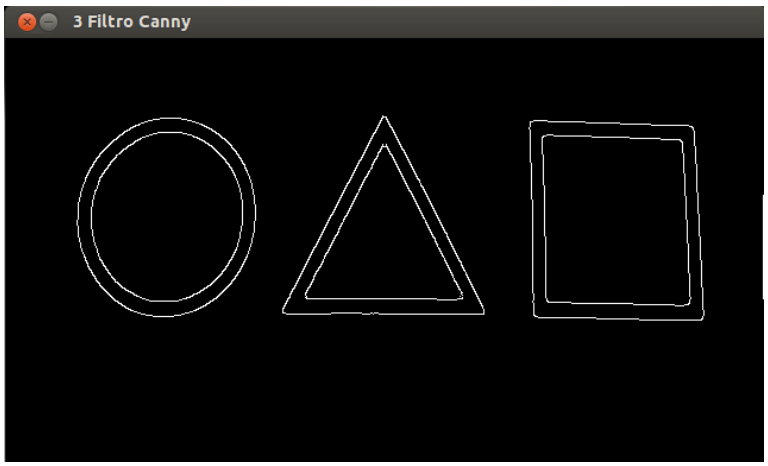


Figura.2.30 Resultado Algoritmo de Canny

Como se vio en el procesamiento de las imágenes por color, es necesario *binarizar* a la imagen que se ha seleccionado como objeto de seguimiento para poder diferenciarlo del entorno eficientemente, y para esto se emplea la función *threshold()* detallada en el apartado siguiente.

2.5.2 SEGMENTACIÓN

2.5.2.1 THRESHOLD( )

Thresholding es tal vez uno de los filtros más simples usados con OpenCV. Éste toma una imagen en escala de grises y la convierte en una imagen “binaria”, en donde todo pixel es blanco o negro dependiendo de si es más claro u oscuro que el límite dado por el threshold.

La función threshold() tiene como argumento un único valor entero entre 0 – 255. Este valor representa el valor de gris (0 será negro y 255 blanco). Después de aplicar el filtro, los pixeles por debajo de este valor serán negros y aquellos sobre el mismo serán blancos. [37]

Las imágenes que se muestran en la siguiente figura, representan el resultado de aplicar este filtro con diferentes valores de threshold.

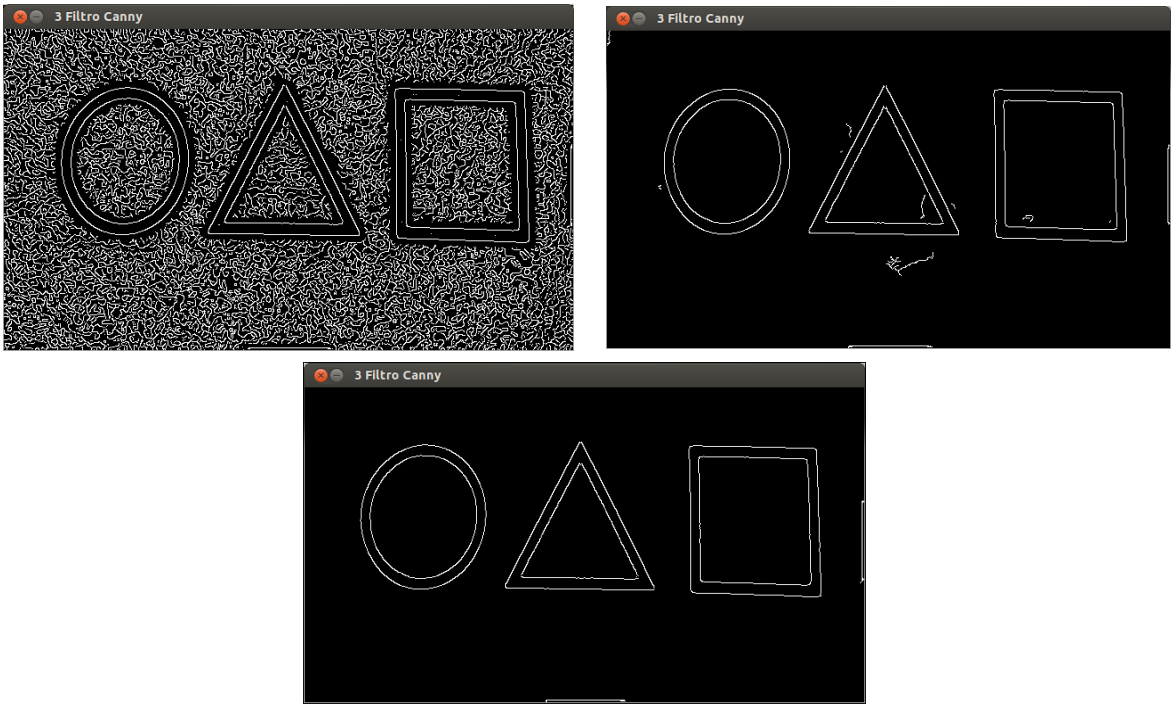


Figura.2.31 Resultado Threshold diferentes umbrales

2.5.3 EXTRACCIÓN DE CARACTERÍSTICAS

Aquí, se somete a la imagen a algoritmos y filtros semejantes a los aplicados en el reconocimiento por color, como son: procesos de dilatación y erosión con la finalidad de realzar la forma y geometría de los objetos en la imagen y evitar encontrarse con bordes imprecisos o discontinuos.

Para definir con claridad los bordes de la imagen considerada como objetivo, se realiza un proceso de *OPENING*; que consiste en realizar primero una erosión que permite eliminar pequeños objetos, pero también se reduce el tamaño de todos los demás objetos en la imagen y después se aplica una dilatación, con la que se vuelve a expandir los objetos a su tamaño inicial menos aquellos pixeles previamente eliminados por la erosión.

Esta operación no preserva estrictamente la geometría de los objetos, ya que erosionando y dilatando solamente se consigue suavizar bordes. [9]

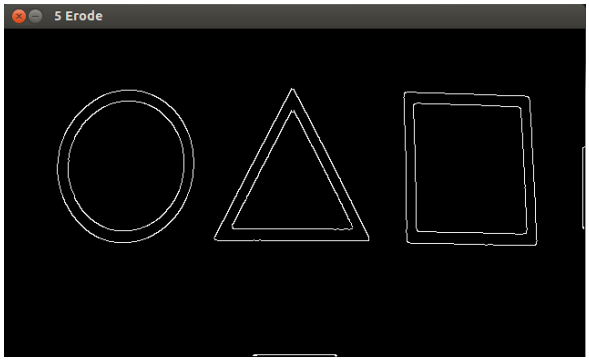


Figura.2.32 Erosionar objetivo

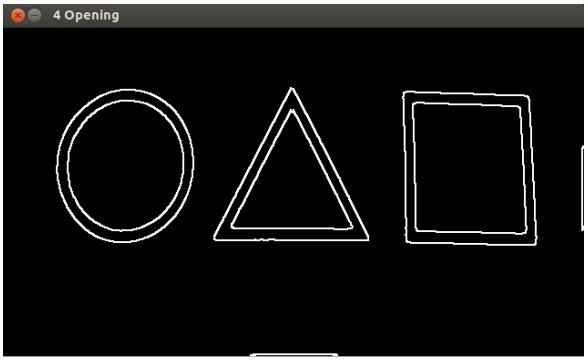


Figura.2.33 Resultado del Opening a la imagen

Además es necesario encontrar los contornos del objeto a seguir con el AR.Drone, para que al compararse con el patrón de forma, puedan identificarse claramente sus características y se consiga clasificarlo adecuadamente.

Haciendo uso de la función findContours(), se recuperan los contornos de la imagen binaria como se había descrito previamente. En esta aplicación en particular de seguimiento por forma, el identificar los contornos del objeto es muy útil para analizar la forma en sí, detectar con precisión el objeto y reconocerlo.

2.5.3.1 Aproximación Poligonal

El objetivo de este post procesamiento de imagen es reconocer en una escena objetos con formas básicas, como círculo, cuadrado, triángulo, etc. OpenCV ofrece la función *approxPolyDP()* para aproximar una curva poligonal con una precisión dada; la idea es aproximar una curva o polígono con otra curva/polígono con menos vértices de manera que la distancia entre las curvas sea menor o igual a la precisión especificada.

La idea es que teniendo una curva compuesta de segmentos de línea, al aplicar este algoritmo, se encuentre una curva similar con menos puntos. Así al final se podrá representar un contorno con un polígono más simple e implícitamente hacer algo de eliminación de ruido, con lo que se logra una forma definida en la que se pueda identificar visiblemente lados, ángulos y otras características importantes. [22]

El siguiente paso para poder finalmente identificar si en el cuadro tomado del video existe alguna de las figuras básicas y poder iniciar el tracking; es comparar el número de lados y el número de ángulos por cada figura que se cierra en la imagen, con las figuras que se tengan como dato, por ejemplo triángulo o cuadrado.

Para identificar un círculo es importante considerar que, se tomó en cuenta que las figuras posibles tendrán 3 o 4 lados, en consecuencia las formas cerradas con más de 5 lados serían círculos. Para asegurarse el programa de que efectivamente se ha encontrado una figura, se ejecuta la función *BoundingRect()* con la que se confirma que las formas registradas sean completamente cerradas.[22]

Una vez identificada la figura es sustancial el optimizar el posterior seguimiento y procurar que a lo largo de este no se vaya a perder el objetivo por confusiones con el entorno, problemas con el movimiento, cuestiones de iluminación, etc., para esto se aplica otro concepto que será de gran utilidad cuando la imagen procesada esté lista para el seguimiento; este nuevo algoritmo es el de *Flujo Óptico.*

2.5.4 FLUJO ÓPTICO

Como se puede ver en este capítulo, para la realización del proyecto era necesaria la implementación de algoritmos de reconocimiento de objetos y seguimiento de los mismos. El seguimiento por forma es un poco más complejo que el seguimiento por características de color, especialmente si no se tiene un ambiente con condiciones controladas en el que no haya perturbaciones importantes; por ello es necesario un algoritmo un tanto más robusto de manera que se facilite el reconocimiento de los objetos.

La idea es evitar como se mencionó, que el objetivo se pierda de vista especialmente por cuestiones de movimiento, como se aprecia en las siguientes imágenes:

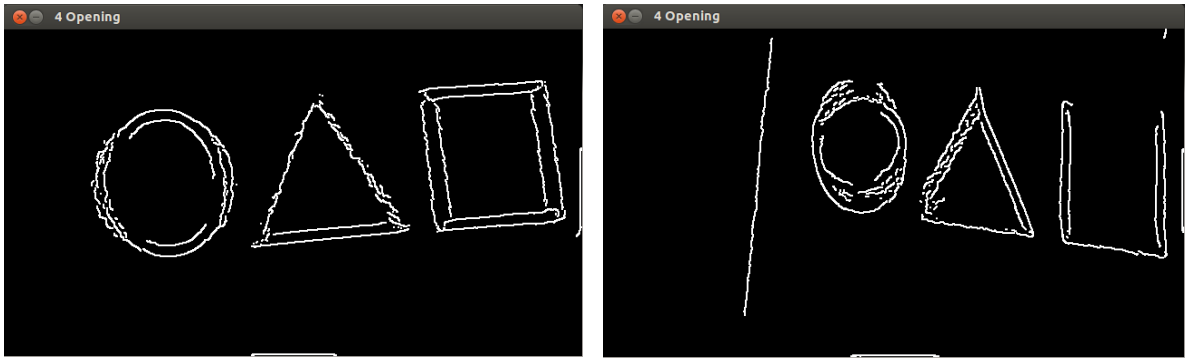


Figura.2.34 Problemas de movimiento y pérdida de objetivos

Uno de los algoritmos más conocidos para seguimiento por forma se llama *Flujo Óptico*, basado en el algoritmo de Lucas y Kanade. El Flujo Óptico se entiende como el movimiento aparente de los píxeles de una imagen de un cuadro a otro dentro de una secuencia de video.

El cálculo del flujo óptico no es un problema sencillo y computarlo para tomas realizadas en un ambiente real puede llegar a ser muy complicado. Por eso los algoritmos existentes como el mencionado se basan en determinadas hipótesis (intensidad constante, rigidez de los objetos, coherencia espacial, entre otras) que generalmente no se cumplen en escenarios reales. [22]

OpenCV ofrece una función en la librería de flujo óptico que permite aplicar el algoritmo de Lucas – Kanade y es: *cvCalcOpticalFlowLK().*

Lucas – Kanade tiene como objetivo encontrar los mejores puntos que definan a un objeto, puntos como las “esquinas”, que tienen la característica de ser encontrarse con mayor facilidad en secuencias de imágenes en video. Estos puntos en esencia significan la presencia de una textura o un borde, y el seguimiento de dichos puntos hace que este algoritmo no dependa de la forma de los objetos a seguir sino más bien de la presencia de los puntos en la imagen, que garantizan la presencia de objetos sin la necesidad de detectarlos.

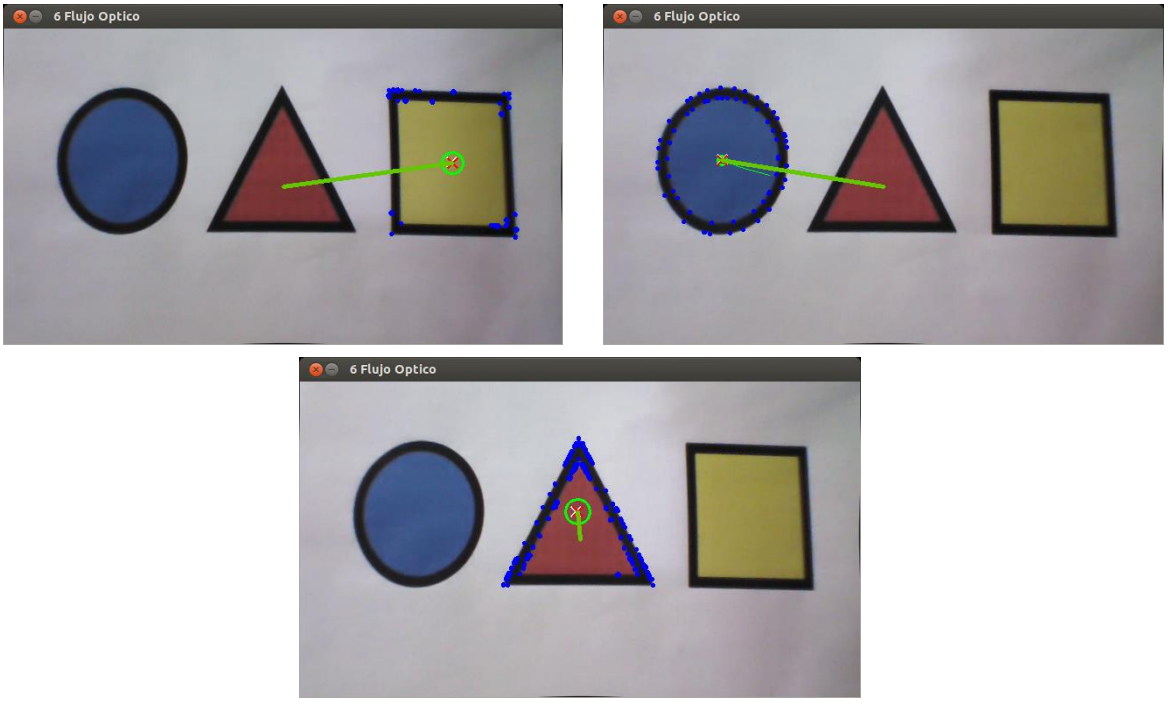


Figura.2.35 Algoritmo de Lucas Kanade para 3 formas diferentes

El hecho de tener estos puntos definiendo el contorno de las figuras permite que con la ayuda de otros algoritmos que se verán posteriormente, el seguimiento sea más preciso, y haya menor posibilidad de perder de foco al objetivo.

CAPÍTULO 3

DESARROLLO E IMPLEMENTACIÓN DEL SISTEMA

3.1 INTRODUCCIÓN

El objetivo del seguimiento se basa en todo momento en tratar de ubicar al objeto buscado (ya sea para el algoritmo desarrollado por color o por forma) en el centro del campo visual de la cámara del cuadricóptero, para ello se busca primero determinar la posición del objeto seguido y posteriormente generar y enviar las respectivas órdenes de control hacia al cuadricóptero a fin de lograr un seguimiento óptimo. El seguimiento visual basado en imágenes (IBVS) [27], inicia con la adquisición y procesamiento de imágenes que permitan la división de la imagen en segmentos manipulables a fin de extraer características representativas para el reconocimiento y la interpretación de los objetos dentro de un espacio de características, para lo cual se utilizan diferentes criterios de control capaces de trabajar en conjunto formando un sistema sólido y eficaz en el seguimiento.

A continuación se muestra el procedimiento que se sigue en el procesamiento de la información embebida en las características más relevantes del objeto, para generar las respectivas órdenes de control en el seguimiento del cuadricóptero al objeto deseado. En el gráfico de abajo se tiene un diagrama de bloques que explica en resumen el proceso que se describe en este capítulo.

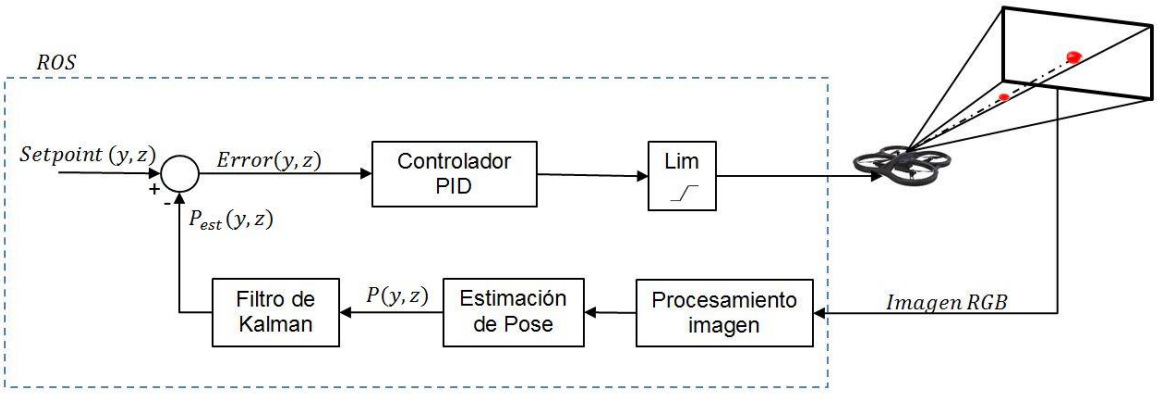


Figura.3.1 Diagrama de bloques del Proceso de Reconocimiento y Seguimiento

3.2 IMPLEMENTACIÓN CON ROS

3.2.1 ¿POR QUÉ SE USA ROS?

El principal motivo por el cual implementar este proyecto con el firmware de ROS, es investigar y conocer los recursos que esta plataforma puede brindar en el desarrollo de aplicaciones con robots, siguiendo el fundamento principal que lo caracteriza, que es apoyar la reutilización de código en la investigación y desarrollo de la robótica.

Usando la infraestructura de ROS en el desarrollo de esta aplicación, se ha creado y configurado algunos nodos que sean capaces de ejecutar la lógica de control en el seguimiento autónomo del cuadricóptero sobre un objeto.

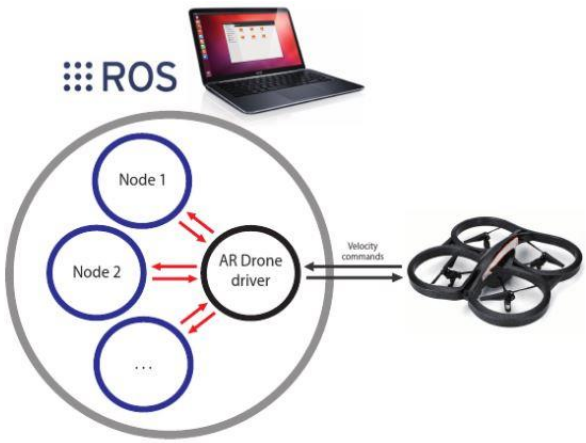


Figura.3.2 Estructura Operacional del Sistema [11]

3.2.2 CREACIÓN Y CONFIGURACIÓN DE UN PAQUETE EN ROS

Se configura una carpeta como área de trabajo de la plataforma, la que contendrá los paquetes de *ardrone_autonomy,* que se encargará de la comunicación con el AR.Drone y el nuevo paquete que se pretende desarrollar para el seguimiento autónomo del cuadrotor hacia un objetivo, a fin de que el firmware sepa su respectivo direccionamiento y las dependencias de cada uno de ellos.

Para crear el nuevo paquete en ROS es necesario, declarar sus diferentes dependencias tanto de librerías como de otros paquetes de los cuales vaya a hacer uso. En este caso, el nuevo paquete dependerá de paquetes estándar de

ROS tales como *image_transport, roscpp, std_msgs, opencv2, cv_bridge, sensor_msgs, ardrone_autonomy, geometry_msgs.*

De igual forma se configuran los archivos *manifest.xml* y *CMakeList.txt* que resultan indispensables para el compilador de *Cmake* utilizado por ROS. En estos archivos se especifican los nodos que se van a correr, los archivos fuente así como los paquete de ROS que se están utilizando en todo el programa

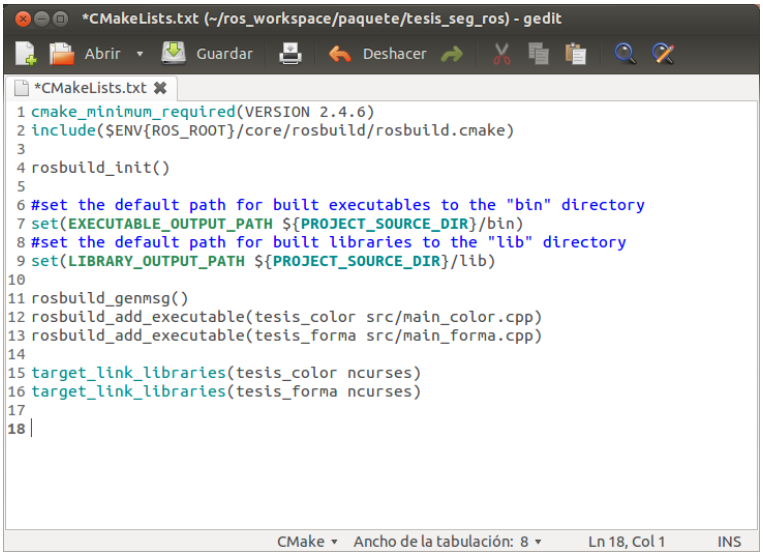


Figura.3.3 Cmakelists.txt

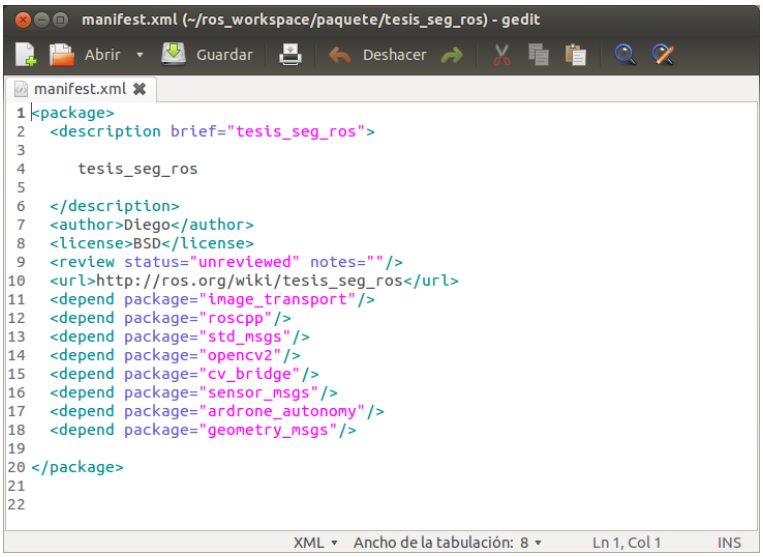


Figura.3.4 Manifest.xml

3.2.3 CREACIÓN Y CONFIGURACIÓN DEL ARCHIVO LAUNCH

Un archivo launch permite correr y ejecutar de manera rápida y eficiente los distintos nodos que previamente se configuran en el mismo, con solo escribir un comando en el terminal. Para este proyecto se han creado y configurado tres archivos launch; uno para correr el nodo de ardrone_driver conjuntamente con el nodo de calibración de la imagen captada por la cámara; cada uno con sus respectivos parámetros, y el segundo y tercer archivo launch para poder correr los nodos de seguimiento con los algoritmos de color y forma respectivamente.

3.2.4 SUSCRIPCIÓN Y PUBLICACIÓN DE TÓPICOS

Como se sabe la comunicación interna en ROS consiste en comunicar los diferentes nodos mediante la suscripción y publicación de tópicos que llevan la información. En este caso se desea recibir y enviar información entre el nodo del *ardrone_driver* y el nodo de “seguimiento”.

3.2.4.1 Suscripción

Para recibir las imágenes trasmitidas por el AR.Drone 2.0, hay que suscribirse al tópico */ardrone/image_raw* en donde llegan las imágenes en su forma original, para luego ser rectificadas por el paquete *image_proc* como se menciona en el capítulo anterior.

También se suscribe al tópico */ardrone/navdata*, que permite tener conocimiento de las variables físicas del estado actual del AR.Drone como: altura, velocidad, aceleración, etc. Que se usarán para comparación entre los valores teóricos y medidos en las diferentes pruebas.

3.2.4.2 Publicación

Para comandar el desplazamiento del cuadrotor para el seguimiento autónomo en sus dos ejes de libertad, se envían los comandos desde el nodo de seguimiento hacia el nodo del ardrone_driver con el tópico cmd_vel.

3.2.5 CORRIENDO EL SISTEMA EN ROS

Para la ejecución de esta aplicación es necesario correr los respectivos archivos launch en el terminal, lo que permite desplegar las ventanas con las imágenes respectivas de cada etapa del seguimiento y los datos que ayudarán al usuario a tener una mejor idea del estado del proceso.

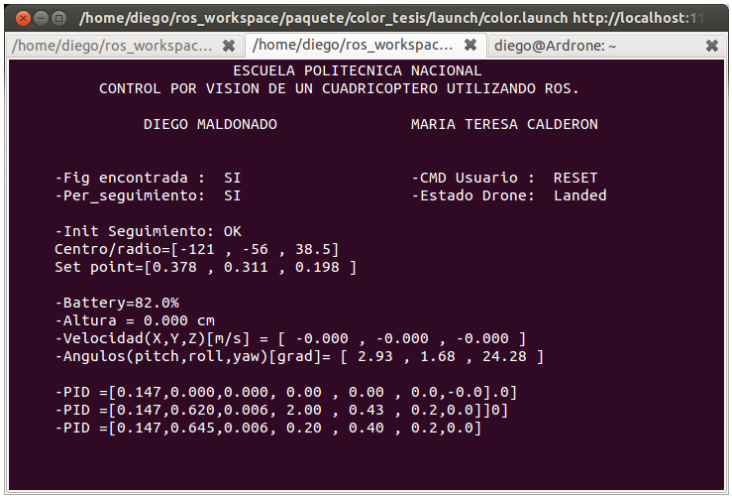


Figura.3.5 Captura de pantalla de la Interfaz de Usuario

El resultado final de la comunicación entre los diferentes nodos mediante los distintos tópicos usados para la implementación de este proyecto se puede resumir con el siguiente diagrama:

Figura.3.6 Esquema de comunicación entre nodos

O de manera completa, a continuación se muestra la comunicación de cada uno de los nodos con todos los tópicos usados para llevar la información respectiva en el procesamiento de imágenes y la determinación de órdenes para controlar al cuadricóptero.

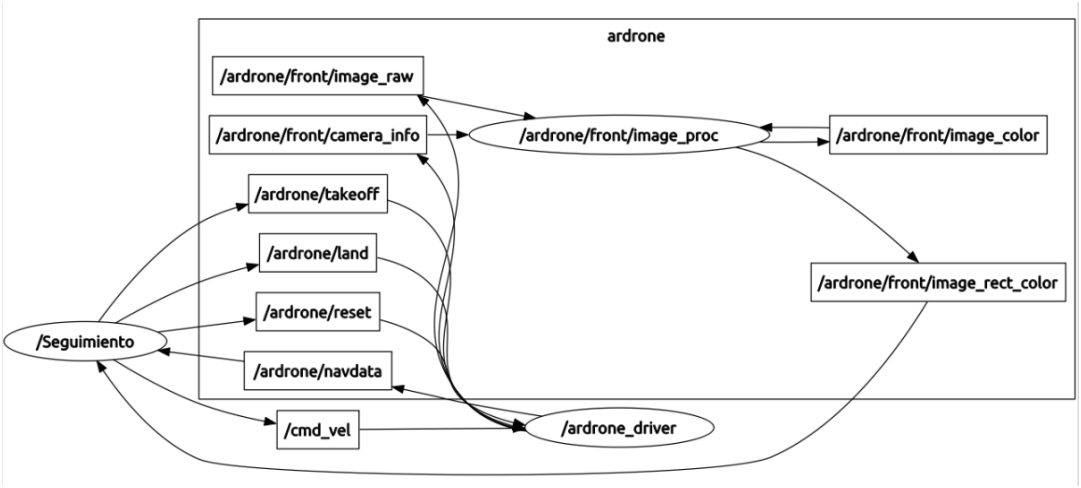


Figura.3.7 Esquema de suscripciones y publicaciones en diferentes temas

3.3 DESARROLLO DE ALGORITMOS DE SEGUIMIENTO

3.3.1 EXTRACCIÓN DE CARACTERÍSTICAS

Después del pre-procesamiento de imagen detallado en el capítulo anterior, se hace una extracción de características que brinda información específica sobre el objeto a ser seguido; información que incluye:

- Ubicación exacta

La ubicación del objeto será el dato más importante en el proceso de seguimiento, porque mediante este se puede determinar el error de posición existente entre el objeto y el cuadricóptero, error que se desea corregir.

Para restarle complejidad al procesamiento de datos y, mejorar la respuesta del seguimiento, la ubicación del objeto se fija respecto de un solo punto, el cual es el centro del mismo objeto.

- Área

El objeto como tal cubre una cierta área dentro del campo visual de la cámara, dicha área se la utiliza para establecer un control alternativo para intentar mantener la distancia que se tiene cuando se inicia el seguimiento entre el objeto y el cuadricóptero, ya que no se cuenta con un sistema de visión estereoscópica que permita establecer el seguimiento considerando la profundidad.

3.3.2 ESTIMACIÓN DE POSE

El Ar.Drone 2.0 tiene abordo una cámara gran angular, diagonal de 90° con una resolución de 1280 x 720 pixeles (720p) [40], siendo una cámara HD de alta resolución.

Para mejorar la velocidad de procesamiento de las imágenes, se trabaja con la mitad de la resolución original de la cámara, es decir con una resolución de 640 x 360 pixeles [19], esto ayudará a que el sistema operativo usado en este proyecto, procese de forma más rápida cada una de las imágenes a fin de que las órdenes de control sean enviadas al cuadricóptero lo más rápido posible para cumplir con un seguimiento eficaz.

En la Figura.3.8 se muestra el área del campo visual de la cámara frontal del cuadricóptero, en la cual el objeto a ser seguido debe visualizarse claramente para poder procesar las respectivas órdenes de control, caso contrario si el objeto buscado ya sea por forma o color no se encuentra dentro de esta área, el AR.Drone entra en modo *HOVERING* o de *PLANEAMIENTO*.

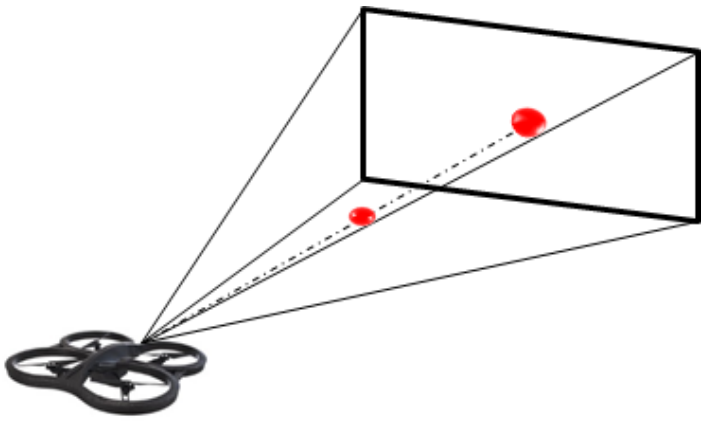


Figura.3.8 Campo visual de la cámara (a)

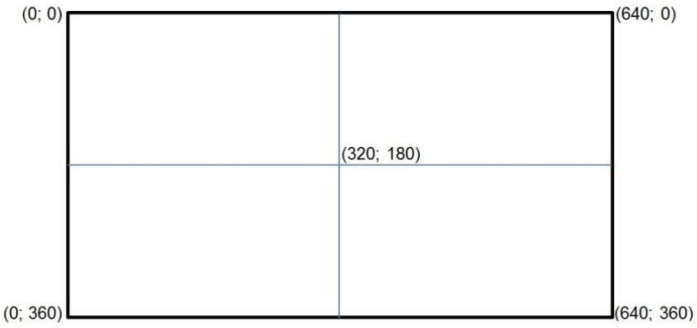


Figura.3.9 Campo visual de la cámara 640x360p (b)

La posición del objeto dentro del campo visual está establecida por dos ejes; de ladeo (roll) y altura (height) de acuerdo a los ejes de desplazamiento del cuadricóptero. Entonces, para el sistema de coordenadas establecido el objeto en cualquier momento de su trayectoria puede estar ocupando una posición $P(y,z)$ como se muestra en la figura a continuación:

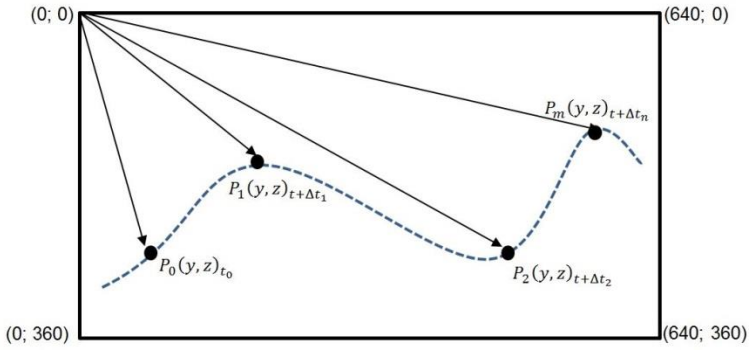


Figura.3.10 Ubicación del objeto en cualquier tiempo

Como se puede notar, al desplazarse el cuerpo, el punto centro no tiene una trayectoria definida dentro del campo visual de la cámara, sino que marca una trayectoria irregular que varía tanto por el movimiento del mismo objeto como por el movimiento del cuadricóptero intentando seguirlo y, a eso se suman los efectos externos que influyen en el procesamiento de la imagen (cambio de iluminación, aparición de sombras, etc.). Este comportamiento causa que el sistema al intentar seguir al objeto se vuelva inestable debido a su fácil variación.

La herramienta que se usa para establecer una corrección y tratar de que la trayectoria sea más suave y continua es un *Filtro de Kalman*.

3.3.2.1 Determinación del error de Pose

El objetivo del seguimiento se basa en todo momento en tratar de ubicar al objeto buscado (ya sea para el algoritmo desarrollado por color o por forma) en el centro del campo visual de la cámara, es decir que dicho objeto se ubique en el punto (320; 180) de la resolución establecida.

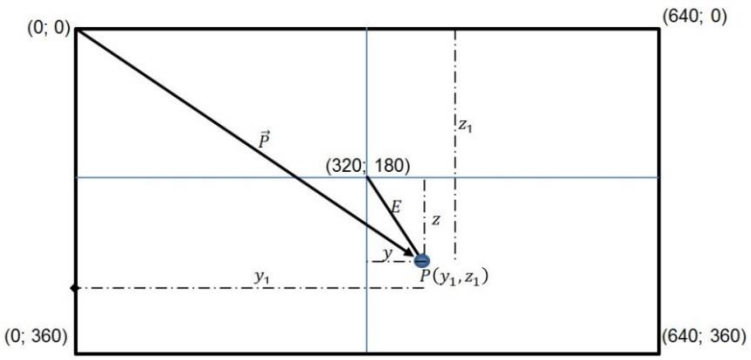


Figura.3.11 Pose del centro del objeto

Para lo cual el *set point* viene a ser siempre el punto centro del campo visual de la cámara, y por ende la diferencia entre este punto y el punto centro del objeto es el error de posición a corregir. Suponiendo que el objeto buscado se encuentra en el punto $P(z_1, \mathrm{y}_1)$ respecto a la esquina superior izquierda del campo de vista de la cámara; para saber el error de posición del centro del objeto respecto al centro de la cámara simplemente se saca la diferencia entre estos dos puntos, es decir;

$$E(y, z) = P(y_1, z_1) - P(320{,}180) \tag{3.1}$$

De acuerdo a la ubicación del set point, el rango de valores que el error de posición podría tener va desde -320 hasta 320 para el eje de ladeo (eje Y), y desde -180 hasta 180 para el eje de altura (eje Z), como se muestra en la figura 3.12.

Figura.3.12 Límites del error de posición

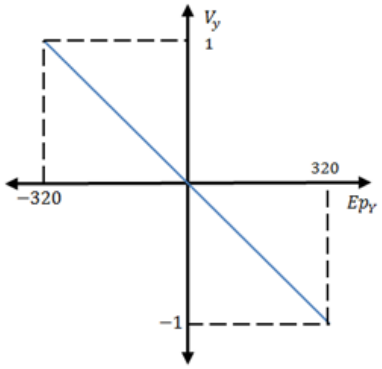
3.3.2.2 Conversión error de Posición- velocidad

Para controlar al AR.Drone usando el firmware de ROS, se hace uso del paquete a*rdrone_autonomy* [40], que internamente tiene un nodo llamado *ardrone_driver*, como se explica en el capítulo 1 de este trabajo; este nodo es el que permite comunicarse con el cuadricóptero, para enviar y/o recibir datos. Para publicar los comandos de vuelo se usa el tópico *cmd_vel* con un rango que oscila en cada variable entre -1 y 1.

La metodología usada para la conversión del error de posición a set point velocidad, consiste en trabajar dicha conversión de forma lineal; es decir se relaciona los límites máximos del error de posición que el objeto puede llegar a alcanzar, con respecto a los límites máximos de los comandos de velocidad lineal que se pueden enviar en cada eje respectivamente. Por ejemplo, para en el eje de ladeo (roll) los límites del error están entre -320 y 320 para cuando el objeto se encuentre en el extremo izquierdo y derecho del campo de vista de la cámara respectivamente.

De acuerdo al driver que maneja los comandos, cuando el objeto se encuentre en el extremo izquierdo siendo el error igual a -320 en este eje, se debería enviar el comando de control con el máximo valor que en este caso seria 1 para que el cuadricóptero se dirija hacia la izquierda hasta el momento que el objeto se ubique cerca del centro del campo visual o set point. Y viceversa, si el objeto se encuentra en el extremo derecho del campo de vista, el error de posición sería

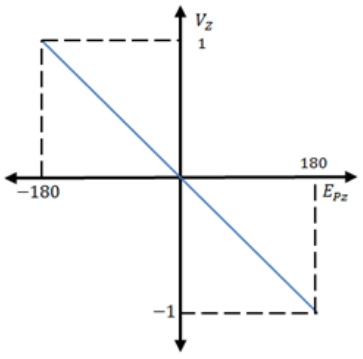
320, para lo cual se debería enviar el valor máximo, es decir -1. Dicho comportamiento esta manejado por la siguiente ecuación y representada por su gráfica.



$$V_y = -\frac{Ep_y}{320} \tag{3.2}$$

Figura.3.13 Conversión Error de Posición-Velocidad eje de Ladeo

Para el eje de altura, se tiene el mismo principio que en el del eje de ladeo, pero en este caso se debe enviar el valor -1 como comando cuando el objeto tenga un error de posición de 180 respecto del set point, para que suba el cuadricóptero, y 1 cuando el objeto tenga un error igual a -180 para que el cuadricóptero descienda hasta que el objeto este dentro de la *zona de histéresis*, que se explicará más adelante. De la misma forma este comportamiento puede ser expresado por la ecuación y gráfica a continuación.



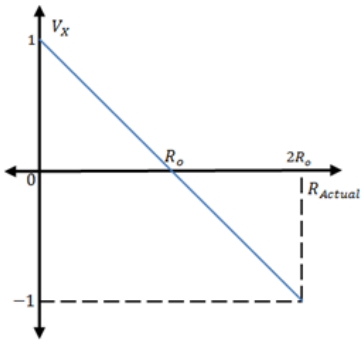
$$V_z = -\frac{E_{Pz}}{180} \tag{3.3}$$

Figura.3.14 Conversión Error de Posición-Velocidad eje altitud

En el caso del eje de cabeceo (pitch), al momento de seleccionar el objeto a seguirse, se guarda el área que cubre dicho objeto en el campo visual, aproximándola al área de un círculo y se calcula su radio inicial R_o. La idea en

este eje particularmente es que el radio del objeto casi siempre tenga el mismo valor, para ello se supone que cuando el objeto se acerca mucho a la cámara y su valor es el doble del valor inicial, es decir $2R_o$, se enviaría el valor máximo del comando; es decir -1, para que el cuadricóptero se aleje del objetivo.

Al contrario cuando el objeto se aleje, y por ende el radio comience a disminuir hasta que su valor sea cero, entonces es necesario que se envíe el máximo valor de 1, para que el cuadricóptero avance hacia adelante. Igualmente que en los anteriores casos dicho comportamiento se lo presenta en base a la siguiente ecuación y a su representación gráfica.



$$V_X = 1 - \frac{R_{Actual}}{R_o} \tag{3.4}$$

Figura.3.15 Conversión Variación del Radio-Velocidad eje de cabeceo

3.3.3 FILTRO DE KALMAN

El Filtro de Kalman se incorpora como herramienta para conseguir la estabilización del cuadricóptero después de implementar el algoritmo de estimación de pose, con el fin de suavizar la trayectoria que sigue el AR.Drone y corregir los errores de estimación en la misma; errores que son causados por los desplazamientos tanto del objeto, como del cuadricóptero y efectos externos a lo largo del recorrido. [38][39]

El filtro es una ecuación de vectores que describe la evolución del estado con el tiempo, es decir mediante el conocimiento del comportamiento actual el filtro de Kalman estima el estado futuro del sistema aplicando un término de corrección proporcional al factor de predicción. Para desarrollar el filtro, es importante diseñar un *OBSERVADOR,* lo que implica estimar el vector de estado a partir de

las salidas del sistema. En este caso por ejemplo, el vector de estado se definiría para la posición del centro del objeto en su desplazamiento por los ejes de ladeo y altura respectivamente y, la variación del área del mismo así $[Py, Pz, A_{obj}]$, y su respectiva variación de velocidad $[\Delta V_Y, \Delta V_Z, \Delta A_{obj}]$.

$$X_k = F.X_{k-1} + W_k \tag{3.5}$$

$$\begin{bmatrix} Py_k \\ Pz_k \\ A_k \\ \Delta y_k \\ \Delta z_k \\ \Delta A_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Py_{k-1} \\ Pz_{k-1} \\ A_{k-1} \\ \Delta y_{k-1} \\ \Delta z_{k-1} \\ \Delta A_{k-1} \end{bmatrix} + W_{t-1} \tag{3.6}$$

Donde X_{k-1} es el vector de estado (posición – velocidad), F es la matriz de transición de estados del sistema, y W_k es el vector que representa el ruido asociado al modelo del sistema.

Para la predicción del estado del objeto es necesario tener un vector de medidas que, usado en un sistema visual serían los valores de la posición y el área del objeto, y está representado por:

$$Z_k = \begin{bmatrix} \overline{Y_k} \\ \overline{Z_k} \\ \overline{A_k} \end{bmatrix} + V_k \tag{3.7}$$

Donde Z_k es el vector de medidas, $[\overline{Y_k}, \overline{Z_k}, \overline{A_k}]^t$ es la posición del objeto respecto a la esquina superior izquierda del campo de vista de la cámara y, V_k es el vector aleatorio que modela la incertidumbre asociada a las medidas. Es muy importante tener en cuenta que el ruido del proceso W_k y el ruido de medición V_k son vectores aleatorios gaussianos con matrices de covarianza Q_k y R_k respectivamente, obtenidas de forma experimental.

Finalmente el resultado del filtro de Kalman es la estimación de pose del centro del objeto y también la estimación del área del mismo; esto permite tener

trayectorias más suaves y finas sin variaciones considerables que vuelvan al sistema oscilatorio.

A continuación, se muestra la estabilización que el filtro de Kalman permite tener en el seguimiento de un globo rojo (reconocimiento por características de color), y a manera demostrativa se ha pintado de color azul la trayectoria original del centro del objeto, y de color verde la trayectoria resultante con la aplicación del filtro de Kalman.

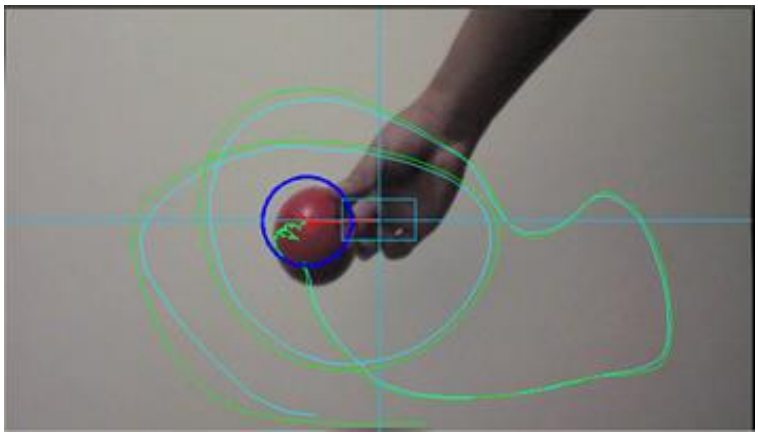


Figura.3.16 Suavización en la trayectoria del objeto

En este punto en el que ya se ha identificado el objeto y se es capaz de conocer su ubicación y el área que ocupa, se da inicio a la nueva etapa del seguimiento, la cual es tratar de mantener siempre dicho objeto dentro del campo visual de la cámara alrededor de un punto fijo, el centro visual de la cámara.

3.3.4 ÁREA DE HISTÉRESIS

3.3.4.1 Histéresis para los ejes de ladeo y altura

El uso de un *área de histéresis* ayuda a que el sistema no se comporte oscilatoriamente, evitando que el cuadricóptero entre en un movimiento de *vaivén*; es decir se tiene un área libre cerca del centro del campo visual donde el centro del objeto pueda estar sin que el controlador ejecute alguna acción, al tiempo que se envía la orden para que el cuadricóptero se quede en la posición actual en la que se encuentre, para el eje de ladeo y de altura. Esta zona de histéresis

comprende el 10% del total de pixeles de cada eje, más o menos un área igual a 64 x 36 pixeles, como se muestra en la figura 3.17.

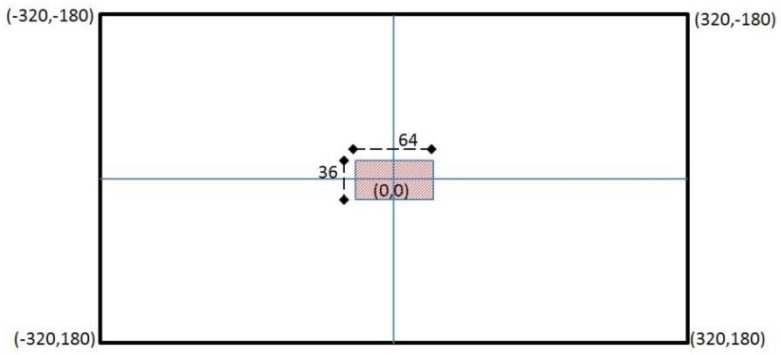


Figura.3.17 Área de Histéresis para eje de ladeo y altitud

3.3.4.2 Histéresis para el eje de cabeceo

Para el eje de cabeceo también se establece una zona de histéresis, la cual se ha establecido teniendo como base el área que el objeto cubre dentro del campo visual de la cámara.

Como se ha visto en el procesamiento de la imagen, el área en muchos de los casos no tiene una forma definida por efectos de cambio de iluminación entre otros factores; por ello esta área se aproxima al área de un círculo de la que se pueda obtener su radio, siendo esta la variable que permitirá realizar el control en este eje.

Así la zona de histéresis para el eje de cabeceo, se determina teniendo en cuenta el 5% del radio máximo que el objeto puede alcanzar dentro del campo de vista de la cámara, es decir $\Delta_{radio\_max} = 180(5\%) = 9$, para que en caso de que la variación del radio del objeto seguido sobrepase el valor máximo al acercarse o alejarse del cuadricóptero, el controlador corrige el error intentando que el objeto en casi todo momento mantenga su radio constante.

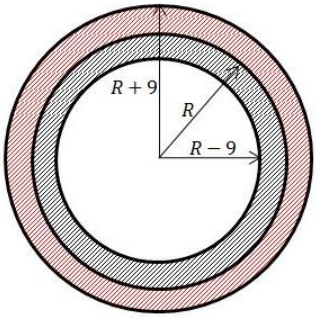


Figura.3.18 Área de Histéresis para eje de cabeceo

3.4 MODELACIÓN DEL CUADRICÓPTERO

Para este proyecto se ha hecho uso de un micro VANT en configuración de cuatro rotores coplanarios (cuadricóptero). [40]

3.4.1 DESCRIPCIÓN DE VUELO CARACTERÍSTICO DEL CUADRICÓPTERO.

Figura.3.19 Descripción del vuelo del Ar.Drone 2.0 [40]

El sistema referencial de este cuadricóptero se basa en el uso de una IMU a bordo del mismo, el cual le permite orientarse basándose en el principio de los ángulos de Euler [11]. Para lograr el desplazamiento del cuadricóptero por sus diferentes ejes se debe cambiar la fuerza ejercida por cada rotor, es decir se debe cambiar la velocidad de giro del rotor para que la hélice pueda ejercer mayor o menor fuerza de empuje de acuerdo a lo requerido. Para conseguir el movimiento hacia adelante o cabeceo (pitch), los rotores 1 y 2, deben bajar su velocidad, y simultáneamente los rotores 3 y 4, deben aumentarla. Para un movimiento opuesto, debe realizarse la secuencia en forma inversa.

El desplazamiento hacia la izquierda se ejecuta siguiendo el mismo procedimiento por pares, pero en este caso se baja la velocidad en los rotores 1 y 4, y se aumenta la velocidad de los rotores 2 y 3. Para irse hacia la derecha se debe realizar lo opuesto.

El movimiento de guiñada o giro (yaw) se lo obtiene a partir de la diferencia en el par de torsión entre cada par de rotores, es decir se acelera los rotores con sentido horario y se desacelera los rotores en sentido anti-horario. Y finalmente para movilizarse hacia arriba se aumenta la velocidad de todos los rotores al mismo tiempo, y para su descenso se baja la velocidad de los mismos. [11]

3.4.2 MODELADO DEL AR.DRONE

Se considera un marco de referencia inercial y un marco de referencia fijo en el cuadricóptero, que se asume como un cuerpo rígido en el espacio, con una fuerza principal (empuje) y tres momentos (pares) [42]. La orientación del cuadricóptero está dada por los tres ángulos de Euler: θ para el cabeceo, ϕ para el ladeo y ψ para el guiñado, como se muestra en el siguiente gráfico:

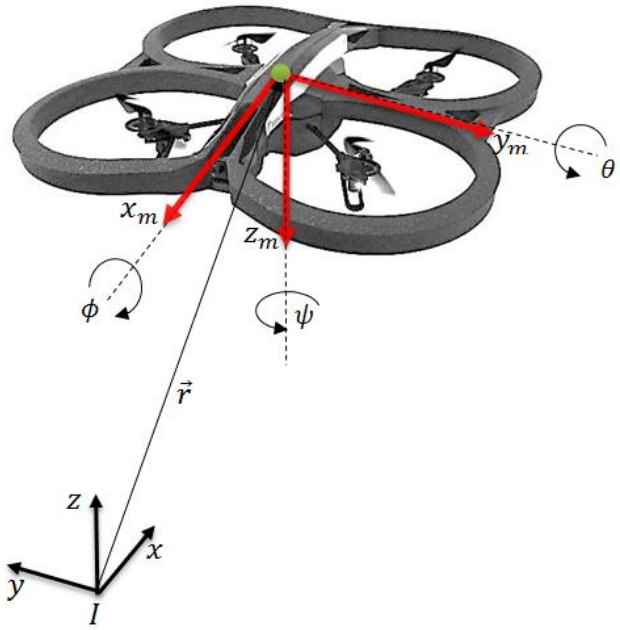


Figura.3.20 Sistema de referencia para la modelación

Es necesario para el control del cuadricóptero estimar la posición y orientación del vehículo respecto a un sistema de referencia inercial, por lo que se llevan a cabo una serie de trasformaciones aplicadas a los vectores del marco referencial fijo en el cuerpo del AR.Drone para tenerlos en el marco de referencia inercial. Esto se representa mediante la matriz rotacional R [41].

$$R = \begin{bmatrix} cos\psi cos\theta & cos\psi sen\theta sen\phi - sen\psi cos\phi & cos\psi sen\theta cos\phi + sen\psi sen\phi \\ sen\psi cos\theta & sen\psi sen\theta sen\phi + cos\psi cos\phi & sen\psi sen\theta cos\phi - cos\psi sen\phi \\ -sen\theta & cos\theta sen\phi & cos\theta cos\phi \end{bmatrix} \quad (3.8)$$

Se puede notar que el modelo para este cuadricóptero, se puede dividir en dos subsistemas; el primer subsistema es la posición del cuadricóptero en el espacio y el segundo su orientación. Cabe aclarar que la posición va a depender de la orientación que tome el cuadricóptero, lo que conlleva a diseñar un lazo interno y otro externo, en donde el primero controla postura, de acuerdo a los ángulos deseados que recibe del lazo externo.

Observando la figura 3.20, el vector *r* nos muestra la posición del cuadricóptero dentro del marco de referencia inercial.

$$\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (3.9)$$

Analizando al cuadricóptero en un diagrama de cuerpo libre, se observaría que las únicas fuerzas que afectan la posición son aquellas que surgen por el empuje de los rotores y la gravedad. Este comportamiento se representa así:

$$\vec{r} = \frac{R.b}{m}.\sum {\omega_i}^2 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - g.\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad (3.10)$$

Siendo R la *Matriz Rotacional*, b el *coeficiente de empuje* de las hélices y m la *masa* del cuadricóptero.

El modelo escogido en base a la literatura analizada; presenta un “modelo” para el control de la velocidad del cuadricóptero de forma aproximada, ya que éste realmente es no lineal, por ello se linealiza alrededor de un punto de equilibrio, el mismo que se ha representado en variables de estado [42].

$$\begin{cases} \dot{\omega} = \begin{bmatrix} 0 & 1 & 0 & \dot{0} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \omega + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{l}{I_x} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{l}{I_y} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} & 0 \end{bmatrix} u \\ u = [u1 \quad u2 \quad u3 \quad u4 \quad g] \end{cases} \tag{3.11}$$

Donde,

$$\omega = \begin{bmatrix} z \\ \dot{z} \\ \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \\ \psi \\ \dot{\psi} \end{bmatrix} \tag{3.12}$$

Las variables I_x I_y e I_z son los momentos de inercia específicos del cuadricóptero, tomados del paper “*Closing the gap between simulation and reality in the sensor and the motion models of an autonomous AR.Drone*”. [45]

Según el modelo matemático del cuadricóptero se puede notar que se tienen cuatro subsistemas desacoplados, lo que lleva a concluir que el sistema puede descomponerse en sus distintos ejes para realizar su análisis y control de forma independiente. El modelo que describe el comportamiento del cuadricóptero por cada eje, es representado por el siguiente modelo en variables de estado.

$$\dot{\omega} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \omega + \begin{bmatrix} 0 \\ r \end{bmatrix} u \tag{3.13}$$

Comparándolo con el modelo de otros sistemas, éste se parece a una masa sin amortiguamiento que puede ser controlada mediante su entrada. Representándolo mediante su función de transferencia se tiene:

$$G = \frac{r}{s^2} \tag{3.14}$$

A continuación se presenta una tabla con los modelos en variables de estado y funciones de transferencia para cada eje de movimiento del AR.Drone.

Tabla.3.1. Modelo del AR.Drone por eje

Eje	Variables de estado	Función de trasferencia
Altura	$\begin{cases} \dot{\omega} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \omega + \begin{bmatrix} 0 & 0 \\ \frac{1}{m} & -1 \end{bmatrix} u \\ v = [1 \quad 0]\omega \quad u = [u_1 \quad g] \end{cases}$	$G(s) = \frac{1}{m.s^2}$
Cabeceo	$\begin{cases} \dot{\omega} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \omega + \begin{bmatrix} 0 \\ l/I_x \end{bmatrix} u \\ v = [1 \quad 0]\omega \quad u = [u_2] \end{cases}$	$G(s) = \frac{l}{I_x.s^2}$
Ladeo	$\begin{cases} \dot{\omega} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \omega + \begin{bmatrix} 0 \\ l/I_y \end{bmatrix} u \\ v = [1 \quad 0]\omega \quad u = [u_3] \end{cases}$	$G(s) = \frac{l}{I_y.s^2}$
Guiñado	$\begin{cases} \dot{\omega} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \omega + \begin{bmatrix} 0 \\ 1/I_z \end{bmatrix} u \\ v = [1 \quad 0]\omega \quad u = [u_4] \end{cases}$	$G(s) = \frac{1}{I_z.s^2}$

Para controlar el desplazamiento del cuadricóptero orientado al seguimiento del objetivo, se utiliza un controlador PID discreto, debido a su rápida respuesta correctora del error. En los siguientes párrafos se presenta una descripción más detallada.

3.4.3 CONTROLADOR PID

El controlador PID (Proporcional-Integral-Derivativo) es una de las herramientas más comunes y eficientes, que se usa en control automático, debido a su rápida y efectiva respuesta correctora para ajustar un proceso.

El principio de un PID básicamente es reducir el error que existe entre el valor medido y el valor que se desea obtener, basándose en la calibración de tres parámetros o componentes.

El primer parámetro es la parte proporcional (P) que se encarga de corregir el error actual, es decir, corrige perturbaciones transitorias, pero no corrige perturbaciones permanentes, lo cual conduce a tener un error permanente. Para corregir el problema del error en estado estable está el segundo parámetro, la parte integral (I) que genera una corrección proporcional a la integral del error, esto asegura que aplicando un esfuerzo de control suficiente, el error de seguimiento se reduce a cero. Para mejorar el funcionamiento del controlador se agrega la parte derivativa (D), que funciona como una predicción de los errores futuros, que ayuda a disminuir el máximo sobre pico [43]. La respuesta de este controlador puede ser descrita como la respuesta de control ante el error.
La ecuación diferencial que representa a un PID en función del tiempo, está dada por:

$$u(t) = K_p e(t) + K_i \int_o^t e(t)d\tau + K_d \frac{d}{dt} e(t) \qquad (3.15)$$

Siendo:
$u(t)$ Respuesta del controlador ante el error
$e(t)$ Error=valor medido-valor requerido
K_p Ganancia proporcional
K_i Ganancia Integral
K_d Ganancia derivativa

Para poder diseñar e implementar un controlador PID es muy importante conocer el tipo de planta o proceso se desea controlar, para según ello poder calcular las constantes o ganancias del controlador. A fin de tener una mejor interpretación e implementación del control, además de cálculos más simples, es preferible representar el controlador como función de transferencia aplicando la transformada de Laplace. Con lo que se obtiene la siguiente ecuación:

$$G(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s} \tag{3.16}$$

3.4.3.1 Implementación Discreta

Para controlar al AR.Drone mediante el firmware de ROS, es necesario discretizar el PID, a fin de conseguir un seudocódigo que sea capaz de ejecutar las funciones del controlador. La forma de discretizar la parte integral es mediante la aproximación trapezoidal y para la derivativa, la aproximación por diferencias infinitas. Ambos términos se los discretiza teniendo en cuenta un tiempo de muestreo Δt en segundos; se tiene entonces [43]:

Término integral:

$$K_i \cdot \int_o^{t_k} e(t).d\tau \; = K_i \sum_0^k e(t_i).\Delta t \tag{3.17}$$

Término derivativo:

$$Kd \cdot \frac{d}{dt} e(t_k) = \frac{e(t_k) - e(t_{k-1})}{\Delta t} \cdot Kd \tag{3.18}$$

3.4.3.2 Efecto Wind-Up Integral

Es muy común en el diseño de un controlador PID que el sistema pueda llegar a ser inestable u oscilante debido a la acumulación descontrolada de su parte integral llamada *EFECTO WIND - UP* o saturación integral, que es la suma consecutiva del error en el tiempo de muestreo.

Esto puede causar largos períodos de sobresalto en la respuesta del sistema, es por ello que internamente en el controlador PID se dan límites máximo superior y máximo inferior respectivamente para controlar la acumulación de la parte integral, además es muy importante que dicha componente se reinicie siempre que el objeto se encuentre dentro de la zona de histéresis. [41]

3.4.3.3 Límites máximos a la salida del PID

Como se había mencionado antes en el apartado de conversión de Posición – Velocidad, el AR.Drone puede ser manejado por comandos que tengan valores entre -1 a 1 mediante el nodo de *ardrone_driver*, siendo éstos los valores máximos que el PID puede enviar. Para esto se ha considerado un limitador a la salida del controlador que mantiene los comandos de control dentro del rango establecido.

3.4.3.4 Seudocódigo

De acuerdo a lo explicado en los puntos anteriores, el seudocódigo utilizado en la implementación del control del cuadricóptero para cumplir con los objetivos propuestos es el siguiente:

```
//proporcional
  pTerm=pGain*error;
//integral
  Integral = Integral + (error*dt);
  iTerm=iGain*Integral;
  if (iTerm>iMax){iTerm=iMax;}
 if (iTerm<-iMax){iTerm=-iMax;}
//derivativo
derivative = (error - previous_error)/dt;
dTerm=dGain*derivative;

//salida
  output = pTerm+iTerm+dTerm;
  if (output>pid_max){output=pid_max;}
  if (output<-pid_max){output=-pid_max;}
  previous_error = error;
```

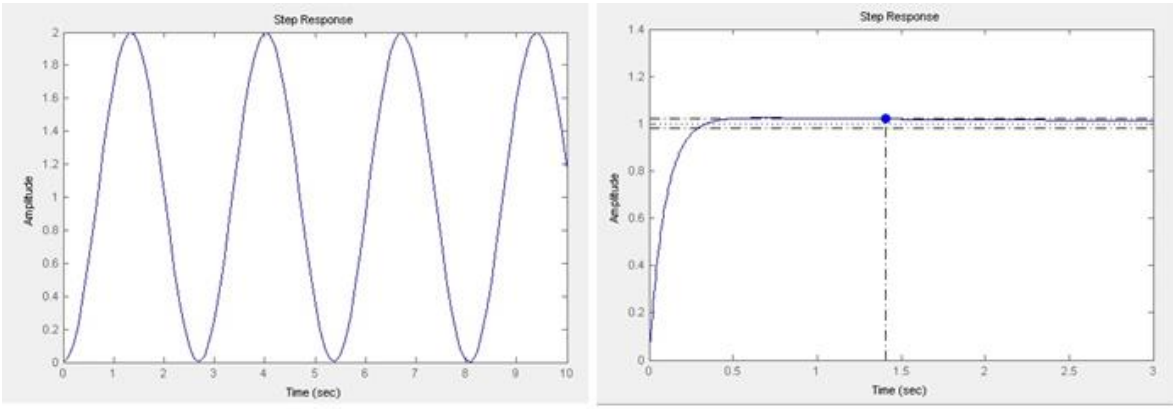
3.4.4 CONTROLADORES PID PARA CADA EJE DEL CUADRICÓPTERO

Una vez obtenidas las funciones de transferencia de cada eje en el que se va desplazar el cuadricóptero: cabeceo, altitud y ladeo respectivamente, se procede diseñar teóricamente los controladores PID adecuados para cada uno usando las herramientas que ofrece el software de MATLAB® [44]. Para encontrar las constantes de cada controlador en el simulador, se hizo primero una serie de cálculos matemáticos y posteriormente se calibraron de forma manual, hasta conseguir un sistema estable, sin que los valores de las constantes sobrepasen los límites máximos permitidos.

A continuación se muestran las gráficas de las respuestas a una función paso simuladas en MATLAB, tanto para la función de transferencia original, como para dicha función pero con el controlador ya implementado; para cada uno de los ejes que se controla.

Ladeo

$$G(s) = \frac{l}{I_Y.S^2} \qquad C(s) = \frac{2S^2 + 0.62S + 0.006}{S} \tag{3.19}$$

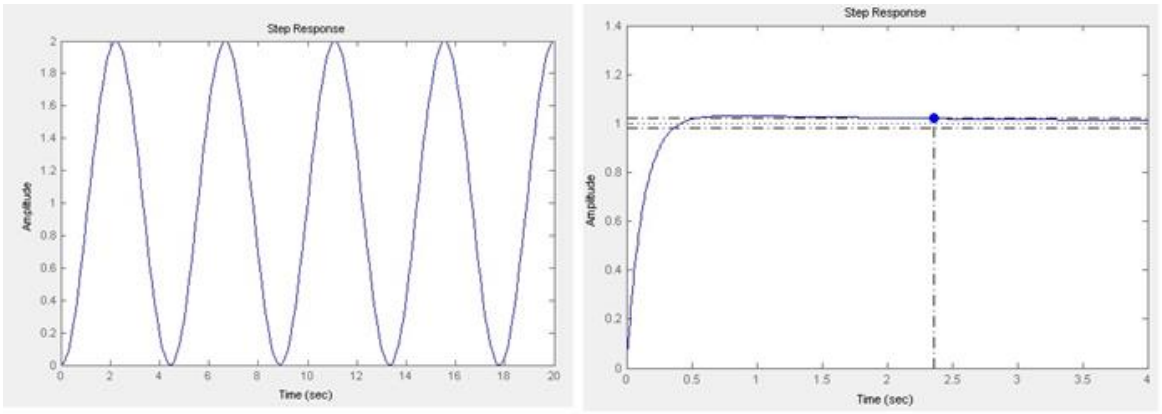


a) Sistema Original b) Sistema Compensado por PID

Figura.3.21 Respuesta para el eje de ladeo

Altitud

$$G = \frac{1}{m.s^2} \qquad C(s) = \frac{3.5S^2 + 1.085S + 0.00105}{S} \tag{3.20}$$

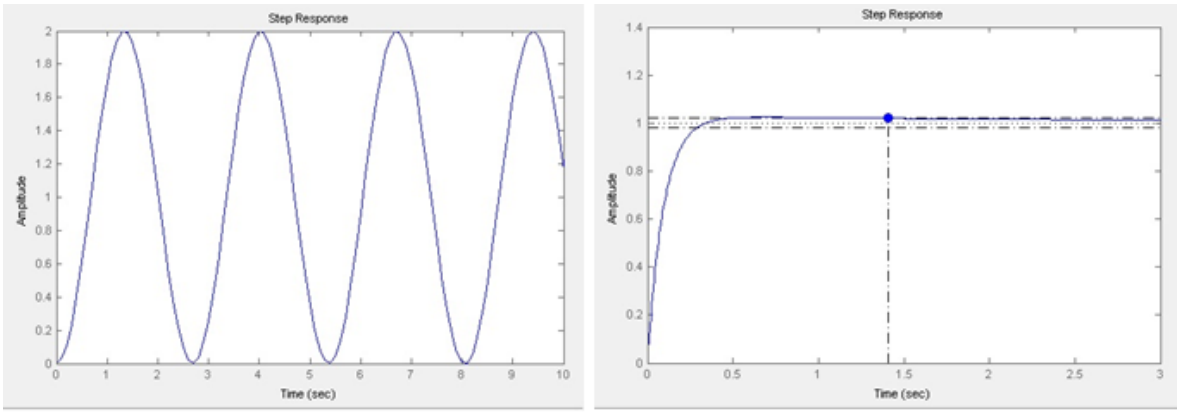


a) Sistema Original b) Sistema Compensado por PID

Figura.3.22 Respuesta para el eje de Altura

Cabeceo

$$G(s) = \frac{l}{I_x.S^2} \qquad C(s) = \frac{2S^2 + 0.62S + 0.006}{S} \tag{3.21}$$



a) Sistema Original b) Sistema Compensado por PID

Figura.3.23 Respuesta para el eje de Cabeceo

Es importante tomar en cuenta que los valores de los momentos de inercia son tomados de [45]; y el valor de la masa se toma de las especificaciones técnicas que provee el fabricante del AR.Drone.

3.5 IMPLEMENTACIÓN DEL SEGUIMIENTO

3.5.1 SEGUIMIENTO POR COLOR

Como se vio en el capítulo anterior el resultado del procesamiento de imagen es tener una captura o un cuadro totalmente monocromático, en blanco y negro, que visualmente representa si se ha encontrado el color del objeto que se busca o no. Por ejemplo, se presenta a continuación el resultado del pre-procesamiento en la búsqueda de un globo rojo.

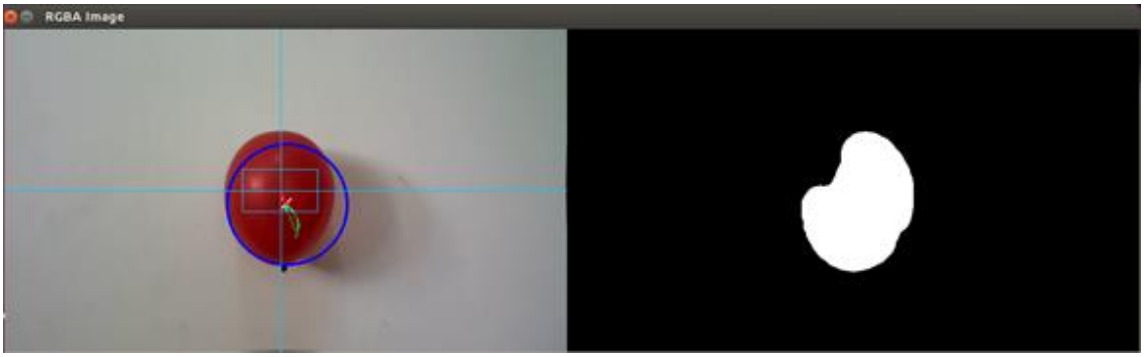


Figura.3.24 Resultado del procesamiento de imagen por Color

Seguidamente, en base a este resultado se puede calcular fácilmente los datos de interés para el seguimiento, como son área (A) y ubicación del centro del objeto P(y, z) como se muestra en la siguiente figura:

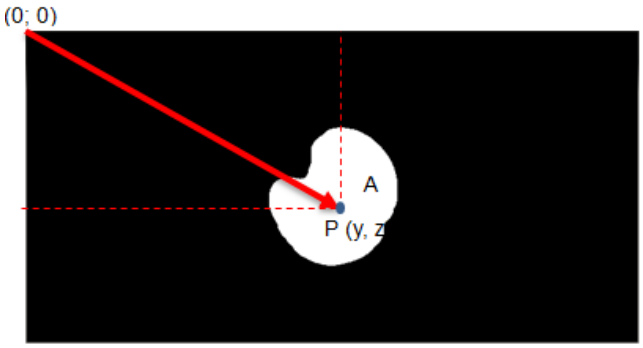
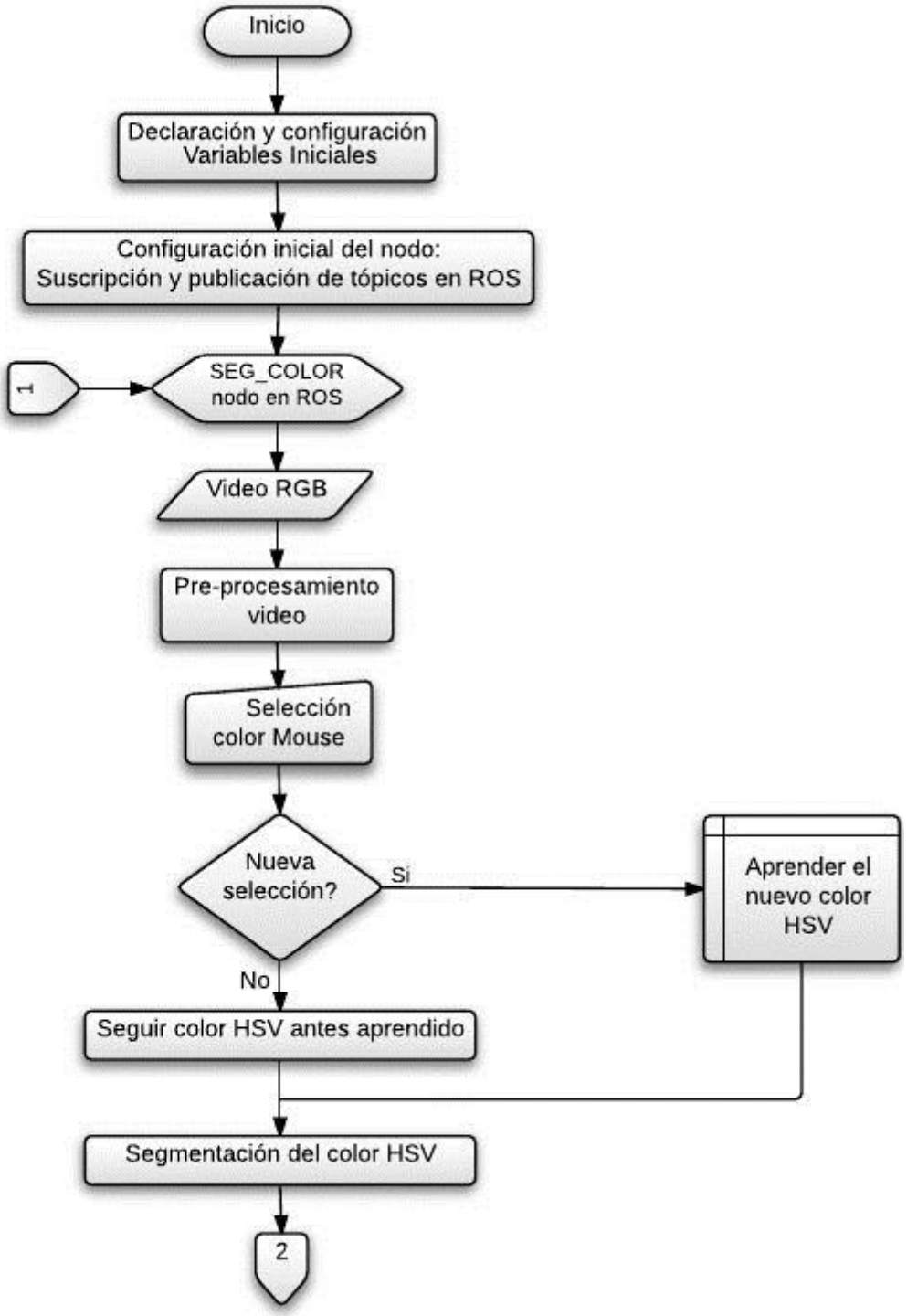


Figura.3.25 Determinación de la ubicación y el área

Una vez determinada la pose y el área del objeto se procede a generar y enviar los comandos respectivos por cada eje, para realizar el control de seguimiento según lo descrito al inicio de este capítulo.

A continuación se presenta el diagrama de flujo, en el cual se describe y explica de manera completa la manera de obtener y procesar la información para comandar el cuadricóptero en el reconocimiento y seguimiento de objetos.



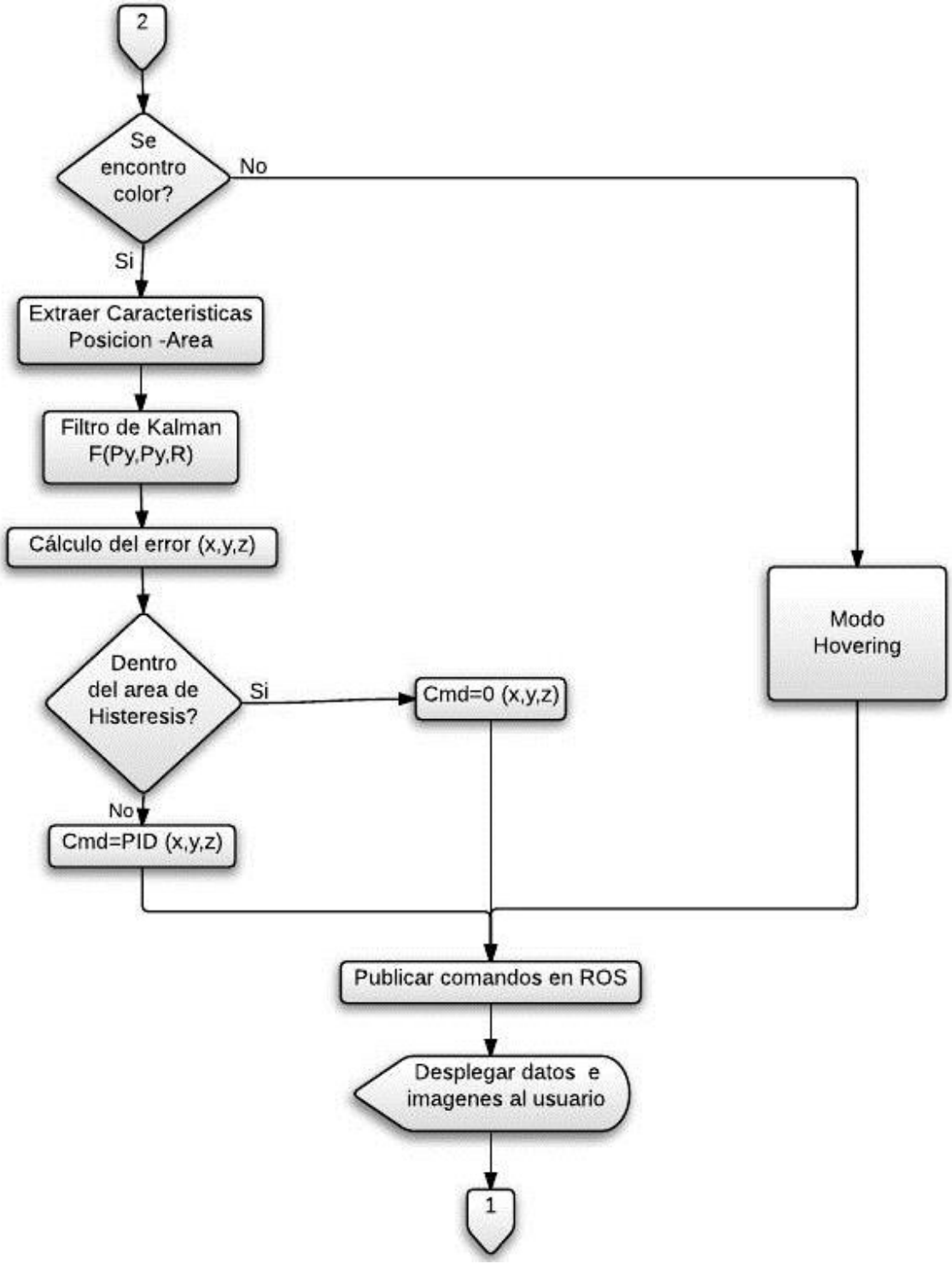


Figura.3.26 Diagrama de Flujo Procesamiento por Color

3.5.2 SEGUIMIENTO POR FORMA

Luego del procesamiento de imagen en búsqueda de formas, el resultado es una imagen monocromática en blanco y negro, la cual representa en color blanco los

cambios significativos de intensidad resumidos en los bordes de las figuras encontradas y en color negro las áreas que tienen una intensidad constante.

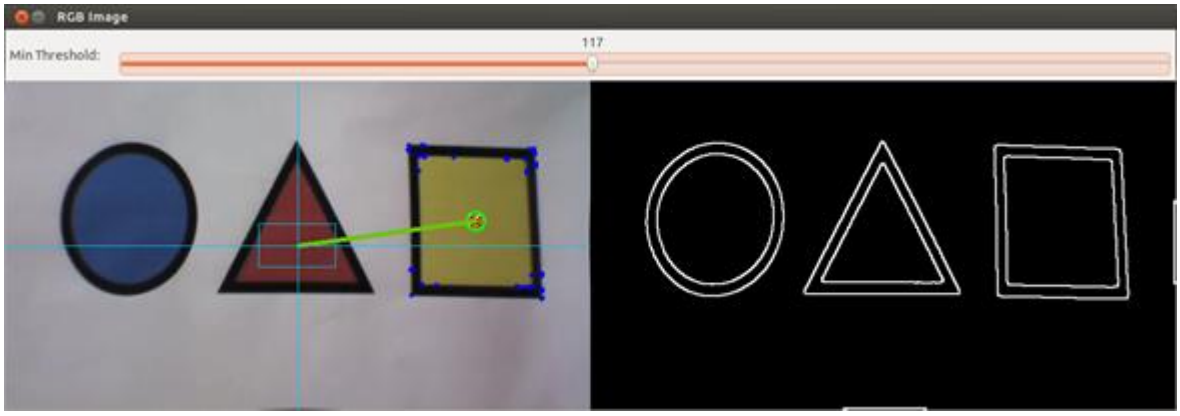


Figura.3.27 Resultado del procesamiento de imagen por Forma

Consecutivamente se busca todos los contornos dentro de la imagen y se los aproxima a la forma básica de un polígono (triangulo, rectángulo, cuadrado, circulo) según el número de lados encontrados. Para que el contorno encontrado sea determinado como una aproximación a una forma básica, este debe cumplir requisitos como:

- El contorno se debe cerrar completamente
- Contar con el número de lados correspondientes a la figura buscada.

Finalmente se calcula el área A que cubre el contorno y la ubicación del centro del objeto P (y, z)

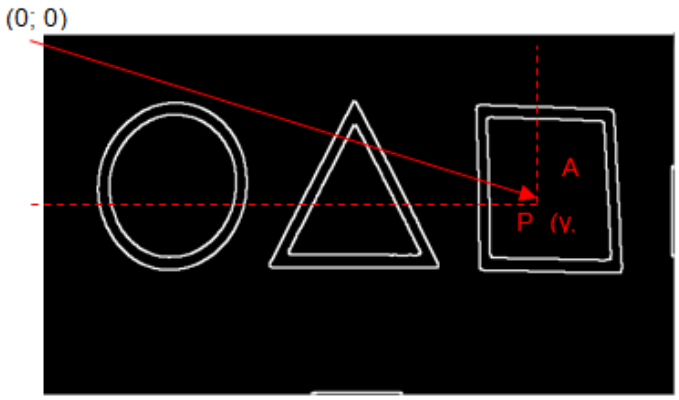
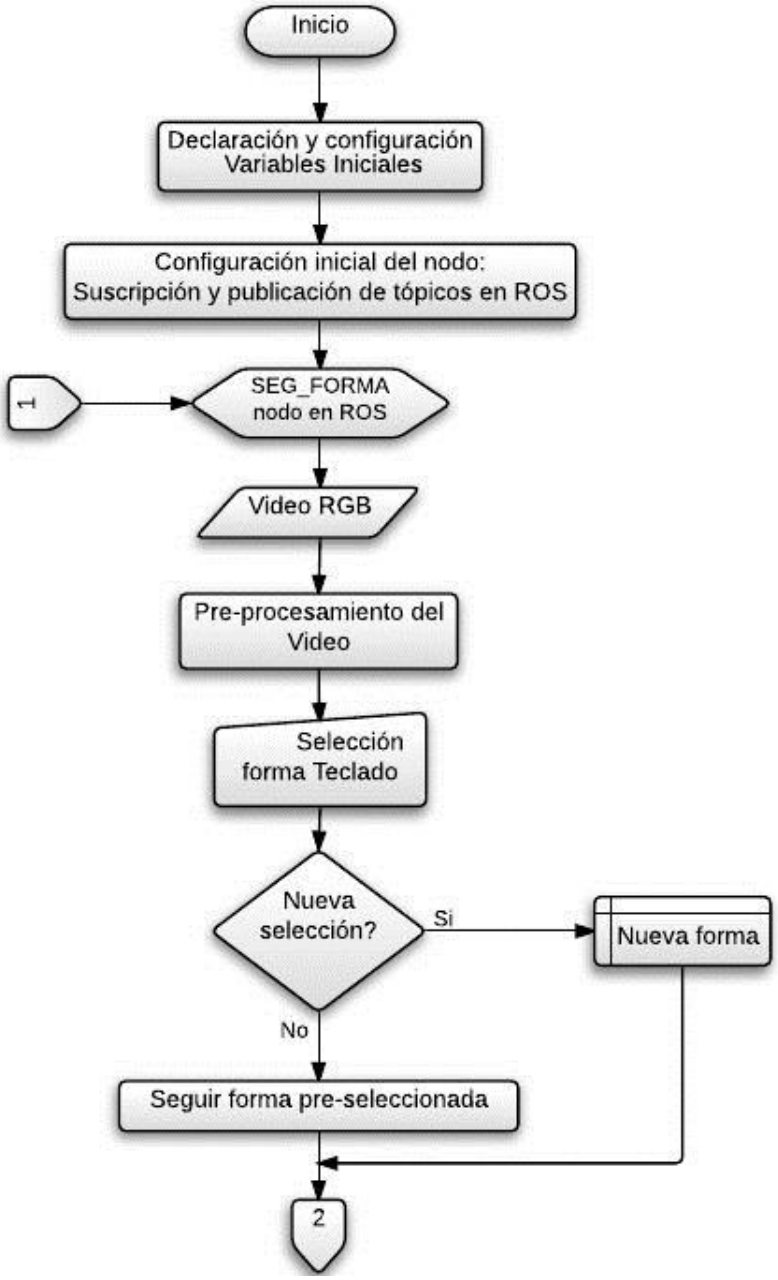


Figura.3.28 Determinación de la ubicación y área del contorno

Una vez determinada la pose y el área del objeto se procede a utilizar el algoritmo de *flujo óptico* para seguir las características más relevantes de la forma permitiendo un seguimiento continuo como se había explicado en el capítulo anterior. Inmediatamente se generan y se envían los comandos respectivos por cada eje para el proceso de seguimiento.

A continuación se presenta el diagrama de flujo, en el cual se describe y explica de manera completa la manera de obtener y procesar la información para comandar el cuadricóptero en el reconocimiento y seguimiento por forma.



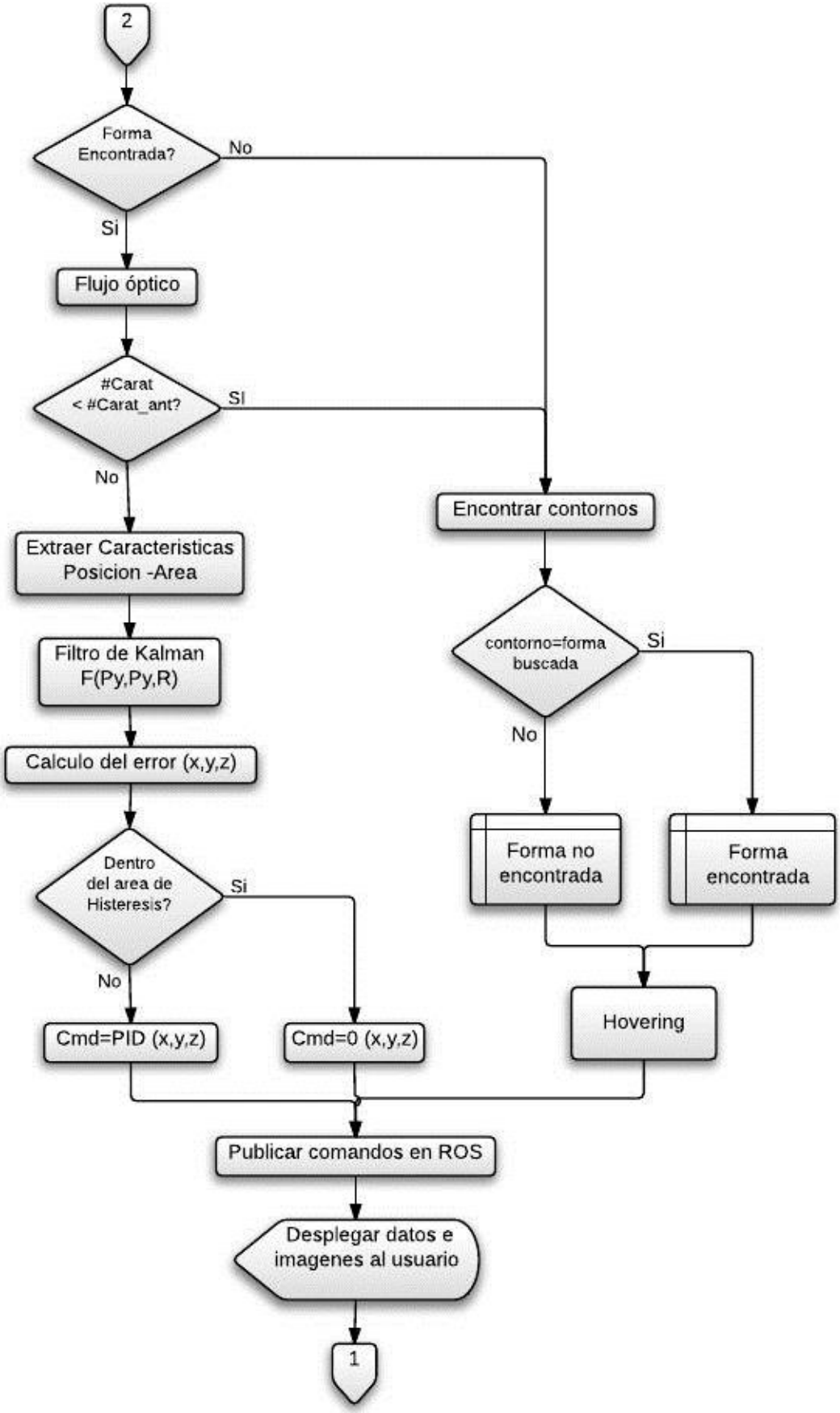


Figura.3.29 Diagrama de Flujo Procesamiento por Forma

CAPÍTULO 4

PRUEBAS Y RESULTADOS

4.1 INTRODUCCIÓN

A lo largo de este capítulo se van a presentar los diferentes resultados experimentales de las técnicas y algoritmos que se describieron en los capítulos anteriores. El conjunto de pruebas se enfocará principalmente en determinar la efectividad y la precisión del software desarrollado, además de comprobar si existe movimiento estable y puntual; también se hacen pruebas adicionales para determinar la capacidad de los algoritmos para recibir la información de la cámara del AR.Drone, encontrar los objetivos e iniciar el seguimiento.

Los datos que proporcionan estas pruebas se utilizan para extraer las diferentes conclusiones y proponer las respectivas soluciones y/o recomendaciones para los problemas que pudieran presentarse en el proyecto. Por otro lado, es importante mencionar que las pruebas se hicieron únicamente considerando dos grados de libertad, es decir el cuadricóptero hará el seguimiento del objetivo solo en el plano.

4.2 CONDICIONES DE AMBIENTE SEMIESTRUCTURADO

4.2.1 AMBIENTE SEMIESTRUCTURADO

En capítulos anteriores se mencionó ciertos aspectos que podrían eventualmente influir de forma negativa en el desarrollo de esta aplicación, situaciones como problemas de iluminación, presencia de sombras, demasiada luz en el entorno, ambientes complejos, muchos objetos en la escena, entre otros. Problemas que dificultan especialmente el reconocimiento del objetivo y en consecuencia impiden un buen desempeño de la parte de seguimiento.

Con las pruebas que se describen en este capítulo se busca determinar las condiciones mínimas para un desarrollo óptimo de la aplicación implementada en este proyecto.

4.2.2 RECONOCIMIENTO POR COLOR

El reconocimiento de objetos según sus características de color, es un método que tiene que combatir los problemas que causan los cambios en una escena que podrían deberse al movimiento de la cámara, al movimiento de los objetos, a cambios en la iluminación, o a cambios en la estructura, forma o tamaño de un objeto. Además en un ambiente con la presencia de muchos objetos moviéndose independientemente en una escena, resulta totalmente problemático el hecho de reconocer un solo objetivo sin que experimente confusiones con objetos de similares características, por lo que sería necesario el uso de restricciones basadas en la naturaleza de los mismos objetos y de su movimiento.

La complejidad de los algoritmos obedece a que éstos deben detectar cambios, determinar las características del movimiento del observador y los objetos, caracterizar los movimientos, recuperar la estructura de los objetos y reconocer a los objetos en movimiento.

En consecuencia las pruebas que se llevaron a cabo en este punto, sirvieron para determinar las características necesarias del ambiente en el que se correrá esta aplicación, de manera que no se presenten problemas como perder de foco al objetivo, o confusiones de éste con el entorno. Los resultados encontrados se presentan en las siguientes imágenes:

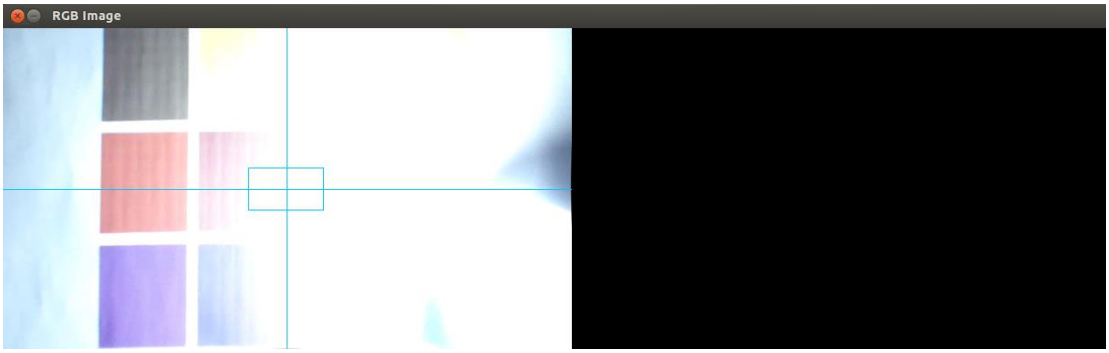


Figura.4.1 Destello de Luz

En la figura.4.1 se observa que ante el exceso de iluminación, el pre procesamiento de la imagen no entrega ningún resultado, es decir no se logra

identificar el objetivo puesto que la mayoría de los colores tienden a un tono de blanco. Algo parecido ocurre cuando por el contrario, no hay iluminación suficiente; los objetos en general tendrán una tonalidad que tiende al gris y habrá confusiones entre los cuerpos presentes en la escena (Figura.4.2).

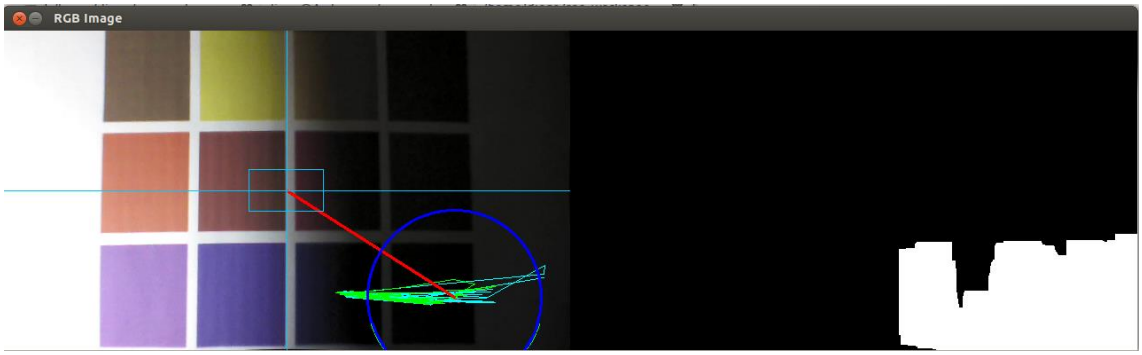


Figura.4.2 Poca iluminación

En el siguiente caso, el programa entrega un mensaje de error señalando que son demasiados elementos con las mismas características de color, y que en consecuencia no se puede hacer el reconocimiento de un objetivo específico como se ve en la figura 4.3.

Figura.4.3 Mensaje de error_demasiados objetos

Otro problema que se encontró realizando las pruebas, es que si la iluminación de la habitación no es uniforme, se producen sombras o bien dependiendo del objeto, éste puede tener más o menos brillo en unas zonas de su cuerpo provocando que el reconocimiento no sea preciso como se ve en los siguientes cuadros:

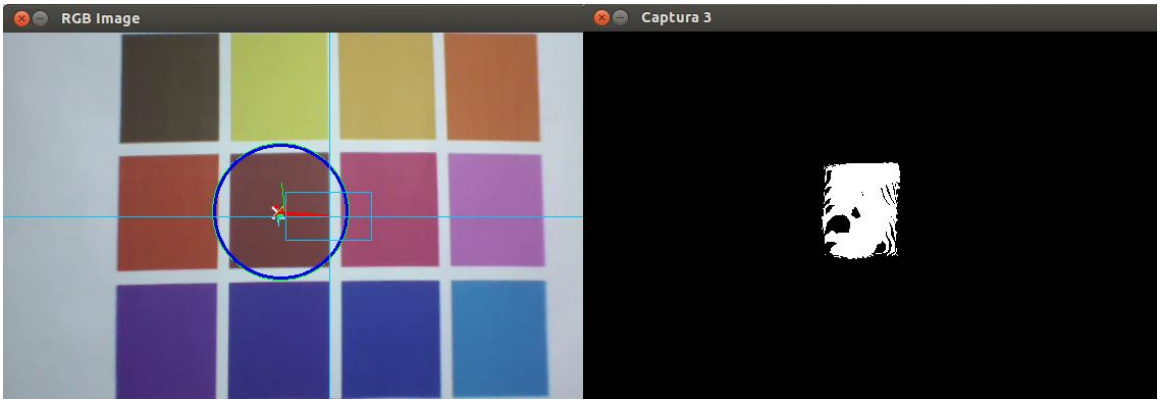


Figura.4.4 Iluminación no uniforme

El hecho de tener sombras o zonas con mayor brillo provoca que el objetivo encontrado no sea exactamente igual al objeto que se capta con la cámara de video, como por ejemplo:

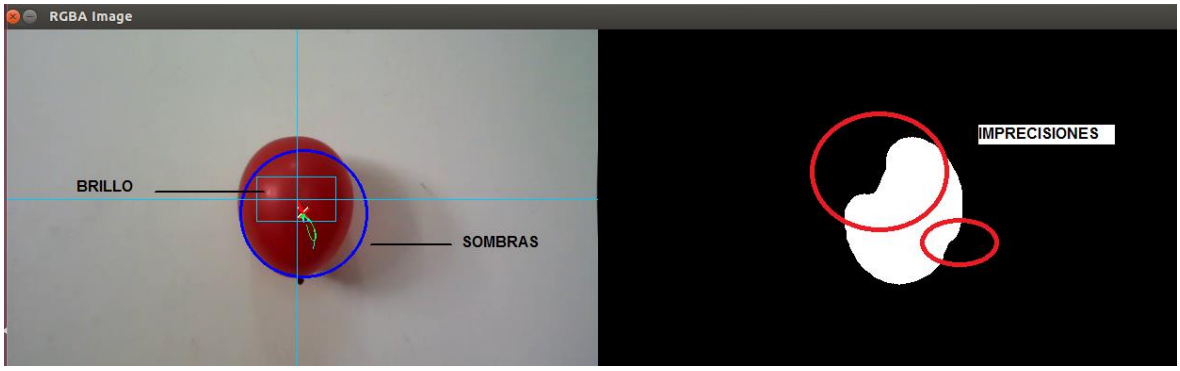


Figura.4.5 Sombras y Brillo

Otra situación podría darse si es que el entorno en donde se ejecute la aplicación tiene muchos objetos de similares características de color, puesto que si bien el algoritmo usado tiene un rango de diferenciación de colores que eventualmente le da cierto grado de robustez, podría pasar que por cuestiones de iluminación o hechos fortuitos dos o más cuerpos se parezcan en color, provocando confusión y afectando el resultado del procesamiento.

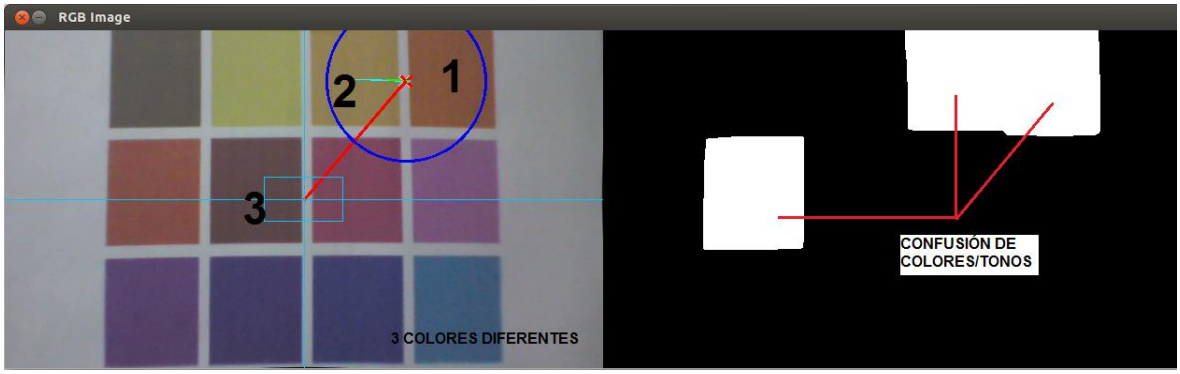


Figura.4.6 Confusión de tonos

Esta situación resulta más evidente si se considera a la imagen en espacio de color HSV que maneja matices (Figura.4.7)

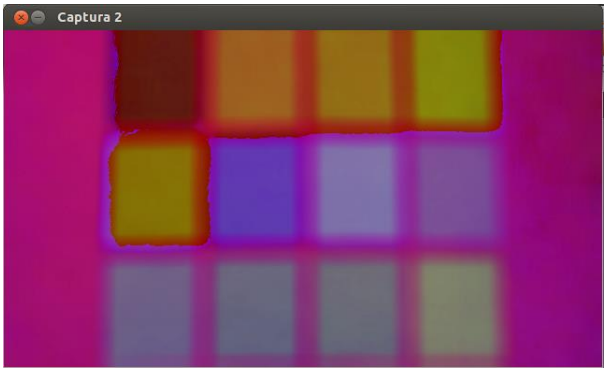


Figura.4.7 Espacio HSV, error

4.2.3 RECONOCIMIENTO POR FORMA

El principal inconveniente que se encuentra para hacer un reconocimiento según la forma básica de un objeto, es que en un entorno con muchas cosas es inevitable que se encuentre más de un objeto con la forma especificada. En este caso, es importante que se tome en cuenta que la aplicación desarrollada en este proyecto se ponga en marcha en una habitación o en general en un lugar en donde no haya demasiadas cosas que pueden parecerse al objetivo.

Por ejemplo, en la siguiente imagen, se tiene como entorno un estudio y lo que se desea hallar para seguimiento es un *cuadrado*, y sucedió que se encontraron muchas cosas con forma similar a un cuadrado:

Figura.4.8 Entorno complejo_formas semejantes

En “escenas” de este tipo, como la de la captura anterior siempre se presentan problemas semejantes independientemente de la forma geométrica que se desee encontrar.

Para cuestiones de reconocimiento de imagen, por los dos métodos que se manejan en este proyecto, estos son los inconvenientes que se encontraron que pueden influir en la obtención de un buen resultado. La idea es entonces, controlar cada una de estas situaciones a fin de evitar que se presenten durante la ejecución de la aplicación de reconocimiento y seguimiento de objetos mediante características de color y forma desarrollada en este trabajo.

4.3 SEGUIMIENTO O TRACKING

En el instante en el que se comienza con el tracking o seguimiento, se pusieron en marcha, otro conjunto de pruebas para comprobar que el movimiento que realiza el AR.Drone sea aproximadamente igual al del objetivo, considerando que el cuadricóptero tiene para este proyecto dos grados de libertad, es decir que su movimiento se limita al plano, como se explica en párrafos anteriores.

En cuanto a seguimiento, como se ve en el *CAPÍTULO 3,* se maneja ciertos filtros y controladores para estabilizar el vuelo del AR.Drone como son el Filtro de Kalman y un Controlador PID, cuyo funcionamiento y respuesta se analiza en los siguientes párrafos.

4.3.1 FILTRO DE KALMAN

El filtro de Kalman constituye una herramienta para la corrección de la trayectoria que debe seguir el AR.Drone, según el movimiento del objetivo identificado. Para interpretar los resultados de las pruebas realizadas se ha considerado que el modelo de la planta obtenido es solo una aproximación del modelo real, por tres motivos:

- No existe un modelo matemático perfecto de un sistema real.
- Existen perturbaciones que no se pueden modelar de una forma determinística.
- Los sensores no son perfectos.

Las siguientes gráficas se tomaron con la ayuda de la herramienta de ROS, *rqtplot()*, que ofrece un plugin para visualizar valores numéricos en un gráfico 2D usando diferentes tipos de trazados. Rqt_plot muestra la gráfica en función de tiempo de los datos publicados en los diferentes temas.

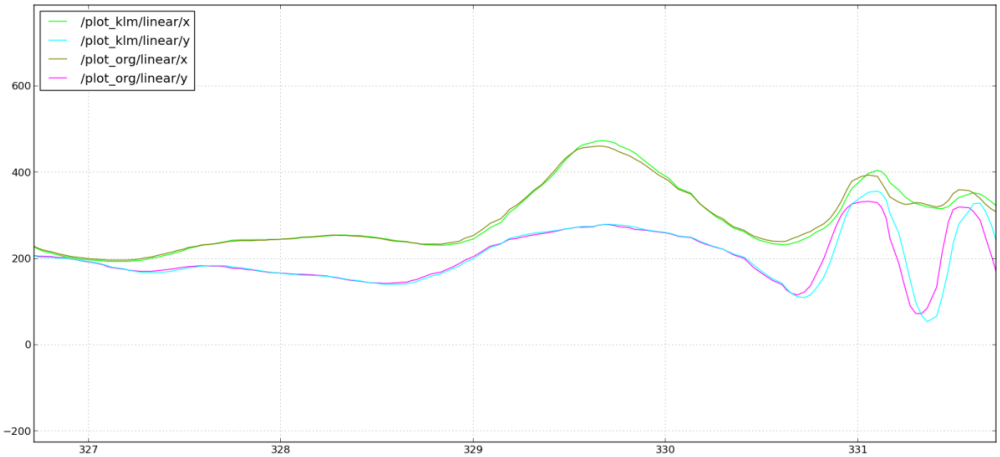


Figura.4.9 Respuesta filtro de Kalman

En la figura 4.9, las gráficas en color café y rosa, muestran la información del movimiento del AR.Drone conforme el objetivo de seguimiento también se mueve en sus dos ejes de libertad (de arriba hacia abajo y de izquierda a derecha). Estos datos son los que entran en el filtro de Kalman para ser procesados; es decir estos datos son las condiciones iniciales para realizar el proceso de predicción y

de corrección del movimiento.Entonces, las curvas en verde y celeste representan la salida de este filtro; como se observa el movimiento es más suave, se puede decir que se trata de hacer que los cambios de posición sean menos bruscos, dando como resultado un movimiento más controlado y preciso.

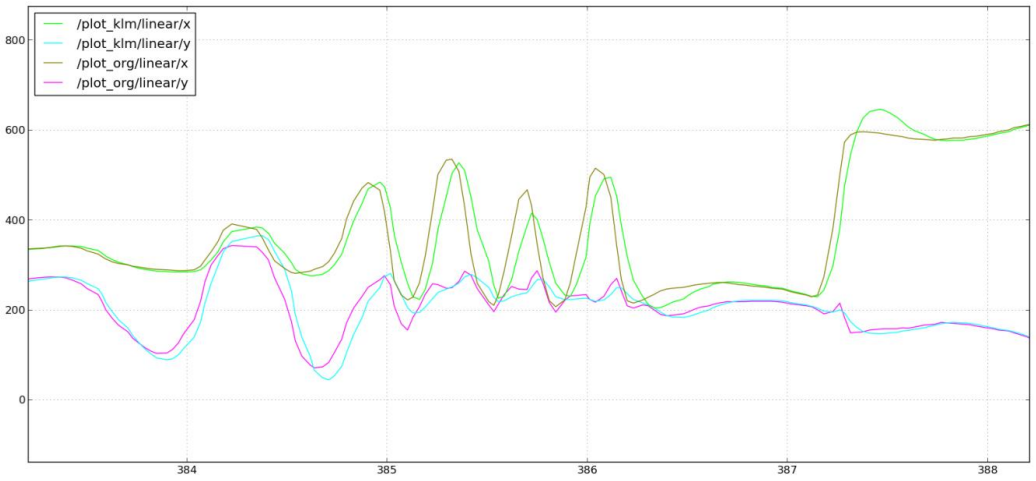


Figura.4.10 Filtro de Kalman_cambios bruscos

En la figura 4.10, se observa claramente el trabajo del filtro de Kalman, puesto que en las gráficas del movimiento sin filtro se ven cambios bruscos en la trayectoria, lo que se aprecia en el AR.Drone como “movimientos torpes” al tratar de seguir a su objetivo. Lo que hace en este caso el filtro es tratar de mantener una trayectoria suave; es decir una trayectoria en donde los cambios de posición se den de manera progresiva dando la sensación de tener mayor estabilidad.

Tomando en cuenta que los tiempos de comunicación son relativamente rápidos, es necesario tener un control más robusto de los movimientos que realice el cuadricóptero, porque al cambiar de posición bruscamente podría desestabilizarse y perder de vista al cuerpo que está siguiendo.

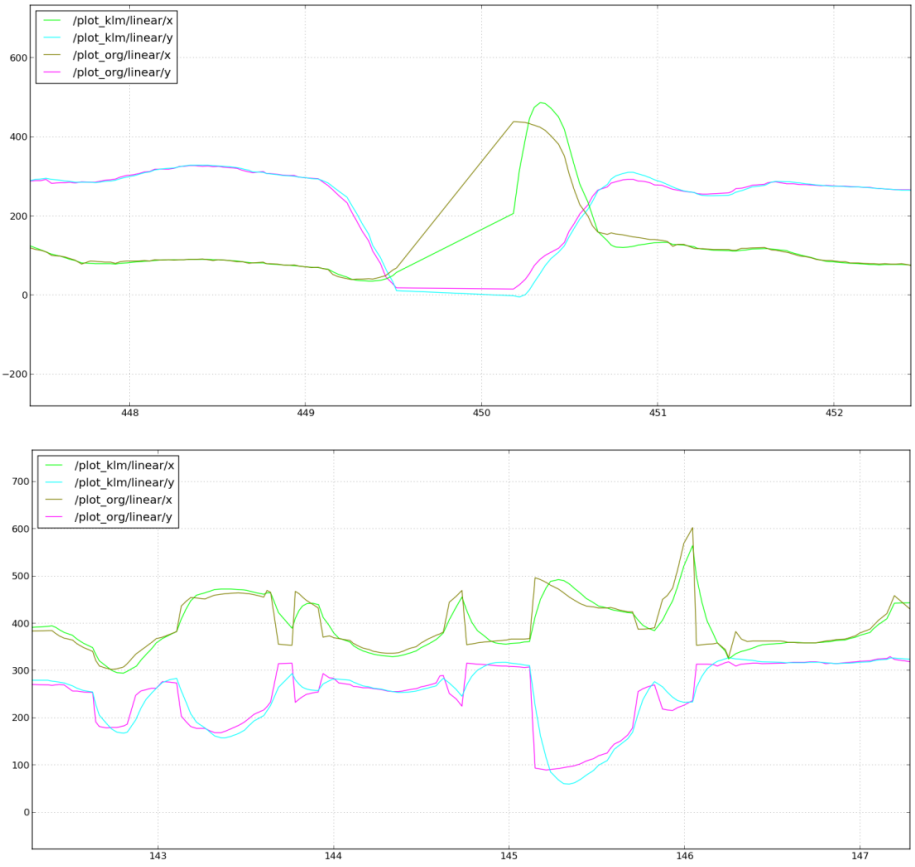


Figura.4.11 Control de movimientos bruscos

En el gráfico 4.11 puede entenderse que a pesar de haber movimientos rápidos del objetivo, el filtro hace que el cuadricóptero sin perder de vista a dicho objetivo haga un movimiento menos “violento” de forma que pueda mantenerse en una posición segura y evite perder el control. Con las pruebas que se hicieron, se comprobó qué tan bien se corrigen las trayectorias y se ajustó el filtro para suavizar los movimientos y que no haya variaciones de posición violentas.

4.3.2 CONTROLADOR PID

En general se sabe que un PID es un control que calcula el error entre un valor medio y el que se quiere obtener, para aplicar una acción correctora que ajuste el proceso.

La respuesta del controlador implementado se puede describir en términos del control ante un error, el grado en el que se llega al “set point”, y el grado de oscilación del sistema. Estos parámetros son los que desea comprobar con una

serie de pruebas, para verificar el comportamiento de la planta, en este caso del AR.Drone ante diferentes movimientos en distintas direcciones y a diferentes velocidades. Antes de interpretar los resultados de las pruebas, hay que entender que el uso de un PID no garantiza control óptimo del sistema o la estabilidad total del mismo.

Con la ayuda del MATLAB, se observó el tipo de comportamiento del sistema y se concluyó que es necesario hacer el diseño de un controlador PID, dadas las respuestas obtenidas para los dos tipos de movimiento que realiza el cuadricóptero en este proyecto, como se observa en las siguientes imágenes:

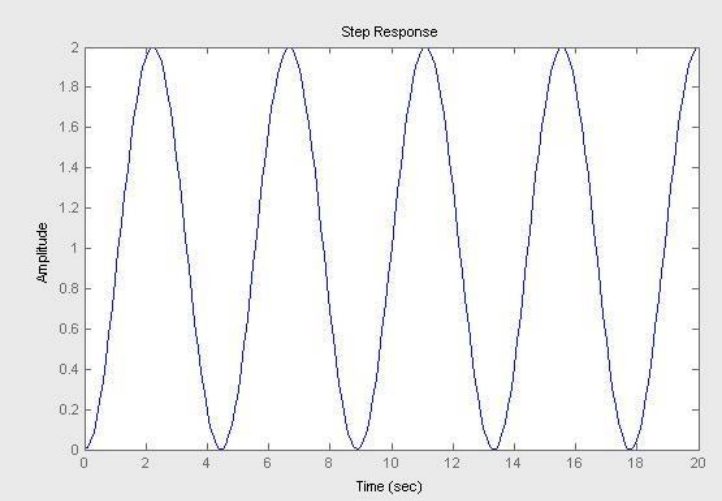


Figura.4.12 Respuesta sistema movimiento vertical

En la figura 4.12 se tiene la gráfica del comportamiento del sistema cuando se hace un movimiento vertical, de arriba hacia abajo, y se puede notar que se tiene un sistema oscilatorio por lo que resulta necesaria la implementación de un controlador.

En la figura 4.13 con una respuesta de similares características a la anterior, se tiene graficado el comportamiento del sistema ante una entrada paso, cuando se realiza el movimiento horizontal, de izquierda a derecha, concluyendo nuevamente que se necesita de un controlador.

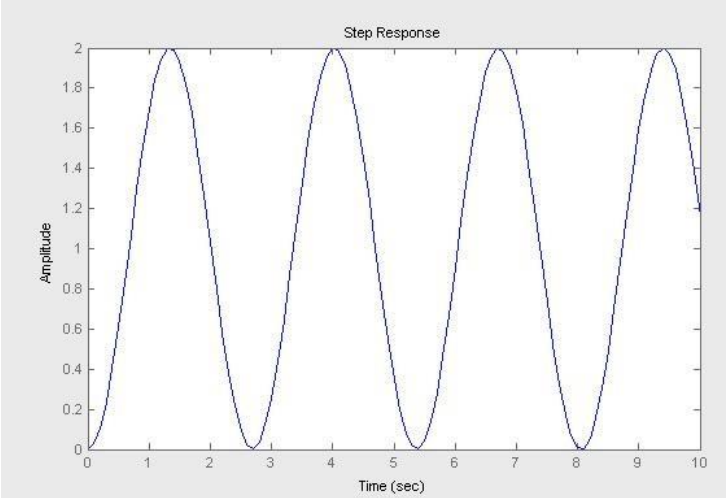


Figura.4.13 Respuesta sistema movimiento horizontal

De las imágenes anteriores, se nota que es importante el diseño de un controlador tipo PID que permita reducir el tiempo de subida, eliminar el error en estado estable, disminuir el tiempo de establecimiento y el sobre impulso, dando como resultado el incremento de la estabilidad del sistema mejorando la respuesta del mismo. El controlador PID que se implementa, busca también eliminar la sensibilidad a las variaciones de parámetros, darle robustez al sistema y que presente rechazo a perturbaciones. Matemáticamente hablando, en el diseño el tiempo de establecimiento debe ser menor a *4 segundos* y el sobre pico menor a un *20%,* especificaciones que garantizan una respuesta estable.

4.3.2.1 Resultados con PID diseñado

Este controlador se diseñó en base a condiciones teóricas, en las que se asume un valor aproximado de sobrepico y ts. Haciendo uso de la herramienta de ROS, rqtplot (), se tomaron gráficas de la precisión con la que el AR.Drone sigue a su objetivo, reconocido mediante el método de características de color o bien de forma. El controlador con el que se hicieron las pruebas mantiene los valores que se hallaron con cálculos matemáticos sin ninguna variación, para cada tipo de movimiento:

Movimiento Vertical:

$$C(s) = \frac{3.5S^2 + 1.085S + 0.00105}{S} \tag{3.20}$$

Movimiento Horizontal:

$$C(s) = \frac{2S^2 + 0.62S + 0.006}{S} \tag{3.19}$$

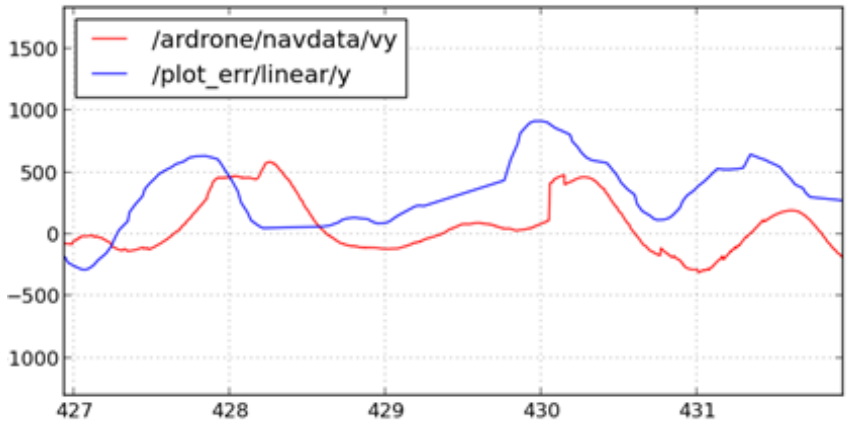


Figura.4.14 PID diseño

En la figura anterior, se presenta un movimiento relativamente suave, sin variaciones y pausado del cuerpo que se está siguiendo; como se ve la respuesta del cuadricóptero es no muy satisfactoria porque se tiene un error de precisión grande, y hay mucha inestabilidad que físicamente se observa como oscilaciones del AR.Drone antes de poder estabilizarse frente al objeto.

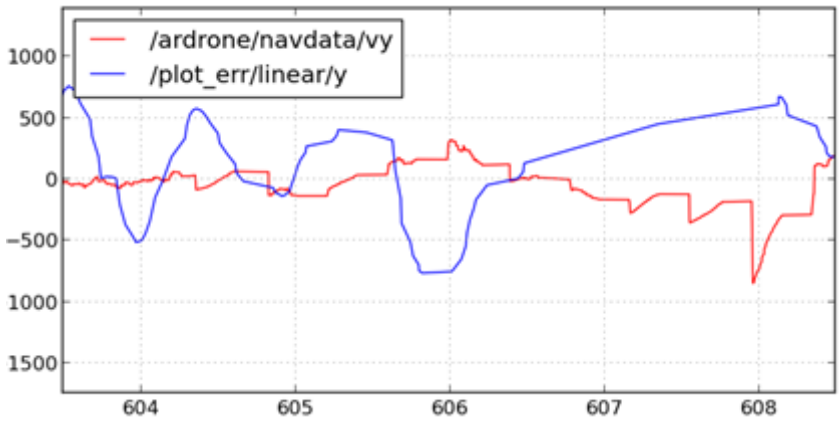


Figura.4.15 PID diseño movimiento rápido

En la figura 4.16 se tiene un movimiento mucho más rápido y agitado del objetivo, y como se aprecia en la gráfica el AR.Drone tiene una respuesta “accidentada”, alejada del “set point”, con un error grande y bastante inestabilidad.

El set point, está dado por la trayectoria que sigue el objetivo, es decir lo que se pretende es que el movimiento del cuadricóptero sea lo más semejante al movimiento del objetivo.

4.3.2.2 Corrección de errores y resultados

En primera instancia se puede creer que la imprecisión que se observa en las gráficas anteriores podría deberse a una mala calibración de las constantes del PID, pero antes de manipularlas es preciso analizar otras posibles situaciones que provoquen este problema.

Con el cuadricóptero en marcha se analiza el comportamiento de éste, para de forma práctica observar el grado de inestabilidad que tengan los movimientos del AR.Drone al seguir al objetivo.
La primera condición crítica en la implementación del seguimiento, está dada por el área de histéresis que se estableció anteriormente para fijar un espacio en el que se ejerza una acción correctora; ésta histéresis plantea que dentro de un área dada el controlador ya no realice ninguna acción sobre el sistema.

Teóricamente se aconseja que el área de histéresis sea de alrededor del 10% del total del campo visual de la cámara. Así que se comprobó si efectivamente se estaba manejando este 10%, obteniendo como resultado que el primer error que ocasionaba la inestabilidad del cuadricóptero es que se estaba trabajando con una histéresis de más del 15%.

Una vez corregido este problema se obtuvieron las siguientes gráficas:

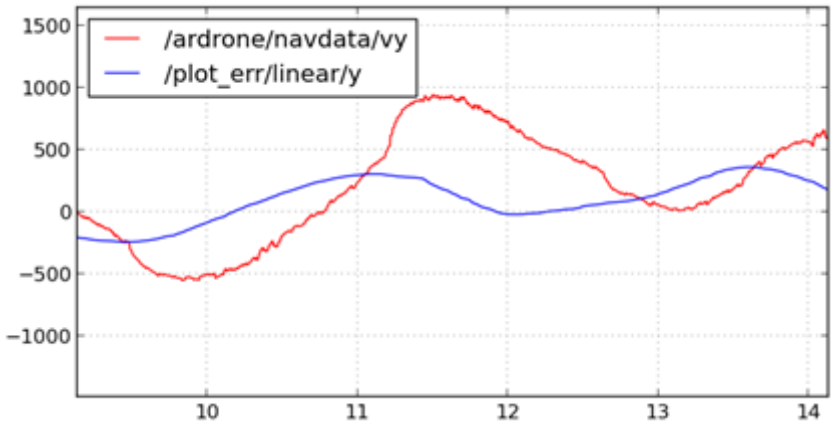


Figura.4.16 Área de histéresis de más del 15%

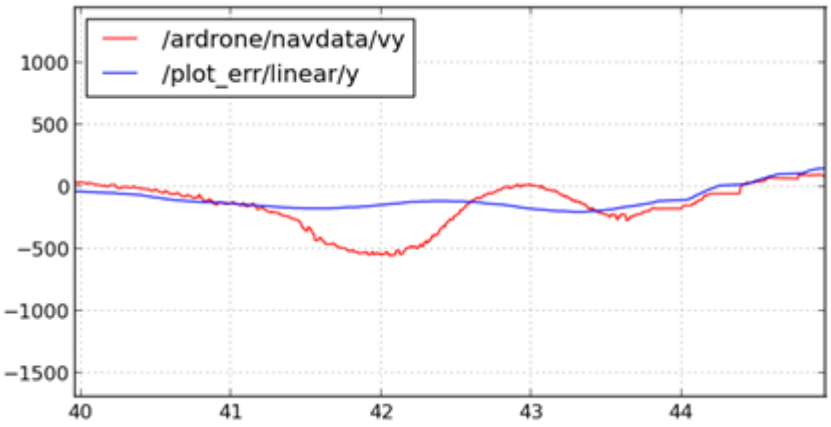


Figura.4.17 Área de histéresis del 10%

En la Figura.4.18 se aprecia que el error entre el set point y la señal tomada del AR.Drone es mucho menor que el observado en etapas anteriores, también se ve como el sistema trata de mantener un movimiento suave ajustándose al valor dado por el set point después de haber calibrado el valor del área de histéresis. Pero también aparece un nuevo inconveniente, se puede notar que existe un retraso entre ambas señales; esta situación podría ser consecuencia de los tiempos de comunicación entre el computador y el cuadricóptero o bien de los tiempos de muestreo que se manejan en el controlador. Aunque estos retrasos son de un valor muy pequeño aparentemente imperceptibles en la práctica, son un problema de imprecisión.

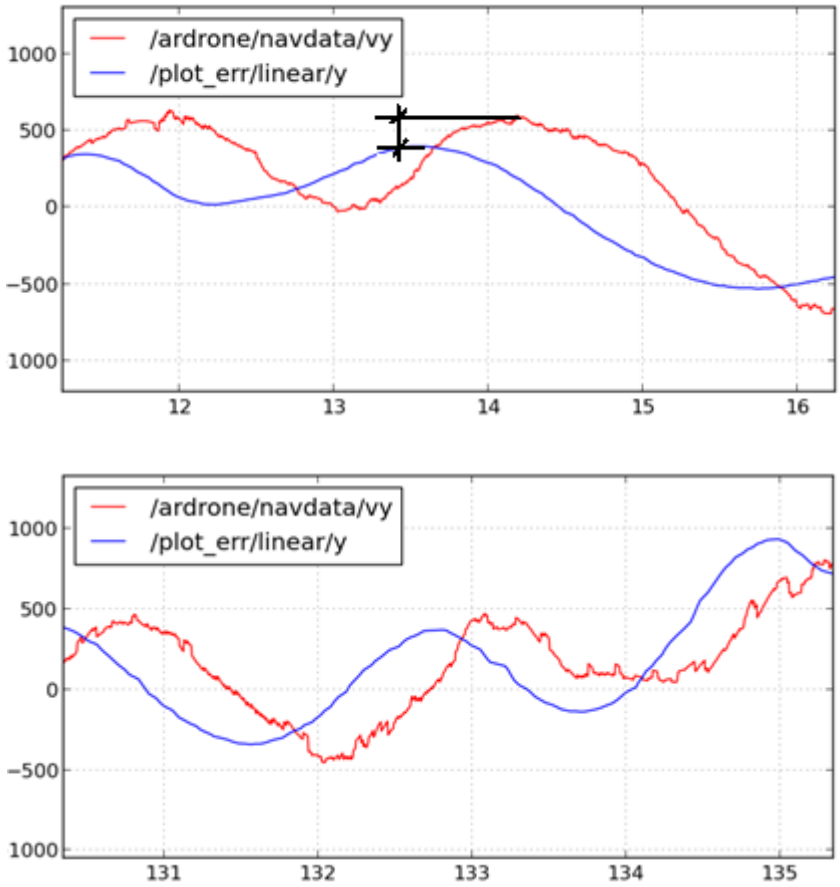


Figura.4.18 Error entre set point y señal real y Retraso

Los tiempos de muestreo tienen mucha influencia en la velocidad en la que responde el sistema, a pesar de que estén en el orden de los milisegundos. Analizando el programa se encontró que la aplicación de este proyecto trabajaba con un tiempo de muestreo de 20ms, un tiempo demasiado corto para que se avance a procesar las órdenes y ejecutarlas de manera óptima.

En teoría se recomienda emplear un tiempo de más o menos 100ms, así que se adoptaron estos valores y los resultados fueron realmente formidables como se observa en las figuras a continuación:

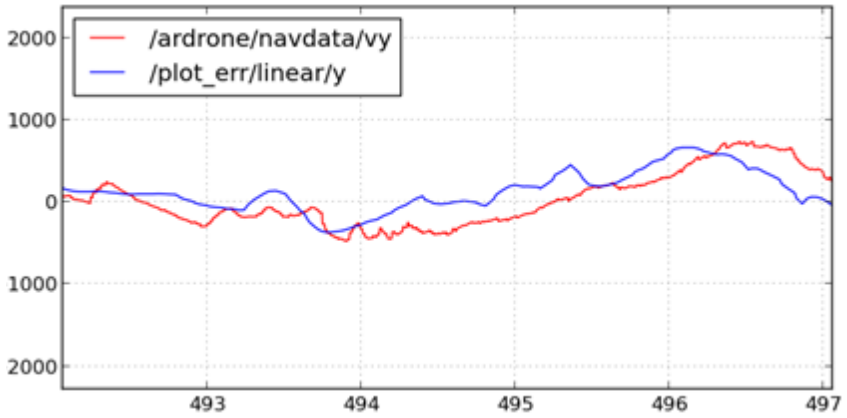


Figura.4.19 Corrección de los tiempos de muestreo

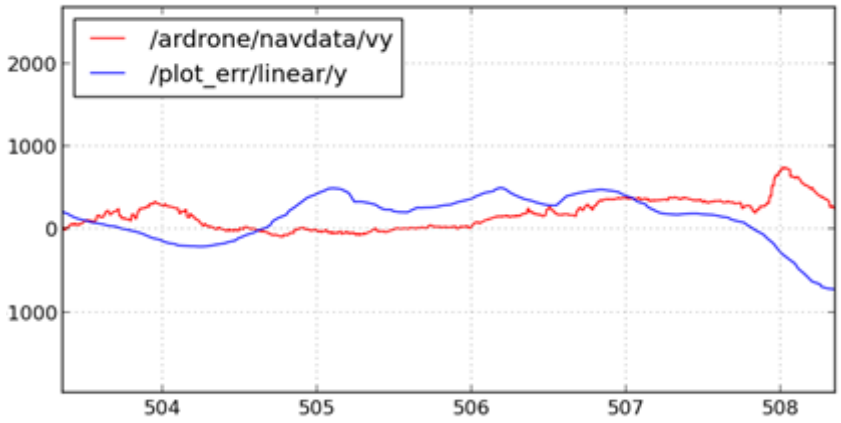


Figura.4.20 Eliminación de retraso en la señal

Los dos problemas hallados, eran los que provocaban directamente el error de precisión en lo que se refiere al proceso de seguimiento, en consecuencia no hubo necesidad de cambiar los valores de las constates de los PID y realizar ningún tipo de calibración en los mismos.

Una situación en particular, resulta del cambio de dirección en el movimiento del AR.Drone, en ese instante se produce una pequeña inestabilidad que no tiene mucha duración pero que se puede reconocer en las capturas del rqtplot ().

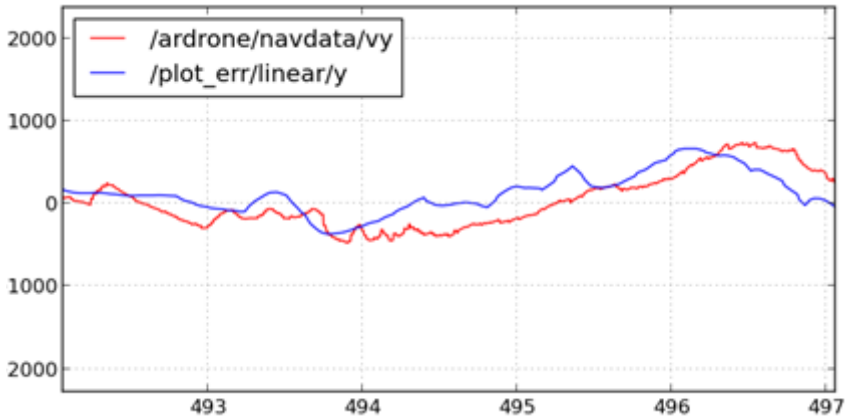


Figura.4.21 Cambios de dirección en el movimiento

También es importante saber que cuando el cuadricóptero está en modo *HOVERING*, significa que el AR.Drone perdió de vista al objetivo. Y si este está estático en un punto, es decir en una situación de ausencia de movimiento, debido a que el objetivo identificado no se mueve; el AR.Drone se mantiene estable y no hay variaciones en su estado que puedan considerarse como *error*.

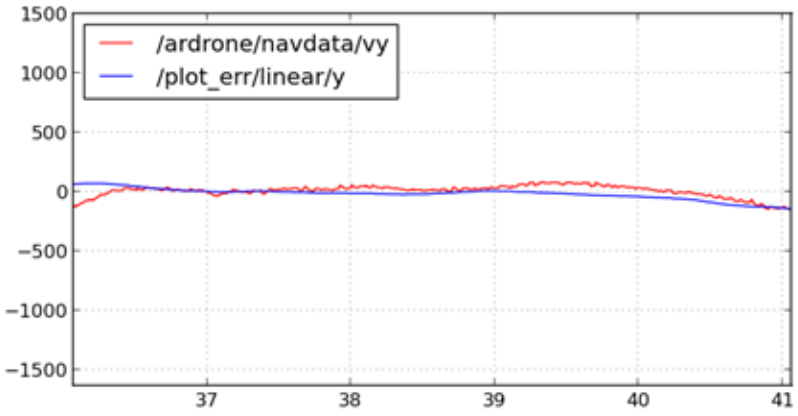


Figura.4.22 _Ausencia de movimiento

Una vez terminadas las pruebas de implementación, se verificó el estado de las hélices, porque con el uso tienden a arquearse y perder su forma original, lo que podría también considerarse como perturbación en el desarrollo del proyecto puesto que influye en el correcto vuelo del cuadricóptero.

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

- El Parrot AR.Drone 2.0, es un cuadricóptero muy versátil como se describe en el Capítulo 1 de este proyecto; ya que con este dispositivo es posible desarrollar un sinnúmero de aplicaciones; porque es resistente a impactos, choques y de fácil programación. Una de las ventajas de usar el AR.Drone 2.0 es que tiene Wi – Fi, lo que hace posible correr programas desde un computador mientras se reciben los datos de los sensores y se envían los comandos.

- Se llegó a la conclusión de que antes del desarrollo de cualquier aplicación, es importante tomar en cuenta que el movimiento del AR.Drone no es suave y es fluctuante mientras se recibe el video desde las cámaras, lo que hace que se dificulte el procesamiento de imágenes.

- Como resultado de la investigación y desarrollo en este proyecto, se puede concluir que mediante el uso de los recursos y herramientas contenidos en el framework ROS, es posible desarrollar diferentes aplicaciones para seguimiento de objetos ya sea por su color o su forma básica característica utilizando el procesamiento de imágenes, a fin de realizar el seguimiento de un objeto determinado con un cuadricóptero en dos grados de libertad mediante visión artificial.

- El framework ROS se caracteriza por apoyar a la reutilización de código para la investigación y desarrollo de la robótica, es decir permite una amplia integración entre diferentes aplicaciones de código abierto para formar sistemas robóticos complejos y elaborados, convirtiéndolo en el sistema preferido por una gran comunidad de desarrolladores; pero así mismo presenta situaciones negativas, ya que al momento de integrar ciertos

repositorios no oficiales desarrollados por su comunidad, los resultados obtenidos dependerán únicamente de la habilidad y experiencia que tenga el desarrollador.

- Para conseguir un seguimiento eficiente se ha utilizado el sistema de control visual basado en imagen para la determinación del posicionamiento del cuadricóptero respecto del objeto en movimiento que se desea seguir, sin embargo, existe la dificultad de que no poder determinar la trayectoria que seguirá el AR.Drone desde su localización inicial hasta la final. De igual forma se evidencia que la calidad del seguimiento mejora con la realización de una estimación robusta del movimiento del objeto seguido, para ello es necesario contar con un sistema de visión artificial robusto con una alta velocidad de procesamiento.

- Durante los experimentos se pudo comprobar la inestabilidad que tiene un cuadricóptero de uso comercial, por lo que el diseño de un controlador capaz de estabilizarlo a pesar de las condiciones externas y las incertidumbres paramétricas es totalmente necesario, especialmente en este proyecto que tiene tareas más exigentes que el vuelo simple de entretenimiento. Por ende se llegó a la conclusión tomando en cuenta el comportamiento característico del AR.Drone (sistema no lineal) y la dificultad de predecir la trayectoria del desplazamiento del objeto a seguir, que se deben unir técnicas de control adecuadas para compensar y corregir de manera eficiente las vulnerabilidades del sistema en el seguimiento. En respuesta a esto fue necesario primeramente la aplicación de un Filtro de Kalman para la estabilización y predicción del movimiento del objeto seguido y segundo un controlador PID para corrección del error de posición durante el seguimiento.

- En cuanto a procesamiento de imagen para hacer el reconocimiento, se concluyó en primer lugar que mientras se hacían las pruebas y se usaba el *rqtplot()* de ROS para graficar las señales, el consumo de recursos en la laptop era muy grande por lo que el pre procesamiento de la imagen, la

captura del objetivo a seguir y el tracking se volvieron un tanto lentos y menos eficientes.

- Cuando se va a trabajar con la librería de OpenCV, como se vio en el Capítulo 2, el video se recibe como mensaje de ROS y es necesario que esté en formato de OpenCV para poder trabajarlo; pues bien, al hacer este cambio de formato se pierden características de *resolución*, pero que no influyen mayormente en el desempeño del procesamiento de las imágenes ni en el resultado final.

- El pre procesamiento de imagen se hizo por 2 métodos, uno considerando el color del objeto cuyos resultados han sido muy buenos debido a la calidad del color que el mismo objeto tiene independientemente de la escena en la que se encuentra, siempre que su tamaño relativo a la escena lo haga lo suficientemente relevante; además tanto la respuesta del procesamiento de imagen como la ejecución de órdenes de control se ejecutan casi de manera inmediata, teniendo un proceso mucho más óptimo. En el segundo método basado en características (features), específicamente en la forma básica del objeto, la precisión y eficiencia ha sido inferior, debido principalmente a la variación de pose de los objetos, al reconocimiento eficaz de las características del objeto, así como a la dificultad de obtener buenos comparadores de *features* en un entorno más complejo y realista. Todas estas tareas requieren mayor capacidad de procesamiento y por eso el proceso es más lento que con el método anterior.

5.2 RECOMENDACIONES

- En la etapa de pruebas, como primer paso se probó el funcionamiento del Ar.Drone con el software de control para iOS y hubo problemas con la conexión Wi – Fi, estos problemas también se presentaron en la comunicación entre la laptop y el cuadricóptero durante algunas de las pruebas. Haciendo un análisis de posibles perturbaciones en el ambiente, se determinó que estas complicaciones se debían a la presencia de mucho

“ruido” y por ello el AR.Drone no respondía a los comandos correctamente; es decir se identificó que el problema era que muchas otras redes inalámbricas utilizaban el mismo canal que el del Wi – Fi que comunica el cuadricópetero con la computador.

Una solución consiste en cambiar el canal utilizado por la red Wi – Fi del AR.Drone, pero esto involucra cambios en la programación del driver del cuadricóptero. Así que una solución más versátil, que es la que se está usando en este proyecto, es aumentar la potencia de la antena Wi – Fi del computador a partir del empleo de un adaptador USB Wireless que brinda mayor cobertura y velocidad en la conexión, con esto las comunicaciones funcionan correctamente.

- Se recomienda implementar un algoritmo de procesado muy robusto y sensible que permita el desarrollo de la aplicación en entornos poco controlados, en los que se presentan sombras aleatorias que en este proyecto significan modificaciones en el color del objeto y oclusiones parciales de los mismos que modifican su forma complicando el reconocimiento y su posterior seguimiento.

- El problema del lento procesamiento de imágenes especialmente cuando se manejan características de formas básicas, puede solucionarse eventualmente con un computador de mayor capacidad y, con un mejor procesador. Esto debido a que cualquier reconocimiento de objetos requiere una capacidad de cómputo muy elevada, y en especial cuando se desea desarrollar procesos en tiempo real.

- Dentro de lo que se refiere al software manejado en el proyecto, una gran ventaja de ROS es que es un sistema completo que permite utilizar tanto el código antiguo de otros proyectos como el código nuevo generado para nuestra aplicación. Es un sistema compatible con repositorios antiguos, esto permite usar la programación existente en otros sistemas, y mover esta programación a casi cualquier máquina ya que el correcto funcionamiento de

ROS solo está garantizado en sistemas de base UNIX, como Ubuntu. Es importante saber que ROS es un sistema soportado por una gran comunidad y por eso tiene muchas ventajas, pero asimismo tiene situaciones negativas, como que al existir gran cantidad de repositorios no todos funcionan en su totalidad y la solución a estos errores dependen también de la comunidad.

- El sistema de control visual basado en imágenes brinda muchos beneficios en la implementación de aplicaciones para el seguimiento de objetivos, pero también presenta algunas dificultades debido al procesamiento de las mismas, es decir a eventos externos tales como condiciones ambientales, ruido, perturbaciones, es por ello que se recomienda trabajar en un ambiente semi-estructurado como el que se detalla en el capítulo 4.

- Para el reconocimiento y seguimiento de un objeto, hay una condición importante a tener en cuenta, el movimiento del cuadricóptero se hace únicamente en dos dimensiones, además se maneja una sola cámara de video (cámara frontal) lo que implica que no se tiene percepción de profundidad; en consecuencia si es que el AR.Drone no se encuentra correctamente ubicado, a una distancia prudente y totalmente “recto” frente al objetivo, tenderá a moverse erróneamente; de *lado*, tratando de mantener su movimiento dentro de los dos ejes de libertad que identifique desde su posición.

- Para trabajos futuros se recomienda contar con un sistema de posicionamiento de cámaras o un sistema GPS, con el que se pueda tener la posición exacta del cuadricóptero en cualquier momento. Para tener un correcto desplazamiento, puede implementarse también un sistema de visión estereoscópica, con 2 cámaras, de manera que se adquiera la ubicación del AR.Drone en las 3 dimensiones posibles y en consecuencia mejorarse el controlador.

REFERENCIAS BIBLIOGRÁFICAS

[1] Pir Zubair Shah, "Pakistan Says U.S. Drone Kills 13", New York Times, June 18, 2009.

[2] Aerial Assassins: Drones. Blog, [Online]. Disponible: http://readanddigest.com/what-is-a-drone/

[3] Leishman, J.G. “Principles of Helicopter Aerodynamics”. New York, NY: Cambridge University Press, 2000.

[4] Skypics Chile, “Noticias de drones civiles en el mundo”, [Online]. Disponible: http://videofotoaerea.cl/drone-chile/drone-usado-por-carabineros/

[5] Pounds, P.; Mahony, R., Corke, P. (December 2006). "Modelling and Control of a Quad-Rotor Robot". In the Proceedings of the Australasian Conference on Robotics and Automation. Auckland, New Zealand.

[6] Coming of Age of Quadrotors, [Online]. Disponible: http://illumin.usc.edu/162/the-quadrotors-coming-of-age/

[7] “What Makes Quadrotors so Popular”, [Online]. Disponible: http://www.quadrotors.net/why-quadrotors-are-so-popular-for-research

[8] Basic Flight Control Laws for Quadrotors. “IMU with Kalman filter”, [Online]. Disponible: http://cog.yonsei.ac.kr/quad/quad.htm

[9] Steve Rossius, “Reconocimiento de objetos mediante WebCam en tiempo Real”, Universidad Politécnica de Valencia.

[10] Wyvern Quadrotor Helicopter, [Online]. Disponible: http://wyvernupenn.blogspot.ca/2010/04/mechanical-design.html

[11] Jara Forcadell Ortiz, Marc Sureda Codina, “Development of autonomous manoeuvres in a quadcopter”, Universidad de Girona, Febrero 2013.

[12] Parrot, “Características y Especificaciones Técnicas”, [Online]. Disponible: http://ardrone2.parrot.com/

[13] “Parrot AR.Drone”, [Online]. Disponible: http://es.wikipedia.org/wiki/Parrot_AR.Drone

[14] Inkyu Sa, Hu He, Van Huynh, Peter Corke. “Monocular Vision based Autonomous Navigation for a Cost-Effective Open-Source MAVs in GPS-denied Environments. Paper.

[15] Daniele Iasella, Stefano Fossati. “Monocular Autonomous Exploration in

Unknown Environment with Low-Cost Quadrotor". Politécnico Di Milano, 2012-2013.

[16] ROS, "Introduction", [Online]. Disponible: http://wiki.ros.org/ROS/...

[17] Álvaro García Cazorla. "ROS: Robot Operating System", Universidad Politécnica de Cartagena, Septiembre 2013.

[18] "Sistema Operativo Robótico", [Online]. Disponible: http://es.wikipedia.org/wiki/Sistema_Operativo_Rob%C3%B3tico

[19] ardrone_autonomy, "A ROS Driver for ARDrone 1.0 & 2.0", [Online]. Disponible:https://github.com/AutonomyLab/ardrone_autonomy/blob/master/README.md

[20] "OpenCV", [Online]. Disponible: http://es.wikipedia.org/wiki/OpenCV

[21] OpenCV, "Introducción a las librerías de OpenCV", [Online]. Disponible: http://web-sisop.disca.upv.es/~imd/cursosAnteriors/2k3-2k4/copiaTreballs/serdelal/trabajoIMD.xml

[22] Gary Bradsky, Adrian Kaehler, "Learning OpenCV. Computer Vision with the OpenCV Library". O'REALLY.

[23] A. Yilmaz. "Object Tracking by Asymmetric Kernel Mean Shift with Automatic Scale and Orientation Selection. Computer Vision and Pattern Recognition", 2007. CVPR'07. IEEE Conference on. 2007.

[24] D. Zhang and G. Lu. "Segmentation of moving objects in image sequence: A review. Circuits, Systems and Signal Process". vol. 20, pp. 143-183, 2001.

[25] Pascual Campoy, Iván Mondragón, Miguel Olivares, Carol Martínez. "Visual Servoing for UAVs". Universidad Politécnica de Madrid.

[26] François Chaumett, Seth Hutchinson. "Visual Servo Control Part 1: Basic Approaches". IEEE, Tutorial.

[27] "Visual Servoing", [Online]. Disponible: http://en.wikipedia.org/wiki/Visual_Servoing

[28] OpenCV Tutorial C++. "OpenCV Lessons", [Online]. Disponible: http://opencv-srf.blogspot.com/p/opencv-c-tutorials.html

[29] Tutorial Camera Calibration, [Online]. Disponible:http://boofcv.org/index.php?title=Tutorial_Camera_Calibration#Lens_Distortion

[30] ROS.org, "Using CVbridge", [Online]. Disponible:

http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeCppCturtle

[31] ARUCO, “Una biblioteca mínimo para aplicaciones de realidad aumentada basada en OpenCV”, [Online]. Disponible: http://realidadaumentadaperu.blogspot.com/2013/07/aruco-una-biblioteca-minimo-para.html

[32] Hilda Caballero Barbosa, “Modelado y Seguimiento de Objetos por medio de Distribución de Color”, 2001.

[33] Eric Yuan's Blog, “Bilateral Filtering”, [Online]. Disponible: http://eric-yuan.me/bilateral-filtering/

[34] Héctor López Paredes, “Detección y seguimiento de objetos con cámaras en movimiento”, Universidad Autónoma de Madrid, 2011.

[35] P. Gil, F. Torres, F. Ortiz, “Detección de Objetos por Segmentación Multinivel combinada de espacios de Color”, Universidad de Alicante, 2004.

[36] Antonio Nicolás Pina, “Clasificación y Búsqueda de Imágenes usando Características Visuales”, Universidad de Murcia, Junio 2010.

[37] OpenCV Processing Book, “Threshold”, [Online]. Disponible: https://github.com/atduskgreg/opencv-processing-book/blob/master/book/filters/threshold.md

[38] Sergio Pereira Ruiz. “Localización de robots mediante filtro de Kalman”.

[39] Iván Fernando Bernal, “On board visual control algorithms for unmanned Aereal Vehicles”, Universidad Politécnica de Madrid, 2011.

[40] Ar Drone 2.0, “Especificaciones Técnicas Ardrone 2.0”, [Online]. Disponible: http://ardrone-2.es/especificaciones-ar-drone-2/

[41] Guilherme Vianna Raffo, “Modelado y Control de un Helicóptero Quadrotor”, Sevilla 2007.

[42] Barquín Murguía, Alberto Isaac. “Implementación de un controlador externo en un cuadricóptero comercial”, Marzo 2014.

[43] “Pid controller”, [Online]. Disponible: http://en.wikipedia.org/wiki/PID_controller

[44] [Online]. Disponible: http://www.mathworks.com/products/matlab/. Revisado el 22 de octubre de 2013

[45] Arnoud Visser, “Closing the gap between simulation and reality in the sensor and motion models of an autonomous AR.Drone”.

ANEXOS

ANEXO A

REQUERIMIENTOS

Para un correcto funcionamiento de este proyecto son necesarias una serie de características y consideraciones mínimas:

SOFTWARE

Se necesita disponer de los siguientes sistemas operativos y programas:

- Ubuntu 12.04
- ROS Groovy
- Paquete ardrone_autonomy
- OpenCV 2.4

HARDWARE

Los requerimientos de hardware incluyen:

- Computador

Procesador i7 de 2.4GHz o superior
Memoria RAM 6GB o superior
Espacio en disco mínimo de 40G

- Adaptador USB D-Link WiFi N150
- AR.Drone 2.0

ANEXO B

MANUAL DE INSTALACIÓN

INSTALACIÓN DE UBUNTU 12.04

1. Lo primero es descargar Ubuntu desde su web oficial; después grabar los archivos ISO en un CD o DVD o bien, en un pendrive.

2. Para iniciar la instalación hay que hacer que el ordenador/portátil arranque el sistema desde el CD/DVD o Pendrive, para ello se debe configurar la BIOS en el Menú Boot Options (Opciones de Arranque), que muestra el ordenador al iniciar y oprimir las teclas F12, F8 o la combinación Ctrl+Alt+Esc.

3. En la primera pantalla el software de instalación muestra dos opciones: **Probar o Instalar**, escogiendo Probar se ejecuta el sistema operativo sin instalar y se puede trabajar con él sin ningún inconveniente; pero para esta aplicación se selecciona Instalar.

4. En la ventana se debe seleccionar también el idioma. En este caso se elige el idioma Español para que la instalación y el sistema Ubuntu se instale en este idioma. A continuación se da click en el botón de Continuar.

Figura. B.1 Inicio de instalación Ubuntu 12.04

5. En este paso se verifica que el ordenador cumpla con todos los requisitos mínimos para la instalación, como por ejemplo: espacio en disco, batería y conexión a internet.

Figura. B.2 Preparación de la Instalación de Ubuntu

Si no se dispone de una conexión a internet por cable y sí por Wi – Fi (inalámbrica), se puede hacer click sobre el ícono de redes (parte superior derecha de la pantalla) para conectarse.

Aquí además si se desea se puede marcar la opción “Instalar Software de Terceros” para que durante el proceso de instalación se descarguen e instalen codecs y temas de licencia que no se instalan por defecto.

Una vez seleccionadas las opciones deseadas oprimir la opción Continuar.

6. En este paso se tienen que considerar diferentes tipos de instalación. Este es el paso más complicado si no se tienen las cosas claras. Las opciones son:

- Instalar junto a otro “sistema operativo”
- Reemplazar “sistema operativo” con Ubuntu
- Algo más

Figura. B.3 Asignar espacio en disco

INSTALAR JUNTO A "OTRO SISTEMA OPERATIVO"

Permite instalar Ubuntu junto a otro Sistema Operativo, con solo desplazar la barra separadora hasta el tamaño deseado para ambas particiones.

Si el sistema operativo acompañante es Windows, no se debe olvidar desfragmentar la partición antes de comenzar la instalación.

Para comenzar la instalación, pulsar en Instalar Ahora.

Figura. B.4 Instalación de Ubuntu junto a Windows

REEMPLAZAR “SISTEMA OPERATIVO” CON UBUNTU

Hay que tener cuidado con esta opción, porque borrará todo el disco para instalar Ubuntu.

Si se tiene una partición de datos, lo recomendable es utilizar la siguiente opción de “algo más”.

Para comenzar la instalación, pulsar en Instalar Ahora.

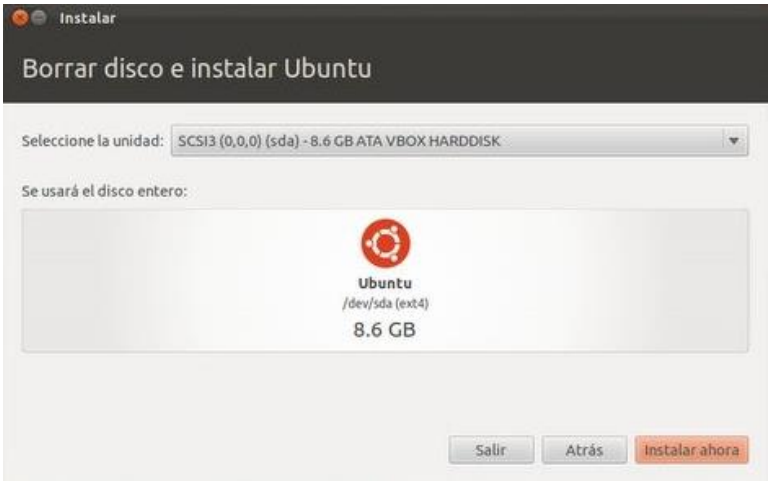


Figura. B.5 Formatear disco e instalar

“ALGO MÁS”

Antes de seguir con la instalación, es importante tener claro ciertos conceptos sobre el tema de particionado.

Particionado: Particionar un disco duro es realizar una división en él de modo que, a efectos prácticos, el sistemas operativo crea que se tiene varios discos duros, cuando en realidad hay un único disco físico dividido en varias partes. De este modo, se pueden modificar o borrar particiones sin afectar a los demás datos del disco.

Los sistemas de archivos indican el modo en que se gestionan los archivos dentro de las particiones. Según su complejidad tienen características como previsión de apagones, posibilidad de recuperar datos, indexación para búsquedas rápidas, reducción de la fragmentación para agilizar la lectura de los datos, etc. Los sistemas de archivos más representativos son:

fat32 o vfat. Sistema de archivos tradicional de MS-DOS y las primeras versiones de Windows. Padece de una gran fragmentación y es un poco inestable.
nfts. Es el nuevo sistema de Windows, usado a partir del 2000 y XP. Es más estable, pero también adolece de información.
ext2. Hasta hace poco era el sistema estándar de Linux. Tiene una fragmentación bajísima, aunque es lento manejando archivos de gran tamaño.
ext4. Es el que usará Ubuntu por defecto.
swap. Es el sistema de archivos para la partición de intercambio de Linux.

Tamaño de las particiones: puesto que a cada partición se le va a dar un uso diferente, cada una tendrá un tamaño diferente.es necesario considerar la partición para la memoria de intercambio que será de tipo swap. Es una costumbre que ésta sea del doble de tamaño que la memoria RAM disponible.

La partición para Ubuntu (montada en /) debe tener al menos 5 GB. Si pretendemos instalar más programas es recomendable darle un poco más de espacio. Para un uso normal, unos 10 GB son convenientes.

Finalmente, es recomendable crear otra **partición para la carpeta personal** (montada en/home). Así se tendrán las configuraciones de las aplicaciones y los archivos personales (documentos, imágenes) en una partición aparte. De este modo, si se quiere reinstalar Ubuntu, se puede formatear sin temor a perder la configuración de programas ni datos. Esta partición se monta en /home.

Preparación del particionamiento: Lo primero que debe hacerse antes de instalar Ubuntu (y particionar el disco) es desfragmentar el disco duro o partición donde esté instalado Windows. Con esta operación se conseguirá que los

diferentes fragmentos de los archivos se junten y sea menos probable su pérdida en el proceso de particionamiento. No olvidar **apuntar el nombre de las particiones** para recordar cual es cual a la hora de la instalación.

Ahora, al escoger la opción de Algo Más en la ventana hay que seguir los siguientes pasos:

1º Seleccionar la **partición para Ubuntu**, pulsando en "cambiar" y en la nueva ventana, elegir en los desplegables, el sistema de ficheros "ext4" y el punto de montaje "/".

2º Seleccionar la **partición para la Carpeta Personal**, pulsando en "cambiar" y en la nueva ventana, elegir en los desplegables, el sistema de ficheros "ext4" y el punto de montaje "/home".

3º Seleccionar la **partición para la Memoria de Intercambio**, pulsando en "cambiar" y en la nueva ventana, elegir en los desplegables, el sistema de ficheros "swap" y el punto de montaje "nada".

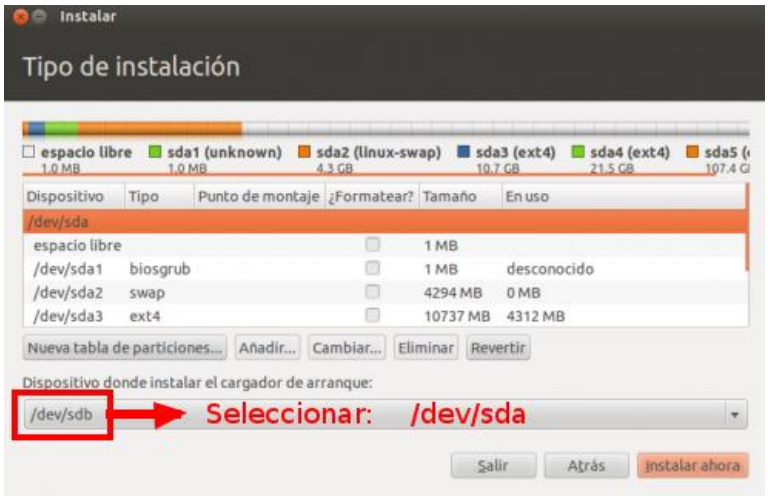


Figura. B.6 Partición del disco

Ahora seleccionar el "**dispositivo donde instalar el cargador de arranque**" (desplegable en la parte inferior de la ventana), donde se selecciona el 1er disco duro "/dev/sda" (no la partición), donde se instalará el GRUB (gestor de arranque

múltiple que se usa para iniciar dos o más sistemas operativos instalados en un mismo ordenador).

7. Aquí se selecciona la localidad de residencia y probablemente dé por defecto una localización casi exacta. Entonces el instalador configurará la zona horaria, moneda, idioma, distribución de teclado, entre otros parámetros.

Figura. B.7 Inicio de la instalación

8. Este paso es importante para utilizar el teclado sin problemas. En la línea de abajo se pueden teclear las letras y símbolos para asegurarse de que es la distribución correcta.

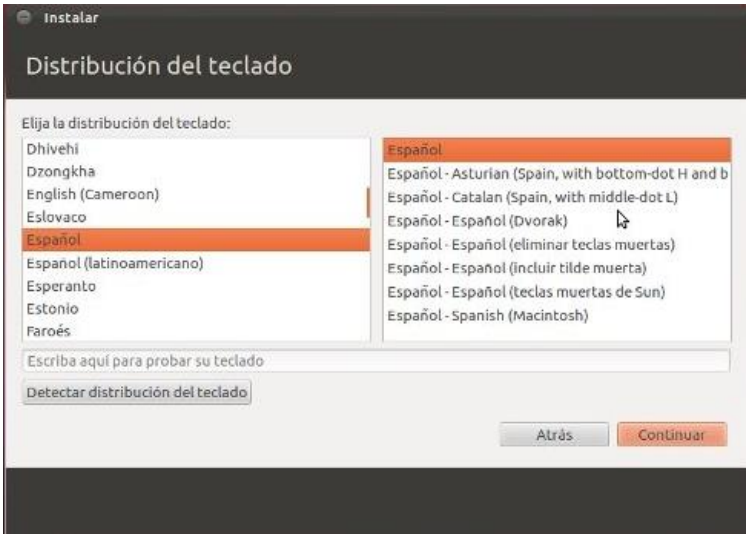


Figura. B.8 Selección de entrada del teclado

Habiendo seleccionado antes correctamente el idioma y la localización no habrá problemas, pero para el idioma español puede haber cierta confusión, ya que hay 2 distribuciones de teclado distintas, una para Latinoamérica y otra para España.

Un truco muy utilizado para asegurarse es pulsar la tecla @ (arroba). Este símbolo se encuentra ubicado en la tecla 2 (dos), para los teclados con idioma Español-Español y en la tecla Q para los teclados Español (latinoamericano)-Español.

9. Aquí el usuario se identifica como Administrador del Sistema. Por defecto, en Ubuntu la cuenta del root viene desactivada. Este usuario será miembro del grupo del root y tendrá privilegios de Administrador.

Hay que tener en cuenta lo siguiente para no tener problemas más tarde:

- Al escribir "su nombre" se rellenará automáticamente la casilla de "Nombre de usuario" .**Escribir todo en** minúsculas; al introducir la 1ª letra en mayúscula en "su nombre", en el usuario aparecerá el mismo nombre con minúscula y ese es su usuario para luego trabajar con comandos, no el de la mayúscula en la 1ª letra.

- El campo del "Nombre de su Equipo" se rellena automáticamente, pero se puede cambiarlo por "casa" o cualquier otro.

- Para que una contraseña sea fuerte debe de tener al menos 8 dígitos y contener mayúsculas, minúsculas, letras, símbolos y números. Dejarla en blanco es una temeridad...

- Activar la casilla "Iniciar sesión automáticamente" puede traer futuros problemas. Dejar activada la que viene por defecto "Solicitar contraseña para iniciar sesión".

- Si se activa la casilla "Cifrar mi carpeta personal", no olvidar nunca las contraseñas que le puestas al cifrado, porque se pueden perder todos los Datos.

10. Ahora solo queda esperar. Una vez instalado, se debe reiniciar el equipo, extraer el CD o USB de instalación y eso es todo. Finalmente se debe actualizar el repositorio de Ubuntu con el comando **sudo apt-get update** ejecutado en un terminal (símbolo del sistema en Windows).

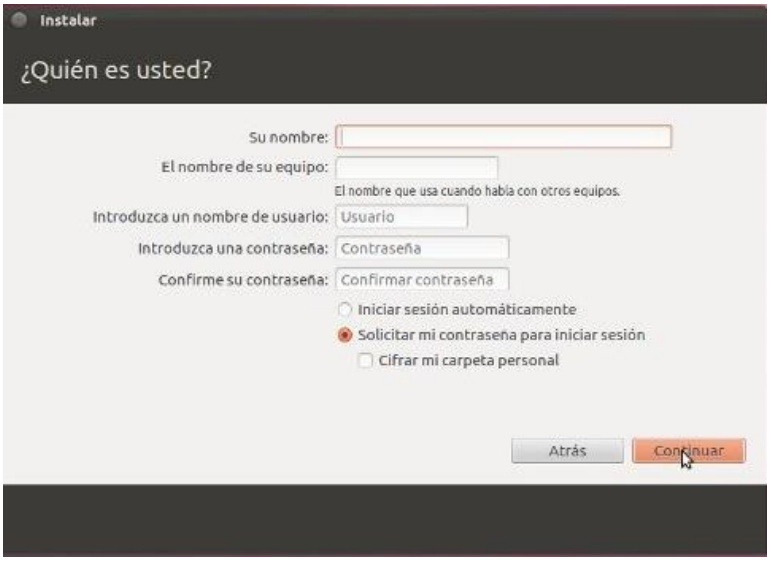


Figura. B.9 Ingreso de datos del usuario

INSTALACIÓN DE ROS (ROBOT OPERATING SYSTEM)

Abrir un nuevo terminal y copiar consecutivamente cada uno de los códigos presentados a continuación:

1. En primer lugar se tiene hacer la configuración inicial de los repositorios.

Para empezar se debe configurar Ubuntu, específicamente apt, para que reconozca los repositorios de ROS y permita instalarlo y actualizarlo. Por defecto en Ubuntu 12.04 ya están habilitados los repositorios universe, multiverse y restricted, lo cual era un paso adicional cuando se instalaba ROS en versiones anteriores de Ubuntu. Para verificarlo se abre la interfaz gráfica de orígenes de software con el siguiente comando:

```
$ python /usr/bin/software-properties-gtk
```

Aquí se verifica que están seleccionados los cuatro tipos de repositorios (main, universe, restricted y multiverse). Ahora, lo que debe hacerse es añadir los repositorios de ROS a la sources.list de software y configurar el acceso con los siguientes comandos:

```
$ 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 -
```

Ahora ya se tienen los repositorios de ROS configurados para su administración desde el comando apt.

2. La segunda parte es básicamente la Instalación y configuración general

Aunque ya se configuró los repositorios de ROS para su funcionamiento con apt, se tiene que corroborar que el listado de paquetes esté actualizado ejecutando:

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

Debido a que ROS contiene gran cantidad de herramientas y utilidades, hay 4 opciones de instalación. La primera opción y la más recomendada es la instalación completa (Desktop-full), la cual incluye todo el core de ROS, librerías genéricas de varios robots, simuladores 2D y 3D, herramientas de visualización y librerías de navegación y percepción.

```
$ sudo apt-get install ros-groovy-desktop-full
```

La segunda opción (Desktop) incluye el core, las librerías de robots y las herramientas de visualización.

```
$ sudo apt-get install ros-groovy-desktop
```

La tercera opción (ROS-base) solamente tiene el core de ROS y librerías de comunicación, nada de interfaces gráficas (GUI).

```
$ sudo apt-get install ros-groovy-ros-base
```

La última opción es instalar paquetes individuales especificando el nombre en el comando:

```
$ sudo apt-get install ros-groovy-PACKAGE
```

por ejemplo:

```
$ sudo apt-get install ros-groovy-slam-gmapping
```

Para la instalación de la versión completa se descargan aproximadamente 900MB, que con una conexión de 4M se demoraría entre 30 minutos y 45 minutos, y tras esto se configuran los paquetes descargados. Después de haber finalizado la instalación, hay que configurar algunas herramientas adicionales y el entorno de trabajo.

Se empieza inicializando **rosdep**, una herramienta que facilita la instalación de algunas dependencias de paquetes y que es necesaria para ejecutar algunos componentes de ROS.

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

Es conveniente que cada vez que se inicie sesión las variables de entorno de ROS sean cargadas automáticamente:

```
$ echo "source /opt/ros/groovy/setup.bash" >> ~/.bashrc
```

```
$ source ~/.bashrc
```

Se verifica que ya están declaradas las variables de entorno de ROS:

```
$ export | grep ROS
declare -x ROS_DISTRO="groovy"
declare -x ROS_ETC_DIR="/opt/ros/groovy/etc/ros"
declare -x ROS_MASTER_URI=http://localhost:11311
declare -x
ROS_PACKAGE_PATH="/opt/ros/groovy/share:/opt/ros/groovy/stacks"
declare -x ROS_ROOT="/opt/ros/groovy/share/ros"
```

Ahora se instala **rosinstall**, una herramienta que ayuda a descargar las fuentes de los paquetes y su instalación. Se descargan aproximadamente 10 MB.

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

Finalmente, se tiene ROS Groovy instalado y configurado en el sistema Ubuntu 12.04.

INSTALACIÓN DE OPENCV

La librería de OpenCV está dentro de los paquetes que contiene ROS, pero es necesario descargar e instalar ciertas dependencias que no vienen por default y que resultan necesarias para aplicaciones un tanto más complejas o que por otro lado ayudan a mejorar el funcionamiento de estas aplicaciones.

Para ello, se ejecuta el siguiente comando en un terminal:

```
sudo apt-get install build-essential libgtk2.0-dev libjpeg-dev libtiff4-dev libjasper-dev libopenexr-dev
 cmake python-dev python-numpy python-tk libtbb-dev libeigen2-dev yasm libfaac-dev libopencore-
amrnb-dev libopencore-amrwb-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev libqt4-de
v libqt4-opengl-dev sphinx-common texlive-latex-extra libv4l-dev libdc1394-22-dev libavcodec-dev l
ibavformat-dev libswscale-dev
```

CREACIÓN DEL ÁREA DE TRABAJO

Luego de instalar ROS, se necesita un área de trabajo, ésta se creará por medio de ros, el cual permite crear un ambiente de trabajo organizado por un sistema de control de versiones. Es importante tener separado los paquetes que se vayan a crear o alterar de los paquetes proporcionados por ROS, por motivos de orden para evitar confusiones. Con el siguiente comando, se crea la carpeta ros, en ella se va a trabajar, y ros se encarga de crear los archivos necesarios setup.bash, setup.sh, setup.zsh, .rosinstall

```
$ ros init [nombre del espacio de trabajo] [localización de los ficheros de ros]
```

En este caso:

```
$ rosws init ros_workspace /opt/ros/groovy
```

Si se abre la carpeta personal se tendría lo siguiente:

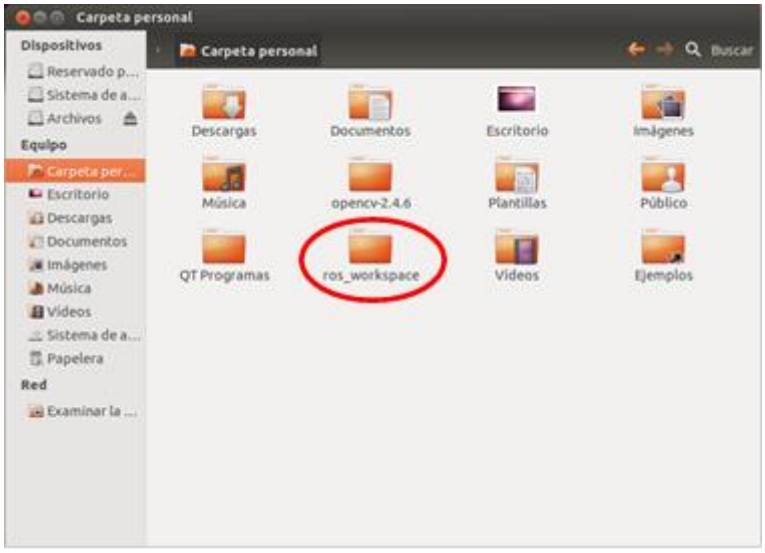


Figura. B.10 Área de trabajo de ROS

Para poder usar esta carpeta y trabajar con los demás paquetes instalados en /opt/ros/groovy y agregar el nuevo directorio a la variable ROS_PACKAGE_PATH. Se ejecuta:

```
$ echo "ros_workspace/setup.bash" >> ~/.bashrc
$ . ~/.bashrc
```

O en la shell a utilizar:

```
$ source ~/ ros_workspace/setup.bash
```

Cada vez que se crea un paquete este se debe agregar al repositorio, este trabajo puede ahorrarse si los paquetes se ponen en un directorio creado y configurado de la siguiente manera:

```
$ mkdir ~/ros_workspace/paquete
$ rosws set ~/ros_workspace/paquete
```

Para actualizar:

```
$ source ~/ros_workspace/setup.bash
```

Para asegurarse que la variable ROS_PACKAGE_PATH funciona, se la imprime:

```
$ echo $ROS_PACKAGE_PATH
```

Salida la cual debe ser parecida a:

```
/home/your_user_name/ros_workspace/paquetes:/opt/ros/groovy/share:/opt/ros/groovy/stacks
```

Abriendo la carpeta de área de trabajo, se encuentran los siguientes archivos:

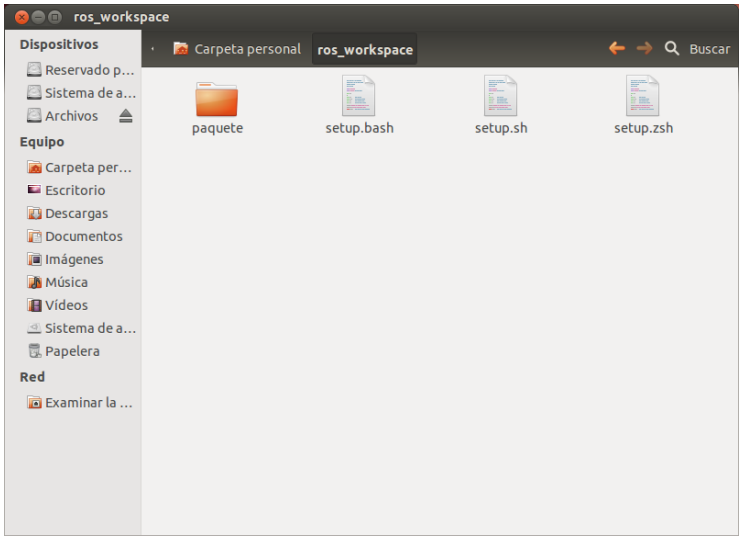


Figura. B.11 Preparación del área de trabajo

Si se desea también se puede configurar el programa para dejar “configurado” el entorno de forma automática para cada vez que se abra un nuevo terminal, se utilizan las siguientes funciones:

```
$ echo"source/opt/ros/fuerte/setup.bash">>~/.bashrc
$echo"exportROS_PACKAGE_PATH=~/miworkspace:$ROS_PACKAGE_PATH">>~/.bashrc
$ echo"exportROS_WORKSPACE=~/miworkspace">>~/.bashrc
$ echo"exportROS_HOSTNAME=localhost">>~/.bashrc
$ echo"exportROS_MASTER_URI=http://localhost:11311">>~/.bashrc
```

INSTALACIÓN DEL PACKAGE ARDRONE_AUTONOMY

Para compilar este paquete se lo hace a través del compilador de catkin, para lo que es necesario en primer lugar crear una carpeta contenedora.

```
$ mkdir ~/ros_workspace/paquete/catkin_ws
$ cd ~/ros_workspace/paquete/catkin_ws
$ mkdir src
$ cd src
```

Se descarga el paquete del repositorio y se compila.

```
$ git clone https://github.com/AutonomyLab/ardrone_autonomy.git
```

```
$ cd ~/catkin_ws
$ catkin_make
```

Al final de todos y cada uno de estos pasos se debe tener un resultado como el que se muestra a continuación en el terminal, y consecuentemente el paquete ya está creado y configurado en su totalidad.

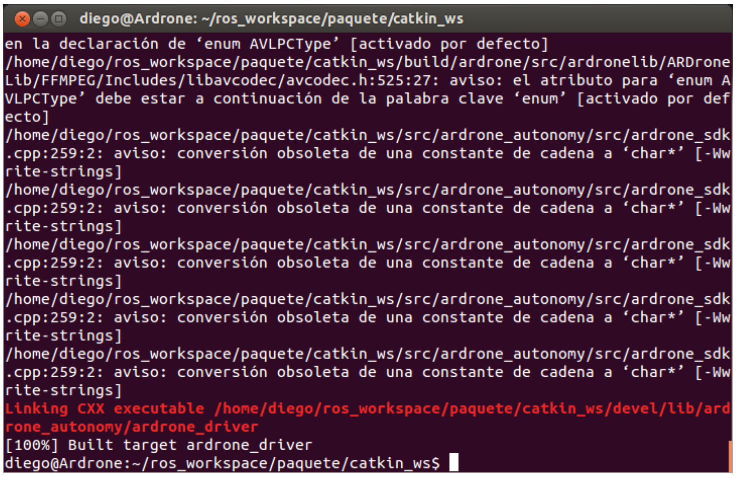


Figura. B.12 Instalación ardrone_autonomy completa

CREACIÓN DE UN PAQUETE

Se accede a la dirección dentro del área de trabajo ya configurada para crear nuestro paquete.

```
$ cd ~/ros_workspace/paquete
```

Se crea el nuevo paquete con el comando

```
$ roscreate [nombre del paquete] [dependencias]
```

Para esta aplicación:

```
$ roscreate-pkg tesis_seg_ros image_transport roscpp std_msgs opencv2 cv_bridge sensor_msgs
ardrone_autonomy geometry_msgs
```

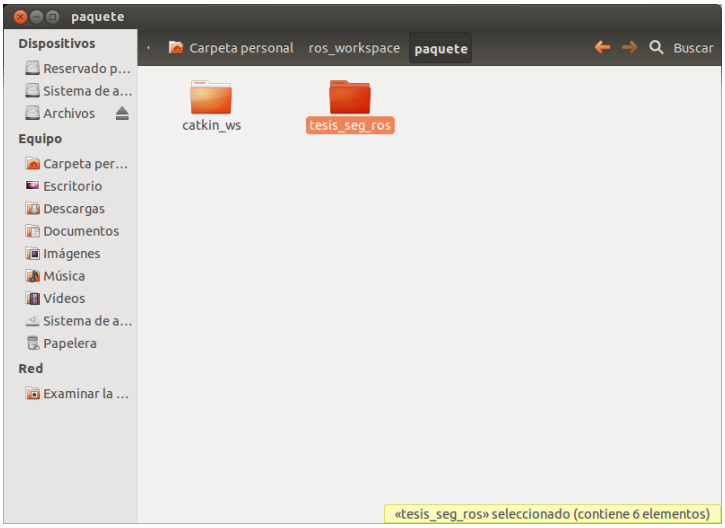


Figura. B.13 creación del nuevo paquete del proyecto

La cual internamente contiene los siguientes archivos y carpetas.

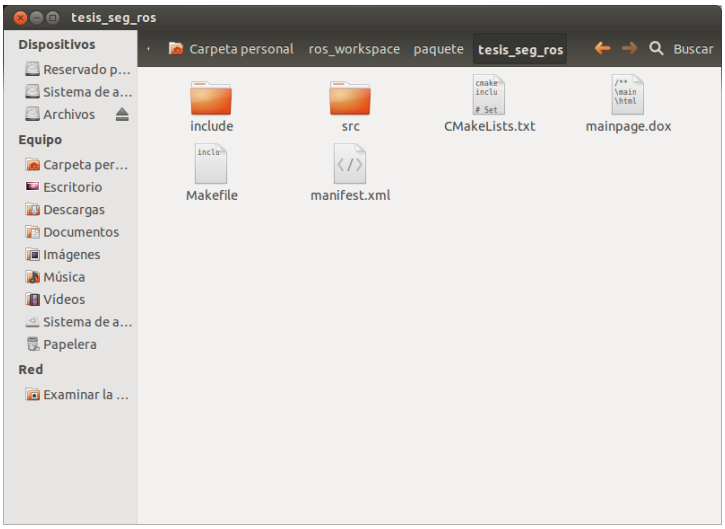


Figura. B.14 Contenido inicial de un nuevo paquete

Se editan los respectivos archivos *Cmakelists.txt* y *manifest.xml* útiles para la compilación del proyecto. Estos archivos tendrán la información necesaria de las dependencias del paquete y la creación de sus archivos ejecutables.

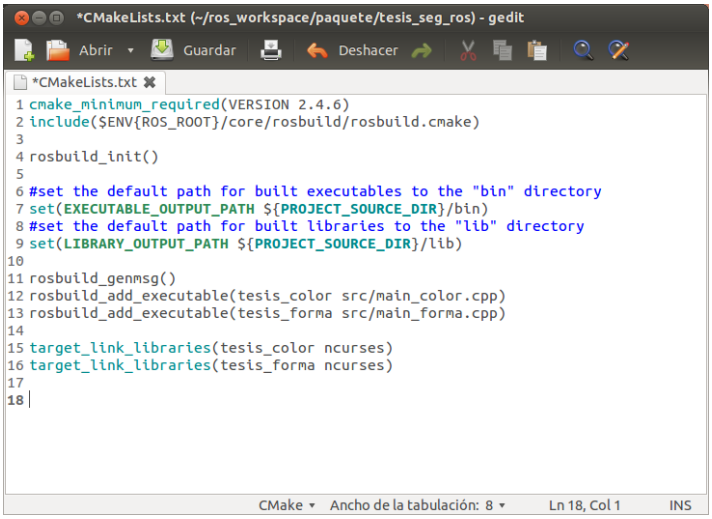


Figura. B.15 Archivo cmakelists.txt para compilación de ejecutables

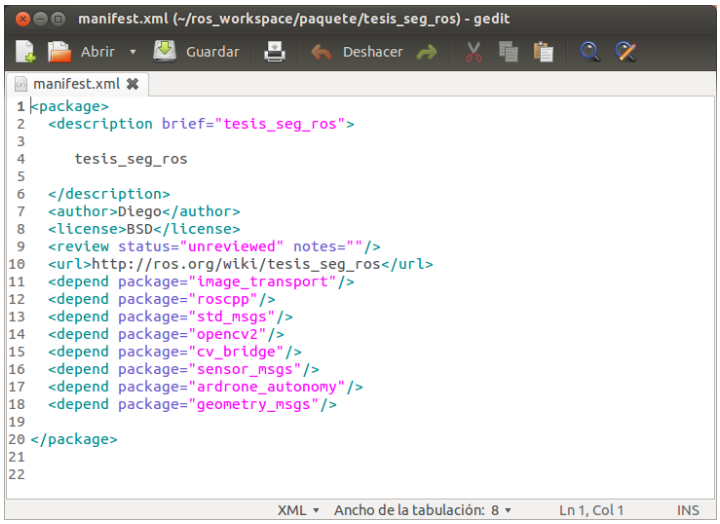


Figura. B.16 Archivo manifest.xml compilación de dependencias

Luego de ello se dirige a la carpeta src donde va a estar almacenado el código fuente. En este caso los diferentes códigos en lenguaje c++ para los seguimientos del AR.Drone a hacia un color o a una forma definida.

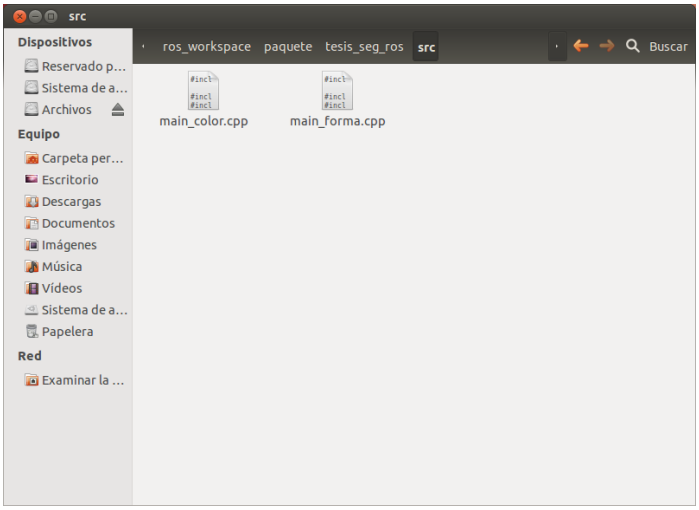


Figura. B.17 Archivos fuente de programación

En la carpeta en donde está el paquete se debe añadir una subcarpeta contenedora de los archivos launch, que permitirán correr las aplicaciones de manera sencilla con un simple código.

Para correr estos archivos se lo hace con el comando

```
$ roslaunch [nombre del paquete] [nombre del archivo launch]
```

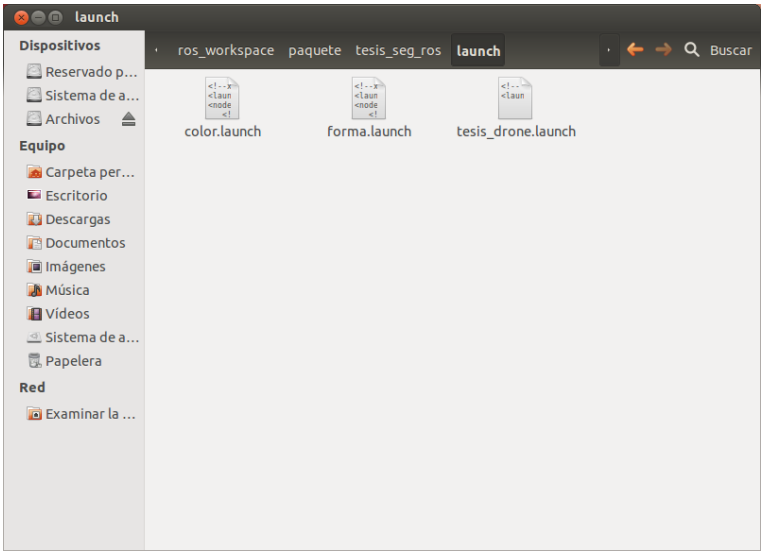


Figura. B.18 Archivos Launch ejecutables

Cada uno de estos archivos contiene una configuración específica para correr los respectivos nodos con sus propios parámetros.

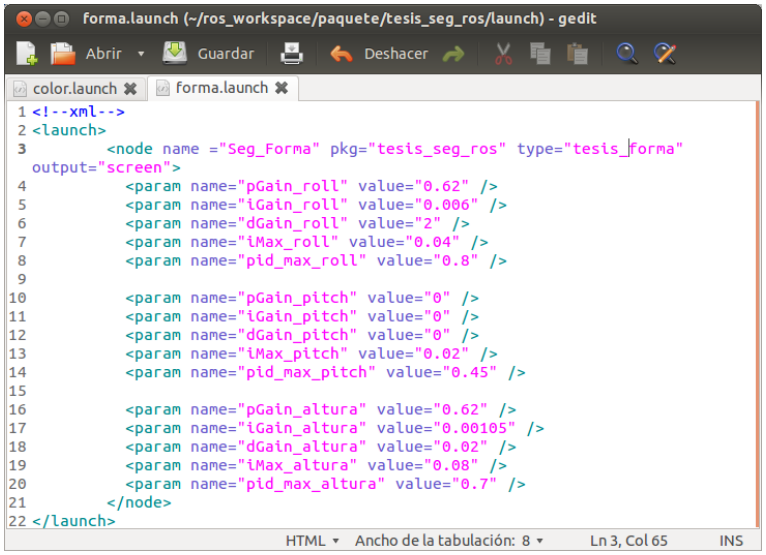


Figura. B.19 Configuración del archivo Launch

Una vez configurados todos los archivos anteriores se abre un nuevo terminal, y se debe compilar el comando:

```
$ Rosmake tesis_seg_ros
```

Lo cual compilará el paquete y si no se tiene ningún error tendrá que aparecer un resultado muy parecido a este.

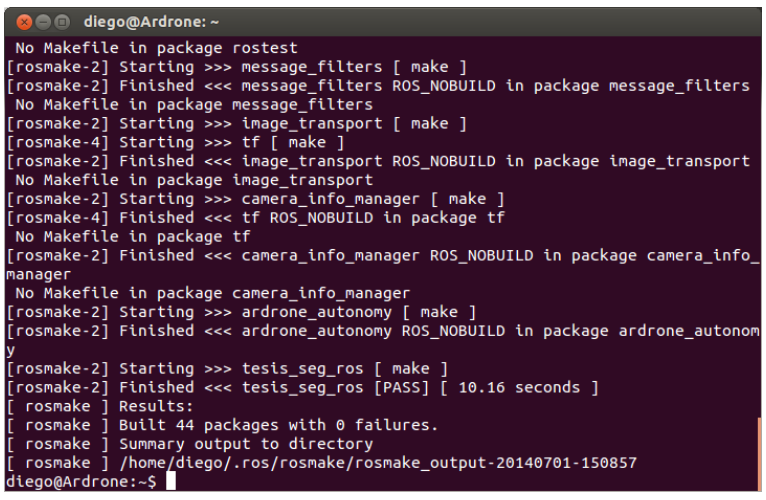


Figura. B.20 Ejecutando Aplicación del seguimiento

Indicando la compilación total del paquete sin errores.

ANEXO C

INSTRUCCIONES DE USO

CONEXIONES DEL HARDWARE

AR.DRONE

Se debe quitar la protección externa del AR.Drone para poder conectar la batería Parrot High Density Battery 1500mA, y posteriormente sujetarla con su seguro de correa. Seguidamente volver a colocar la protección. Esperar a que los 4 leds bajo el AR.Drone se enciendan de color verde, lo cual significa que el AR.Drone está listo para ser comandado.

Figura. C.1 Batería High Density Parrot

COMPUTADOR

Conectar el dispositivo receptor de Wi – Fi al puerto USB de la computadora

Figura. C.2 Adaptador wifi dlink

ESTABLECIMIENTO DE LA CONEXIÓN

Conectarse a la red wifi emitida por el cuadricóptero de nombre Ardrone2_xxxx

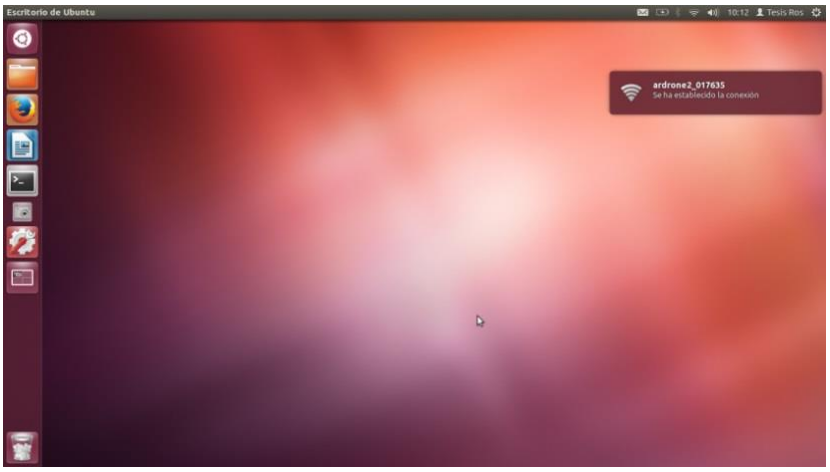


Figura. C.3 Establecer conexión con Ar-Drone 2.0

INTERFAZ DE USUARIO

Como todos los sistemas operativos derivados de Unix, GNU/Linux se dispone de un intérprete de órdenes o terminal (en inglés se utiliza la palabra shell) que hace de interfaz entre el usuario y el propio sistema operativo, en el mismo que se pondrán los comandos respectivos para correr este proyecto.

Para abrir un nuevo terminal presionamos las teclas Ctrl + Alt + T.

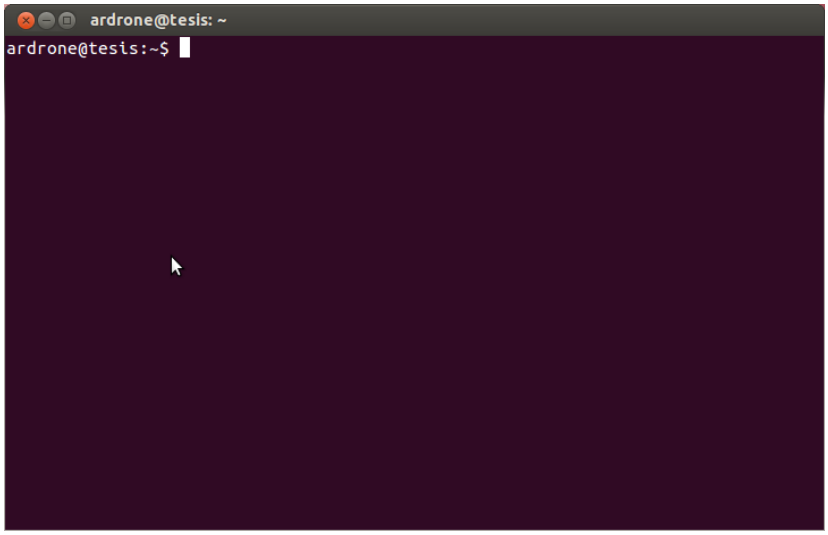


Figura. C.4 Terminal de Usuario de Ubuntu

De manera rápida se comprobará si el firmware ROS está bien instalado en el ordenador, introduciendo en el terminal.

```
$ roscd
```

Comando que lleva a la carpeta configurada como área de trabajo para futuros proyectos en ROS. En este caso se dirige a la carpeta “ros_workspace”, contenedora de esta aplicación.

Y adicionalmente se buscará el paquete contenedor de los archivos ejecutables que permiten la implementación de este proyecto. Se digitará:

```
$ rospack find tesis_seg_ros
```

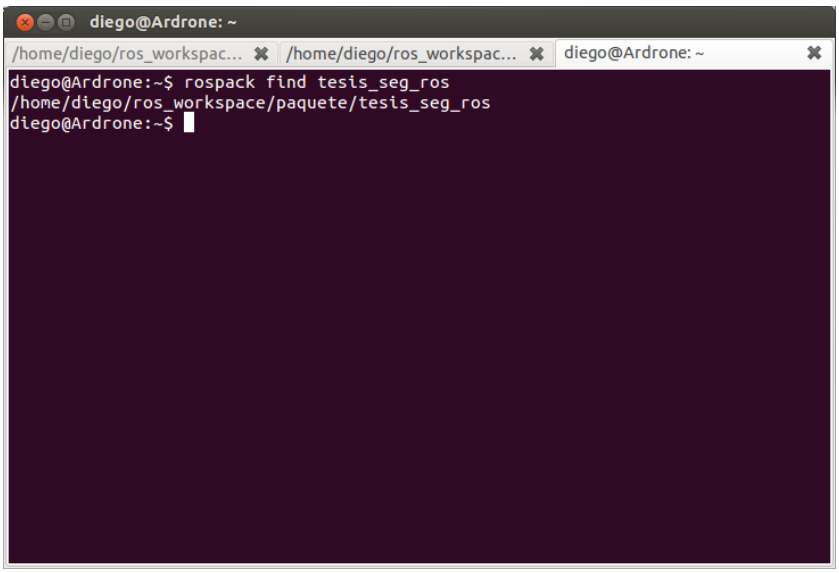


Figura. C.5 Rospack find

Si ha sido encontrado el paquete, el terminal desplegará la dirección contenedora del paquete.

SEGUIMIENTO POR SEGMENTACIÓN DE COLOR

Para iniciar el seguimiento por color se debe ejecutar dos archivos launch que contienen la configuración inicial del seguimiento.

El primer archivo se encargará de correr el nodo del *ardrone_driver, encargado de establecer la comunicación con el AR.Drone 2.0* utilizando el siguiente comando:

```
$ roslaunch tesis_seg_ros tesis_drone.launch
```

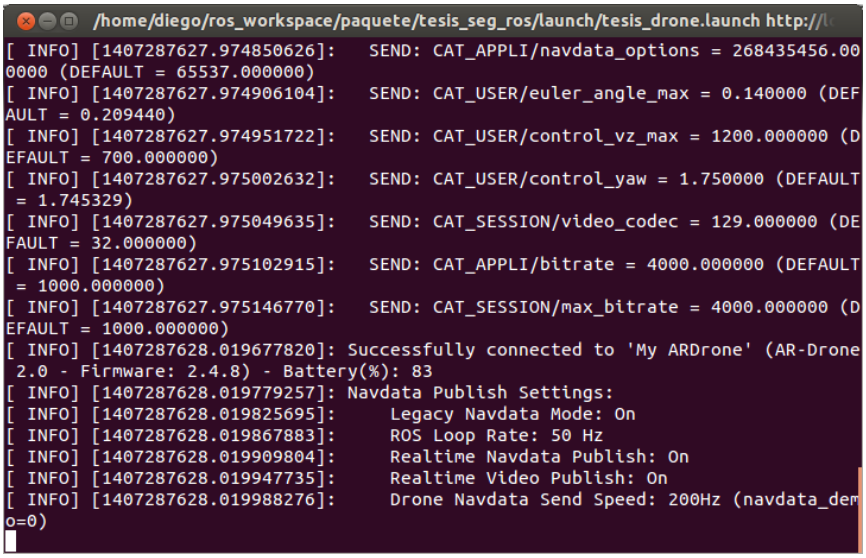


Figura. C.6 Correr el nodo de seguimiento del AR.Drone

El segundo archivo launch se encargará de ejecutar la aplicación de seguimiento. Para lo cual abrirá una nueva pestaña del terminal y se digita:

```
$ roslaunch tesis_seg_ros tesis_color.launch
```

Inmediatamente se va a desplegar la interfaz de usuario, conteniendo los datos más relevantes del control en todo el seguimiento, conjuntamente con una nueva ventana, presentando el video original transmitido por el cuadricóptero (parte izquierda de la ventana) y el resultado final del procesamiento de la imagen (parte derecha de la ventana).

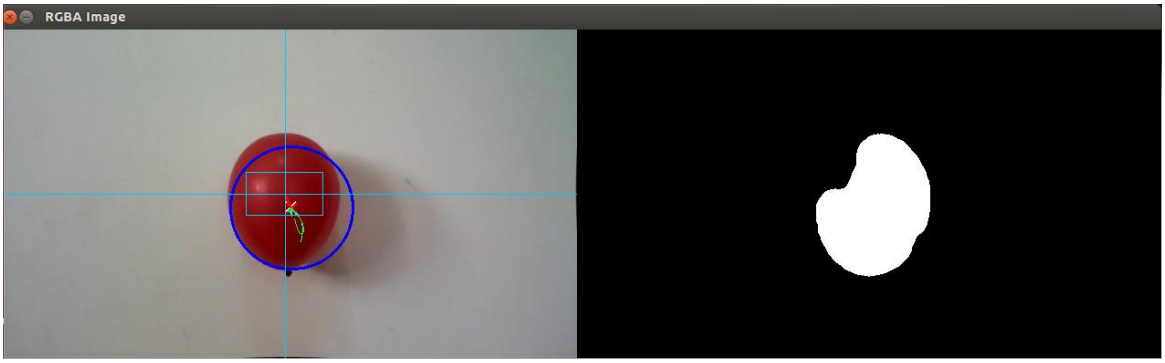


Figura. C.7 Procesamiento por color

La información presentada en la interfaz de usuario esta detallada a continuación.

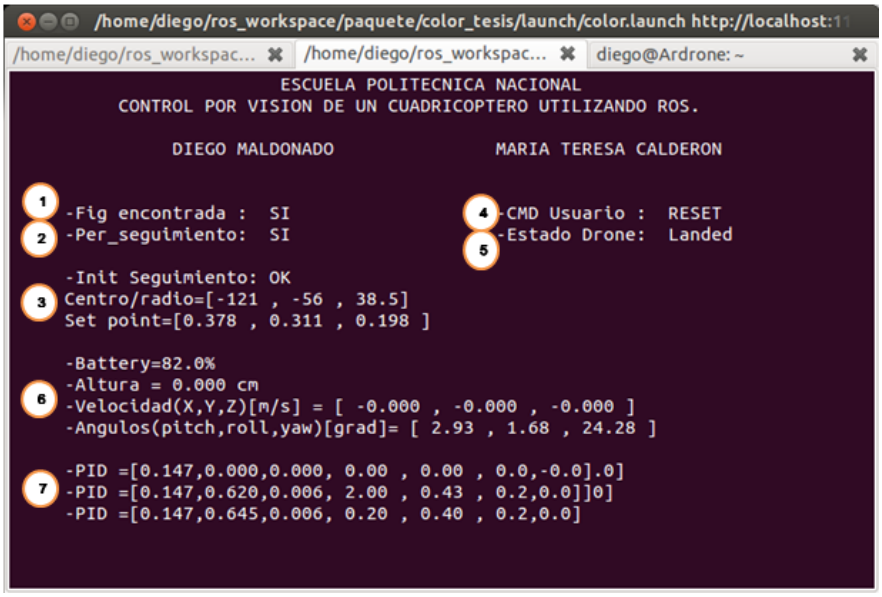


Figura. C.8 Interfaz de Usuario de seguimiento por color

1. **Figura encontrada**: indica si se ha encontrado o no alguna forma con el color determinado que se busca para seguimiento.
2. **Permiso de seguimiento:** muestra si el usuario da o no la orden para el inicio del seguimiento por parte del quadrotor al color determinado.
3. **Inicio del seguimiento:** en caso de ser positivos los anteriores anuncios, es decir si y solamente si se ha encontrado alguna forma con el color que se desea seguir y el usuario ha dado el permiso de seguimiento, entonces el ordenador comenzará a ejecutar las respectivas órdenes de control para ser enviadas al AR.Drone.

Siendo el seguimiento un hecho, se despliega la información sobre la ubicación del objeto dentro del campo visual de la cámara y el setpoint calculado para corregir el error de posición encontrado.

4. **CMD usuario:** despliega los respectivos comandos que el usuario ingresa al ordenador para control del quadrotor. En los cuales van a estar los estados: Take Off, Land, Reset, Flattrim
5. **Estado AR.Drone:** despliega el respectivo estado del AR.Drone en su operación, es decir Unknown, Inited, Landed, Flying, Hovering, Test, Taking off , Landing , Looping
6. **IMU ON BOARD**: presenta información de la IMU a bordo del AR.Drone que permite al usuario tener más conocimiento de las características de vuelo en tiempo real del AR.Drone; tales como velocidad, altura o batería.
7. **PID:** se presenta los valores de las ganancias del modelo matemático para el controlador PID, con sus respectivos valores.

ÓRDENES DEL USUARIO

Para comandar el seguimiento en cada instante, el usuario puede enviar diferentes órdenes de control desde el computador mediante el teclado, para ello se ha designado las siguientes teclas.

Comandos desde teclado		
Tecla	**Función**	**Descripción**
T	Take Off	Orden para despegar al quadrotor
L	Land	Orden para aterrizar al quadrotor
R	Reset	Resetea los parámetros del AR.Drone
F	Flattrim	Calibra la IMU a bordo del AR.Drone, para condiciones iniciales de vuelo
P	Permiso Seguimiento	Permite o no el seguimiento del quadrotor al objeto.

SELECCIÓN DEL COLOR A SEGUIR.

Para seleccionar el color, solamente se debe dar click sobre el color deseado en la ventana donde se transmite el video, y automáticamente el sistema aprenderá la tonalidad del color. Para seleccionar un nuevo color a seguir es necesario volver a dar click sobre el objeto con el nuevo color deseado y se cambiará automáticamente.

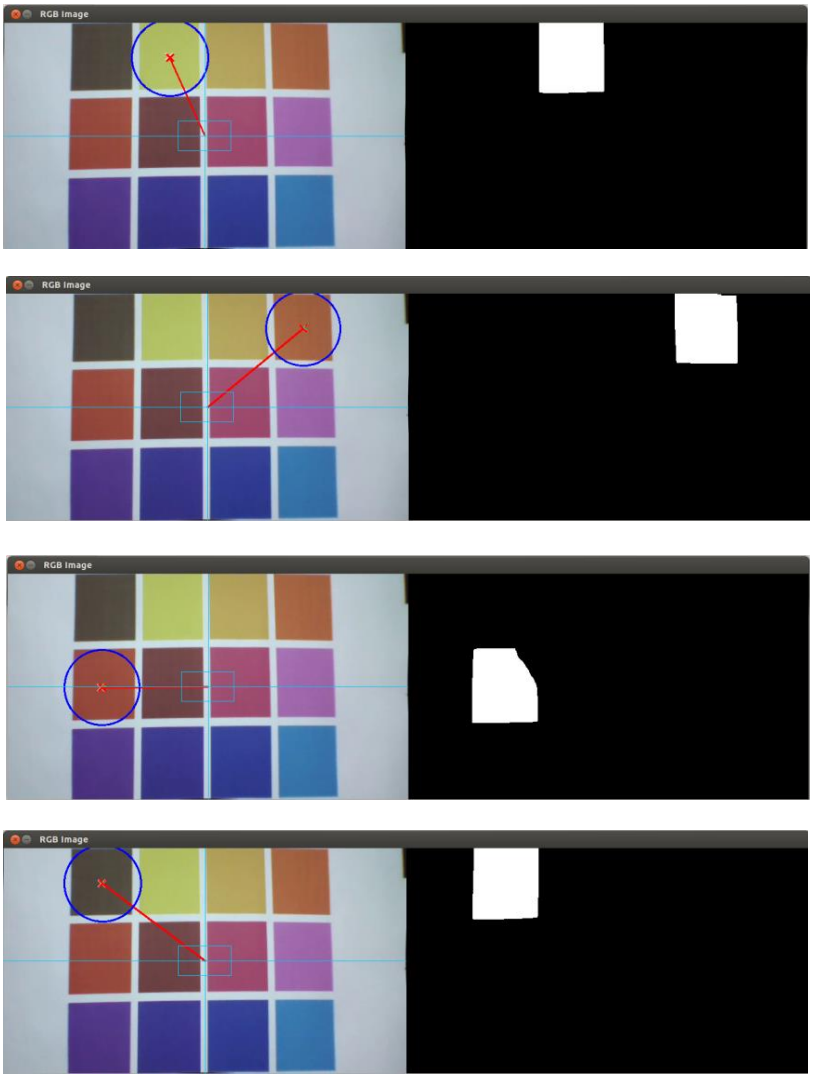


Figura. C.9 Selección del color mediante el Mouse

Visualmente se presenta la estabilización efectuada por parte del filtro de kalman en donde se ha pintado de color azul la trayectoria original del centro del objeto, y de color verde la trayectoria resultante del filtro.

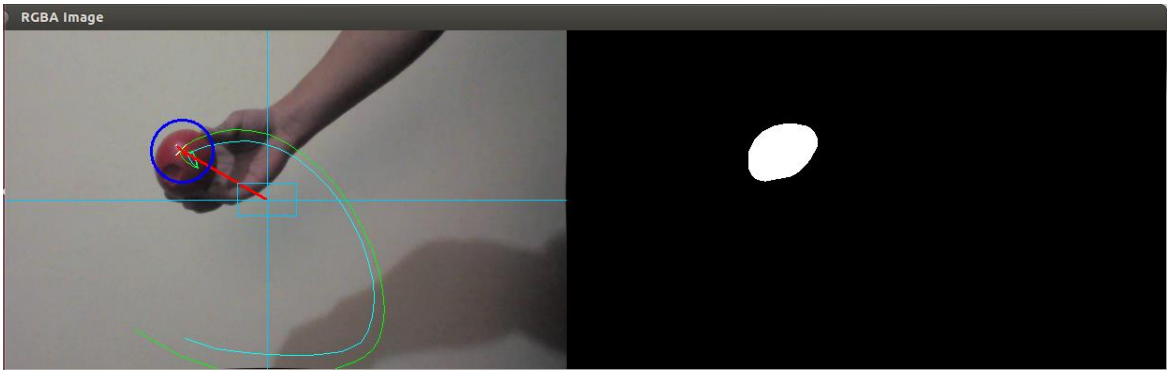


Figura. C.10 Estabilización del seguimiento por el Filtro de Kalman

El problema principal se presenta cuando el medio en el que se desenvuelve el objeto seguido, es muy parecido a las características del mismo, es por ello que cuando sucede dicho evento se despliega en la ventana principal un mensaje informativo, dando a conocer al usuario este problema.

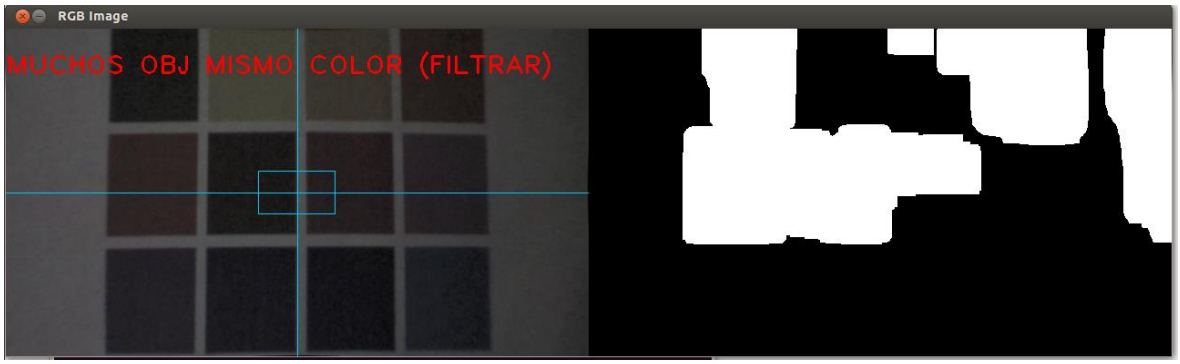


Figura. C.11 Problema de iluminación

SEGUIMIENTO POR FORMA

Para iniciar el seguimiento por Forma de igual forma se debe ejecutar dos archivos launch contenedores de la configuración inicial del seguimiento.

El primer archivo se encargará de correr el nodo del *ardrone_driver, encargado de establecer la comunicación con el AR.Drone 2.0* utilizando el siguiente comando:

```
$ roslaunch tesis_seg_ros tesis_drone.launch
```

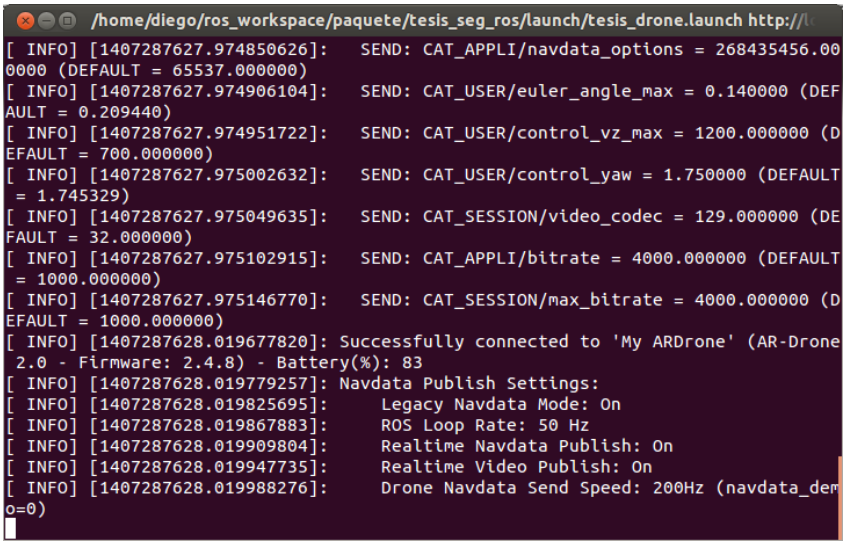


Figura. C.12 Correr el nodo de seguimiento del AR.Drone

El segundo archivo launch se encargará de ejecutar la aplicación de seguimiento. Para lo cual abrirá una nueva pestaña del terminal y se digita:

```
$ roslaunch  tesis_seg_ros  color.launch
```

Inmediatamente se va a desplegar la interfaz de usuario, conteniendo los datos más relevantes del control en todo el seguimiento, conjuntamente con una nueva ventana, presentando el video original transmitido por el quadrotor (parte izquierda de la ventana) y el resultado final del procesamiento de la imagen (parte derecha de la ventana).

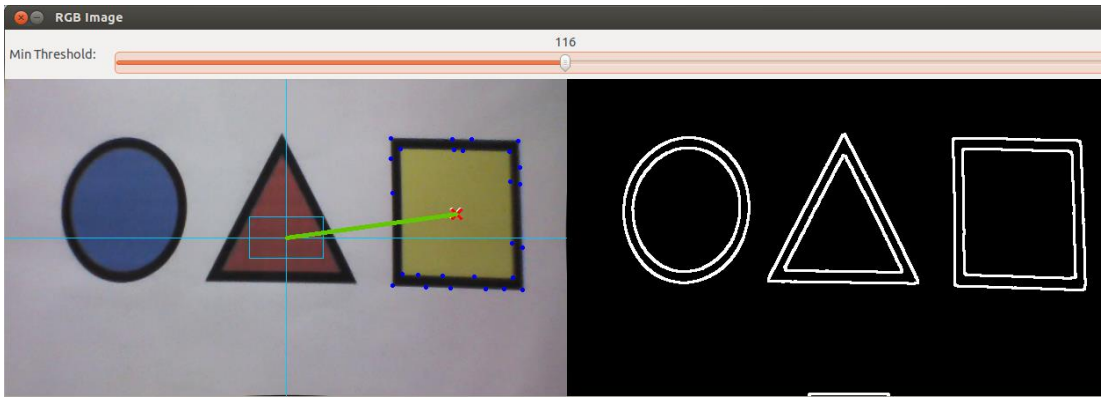


Figura. C.13 Procesamiento por forma

La información presentada en la interfaz de usuario de detalla a continuación:

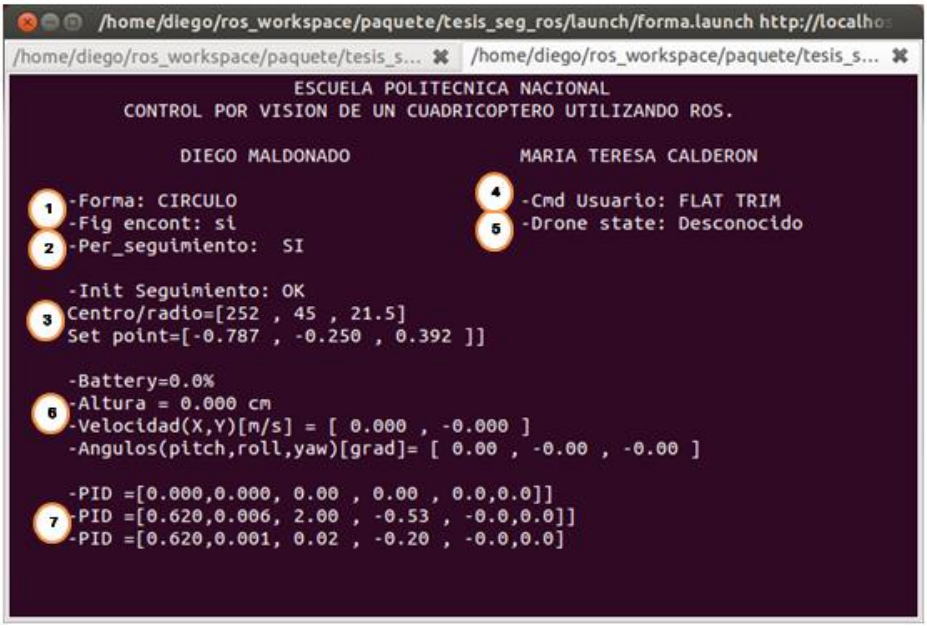


Figura. C.14 Interfaz de usuario del seguimiento de forma

1. **Forma seleccionada y encontrada**: indica la forma deseada para el seguimiento y si se la ha encontrado o no.
2. **Permiso de seguimiento:** muestra si el usuario da o no la orden para el inicio del seguimiento por parte del quadrotor al color determinado.
3. **Inicio del seguimiento:** en caso de ser positivas los anteriores anuncios, es decir si y solamente si se ha encontrado alguna forma con el color que se desea seguir y el usuario ha dado el permiso de seguimiento, entonces el ordenador comenzará a ejecutar las respectivas órdenes de control para ser enviadas al AR.Drone.

Siendo el seguimiento un hecho, se despliega la información sobre la ubicación del objeto dentro del campo visual de la cámara y el setpoint calculado para corregir el error de posición encontrado.

4. **CMD usuario:** despliega los respectivos comandos que el usuario ingresa al ordenador para control del quadrotor. En los cuales van a estar los estados: Take Off, Land, Reset, Flattrim

5. **Estado AR.Drone:** despliega el respectivo estado del AR.Drone en su operación, es decir Unknown, Inited, Landed, Flying, Hovering, Test, Taking off , Landing , Looping
6. **IMU ON BOARD**: presenta información de la IMU a bordo del AR.Drone que permite al usuario tener más conocimiento de las características de vuelo en tiempo real del AR.Drone; tales como velocidad, altura, Batería,
7. **PID:** se presenta los valores de las ganancias del modelo matemático para el controlador PID, con sus respectivos valores.

ÓRDENES DEL USUARIO

Para comandar el funcionamiento del seguimiento en cada instante, el usuario puede enviar diferentes órdenes de control desde el computador mediante el teclado, para ello se ha designado las siguientes teclas.

	Comandos desde teclado		
Comandos De vuelo	**Tecla**	**función**	**descripción**
	T	Take Off	Orden para despegar al quadrotor
	L	Land	Orden para aterrizar al quadrotor
	R	Reset	Resetea los parámetros del AR.Drone
	F	Flattrim	Calibra la IMU a bordo del AR.Drone, para condiciones iniciales de vuelo
	P	Permiso Seguimiento	Permite o no el seguimiento del quadrotor al objeto.
Selección Objeto	Q	No buscar formas	No se seguirá a ninguna forma
	W	Seguimiento a triángulos	Se buscará y seguirá solamente a formas triangulares
	A	Seguimiento a círculos	Se buscará y seguirá solamente a formas circulares
	S	Seguimiento a	Se buscará y seguirá solamente a

		Rect/Cuadrados	formas rectangulares
	D	Eliminar puntos seleccionados por flujo óptico	Deseleccionará los puntos seguidos por el flujo óptico y permitirá volver la búsqueda de nuevas características del objeto.

OBTENER DATOS DEL SISTEMA

El comando rostopic proporciona información acerca de los tópicos, y se puede acceder a las diferentes opciones que ofrece esta función con el comando:

```
$ rostopic –h
```

Donde se mostrará la ayuda de esta función.

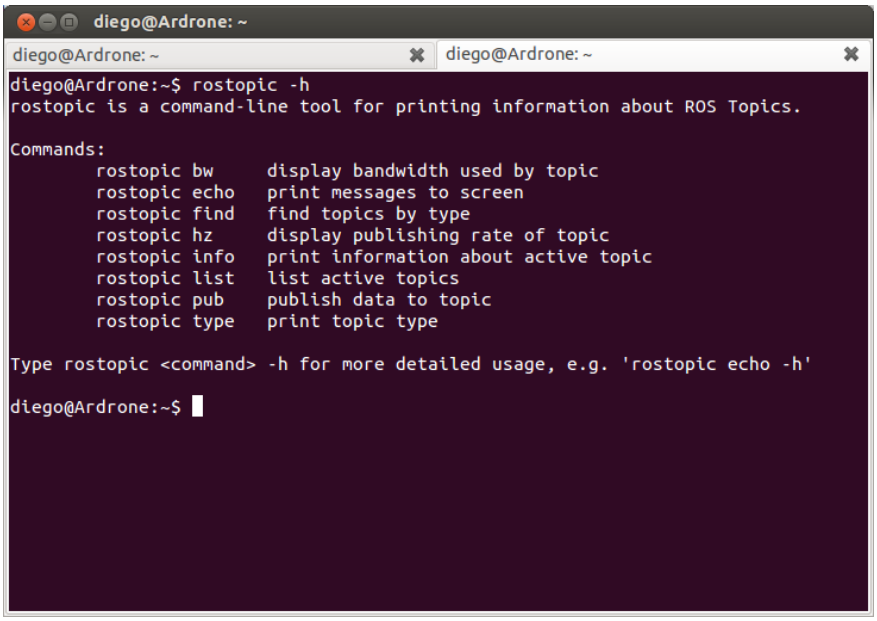


Figura. C.15 Comando rostopic -h

rostopic list

Este comando devuelve una lista de todos los tópicos a los que se está subscrito y publicando, este comando también tiene su propia ayuda, para acceder a ella solo hay que introducir en un terminal nuevo:

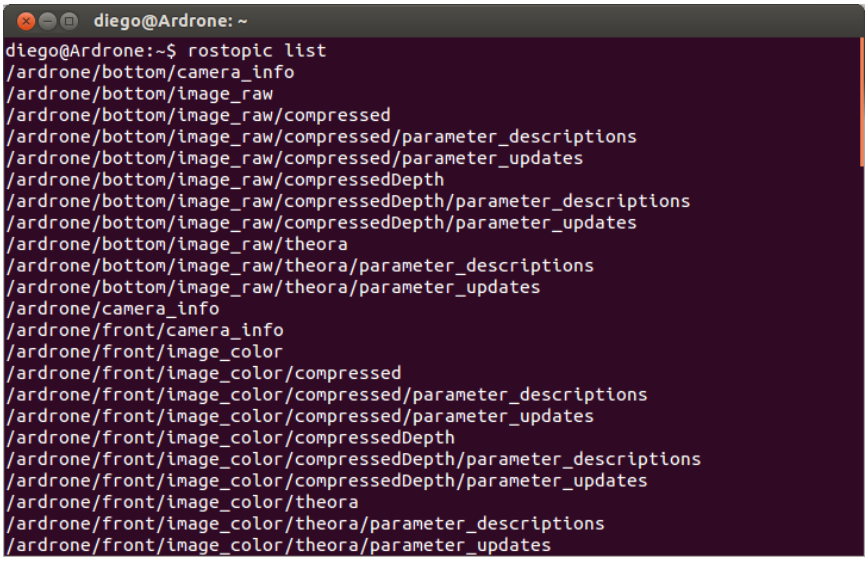


Figura. C.16 Comando Rostopic list

Para obtener más información acerca de los tópicos, de puede usar el comando rostopic info, el cual presentará información del nodo que lo esté publicando así también de los nodos que se han suscrito a él.

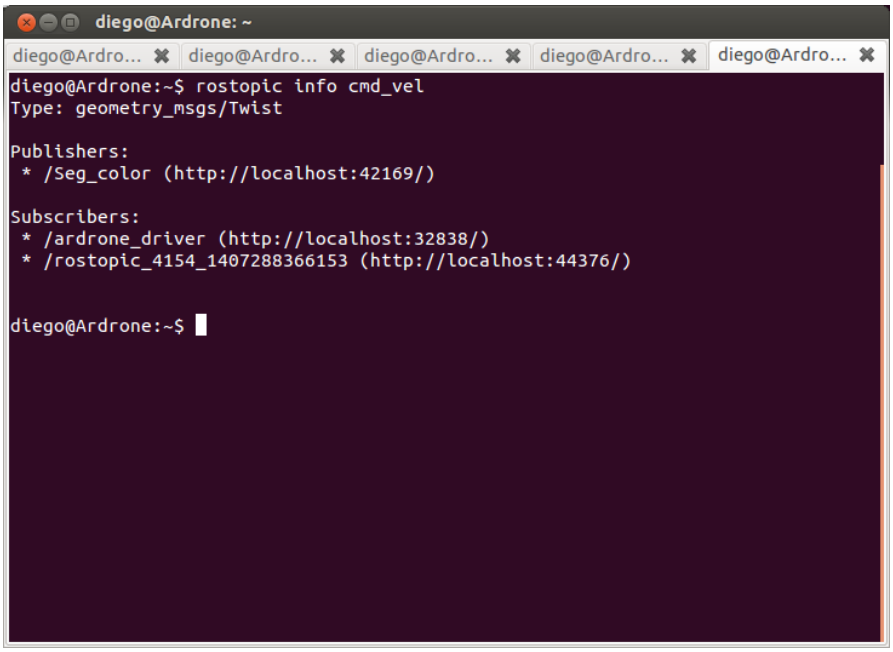


Figura. C.17 Comando rostopic info

Información específica de cada tópico se puede obtener mediante Rostopic type, y secuencialmente se puede usar la función show para desplegar elementos internos del tema.

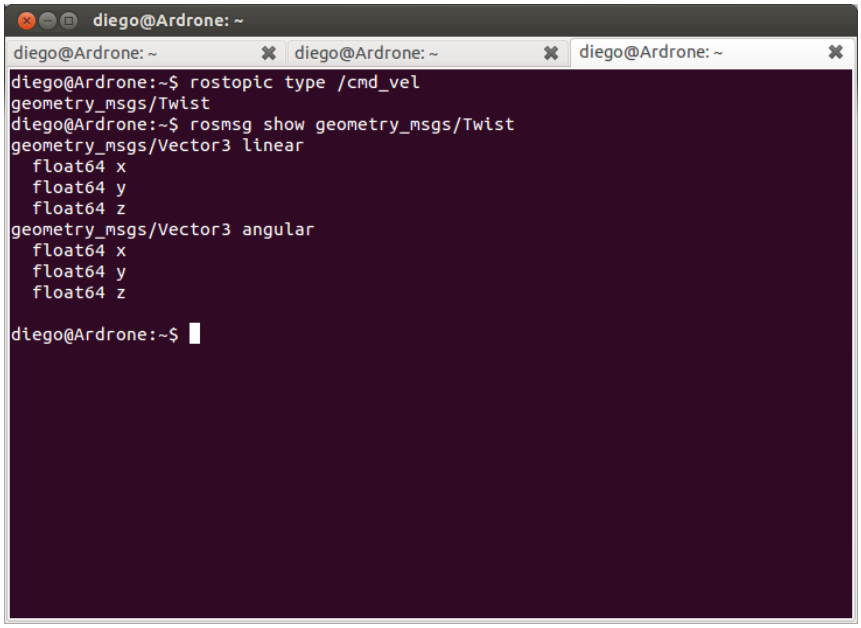


Figura. C.18 Comando Rostopic type

rostopic hz

Es el encargado de mandar la información a la frecuencia requerida

El comando tiene la forma:

```
$ rostopic hz [topic]
```

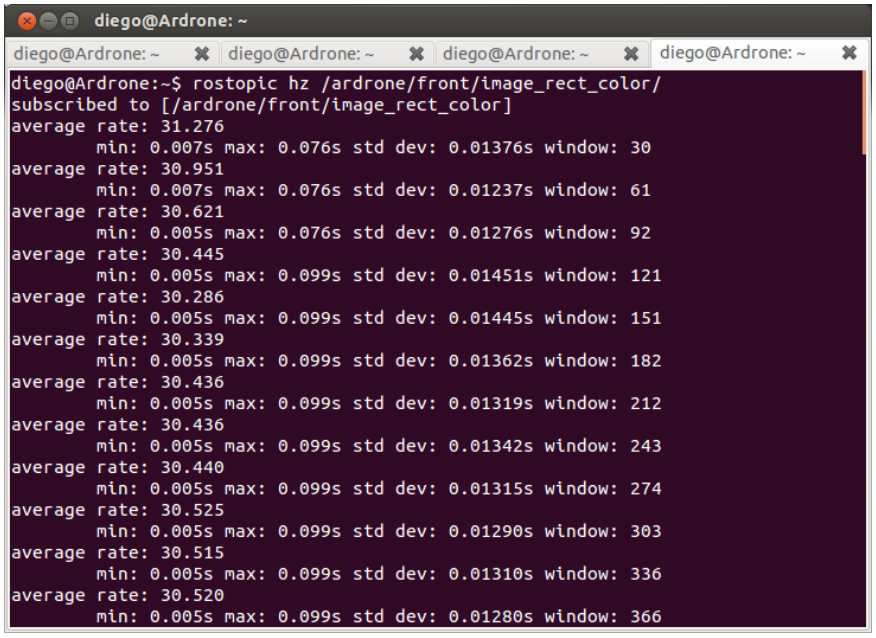


Figura. C.19 Comando rostopic hz

REPRESENTACIÓN DE LA COMUNICACIÓN EN ROS

Ros tiene herramientas y recursos que le permiten al usuario entender de forma más fácil su comunicación interna, además de presentar información relevante al usuario, para ello se pone en disposición el paquete rqt_graph encargado de graficar y presentar al usuario la comunicación en sus diferentes nodos, utilizando el comando.

```
$ rosrun rqt_graph rqt_graph
```

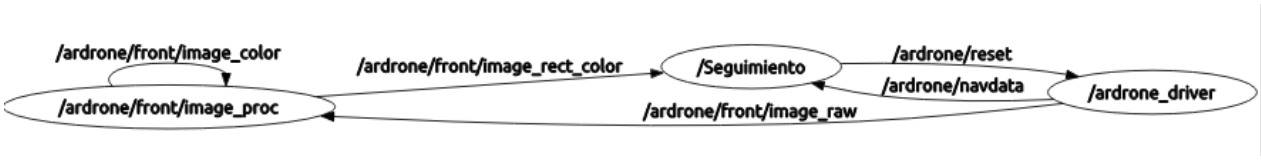


Figura. C.20 Representación gráfica de comunicación entre nodos

O de manera completa a continuación se muestra la comunicación de cada uno de los nodos con todos los tópicos usados para llevar la información respectiva en el procesamiento de imágenes y la determinación de órdenes para controlar al quadrotor.

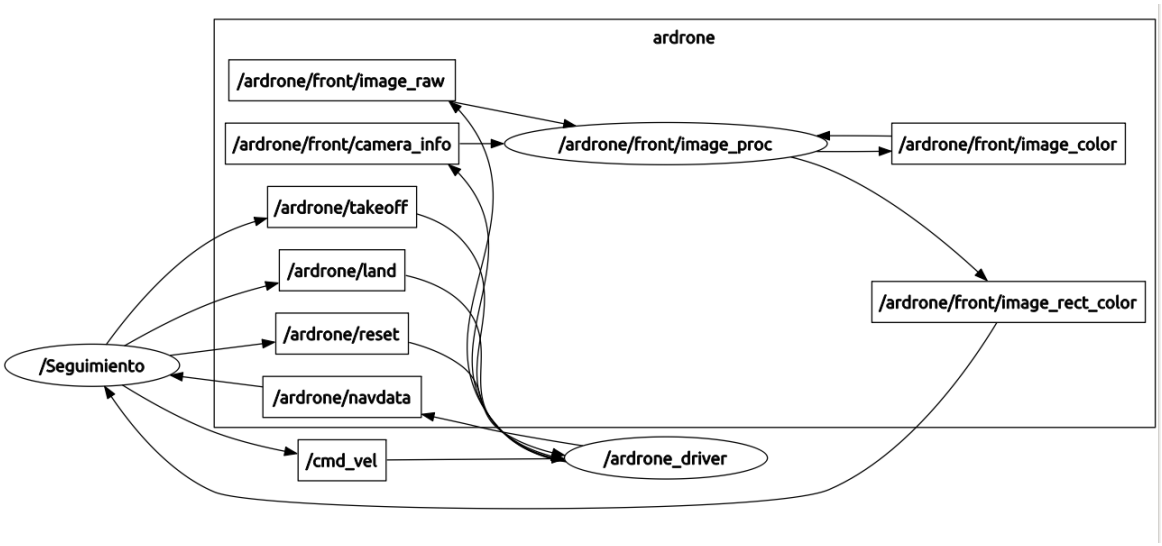


Figura. C.21 Representación Gráfica de comunicación entre nodos modo extendido

GRAFICAR DATOS CON RQT_PLOT

rqt_plot muestra un gráfico de tiempo de desplazamiento de los datos publicados sobre temas.

```
$ rosrun rqt_plot rqt_plot
```

En este caso servirá para comprender el comportamiento del quadrotor en el seguimiento al objeto

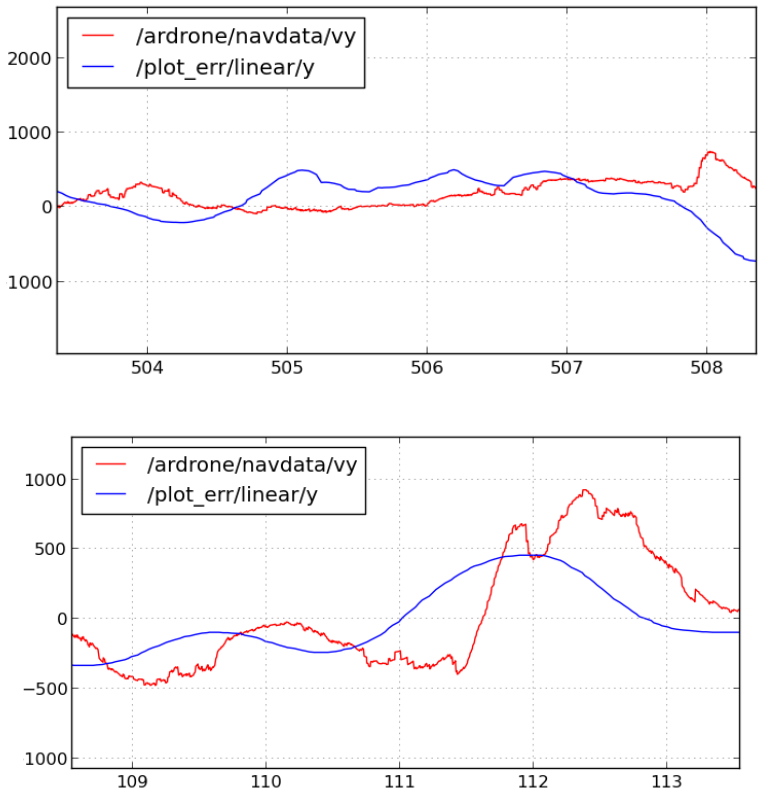


Figura. C.22 Gráficos de datos mediante rqt-plot