UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERÍA MECÁNICA

DESARROLLO DE TÉCNICAS PROBABILÍSTICAS DE LOCALIZACIÓN PARA LA NAVEGACIÓN AUTÓNOMA DE UN VEHÍCULO AÉREO MULTIRROTOR

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE:
INGENIERO MECATRÓNICO

EDWIN GUERRERO JARA

PROMOCIÓN 2009 - I

LIMA - PERÚ

2015

Índice General

PRÓLOGO 1
CAPÍTULO I INTRODUCCIÓN 4
1.1 Motivación 4
1.2 Antecedentes 7
1.2.1 Avances en la última década 7
1.3 Principio dinámico del cuadricóptero 10
1.4 Cuadricópteros actuales 12
1.4.1 Cuadricópteros comerciales más conocidos 13
1.4.2 Aplicaciones de los cuadricópteros 16
1.5 Navegación autónoma 20
1.6 Objetivos 24
1.6.1 Objetivo general 24
1.6.2 Objetivos específicos 24
1.7 Alcances y limitaciones 25
1.8 Estándares de calidad y regulaciones de operación 26
CAPÍTULO II MARCO TEÓRICO 29
2.1 Representación geométrica 29
2.1.1 Coordenadas homogéneas 30
2.1.2 Transformaciones en coordenadas homogéneas 31
2.1.3 Coordenadas globales y locales 33
2.1.4 Ángulos de Euler 35
2.2 Seguimiento de Trayectoria 37
2.2.1 Control punto a punto 38
2.2.2 Control de trayectoria continua 40

2.3 Estimación de estado 41
2.3.1 Modelo de estado 41
2.3.2 Observadores de estado 42
2.3.3 Robótica probabilística 44
2.3.4 Teoría de probabilidades 46
2.3.5 Filtro de Bayes 50
2.3.6 Filtro de Kalman 53
2.3.7 Filtro de Kalman extendido (EKF) 58
CAPÍTULO III SIMULADOR DE VUELO 61
3.1 Simulador en Gazebo - ROS 61
3.1.1 Principios del simulador 65
3.1.2 Estructura del paquete de simulación *tum_simulator* 67
3.2 Simulador JavaScript 71
3.2.1 Principios del simulador 72
CAPÍTULO IV SOLUCIÓN PROPUESTA 75
4.1 Modelo odométrico 75
4.2 Visión monocular 78
4.3 Comandos de vuelo 81
4.4 Control de trayectoria 82
4.4.1 Controlador PID 83
4.4.2 Programación con el simulador de JavaScript 85
4.4.3 Programación con el simulador de Gazebo (*tum_simulator*) 90
4.5 Localización del cuadricóptero 94
4.5.1 Inicialización 95
4.5.2 Programación 96
CAPÍTULO V RESULTADOS DE LA PROPUESTA 100
5.1 Resultados de simulación del control de trayectoria 100
5.2 Implementación del control de trayectoria 101
5.3 Resultados de la simulación de localización 104
5.4 Implementación del algoritmo de localización 107
CONCLUSIONES 109
RECOMENDACIONES Y TRABAJOS FUTUROS 111
BIBLIOGRAFÍA 112

APÉNDICE A CARACTERÍSTICAS Y ESPECIFICACIONES TÉCNICAS DEL PARROT AR.DRONE 2.0 114
APÉNDICE B PLANOS DEL PARROT AR.DRONE 2.0 115

Índice de Figuras

Figura 1.1: Diagrama de fuerzas de empuje de cada rotor de un cuadricóptero (thrust). Se puede observar los tres grados de libertad (DOF) descritos anteriormente en los ángulos de Euler: pitch, roll y yaw. 10
Figura 1.2: Principio de vuelo de un cuadricóptero. 11
Figura 1.3: Sistema de coordenadas *xyz* del cuadricóptero. 12
Figura 1.4: Parrot AR.Drone 1.0. 14
Figura 1.5: Parrot AR.Drone 2.0. 15
Figura 1.6: Phantom Vision 2.0. 15
Figura 1.7: Otras plataformas comerciales. 16
Figura 1.8: Vehículo aéreo multirrotor explorando interior de construcción colapsada. 17
Figura 1.9: Cuadricóptero usado para riego de zonas de cultivo 18
Figura 1.10: Vehículos aéreos multirrotores utilizados para transporte de mercancía, se observa los usados por Amazon y Domino's Pizza. 18
Figura 1.11: Filmación aérea realizada por el cuadricóptero comercial Phantom. 19
Figura 1.12: Aprendiendo a seguir una trayectoria. The Flying Machine Arena, ETH Zürich: http://www.youtube.com/watch?v=goVuP5TJIUU. Angela P. Schoellig, Fabian L. Muller, and Raffaello D'Andrea, "Optimization-Based Iterative Learning for Precise Quadrocopter Trajectory Tracking", Autonomous Robots Volume 33, Number 1-2, pp.103–127, 2012 20
Figura 1.13: Maniobras agresivas precisas. GRASP Lab, University of Pennsylvania, http://youtu.be/YQIMGV5vtd4. N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar. The GRASP multiple micro UAV testbed. IEEE Robotics and Automation Magazine, Vol. 17, No. 3. 2010. Daniel Mellinger, Nathan Michael, and Vijay Kumar. Trajectory Generation and Control for Precise Aggressive Maneuvers with Cuadricópteros. International Journal of Robotics Research Apr. 2012 21
Figura 1.14: Cuadricóptero malabarista. The Flying Machine Arena, ETH Zürich: http://www.youtube.com/watch?v=3CR5y8qZf0Y. Mark Muller, Sergei Lupashin, and Raffaello D'Andrea, "Quadrocopter Ball Juggling", IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.5113–5120, 2011. 21
Figura 1.15: Construcción en equipo. GRASP Lab, University of Pennsylvania, http://youtu.be/W18Z3UnnS_0. Quentin Lindsey, Daniel Mellinger and Vijay

Kumar 'Construction with cuadricóptero teams," Autonomous Robots, 33, (3), 2012.....22
Figura 1.16: Interacción con un cuadricóptero via Kinect. The Flying Machine Arena ETH Zürich: http://www.youtube.com/watch?v=A52FqfOi0Ek.....22
Figura 1.17: (a) Ejemplo de la estimación de la posición del robot y mapa del entorno mediante filtro de Kalman. (b) Robot acuatico Oberon desarrollado en la Universidad de Sydney. (Imágenes por Stefan Williams and Hugh Durrant-Whyte, Australian Centre for Field Robotics).....24
Figura 2.1: Sistema de coordenadas cartesianas espaciales. 29
Figura 2.2: Curva polinomial definida en coordenadas homogéneas (azul) y su proyección en el plano (w = 1). 31
Figura 2.3: Ejemplo de sistemas de coordenadas globales y locales de un vehículo. 34
Figura 2.4: Angulos de Euler roll, pitch y yaw en un aeroplano.....35
Figura 2.5: Rotaciones de Euler de acuerdo a la convención ZYX. 36
Figura 2.6: Aproximación polinómica fragmentaria. 40
Figura 2.7: Curvas parametrizadas en el tiempo. 40
Figura 2.8: Transformación de una representación única de la posición a una representación probabilística. 46
Figura 2.9: Representación gráfica de las variables aleatorias A y B.....47
Figura 2.10: Distribución Normal.....54
Figura 3.1: Simulador de un PR2 en Gazebo. 63
Figura 3.2: Simulador de un cuadricóptero en Gazebo.....64
Figura 3.3: Diagrama mostrando los nodos (encerrados en elipses) y tópicos de la ejecución de los algoritmos en el simulador *tum_simulator*. 67
Figura 3.4: Simulador del Parrot AR.Drone en Gazebo. 68
Figura 3.5: Diagrama de bloques de la operación del cuadricóptero mediante joystick a través del driver en ROS.....69
Figura 3.6: Diagrama de bloques de la operación del simulador en Gazebo mediante joystick. 69
Figura 3.7: Impresión en pantalla de los datos de navegación de simulador *tum_simulator* obtenidos mediante el tópico **/ardrone/navdata**. 70
Figura 3.8: Simulador basado en JavaScript. En la parte superior se simula el vuelo del Parrot AR.Drone, en la parte central se grafican las velocidades lineales y en la parte inferior se muestra parte del algoritmo e impresiones en pantalla de variables. 72
Figura 4.1: Ejes de coordenadas del cuadricóptero, donde se ilustra las velocidades de avance y el angulo de giro ψ. 77
Figura 4.2: Sistema de coordenadas global: Las velocidades de avance que brinda el cuadricóptero están en un sistema de coordenadas relativo al mismo. 77
Figura 4.3: Transformaciones entre sistema de coordenadas. 79
Figura 4.4: Control de lazo cerrado. 84
Figura 4.5: Control PD. Conforme Kd aumenta se consigue una mejor amortiguación de la respuesta de control. 84
Figura 4.6: Control de trayectoria (pruebas en simulador de Javascript). 86

Figura 4.7: Respuesta del controlador de trayectoria. 87
Figura 4.8: Control de trayectoria con respuesta sub-amortiguada. 89
Figura 4.9: Respuesta sub-amortiguada del controlador.....89
Figura 4.10: Respuesta con oscilaciones.....90
Figura 4.11: Simulación del controlador PD utilizando el *tum_simulator*. 91
Figura 4.12: Diagrama de flujos representando el contenido del nodo /Code. 93
Figura 4.13: Diagrama de bloques para el control de posición del cuadricóptero mediante estimación de estado aplicando filtro de Kalman. 94
Figura 5.1: Medición del punto de despegue al punto de aterrizaje para ser comparado con el cálculo de la odometría. 102
Figura 5.2: Implementación del controlador PD en el Parrot AR.Drone 2.0.....104
Figura 5.3: Trayectoria seguida por el cuadricóptero empleando filtro de Kalman. 105
Figura 5.4: Seguimiento de trayectoria sin filtro de Kalman, se observa un desvío de la trayectoria deseada debido al alto ruido del modelo odométrico, representado por la elipse roja como una muy alta covarianza.....106
Figura 5.5: Imagen prediseñada que es detectada por defecto por la unidad de procesamiento del Parrot AR.Drone. 107
Figura 5.6: Implementación del EKF en el Parrot AR.Drone mediante detección de *landmarks*. 108

Índice de Tablas

Tabla 1.1: Configuraciones comunes para MAVs. 8
Tabla 1.2: Comparación de MFRs, (1: malo 4: muy bueno) 9
Tabla 2.1: Transformaciones homogéneas en dos dimensiones P2. 33
Tabla 2.2: Clasificación de sensores usados robótica móvil. A: activo P: pasivo, PC: propioceptivo, EC: exteroceptivo. 44
Tabla 3.1: Información de navegación de interés del Parrot AR.Drone 2.0. 71
Tabla 4.1: Entradas de control del cuadricóptero AR Drone Parrot 2.0. 82
Tabla 5.1: Mediciones de odometría. 103

PRÓLOGO

En los últimos años se han venido desarrollando algoritmos para un mejor control y autonomía en el vuelo de vehículos aéreos no tripulados. El amplio rango de posibles aplicaciones que se le pueden dar a estos vehículos, han hecho que muchas universidades e instituciones de investigación y desarrollo de prestigio a nivel mundial se enfoquen en la elaboración de mejores técnicas de control dinámico y navegación autónoma para estos vehículos, siendo los de diseño de cuatro hélices distribuidas en forma de cruz llamados cuadricópteros los más usados.

Los contenidos teóricos principales utilizados en el presente trabajo son la robótica probabilística, la ingeniería de control clásica y la representación geométrica tridimensional.

Para la prueba en simulación y solo con fines de visualización de los algoritmos experimentales elaborados, se ha optado por el uso de un simulador basado en JavaScript. Asimismo, para el seguimiento de trayectoria se ha usado un simulador en Gazebo para su posterior implementación. Ambos simuladores han sido desarrollados por el Grupo de Visión Artificial de la Universidad Técnica de Múnich. Mayor información se brinda en el capítulo respectivo.

Este trabajo está distribuido de tal modo que facilite el entendimiento de las bases teóricas y su aplicación en la navegación del cuadricóptero. De igual manera se espera que sirva como base para futuros investigadores que deseen continuar con las técnicas aquí expuestas u otras más avanzadas en simulación o en pruebas físicas.

La presente tesis está dividida en seis capítulos resumidos a continuación:

Capítulo I: capítulo de introducción que contiene la motivación de la tesis, breve reseña del desarrollo de vehículos aéreos no tripulados en la última década, así como los avances en la navegación autónoma para estos vehículos. Finalmente se determinan los objetivos, alcances y limitaciones del presente trabajo.

Capítulo II: se presentan las teorías necesarias para el alcance de nuestro objetivo, las cuales son la representación geométrica del estado de un cuerpo en el espacio, la teoría para el seguimiento de trayectoria de un vehículo, y la estimación de estado mediante algoritmos probabilísticos de localización.

Capítulo III: se presentan dos plataformas de simulación de vuelo del cuadricóptero. Una de ellas es Gazebo -una de las más robustas en la actualidad- con la que simulamos el vuelo del cuadricóptero comercial Parrot AR.Drone y su información de navegación, los algoritmos desarrollados en esta plataforma pueden ser trasladados directamente al cuadricóptero comercial mediante conexión WiFi. El otro entorno de simulación es un paquete JavaScript, con el que se probarán los algoritmos probabilísticos de navegación autónoma con fines de visualización.

Capítulo IV: en este capítulo se desarrollan los modelos matemáticos de estado del vehículo aéreo, el control de trayectoria deseada para el vuelo del cuadricóptero a

través de un controlador PID, el que es usado como base en el desarrollo del algoritmo de estimación de estado, en donde se presentan los algoritmos expuestos en el Capítulo II aplicados a los modelos de estado del cuadricóptero. El controlador de trayectoria desarrollado en este capítulo será trasladado al Parrot AR.Drone 2.0 detallando los resultados obtenidos.

Capítulo V: se muestran los resultados de la solución propuesta en el capítulo anterior, con la que se habrá probado cierto grado de autonomía en la navegación del cuadricóptero para el seguimiento de una trayectoria deseada.

La presente tesis fue elaborada a comienzos de un proyecto de investigación para la navegación autónoma del cuadricóptero financiada por el Instituto General de Investigación - IGI de la Universidad Nacional de Ingeniería. A la fecha de la conclusión de la tesis, el estado del proyecto se encuentra en sus fases iniciales, el cual tiene como objetivo el desarrollo y la implementación de algoritmos probabilísticos de localización para lograr la autonomía de vuelo del cuadricóptero comercial Parrot AR.Drone 2.0. Los algoritmos aquí desarrollados son la base principal de algoritmos más avanzados como el EFK SLAM, que se trata de la estimación de la posición y el mapeo simultáneo del mismo haciendo uso del filtro de Kalman extendido.

CAPÍTULO I
INTRODUCCIÓN

1.1 Motivación

En los últimos años, los vehículos aéreos pequeños (MAVs por sus siglas en inglés: Miniature Aerial Vehicles), se han convertido en una herramienta importante no solo en el dominio militar, sino también en ambientes urbanos. Particularmente, los cuadricópteros han empezado a ser muy populares, especialmente para propósitos de observación y exploración en entornos abiertos y ambientes cerrados, debido a su alta maniobrabilidad, así como también para la recopilación de información en lugares de difícil acceso como la toma de fotografías o grabaciones de video, manipulación de objetos o simplemente como juguetes de alta tecnología [4].

Existen numerosos ejemplos de aplicaciones prácticas que se le han dado a los MAVs, como por ejemplo en labores de exploración para la inspección de los reactores nucleares dañados en Fukushima desde Marzo del año 2011 en Japón y para observaciones aéreas y el monitoreo de situaciones como eventos deportivos o protestas en las calles [11].

Sin embargo, existen aún múltiples aplicaciones potenciales de estos vehículos aéreos, como por ejemplo la exploración rápida en los interiores de construcciones colapsadas por terremotos u otro tipo de desastres naturales o terroristas, sin arriesgar la vida de ningún rescatista, para la búsqueda de sobrevivientes. Equipados con cámaras de alta resolución, pueden ser usados también para la toma de fotografías aéreas y para la toma de videos en diversos eventos deportivos tal como ya se ha puesto de moda.

El cuadricóptero al tener un comportamiento de vuelo similar al de un helicóptero, tiene la capacidad de aterrizar y despegar verticalmente y de mantenerse inmóvil en pleno vuelo sobre cierto punto de interés, y de moverse en cualquier dirección en cualquier momento, sin antes tener que rotar. Estas características del cuadricóptero, a diferencia de los aeroplanos tradicionales, les dan la capacidad de maniobrar en espacios altamente reducidos, como pasadizos y oficinas, lo que los hace ideales para exploración en interiores o de alta densidad de obstáculos.

Para la navegación, los más modernos MAVs cuentan con una amplia gama de sensores. La unidad de medición inercial (IMU por sus siglas en inglés: Inertial Measurement Unit), mide la aceleración y orientación del vehículo aéreo. Algunos cuentan con sistema de posicionamiento global GPS (Global Positioning System), que le da la capacidad de conocer su posición absoluta, permitiéndoles navegar autónomamente en ambientes abiertos y despejados, o simplemente mantenerse en cierta ubicación deseada sin desviarse. Aun siendo controlados remotamente por un piloto experimentado, estos sistemas de navegación sirven de gran ayuda, reduciendo significativamente la carga de trabajo del piloto.

Sin embargo, al navegar en entornos cerrados o desconocidos, donde la señal GPS no está disponible o es inviable, se requiere de otras alternativas de localización. Existen también una amplia variedad de sensores usados para esta tarea, como por ejemplo los sensores económicos de ultrasonido que miden distancias a objetos cercanos en una dirección en particular o los sensores láser de alta resolución que brindan una imagen del entorno, pero siendo estos muy costosos. Una de las maneras más directas de recoger información del entorno es mediante las cámaras digitales, las que pueden entregar mucha información y son relativamente económicas de bajo consumo de energía, pequeñas y ligeras y por ende, particularmente perfectas para los MAVs. Por otro lado, el procesamiento de esta información es una tarea muy compleja, que requiere una alta capacidad computacional.

La principal característica de un sistema capaz de navegar autónomamente, es la capacidad de localizarse por sí mismo en el entorno que lo rodea. Particularmente esta capacidad es muy complicada en el caso de vehículos aéreos, a diferencia de los vehículos móviles en tierra los cuales pueden permanecer inmóviles sin problemas, para los vehículos voladores el mantener una posición en el aire requiere contrarrestar perturbaciones que inducen su movimiento, lo que a su vez requiere un método para detectar estos movimientos inducidos.

La tarea de localización de un robot ha sido el centro de muchas investigaciones en el campo de la visión computacional y la robótica probabilística, la cual es ampliamente conocida como el problema de la localización y mapeo simultaneo (SLAM por sus siglas en inglés: Simultaneous Localization and Mapping). La idea general de esta

técnica es que usando la información de los sensores del robot se genera un mapa del entorno. Este mapa es usado para volver a estimar la posición del robot [14].

Una vez estimada la posición del MAV, esta puede ser usada para ir y llegar a una posición deseada o seguir una trayectoria. Además, al ubicarse autónomamente en su entorno, reduce significativamente la carga de trabajo de un piloto, haciendo el control manual del vehículo aéreo mucho más fácil, al compensar automáticamente los desvíos que pueda tener debido a las perturbaciones. Un ejemplo de ello es una aplicación desarrollada en la Universidad Técnica de Múnich que le permiten a un cuadricóptero navegar explorar y localizar objetos de interés de forma autónoma, en ambientes cerrados y desconocidos [1].

Existe entonces un área potencial de desarrollo para mejorar la autonomía de vuelo de estos vehículos aéreos en la que la cámara digital a bordo sea una importante fuente de información para el logro de una navegación autónoma utilizando los alcances en visión computacional y robótica probabilística.

1.2 Antecedentes

1.2.1 Avances en la última década

Los avances en la tecnología de procesadores, la miniaturización de los sensores y la teoría de control han contribuido al desarrollo de MAVs.

Los robots voladores pequeños (MFRs por sus siglas en inglés: *Miniature Flying Robots*) ofrecen mejores ventajas para la navegación en interiores de edificaciones o ambientes congestionados, debido a su tamaño y modo de vuelo. Existen muchas

aplicaciones de este tipo de robots, como el rescate en desastres naturales, inspección de fallas en construcciones, etc.

Diversos modelos dinámicos son usados para los MAVs, a continuación mostramos en la Tabla 1.1 estos modelos señalando sus ventajas y desventajas [15].

Asimismo, en la Tabla 1.2 se presenta una breve comparación entre diversos modelos de robots voladores pequeños (MFRs), los cuales tienen la capacidad de despegue y aterrizaje en vertical, y por ende pueden también mantenerse inmóvil en pleno vuelo [15].

Tabla 1.1: Configuraciones comunes para MAVs.

Configuration	Picture	Advantages	Drawbacks
Fixed-wing (AeroVironment)		- Simple mechanics - Silent operation	- No hovering
Single (A.V de Rostyne)		- Good controllability and maneuverability	- Complex mechanics - Large rotor - Long tail boom
Axial rotor (Maryland Univ.)		- Compactness - Simple mechanics	- Complex control - Weak maneuverability
Coaxial rotors (ETHZ)		- Compactness - Simple mechanics	- Complex aerodynamics
Tandem rotors (Heudiasyc)		- Good controllability and maneuverability - No aerodynamics Interference	- Complex mechanics - Large size
Quadrotors (ETHZ)		- Good maneuverability - Simple mechanics - Increased payload	- High energy consumption - Large size

Tabla 1.1: Configuraciones comunes para MAVs (continuación).

Configuration	Picture	Advantages	Drawbacks
Blimp (EPFL)		- Low power consumption - Auto-lift	- Large size - Weak maneuverability
Hybrid (MIT)		- Good maneuverability - Good survivability	- Large size - Complex design
Bird-like (Caltech)		- Good maneuverability - Low power Consumption	- Complex mechanics - Complex control
Insect-like (UC Berkeley)		- Good maneuverability - Compactness	- Complex mechanics - Complex control
Fish-like (US Naval Lab)		- Multimode mobility - Efficient Aerodynamics	- Complex control - Weak maneuverability

Tabla 1.2: Comparación de MFRs, (1: malo, 4: muy bueno).

	A	B	C	D	E	F	G	H
Power cost	2	2	2	2	1	4	3	3
Control cost	1	1	4	2	3	3	2	1
Payload/volume	2	2	4	3	3	1	2	1
Maneuverability	4	2	2	3	3	1	3	3
Mechanics simplicity	1	3	3	1	4	4	1	1
Aerodynamics complexity	1	1	1	1	4	3	1	1
Low speed flight	4	3	4	3	4	4	2	2
High speed flight	2	4	1	2	3	1	3	3
Miniaturization	2	3	4	2	3	1	2	4
Survivability	1	3	3	1	1	3	2	3
Stationary flight	4	4	4	4	4	3	1	2
Total	**24**	**28**	**32**	**24**	**33**	**28**	**22**	**24**

A=Single rotor, B=Axial rotor, C=Coaxial rotors, D=Tandem rotors, E=Quadrotor, F=Blimp, G=Bird-like, H=Insect-like.

Según se observa, el modelo cuadricóptero y rotor coaxial presentan los mejores puntajes promedios de las características presentadas en la Tabla 1.2. Para la presente

tesis, se va a usar el modelo cuadricóptero, quien obtuvo el mayor puntaje en la tabla, y es ampliamente usado en fines de investigación y fines comerciales.

1.3 Principio dinámico del cuadricóptero

El diseño mecánico de un cuadricóptero consta de cuatro hélices en una configuración en cruz. Manteniendo el giro de un par de hélices en sentido opuesto al otro par de hélices, elimina la necesidad de un rotor de cola a diferencia de un helicóptero (ver Figura 1.1).

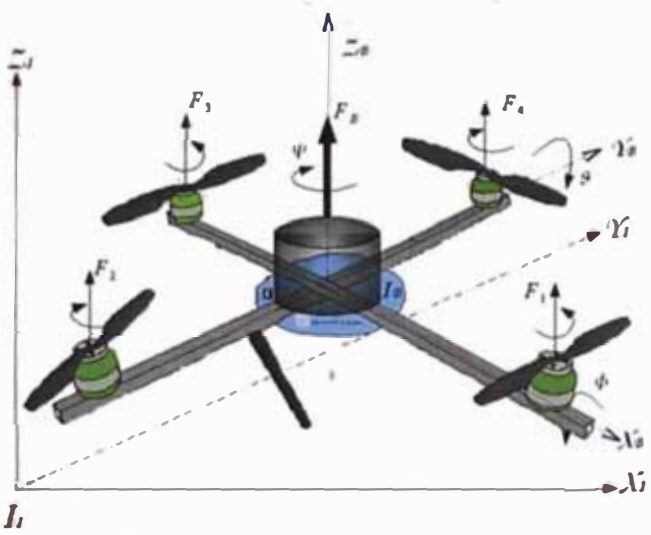


Figura 1.1: Diagrama de fuerzas de empuje de cada rotor de un cuadricóptero (thrust). Se puede observar los tres grados de libertad (DOF) descritos anteriormente en los ángulos de Euler: pitch, roll y yaw.

En consecuencia, la rotación vertical (alrededor del eje z en la figura) se consigue mediante la creación de una diferencia en las velocidades angulares entre los dos pares de rotores (ver Figura 1.2a y Figura 1.2b).

Aumentando o disminuyendo simultáneamente las velocidades de las cuatro hélices permite el ascenso y descenso del cuadricóptero (ver Figura 1.2c). Los movimientos

horizontales del cuadricóptero, sean de avance y retroceso, o marcha de lado, se logra mediante una inclinación sobre sus ejes lateral y longitudinal *x* y *y*, esto es posible por el con el cambio coordinado de las velocidades de las hélices de un par opuesto de rotores (ver Figura 1.2d).

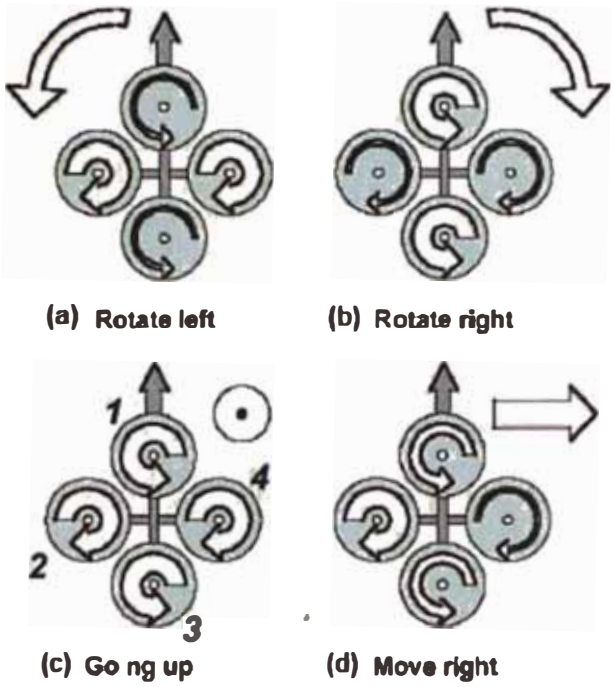


Figura 1.2: Principio de vuelo de un cuadricóptero.

A pesar de los cuatro rotores como actuadores, el cuadricóptero es un sistema subactuado y dinámicamente inestable. Inclusive, los cuadricópteros de la clase MAV requieren un diámetro muy pequeño para el rotor que lo penaliza en términos de eficiencia. Sin embargo, como se indicó en la Tabla 1.2, la mecánica simple de los cuadricópteros y el la capacidad de la carga útil son su principal ventaja.

La dinámica de un cuerpo rígido sujeto a fuerzas externas aplicados a su centro de masa, expresados en el sistema inercial del cuerpo, según las ecuaciones de Newton-Euler [6], son:

$$\begin{bmatrix} mI_{3\times3} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \dot{V} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} \omega \times mV \\ \omega \times I\omega \end{bmatrix} = \begin{bmatrix} F \\ \tau \end{bmatrix} \tag{1.1}$$

Donde $I \in R^{(3x3)}$ es la matriz inercial, V es la velocidad lineal del cuerpo y ω es la velocidad angular del cuerpo. F y τ son la fuerza y el torque respectivamente aplicados al cuerpo, mientras m es la masa.

Considerando el sistema global E y el sistema inercial B del cuadricóptero mostrados en la Figura 1.3. Usando los ángulos de Euler, la orientación del cuadricóptero en el espacio está dada por la matriz de rotación R, que expresa la transformación del sistema B al sistema E, presentada en la ecuación (2.16).

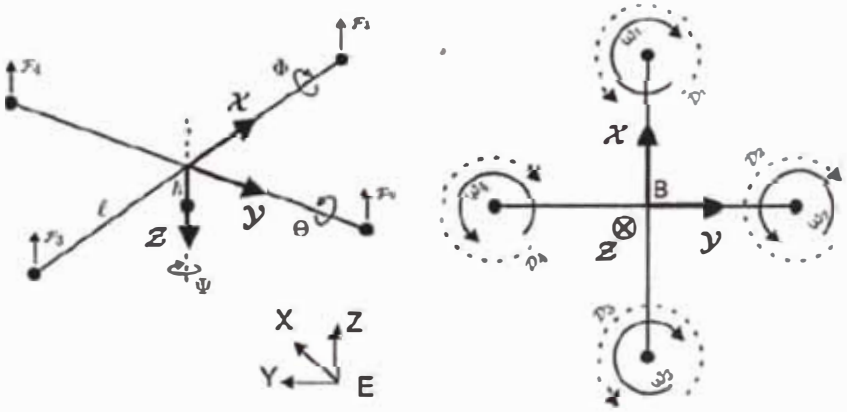


Figura 1.3: Sistema de coordenadas *xyz* del cuadricóptero.

1.4 Cuadricópteros actuales

Como fue mencionado anteriormente, debido a los progresos en la tecnología en la última década, los MAVs, empezaron a ser más populares para muchas

aplicaciones. La necesidad de contar con vehículos voladores con gran maniobralidad y habilidad de mantenerse inmóvil en vuelo ha conducido al incremento en la investigación de los cuadricópteros. Su diseño de cuatro rotores permite a los cuadricópteros ser relativamente simple en diseño, fiables y fáciles de pilotear. Las últimas investigaciones continúan incrementando la viabilidad del uso de los cuadricópteros, generando avances en las comunicaciones entre los robots, exploración del entorno y la maniobrabilidad. Si todas estas cualidades en desarrollo se pudieran juntar, los cuadricópteros serían capaces de misiones autónomas avanzadas que en la actualidad no son posibles con ningún otro vehículo aéreo.

Actualmente en el mercado existen muchas plataformas comerciales para el uso personal, a continuación se hará una presentación de algunas de ellas.

1.4.1 Cuadricópteros comerciales más conocidos

- **Parrot AR.Drone**

Es un cuadricóptero controlado por radiofrecuencia elaborado por la compañía francesa Parrot. Está diseñado para ser controlado por smartphones y tablets con sistemas operativos iOS o Android. Existen actualmente dos versiones en el mercado:

La versión 1.0 (ver Figura 1.4) fue lanzada en el año 2010, su plataforma está construida de nylon y fibra de carbono, midiendo 57 cm de largo. Se suministran dos cascos intercambiables con la estructura, uno diseñado para interiores, hecho de espuma, que protege a las hélices, y otro hecho de plástico para ser usada al aire libre, que le permite una mayor maniobrabilidad. En total, el AR.Drone cuenta con seis

grados de libertad, con una unidad de medición inercial IMU miniaturizado para su uso en la estabilización.

Figura 1.4: Parrot AR.Drone 1.0.

Dentro de la estructura del cuadricóptero, una gama de sensores para la asistencia de vuelo, permiten a la interfaz ser usada por los pilotos de manera más simple, y haciendo que su vuelo avanzado sea más fácil. El procesador a bordo ejecuta el sistema operativo Linux, y se comunica con el piloto a través de una conexión Wi-Fi. Los sensores a bordo incluyen un altímetro de ultrasonidos, que se utiliza para proporcionar la estabilización vertical hasta 6 m. Los rotores son alimentados por 15 vatios, con motores sin escobillas alimentados por una batería de polímero de litio de 11.1 voltios. Esto proporciona aproximadamente 12 minutos de tiempo de vuelo a una velocidad de 5 m/s.

La versión 2.0 es la sucesora, y fue lanzada en el año 2012 en Las Vegas (ver Figura 1.5). En lugar de rediseñar el producto, se hicieron mejoras a su funcionalidad. El hardware del AR.Drone 2.0 se ha actualizado de manera significativa para mejorar las funciones del cuadricóptero. La calidad de la cámara se incrementó, y muchos de los

sensores a bordo se hicieron más sensibles, lo que permite un mayor control. El altímetro de ultrasonido se mejoró con la adición de un sensor de presión de aire, lo que permite un vuelo más estable y el Wi-Fi se ha actualizado a un nuevo estándar. Otras mejoras de los sensores son la actualización del giroscopio a una versión de 3 ejes, junto con un acelerómetro de 3 ejes y magnetómetro [APÉNDICE A].

Figura 1.5: Parrot AR.Drone 2.0.

- **Phantom**

Es una serie de plataformas comerciales desarrolladas por DJI Innovations, su aplicación está más enfocada a la toma de imágenes y videos aéreos, debido a su cámara integrada de alta resolución (ver Figura 1.6).

Figura 1.6: Phantom Vision 2.0.

- **Otras plataformas**

Existen muchas otras plataformas comerciales (ver Figura 1.7), así como también existen opciones de hardware y software open-sources, basadas en Arduino, para la construcción y desarrollo de cuadricópteros, como por ejemplo el ArduCopter.

Figura 1.7: Otras plataformas comerciales.

1.4.2 Aplicaciones de los cuadricópteros

Muchas han sido las aplicaciones comerciales que se están dando a los vehículos multirrotores en la actualidad. A continuación se darán ejemplos de las más importantes.

- **Misiones de búsqueda y rescate**

Estos vehículos aéreos pueden acceder a zonas de alto riesgo para el hombre, como ambientes con alta radiación o zonas de difícil acceso por tierra, sin exponer en riesgo la integridad del rescatista. De esta manera, pueden observar daños materiales y encontrar personas aisladas y atrapadas en las zonas de peligro, para posteriormente poder ser rescatadas.

- **Inspecciones de infraestructura**

La capacidad de mantenerse en vuelo sobre un punto fijo, su pequeñez y maniobrabilidad, hacen de estos vehículos aéreos idóneos para realizar inspecciones en interiores y exteriores, como puentes, edificios, presas e infraestructura en general. Con la cámara digital a bordo, pueden grabar y fotografiar daños en las infraestructuras de manera rápida y sencilla, tele-operados desde tierra (ver Figura 1.8).

Figura 1.8: Vehículo aéreo multirrotor explorando interior de construcción colapsada.

- **Inspección y riego de campos de agricultura**

Otro uso de estos vehículos aéreos es la inspección y el riego de campos de cultivo, eliminando la necesidad de implementar un sistema de riego y/o fumigación como el mostrado en la Figura 1.9.

- **Mapeo aéreo**

Otra de las aplicaciones de los vehículos aéreos multirrotores es la construcción de mapas topológicos de una zona determinada de manera precisa, debido a su capacidad de sobrevolar la zona de manera constante.

Figura 1.9: Cuadricóptero usado para riego de zonas de cultivo.

- **Transporte de objetos**

Recientemente muchas compañías de distribución de productos están optando por el envío de su mercancía mediante vehículos aéreos no tripulados, como es el caso de Amazon con su programa Amazon Prime Air, que para el año 2015, espera obtener los permisos de la Administración Federal de Aviación de los EEUU para el uso comercial en la entrega de productos al usuario final (ver Figura 1.10).

Figura 1.10: Vehículos aéreos multirrotores utilizados para transporte de mercancía, se observa los usados por Amazon y Domino's Pizza.

- **Seguridad ciudadana**

Recientemente, diversos municipios en la ciudad de Lima, han adquirido estos vehículos aéreos para ser utilizados en el patrullaje de calles, manifestaciones u otros eventos donde se requiera vigilancia. Operados remotamente se puede observar en tiempo real las capturas de imágenes y video de la cámara digital incorporada a estos vehículos. Algunos candidatos a las recientes elecciones municipales plantearon el uso de estos vehículos aéreos como un reforzamiento a la seguridad ciudadana.

- **Fotografía y cinematografía**

Muchos multirrotores comerciales traen una cámara digital incorporada o con una estructura para poder adaptar cámaras filmadoras profesionales, y son usados para grabaciones aéreas para diversas actividades, como por ejemplo, eventos deportivos, documentales, o cualquier otro uso particular (Figura 1.11). Un ejemplo de esta aplicación en nuestra ciudad se dio en la procesión del Señor de los Milagros en Octubre pasado, donde se usaron drones para transmitir en vivo y en directo el evento vía Youtube.

Figura 1.11: Filmación aérea realizada por el cuadricóptero comercial Phantom.

1.5 Navegación autónoma

Los trabajos previos relacionados al vuelo autónomo con cuadricópteros, pueden ser categorizados en dos áreas principales de investigación:

Una de ellas tiene como objetivo el control óptimo y ágil de un cuadricóptero. Muchos resultados han sido publicados al respecto. Estos trabajos sin embargo se guían en sistemas de monitoreo externo, restringiendo su uso a un laboratorio. Las investigaciones más resaltantes en esta área se han venido realizando en el Flying Machine Arena del Instituto de Control de Sistemas Dinámicos de la ETH de Zúrich y en el Laboratorio General de Robótica, Automatización y Percepción - GRASP de la Universidad de Pennsylvania. A continuación se muestra algunas imágenes de estas investigaciones (Figuras del Figura 1.12 al Figura 1.16):

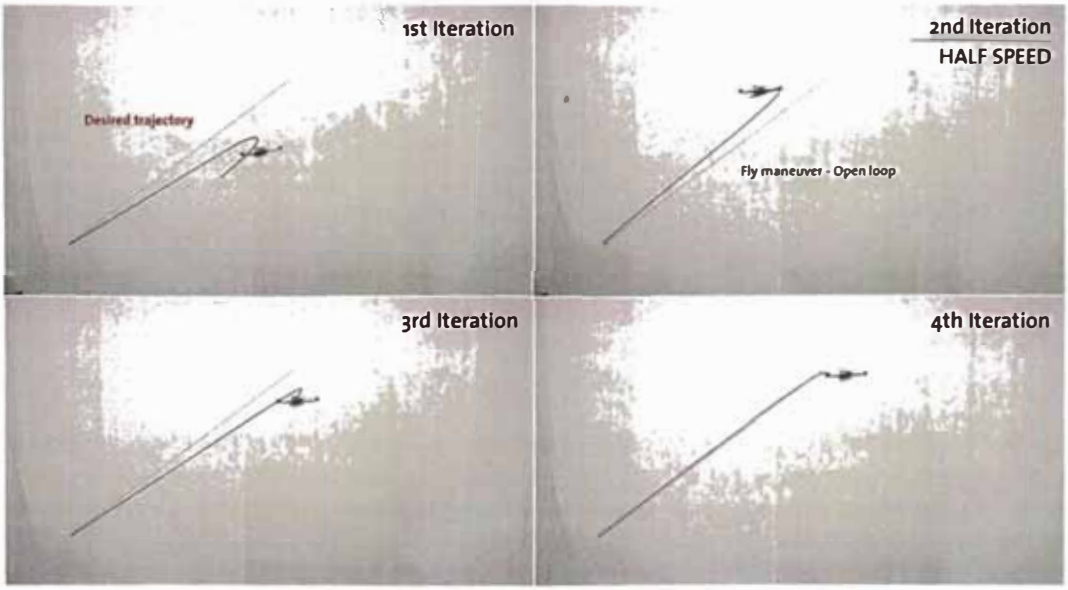


Figura 1.12: Aprendiendo a seguir una trayectoria. The Flying Machine Arena, ETH Zürich: http://www.youtube.com/watch?v=goVuP5TJIUU. Angela P. Schoellig, Fabian L. Muller, and Raffaello D'Andrea, "Optimization-Based Iterative Learning for Precise Quadrocopter Trajectory Tracking", Autonomous Robots, Volume 33, Number 1-2, pp.103–127, 2012.

Figura 1.13: Maniobras agresivas precisas. GRASP Lab, University of Pennsylvania, http://youtu.be/YQIMGV5vtd4. N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar. The GRASP multiple micro UAV testbed. IEEE Robotics and Automation Magazine, Vol. 17, No. 3. 2010. Daniel Mellinger, Nathan Michael, and Vijay Kumar. Trajectory Generation and Control for Precise Aggressive Maneuvers with quadrotors. International Journal of Robotics Research, Apr. 2012.

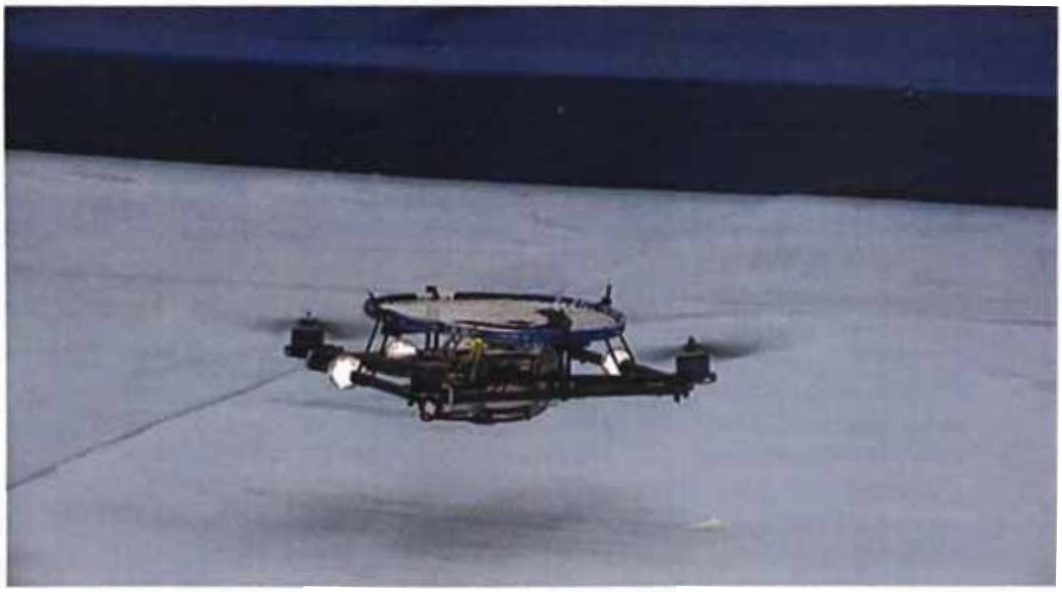


Figura 1.14: Cuadricóptero malabarista. The Flying Machine Arena, ETH Zürich: http://www.youtube.com/watch?v=3CR5y8qZf0Y. Mark Muller, Sergei Lupashin, and Raffaello D'Andrea, "Quadrocopter Ball Juggling", IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.5113–5120, 2011.

Figura 1.15: Construcción en equipo. GRASP Lab, University of Pennsylvania, http://youtu.be/W18Z3UnnS_0. Quentin Lindsey, Daniel Mellinger and Vijay Kumar, "Construction with quadrotors teams," Autonomous Robots, 33, (3), 2012.

Figura 1.16: Interacción con un cuadricóptero via Kinect. The Flying Machine Arena, ETH Zürich: http://www.youtube.com/watch?v=A52FqfOi0Ek

La segunda área de investigación se enfoca en el aprendizaje de un mapa establecido del entorno, el cual es luego reproducido por el cuadricóptero siguiendo la misma

trayectoria. Para estos fines y en ambientes abiertos, existen soluciones comerciales mediante el uso de GPS a bordo. Sin embargo, para la navegación en interiores o en ambientes donde la señal GPS es inviable, se usan los sensores a bordo para la observación del entorno y la estimación de su posición mientras a su vez se va construyendo un mapa del entorno que servirá también para su localización. Esta técnica es ampliamente conocida como el problema de SLAM, ya mencionado anteriormente.

Históricamente, el problema de SLAM ha sido formulado por primera vez en el año 1986, y es considerado ser el principal prerrequisito de los robots verdaderamente autónomos [2] [5]. La primera solución propuesta estuvo basada en el filtro de Kalman extendido (EKF), conocida como EKF-SLAM, donde las posiciones de marcadores llamados *landmarks*, como la posición actual del robot son representados en conjunto en un vector de estado x.

El EKF SLAM ha sido aplicado satisfactoriamente en un rango amplio de problemas de navegación, involucrando vehículos aéreos, submarinos, etc. [9]. La Figura 1.17a muestra un ejemplo del resultado obtenido mediante la implementación del EKF SLAM. Se observa el mapa obtenido bajo el agua por el robot acuático Oberon, desarrollado en la Universidad de Sydney, Australia, como se muestra la Figura 1.17b, el robot está equipado con un sonar que produce mediciones de alta resolución hasta 50 metros de distancia. Para facilitar el problema de mapeo, se colocó objetos verticales en el agua, que pueden ser identificados por el sonar más fácilmente.

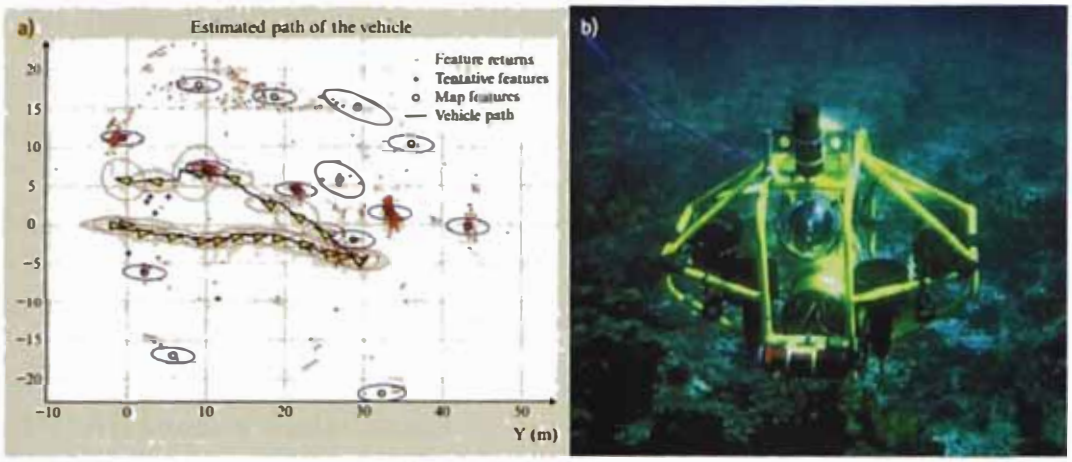


Figura 1.17: (a) Ejemplo de la estimación de la posición del robot y mapa del entorno mediante filtro de Kalman. (b) Robot acuatico Oberon desarrollado en la Universidad de Sydney. (Imágenes por Stefan Williams and Hugh Durrant-Whyte, Australian Centre for Field Robotics).

1.6 Objetivos

1.6.1 Objetivo general

Desarrollar algoritmos de localización para la navegación autónoma del vehículo aéreo multirrotor Parrot Ar.Drone en el seguimiento de una trayectoria definida en el ambiente tridimensional, con la menor intervención humana, utilizando la información de sus sensores a bordo incluyendo la cámara digital.

1.6.2 Objetivos específicos

1. Revisar los fundamentos teóricos para el desarrollo de los algoritmos aplicados al cuadricóptero.
2. Mostrar la simulación de vuelo del Parrot AR.Drone.
3. Desarrollar el algoritmo de estimación de estado (localización) para el cuadricóptero, teniendo la información de la detección de *landmarks* a través de su cámara digital incorporada.

4. Desarrollar el algoritmo de control de posición en un plano horizontal (bidimensional) para el seguimiento de trayectorias e implementarlo en el Parrot AR.Drone.
5. Probar un vuelo autónomo, en el que el cuadricóptero siga una trayectoria definida y contrarreste perturbaciones sin intervención humana.

1.7 Alcances y limitaciones

1. Los algoritmos de estimación de estado requieren de la ubicación predefinida de *landmarks* en tierra, los cuales han de ser detectados por la cámara digital incorporada en el cuadricóptero. La presente tesis no se enfoca ni desarrolla los algoritmos de visión computacional para la detección de dichos *landmarks*, sin embargo se hace uso de una imagen prediseñada la cual es detectaba por la unidad de procesamiento del Parrot AR.Drone, y a partir de ello se desarrolla el modelo matemático de observación de *landmarks* para la corrección de la posición del cuadricóptero en el filtro de Kalman, desarrollado en el capítulo con la solución propuesta.
2. Para la simulación de los algoritmos de localización, se va a usar un simulador programado en JavaScript desarrollado en la Universidad Técnica de Munich (TUM), el cual permite la simulación de la detección de *landmarks* mencionado en el punto anterior. Este simulador solo será usado con fines de visualización. Posteriormente se hará uso de un simulador en Gazebo, el cual nos facilita traspasar los algoritmos directamente al Parrot AR.Drone gracias a que utiliza el driver *ardrone_autonomy* desarrollado en ROS. Todos los algoritmos desarrollados serán programados en lenguaje Python.

3. Para una completa autonomía en la navegación de un vehículo aéreo, no se requeriría del conocimiento previo de *landmarks* en el entorno. Existen diversas técnicas que pertenecen al problema de SLAM, en el cual se realiza una localización y un mapeo simultaneo del entorno. En la presente tesis se desarrollan los algoritmos de localización como una introducción a la navegación autónoma. En el presente capítulo se dio una breve reseña del problema de SLAM basada en el algoritmo de localización a implementar.

1.8 Estándares de calidad y regulaciones de operación

La empresa Parrot fue fundada en 1994 por Henri Seydoux, Parrot crea, desarrolla y vende productos de tecnología de consumo avanzados para Smartphones y tabletas.

Parrot ofrece la gama de sistemas de comunicación manos libres para el coche más amplia del mercado. Su reconocida experiencia a nivel global en el campo de la conectividad móvil y los servicios multimedia asociados al Smartphone, ha posicionado a Parrot como un jugador clave en el mercado del infoentretenimiento en el coche.

Además, Parrot diseña productos multimedia inalámbricos de alta gama dedicados al sonido, y explora nuevas posibilidades con las tecnologías Bluetooth Smart.

Finalmente, Parrot se está expandiendo en el mercado de vehículos aéreos no tripulados (UAV), con el Parrot AR.Drone, el primer cuadricóptero pilotado vía Wi-Fi y también con nuevas soluciones para aplicar el mercado de UAV al uso profesional.

Parrot con sede central en París, cuenta en la actualidad con más de 850 empleados en todo el mundo y genera la mayoría de sus ventas en el exterior.

Los productos Parrot cuentan con los siguientes estándares de calidad:

- **ISO** 9001
- **ISO TS** 16949
- **ISO** 14001
- **OHSAS** 18001

Actualmente en el país existe un proyecto de ley (PL 3872 / 2014-CR) para la regulación del uso de vehículos aéreos no tripulados para toda persona natural o jurídica que opere estos vehículos dentro del espacio aéreo nacional.

Según el proyecto de ley, estas aeronaves de uso civil podrán emplearse para los siguientes trabajos:

a) Monitoreo y evaluación de desastres naturales
b) Patrulla costera y fronteriza
c) Apoyo en búsqueda de rescate
d) Detección y control de incendios
e) Trabajos de investigación y científicos
f) Realización de estudios de impacto ambiental
g) Tratamientos aéreos y fitosanitarios
h) Vigilancia
i) Publicidad

j) Otros que se establezcan en el reglamento

La Dirección General de Aeronáutica Civil del Ministerio de Transportes y Comunicaciones será la encargada de otorgar el permiso a las personas naturales o jurídicas que operen estas aeronaves para realizar cualquier actividad comercial o civil, en merito a las condiciones características y el lugar en donde vayan a operar.

CAPÍTULO II
MARCO TEÓRICO

A continuación se presenta el conjunto de teorías necesarias para el desarrollo de la presente tesis.

2.1 Representación geométrica

Para representar la posición de un cuerpo en el espacio, es necesario definir un sistema de coordenadas en el espacio euclídeo de tres dimensiones. El sistema de coordenadas a usar será el cartesiano representado por los ejes x, y y z como es mostrado en la Figura 2.1.

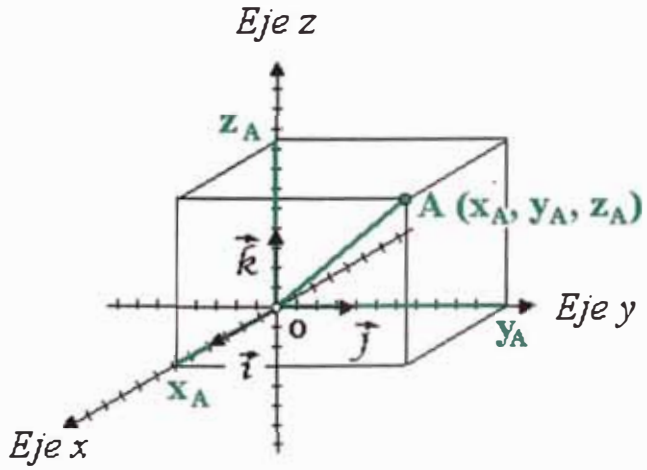


Figura 2.1: Sistema de coordenadas cartesianas espaciales.

De la figura 2.1, el punto A esta representado por el vector $(x_A, y_A, z_A) \in R^3$ que indica su posición en el espacio tridimensional.

2.1.1 Coordenadas homogéneas

Otra representación se puede dar mediante un vector aumentado $\bar{\mathbf{x}}_A$. El vector $\mathbf{x}_A = (x_A, y_A, z_A)$ fue la representación cartesiana del punto A, mientras el vector $(x_A, y_A, z_A, 1) \in R^4$ se le denomina vector aumentado.

La finalidad de esta nueva representación es facilitar los cálculos matemáticos en las transformaciones de sistemas paralelos y en el de las proyecciones de imágenes como es el caso que lo generan las cámaras digitales. Esta nueva representación está dentro de la geometría proyectiva y se le denomina coordenadas homogéneas. Algunas ventajas de las coordenadas homogéneas son la simplicidad de los cálculos matemáticos, los puntos en el infinito pueden ser representados por coordenadas finitas, y una sola matriz puede representar múltiples transformaciones en el espacio euclídeo.

Definimos entonces las coordenadas homogéneas para el punto A de la siguiente manera:

$$\tilde{\mathbf{x}}_A = \begin{pmatrix} \omega x_A \\ \omega y_A \\ \omega z_A \\ \omega \end{pmatrix} = \begin{pmatrix} \tilde{x}_A \\ \tilde{y}_A \\ \tilde{z}_A \\ \omega \end{pmatrix} \in P^3 \qquad (2.1)$$

Una ilustración de las coordenadas homogéneas se presenta en la Figura 2.2.

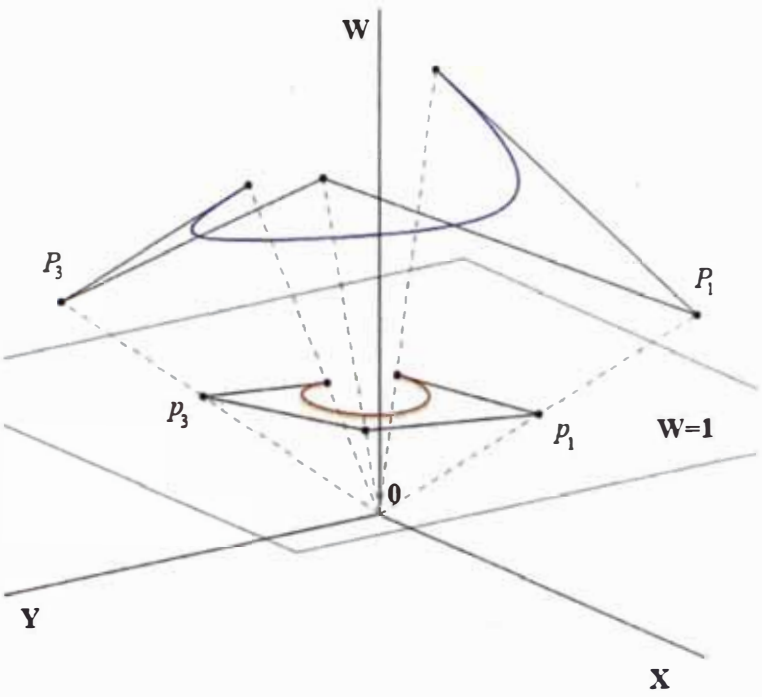


Figura 2.2: Curva polinomial definida en coordenadas homogéneas (azul) y su proyección en el plano (w = 1).

2.1.2 Transformaciones en coordenadas homogéneas

Una transformación es un mapeo lineal invertible expresado de la siguiente manera, donde M es una matriz de dimensiones M_{4x4} para un espacio euclídeo de tres dimensiones [13]:

$$\mathbf{x}' = M\mathbf{x} \tag{2.2}$$

- **Traslación**

La traslación en el espacio tridimensional está representado por el vector de translación $\mathbf{t} = (t_x, t_y, t_z)$, que contiene tres parámetros, en la matriz M como se indica:

$$M = \begin{pmatrix} I & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \tag{2.3}$$

Donde I es la matriz de identidad de dimensiones I_{3x3}.

- **Rotación**

La rotación está representado por la matriz de rotación **R**, que contiene tres parámetros, en la matriz M como se indica:

$$M = \begin{pmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \tag{2.4}$$

La matriz de rotación **R** es una matriz ortogonal de dimensiones $\mathbf{R}_{3x3}$.

- **Transformación euclidiana**

Es la transformación del movimiento de un cuerpo rígido, la que representa una traslación y una rotación consecutiva:

$$M = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \tag{2.5}$$

- **Transformación de semejanza**

Transformación que contiene siete parámetros, donde m es un factor de escala, esta transformación preserva los ángulos entre líneas y planos:

$$M = \begin{pmatrix} \mathrm{m}\mathbf{R} & \mathbf{t} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \tag{2.6}$$

- **Transformación de afinidad**

Transformación que contiene doce parámetros, las líneas y planos paralelos permanecen paralelos en la transformación:

$$M = \begin{pmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \tag{2.7}$$

Un resumen de todas estas transformaciones en coordenadas homogéneas se pueden encontrar en la Tabla 2.1.

De las transformaciones presentadas, la transformación euclidiana es la más usada en robótica, porque describe el movimiento de un robot en el espacio, descrito por un movimiento de traslación y una rotación (ver ecuación (2.5)).

Tabla 2.1: Transformaciones homogéneas en dos dimensiones P^2.

2D Transformation	Figure	d. o. f.	H	H
Translation		2	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} I & t \\ 0^T & 1 \end{bmatrix}$
Mirroring at y-axis		1	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} Z & 0 \\ 0^T & 1 \end{bmatrix}$
Rotation		1	$\begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} R & 0 \\ 0^T & 1 \end{bmatrix}$
Motion		3	$\begin{bmatrix} \cos\varphi & -\sin\varphi & t_x \\ \sin\varphi & \cos\varphi & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}$
Similarity		4	$\begin{bmatrix} a & -b & t_x \\ b & a & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \lambda R & t \\ 0^T & 1 \end{bmatrix}$
Scale difference		1	$\begin{bmatrix} 1+m/2 & 0 & 0 \\ 0 & 1-m/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} D & 0 \\ 0^T & 1 \end{bmatrix}$
Shear		1	$\begin{bmatrix} 1 & s/2 & 0 \\ s/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} S & 0 \\ 0^T & 1 \end{bmatrix}$
Asym. shear		1	$\begin{bmatrix} 1 & s' & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} S' & 0 \\ 0^T & 1 \end{bmatrix}$
Affinity		6	$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix}$
Projectivity		8	$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$	$\begin{bmatrix} A & t \\ p^T & 1/\lambda \end{bmatrix}$

2.1.3 Coordenadas globales y locales

En robótica es necesario presentar dos sistemas de coordenadas de referencia, un sistema de coordenadas global, fijo al entorno que rodea al robot, y otro sistema de coordenadas inercial o local, generalmente fijo al centro de masa del robot móvil (ver Figura 2.3). El motivo de contar con un sistema de coordenadas inercial, es por ejemplo

para el cálculo de la posición en el entorno de un objeto observado por el robot, mediante la matriz de transformación presentada en la ecuación (2.5), conociendo la posición relativa del objeto al robot.

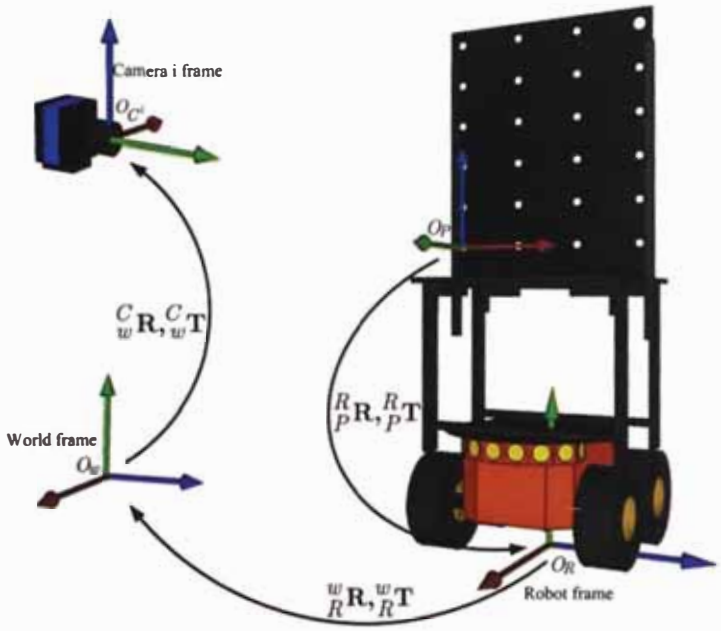


Figura 2.3: Ejemplo de sistemas de coordenadas globales y locales de un vehículo.

- **Transformación local a global**

Como fue mencionado previamente, usualmente un robot móvil observa un objeto representado en su sistema de coordenadas inercial, y se quisiera conocer la posición de dicho objeto en el sistema de coordenadas global. Esto se logra mediante la matriz de transformación que define la posición y orientación del robot en dicho instante de tiempo, por lo que se tiene lo siguiente:

$$\mathbf{x}_{global} = M_t \mathbf{x}_{local} \tag{2.8}$$

$$M_t = \begin{pmatrix} \mathbf{R}_t & \mathbf{t}_t \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \tag{2.9}$$

- **Transformación global a local**

De igual manera, también se requiere alcanzar una posición definida en la referencia global, ingresando comando locales al robot. Esto se logra mediante la matriz de transformación inversa:

$$\mathbf{x}_{local} = {M_t}^{-1}\mathbf{x}_{global} \tag{2.10}$$

$${M_t}^{-1} = \begin{pmatrix} {\mathbf{R}_t}^{\mathrm{T}} & -{\mathbf{R}_t}^{\mathrm{T}}\mathbf{t}_t \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \tag{2.11}$$

2.1.4 Ángulos de Euler

Los ángulos de Euler representan la orientación de un cuerpo respecto a un sistema de coordenadas de referencia XYZ o con respecto a otro cuerpo rígido. Para describir la orientación en el espacio de un cuerpo en términos de ángulos de Euler es necesario definir tres rotaciones comúnmente usadas en navegación aérea según norma DIN 9300 mostrados en la Figura 2.4.

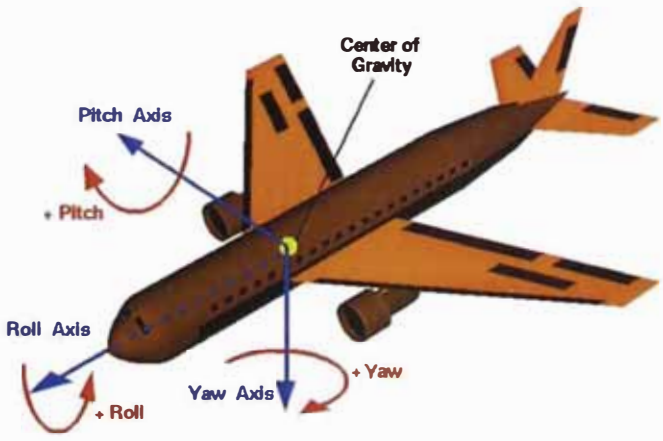


Figura 2.4: Ángulos de Euler, roll, pitch y yaw en un aeroplano.

Donde:

- ϕ (roll) describe la rotación alrededor del eje x.
- θ (pitch) describe la rotación alrededor del eje y.
- ψ (yaw) describe la rotación alrededor del eje z.

De acuerdo al teorema de rotaciones de Euler [16], cualquier rotación de un cuerpo rígido en el espacio tridimensional, puede ser descrito usando estos tres ángulos de Euler. Existes muchas convenciones para la representación de la rotación de un cuerpo usando los ángulos de Euler. La llamada convención ZYX (ver Figura 2.5) es la más comúnmente utilizada en ingeniería aeronáutica.

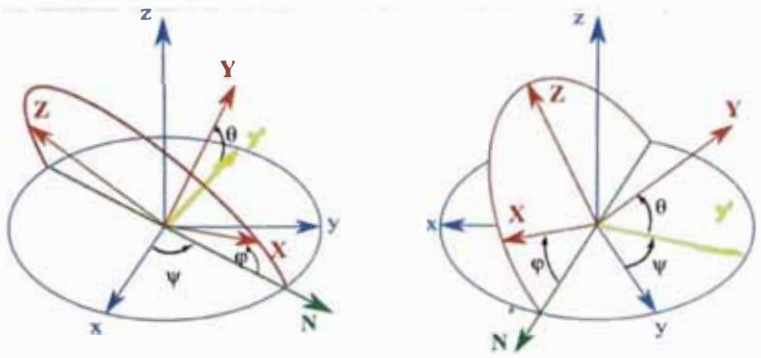


Figura 2.5: Rotaciones de Euler de acuerdo a la convención ZYX.

En esta convención, la primera rotación es realizada alrededor del eje z por un ángulo ψ, la segunda rotación es realizada alrededor del eje y por un ángulo θ y la tercera rotación es realizada sobre el eje x por un ángulo ϕ.

La matriz de rotación **R** definida por las tres rotaciones consecutivas de acuerdo a la convención ZYX es entonces la siguiente:

$$\mathbf{R} = \mathbf{R}_Z(\psi)\mathbf{R}_Y(\theta)\mathbf{R}_X(\phi) \tag{2.12}$$

Donde:

$$\mathbf{R}_Z(\psi) = \begin{pmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.13}$$

$$\mathbf{R}_Y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \tag{2.14}$$

$$\mathbf{R}_X(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{pmatrix} \tag{2.15}$$

Por lo que la matriz de rotación **R** usando los términos s para el seno y c para el coseno, es la siguiente:

$$\mathbf{R} = \begin{pmatrix} c(\psi)s(\theta) & c(\psi)s(\theta)s(\phi) - s(\psi)c(\phi) & c(\psi)s(\theta)c(\phi) + s(\psi)s(\phi) \\ s(\psi)c(\theta) & s(\psi)s(\theta)s(\phi) + c(\psi)c(\phi) & s(\psi)s(\theta)c(\phi) - c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{pmatrix} \tag{2.16}$$

2.2 Seguimiento de Trayectoria

Cuando se realiza el control de seguimiento de trayectorias de robots móviles esta generalmente está determinada por una ecuación matemática, sin embargo también es deseable en algunos casos que el robot siga cualquier otra trayectoria definida por puntos en el plano. Existen entonces dos formas básicas para definir la trayectoria [7]:

- Suministrando puntos consecutivos e ignorando la trayectoria espacial que describe el robot entre cada dos puntos.
- Especificando la trayectoria por una curva paramétrica en el tiempo, tal como una recta o una parábola, que debe describir el robot en el espacio de trabajo.

La primera alternativa, denominada también control punto a punto, sólo tiene interés práctico cuando los puntos están suficientemente separados, ya que en caso contrario, sería muy laborioso especificar todos los puntos intermedios. Su contraparte es que los puntos tampoco puedan estar muy separados ya que existe un alto riesgo de que se generen movimientos imprevisibles o no controlados. Para la solución del problema del control punto a punto existen algoritmos que realizan la interpolación entre los puntos específicos, siendo los más comunes los polinomios cúbicos y los splines haciendo posible realizar el control del robot para que pase por dichos puntos.

La segunda forma para realizar el seguimiento de trayectorias es utilizando curvas paramétricas en el tiempo como trayectoria deseada que se denomina tradicionalmente control de trayectoria continua, el control debe hacer que el robot reproduzca lo más exacto posible la trayectoria especificada.

Para la generación de trayectorias, en ambos casos, es necesario tener en cuenta lo siguiente:

- Las restricciones cinemáticas y dinámicas involucradas.
- Las trayectorias generadas deben ser suaves, lo que implica restricciones sobre las derivadas.
- Al menos la primera derivada debe ser continua, pudiendo exigirse también la continuidad en derivadas de orden superior.
- La generación de trayectorias debe ser computacionalmente eficiente.

2.2.1 Control punto a punto

Para la solución de la generación de trayectorias por control punto a punto son comunes los siguientes métodos de interpolación:

- **Interpolación polinómica**

Teniendo una trayectoria cualquiera definida y continua en un intervalo se puede encontrar una función que se aproxime lo más posible a esta trayectoria. Una de las funciones más conocidas para realizar esto son los polinomios algebraicos de la forma $P_m(x) = a_n x^n + a_{n-1} x^{n-1} \dots a_1 x + a_0$ donde n es un número entero no negativo y $a_0 \dots a_n$ son constantes reales.

Empleando polinomios cúbicos se puede aproximar lo más posible a una trayectoria definida por puntos, pero si la trayectoria tiene muchos puntos de inflexión el polinomio cúbico ya no es representativo.

- **Funciones spline**

La interpolación polinómica se basa en la sustitución de una función o una tabla de valores por un polinomio que toma dichos valores. Cuando el número de puntos aumenta, también aumenta el grado del polinomio, que se hace más oscilante, lo cual se traduce en un aumento de errores en la trayectoria generada. Un enfoque alternativo a la utilización de polinomios de grado alto es el uso de polinomios de grado menor en subintervalos. El procedimiento consiste en dividir el intervalo en una serie de sub intervalos, y en cada subintervalo definir un polinomio cúbico. A este método se le conoce como aproximación polinómica fragmentaria, o splines. Esta aproximación consiste en unir una serie de puntos dados mediante una serie de polinomios cúbicos, tal como se muestra en la Figura 2.6.

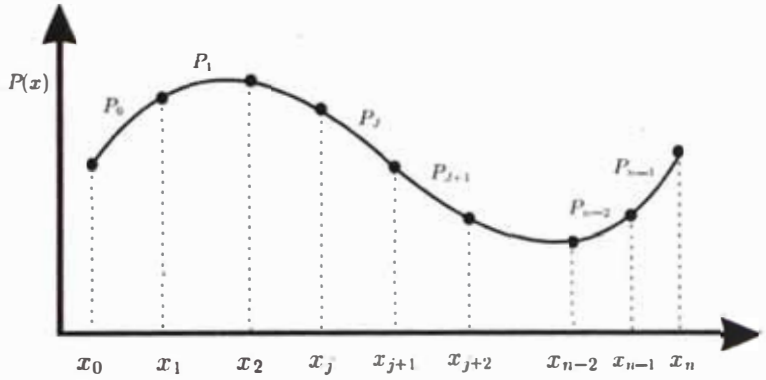


Figura 2.6: Aproximación polinómica fragmentaria.

2.2.2 Control de trayectoria continua

Se puede usar las coordenadas x y y para describir la ubicación del robot móvil en cualquier punto de su recorrido.

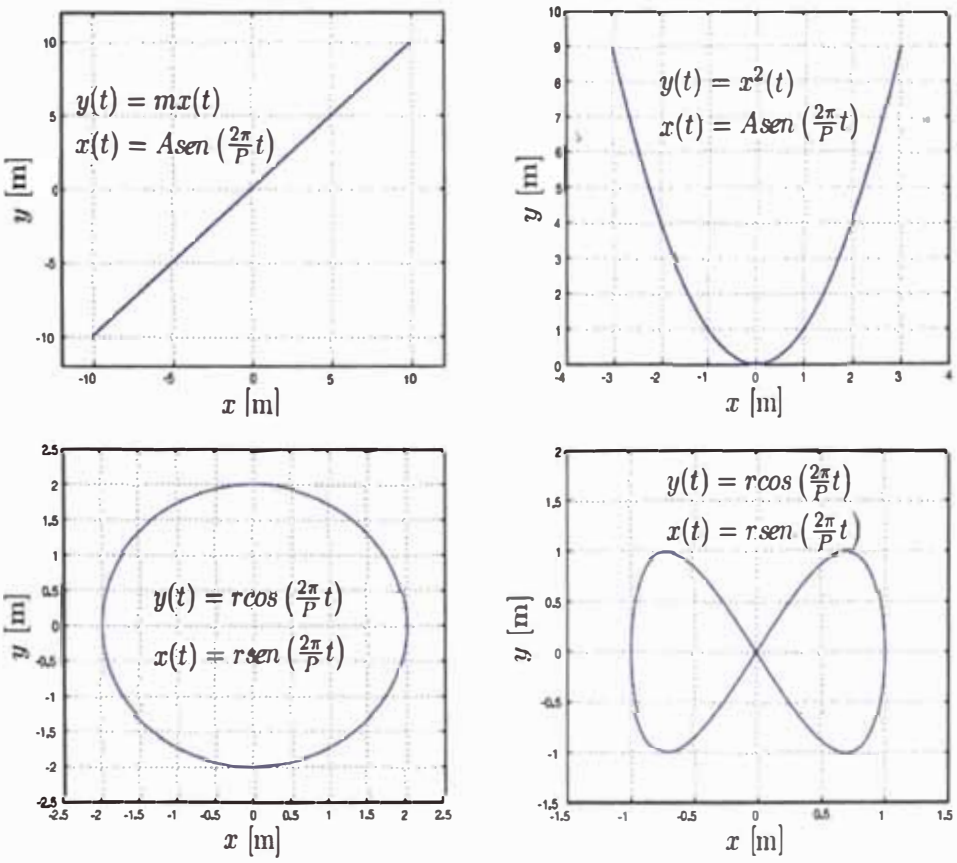


Figura 2.7: Curvas parametrizadas en el tiempo.

Sin embargo, las coordenadas no incidían en el recorrido cuando el robot se encuentra en diferente ubicación, por tal motivo en ocasiones, dos variables no son suficientes para describir completamente una situación de gráfica. Una alternativa de solución es usar ecuaciones paramétricas para describir las coordenadas x y y de un punto como funciones de una tercera variable, tiempo t, que se denomina parámetro. Por tanto una curva paramétrica es la que está definida por ecuaciones paramétricas. Algunas de las curvas paramétricas en el tiempo más comunes son mostradas en la Figura 2.7.

2.3 Estimación de estado

La estimación de estado es la solución al problema fundamental de localización de un robot móvil. Esta solución se basa en la teoría de probabilidades que estaremos describiendo en el presente subcapítulo.

2.3.1 Modelo de estado

Es la ecuación que define el comportamiento en el tiempo del robot. De manera simplificada, la ecuación es la siguiente:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \tag{2.17}$$

Donde el estado estimado $\mathbf{x}_{t+1}$ está dado por el estado actual $\mathbf{x}_t$ y las entradas de control o acciones $\mathbf{u}_t$.

Como se puede observar, podemos estimar el estado en el que se encontrará posicionado nuestro robot en el tiempo $t + 1$, conociendo el estado en el que se encuentra actualmente y las acciones $\mathbf{u}_t$ del robot, que son las lecturas de las velocidades obtenidas de su odometría.

Este modelo predictor de la posición, no concuerda con el estado real del robot, debido a que todas las lecturas hechas por sus sensores de odometría (en los casos de robots móviles con ruedas) - también llamados sensores de velocidad para lo demás tipos de vehículos móviles o sensores de odometría visual como es nuestro caso - contienen alto ruido, y por ende no está garantizado que lo que el robot, por su modelo odométrico, determina donde está es realmente su posición real.

Es por ello que para una mejor representación del estado estimado del robot, se necesita también inferir su estado actual de las observaciones de otros sensores así como lo hemos hecho de las acciones (movimientos) ejecutadas.

2.3.2 Observadores de estado

Una de las tareas más importantes de un vehículo autónomo es la adquisición de conocimiento del entorno que lo rodea. Esto es realizado con diversos sensores, extrayendo la información significativa de sus mediciones.

Una amplia variedad de sensores son usados en robótica móvil [3]. Algunos de ellos son usados para mediciones internas del robot, como por ejemplo la temperatura de sus tarjetas electrónicas o la velocidad de rotación de sus motores. Otros sensores más sofisticados son usados para la adquisición de información del entorno que lo rodea e inclusive la medición de su posición global a través de GPS.

Los sensores usados en los robots móviles pueden ser clasificados en dos grandes tipos [10] p. 101:

- Sensores Propioceptivos, los cuales realizan mediciones de las características internas del robot, como su temperatura, velocidad de rotación de sus motores, nivel de baterías, etc.
- Sensores Exteroceptivos, los cuales adquieren información del entorno en el que se encuentra el robot, como por ejemplo distancias, intensidad lumínica, obstáculos cercanos, etc.

La mayor variedad de sensores usados en robótica móvil puede ser encontrados en la Tabla 2.2, debidamente clasificados [10] p. 104.

Para nuestro caso, el cuadricóptero usa la cámara inferior junto a su unidad de medición inercial IMU para el cálculo de sus velocidades [ref Anexo]. En este caso, esta cámara CMOS fue utilizada para inferir sus características internas (velocidades) por lo que se tendría que clasificar como un sensor propioceptivo al igual que los sensores en los motores en el caso de robots móviles con ruedas, como se muestra en la Tabla 2.2. Sin embargo, se va a usar la misma cámara para el reconocimiento de marcadores (*landmarks*) que serán colocados en el suelo, lo cual formará el modelo de observación para facilitar la estimación de la posición actual del cuadricóptero. En este caso, la cámara CMOS será usada para obtener información del entorno y por ende será clasificado como un sensor exteroceptivo como es mostrado en la Tabla 2.2.

La representación del modelo de las observaciones de los sensores puede ser expresada con la siguiente ecuación [14]:

$$\mathbf{z}_t = h(\mathbf{x}_t) \tag{2.18}$$

Donde $\mathbf{z}_t$ es la lectura del sensor, $\mathbf{x}_t$ es el estado actual del robot, y h es la función de observación que describe el modelo del sensor.

Tabla 2.2: Clasificación de sensores usados robótica móvil. A: activo, P: pasivo, PC: propioceptivo, EC: exteroceptivo.

General classification (typical use)	Sensor Sensor System	PC or EC	A or P
Tactile sensors (detection of physical contact or closeness; security switches)	Contact switches, bumpers	EC	P
	Optical barriers	EC	A
	Noncontact proximity sensors	EC	A
Wheel/motor sensors (wheel/motor speed and position)	Brush encoders	PC	P
	Potentiometers	PC	P
	Synchros, resolvers	PC	A
	Optical encoders	PC	A
	Magnetic encoders	PC	A
	Inductive encoders	PC	A
	Capacitive encoders	PC	A
Heading sensors (orientation of the robot in relation to a fixed reference frame)	Compass	EC	P
	Gyroscopes	PC	P
	Inclinometers	EC	A/P
Acceleration sensor	Accelerometer	PC	P
Ground beacons (localization in a fixed reference frame)	GPS	EC	A
	Active optical or RF beacons	EC	A
	Active ultrasonic beacons	EC	A
	Reflective beacons	EC	A
Active ranging (reflectivity, time-of-flight, and geometric triangulation)	Reflectivity sensors	EC	A
	Ultrasonic sensor	EC	A
	Laser rangefinder	EC	A
	Optical triangulation (1D)	EC	A
	Structured light (2D)	EC	A
Motion/speed sensors (speed relative to fixed or moving objects)	Doppler radar	EC	A
	Doppler sound	EC	A
Vision sensors (visual ranging, whole-image analysis, segmentation, object recognition)	CCD/CMOS camera(s)	EC	P
	Visual ranging packages		
	Object tracking packages		

2.3.3 Robótica probabilística

Tanto la representación del modelo de las observaciones de los sensores como del movimiento del robot, son imprecisos y no concuerdan con su estado real, debido siempre al ruido presente en los sensores.

Es por ello que se requiere de una representación no única que defina el estado estimado del robot, con un rango de posibles valores del estado para una misma unidad de tiempo.

Robótica probabilística es una creciente área de la robótica, que se encarga del estudio de la incertidumbre en la percepción y el control. Su función principal es la representación matemática de la incertidumbre basándose en la teoría de las probabilidades.

En esta materia, las nuevas representaciones para los modelos de estado de las acciones (odometría) y de las observaciones del robot se trasladan a modelos probabilísticos de estado los cuales son presentados a continuación:

- **Modelo probabilístico de estado**

En la sección 2.3.1, se vio que el modelo del movimiento del robot, estaba expresado por la función $f(\mathbf{x}_t, \mathbf{u}_t)$. Para efectos de estimación de estado, la representación probabilística del movimiento es la siguiente [14]:

$$p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{u}_t) \tag{2.19}$$

En esta nueva expresión, las variables $\mathbf{x}_t$ y $\mathbf{u}_t$ son variables de estado que dejan de tomar un valor único, para ahora contener un número determinado de valores o un rango continuo de valores, esto dependiendo si se optan por variables discretas o continuas respectivamente. De esta manera, el estado de robot no está determinado en un único valor, sino dentro de una incertidumbre determinada por la distribución probabilística de dichas variables (ver Figura 2.8).

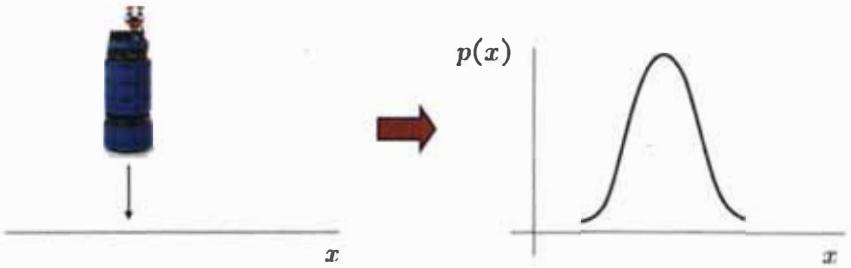


Figura 2.8: Transformación de una representación única de la posición a una representación probabilística.

- **Modelo probabilístico de los observadores de estado**

En la sección 2.3.2, se vio que el modelo de las observaciones de los sensores estaba representado por la ecuación $\mathbf{z}_t = h(\mathbf{x}_t)$.

Para efectos de estimación, el nuevo modelo probabilístico de las observaciones del sensor está representada por la siguiente expresión [14]:

$$p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m}) \tag{2.20}$$

El cual indica la probabilidad de la observación $\mathbf{z}_t$, conociendo el estado $\mathbf{x}_t$ (posición) del robot y el mapa $\mathbf{m}$ de su entorno.

De igual manera, la variable $\mathbf{z}_t$ representa la incertidumbre de la observación determinada por una distribución de probabilidad.

Antes de continuar con el desarrollo de las nuevas representaciones probabilísticas para la estimación del estado del robot aplicado al cuadricóptero, se ofrece una breve introducción de la teoría de probabilidades y sus propiedades.

2.3.4 Teoría de probabilidades

La teoría de la probabilidad es el estudio de procesos aleatorios estocásticos que pueden producir un diverso número de resultados. Por ejemplo, al tirar un dado se presentan 6 posibles resultados, que pueden tomar los valores x:{1, 2, 3, 4, 5, 6}. Denotemos como X al resultado que se puede obtener al tirar el dado. En este caso X es una variable aleatoria discreta, ya que toma un determinado número de valores.

$$p(X = x) \tag{2.21}$$

La probabilidad para que X tome el valor 4 por ejemplo, asumiendo una misma probabilidad para cada valor, es de 1/6. Esto lo podemos expresar de la siguiente manera:

$$p(X = 4) = 1/6 \tag{2.22}$$

- **Axiomas de la teoría de probabilidades**

  Dado el conjunto A y B graficado a continuación en la Figura 2.9:

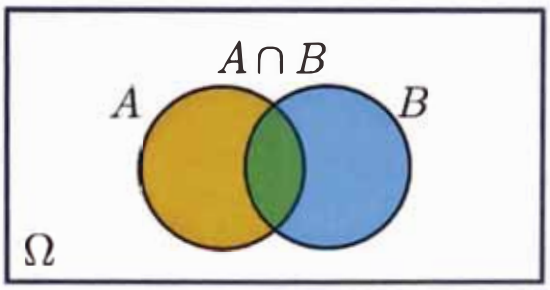


Figura 2.9: Representación gráfica de las variables aleatorias A y B.

Se define lo siguiente [12]:

1. $0 \leq p(A) \leq 1$
2. $p(\Omega) = 1$
3. $p(\phi) = 0$
4. $p(A \cup B) = p(A) + p(B) - p(A \cap B)$

- **Variables aleatorias**

Las variables aleatorias tomar un número determinado de valores o conforman un rango continuo de valores, dependiendo si son variables discretas o continuas.

- **Variables aleatorias discretas**
  - X denota una variable aleatoria
  - X puede tomar un numero contable de valores en $\{x_1, x_2, \ldots x_n\}$
  - $p(X = x_i)$ es la probabilidad que la variable aleatoria X tome el valor de x_i. Esto es simplificado a $p(x_i)$
  - $\sum_x p(x) = 1$
- **Variables aleatorias continuas**
  - X toma valores continuos
  - $p(X = x)$ o $p(x)$ es llamado la función de densidad de probabilidad (PDF). $p(x \in [a, b]) = \int_a^b p(x)dx$
  - $\int p(x)dx = 1$

- **Probabilidad condicional**
  - $p(X = x \wedge Y = y) = p(x, y)$
  - Si X y Y son independientes, entonces $p(x, y) = p(x)p(y)$
  - $P(x|y)$ es la probabilidad de x dado y, se define entonces $p(x|y)p(y) = p(x, y)$
  - Si X y Y son independientes entonces $p(x|y) = p(x)$

- **Independencia condicional**

La definición de la independencia condicional viene dada por la siguiente expresión: $p(x, y|z) = p(x|z)p(y|z)$

Equivalente a:

- $p(x|z) = p(x|y, z)$
- $p(y|z) = p(y|x, z)$

Nota: esto no necesariamente significa que X y Y son independientes.

- **Teorema de la probabilidad total**
  - Caso Discreto: $p(x) = \sum p(x|y_i)p(y_i)$
  - Caso Continuo: $p(x) = \int p(x|y)p(y)dy$

- **Estimación de la data**

Sean los valores aleatorios discretos $\mathbf{x}$: $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$

- La media está definida por: $\mu = \sum_i x_i$
- La covarianza está definida por: $\Sigma = \frac{1}{n-1}\sum_i(\mathbf{x}_i - \mu)(\mathbf{x}_i - u)^T$

- **Regla de Bayes**

Si queremos estimar el estado actual del robot debemos obtener una relación entre los modelos probabilísticos del robot descritos anteriormente en la sección 2.3.3. Esta relación la podemos obtener gracias a la Regla de Bayes de la teoría de probabilidades.

De la definición de la Probabilidad Condicional, se obtiene:

$$p(x, y) = p(x|y)p(y) = p(y|x)p(x) \tag{2.23}$$

Del cual obtenemos la Regla de Bayes que se expresa de la siguiente forma:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{2.24}$$

$p(y)$ puede ser computacionalmente costoso de obtener, pero lo podemos suprimir por normalización.

$$p(x|y) = \eta\, p(y|x)p(x) \tag{2.25}$$

- **Supuesto de Markov**

En teoría de probabilidades, el supuesto de Markov se refiere a la propiedad de ciertos procesos estocásticos por la cual "carecen de memoria", lo que significa que la distribución de probabilidad del valor futuro de una variable aleatoria depende de su valor presente, pero es independiente de la historia de dicha variable.

En nuestro caso, hemos hecho uso de este supuesto para definir la probabilidad de la observación de los sensores dado solo el estado actual del robot, no siendo necesario conocer los estados previos (ver ecuación (2.20)) [14] p. 21:

$$p(z_t \mid x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t \mid x_t) \tag{2.26}$$

De igual manera se ha aplicado para definir la probabilidad del estado actual del robot, que solamente depende del estado anterior y la acción (entrada de control) actual (ver ecuación (2.19)) [14] p. 21:

$$p(x_t \mid x_{0:t-1}, z_{1:t}, u_{1:t}) = p(x_t \mid x_{t-1}, u_t) \tag{2.27}$$

2.3.5 Filtro de Bayes

Ahora que tenemos el modelo probabilístico de la observación de los sensores del robot, que solo depende del estado actual del mismo, y a su vez tenemos el modelo probabilístico del estado actual, que solo depende de su estado anterior y la entrada de

control, se quiere hallar un único modelo para estimar el estado actual del robot, dada las observaciones de los sensores y la entrada de control, en otras palabras, se quiere fusionar ambos modelos para obtener una estimación del estado actual del robot. A esta estimación la llamaremos *belief* (esperanza) y la obtenemos gracias al filtro de Bayes:

Tenemos nuestros modelos representados por las ecuaciones (2.19) y (2.20), y deseamos una sola expresión que estime el estado actual, dadas las acciones y observaciones del robot. Esta expresión está representada por la siguiente ecuación:

$$p(x_t \mid z_{1:t}, u_{1:t}) \tag{2.28}$$

A lo cual llamaremos $bel(x_t)$ [14] p. 22:

$$bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t}) \tag{2.29}$$

De la Regla de Bayes, podemos representar la esperanza de la siguiente manera:

$$bel(x_t) = \eta\, p(z_t \mid x_t, z_{1:t-1}, u_{1:t})\, p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{2.30}$$

Por el supuesto de Markov, se ha descartado las observaciones y acciones pasadas (ver ecuación (2.26)), simplificando la expresión a:

$$bel(x_t) = \eta\, p(z_t \mid x_t)\, p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{2.31}$$

Por el teorema de probabilidad total, se infiere que:

$$p(x_t \mid z_{1:t-1}, u_{1:t})$$

$$= \int_{x_{t-1}} p(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t})\, p(x_{t-1} \mid z_{1:t-1}, u_{1:t})\, dx_{t-1} \tag{2.32}$$

Reemplazándolo a la ecuación (2.31), tenemos:

$$bel(x_t)$$

$$= \eta\, p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid x_{t-1}, z_{1:t-1}, u_{1:t})\, p(x_{t-1} \mid z_{1:t-1}, u_{1:t})\, dx_{t-1} \tag{2.33}$$

Por el supuesto de Markov, se ha descartado todas las observaciones y acciones pasadas (ver ecuación (2.27)), simplificando la expresión a la siguiente:

$$bel(x_t)$$

$$= \eta\, p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid x_{t-1}, u_t)\, p(x_{t-1} \mid z_{1:t-1}, u_{1:t}) dx_{t-1} \tag{2.34}$$

La probabilidad del estado x_{t-1} en la expresión $p(x_{t-1} \mid z_{1:t-1}, u_{1:t})$, no depende de la acción u_t, por lo que se tiene lo siguiente:

$$p(x_{t-1} \mid z_{1:t-1}, u_{1:t}) = p(x_{t-1} \mid z_{1:t-1}, u_{1:t-1}) \tag{2.35}$$

Reemplazando en la ecuación (2.34), tenemos:

$$bel(x_t)$$

$$= \eta\, p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid x_{t-1}, u_t)\, p(x_{t-1} \mid z_{1:t-1}, u_{1:t-1}) dx_{t-1} \tag{2.36}$$

De la definición inicial (2.29), tenemos:

$$bel(x_{t-1}) = p(x_{t-1} \mid z_{1:t-1}, u_{1:t-1}) \tag{2.37}$$

Reemplazando en la ecuación (2.36), obtenemos:

$$bel(x_t) = \eta\, p(z_t \mid x_t) \int_{x_{t-1}} p(x_t \mid x_{t-1}, u_t)\, bel(x_{t-1}) dx_{t-1} \tag{2.38}$$

Como se observa, se ha obtenido una expresión recursiva para la estimación del estado $bel(x_t)$.

Esta expresión se divide en dos etapas, a las cuales se le llaman Predicción y Corrección [14] p. 22.

Predicción

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\, bel(x_{t-1}) d\, x_{t-1} \tag{2.39}$$

En esta etapa, obtenemos una predicción de estado actual del robot, dada la estimación anterior y nuestro modelo del movimiento del robot.

Corrección

$$bel(x_t) = \eta\, p(z_t \mid x_t)\ \overline{bel}(x_t) \tag{2.40}$$

Es esta etapa, obtenemos la estimación actual del robot, dada nuestra predicción en la etapa anterior, y la observación de los sensores.

2.3.6 Filtro de Kalman

El filtro de Kalman es un filtro de Bayes de variables continuas, cuyos estados probabilísticos son representados mediante una distribución normal (distribución gaussiana).

El filtro de Kalman es muy eficiente y presenta una solución óptima para sistemas lineales.

Su campo de aplicación es muy amplio. Se utiliza en ciencias económicas, en predicciones climatológicas, navegación satelital y robótica.

Para desarrollar mejor el filtro de Kalman, se dará un breve repaso de la distribución normal.

- **Distribución normal**

  La representación matemática de la distribución Gaussiana es la siguiente:

  - **Para una sola variable**

$$X \sim N(\mu, \sigma^2) \tag{2.41}$$

$$p(X = x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right) \tag{2.42}$$

  Donde μ es la media y σ^2 la varianza, siendo σ la desviación estándar (ver Figura 2.10).

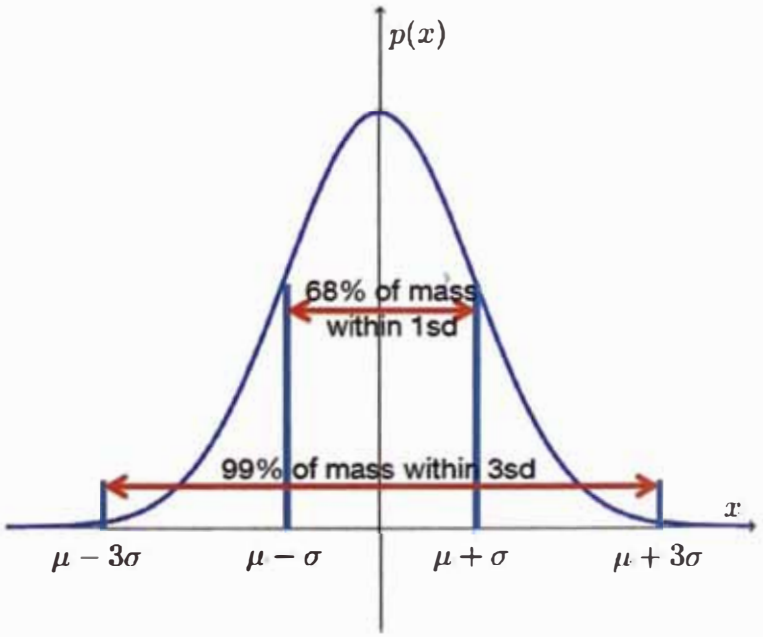


Figura 2.10: Distribución Normal.

  - **Para multivariable:**

$$\mathbf{X} \sim N(\mu, \Sigma) \tag{2.43}$$

$$p(\mathbf{X} = \mathbf{x}) = N(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$= \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \quad (2.44)$$

Donde la media $\boldsymbol{\mu} \in \mathbf{R}^n$ y la covarianza $\boldsymbol{\Sigma} \in \mathbf{R}^{n \times n}$.

- **Propiedades de la distribución normal**

  - Transformación lineal:

    Sea $\mathbf{X} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ y $\mathbf{Y} = \mathbf{AX} + \mathbf{B}$, entonces $\mathbf{Y} \sim N(\mathbf{A}\boldsymbol{\mu} + \mathbf{B}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^T)$ (2.45)

  - Intersección de dos distribuciones Gaussianas:

    $\mathbf{X}_1 \sim N(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ y $\mathbf{X}_2 \sim N(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, entonces

$$p(\mathbf{X}_1)p(\mathbf{X}_2) = N\left(\frac{\boldsymbol{\Sigma}_2}{\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2}\boldsymbol{\mu}_1 + \frac{\boldsymbol{\Sigma}_1}{\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2}\boldsymbol{\mu}_2, \frac{1}{\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1}}\right) \quad (2.46)$$

- **Desarrollo del filtro de Kalman**

Para el desarrollo del filtro de Kalman, se requiere estimar el estado actual del robot, como una variable discreta x_t, que es gobernado por la siguiente ecuación diferencial estocástica:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \quad (2.47)$$

Y de las observaciones en el estado actual:

$$z_t = C_t x_t + \delta_t \quad (2.48)$$

Donde:

A_t Matriz (n x n) que describe como el estado evoluciona de $t - 1$ a t sin entradas de control ni ruido.

B_t Matriz (n x l) que describe como la entrada de control u_t cambia el estado de $t-1$ a t.

C_t Matriz (k x n) que describe como mapear el estado x_t a una observación z_t.

ϵ_t y δ_t Variables aleatorias que representan el ruido en el proceso y observaciones, asumiendo que son independientes y normalmente distribuidos con media cero y covarianza Q_t y R_t respectivamente: $\epsilon_t \sim N(\mathbf{0}, \mathbf{Q})$ y $\delta_t \sim N(\mathbf{0}, \mathbf{R})$.

Se considera lo siguiente:

1. La esperanza inicial es Gaussiana: $\text{bel}(\mathbf{x}_0) = N(\mathbf{x}_0; \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$
2. El estado también es Gaussiano: $\mathbf{x}_t \sim N(\mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t, \mathbf{Q})$
3. Las observaciones también contienes una distribución normal: $\mathbf{z}_t \sim N(\mathbf{C}\mathbf{x}_t, \mathbf{R})$

Del filtro de Bayes tenemos las dos etapas:

Predicción:

La predicción del estado actual del robot, obtenida en la ecuación (2.39), es la siguiente:

$$\overline{\text{bel}}(\mathbf{x}_t) = \int p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t) \text{bel}(\mathbf{x}_{t-1}) \text{d}\, \mathbf{x}_{t-1} \tag{2.49}$$

Corrección:

Aplicando el modelo del sensor, se corrige la predicción anterior, mediante la ecuación (2.40) obtenida previamente:

$$\text{bel}(\mathbf{x}_t) = \eta\, p(\mathbf{z}_t \mid \mathbf{x}_t) \overline{\text{bel}}(\mathbf{x}_t) \tag{2.50}$$

Como fue descrito, las variables de estado del filtro de Kalman están representados mediante distribuciones normales, por lo tanto, reemplazando los modelos de distribución normal, se obtiene:

Predicción:

$$\overline{\text{bel}}(\mathbf{x}_t) = \int \underbrace{p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)}_{N(\mathbf{x}_t;\mathbf{A}\mathbf{x}_{t-1}+\mathbf{B}\mathbf{u}_t,\mathbf{Q})} \quad \underbrace{\text{bel}(\mathbf{x}_{t-1})}_{N(\mathbf{x}_{t-1};\mu_{t-1},\Sigma_{t-1})} \quad \mathrm{d}\,\mathbf{x}_{t-1} \tag{2.51}$$

$$\overline{\text{bel}}(\mathbf{x}_t) = N(\mathbf{x}_t; \mathbf{A}\boldsymbol{\mu}_{t-1} + \mathbf{B}\mathbf{u}_t, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^{\mathrm{T}} + \mathbf{Q}) \tag{2.52}$$

$$\overline{\text{bel}}(\mathbf{x}_t) = N(\mathbf{x}_t;\, \bar{\mu}_t, \bar{\Sigma}_t) \tag{2.53}$$

Corrección:

$$\text{bel}(\mathbf{x}_t) = \underbrace{\eta\, p(\mathbf{z}_t \mid \mathbf{x}_t)}_{N(\mathbf{z}_t;\mathbf{C}\mathbf{x}_t,\mathbf{R})} \quad \underbrace{\overline{\text{bel}}(\mathbf{x}_t)}_{N(\mathbf{x}_t;\,\bar{\mu}_t,\bar{\Sigma}_t)} \tag{2.54}$$

$$\text{bel}(\mathbf{x}_t) = N(\mathbf{x}_t; \bar{\mu}_t + \mathbf{K}_t(\mathbf{z}_t - \mathbf{C}\bar{\mu}), (\mathbf{I} - \mathbf{K}_t\mathbf{C})\bar{\Sigma}) \tag{2.55}$$

$$\text{bel}(\mathbf{x}_t) = N(\mathbf{x}_t;\, \mu_t, \Sigma_t) \tag{2.56}$$

Siendo $\mathbf{K}_t$ la ganancia de Kalman:

$$\mathbf{K}_t = \bar{\Sigma}_t\mathbf{C}^{\mathrm{T}}(\mathbf{C}\bar{\Sigma}_t\mathbf{C}^{\mathrm{T}} + \mathbf{R})^{-1} \tag{2.57}$$

Del cual se obtiene el algoritmo del filtro de Kalman:

Predicción:

En la predicción del estado $\mathbf{x}_t$, la media de su distribución está dada por:

$$\bar{\mu}_t = \mathbf{A}\boldsymbol{\mu}_{t-1} + \mathbf{B}\mathbf{u}_t \tag{2.58}$$

Con una covarianza dada por:

$$\bar{\Sigma}_t = \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^{\mathrm{T}} + \mathbf{Q} \tag{2.59}$$

Corrección:

En la etapa de corrección. La media de la distribución de nuestro estado estimado $\mathbf{x}_t$, está dada por:

$$\mu_t = \bar{\mu}_t + \mathbf{K}_t(\mathbf{z}_t - \mathbf{C}\bar{\mu}_t) \tag{2.60}$$

Con una covarianza dada por:

$$\mathbf{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{C})\bar{\mathbf{\Sigma}}_t \tag{2.61}$$

Siendo $\mathbf{K}_t$ la ganancia de Kalman definida en la ecuación (2.57).

Este algoritmo presenta una solución óptima para sistemas lineales con distribución Gaussiana.

Para nuestro caso, y la mayoría de robots, el modelo matemático del robot es un sistema no lineal.

Para aplicar el modelo del cuadricóptero al filtro de Kalman, podemos linealizar nuestro sistema mediante aproximaciones de Taylor.

2.3.7 Filtro de Kalman extendido (EKF)

El filtro de Kalman Extendido contiene el mismo concepto al filtro de Kalman original, aplicado a sistemas no lineales, que han sido linealizados mediante aproximaciones de Taylor.

Para el filtro de Kalman, las ecuaciones de estado estaban representados por las ecuaciones lineales (2.47) y (2.48).

Para el caso de sistemas no lineales, las ecuaciones de estado están representadas por las siguientes ecuaciones [14] p. 48:

$$x_t = g(x_{t-1}, u_t) + \epsilon_t \tag{2.62}$$

$$z_t = h(x_t) + \delta_t \tag{2.63}$$

Aplicando entonces aproximación de Taylor, nuestra función de estado aproximado es la siguiente:

$$g(\mathbf{x}_{t-1}, \boldsymbol{u}_t) \approx g(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) + \left.\frac{\partial g(\mathbf{x}, \mathbf{u}_t)}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mu_{t-1}} (\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}) \tag{2.64}$$

Donde $\mathbf{G}_t$ es el jacobiano de la función $g(\mathbf{x}, \mathbf{u}_t)$:

$$\mathbf{G}_t = \left.\frac{\partial g(\mathbf{x}, \mathbf{u}_t)}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mu_{t-1}} \tag{2.65}$$

Reemplazamos $\mathbf{G}_t$ en la ecuación (2.64):

$$g(\mathbf{x}_{t-1}, \boldsymbol{u}_t) = g(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) + \mathbf{G}_t(\mathbf{x}_{t-1} - \boldsymbol{\mu}_{t-1}) \tag{2.66}$$

De igual manera, aplicamos expansión de Taylor al modelo del sensor:

$$h(\mathbf{x}_t) \approx h(\overline{\boldsymbol{\mu}}_t) + \left.\frac{\partial h(\mathbf{x}, \mathbf{u}_t)}{\partial \mathbf{x}}\right|_{\mathbf{x}=\overline{\mu}_t} (\mathbf{x}_t - \overline{\boldsymbol{\mu}}_t) \tag{2.67}$$

Donde $\mathbf{H}_t$ es el jacobiano de la función $h(\mathbf{x}, \mathbf{u}_t)$:

$$\mathbf{H}_t = \left.\frac{\partial h(\mathbf{x}, \mathbf{u}_t)}{\partial \mathbf{x}}\right|_{\mathbf{x}=\overline{\mu}_t} \tag{2.68}$$

Obtenemos la siguiente expresión:

$$h(\mathbf{x}_t) = h(\overline{\boldsymbol{\mu}}_t) + \mathbf{H}_t(\mathbf{x}_t - \overline{\boldsymbol{\mu}}_t) \tag{2.69}$$

Del cual se obtiene el algoritmo para el filtro de Kalman Extendido [14] p. 51:

Predicción:

$$\overline{\boldsymbol{\mu}}_t = g(\boldsymbol{\mu}_{t-1}, \mathbf{u}_t) \tag{2.70}$$

$$\overline{\boldsymbol{\Sigma}}_t = \mathbf{G}_t \boldsymbol{\Sigma} \mathbf{G}_t^{\mathrm{T}} + \mathbf{Q} \tag{2.71}$$

Con $\mathbf{G}_t$ definido en la ecuación (2.65).

Corrección:

$$\mu_t = \bar{\mu}_t + \mathbf{K}_t(\mathbf{z}_t - h(\bar{\mu}_t)) \tag{2.72}$$

$$\Sigma_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\bar{\Sigma}_t \tag{2.73}$$

Con ganancia de Kalman $\mathbf{K}_t$:

$$\mathbf{K}_t = \bar{\Sigma}_t\mathbf{H}_t^{\mathrm{T}}(\mathbf{H}_t\bar{\Sigma}_t\mathbf{H}_t^{\mathrm{T}} + \mathbf{R})^{-1} \tag{2.74}$$

Y $\mathbf{H}_t$ definido en la ecuación (2.68).

CAPITULO III
SIMULADOR DE VUELO

Para la implementación de los algoritmos a desarrollar en la presente tesis y para la muestra de resultados, es necesario contar con un simulador programable que modele el vuelo de un cuadricóptero, más específicamente, emule las características dinámicas del AR Drone 2.0 así como de su información de navegación que nos ofrece. Para ello, se han encontrado dos paquetes de simulación, una de ellas llamada *tum_simulator*, desarrollada para el software de simulación Gazebo, programable en la multi-plataforma ROS, de sus siglas en inglés Robot Operating System, en el sistema operativo Ubuntu, y la segunda es un paquete de simulación desarrollado con JavaScript. Para la programación de ambos se va hacer uso del lenguaje de programación Python.

A continuación se presenta una introducción de ambas plataformas y de sus principios de simulación.

3.1 Simulador en Gazebo - ROS

ROS (Robot Operating System) es un entorno de programación multi-plataforma muy flexible, ampliamente usado por la comunidad de investigadores en la robótica, para la elaboración de software para diversos sistemas robóticos. Contiene

una muy amplia gama de herramientas y librerías que simplifican la tarea de crear un comportamiento complejo y robusto de la dinámica de un robot con una amplia variedad de plataformas.

ROS nació como un proyecto grande con muchos antecesores y contribuyentes. La necesidad de un marco de colaboración abierta fue sentida por muchas personas en la comunidad, y muchos proyectos empezaron a crearse para alcanzar este objetivo.

Varios esfuerzos de la Universidad de Stanford a mediados de los años 2000 que implicaban la integración de sistemas embebidos e Inteligencia Artificial, como el Robot (STAIR) de Stanford, y robots personales PR (ver Figura 3.1), crearon prototipos internos de sistemas de software flexibles y dinámicos destinados a la robótica. En el 2007, Willow Garage, una incubadora visionaria de la robótica, proveía siempre de importantes recursos para ampliar estos conceptos y crear implementaciones bien probadas. Este esfuerzo se ha visto impulsado por innumerables investigadores que contribuyeron con su tiempo y experiencia para el núcleo central de ROS y sus paquetes de software fundamentales. En todo momento, el software fue desarrollado para uso abierto bajo la licencia de código abierto BSD, y poco a poco se ha convertido en una plataforma ampliamente utilizada en la comunidad de investigación de la robótica. Como resultado, ROS fue construido desde cero para fomentar el desarrollo de software colaborativo de robótica.

El ecosistema de ROS se compone actualmente de decenas de miles de usuarios en todo el mundo, trabajando en dominios que van desde proyectos por hobby a los grandes sistemas de automatización industrial.

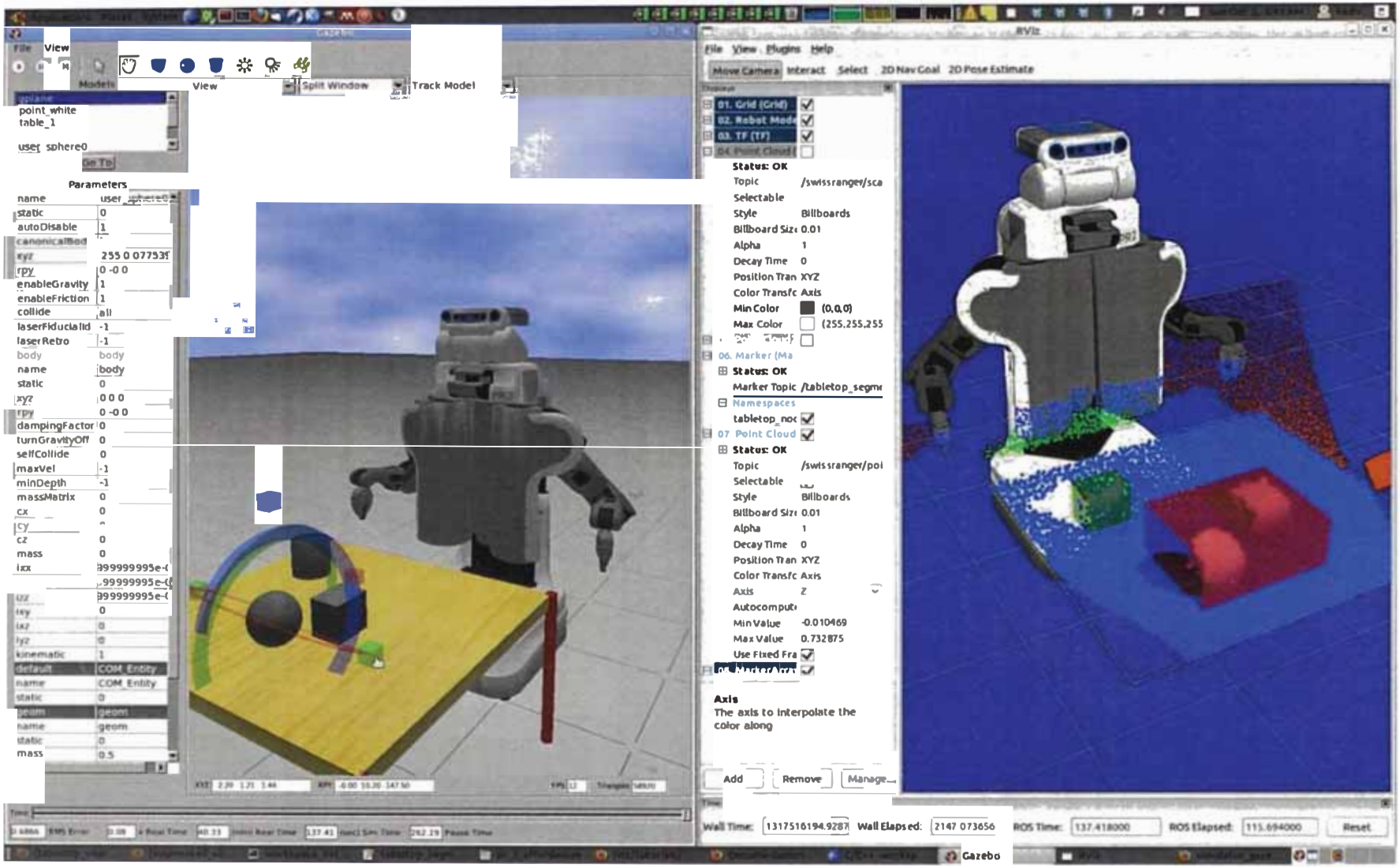


Figura 3.1: Simulador de un PR2 en Gazebo.

Gazebo es una plataforma de simulación de código abierto bajo licencia Apache 2.0. Es probablemente la plataforma más completa para la simulación de sistemas dinámicos. Algunas de sus principales características son sus gráficos 3D avanzados, que ofrece una representación realista de ambientes incluyendo iluminación de alta calidad, sombras y texturas. Ofrece también la generación de información de sensores, incluyendo ruido de sensores laser, cámaras 2D/3D, sensores estilo Kinect, etc. Finalmente también incluye muchos modelos robóticos como el PR2, Pioneer2 DX, iRobot, etc. Ver http://gazebosim.org/ para mayor información.

Gazebo entonces se integra a ROS como una herramienta de simulación de sistemas dinámicos.

Actualmente, existe una reducida gama de paquetes de simulación de vuelo de cuadricópteros en Gazebo, uno de ellos es por ejemplo el paquete llamado *hector_cuadricóptero* (ver Figura 3.2).Ver http://wiki.ros.org/hector_cuadricóptero para mayor información. Para la presente tesis, el paquete de simulación a usar es llamado *tum simulator* el cual se describe más adelante.

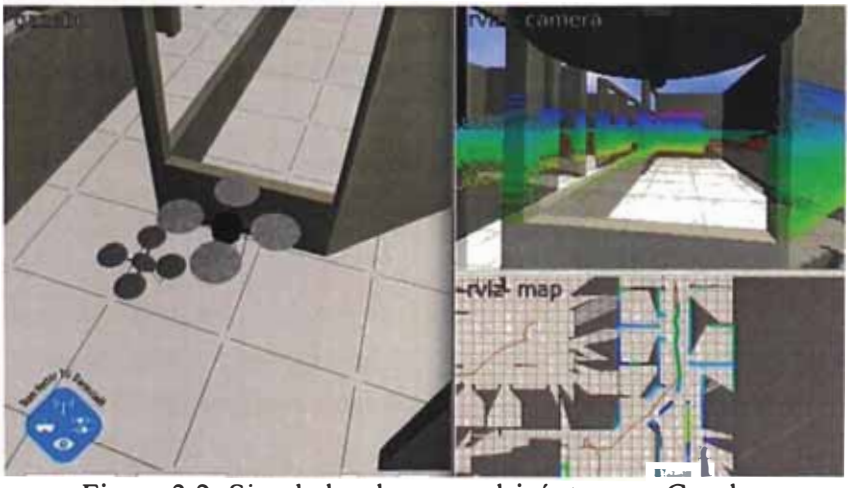


Figura 3.2: Simulador de un cuadricóptero en Gazebo.

3.1.1 Principios del simulador

Para entender el funcionamiento de la multiplataforma ROS, se dará una breve reseña de los principales conceptos de ROS:

- Nodos

Un nodo es un proceso que realiza el cómputo. Los nodos se combinan en un gráfico y se comunican entre sí utilizando tópicos, servicios RPC (Remote Procedure Calls) y servidores de parámetros. Estos nodos están destinados a operar a una escala pequeña, un sistema de control de un robot por ejemplo, normalmente comprenderá muchos nodos; un nodo controla un sensor láser, un nodo controla las ruedas del robot, un nodo lleva a cabo la localización, un nodo lleva a cabo la planificación de la trayectoria, otro nodo proporciona una vista gráfica del sistema, y así sucesivamente.

- Tópicos

Son los buses mediante los cuales los nodos intercambian mensajes. Los tópicos tienen una semántica anónima de publicación/suscripción. En general, los nodos no son conscientes de con quién se están comunicando. En vez de ello, los nodos que se interesan en la data se suscriben a un tópico relevante, y los nodos que generan data publican a un tópico relevante. Puede haber múltiples publicadores y suscriptores a un tópico.

- Mensajes

Los nodos se comunican entre ellos mediante la publicación de mensajes a los tópicos. Un mensaje es una estructura de datos simple, que comprende campos de varios tipos de datos. Se admiten tipos de datos estándar (entero, de coma flotante,

booleanos, etc.) así como los arreglos de dichos tipos. Los mensajes pueden incluir estructuras anidadas y matrices (similar a las estructuras en lenguaje C). Los nodos también pueden intercambiar un mensaje de solicitud y respuesta como parte de una llamada de servicio de ROS. Estos mensajes de solicitud y respuesta se definen en archivos srv.

- Servicios

El modelo de publicación/suscripción es un paradigma de comunicación muy flexible, pero su transporte unidireccional no es apropiado para una solicitud/respuesta RPC, que a menudo se requieren en un sistema distribuido. La solicitud/respuesta se realiza a través de un servicio, que se define por un par de mensajes: uno para la solicitud y otro para la respuesta. Un nodo en ROS ofrece un servicio bajo un nombre *string*, y un cliente llama al servicio mediante el envío de un mensaje de solicitud y espera por la respuesta. Como fue dicho, los servicios se definen usando archivos srv, que se compilan en el código fuente de una librería cliente de ROS.

- Paquetes

Son la unidad principal para la organización de software en ROS. Un paquete puede contener procesos de cómputo de ROS (nodos), librerías, sets de data, archivos de configuración, y demás componentes que son organizados juntos.

A continuación se muestra la Figura 3.3, que grafica el uso de estos conceptos en la simulación de los algoritmos de control. Se describe con mayor detalle la estructura de funcionamiento de la simulación en el siguiente punto.

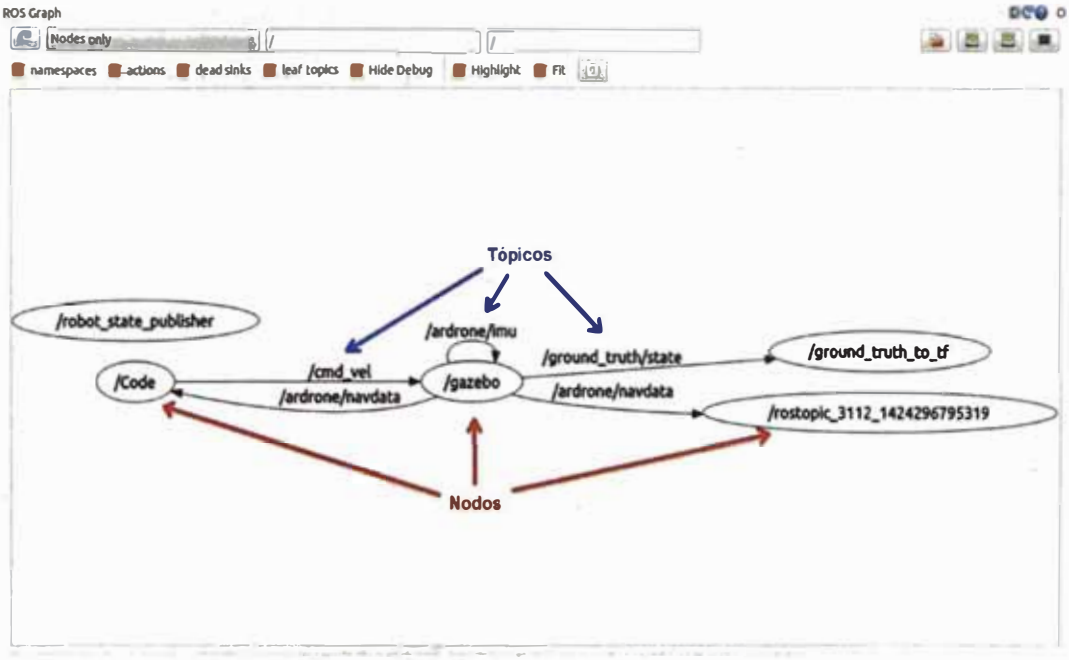


Figura 3.3: Diagrama mostrando los nodos (encerrados en elipses) y tópicos de la ejecución de los algoritmos en el simulador *tum_simulator*.

3.1.2 Estructura del paquete de simulación *tum_simulator*

Como se verá en el CAPÍTULO IV, para la simulación y su posterior implementación del modelo odométrico y del controlador de trayectoria para el Parrot AR.Drone 2.0, se hará uso de un paquete en Gazebo implementado por el Grupo de Visión Artificial de la Universidad Técnica de Múnich denominado *tum_simulator* (ver Figura 3.4). En este punto, se verá la estructura de funcionamiento de dicho paquete.

Para desarrollo e implementación de los algoritmos de control, la conexión con el Parrot AR.Drone 2.0 se hace vía WiFi a través del driver *ardrone_autonomy* implementado en ROS (ver Figura 3.5). Este driver está basado en la versión oficial AR.Drone SDK 2.0 (https://projects.ardrone.org/projects/show/ardrone-api) y fue

elaborado por el Laboratorio de Autonomía de la Universidad Simon Fraser. Mayor información respecto al driver en la siguiente dirección de su repositorio: https://github.com/AutonomyLab/ardrone_autonomy.

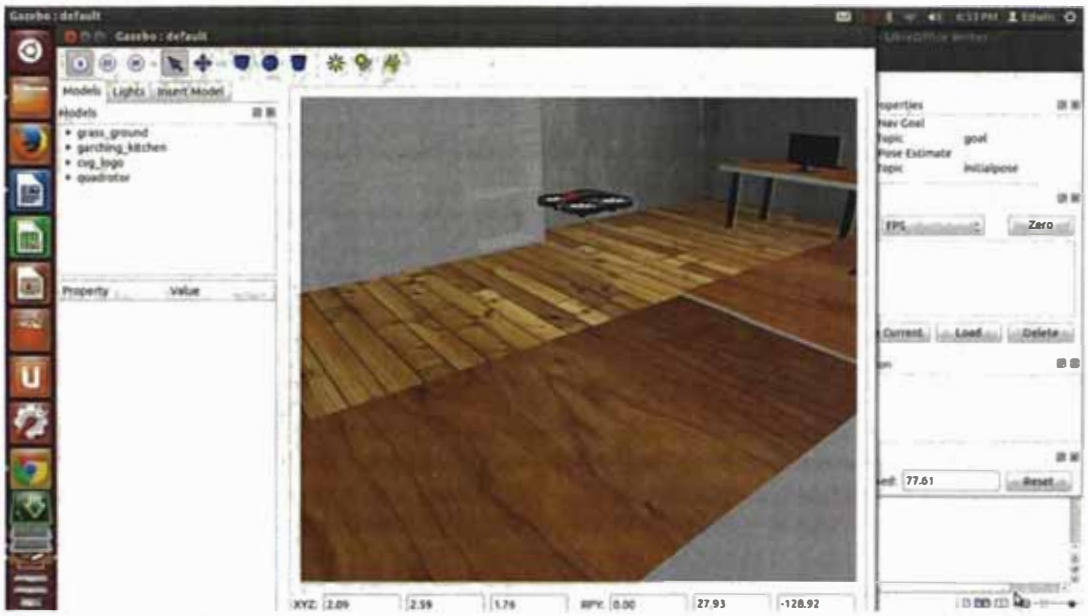


Figura 3.4: Simulador del Parrot AR.Drone en Gazebo.

Para la simulación en Gazebo, el paquete *tum_simulator* tiene como dependencia el driver *ardrone_autonomy*, por lo que los algoritmos programados en esta plataforma pueden ser trasladados directamente al Parrot AR.Drone mediante la conexión WiFi entre la computadora y el cuadricóptero.

El paquete *tum_simulator* fue descargado de http://wiki.ros.org/tum_simulator cuya estructura de funcionamiento se muestra en la Figura 3.6. Se puede observar la similitud con la estructura del driver *ardrone_autonomy* mostrado en la Figura 3.5, notándose que el paquete en Gazebo reemplaza (simula) la comunicación vía WiFi y el driver del cuadricóptero. Esto significa que los algoritmos programados en esta

plataforma pueden ser trasladados directamente al Parrot AR.Drone mediante el driver *ardrone_autonomy* vía WiFi.

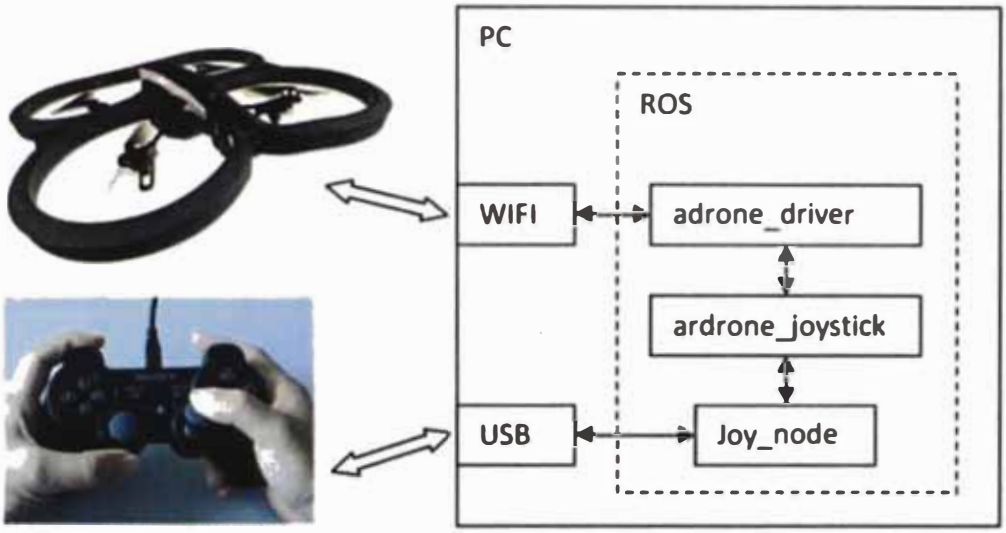


Figura 3.5: Diagrama de bloques de la operación del cuadricóptero mediante joystick a través del driver en ROS.

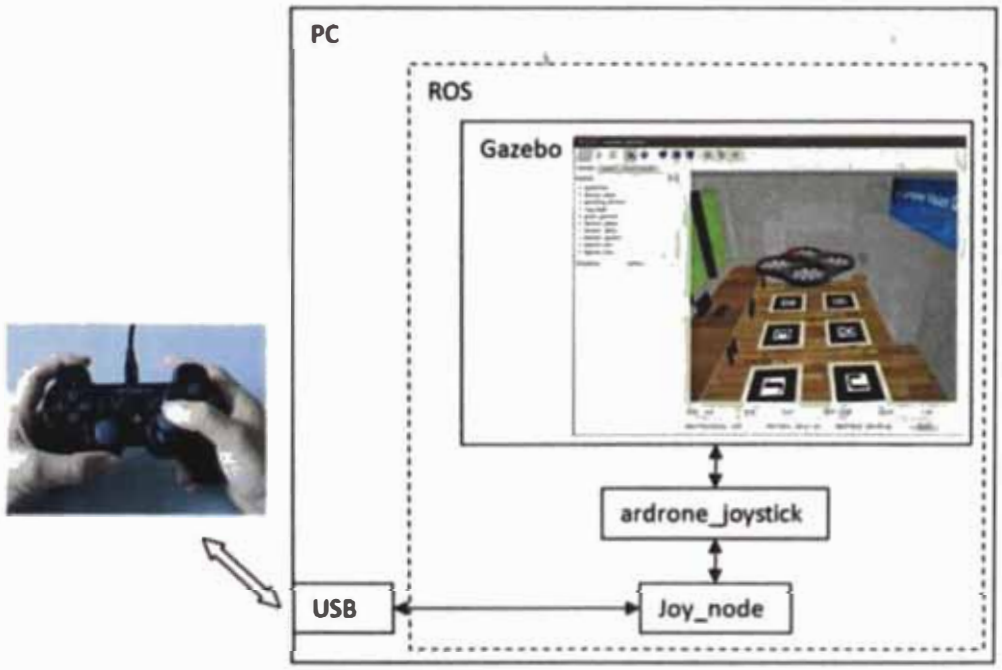


Figura 3.6: Diagrama de bloques de la operación del simulador en Gazebo mediante joystick.

Según la información mostrada en el repositorio del driver en ROS, la información de navegación del Parrot AR.Drone es entregada a través del tópico **/ardrone/navdata** como se muestra en la Figura 3.7, la cual contiene información del sistema a bordo y del cual es de interés las velocidades lineales obtenidas de su sistema interno de odometría visual, y de los ángulos de rotación, obtenidos de su sistema de medición inercial (ver Tabla 3.1).

Figura 3.7: Impresión en pantalla de los datos de navegación de simulador *tum_simulator* obtenidos mediante el tópico **/ardrone/navdata**.

Tabla 3.1: Información de navegación de interés del Parrot AR.Drone 2.0.

- vx, vy, vz: Linear velocity (mm/s)
- rotX: Left/right tilt in degrees (rotation about the X axis)
- rotY: Forward/backward tilt in degrees (rotation about the Y axis)
- rotZ: Orientation in degrees (rotation about the Z axis)

Se puede apreciar en la Figura 3.3 la lectura de los datos de navegación que nos entrega el simulador a través del tópico **/ardrone/navdata**, siendo estos procesados por el nodo */Code*, el cual contiene los algoritmos de control desarrollados, el que a su vez entrega los comandos de control al simulador a través del tópico **/cmd_vel**. Se describirá con mayor detalle la implementación de los algoritmos de control en el próximo capítulo.

3.2 Simulador JavaScript

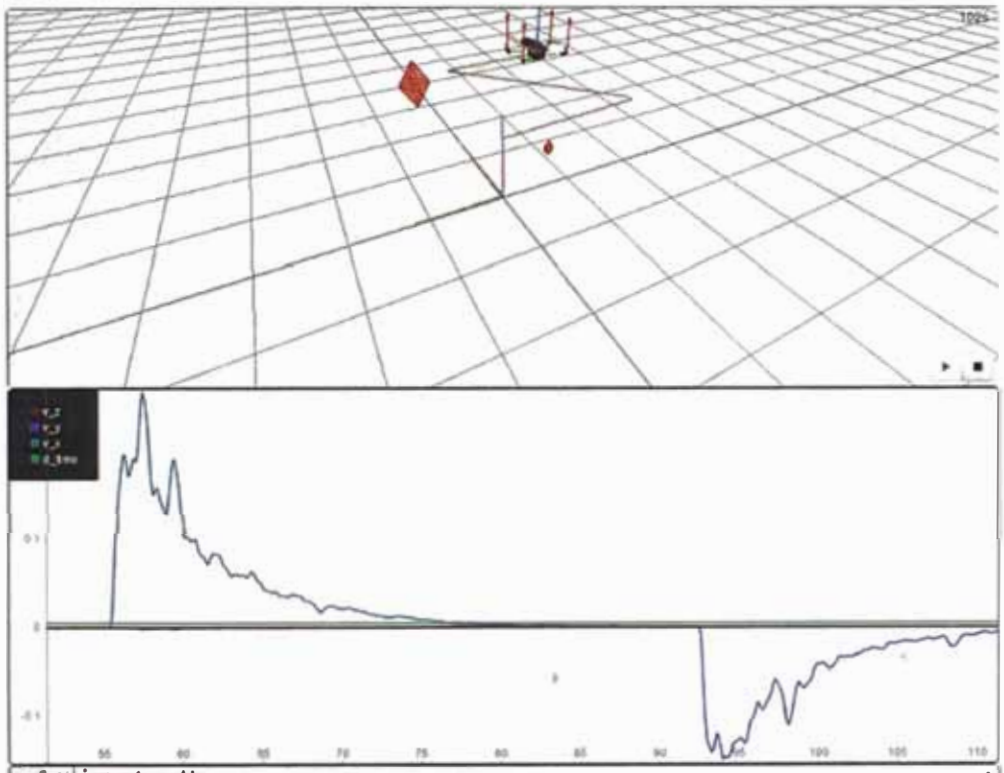
Para las diversas etapas de desarrollo de los algoritmos de control y localización, haremos uso también de un simulador programado en JavaScript desarrollado por el mismo Grupo de Visión Artificial de la Universidad Técnica de Múnich - TUM.

JavaScript es un lenguaje de programación orientado a objetos que se utiliza principalmente del lado del cliente, permitiendo crear efectos atractivos y dinámicos en páginas web. Los navegadores modernos interpretan el código JavaScript integrado en las páginas web.

La ventaja es que al estar alojado en el ordenador del usuario (cliente) los efectos creados por JavaScript son muy rápidos y dinámicos. Al ser un lenguaje de programación permite toda la potencia de la programación como uso de variables, condicionales, bucles, etc.

3.2.1 Principios del simulador

Las órdenes al cuadricóptero del simulador JavaScript mostrado en la Figura 3.8 pueden ser programadas mediante lenguaje Python.



```
1  import math
2  import plot
3  import time
4  from simulator.controller import MissionPlanner, PositionController
5
6  use_planned_mission = True
7
8
9  class InternalCode:
10     def __init__(self, simulator):
11         print("initializing simulation at %s" % time.time())
12
13         self.simulator = simulator
14
15         if use_planned_mission:
16             mission = MissionPlanner()
```

```
initializing simulation at 1424357880.54
updating setpoint, position: [[0 0 0.5]] yaw: 0.0
updating setpoint, position: [[0 -1 0.5]] yaw: 0.0
updating setpoint, position: [[0 -1 0.5]] yaw: 0.785398163397
updating setpoint, position: [[1.0099 -0.010029 0.5]] yaw: 0.785398163397
```

Figura 3.8: Simulador basado en JavaScript. En la parte superior se simula el vuelo del Parrot AR.Drone, en la parte central se grafican las velocidades lineales y en la parte inferior se muestra parte del algoritmo e impresiones en pantalla de variables.

El simulador está basado en la dinámica y datos de navegación del cuadricóptero comercial Parrot AR Drone 2.0, el cual nos brinda las velocidades relativas a su eje de coordenadas obtenidas de su sistema de odometría visual. Así mismo, nos brinda los ángulos de rotación relativos a sus ejes obtenidos de su unidad de medición inercial IMU.

El simulador, al igual que el driver del cuadricóptero entrega la información de navegación en la variable clase **navdata**, de la cual se obtienen las velocidades lineales y ángulos de rotación, la suscripción ver siguiente código:

```
1    def measurement_callback(self, t, dt, navdata):
2        rot = self.rotation_to_world(navdata.rotZ)
3        self.state.lin_velocity = np.dot(rot, np.array([[navdata.vx], [navdata.vy],
4        [navdata.vz]]))
```

La función **measurement callback** es una función interna del simulador la cual es llamada con la frecuencia de lectura del **navdata**, la cual es efectuada normalmente cada 5ms [8] p. 39. Los datos de interés de la clase **navdata** son las velocidades lineales y ángulos de rotación según fue mostrado en la Tabla 3.1. Estos datos son similares a las lecturas del tópico **/ardrone/navdata** del simulador en Gazebo; por ejemplo, para leer el ángulo de giro ψ, se refiere a la variable rotZ, llamándola de la siguiente manera: **navdata.rotZ**, como se observa en la línea 2 del código de arriba de igual manera las velocidades lineales *vx*, *vy* y *vz* son obtenidas de la clase **navdata** de la siguiente manera respectivamente: **navdata.vx**, **navdata.xy**, **navdata.vz**. La explicación del código será vista en el próximo capítulo.

Con esta información podemos desarrollar los algoritmos para el seguimiento de trayectorias y localización mediante estimación de estado haciendo uso de ambas plataformas de simulación para visualización y posterior implementación.

CAPITULO IV
SOLUCIÓN PROPUESTA

4.1 Modelo odométrico

La odometría es la predicción de la posición de un robot en el tiempo $t + 1$, conociendo su posición en el tiempo t y el movimiento ejecutado en la unidad de tiempo.

Para conocer el movimiento del robot durante dicha unidad de tiempo, se toma información de sus sensores de odometría, los cuales pueden ser encoders ópticos para el caso de vehículos con ruedas, o mediante una cámara como se da en el caso del cuadricóptero, a lo cual se le denomina odometría visual.

El cuadricóptero a través del tópico **/ardrone/navdata,** nos brinda información de su estado actual [8], la lectura de esta información será explicada en la sección 4.3. La información que nos brinda respecto a su estado de navegación son la velocidad de avance en el eje x v_x, la velocidad de avance en el eje y v_y, la velocidad de subida o bajada v_z y los ángulos de rotación respecto a sus ejes de rotación.

Con la información brindada, queremos conocer la posición actual de cuadricóptero, para lo cual debemos integrar la velocidad respecto al tiempo.

Tenemos las siguientes ecuaciones:

$$\dot{\boldsymbol{x}} = \frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{v} \tag{4.1}$$

$$\boldsymbol{x} = \int \boldsymbol{v}\, \partial t \tag{4.2}$$

Al ser la velocidad una variable discreta, podemos aproximar la derivada de la posición de la siguiente manera:

$$\frac{\partial \boldsymbol{x}}{\partial t} \cong \frac{\boldsymbol{x}_{t+1} - \boldsymbol{x}_t}{\Delta t} \tag{4.3}$$

Con lo cual obtenemos la siguiente aproximación que nos permite predecir la posición del robot en el tiempo $t + 1$ sabiendo la posición en el tiempo t y la velocidad en el instante de tiempo t:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \dot{\boldsymbol{x}}_t \Delta t \tag{4.4}$$

Como se observa en la Figura 4.1 y la Figura 4.2, las velocidades de avance del cuadricóptero están en el sistema de coordenadas del mismo, por lo que se requiere llevarlo un sistema de coordenadas global.

La transformación de coordenadas se realiza mediante una multiplicación con la matriz de rotación **R**.

La velocidad de avance del cuadricóptero:

$$\boldsymbol{v}_Q = \begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} \tag{4.5}$$

La transformación de sistemas de coordenadas se realiza mediante la matriz de rotación:

$$v_W = \mathbf{R} v_Q \tag{4.6}$$

$$\mathbf{R} = \begin{pmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{pmatrix} \tag{4.7}$$

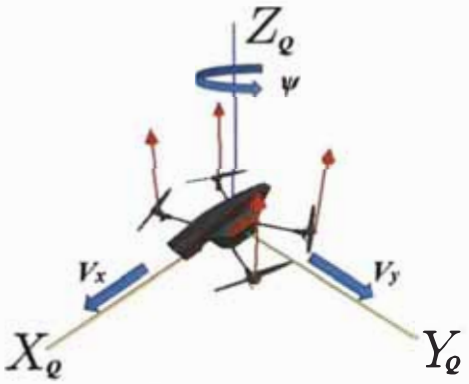


Figura 4.1: Ejes de coordenadas del cuadricóptero, donde se ilustra las velocidades de avance y el angulo de giro ψ.

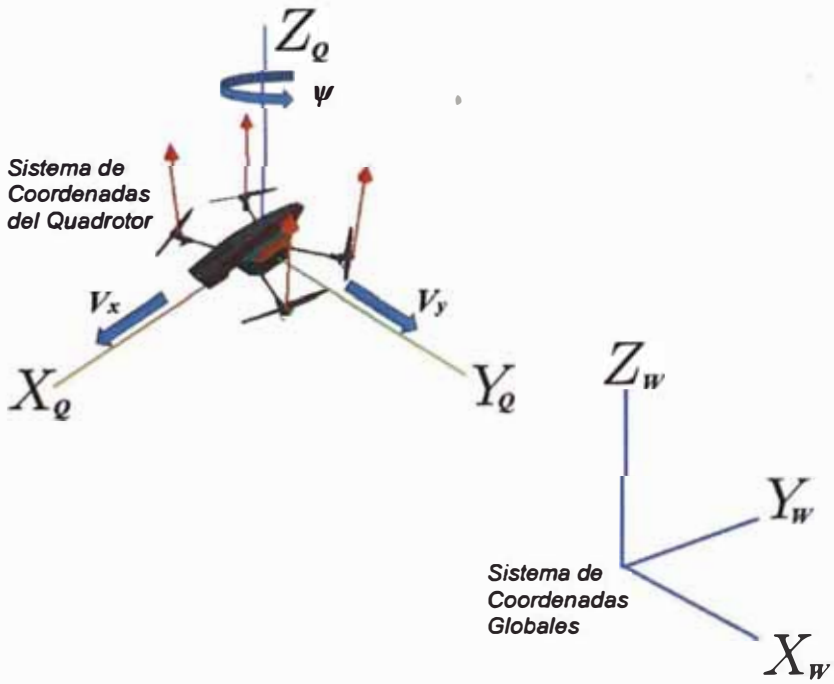


Figura 4.2: Sistema de coordenadas global: Las velocidades de avance que brinda el cuadricóptero están en un sistema de coordenadas relativo al mismo.

Reemplazamos la velocidad global en la ecuación (4.4), para obtener la predicción de la posición para el tiempo $t + 1$:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \mathbf{R}_t \boldsymbol{v}_Q \Delta t \tag{4.8}$$

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \end{pmatrix} + \begin{pmatrix} \cos(\psi_t) & -\sin(\psi_t) \\ \sin(\psi_t) & \cos(\psi_t) \end{pmatrix} \cdot \begin{pmatrix} \dot{x}_t \\ \dot{y}_t \end{pmatrix} \Delta t \tag{4.9}$$

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \end{pmatrix} = \begin{pmatrix} x_t + \Delta t(cos(\psi_t)\, \dot{x}_t - \sin(\psi_t)\dot{y}_t) \\ x_t + \Delta t(\sin(\psi_t)\, \dot{x}_t + \cos(\psi_t)\dot{y}_t) \end{pmatrix} \tag{4.10}$$

De igual manera, aplicamos la ecuación (4.4) para conocer el ángulo de orientación ψ para el tiempo $t + 1$:

$$\psi_{t+1} = \psi_t + \dot{\psi}_t \Delta t \tag{4.11}$$

De las ecuaciones (4.10) y (4.11), tenemos la predicción de la posición y orientación del cuadricóptero para el tiempo $t + 1$, a lo cual lo llamaremos modelo odométrico:

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \psi_{t+1} \end{pmatrix} = \begin{pmatrix} x_t + \Delta t(\cos(\psi_t)\, \dot{x}_t - \sin(\psi_t)\dot{y}_t) \\ y_t + \Delta t(\sin(\psi_t)\, \dot{x}_t + \cos(\psi_t)\dot{y}_t) \\ \psi_t + \Delta t \dot{\psi}_t \end{pmatrix} \tag{4.12}$$

El modelo odométrico desarrollado en las ecuaciones será utilizado como una estimación previa de la posición del cuadricóptero en la sección 4.4 como el *feedback* del controlador de trayectoria, y posteriormente con el filtro de Kalman para la corrección y una mejor estimación de la posición del cuadricóptero.

4.2 Visión monocular

Como se verá más adelante en el subcapítulo 4.5 y en los resultados del CAPÍTULO V, para la navegación del cuadricóptero, se tendrá que colocar marcadores (*landmarks*) en el suelo sobre la proyección de la trayectoria del cuadricóptero, que serán detectados a través de una cámara digital incorporada en el

robot, para así poder corregir desvíos en la trayectoria deseada del cuadricóptero, y localizarse mejor en su entorno.

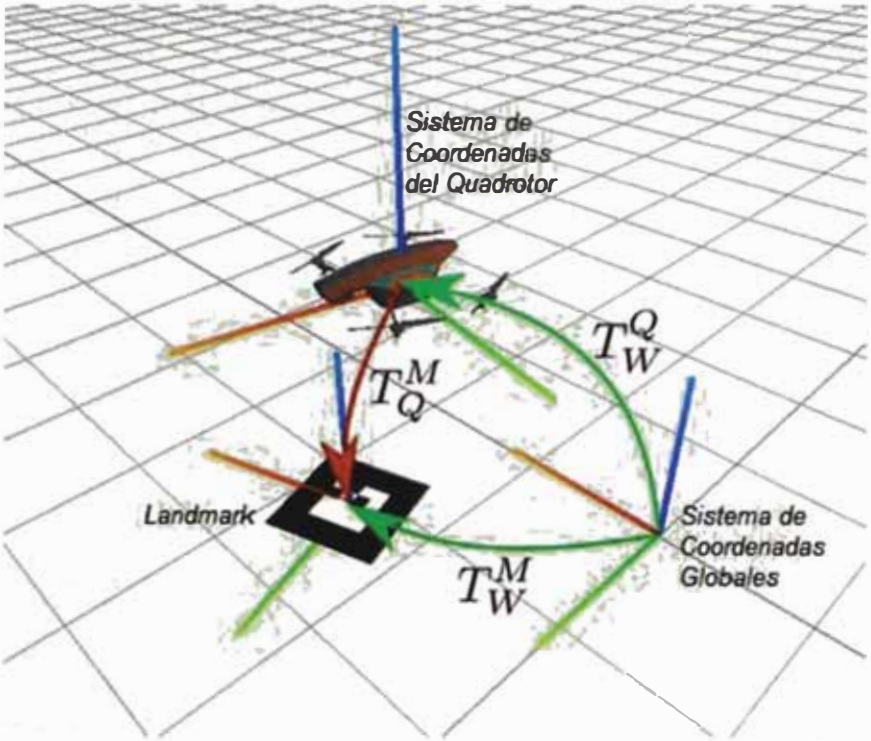


Figura 4.3: Transformaciones entre sistema de coordenadas.

Podemos calcular entonces la posición del *landmark* con respecto al cuadricóptero, conociendo la posición del *landmark* en el sistema de coordenadas global y la matriz de transformación del cuadricóptero (ver Figura 4.3). La posición del *landmark* calculada, relativa al cuadricóptero, será posteriormente comparada con la posición observada por la cámara del cuadricóptero, lo que servirá para la corrección de la posición estimada del cuadricóptero mediante un filtro de Kalman (Sección 2.3.6).

En la Figura 4.3, tenemos:

- T_W^Q es la posición del cuadricóptero que puede expresarse en una matriz de transformación en coordenadas homogéneas.

- T_W^M es la posición del *landmark* ya definida en el sistema de coordenadas global
- T_Q^M es la posición relativa del *landmark*, que puede ser observada por la cámara inferior del cuadricóptero.

De la ecuación (2.10), se obtiene la posición del *landmark* respecto al cuadricóptero:

$$T_Q^M = T_Q^W T_W^M \tag{4.13}$$

Donde:

$$T_Q^W = \left(T_W^Q\right)^{-1} \tag{4.14}$$

$$T_W^Q = \begin{pmatrix} \mathbf{R}_t & \mathbf{t}_t \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \tag{4.15}$$

$$T_W^Q = \begin{pmatrix} \cos(\psi_t) & -\sin(\psi_t) & x_t \\ \sin(\psi_t) & \cos(\psi_t) & y_t \\ 0 & 0 & 1 \end{pmatrix} \tag{4.16}$$

De la ecuación (2.11), tenemos:

$$T_Q^W = \left(T_W^Q\right)^{-1} = \begin{pmatrix} \mathbf{R}_t & \mathbf{t}_t \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix}^{-1} \tag{4.17}$$

$$T_Q^W = \begin{pmatrix} \mathbf{R}_t{}^{\mathrm{T}} & -\mathbf{R}_t{}^{\mathrm{T}}\mathbf{t}_t \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \tag{4.18}$$

$$T_Q^W = \begin{pmatrix} \cos(\psi_t) & \sin(\psi_t) & -x_t\cos(\psi_t) - y_t\sin(\psi_t) \\ -\sin(\psi_t) & \cos(\psi_t) & x_t\sin(\psi_t) - y_t\cos(\psi_t) \\ 0 & 0 & 1 \end{pmatrix} \tag{4.19}$$

Se conoce la posición del *landmark* en el sistema global representado por:

$$T_W^M = \begin{pmatrix} x_W^M \\ y_W^M \end{pmatrix} \tag{4.20}$$

En coordenadas homogéneas:

$$T_W^M = \begin{pmatrix} x_W^M \\ y_W^M \\ 1 \end{pmatrix} \tag{4.21}$$

Reemplazando las ecuaciones (4.19) y (4.21) en la ecuación (4.13), se tiene que:

$$T_Q^M = \begin{pmatrix} \cos(\psi_t) & \sin(\psi_t) & -x_t\cos(\psi_t) - y_t\sin(\psi_t) \\ -\sin(\psi_t) & \cos(\psi_t) & x_t\sin(\psi_t) - y_t\cos(\psi_t) \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_W^M \\ y_W^M \\ 1 \end{pmatrix} \tag{4.22}$$

$$T_Q^M = \begin{pmatrix} (x_W^M - x_t)\cos(\psi_t) + (y_W^M - y_t)\sin(\psi_t) \\ -(x_W^M - x_t)\sin(\psi_t) + (y_W^M - y_t)\cos(\psi_t) \\ 1 \end{pmatrix} \tag{4.23}$$

En coordenadas cartesianas:

$$T_Q^M = \begin{pmatrix} (x_W^M - x_t)\cos(\psi_t) + (y_W^M - y_t)\sin(\psi_t) \\ -(x_W^M - x_t)\sin(\psi_t) + (y_W^M - y_t)\cos(\psi_t) \end{pmatrix} \tag{4.24}$$

La orientación relativa del *landmark* con respecto al cuadricóptero, es simplemente la diferencia de sus ángulos:

$$\psi_M^Q = \psi_W^M - \psi_t \tag{4.25}$$

Las ecuaciones (4.24) y (4.25) serán implementadas en simulación en la sección 4.5.2 en la etapa de corrección del filtro de Kalman.

4.3 Comandos de vuelo

Como se encuentra descrito en la referencia bibliográfica [8] p. 17, el sistema de control del cuadricóptero ofrece un control de vuelo en base a comandos de vuelo los cuales son las velocidades lineales en los ejes *x, y* y *z*, y la velocidad de giro (*yaw*) mostrados en la Tabla 4.1. Asimismo comandos tipo booleano para el despegue y aterrizaje (*take off* & *land flags*).

Tabla 4.1: Entradas de control del cuadricóptero AR Drone Parrot 2.0.

-linear.x: move backward
+linear.x: move forward
-linear.y: move right
+linear.y: move left
-linear.z: move down
+linear.z: move up
-angular.z: turn left
+angular.z: turn right

De manera similar a como los datos de navegación de cuadricóptero son obtenidos del tópico **/ardrone/navdata** mediante la suscripción al mismo, los comandos de velocidad son publicados al tópico **/cmd_vel** en los que se encuentran los comandos mostrados en la Tabla 4.1. Los valores de entrada a estos comandos se encuentran definidos en el rango de [-1, 1].

4.4 Control de trayectoria

Para el seguimiento de la trayectoria deseada, se ha de diseñar un controlador con retroalimentación de estado. Este controlador tendrá entonces como entradas al valor de referencia del estado deseado, y al estado actual estimado, el cual solo ha sido obtenido por el momento del modelo odométrico del cuadricóptero.

En el diseño del controlador de trayectoria del cuadricóptero, se debe tener presentes los siguientes puntos:

- Debe ser tolerante a **perturbaciones** del entorno como el viento u alguna acción humana.
- Debe ser tolerante al **ruido** de los sensores de odometría visual, siendo la cámara inferior del cuadricóptero el sensor principal. Nota: más adelante se verá que la posición obtenida por odometría visual es muy imprecisa y conlleva a grandes

errores en la estimación de la posición, por lo que se incluirá información de observaciones mediante un filtro de Kalman.

- Debe permitir un avance **estable** con pocas oscilaciones durante todo el tramo de la trayectoria deseada.

Todas estas características importantes, las podemos obtener mediante un controlador proporcional integral derivativo PID.

4.4.1 Controlador PID

El controlador PID es una extensión más desarrollada al clásico control ON/OFF de muchos procesos. Es un controlador de lazo cerrado y es el más usado dentro de la teoría de control clásica. Como su nombre lo indica, combina tres acciones importantes: proporcional, integral y derivativo.

La representación matemática del controlador PID, para una variable continua, esta expresado de la siguiente manera:

$$u_t = K_p e(t) + K_i \int e(t)\partial t + K_d \frac{\partial e(t)}{\partial t} \tag{4.26}$$

Donde K_p, K_i y K_d son los parámetros de la parte proporcional, integral y derivativa del controlador respectivamente.

Para una variable discreta, la expresión matemática del controlador PID es la siguiente:

$$u(k) = K_p e(k) + K_i \sum_{i=0}^{k} e(i)\Delta t + K_d \frac{e(k) - e(k-1)}{\Delta t} \tag{4.27}$$

Para nuestro caso, el esquema de control de lazo cerrado se muestra en la Figura 4.4.

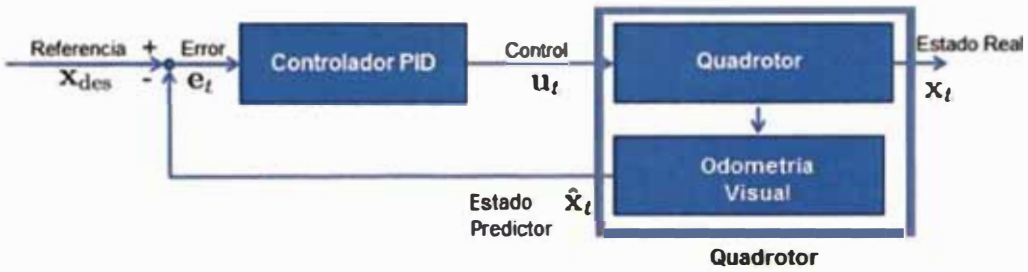


Figura 4.4: Control de lazo cerrado.

Se optará por descartar la parte integral del controlador, por lo que se trabajará con la parte proporcional y derivativa, siendo denominado a este controlador como controlador PD (ver Figura 4.5).

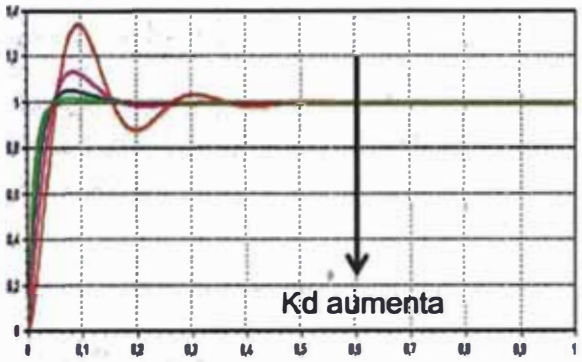


Figura 4.5: Control PD. Conforme Kd aumenta se consigue una mejor amortiguación de la respuesta de control.

Las propiedades del controlador PD son las siguientes:

- Mejora la estabilidad
- Disminuye las oscilaciones
- Aumenta la velocidad de respuesta

El controlador PD está representado de la siguiente manera:

$$u(k) = K_p e(k) + K_d \dot{e}(k) \tag{4.28}$$

$$u(k) = K_p\big(x_d(k) - \hat{x}(k)\big) + K_d\big(\dot{x}_d(k) - \hat{\dot{x}}(k)\big) \tag{4.29}$$

$$u_t = K_p\big(x_{d_t} - x_t\big) + K_d\big(\dot{x}_{d_t} - \dot{x}_t\big) \tag{4.30}$$

Siendo x_{d_t} y $\dot{x}_{d_t}$ la posición y velocidad deseada para el instante t, así como x_t la posición estimada obtenida en la ecuación (4.12).

4.4.2 Programación con el simulador de JavaScript

En el concepto de seguimiento de trayectoria, se ha optado por un control de trayectoria parametrizado, y se ha escogido una trayectoria de forma circular, ecuación mostrada anteriormente en la Figura 2.7, que es programada de la siguiente manera:

```
1    def update_setpoint(self, t):
2        self.radio = 1
3        self.state_desired.position[0] = self.radio*math.sin(math.pi / 10.0 * t)
4        self.state_desired.position[1] = self.radio*math.cos(math.pi / 10.0 * t)
5        self.state_desired.position[2] = 1.5
```

La variable state_desired.position de tipo array contiene la posición deseada en metros en los ejes *x*, *y* y *z*. Se observa que para este caso el radio de la circunferencia es de un metro, el periodo es de 20 segundos y la altura deseada es de 1.5 metros. En la Figura 4.6 se puede observar la trayectoria circular programada ejecutada en el simulador de JavaScript. La línea celeste representa la trayectoria deseada programada líneas arriba. El error $e(t)$ del controlador (ver ecuación (4.26)) será la diferencia de la posición deseada en el tiempo t programada anteriormente y la posición actual estimada, por el momento adquirida solo del modelo odométrico del cuadricóptero. Para efectos de simulación y prueba del controlador PD, se reconoce a la posición obtenida por la odometría como la posición estimada del cuadricóptero. En la Figura 4.6 la línea azul

representa la posición obtenida por odometría, mientras la línea morada representa la posición real del simulador.

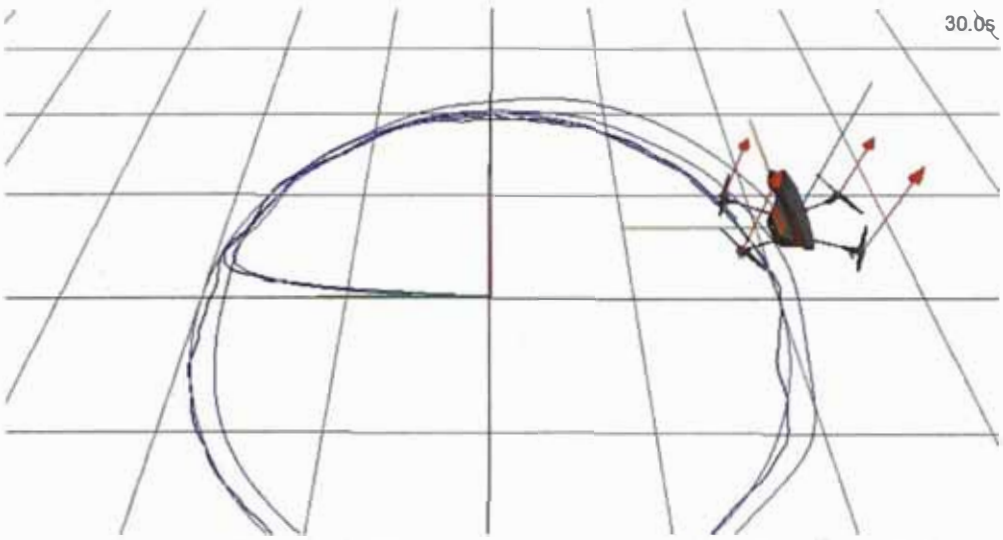


Figura 4.6: Control de trayectoria (pruebas en simulador de Javascript).

La estimación de posición la obtenemos de las ecuaciones de odometría desarrolladas en el subcapítulo 4.1. Su programación se muestra a continuación:

```
1    def measurement_callback(self, t, dt, navdata):
2        rot = self.rotation_to_world(navdata.rotZ)
3        updated_state.velocity = np.dot(rot, np.array([[navdata.vx], [navdata.vy],
4        [navdata.vz]]))
5        updated_state.position += dt * updated_state.velocity
6        self.state.append(updated_state);
```

En la línea 2 se obtiene la matriz de rotación a través de la función **rotation_to_world**. De la línea 3 a la 5, se programa la odometría para la estimación de las posiciones en las coordenadas globales x, y y z, según la ecuación (4.9).

La función **rotation_to_world** se define de acuerdo a la ecuación (4.7):

```
1  def rotation_to_world(self, yaw):
2      return np.array([[cos(yaw), -sin(yaw), 0], [sin(yaw), cos(yaw), 0], [0, 0, 1]])
```

La función **measurement_callback** es llamada internamente por el simulador en cuanto una lectura de la clase **navdata** es efectuada.

La programación del controlador se muestra a continuación, los parámetros proporcionales y derivativos del controlador han sido seteados de manera experimental y logran un adecuado control de la posición deseada según se observa en la Figura 4.6.

```
1    def __init__(self):
2        Kp_xy = 1.7
3        Kp_z = 0.5
4        Kd_xy = 0.8
5        Kd  z = 0.25
6        self.Kp = np.array([[Kp_xy, Kp_xy, Kp_z]]).T
7        self.Kd = np.array([[Kd_xy, Kd_xy, Kd_z]]).T

8    def compute_control_command(self, t, dt, state, state_desired):
9        self.plot(state.position, state_desired.position)
10       u = self.Kp * (state_desired.position - state.position) + self.Kd *
11       (state_desired.velocity - state.velocity)
12       return u
```

De la ecuación (4.30) se escriben las líneas 10 y 11. Con la función plot de la línea 9 se grafica la posición estimada y la posición deseada mostrados en la Figura 4.7.

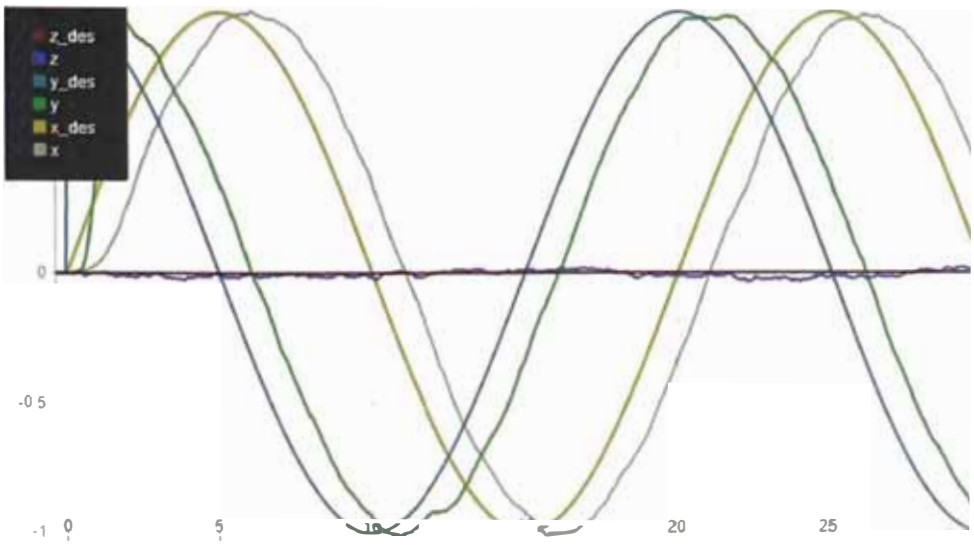


Figura 4.7: Respuesta del controlador de trayectoria.

Los parámetros del controlador son definidos en **__init__(self)**. Los valores de estos parámetros fueron sintonizados partiendo de valores referenciales cercanos a la unidad, variándolos de a pocos en múltiples intentos, hasta llegar a los valores mostrados arriba que logran una respuesta de control es rápida, sin oscilaciones y estable, según se observa en la Figura 4.7.

Finalmente dentro de la función **measurement callback** se añade la salida del controlador como comandos de entrada al simulador. Mayor detalle de la función **set_input_world** de la línea 4 será vista con el simulador de Gazebo:

```
1    def measurement_callback(self, t, dt, navdata):
2        lin_vel = self.user.compute_control_command(t, dt, current,
3        self.copy_state(self.state_desired))
4        self.simulator.set_input_world(lin_vel, 0)
```

Como fue mencionado, se presentó los valores de los parámetros del controlador PID que logran un buen control de trayectoria. A continuación describimos las respuestas obtenidas variando estos valores en unos cuantos decimales.

Si el parámetro derivativo disminuye, se consigue una respuesta sub-amortiguada como es mostrado en las Figura 4.8 y Figura 4.9, lo cual no es deseado.

En la Figura 4.9 se puede observar un pico en la respuesta de la posición en el eje y debido al bajo valor del parámetro derivativo, esto se traduce en como una ligera desviación del vehículo al intentar alcanzar la posición deseada (ver Figura 4.8).

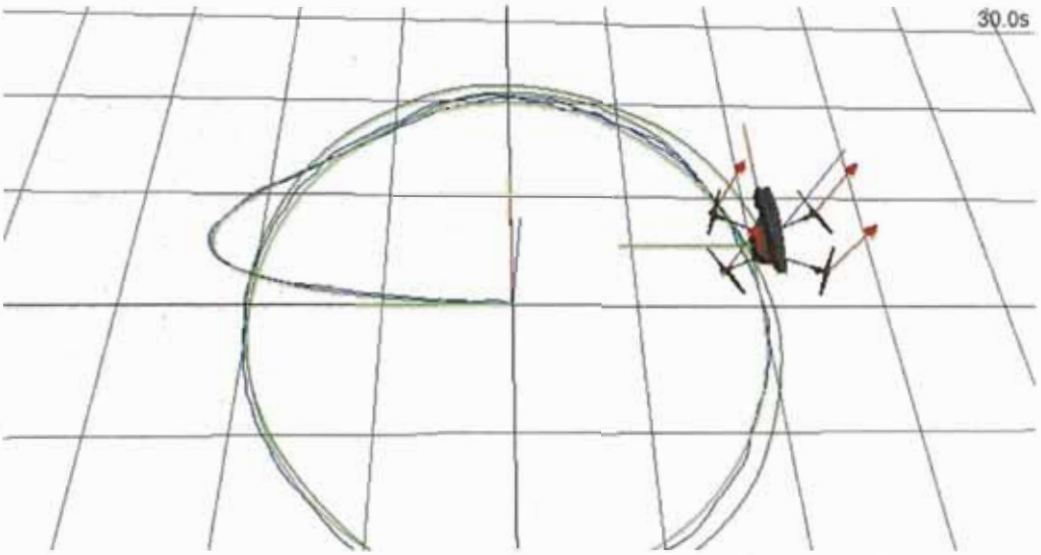


Figura 4.8: Control de trayectoria con respuesta sub-amortiguada.

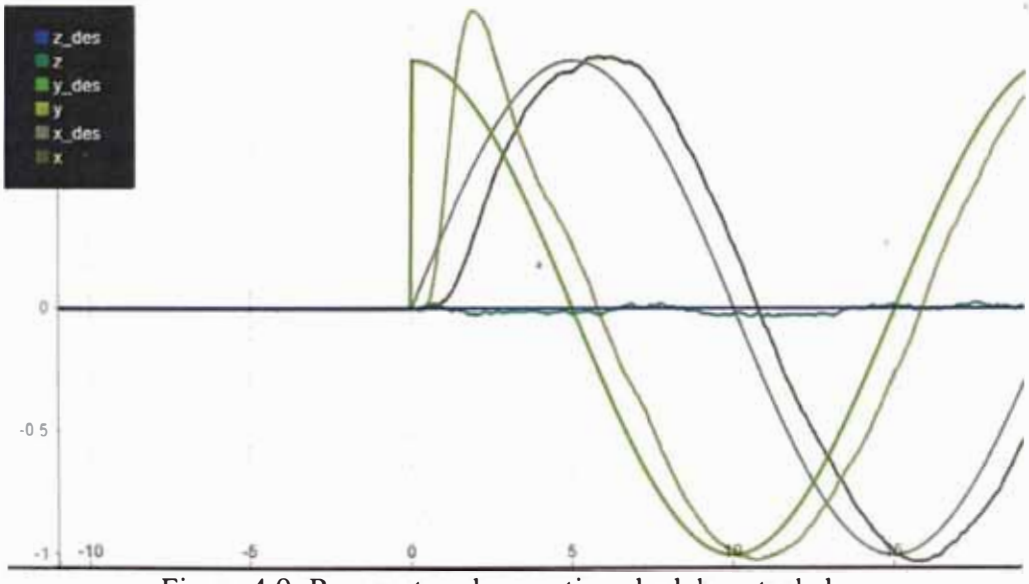


Figura 4.9: Respuesta sub-amortiguada del controlador.

Asimismo, si se aumenta el parámetro derivativo, la respuesta del controlador oscila bastante según se observa en la Figura 4.10a. Este comportamiento es muy similar cuando se aumenta el parámetro proporcional, obteniendo una respuesta con muchos transitorios (ver Figura 4.10b), la diferencia es que en la Figura 4.10a el sistema es sobre-amortiguado a diferencia de la Figura 4.10b que el sistema es sub-amortiguado.

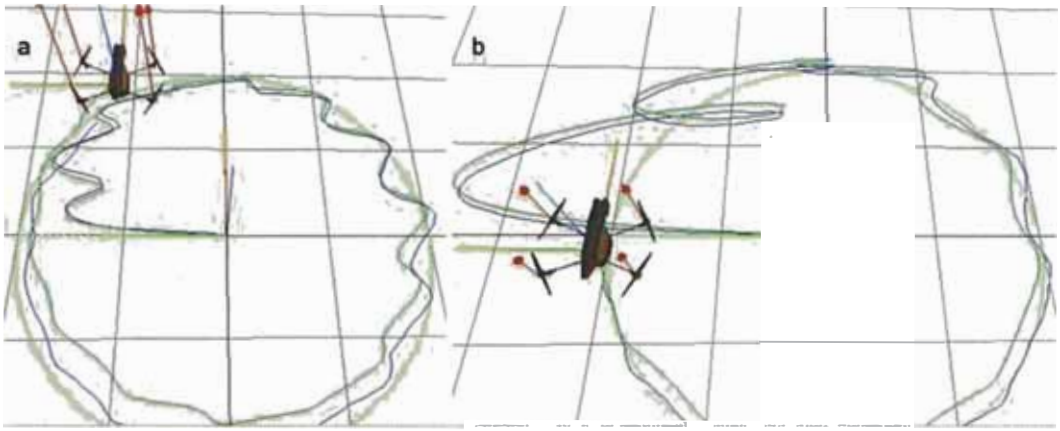


Figura 4.10: Respuesta con oscilaciones.

4.4.3 Programación con el simulador de Gazebo (*tum_simulator*)

Un paso previo a la implementación del controlador de trayectorias en el Parrot AR.Drone, es la programación en ROS haciendo uso del simulador en Gazebo *tum_simulator* (ver Figura 4.11).

Similar a la programación realizada con el simulador JavaScript, en esta nueva plataforma se va a definir la función **ReceiveNavdata** la cual va a ser suscrita al tópico **/ardrone/navdata** del simulador, el cual es el mismo tópico enviado por el Parrot AR.Drone vía conexión WiFi [8]:

```
1      self.subNavdata =
2  rospy.Subscriber('/ardrone/navdata',Navdata,self.ReceiveNavdata)
```

La lectura del tópico **/ardrone/navdata** es realizada, con la frecuencia de aproximadamente 200Hz [8] p. 39.

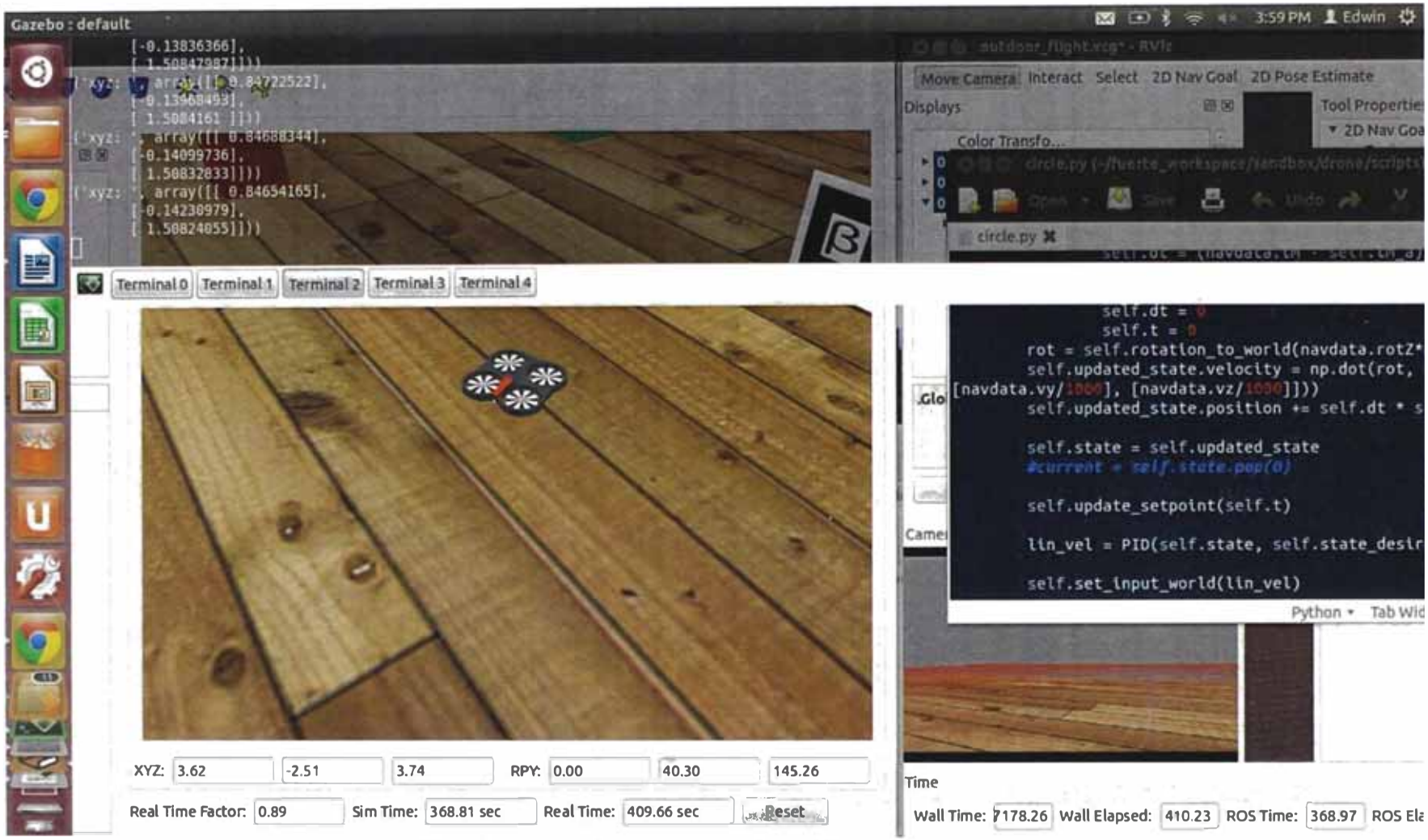


Figura 4.11: Simulación del controlador PD utilizando el *tum_simulator*.

La programación de la odometría y el controlador de trayectoria es muy similar al del simulador de JavaScript. La función **set_input_world** envía los comandos de control al tópico **/cmd_vel** como se muestra a continuación:

```
1    def set_input_world(self, lin_vel):
2        twist = Twist()
3        self.cmd = lin_vel.compute_control_command()
4        twist.linear.x = self.cmd[0,0]
5        twist.linear.y = self.cmd[1,0]
6        twist.linear.z = self.cmd[2,0]
7        self.pub.publish(twist)
```

La función lin_vel.compute_control_command contiene la salida del controlador, que son los comandos de control que van como entrada al cuadricóptero.

Los resultados obtenidos con este simulador son muy similares a los presentados en el punto anterior donde se utilizó el simulador JavaScript.

Hasta el momento se ha obtenido comando de control para las velocidades lineales del cuadricóptero, sin variar su ángulo de giro ψ. Se puede añadir un controlador adicional para este ángulo de la siguiente manera:

```
1    def ReceiveNavdata(self,navdata):
2        lin_vel = PD_lin(self.state, self.state_desired)
3        ang_vel = PD_ang(self.state, self.state_desired)
4        self.set_input_world(lin_vel, ang_vel)
```

Se observa que la función **set_input_world** ahora tiene dos argumentos de entrada, lin_vel con los comandos de control para las velocidades lineales, y ang_vel con los comandos de control para las velocides angulares del cuadricóptero.

A la función **set_input_world** se le agrega el segundo argumento según se muestra en las líneas 4 y 8 del siguiente código:

```
1    def set_input_world(self, lin_vel, ang_vel):
2        twist = Twist()
3        self.cmd = lin_vel.compute_control_command()
4        self.cmd_ang = ang_vel.compute_control_command()
5        twist.linear.x = self.cmd[0,0]
6        twist.linear.y = self.cmd[1,0]
7        twist.linear.z = self.cmd[2,0]
8        twist.angular.z = self.cmd_ang
9        self.pub.publish(twist)
```

Las clases PD_lin y PD_ang contienen a la función **compute_control_command** visto anteriormente.

Estos algoritmos forman parte del nodo /Code visto en la Figura 3.3. Este nodo fue creado mediante programación en la multiplataforma ROS conteniendo las líneas de código expuestas en el presente capítulo. Un diagrama de flujo que representa mejor el contenido del nodo se puede observar en la Figura 4.12. Como se puede observar, el contenido del nodo /Code se basa en el diagrama mostrado en la Figura 4.4.

Los algoritmos desarrollados en esta plataforma se pueden trasladar directamente al cuadricóptero, vía conexión WiFi.

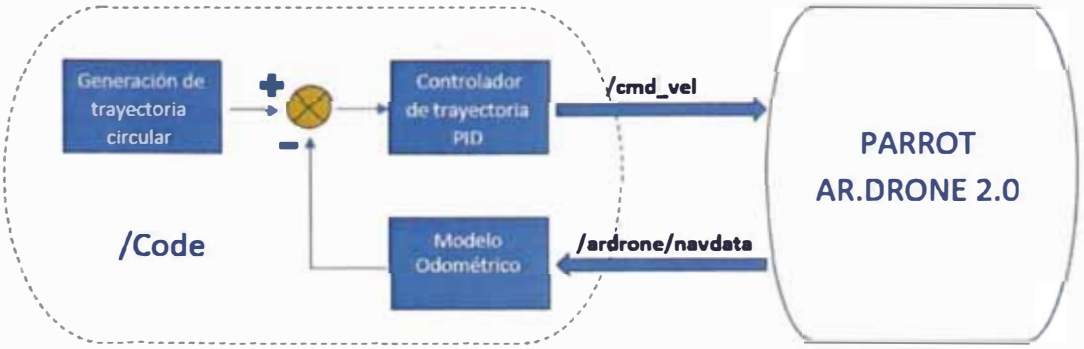


Figura 4.12: Diagrama de flujos representando el contenido del nodo /Code.

4.5 Localización del cuadricóptero

En la presente sección, se va a desarrollar el algoritmo del filtro de Kalman Extendido con los modelos del cuadricóptero desarrollados en el CAPÍTULO II.

El estado estimado a la salida del filtro de Kalman será el *feedback* de la posición deseada de referencia, con el que se determina el error de posición que será la entrada para nuestro controlador de trayectoria PID desarrollado en el capítulo de Control de Trayectoria, sección 4.4 (ver Figura 4.13).

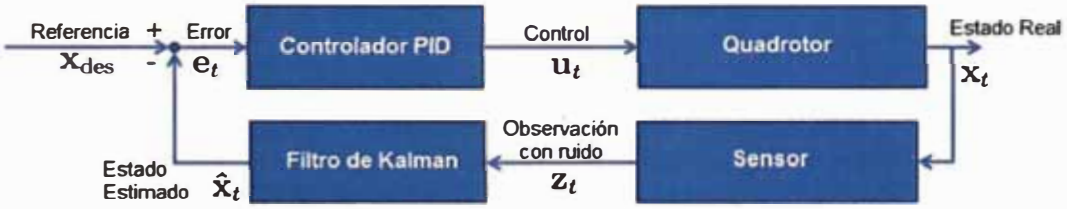


Figura 4.13: Diagrama de bloques para el control de posición del cuadricóptero mediante estimación de estado aplicando filtro de Kalman.

La función de estado del cuadricóptero contiene como elementos la posición x y y en las coordenadas globales, y la orientación ψ.

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ \psi \end{pmatrix} \tag{4.31}$$

Como fue descrito en la odometría del cuadricóptero (ver sección 4.1), el modelo matemático del movimiento del cuadricóptero es:

$$f(\mathbf{x}_t, \mathbf{u}_t) = \begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \psi_{t+1} \end{pmatrix} = \begin{pmatrix} x_t + \Delta t(\cos(\psi_t)\,\dot{x}_t - \sin(\psi_t)\dot{y}_t) \\ y_t + \Delta t(\sin(\psi_t)\,\dot{x}_t + \cos(\psi_t)\dot{y}_t) \\ \psi_t + \Delta t\dot{\psi}_t \end{pmatrix} \tag{4.32}$$

Para la implementación de filtro de Kalman Extendido, se necesita la matriz Jacobiana de la función f tal como fue definido en la ecuación (2.65). Se obtiene lo siguiente:

$$\mathbf{F} = \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\Big|_{\mathbf{x}_t} = \begin{pmatrix} 1 & 0 & \Delta t(-\sin(\psi_t)\dot{x}_t - \cos(\psi_t)\,\dot{y}_t) \\ 0 & 1 & \Delta t(\cos(\psi_t)\,\dot{x}_t - \sin(\psi_t)\dot{y}_t) \\ 0 & 0 & 1 \end{pmatrix} \tag{4.33}$$

La observación del *landmark* a través de la cámara del cuadricóptero contiene como elementos la posición relativa x_m y y_m en las coordenadas del cuadricóptero, y la orientación relativa ψ_m.

$$\mathbf{z} = \begin{pmatrix} x_m \\ y_m \\ \psi_m \end{pmatrix} \tag{4.34}$$

La ecuación del modelo de observación de la cámara del cuadricóptero $\mathbf{z} = h(\mathbf{x}_t)$ fue obtenida en la sección 4.2:

$$h(\mathbf{x}_t) = \begin{pmatrix} \cos(\psi_t).(x_W^M - x_t) + \sin(\psi_t).(y_W^M - y_t) \\ -\sin(\psi_t).(x_W^M - x_t) + \cos(\psi_t).(y_W^M - y_t) \\ \psi_W^M - \psi_t \end{pmatrix} \tag{4.35}$$

Para la implementación de filtro de Kalman Extendido, se necesita de igual manera la matriz Jacobiana de la ecuación de observación h tal como se definió en la ecuación (2.68). Se obtiene lo siguiente:

$$\mathbf{H} = \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}_t}$$

$$= \begin{pmatrix} -\cos(\psi_t) & -\sin(\psi_t) & -\sin(\psi_t).(x_m - x_t) + \cos(\psi_t).(y_m - y_t) \\ \sin(\psi_t) & -\cos(\psi_t) & -\cos(\psi_t).(x_m - x_t) - \sin(\psi_t).(y_m - y_t) \\ 0 & 0 & -1 \end{pmatrix} \tag{4.36}$$

4.5.1 Inicialización

Se define el ruido en el proceso de la odometría visual y en las observaciones de la cámara a través de las covarianzas **Q** y **R** respectivamente. Siendo el estado $\mathbf{x}_t$ y

las observaciones $\mathbf{z}_t \in R^3$, sus covarianzas pertenecen a la dimensión R^{3x3}. Asimismo se espera que el ruido sea constante e invariable en el tiempo:

$$\mathbf{Q} = \begin{pmatrix} {\epsilon_x}^2 & 0 & 0 \\ 0 & {\epsilon_y}^2 & 0 \\ 0 & 0 & {\epsilon_\psi}^2 \end{pmatrix} \tag{4.37}$$

$$\mathbf{R} = \begin{pmatrix} {\delta_x}^2 & 0 & 0 \\ 0 & {\delta_y}^2 & 0 \\ 0 & 0 & {\delta_\psi}^2 \end{pmatrix} \tag{4.38}$$

De igual manera se inicializa con un valor determinado cercano o igual a cero a la covarianza del filtro de Kalman:

$$\mathbf{\Sigma} = \alpha \mathbf{I}^{3x3} \tag{4.39}$$

Donde α es una constante e **I** es la matriz de identidad.

4.5.2 Programación

A continuación se muestra la programación del filtro de Kalman extendido, las líneas de código que se presentan a continuación son la transcripción en lenguaje Python de las ecuaciones del algoritmo, las que se dan referencia en el comentario previo que inicia con el símbolo numeral # o entre los símbolos ''' como encabezado al iniciar el código.

- **Inicialización**

```
1    def __init__(self):
       # ruido en la odometría, ver ecuación (4.37):
2      pos_noise_std = 0.005
3      yaw_noise_std = 0.005
4      self.Q = np.array([
5        [pos_noise_std*pos_noise_std,0,0],
6        [0,pos_noise_std*pos_noise_std,0],
7        [0,0,yaw_noise_std*yaw_noise_std]
```

```
8       ])
        # ruido en las observaciones, ver ecuación (4.38):
9       z_pos_noise_std = 0.005
10      z_yaw_noise_std = 0.03
11      self.R = np.array([
12        [z_pos_noise_std*z_pos_noise_std,0,0],
13        [0,z_pos_noise_std*z_pos_noise_std,0],
14        [0,0,z_yaw_noise_std*z_yaw_noise_std]
15      ])
        # vector de estado [x, y, yaw] en coordenadas globales (ecuación (4.34)):
16      self.x = np.zeros((3,1))
        # matriz 3x3 de covarianza de estado (ecuación (4.37)):
17      self.alpha = 0.01
18      self.sigma = self.alpha * np.identity(3)
```

- **EKF – predicción**

```
1     def predictState(self, dt, x, u_linear_velocity, u_yaw_velocity):
        '''
          Odometría: Predice el estado futuro usando el estado actual y las
          entradas de control (velocidades lineales ángulo de giro yaw), ver
          ecuación (4.12). La función rotation es la matriz de rotación
          previamente definida según la ecuación (4.7).
        '''
2       x_p = np.zeros((3, 1))
3       x_p[0:2] = x[0:2] + dt * np.dot(self.rotation(x[2]), u_linear_velocity)
4       x_p[2]   = x[2]   + dt * u_yaw_velocity
5       x_p[2]   = self.normalizeYaw(x_p[2])
6       return x_p

7     def calculatePredictStateJacobian(self, dt, x, u_linear_velocity,
8     u_yaw_velocity):
        '''
        Calcula la matriz Jacobiana del modelo odométrico, ver ecuación (4.33):
        '''
9       s_yaw = math.sin(x[2])
10      c_yaw = math.cos(x[2])
11      dRotation_dYaw = np.array([
12        [-s_yaw, -c_yaw],
13        [ c_yaw, -s_yaw]
14      ])
15      F = np.identity(3)
16      F[0:2, 2] = dt * np.dot(dRotation_dYaw, u_linear_velocity)
17      return F
```

```
18    def predictCovariance(self, sigma, F, Q):
        #Predice la covarianza de acuerdo a la ecuación (2.71):
19      return np.dot(F, np.dot(sigma, F.T)) + Q
```

- **EKF – corrección**

```
1     def predictMeasurement(self, x, marker_position_world,
2     marker_yaw_world):
        '''
        Predice la posición del landmark respecto al cuadricóptero, ver ecuación
        (4.35):
        '''
3       x_w = marker_position_world[0]
4       y_w = marker_position_world[1]
5       s_yaw = math.sin(x[2])
6       c_yaw = math.cos(x[2])
7       z_predicted = np.array([
8         [c_yaw*(x_w-x[0])+s_yaw*(y_w-x[1])],
9         [-s_yaw*(x_w-x[0])+c_yaw*(y_w-x[1])],
10        [marker_yaw_world-x[2]]
11      ])
12      return z_predicted

13    def calculatePredictMeasurementJacobian(self, x,
14    marker_position_world, marker_yaw_world):
        #Matriz Jacobiana del Modelo de observación, ver ecuación (4.36):
15      s_yaw = math.sin(x[2])
16      c_yaw = math.cos(x[2])
17      x_w = marker_position_world[0]
18      y_w = marker_position_world[1]
19      H = np.array([
20        [-c_yaw, -s_yaw, -s_yaw*(x_w-x[0]) + c_yaw*(y_w-x[1])],
21        [s_yaw, -c_yaw, -c_yaw*(x_w-x[0]) - s_yaw*(y_w-x[1])],
22        [0, 0, -1]
23      ])
24      return H

25    def calculateKalmanGain(self, sigma_p, H, R):
        #Calcula la ganancia de Kalman, ver ecuación (2.74):
26      return np.dot(np.dot(sigma_p, H.T), np.linalg.inv(np.dot(H,
27      np.dot(sigma_p, H.T)) + R))

28    def correctState(self, K, x_predicted, z, z_predicted):
        '''
```

```
        Corrige el estado estimado usando la ganancia de Kalman, las
        observaciones de la cámara, el modelo de las observaciones y el modelo
        odométrico previamente calculado, ver ecuación (2.72):
29      '''
30      x_corrected = x_predicted + np.dot(K,(z - z_predicted))
        return x  corrected
31
    def correctCovariance(self, sigma_p, K, H):
32      #Corrige la covarianza del estado estimado, ver ecuación (2.73):
        return np.dot(np.identity(3) - np.dot(K, H), sigma_p)
```

CAPÍTULO V
RESULTADOS DE LA PROPUESTA

5.1 Resultados de simulación del control de trayectoria

En la prueba de simulación del controlador PD, dados los parámetros indicados, se logra un seguimiento de trayectoria suave sin oscilaciones (ver Figura 4.8). En múltiples pruebas de la sintonización de los parámetros del controlador, se observó oscilaciones y desviaciones de la trayectoria deseada (ver Figura 4.10).

Por ejemplo, al disminuir pocas décimas el valor del parámetro derivativo, se obtiene una respuesta sub-amortiguada, en la que el cuadricóptero al despegar y querer llegar a la circunferencia deseada desde su punto de partida (centro de la circunferencia), sobrepasa ligeramente el radio de la circunferencia para luego regresar a la misma y estabilizarse (ver Figura 4.9).

De igual manera, aumentando mesuradamente el valor del parámetro derivativo, se obtiene una respuesta oscilatoria durante todo el recorrido del cuadricóptero (ver Figura 4.10).

Con otros valores de parámetros del controlador, se consiguen respuestas inestables, en la que el cuadricóptero sigue trayectorias espirales alejándose del centro, o desviándose significativamente de la posición deseada.

De esta manera, se consiguió experimentalmente sintonizar los parámetros a los valores indicados en las líneas de programación. Con estos valores, se dejó en simulación un tiempo prudente comprobando el correcto seguimiento de la trayectoria deseada. Al pasar el tiempo sin embargo, el cuadricóptero se desvía ligeramente, esto debido a la falta de precisión de la estimación de la posición por medio de la odometría inclusive en simulación.

Existen múltiples variables que no son simuladas como por ejemplo el viento. Debido a ello, a continuación se va a implementar los algoritmos desarrollados en el cuadricóptero comercial Parrot AR.Drone 2.0 para la verificación de los resultados en simulación.

5.2 Implementación del control de trayectoria

Antes de trasladar el algoritmo desarrollado de control de trayectoria al Parrot AR.Drone 2.0, se probó el funcionamiento del algoritmo de la odometría visto anteriormente en la sección 4.1:

```
1   def ReceiveNavdata(self, t, dt, navdata):
2       rot = self.rotation_to_world(navdata.rotZ)
3       updated_state.velocity = np.dot(rot, np.array([[navdata.vx], [navdata.vy],
4       [navdata.vz]]))
5       updated_state.position += dt * updated_state.velocity
6       self.state.append(updated_state);
```

Se enviaron comandos de avance lineal en los ejes x y y del cuadricóptero con una velocidad mesurada, para comparar el avance real en vuelo medido con una cinta métrica desde el punto de despegue al punto de aterrizaje, contrastado con el cálculo de la estimación de la posición mediante la odometría (ver Figura 5.1). Los resultados obtenidos fueron aceptables, el rango de error se situó con una media de 0.1 y 0.04 metros y una desviación estándar de 0.12 y 0.23 metros en x y y respectivamente (ver Tabla 5.1). Asimismo, se comprobó que los cambios bruscos en vuelo debido a perturbaciones como ráfagas de viento y fuerzas externas como el empujarlo intencionalmente en pleno vuelo, son percibidas por los sensores internos del cuadricóptero y trasladadas en cambios bruscos de sus velocidades lineales que nos entrega a través de su información de navegación en el tópico **/ardrone/navdata**. En consecuencia, la posición estimada calculada por el modelo odométrico también contempló estos cambios bruscos de posición debido a las perturbaciones, pero con cierto margen de error cercanos a los mostrados en la Tabla 5.1.

Figura 5.1: Medición del punto de despegue al punto de aterrizaje para ser comparado con el cálculo de la odometría.

Tabla 5.1: Mediciones de odometría.

Odometría		**Real**	
x	*y*	*x*	*y*
1.02	2.05	1.00	1.80
0.32	1.72	0.50	2.00
1.18	1.15	1.40	1.20
-1.44	1.22	-1.60	1.40
1.15	1.19	1.30	1.40
1.07	-1.32	1.30	-1.50

Luego de las pruebas de odometría, el algoritmo desarrollado en la sección 4.4.3 fue trasladado directamente al cuadricóptero comercial Parrot AR.Drone 2.0 obteniendo respuestas muy alejadas a los resultados de las simulaciones; esto debido principalmente a factores externos del mundo real que no se contemplan en las simulaciones, como por ejemplo el viento. A pesar de ello, se pudo corregir estas dificultades mediante otra sintonización de los parámetros del controlador, que tuvieron que ser disminuidos en cierto porcentaje. Con nuevos valores de parámetros un poco más bajos que los que habían sido sintonizados en simulación, se logró un control aceptable de la trayectoria deseada. Sin embargo, no se pudo evitar las desviaciones constantes de la trayectoria deseada debido a las ráfagas de viento que siempre estaban presentes en las pruebas experimentales, siendo este factor el principal desestabilizante del vuelo controlado del cuadricóptero. En la Figura 5.2 se muestra una foto de las pruebas realizadas con el Parrot AR.Drone 2.0.

En los **videos anexos** a la presente tesis se pueden observar al cuadricóptero en vuelo ejecutando los algoritmos de control desarrollados en la presente tesis, tanto de las simulaciones como de la implementación física.

Figura 5.2: Implementación del controlador PD en el Parrot AR.Drone 2.0.

5.3 Resultados de la simulación de localización

Para la prueba del filtro de Kalman, se ha definido el seguimiento de una trayectoria cuadrada mostrada en la Figura 5.3. Como se observa en la figura, se han colocado *landmarks* en el suelo en la proyección de dicha trayectoria separados por una corta distancia arbitraria entre ellos. Las líneas amarillas simbolizan la observación del *landmark* realizada por la cámara del cuadricóptero. En la simulación, se puede observar el incremento de la covarianza del modelo odométrico conforme el cuadricóptero avanza, que es eliminada inmediatamente después de la observación de algún *landmark*. La línea roja representa la posición del cuadricóptero según su modelo odométrico. La línea celeste representa la posición real del vehículo.

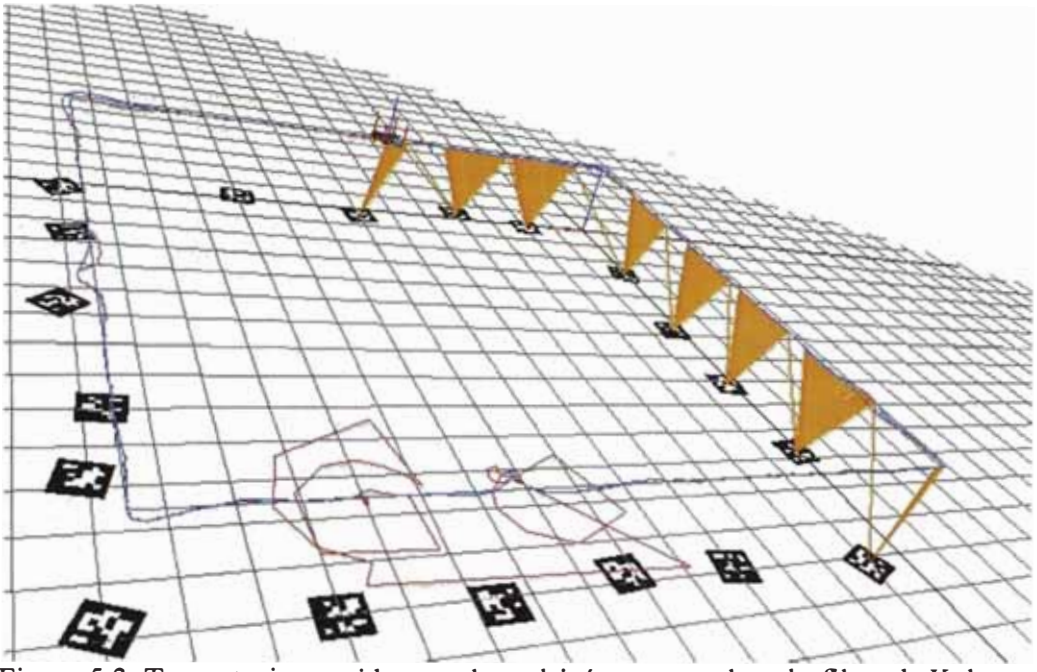


Figura 5.3: Trayectoria seguida por el cuadricóptero empleando filtro de Kalman.

Se observa que debido a la observación de los *landmarks* para la corrección de la posición, el cuadricóptero logra mantener la trayectoria deseada sin desviarse de ella, lo que no sucede con la ausencia de la implementación de la corrección de la posición por las observaciones de *landmarks*, según se comenta a continuación.

En la simulación mostrada en la Figura 5.4 se ha eliminado intencionalmente el filtro de Kalman. Se puede observar un desvío importante de la trayectoria deseada, debido a la gran diferencia entre la posición calculada por el modelo odométrico del cuadricóptero y la posición real, después de haber avanzado cierta distancia, ya que mientras el vehículo cree estar en la posición correcta según el modelo odométrico, en realidad se va alejando cada vez más de la posición deseada, a causa de la información imprecisa obtenida por sus sensores de odometría (velocidades lineales y angulares). Nota: En la Figura 5.4 también se representa las observaciones de *landmarks*, pero estas no fueron consideradas en el algoritmo debido a la ausencia del filtro de Kalman.

La elipse de color rojo alrededor del cuadricóptero representa la incertidumbre calculada por la matriz de covarianza, la cual crece conforme el vehículo avanza por ausencia de la corrección efectuada mediante el filtro de Kalman. Esta crece de manera cuadrática según el ruido en la odometría, lo cual se consideró constante.

La covarianza en una representación matemática de la incertidumbre, pero no refleja necesariamente la desviación del vehículo de su trayectoria deseada, la cual depende mucho de factores externos como el viento.

Comparando ambas figuras se puede observar con claridad el gran aporte de la corrección de la posición mediante las observaciones de *landmarks* para la estimación del estado del cuadricóptero, brindando un adecuado seguimiento de la trayectoria deseada de manera autónoma, sin necesidad de ningún mando remoto.

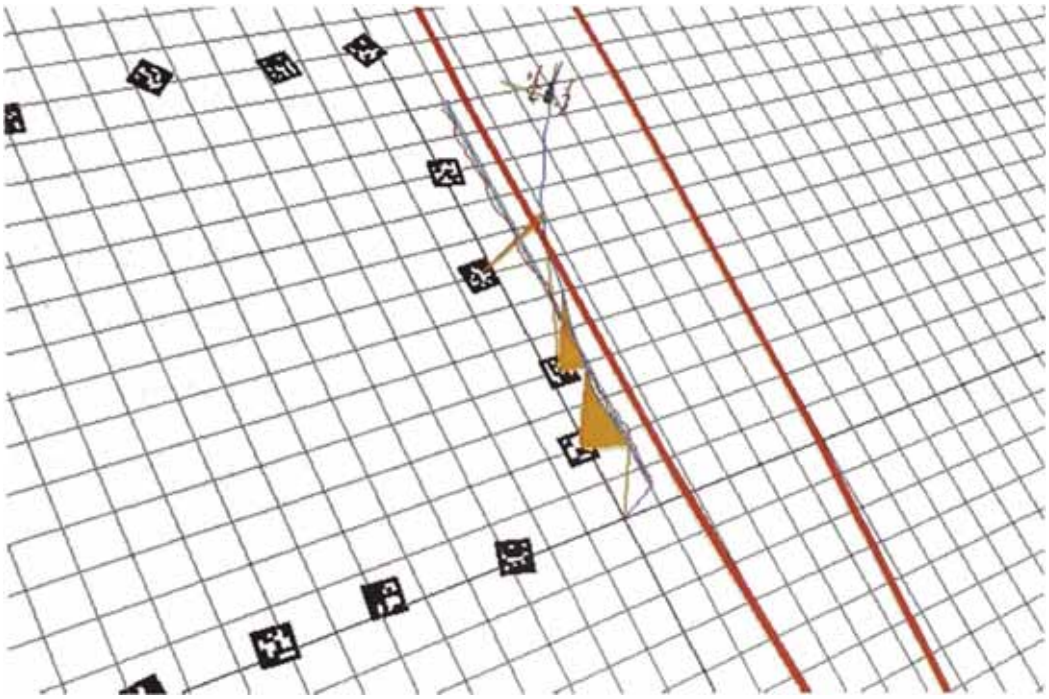


Figura 5.4: Seguimiento de trayectoria sin filtro de Kalman, se observa un desvío de la trayectoria deseada debido al alto ruido del modelo odométrico, representado por la elipse roja como una muy alta covarianza.

5.4 Implementación del algoritmo de localización

Para la implementación del filtro de Kalman al Parrot AR.Drone, se ha utilizado como *landmark* una imagen prediseñada la cual es detectada automáticamente por la unidad de procesamiento del cuadricóptero (Ver Figura 5.5).

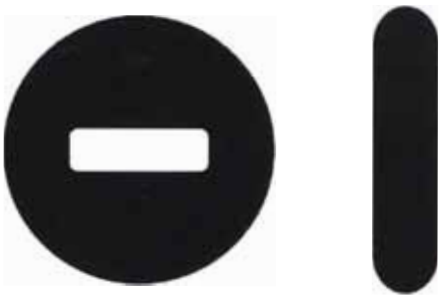


Figura 5.5: Imagen prediseñada que es detectada por defecto por la unidad de procesamiento del Parrot AR.Drone.

Esta imagen fue impresa y colocada cada cierta distancia en la proyección de la trayectoria deseada, para que sea detectaba por la cámara digital inferior del Parrot AR.Drone, el cual al ser detectaba, nos envía información respecto a la distancia aproximada a la que se encuentra y su posición relativa al cuadro de la imagen recibida. Esta información es procesada para obtener la posición relativa x_m y y_m del *landmark* en las coordenadas del cuadricóptero, y la orientación relativa ψ_m, añadidas al vector de la ecuación (4.34).

Al detectar a esta única imagen como *landmark*, se ha optado por cambiar la orientación de la imagen en el suelo 180° para los *landmarks* de orden impar con respecto a los de orden par, siendo necesaria la detección de todos los *landmark* en orden consecutivo para no perder la ilación del orden de cada uno de ellos. Esta

limitación se vería solucionada al tener una imagen distinta por cada posición, pero teniendo como consecuencia la detección de cada una de esas imágenes.

En las pruebas experimentales de la implementación del algoritmo de localización, se logró un comportamiento similar al de la simulación, pero con mayores perturbaciones debido al viento y con error mucho mayor de la estación de la posición según su modelo odométrico. Finalmente, esta posición equivocada es corregida apenas se detecta un *landmark*, mejorando su posición según la trayectoria deseada previamente definida. Asimismo se observa un comportamiento oscilatorio en el seguimiento de la trayectoria, lo que se puede corregir ajustando nuevamente los parámetros del controlador PD sintonizados previamente.

Figura 5.6: Implementación del EKF en el Parrot AR.Drone mediante detección de *landmarks*.

La implementación se muestra también en los **videos anexos** a la presente tesis.

CONCLUSIONES

1. Mediante el uso del filtro de Kalman se logra una estimación eficiente de la posición en vuelo del cuadricóptero, haciendo uso de las observaciones de *landmarks* a través de su cámara digital y de su información de navegación obtenida por odometría visual. De esta manera, se consigue el seguimiento de una trayectoria deseada, evitando la desorientación gracias a la localización continua en el entorno, eliminando la necesidad de ser controlado remotamente para el cumplimiento de este objetivo, proporcionando así cierto grado de autonomía en la navegación.
2. Los modelos matemáticos obtenidos para el cuadricóptero, son modelos no lineales como la mayoría de sistemas dinámicos, sin embargo, mediante su linealización por aproximaciones de Taylor, fue posible adaptar al sistema para su estimación de estado a través del denominado filtro de Kalman Extendido – EKF. Los resultados obtenidos fueron muy positivos.
3. En la actualidad existe un amplio desarrollo de paquetes de simulación para una amplia variedad de sistemas robóticos, de libre acceso por ser desarrollados bajo licencia de código abierto, en plataformas multi-colaborativas como es el caso de la plataforma de simulación Gazebo y la multiplataforma ROS. Esto incentiva a la

investigación y desarrollo de técnicas y algoritmos que pueden ser testeados en dichos entornos de programación.

4. Para el control del seguimiento de una trayectoria deseada, resulta muy práctico el uso de un controlador PID o PD, siendo la entrada del controlador el error generado por la diferencia entre el estado deseado y el estado estimado. Los resultados obtenidos en la simulación tanto en condiciones ideales y reales confirman el adecuado uso de este controlador clásico.
5. La estimación de la posición del cuadricóptero a través del modelo odométrico presentado en el capítulo correspondiente, se vuelve muy imprecisa conforme avanza el cuadricóptero. Esto lo podemos observar en el seguimiento de la trayectoria circular programada, que al dar más vueltas alrededor de la circunferencia programada, el centro de la misma se desvía y la trayectoria va perdiendo la referencia original. También se observa esto en el seguimiento de las trayectorias lineales. Es este el motivo por el cual se desarrolla una mejor estimación de estado a través de las observaciones de la cámara digital del cuadricóptero, mediante el filtro de Kalman.

RECOMENDACIONES Y TRABAJOS FUTUROS

1. Respecto al reconocimiento de *landmarks*, seria recomendado la identificación de otras figuras como *landmarks* o de la misma añadiendo colores u otra característica, esto anularía la necesidad de reconocer cada *landmark* en orden consecutivo.

2. Como limitación, se debe tener definidas las ubicaciones de los *landmarks*, esta limitación se puede anular añadiendo algoritmos de mapeo simultaneo, lo que también requeriría la identificación única de cada *landmark* por ubicación.

3. Asimismo, se puede hacer uso de la cámara frontal para el reconocimiento de *landmarks*, los cuales podrían ser diversas características únicas de la imagen recibida.

4. Actualmente se viene desarrollando muchas técnicas de reconocimiento de *landmarks* y localización y mapeo simultáneo (SLAM) en el campo de la visión computacional y la robótica.

BIBLIOGRAFÍA

[1] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy. *Autonomous navigation and exploration of a quadrotor helicopter in gps-denied indoor environments*. Association for Unmanned Vehicle Systems International (IARC), 2009.

[2] H. Durrant-Whyte and T. Bailey. *Simultaneous localization and mapping: Part I*. Robotics & Automation Magazine, 13(2): 99 – 110, 2006.

[3] H. R. Everett. *Sensors for mobile robots, theory and application*. CRC Press, 1995.

[4] D. Mellinger, N. Michael, V. Kumar. *Trajectory Generation and Control for Precise Aggressive Maneuvers with quadrotors*. International Journal of Robotics Research, Apr. 2012.

[5] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. *FastSLAM: A factored solution to the simultaneous localization and mapping problem*. In Proc. of the AAAI National Conference on Artificial Intelligence, 2002.

[6] R. Murray, et al., *A Mathematical Introduction to Robotic Manipulation*, CRC, Boca Raton, FL 1994.

[7] A. Ollero, *Robótica Manipuladores y robots móviles*, Marcombo, Barcelona, 2001.

[8] S. Piskorski, N. Brulez, P. Eline. *AR.Drone Developer Guide*, SDK 1.7, May 17, 2011.

[9] B. Siciliano, O. Khatib. *Handbook of Robotics*. Springer, 2008.

[10] R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza. *Introduction to Autonomous Mobile Robots*. The MIT Press, Second edition, 2011.

[11] M. Smith. *Flying drone peers into Japan's damaged reactors*. CNN: http://edition.cnn.com/2011/WORLD/asiapcf/04/10/japan.nuclear.reactors/

[12] J. F. Suarez. *Introducción a la teoría de probabilidad*. Universidad Nacional de Colombia, 2002.

[13] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[14] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, Boston, USA, 2005.

[15] K. P. Valavanis. *Advances in unmanned aerial vehicles*, volume 33. Springer, Tampa, FL, 2007.

[16] E. W. Weisstein. *Euler Angles*, from MathWorld - A Wolfram Web Resource. Available online: http://mathworld.wolfram.com/EulerAngles.html.