Pontificia Universidad Javeriana
Facultad de Ingeniería

Aterrizaje autónomo de un Quadrotor sobre una plataforma en movimiento

Maestría en Ingeniería electrónica

Álvaro Gerardo Otero Caicedo
Ingeniero Electrónico PUJ

Colombia 2014

CONTEXTO

El presente documento es un proyecto realizado en el ámbito de los vehículos aéreos no tripulados conocidos como UAV. Se da a conocer una técnica de control para un quadrotor basado en el posicionamiento por GPS y procesamiento de imágenes que permita compensar el error dado por el sensor GPS del quadrotor para lograr el aterrizaje autónomo sobre una plataforma de aterrizaje marcada de color rojo. El proceso de control se basa en IBVS, una técnica de control que permite ajustar la posición de un sistema basado en la información extraída por un sensor óptico. El montaje del quadrotor y las plataformas de procesamiento visual y control principal del sistema se basan en código Open Source y Open Hardware

1. INDICE GENERAL

2. INDICE DE FIGURAS

3. INDICE DE TABLAS

4. INTRODUCCIÓN

Los helicópteros son hoy en día uno de los modelos de aeronaves más usado para diferentes tipos de tareas, por su naturaleza estas aeronaves tienen la capacidad de mantener vuelo estacionario gracias a que la sustentación se genera por giro de su hélice (rotor) principal. También se conocen como aeronaves tipo VTOL (Vertical Take off and Landing), una característica particular que genera gran interés a la comunidad que estudia el mejoramiento de las capacidades de vuelo.

En los últimos años se han dado grandes avances tecnológicos en cuanto a sensores de tamaño reducido, microcontroladores, fuentes de alimentación o baterías, lo que han permitido miniaturizar este tipo de aeronaves para ser implementados en distintas aplicaciones civiles y militares donde se requiere dotar a la aeronave de ciertas capacidades autónomas. Este tipo de aeronaves no tripuladas se conocen como UAV (por sus siglas en inglés UnmannedAerialVeihcle) y UAS (por sus siglas en inglés UnnmannedAerialSystem) las cuales se han venido implementando en diferentes tareas donde se dispone de entornos de difícil acceso o simplemente para facilitar algunas tareas como por ejemplo el mapeo de terrenos agrícolas, transporte de mercancía etc (Skrzypietz, 2012).

Cuando hablamos de UAV o UASpodemosclasificarlos de acuerdo al número de rotores, helicópteros (1 rotor) tricóptero (3 rotores) hexacopter (6 rotores) etc. En este proyecto nos centraremos en una aeronave tipo VOLT de cuatro rotores, conocido como Quadrotor.En muchos casos estos vehículos aéreos son utilizados en la automatización de procesos industriales, al igual que en desastres naturales, búsquedas y rescates, inspección de instalaciones y estructuras, vigilancia y realización de mapas de terrenos (Real, 2009).

La implementación y desarrollo de sistemas de control para estos vehículos no es trivial, debido principalmente a la dinámica característica e inherente de los sistemas aerodinámicos, los cuales son multivariables, subactuados, y poseen diversas características no lineales (Moses Bangura, 2012), esto significa que las técnicas clásicas, lineales y mono variables se ven limitadas al generar un sistema de control eficiente, fiable y robusto. El control de estos sistemas requiere la implementación de técnicas avanzadas que permitan considerar las características propias del sistema y las incertidumbres propias de cualquier modelo. En este orden de ideas sería apropiado el uso de técnicas de modelado y control no lineal modernas las cuales permitan lograr un control eficiente, fiable y robusto en las distintas condiciones de vuelo que pueden presentarse (Vuelo autónomo, control de estabilidad, vuelo estacionario, vuelo en un punto fijo, aterrizaje y despegue).

Existen diferentes técnicas para controlar este tipo de aeronaves por ejemplo (Santos, 2010) propone realizar el control de estabilidad de un Quadrotor a partir de un sistema de

control difuso. Otros como (José Luis, 2012) define un sistema que a partir de filtros de Kalman y redes wireless estima la localización o posicionamiento del UAV, aplicando sus resultados, por ejemplo, a realizar vuelo estacionario más preciso en interiores. De igual manera se han aplicado sensores ópticos (cámaras) como por ejemplo en (Lozano, 2011) para realizar la estimación del flujo óptico, el cual analiza el movimiento aparente de los patrones de intensidad de una imagen producidas por el movimiento relativo entre el sensor (cámara) y el entorno, logrando mejorar la estabilización de la aeronave al minimizar el cambio de flujo óptico. Otros tipos de control como el que plantea (K. E. Wenzel, 2010) donde se implementan sistemas de control para lograr un aterrizaje (o despegue) autónomo de un quadrotor.

Actualmente en Colombia ha habido un incremento en los estudios realizados sobre control sobre UAVs, pero poco a poco se ha incrementado el interés sobre de estas aeronaves; trabajos como (Iván F. Mondragón, 2011), que implementan un sistema de seguimiento para un UAV basado en visión artificial son ejemplo de la calidad de trabajos que se han desarrollado en los últimos años. De acuerdo a esto surge un gran interés para desarrollar un proyecto que unifique varías líneas de investigación (de las mencionadas anteriormente) y lograr dar un gran aporte al estado del arte a nivel regional.

La idea de unificar varias líneas de investigación surge de la necesidad de realizar un sistema de control que permita un vuelo autónomo y que satisfaga tareas más complejas de las que actualmente muchos quadrotors son capaces de realizar, por ejemplo, es posible adquirir un quadrotor que tenga la capacidad de volar de manera autónoma basado en seguimientos de puntos GPS (llamados waypoints) sin embargo el interés particular sobre estas aeronaves motiva a complementar estas funcionalidad y mejorarlas con nuevas tareas o mejore desempeño en el controlador. Una de las funcionalidades básicas que puede complementarse es el aterrizaje, ya que dependiendo de la situación puede tornarse una tarea compleja, por ejemplo, el aterrizaje sobre una plataforma móvil, o el aterrizaje en una zona segura, delimitada por unos cuantos metros. En estos aspectos los Quadrotors actuales se ven limitados, ya que poseen un sistema de “guía” basado en un sensor GPS comercial, el cual da una buena aproximación en distancias mayores entre 2 y 5 metros, sin embargo si se requiere aterrizar en un lugar específico el modelo se verá limitado por el error en el posicionamiento. La solución que se plantea en este proyecto es complementar el sistema de posicionamiento del control principal basado en GPS con un segundo sistema de control capaz de detectar una marca que le indique el lugar correcto para realizar el aterrizaje. Para esto implementó un control servo-visual. El control servo visual es una herramienta de control jerárquica (detectar y ejecutar) donde se usa el sensor óptico (cámara) para brindar las entradas al controlador quien debe reaccionar en diferentes acciones definidas. El control se basa en el seguimiento visual de características (color) para proporcionar una posición más exacta de dónde el quadrotor debe aterrizar y así poder corregir el error del GPS.

En este orden de ideas se desarrolló un sistema capaz de posicionarse sobre un punto GPS (waypoint) y realizar un aterrizar guiado (sobre una plataforma fija), basado en visión por computador; todo de manera autónoma (Despegue, posicionamiento, aterrizaje).

El controlador se compone de dos sistemas: Primero un controlador central o control de bajo nivel encargado de estabilizar y dirigir al quadrotor en cualquier dirección, y segundo un controlador de alto nivel capaz de procesar los datos del sistema inercial (IMU), el conjunto de sensores y la información captada por una cámara a bordo con el fin de comandar al control de bajo nivel y así lograr realizar el posicionamiento de precisión del quadrotor sobre un punto GPS.

5. MARCO TEÓRICO

Un quadrotor es un sistema basado en la unión de cuatro rotores independientes, estos son los encargados de proporcionar sustentación y el movimiento en todas las direcciones; La ubicación en la estructura de los rotores generalmente describe una cruz con dos rotores alineados de forma horizontal y dos de forma vertical. Cuando se genera un cambio en la velocidad de giro o el torque en los motores se produce un cambio los ángulos de alabeo, Cabeceo y cuñada (Roll, pitch y yaw en inglés) los cuales describen el movimiento en el espacio del UAV. Un cambio en el ángulo del Alabeo o Roll logra un desplazamiento horizontal, de izquierda a derecha o de derecha a izquierda, un cambio en el ángulo Pitch logra un desplazamiento hacia adelante o hacia atrás, un cambio en Yaw causa un movimiento rotacional sobre el eje vertical de la aeronave. El sistema de control debe ser capaz de controlar estos ángulos para lograr una estabilidad en el quadrotor que le permita volar en cualquier dirección manteniendo siempre una compensación adecuado en los tres ángulos.

Fig. 1.*Descripción de los ángulos fundamentales del QuadrotorAscTecHummingbird (Ascending Technologies, 2014)*

5.1. Modelo del Quadrotor

A pesar de que el trabajo se centra en la temática del control visual, es importante considerar los aspectos físicos del quadrotor, por lo tanto a continuación se realiza una descripción breve y simplificada de la formulación matemática para las leyes físicas que describen la orientación y posición del quadrotor. Sin embargo se sugiere dirigirse a los

apuntes bibliográficos como (Raffo, 2007) donde se realiza en detalle el modelado cinemático y dinámico de un quadrotor.

Para la descripción analítica de la orientación y posición del quadrotor es importante considerar que el modelo se representa mediante las formulaciones de cuerpo rígido en el espacio, donde se consideran 4 fuerzas que actúan sobre el quadrotor en el espacio libre: fuerza de empuje y tres torques o pares

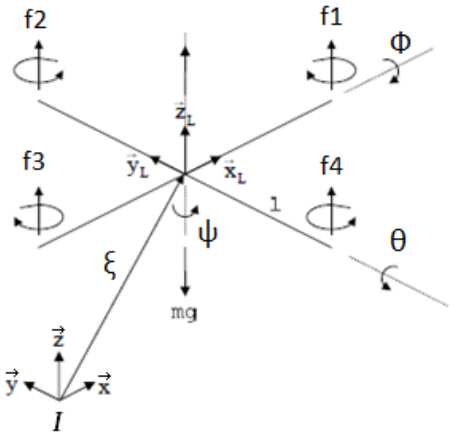

Fig. 2.*Sistema de coordenadas del quadrotor. (Bouabdallah, 2004)*

Las fuerzas f1, f2, f3, y f4 corresponden a la sustentación generado por los motores. Cuando se produce un cambio de iguales proporciones en las cuatro fuerzas se logra cambios en la dirección perpendicular a los motores del quadrotor. Con un cambio conjunto dos más fuerzas es posible generar un direccionamiento en todos los ejes. Por ejemplo un cambio en Ƒ2 y Ƒ4 modifica el ángulo de Roll (ϕ), un cambio en Ƒ1, Ƒ3 modifica el ángulo Pitch (θ) y un cambio en (f1, f3) y (f2, f4) modifica el ángulo yaw (ψ) (para este ángulo es importante tener en cuenta la dirección de giro de los motores).

Adicional a las fuerzas descritas, es necesario considerar los efectos giroscópicos tanto en cuerpo rígido rotando en el espacio, como el de la rotación de las cuatro hélices. En la tabla 1 se mencionan dichos efectos donde Ω corresponde a la velocidad del rotor, J_R corresponde al momento de inercia rotacional que se genera en el eje del rotor, l representa la distancia entre el centro de masa del quadrotor y los rotores, J corresponde al momento de inercia del cuerpo rígido y ϕ, θ y ψ son los ángulos de Tait-Bryan.

Efectos	Fuentes	Formulación
Efectos aerodinámicos	Rotación de los rotores	$C\Omega^2$
	Giro de hélices	
Pares inerciales opuestos	Cambio en la velocidad de rotación de los rotores	$J_R\dot{\Omega}$
Efectos de la gravedad	Posición del centro de masa	l
Efectos giroscópicos	Cambio en la orientación del cuerpo rígido	$J\theta\psi$
	Cambio de la orientación del plano de los rotores	$J_R\Omega\theta, \phi$
Fricción	Todos los movimientos del quadrotor	$C\dot{\phi}, \dot{\theta}, \dot{\psi}$

Tabla 1. *Principales efectos físicos que actúan sobre un quadrotor. (Bouabdallah, 2004)*

La descripción matemática de la posición y la orientación del quadrotor, referenciado a un sistema de coordenado inercial {W}, considera la descripción de éste mediante un sistema de coordenadas ubicando el origen en su centro de masa. De acuerdo a esto se define un sistema de coordenadas en donde se considera que $B = \{\acute{X}_L, \acute{Y}_L, \acute{Z}_L\}$ fijo al quadrotor y fijo al espacio. Adicional a esto, el vector ζ = $\{x, y, z\}$ representa la posición del quadrotor con respecto al sistema inercial W y gracias a esto podemos afirmar que la orientación del quadrotor viene dada por la matriz de rotación $R_w: B \rightarrow W$ donde R_w es una matriz de rotación RPY.

Por otro lado la rotación de un cuerpo rígido puede ser expresada a partir de los ángulos de Tait-Bryan sobre un espacio euclidiano tridimensional como se describe a continuación:

- ✓ **Rotación según $\vec{x}$** : Es el primer giro que correspondiente al ángulo de roll (ϕ) y se realiza alrededor del eje $\vec{x}$.

- ✓ **Rotación según $\vec{y}$** : El segundo giro se realiza alrededor del eje $\vec{y}$ conocido como el ángulo pitch θ.

- ✓ **Rotación según $\vec{z}$** : Es el tercer giro y última rotación corresponde al ángulo de yaw (ψ) el cual se realiza alrededor del eje $\vec{z}$.

A partir de esto podemos definir la matriz de rotación completa de del vector B respecto a al sistema de coordenadas inercial W, el cual representa la orientación de cuerpo rígido rotando sobre cada eje.

La matriz R_w denominada matriz de coseno directa viene dada por:

$$R_w = R(z,\psi) \cdot R(y,\theta) \cdot R(z,\phi) \qquad (1)$$

La matriz de rotación del sistema de coordenadas B es la transpuesta de R_w debido a que es ortogonal. Gracias a esto y a la relación de la derivada de la matriz ortogonal con una matriz anti-simétrica, es posible obtener las ecuaciones cinemáticas de rotación que establecen una relación entre las velocidades angulares del quadrotor.

$$\omega = \begin{bmatrix} 1 & 0 & -sin\theta \\ 0 & cos\phi & sin\phi cos\theta \\ 0 & -sin\phi & cos\phi cos\theta \end{bmatrix} \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \qquad (2)$$

Cuando consideramos los movimientos rotacionales hablamos de las velocidades angulares en los tres ejes del sistema coordenado $[\overrightarrow{x_L}, \overrightarrow{y_L}, \overrightarrow{z_L}]$, velocidad angular de Roll, pitch y yaw. Estas velocidades rotacionales se deben al par generado por la fuerza de giro de los rotores las cuales describen los diferentes momentos en los tres ejes: momento de roll (L), de pitch (M) y de yaw (N). El movimiento trasnacional está caracterizado por las componentes de la velocidad $v_0$en los tres ejes inerciales con relación a la velocidad absoluta del quadrotor V_0donde existe una relación directa entre v_0y V_0dado por la siguiente expresión.

$$v_0 = R_w \bullet V_0 \qquad (3)$$

5.1.1. Representación dinámica del Quadrotor

A partir de la formulación Newton-Euler es posible obtener en el sistema de coordenadas $\boldsymbol{B}$ la dinámica del quadrotor de cuerpo rígido sujeto a fuerzas externas aplicadas al centro de masa.

$$\begin{bmatrix} mI_{3x3} & 0 \\ 0 & J \end{bmatrix} \begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} \omega \times mV \\ \omega \times J\omega \end{bmatrix} = \begin{bmatrix} F + F_d \\ \tau + \tau_d \end{bmatrix} \qquad (4)$$

Donde $J \in R^{3x3}$es el tensor de inercia, I_{3x3} es la matriz identidad, v es el vector de velocidad trasnacional o velocidad lineal, **ω** es el vector de velocidad angular y m la masa del quadrotor.

Ahora bien, considerando el vector de estado $\zeta = [\xi\ \acute{v}\ \eta\ \omega]^T$ donde ξ es la posición, $\acute{v}$ la velocidad lineal representada para el sistema inercial W, el vector de ángulos de rotación del sistema dado por $\eta = [\phi\ \theta\ \psi]$ y ω la velocidad angular del quadrotor. Gracias a esto podemos describir las ecuaciones de cuerpo rígido en la ecuación (5)

$$\begin{cases} \dot{\xi} = v \\ mv' = \dot{R_I} F_B \\ \dot{R_w} = R_w S(w) \\ J\,\dot{\omega} = -\omega \times J\omega + \tau_b \end{cases} \quad donde \;\; \dot{\xi} = v' = R_w V \;\; y \;\; S(\omega) = R_w^T \dot{R_w} \qquad (5)$$

Teniendo en cuenta que F_B y τ_B son las fuerzas externas y pares del quadrotor, las cuales básicamente consisten en el peso del quadrotor, el vector de fuerzas aerodinámicas y los pares generados por los cuatro rotores y adicionalmente considerando la suma de las fuerzas de los rotores, es posible obtener la siguiente ecuación no lineal que caracteriza el sistema.

$$\dot{\zeta} = f(\zeta) + \sum_{i=1}^{4} g_i(\zeta) U_i \qquad (6)$$

Donde ζ es el vector de estados $\zeta = [\xi\ v'\ \eta\ \omega\ ]^T$, U_i entrada de control aplicada al motor i-ésimo y f y g se describen en (Raffo, 2007).

Como se menciona anteriormente el sistema de control que mantiene el quadrotor en vuelo y le brinda la posibilidad de desplazarse en todas las direcciones no es un tema de interés para este trabajo. Nos importa conocer el funcionamiento para comprender el sistema en su totalidad, más allá de diseñar un controlador, ya que se supone que este sistema cuenta con dicho controlador, y es lo suficientemente estable para nuestras aplicaciones.

Existe en el mercado una gran variedad de controladores que brindan funcionalidades básicas como vuelo estacionario, desplazamiento en todas las direcciones, y algunas un poco más complejas como vuelos autónomos bajo ciertas circunstancias previamente programadas,podemos mencionar, AeroQuad, DJI NAZA, Arducopter (Hyon Lim, 2014) cada uno con sus cualidades que los hacen en ciertos aspectos mejores que otros.

Como un objetivo personal de este proyecto se plantea realizar un aporte a la comunidad académica que no cuenta con los recursos suficientes para adquirir una plataforma de desarrollo robusta comoAstecPelican[1] pero sí con recursos suficientes para un proyecto Open Source.

Con esta condición, se plantea encontrar un controlador de vuelo que sea Open Source y es aquí donde surge el criterio de selección para e controlador principal del quadrotor, Ardupilot. Su nombre se debe a que está basado en el micro controlador Atmega 2560 compatible con Arduino. Entre sus características principales cuenta con un giroscopio de 3 ejes, un acelerómetro de 3 ejes, un barómetro, un magnetómetro y un receptor GPS (3DRobotics, 2013). Además de ser Open Source y Open Hardware cuenta con un

[1]AstecPelican (Ascending Technologies, 2014)

ambiente de desarrollo compatible con Arduino, una característica muy interesante dada la gran comunidad que desarrolla para este tipo de plataformas.

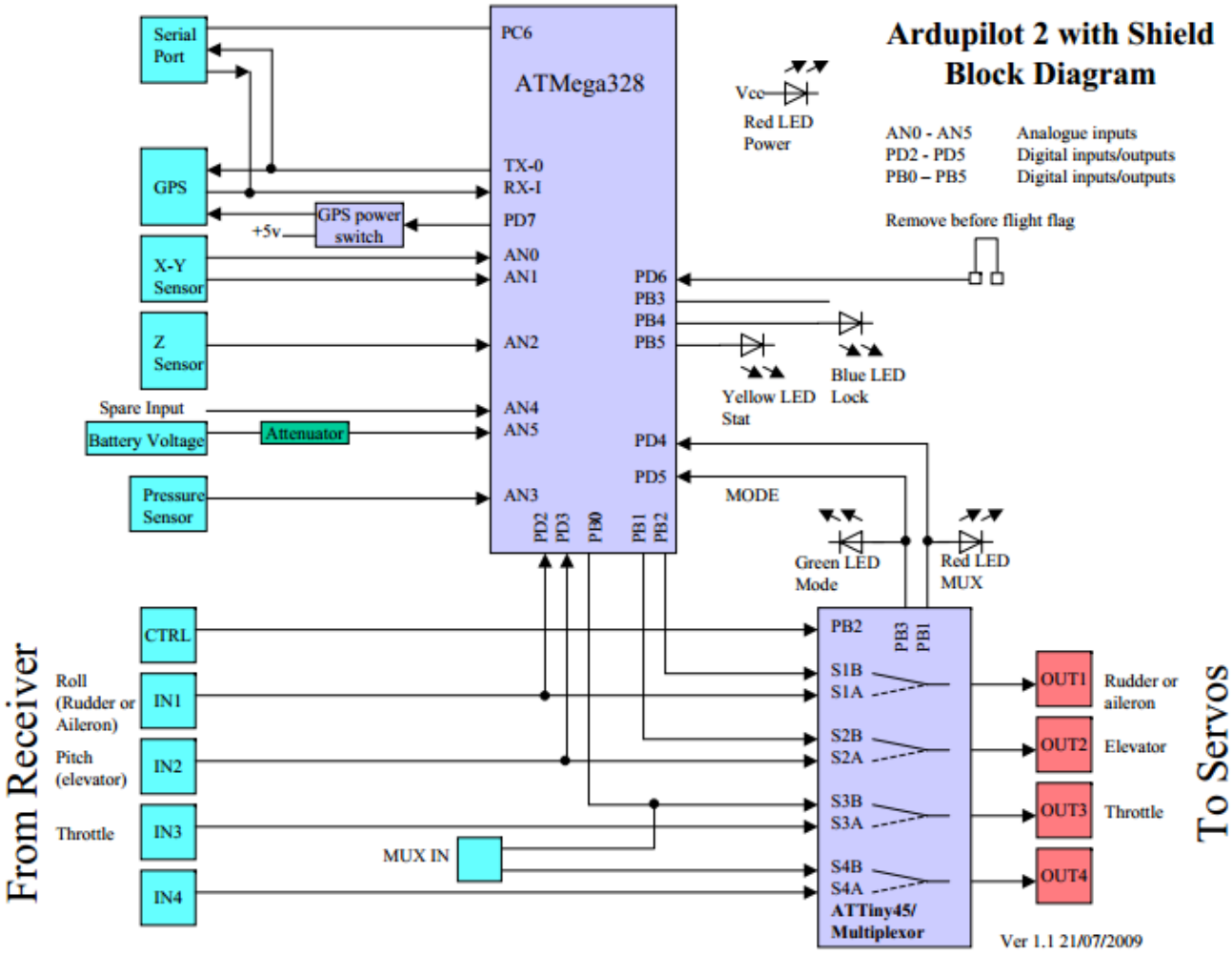

***Fig. 3**. Diagrama de bloques interno de Ardupilot (3DRobotics, 2013)*

5.2. Modelo de la cámara

Al igual que el ojo humano, los sensores ópticos o cámaras generan una proyección bidimensional de toda la información captada en la escena sobre un plano llamado plano de la imagen. Uno de los principales problemas de la proyección es que genera perdida de información como por ejemplo profundidad, percepción real de las medidas, cantidad de luz, color etc. Debido a esto se han desarrollado varios modelos que permiten generar una proyección de la imagen más precisa. El modelo de proyección de perspectiva, modelo Pinole (Zisserman, 2014),bodyframe (uno de los más utilizados dada su simplicidad y buenas aproximaciones a los sistemas reales) define un plano π llamado plano de la imagen, y el centro óptico llamado C. A partir de estas dos medidas se define la distancia focal f co-

mo la distancia entre el centro óptico y el plano de la imagen, todo esto bajo un sistema de coordenadas del entorno W.

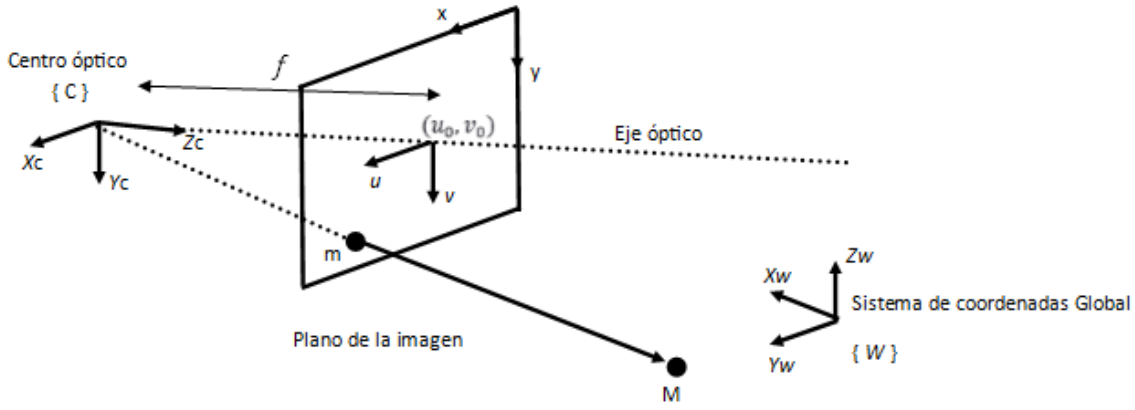

Fig. 4.*Diagrama de referencia del Modelo Pinhole*

En la figura 4 se representa el sistema de coordenadas de la cámara. Donde el centro óptico de la cámara viene dado por C, el cual brinda la alineación de los ejes de la cámara con respecto al plano de la imagen. El punto espacial M (con coordenadas $[x_c, y_c, z_c]$ y con referencia al sistema de coordenadas de la cámara) se representa en el plano de la imagen por un punto m con coordenadas (u, v), las cuales se denominan coordenadas centrales, esto debido a que se referencian al punto central del plano de la imagen (u_0, v_0). Adicional a esto el punto principal de la cámara se encuentra a una distancia f del foco conocida como distancia focal.

Gracias a esto podemos generar una relación entre los puntos sobre el plano y el sistema coordenadas de la cámara dada por la siguiente expresión:

$$\frac{u}{X_c} = \frac{v}{Y_c} + \frac{f}{Z_c} \qquad (7)$$

Esta expresión se denomina transformación de perspectiva ya que relaciona el sistema de coordenadas de la cámara con el sistema de coordenadas del plano de la imagen.

5.3. Control servo-visual

El control servo-visual se define como e*l uso de información visual procedente de una o varias cámaras de video, ya sean fijas o móviles, para controlar la posición y/u orientación*

del efector final del robot con respecto a un objeto o a un conjunto de características visuales de éste (dependiendo de la tarea por realizar (Corke, 1996)*).*

Es importante diferenciar entre control visual y control servo-visual. En el control visual los movimientos del robot son controlados con la información extraída de la escena por la cámara, es decir que un cambio en la escena captado por la cámara produce un cambio en la conducta del robot. El funcionamiento básico es el siguiente:

1. Captura de las imágenes
2. Extracción e interpretación de la escena
3. Determinar la tarea a realzar
4. Ejecutar la acción sobre el robot.

Este es un esquema de control en lazo abierto, donde la información extraída de la cámara no se usa para la realimentación del sistema, es decir es un sistema donde la extracción de la información y el control del robot son dos tareas desacopladas; el robot en todo momento asume que el sistema permanece estático.

El proceso del control Servo-visual consiste en usar la información de la cámara para cerrar el lazo de control y realimentar el sistema, por lo tanto no existe una relación directa ente el controlador principal y en controlado visual, así el control visual puede independizarse de todo el comportamiento no lineal. Particularmente en este proyecto se tiene una posición a la que el Robot (en este caso un Quadrotor) debe dirigirse dada por el sensor GPS, sin embargo la posición deseada (zona de aterrizaje) difiere de la posición real captada por el GPS, por el error presente en el sensor (Aproximadamente de 2.5 m a 5 m en exteriores[2]), por lo tanto el error entre la posición real y la posición deseada debe ser compensado por la información captada por la cámara en todo instante de tiempo, quien en respuesta debe ajustar y minimizar el error de la posición del Quadrotor enviando la información precisa al controlador principal (Ardupilot).

La capacidad de extraer información visual es muy útil en muchos aspectos, principalmente porque brinda un contexto más preciso de la escena en la que está inmerso el Quadrotor. La información extraída de la escena puede clasificarse en parámetros y características de acuerdo a las necesidades particulares. Hablamos de un parámetro de la imagen al referirnos a un valor real que corresponde a una característica de la imagen, este parámetro depende de la naturaleza de la captura y puede ser interpretado como una proyección de las características físicas del objeto que nos interese analizar por ejemplo la longitud de un contorno, las coordenadas el centro del objeto etc. Las características son la información particular del objeto, como tamaño, color, contorno etc.

[2] Información tomada de la página del fabricante. Datasheet (u-blox, 2014)

La ubicación de la cámara también juega un papel muy importante al definir el tipo de control que se desea realizar.Si la cámara se fija sobre el robot recibe el nombre de cámara en mano. En esta configuración existe una relación directa entre el sistema de coordenadas de la cámara y el del robot, por lo tanto un movimiento de la cámara en el espacio implica un movimiento del robot.Si la cámara se fija sobre una superficie que permite captar un plano de la dinámica del robot se conoce como cámara fija, donde el control se realiza a través de las perspectivas captadas por la cámara, por lo tanto la posición de la cámara es independiente a los movimientos del robot haciendo que se necesario estimar la posición relativa entre la cámara y su sistema de coordenadas en cada acción de control realzada.

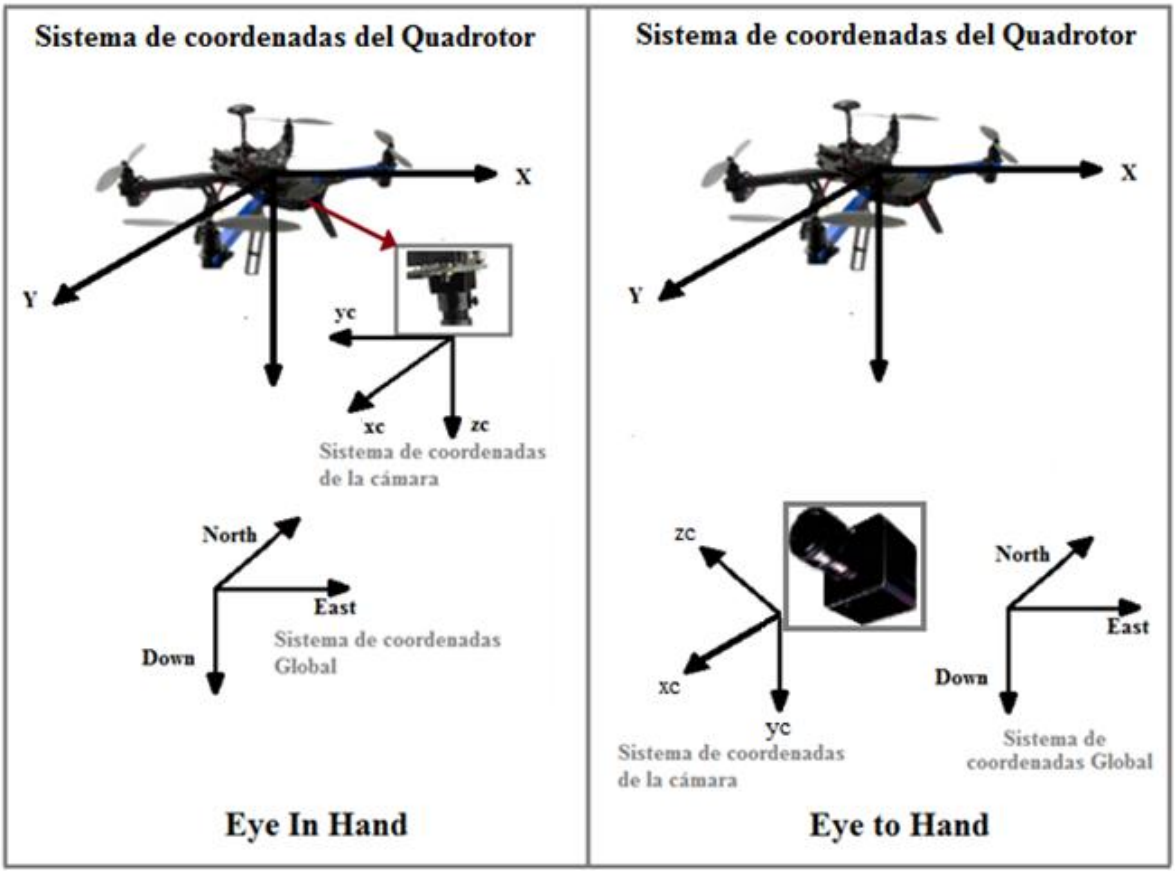

***Fig. 5**. Configuración de cámara en mano (Eye In Hand) y cámara fija (Eye to Hand).*

5.3.1. Esquemas de control servo-visual

El principal criterio para definir un esquema es considerar cómo es la interacción entre el sistema de control visual y el sistema encargado de generar el posicionamiento del robot, en el caso del quadrotor, los rotores. En (Mondragón, 2011) podemos encontrar dos esquemas de control definidos, primero un sistema de control directo (Direct Visual Servoing), donde el sistema de visión envía directamente las señales de control al robot

para estabilizar el sistema, y un segundo esquema donde el control visual es usado como una entrada de referencia al controlador principal, el cual se conoce como control “mirar y mover dinámico” (Dynamic look and move).

Control directo (Direct Visual Servoing)

Esta técnica de control el controlador transforma la información visual en señales de control que se envían a los actuadores del robot. En esta configuración existe un único lazo de control que realimenta el sistema y proporciona las señales de referencia que afectan directamente a los actuadores del robot, en nuestro caso, los rotores. Éste también es el encargado de estabilizar al sistema y debe ser capaz de responder a una frecuencia mayor o igual a la frecuencia de operación de los actuadores. De aquí se despenden en general dos métodos de control, control basado en imagen (Image Based Visual Servoing) y control basado en posición (Position Based Visual Servoing)

Control IBVS (Image Based Visual Servoing)

En este método de control las características visuales de la imagen son utilizadas para calcular la posición y orientación del objeto de control en el espacio, donde se considera las dimensiones reales del robot, el modelo de la cámara y la relación geométrica cámara robot, lo que provoca que sea muy sensible a perturbaciones (Hutchinson, 1996). Su función de error está dada por la diferencia entre la posición y orientación actual del objeto y su posición y orientación deseada.

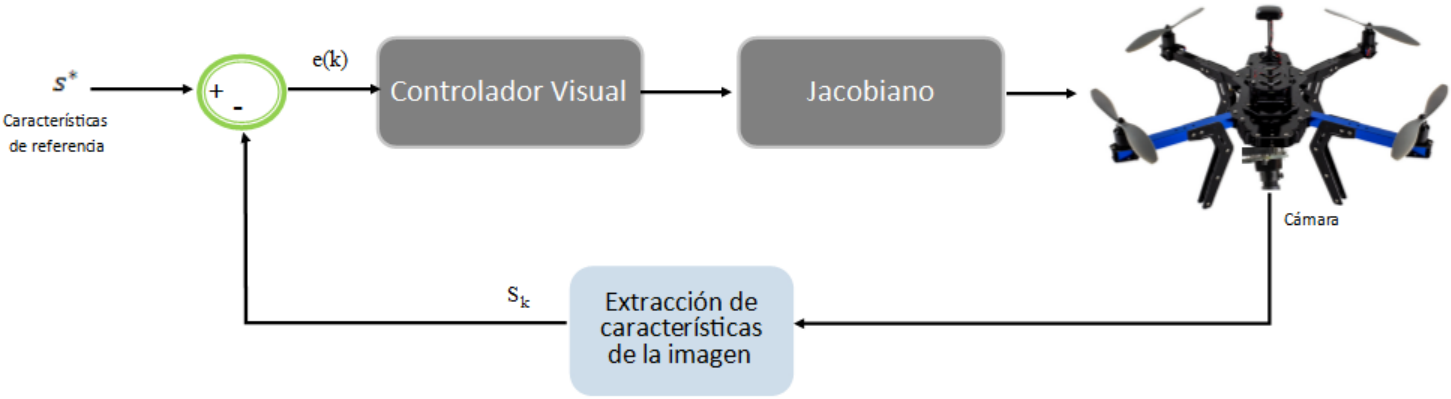

Fig. 6.*Esquema de control servo-visual IBVS*

Control PBVS (Position Based Visual Servoing)

En esta estrategia de control la función de error y la ley de control están definidas directamente en el plano 2D de la imagen y no es necesario un conocimiento previo del modelo tridimensional del objeto. Básicamente la tarea de control trata de reducir una

medida de distancia entre la posición actual y deseada del objetivo en la imagen. Esta técnica es muy robusta ante los posibles errores en la definición del modelo de los objetos, además de reducirse el tiempo de ciclo de control.

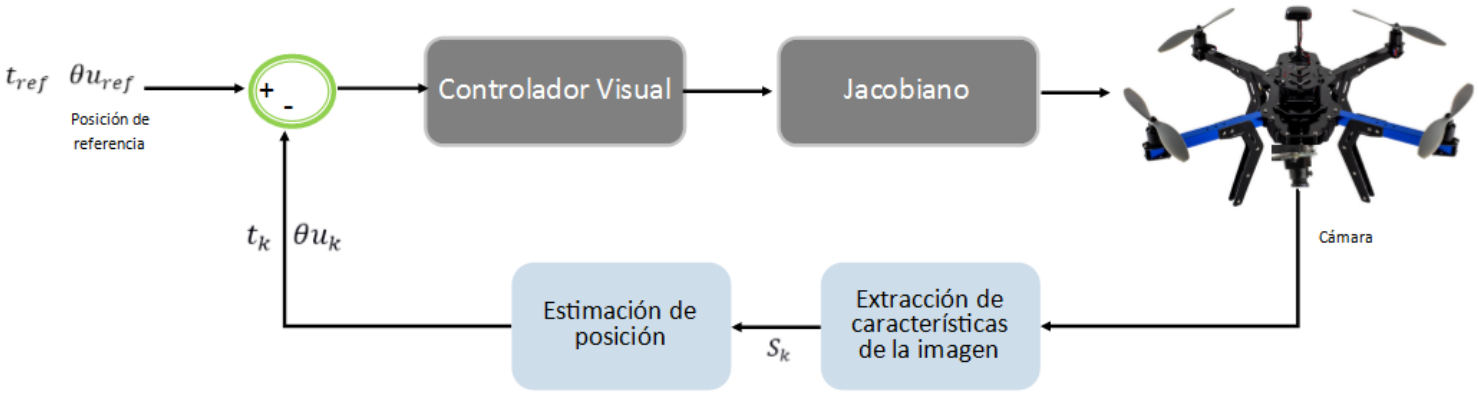

Fig. 7.*Esquema de control servo-visual PBVS.*

5.3.2. Control mirar y mover dinámico (Dynamic look and move).

Este es un tipo de control jerárquico en donde el control visual brinda la información para tomarse como referencia al control principal. En esta configuración el lazo del control visual existe por fuera del lazo de control del sistema principal (control de bajo nivel), el cual es el encargado de estabilizar el sistema y modificar la dinámica de los actuadores. Esta configuración es una de las más usadas en la actualidad porque mientras existe un sistema de control de bajo nivel que estabiliza el sistema, es posible enviar comandos cartesianos y de posición como referencias que modifican la dinámica del sistema indirectamente sin entrar al lazo del control principal. Este es el esquema de control que más nos interesa en este proyecto, debido a que asumimos que el control de bajo nivel se encarga de gestionar todas las acciones de traslación del quadrotor, mientras que el control de alto nivel se encarga de enviar señales de referencia al control principal basadas en control servo visual. Al igual que en el control visual directo, existen dos técnicas de control principales, control servo visual dinámico basado en imágenes, que lo llamaremos DynamicImageBased Look and Move y Control servo visual dinámico basado en posición que lo llamaremos Dynamic Pose Based Look and Move.

Dynamic Image Based Look and move.

Particularmente en esta técnica de control, la función de error y la regla de control son definidas en base a las características captadas directamente desde la escena de la imagen Este es un método que reduce la cantidad de información procesada por el controlador de alto nivel (controlador visual) porque estimar la posición del objeto en cada momento no es necesario como en el caso de control visual por posición.

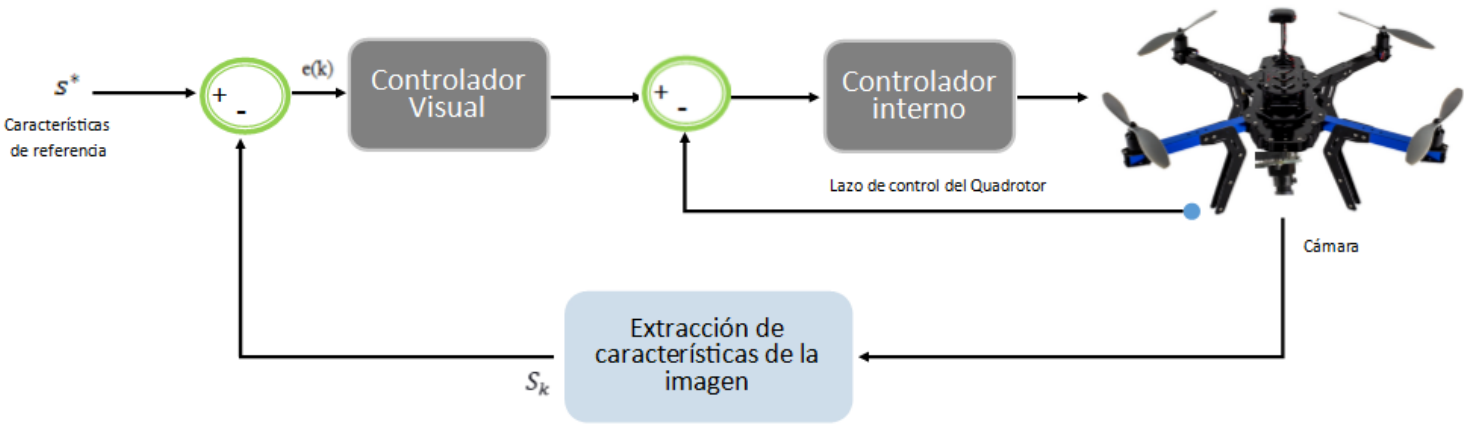

Fig.8. *Dynamic Image Based Look and move.*

En el caso ideal un controlador IBVS utiliza las características de la imagen para definir los parámetros $\boldsymbol{s}$ para los cuales el controlador está diseñado a responder. Estas características pueden ser definidas directamente como una representación en pixeles $s = (u, v)$ o o también de acuerdo a la definición del plano de la imagen $s = (x, y)$.

Esta configuración es independiente de la distancia focal de la cámara, el plano de la imagen y la distorsión del lente, es decir, de los parámetros intrínsecos de la cámara. Además de esto el error puede ser estimado, de acuerdo a (Chaumette, 2006) entre las características medidas en cada instante de tiempo y las características deseadas en la imagen y puede ser calculado a partir de $e = s - s^*$. A pesar que el error está definido en el plano de la imagen, es necesario definir una relación entre los cambios en el espacio de la imagen y la posición del quadrotor, donde en general se usa el Jacobiano o matriz de interacción.

En este aspecto una de las aproximaciones más robustas es la implementación de un control de velocidad, debido a que los cambios en posición son más "suaves" que los cambios de posición con coordenadas cartesianas, ya que la velocidad de traslación es un parámetro que puede ser definido. Para obtener el Jacobianoes necesario establecer la relación entre la velocidad de la cámara y las variaciones de tiempo para $\boldsymbol{s}$, en este orden de ideas si asumimos un punto en el espacio representado en el sistema de coordenadas de la cámara $X^c = [X, Y, Z]^T$ el cual es proyectado en las coordenadas normalizadas del plano de la imagen $s = (x_n, y_n)$ podemos encontrar la relación (de acuerdo al modelo Pinhole) entre los dos espacios cartesianos a partir de la siguiente expresión

$$x_n = \frac{X}{Z} \qquad y_n = \frac{y}{Z} \tag{8}$$

Donde $\frac{X}{Z} = \frac{u - c_u}{f\alpha}$ y $\frac{Y}{Z} = \frac{v - c_v}{f}$, de los cuales c_u, c_v, f, α son los parámetros intrínsecos de la

cámara. La derivada del tiempo de la proyección de los puntos en el plano de la imagen está dada por

$$\dot{x}_n = \frac{\dot{X}}{Z} - X\,\frac{\dot{Z}}{Z^2} \qquad \dot{y}_n = \frac{\dot{Y}}{Z} - Y\,\frac{\dot{Z}}{Z^2} \tag{9}$$

Así para relacionar la velocidad del punto X^c con la velocidad espacial de la cámara es necesario definir este parámetro como $V_c = (v_c, \omega_c)$, y de igual forma la velocidad lineal instantánea $v_c = \left[v_x,\ v_y, v_z\right]$ y la velocidad angular instantánea $\omega_c = \left[\omega_x, \omega_y, \omega_z\right]^T$.

A partir de esto podemos definir la derivada del tiempo del punto X^c

$$\begin{aligned} \dot{X} &= -v_x - \omega_y Z + \omega_z Y \\ \dot{Y} &= -v_y - \omega_z X + \omega_x Z \\ \dot{Z} &= -v_z - \omega_x Y + \omega_y X \end{aligned} \tag{10}$$

Reemplazando las ecuaciones (10) en(9) y considerando la proyección en el plano de la imagen definido en la ecuación(8) se obtiene la siguiente expresión.

$$\dot{x}_n = -\frac{v_x}{Z} + \frac{x_n v_z}{Z} x_n y_n \omega_x + (x_n^2 + 1)\omega_y + y_n \omega_z \tag{11}$$

$$\dot{y}_n = -\frac{v_y}{Z} + \frac{y_n v_z}{Z} + (y_n^2 + 1)\omega_x - x_n y_n \omega_z - x_n \omega_z \tag{12}$$

Las ecuaciones (11) y (12) pueden ser escritas como $\dot{s} = L_s V_c$, donde L_s es el Jacobiano o matriz de interacción:

$$L_s = \begin{bmatrix} -1/Z & 0 & x_n/Z & x_n y_n & -(x_n^2 + 1) & y_n \\ 0 & -1/Z & y_n/Z & (y_n^2 + 1) & -x_n y_n & -x_n \end{bmatrix} \tag{13}$$

De la ecuación (13) es posible observar que existe una dependencia con la coordenada Z, la cual está ligada al sistema de referencias de la cámara X^c, por lo tanto para diseñar un controlador (con la formulación descrita en la ecuación (13)) se requiere que Z se estime en todo instante de tiempo. Esto debido a que no siempre es posible conocer el valor de Z, el cual es una variación de la matriz del Jacobiano, conocido como Jacobiano estimado $\widehat{L_s}$. Usualmente el método de calcular el Jacobiano estimado consiste en tratar de estimar los valores de Z tan reales como sea posible usando un método adaptivo descrito en (Hutchinson, 1996). Otro método popular define el valor de Z como una constante, en este caso $\widehat{L_s}$ es constante y sólo es necesario definir las coordenadas Z deseadas.

Si V_c es usada como la entrada deseada al sistema de bajo nivel (controlador principal) la relación entre la velocidad de la cámara y la variación en tiempo del error puede ser definida como $\dot{e} = L_e V_c$, donde $L_e = L_s$. Usualmente se toma el error como una función exponencial desacoplada decreciente permitiendo transmitir los parámetros de velocidad al controlador de bajo nivel de la siguiente forma

$$V_c = -\lambda L_e^{+} e \qquad (14)$$

O cuándo sólo se tiene el Jacobiano estimado $V_c = -\lambda \widehat{L_e^{+}} e$, donde L_e^{+} es la pseudo-inversa de la matriz Moore-Penrose L_e, ($L_e^{+} = (L_e^T L_e)^{-1} L_e^T$) y $\widehat{L_e^{+}}$ es la pseudo-inversa del Jacobiano estimado cuando Z no es un parámetro disponible. El controlador que se obtiene asume que los cambios en la dinámica del quadrotor son despreciables cuando el control de bajo nivel procesa una orden enviada por el control de visión.

Dynamic Pose Based look and move

En esta técnica de control las características extraídas de la imagen son usadas para estimar la posición de la cámara respecto a un sistema de referencias definido. En concreto el controlador visual se limita a calcular el error de posición entre la posición de referencia y la posición estimada, por lo tanto se generan referencias de traslación o posición que son interpretadas por el controlador principal.

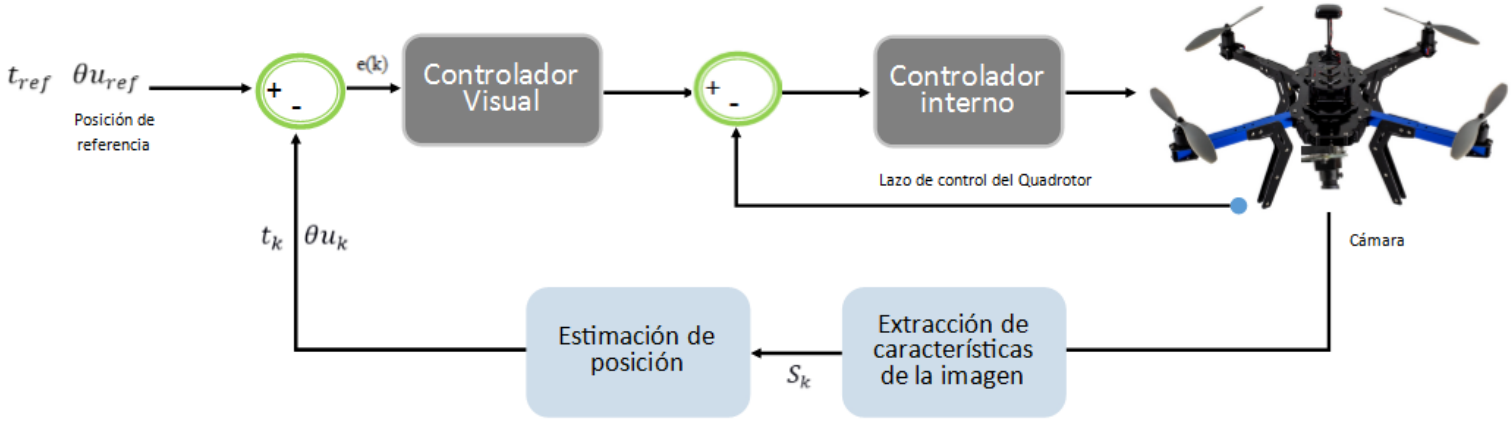

Fig.9. *Dynamic Pose Based look and move*

Usualmente se definen las características en función del sistema de coordenadas parar representar la función de error y la posición de la cámara, siendo esta última definida como posición de referencia $s^* = (t^*, \theta u^*)$ y la posición medida como $s = (t, \theta u)$ donde t y θu se definen respectivamente como el vector de traslación y la orientación (el ángulo de rotación θ a través del vector u). En este esquema el controlador depende de cómo las características s y s^*son seleccionadas, (Chaumette, 2006) propone dos estrategias ´para

este fin: Seleccionar las características considerando si el frame de referencia está contenido en el frame de procesamiento actual $\mathfrak{F}_c$, el frame deseado $\mathfrak{F}_{c^*}$ o el frame de referencia $\mathfrak{F}_0$. La traslación t se define de forma relativa al frame de referencia $\mathfrak{F}_0$, y el conjunto de características $s = ({}^{c}t_0, \theta u)$ y sus respectivas referencias $s^* = ({}^{c^*}t_0, 0_{3\times1})$, done $0_{3\times1}$ es un vector de ceros.Adicional a esto es posible expresar la ecuación que representa la función de errorcomo $e = ({}^{c}t_0 - {}^{c^*}t_0, \theta u)$ el cual está relacionado con la velocidad de la cámara a través de la matriz $L_e$con $\dot{e} = L_e V_c$.

$$L_e = \begin{bmatrix} -I_{3\times3} & [{}^{c}t_0]_x \\ 0_{3\times3} & L_{\theta u} \end{bmatrix} \qquad (15)$$

Donde $I_{3\times3}$ es una matriz identidad 3×3 y L_{θ_u} es lo correspondiente a la orientación del objeto en el Jacobiano, y viene dada por la siguiente expresión

$$L_{\theta_u} = I_{3\times3} - \frac{\theta}{2}[u]_x + \left(1 - \frac{sinc(\theta)}{sinc^2(\theta/2)}\right)[u]_x^2 \qquad (16)$$

$$\text{Donde } [u]_x^2 = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix} \qquad (17)$$

$$\text{Recordad que } sinc(x) \begin{cases} 1 & para\ x = 0 \\ \frac{sen\ x}{x} & para\ otro \end{cases} \qquad (18)$$

De esta forma podemos encontrar la entrada de control $V_c = -\lambda\, L_e^{-1}\dot{e}$ obteniendo la matriz inversa del Jacobiano estimado con la siguiente ecuación:

$$L_e = \begin{bmatrix} -I_{3\times3} & [{}^{c}t_0]_x L_{\theta u}^{-1} \\ 0_{3\times3} & L_{\theta u} \end{bmatrix} \qquad (19)$$

Ahora al considerar que $L_{\theta u}^{-1}\,\theta u = \theta u$ y usando la matriz inversa del Jacobiano estimado podemos encontrar el controlador, compuesto por la velocidad lineal $v_c = \left[v_x, v_y, v_z\right]^T$ y angular $\omega = \left[\omega_x, \omega_y, \omega_z\right]^T$

$$v_c = -\lambda\left(\left({}^{c^*}t_0 - {}^{c}t_0\right) + [\,{}^{c}t_0]_x\,\theta u\,\right) \quad (20)$$

$$\omega_c = -\lambda\,\theta u \quad (21)$$

Otro esquema de control, basado en posición, define las características medidas o tomadas y las deseadas respectivamente como $s = ({}^{c}t_c, \theta u)$ y $s^* = (0_{3\times1}, 0_{3\times1})$, lo que resulta que la función de error sea $s - s^* = e = s$. Esto permite que el Jacobiano relacione la velocidad de la cámara V_c con el error en la velocidad $\dot{e} = L_e\,V_c$ y genere un desacople entre los movimientos traslacionales y rotacionales

$$L_e = \begin{bmatrix} {}^{c^*}R_c & 0_{3\times3} \\ 0_{3\times3} & L_{\theta u} \end{bmatrix} \quad (22)$$

Donde ${}^{c^*}R_c$ es la matriz de rotación que sirve para identificar la orientación del frame actual con el frame deseado. Finalmente con esto podemos generar la ley de control que rige al controlador con la siguiente expresión:

$$v_c = -\lambda^* R_c^T t_c \quad (23)$$

$$\omega_c = -\lambda\theta u \quad (24)$$

5.3.2. Diseño del control servo visual

En esta sección se da a conocer cuál es la estrategia de control implementada para que el quadrotor tenga la capacidad de ajustar la posición horizontal y vertical cuando detecta la zona de aterrizaje en tierra. El proceso consiste en el envío,desde la plataforma de aterrizaje,de las coordenadas GPS, las cuales se procesan en el quadrotor por el controlador de bajo nivel. Éste se encarga de dirigir el Quadrotor hasta la posición deseada y una vez se detecta la zona de aterrizaje, el controlador de alto nivel ajusta la posición vertical y horizontal del quadrotor para lograr el aterrizaje sobre la plataforma.

Básicamente se requiere de dos etapas de control visual conjuntas, una que se encargue de controlar la posición horizontal del quadrotor, ajustando los ángulos del Roll y Pitch para mantener imagen de la plataforma siempre en el centro del plano de la imagen captado por la cámara a bordo. En esta etapa no se tiene en cuenta el ángulo yaw ya que se restringe a que el quadrotor y la plataforma siempre están alineados hacia el frente. La etapa siguiente consiste en descender el Quadrotor, para esto se requiere ubicar la plataforma en el plano de la imagen, y con la ayuda del control de los ángulos Roll y Pitch mantener la plataforma en el centro del plano de la imagen mientras se realiza el descenso del Quadrotor.

Control visual horizontal (ajuste de los ángulos Roll y Pitch)

Este control consiste en mantener la plataforma de aterrizaje siempre centrada en el plano de la imagen. La plataforma tiene una forma cuadrara con una proyección de su centro dada por $\boldsymbol{x_t} = [x_t, y_t]^T$ el cual puede ser proyectado en el plano de la cámara con la ecuación $\boldsymbol{X}_{Tc} = [X_{Tc}, Y_{Tc}, Z_{Tc}]^T$. De acuerdo a lo definido en la sección 5.3.1. podemos definir los píxeles de las propiedades proyectadas como $s = [x_t\ , y_t]^T$ como características para la ley de control. Ahora bien, considerando que la finalidad del sistema de control es mantener la el objeto de interés siempre en el centro (en nuestro caso la plataforma de aterrizaje) podemos describir las características deseadas como:

$$s^* = [x^*\ , y^*]^T = \left[\frac{AnchoImg}{2} - c_u\ , \frac{AltoImg}{2} - c_v\right]^T \qquad (25)$$

Donde $C_u\ y\ C_v$ corresponden al centro de masa de la plataforma de aterrizaje, AnchoImag. y AltoImag. A las medidas del plano de la imagen captado por la cámara y $e = s - s^*$ el error que se integra al controlador.

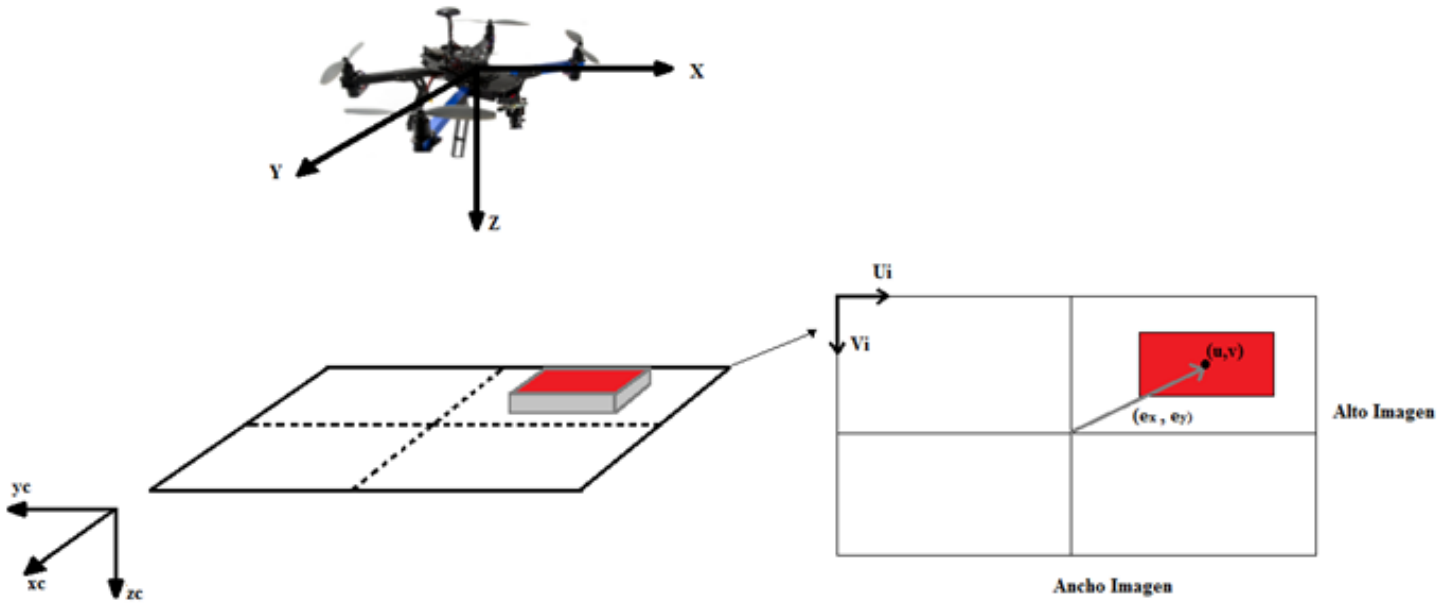

Fig. 10. *Sistema de referencia Quadrotor - Plataforma de aterrizaje.*

Con el fin de diseñar un controlador de velocidad traslacional, y considerando las ecuaciones dadas por el modelo de la cámara Pinhole, las características se proyectan en el plano de la imagen como:

$$x_t = \frac{X_{Tc}}{Z_{Tc}} \qquad (26)$$

$$y_t = \frac{Y_{Tc}}{Z_{Tc}} \qquad (27)$$

Donde sus derivadas de tiempo están dadas por

$$\dot{x}_t = \frac{\dot{X}_{Tc}}{Z_{Tc}} - X_{Tc}\frac{\dot{Z}_{Tc}}{Z_{Tc}^2} = \frac{\dot{X}_{Tc}}{Z_{Tc}} - x_t\frac{\dot{Z}_{Tc}}{Z_{Tc}} \qquad (28)$$

$$\dot{y}_t = \frac{\dot{Y}_{Tc}}{Z_{Tc}} - Y_{Tc}\frac{\dot{Z}_{Tc}}{Z_{Tc}^2} = \frac{\dot{Y}_{Tc}}{Z_{Tc}} - y_t\frac{\dot{Z}_{Tc}}{Z_{Tc}} \qquad (29)$$

La derivada de $\dot{X}_{Tc}$ se obtiene de igual forma que en la ecuación(10), por lo tanto integrando este factor en la ecuación(28) y (29) y considerando la proyección de las características definidas en(26) y (27) es posible obtener el Jacobiano o matriz de integración dada por la siguiente expresión

$$L_s = \begin{bmatrix} -1/Z & 0 & x_n/Z & x_n y_n & -(x_n^2+1) & y_n \\ 0 & -1/Z & y_n/Z & (y_n^2+1) & -x_n y_n & -x_n \end{bmatrix} \qquad (30)$$

Podemos realizar una simplificación de la matriz anterior realizando las siguientes aproximaciones:

- ✓ Si suponemos un control de bajo nivel lo suficientemente robusto y estable podemos asumir que en vuelo los ángulos de roll y pitch son constantes y muy cercanos a 0, lo que se traduce en velocidades angulares de la cámara despreciables en X y Y. $\omega x_c \approx \omega y_C \approx 0$
- ✓ La velocidad inicial de la cámara en Z es cero; $vz_c = 0 \leftrightarrow t = 0$
- ✓ El ángulo yaw del quadrotor siempre será constante e igual al inicial del momento de búsqueda de la marca.

Teniendo en cuenta lo anterior podemos reducir la matriz a la siguiente expresión.

$$L_s = \begin{bmatrix} -1/Z & 0 \\ 0 & -1/Z \end{bmatrix} \qquad (31)$$

Con su inversa correspondiente

$$\widehat{L_e} = \begin{bmatrix} -Z & 0 \\ 0 & -Z \end{bmatrix} \qquad (32)$$

De esta forma podemos construir el control encargado de enviar los comandos de velocidad $V_c - \lambda \widehat{L_e}^{+} e$, el cual se describe en la siguiente ecuación.

$$\begin{bmatrix} v_{Xc} \\ v_{Yc} \end{bmatrix} = -\lambda \begin{bmatrix} 0 & -Z \\ -Z & 0 \end{bmatrix} \begin{bmatrix} x_t - x^* \\ y_t - y^* \end{bmatrix} \qquad (33)$$

Se observa que el sistema de coordenadas de la cámara, esta rotado 90° en el eje Z con respecto al eje de coordenadas de la aeronave. Por esta razón el eje X de la aeronave corresponde al eje –Y de la cámara, el eje Y de la aeronave corresponde al eje X de la cámara y el Z es igual.

Para el cálculo o estimación de Z se usa el valor del estimador de estado entregado por el ardupilot (GPS corregido con IMU y filtro de kalman) el cual corresponde a la altura del quadrotor

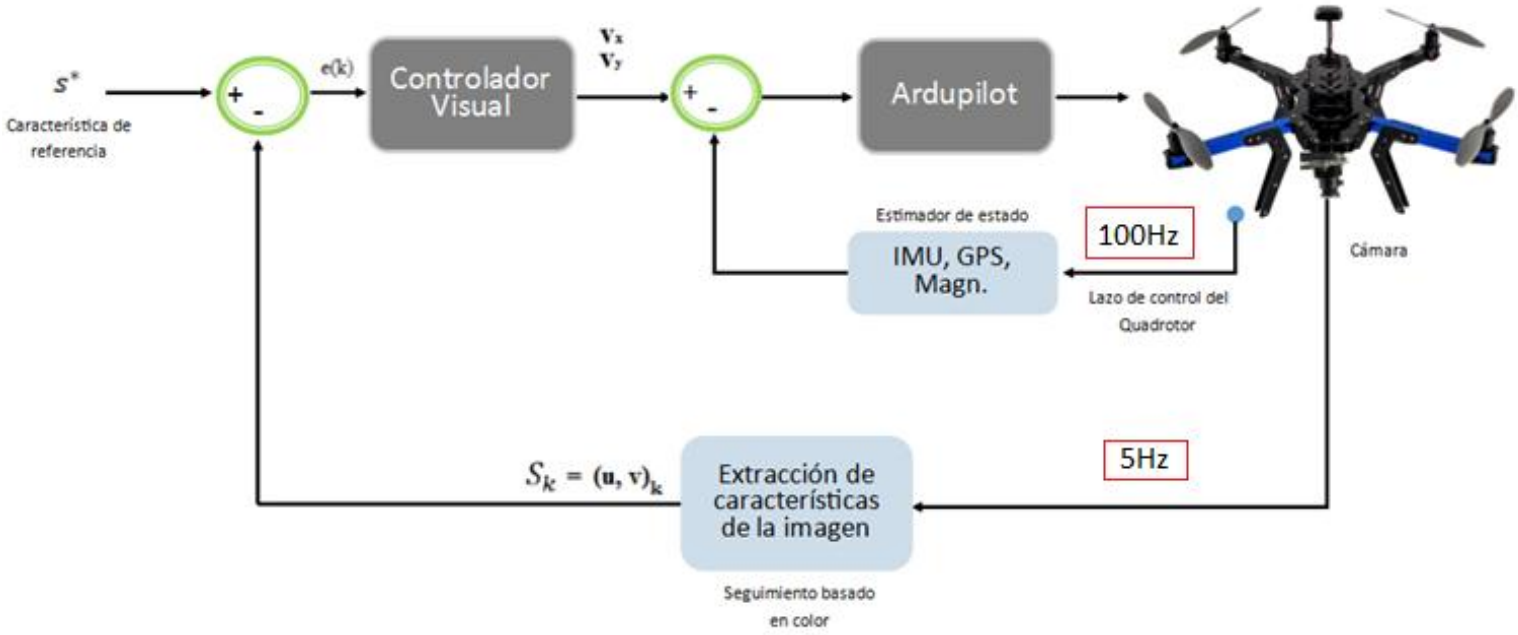

Fig. 11. *Diagrama de bloques del sistema "Dynamic Image Based look and move" para el control de aterrizaje de un quadrotor sobre una plataforma basado en seguimiento por características de color*

Control visual horizontal (Aterrizaje)

En principio el control de descenso no requiere una estrategia de control definida en el diseño del controlador visual, dado que una vez el quadrotor ubica la plataforma de aterrizaje por medio de la cámara se activa un descenso controlado a 30 cm/s, esto es posible gracias a que Ardupilot permite, mediante el sensor de presión o barómetro y el GPS, realizar un descenso controlado a una velocidad establecida, en nuestro caso 30cm/s. Esto se logra gracias a la API de Ardupilot que permite activar el modo de descenso controlado. Una vez se activa el descenso Ardupilot toma como referencia la posición GPS e intenta aterrizar manteniendo esta posición constante. Sin embargo este aterrizaje requiere un ajuste de la posición horizontal del Quadrotor para compensar el error del GPS y lograr aterrizar el quadrotor sobre la zona deseada.

5.4. Sistema de procesamiento a bordo

El campo de desarrollo de visón por computador ha sido ampliamente estudiado por muchas áreas del conocimiento, se han desarrollado muchos algoritmos que pretenden solucionar algunos problemas en campo de visión para robots, algoritmos para extracción de características, formas y gestos, seguimiento de objetos, estimación de posición, reconocimiento de características etc. El gran problema radica en que el análisis de imágenes demanda una gran capacidad de procesamiento, esto implica que para un vehículo aéreo con capacidad de carga reducida el procesamiento a bordo deba hacerse con módulos de computo de tamaño reducido y prestaciones limitadas, sin embargo en los últimos años los avances tecnológicos han permitido tener módulos de computo de un tamaño muy reducido, con muy buen desempeño y capacidad de procesamiento, lo que los convierte en buenos candidatos para implmentar algoritmos de visión en un vehículo aéreo como un quadrotor. Unas de las particularidades que se desea buscar en este tipo de plataformas es que sean Open Source, esto debido en gran medida a reducir los costos de implementación y también para tener una plataforma que pueda ser reprogramada de acuerdo a las necesidades. Raspberry Pi, Beaglebone Black, Odroid son tres placas que hoy en día se viene usando en muchos proyectos de sistemas enbebidos. De estos módulos podemos destacar su tamaño, peso, costos reducidos, y una capacidad de cómputo realmente buena (Procesadores de 1Ghz, Dual Core, 1GB RAM y hasta 32GB de almacenamiento flash). Unos de los módulos que destaca en el mercado es Beaglebone Black, una plataforma que no es tan robusta como Odroid, pero si mejor que Raspberry en cuanto a capacidad de procesamiento. Esta plataforma posee una gran cantidad de entradas análogas y digitales y un conjunto de varias UART las cuales son muy útiles a la hora de interconectar diferentes sistemas.

A partir de la información de la tabla 2 podemos afirmar que Odroid es la placa con mejores prestaciones pero con mayor costo que RASPBERRY PI y Beaglebone Black (BBB). BBB se sitúa en un intermedio entre Raspberry y Odroid, tanto en costos como en

capacidad de procesamiento, por esta razón se escogió este módulo como control principal de alto nivel.

Para poder controlar la aeronave de manera autónoma con BBB es necesario realizar una fusión sensorial con la IMU del Quadrotor, de esta manera se logra tener toda la información de los ángulos Pitch, Roll, Yaw, Altura, posición GPS, nivel de batería etc., y lograr una integración con la información captada por la cámara. Esta fusión sensorial se realiza mediante la comunicación mediante la UART del control de bajo nivel (Ardupilot) y la UART de BBB, ya que gracias a la API proporcionada por Ardupilot podemos empaquetar toda la información sensorial a través de la UART y comunicarla a BBB

	BeagleBone Black	**Raspberry Pi**	**Odroid XU3**
Precio	45 USD	35 USD	149 USD
SoC	1GHz TI Sitara AM3359 ARM Cortex A8	700 MHz ARM1176JZFS	Samsung Exynos5422 ARM® Cortex™-A15 Quad 2.0GHz/Cortex™-A7 Quad 1.4GHz
RAM	512 MB DDR3L @ 400 MHz	512 MB SDRAM @ 400 MHz	2Gbyte LPDDR3 RAM PoP (933Mhz, 14.9GB/s memory bandwidth, 2x32bit bus)
Storage	2 GB on-board eMMC, MicroSD	SD	eMMC module socket : eMMC 5.0 Flash Storage (up to 64GByte) , MicroSD Card Slot (up to 64GByte)
Video	1 Mini-HDMI	1 HDMI, 1 video compuesto	1 HDMI
O.S.	Angstrom, Ubuntu, Android, ArchLinux, Gentoo, Minix, RISC OS	Raspbian, Android, ArchLinux, FreeBSD, Fedora, RISC OS	Ubuntu 14.04 + OpenGL ES + OpenCL on Kernel LTS 3.10, Android 4.4.2 con Kernel LTS 3.10
Consumo mA / V	210-460 mA @ 5V	150-350 mA @ 5V	1000mA @ 5V
GPIO	65 Pines I/O	8 Pines I/O	28 Pines I/O
Puertos	1 USB Host, 1 Mini-USB Cliente, 1 10/100 Mbps Ethernet	2 USB Hosts, 1 Micro-USB Alimentación, 1 10/100 Mbps Ethernet	1 USB 3.0 OTG, 4 USB 3.0 Host , 1 10/100 Mbps Ethernet

Tabla 2. Comparativa de características de Beaglebone Black, Raspberry Pi y Odroid XU3

6. ESPECIFICACIONES

En esta sección se describe cuáles son las características del sistema, a nivel de hardware y software, particularmente el quadrotor utilizado, el sistema de control de bajo nivel, el sistema de control de alto nivel junto con los componentes visión y la plataforma de aterrizaje.

6.1. Hardware del quadrotor

El quadrotor usado en este proyecto se conoce como Arducopter, fue adquirido en la tienda de la empresa 3DRobotics, quienes son los creadores de Ardupilot. Este quadrotor cuenta con las siguientes características principales.

Fig.12.*QuadrotorArducopter X4 3DRobotics*[3]*.*

- ✓ 4 motores brushless outrunner de 880Kv AC2836 y 4 ESC (electronic speed control) de 20A con firmware Simonk, 4 Proppellers 11X47 push pull y Chasis en aluminio
- ✓ Sensor GPS u-bloxLEA-6H GPS con compas a una frecuencia de 5Hz
- ✓ Radio para telemetría a 915Mhz
- ✓ Batería de polímero de litio de 5000mAh
- ✓ Peso total 2.5Kg, capacidad de carga 400g

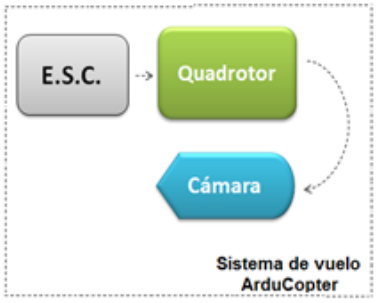

Fig. 13*. Diagrama de entradas y salidas (general) del sistema de vuelo*

[3] 3DRoboticsQuadrotor (3DRobotics, 2013)

6.2. Control de bajo nivel

Como se describe anteriormente se usó el controlador de 3Drobotics llamado Ardupilot o APM, el cual es un sistema de control para un quadrotor basado en arduino. Esto permite una gran posibilidad de adaptación del control a las características particulares de cada proyecto. Entre sus principales características se encuentran:

***Fig.14**. ControladorArdupilot APM 2.6 3DRobotics*[4]

- ✓ Compatible con arduino, MicrocontroladorAtmega 2560 de Atmel
- ✓ Giroscopio y acelerómetro MPU-6000 de 3 ejes
- ✓ Barómetro MS5611-01BA03de alta resolución
- ✓ Compas digital HMC5883L-TR de 3 ejes
- ✓ 4MB de almacenamiento On-Board para logs de vuelo, 1 puerto USB
- ✓ 14 Entradas/Salidas digitales, 8 Salidas PWM para motores brushless
- ✓ 8 Canales de entrada para receptor y 4 modos de vuelo[5] usados
- ✓ Protocolo de comunicación Mavlink

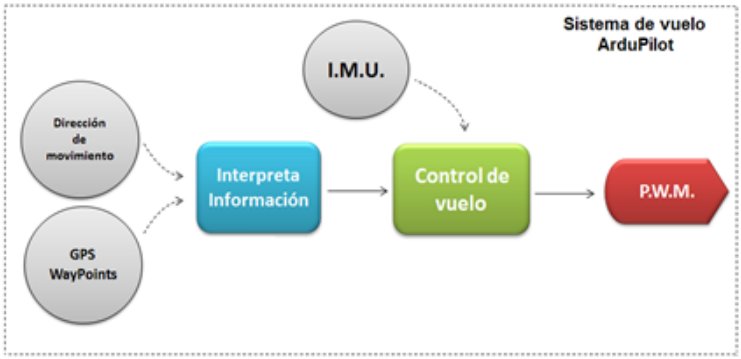

***Fig. 15.** Diagrama de entradas y salidas (general) del sistema de vuelo*

[4] 3DRobotics Autopilot (3DRobotics, 2013)
[5] Modo TAKEOFF: Despegue autónomo, Modo GUIDED: Desplazamiento mediante coordenadas GPS, Modo POSHOLD: Posicionamiento estático en un punto GPS, Modo LAND: Aterrizaje controlado a 30cm/s

6.3. Control de alto nivel

Control Principal.

Basado en la placa Beagleboneblack el controlador de alto nivel es capaz de crear un ambiente te desarrollo mediante la implementación de un sistema operativo Linux basado en el kernel 3.8. Entre sus principales características se destacan.

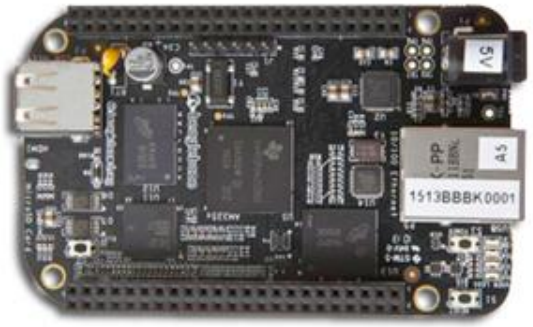

Fig.16. *BeagleBoneBlacj de Texas Instruments*[6]

- ✓ Procesador ARM Cortex-A8 AM335X a 1GHz con 512MB de memoria RAM
- ✓ 4GB de almacenamiento On-board y puerto para tarjeta uSD
- ✓ Linux Kernel 3.8, Puerto USB 2.0 modo cliente, Conector de Red RJ-45 10/100
- ✓ Salida HDMI, 4 UART, 69 GPIO, puerto SPI, I2C
- ✓ Librerías de visión OpenCV
- ✓ Protocolo Mavlink para comunicación con Ardupilot
- ✓ Consumo 210-460 mA@5V y Peso de 40g

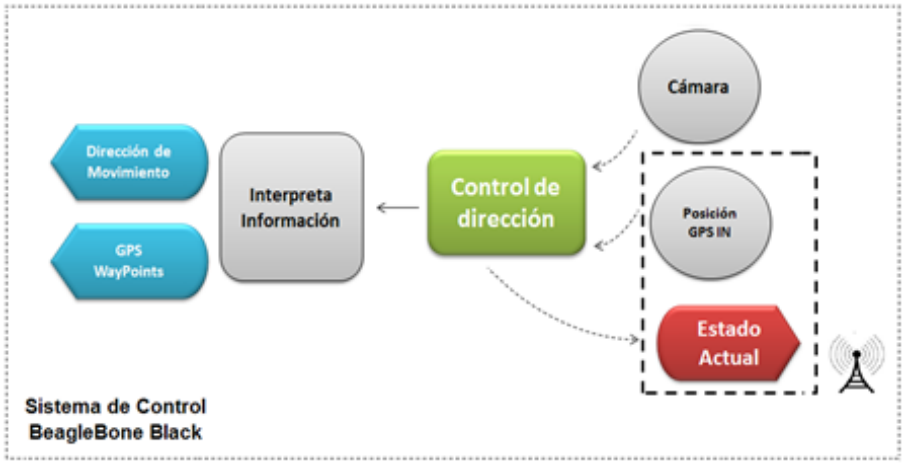

Fig. 17. *Diagrama entradas y salidas (general) del sistema de control basado en BBB*

[6]BeagleBone Black (Instruments, 2014)

Cámara a bordo

***Fig.18**. Logitech HD Pro USB Webcam C920*

- ✓ Interfaz USB 2.0 con compresión en formato MJPEH y H.264
- ✓ Auto enfoque, Lentes Carl-Zeiss

6.4. Plataforma de aterrizaje

En esta sección se da a conocer cuáles son las características de la plataforma, y cómo se define lo que el sistema de visión debe detectar como plataforma de aterrizaje.

En principio para que el sistema de visión tenga facilidad de detectar la plataforma se diseñó una con dimensiones rectangulares, aproximadamente de 100cm de largo por 50cm de ancho. La plataforma se recubre con una lámina roja y se ubica en un entorno donde este color tenga un buen contraste con respecto al medio. La capacidad de enviar las coordenadas por GPS se logra mediante el uso de un ambiente de desarrollo Linux con las librerías de comunicación para APM llamadas Mavlink. Debido a que este tipo de librearías es funcional bajo un ambiente Debian, se usó un sistema embebido con el que se contaba en el momento de realización del proyecto, el cual es Raspberry Pi, que junto con un módulo de recepción GPS USB y un enlace wifi es capaz de enviar las coordenadas GPS al quadrotor.

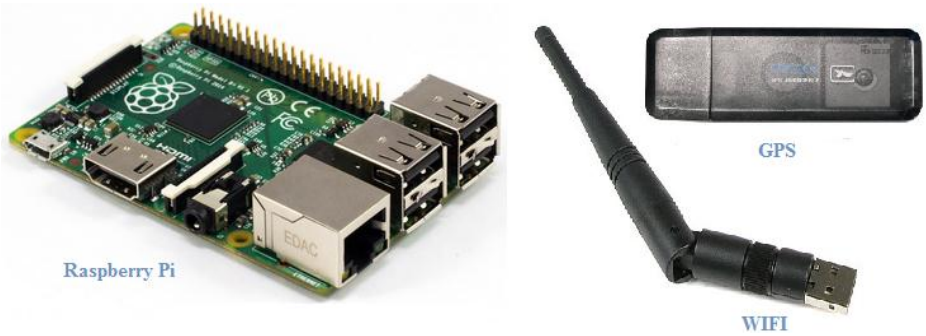

***Fig. 19.** Componentes de la plataforma de aterrizaje, Plataforma Linux Raspberry Pi[7], Modulo WIFI UWN200[8] y Módulo GPS ND-100S[9]*

[7] Raspberry Pi (Raspberry Pi, 2014)
[8]Módulo WIFI (Logic Supply, 2014)
[9]Módulo GPS (Globalsat Products, 2014)

6.5. Diagrama general del sistema

El sistema general está conformado por 4 subsistemas, los cuales interactúan entre sí para realizar la acción deseada. El sistema debe seguir 3 etapas principales para lograr el objetivo planteado.

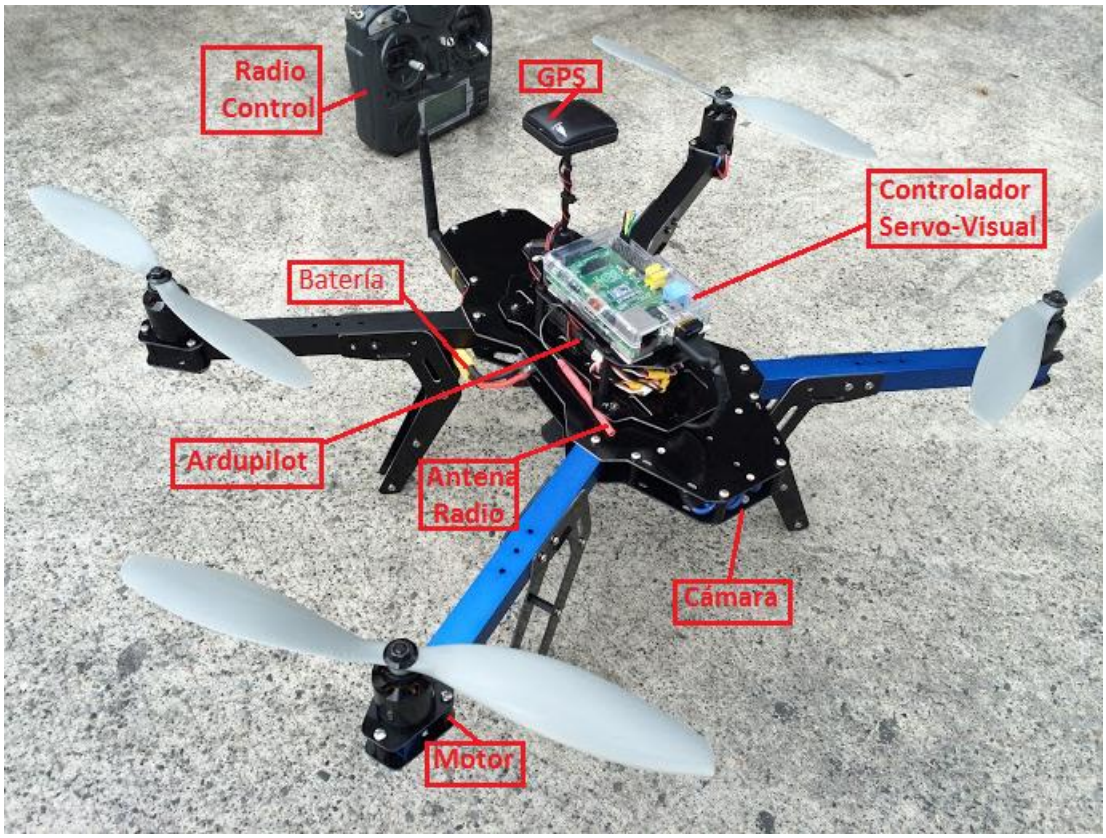

Fig. 20. Montaje real del Quadrotor

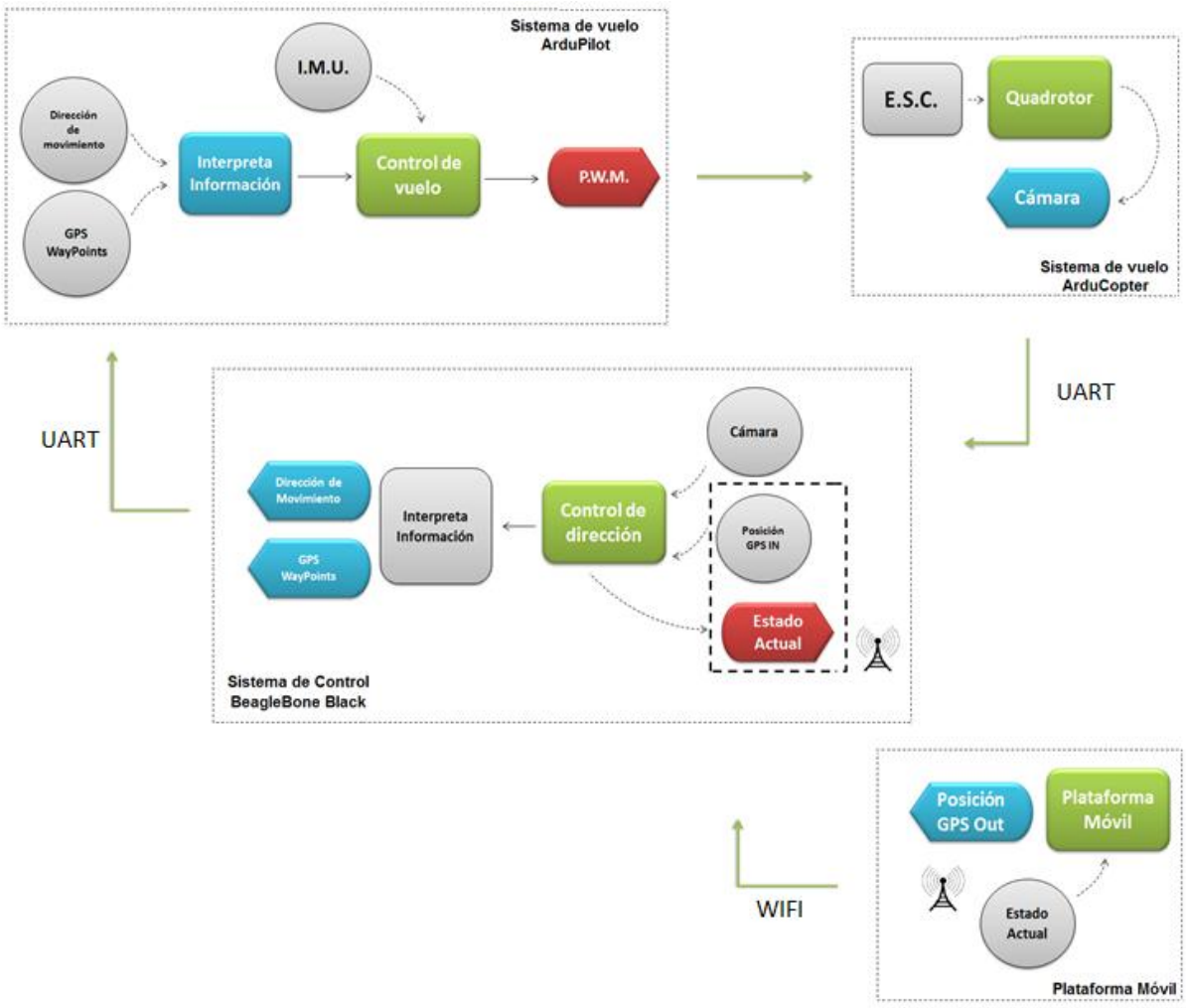

Fig. 21. *Diagrama general del sistema propuesto*

En la figura 21 se observa un diagrama general del sistema implementado. El controlador servo-visual recibe las coordenadas GPS de la plataforma móvil mediante un enlace wifi, y la información procesada por la cámara para enviar los comandos de control a Ardupilot quien se encarga de ejecutar la acción sobre el quadrotor. El proceso que describe el sistema se divide en 3 etapas fundamentales:

1. **Etapa 1 - Despegue:**

   Mediante las funcionalidades de Ardupilot se realiza el despegue, el cual por defecto está parametrizado para elevar al quadrotor a una altura entre 3m y 30m

2. **Etapa 2 - Desplazamiento**:

   Mientras se realiza el despegue, Ardupilot puede dirigir al quadrotor hasta la posición GPS deseada, para esto la plataforma siempre envía su posición al

quadrotor mediante la conexión wifi. BeagleBone Black interpreta la información GPS y mantiene el modo de vuelo de Ardupilot en **GUIDED** el cual permite al quadrotor seguir una posición GPS deseada. Una vez el quadrotor se desplace hasta la plataforma se concluye la etapa de desplazamiento y el controlado visual toma el control del quadrotor.

3. **Etapa 3 - Seguimiento óptico:**

   El controlador visual está activo en todas las etapas descritas por el quadrotor. Cuando se detecta la plataforma en tierra se hace un cambio en el modo de vuelo, de **GUIDED** a **POSHOLD**, el cual intenta mantener al quadrotor en un punto fijo en el aire. Este punto fijo está dado por el punto GPS justo cuando se activa el modo POSHOLD.

4. **Etapa 4 – Aterrizaje**

   Una vez el quadrotor alcanza la plataforma de aterrizaje se procede a realizar el aterrizaje. En este proceso el modo **LAND** se activa, el cual genera un desplazamiento vertical hacia abajo, manteniendo la posición GPS desde donde fue activado, a una velocidad por defecto de 30cm/s.

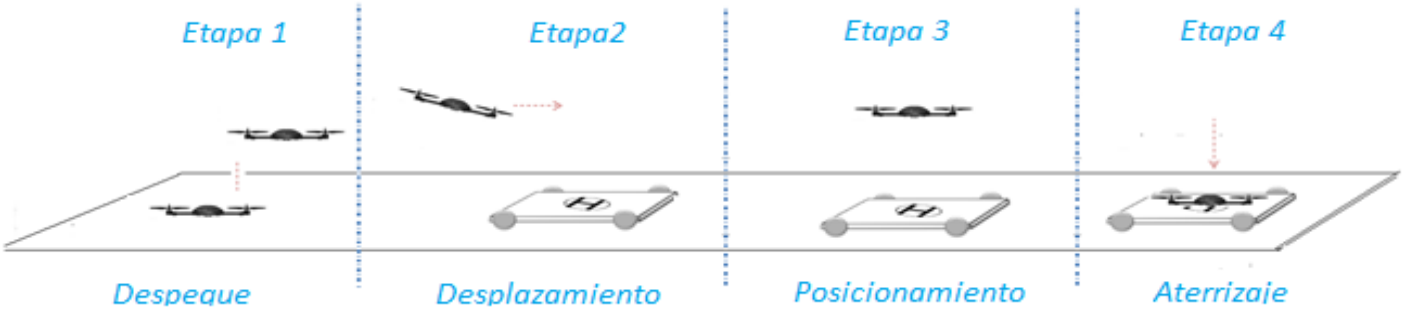

Fig. 22. *Descripción de las etapas que debe realizar el quadrotor*

7. DESARROLLOS

En el contexto de este proyecto es importante implementar dos mecanismos de vuelo. El primero consiste en evaluar el comportamiento del quadrotor cuando intenta aterrizar sobre una plataforma fija en tierra mientras que la segunda consiste en aterrizar el quadrotor sobre la plataforma en movimiento. Para ambos casos, el esquema de control es el mismo, ya que lo que se plantea en la sección 5.3 es un sistema de control visual capaz de enviar comandos de velocidad al quadrotor para lograr centrar la plataforma en el centro de la imagen y aterrizar, esto hace posible a que el control sea capaz de centrar siempre la plataforma a pesar de que ésta se esté moviendo.

7.1. Sistema de control de vuelo

En el sistema de control de vuelo fue necesario realizar algunos ajustes a nivel de hardware y software para poder unificarlo con el controlador visual. Para esto, como se menciona anteriormente, se hizo uso de los puertos seriales (UART) de cada uno de los dispositivos para realizar la comunicación entre ellos. Se interconectaron la UART de telemetría de Ardupilot con la UART 4 de BeagleBone Black, gracias a esto es posible transmitir información desde y hacia el controlador de vuelo. Adicional a lo anterior fue necesario implementar un mecanismo que permita interpretar toda la información referente a sensores, IMU, GPS etc. desde el controlador de vuelo. Para esto se usó el protocolo MAVLink (Micro AerialVehicle Link), el cual consiste en una serie de instrucciones que pueden ser interpretados y ejecutados por una gran variedad de controladores de vuelo. Se recomienda al lector dirigirse a la página oficial (Meier, 2014) para tener una referencia más amplia del protocolo. Gracias a este protocolo fue posible implementar una rutina en BBB que permita leer la información recibida a través de la UART4 y decodificarla con MAVLink.

Ardupilot posee gran variedad de modos de vuelo, que permiten a quadrotor responder de diferentes maneras. En este proyecto se hizo uso de tres modos de vuelo principalmente.

Modo TakeOff: En este modo el controlador genera un despegue controlado, basado en la posición GPS de partida, es decir, Ardupilot toma como referencia la posición GPS antes de despegue e intenta mantener esta posición hasta alcanzar una altura predeterminada por el usuario. Una ventaja de este vuelo es que tiene la capacidad de activar de forma paralela el un modo de seguimiento GPS llamado GUIDED mientras el quadrotor despega.

Modo Guided: En este modo Ardupilot es capaz de interpretar una posición GPS enviada por un sistema externo y dirigir el quadrotor hasta dicha posición.

Modo PosHold: En este modo de vuelo el controlador de vuelo usa una unificación de todos los sensores (IMU, Barómetro, GPS) para tratar de mantener una posición constante

en vuelo, esto quiere decir que se ajustan constantemente los ángulos de Roll, Pitch, Yaw y la propulsión de los motores conocida como Throttle. Se puede considerar que este modo como vuelo estacionario y dentro de todos los esquemas de control planteados se supone que el quadrotor se mantiene estable sobre un punto del espacio.

Modo Land: En este modo el controlador de vuelo toma como referencia la posición GPS cuando el modo se activa e intenta realizar un aterrizaje a 30cm/s. Por otro lado no existe un modo de vuelo que permita realizar un despegue controlado hasta una altura indicada, sin embargo puede realizarse gracias al protocolo MAVLink. Para este caso lo primero es activar el modo PosHold en tierra, posteriormente enviar un comando para incrementar la potencia de los motores y con la ayuda de BBB monitorear la altura del quadrotor. Una vez se alcance la altura deseada se mantiene la potencia de los motores estable y se deja que el control de vuelo mantenga al quadrotor gracias al modo PosHold.

Gracias a los modos descritos anteriormente, se define el algoritmo que el control de vuelo que debe llevar a cabo el quadrotor para realizar el aterrizaje sobre una plataforma en tierra.

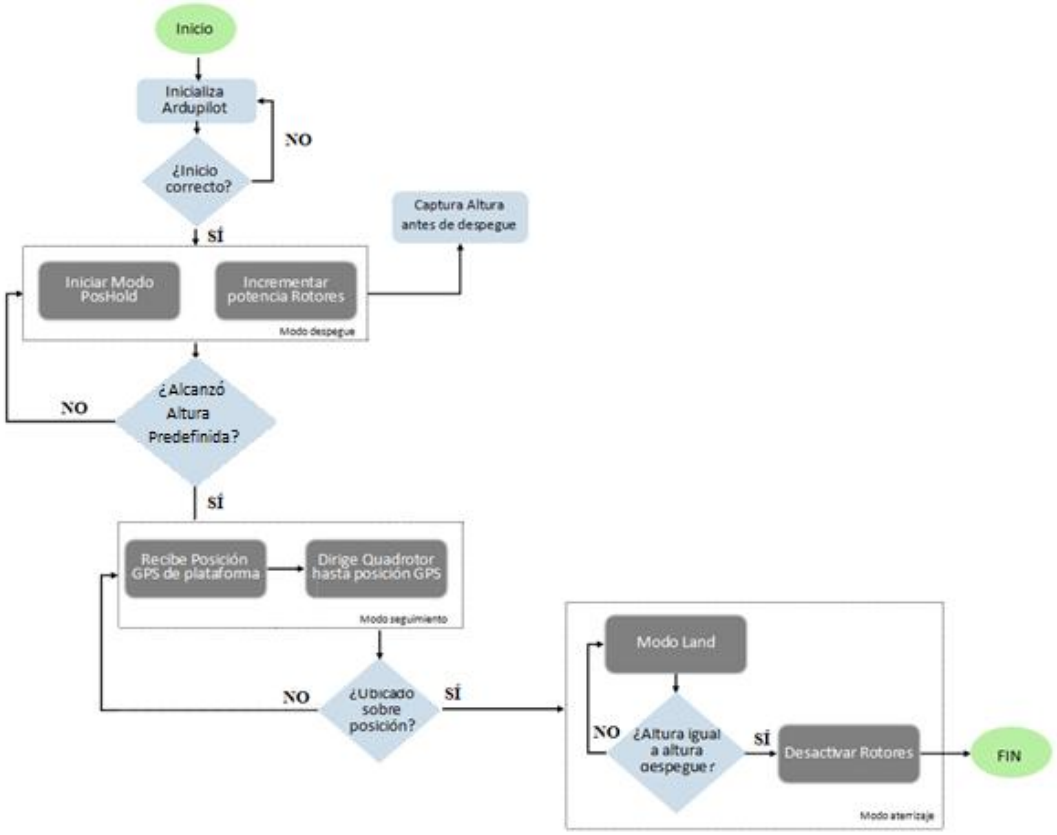

Fig. 23. *Diagrama de bloques del algoritmo propuesto para el proceso de aterrizaje del quadrotor*

En la figura 23 se describe el proceso implementado en el control de vuelo para realizar el aterrizaje sobre la plataforma. En principio el control inicializa la calibración de sensores e informa cuando el proceso concluye, una vez hecho esto se activa el modo PosHold y se aumenta gradualmente la señal que controla la potencia en los rotores a través de MAVLink. Antes de despegar se toma como referencia la altura de despegue para tener una referencia al momento de aterrizar, es decir para saber cuándo el quadrotor ha aterrizado (aterriza si altura final es aproximadamente igual a altura final). La recepción de coordenadas GPS, provenientes de la plataforma en tierra se mantiene activa hasta que el quadrotor detecta la plataforma. Una vez esta posición es alcanzada y el sistema de visión ha detectado la plataforma se procede a activar el descenso mediante el modo Land.

7.2. Sistema de control visual

Este controlador entra a afectar directamente la entapa en la cual el quadrotor inicia el descenso hacia la plataforma de aterrizaje. El controlador visual activa el modo Land cuando la plataforma se detecta. En este punto el controlador visual envía comando de velocidad en las coordenadas X y Y (basado en el sistema de referencias del quadrotor) para intentar mantener entrada la plataforma de aterrizaje (más información en la sección 5.3.2). En este caso el controlador visual describe el siguiente algoritmo.

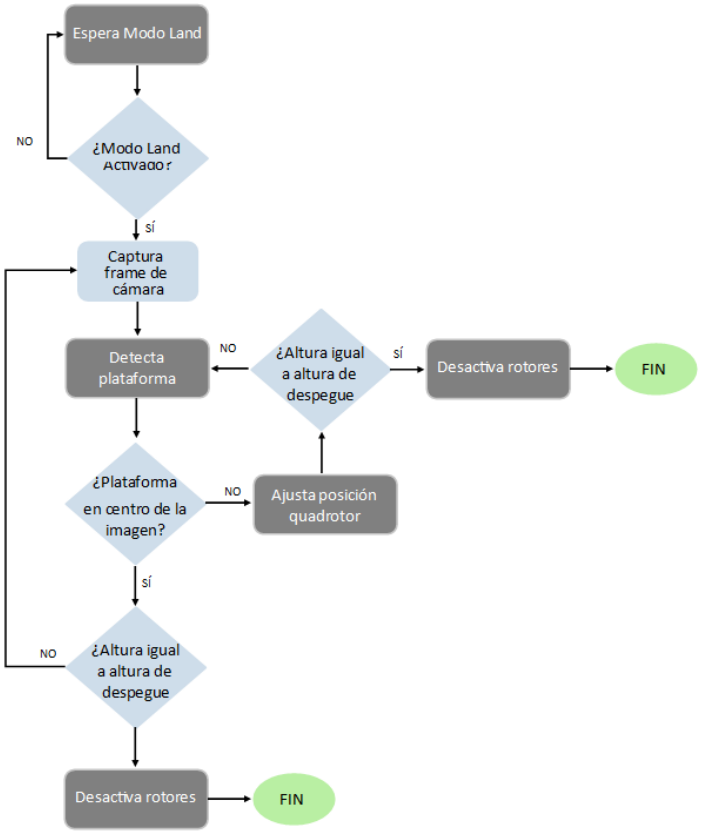

Fig. 24. *Diagrama de bloques del algoritmo de control servo-visual*

7.3. Plataforma de aterrizaje

Finalmente la plataforma de aterrizaje es la encargada de “guiar” al quadrotor hasta el punto donde debe descender. Para esto se genera una conexión a través del protocolo MAVLink entre Raspberry Pi y Ardupilot mediante una red WIFI. La plataforma cuenta con un sensor GPS que permite detectar su posición, la cual es transformada en paquetes MAVLink y enviada al controlador del vuelo del quadrotor (Ardupilot).

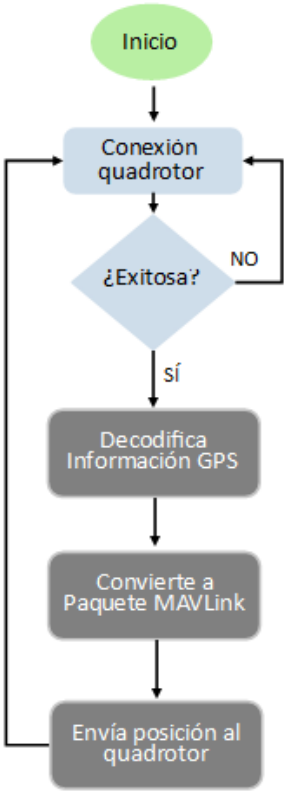

Fig. 25. *Diagrama de bloques del algoritmo para la plataforma de aterrizaje*

8. ANÁLISIS DE RESULTADOS

En esta sección se da a conocer el proceso de pruebas llevado a cabo. Principalmente se enfoca en analizar qué tan buena resulta la corrección de la posición GPS del Quadrotor cuando se usa el control servo visual. Para esto se envía desde la estación de tierra la coordenada GPS a la cual el Quadrotor debe dirigirse, una vez ubicada, el control de alto nivel dirige el Quadrotor hasta la posición deseada ubicando con la cámara la plataforma de color rojo.

Fig. 26. *Quadrotor dirigiéndose hacia la posición GPS de la plataforma de aterrizaje*

La figura describe varias tomas capturadas desde la cámara interna del quadrotor, donde se observa todo el proceso que describe el quadrotor, desde el despegue hasta el aterrizaje.

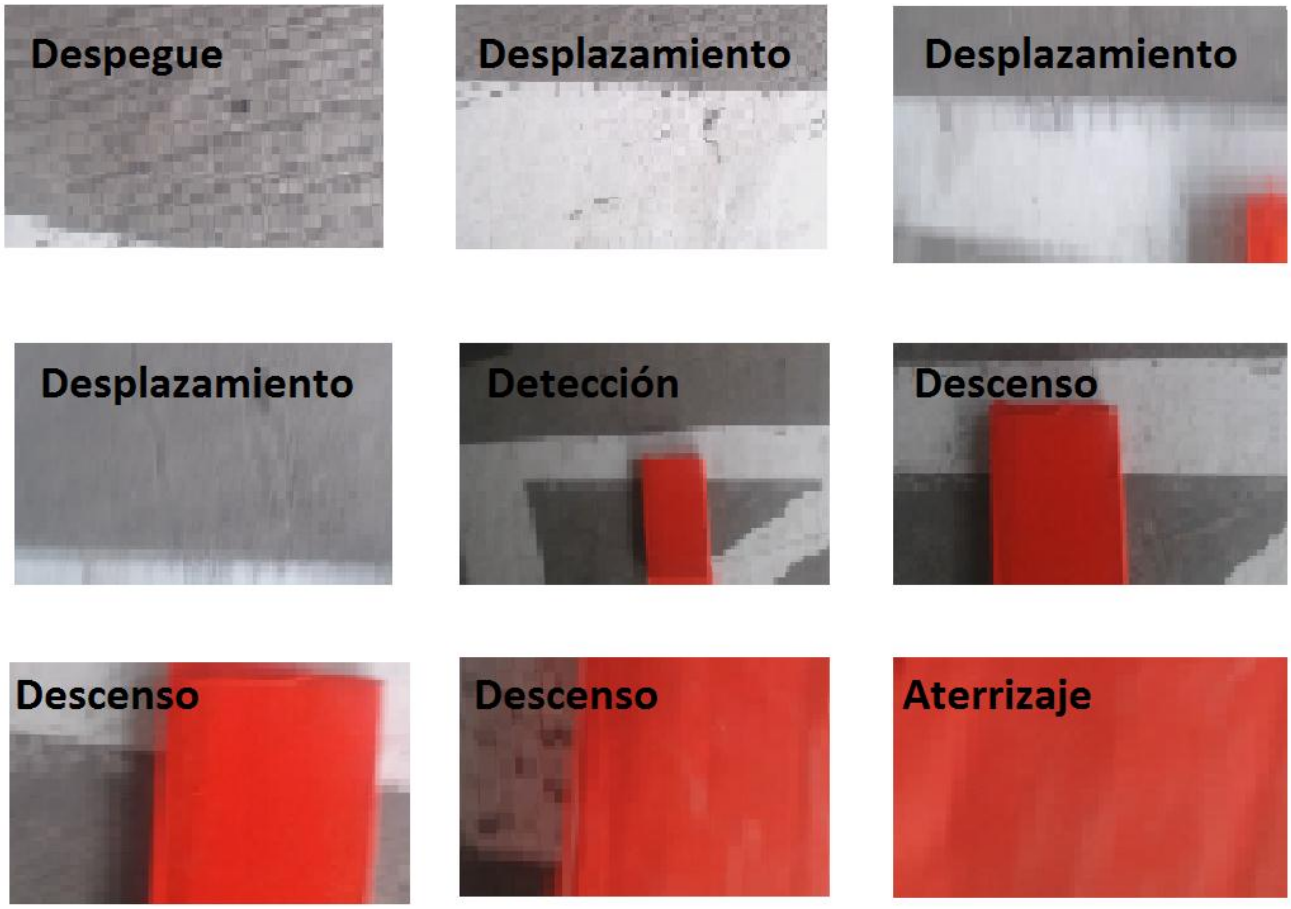

Fig. 27. *Descripción de las etapas o fases de vuelo cumplidas por el quadrotor.*

En la figura 27 se observa una etapa de despegue, en la que el quadrotor toma altura, seguida de una etapa de desplazamiento hacia la posición GPS de la plataforma, finalmente las etapas de descenso y aterrizaje donde el control servo visual ajusta la posición del quadrotor para lograr aterrizar sobre la plataforma.

En la etapa de despegue el controlador de bajo nivel toma como base la posición GPS antes del despegue y aumenta la propulsión de los rotores para despegar. Una vez realizado el despegue se aceptan coordenadas GPS entrantes provenientes de la plataforma de aterrizaje, momento en el cual el controlador de bajo nivel genera el desplazamiento hasta la posición GPS enviada por la plataforma. Justo en el momento de detectar la plataforma, el controlador de bajo nivel conmuta al modo PosHold e inicia el descenso hacia la plataforma, siempre que la marca roja se haya detectado por más de 3 segundos. Cuando se inicia el descenso el controlador servo visual envía la orden de conmutar al modo Land y toma control de los ángulos de Roll y Pitch para ajustar la posición del quadrotor en todo momento y lograr realizar el aterrizaje.

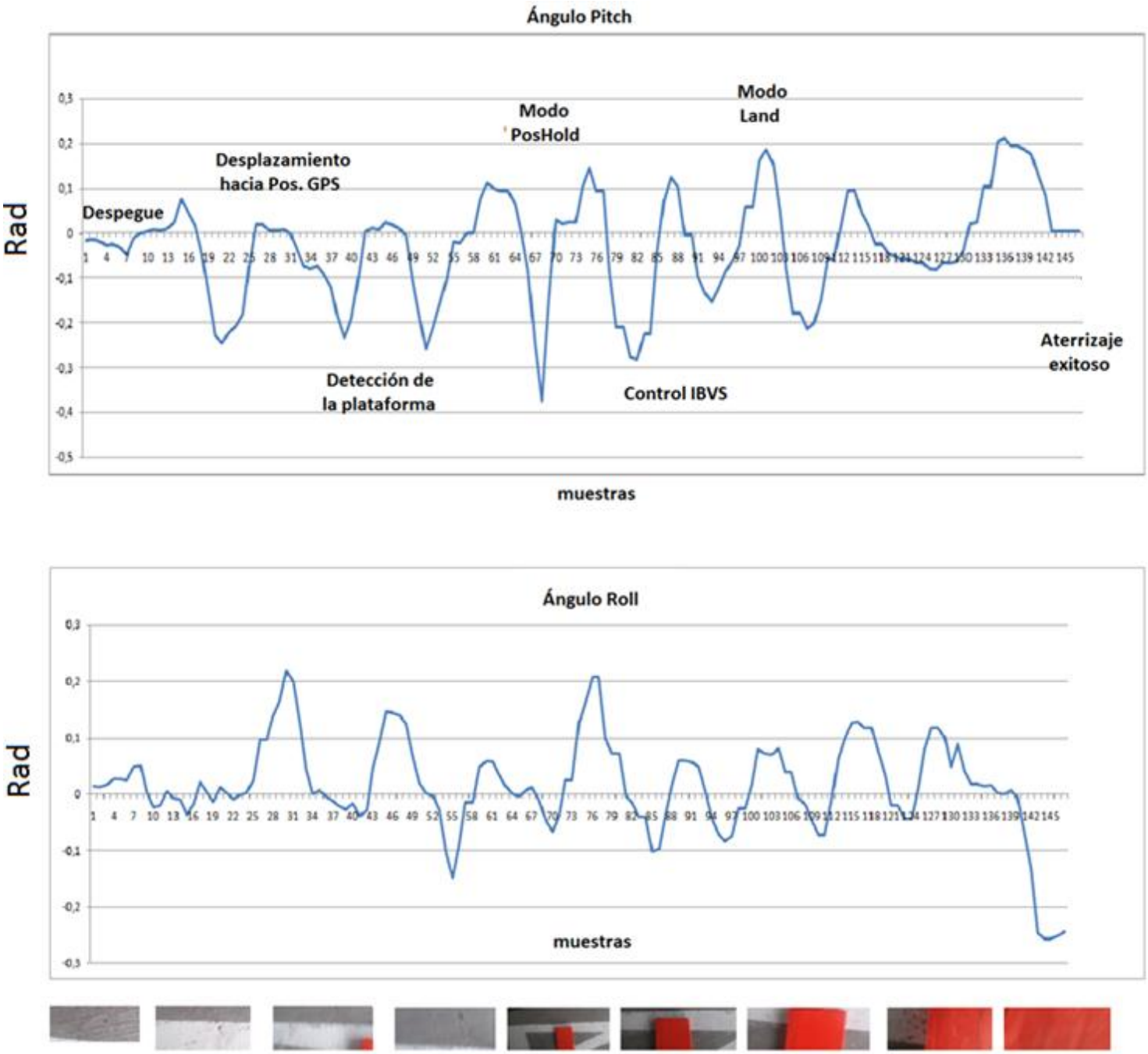

Fig. 28. Gráfico de los ángulos Roll y Pitch desde el despegue hasta el aterrizaje sobre la plataforma. La descripción de los modos de vuelo (en la figura de Roll) describe los estados ejecutados por el sistema de acuerdo a los eventos capturados por el controlador servo visual. En la parte inferior se observa las imágenes capturadas por el sistema correspondiente a cada fase de vuelo.

En la figura *28* se observa el comportamiento de los ángulos de Roll y Pitch cuando se ejecuta un desplazamiento guiado por GPS hasta la plataforma, una estabilización del quadrotor de la imagen captada y un descenso controlado. En primera instancia el quadrotor despega desde su posición inicial, ejecuta la secuencia que permite ser guiado por la plataforma de aterrizaje quien envía su coordenada GPS, una vez posicionado sobre la coordenada y detectada la plataforma se activa el modo PosHold que permite mantener el quadrotor en una posición estable sobre la plataforma. Una vez se detecta la plataforma por un periodo de 3 segundos se da inicio al descenso, Modo Land, donde a medida que el quadrotor disminuye su altura, el controlar servo visual intenta mantener la plataforma centrada para de esta forma realizar un aterrizaje exitoso.

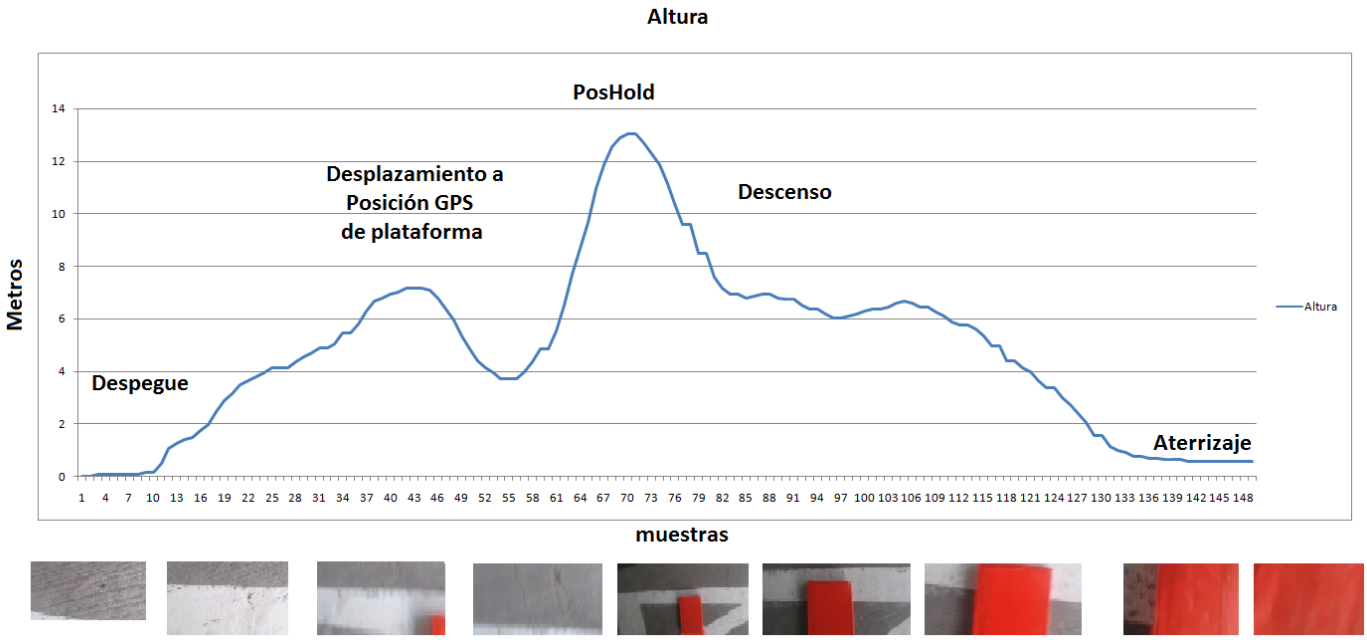

Fig. 29. Comportamiento de la altura en todas las etapas del proceso (despegue - aterrizaje).

En la figura 29 se observa cómo es el comportamiento del quadrotor referente a su altura. El despegue se hace de forma suave, gracias al controlador de bajo nivel, posteriormente se procede a hacer el seguimiento de la posición GPS de la plataforma de aterrizaje, sin embargo el quadrotor sufre una descompensación de aproximadamente dos metros de altura. Esto es un comportamiento inesperado de Ardupilot. En el desplazamiento se espera que la ganancia de altura se realice de manera constante, sin embargo el comportamiento es diferente. Con la información proporcionada por Ardupilot no es posible afirmar a qué se debe esta descompensación de altura, debido a que Ardupilot se limita, en el protocolo Mavlink a enviar información referente a la IMU (Attitude), GPS, Velocidad pudo ser ocasionada por la pérdida de referencia de uno o varios satélites GPS,

lo que puede llegar a descompensar la altura del quadrotor mientras se adapta a la pérdida. Posterior a esta descompensación el controlador de bajo nivel compensa la pérdida de altura hasta que el controlador servo visual detecta la marca y da inicio al descenso hacia la plataforma. Se puede observar que la altura final es mayor a la altura inicial, esto en consecuencia de que el quadrotor ha aterrizado sobre la plataforma

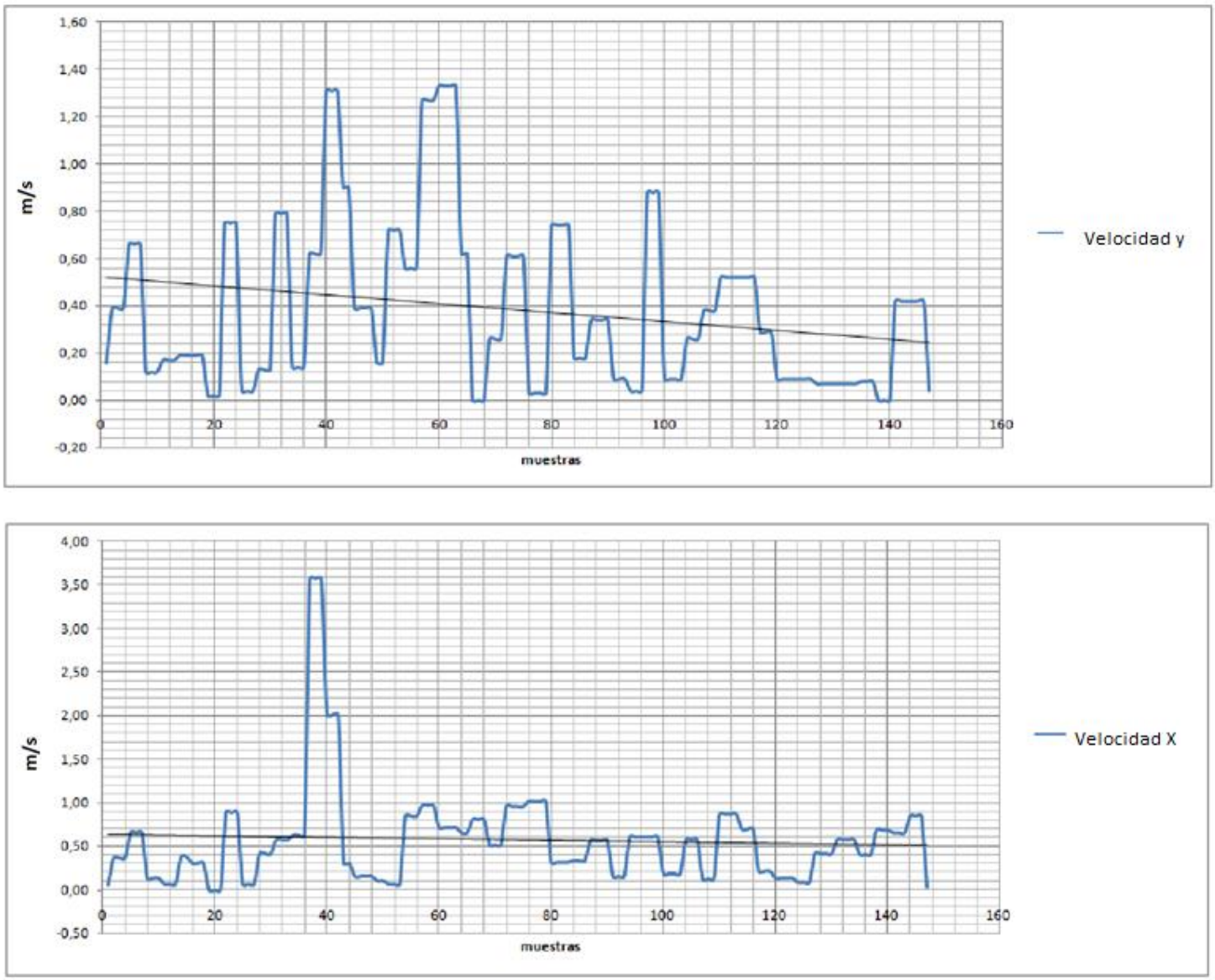

Fig. 30. Velocidad de desplazamiento en los ejes horizontales mientras se ejecuta el proceso de despegue, desplazamiento y aterrizaje.

En la figura *30* se observa cómo se afectan las velocidades en los ejes horizontales en cada etapa del proceso de aterrizaje. Al iniciar el despegue la velocidad en x aumenta considerablemente, debido a que se da inicio al desplazamiento hacia la plataforma de aterrizaje, aquí se observa que la velocidad en y no se ve afectada considerablemente, es decir el quadrotor mantiene un desplazamiento aproximadamente lineal hacia la plataforma de aterrizaje. Cuando se llega hasta la coordenada GPS de la plataforma se ejecuta el modo PosHold para posteriormente dar inicio a la secuencia de aterrizaje. Cuando se ejecuta este modo se observa que la velocidad en x se ajusta de una manera más agresiva que la velocidad en y, esto se debe a que el control IBVS logra estabilizar la imagen de la plataforma de aterrizaje captada por la cámara en el eje Y, pero debe ajustar constantemente la posición en el eje X para mantener centrada la imagen. A medida que se acerca al punto de aterrizaje, las velocidades se ajustan con menor intensidad debido a que la imagen de la plataforma tiene a estar siempre centrada. Finalmente las el quadrotor aterriza llevando las velocidades en los ejes X y Y a cero.

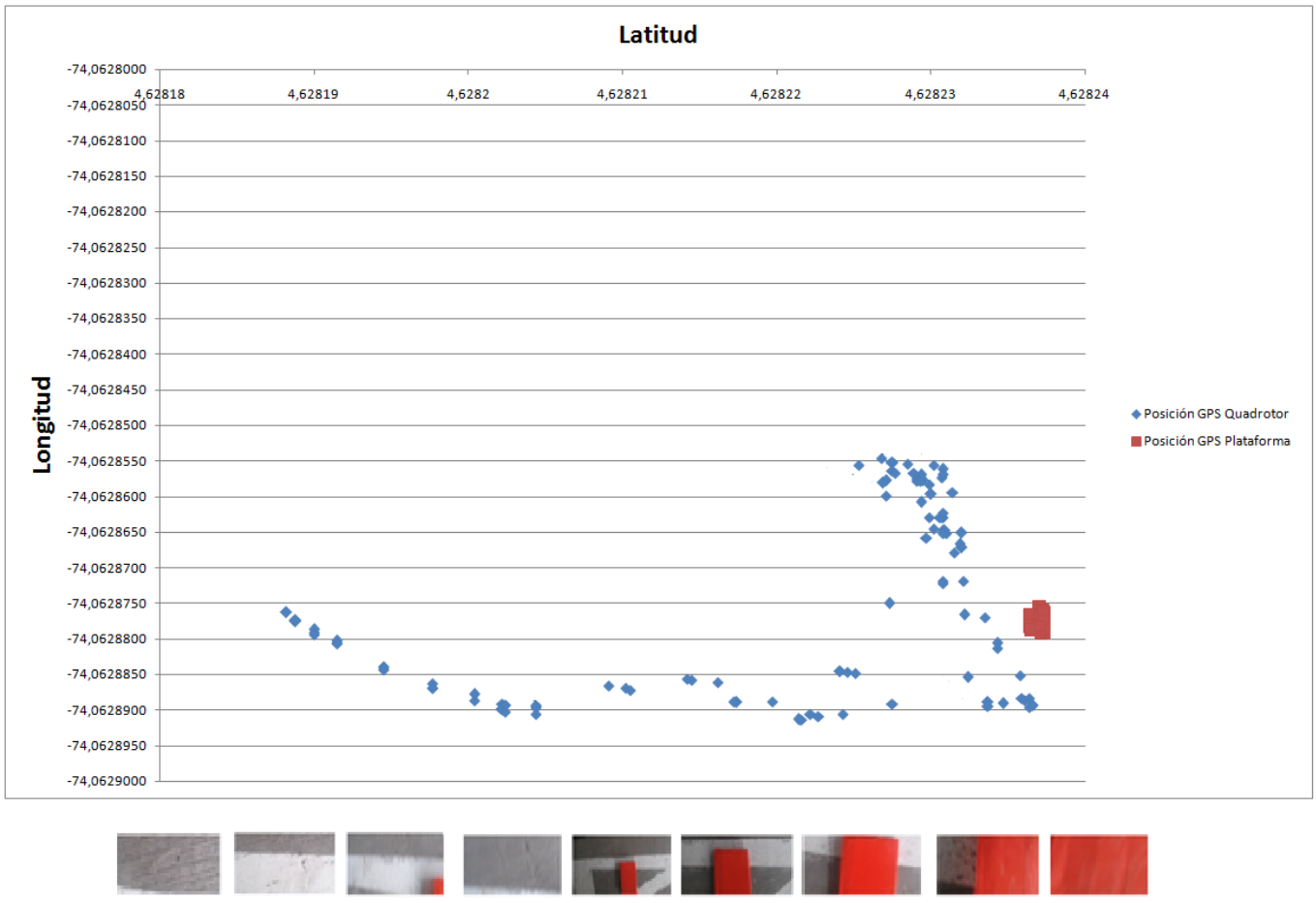

Fig. 31. Comportamiento latitudinal y longitudinal del quadrotor al desplazarse hacia la plataforma de aterrizaje

En la figura 31 se observa que el quadrotor realiza un desplazamiento hacia la coordenada GPS enviada por la plataforma de aterrizaje, justo cuando éste se posiciona sobre ésta el controlador IBVS comienza a ajustar la posición del quadrotor para intentar mantener la plataforma centrada, esto se observa en el conjunto de puntos GPS ejecutados hasta concluir con el aterrizaje. Podemos observar que cuando la plataforma de aterrizaje es detectada y se da inicio al descenso la posición representada en la coordenada de latitud se mantiene sin cambios “agresivos” mientras que en la longitud los, ajustes de la posición son más evidentes.

Al realizar una comparación entre la ejecución de la prueba con, y sin el controlador IBVS podemos evaluar la corrección dada por el controlador a la posición GPS del quadrotor.

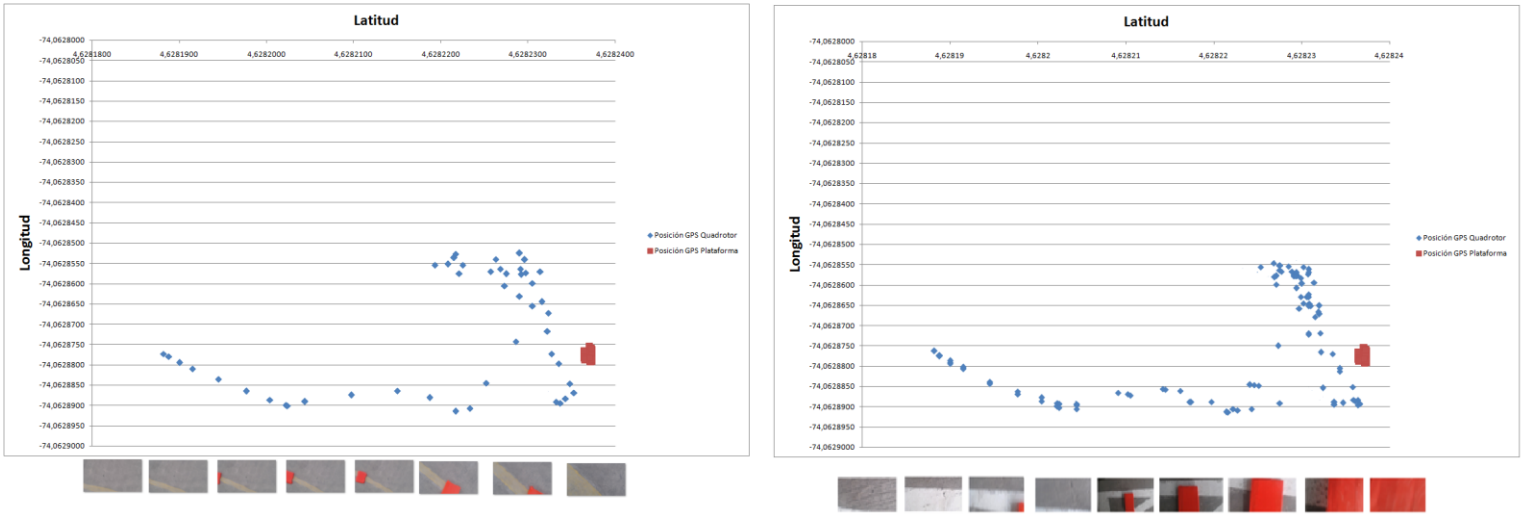

Fig. 32. Comparación del posicionamiento GPS realizado por el quadrotor con y sin control servo visual.

En la *figura 32* se observa que en ausencia del control servo visual el quadrotor describe una trayectoria regular hasta llegar a la posición que envía la plataforma, una vez llega a ella inicia el descenso, donde se presenta un desequilibrio de la posición ya que el Modo Land, como se explica en la sección 6, el Ardupilot intenta mantener la posición GPS que se detectó justo cuando se activa el modo Land, debido al error en el GPS la posición del quadrotor con respecto a la plataforma se ve afectada, mientras que si el controlador visual ajusta los ángulos Roll y Pitch la trayectoria descrita trata de aproximarse más a aterrizar en la plataforma. Gracias a esto se observa que el control servo visual sí corrige la posición GPS del quadrotor, esto se evidencia en la cantidad de coordenadas que se observan en la figura 32 (con IBSV) ya que continuamente el controlador visual, ajusta la posición del quadrotor, lo que se traduce en más coordenadas GPS descritas por el quadrotor.

Para realizar una validación más precisa del comportamiento del quadrotor con IBVS y sin IBVS realizamos una conversión de los datos longitudinales y latitudinales a la escala métrica y calcular la distancia entre la coordenada GPS del quadrotor y la coordenada GPS de la plataforma. Para realizar este proceso se usó la descripción publicada en *(Veness, 2013)*

$$Distancia_{km} = Radio_{Tierra} * acos(sin(lat1) * sen(lat2) + cos(lat1) * cos(lat2) * cos(lon1 - lon2)) \quad (34)$$

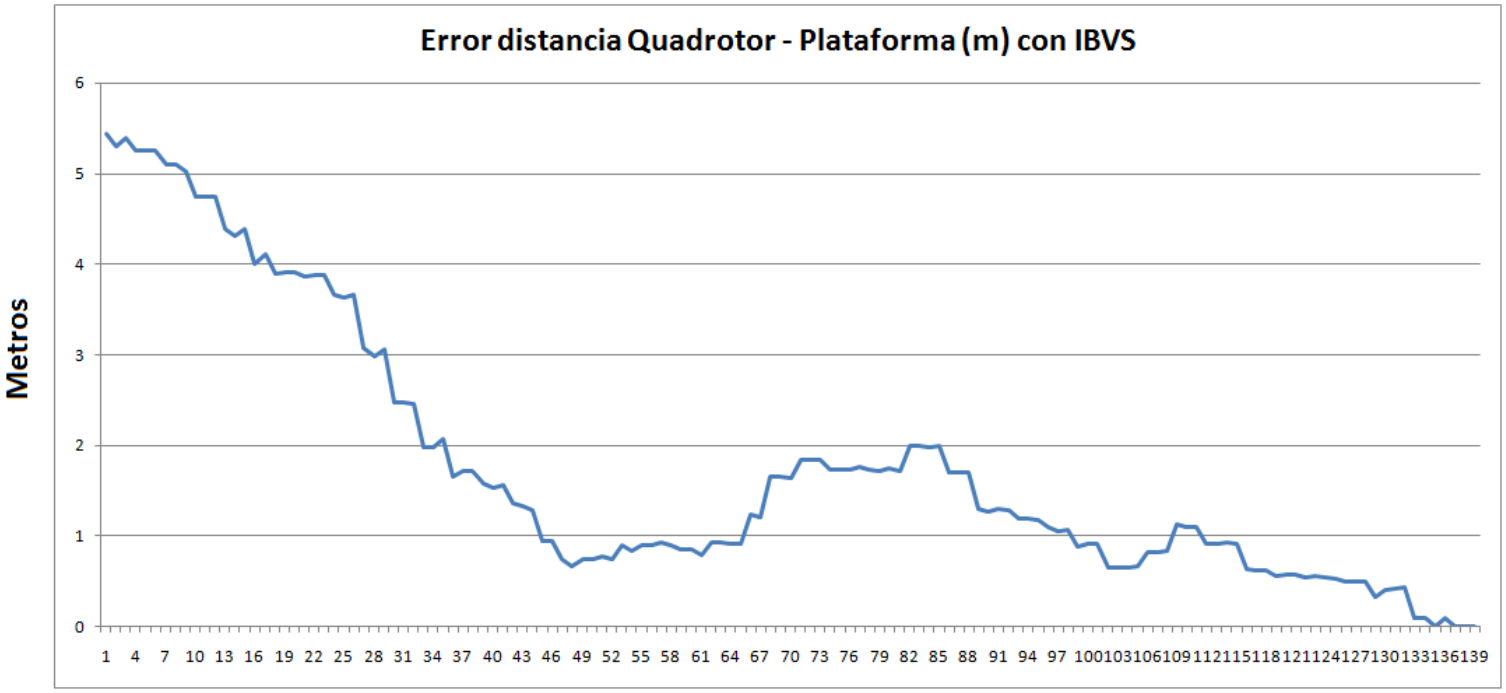

Fig. 33. Erro (m) entre la distancia entre el quadrotor y la plataforma de aterrizaje desde el despegue hasta el aterrizaje usando IBVS. Error Cuadrático Medio EMC = 5,04 m^2

La figura 33 describe cómo el quadrotor se acerca a la plataforma de aterrizaje. Justo al final de las muestras, el quadrotor logra alcanzar la plataforma de aterrizaje. Eso se concluye teniendo en cuenta que el error disminuye a medida que el quadrotor se acerca a la plataforma de aterrizaje.

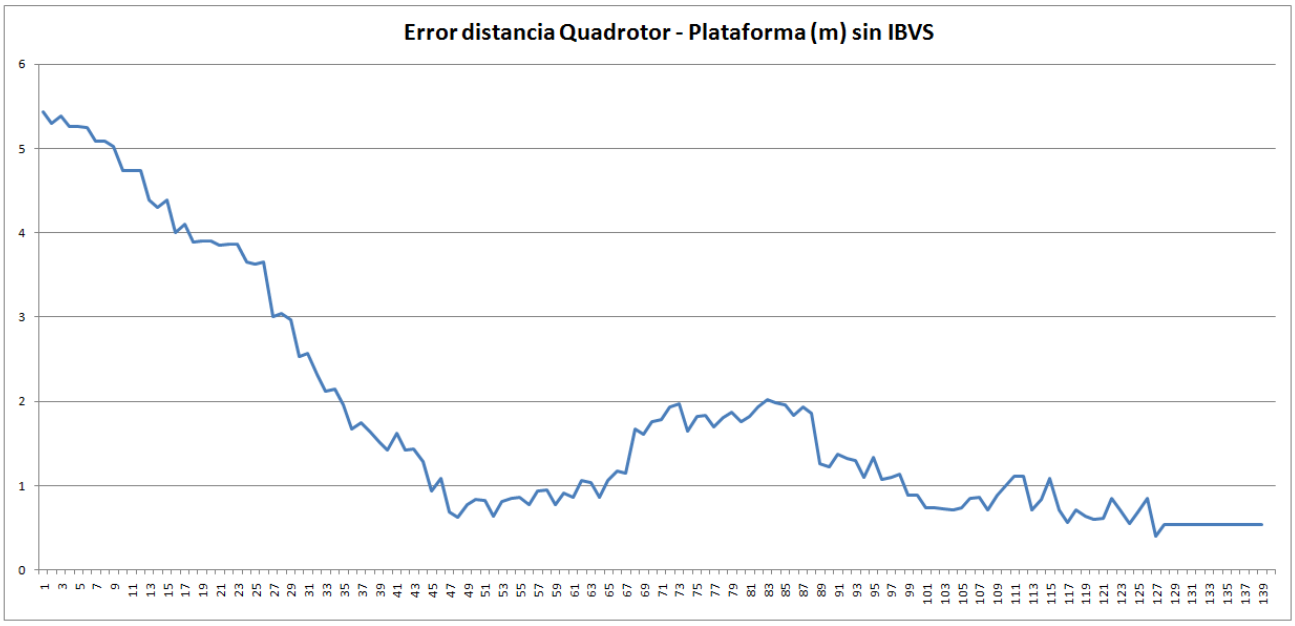

Fig. 34. Erro (m) entre la distancia entre el quadrotor y la plataforma de aterrizaje desde el despegue hasta el aterrizaje sin usar IBVS. Error Cuadrático Medio EMC = 5,11 m^2

En este caso es más evidente que la prueba realizada con el controlador servo visual es más eficiente ya que logra minimizar el error de posición entre el quadrotor y la plataforma lo que se traduce en aterrizar sobre la plataforma, mientras que en la prueba realizada sin IBVS el error se minimiza en proporciones similares al de la *Fig. 33* sin embargo el error se estabiliza en un valor mayor a cero, esto significa que el quadrotor se no logra aterrizar sobre la plataforma, lo que nos confirma el efectivo funcionamiento del control servo visual al compensar la posición GPS enviada por la plataforma de aterrizaje al quadrotor.

9. CONCLUSIONES

En el desarrollo e implementación del controlador visual se concluye que es acertado usar un las salidas de control basadas en la velocidad de translación en los ejes, ya que esto hace los movimientos traslacionales sean suaves, de tal manera que la cámara no se vea afectada con las vibraciones generadas por el quadrotor al desplazarse en el espacio.

Uno de los puntos a considerar en este proyecto sin duda es la capacidad de procesamiento del controlador visual ya que constituye una limitante muy importante. La capacidad de procesamiento a bordo se define en un aspecto fundamental; FPS (Frames Por Segundo), esta característica nos da una idea de qué tan bueno puede llegar a ser el controlador visual. Entre mayor sea la capacidad de procesamiento del controlador visual, mayor es la eficiencia del algoritmo implementado ya que el controlador puede procesar las imágenes con mayor fluidez y responder a los cambios captados por la cámara con mayor rapidez. En las pruebas del controlador visual se observó que éste tiene una respuesta de 5 a 10 FPS, una velocidad de procesamiento aceptable para una plataforma como BBB, sin embargo lo ideal en este tipo de controladores es tener la información del entorno tan rápido como sea posible. También la cámara es una limitante, ya que la capacidad de ajustarse a las condiciones de luminosidad redice el FrameRate que es capaz de procesar.A pesar de la capacidad de procesamiento limitada, el controlador responde de acuerdo a lo esperado y logra controlar el quadrotor de manera precisa bajo óptimas condiciones de luminosidad y climatológicas.

El FrameRate afecta al momento de estabilizar la imagen de la plataforma en el centro del plano, esto se evidencia en los continuos cambios de posición de la Fig. 32 *(con IBVS)*, ya que con un FrameRate superior se podrían conseguir valores menos dispersos, lo que se traduce en movimientos más suaves, ya que la diferencia de posición entre un frame antes y uno después hace que el controlador servo visual deba ajustar la posición del quadrotor de manera “brusca” para mantener centrada la plataforma.

Se observa que el controlador servo visual, ayuda a corregir la posición GPS del sistema, sin embargo, el resultado esperado no es de acuerdo a lo esperado, ya que cuando no se usa el controlador servo visual, la posición del Quadrotor al aterrizar, con respecto a la posición del quadrotor al aterrizar con ayuda del controlador servo visual son muy cercanas, lo que nos da a ente entender que en exteriores, el GPS no es tan impreciso, y depende de munchas condiciones, climatológicas, ruido producido por otros sistemas como antenas que puedan afectar el desempeño. Por lo tanto teniendo una un buen espacio de prueba, fuera de interferencias electromagnéticas, el GPS tiene un desempeño bastante bueno.

10. BIBLIOGRAFÍA

3DRobotics. (04 de 12 de 2013). *APM 2.6 set*. Obtenido de 3DRobotics: http://store.3drobotics.com/products/apm-2-6-kit-1
Ascending Technologies. (05 de 05 de 2014). *asctec.* Obtenido de Ascending Technologies: http://www.asctec.de/en/uav-uas-drone-products/asctec-hummingbird/

Bouabdallah, P. M. (2004). Design and Control of an Indoor Micro Quadrotor. *IEEE Int. Conf. on Rob. and Automat., volume 5* (págs. 4393–4398). New Orleans, USA: IEEE.
Chaumette, F. a. (2006). Visual servo control. I. basic aproaches. *Robotics and Aitomation Magazine 13* , 82-90.

Corke, P. (1996). *Visual Control of Robots: high performance visual servoing.* Australia: Research Studies Press LTDCSIRO. Division of Manufacturing Technology.

Globalsat Products. (01 de 11 de 2014). Obtenido de Globalsat: http://www.globalsat.com.tw/products-page.php?menu=2&gs_en_product_id=2&gs_en_product_cnt_id=61

Hutchinson, S. H. (1996). A tutorial on visual servo control. *Transaction on Robotics and automation volume 12* , 651-670.

Hyon Lim, J. P. (28 de 01 de 2014). *Build your own quadrotor.* Obtenido de Open-Source Projects on Unmanned Aerial Vehicles: http://ricardofm.com/cms/attachments/article/89/build-your-own-quadrotor.pdf

Instruments, T. (07 de 04 de 2014). *beagleboard*. Obtenido de BeagleBone Black: http://beagleboard.org/black

Iván F. Mondragón, P. C. (2011). 3D Object following based on visual information for Unmanned Aerial Vehicles. *The Latin American Robotics Competition (LARC).* Bogotá, Colombia.

José Luis, R. L. (2012). Real-time localization of an UAV using Kalman filter and a Wireless Sensor Network. *Journal of intelligent & Robotic System* , Vol. 65 283-293.
K. E. Wenzel, A. M. (2010). Automatic Take Off, Tracking and Landing of a Miniature UAV on a Moving Carrier Vehicle. *Journal of Intelligent & Robotic Systems Vol. 61* , 221-238.

Logic Supply. (01 de 11 de 2014). *Wireless adapter uwn200*. Obtenido de logicsupply: http://www.logicsupply.com/components/networking/wireless/uwn200/

Lozano, R. Z. (2011). Stabilization of a helicopter using optical flow. *Electrical Engineering Computing Science and Automatic Control (CCE), 2011 8th International Conference on*

(págs. 1 - 6). Merida City: IEEE.

Meier, L. (01 de 11 de 2014). *QGroundControl*. Obtenido de Mavlink Protocol: http://qgroundcontrol.org/mavlink/start

Mondragón, I. (2011). *On-board visual control algorithms for Unmanned Aerial Vehicles.* Madrid, España: Universidad Politécnica de Madrid.

Moses Bangura, R. M. (2012). *Nonlinear Dynamic Modeling for High Performance Control of a Quadrotor.* Victoria University of Wellington. New Zealand: Proceedings of Australasian Conference on Robotics and Automation.

Raffo, G. V. (2007). *Modelado y control de un helicóptero quadrotor.* Sevilla: Universidad de Sevilla.

Raspberry Pi. (01 de 04 de 2014). *The raspberrypi org.* Obtenido de Raspberry Pi: http://www.raspberrypi.org/

Real, C. N. (2009). *Control de un Quadrotor mediante la plataforma Arduino.* Universidad politécnica de Cataluña.

Santos, M. (2010). *Intelligent fuzzy controller of a quadrotor.* Madrid, Spain : Fac. de Inf., Univ. Complutense.

Skrzypietz, T. (2012). *Unmanned Aircraft Systems for Civilian Missions.* Brandenburg Institute for Society and Security.

u-blox. (04 de 11 de 2014). *u-blox*. Obtenido de Products GPS G6 HW: https://www.u-blox.com/images/downloads/Product_Docs/LEA-6_ProductSummary_%28GPS.G6-HW-09002%29.pdf

Veness, C. (06 de 11 de 2013). *cpearson.* Recuperado el 05 de 11 de 2014, de Latitude, Longitude and Great Circles: http://bluemm.blogspot.com/2007/01/excel-formula-to-calculate-distance.html

Zisserman, R. H. (5 de 11 de 2014). *Mulple View Geometry in Computer Vision.* Obtenido de robots: http://www.robots.ox.ac.uk/~vgg/hzbook/

11. ANEXOS

Anexo 1. Código control Visual

```
fromsubprocess import Popen, PIPE, STDOUT

from droneapi.lib import VehicleMode
frompymavlink    import mavutil

importAdafruit_BBIO.UART as UART
import time
importos
import cv2
import sys
import serial
importnumpy as np

#-------------------------------------------------------------------------------------------------------------
------------#
#-----------------------------------------------------------Functions-----------------------------------------
----------------#
#-------------------------------------------------------------------------------------------------------------
------------#

def delay(seconds):

        global control5
        globaltimesleep
        global entrada
        global ReffTime3

        ok = 0
        if control5 != 1:

                if entrada == 0:
                        ReffTime3 = time.time()
                        entrada = 0
                else:
```

```
                    ReffTime3 = time.time()

            control5 = 1

        CurrentTime = time.time()
        timesleep = (CurrentTime - ReffTime3)

        iftimesleep> seconds:
            entrada = 1
            timesleep = 0
            control5 = 0
            ok = 1

        return ok


#-------------------------------------------------------------------------------------------------------------------
-----------#
#-----------------------------------------------------Data declaration---------------------------------------
-----------------#
#-------------------------------------------------------------------------------------------------------------------
-----------#

counter = 0

threshold  = 200;
threshold2 = 10000;
j = 0;
control  = 0
control2 = 0
control3 = 0
control4 = 0
control5 = 0
control6 = 0
entrada2 = 0

timesleep = 0
entrada   = 0
ReffTime3 = 0

NotDetecTime = 0

throttle = 0
AuxTime2 = 0
```

```
#Mejor filtro rojo dia
H_MIN = 168;
S_MIN = 176;
V_MIN = 52;

H_MAX = 181;
S_MAX = 211;
V_MAX = 180;

min_red = np.array([H_MIN,S_MIN,V_MIN])
max_red = np.array([H_MAX,S_MAX,V_MAX])

pt1  =np.array([30,30])
pt2  =np.array([402,210])
centroid = np.array([0,0])
i = 0


#-------------------------------------------------------------------------------------------------------------
------------#
#-----------------------------------------------------Start logic-----------------------------------------
--------------#
#-------------------------------------------------------------------------------------------------------------
------------#
UART.setup("UART4")

os.system('reset')

start_time = time.time()

fname1 = "location"
fname2 = "velocity"
fname3 = "IMU"

while True:

        ifos.path.isfile(fname1):
                fname1 = fname1+str(i)+'.txt'
                break
        else:
                i = 0
                break
        i += 1
```

```
while True:

        ifos.path.isfile(fname2):
                fname2 = fname2+str(i)+'.txt'
                break
        else:
                i = 0
                break
        i += 1

while True:

        ifos.path.isfile(fname3):
                fname3 = fname3+str(i)+'.txt'
                break
        else:
                i = 0
                break
        i += 1

location = open(str(fname1), "w")
velocity = open(str(fname2), "w")
IMU      = open(str(fname3), "w")

print 'Logs files are: ', fname1, ' ' , fname2, ' ' , fname3

#-----------------------------------------------#
#----------------Preparing Quadrotor--------------#
#-----------------------------------------------#

capture = cv2.VideoCapture(0)
capture.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 432)
capture.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 240)

#cv2.namedWindow("ImgThresh",cv2.WINDOW_AUTOSIZE)
cv2.namedWindow("frame",cv2.WINDOW_AUTOSIZE)
#cv2.namedWindow("hsv",cv2.WINDOW_AUTOSIZE)


print 'Starting APM'

api = local_connect()
v = api.get_vehicles()[0]
```

```
rval, fram = capture.read()
fram = cv2.flip(fram,1)
iffram is None:
print "Get frame Error"

file = 'log' + str(i) + '.jpg'
cv2.imwrite(file, fram)
location.write('log' + str(i) + ' ' + str(v.location))
IMU.write(     'log' + str(i) + ' ' + str(v.attitude))
velocity.write( 'log' + str(i) + ' ' + str(v.velocity))

i=i+1

v.channel_override = { "1" : 0, "2" : 0, "3" : 0, "4" : 0, "5" : 0, }
v.flush()
time.sleep(0.5)

#PosHold
#v.channel_override = { "5" : 1300 }
#v.flush()
print 'Ready'

time.sleep(0.5)
print 'arming throttle'
v.armed = True
v.flush()
print 'Ready'

time.sleep(1.5)

ifv.armed == False:
        v.armed = True
        v.flush()

throttle = 1650
v.channel_override = { "3" : throttle }
v.flush()
print 'take-off'

rval, fram = capture.read()
fram = cv2.flip(fram,1)
iffram is None:
print "Get frame Error"
```

```
file = 'log' + str(i) + '.jpg'
cv2.imwrite(file, fram)
location.write('log' + str(i) + ' ' + str(v.location))
IMU.write(     'log' + str(i) + ' ' + str(v.attitude))
velocity.write( 'log' + str(i) + ' ' + str(v.velocity))
i=i+1

#-----------------------------------------------#
#---------------Start visino logic--------------#
#-----------------------------------------------#


#while True and v.mode<> "STABILIZE":
while True:

rval, fram = capture.read()
fram = cv2.flip(fram,1)
iffram is None:
                print 'get frame error'
        break

imgHSV = cv2.cvtColor(fram, cv2.COLOR_BGR2HSV)
imgThresh = cv2.inRange(imgHSV,min_red, max_red)

    #imgThresh = cv2.erode(imgThresh,None, iterations = 1)
    #imgThresh = cv2.dilate(imgThresh,None, iterations = 7)

contours, hierarchy = cv2.findContours(imgThresh, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)

        objects = len(contours)

        if throttle < 1300:
                print 'The Quadrotor has landed'
                break

        if objects < 1 and control == 0:
                if control2 != 1:
                        ReffTime = time.time()
                        print '\nNot Rover detected'
                control2 = 1

                CurrentTime = time.time()
```

```
                NotDetecTime = int (CurrentTime - ReffTime)
                out3 = 'time ' + str(NotDetecTime)
                sys.stdout.write('\r'+out3)
                sys.stdout.flush()
                ifNotDetecTime> 15:
                        control2 = 0
                        print '\nCannot detect Rover, landing by security'
                        v.mode = VehicleMode("LAND")
                        v.flush()
                        v.channel_override = { "1" : 0, "2" : 0, "4" : 0, "5" : 0, }
                        v.flush()
                        break


        elif (objects >= 1 and control == 0) or (objects >= 1 and control == 2):
                print '\nRover detected'
                control = 3
                NotDetecTime = 0
                v.mode = VehicleMode("LAND")
                v.flush()


elif (objects < 1 and control == 1) or (objects < 1 and control == 3):
print '\nRover loss'
                control = 2
                NotDetecTime = 0


        elif objects >= 1 and control == 1:
                print 'Setting Quadrotor position for land on Rover'
                NotDetecTime = 0


        if control == 2:
                #v.channel_override = { "5" : 1300 }
                #v.flush()

                NotDetecTime = 0

                if throttle < 1880:

                        if delay(3) == 1:
                                throttle += 110
```

```
                                out6 = 'Tryig to detect rover, setting throttle at: ' +
str(throttle)
                                v.channel_override = { "3" : throttle }
                                v.flush()
                                sys.stdout.write('\r'+out6)
                                sys.stdout.flush()

                        elif throttle > 1880:
                                print '\nMax. Altitude gotten'
                                NotDetecTime = 0
                                control = 0
                                control2 = 0

                forcnt in contours:
                        j += 1
                        contour_area = cv2.contourArea(cnt)
                        if (contour_area> threshold and contour_area< threshold2) and control ==
3:
                                x,y,w,h = cv2.boundingRect(cnt)
cv2.rectangle(fram, (x,y), (x+w,y+h), (0,255,0), 1)
                                centroidx = x+w/2
centroidy = y+h/2

                                if control4 == 0:

                                        #Z1
                                        if (centroidx>= 0 and centroidy>=0) and (centroidx< 128 and
centroidy< 70):
                                                out2 = 'object detected in Z1\n'
                                                sys.stdout.write('\r'+out2)
                                                sys.stdout.flush()

                                                if entrada2 == 0:

                                                        control4 = 1
                                                        channel  = 2
                                                        PWM = 1279
                                                        v.channel_override = { str(channel) : PWM }
                                                        v.flush()
                                                        print command2

                                                elif entrada2 == 1 and delay(0.05) == 1:

                                                        control4 = 1
```

```
                                channel  = 2
                                entrada2 = 0
                                PWM = 1279
                                v.channel_override = { str(channel) : PWM }
                                v.flush()
                                print command2
                        #Z2
                        if (centroidx>= 0  and centroidy>  70) and (centroidx<  128
and centroidy< 170):
                                out2 = 'object detected in Z2\n'
                                sys.stdout.write('\r'+out2)
                                sys.stdout.flush()

                                if entrada2 == 0:

                                        control4 = 1
                                        channel  = 1
                                        PWM = 1679
                                        v.channel_override = { str(channel) : PWM }
                                        v.flush()
                                        print command2

                                elif entrada2 == 1 and delay(0.05) == 1:

                                        control4 = 1
                                        channel  = 1
                                        entrada2 = 0
                                        PWM = 1679
                                        v.channel_override = { str(channel) : PWM }
                                        v.flush()
                                        print command2
                        #Z3
                        if (centroidx>= 0  and centroidy> 170) and (centroidx<  128
and centroidy<= 240):
                                out2 = 'object detected in Z3\n'
                                sys.stdout.write('\r'+out2)
                                sys.stdout.flush()

                                if entrada2 == 0:

                                        control4 = 1
                                        channel  = 2
                                        PWM = 1679
                                        v.channel_override = { str(channel) : PWM }
```

```
                        v.flush()
                        print command2

                elif entrada2 == 1 and delay(0.05) == 1:

                        control4 = 1
                        channel  = 2
                        entrada2 = 0
                        PWM = 1679
                        v.channel_override = { str(channel) : PWM }
                        v.flush()
                        print command2
        #Z4
        if (centroidx> 128 and centroidy> 170) and (centroidx<  304
and centroidy<= 240):
                out2 = 'object detected in Z4\n'
                sys.stdout.write('\r'+out2)
                sys.stdout.flush()

                if entrada2 == 0:

                        control4 = 1
                        channel  = 2
                        PWM = 1679
                        v.channel_override = { str(channel) : PWM }
                        v.flush()
                        print command2

                elif entrada2 == 1 and delay(0.05) == 1:

                        control4 = 1
                        channel  = 2
                        entrada2 = 0
                        PWM = 1679
                        v.channel_override = { str(channel) : PWM }
                        v.flush()
                        print command2
        #Z5
        if (centroidx> 304 and centroidy> 170) and (centroidx<= 432
and centroidy<= 240):

                out2 = 'object detected in Z5\n'
                sys.stdout.write('\r'+out2)
                sys.stdout.flush()
```

```
                    if entrada2 == 0:

                            control4 = 1
                            channel  = 2
                            PWM = 1679
                            v.channel_override = { str(channel) : PWM }
                            v.flush()
                            print command2

                    elif entrada2 == 1 and delay(0.05) == 1:

                            control4 = 1
                            channel  = 2
                            entrada2 = 0
                            PWM = 1279
                            v.channel_override = { str(channel) : PWM }
                            v.flush()
                            print command2
                #Z6
                if (centroidx> 304 and centroidy> 70)  and (centroidx<= 432
and centroidy< 170):
                    out2 = 'object detected in Z6\n'
                    sys.stdout.write('\r'+out2)
                    sys.stdout.flush()

                    if entrada2 == 0:

                            control4 = 1
                            channel  = 1
                            PWM = 1279
                            v.channel_override = { str(channel) : PWM }
                            v.flush()
                            print command2

                    elif entrada2 == 1 and delay(0.05) == 1:

                            control4 = 1
                            channel  = 1
                            entrada2 = 0
                            PWM = 1279
                            v.channel_override = { str(channel) : PWM }
                            v.flush()
                            print command2
```

```
                        #Z7
                        if (centroidx> 304 and centroidy>= 0)  and (centroidx<= 432
and centroidy< 70):
                                out2 = 'object detected in Z7\n'
                                sys.stdout.write('\r'+out2)
                                sys.stdout.flush()

                                if entrada2 == 0:

                                        control4 = 1
                                        channel  = 2
                                        PWM = 1279
                                        v.channel_override = { str(channel) : PWM }
                                        v.flush()
                                        print command2

                                elif entrada2 == 1 and delay(0.05) == 1:

                                        control4 = 1
                                        channel  = 2
                                        entrada2 = 0
                                        PWM = 1279
                                        v.channel_override = { str(channel) : PWM }
                                        v.flush()
                                        print command2
                        #Z8
                        if (centroidx> 128 and centroidy>= 0)  and (centroidx<  304
and centroidy< 70):
                                out2 = 'object detected in Z8\n'
                                sys.stdout.write('\r'+out2)
                                sys.stdout.flush()

                                if entrada2 == 0:

                                        control4 = 1
                                        channel  = 2
                                        PWM = 1279
                                        v.channel_override = { str(channel) : PWM }
                                        v.flush()
                                        print command2

                                elif entrada2 == 1 and delay(0.05) == 1:

                                        control4 = 1
```

```
                                        channel  = 2
                                        entrada2 = 0
                                        PWM = 1279
                                        v.channel_override = { str(channel) : PWM }
                                        v.flush()
                                        print command2
                        #Z0
                if (centroidx>= 128 and centroidy>= 70)  and (centroidx<=  304  and centroidy<= 170):

                                        out2 = 'object detected in Z0\n'
                                        sys.stdout.write('\r'+out2)
                                        sys.stdout.flush()

                                        channel = 3
                                        control4 = 1
                                        #throttle -= 105

                        else:
                                if channel <> 3:
                                        if delay(0.05) == 1:
                                                entrada2 = 1
                                                control4 = 0
                                                PWM = 0
                                                v.channel_override = { str(channel) : PWM }
                                                v.flush()
                                                print command2

                elif (contour_area>= threshold2) and control == 3:
                        cv2.imwrite(   'log' + str(i) + '.jpg', fram)
                        location.write('log' + str(i) + ' ' + str(v.location))
                        velocity.write('log' + str(i) + ' ' + str(v.attitude))
                        IMU.write(     'log' + str(i) + ' ' + str(v.velocity))
                        i += 1
                        break


        cv2.rectangle(fram, (128,70), (304,170), (0,0,255), 1)

        #cv2.imshow("ImgThresh", imgThresh)
cv2.imshow("frame", fram)
    #cv2.imshow("hsv", imgHSV)

    #counter += 1
```

```
        #currtime = time.time()
        #numsecs = currtime - start_time
        #fsp  = counter / numsecs

        #print "FPS Promedio:",fsp

            if delay(3) == 1:
                    cv2.imwrite(   'log' + str(i) + '.jpg', fram)
                    location.write('log' + str(i) + ' ' + str(v.location))
                    IMU.write(     'log' + str(i) + ' ' + str(v.attitude))
                    velocity.write( 'log' + str(i) + ' ' + str(v.velocity))
                    i += 1

            k = cv2.waitKey(10)
            if k == 27:
                    v.channel_override = { "1" : 0, "2" : 0, "3" : 0, "4" : 0, "5" : 0, }
                    v.flush()
                    v.mode = VehicleMode("STABILIZE")
                    v.flush()
            break


whilev.mode == "LAND":

            ifv.armed == False:
                    print 'the Quadrotor has landed'
                    break

            elifv.armed == True:
                    out3 = 'Quadrotor is landing'
                    sys.stdout.write('\r'+out3)
                    sys.stdout.flush()

print 'Finish :)'

ifv.armed == True:
            v.armed = False
            v.flush()

v.channel_override = { "1" : 0, "2" : 0, "3" : 0, "4" : 0, "5" : 0 }
v.flush()

location.close()
velocity.close()
```

```
IMU.close()

cv2.destroyAllWindows()
capture.release()
```