



**UNIVERSIDAD CATÓLICA DEL NORTE
FACULTAD DE INGENIERÍA Y CIENCIAS GEOLÓGICAS
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN**

**CONTROL DE VUELO DE UN VEHÍCULO AÉREO NO TRIPULADO A
BAJA ALTURA BASADO EN FLUJO ÓPTICO**

**TESIS PARA OPTAR POR EL GRADO DE MAGÍSTER INGENIERÍA EN
INFORMÁTICA**

**Tesista
CARLOS ANDRÉS VELÁSQUEZ OSPINA**

**Profesor Tutor
MG. RICARDO MARCIAL PEREZ SÁNCHEZ**

**Antofagasta, Chile
Mayo 30 de 2021**

Sumario

1.RESUMEN.....	7
2.DEFINICIONES ÚTILES AL ACTUAL DOCUMENTO.....	9
3.OBJETIVO DE LA TESIS.....	10
3.1OBJETIVO GENERAL.....	10
3.2OBJETIVOS ESPECÍFICOS.....	10
4.MARCO TEÓRICO.....	11
4.1ROBOT MÓVIL.....	11
4.2ROBÓTICA MÓVIL AÉREA.....	11
4.3CUADRIROTOR.....	12
4.4CONFIGURACIONES.....	13
4.5FUNDAMENTOS DE VUELO.....	17
4.5.1Throttle:.....	21
4.5.2Yaw:.....	22
4.5.3Pitch.....	22
4.5.4Roll.....	23
4.6SENSORES.....	24
4.6.1Odometría.....	24
4.6.2Central Inercial (IMU).....	24
4.6.3GPS.....	25
4.7CONTROL DE VUELO.....	26
4.7.1CONTROL DE POSTURA.....	28
4.7.2Control de.....	28
4.7.3Control de roll.....	32
4.7.4Control de yaw ()......	33
4.8CONTROL DE POSICIÓN.....	35
4.9CONTROL DE ALTURA.....	36
4.10CONTROL GENERAL DEL SISTEMA.....	38
4.11FLUJO ÓPTICO.....	40
4.12MÉTODOS PARA LA ESTIMACIÓN DEL FLUJO ÓPTICO.....	43
4.12.1MÉTODOS DIFERENCIALES.....	43
4.13ESTIMACIÓN DEL FLUJO ÓPTICO.....	45
4.13.1Estimación del movimiento basado en gradiente.....	46
4.13.2Dirección normal.....	48
4.13.3Área de regresión basada en mínimos cuadrados.....	50
4.13.4Problema de apertura.....	51
5.ESTADO DE ARTE.....	51
5.1FLUJO ÓPTICO.....	52
5.2CONTROL DE VUELO.....	54
6.HIPÓTESIS DE TRABAJO.....	59
6.1PREGUNTAS DE INVESTIGACIÓN.....	59
6.2HIPÓTESIS.....	59

7.DESARROLLO DE LA TESIS.....	61
7.1PROBLEMA A RESOLVER.....	61
7.2DESCRIPCIÓN DEL TRABAJO REALIZADO.....	61
7.3DESARROLLO DE LA CAPTURA DE FLUJO ÓPTICO.....	62
7.3.1Cálculo de desplazamiento.....	62
7.3.2Captura de desplazamiento en los ejes coordenados XY.....	63
7.4PRUEBAS DE CAPTURA DE DESPLAZAMIENTO.....	66
7.5DESARROLLO DE UN PRIMER PROTOTIPO TERRESTRE.....	66
7.5.1Objetivo de desarrollo del prototipo terrestre inicial.....	66
7.5.2Generación de pruebas de navegación.....	67
7.5.3Procedimiento.....	69
7.5.4Análisis de las pruebas.....	72
7.6PREPARACIÓN DE UN CUADRICÓPTERO PARA LA NAVEGACIÓN CON FLUJO ÓPTICO.....	73
7.6.1Descripción del cuadricóptero.....	73
7.6.2Instalación de librerías para captura de flujo óptico.....	76
7.7SOFTWARE.....	76
7.7.1Ubuntu.....	77
7.7.2ROS (Sistema Operativo Robótico).....	77
7.7.3Driver bebop_autonomy.....	80
7.8SIMULACIÓN CON SPHINX.....	81
7.9INCORPORACIÓN DE ALGORITMOS DE NAVEGACIÓN AUTÓNOMA.....	83
7.9.1Comunicación con el drone (maestro-esclavo).....	85
7.9.2Publicación de tópicos correspondientes al nodo de flujo óptico.....	88
7.9.3Envío de ordenes de navegación al drone.....	89
7.10PREPARACIÓN DE PRUEBAS FINALES DE NAVEGACIÓN.....	91
7.10.1Prueba de comandos.....	93
8.ANÁLISIS DE RESULTADOS Y CONCLUSIONES DE LA TESIS.....	97
8.1RESULTADOS.....	97
8.2CONCLUSIONES.....	100
9.PLAN DE ACTIVIDADES.....	102

Índice de figuras

Figura 1: Configuración en X, tomado de Luque (2014) [CITATION Luq14 \l 3082]8...	14
Figura 2: Configuración en +, tomado de Luque (2014).....	14
Figura 3: Configuración Y4, tomado de http://zawiki.praxis-arbor.ch/doku.php/tschinz:multicopter (2016).....	15
Figura 4: Configuración VTail, tomado de http://www.lynxmotion.com/p-896-hunter-vtail-500-quadcopter-base-combo-kit.aspx (2016).....	16
Figura 5: Ángulos de Euler, tomado de Luque (2014).....	18
Figura 6: Giro en ángulo Yaw, tomado de Luque (2014).....	19
Figura 7: Giro en ángulo Pitch, tomado de Luque (2014).....	19
Figura 8: Giro en ángulo Roll, tomado de Luque (2014).....	20
Figura 9: Convención Potencia de giro, tomado de https://3dr.com/kb/ (2016).....	21
Figura 10: Movimiento ascendente, tomado de https://3dr.com/kb/ (2016).....	21
Figura 11: Movimiento descendente, tomado de https://3dr.com/kb/ (2016).....	21
Figura 12: Giro sentido horario, tomado de https://3dr.com/kb/ (2016).....	22
Figura 13: Giro sentido anti-horario, tomado de https://3dr.com/kb/ (2016).....	22
Figura 14: Movimiento hacia adelante, tomado de https://3dr.com/kb/ (2016).....	22
Figura 15: Movimiento hacia atrás, tomado de https://3dr.com/kb/ (2016).....	23
Figura 16: Movimiento hacia la derecha, tomado de https://3dr.com/kb/ (2016).....	23
Figura 17: Movimiento hacia la izquierda, tomado de https://3dr.com/kb/ (2016).....	23
Figura 18: Sistema de GPS para un UAV, tomado de Campo (2014).....	26
Figura 19: Diagrama de control de vuelo, tomado de Pico (2012).....	27
Figura 20: Diagrama del control de postura del cuadricóptero.....	28
Figura 21: Ángulos determinan la postura de cuadricóptero, tomado de Pico (2012).....	29
Figura 22: Diagrama del controlador de pitch, tomado de Pico (2012).....	30
Figura 23: Diagrama de control PID de roll, tomado de Pico (2012).....	32
Figura 24: Diagrama del controlador PID de yaw, tomado de Pico (2012).....	34
Figura 25: Diagrama de control de posición, tomado de Pico (2012).....	36
Figura 26: Diagrama del controlador PID de altura, tomado de Pico (2012).....	38
Figura 27: Diagrama del control general del sistema, tomado de Pico (2012).....	40
Figura 28: Vista de perfil de las partes involucradas en el sistema de un sensor de flujo óptico, tomado de Tresanchez (2011).....	41
Figura 29: Esquema básico de funcionamiento del sensor de flujo óptico, tomado de Tresanchez (2011).....	42
Figura 30: Estimación de señal de primer orden exacto, tomado de Riascos (2015).....	47
Figura 31: Estimación de señal no lineal de orden superior, tomado de Riascos (2015).....	48
Figura 32: Restricción de gradiente de la velocidad en la dirección normal, tomado de Riascos (2015) [18].....	49
Figura 33: Línea de restricción de movimiento, tomado de Riascos (2015).....	49

Figura 34: Problema de apertura, tomado de Riascos (2015) [CITATION Ria15 \l 3082]18	51
Figura 35: Sensor de Flujo Óptico PX4Flow.....	63
Figure 36: Representación gráfica del algoritmo SAD.....	65
Figura 37: Escenario de prueba de navegación terrestre.....	67
Figura 38: Robot Parallax junto al sensor PX4Flow.....	68
Figura 39: Robot diferencial PARALLAX.....	69
Figura 40: Trayectoria vista superior.....	71
Figura 41: Trayectoria XY realizada.....	72
Figura 42: Cuadricóptero Bebop 2 Power, configuración X, tomado de https://www.parrot.com/es/drones/parrot-bebop-2-power/piezas-repuesto (2019).....	74
Figura 43: Raspberry Pi Zero W.....	75
Figure 44: Terminal Linux con los tópicos publicados disponibles para drone.....	80
Figure 45: Entorno de simulación Gazebo- para el drone Bebop2 Parrot-Sphinx.....	83
Figure 46: Movimientos generados por los parámetros de control del drone [44].....	85
Figure 47: Comunicación esclavo-maestro (Bebop 2- Laptop Linux(ROS)).....	86
Figure 48: Parámetros de configuración en archivo config.yaml.....	87
Figure 49: Esquema de control PID para el drone Bebop 2.....	91
Figure 50: Entorno simulado para el entorno Gazebo y el drone Bebop 2.....	92
Figure 51: Interfaz gráfica para el control manual del movimiento del drone.....	93
Figure 52: Sphinx-Dashboard.....	94
Figure 53 Desplazamiento en [m] en función del tiempo. Leyenda: posición en Z (color verde), posición en X (color azul), posición en Y (color purpura).....	95
Figure 54: Desplazamiento en [m] en función del tiempo. Leyenda: posición en X (color azul), posición en Y (color purpura), posición en Z (color verde).....	96
Figure 55: Desplazamiento producido en las coordenadas XYZ.....	98
Figure 56: Trayectoria cuadrada, plano XY, 4 x 4 metros.....	98
Figure 57: Desplazamiento producido en el eje Z.....	99
Figure 58: Desplazamiento del drone en las coordenadas XYZ.....	99

Índice de tablas

Tabla 1: Velocidad vs distancia focal.....	64
Tabla 2: Tópicos principales para el control de movimiento del drone.....	83
Tabla 3: Efectos de las constantes de control respecto a la acción de control.....	91

1. RESUMEN

Una amplia variedad de técnicas para la navegación visual utilizando sistemas de visión en robots han sido desarrolladas en los últimos años. Estas técnicas actualmente están siendo utilizadas en los vehículos aéreos autónomos con el propósito de incrementar su autonomía y su adaptabilidad para desarrollar diversas tareas.

Los últimos años han sido testigos de un gran interés en el desarrollo de los vehículos aéreos autónomos (Autonomous Unmanned Aerial Vehicles UAV). Este interés es debido al hecho de que estos vehículos tienen aplicaciones civiles y militares de gran impacto, como operaciones de investigación, de rescate, de vigilancia aérea, de inspección y monitoreo ambiental, entre otras.

Las aplicaciones mencionadas anteriormente requieren de robots aéreos con habilidad y autonomía para operar en ambientes desconocidos, hostiles o que no están bien estructurados geométricamente en donde la señal GPS (Global Positioning System) no esté disponible. En esta propuesta de anteproyecto de tesis se plantea desarrollar una odometría de bajo costo para un vehículo aéreo no tripulado para estimar el desplazamiento espacial de éste con respecto al suelo.

La odometría en un robot es el conjunto de medidas obtenidas desde sus sensores internos, que le permiten determinar sus desplazamientos en el plano o en el espacio. En los robots terrestres la odometría viene dada específicamente por los encoders que tiene asociado a cada rueda y que en definitiva cuentan los pasos de cada una de ellas y a través de esta información se deduce el desplazamiento en el plano XY del robot.

En los robots aéreos, la odometría se refiere al conjunto de medidas de los sensores internos (giroscopio, compás, barómetro), que permiten estimar los desplazamientos del robot, esta odometría junto a la información entregada por un GPS interno, permite el cálculo de la posición espacial del robot. El barómetro que es parte de la odometría del robot aporta con la medidas de altitud, que señalan la distancia que existe entre el nivel del mar y la posición del robot. Esta medida no entrega información de la distancia desde el robot hasta el real nivel del suelo, por esta razón los cambios en el nivel del relieve del suelo no son detectado por el barómetro, por esto, se ha decidido intervenir la odometría del robot incorporando a ella un sensor de flujo óptico, el cual si permite determinar variaciones en el nivel del suelo, cuando el vuelo es a baja altura (menos de 3 metros).

La propuesta se enfoca en el uso del flujo óptico para estimar la altura de un vehículo aéreo autónomo con respecto al piso para estabilizar su vuelo y de esta forma lograr que este se desplace a baja altura de manera controlada. Las cámaras son transductores muy útiles que proveen la suficiente información alrededor del medio en que se desenvuelve un robot. La mejora en la calidad digital de la imagen y el incremento en el desempeño del análisis y procesamiento de imágenes son tales que, actualmente, la visión por computadora es una herramienta adecuada para estimar parámetros con respecto a la posición de un vehículo aéreo autónomo.

En este documento se reportan los desarrollos que se han llevado a cabo en cada una de las áreas del saber que se aplican en la elaboración de esta tesis, además se describen los materiales a utilizar y se plantea una metodología de trabajo con un cronograma estimado.

2. DEFINICIONES ÚTILES AL ACTUAL DOCUMENTO

Ambiente controlado: Es un entorno al aire libre, no urbano, con variaciones de relieves lo suficiente para no convertirse en obstáculos de la trayectoria.


Obstáculo: Variaciones del relieve del terreno con pendiente de sus paredes, superiores a las posibles de detectar, por sensores dispuestos en la zona inferior del vehículo. Esto porque lo que se desea resolver es el vuelo autónomo controlado observando la superficie del terreno y no la evasión de obstáculos emergentes estáticos o móviles.

Trayectoria o vuelo de baja altura: Es el vuelo controlado, a una distancia no superior 1,5 metros constante, desde el nivel del terreno.

Ángulos de navegación: Son un tipo de ángulos de Euler usados para describir la orientación de un objeto en tres dimensiones. Dado un sistema coordenadas de tres ejes fijos XYZ, ortogonales entre sí, definido en un vehículo aéreo, en donde el eje X se extiende hacia la parte delantera de la nave, el eje Y se extiende por el lado derecho y el eje Z se extiende por la parte inferior, existen tres rotaciones principales sobre cada uno de estos ejes conocidas como Yaw, Pitch y Roll.

Yaw: También se conoce como guiñada, está definido por el ángulo de giro ψ , éste se refiere al movimiento cuando el vehículo gira sobre su eje vertical Z.

Pitch: También se conoce como cabeceo, está definido por el ángulo de giro θ , éste se refiere al movimiento cuando el vehículo gira sobre su eje vertical Y.

Roll: También se conoce como cabeceo, está definido por el ángulo de giro ϕ , éste se refiere al movimiento cuando el vehículo gira sobre su eje vertical X.

3. OBJETIVO DE LA TESIS

3.1 OBJETIVO GENERAL

Dotar un UAV con capacidad de vuelo a baja altura siguiendo una trayectoria, considerando el relieve del terreno por donde se desplaza, en base a un sensor de flujo óptico.

Restricciones:

- No considera evasión de obstáculos.
- Entorno controlado

3.2 OBJETIVOS ESPECÍFICOS

- Obtener información útil del sensor de flujo óptico para incorporarlos a los algoritmos de control de vuelo a baja altura.
- Estudiar los algoritmos típicos de control de vuelo y de seguimiento de trayectoria de un vehículo aéreo no tripulado.
- Definir un algoritmo de navegación, que permita al vehículo aéreo no tripulado volar a baja altura de manera autónoma (modificando uno ya existente o creando uno nuevo).
- Establecer los experimentos que permitan lograr comprobar el cumplimiento de los objetivos planteados.

4. MARCO TEÓRICO

4.1 ROBOT MÓVIL

Un robot móvil es una máquina automática que es capaz de trasladarse en cualquier ambiente dado. Los robots móviles tienen la capacidad de moverse en su entorno y no se fijan a una ubicación física [1].

La autonomía de un robot móvil se basa en el sistema de navegación autónoma, en estos sistemas se incluyen tareas de planificación, percepción y control. El problema de planificación puede descomponerse en planificación global de la misión, de la ruta, de la trayectoria y finalmente evitar obstáculos no esperados [2].

Los robots móviles son un foco importante de la investigación actual y casi cada universidad importante que tenga uno o más laboratorios que se centran en la investigación de robots móviles [3]. Los robots móviles se encuentran también en la industria y los servicios. Los robots domésticos ya son productos de consumo que realizan ciertas tareas del hogar, como pasar la aspiradora o realizar jardinería.

4.2 ROBÓTICA MÓVIL AÉREA

Los robots aéreos son aeronaves no tripuladas como helicópteros o pequeños aviones operados a control remoto que pueden proporcionar imágenes aéreas para reconocimiento

de terreno y superficie, son muy útiles en problemas de análisis de tráfico e inspección de edificios [4].

También la demanda de las aplicaciones tales como recolección de datos, mantenimiento de instalaciones en ambientes naturales a los que al ser humano se le dificulta llegar y probablemente hasta le sea imposible, ha llevado al hombre a desarrollar vehículos aéreos o submarinos. Dichos robots resultan de la evolución de vehículos totalmente tele-operados por el ser humano. Un primer problema con el que se puede topar en el desarrollo de este tipo de robots es la necesidad de obtener información de forma continua, que es el caso de los robots móviles. En este caso el robot no puede detenerse, forzosamente tiene que recibir datos de su movimiento en el espacio, regular su velocidad y posicionar sus actuadores en forma simultánea. Comúnmente los robots móviles utilizan un Sistema de Posicionamiento Global (GPS), acelerómetros, giróscopos, telémetros y otros sensores sofisticados para ubicarse en el espacio tridimensional. Una consideración importante es tener información continua en tierra de la carga de la batería, con el objeto de activar algún mecanismo automático de aterrizaje seguro en caso de una baja de energía [5].

Estos robots contienen un sistema redundante de sensores para su posicionamiento, de forma tal que ante cualquier interrupción momentánea de uno o más de ellos, pueda recuperarse de la falla. Este sistema de redundancia permite mayor fiabilidad y precisión en la lectura de datos [4].

4.3 CUADRIROTOR

Un Cuadricóptero o Cuadrirotor, es una aeronave de ala giratoria, propulsada por cuatro rotores. Normalmente éstos se encuentran colocados simétricamente, formando cada uno un vértice de un cuadrado imaginario. Al ser un tipo de helicóptero, tanto la sustentación como

la propulsión se basan en el aire impulsado por las hélices de sus rotores. En cambio, a diferencia de los helicópteros que varían el ángulo de incidencia (inclinación) de las palas de las hélices para maniobrar (tanto las palas del rotor principal como las del rotor de cola), los cuadricópteros (como los demás multicopteros modernos) basan sus maniobras en el cambio de revoluciones de sus distintos motores, usando así hélices con palas de paso fijo que simplifican en gran medida la mecánica de este tipo de aeronaves [6] [7].

4.4 CONFIGURACIONES

Mayormente, hay dos tipos o configuraciones de cuadricópteros, los de tipo x (en aspa o equis) (Figura 1) y los de tipo + (en cruz) (Figura2). Estas denominaciones se deben a lo que considera como la parte delantera de la aeronave [6] [8].

En un cuadricóptero configurado en X, se tienen dos rotores delanteros y dos rotores traseros (y redundantemente, dos rotores izquierdos y dos rotores derechos). Por hacer una analogía se puede comparar con el posicionamiento de las ruedas de un automóvil. Este tipo de montaje es más popular para llevar cámaras en ellos, pues la cámara no tiene ningún brazo en su campo de visión (contando que las cámaras suelen apuntar hacia adelante, aunque hay algunas rotatorias) [8] [6].

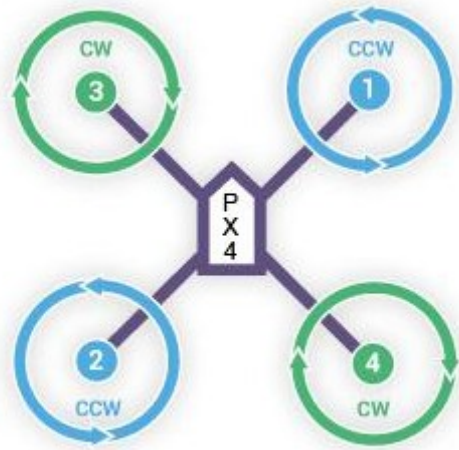


Figura 1: Configuración en X, tomado de Luque (2014) [CITATION Luq14 \l 3082]8

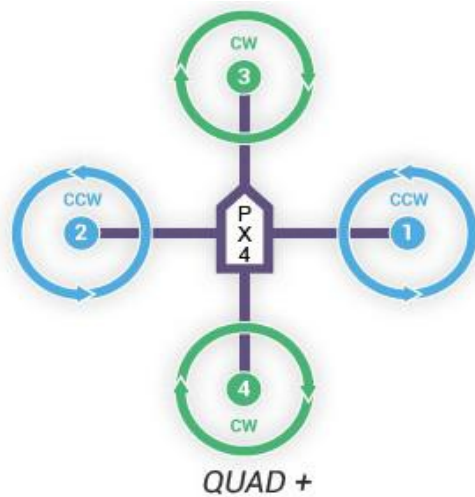


Figura 2: Configuración en +, tomado de Luque (2014)

Por otra parte, la configuración en + conlleva tener un rotor delantero, uno trasero, uno izquierdo y uno derecho. Se puede comparar este tipo de configuración con las partes de un avión siendo el rotor delantero la cabeza, el rotor trasero la cola y los rotores laterales las alas del avión. Este tipo de montaje es más popular entre los aficionados al aeromodelismo y las acrobacias, sobre todo para los que tienen aviones de radiocontrol por su similitud con éstos, y por ser más sencillo de programar y entender [8] [7] [6].

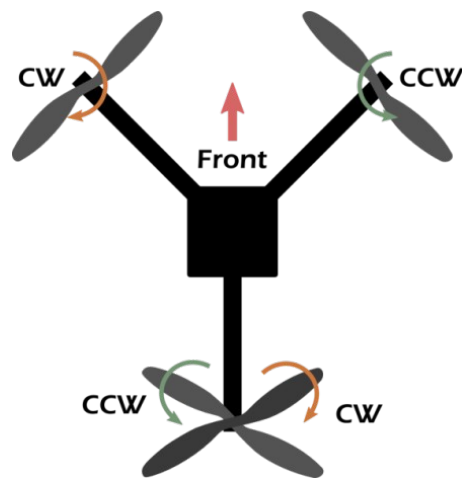


Figura 3: Configuración Y4, tomado de <http://zawiki.praxis-arbor.ch/doku.php/tschinz:multicopter> (2016)

Últimamente, han aparecido otros dos tipos de cuadricópteros, cuyos rotores no están colocados formando un cuadrado, los del tipo Y4 (Figura 3) y los VTail (Figura 4). Por definición son cuadricópteros (son aeronaves de ala móvil y propulsadas por cuatro

rotores), pero su comportamiento y forma es similar al de los tricópteros (multicópteros de 3 rotores), aunque aumentan la estabilidad de estos últimos gracias a tener un número par de rotores [7] [8].

La configuración Y4 consiste en una estructura en forma de Y, teniendo en su parte delantera dos rotores (uno en cada punta de los brazos de la Y) y otros dos rotores coaxiales en la cola (dos rotores sobre el mismo eje vertical, uno encima del otro). Esta forma es una evolución de los tricópteros convencionales, pero evitando el servo necesario para el giro sobre el eje vertical (guiñada o *yawing*), usando los dos rotores coaxiales, girando en dirección contraria el uno del otro. Con esto se logra alrededor de 1/3 más de fuerza de sustentación (pues el peso no varía mucho al ser la estructura similar al tricóptero), y una mayor estabilidad y fiabilidad, pues no se depende de ningún servo [8][6][7].



Figura 4: Configuración VTail, tomado de <http://www.lynxmotion.com/p-896-hunter-vtail-500-quadcopter-base-combo-kit.aspx> (2016)

La configuración VTail corresponde a una mezcla entre los cuadricópteros en X y los tricópteros. Como los cuadricópteros, tiene dos rotores delanteros y dos traseros, pero estando los delanteros colocados cada uno sobre un brazo (como en la configuración X

convencional), los traseros se encuentran sobre un único brazo trasero (a modo de cola) colocado uno a cada lado de su extremo, he inclinados alrededor de él en un ángulo determinado. Esta configuración no es muy popular, debido a que es menos eficiente que la Y4 (puesto que el aire propulsado por los rotores de cola no se entrecruza y no se destina todo a la sustentación debido a su inclinación) y a que su control de motores es algo más complicado. Pero por otra parte, es una configuración más estable (gracias a la parte de empuje lateral de sus rotores de cola), y su orientación es más reconocible a distancia gracias a su clara distinción entre cabeza y cola [7][6][8].

En este proyecto se utilizará una configuración en X, debido a que su comportamiento es más estable y se adapta mejor a vuelos bajos y lentos, considerando además que el cuadricóptero que se dispone permite un control de vuelo para una configuración en X.

4.5 FUNDAMENTOS DE VUELO

El vuelo de los cuadricópteros se basa en la sustentación obtenida mediante el impulso del aire por sus hélices, y en las maniobras logradas mediante el cambio de velocidad de sus rotores. Podemos ver los cuadricópteros como máquinas voladoras con 6 Grados de Libertad, ya que se pueden desplazar y rotar sobre los 3 ejes del espacio tridimensional [8].

Aunque esto sea así, en un instante de tiempo dado, sólo se puede indicar al cuadricóptero 4 movimientos diferentes. Como toda aeronave, para conocer su orientación espacial sólo necesitamos tres valores, los Ángulos de Euler mostrados en la Figura 5 (o ángulos de vuelo) [8].

Estos ángulos son la diferencia de orientación que hay entre el sistema de referencia de tierra (eje X apuntando al Norte, eje Y al Este y eje Z hacia abajo) o *NED*, y el sistema de

referencia de vuelo (eje X apuntando a la cabeza del aeronave, eje Y a la derecha, y eje Z, perpendicular a ambos, apuntando hacia la parte de abajo del aeronave) [8].

Para entender los denominados ángulos de vuelo, se supone una situación inicial en la que los dos sistemas de referencia (tierra y vuelo) estas alineados [8].

El ángulo Yaw (ψ) (Figura 6) es el ángulo girado sobre el eje Z (de ambos sistemas pues están alineados), logrando girar los ejes X e Y del sistema de referencia de vuelo respecto al de tierra. Esto lleva a tener los ejes X_{v1} e Y_{v1} [8].

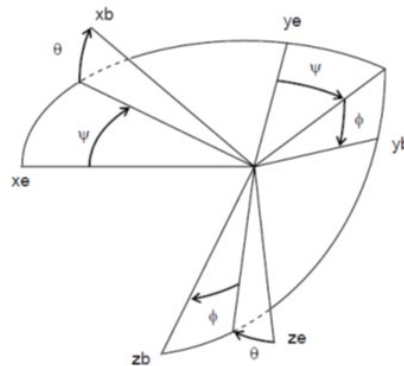


Figura 5: Ángulos de Euler, tomado de Luque (2014).

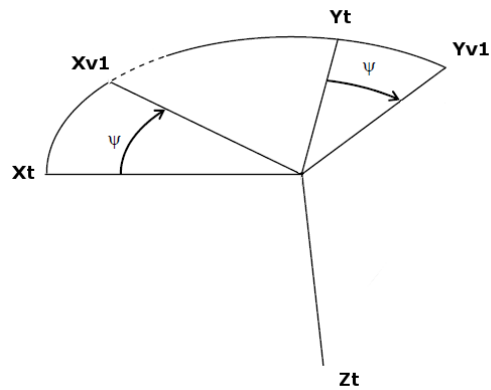


Figura 6: Giro en ángulo Yaw, tomado de Luque (2014)

Ahora se procede a girar sobre el eje Y_{v1} , dando como resultado un determinado Pitch (θ) (Figura 7) y logrando girar los ejes X y Z del vuelo, dando como resultado los ejes X_{v2} y Z_{v2} .

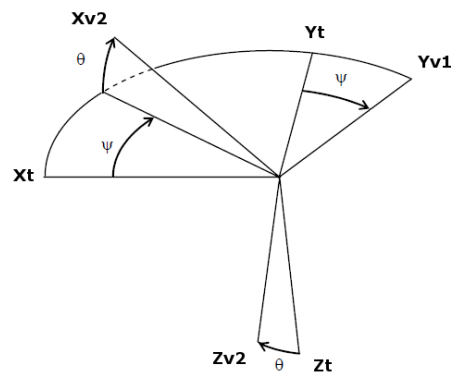

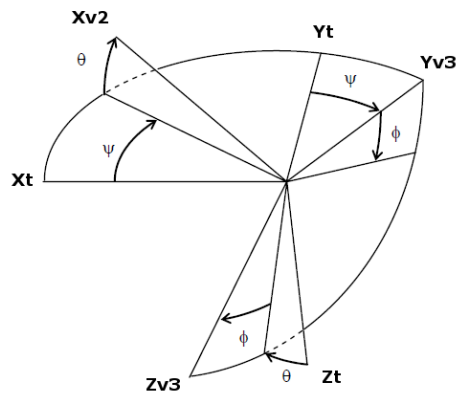


Figura 7: Giro en ángulo Pitch, tomado de Luque (2014)

Por último giramos alrededor del eje X_{v2} dando como resultado un determinado Roll (ϕ) (Figura 8), y los ejes Y_{v3} y z_{v3} . 



*Figura 8: Giro en ángulo Roll,
tomado de Luque (2014)*

Para el control del movimiento se debe controlar individualmente cada motor, para que, en su conjunto, se logre el movimiento esperado.

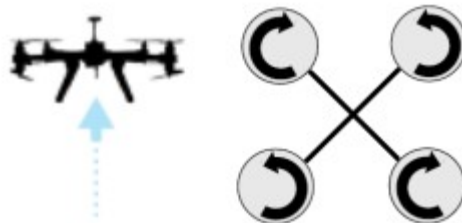
Para poder entender los posibles movimientos de los motores, se definirán éstos de acuerdo a la convención mostrada en la Figura 9.



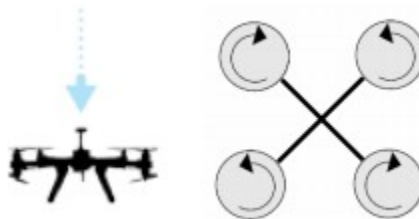
*Figura 9:
Convención Potencia
de giro, tomado de
<https://3dr.com/kb/>
(2016)*

Las cuatro maniobras básicas de un cuadricóptero que se puedan realizar de acuerdo al movimiento de sus motores, son mostrados en la Figuras 10 hasta la 17:

4.5.1 Throttle:



*Figura 10: Movimiento ascendente,
tc*



*Figura 11: Movimiento descendente,
tomado de <https://3dr.com/kb/> (2016)*

4.5.2 Yaw:

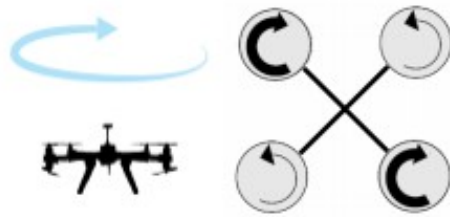


Figura 12: Giro sentido horario, tomado de <https://3dr.com/kb/> (2016)

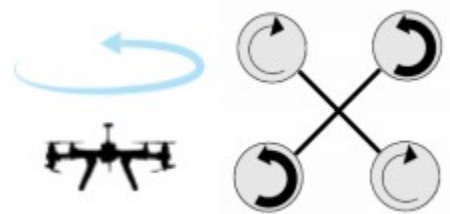


Figura 13: Giro sentido anti-horario, tomado de <https://3dr.com/kb/> (2016)

4.5.3 Pitch

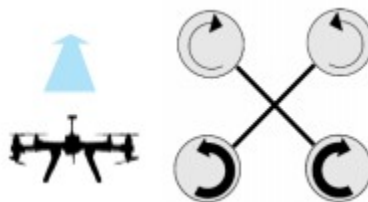


Figura 14: Movimiento hacia adelante, tomado de <https://3dr.com/kb/> (2016)

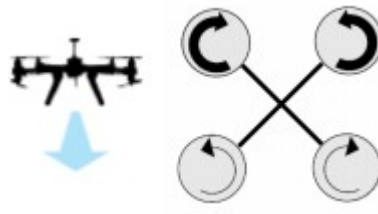


Figura 15: Movimiento hacia atrás, tomado de <https://3dr.com/kb/> (2016)

4.5.4 Roll

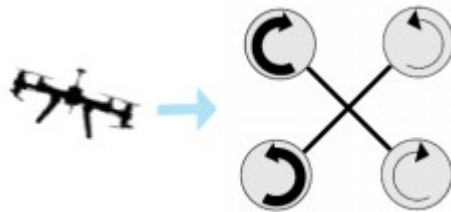


Figura 16: Movimiento hacia la derecha, tomado de <https://3dr.com/kb/> (2016)

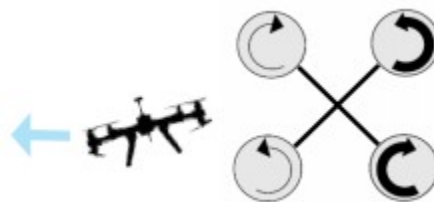


Figura 17: Movimiento hacia la izquierda, tomado de <https://3dr.com/kb/> (2016)

4.6 SENSORES

Una aeronave capaz de realizar vuelos autónomos con una mínima intervención del piloto necesita incluir algunos sensores que le ayuden a mantener un vuelo estable y seguro.

4.6.1 Odometría

Odometría viene del griego *hodos* que significa viaje y *metron* que significa medición. La odometría se ocupa de estimar el cambio en la posición y orientación en el tiempo de un robot o vehículo usando la información proporcionada por sensores de movimiento. En particular, para este proyecto los datos que se usarán para hacer dicha estimación serán proporcionados por una sola cámara y algunos sensores adicionales de su odometría [9].

Es importante mencionar que la odometría es sensible a múltiples tipos de errores y por esta característica, se sugiere que no debe utilizarse como único método de localización, pues el error generado por la inexactitud de las medidas es acumulativo, por lo tanto, después de cierta cantidad de tiempo sin realizar ningún tipo de corrección, la estimación del movimiento de la odometría puede variar considerablemente del movimiento real. Por esta última razón la odometría se ha utilizado como un método complementario para apoyar otras estimaciones de posición como las brindadas por el GPS [9] [10].

4.6.2 Central Inercial (IMU)

Se denomina Central Inercial (IMU) al conjunto de sensores que proporcionan la medición de la posición angular del vehículo (ϕ, θ, ψ), así como su velocidad angular (ϕ', θ', ψ'), todo esto es posible gracias a los acelerómetros y girómetros, respectivamente, que conforman a la central inercial (IMU) [7].

Los acelerómetros, son sensores que miden la aceleración lineal en una, dos o tres dimensiones, esto es, en tres direcciones del espacio orto-normal, es decir, en tres direcciones perpendiculares entre sí. Esta característica permite medir la inclinación de un cuerpo, puesto que es posible determinar con el acelerómetro la componente de la aceleración provocada por la gravedad que actúa sobre el cuerpo. Un acelerómetro también es usado para determinar la posición angular de un cuerpo, a partir de la medición de su aceleración. Por otro lado, el girómetro, es un dispositivo formado esencialmente por un cuerpo con simetría de rotación que gira alrededor de su eje de simetría. Cuando se somete el giroscopio a un momento de fuerza que tiende a cambiar la orientación del eje de rotación. Su eje de rotación, en lugar de cambiar de dirección como lo haría un cuerpo que no girase, cambia de orientación en una dirección perpendicular a la dirección intuitiva. Con los girómetros podemos medir la rapidez con la que cambia un objeto y poder así conocer la velocidad angular del objeto [7][11].

4.6.3 GPS

Un GPS, Sistema de Posicionamiento Global (Global Position System), integrado en algún dispositivo, calcula la posición de dicho dispositivo tomando el tiempo de las señales enviadas por un conjunto de satélites que encuentran orbitando alrededor del planeta. Los satélites envían continuamente mensajes que los dispositivos receptor GPS reciben, estos mensajes contienen datos sobre el tiempo en el cual el mensaje fue transmitido y la posición del satélite al momento de transmitir el mensaje [12][13].

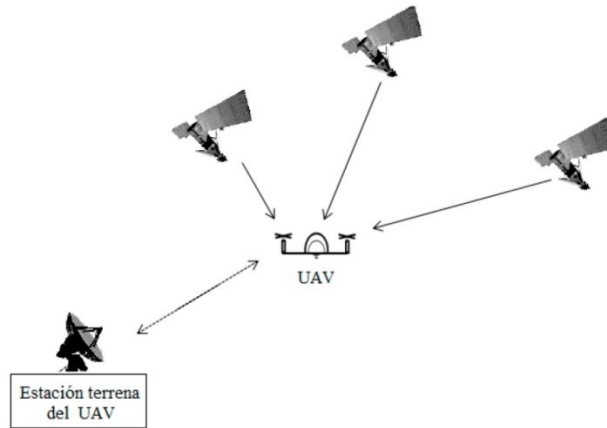


Figura 18: Sistema de GPS para un UAV, tomado de Campo (2014)

El dispositivo receptor GPS recibe los mensajes que los satélites están enviando constantemente, calcula cuánto fue el tiempo que tardó en llegar el mensaje y luego calcula la distancia del dispositivo a cada satélite. Posteriormente, con la información de las posiciones de los satélites, las distancias del dispositivo hacia ellos y con la ayuda de la trilateración (Figura 18), el dispositivo puede computar la posición aproximada en donde se encuentra. La posición se puede desplegar en el formato de latitud o longitud, o en un mapa digital que referencie la ubicación del dispositivo.

4.7 CONTROL DE VUELO

El movimiento de un cuadricóptero tiene seis grados de libertad, definidos por el vector $q = (x, y, z, \theta, \phi, \psi) \in \mathbb{R}^6$ donde $p = (x, y, z) \in \mathbb{R}^3$ denota la posición del centro de masa del vehículo con respecto a un eje de referencia en tierra y $\alpha = (\theta, \phi, \psi) \in \mathbb{R}^3$ son los ángulos de Euler *pitch*, *roll* y *yaw* que representan la postura [13].

Un diagrama general del control en lazo cerrado del cuadricóptero se muestra en la Figura 19.

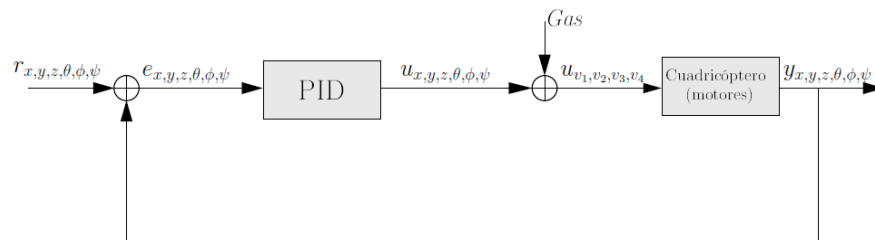


Figura 19: Diagrama de control de vuelo, tomado de Pico (2012)

La señal de referencia $r_{x,y,z,\theta,\phi,\psi}$ indica la postura y posición deseada del vehículo, mientras que $y_{x,y,z,\theta,\phi,\psi}$ es la variable manipulada, que es la posición y postura reales medidas. La señal de error $e_{x,y,z,\theta,\phi,\psi}$ es la diferencia entre la posición y postura deseada menos las reales $e=r-y$. El proceso controlador es un control (controles) PID cuya variable de control $u_{x,y,z,\theta,\phi,\psi}$ es sumada a la variable Gas (valores iniciales de potencia), obteniendo $u_{v1,v2,v3,v4}$ que sean las señales actuantes aplicadas a cada motor

xxx.



La variable Gas, es un valor de referencia que indica la velocidad base utilizada por los motores. Esta se aplica con la misma magnitud en cada motor y, a partir de ese valor, las señales actuantes de los controladores PID son sumadas (o restadas) [14].

En general, el control total del cuadricóptero se divide en 5 controladores, 3 controladores PID para el control de postura, dejando encargado de a un PID por cada uno de los ángulos de Euler *pitch*, *roll* y *yaw*, un control PID que se encargue de la variable altura “z”, y un control PID que corrige la posición bidimensional en los ejes “x” e “y” [15].

4.7.1 CONTROL DE POSTURA

El control automático de postura se subdivide en tres controladores PID: uno para el *pitch*, uno para el *roll* y otro para el *yaw*. La Figura 20 muestra el diagrama de control automático de postura, donde se toman como referencia r los ángulos (θ, ϕ, ψ) deseados, que por medio de la retroalimentación “y” se obtienen los errores e_θ, e_ϕ, e_ψ . Cada señal de error es tomada como entrada de cada controlador PID del ángulo correspondiente, obteniéndose así tres variables de control, que combinadas resultan en la señal actuante de postura $u_{\theta, \phi, \psi}$, la cual se suma a la variable *Gas* y a la señal actuante de posición para después ser aplicada a los motores [14].

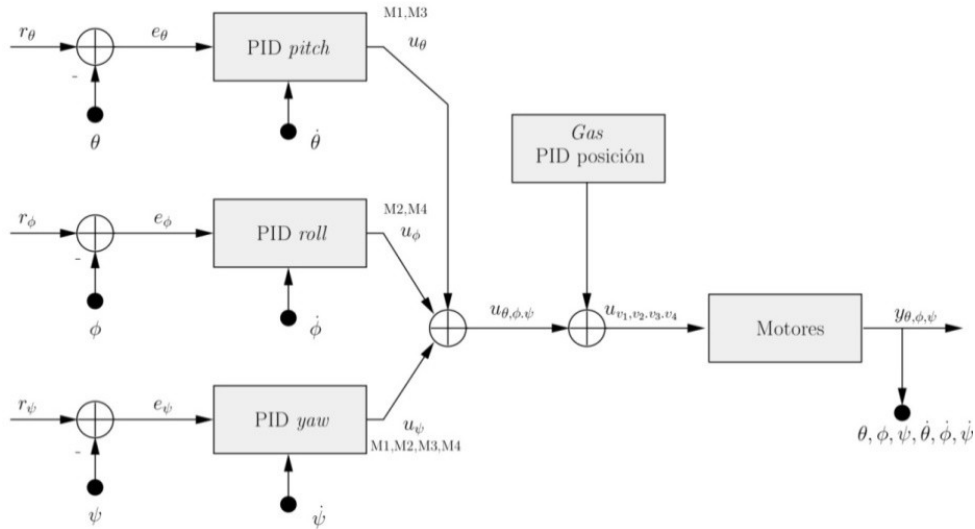


Figura 20: Diagrama del control de postura del cuadricóptero

4.7.2 Control de $pitch(\theta)$

Como puede verse en la Figura 21, el ángulo θ representa los grados de rotación en el eje y del sistema de coordenadas del cuadricóptero. Al subir el frente del vehículo, se obtienen ángulos positivos y al bajarlo ángulos negativos. Con esto se puede observar que el control automático de θ sólo actúa sobre los motores 1 y 3 [13].

Para elevar el frente, se aumenta la velocidad en el motor 1 y se disminuye en la misma magnitud en el motor 3, haciendo lo contrario para bajarlo.

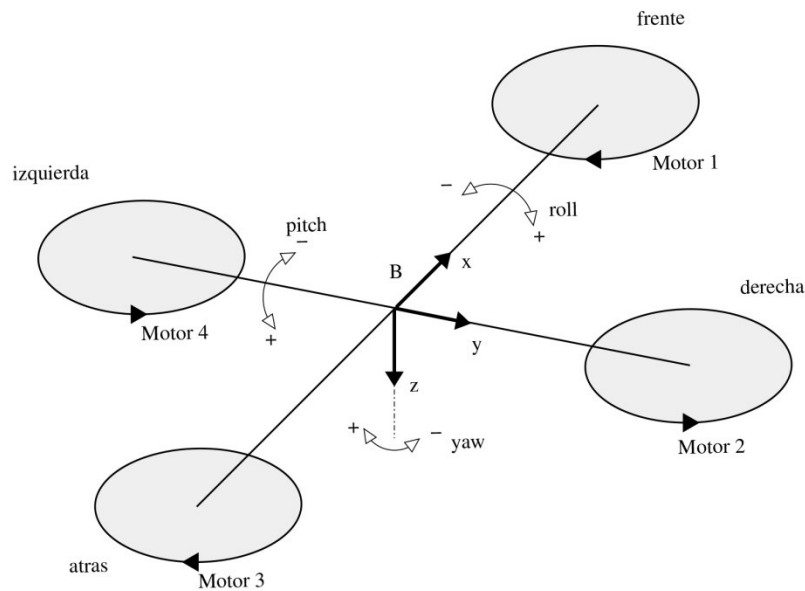


Figura 21: Ángulos determinan la postura de cuadricóptero, tomado de Pico (2012).

La Figura 22 muestra el sistema de control PID para el ángulo θ . Para éste y los demás controladores de postura se utilizan una versión modificada del PID básico. En ésta, el control proporcional e integral utilizan la señal de error “e” como entrada, pero la parte derivativa utiliza la derivada de la señal de retroalimentación “y”.

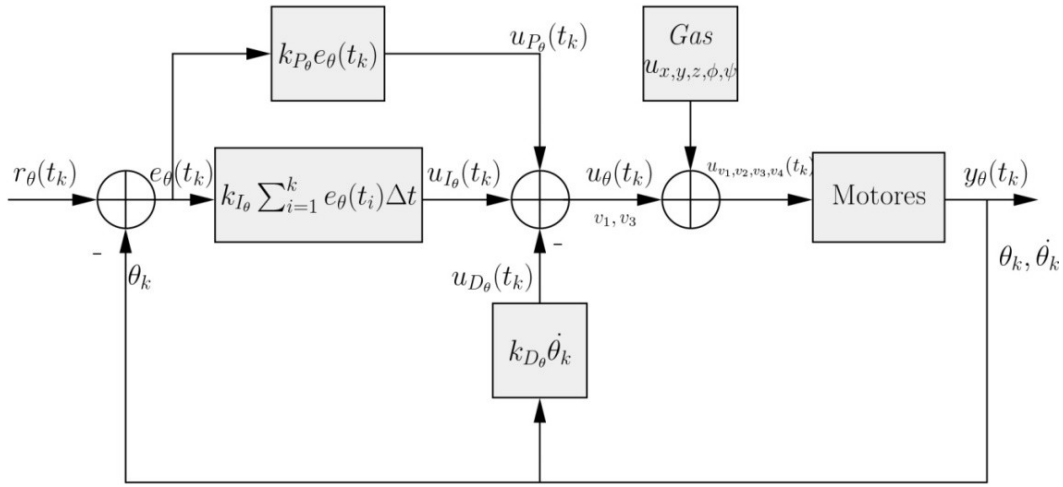


Figura 22: Diagrama del controlador de pitch, tomado de Pico (2012).

Este cambio se hace para evitar el fenómeno denominado “patada derivativa”, el cual sucede en ciertos sistemas al cambiar repentinamente la entrada de referencia r , lo que provoca un impulso en la señal de error, el cual se transmite a la salida de la señal de control derivativo, provocando cambios bruscos en la acción de control.

Al utilizar como entrada la derivada de la señal de retroalimentación, en el componente derivativo, los cambios repentinos en la señal de referencia no afectan el desempeño del control derivativo, obteniendo un sistema más estable.

Una ventaja más de esta configuración es que la derivada de la señal de salida (variable manipulada, posición angular) es la velocidad angular, que puede ser obtenida directamente de los sensores del vehículo evitando la necesidad de hacer aproximaciones basadas en la posición.

Continuando con el diagrama, la entrada de referencia r_θ corresponde los grados de inclinación deseados y la señal $e_\theta(t_k) = r_\theta(t_k) - y_\theta(t_k)$ es el error en el instante de tiempo

t_k , que es utilizado por los términos proporcional e integral. La variable de control $u_{p\theta}$ obtenida del término proporcional queda de la siguiente manera:

$$u_{p\theta}(t_k) = K P_{\theta} * e_{\theta}(t_k)$$

Donde $K P_{\theta}$ es la constante proporcional en θ y $e_{\theta}(t_k)$ es el error en el instante de tiempo t_k

La variable de control $u_{I\theta}$ aportada por el término integral se obtiene con la fórmula:

$$u_{I\theta}(t_k) = K I_{\theta} \sum_{i=1}^k e_{\theta}(t_i) \Delta t,$$

Donde $K I_{\theta}$ es la constante integral en θ , $e(t_i)$ es el error en tiempo continuo en el i -ésimo instante de muestreo Δt es el intervalo de muestreo.

La acción de control $u_{D\theta}$ del término derivativo es la siguiente:

$$u_{D\theta}(t_k) = K D_{\theta} * \dot{\theta}(t_k)$$

Donde $K D_{\theta}$ es la constante derivativa en θ u $\dot{\theta}(t_k)$ es la velocidad angular en el eje y en el instante de tiempo t_k .

Por lo que la señal actuante en θ es:

$$u_{\theta}(t_k) = u_{P_{\theta}}(t_k) + u_{I_{\theta}}(t_k) - u_{D_{\theta}}(t_k) = K P_{\theta} * e_{\theta}(t_k) + K I_{\theta} \sum_{i=1}^k e_{\theta}(t_i) \Delta t - K D_{\theta} * \dot{\theta}(t_k)$$

En conclusión, el aporte de la variable de control $u_{\theta}(t_k)$ sobre las velocidades de los motores es:

$$u_{v_1}(t_k) = u_{n_{v_1}} + u_{\theta}(t_k), u_{v_3}(t_k) = u_{n_{v_3}} - u_{\theta}(t_k)$$

Donde $u_{v_1}(t_k)$ y $u_{v_3}(t_k)$ son las señales actuantes en los motores 1 y 3, y u_n representa las señales actuantes de los demás controladores.

4.7.3 Control de roll (ϕ)

El ángulo ϕ representa la rotación en el eje x en el sistema de coordenadas del cuadricóptero. Elevar el lado derecho del vehículo corresponde a un giro negativo, mientras que bajarlo (elevar el lado izquierdo) corresponde a un giro positivo. Para elevar el lado derecho, se aumenta la velocidad en el motor 2 y se disminuye en la misma proporción en el motor 4, para bajarlo se hace lo contrario. Con esto se puede ver que el controlador del ángulo ϕ en el cuadricóptero sólo actúa sobre los motores 2 y 4 [13].

La Figura 23 muestra el diagrama del controlador PID de *roll*. El esquema es el mismo que en el controlador de *pitch*, con la diferencia de que actúan sobre diferentes motores.

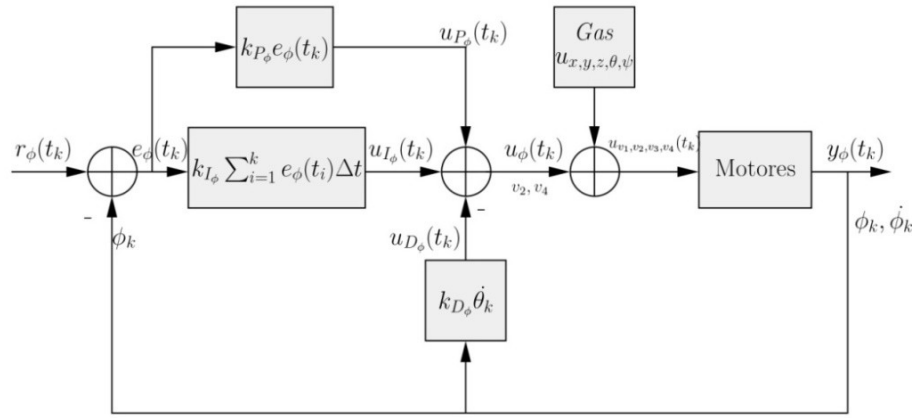


Figura 23: Diagrama de control PID de roll, tomado de Pico (2012)

La fórmula del controlador PID para ϕ es la siguiente:

$$u_{\phi}(t_k) = u_{P_{\phi}}(t_k) + u_{I_{\phi}}(t_k) - u_{D_{\phi}}(t_k) = K P_{\phi} * e_{\phi}(t_k) + K I_{\phi} \sum_{i=1}^k e_{\phi}(t_i) \Delta t - K D_{\phi} * \dot{\phi}(t_k)$$

Dónde:

$u_{\varphi}(t_k)$ es la variable de control en φ .

$u_{P_{\varphi}}(t_k), u_{I_{\varphi}}(t_k), u_{D_{\varphi}}(t_k)$ son las variables de control proporcional, integral y derivativo en φ .

$e(t_i)$ es la señal de error en φ .

$K P_{\varphi}, K I_{\varphi}, K D_{\varphi}$ son las constantes proporcional, derivativa e integral en φ .

$\dot{\varphi}(t_k)$ es la velocidad angular en el eje x en el sistema de coordenadas del vehículo.

El aporte de la variable de control $u_{\varphi}(t_k)$ sobre las velocidades de los motores es:

$$u_{v_2}(t_k) = u_{n_{v_2}} - u_{\varphi}(t_k), u_{v_4}(t_k) = u_{n_{v_4}} + u_{\varphi}(t_k)$$

Donde $u_{v_2}(t_k)$ y $u_{n_{v_4}}$ son las señales actuantes en los motores 2 y 4, y u_n representa las señales actuantes de los demás controladores.

4.7.4 Control de yaw (ψ)

El ángulo yaw representa un movimiento de rotación en el eje z del sistema de coordenadas del vehículo (Figura 21). El giro es negativo en el sentido de las manecillas del reloj y positivo en contra de las manecillas [13].

Al aumentar la velocidad en los motores 1 y 3, y disminuirla en la misma magnitud en los motores 2 y 4, se produce un giro negativo, mientras que para producir un giro positivo se hace lo contrario.

A diferencia de los controladores de los ángulos θ y ϕ , el controlador PID del ángulo ψ actúa sobre los cuatro motores del cuadricóptero.

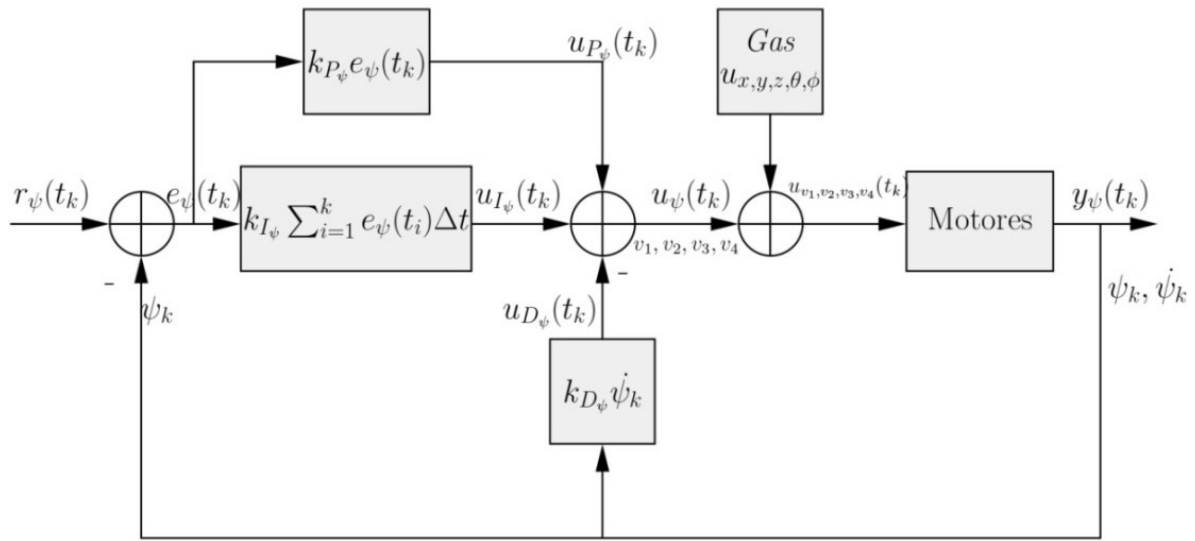


Figura 24: Diagrama del controlador PID de yaw, tomado de Pico (2012).

De la Figura 24 se obtiene la ecuación del controlador PID para ψ , la cual se define de la siguiente manera:

$$u_{\psi}(t_k) = u_{P_{\psi}}(t_k) + u_{I_{\psi}}(t_k) - u_{D_{\psi}}(t_k) = K P_{\psi} * e_{\psi}(t_k) + K I_{\psi} \sum_{i=1}^k e_{\psi}(t_i) \Delta t - K D_{\psi} * \dot{\psi}(t_k)$$

Donde:

$u_{\psi}(t_k)$ es la variable de control en ψ .

$u_{P_\psi}(t_k), u_{I_\psi}(t_k), u_{D_\psi}(t_k)$ son las variables de control proporcional, integral y derivativo en ψ .

$e(t_i)$ es la señal de error en ψ .

$K P_\psi, K I_\psi, K D_\psi$ son las constantes proporcional, derivativa e integral en ψ .

$\dot{\psi}(t_k)$ es la velocidad angular en el eje z en el sistema de coordenadas del vehículo.

El aporte de la variable de control $u_\psi(t_k)$ sobre las velocidades de los motores es:

$$\begin{aligned} u_{v_1}(t_k) &= u_{n_{v_1}} - u_\psi(t_k), u_{v_3}(t_k) = u_{n_{v_3}} - u_\psi(t_k) \\ u_{v_2}(t_k) &= u_{n_{v_2}} + u_\psi(t_k), u_{v_4}(t_k) = u_{n_{v_4}} - u_\psi(t_k) \end{aligned}$$

Donde $u_{v_1}(t_k), u_{v_2}(t_k), u_{v_3}(t_k)$ y $u_{v_4}(t_k)$ y son las señales actuantes en los motores 1, 2, 3 y 4, y u_n representa las señales actuantes de los demás controladores.

4.8 CONTROL DE POSICIÓN

El control de posición se divide en dos subcontroles: un control automático de altura y un control manual de posición en los ejes “x” e “y” con respecto al sistema de coordenadas en tierra. Como se muestra en la Figura 25, se utiliza un controlador PID para la altura. Para mover el vehículo en las direcciones “x” e “y”, se varían los ángulos de la postura (por ejemplo, para ir hacia adelante se cambia el valor de referencia del controlador de *pitch* a un ángulo negativo), por lo que el control manual consiste en enviar los valores de referencia r_θ, r_ϕ y r_ψ al controlador de postura. Al combinar las variables de control de postura, altura y *Gas*, se obtiene la señal actuante total del sistema [14].

4.9 CONTROL DE ALTURA

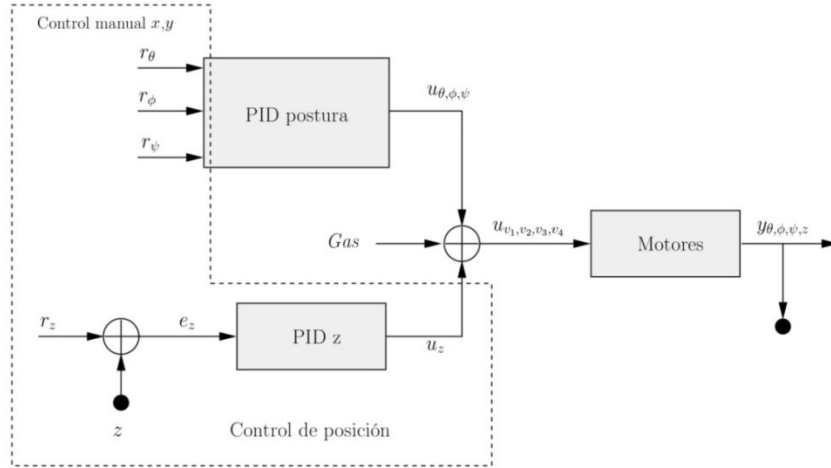


Figura 25: Diagrama de control de posición, tomado de Pico (2012).

El control de altura se hace variando la velocidad de todos los motores con la misma magnitud. Para elevar el vehículo se incrementa y para bajarlo se decrementa. Como puede verse en la Figura 26, se implementa un control PID clásico donde $r_z(t_k)$ es la altura deseada y la señal de error $e_z(t_k) = r_z(t_k) - y_z(t_k)$ es la diferencia entre la altura deseada y la altura real del vehículo medida por los sensores. Esta señal de error es tomada por las partes de control proporcional, integral y derivativa, de las cuales se obtienen las variables de control $u_{p_z}(t_k)$, $u_{I_z}(t_k)$ y $u_{D_z}(t_k)$. La suma de estas variables resulta en la señal actuante de altura, que junto con las variables de control de postura y Gas , es aplicada a los motores del cuadricóptero [14].

La variable de control u_{p_z} obtenida del término proporcional queda de la siguiente manera:

$$u_{P_z}(t_k) = K P_z * e_z(t_k)$$

Donde $K P_z$ es la constante proporcional en z y $e_z(t_k)$ es el error en el instante de tiempo t_k

La variable de control u_{I_z} aportada por el término integral se obtiene con la fórmula:

$$u_{I_z}(t_k) = K I_z \sum_{i=1}^k e_z(t_i) \Delta t,$$

Donde $K I_z$ es la constante integral en z, $e_z(t_i)$ es el error en tiempo continuo en el i-ésimo instante de muestreo Δt es el intervalo de muestreo.

La acción de control u_{D_z} del término derivativo es la siguiente:

$$u_{D_z}(t_k) = K \frac{D_z * e_z(t_k) - e_z(t_{k-1})}{\Delta t}$$

Donde $K D_z$ es la constante derivativa en z u $e_z(t_k)$ es el error en el instante de tiempo t_k y $e_z(t_{k-1})$ es el error en el muestreo anterior.

Por lo que la señal actuante en z es:

$$u_z(t_k) = u_{P_z}(t_k) + u_{I_z}(t_k) - u_{D_z}(t_k) = K P_z * e_z(t_k) + K I_z \sum_{i=1}^k e_z(t_i) \Delta t - K \frac{D_z * e_z(t_k) - e_z(t_k)}{\Delta t}$$

En conclusión, el aporte de la variable de control u_z sobre las velocidades de los motores es:

$$u_{v_1}(t_k) = u_{n_{v_1}} + u_z(t_k), u_{v_2}(t_k) = u_{n_{v_2}} + u_z(t_k)$$

$$u_{v_3}(t_k) = u_{n_{v_3}} + u_z(t_k), u_{v_4}(t_k) = u_{n_{v_4}} + u_z(t_k)$$

Donde $u_{v_1}(t_k)$, $u_{v_2}(t_k)$, $u_{v_3}(t_k)$ y $u_{v_4}(t_k)$ son las señales actuantes en los motores 1, 2, 3 y 4 respectivamente, y u_n representa las señales actuantes de los demás controladores.

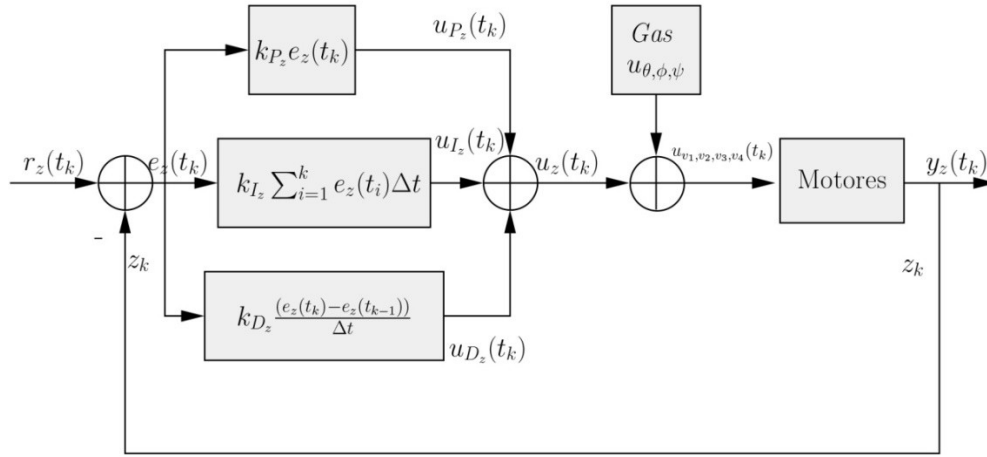


Figura 26: Diagrama del controlador PID de altura, tomado de Pico (2012).

4.10 CONTROL GENERAL DEL SISTEMA

La Figura 27 muestra el diagrama del sistema de control del cuadricóptero. Éste cuenta con cuatro controladores PID: tres para la postura y uno para la altura. Cada señal actuante de estos controladores tiene un aporte sobre la velocidad de cada uno de los cuatro motores del vehículo [14]. La variable de control total en cada motor es la suma de los aportes de todos los controladores PID más la variable de referencia Gas .

Uniendo las variables de control de las ecuaciones 5, 7, 9, 10, 15 y 16 se obtiene:

$$\begin{aligned}
u_{v_1}(tk) &= Gas + u_{\theta}(t_k) - u_{\psi}(t_k) + u_z(t_k) \\
u_{v_2}(tk) &= Gas - u_{\phi}(t_k) + u_{\psi}(t_k) + u_z(t_k) \\
u_{v_3}(tk) &= Gas - u_{\theta}(t_k) - u_{\psi}(t_k) + u_z(t_k) \\
u_{v_4}(tk) &= Gas + u_{\phi}(t_k) + u_{\psi}(t_k) + u_z(t_k)
\end{aligned}$$

Donde uv_1, uv_2, uv_3 y uv_4 , son las variables de control aplicadas a los motores 1, 2, 3 y 4 respectivamente.

u_{θ} , es la variable de control en *pitch*.

u_{ϕ} , es la variable de control en *roll*.

u_{ψ} , es la variable de control en *yaw*.

u_z , es la variable de control de altura.

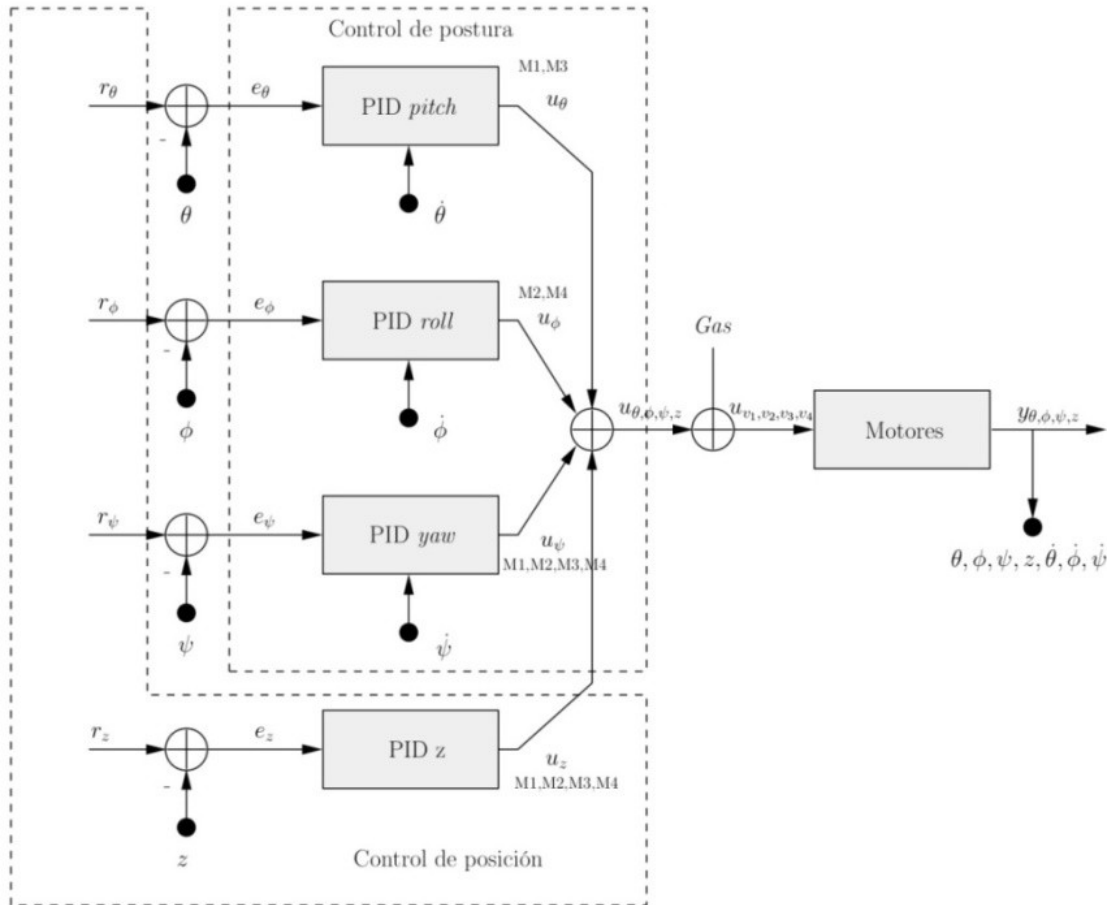


Figura 27: Diagrama del control general del sistema, tomado de Pico (2012).

4.11 FLUJO ÓPTICO

Una de las aplicaciones típicas del flujo óptico, es el diseño de *mouse* modernos, en la descripción de su funcionamiento se basará la explicación de cómo es posible determinar el desplazamiento a partir de imágenes consecutivas del suelo. Se utilizará este ejemplo porque la forma de calcular el desplazamiento en el plano por el que se desplaza el mouse, tiene la misma lógica que se utiliza para calcular el desplazamiento en el espacio, a partir de la cámara de flujo óptico montada en la parte inferior del vehículo aéreo, pero es más

simple de explicar. Además en [16], se explica cómo se logra calcular los desplazamiento de un cuadricóptero respecto al suelo, utilizando el sensor de un mouse modificado.

En sensores basados en CMOS y un microprocesador de señal digital (DSP) la medición del flujo óptico se realiza por medio del análisis comparativo en una secuencia de imágenes captadas por un sistema de visión. Este análisis pretende cuantificar la dirección y cantidad de movimiento producido que se conoce como flujo óptico, y este se interpreta como el desplazamiento relativo bidimensional en el plano cartesiano XY. Para estimar el flujo óptico existe un sin número de técnicas [17], sin embargo, debido a su aplicación y sus limitaciones, estos sensores de bajo coste poseen algoritmos de procesamiento de imagen optimizados y adaptados con una aritmética computacional menor.

El principio de funcionamiento de un sensor de flujo óptico estándar está formado por un chip o sensor principal que incorpora una cámara integrada CMOS y un microprocesador de señal digital (DSP), una fuente de luz (normalmente luz infrarroja o láser) que ilumina la superficie, y un conductor de luz con un par de lentes de plástico esféricas, una para focalizar la imagen capturada en una determinada distancia y la otra para concentrar la iluminación producida por la fuente de luz en la zona visible por la cámara (Figura 28) [16].

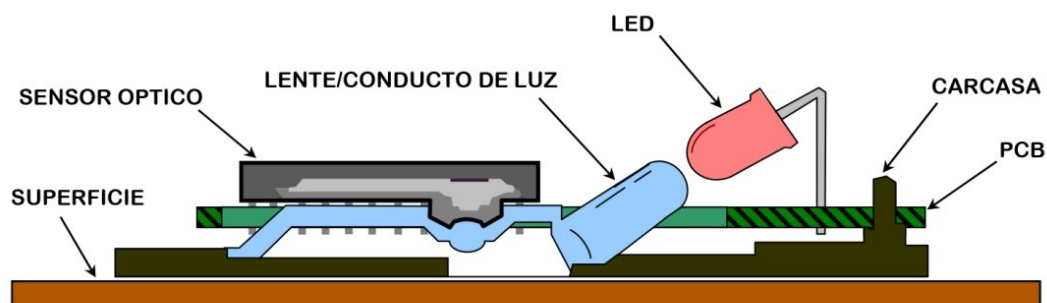


Figura 28: Vista de perfil de las partes involucradas en el sistema de un sensor de flujo óptico, tomado de Tresanchez (2011).

En la Figura 29 se muestra esquemáticamente el funcionamiento básico del sensor de flujo óptico. En primer lugar, el DSP lee la imagen mediante la matriz de fotodiodos que forman la CMOS acumulando una carga eléctrica en cada celda proporcional a la intensidad de la luz que incide sobre ella, a mayor intensidad luminosa, mayor carga acumulada. La incorporación de una fuente de luz, por un lado, ayuda a obtener una iluminación constante para cada celda, y por otro, como está situada en posición diagonal provoca sombras en las pequeñas imperfecciones de la superficie lo que facilita el correcto funcionamiento del algoritmo de flujo óptico (Figura 29). La intensidad de la fuente de luz es controlada por el algoritmo de autoexposición ejecutado en el DSP interno del sensor. Una vez los valores de la CMOS (imagen) son almacenados en un buffer, el DSP se encarga también de ejecutar un algoritmo de flujo óptico para calcular el desplazamiento relativo entre la imagen capturada y la anterior. Finalmente, estos datos de desplazamiento son guardados en dos registros de valores enteros que pueden ser leídos mediante un puerto de comunicaciones serie y expresan el número de pulsos (data counts) que corresponden al incremento de desplazamiento desde la última lectura realizada [16].

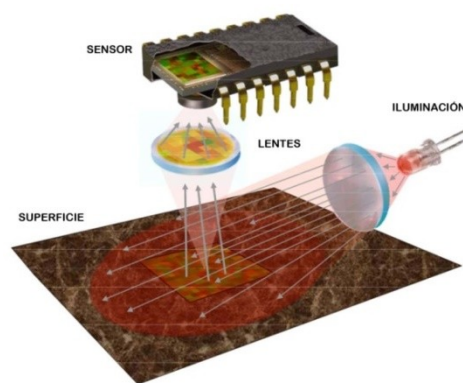


Figura 29: Esquema básico de funcionamiento del sensor de flujo óptico, tomado de Tresanchez (2011)

Con este principio de funcionamiento, un sensor de flujo óptico puede llegar a generar resoluciones de desplazamiento muy elevadas (hasta 5670 cpi con el sensor ADNS-9500) debido a la capacidad de detectar detalles microscópicos de la superficie. Sin embargo, la resolución del sensor es altamente dependiente de la configuración del sistema de visión (campo de visión) y del tamaño de la CMOS del sensor. Cuantos más píxeles se disponga y menor sea el campo de visión, más resolución se dispondrá. Otros factores que influyen en el funcionamiento del sensor de flujo óptico son la altura de trabajo (distancia entre sensor y superficie) y la velocidad del desplazamiento que se desea medir [16].

Si se analiza la velocidad máxima de desplazamiento, o velocidad en que el algoritmo de flujo óptico deja de funcionar, éste depende principalmente de la resolución del sensor y del tiempo en leer y procesar una imagen (tasa de muestreo). Según el principal fabricante, Avago Technologies, el sensor de flujo óptico puede llegar a procesar hasta 11750 imágenes por segundo (fps) con el modelo ADNS-9500 [16].

4.12 MÉTODOS PARA LA ESTIMACIÓN DEL FLUJO ÓPTICO

A continuación se realiza una breve descripción de los métodos más representativos para la estimación del flujo óptico reportados en la literatura.

4.12.1 MÉTODOS DIFERENCIALES

Calculan el desplazamiento de los píxeles a partir de las derivadas espaciales o espacio-temporales de las intensidades de la imagen. Una limitación que tiene este tipo de métodos es que obliga a que las derivadas sean computables en el dominio de la imagen. En función de cómo se use la información de las derivadas. Estableció la siguiente subcategoría para los métodos diferenciales:

- **Métodos locales:** utilizan la información en una vecindad alrededor de un píxel para estimar su movimiento. El método más representativo de esta familia es el de Lucas-Kanade. Este método calcula el desplazamiento a partir de la **militarización** de la ecuación del flujo óptico alrededor de una ventana centrado en un píxel.

$$\frac{dI(x, y, t)}{dt} = \frac{\partial I(x, y, t)}{\partial x} \frac{dx}{dt} + \frac{\partial I(x, y, t)}{\partial y} \frac{dy}{dt} + \frac{\partial I(x, y, t)}{\partial t} = 0$$

- **Métodos globales:** añaden como restricción global un término de regularización sobre el flujo. Esta restricción supone que el campo de desplazamiento es suave. Con este tipo de métodos se obtienen campos de desplazamientos densos. El método más representativo es el de Horn-Schunck.
- **Métodos de contorno:** utilizan la información de los bordes de los objetos para detectar el desplazamiento. Aplican técnicas diferenciales para la extracción de determinadas estructuras en la imagen para luego establecer correspondencias entre estas estructuras. Entre los métodos más relevantes pertenecientes a este grupo se destaca porque realiza un suavizado sobre el contorno extraído de la imagen, permitiendo una mejor estimación del flujo óptico.
- **Métodos variacionales:** La idea que subyace a este tipo de métodos es la definición de una energía que penaliza las desviaciones respecto a las restricciones impuestas en el modelo. Una de las ventajas que ofrecen es que todas estas restricciones están presentes en la energía; no existe ningún tipo de suposición adicional que no se refleje en el modelo y si en la implementación. A diferencia de otro tipo de técnicas las estimaciones son densas por lo que no es necesario realizar ningún proceso de

interpretación. La solución se obtiene directamente mediante la **militarización** de esta energía.

$$u^{k+1} = u^{-k} - \frac{I_x(I_x u^{-k} + I_y v^{-k} + I_t)}{\alpha^2 + I_x^2 + I_y^2}$$

$$v^{k+1} = v^{-k} - \frac{I_y(I_x u^{-k} + I_y v^{-k} + I_t)}{\alpha^2 + I_x^2 + I_y^2}$$

4.13 ESTIMACIÓN DEL FLUJO ÓPTICO

Las imágenes son datos observados en el cual se quiere registrar en la escena, y el desplazamiento de los píxeles no es más que la proyección en una imagen del movimiento tridimensional. No obstante, puede ocurrir que al estimar el flujo óptico se tenga varias soluciones para el mismo desplazamiento, lo cual conlleva a que se provoque un problema mal condicionado, para convertir este problema en bien condicionado se deben cumplir tres condiciones principalmente: existencia, unicidad, estabilización de la solución. Para lograr estas tres condiciones se debe incorporar un elemento el cual estabilice, normalmente en forma de término de regularización o suavizado. El desplazamiento de píxeles en la imagen provoca un movimiento en la escena, pero hay gran cantidad de situaciones en la que se produce el movimiento de píxeles pero no de objetos o viceversa, por lo cual existen diferentes factores que dificultan la estimación exacta del flujo óptico [18].

Si se representa una imagen como aplicación $I: (x, y, t) \rightarrow I(x, y, t)$ donde (x, y) representa la coordenada espacial de la imagen y t el tiempo, se tendría una secuencia de imágenes con variación en el tiempo en su intensidad de las coordenadas. El vector de

desplazamiento se define como función $h(x, y, t) = (u(x, y, t), v(x, y, t))^T$ y representa el movimiento horizontal y vertical de los píxeles a través de la secuencia de imágenes. Se suele suponer que alguna propiedad presente en la imagen no varía a lo largo del tiempo, se representa como [3][4][5][6][7][8]:

$$f(x+u, y+v, t+1) - f(x, y, t) = 0$$

Donde f es algún tipo de propiedad en la imagen y $t, t+1$ representan imágenes en diferentes instantes de tiempo. La intensidad presente en los píxeles de la imagen indica la radiación luminosa que refleja una superficie en los objetos. Hay diferentes modelos para la representación de tipos de superficie. Uno de ellos es una superficie lambertiana en donde el brillo aparente es el mismo en todas las direcciones de vista. Un píxel que pertenezca a este tipo de superficie tendrá igual valor de intensidad en todas las secuencias de imágenes [18].

4.13.1 Estimación del movimiento basado en gradiente

Sea $f(x)$ una función de la posición espacial, y asumiendo que $f(x)$ es simplemente trasladada por d entre el tiempo 1 y 2.

$$f_2(x) = f_1(x-d)$$

Se puede expresar la señal desplazada como expansión de Taylor de $f_1(x)$ sobre x

$$f_1(x-d) = f_1(x) - df_1'(x) + O$$

En cuyo caso la diferencia entre las dos señales está dada por

$$f_2(x) - f_1(x) = -df_1'(x) + O$$

Se obtiene una primera aproximación para el desplazamiento de la siguiente forma:

$$d = \frac{f_1(x) - f_2(x)}{f_1'(x)}$$

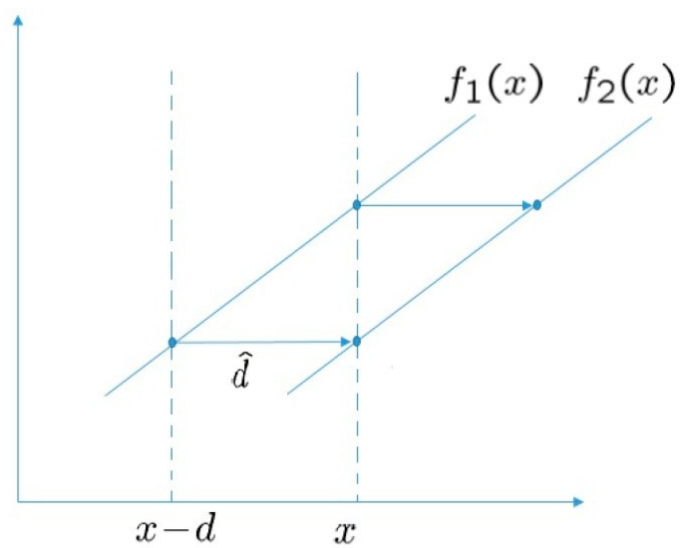


Figura 30: Estimación de señal de primer orden exacto, tomado de Riascos (2015).

Para señales no lineales, la exactitud de la aproximación depende de la magnitud del desplazamiento y de la estructura de la señal de orden superior (Figura 30 y 31).

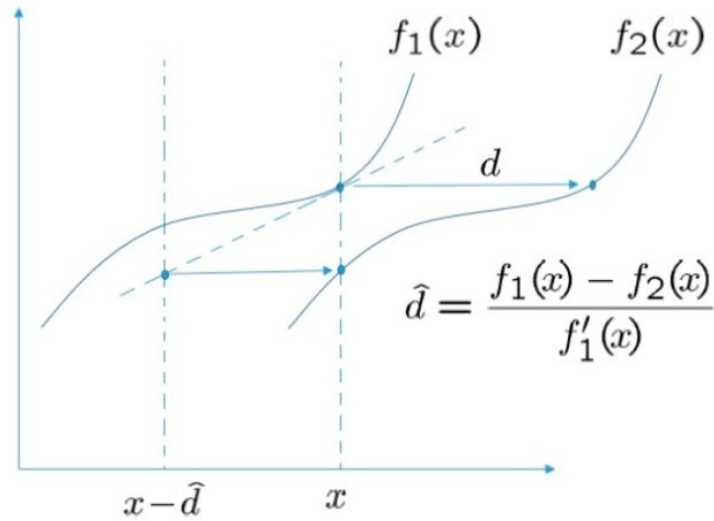


Figura 31: Estimación de señal no lineal de orden superior, tomado de Riascos (2015).

4.13.2 Dirección normal

Un píxel perteneciente a una superficie lambertiana donde el brillo es aparente en todas las direcciones mantendrá el mismo valor de intensidad en todas las secuencias de imágenes. Si se asume que las superficies de los objetos son lambertianas se hace una sustitución f por I en la ecuación 24 para representar esta invarianza. Esta expresión es no lineal, lo cual se puede prevenir si se realiza expansiones de Taylor, esto permite excluir los términos que son de orden superior. Esta nueva expresión adquiere el nombre de *ecuación de restricción del flujo óptico*, solo cuando $f = I$ [18].

$$I_x u + I_y v + I_t = 0$$

Los subíndices son derivadas parciales. De la expresión 29 no es posible poder determinar el vector desplazamiento de una sola interpretación posible ya que se tiene dos incógnitas y una sola ecuación. Solo es posible realizar la estimación de la componente en la dirección

normal a la curva de nivel, en la dirección del gradiente (Figura 32). Esto es lo que se conoce como el problema de *apertura* [18].



Figura 32: Restricción de gradiente de la velocidad en la dirección normal, tomado de Riascos (2015) [18]

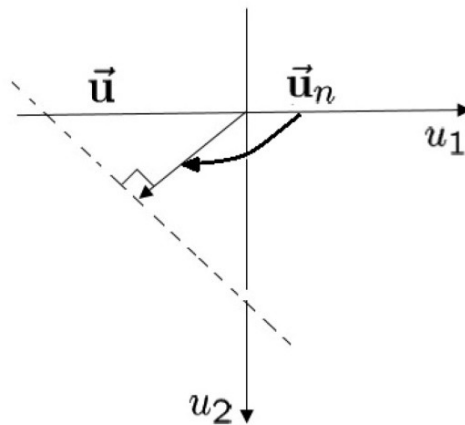


Figura 33: Línea de restricción de movimiento, tomado de Riascos (2015)

Como la restricción del gradiente de la velocidad en orientación de la imagen local, no limita tangencialmente con la velocidad, solo es posible realizar la estimación de la componente en la dirección normal a la curva de nivel, cuando la magnitud del gradiente es

cero, no se obtiene ninguna restricción, en cualquier caso es necesario más limitaciones para estimar ambos elementos de la velocidad $2D\dot{u} = (u_1, u_2)^T$ (Figura 33).

4.13.3 Área de regresión basada en mínimos cuadrados

Las restricciones de flujo óptico no se satisfacen exactamente, lo que se busca es la velocidad que minimiza el error cuadrático en cada restricción a lo que se le conoce con el nombre de estimación de velocidad por mínimos cuadrados, para esta técnica hay que tener en cuenta que la eficiencia puede cambiar y se representa como:

$$E(u_1, u_2) = \sum_{x,y} g(x, y) [u_1 f_x(x, y, t) \mp u_2 f_y(x, y, t) \mp f_t(x, y, t)]^2$$

Donde $g(x, y)$ es una ventana de pasa bajo que ayuda a dar más peso a restricciones en el centro de la región.

Para la solución del área de la regresión se hace diferenciales de E con respecto a (u_1, u_2) e igualar a cero.

$$\frac{\partial E(u_1, u_2)}{\partial u_1} = \sum_{xy} g(x, y) [u_1 f_x^2 \mp u_2 f_x f_y \mp f_x f_t] = 0$$

$$\frac{\partial E(u_1, u_2)}{\partial u_2} = \sum_{xy} g(x, y) [u_2 f_y^2 \mp u_1 f_x f_y \mp f_y f_t] = 0$$

Se tiene dos ecuaciones de forma lineal para la restricción tanto para u_1 y u_2 , estas pueden ser representadas de forma matricial como: $M\dot{u} \mp \dot{b} = 0$, $-\dot{M}^{-1}\dot{b}$ asumiendo que M^{-1} existe, donde

$$M = \sum g \begin{pmatrix} f_x \\ f_y \end{pmatrix} (f_x, f_y) = \begin{bmatrix} \sum g f_x^2 & \sum g f_x f_y \\ \sum g f_x f_y & \sum g f_y^2 \end{bmatrix}$$

$$\dot{b} = \sum g f_t \begin{pmatrix} f_x \\ f_y \end{pmatrix} = \begin{pmatrix} \sum g f_x f_t \\ \sum g f_y f_t \end{pmatrix}$$

4.13.4 Problema de apertura

Consiste en conocer el desplazamiento de la componente normal en la dirección del gradiente de los píxeles de una línea. Pero se desconoce si la línea se ha desplazado también en la dirección de la componente ortogonal a la dirección del gradiente (Figura 34).

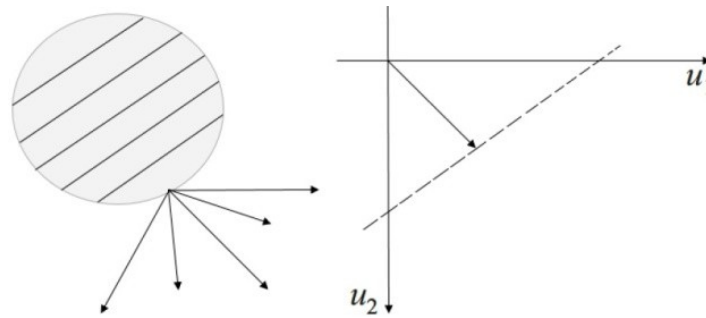


Figura 34: Problema de apertura, tomado de Riascos (2015) [CITATION Ria15 \l 3082]18

Cuando todos los gradientes de la imagen son paralelos, la solución de la matriz normal para los mínimos cuadrados se vuelve simple, por ejemplo para gradientes $m(x, y)\hat{n}$ donde $m(x, y)$ es la magnitud del gradiente del píxel (x, y) [4][9][18].

$$M = \left(\sum g(x, y) m^2(x, y) \hat{n} \hat{n}^T \right)$$

En este apartado se realiza una revisión de aquellos trabajos que por tener un fin similar al de esta tesis, pudieran servir como base de discusión para desarrollar una solución al problema a resolver. Para ello, se ha realizado una primera introducción al flujo óptico por computador. Posteriormente se especifican las diversas técnicas de control de vuelo. A continuación se describen algunas publicaciones realizadas sobre el flujo óptico:

5.1 FLUJO ÓPTICO

T.W. Ng [19] introduce el sensor de flujo óptico como alternativa económica para medir desplazamientos siendo uno de los pioneros en esta técnica, en [20] se puede ver una aplicación desarrollada para este fin. Para poder realizar dispositivos y aplicaciones basadas en este tipo de sensor es necesario hacer una caracterización estudiando sus atributos principales como, por ejemplo, el error de desplazamiento, la sensibilidad en variaciones de la altura entre el sensor y la superficie a medir, la influencia de la iluminación y el tipo de suelo, la importancia de la ubicación del sensor en una plataforma robótica para conocer la trayectoria seguida, y la sensibilidad a los cambios de velocidad y aceleración del sensor.

Uno de los peores escenarios que se pueden presentar para el sensor de flujo óptico al ser considerado como un sensor de medida de desplazamiento es la acumulación de errores producidos durante las estimaciones de las medidas. En el caso de aplicaciones con robots móviles la estimación del desplazamiento está sujeta tanto a errores sistemáticos como no sistemáticos [21][22]. También, estos errores son inherentes a las alteraciones en el entorno de trabajo debido a los cambios del entorno físico o a la intervención del operador humano, y también por algunos defectos del propio sensor como la incapacidad de reconocer movimiento en una superficie homogénea o cuando la distancia entre sensor y superficie es demasiado alta. Por otro lado se pueden producir errores por el mal estado o el desgaste de la superficie de trabajo como grietas, golpes o suciedad acumulada, o incluso por un deslizamiento excesivo e incontrolado de las ruedas producido por la presencia de líquido o

una colisión, en [23] se ha trabajado para encontrar las condiciones de la superficie en donde la medición del sensor es más preciso. Las medidas originales de la cantidad desplazamiento en función del tiempo son basadas en el contacto con la superficie del suelo, la cual genera un deslizamiento diferente dependiendo del tipo de superficie produciendo errores acumulativos [24]. El uso del algoritmo implementado en [25] los robots móviles son capaces de desplazarse dada una trayectoria sin basarse en orientación de un sensor GPS en un trayecto relativamente largo con un pequeño error de posicionamiento. En varios proyectos científicos se están investigando el uso del sensor de flujo óptico para conocer la trayectoria de un robot móvil en función del tiempo tratando de identificar diferentes fuentes de error como cambios en la altura respecto al suelo, tipos de superficies, condiciones de iluminación, todo esto juntamente con la velocidad y la aceleración de desplazamiento [26][27][24]. En [28] se hace referencia al error de orientación el cual provoca errores considerables en el posicionamiento que se incrementan de forma proporcional con la distancia recorrida. En [29] se concluye que el error de orientación es el más significativo ya que provoca un error lateral ilimitado que crece también de forma proporcional con la distancia.

Los sensores de flujo óptico convencionalmente se han empleado como sensores de movimiento en dos dimensiones, sin embargo cuando se requiere de movimientos más complejos como el de múltiples grados de libertad (MDOF) pueden encontrarse interesantes para una gran variedad de aplicaciones en la robótica. En [29] se ha presentado un sistema formado por un sensor dual sin contacto para medir movimientos planos y esféricos con tres grados de libertad (3-DOF). Este sistema es capaz de detectar cambios microscópicos en una secuencia de imágenes, y estimar el desplazamiento angular y el centro instantáneo de rotación de superficie en movimiento.

En [30] se documenta el desarrollo de un nuevo sistema sensor con el propósito de captar variaciones en la altura y la translación de la superficie simultáneamente. Además, se ha

encontrado que la sensibilidad del sensor de flujo óptico es cambiante en diversas direcciones de translación. S-L Jeng and W-H Chieng P-L Wu [31] emplearon un dispositivo formado por varios sensores de flujo óptico para determinar las mediciones de los movimientos planos y esféricos con tres grados de libertad (3-DOF) con los métodos de mínimos cuadrados lineales (LLS) y mínimos cuadrados no lineales (NLS) alrededor de dos trayectorias circulares (no-orientada y auto-orientada). El dispositivo ha permitido evaluar la posición y la orientación de la estructura principal del sistema. Los resultados conseguidos muestran que cuando el movimiento es no-orientado se ha de utilizar el método LLS mientras que cuando se trata de una trayectoria auto-orientada el método LLS produce un error de trayectoria sinusoidal. El método NLS ha proporcionado mejores resultados en la precisión de la posición estimada, en una orden de magnitud. En los casos que ha habido variaciones en un desplazamiento incremental se ha visto que existen errores acumulativos.

5.2 CONTROL DE VUELO

A continuación, se describen algunas publicaciones realizadas sobre el control de vuelo:

En la literatura existen desde hace varias décadas estudios del control en el área de los vehículos aéreos no tripulados. La representación de los sistemas dinámicos que representan estos tipos de plataformas ha sido ampliamente documentado [32][33][34][35], donde se detalla un cuidadoso análisis sobre las propiedades aerodinámicas y de criterios de estabilidad. Debido a esta inestabilidad presente en este tipo de vehículos, los primeros estudios se dedicaron en desarrollar controladores para hacer estable a estos sistemas.

Guilherme Vianna Raffo [35] en su tesis presenta el diseño de un control realimentado robusto basado en H_∞ , de esta misma forma en el documento [36] se presenta un

controlador basado en lógica difusa para la estabilización de un *SUAV* (Small Unmanned Aerial Vehicle). Otras propuestas para el control se muestran [32][37][38], entre otros. El Laboratorio de Robótica Aeroespacial de la Universidad de Stanford [39] es uno de los pioneros en usar el GPS como sensor principal para la navegación sustituyendo a la unidad de medida inercial (IMU), la cual, y por historia, es el sensor que se había usado como principal. Este sistema estaba dotado de un GPS con 4 receptores y 4 antenas colocadas en puntos estratégicos del aparato, con lo que se determina su posición, velocidad, altitud e información angular.

En la actualidad la visión artificial o por computador ha tomado una gran relevancia convirtiéndose en una de las áreas más importantes en el mundo de la robótica. En los últimos años, las capacidades hardware de los sistemas embebidos fueron mejoradas lo que ha permitido llevar a cabo una gran variedad de aplicaciones visuales sofisticadas. Lo anterior era la más grande limitante que no permitía que el control visual lograra los avances que conocemos en la actualidad que ha tenido estos años atrás. Gracias a nuevos desarrollos en hardware y de procesadores de mayor capacidad y con soporte para programación paralela, y de cámaras de mayores prestaciones a un coste relativamente asequible, han hecho posible que el control visual sea ya una realidad.

Cuando se presentan términos como control visual aplicados a vehículos aéreos autónomos, se hace referencia al uso de la información visual del procesamiento de imágenes para el control de velocidad, posición absoluta o relativa, o para el control de la orientación. Al igual que la literatura sobre el control visual contempla el término cámara-en-mano (*eye-in-hand*) para ciertas configuraciones en robots articulados, para el caso de vehículos aéreos robotizados el término adecuado sería el de cámara a bordo (*on-board camera*) [33].

En [40] se presenta la estabilización de vuelo de un helicóptero con cuatro rotores utilizando transductores inerciales y la información visual de una cámara. Utilizando el

flujo óptico experimentado por el sensor visual se realiza la estimación de la velocidad traslacional en el plano XY , de la posición y del ángulo de guiñada del helicóptero.

En [41] el autor utiliza conceptos de procesamiento digital de imágenes (algoritmos de localización y seguimiento), con conceptos de control de vehículos aéreos autónomos (basados o no en control visual) incluyendo además el modelo del controlador usado para garantizar la estabilidad del quadrotor; también incorpora la formulación del modelo de la cámara, algoritmos para la calibración de la misma; fusión sensorial entre los datos de la cámara con los del sensor inercial (IMU), cuestiones relacionadas con geometría proyectiva incluyendo además la descomposición de la matriz de homografía, y, además, la realización de experimentos en el laboratorio.

En [42] se presenta una aplicación de inspección, patrullaje y monitoreo, reconstrucción digital aérea para un UAV dotado de flujo óptico.

Este estado del arte presenta alguna información relevante acerca del flujo óptico y algunas de sus aplicaciones en el control de navegación de vehículos aéreos no tripulados.

En la Tabla 1 se presentan de forma ilustrativa la relevancia de los artículos discutidos.



Autor	Paper	Citas	Año
James P. Ostrowski, Robert Mahony. Erdinç Altug,	Control of a quadrotor Helicopter using Visual Feedback	419	2002
Tarek Hamel, and Robert Mahony	A Practical Visual Servo Control for an Unmanned Aerial Vehicle	237	2008
I. Valgañón and R. Pernia, J.Palacin	The optical mouse for indoor mobile robot odometry measurement	88	2009
U. Minoni and A. Signorini	Low-cost optical motion sensors: An experimental	46	2006
Ravi Prasanth and Raman K. Mehra. Jovan D.	A Multi-Layer Control Architecture for Unmanned Aerial Vehicles	42	2002

Boskovic			
Tarek Hamel, Robert Mahony. David Suter	Visual Servo Control using homography estimation for the stabilization of an X4-flyer	33	2010
A.P. Roskilly and R. Norman N. Tunwattana	Investigations into the effects of illumination and acceleration on optical mouse sensors as contact-free 2D measurement devices	21	2009
Guilherme Vianna Raffo	Tesis de Master	14	2007
Liang Yan, I-Ming Chen, Zhongwei Guo, Yan Lang, and Yunhua Li	A Three Degree-of-Freedom Optical Orientation Measurement Method for Spherical Actuator Applications	13	2011
Haiyang Chao, Yu Gu, and M., Napolitano	A survey of optical flow techniques for UAV navigation applications	13	2013
D. J. Hyun, H. S. Yang and H. S. Park H. R. Park	A dead reckoning sensor system and a tracking algorithm for mobile robot	8	2009
Robert Mahony. Tarek Hamel	Image based visual servo control for a class of aerial robotic systems.	4	2007
Xusheng Lei, Song Wang, Yongliang Wu and Tianmiao Wang Jianhong Liang	A Small Unmanned Aerial Vehicle for Polar Research,	4	2008
Nicholas Kottenstette	Constructive Non-Linear Control Design With Applications to Quad-Rotor and Fixed-Wing Aircraft	4	2010
Jesús María Gonzalez Villagómez	Realimentación visual para el control de un vehículo aéreo cuatrimotor no tripulado	4	2010
R. Ross and J.Devlin	Analysis of Real-Time Velocity Compensation for Outdoor Optical Mouse Sensor Odometry	4	2014
S-L Jeng and W-H Chieng P-L Wu	Least Squares Approach to Odometry based on Multiple Optical Mouse Sensors	3	2010
Hugo Romero, Sergio Salazar, and Juan Escareño	Estabilización de un mini helicóptero de cuatro rotores basada en flujo óptico y sensores inerciales	3	2010
Steven Bell	High-Precision Robot Odometry Using an Array of Optical Mice	3	2011
M. Matteucci and M. Restelli, A. Bonarini	Dead reckoning for mobile robots using two	0	2004
L., Campoy Mejias	COLIBRI: Vehículo Aéreo Autónomo Guiado por Visión para Inspección y Vigilancia	0	2007
Ondrej Spinka	RAMA - a Low-Cost Modular Control System for Unmanned Aerial Vehicles	0	2009
S. Krivic, A. Mrzic, J.	Optimization based algorithm for correction of	0	2013

Velagic, and N. Osmic	systematic odometry errors of mobile robot		
R.K. Ghosh, V. Kataria, and V. Mishra	Indoor navigation system using optical mouse sensor and smartphone	0	2014
Azizi, A.; Vossoughi, G.	Empirical study on effect of surface texture and grain size on displacement measurement of optical flow sensors	0	2014
E.J. Kreinar and R.D. Quinn	Odometry error estimation for a differential drive robot snowplow	0	2014
Yanming Pei and L. Kleeman	Online robot odometry calibration over multiple regions classified by floor colour	0	2015

6. HIPÓTESIS DE TRABAJO

6.1 PREGUNTAS DE INVESTIGACIÓN

En este trabajo se plantea la alternativa de utilizar un sensor de flujo óptico como reemplazo de los altímetros convencionales, de lo cual surgen las siguientes preguntas:

¿Es posible lograr el control de vuelo de un UAV a baja altura utilizando la información de un sensor de flujo óptico?

¿Es posible que un UAV siga de forma adecuada y autónoma una trayectoria con la información del sensor de flujo óptico implementado?

6.2 HIPÓTESIS

A continuación se plantea la hipótesis de acuerdo a la problemática establecida.

Agregando un sensor de flujo óptico a la odometría típica de un UAV es posible realizar una trayectoria a baja altura de manera autónoma en un ambiente controlado.

Para comprobar el alcance de esta hipótesis se definen las métricas de eficiencia de desempeño de los algoritmos implementados, en donde se comparan los métodos convencionales con el actual implementado.

7. DESARROLLO DE LA TESIS

7.1 PROBLEMA A RESOLVER

Se propone en este trabajo de tesis agregar a la odometría estándar de un vehículo aéreo no tripulado, un sensor de flujo óptico e intervenir sus lazos de control para que permitan obtener la estabilidad y la capacidad de seguir una trayectoria en un vuelo a baja altura, con la suficiente autonomía para adaptarse al relieve de la superficie sobre la cual navega.

7.2 DESCRIPCIÓN DEL TRABAJO REALIZADO

Esta tesis de grado se ha dividido en cuatro secciones, en la primera de estas se realiza un análisis del sensor de flujo óptico en donde se realizan pruebas experimentales para comprobar y entender su funcionamiento del flujo óptico en un ambiente controlado bajo la implementación de ciertos algoritmos para la captura de información proveniente de este, además se prepara el hardware necesario para la implementación de las demás pruebas en las secciones siguientes. Luego de comprobar que la información que entrega el sensor corresponde a medidas correctas de acuerdo a un desplazamiento realizado sobre él, se procederá a diseñar una prueba sobre un robot móvil terrestre en donde se reemplazará la odometría estándar de este por la implementación sistema de odometría visual basada en flujo óptico, el robot deberá responder de manera autónoma de acuerdo a la información proveniente del sensor, lo cual demostrará la capacidad de este para obtener las coordenadas XY en un determinado desplazamiento para una trayectoria definida en un plano cartesiano, este experimento se llevara a cabo en un entorno libre de obstáculos, será sobre una superficie plana con luz natural. La tercera sección se implementará el sensor de flujo óptico sobre el cuadricóptero de tal manera que pueda determinar la trayectoria sobre la superficie sobre la cual navega además pueda controlar el vuelo a baja altura siguiendo el

relieve de la superficie ~~sobre la cual navega~~, que para este trabajo distancias inferiores a un metro de altura.

7.3 DESARROLLO DE LA CAPTURA DE FLUJO ÓPTICO

En esta sección se comienza a realizar pruebas con el sensor de flujo de óptico PX4Flow en una plataforma experimental en la que no se incorporara el cuadricóptero. El propósito de este primer acercamiento con el sensor seleccionado se hace con la intención de entender su funcionamiento, capturar la información que entrega y determinar su debido procesamiento. Este sensor se integrará con la tarjeta Raspberry Pi Zero formando un sistema embebido incorporado al drone, el cual será encargado de capturar la información de flujo óptico y a través de un algoritmo de control producirá las órdenes de comando que deberán ser transmitidas al sistema principal de auto pilotaje.

7.3.1 Cálculo de desplazamiento

Con el propósito de incorporar una mayor autonomía del robot móvil en cuanto a su desplazamiento se refiere y de una manera precisa siguiendo el relieve del terreno por el cual navega, se procede al diseño de un algoritmo de captura de la información suministrada por el sensor de flujo óptico, el cual permita llevar las velocidades de desplazamiento producidos a un sistema cartesiano X,Y, además de incorporar la altura sobre la cual navega para luego incorporarla al lazo de control de altura de vuelo del cuadricóptero. Para llevar a cabo lo anteriormente mencionado se desarrollará un algoritmo en escrito en el lenguaje Python con el que se capturará la información de flujo óptico producida por el desplazamiento de acuerdo a una trayectoria seguida por el sensor. Esta trayectoria capturada deberá coincidir con el recorrido realizado con el sensor.

7.3.2 Captura de desplazamiento en los ejes coordenados XY

En esta sesión se describe el hardware y el software usados para la captura de información proveniente del sensor de flujo óptico, además de los algoritmos para el tratamiento de la información obtenida.

Para este proyecto se seleccionó el sensor PX4FLOW, compuesto por un microprocesador Stm32f405, 168 MHz Cortex M4F (gflops 128 + 64 KB de RAM), cámara CMOS MT9V034, giróscopo y sonar para las mediciones de altura. La cámara tiene una resolución de 752 X480 pixeles, pero la imagen es dividida en una matriz de 4X4. Esta puede ser usada con diferentes lentes, pero en este caso se usará uno con una distancia focal de 16 mm.

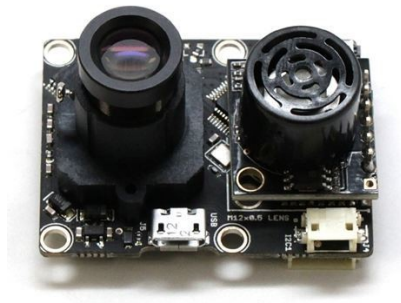


Figura 35: Sensor de Flujo Óptico PX4Flow

El fabricante proporciona las máximas medidas teóricas para diferentes distancias focales como se muestra en la tabla 1. Sin embargo resultados prácticos reportados en la tesis [43] indican que medidas validas están restringidas a una altura máxima de 5 metros y una máxima velocidad horizontal de 2 m/s a una altitud de 1.5m.

Altura	1m	3m	10m
16 mm lens	2.4 m/s	7.2 m/s	24 m/s
8 mm lens	4.8 m/s	14.4 m/s	48 m/s
6mm lens	6.4 m/s	19.2 m/s	64 m/s
4 mm lens	9.6 m/s	28.8 m/s	96 m/s

Tabla 1: Velocidad vs distancia focal




Adicionalmente también es importante tener en cuenta que el sensor por diseño no calcula el flujo óptico en distancias menores a 30 cm, por lo cual se debe respetar esta altura mínima en la plataforma de experimentación para obtener mediciones válidas, además la velocidad de navegación debe ser menor a 2 m/s como se indicó anteriormente.

La Raspberry Pi 2 B+ fue seleccionada como hardware de captura y procesamiento de la información. La Raspberry Pi es un computador de tamaño reducido basado en el procesador ARM11 con frecuencia de 900MHZ, 1GB de memoria RAM. El sistema operativo incorporado está basado en la distribución Debian de Linux. Gracias a su bajo costo y su disponibilidad en el comercio la han convertido en uno de los sistemas embebidos con mayor popularidad en el desarrollo de diversas aplicaciones.

Para la captura de la información de flujo óptico se implementó el paquete PyPX4Flow desarrollado por Simon Dlevi, escrito en el lenguaje Python. Esta librería recupera los datos del sensor a través de una interfaz de comunicación serial para lo cual hay que instalar la dependencia PySerial sobre la tarjeta Raspberry Pi.

La información de flujo óptico que el sensor proporciona es obtenida a través de la implementación del algoritmo SAD (sum of absolute differences) como se muestra en la figura 36. El valor SAD de un bloque de referencia de píxeles del fotograma actual y anterior se compara con los valores SAD dentro del área de búsqueda. La posición de la mejor coincidencia en el área de búsqueda se selecciona como el valor de flujo resultante

en la dirección X y Y, para calcular la velocidad V_x y V_y se estima la altura de la cámara por medio del sensor ultrasónico y la distancia focal del lente. xxx 

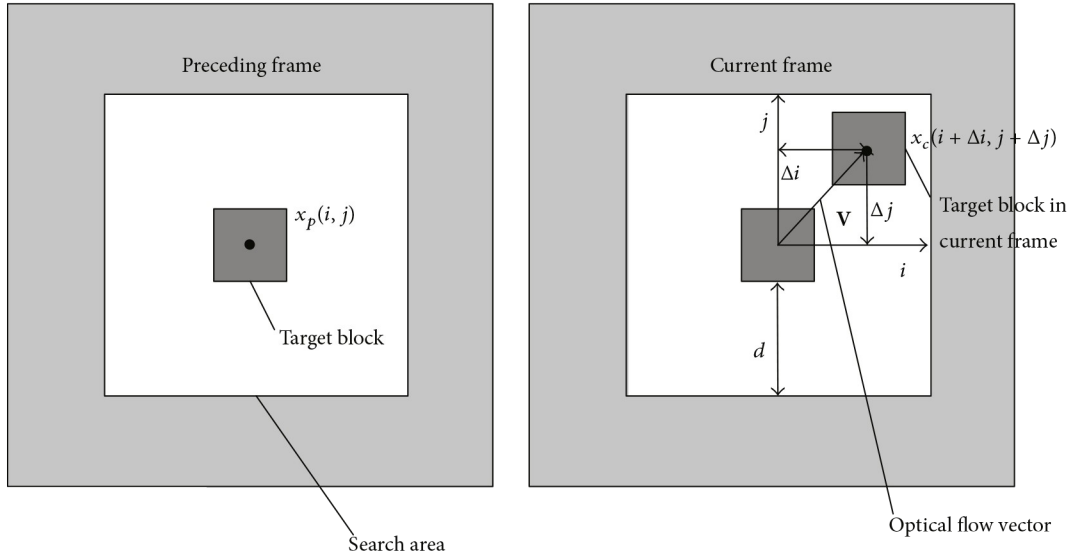


Figure 36: Representación gráfica del algoritmo SAD

Con el método *getFlowComp()* se obtiene el flujo óptico en metros por segundo, en donde *flowCompX* y *flowCompY* corresponden a las velocidades en la direcciones X y Y respectivamente de acuerdo al desplazamiento del sensor sobre una superficie.

$$flowCompX, flowCompY = getFlowComp()$$

Para convertir estas velocidades en desplazamiento utilizando un sistema de coordenadas XY se procede a la integración, donde dt es el tiempo que transcurre entre la toma de cada dato, para este caso se tomó un tiempo de muestreo de 0.1 segundos. Cada dato producido se almacena en las variables X_{accum} y Y_{accum} , las cuales se registran en un archivo de texto plano para su posterior análisis.

```

X_acum += flowCompX * dt
Y_acum += flowCompY * dt

logfile = open(datalog.cvs, 'w')
logfile.write('%+3.3f, %+3.3f' % (X_accum, Y_accum))

```

7.4 PRUEBAS DE CAPTURA DE DESPLAZAMIENTO

De las pruebas realizadas se puede observar que el sensor funciona satisfactoriamente en interiores como en exteriores, permitiendo obtener las coordenadas de desplazamiento de este sobre superficies terrestres, entregando información mas precisa que la habitualmente se obtiene con sensores convencionales como el GPS, que pueden presentar un error de hasta dos metros en sus mediciones, incluso puede reemplazar la odometría en donde el uso del GPS es negado.

Adicionalmente, se debe tener en cuenta las restricciones propias del sistema de captura, tales como una baja velocidad de movimiento del sensor sobre la superficie sobre la que se navega para evitar el problema que se puede generar cuando la diferencia de posición de dos frames es pequeña, esto también se puede controlar con una alta tasa de muestreo de la cámara.

7.5 DESARROLLO DE UN PRIMER PROTOTIPO TERRESTRE

7.5.1 Objetivo de desarrollo del prototipo terrestre inicial

El desarrollo de esta prueba tiene como objeto comprobar la viabilidad del flujo óptico como metodología para el cálculo de la odometría en un robot móvil de tal manera que permita obtener la posición del vehículo en cada momento expresada mediante coordenadas

XY cuando este navega en un ambiente desconocido. En esta prueba no se consideran obstáculos ni el control de una trayectoria específica, lo que se busca es que el sensor capte de una manera aceptable el desplazamiento del robot de acuerdo a una trayectoria preestablecida en él.

7.5.2 Generación de pruebas de navegación

Para realizar las pruebas se seleccionó un lugar a cielo abierto dentro de la universidad (UCN), el sitio es una superficie plana dura y libre de obstáculos, se utilizó luz natural, este aspecto es relevante, debido a que se empleó un sistema de visión artificial que depende altamente de la luminosidad sobre el terreno sobre el cual navega (ver figura 37).

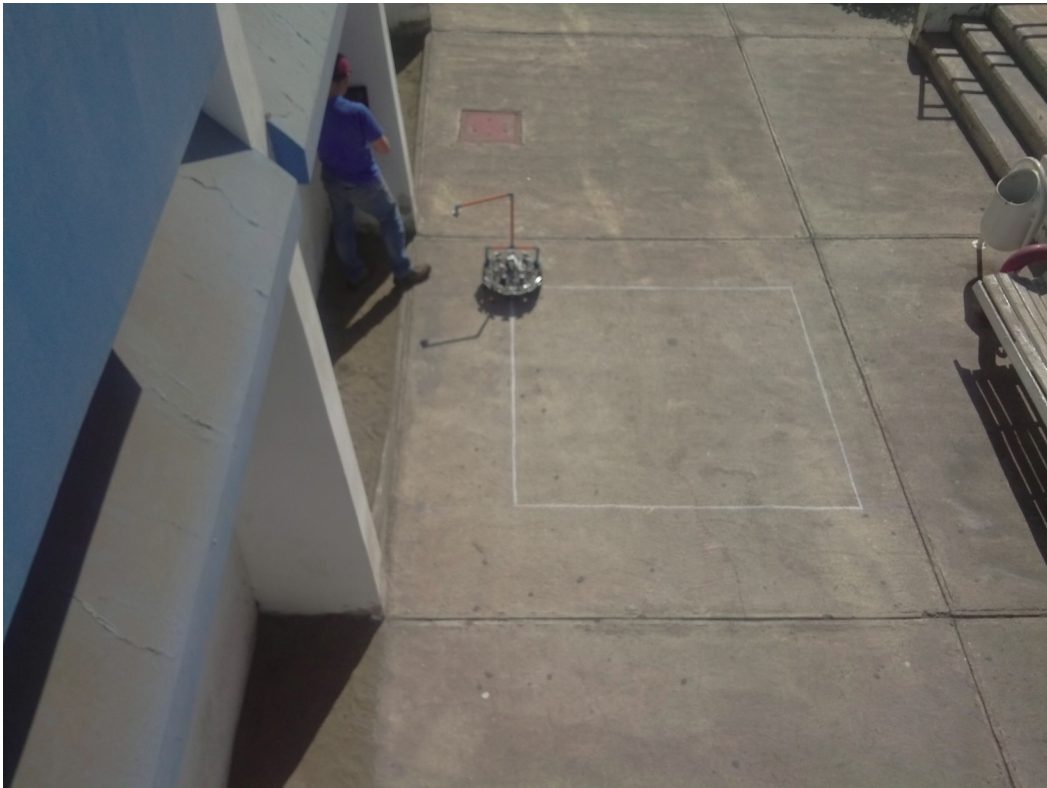


Figura 37: Escenario de prueba de navegación terrestre.

El robot móvil “Robot Base Full Kit” mostrado en la figura 38, fue desarrollado por la empresa Parallax Inc, cuenta con dos ruedas neumáticas de 6 pulgadas de diámetro, ubicadas de forma diametralmente opuestas con una separación de 16 pulgadas entre ellas. Su hardware de control está compuesto por la Tarjeta Propeller Platform USB, 8-core Parallax Propeller chip que permite el control del hardware, el driver HB-25 Motor controller que controla el sentido y velocidad de una rueda mediante de un ancho de pulso variable y motores brushless de corriente continua, provistos de reductores de velocidad y encoders de cuadratura de 36 posiciones.

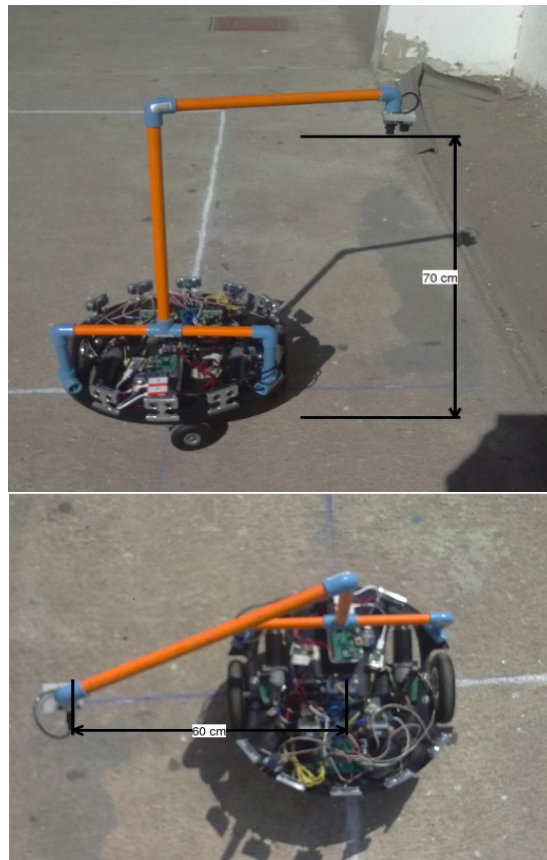


Figura 38: Robot Parallax junto al sensor PX4Flow

Para incorporar el sensor de flujo óptico sobre el robot se construyó una estructura en tubos PVC y se adecuó sobre este como se muestra en la figura 38, de tal manera que el sensor queda a 70 cm de distancia sobre el suelo y a 60 cm de distancia del centro geométrico del robot. Adicionalmente se ha incorporado la tarjeta Raspberry Pi, esta se encarga de la captura la información de flujo óptico proporcionada por el sensor a través de la interfaz serial USB y también permite coordinar la comunicación entre el hardware de control con el computador portátil externo, enlazados mediante el protocolo SSH. Desde el computador portátil se envían las ordenes de control hacia la Raspberry Pi para que esta ejecute la trayectoria que se desea seguir y es quien le asigna los valores al control del hardware para que se produzca el desplazamiento del robot.



*Figura 39: Robot diferencial
PARALLAX*

7.5.3 Procedimiento

El vehículo móvil se ubicó en el entorno seleccionado para la prueba, luego se establece comunicación mediante un enlace SSH desde el cual se lanza el algoritmo con la cual se le envía órdenes desde la Raspberry Pi hacia el sistema de control de hardware y este a su vez le retorna información de su estado actual. El sistema anterior funciona de la siguiente manera:

- Se establece el enlace de comunicación (PC-Raspberry Pi) - Robot (control de hardware).
- Se verifica posición inicial del robot, este parte de la coordenada (0, 0).
- Se establece la tarea que se desea realiza, en este caso se definió una trayectoria cuadrada de 2 x 2 metros. Esta trayectoria se programó utilizando el lenguaje Python, en donde se generó una matriz con los valores de avance (000) y giro (000) expresados mediante tres dígitos ASCII, ambos string equivalen a un valor decimal que representan la cantidad en centímetros y en grados respectivamente que corresponde a la orden de avance que deben cumplir el robot durante el desplazamiento. El valor máximo permitido en una orden para el avance son 999 cm y para el giro son 180 grados.
- Se verifica si el robot está listo para recibir órdenes, esto se hace enviando el carácter en ASCII "L", el cual retorna el string "+00001" si está libre y "+00000" si está ocupado.
- Sólo se envía una orden de avance o de giro a la vez desde la Raspberry Pi cada vez que el robot queda en el estado libre según sea el caso hasta completar el cuadrado. Cuando esta condición "libre" se cumple se envía el carácter "M" que indica la orden de movimiento más la distancia "000" requerida acompañada de un "0" que indica que el movimiento es de avance (ejemplo, "M0000"). Para el giro se envía el carácter "G" más el ángulo de giro 000 acompañado de "0" que indica que el giro es en sentido del reloj ó "1" en contra.
- El robot, de acuerdo a su odometría, navega basado en una arquitectura deliberativa basada en el paradigma "sensorización-planificación-actuación" de acuerdo a la

información recibida hasta complementar la misión asignada, también es importante recalcar que este no responde a factores externos como obstáculos estáticos o dinámicos.

- Paralelamente al desplazamiento del robot, el sensor de flujo óptico proporciona el desplazamiento **generados** mediante la captura de las coordenadas XY. Estos datos se almacenan en un **fichero** de texto plano con extensión *.csv para su posterior análisis.

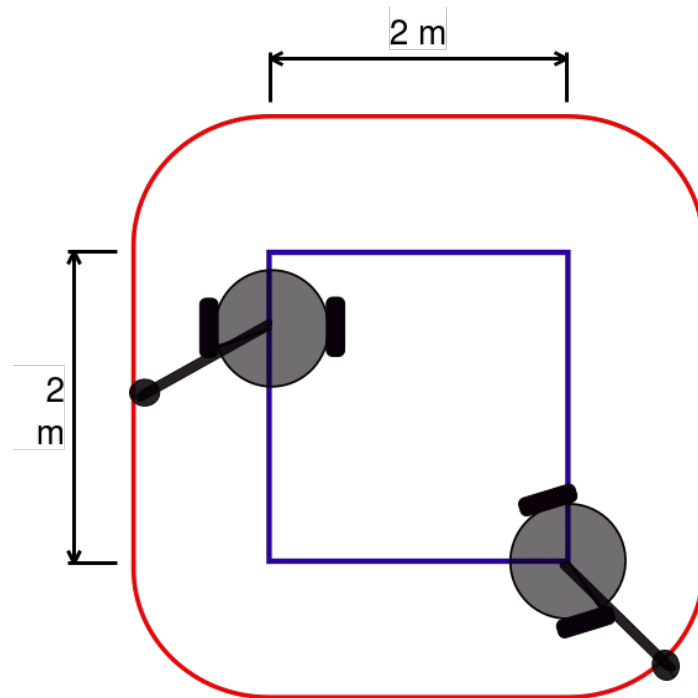


Figura 40: Trayectoria vista superior

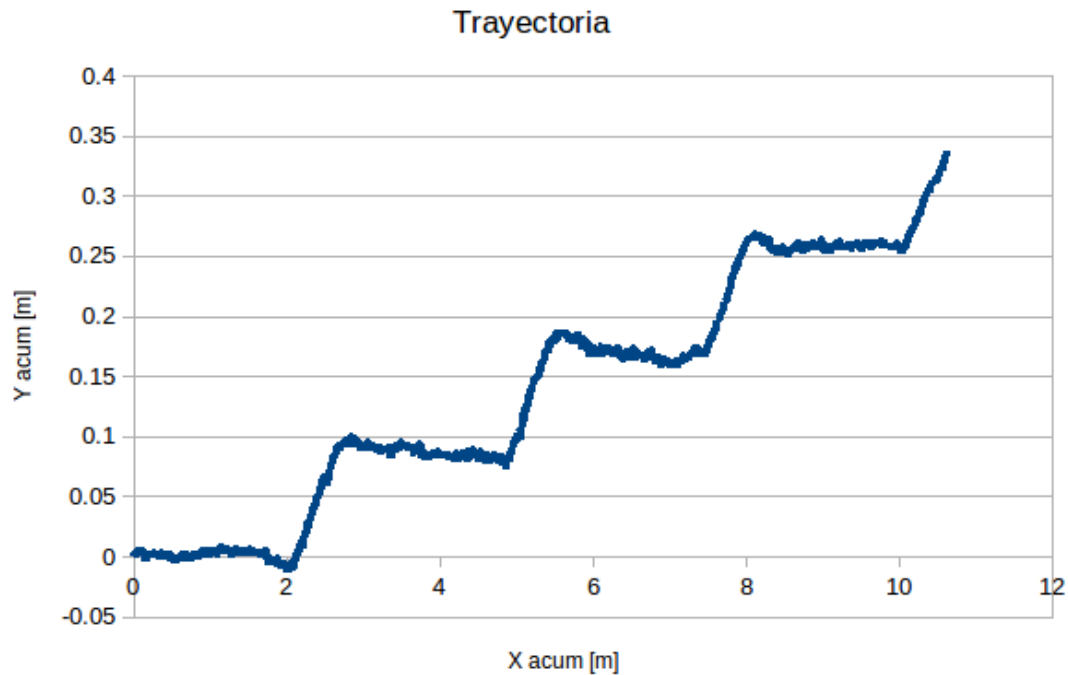


Figura 41: Trayectoria XY realizada

7.5.4 Análisis de las pruebas

En la figura 41 se muestra la trayectoria seguida por el robot, se ve como en sentido del eje X se acumula el desplazamiento del robot expresado en metros, este aumento corresponde a los trayectos de dos metros por cada lado del cuadrado que es recorrido por el robot. También vemos como el desplazamiento total suma 10 metros cuando debería ser 8 metros, esto se debe al error acumulado en cada giro del robot cuando este completa los dos metros recorridos, recordando la figura 38 el sensor de flujo óptico se instaló en un costado lateral del robot y cuando este gira 90° sobre su propio eje al completar el trayecto de 2 metros el sensor sufre un desplazamiento en el eje tanto en el eje X como en Y describiendo un arco,

aun cuando el robot no se ha desplazado, solo ha cambiado su orientación, visto en la figura 40 en donde el recuadro azul es la trayectoria del robot en torno a su centro geométrico y en rojo vemos la trayectoria del desplazamiento del robot en torno al sensor ubicado lateralmente.

7.6 PREPARACIÓN DE UN CUADRICÓPTERO PARA LA NAVEGACIÓN CON FLUJO ÓPTICO

7.6.1 Descripción del cuadricóptero

Se desea utilizar el cuadricóptero comercial Bebop 2 Power (Figura 35), fabricado por la casa Parrot. Este UAV es particularmente adecuado para vuelos al aire libre y captura de vídeo en alta definición, puede ser controlado con radio-controladores propio o a través de un aplicativo móvil para Smartphone o tablet. Su configuración es en equis como se muestra en la figura 19. El drone está provisto de sensores a bordo para la navegación autónoma a través del uso GPS para la guianza. También está dotado de una cámara frontal para fotografía. El UAV está provisto de motores brushless (sin escobillas). Cada uno de ellos es controlado por un ESC (Electronic Speed Controller) o también llamado variador.

Componentes técnicos:

- Diámetro del eje del motor (con exclusión de las hélices): 565 mm
- Diámetro exterior (incluidas las hélices): 844 mm
- Altura total: 203 mm
- Tamaño comprimido (brazos, hélices y tren de aterrizaje eliminados): 616 x 146 x 103 mm

- Peso (excluyendo la batería y el receptor RC): 1,05 kg
- Peso, listo para el vuelo (incluyendo 5300 mAh de la batería y el receptor Spektrum AR610): 1,47 kg
- Peso máximo de despegue (MTOW): 2,8 kg
- Carga máxima útil (excluyendo una batería de 5300 mAh): 1,4 kg



Figura 42: Cuadricóptero Bebop 2 Power, configuración X, tomado de <https://www.parrot.com/es/drones/parrot-bebop-2-power/piezas-repuesto> (2019)

Al cuadricóptero se le agregará un sensor de flujo óptico y uno de ultrasonido externo desarrollado por 3DRobotics (ver Figura 21) que proporcionan información de velocidad del dron con respecto al suelo, posición y altura, aunque el dron Bebop 2 cuenta con su propio sensor de flujo óptico, se decidió incorporar otro externo con el fin de obtener una comparación del desempeño de ambos sensores y confirmar la viabilidad de implementación de este tipo de sensores para la obtención de la odometría de un cuadricóptero cuando este vuela a baja altura o en ambientes con GPS negado.



El sensor px4flow solo cuenta con puertos de comunicación seriales (físicos) y el drone no permite la incorporación directa de este sensor, por lo tanto se decidió empotrarlo en una Raspberry Pi Zero W, este sistema embebido cuenta con la posibilidad de comunicarse a través de puertos inalámbricos como puertos físicos.

La Raspberry Pi Zero W (ver figura 39) es una computadora de baja prestaciones basado en el procesador BCM2835 de un solo núcleo y sus dimensiones tan solo son de 65*30 mm. Para esta tarjeta se han desarrollado varios sistemas operativos, en este caso se optó por distribución Raspbian Stretch, ya que este es una sistema estable y ampliamente usado, ademas cuenta con un buen soporte prestado por los desarrolladores de este. Raspbian es una distribución libre del sistema operativo GNU/Linux basado en Debian dedicado a los sistemas embebidos Raspberry Pi.

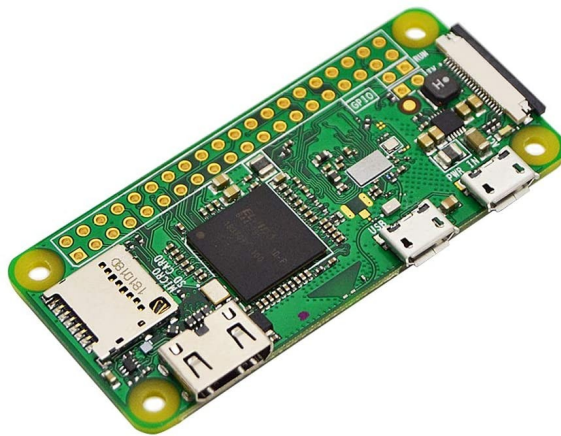


Figura 43: Raspberry Pi Zero W

Especificaciones técnicas de la Raspberry Pi Zero W

- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- 802,11b/g/n wireless LAN
- 1GHz, single-core CPU
- 512MB RAM
- Mini HDMI and USB On-The-Go ports
- Micro USB power
- HAT-compatible 40-pin header
- Composite video and reset headers
- CSI camera connector

7.6.2 Instalación de librerías para captura de flujo óptico

Luego de haber confirmado la comunicación del dron con la Pixhawk, se migro hacia la tarjeta Raspberry Pi, puesto que se ha decidido que el sistema navegación desarrollado irá empotrado en el propio dron en donde irán las rutinas de captura y procesamiento de flujo óptico, como también las ordenes de control de navegación de este, con lo cual se pretende demostrar que la captura del desplazamiento del flujo óptico permite dotar de autonomía al vehículo aéreo sin que sea necesario del radio operador para un vuelo a baja altura.

7.7 SOFTWARE

A continuación se describe el software utilizado para interactuar con el dron aéreo que permite la interacción de este con el computador y el posterior desarrollo de algoritmos de control.

7.7.1 Ubuntu



Ubuntu es el sistema elegido para implementar ROS. Es un sistema operativo basado en GNU/Linux que se distribuye como software libre. Está orientado para el usuario promedio, con una interfaz de usuario intuitiva y fácil de usar. Está compuesto compuesto de múltiple software libre o de código abierto. Es una de la distribuciones de Linux mas utilizadas y es ampliamente usado como servidor web [44]. Para esta tesis se eligió la distribución de Ubuntu 16.04 LTS conocida, también como Xenial Xerus. La elección de este sistema operativo se debe a que es la distribución si bien no es la mas reciente, pues actualmente existe la distribución 20.04 LTS, si es la más estable y con mayor soporte, además el paquete escrito para ROS *bebop_autonomy*, que será descrito mas adelante, está escrito para la distribución de ROS Indigo y ROS Kinetic, en donde ambas solo son compatible para la distribución Xenial elegida.

7.7.2 ROS (Sistema Operativo Robótico)

ROS es un middleware framework de código abierto basado en Linux para su uso modular en aplicaciones robóticas, ya sea para un robot físico o uno simulado, proporcionando abstracción de hardware, control de bajo nivel, comunicación entre procesos y gestión de paquetes de software. Es diseñado por Willow Garage y actualmente es mantenido por la Open Source Robotics Foundation. ROS no debe considerarse un sistema operativo como tal, sino como un conjunto de herramientas de software que permite interactuar conjuntamente con otros sistemas operativos para prestar nuevos servicios cuando se desarrolla software para robots, lo cual permite el ahorro de tiempo y energía al reutilizar estas herramientas organizadas en paquetes.

Nomenclatura y conceptos básicos

- **Repositorio:** es una colección de paquetes que comparte un mismo sistema de control de versiones y pueden actualizarse mediante una herramienta que proporciona el entorno de ROS.
- **Paquete (Package):** es la unidad de organización de software de código ROS. Cada paquete puede contener librerías, nodos, scripts, archivos de configuración, etc.
- **Pila (stack):** es una colección de paquetes con funcionalidad relacionada.
- **Metapaquetes (Metapackages):** son paquetes especializados cuyo propósito es representar a un grupo relacionado de paquetes.
- **Nodo:** es un ejecutable que usa ROS para comunicarse con otros nodos por medio de los topics o servicios. ROS está diseñado para ser modular por lo que un sistema de control robótico tendrá normalmente varios nodos. Un nodo de ROS estará escrito normalmente en C++ o Python, utilizando las librerías roscpp o rospy.
- **Maestro:** el Maestro de ROS provee de un nombre de registro y cuida del resto del grafo de computación. Sin el Maestro, los nodos no podrían encontrarse los unos con los otros, cambiar mensajes o invocar servicios.
- **Servidor de parámetros:** permite guardar y recuperar los datos de una localización específica.

- **Mensajes:** los nodos se comunican entre ellos mediante el uso de mensajes. Un mensaje es una estructura simple de datos que contiene campos determinados (entero, lógico, float, etc).
- **Publisher:** o publicadores, esta función publica el mensaje creado mediante un tópico, son una lista de strings, formados la siguiente manera, [[topic1, [topic1Publisher1...topic1PublisherN]]...]

- **Subscriber:** o subscriptores, esta función escribe el mensaje escuchado en el tópico. Son una lista de strings, formados de la siguiente manera [[topic1, [topic1Subscriber...topic1SubscriberN]]...]
- **Tópicos:**  los mensajes entre nodos se mandan mediante una vía de transporte con la semántica de publicación/suscripción. Un nodo envía un mensaje por medio de una publicación a un tópico dado. El tópico es un nombre que es usado para identificar el contenido del mensaje. El nodo que está interesado en cierto tipo de dato se suscribirá al tópico apropiado. Puede haber múltiples nodos publicadores y suscriptores para un solo tópico y solo **un nodos** que pueda publicar/suscribirse a varios tópicos.
- **Servicios:** se utiliza para interacciones de petición/respuesta. **Un nodos** ofrece un servicio bajo un cierto nombre, y otro nodo solicita dicho servicio enviando una petición y esperando una respuesta.
- **Bags:** es un método para guardar y reproducir datos de mensaje de ROS. Es de utilidad para el desarrollo y las pruebas de algoritmos. Ya que a veces puede ser difícil obtener información útil de algunos sistemas.

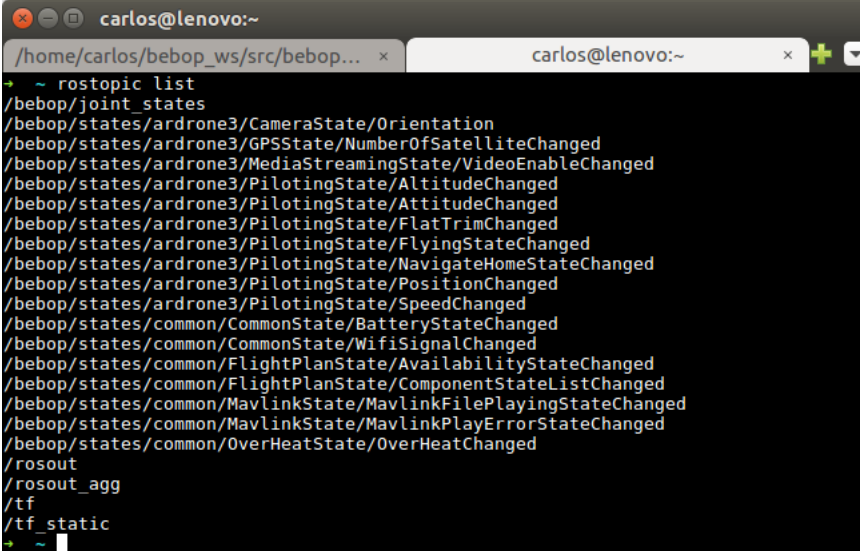
7.7.3 Driver bebop_autonomy

Es un driver para ROS desarrollado para los drone Bebop 1.0 y 2.0 de la marca Parrot, basado en la aplicación oficial ARDroneSDK3 de Parrot. Ofrece las mismas funcionalidades ofrecidas por la interfaz gráfica para dispositivos móviles que proporciona el fabricante, solo que esta herramienta de ROS permite personalizar y desarrollar aplicaciones propias adaptadas a nuestras necesidades de acuerdo a los algoritmos implementados. Este paquete se puede iniciar como un nodo o como un nodelet.

Una vez ejecutado el driver permitirá crear topics, los cuales podrán ser intercambiado por los nodos a través de lo Subscriber para captarlos o Publisher para ponerlos disponibles para otros nodos, permitiendo el intercambio de mensajes entre procesos. Para ejecutarlo es necesario utilizar el comando roslaunch, debido a que este driver es un archivo tipo .launch. El comando tiene la siguiente sintaxis para lanzarlo:

\$roslaunch [carpeta contenedora] [archivo.launch]

\$roslaunch bebop_driver bebop_node.launch

A terminal window titled 'carlos@lenovo:~' showing the output of the 'rostopic list' command. The output lists various ROS topics for the bebop_driver, including joint states, camera state, GPS state, media streaming state, piloting state, and common state topics. The topics are listed in a single column, with some topics having sub-entries. The terminal window has a dark background and a light-colored text. The window title bar shows the user 'carlos@lenovo' and the current directory '~'. There are also some window control icons (minimize, maximize, close) on the left side of the title bar.

```
carlos@lenovo:~  
+ ~ rostopic list  
/bebop/joint_states  
/bebop/states/ardrone3/CameraState/Orientation  
/bebop/states/ardrone3/GPSState/NumberOfSatelliteChanged  
/bebop/states/ardrone3/MediaStreamingState/VideoEnableChanged  
/bebop/states/ardrone3/PilotingState/AltitudeChanged  
/bebop/states/ardrone3/PilotingState/AttitudeChanged  
/bebop/states/ardrone3/PilotingState/FlatTrimChanged  
/bebop/states/ardrone3/PilotingState/FlyingStateChanged  
/bebop/states/ardrone3/PilotingState/NavigateHomeStateChanged  
/bebop/states/ardrone3/PilotingState/PositionChanged  
/bebop/states/ardrone3/PilotingState/SpeedChanged  
/bebop/states/common/CommonState/BatteryStateChanged  
/bebop/states/common/CommonState/WifiSignalChanged  
/bebop/states/common/FlightPlanState/AvailabilityStateChanged  
/bebop/states/common/FlightPlanState/ComponentStateListChanged  
/bebop/states/common/MavlinkState/MavlinkFilePlayingStateChanged  
/bebop/states/common/MavlinkState/MavlinkPlayErrorStateChanged  
/bebop/states/common/OverHeatState/OverHeatChanged  
/rosout  
/rosout_agg  
/tf  
/tf_static  
+ ~
```

Figure 44: Terminal Linux con los tópicos publicados disponibles para drone

Una vez el driver es ejecutado son generados los tópicos que permiten recuperar toda la información del estado del dron como las acciones que se pueden ejecutar sobre él, tales como, la información de la IMU, coordenadas GPS, nivel de batería y el envío de comando para tele operar el dron, entre otras (ver Figura 44).

7.8 SIMULACIÓN CON SPHINX

Sphinx es un simulador gráfico para los drones comerciales de Parrot basado en Gazebo dentro del entorno de ROS. Este **soporte** las referencias Bebop 1 y 2, además de las versiones de los mini Disco, Airbone, Mambo y Swing. Con este simulador podemos obtener las mismas funcionalidades que tenemos con el dron real incluidas las conexiones. El simulador genera una conexión WiFi directa hacia el dron virtual acaparándolo por completo, inhabilitándolo, por ejemplo, para la navegación web. Esto hace que sea necesario usar un adaptador de internet adicional, ya que el del propio computador es usado para generar este enlace. Todos los tópicos utilizados por el dron virtual son los mismos que emplea el dron real cuando se emplea el middleware de programación ROS, lo que facilita la implementación de la simulación y permite migrar al dron **real real** sin tener que hacer ninguna modificación en los scripts generados para este desarrollo. Los tópicos utilizados en la simulación son los siguientes:

Para el desarrollo de este trabajo se utiliza el simulador *Sphinx* debido a que este facilita la implementación de algoritmos de una manera segura y facilita la necesidad de un amplio y seguro espacio de trabajo. Este también permite crear entornos de simulación por medio de archivos de configuración, permitiendo la incorporación de varias características deseadas en el dron, que incluye en acceso a la cámara y la información de estado de batería.

Para ejecutar el simulador sobre nuestra máquina Linux ejecutamos los siguientes pasos:

Se inicia el servicio *Firmwared* instalado ingresando el siguiente comando

```
$sudo systemctl start firmwared.service  
$sudo firmwared
```

A continuación se comprueba que el servicio este activo con el comando siguiente:

```
$ fdc ping  
PONG ← retorno
```

El nombre de la interfaz WiFi utilizada por el drone virtual se comprueba con el comando:

```
$iwconfig
```

Finalmente se lanza la simulación del drone, esto se hace el archivo con extensión *.drone* ubicado junto con la instalación de Parrot-Sphinx en la dirección */opt/parrot-sphinx/usr/share/sphinx/drones/*, allí se encuentran varias versiones de drones simulados. También hay que recordar el nombre de la interfaz WiFi creada, si esta es diferente a wlan0 se debe cambiar por la obtenida en el paso anterior modificandolo en el archivo mencionado, teniendo esto claro podemos ejecutar el siguiente comando para tener el drone dentro de nuestro entorno de simulación:

```
$ sphinx /opt/parrot-sphinx/usr/share/sphinx/drones/bebop2.drone
```

Para conectarse con este drone virtual procedemos de la misma forma como se haría con el drone real, en **esto** caso lo hacemos a través del paquete *bebop_autonomy* para el middleware ROS que se explicó anteriormente, **subscribiendo-nos** a los tópicos que permiten obtener información del drone producidos durante la navegación, y de los tópicos que permiten el control de este. En la figura 44 vemos el drone dentro del entorno de simulación, el cual se encuentra listo para conectarse con él para interactuar a través de los nodos de ROS para que realice la navegación que deseamos programar en este.

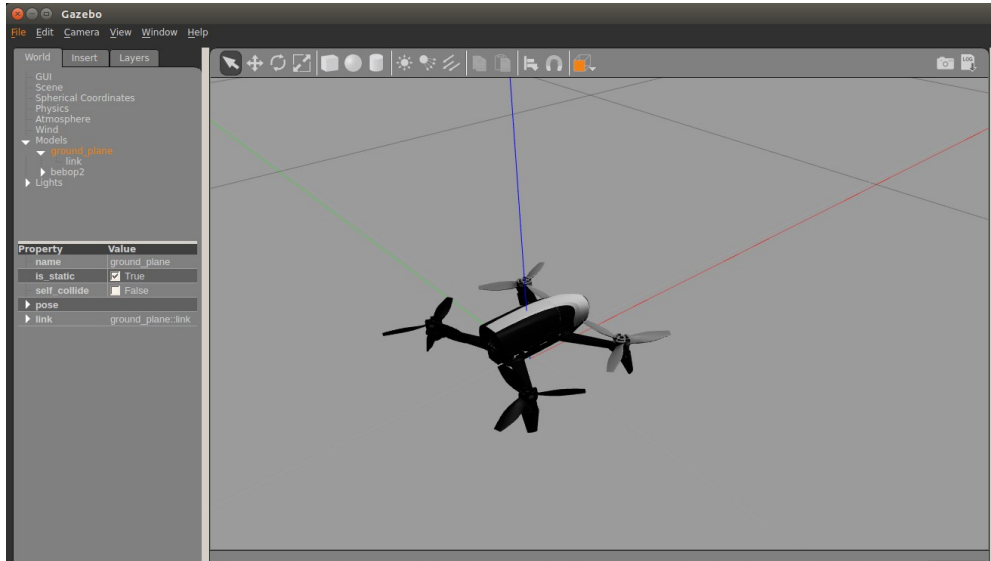


Figure 45: Entorno de simulación Gazebo- para el drone Bebop2 Parrot-Sphinx

7.9 INCORPORACIÓN DE ALGORITMOS DE NAVEGACIÓN AUTÓNOMA

Para la navegación autónoma del drone se han desarrollado varios scripts ~~desarrollado~~ en el lenguaje Python (figura 40). El driver *bebop_autonomy* utilizado contiene los tópicos principales de navegación del drone (despegue, navegación, aterrizaje). En la tabla 2 se resumen los comandos básicos para el control del drone.

TOPIC	TIPO DE MENSAJE	DESCRIPCIÓN
<i>Takeoff</i>	std_msgs/Empty	Inicia el despegue
<i>land</i>	std_msgs/Empty	Aterrizaje seguro
<i>reset</i>	std_msgs/Empty	Paro de emergencia
<i>cmd_vel</i>	std_msgs/Empty	Control de movimientos

Tabla 2: Tópicos principales para el control de movimiento del drone

El tópico *cmd_vel* publica un mensaje con un valor en el rango $[-1, 1]$, en el caso de que se reciban algún otro comando, como por ejemplo, *takeoff*, *land* o *reset*, el tópico de velocidad vuelve a cero.

A continuación se presentan la forma correcta de utilizar los comandos desde un script Python para la creación de un nodo de tipo *Publisher*.

```
pub1 = rospy.Publisher('/bebeop/Takeoff', Empty, queue_size=1)
pub2 = rospy.Publisher('/bebop/land', Empty, queue_size=1)
# despegue del drone
pub1.publish()
# aterrizaje
pub2.publish()
```

Para enviar permitir el movimiento del drone, después de que este ha despegado, el mensaje que se debe publicar debe ser de tipo *geometry_msgs/Twist*, el rango de la variable que componen el mensaje a publicar comprende el rango numérico desde $[-1,1]$, en donde el valor 0 no tiene efecto sobre la velocidad, un valor mayor de cero tiene efecto hacia el avance positivo, siendo 1 el mayor valor de velocidad permitido, para valores negativos permite un avance negativo siendo el -1 el valor máximo permitido. Esto se cumple para cada velocidad lineal y angular permitida para el drone disponible en el tópico *Twist* (ver figura 46).

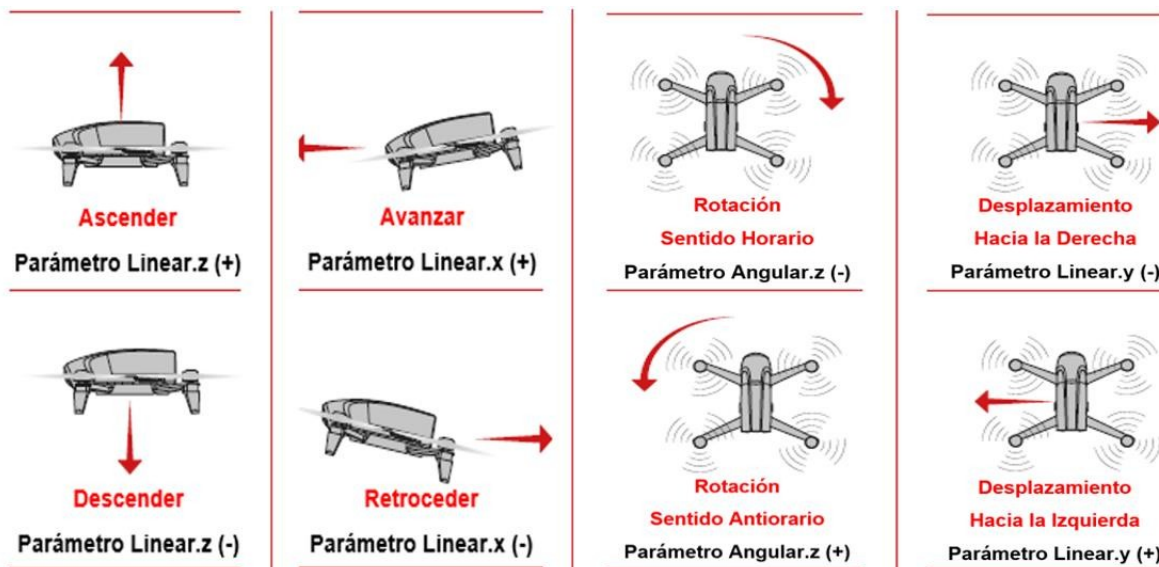


Figure 46: Movimientos generados por los parámetros de control del drone [44]

A continuación se presenta un ejemplo del uso del comando Twist dentro de un script Python.

```
pub = rospy.Publisher('/bebop/cmd_vel', Twist, queue_size = 1)
twist = Twist()
twist.linear.x = x*speed
twist.linear.y = y*speed
twist.linear.z = z*speed
twist.angular.x = 0 twist.angular.y = 0
twist.angular.z = th*turn
pub.publish(twist)
```

7.9.1 Comunicación con el drone (maestro-esclavo)

Como se muestra en la figura 46 tenemos un computador portátil (Lenovo ideapad 330S, I7 8th Gen) actuando como sistema maestro, como ya se ha dicho anteriormente este opera bajo el sistema operativo Ubuntu 16.04 LTS, en el cual se ha instalado la distribución de

ROS Kinetic. El sistema operativo ROS se basa en una arquitectura Maestro-Esclavo, en este desarrollo, como se muestra en la figura 46, es la Laptop, la cual actúa como Maestro y es en la cual se ejecutará ROS, los demás dispositivos conectados a él, en el que se ejecuten nodos vinculándose al Maestro, serán esclavos, el drone Bebop será el Esclavo en este caso.

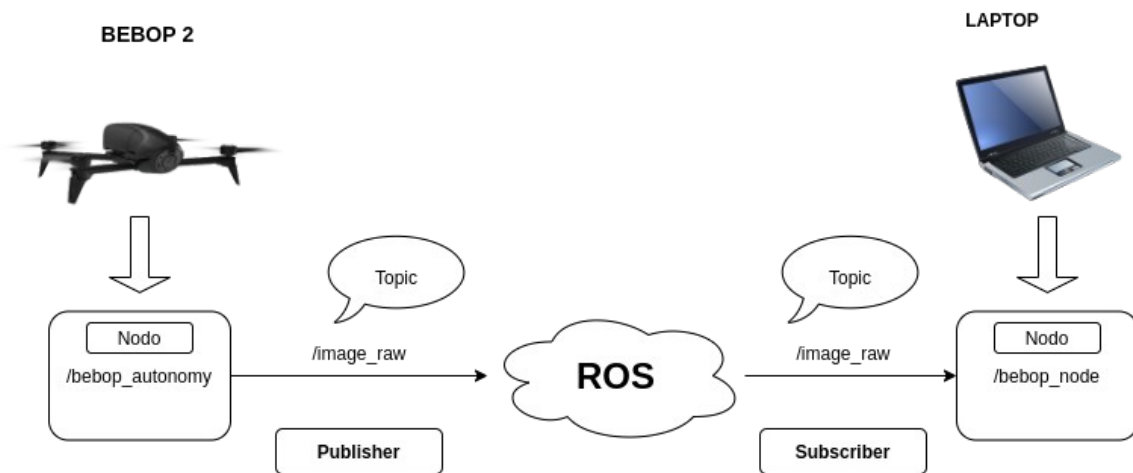


Figure 47: Comunicación esclavo-maestro (Bebop 2- Laptop Linux(ROS)).

Los nodos establecen comunicación entre sí transmitiendo mensajes mediante Tópicos, servicios de llamada a procedimiento remoto y servidor de parámetros, estos nodos están diseñados para operar a bajo nivel, por tanto, los sistemas de control de un robot normalmente se componen de muchos nodos. Por otro lado, por cada tópico publicado es enviado un mensaje específico.

Para el desarrollo de este trabajo se tendrá un *tópico* llamado *cmd_vel* en las que se publicarán las velocidades de desplazamiento asignadas al drone. Estos mensajes son de tipo *geometry_msgs/Twist.msg* y los mensajes *Twist.msg* a su vez están formados por dos vectores de tres componentes, cada uno indica la velocidad lineal y angular de los ejes X, Y y Z.

Para obtener la información de desplazamiento y posición del dron, se utilizará el SDK oficial de Parrot, descrito anteriormente. Este driver permite controlar los movimientos básicos del dron, como despegue, elevación, aterrizaje, girar y navegar.

Después de haberlo instalado, se requiere configurar las condiciones vuelo en las cuales se llevará a cabo la simulación. Esto se hace modificando el archivo ejecutable *.launch* y el de configuración ubicados dentro las carpetas del driver. Al ejecutar el archivo *.launch* se inicializan las condiciones predeterminadas en el archivo de configuración *.yaml* *K* que se encuentra en la carpeta *config* (verfigura 47).

```
states:
  enable_commonstate_batterystatechanged: true
  enable_commonstate_wifisignalchanged: true
  enable_overheatstate_overheatchanged: true
  enable_controllerstate_ispilotingchanged: true
  enable_mavlinkstate_mavlinkfileplayingstatechanged: true
  enable_mavlinkstate_mavlinkplayerrorstatechanged: true
  enable_flightplanstate_availabilitystatechanged: true
  enable_flightplanstate_componentstatelistchanged: true
  enable_pilotingstate_altitudechanged: true
  enable_pilotingstate_flattrimchanged: true
  enable_pilotingstate_flyingstatechanged: true
  enable_pilotingstate_navigatehomestatechanged: true
  enable_pilotingstate_positionchanged: true
  enable_pilotingstate_speedchanged: true
  enable_pilotingstate_attitudechanged: true
  enable_pilotingstate_positionchanged: true
  enable_altitudechanged: true
  enable_autotakeoffmodechanged: true
  enable_mediastreamingstate_videoenablechanged: true
  enable_camerastate_orientation: true
  enable_gpsstate_numberofsatellitechanged: true
  enable_numberofsatellitechanged: true

reset settings: true
publish_odom_tf: true
odom_frame_id: "odom"
cmd_vel_timeout: 0.2

# restricciones para navegación en el entorno gazebo
PilotingSettingsMaxAltitudeCurrent: 15
PilotingSettingsMaxTiltCurrent: 5.0
PilotingSettingsMaxDistanceValue: 10.0
PilotingSettingsNoFlyOverMaxDistanceShouldnotflyover: 1
PilotingSettingsMinAltitudeCurrent: 0.0
SpeedSettingsMaxVerticalSpeedCurrent: 0.2
SpeedSettingsMaxRotationSpeedCurrent: 10
SpeedSettingsOutdoorOutdoor: 0
```

Figure 48: Parámetros de configuración en archivo *config.yaml*

Los nuevos parámetros agregados al archivo `.yaml` según la figura 47 para la configuración del drone son los siguientes:

- **Altitud Máxima:** altura máxima a la que el drone podrá elevarse.

PilotingSettingMaxaltitudeCurrent: 15

- **Máxima Inclinación:** en grados, que el drone tomará para realizar desplazamientos en los ejes X e Y.

PilotinSettingMaxCurrent: 5.0

- **Máxima Distancia:** el drone solo se podrá alejar de la estación en tierra.

PilotingSettingMaxDistanceValue: 10.0

- **Altitud mínima:** distancia a la que el drone puede volar.

PilotingSettingMinAltitudeCurrent: 0.0

- **Máxima Velocidad de Rotación:** grados que permite al drone girar por segundo.

SpeedSettingMaxRotationCurrent: 10

7.9.2 Publicación de tópicos correspondientes al nodo de flujo óptico

Dentro de el espacio de trabajo creado para ROS se procederá a implementar el nodo publicador con el script Python que permite capturar la información del sensor de flujo óptico proveniente del drone Bebop 2 simulado. Para ello se utilizará el tópico `/bebop/odom` que contiene el mensaje tipo `nav_msgs.msg`. A continuación se presenta el ejemplo escrito en Python del correcto uso de los tópicos dentro de un script:

```
subscriber_odom = rospy.Subscriber("/bebop/odom", Odometry, bebop_pose)
```


En donde *bebop_pose* corresponde al llamado de la función que retorna los valores de cuaterniones correspondientes al desplazamiento del drone, como se muestra en el siguiente código:

```
def bebop_pose(msg):
    bebopose = msg.pose.pose
    quat = bebopose.orientation
    roll, pitch, yaw = tf.transformations.euler_from_quaternion(
        (quat.x, quat.y, quat.z, quat.w))
```

7.9.3 Envío de ordenes de navegación al drone

En este nodo se crea un script el ejecutara las ordenes de vuelo del drone, para tal fin se utilizará el mensaje tipo *geometry_msgs/Twist* mediante el tópico *bebop/cmd_vel*. Este tópico esta compuesto por las componentes de velocidad lineal en los ejes *X*, *Y*, *Z* y la velocidad angular para el eje de rotación *yaw*. Estas ordenes de control son necesarias para que permitan que el drone navegue por la trayectoria deseada, y deben ser calculados por un algoritmo de control, en este caso, se utilizo un algoritmo de control PID estándar para las velocidades de cada uno de los ejes (*V_x*, *V_y*, *V_z*). Los comandos para cada eje se obtienen a través del error calculado entre la trayectoria preestablecida a través de una serie de waypoint y la posición real estimada del drone, la cual es proporcionada por el mensaje tipo *nav_msgs.msg* mediante el tópico */bebop/odom*. Un nuevo waypoint o setpoint es asignado al PID cada vez cada uno de estos alcanza su objetivo, es decir, cada que el drone se acerca a la referencia con un error inferior al 10% un nuevo punto de referencia es asignado en cada una de las coordenadas (*x,y,z*) hasta que el trayecto asignado es cumplido en su totalidad por el drone.

Para el procedimiento anterior hay que tener en cuenta que el drone no entrega el valor de el ángulo de rotación *yaw*, este debe ser calculado a través de los cuaterniones mediante la

incorporación de los ángulos de euler, obtenidos mediante el tópico */bebop/odom*, esta conversión se obtiene a través de la función *transformation.euler_from_quaternion(pose.orientation)* proporcionado por la librería *tf* de ROS, tal como se muestra a continuación:

```
#type(pose) = geometry_msgs.msg.Pose
quaternion = (
    pose.orientation.x,
    pose.orientation.y,
    pose.orientation.z,
    pose.orientation.w
)
euler = tf.transformations.euler_from_quaternion(quaternion)
roll = euler[0]
pitch = euler[1]
yaw = euler[2]
```

Para capturar la información de flujo óptico proveniente del sensor externo incorporado al drone y compararlos con la odometría propia del drone Bebop, se diseñó una trayectoria cuadrada, que comprende un recorrido de 4 metros por lado de este trayecto. La trayectoria se generó desarrollando una matriz con los puntos de referencia para cada eje coordenado (X, Y, Z) por los cuales el drone debe navegar. Para lograr que el drone se desplace a través de esta trayectoria deseada se ha incluido un control PID, el cual permite controlar el movimiento del drone basado en la estimación de su posición (ver Tabla 3). El controlador requiere como referencia los puntos deseados $r = (x, y, z, yaw)$ que fueron asignados anteriormente, estos puntos son comparados por la pose actual estimada, en donde la diferencia entre ambos puntos (pose actual estimada y pose deseada) es el error que será minimizado por el control PID enviando al drone los comandos apropiados de velocidad

(v_x , v_y , v_z , v_{yaw}). Estos comando ocasionan el desplazamiento del drone hacia el punto de referencia. A continuación se dará mas detalle de lo anterior mencionado.

	Tr	Mp	Ts	ecc
Kp	Disminuye	Aumenta	Poco efecto	Disminuye
ki	Disminuye	Aumenta	Aumenta	Elimina
kd	Poco efecto	Disminuye	Disminuye	Poco Efecto

Tabla 3: Efectos de las constantes de control respecto a la acción de control

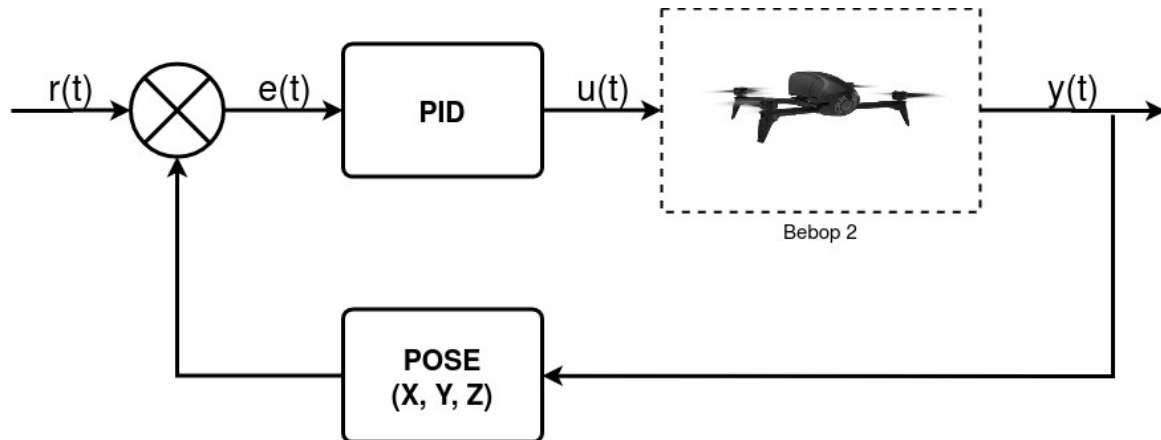


Figure 49: Esquema de control PID para el drone Bebop 2

7.10 PREPARACIÓN DE PRUEBAS FINALES DE NAVEGACIÓN

Para realizar los vuelos experimentales y recopilar datos producidos, se seleccionó un escenario virtual diseñado en el entorno gráfico 3D de Gazebo, por tanto, se han ejecutado las diversas herramientas seleccionadas del sistema ROS y permiten la solución a la implementación de la propuesta desarrollada en esta tesis. Todas las pruebas fueron

efectuadas en un ambiente libre de obstáculos, lo suficientemente amplio para poder realizar un vuelo dentro de una trayectoria cuadrada de 4 metros de lado.

Para el experimento se seleccionó un sector urbano residencial, construido a partir de las herramientas de diseño suministradas por Gazebo (ver figura 49), el cual permite lograr un comportamiento realista de la simulación e implementar los algoritmos diseñados de acuerdo con lo planteado a la tesis.

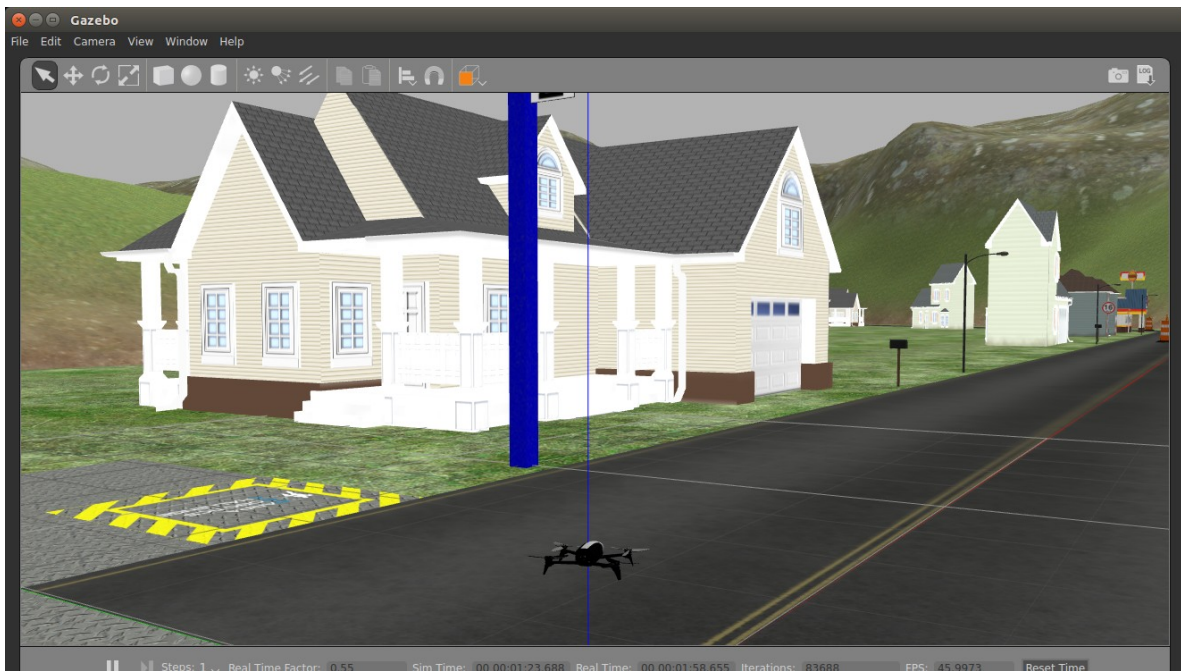


Figure 50: Entorno simulado para el entorno Gazebo y el drone Bebop 2

```
$ sphinx <path/to/my.world> <path/to/my.drone>
```

```
$ sudo sphinx /opt/parrot-sphinx/usr/share/sphinx/worlds/outdoor_5.world  
/opt/parrot-sphinx/usr/share/sphinx/drones/bebop2.drone::with_front_cam=false
```

7.10.1 Prueba de comandos

La siguiente prueba consiste en probar la conexión del drone simulado con los script diseñados, para este fin se creó una interfaz gráfica, en la cual se le envía al drone los comandos de navegación, como el de despegue, aterrizaje, y los demás movimientos permitidos **por el drone a través**. A continuación se muestran los tópicos utilizados para permitir las acciones para la navegación del drone utilizados para el desarrollo de la interfaz mostrada en la figura 50.

```
# ROS
self.takeoff_pub = rospy.Publisher("/bebop/takeoff", Empty,
queue_size=10)
self.land_pub = rospy.Publisher("/bebop/land", Empty, queue_size=10)
self.vel_pub = rospy.Publisher("/bebop/cmd_vel", Twist, queue_size=10)
self.emer_pub = rospy.Publisher("/bebop/reset", Empty, queue_size=10)
```

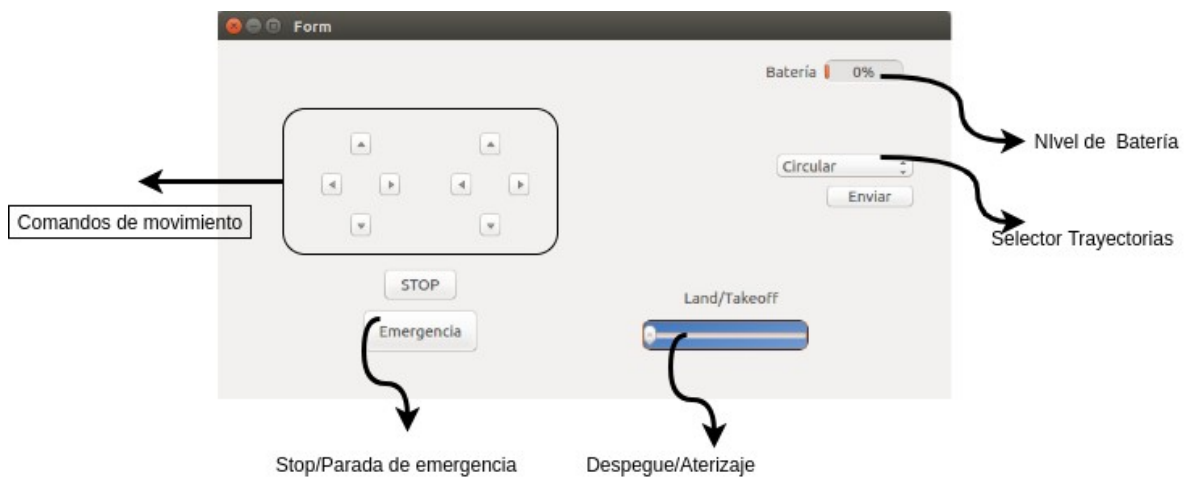


Figure 51: Interfaz gráfica para el control manual del movimiento del drone

Esta interfaz está escrita en Python y la librería para el desarrollo de aplicaciones gráficas PyQt5 y es ejecutada como un nodo ROS a través de comando siguiente:


```
$ rosrun control_manual manualGui.py
```

En las figuras 52 y 53 se representan los desplazamientos producido por el drone al ser operado manualmente por la interfaz gráfica anterior. Estos datos son recuperados del dashboard web publicado por el simulador Parrot-Sphinx (ver figura 51), accesible en <http://localhost:9002>. Por medio de esta interfaz es posible graficar en tiempo real la información producida por el drone durante la simulación, a través de los tópicos publicados mediante la ejecución del driver *bebop_autonomy* y los scripts desarrollados a partir de este.



Figure 52: Sphinx-Dashboard



En la gráfica 52 se presenta la prueba de despegue y aterrizaje del drone. La línea verde representa el desplazamiento del drone en el eje Z, allí se ve como el drone parte de la posición cero al darle la orden de despegue y alcanza la altura de 1 metro aproximadamente y se estabiliza en esta posición, luego de 30 segundos se le da de nuevo la orden de aterrizaje regresando a la posición de origen. En la gráfica 53, a diferencia de la prueba anterior, se probó el desplazamiento del drone en el plano XY. La línea verde representa el despegue y aterrizaje del drone, la azul y purpura el desplazamiento en el plano XY, allí se ve como después de haber despegado y estabilizarse a un metro del suelo, el drone realiza un desplazamiento **fuero** del origen y luego regresa a él, para luego aterrizar de nuevo. De esta manera comprobamos que los script de Python y los nodos de ROS funcionan perfectamente, comunicándose adecuadamente con el drone simulado, lo cual permite **mas**  adelante probar los algoritmos para el vuelo autónomo.

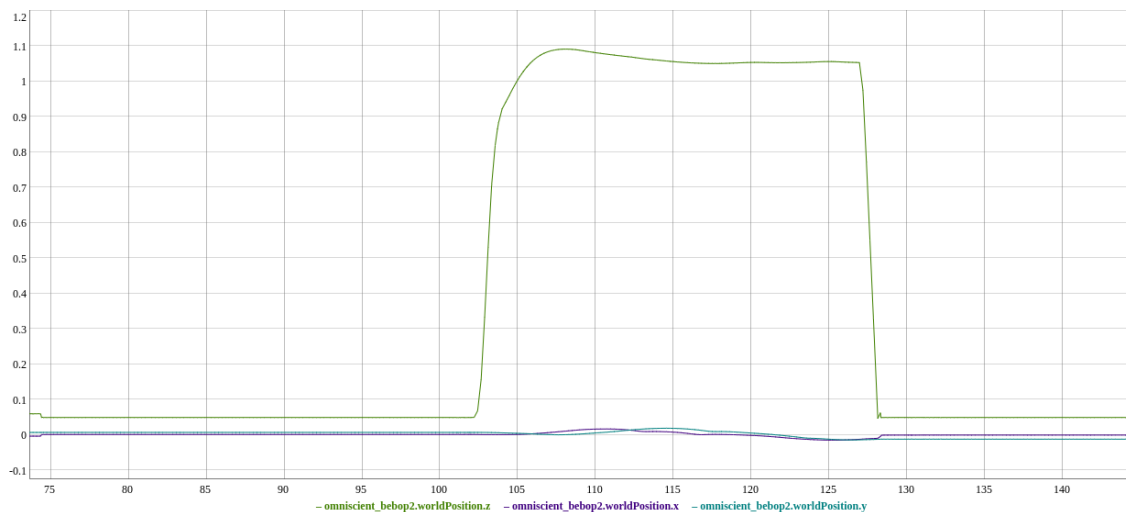


Figure 53 Desplazamiento en [m] en función del tiempo. Leyenda: posición en Z (color verde), posición en X (color azul), posición en Y (color purpura)

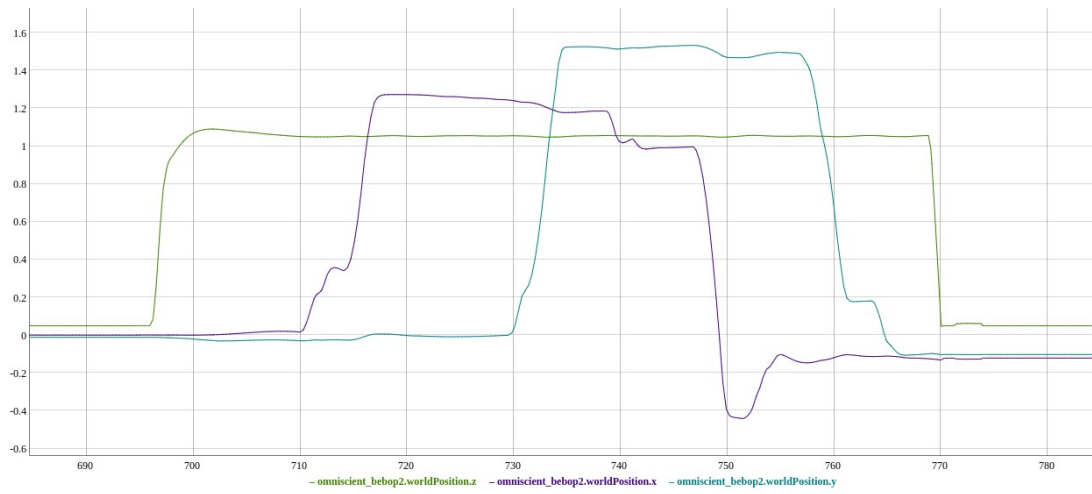


Figure 54: Desplazamiento en [m] en función del tiempo. Leyenda: posición en X (color azul), posición en Y (color purpura), posición en Z (color verde)

8. ANÁLISIS DE RESULTADOS Y CONCLUSIONES DE LA TESIS

8.1 RESULTADOS

Para validar la consecución de los objetivos planteados para esta tesis, se ha realizado una serie de experimentos sobre la solución diseñada, que han sido detallados a lo largo de este documento.

Como se ha descrito anteriormente, se ha utilizado Gazebo como herramienta de simulación junto con el drone virtual Sphinx-Parrot. De este mismo modo, las pruebas realizadas se ajustan a este software. Para ello, se han ejecutado los nodos ROS desarrollados que conforman la solución, y estos a su vez son alimentados con los datos suministrados por el entorno y el drone virtual usado.

En la figura 54 se muestra los resultados producidos del vuelo realizado, como ya se ha mencionado anteriormente, fue asignada una trayectoria cuadra de 4 metros en el plano XY (ver Figura 55) y a un metro de altura del suelo (ver figura 57). En ambas figuras se ve como el drone sigue satisfactoriamente el vuelo de acuerdo a la trayectoria asignada. Este experimento se realizad en varias ocasiones confirmando la fiabilidad del algoritmo usado.

En las gráficas 56, 57 y 58 se evidencia un error cercano al 20% del recorrido obtenido con el deseado en cada coordenada del movimiento producido, esto se debe a la estrategia de control empleada, en donde se utilizó un controlador PID estándar. Este resultado se puede mejorar usando un control mas robusto, como por ejemplo un LQR (por sus siglas en inglés Linear-Quadratic Regulator) o un controlador borroso PID, que podría realizarse en un trabajo a futuro, pero en este documento solo se pretende demostrar la capacidad del sensor de flujo óptico como instrumento de medición del desplazamiento del drone. Por tanto, para nuestro interés, solo es necesario observar que el drone es capaz de obtener su posición en el entorno explorado y seguir un plan de vuelo de acuerdo a unas condiciones iniciales.

Las pruebas con el drone real y la comparación del los sensores de flujo óptico, entre el sensor propio del drone Bebop y el sensor PX4Flow como se había mencionado anteriormente, no se realizaron debido a las condiciones que generó la pandemia y la imposibilidad de encontrar un espacio acorde para aquellas pruebas.

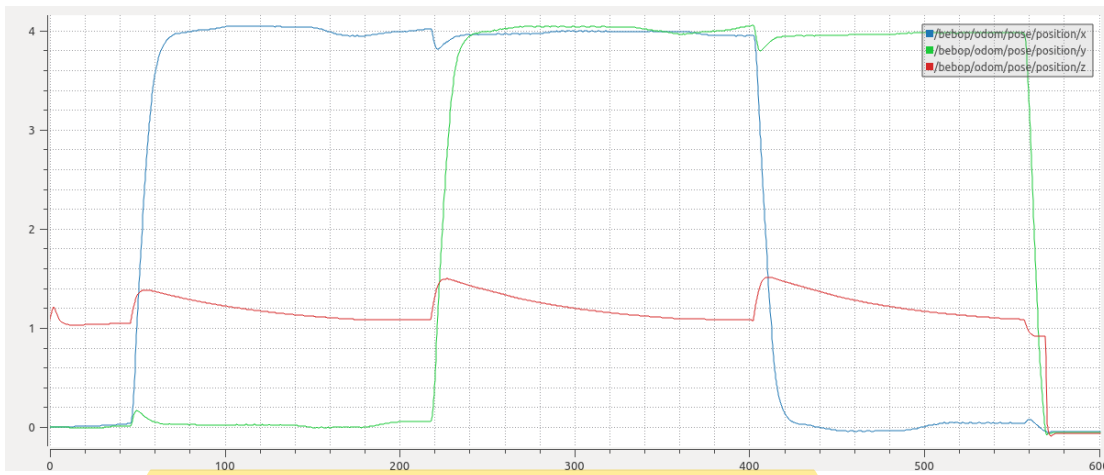


Figure 55: Desplazamiento producido en las coordenadas XYZ

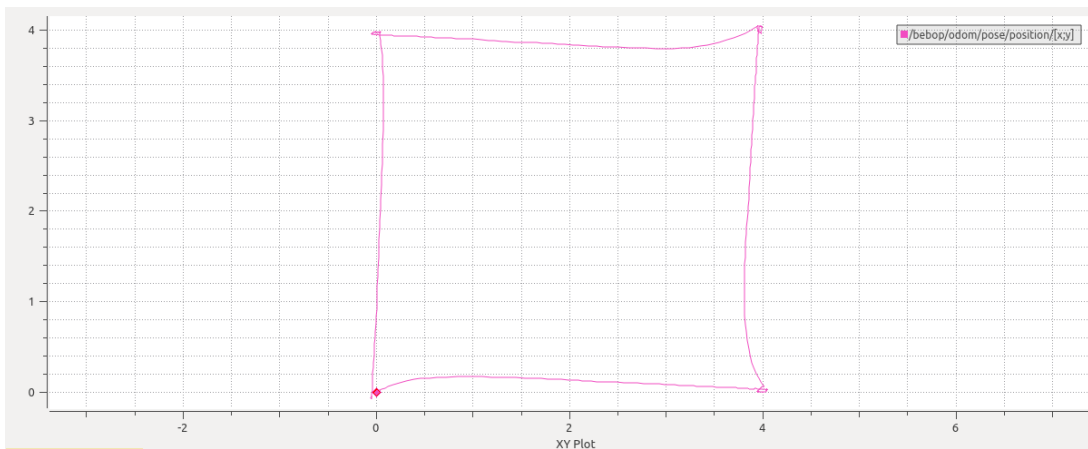


Figure 56: Trayectoria cuadrada, plano XY, 4 x 4 metros.

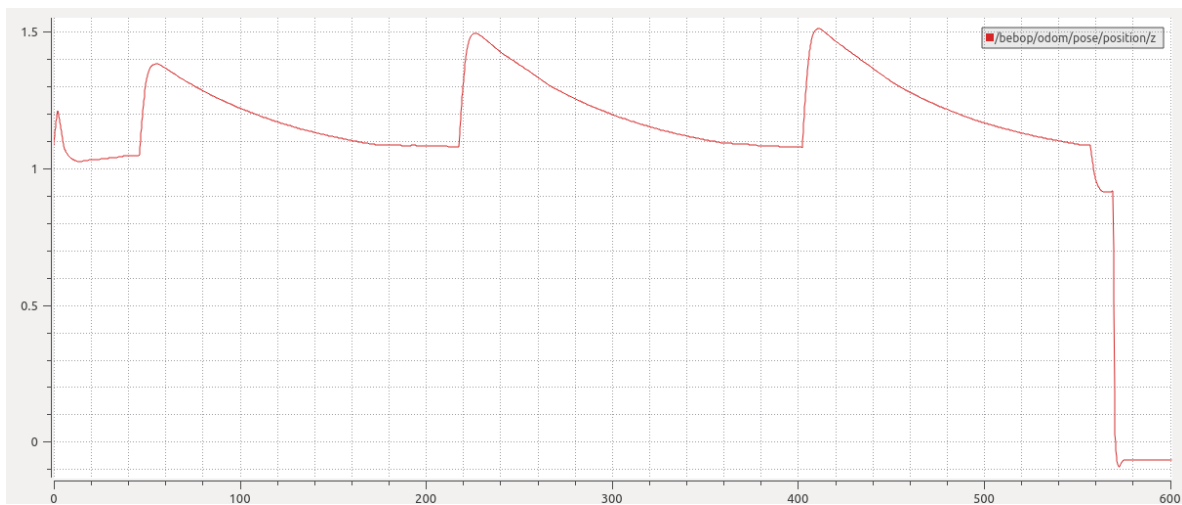


Figure 57: Desplazamiento producido en el eje Z

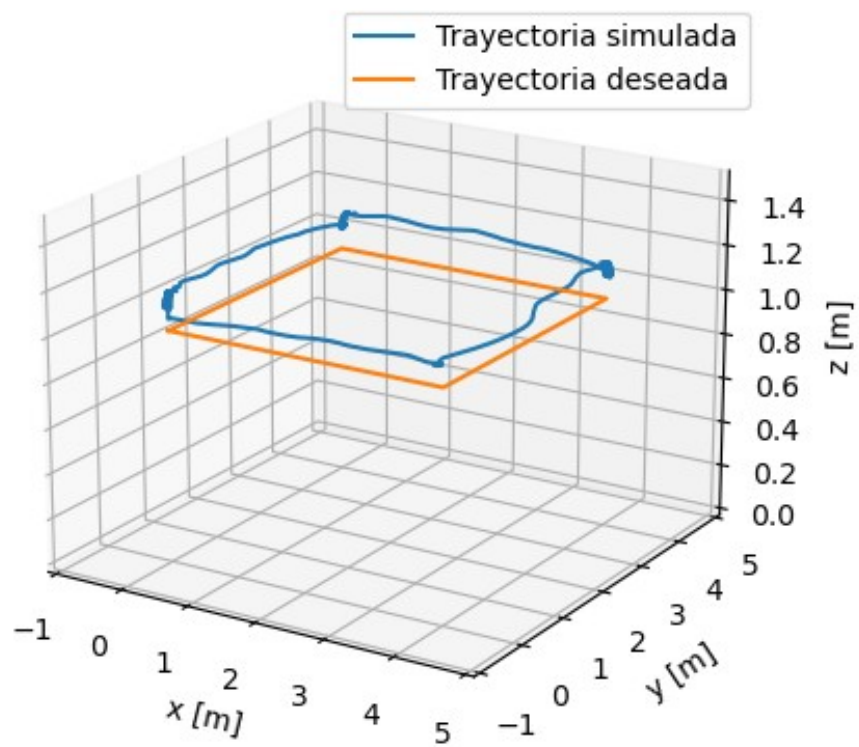



Figure 58: Desplazamiento del drone en las coordenadas XYZ

8.2 CONCLUSIONES

En esta tesis, se presento un sistema de control de vuelo que permite la navegación a baja altura para un drone Parrot Bebop, usando el Sistema Operativo Robótico (ROS). El objetivo del controlador es que el drone navegara dentro de un entorno de Gazebo demostrando que es posible que un drone se puede desplazar a través de una trayectoria definida volando a poca distancia del suelo, en donde se definió que baja altura comprende distancias menores a 1 metro.  Nosotros presentamos primero la implementación de algoritmos para el control de vuelo de manera autónoma para el simulador Gazebo y el drone virtual Sphinx-Parrot, y el controlador de como un paquete ROS. Luego, se presentamos un algoritmo de control, que es utilizado para que el drone pueda cumplir con su misión de manera autónoma. Finalmente, se recopilaron información obtenida del vuelo para demostrar la capacidad de los algoritmos usados para el vuelo a baja altura dentro de un entorno controlado.

Se utilizó el sistema operativo ROS para establecer la comunicación con el drone simulado, Sphinx-Parrot, y el algoritmo de control. De esta forma se logró enviar los comandos de navegación de manera satisfactoria, la navegación autónoma como manual del drone acorde a la trayectoria cuadrada establecida previamente, dentro de un entorno simulado Gazebo. El driver *bebop_autonomy*, permitió crear los nodos con los mensajes para el control de vuelo y la recuperación de datos de navegación del drone. La información recuperada se registró a través del comando *rosviz* que permite almacenar toda la información que se produce durante la ejecución de la aplicación robótica que se requiera analizar posteriormente, y luego fueron representadas gráficamente y posteriormente analizadas por medio del la librería *bagpy* de python.

El sensor Px4Flow fue probado usando un robot móvil terrestre diferencial, en un ambiente controlado sin obstáculos, sobre una superficie plana y rígida. Este sensor demostró que puede obtener las coordenadas de desplazamiento XY. De esta manera, se comprobó que es factible utilizar algoritmos basados en la información de la velocidad proveniente de sensores de flujo óptico, pudiendo reemplazar o complementar con estos la odometría tradicional, basada en sensores inerciales IMU's o sensores tipo encoders, habitualmente usados en robótica móvil terrestre.

Para la evaluación de los algoritmos implementados se realizaron vuelos dentro del entorno simulado, obteniendo pruebas satisfactorias comparando la trayectoria ideal con la obtenida en dicha simulación, los errores producidos son inferiores a al 10%, por lo que demuestra que se puede utilizar el drone a bajas alturas, pues este porcentaje equivalen a un valor inferior a los 10 centímetros, y se podría reducir mucho mas implementando un sistemas de control mas robusto y eficiente.

Finalmente, la estrecha relación entre las herramientas que suministra ROS y Gazebo permite la simulación completa de vehículos aéreos no tripulados (drones), incluida la detección a bajo nivel, la dinámica de vuelo y la detección externa utilizando cualquier sensor disponible para la simulación en Gazebo. Logrando de esta manera el desarrollo de cualquier aplicación robótica de manera simulada antes de ser implementada en un drone real, permitiendo hacer pruebas de manera más rápida y segura, antes de su puesta en marcha en robots reales.

9. PLAN DE ACTIVIDADES

[illegible]

Bibliografía

- [1] Jara Páez , M. A., & Pacheco Cunduri, M. A.. Diseño e implementación de un robot cartesiano, para el montaje de tapa y/o pasador, en el proceso de paletizado. 2013. Escuela Superior Politécnica de Chimborazo.
- [2] Ferrer Ramos, A. Seguimiento de objetos móviles mediante escáner láser radial con el vehículo autónomo ROMEO-3R. 2010. Universidad de Sevilla, Sevilla.
- [3] Araújo, A., Portugal, D., Couceiro, M. S., Sales, J., & Rocha, R. P. Desarrollo de un robot móvil compacto integrado en el middleware ROS. 2014. Revista Iberoamericana de Automática e Informática Industrial.
- [4] Reyes Cortés, F. ROBÓTICA: CONTROL DE ROBOTS MANIPULADORES (Primera ed.). 2011. España: MARCOMBO, S.A.
- [5] Ojeda Bustamante, W., Flores Velázquez, J., & Unland Weiss, H. Drones y sistemas de información geográfica en la ingeniería hidroagrícola. 2014. Instituto Mexicano de Tecnología del Agua, México.
- [6] Vargas Fonseca, L. Desarrollo de algoritmos para el seguimiento de trayectorias de un quadrotor utilizando técnicas modernas de control con álgebra lineal. 2015. Escuela Politécnica Nacional.
- [7] Escamilla Núñez, R. . 2010. Instituto Politécnico Nacional, México D.F.
- [8] Luque Rodríguez, O.. Estudio y desarrollo de un sistema de control de un cuadricóptero. 2014. Universidad del País Vasco, País Vasco.
- [9] Agudelo España, D. A., Silva López, J. S., & Gil López, J. D. Estimación de la localización de un vehículo usando un sistema de visión por computador. 2013. Universidad Tecnológica de Pereira, Pereira.
- [10] Rivas González, E. Localización mediante Differential Evolution de un robot móvil. 2011. Universidad Carlos III de Madrid, Madrid.
- [11] Calderón Jácome, M. T. (s.f.). Control por visión de un cuadricóptero utilizando ROS. 2017. Escuela Politécnica Nacional, Quito.
- [12] Campo C, D., Benjumea G, A., & Vélez S, C. Estudio de pilotos automáticos enfocados en la configuración de ala volante para vehículos aéreos no tripulados. 2014. EAFIT.
- [13] G. S. Control embebido de robots móviles con recursos limitados basados en flujo óptico. 2011. Revista Iberoamericana de Automática e Informática industrial.
- [14] Pico Villalpando, A. Diseño e implementación de un sistema de control para un cuadricóptero. 2012. Instituto Politécnico Nacional, México D.F.
- [15] Artymiak, Jacek. LibreOffice Calc Functions and Formulas Tips. 2011. .
- [16] Tresanchez Ribes, M. Aplicación de sensores de flujo óptico para el desarrollo de nuevos sistemas de medida de bajo coste. 2011. Universitat de Lleida, Lleida.

- [17] Chao, H., Gu, Y., & Napolitano, M. A survey of optical flow techniques for UAV navigation applications. . Unmanned Aircraft Systems (ICUAS).
- [18] Riascos Segura, J. S., & Cardona Gallego, Y. A.. Determinación de la cinemática de objetos móviles bajo condiciones controladas mediante imágenes empleando técnicas de flujo óptico. 2015. Universidad Tecnológica de Pereira, Pereira.
- [19] T.W., N. The optical mouse as a two-dimensional displacement sensor. 2003. Sensor and Actuators A107.
- [20] Ghosh, R., V, K., & V, M.. Indoor Positioning and Indoor Navigation (IPIN). 2013. 2014 International Conference.
- [21] Pei, Y., & L, K. Online robot odometry calibration over multiple regions classified by floor colour. 2015. Mechatronics and Automation (ICMA).
- [22] A. Mrzic, S. K., Velagic, J., & Osmic, N. Optimization based algorithm for correction of systematic odometry errors of mobile robot. 2013. Control Conference (ASCC).
- [23] Azizi, A., & Vossoughi, G. Empirical study on effect of surface texture and grain size on displacement measurement of optical flow sensors. 2014. Robotics and Mechatronics (ICRoM).
- [24] Roskilly, A., & Norman N, R. Investigations into the effects of illumination and acceleration on optical mouse sensors as contact-free 2D measurement devices. 2009. Sensors and Actuators A 149.
- [25] Hyun, D., Yang, H., Park, H., & Park, H. A dead reckoning sensor system and a tracking algorithm for mobile robot. 2009. ICROS-SICE International Joint Conference.
- [26] Valgañón, I., & Pernia, R. The optical mouse for indoor mobile robot odometry measurement. 2006. .
- [27] Minoni, U., & Signorini, A. Low-cost optical motion sensors: An experimental. Sensors and Actuators A 128. 2006. .
- [28] Kreinar, E. J., & Quinn, R. D. Odometry error estimation for a differential drive robot snowplow. Position. 2014. Location and Navigation Symposium - PLANS 2014.
- [29] Yang, L., Chen, I.-M., Guo, Z., Lang, Y., & Li, Y. A Three Degree-of-Freedom Optical Orientation Measurement Method for Spherical Actuator Applications. 2011. Automation Science and Engineering, IEEE Transactions.
- [30] Bell, S. High-Precision Robot Odometry Using an Array of Optical Mice. 2011. (O. C. University, Ed.) Oklahoma.
- [31] Jeng, S.-L., Chieng, W.-H., & Wu, P.-L. Least Squares Approach to Odometry based on Multiple Optical Mouse Sensors. 2010. The 5th IEEE Conference on Industrial Electronics and Applications.
- [32] Prasanth, R., Mehra, R. K., & Boskovic, J. D. Multi-Layer Control Architecture for Unmanned Aerial Vehicles. 2002. Proceedings of the American Control Conference Anchorage.
- [33] Hamel, T., Mahony, R., & Guenard, N. A Practical Visual Servo Control for an Unmanned Aerial Vehicle. 2008. IEEE Transactions on robotics.
- [34] Ostrwski, J. P., Mahony, R., & Altug, E. Control of a quadrotor Helicopter using Visual Feedback. 2002. Proceedings of the 2002 IEEE International Conference on Robotics & Automation Washington, D.C.

- [35] Vianna Raffo, G. MODELADO Y CONTROL DE UNHELICÓPTERO QUADROTOR. 2007. .
- [36] Lei, X., Wang, S., Wu, Y., Wang, T., & Liang, J. A Small Unmanned Aerial Vehicle for Polar Research. 2008. Proceedings of the IEEE International Conference on Automation and Logistics Qingdao.
- [37] Spinka, O. RAMA - a Low-Cost Modular Control System for Unmanned Aerial Vehicles. 2009. Czech Technical University in Prague, Prague.
- [38] Kottenstette., N. Constructive Non-Linear Control Design With Applications to Quad-Rotor and Fixed-Wing Aircraft. 2010. V. University.
- [39] Stanford University HUMMINGBIRD Aerospace Robotics Laboratory. <http://www.stanford.edu/>. 2010. <https://web.stanford.edu/group/ar1/projects/hummingbird-autonomous-helicopter>.
- [40] Romero, H., Salazar, S., & Escareño, J. REVISTA IBEROAMERICANA DE AUTOMATICA E INFORMATICA INDUSTRIAL (RIAI). 2010. Estabilización de un mini helicóptero de cuatro rotores basada en flujo óptico y sensores inerciales.
- [41] Gonzalez Villagómez, J. M. Realimentación visual para el control de un vehículo aéreo cuatrimotor no tripulado. 2010. Universidad de Sevilla, Sevilla.
- [42] Mejías, L., & Campoy, P. (s.f.). COLIBRI: Vehículo Aéreo Autónomo Guiado por Visión para Inspección y Vigilancia.. 2007. U. P. Madrid, Ed.
- [43] Tomáš Novák. Localization of Unmanned Aerial Vehicles Using an Optical Flow in Camera Images. 2017. .
- [44] /statowl.com. Operating System Version Usage. 2020. <https://stats.wikimedia.org/wikimedia/squids/SquidReportOperatingSystems.htm>.