BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA ELECTRÓNICA

MAESTRÍA EN CIENCIAS DE LA ELECTRÓNICA
OPCIÓN EN AUTOMATIZACIÓN

"Control MPC de un cuadricóptero para el seguimiento de trayectorias basado en odometría visual"

T E S I S

Presentada para obtener el título de:
Maestro en Ciencias de la Electrónica, Opción en Automatización

Presenta:

Ing. Eduardo Salazar Hidalgo*

Directores:

Dra. Josefina Castañeda Camacho (FCE-BUAP)

Dr. César Martínez Torres (EDEI-UDLAP)

Dr. José Martínez Carranza (INAOE)

Puebla, México **Enero 2021**

***BECARIO CONACYT**

BUAP®

AGRADECIMIENTOS

Agradezco a mi familia por su apoyo incondicional a lo largo de mi proceso de formación.

A la Benemérita Universidad Autónoma de Puebla por abrirme sus puertas y darme la oportunidad de llevar a cabo mis estudios de maestría.

Al Consejo Nacional de Ciencia y Tecnología por el apoyo económico brindado a lo largo de este periodo.

A mis asesores por su atención y tiempo proporcionados durante el desarrollo de mi trabajo de investigación.

A mis profesores por haberme compartido sus conocimientos y por hacerme madurar como investigador.

A la UDLAP y al INAOE por abrirme sus puertas.

Y a mis compañeros de la generación MCEA 2018 con quienes compartí agradables momentos.

"Quiet people have the loudest minds."

— Stephen Hawking

DEDICATORIA

Para mis padres María Teresa y Ángel, y a mi pareja sentimental Flor Lizeth.

Índice general

Índice de figuras

Índice de tablas

1 Introducción

Los vehículos aéreos no tripulados *Unmanned Aerial Vehicles* (UAV's por sus siglas en Inglés) son vehículos sin tripulación, capaces de mantener de manera autónoma un nivel de vuelo controlado. El diseño de los UAV's tiene una amplia variedad de formas, tamaños, configuraciones y características. Debido a la gran variedad existente, la comunidad científica los ha clasificado de acuerdo a su peso y rango de operación tal como se muestra en la Tabla 1.1.

Tipo	**Peso máximo**	**Rango máximo**
Nano	200 grs	5 Km
Micro	2 Kg	25 Km
Mini	20 Kg	40 Km
Ligero	50 Kg	70 Km
Pequeño	150 Kg	150 Km

Tabla 1.1: Clasificación de UAV's

Especial interés han recibido los micro vehículos aéreos *Micro Aerial Vehicles* (MAV's por sus siglas en Inglés) debido a su tamaño reducido. En la Figura 1.1 se muestran las subcategorías que comprende ente tipo de vehículos [1].

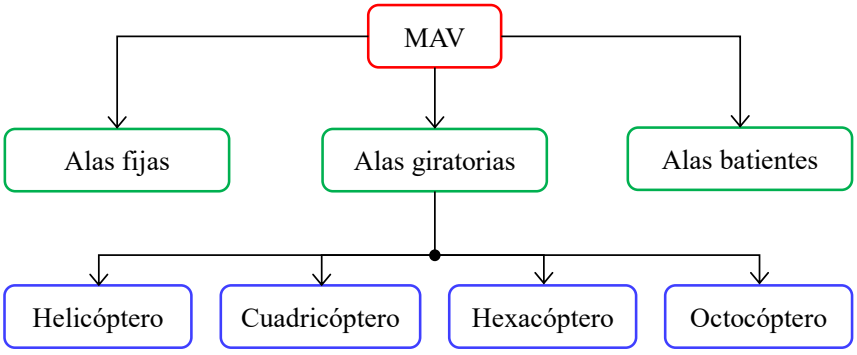


Figura 1.1: Clasificación de MAV's.

De entre ellos los cuadricópteros han cobrado interés debido a sus particulares características que lo diferencian de otros tipos de vehículos aéreos, tales como: su estructura simple, la habilidad de mantenerse en vuelo estacionario y la capacidad de despegar y aterrizar de forma vertical. Un cuadricóptero como su nombre lo indica es un vehículo con cuatro rotores los cuales se encuentran distribuidos de manera estratégica para formar una estructura en forma de cruz. Cada rotor se ubica en cada extremo de forma equidistante hacia el centro de masa del cuadricóptero.

1.1. Antecedentes

1.1.1. Vehículos Cuadricópteros

El primer vehículo tipo cuadrirrotor fue presentado en Francia por los hermanos Brèguet en 1907 y es conocido como el Giroplano Brèguet-Richet No. 1, Figura 1.2. Cada rotor consistía de dos hélices tipo biplano las cuales estaban interconectadas mediante un complejo sistema de poleas. El vehículo estaba propulsado por un motor de combustión interna. Los cuatro rotores fueron configurados del tal forma que un par girara en sentido horario y el otro par en sentido antihorario con el objetivo de cancelar los efectos de torsión. Este diseño se sigue utilizando en los cuadricópteros modernos el cual se ha convertido en un estándar. Debido a inestabilidad del modelo y por falta de un sistema de control adecuado, el vehículo resulto ser poco práctico [2].

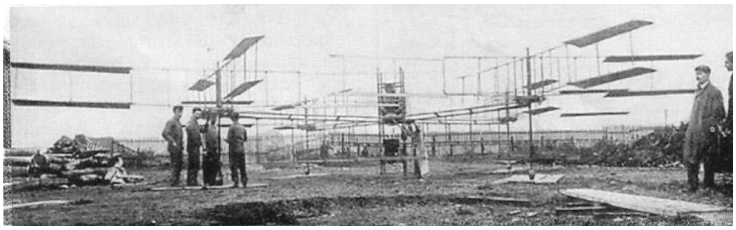


Figura 1.2: Giroplano Breguet-Richet No. 1.

En 1920 Étienne Oehmichen experimentó con el prototipo Oehmichen No. 2, Figura 1.3, el cual incorporaba 4 rotores de dos aspas ubicados en los extremos del marco principal construido a partir de tubos de metal. Las hélices utilizadas permitían variar el ángulo de ataque por lo que proveía cierto grado de control. Cinco hélices ubicadas en los laterales permitían controlar el ángulo en *yaw*. Este modelo es considerado más bien como una combinación entre un cuadrirrotor y un helicóptero. El vehículo tuvo cierto grado de estabilidad permitiendo realizar más de mil vuelos de prueba. Consiguió permanecer varios minutos en el aire y en 1924 logró desplazarse una distancia de 360 metros. Su máximo logro fue el haber viajado 1 kilómetro de distancia en un tiempo de 7 minutos [3].

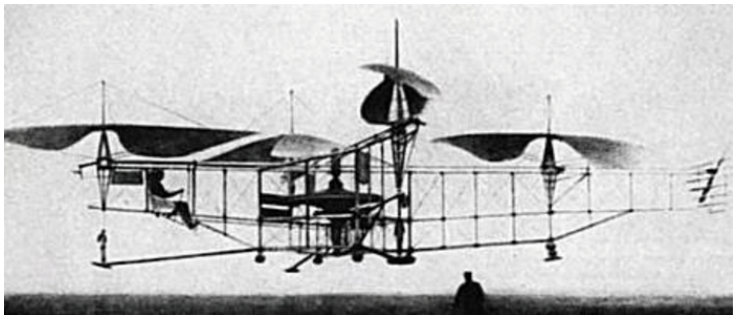


Figura 1.3: Cuadrirrotor Oehmichen No. 2.

En 1922 George de Bothezat e Ivan Jerome mejoraron el diseño de Oehmichen al incluir rotores de 6 aspas, Figura 1.4. Además de añadir ruedas en la parte inferior del vehículo con el objetivo de conseguir despegues y aterrizajes más suaves. Este vehículo efectuó su primer vuelo en octubre de 1922 y hasta 1923 ya había completado más de 100 vuelos exitosos, alcanzando una altura máxima de 5 metros [4].

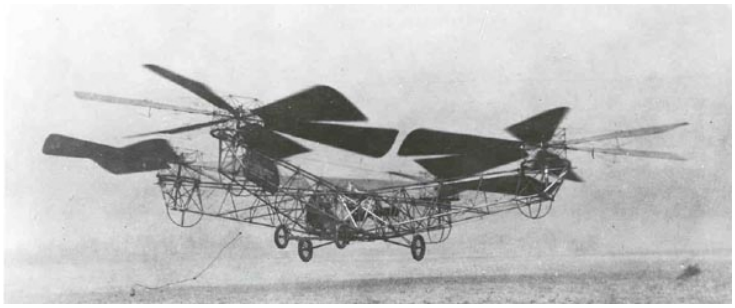


Figura 1.4: Modelo de George de Bothezat e Ivan Jerome.

A pesar del interés inicial en el desarrollo de cuadricópteros, no hubo avances significativos durante las siguientes dos décadas, no fue sino hasta 1956 cuando el vehículo Convertawings Modelo A fue presentado, Figura 1.5. Los rotores de este vehículo eran propulsados por dos motores. Este cuadrirrotor fue el primero el efectuar desplazamientos frontales. Diferentes pruebas de vuelo fueron efectuadas de manera exitosa, sin embargo, debido al carente interés comercial el proyecto fue abandonado, [4].

Figura 1.5: Cuadrirrotor Convertawings Modelo A.

En 1958 la empresa Curtiss Wright diseñó el modelo Curtiss-Wright VZ-7 para el ejército de los Estados Unidos, Figura 1.6. La innovación de este cuadricóptero es que se utilizaron 4 controladores de velocidad, uno para cada rotor. Este diseño es considerado el precursor de los cuadricópteros modernos.

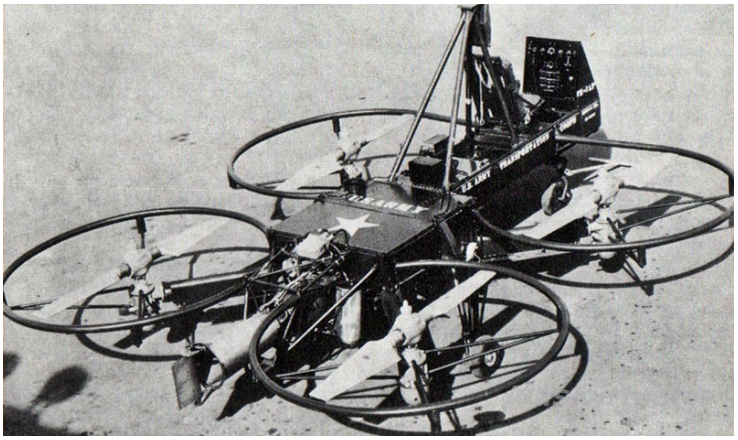


Figura 1.6: Curtiss-Wright VZ-7.

La investigación relacionada con cuadricópteros ha incrementado especialmente en el nuevo milenio debido a los avances en las tecnologías de la computación, sensores, y almacenamiento de energía. Los modelos actuales se caracterizan por ser de tamaño reducido, Figura 1.7.

Figura 1.7: Dron Phantom 3 de la empresa DJI.

1.1.2. Estrategias de control

Los vehículos cuadrirrotores han cobrado especial interés en el área de control automático debido a la naturaleza no lineal, multivariable y subactuada del modelo dinámico. El control de un cuadricóptero se logra variando las velocidades angulares de cada rotor, los cuales generalmente están constituidos por motores eléctricos. El principio básico para el control de un vehículo cuadrirrotor consiste en calcular las velocidades angulares adecuadas para que el vehículo siga una trayectoria determinada.

Frecuentemente el modelo dinámico es dividido en subsistemas para llevar a cabo su análisis por separado y así diseñar controladores independientes. Se han propuesto una gran variedad de estrategias de control para este tipo de plataformas tales como: controladores PID [5], [6] los cuales fueron de los primeros tipos de controladores empleados en estos sistemas; Backstepping por ejemplo: [7], en donde el modelo dinámico es dividido en tres subsistemas tomando en consideración la dinámica de los rotores, el controlador se diseña basándose en la teoría de estabilidad de Lyapunov; no lineales $\mathcal{H}_\infty$ como en [8] donde el modelo es dividido en dos subsistemas, un controlador MPC para el control de movimientos traslacionales y un controlador $\mathcal{H}_\infty$ para el control de la orientación del vehículo; LQR [9] y [10] los cuales emplean el modelo dinámico linealizado en condición de vuelo estacionario para el cálculo de la señal de control; PD, por ejemplo: [11], el cual emplea el modelo dinámico no lineal obtenido a partir de las ecuaciones de movimiento de Euler-Lagrange; par calculado [12], este compensador no lineal se emplea para resolver el problema de seguimiento de trayectorias; entre otros. En [13] se realiza una recopilación de los trabajos dedicados al control de cuadricópteros empleando distintas estrategias. En la Figura 1.8 se clasifican las estrategias de control encontradas en la literatura.

A pesar de que existe una gran variedad de algoritmos de control, la mayoría no toma en consideración la posible saturación en los actuadores, así como posibles restricciones físicas. Por tal motivo, en la última década ha habido un incremento en el diseño de controladores predictivos para vehículos aéreos, especialmente controladores MPC *Model Predictive Control*. Este tipo de controladores tienen la habilidad de incluir restricciones en el proceso de optimización, y con ello obtener una secuencia de control capaz de cumplir con el objetivo de control obedeciendo las restricciones establecidas.

El control MPC puede ser empleado tanto en sistemas lineales como en sistemas no lineales. La sintonización de controladores MPC resulta una tarea sencilla.

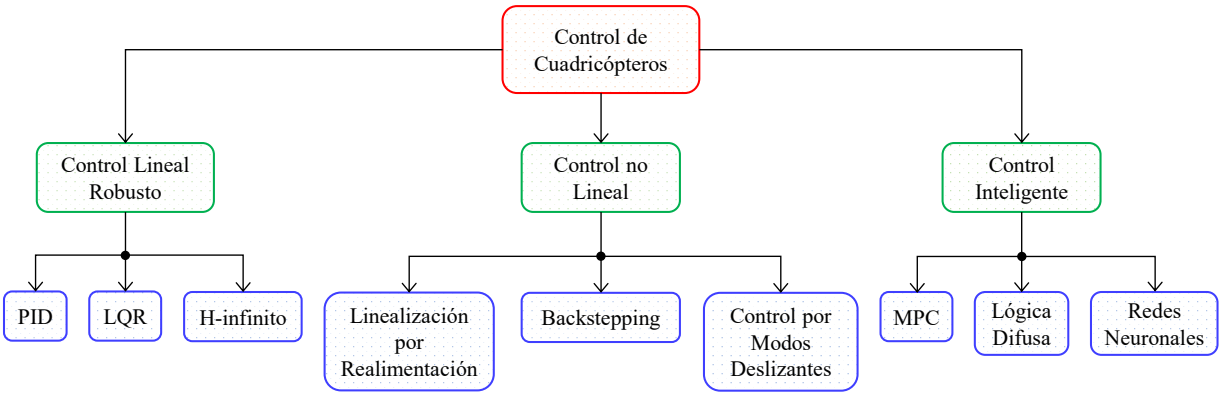


Figura 1.8: Estrategias para el control de cuadricópteros.

En la literatura se pueden encontrar una gran variedad de controladores MPC para el control de cuadricópteros. En [14] se presenta un controlador MPC para el seguimiento de trayectorias de un cuadricóptero donde únicamente se emplea un controlador MPC que involucra la posición y orientación. Los resultados obtenidos son comparados con el tradicional controlador PID y con un controlador Backstepping no lineal. En [15] y [16] se emplean controladores MPC en cascada que utilizan funciones ortonormales para el cálculo de la secuencia de control, esta estructura divide la dinámica del cuadricóptero en dos subsistemas: traslación y rotación, los cuales son controlados por dos algoritmos independientes cada uno con diferentes ganancias de control. Finalmente en [17] de igual forma se emplean funciones ortonormales para calcular la secuencia de control, sin embargo, el modelo dinámico es dividido en tres subsistemas cada uno con su respectivo controlador MPC. Existen trabajos en los que se emplean controladores tradicionales en conjunto con un controlador MPC, tal es el caso de [18] en donde se usa un control $\mathcal{H}_\infty$ para controlar los movimientos rotacionales y un MPC para el control del subsistema de traslación.

1.2. Estado del Arte

Existe actualmente una área de investigación activa que tiene como objetivo principal desarrollar estrategias de control para volver a los cuadricópteros vehículos completamente autónomos. Alrededor del mundo hay distintos centros de investigación enfocados al estudio de MAV's, especialmente trabajos relacionados con cuadricópteros. Estos grupos hacen uso de plataformas comerciales e incluso algunos de ellos se dedican a la construcción de prototipos. El incremento en las investigaciones relacionadas con vehículos cuadrirrotores se debe al creciente número de aplicaciones cotidianas, tales como: operaciones de búsqueda aérea [19], inspección de edificios [20], agricultura [21], vigilancia de tráfico en carreteras [22], recolección de información para la predicción meteorológica [23], vigilancia de bosques [24], fotografía aérea [25], etc.

1.3. Justificación

El Control Predictivo de Modelo tiene la característica principal de adelantarse a la aparición de perturbaciones, por lo que es de especial interés en el seguimiento de trayectorias de vehículos aéreos. Actualmente y gracias a que las computadoras a bordo de los drones han aumentado su capacidad de cómputo es posible implementar controladores de bajo y alto nivel.

1.4. Objetivos

1.4.1. General

Diseño de un controlador basado en control predictivo de modelo y odometría visual para el seguimiento de trayectorias de un dron basado en puntos de referencia.

1.4.2. Particulares

- Estudiar los diferentes métodos de localización basados en odometría visual.
- Estudiar los métodos de control predictivo de modelo.
- Seleccionar el método de localización a utilizar.
- Seleccionar el método de control predictivo basado en modelo a utilizar.
- Implementar el método de localización basado en odometría.
- Implementar el método de control basado en modelo para seguimiento de trayectorias.
- Diseñar un marco experimental para evaluar la localización visual y el seguimiento de trayectorias utilizando el MPC.
- Publicación de resultados.
- Escritura de la tesis

2 Modelo Dinámico

Un cuadricóptero es un tipo de helicóptero con cuatro rotores, los cuales se encuentran ubicados estratégicamente para formar una estructura tipo "X" (Figura 2.1). Las hélices generan un empuje en dirección perpendicular al plano definido por los rotores. La posición y orientación del cuadricóptero se pueden controlar variando las velocidades angulares de los rotores, sin embargo, esto resulta ser un reto debido a que se cuentan únicamente con cuatro motores contra los seis grados de libertad del dron lo cual hace que el cuadricóptero sea considerado como un sistema subactuado. Los motores se encuentran divididos en dos pares, cada par se caracteriza por su ubicación opuesta en la estructura del cuadricóptero, además de girar en el mismo sentido. Los motores 1 y 3 giran en el sentido contrario de las manecillas del reloj, mientras que los motores 2 y 4 giran en el sentido de las manecillas del reloj.

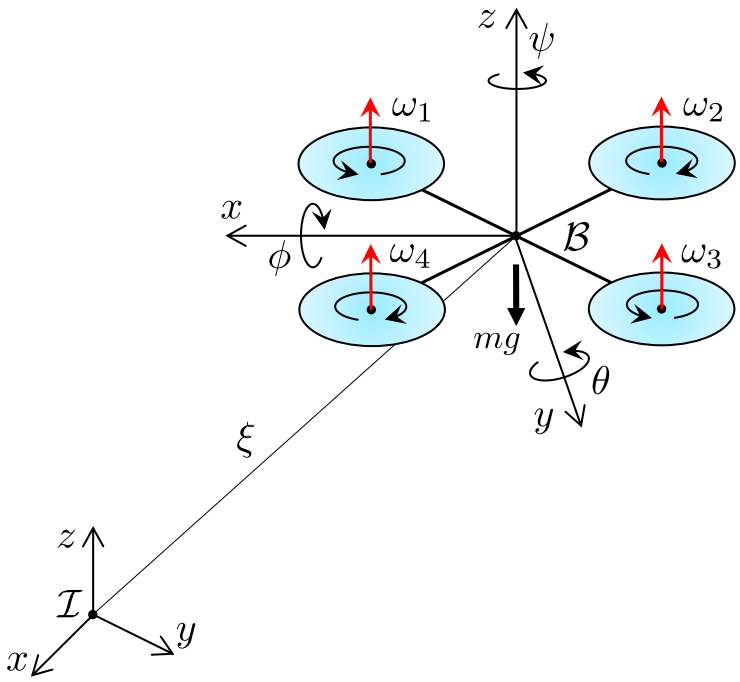


Figura 2.1: Diagrama de cuerpo libre de un Cuadricóptero.

Dependiendo de la velocidad de rotación de cada motor es posible efectuar cada uno de los movimientos básicos del cuadricóptero. Al aumentar o disminuir la velocidad angular de los cuatro rotores de manera simultánea se obtiene un movimiento vertical a lo largo del eje z (Figura 2.2).

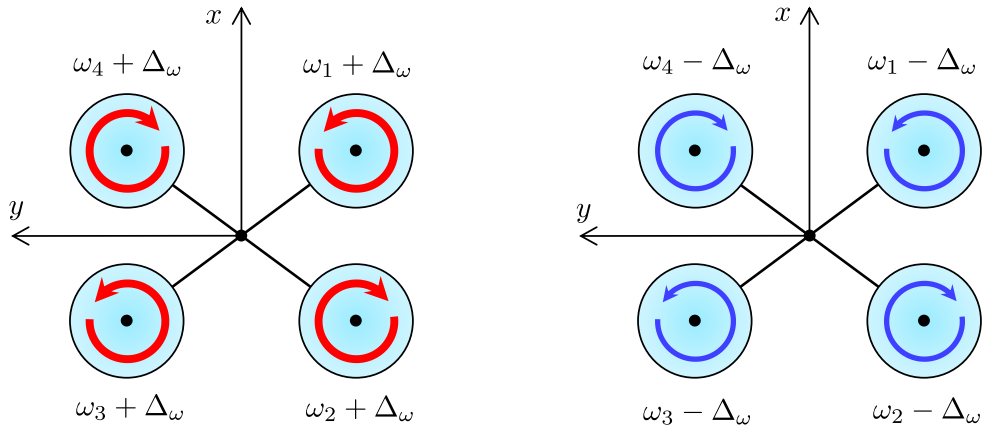


Figura 2.2: Movimiento vertical.

Al aumentar la velocidad en el par de rotores traseros (2 y 3), y al mismo tiempo disminuyendo la velocidad en el par de rotores frontales (1 y 4), se obtiene un desplazamiento frontal en dirección x (movimiento en *pitch*). De manera opuesta, se obtiene un desplazamiento en dirección $-x$ (Figura 2.3).

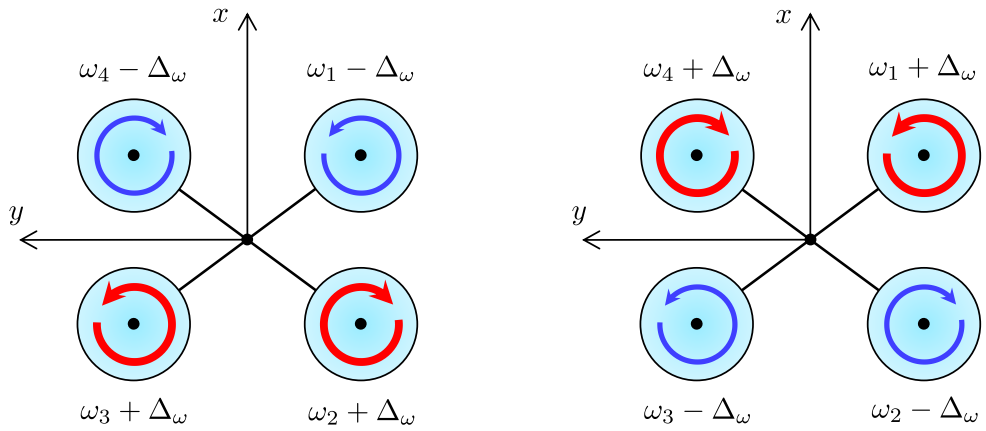


Figura 2.3: Movimiento en *pitch*.

Aumentando la velocidad en el par de rotores de la derecha (1 y 2), y al mismo tiempo disminuyendo la velocidad en el par de rotores de la izquierda (3 y 4), se obtiene un desplazamiento lateral en dirección y (movimiento en *roll*). De manera opuesta, se obtiene un desplazamiento en dirección $-y$ (Figura 2.4).

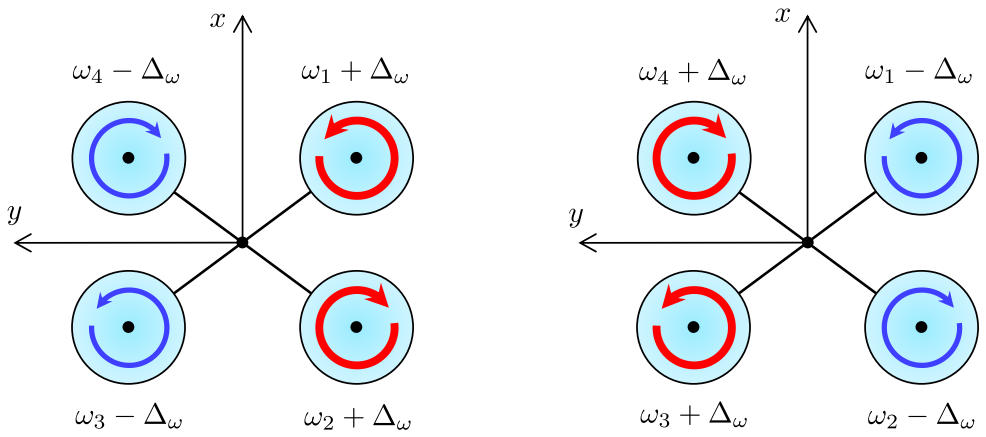


Figura 2.4: Movimiento en *roll*.

Incrementando la velocidad en el par de rotores (1 y 3), y al mismo tiempo disminuyendo la velocidad en el par de rotores (2 y 4), se obtiene una de rotación alrededor del eje z (movimiento en *yaw*) en el sentido de giro contrario a las manecillas del reloj. De manera opuesta, se obtiene una rotación en el sentido de giro a las manecillas del reloj (Figura 2.5).

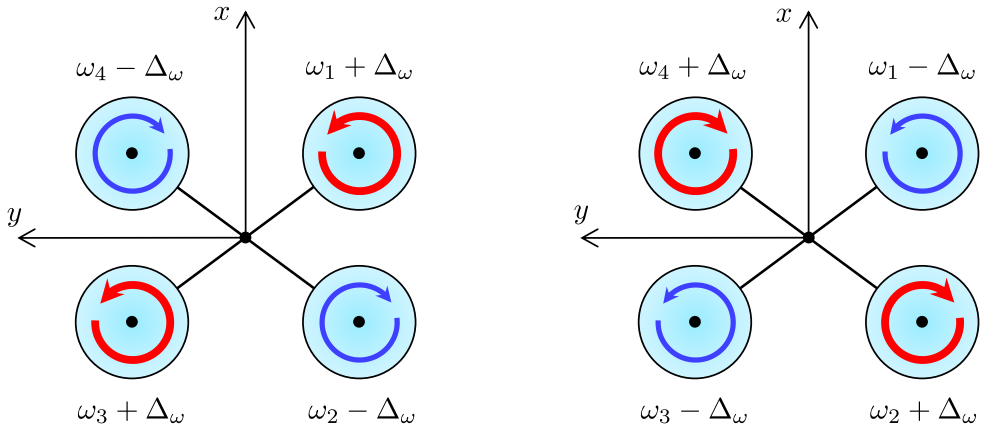


Figura 2.5: Movimiento en *yaw*.

En la literatura existen principalmente dos enfoques para la obtención de las ecuaciones de movimiento que describen la dinámica de un cuadricóptero: Las ecuaciones de Euler-Lagrange [26] y las ecuaciones de Newton-Euler [26]. Sin embargo, para este trabajo se ha decidido seguir el formalismo de Euler-Lagrange debido a que este método describe de manera explícita el balance de energía de un cuerpo en movimiento. El modelo dinámico de un cuadricóptero consiste en un conjunto de ecuaciones diferenciales que describen los movimientos traslacional y rotacional.

Antes de efectuar dicho estudio se hacen las siguientes suposiciones:

- El cuadricóptero es un cuerpo rígido.
- La estructura mecánica es simétrica.
- Las hélices se consideran rígidas.
- El efecto suelo es ignorado.

La posición relativa del sistema de referencia del vehículo $\{B\}$ con respecto al sistema de referencia inercial $\{I\}$ se encuentra definida por el vector $\boldsymbol{\xi} \in \mathbb{R}^3$ (Figura 2.1).

$$\boldsymbol{\xi} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.1}$$

Así mismo la orientación del vehículo estará representada por el vector $\boldsymbol{\eta} \in \mathbb{R}^3$ el cual está compuesto por los ángulos de Euler.

$$\boldsymbol{\eta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \tag{2.2}$$

Por lo tanto, las coordenadas generalizadas del vehículo estarán dadas por el vector $\boldsymbol{q} \in \mathbb{R}^6$.

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{\eta} \end{bmatrix} \tag{2.3}$$

De esta forma se definen las ecuaciones de movimiento de Euler-Lagrange:

$$\frac{d}{dt}\left[\frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \dot{\boldsymbol{q}}}\right] - \frac{\partial \mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \boldsymbol{q}} = \begin{bmatrix} \boldsymbol{F}_\xi \\ \boldsymbol{\tau} \end{bmatrix} \tag{2.4}$$

Donde $\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ representa el Lagrangiano del sistema, el cual resulta de la diferencia entre la energía cinética $\mathcal{K}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ y la energía potencial $\mathcal{U}(\boldsymbol{q})$.

$$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \mathcal{K}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \mathcal{U}(\boldsymbol{q}) \tag{2.5}$$

Como se puede notar la dinámica total del sistema está compuesta por la dinámica traslacional y la dinámica rotacional. Por tanto el análisis se lleva a cabo de forma independiente, sin embargo, al final podremos darnos cuenta de que ambos subsistemas se encuentran íntimamente interconectados.

2.1. Ecuaciones de movimiento traslacional

La energía cinética del robot está dada por la siguiente expresión:

$$\mathcal{K}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) = \frac{1}{2} m \dot{\boldsymbol{\xi}}^T \dot{\boldsymbol{\xi}} \tag{2.6}$$

Donde m representa la masa del vehículo. La energía potencial está relacionada directamente con la altura del cuadricóptero.

$$\mathcal{U}(\boldsymbol{\xi}) = mgz \tag{2.7}$$

Donde g representa la constante de aceleración gravitacional. De esta forma, el Lagrangiano de la dinámica traslacional queda expresado de la siguiente manera:

$$\mathcal{L}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) = \frac{1}{2} m \dot{\boldsymbol{\xi}}^T \dot{\boldsymbol{\xi}} - mgz \tag{2.8}$$

Las ecuaciones de movimiento de Euler-Lagrange para el sistema de traslación están determinadas por la siguiente expresión:

$$\frac{d}{dt}\left[\frac{\partial \mathcal{L}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}})}{\partial \dot{\boldsymbol{\xi}}}\right] - \frac{\partial \mathcal{L}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}})}{\partial \boldsymbol{\xi}} = \boldsymbol{F}_\xi \tag{2.9}$$

Evaluando las derivadas se obtiene la siguiente expresión:

$$m\ddot{\boldsymbol{\xi}} + mg = \boldsymbol{F}_\xi \tag{2.10}$$

Donde $\boldsymbol{F}_\xi \in \mathbb{R}^3$ representa la fuerza aplicada al cuadricóptero generada por empuje total de los rotores, misma que se representa de la siguiente forma:

$$\boldsymbol{F}_\xi = R\hat{\boldsymbol{F}} \tag{2.11}$$

$R \in \mathbb{R}^{3\times 3}$ determina la orientación del vehículo relativa al sistema de coordenadas inercial $\{I\}$. $R(\boldsymbol{\eta}) \in SO(3)$ es el resultado de la multiplicación de las siguientes matrices de rotación [27]:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi & C_\phi \end{bmatrix} \tag{2.12}$$

$$R_y(\theta) = \begin{bmatrix} C_\theta & 0 & S_\theta \\ 0 & 1 & 0 \\ -S_\theta & 0 & C_\theta \end{bmatrix} \tag{2.13}$$

$$R_z(\psi) = \begin{bmatrix} C_\psi & -S_\psi & 0 \\ S_\psi & C_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

Donde $S_\phi = \text{sen}(\phi)$, $C_\phi = \cos(\phi)$, $S_\theta = \text{sen}(\theta)$, $C_\theta = \cos(\theta)$, $S_\psi = \text{sen}(\psi)$ y $C_\psi = \cos(\psi)$.

$$\begin{aligned} R(\boldsymbol{\eta}) &= R_z(\psi)R_y(\theta)R_x(\phi) \\ &= \begin{bmatrix} C_\theta C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\theta & S_\phi C_\theta & C_\phi C_\theta \end{bmatrix} \end{aligned} \tag{2.15}$$

Al mismo tiempo se tiene que:

$$\hat{\boldsymbol{F}} = \begin{bmatrix} 0 \\ 0 \\ F \end{bmatrix} \tag{2.16}$$

Donde F representa la sumatoria de los empujes ejercidos por cada rotor.

$$F = \sum_{i=1}^{4} f_i \tag{2.17}$$

El empuje de cada motor está dado por:

$$f_i = c_T \rho A_i r_i^2 \omega_i^2 \tag{2.18}$$

Para $i = 1, 2, 3, 4$. c_T es el coeficiente de empuje que depende directamente de la geometría de las hélices. ρ representa la densidad del aire. A_i es el área del disco generada por la rotación de la i-ésima hélice, r_i indica el radio de la misma. Por último, $\omega_i \in \mathbb{R}$ es la velocidad angular del i-ésimo motor. De forma alterna, la expresión anterior se puede reescribir de la siguiente forma:

$$f_i = k_T \omega_i^2 \tag{2.19}$$

Donde $k_T > 0$ es la constante de empuje. Sustituyendo a (2.15) y (2.16) en (2.11) se obtiene la siguiente ecuación:

$$\boldsymbol{F}_\xi = \begin{bmatrix} C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\phi S_\theta S_\psi - S_\phi C_\psi \\ C_\phi C_\theta \end{bmatrix} F \tag{2.20}$$

Sustituyendo a (2.20) en (2.10) se obtiene el siguiente sistema de ecuaciones:

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\phi S_\theta S_\psi - S_\phi C_\psi \\ C_\phi C_\theta \end{bmatrix} F - \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{2.21}$$

Estas ecuaciones describen la dinámica traslacional de cuadricóptero.

2.2. Ecuaciones de movimiento rotacional

Para este caso, la energía cinética estará dada por la siguiente expresión:

$$\mathcal{K}(\boldsymbol{\Omega}, \dot{\boldsymbol{\Omega}}) = \frac{1}{2}\boldsymbol{\Omega}^T I \boldsymbol{\Omega} \tag{2.22}$$

Donde $\boldsymbol{\Omega} \in \mathbb{R}^3$ representa el vector de velocidades angulares de $\{B\}$ con respecto a $\{I\}$ expresado en $\{B\}$.

$$\boldsymbol{\Omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.23}$$

Además, suponiendo que la distribución de masa del cuadricóptero es simétrica se considera el tensor de inercia $I \in \mathbb{R}^{3\times3}$ la cual es una matriz diagonal con la siguiente estructura.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{2.24}$$

Las componentes del vector de velocidad angular del cuadricóptero en el sistema de referencia $\{B\}$ se pueden representar de la siguiente forma:

$$\boldsymbol{\Omega} = \dot{\phi}\boldsymbol{i} + \dot{\theta}R_x^T(\phi)\boldsymbol{j} + \dot{\psi}R_x^T(\phi)R_y^T(\theta)\boldsymbol{k}$$

O bien,

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_\phi & S_\phi \\ 0 & -S_\phi & C_\phi \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} C_\theta & 0 & -S_\theta \\ S_\phi S_\theta & C_\phi & S_\phi C_\theta \\ C_\phi S_\theta & -S_\phi & C_\phi C_\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{2.25}$$

Efectuando las operaciones correspondientes y agrupando términos, se obtiene la siguiente expresión:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & S_\phi C_\theta \\ 0 & -S_\phi & C_\phi C_\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.26}$$

$$\boldsymbol{\Omega} = W_\eta \dot{\boldsymbol{\eta}} \tag{2.27}$$

Donde $W_\eta \in \mathbb{R}^{3\times3}$ es el Jacobiano del sistema de rotación.

$$W_\eta = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & S_\phi C_\theta \\ 0 & -S_\phi & C_\phi C_\theta \end{bmatrix} \tag{2.28}$$

De forma alternativa se tiene la siguiente expresión para el vector de velocidades angulares.

$$\boldsymbol{\Omega} = \begin{bmatrix} \dot{\phi} - \dot{\psi} S_\theta \\ \dot{\psi} S_\phi C_\theta + \dot{\theta} C_\phi \\ \dot{\psi} C_\phi C_\theta - \dot{\theta} S_\phi \end{bmatrix} \tag{2.29}$$

Sustituyendo a (2.27) en (2.22) se tiene lo siguiente:

$$\mathcal{K}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) = \frac{1}{2} \dot{\boldsymbol{\eta}}^T W_\eta^T I W_\eta \dot{\boldsymbol{\eta}}$$

De la expresión anterior es posible definir la matriz de inercia del cuadricóptero $M(\boldsymbol{\eta}) \in \mathbb{R}^{3\times 3}$ la cual es definida positiva y por lo tanto simétrica.

$$M(\boldsymbol{\eta}) = W_\eta^T I W_\eta \tag{2.30}$$

Por lo tanto, la energía cinética para el sistema de rotación se puede reescribir de la siguiente manera:

$$\mathcal{K}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) = \frac{1}{2} \dot{\boldsymbol{\eta}}^T M(\boldsymbol{\eta}) \dot{\boldsymbol{\eta}} \tag{2.31}$$

De esta forma, el Lagrangiano de la dinámica rotacional queda expresado de la siguiente manera, nótese que para la dinámica rotacional no se toma en cuenta la energía potencial debido a que para este subsistema no existe variación en la altura.

$$\mathcal{L}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) = \frac{1}{2} \dot{\boldsymbol{\eta}}^T M(\boldsymbol{\eta}) \dot{\boldsymbol{\eta}} \tag{2.32}$$

Las ecuaciones de movimiento de Euler-Lagrange para el sistema de rotación están determinadas por la siguiente expresión:

$$\frac{d}{dt} \left[\frac{\partial \mathcal{L}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})}{\partial \dot{\boldsymbol{\eta}}} \right] - \frac{\partial \mathcal{L}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})}{\partial \boldsymbol{\eta}} = \boldsymbol{\tau} \tag{2.33}$$

Para:

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} (f_3 - f_1) l \\ (f_2 - f_4) l \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} \tag{2.34}$$

Donde l es la distancia entre los ejes de los motores y el centro de gravedad del vehículo. M_i es el par generado alrededor del eje de rotación del i-ésimo rotor.

$$M_i = k_D \omega_i^2 \tag{2.35}$$

k_D es conocido como coeficiente de arrastre. Finalmente se obtienen las siguientes expresiones:

$$\left[\frac{\partial \mathcal{L}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})}{\partial \dot{\boldsymbol{\eta}}} \right] = M(\boldsymbol{\eta}) \dot{\boldsymbol{\eta}} \tag{2.36}$$

$$\frac{d}{dt} \left[\frac{\partial \mathcal{L}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})}{\partial \dot{\boldsymbol{\eta}}} \right] = M(\boldsymbol{\eta}) \ddot{\boldsymbol{\eta}} + \dot{M}(\boldsymbol{\eta}) \dot{\boldsymbol{\eta}} \tag{2.37}$$

$$\left[\frac{\partial \mathcal{L}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})}{\partial \boldsymbol{\eta}} \right] = \frac{1}{2} \frac{\partial}{\partial \boldsymbol{\eta}} \left[\dot{\boldsymbol{\eta}}^T M(\boldsymbol{\eta}) \dot{\boldsymbol{\eta}} \right] \tag{2.38}$$

De esta forma:

$$M(\boldsymbol{\eta})\ddot{\boldsymbol{\eta}} + \dot{M}(\boldsymbol{\eta})\dot{\boldsymbol{\eta}} - \frac{1}{2}\frac{\partial}{\partial \boldsymbol{\eta}}\left[\dot{\boldsymbol{\eta}}^T M(\boldsymbol{\eta})\dot{\boldsymbol{\eta}}\right] = \boldsymbol{\tau} \tag{2.39}$$

$$M(\boldsymbol{\eta})\ddot{\boldsymbol{\eta}} + \left[\dot{M}(\boldsymbol{\eta}) - \frac{1}{2}\frac{\partial}{\partial \boldsymbol{\eta}}\left[\dot{\boldsymbol{\eta}}^T M(\boldsymbol{\eta})\right]\right]\dot{\boldsymbol{\eta}} = \boldsymbol{\tau} \tag{2.40}$$

Esta última expresión puede reescribirse de la siguiente manera:

$$M(\boldsymbol{\eta})\ddot{\boldsymbol{\eta}} + C(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})\dot{\boldsymbol{\eta}} = \boldsymbol{\tau} \tag{2.41}$$

Aquí $C(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) = \dot{M}(\boldsymbol{\eta}) - \frac{1}{2}\frac{\partial}{\partial \boldsymbol{\eta}}\left[\dot{\boldsymbol{\eta}}^T M(\boldsymbol{\eta})\right]$ representa la matriz de fuerzas centrípetas y de Coriolis [28]. Con lo anterior se obtiene la expresión del sistema de ecuaciones para la dinámica rotacional del cuadricóptero.

$$M(\boldsymbol{\eta})\ddot{\boldsymbol{\eta}} = \boldsymbol{\tau} - C(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})\dot{\boldsymbol{\eta}} \tag{2.42}$$

Donde la matriz de inercia queda representada de la siguiente manera:

$$M(\boldsymbol{\eta}) = \begin{bmatrix} I_{xx} & 0 & -I_{xx}S_\theta \\ 0 & I_{yy}C_\phi^2 + I_{zz}S_\phi^2 & [I_{yy} - I_{zz}]S_\phi C_\phi C_\theta \\ -I_{xx}S_\theta & [I_{yy} - I_{zz}]S_\phi C_\phi C_\theta & I_{xx}S_\theta^2 + I_{yy}S_\phi^2 C_\theta^2 + I_{zz}C_\phi^2 C_\theta^2 \end{bmatrix} \tag{2.43}$$

La matriz de fuerzas centrípetas y de Coriolis se obtiene a partir de la matriz de inercia con la ayuda de los símbolos de Christoffel [28].

$$C(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \tag{2.44}$$

$$\begin{aligned}
C_{11} &= 0 \\
C_{12} &= (I_{yy} - I_{zz})(\dot{\theta}C_\phi S_\phi + \dot{\psi}S_\phi^2 C_\theta) + (I_{zz} - I_{yy})\dot{\psi}C_\phi^2 C_\theta - I_{xx}\dot{\psi}C_\theta \\
C_{13} &= (I_{zz} - I_{yy})\dot{\psi}C_\phi S_\phi C_\theta^2 \\
C_{21} &= (I_{zz} - I_{yy})(\dot{\theta}C_\phi S_\phi + \dot{\psi}S_\phi^2 C_\theta) + (I_{yy} - I_{zz})\dot{\psi}C_\phi^2 C_\theta + I_{xx}\dot{\psi}C_\theta \\
C_{22} &= (I_{zz} - I_{yy})\dot{\phi}C_\phi S_\phi \\
C_{23} &= -I_{xx}\dot{\psi}S_\theta C_\theta + I_{yy}\dot{\psi}S_\phi^2 S_\theta C_\theta + I_{zz}\dot{\psi}C_\phi^2 S_\theta C_\theta \\
C_{31} &= (I_{yy} - I_{zz})\dot{\psi}C_\theta^2 S_\phi C_\phi - I_{xx}\dot{\theta}C_\theta \\
C_{32} &= (I_{zz} - I_{yy})(\dot{\theta}C_\phi S_\phi S_\theta + \dot{\phi}S_\phi^2 C_\theta) + (I_{yy} - I_{zz})\dot{\phi}C_\phi^2 C_\theta \\
&\quad + I_{xx}\dot{\psi}S_\theta C_\theta - I_{yy}\dot{\psi}S_\phi^2 S_\theta C_\theta - I_{zz}\dot{\psi}C_\phi^2 S_\theta C_\theta \\
C_{33} &= (I_{yy} - I_{zz})\dot{\phi}C_\phi S_\phi C_\theta^2 - I_{yy}\dot{\theta}S_\phi^2 C_\theta S_\theta - I_{zz}\dot{\theta}C_\phi^2 C_\theta S_\theta + I_{xx}\dot{\theta}C_\theta S_\theta
\end{aligned}$$

Las ecuaciones (2.21) y (2.41) describen la dinámica completa de un cuadricóptero, como se puede apreciar dichas expresiones se componen de términos no lineales. Las entradas al sistema se encuentran determinadas por las ecuaciones (2.17) y (2.34), mismas que se pueden escribir de forma matricial:

$$\begin{bmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} k_T & k_T & k_T & k_T \\ -lk_T & 0 & lk_T & 0 \\ 0 & lk_T & 0 & -lk_T \\ k_D & -k_D & k_D & -k_D \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{2.45}$$

En la Figura 2.6 se pueden notar las entradas y salidas del sistema.

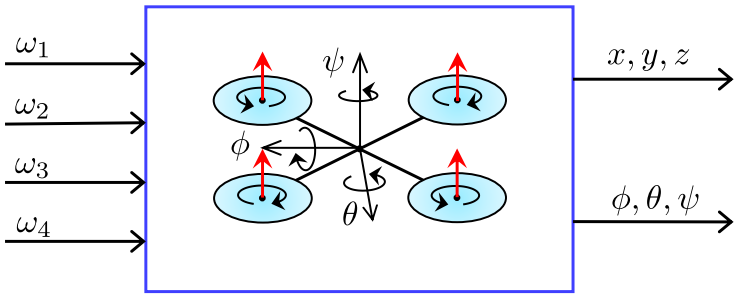


Figura 2.6: Entradas y salidas del modelo dinámico.

En la siguiente sección se presenta un procedimiento para llevar a cabo la linealización del modelo dinámico para su posterior empleo en el diseño de controladores.

2.3. Linealización del modelo dinámico

Debido a la naturaleza subactuada de un cuadricóptero únicamente las coordenadas z, ϕ, θ y ψ se pueden controlar de manera directa tal como se aprecia en las Figuras 2.2, 2.3, 2.4 y 2.5. Las coordenadas x e y se pueden controlar indirectamente a través de los ángulos θ y ϕ respectivamente. La linealización del modelo dinámico se lleva a cabo considerando la situación en la que el vehículo se encuentra en vuelo estacionario (*hovering*), en donde se satisfacen las siguientes condiciones de operación:

$$\begin{aligned} \boldsymbol{\xi} &= \boldsymbol{\xi}_0 \\ \phi &= 0 \\ \theta &= 0 \\ \psi &= \psi_d \end{aligned} \tag{2.46}$$

Lo cual implica,

$$\begin{aligned} \dot{\boldsymbol{\xi}} &= \mathbf{0} \\ \dot{\phi} &= 0 \\ \dot{\theta} &= 0 \\ \dot{\psi} &= 0 \end{aligned} \tag{2.47}$$

Donde $\boldsymbol{\xi}_0$ es un vector de posición constante y ψ_d el ángulo en *yaw* constante deseado. Empleando la aproximación para ángulos pequeños se obtienen las siguientes aproximaciones:

$$\begin{aligned} \operatorname{sen}\phi &\approx \phi \approx 0 \\ \operatorname{sen}\theta &\approx \theta \approx 0 \\ \cos\phi &\approx 1 \\ \cos\theta &\approx 1 \end{aligned} \tag{2.48}$$

Considerando lo anterior, la expresión (2.41) adquiere la siguiente forma:

$$\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{2.49}$$

Como se puede apreciar, la matriz de inercia $M(\boldsymbol{\eta})$ se reduce a una matriz diagonal y, debido a que la matriz de fuerzas centrípetas y de Coriolis $C(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})$ multiplica al vector de velocidades angulares, esta se vuelve cero. Adicionalmente, W_η se reduce a la matriz identidad, por lo que la expresión (2.26) se reduce a lo siguiente:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.50}$$

Con lo anterior se tiene que las velocidades angulares son aproximadamente las mismas tanto en el sistema de referencia inercial $\{I\}$, como en el sistema de referencia unido al vehículo $\{B\}$.

$$\begin{aligned} p &\approx \dot{\phi} \\ q &\approx \dot{\theta} \\ r &\approx \dot{\psi} \end{aligned} \tag{2.51}$$

Con las consideraciones anteriores es posible obtener el modelo dinámico linealizado del cuadricóptero de la siguiente manera:

$$\begin{aligned} \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} &= \begin{bmatrix} \theta C_{\psi_d} + \phi S_{\psi_d} \\ \theta S_{\psi_d} - \phi C_{\psi_d} \\ 1 \end{bmatrix} \frac{F}{m} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \\ \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} &= \begin{bmatrix} \tau_\phi / I_{xx} \\ \tau_\theta / I_{yy} \\ \tau_\psi / I_{zz} \end{bmatrix} \end{aligned} \tag{2.52}$$

Cuando el vehículo se encuentra en condición de vuelo estacionario, la entrada de control F debe ser capaz de compensar los efectos gravitacionales para mantener al cuadricóptero suspendido de forma estable en el plano $x-y$. Por lo tanto se sustituye $F = mg$ en los componentes x e y de las expresiones previas. Como resultado se obtienen las siguientes ecuaciones:

$$\begin{aligned}
\ddot{x} &= g\left[\theta C_{\psi_d} + \phi S_{\psi_d}\right] \\
\ddot{y} &= g\left[\theta S_{\psi_d} - \phi C_{\psi_d}\right] \\
\ddot{z} &= F/m - g \\
\ddot{\phi} &= \tau_\phi / I_{xx} \\
\ddot{\theta} &= \tau_\theta / I_{yy} \\
\ddot{\psi} &= \tau_\psi / I_{zz}
\end{aligned} \tag{2.53}$$

Si se considera $\psi_d = 0$ las ecuaciones anteriores se reducen a lo siguiente:

$$\begin{aligned}
\ddot{x} &= g\theta \\
\ddot{y} &= -g\phi \\
\ddot{z} &= F/m - g \\
\ddot{\phi} &= \tau_\phi / I_{xx} \\
\ddot{\theta} &= \tau_\theta / I_{yy} \\
\ddot{\psi} &= \tau_\psi / I_{zz}
\end{aligned} \tag{2.54}$$

En su forma matricial:

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \phi \\ \theta \\ \psi \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \phi \\ \theta \\ \psi \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_{xx}} & 0 & 0 \\ 0 & 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \begin{bmatrix} G \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{2.55}$$

Donde $G = F - mg$. Como se verá más adelante, las expresiones resultantes serán de gran utilidad durante la etapa de diseño de controladores para el seguimiento de trayectorias.

3

Control Predictivo de Modelo

El Control Predictivo de Modelo MPC, *Model Predictive Control* se desarrolló a finales de los años setenta y ha tenido un desarrollo considerable desde entonces. El término Control Predictivo no designa a una estrategia de control particular, sino a un conjunto de métodos de control que hacen uso explícito de un modelo de la planta para obtener la señal de control minimizando una función de costo, [29], [30], [31]. Estos métodos de control dan lugar a controladores que tienen básicamente la misma estructura y los mismos elementos:

- Uso explícito de un modelo para predecir la evolución del proceso en los instantes futuros.
- Minimización de una función de costo.
- Uso de un horizonte de control finito y deslizante que implica el cálculo de la secuencia de control óptima para todo el horizonte pero con la aplicación de la primera señal de la secuencia de control y la repetición de todo el proceso en el siguiente instante de muestreo.

Los distintos algoritmos de control predictivo difieren en el tipo de modelo utilizado para representar al sistema, las perturbaciones y la función de costo considerada. El control MPC presenta una serie de ventajas sobre otros métodos, entre las que destacan:

- Es una técnica particularmente atractiva para los operadores que requieren pocos conocimientos de control porque los conceptos son muy intuitivos y la sintonización relativamente simple.
- Se puede utilizar para controlar una gran variedad de procesos, desde procesos muy simples hasta procesos con dinámicas complejas como procesos con grandes tiempos muertos, procesos de fase no mínima, procesos inestables o procesos multivariables.
- Su carácter predictivo lo hace compensar intrínsecamente los tiempos muertos.
- Introduce un control anticipativo y de forma natural se compensan las perturbaciones medibles.

- La ley de control resultante es fácilmente implementable.
- Es muy útil cuando se conocen las referencias futuras, como ocurre en el caso de robótica o procesos por lotes.
- Permite tratar las restricciones de una forma sistemática y conceptualmente muy simple durante la fase de diseño.

Como es lógico, también tiene sus inconvenientes:

- Aunque su implementación no es compleja, resulta más difícil que la de los clásicos controladores PID.
- Si la dinámica del sistema no cambia y no existen restricciones, la mayor parte de los cálculos se puede realizar fuera de línea y el controlador resultante es simple, pudiéndose aplicar a plantas con dinámicas rápidas; en caso contrario, los requisitos de cálculo son mucho mayores.
- La mayor dificultad que presenta para su aplicación es la necesidad de un modelo apropiado del proceso cuya obtención requiere conocimientos mínimos de control.

El control predictivo ha demostrado ser en la práctica una estrategia razonable de control y ha sido aplicado con éxito a numerosos procesos industriales.

3.1. Estrategia del MPC

La metodología de todos los controladores pertenecientes a la familia MPC se caracteriza por la siguiente estrategia (Figura 3.1):

1. Las salidas futuras para un horizonte determinado N_p, llamado horizonte de predicción, se predicen cada instante k utilizando el modelo de la planta. Estas predicciones de la salida $\hat{y}(k+j|k)$ para $j = 1 \dots N_p$ dependen de los valores conocidos hasta el instante k (entradas y salidas conocidas) y de las señales de control $u(k+j|k)$, $j = 0 \dots N_c - 1$, que han de ser calculadas y enviadas al sistema.
2. La secuencia de señales de control futuras se calcula minimizando un criterio para mantener al proceso lo más cerca posible de la trayectoria de referencia. Este criterio toma normalmente la forma de una función cuadrática del error entre la salida predicha y la trayectoria de referencias futuras. En la mayor parte de los casos se incluye también el esfuerzo de control dentro de la función de costo. La solución explícita se puede obtener cuando el criterio es cuadrático y el modelo lineal; en caso contrario se ha de utilizar un método numérico para buscar la solución.
3. La señal de control $u(k|k)$ se envía a la planta mientras que el resto de las señales calculadas no se consideran, ya que en el instante siguiente de muestreo $\hat{y}(k+1)$ es ya conocida y los pasos anteriores se repiten con este nuevo valor. Por lo que $u(k+1|k+1)$ se calcula con información diferente y en principio será también diferente de $u(k+1|k)$.

La Figura 3.2 muestra la estructura básica necesaria para implementar el MPC. Se usa un modelo o planta para predecir la evolución de la salida o estado del proceso a partir de las señales de entrada y salidas conocidas. La acciones de control futuras se calculan con el optimizador, que considera la función del coste y las posibles restricciones. El modelo de proceso juega, en consecuencia, un papel decisivo en el controlador.

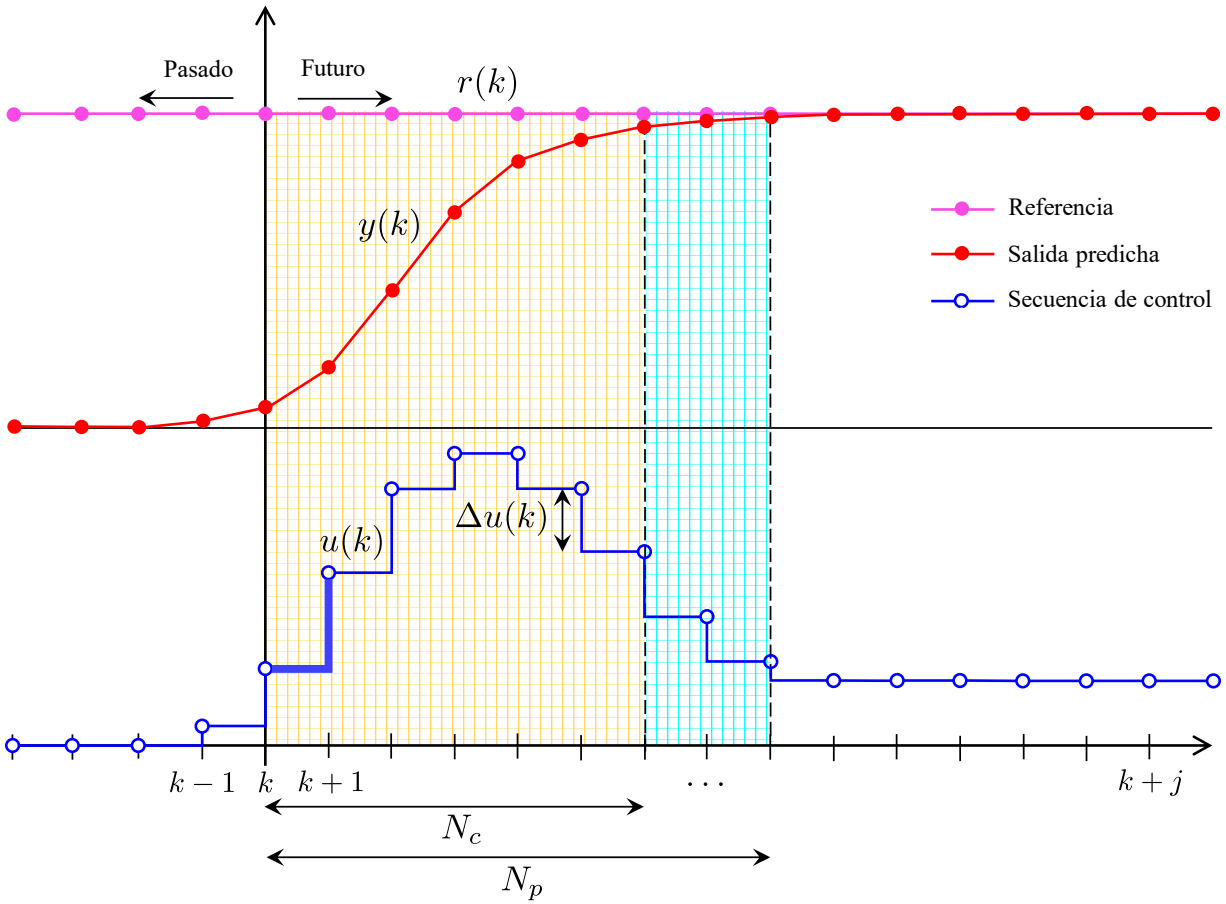


Figura 3.1: Estrategia del MPC.

El modelo elegido debe ser capaz de capturar la dinámica del proceso para predecir de forma precisa la evolución del sistema. Al mismo tiempo, debe ser suficientemente simple de implementar y entender. Las distintas metodologías del control predictivo difieren fundamentalmente en el tipo de modelo utilizado. El optimizador es otra parte fundamental de la estructura, ya que permite obtener las acciones de control a aplicar. Si la función de coste es cuadrática, el modelo lineal y no existen restricciones, la solución se puede obtener de forma analítica. Si este no es el caso se ha de acudir a un algoritmo numérico de optimización que requiere una mayor demanda de coste computacional. El tamaño del problema resultante depende del número de variables, de los horizontes de control y predicción y del número de restricciones, aunque se puede decir que en general los problemas de optimización resultantes en este contexto son problemas más bien modestos [32].

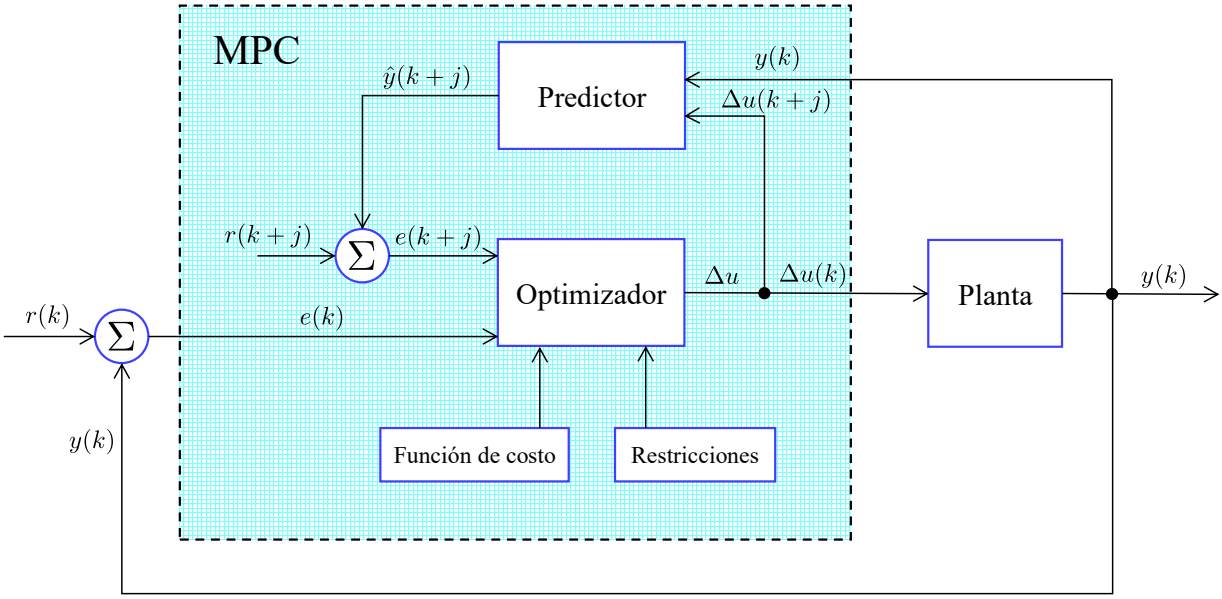


Figura 3.2: Estructura básica del MPC.

3.2. MPC lineal

Existen dos razones principales para el uso de controladores MPC lineales: La identificación de un modelo lineal resulta una tarea fácil cuando se cuentan con los datos de la planta, además los modelos lineales presentan un buen desempeño cuando la planta opera cerca del punto de equilibrio. La mayoría de las veces el uso de un modelo lineal resulta más que suficiente para obtener buenos resultados. Además, el uso de un modelo lineal en conjunto con una función de costo cuadrática da lugar a un problema de optimización convexo cuya solución es bien conocida dentro de la comunidad científica.

Considérese un sistema lineal con n estados y m entradas. A continuación procedemos a definir a $\boldsymbol{x} \in \mathbb{R}^n$ como el vector de estados y $\boldsymbol{u} \in \mathbb{R}^m$ como el vector de entradas de control. De forma similar definimos a $\boldsymbol{x}^{ref} \in \mathbb{R}^n$ como los estados de referencia y a $\boldsymbol{u}^{ref} \in \mathbb{R}^m$ como la secuencia de control en estado estacionario. Tomando en cuenta restricciones lineales, el problema de optimización se formula de la siguiente manera [33]:

$$
\begin{aligned}
\underset{\boldsymbol{U},\boldsymbol{X}}{\text{mín}} \quad & \sum_{k=0}^{N-1} \left(\left\| \boldsymbol{x}_k - \boldsymbol{x}_k^{ref} \right\|_{Q_x}^2 + \left\| \boldsymbol{u}_k - \boldsymbol{u}_k^{ref} \right\|_{R_u}^2 + \left\| \boldsymbol{u}_k - \boldsymbol{u}_{k-1} \right\|_{R_\Delta}^2 \right) \\
& + \left\| \boldsymbol{x}_N - \boldsymbol{x}_N^{ref} \right\|_P^2 \\
\text{s.a.} \quad & \boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k + B_d\boldsymbol{d}_k; \\
& \boldsymbol{d}_{k+1} = \boldsymbol{d}_k, \quad k = 0, \ldots, N-1 \\
& \boldsymbol{u}_k \in \mathbb{U} \\
& \boldsymbol{x}_0 = \boldsymbol{x}(t_0), \quad \boldsymbol{d}_0 = \boldsymbol{d}(t_0).
\end{aligned}
\tag{3.1}
$$

Donde $Q_x \in \mathbb{R}^{n\times n} \succeq 0$ representa el peso dado a la minimización del error de estados en el proceso de optimización, $R_u \in \mathbb{R}^{m\times m} \succ 0$ indica el peso dado a la minimización de la secuencia de control, $R_\Delta \in \mathbb{R}^{m\times m} \succeq 0$ denota el peso dado a la minimización de los incrementos en la secuencia de control, $P \in \mathbb{R}^{n\times n}$ es el peso dado al error de estados terminal. $\boldsymbol{d}_k$ son las perturbaciones externas estimadas y N el horizonte de predicción.

3.3. MPC no lineal

El comportamiento de los sistemas dinámicos se representa de mejor forma empleando ecuaciones diferenciales no lineales las cuales capturan la dinámica de los sistemas de una manera más precisa. El MPC no lineal hace uso del modelo dinámico no lineal para el cálculo de las señales de control óptimas en presencia de restricciones. La extensión de las ideas del MPC lineal a procesos no lineales resulta realmente sencillo, al menos de forma conceptual. En este caso, el problema de optimización se define de la siguiente forma [33]:

$$\begin{aligned} \underset{\boldsymbol{U},\boldsymbol{X}}{\text{mín}} \quad & \int_{t=0}^{T} \left(\left\| \boldsymbol{x}(t) - \boldsymbol{x}_{ref}(t) \right\|_{Q_x}^2 + \left\| \boldsymbol{u}(t) - \boldsymbol{u}_{ref}(t) \right\|_{R_u}^2 + \left\| \boldsymbol{u}(t) - \boldsymbol{u}(t-1) \right\|_{R_\Delta}^2 \right) dt \\ & + \left\| \boldsymbol{x}(T) - \boldsymbol{x}_{ref}(T) \right\|_P^2 \\ \text{s.a.} \quad & \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}); \\ & \boldsymbol{u}(t) \in \mathbb{U} \\ & \boldsymbol{x}(0) = \boldsymbol{x}(t_0). \end{aligned} \tag{3.2}$$

Donde $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ es el modelo dinámico no lineal. A pesar de los avances relacionados con el uso de controladores MPC no lineales, aún existen dificultades derivadas del uso de modelos no lineales, tales como:

- Existe una carencia de técnicas de identificación para sistemas no lineales.
- El problema de optimización resultante es no convexo y requiere de un mayor tiempo de cómputo.
- El estudio de la estabilidad y robustez resulta en una tarea compleja para modelos no lineales.

Algunos de los problemas anteriores han sido resueltos de forma parcial y se esperan tener avances importantes a corto plazo debido a la intensa investigación en esta área.

4 Odometría Visual

4.1. Fundamentos

La visión humana es un sentido muy poderoso la cual nos permite percibir una enorme cantidad de información sobre el entorno que nos rodea. Es por ello que en los últimos años la comunidad científica ha tratado de dotar a robots con sensores que emulen el sistema de visión humano. Los sistemas de visión por computador se dividen principalmente en dos etapas. La primera parte consiste en el desarrollo de dispositivos capaces de convertir la luz en imágenes digitales. La segunda etapa se enfoca en el desarrollo de algoritmos para el procesamiento de dichas imágenes y de esta forma efectuar tareas de propósito específico tales como: estimación de profundidad, detección de movimiento, detección de color, detección de características, etc. [34]. Debido a la importancia de este tema, a continuación se hace una revisión de los fundamentos relacionados con la visión por computadora.

4.1.1. Modelo de la cámara obscura

El diagrama de la Figura 4.1 muestra el modelo básico de una lente delgada.

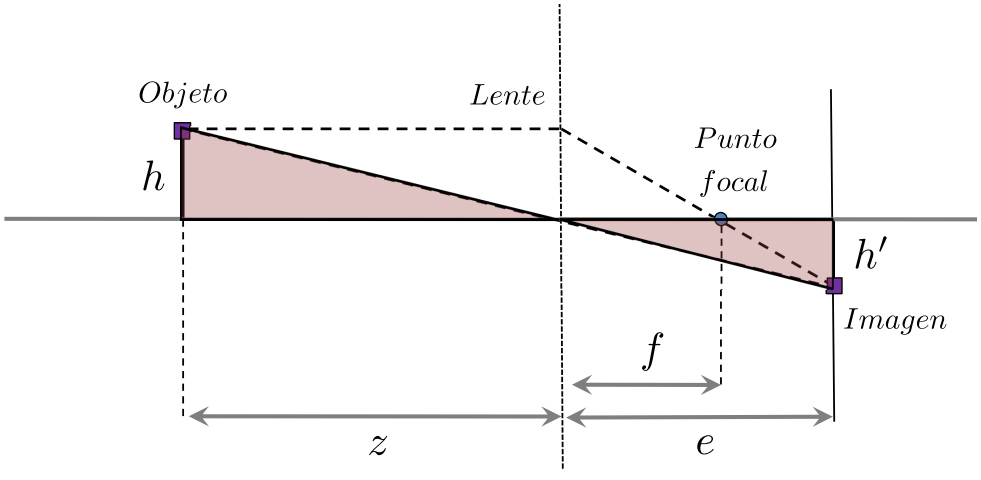


Figura 4.1: Diagrama simplificado de la formación de una imagen.

Empleando el principio de los triángulos semejantes se puede obtener la siguiente relación:

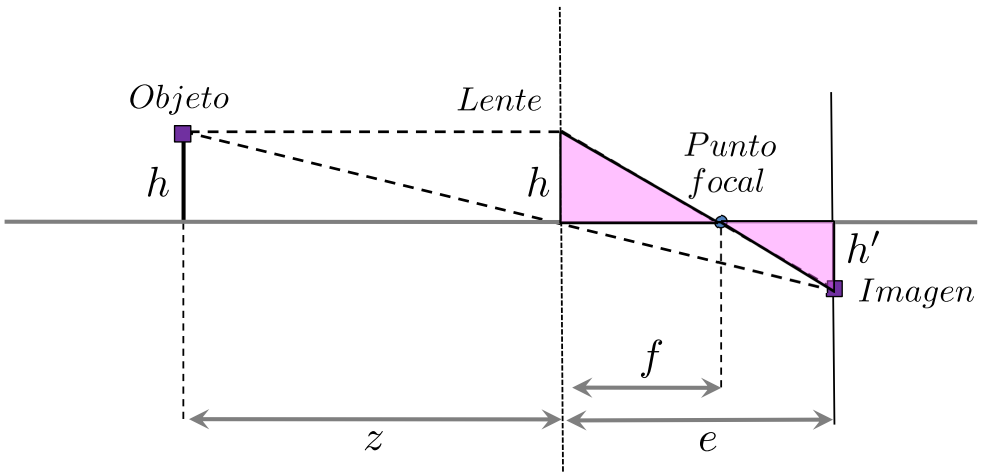


Figura 4.2: Diagrama de la lente delgada.

$$\frac{h'}{h} = \frac{e}{z}$$

$$\frac{h'}{h} = \frac{e-f}{f} = \frac{e}{f} - 1$$

$$\frac{e}{f} - 1 = \frac{e}{z}$$

$$\frac{1}{f} = \frac{1}{z} + \frac{1}{e} \tag{4.1}$$

Donde z representa la distancia hacia el objeto, e la distancia hacia la imagen, y f la distancia focal de la lente. Cualquier objeto en la escena que satisfaga con la ecuación (4.1) estará enfocado. Es necesario ajustar el plano de la imagen de tal forma que los objetos en el infinito se mantengan enfocados. Mientras los objetos en la escena parezcan estar más alejados, el plano de la imagen estará más cerca del plano focal. De la ecuación anterior se hace la siguiente consideración.

$$\frac{1}{z} \approx 0$$

Por lo tanto:

$$\frac{1}{f} \approx \frac{1}{e}$$

$$f \approx e$$

Finalmente se obtiene la siguiente relación:

$$\frac{h'}{h} = \frac{f}{z} \tag{4.2}$$

O bien,

$$h' = \frac{f}{z}h \tag{4.3}$$

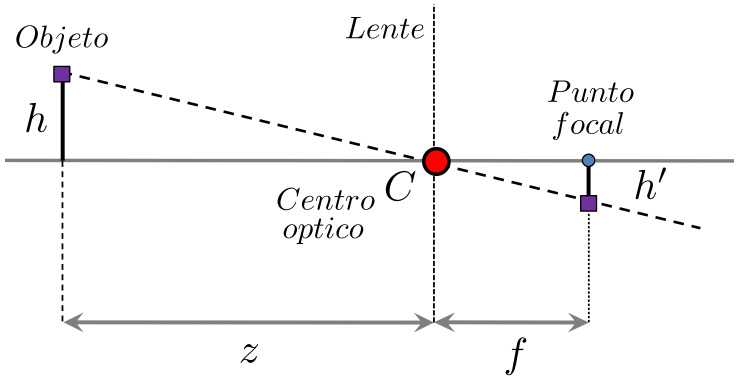


Figura 4.3: Modelo aproximado de la cámara obscura.

La expresión (4.3) explica por qué en una imagen digital los objetos lejanos aparentan ser mas pequeños.

4.1.2. Modelo matemático de una cámara

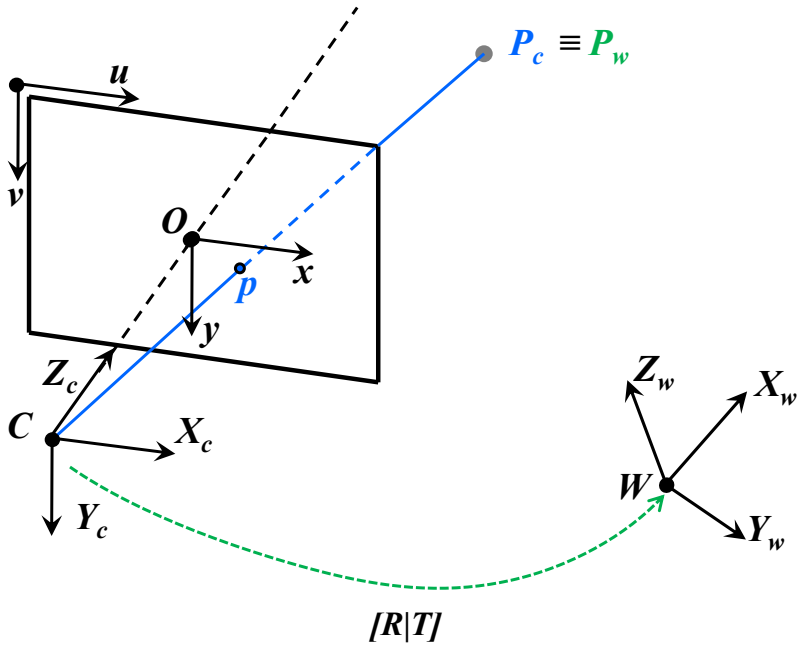


Figura 4.4: Geometría de una cámara que emplea el principio de la cámara obscura.

Considérese a (X_c, Y_c, Z_c) como las componentes del sistema de coordenadas de la cámara con origen en C y el eje Z_c coincidente con el eje óptico. A continuación se introduce

un sistema de referencia bidimensional (x, y) para el plano de la imagen con origen en O y los ejes u y v alineados con los ejes x y y respectivamente [34]. El objetivo principal consiste en mapear un punto arbitrario representado en el sistema de referencia global P_w a su equivalente en coordenadas dentro de la imagen $(u,\ v)$ dada en píxeles. El proceso completo consta de las siguientes etapas:

1. Representar P_c a su equivalente en coordenadas bidimensionales $(x,\ y)$.
2. Convertir $(x,\ y)$ a $(u,\ v)$ dada en píxeles.
3. Realizar una transformación para representar a P_c en coordenadas con respecto al sistema de referencia global P_w.

Paso 1: Por medio del principio de triángulos semejantes es posible obtener las siguientes relaciones:

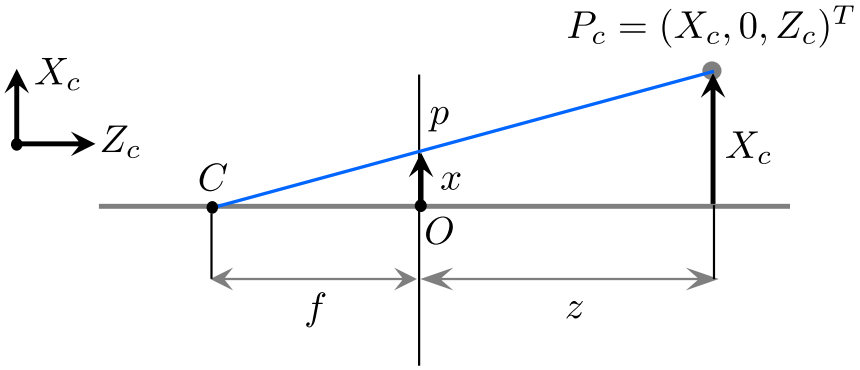


Figura 4.5: Proyección del punto $\boldsymbol{P_c}$ en el plano de la imagen.

$$\frac{x}{f} = \frac{X_c}{Z_c} \quad \Rightarrow \quad x = \frac{fX_c}{Z_c} \tag{4.4}$$

$$\frac{y}{f} = \frac{Y_c}{Z_c} \quad \Rightarrow \quad y = \frac{fY_c}{Z_c} \tag{4.5}$$

De este modo el punto $\boldsymbol{P_c} = (X_c, Y_c, Z_c)$ es proyectado en el plano de la imagen $\boldsymbol{p} = (x, y, f)$. Ignorando a f, el mapeo $\mathbb{R}^3 \mapsto \mathbb{R}^2$ es llevado a cabo mediante la siguiente expresión:

$$(X_c, Y_c, Z_c)^T \mapsto \left(\frac{fX_c}{Z_c}, \frac{fY_c}{Z_c}\right)^T \tag{4.6}$$

Las ecuaciones (4.4) y (4.5) pueden ser representadas mediante coordenadas homogéneas. Considérese los siguientes vectores:

$$\tilde{\boldsymbol{p}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \tilde{\boldsymbol{P}}_c = \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{4.7}$$

Como las coordenadas homogéneas de $\boldsymbol{p}$ y $\boldsymbol{P}_c$ respectivamente. Las ecuaciones de proyección se pueden reescribir de forma matricial:

$$\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{4.8}$$

Nótese que $\lambda = Z_c$ representa a la tercera coordenada de $\boldsymbol{p}$ la cual coincide con la distancia que hay entre la base de la cámara y el plano de la imagen, también conocida como escala. La ecuación (4.6) asume que el origen del sistema de coordenadas del plano de la imagen se ubica sobre el eje óptico, lo que en la práctica no siempre resulta ser cierto.

Paso 2: Al tratarse de píxeles, el origen del sistema de coordenadas se suele ubicar en la esquina superior izquierda del plano de la imagen, por lo que es necesario agregar un *offset*. Además, debido a que las coordenadas de un punto en la imagen se miden en píxeles se debe considerar un factor de escala para las direcciones horizontal y vertical.

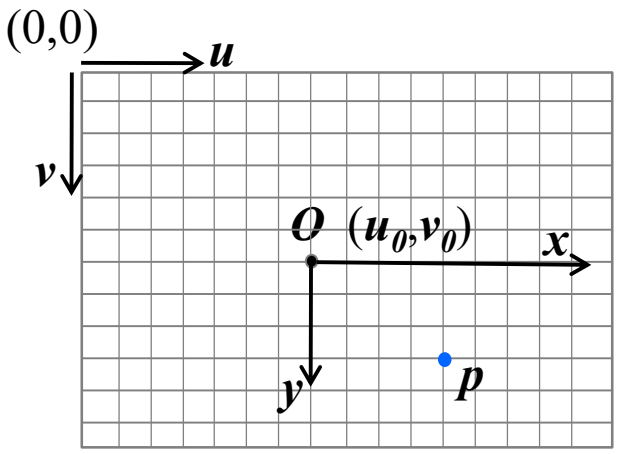


Figura 4.6: Plano de la imagen en píxeles.

De este modo se tienen las siguientes ecuaciones:

$$u = u_0 + k_u x \quad \Rightarrow \quad u = u_0 + \frac{k_u f X_c}{Z_c} \tag{4.9}$$

$$v = v_0 + k_v y \quad \Rightarrow \quad v = v_0 + \frac{k_v f Y_c}{Z_c} \tag{4.10}$$

Para efectuar el siguiente mapeo.

$$(X_c, Y_c, Z_c)^T \mapsto \left(u_0 + \frac{k_u f X_c}{Z_c}, v_0 + \frac{k_v f Y_c}{Z_c} \right)^T \tag{4.11}$$

Lo anterior es expresado en coordenadas homogéneas.

$$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} k_u f & 0 & u_0 & 0 \\ 0 & k_v f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{4.12}$$

Paso 3: Suponiendo que el sistema de referencia global (X_w, Y_w, Z_w) no coincide con el sistema de referencia de la cámara (X_c, Y_c, Z_c) es necesario añadir una transformación entre ambos sistemas (ver Figura 4.4). Dicha transformación está compuesta por una rotación R seguida por una traslación $\boldsymbol{t}$.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + t \tag{4.13}$$

Con esta transformación la ecuación (4.12) se puede reescribir de la siguiente forma:

$$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} k_u f & 0 & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{4.14}$$

$$\lambda \tilde{\boldsymbol{p}} = K[R|\boldsymbol{t}]\tilde{\boldsymbol{P}}_w \tag{4.15}$$

Donde K se conoce como *matriz de parámetros intrínsecos* y $[R|\boldsymbol{t}]$ como *matriz de parámetros extrínsecos* [35]. Estos parámetros se obtienen durante el proceso de calibración de la cámara.

4.1.2.1. Modelo de distorsión

La ecuación (4.15) describe como una cámara efectúa el mapeo de puntos tridimensionales dados en coordenadas con respecto al sistema de referencia global a puntos bidimensionales en el plano de la imagen. Sin embargo, las cámaras incorporan lentes que a menudo distorsionan las imágenes con mayor o menor medida dependiendo de su fabricación. Afortunadamente el modelo de la cámara puede ser extendido para contemplar el modelo de distorsión.

$$u_d = u_0 + \delta_u$$
$$v_d = v_0 + \delta_v$$

Para la mayoría de las cámaras un modelo cuadrático produce buenos resultados.

$$\begin{bmatrix} \delta_u \\ \delta_v \end{bmatrix} = \begin{bmatrix} (u - u_0)\,(1 + k_1 r^2) \\ (v - v_0)\,(1 + k_1 r^2) \end{bmatrix} \tag{4.16}$$

Donde $r^2 = (u - u_0)^2 + (v - v_0)^2$.

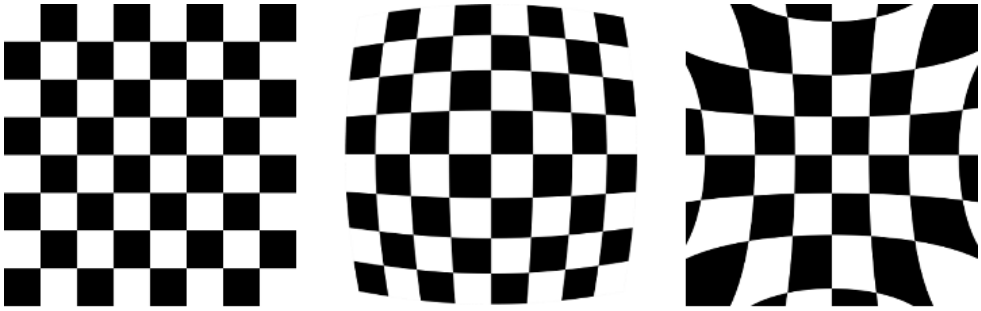


Figura 4.7: Efecto que induce la distorsión radial en las imágenes.

En el modelo anterior únicamente se contempla la *distorsión radial* la cual se manifiesta cuando se proyectan líneas curvas en lugar de líneas rectas (ver Figura 4.7). Sin embargo, el modelo de distorsión se puede extender considerando términos de orden superior dentro del modelo de distorsión radial, así como la inclusión de una expresión para la *distorsión tangencial* la cual surge cuando la lente no se encuentra alineada de forma paralela al plano de la imagen.

$$\begin{bmatrix}\delta_u \\ \delta_v\end{bmatrix} = \underbrace{\begin{bmatrix}(u-u_0)\left(1+k_1r^2+k_2r^4+k_3r^6\right) \\ (v-v_0)\left(1+k_1r^2+k_2r^4+k_3r^6\right)\end{bmatrix}}_{\text{radial}} + \underbrace{\begin{bmatrix}2p_1\left(u-u_0\right)\left(v-v_0\right)+p_2\left[r^2+2\left(u-u_0\right)^2\right] \\ p_1\left[r^2+2\left(v-v_0\right)^2\right]+2p_2\left(u-u_0\right)\left(v-v_0\right)\end{bmatrix}}_{\text{tangencial}} \tag{4.17}$$

De este modo, considerando el modelo de distorsión completo se obtienen cinco parámetros adicionales los cuales también son considerados como parámetros intrínsecos.

$$\boldsymbol{D} = \begin{bmatrix}k_1 & k_2 & p_1 & p_2 & k_3\end{bmatrix}^T \tag{4.18}$$

4.1.3. Calibración de la cámara

El proceso de calibración consiste básicamente en obtener los parámetros intrínsecos y extrínsecos del modelo de la cámara. Conociendo las coordenadas en píxeles de los puntos en la imagen $\tilde{\boldsymbol{p}}$ y sus correspondientes coordenadas tridimensionales $\tilde{\boldsymbol{P}}$, es posible calcular los parámetros K, R y $\boldsymbol{t}$ al resolver la ecuación (4.15).

4.1.3.1. Calibración de la cámara por medio de mallas planares

Este método consiste en ubicar frente a la cámara un objeto con cierto número de puntos de los cuales se conocen sus coordenadas tridimensionales, comúnmente se suele utilizar una cuadrícula semejante a un tablero de ajedrez (aunque no es el único) debido a que resulta más fácil efectuar la detección de las esquinas. Esta técnica requiere que la cámara capture varias imágenes de la cuadrícula con diferentes posiciones y orientaciones (mínimo 6), cuidando que dicho tablero ocupe la mayor parte del campo de visión de la cámara.

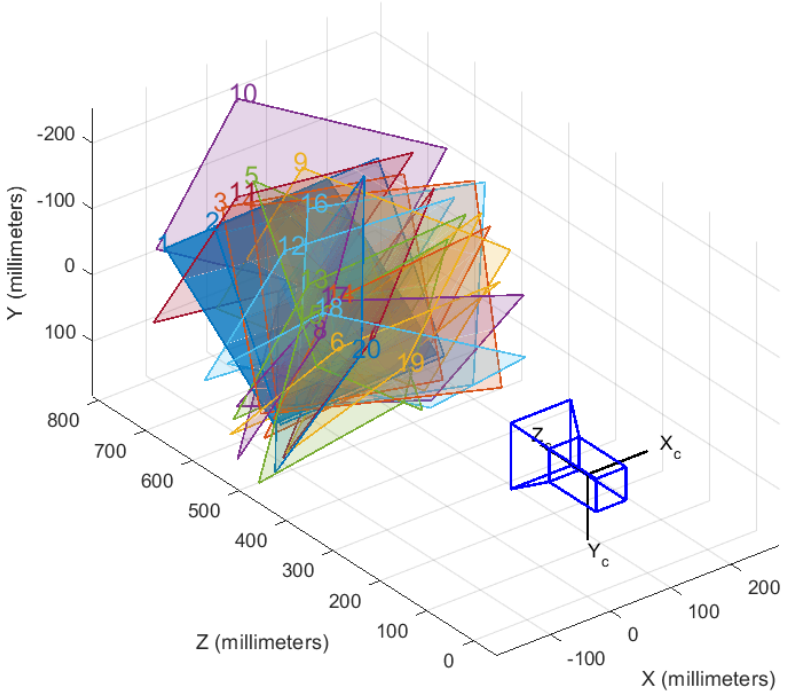


Figura 4.8: Calibración por medio de mallas planares.

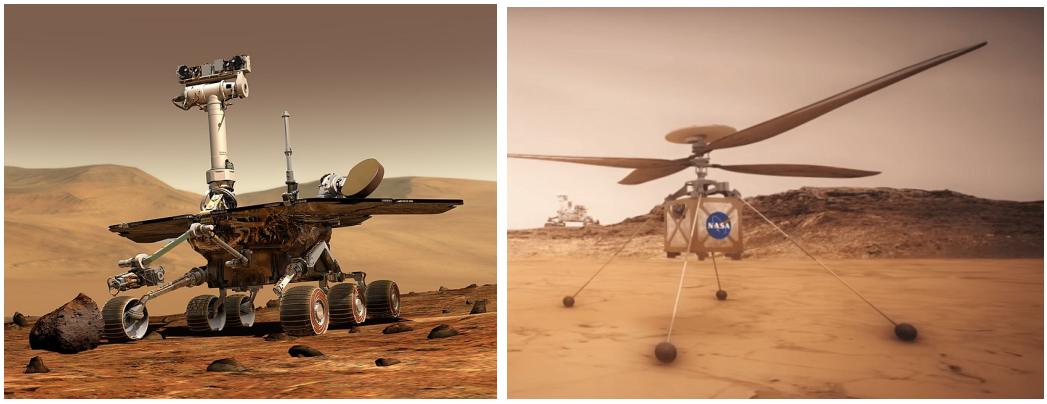
Empleando algoritmos de optimización es posible determinar los parámetros intrínsecos y extrínsecos previamente descritos. La precisión de la calibración incrementa a medida que se aumenta el número de imágenes utilizadas. Estos algoritmos han sido implementados de forma exitosa en plataformas como MATLAB[1] y OpenCV[2].

4.2. Odometría Visual

La odometría visual consiste básicamente en estimar la pose de un agente teniendo como entrada únicamente imágenes proporcionadas por una o varias cámaras. El término fue elegido por su similitud a la odometría convencional, la cual de manera incremental estima la pose de un vehículo al integrar el número de vueltas de las ruedas. La odometría visual opera del mismo modo al estimar de forma incremental la pose de un vehículo examinando los cambios que el movimiento induce en las imágenes. La ventaja que ofrece la odometría visual con respecto a la odometría convencional es que la primera no se ve afectada por el deslizamiento de las ruedas. A menudo la odometría visual es usada en conjunto con otro tipo de sensores (Encoders, GPS, IMU y Láseres) para concebir sistemas de localización más precisos. En años recientes ha incrementado el interés de la comunidad científica por dotar a robots con sistemas de odometría visual para volverlos completamente autónomos. El éxito de esta tecnología es tal, que robots exploradores en Marte han sido dotados con este tipo de sistemas, tal es el caso de los rovers *Spirit*

[1] http://www.vision.caltech.edu/bouguetj/calib_doc/
[2] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

y *Opportunity* [36], [37], además del vehículo aéreo *Mars Helicopter Ingenuity* el cual incorpora un sistema de odometría visual inercial.



(a) Rover Spirit **(b)** Dron Ingenuity

Figura 4.9: Robots exploradores que incorporan sistemas de odometría visual.

Un aspecto en contra de los sistemas de odometría visual es que la trayectoria estimada solo se puede recuperar a un factor de escala. Afortunadamente, la escala absoluta se puede determinar a partir de mediciones directas, restricciones de movimiento o mediante la integración con otros sensores.

4.2.1. Antecedentes

El problema de estimar la pose de un vehículo a partir de una entrada visual comenzó a principios de la década de 1980 y fue descrito por Hans Moravec (Figura 4.10) en su tesis doctoral[3]. Durante el periodo de 1980-2000 la mayor parte de las investigaciones se realizaron para su implementación en robots exploradores y fue motivada por el programa de exploración espacial de Marte de la NASA en un esfuerzo para proporcionar a los rovers la habilidad de efectuar la estimación de sus estados.

Figura 4.10: Hans Moravec.

[3]https://frc.ri.cmu.edu/~hpm/project.archive/robot.papers/1975.cart/1980.html.thesis/index.html

No fue sino hasta 2004 cuando la comunidad científica retomó el interés en esta área con el artículo de David Nister [38] en donde se acuñó por primera vez el término "Odometría Visual". A partir de este año ha habido un gran incremento en el desarrollo e implementación de algoritmos de odometría visual, en parte debido al aumento en la capacidad de procesamiento de los computadores. Actualmente existe un enfoque que tiene como objetivo el desarrollo de sistemas de odometría visual para vehículos aéreos.

4.2.2. Formulación del problema

En este caso el análisis se llevará a cabo considerando un sistema de odometría visual monocular. Para extender al caso estéreo basta con considerar la distancia existente entre ambas cámaras, tomando como origen la cámara del lado izquierdo. Un agente se mueve a través de un entorno tomando imágenes con una cámara en instantes de tiempo discreto k. En el caso de un sistema monocular, el conjunto de imágenes tomadas en los tiempos k estará denotado por $I_{0:n} = \{I_0, \ldots, I_n\}$ [39], [40]. La Figura 4.11 ilustra esta configuración.

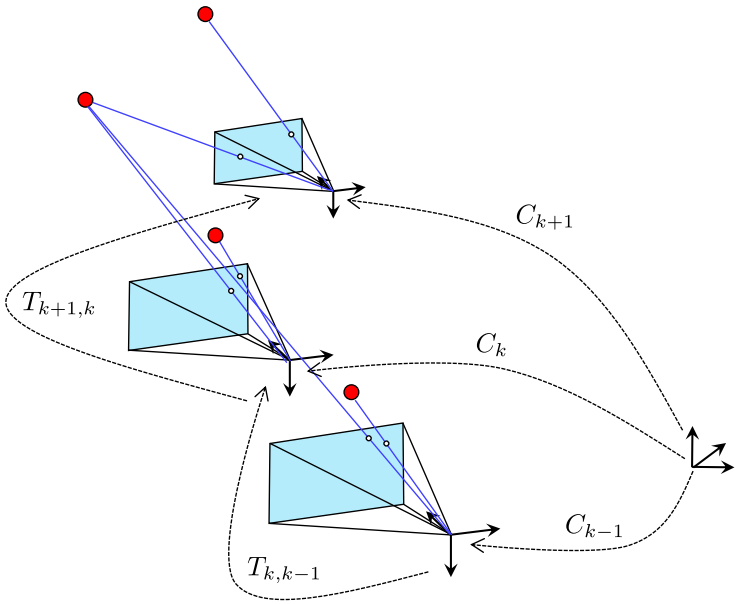


Figura 4.11: Ilustración del problema de Odometría Visual. Las poses relativas $T_{k,k-1}$ de las posiciones adyacentes de la cámara son calculadas a partir de características visuales y posteriormente concatenadas para obtener la pose absoluta C_k con respecto al sistema de coordenadas inicial en $k = 0$.

Por simplicidad, se considera que el sistema de coordenadas de la cámara es también el sistema de coordenadas del agente. Dos posiciones de la cámara en instantes de tiempo adyacentes $k-1$ y k están relacionadas por la transformación $T_{k,k-1} \in \mathbb{R}^{4\times4}$ de la siguiente forma:

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & \boldsymbol{t}_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{4.19}$$

Donde $R_{k,k-1} \in SO(3)$ es la matriz de rotación, y $\boldsymbol{t}_{k,k-1} \in \mathbb{R}^{3\times1}$ el vector de traslación. El conjunto $T_{1:n} = \{T_{1,0}, \ldots, T_{n,n-1}\}$ contiene todos los movimientos subsecuentes.

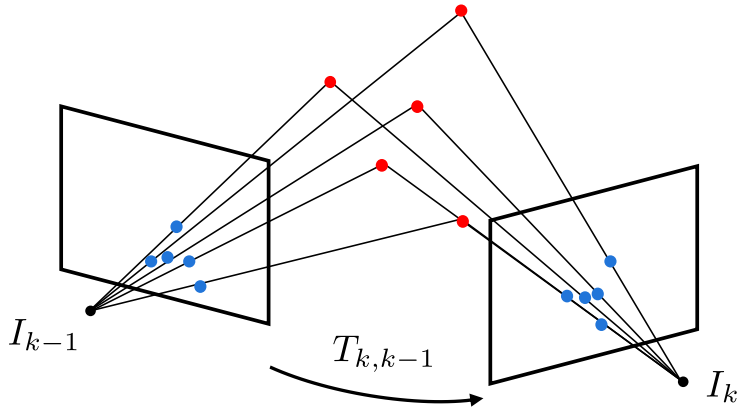


Figura 4.12: Transformación entre dos imágenes consecutivas.

Para simplificar la notación, desde ahora en adelante, se emplea T_k en lugar de $T_{k,k-1}$. El conjunto de poses de la cámara $C_{0:n} = \{C_0, \ldots, C_n\}$ contiene las transformaciones de la cámara con respecto al sistema de coordenadas inicial en $k = 0$. La pose actual C_n puede ser calculada concatenando todas las transformaciones T_k donde $k = 1 \ldots n$. Por lo tanto:

$$C_n = C_{n-1} T_n \tag{4.20}$$

Con C_0 siendo la pose de la cámara en el instante $k = 0$. Después de este paso se puede realizar un refinamiento iterativo sobre las últimas m poses para obtener una estimación más precisa de la trayectoria local.

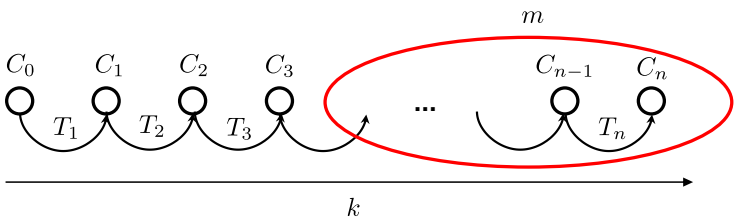


Figura 4.13: Refinamiento iterativo sobre las últimas m poses.

Este refinamiento iterativo funciona al minimizar la suma de los cuadrados de los errores de reproyección de los puntos 3D reproyectados en las últimas m imágenes, conocido como *windowed-bundle adjustment*. Los puntos tridimensionales se obtienen por triangulación de los puntos detectados en cada imagen.

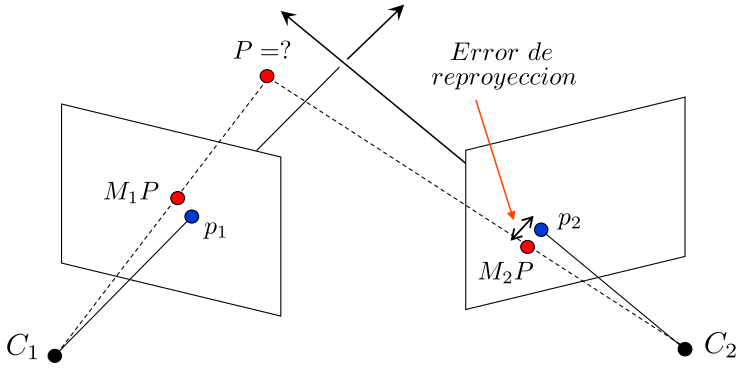


Figura 4.14: Error de reproyección.

El diagrama a bloques del algoritmo de odometría visual se resume en la Figura 4.15. Para cada nueva imagen I_k, el primer paso consiste en detectar las características más sobresalientes de la imagen. Para el segundo paso se puede emplear ya sea la correspondencia entre los puntos de la imagen actual I_k con los detectados en la imagen anterior I_{k-1} o el seguimiento de los mismos en imágenes posteriores. El tercer paso consiste en calcular el movimiento relativo T_k entre las imágenes I_{k-1} e I_k (Figura 4.12). Para el caso monocular unicamente se puede estimar el movimiento a partir de correspondencias 2D-2D debido a que de antemano se supone que no se tiene conocimiento de las dimensiones de objetos 3D en la escena. La pose de la cámara C_k se calcula mediante la concatenación de T_k con las poses anteriores. Finalmente, se puede realizar un refinamiento de las últimas m imágenes para obtener una estimación más precisa de la trayectoria.

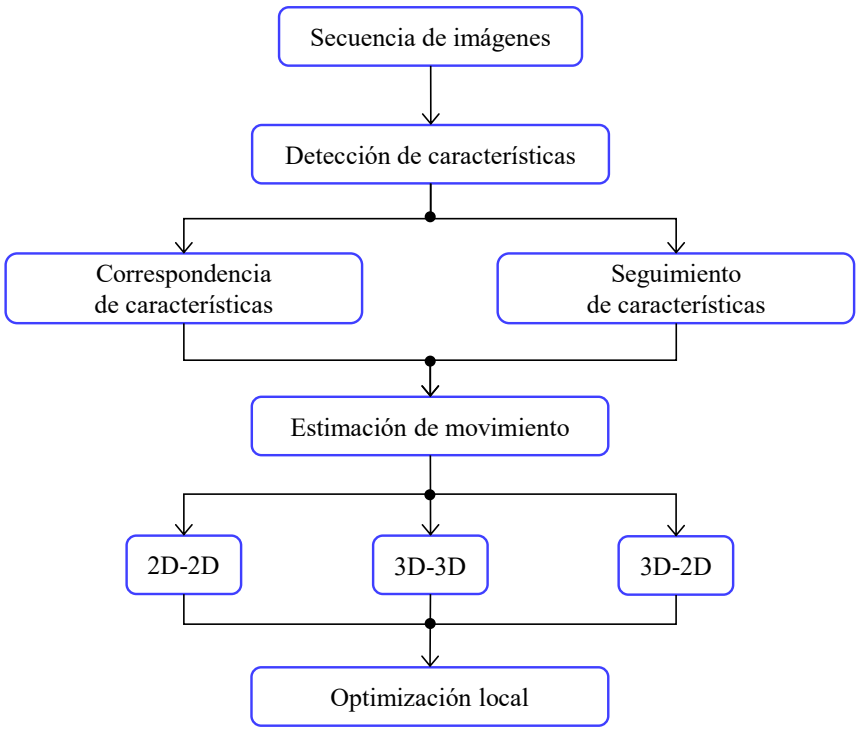


Figura 4.15: Diagrama a bloques que muestra los componentes principales de un sistema basado en odometría visual.

4.2.3. Detección de características

El concepto de puntos de interés se basa en la idea de trabajar con puntos específicos de una imagen para su análisis de forma local en lugar de procesar la imagen como un todo. Esta aproximación funciona bien mientras se cuente con suficientes puntos en la escena, además de que estos puntos sean lo suficientemente distinguibles para que puedan ser localizados de manera precisa. Idealmente los puntos de interés deberían ser detectados en la misma escena, sin importar el punto de vista, escala, u orientación desde la cual la imagen haya sido tomada. Existen dos enfoques principales para calcular el movimiento relativo T_k: métodos basados en apariencia, los cuales utilizan la información de la intensidad de todos los píxeles en la imagen, y métodos basados en características, los cuales solo utilizan características destacadas y repetibles. Los métodos basados en apariencia son menos precisos que los métodos basados en características y demandan mayor capacidad de cómputo. Los métodos basados en características requieren de la capacidad de relacionar (o rastrear) las características a través de las imágenes, pero resultan ser más rápidos y precisos. Por consecuencia, la mayoría de las implementaciones de odometría visual emplean métodos basados en características.

4.2.3.1. Detectores de características

Uno de los detectores de esquinas más conocido es el desarrollado por Chris Harris y Mike Stephens en 1988 [41]. El cual básicamente se encarga de calcular la diferencia en intensidad para un desplazamiento el todas direcciones, a partir de los valores propios obtenidos en cierta región, es posible detectar esquinas. El detector *GFTT (Good Features To Track)* propuesto por Shi y Tomasi [42] es una versión mejorada del detector de Harris, el cual hace que las esquinas detectadas se encuentren distribuidas de manera más uniforme dentro de la imagen, además evita el uso del parámetro k.

(a) Detector Harris **(b)** Detector GFTT

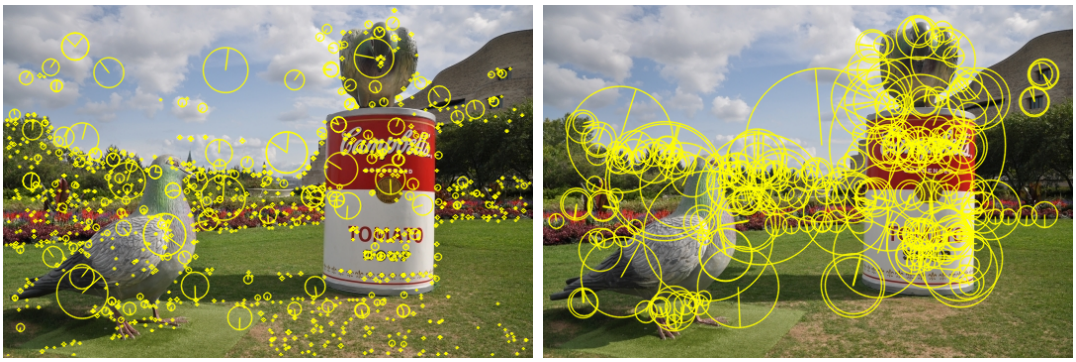
Figura 4.16: Detectores basados en el algoritmo de Harris

Los detectores anteriores constituyen una solución sencilla al problema de detección, sin embargo, ambos demandan considerable carga computacional, ya que el algoritmo en sí implica el uso de derivadas. El operador *FAST (Features from Accelerated Segment Test)* [43] fue concebido para realizar una rápida detección de los puntos de interés, su

algoritmo se basa en la comparación rápida de un conjunto pequeño de píxeles. Este algoritmo es recomendado para aplicaciones donde varios puntos deben ser detectados de forma simultánea y de manera eficiente.

Figura 4.17: Detector FAST

Una de las propiedades más importantes de los detectores es la invarianza al cambio en la escala, es decir, la capacidad de reconocer características sin importar el cambio en la escala en imágenes consecutivas. Dentro de los detectores invariantes ante el cambio en la escala sobresale el operador *SIFT Scale-Invariant Feature Transform* [44]. Como se aprecia en la Figura 4.18 los círculos trazados son proporcionales a la escala de cada característica detectada, de forma similar la línea en el interior de los círculos índica la orientación de cada característica. El detector *SURF Speeded Up Robust Features* [45] parte del detector *SIFT*, sin embargo, este emplea distintos algoritmos que reducen la carga computacional lo que da lugar a un operador más eficiente. Ambos detectores poseen robustos descriptores que permiten reconocer los mismos puntos de interés en imágenes subsecuentes. Es importante mencionar que tanto el detector *SIFT* como el detector *SURF* se encuentran patentados por lo que su uso comercial se encuentra sujeto a términos y licencias.

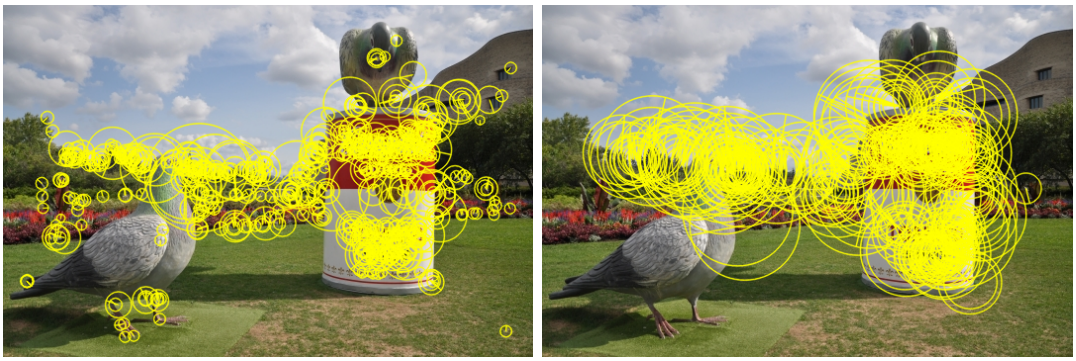


(a) Detector SIFT **(b)** Detector SURF

Figura 4.18: Detectores patentados

Recientemente se han propuesto un par de detectores que cumplan con la tarea de una rápida detección así como invarianza a los cambios en la escala, el detector *BRISK*

Binary Robust Invariant Scalable Keypoints [46] se basa en el operador *FAST* para la detección y para la propiedad de la invarianza a la escala emplea un algoritmo similar al de *SIFT*, la desventaja que presenta es su alto costo computacional. *ORB Oriented FAST and Rotated BRIEF* [47] es otro tipo de detector eficiente e invariante al cambio de escala el cual comparte características similares al detector *BRISK*. Ambos detectores surgen como una alternativa libre a los operadores *SIFT* y *SURF*.



(a) Detector BRISK **(b)** Detector ORB

Figura 4.19: Detectores invariantes al cambio en la escala

4.2.3.2. Descripción y correspondencia entre características

Una vez detectados los puntos de interés dentro de una imagen, el siguiente paso consiste en detectar las mismas características en la imagen consecuente. Ciertamente un solo píxel no es suficiente para realizar una comparación entre dos características, es por ello que se sugiere efectuar el análisis considerando un área alrededor de dichas características, si las dos áreas corresponden al mismo elemento en la escena, entonces, uno podría esperar que los píxeles exhiban valores similares.

Figura 4.20: Correspondencias entre características

Algunos de los detectores previamente descritos cuentan con descriptores los cuales no son más que vectores de 1 o 2 dimensiones compuestos de valores binarios, enteros, o de

punto flotante que describen el punto de interés y su vecindario. Un buen descriptor debe ser capaz de representar de manera única a cada una de las características detectadas en la imagen; debe ser robusto a posibles cambios en la iluminación o variaciones geométricas. Una de las aplicaciones más comunes que emplea detectores de características en conjunto con descriptores es la correspondencia entre imágenes subsecuentes.

4.2.3.3. Flujo óptico

El flujo óptico se define como el patrón de aparente movimiento de objetos dentro de dos imágenes consecutivas causado por el desplazamiento de la cámara. En [48] Bruce Lucas y Takeo Kanade describen detalladamente un algoritmo que permite implementar este procedimiento tomando en cuenta las siguientes consideraciones:

- La intensidad de los píxeles que conforman un objeto no cambian en imágenes consecutivas.
- Píxeles en un vecindario aparentan movimiento similar.

Figura 4.21: Flujo óptico

Una de las aplicaciones más interesantes de esta técnica se encuentra relacionada con el seguimiento de puntos de interés a través de imágenes. Para iniciar el proceso de seguimiento, primeramente se deben detectar las características en una primera imagen, a continuación, empleando el algoritmo de flujo óptico estimar la ubicación de dichas características en las imágenes subsecuentes. A medida que la escena cambie completamente con respecto a la imagen inicial, inevitablemente llegará el momento en que se pierda el rastro de algunas de características iniciales, por tal motivo es buena idea detectar nuevas características cada vez que el número de puntos de interés se reduzca por debajo de un umbral preestablecido.

4.2.4. Estimación de movimiento 2D-2D

En esta etapa, se calcula el movimiento de la cámara entre la imagen actual y la imagen anterior.

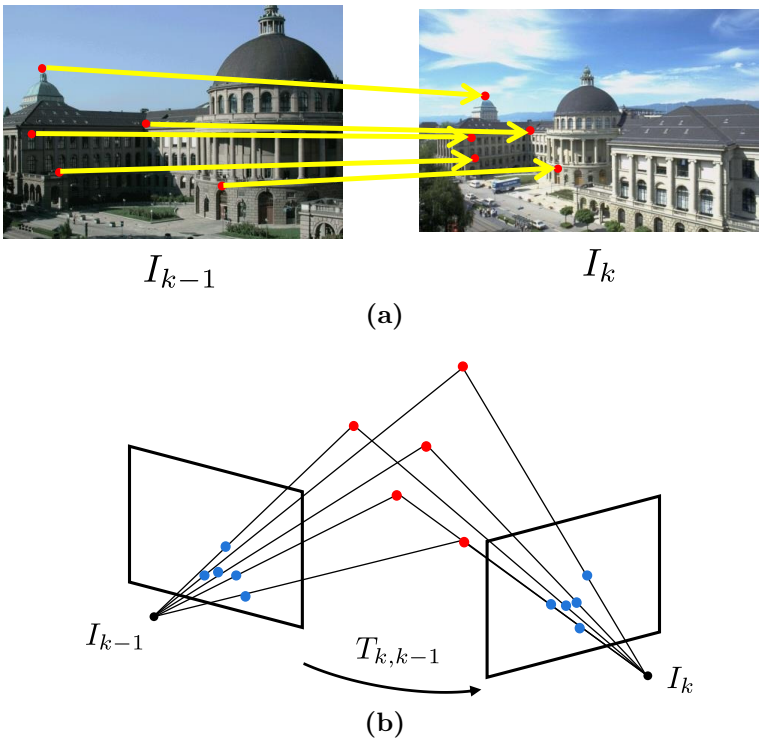


Figura 4.22: Estimación de movimiento a partir de correspondencias 2D-2D.

Mediante la concatenación de todos estos movimientos individuales, se puede recuperar la trayectoria completa de la cámara y por lo tanto del agente. A continuación se explica cómo la transformación T_k entre dos imágenes I_{k-1} e I_k se puede calcular a partir de dos conjuntos de características f_{k-1}, f_k. Aquí únicamente se abordará el caso en el que las correspondencias se especifican en 2D-2D (puntos representados en coordenadas de las imágenes) debido a que el sistema en cuestión incorpora una cámara monocular.

4.2.4.1. Cálculo de la matriz esencial

Las relaciones geométricas entre el par de imágenes I_k e I_{k-1} de una cámara calibrada están descritas por la denominada matriz esencial $E \in \mathbb{R}^{3\times3}$. Esta contiene los parámetros de movimiento de la cámara:

$$E_k \simeq \hat{t}_k R_k \tag{4.21}$$

Donde:

$$\hat{t}_k = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \qquad R_k = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.22}$$

El símbolo $\simeq$ es usado para denotar que la igualdad es válida a un factor de escala. La matriz esencial puede ser calculada a partir de correspondencias 2D-2D para posteriormente extraer la rotación y traslación directamente de E. La estimación del movimiento 2D-2D hace uso de la restricción epipolar la cual se describe a continuación:

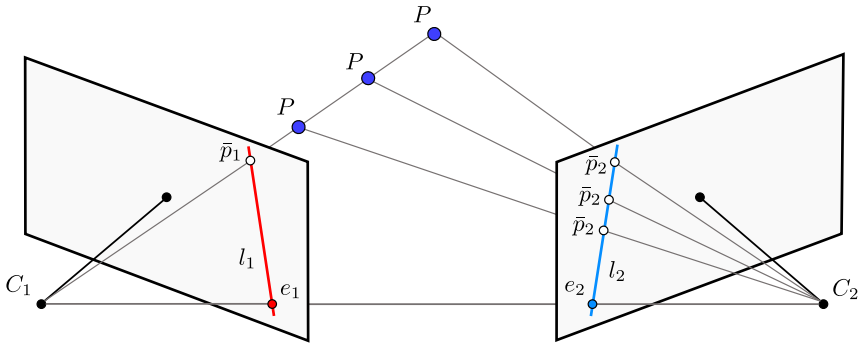


Figura 4.23: Restricción epipolar.

Considérese el esquema de la Figura 4.23, utilizando únicamente la imagen de la izquierda C_1 es imposible hallar el correspondiente punto 3D P de $\bar{p}_1$ debido a que cada punto sobre la línea C_1P se proyecta sobre el mismo punto de la imagen. Al añadir una nueva imagen ahora varios puntos de la línea C_1P proyectan diferentes puntos $\bar{p}_2$ en la segunda imagen. Con las dos imágenes es posible obtener la correcta ubicación de P mediante triangulación. La proyección de distintos puntos sobre C_1P forman una línea en la segunda imagen l_2 conocida como línea epipolar correspondiente al punto $\bar{p}_1$. Lo anterior significa que para encontrar el punto $\bar{p}_1$ en la segunda imagen basta con buscar a lo largo de la línea l_2 lo mismo aplica con cada punto cada uno con su correspondiente línea epipolar. Lo anterior se conoce como restricción epipolar. El plano formado por C_1PC_2 es llamado plano epipolar. En la imagen 4.23 C_1 y C_2 son los centros de las cámaras, la proyección de C_2 aparece en la primera imagen e_1 conocido como epípolo de forma inversa e_2 es la proyección de C_1 en la segunda imagen. En algunos casos no será posible ubicar el epípolo dentro la imagen, el cual podría estar fuera de ella. Todas las líneas epipolares deben pasar por su correspondiente epípolo. La restricción epipolar se define de la siguiente manera:

$$\bar{p}_2^T E \bar{p}_1 = 0 \tag{4.23}$$

Donde $\bar{p}_2$ es la ubicación de una característica en la imagen actual I_k y $\bar{p}_1$ es la ubicación de la misma característica en la imagen previa I_{k-1}. $\bar{p}_1$ y $\bar{p}_2$ son coordenadas de la imagen normalizadas de la forma: $\bar{p} = [\bar{u}, \bar{v}, 1]^T$. La matriz esencial se calcula a partir de correspondencias 2D-2D haciendo uso de la restricción epipolar. Una solución simple y directa para $n \geq 8$ puntos no coplanares es el algoritmo de ocho puntos de Longuet-Higgins [49], el cual se muestra a continuación. Cada correspondencia entre dos puntos genera una restricción de la siguiente forma:

$$\begin{bmatrix} \bar{u}_2\bar{u}_1 & \bar{u}_2\bar{v}_1 & \bar{u}_2 & \bar{v}_2\bar{u}_1 & \bar{v}_2\bar{v}_1 & \bar{v}_2 & \bar{u}_1 & \bar{v}_1 & 1 \end{bmatrix} E = 0 \tag{4.24}$$

Donde:

$$E = \begin{bmatrix} e_{11} & e_{12} & e_{13} & e_{21} & e_{22} & e_{23} & e_{31} & e_{32} & e_{33} \end{bmatrix}^T \tag{4.25}$$

Al apilar restricciones de ocho puntos se obtiene el sistema de ecuaciones lineal $QE = 0$. Este sistema de ecuaciones homogéneo se puede resolver utilizando la técnica de descomposición en valores singulares (SVD) y con ello obtener los parámetros de E. Tener más de ocho puntos conlleva a tener un sistema sobre-determinado. La descomposición en valores singulares de Q tiene la forma $Q = USV^T$, y la estimación por mínimos cuadrados de E con $\|E\| = 1$ se puede encontrar en la última columna de V. Sin embargo, esta estimación lineal de E no cumple con las restricciones internas de una matriz esencial, las cuales provienen de la multiplicación de la matriz de rotación R y la matriz de traslación antisimétrica $\hat{t}$. Estas restricciones son visibles en los valores singulares de la matriz esencial. Una matriz esencial válida después de la descomposición en valores singulares es $E = USV^T$ y tiene su diagonal $\text{diag}(S) = \{s, s, 0\}$, lo que significa que los valores singulares primero y segundo son iguales y el tercero es cero. Para obtener una matriz esencial E válida que cumpla con las restricciones, la solución debe proyectarse en el espacio de matrices esenciales válidas. La matriz esencial proyectada es $\bar{E} = U\,\text{diag}\,\{1, 1, 0\}\,V^T$. Observe que la solución del algoritmo de ocho puntos se degenera cuando los puntos 3D son coplanares. Finalmente, observe que el algoritmo de ocho puntos funciona tanto para cámaras calibradas (perspectiva u omnidireccional) como para no calibradas (solo perspectiva).

4.2.4.2. Extracción de R y t a partir de E

A partir de la estimación de E, se pueden extraer las partes de rotación y traslación. En general, hay cuatro soluciones diferentes para R, t para una matriz esencial; sin embargo, mediante la triangulación de un solo punto, se puede identificar el par R, t adecuado. Las cuatro soluciones son:

$$\begin{aligned} R &= U(\pm W^T)V^T \\ \hat{t} &= U(\pm W)SU^T \end{aligned} \tag{4.26}$$

Donde:

$$W^T = \begin{bmatrix} 0 & \pm 1 & 0 \\ \mp 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.27}$$

La solución correcta es en la que el punto triangulado se encuentra frente a las dos cámaras (Figura 4.24a). Después de seleccionar la solución correcta se debe realizar una optimización no lineal de los parámetros de rotación y traslación utilizando la estimación de R, t como valores iniciales.

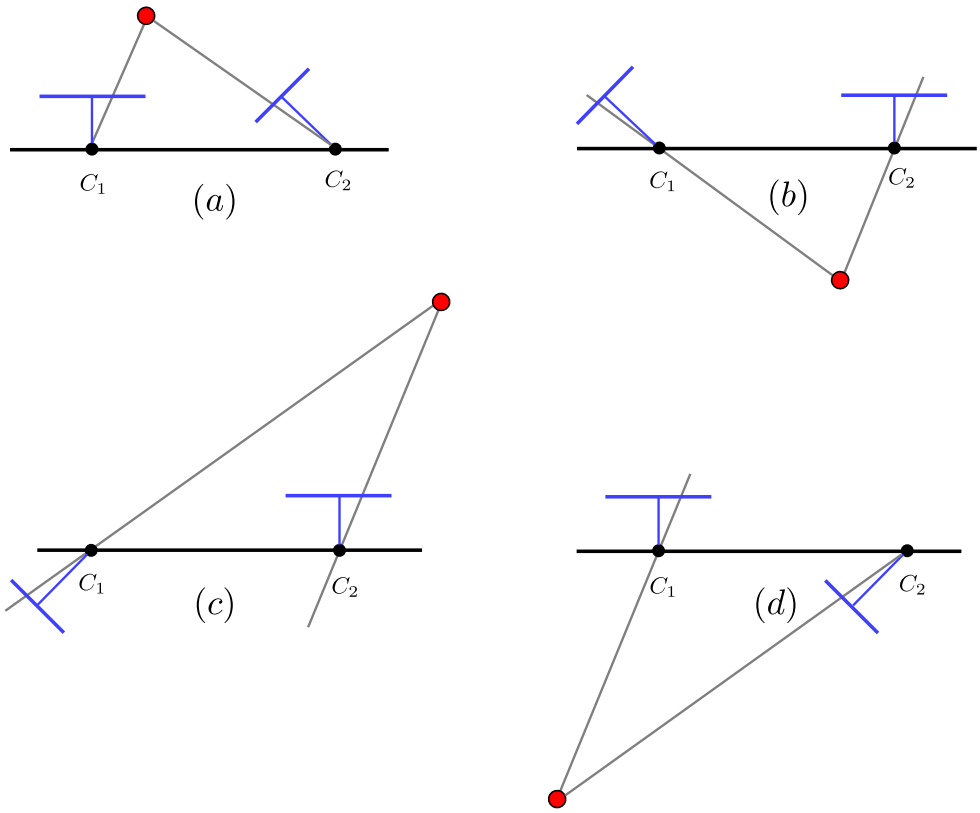


Figura 4.24: Posibles soluciones para R y t.

4.2.4.3. Cálculo de la escala relativa

Para recuperar la trayectoria de una secuencia de imágenes, las diferentes transformaciones $T_{0:n}$ deben ser concatenadas. Para esto, las escalas relativas deben calcularse de forma apropiada, ya que la escala absoluta de la traslación no se puede calcular solamente con dos imágenes. Sin embargo, es posible calcular las escalas relativas para las transformaciones posteriores. Una forma de hacerlo es triangular los puntos 3D de X_{k-1} y X_k a partir de dos pares de imágenes. A partir de los puntos 3D correspondientes, se pueden calcular las distancias relativas entre cualquier combinación de dos conjuntos de puntos 3D. La escala apropiada se puede determinar a partir de la relación de la distancia r entre un par de puntos en X_{k-1} y un par en X_k.

$$r = \frac{\|X_{k-1,i} - X_{k-1,j}\|}{\|X_{k,i} - X_{k,j}\|} \tag{4.28}$$

Por robustez, se calculan las relaciones de escala para muchos pares de puntos y se utiliza la media (o en presencia de valores atípicos, la mediana). El vector de traslación t se escala con esta relación de distancia. Observe que el cálculo de escala relativa requiere que las características se asocien (o rastreen) en múltiples imágenes (al menos tres). En lugar de realizar una triangulación explícita de los puntos 3D, la escala también se puede recuperar aprovechando la restricción trifocal entre correspondencias de características 2D en tres vistas. Por último, el algoritmo de odometría visual para correspondencias 2D-2D se resume en el Algoritmo 1.

Algoritmo 1. Odometría visual a partir de correspondencias 2D-2D

1: Capturar una nueva imagen I_k.
2: Extraer y relacionar características entre I_{k-1} e I_k.
3: Calcular la matriz esencial para el par de imágenes I_{k-1}, I_k.
4: Descomponer la matriz esencial en R_k y t_k, y formar a T_k.
5: Calcular la escala relativa y escalar a t_k.
6: Concatenar la transformación calculando a $C_k = C_{k-1}T_k$.
7: Repetir desde 1.

5 Plataforma experimental

El sistema de control se implementará dentro del entorno de ROS. La versión que se empleará en este trabajo será la de ROS Kinetic Kame debido a que es una versión de soporte prolongado. La versión de ROS indicada previamente se instalará dentro del sistema operativo Ubuntu 16.04 LTS. La plataforma de hardware será el cuadricóptero Bebop 2 de la compañía Francesa Parrot.

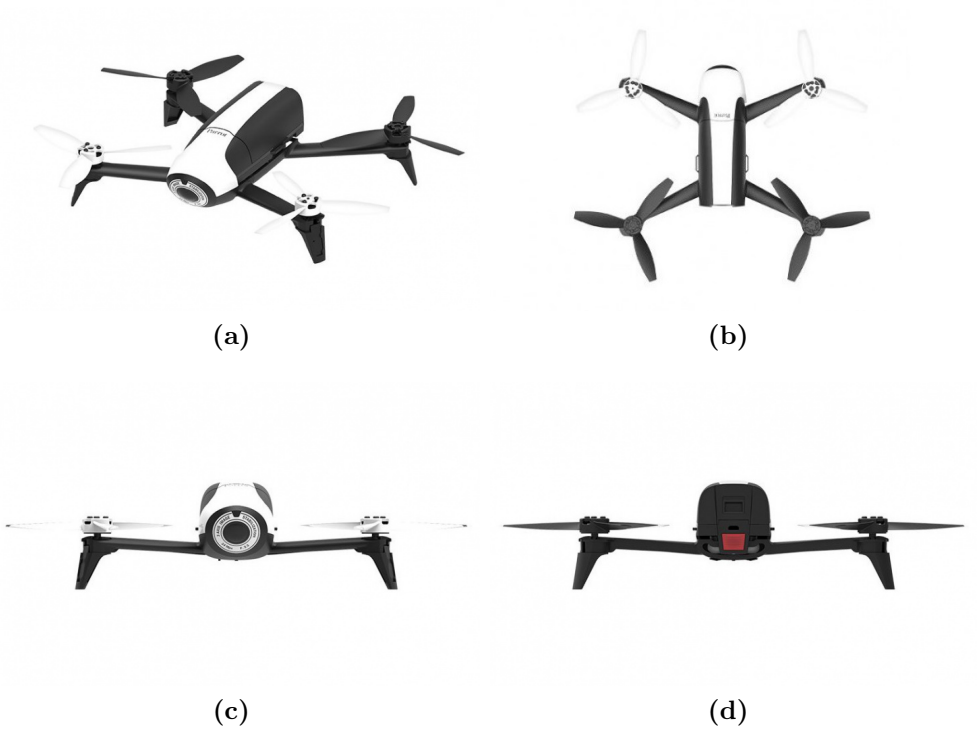
5.1. Dron Bebop 2

Parrot es una compañía de telecomunicaciones con sede en París, Francia, la cual produce productos tales como el cuadricóptero Bebop 2. Este modelo fue lanzado al mercado en noviembre del 2016, el cual resulta ser la versión mejorada de su antecesor el Bebop 1, una comparación entre ambos modelos se puede encontrar en el sitio web[1]. En el sitio oficial[2] del Bebop 2 se puede encontrar una descripción más detallada sobre las características de este modelo, a continuación se enlistan las más sobresalientes.

Característica	**Valor**
Precio aproximado	550 USD
Autonomía de vuelo	25 minutos
Velocidad horizontal máxima	16.6 m/s
Velocidad vertical máxima	5.8 m/s
Alcance de la señal Wifi	hasta 300 metros
Cámara	14 Megapíxeles; FOV: 180°
Peso	500 gramos
Dimensiones	38.2 cm x 32.8 cm x 9 cm
Diámetro de las hélices	15.2 cm

Tabla 5.1: Especificaciones generales del Bebop 2.

[1]https://blog.parrot.com/2016/01/12/comparison-bebop-2-vs-bebop-drone/
[2]https://www.parrot.com/es/drones/parrot-bebop-2



(a) **(b)**

(c) **(d)**

Figura 5.1: Dron Bebop 2 de la compañía Parrot.

Este dron tiene un peso de 500 gramos de aspecto compacto y diseño aerodinámico con una autonomía de 25 minutos. Sus prestaciones le permiten volar y filmar al mismo tiempo con excelente resolución, tanto en interiores como en exteriores. La cámara incorporada de 14 megapíxeles permite filmar en Full HD 1080p con un ángulo de captura de 180° en horizontal y vertical [50].

Figura 5.2: Dron Bebop 2 en pleno vuelo.

Este vehículo puede alcanzar una velocidad máxima de 60 km/h en horizontal y 21 km/h en vertical. Puede alcanzar su velocidad máxima en 14 segundos y resistir vientos de hasta 60 km/h. Además, puede detenerse por completo en poco más de 4 segundos [50].

5.2. ROS

ROS *Robot Operating System* o sistema operativo robótico es un *framework* para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo. ROS es considerado más bien como un meta-sistema operativo, ya que es instalado sobre un sistema operativo ya existente. ROS se desarrolló originalmente en 2007 bajo el nombre de *switchyard* por el Laboratorio de Inteligencia Artificial de Stanford. ROS provee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros. La librería está orientada para el sistema Ubuntu Linux, a pesar de que existen adaptaciones para otros S.O. esta no garantiza su estabilidad. ROS está bajo la licencia open source, BSD *Berkeley Software Distribution* [51].

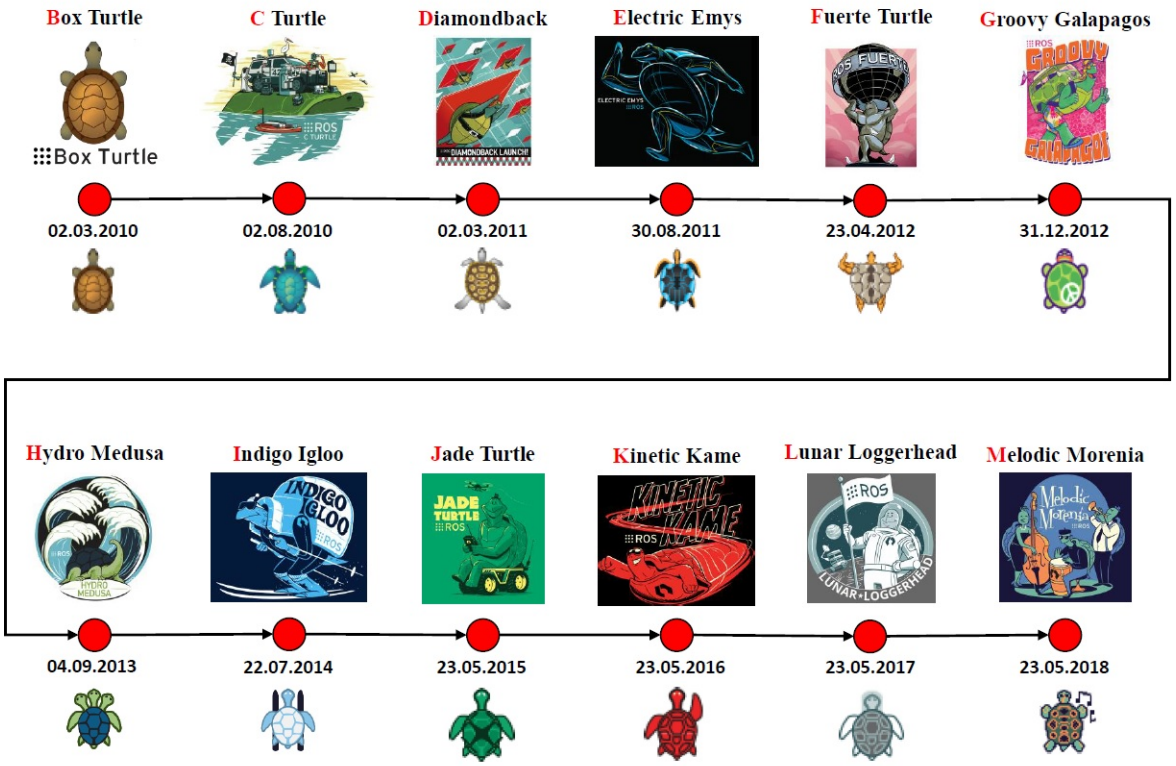
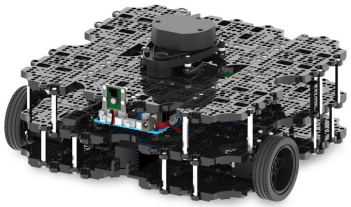


Figura 5.3: Versiones de ROS.

Normalmente las versiones de ROS están referidas por un sobrenombre en vez de por una versión numérica. Dichas versiones se encuentran ordenadas de forma alfabética tomando como referencia la primera letra del sobrenombre [52]. Debido al éxito de ROS, distintas instituciones y compañías han comenzado a adaptar sus productos para ser usados en ROS. Existe una larga lista de robots compatibles con ROS, mismos que se pueden

encontrar en el sitio oficial de ROS[3]. En dicho sitio se pueden encontrar una gran variedad de robots los cuales se subdividen en 4 categorías principales: Aéreos, Terrestres, Marinos y Manipuladores. En la Figura 5.4 se muestran algunos de estos robots.



(a) Robot móvil Turtlebot3 Waffle PI

(b) Dron Bebop 2

(c) Robot industrial Baxter

(d) Vehiculo terrestre Husky

Figura 5.4: Ejemplos de plataformas de hardware compatibles con ROS.

5.2.1. Terminología

5.2.1.1. Nodos

Un nodo[4] se puede pensar como un programa ejecutable. ROS recomienda crear un nodo para cada tarea con el afán de promover su reusabilidad. Por ejemplo, para el caso de un robot móvil, el programa principal se suele subdividir en funciones específicas tales como: percepción, navegación, control y reconocimiento de obstáculos. Un nodo necesita registrase ante el maestro para iniciar la transmisión de información por medio de Tópicos, Servicios o Acciones. Los nodos que deseen recibir información necesitan solicitarlo ante el maestro para que este habilite la transmisión de datos (Figura 5.5).

[3]https://robots.ros.org/
[4]http://wiki.ros.org/Nodes

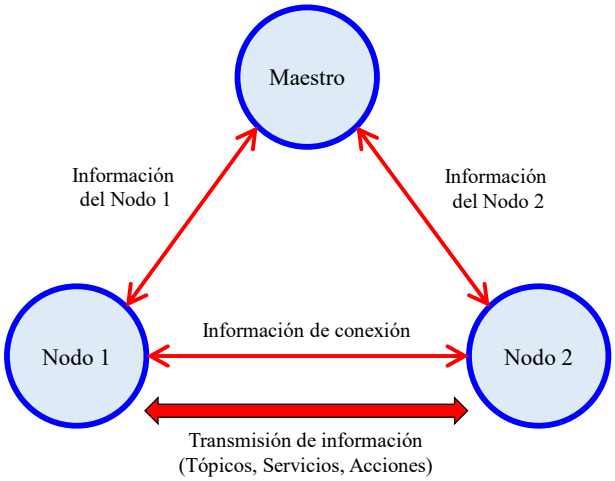


Figura 5.5: Metodología de comunicación entre nodos.

5.2.1.2. Tópicos

La comunicación a través de tópicos[5] necesita de un nodo publicador y un nodo subscriptor ambos con el mismo tipo de mensaje. Es importante que ambos nodos se refieran al mismo tópico. El nodo subscriptor se conecta directamente al nodo publicador para iniciar la transmisión de información. La comunicación mediante tópicos es un tipo de comunicación unidireccional asíncrona tal como se aprecia en la Figura 5.6.

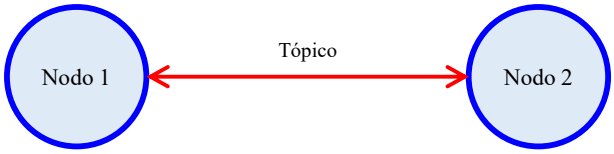


Figura 5.6: Comunicación a través de tópicos.

5.2.1.3. Servicios

Un servicio[6] es una forma de comunicación bidireccional síncrona de tipo cliente-servidor. El proceso inicia con la solicitud del cliente para un servicio en particular (generalmente para la ejecución de una tarea) y al término se obtiene una respuesta del servidor indicando la finalización de la misma. La comunicación entre el cliente y el servidor finaliza una vez que la tarea ha sido completada. La Figura 5.7 muestra una descripción más gráfica.

[5] http://wiki.ros.org/Topics
[6] http://wiki.ros.org/Services

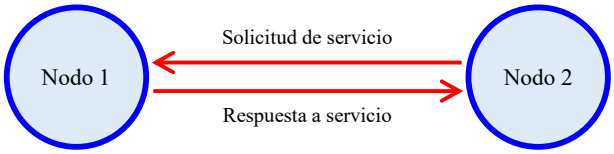


Figura 5.7: Comunicación a través de servicios.

5.2.1.4. Acciones

La comunicación entre nodos con base en acciones[7] se emplea cuando una tarea toma un largo tiempo en ser completada, por lo tanto es necesaria información adicional. Este método de comunicación es similar al de cliente-servidor con algunos componentes adicionales, los cuales nos permiten conocer el progreso de la tarea encomendada (Figura 5.8).

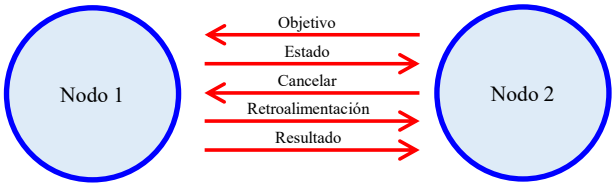


Figura 5.8: Comunicación a través de acciones.

5.2.2. Instalación de ROS Kinetic Kame

Para efectuar la instalación de ROS Kinetic Kame se da por hecho que se cuenta con un computador corriendo Ubuntu 16.04 LTS, no se recomienda hacer uso de una máquina virtual. La instalación de Ubuntu 16.04 se puede realizar siguiendo las instrucciones del sitio web[8], eligiendo adecuadamente la versión, misma que se puede descargar desde su página oficial[9]. Todo el proceso se lleva a cabo dentro de la terminal de Ubuntu por lo que es necesario tener conocimientos sobre los comandos básicos de Linux, en su defecto, se recomienda revisar tutoriales sobre el tema[10]. A continuación se enumeran los pasos a seguir para la correcta instalación de ROS Kinetic Kame, si se desean más detalles sobre el proceso de instalación, se recomienda visitar la página oficial de ROS[11].

[7] http://wiki.ros.org/actionlib
[8] https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop#0
[9] https://ubuntu.com/download/alternative-downloads
[10] http://www.ee.surrey.ac.uk/Teaching/Unix/
[11] http://wiki.ros.org/kinetic/Installation/Ubuntu

5.2.2.1. 1. Configurar repositorios

Para comenzar, es necesario configurar los repositorios de Ubuntu. Para ello se accede al administrador de Software de Ubuntu, en la pestaña **Software de Ubuntu** se elige la opción **Orígenes y actualizaciones**, y en **Software de Ubuntu** se marcan las casillas correspondientes de acuerdo a la Figura 5.9.

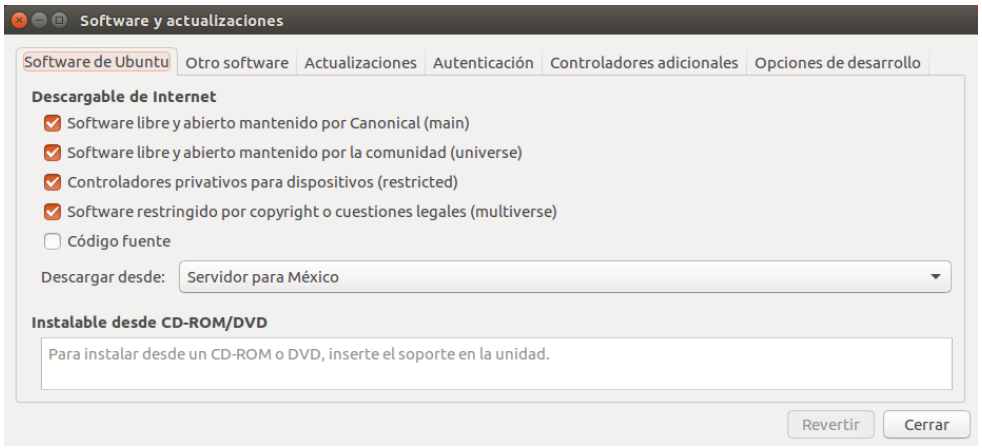


Figura 5.9: Configuración de los repositorios.

5.2.2.2. 2. Agregar repositorios

El siguiente comando añade los repositorios para la descarga de los paquetes necesarios.

```
$ sudo sh -c ‘echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
main" > /etc/apt/sources.list.d/ros-latest.list’
```

5.2.2.3. 3. Key

Este paso consiste en confirmar que el origen del código es correcto. Normalmente cuando se agrega un nuevo repositorio se tienen que añadir las llaves de dicho repositorio.

```
$ sudo apt-key adv --keyserver ‘hkp://keyserver.ubuntu.com:80’ --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Si se presentan problemas al conectar con el servidor se recomienda sustituir dentro de las comillas la URL `hkp://keyserver.ubuntu.com:80` por `hkp://pgp.mit.edu:80`.

5.2.2.4. 4. Instalando ROS Kinetic Kame

Antes de comenzar con la instalación se sugiere tener actualizado el sistema para evitar problemas con librerías y versiones de software.

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

El siguiente comando instalará la versión completa de ROS la cual incluye ROS, rqt, RViz, Gazebo y librerías relacionadas con percepción y navegación.

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

5.2.2.5. 5. Inicialización de rosdep

Antes de usar ROS se necesita inicializar rosdep, la cual es una herramienta que permite descargar e instalar paquetes externos.

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

5.2.2.6. 6. Configuración del entorno

Es conveniente que cuando se ejecute una nueva terminal las variables de entorno de ROS sean agregadas automáticamente.

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

5.2.2.7. 7. Instalación de las dependencias para la construcción de paquetes

El siguiente paso consiste en instalar una herramienta que permita la instalación de paquetes con un solo comando.

```
$ sudo apt install python-rosinstall python-rosinstall-generator
python-wstool build-essential
```

5.2.2.8. 8. Examinando el entorno de ROS

El entorno ROS se configura a través de una serie de variables que le indican al sistema donde encontrar los paquetes. Para verificar si las variables de entorno de ROS se encuentran configuradas adecuadamente se hace uso del siguiente comando:

```
$ printenv | grep ROS
```

Si todo se encuentra funcionando correctamente, la salida del comando anterior debería ser similar a la siguiente:

```
declare -x ROSLISP_PACKAGE_DIRECTORIES=""
declare -x ROS_DISTRO="kinetic"
declare -x ROS_ETC_DIR="/opt/ros/kinetic/etc/ros"
declare -x ROS_MASTER_URI="http://localhost:11311"
declare -x ROS_PACKAGE_PATH="/opt/ros/kinetic/share"
declare -x ROS_ROOT="/opt/ros/kinetic/share/ros"
declare -x ROS_VERSION="1"
```

5.2.2.9. 9. Creación de un espacio de trabajo

Un espacio de trabajo es básicamente una carpeta en donde se almacenarán y modificarán los paquetes de ROS. Para la creación de dicho espacio de trabajo es suficiente con ingresar los siguientes comandos.

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

A continuación se actualiza el archivo de configuración.

```
$ source devel/setup.bash
```

Lo anterior se puede realizar de forma automática añadiendo dicha instrucción en el archivo de configuración.

```
$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

Para verificar si el espacio de trabajo se encuentra propiamente configurado se debe ingresar el siguiente comando:

```
$ echo $ROS_PACKAGE_PATH
```

Si todo se encuentra configurado de manera correcta se debe obtener la siguiente respuesta en la terminal.

```
/home/usuario/catkin_ws/src:/opt/ros/kinetic/share
```

5.2.2.10. 10. Iniciando ROS

Para comprobar la correcta instalación de ROS Kinetic Kame se procede a abrir una terminal en Ubuntu para la Inicialización de ROS.

```
$ roscore
```

A continuación con la combinación de teclas `ctrl+alt+t` se abre una nueva terminal y se ejecuta el siguiente comando:

```
$ rosrun turtlesim turtlesim_node
```

Si la instalación de ROS Kinetic Kame se efectuó de manera correcta se abrirá una ventana como el de la Figura 5.10.

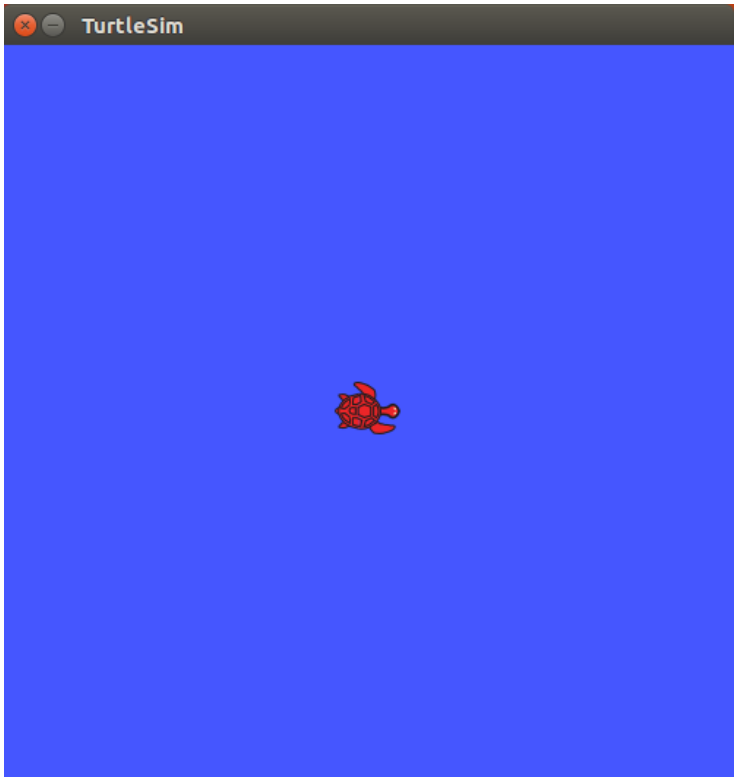


Figura 5.10: Ejecución de ros.

Ahora que ya se cuenta con ROS, se recomienda comenzar con los tutoriales del sitio oficial[12], además de revisar referencias especializadas [53], [54], [55], [56].

5.2.3. Paquete bebop_autonomy

El dron Bebop 2 puede ser controlado a través de ROS por medio del driver `bebop_autonomy` el cual fue desarrollado por Mani Monajjemi de la universidad Simon Fraser. Este paquete se basa en el kit de desarrollo Parrot ARDrone SDK3. En su sitio oficial[13] se puede encontrar información detallada sobre las características este paquete. A continuación se presentan los pasos para la instalación de paquete `bebop_autonomy`, mismos que pueden encontrarse en la sección de instalación[14] del sitio antes mencionado. Los comandos que a continuación se presentan fueron probados en Ubuntu 16.04 con ROS Kinetic Kame.

5.2.3.1. Instalación

Como primer nos ubicamos dentro de la carpeta `/src` de nuestro espacio de trabajo.

```
$ cd catkin_ws/src
```

A continuación se lleva a cabo la descarga del paquete desde un repositorio de *github*.

[12]http://wiki.ros.org/ROS/Tutorials
[13]https://bebop-autonomy.readthedocs.io/en/latest/
[14]https://bebop-autonomy.readthedocs.io/en/latest/installation.html

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

Volvemos a nuestro espacio de trabajo y compilamos.

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

En la Figura 5.11 se muestra el paquete recién descargado el cual está integrado por un conjunto de subcarpertas en la que `/bebop_driver` cobra más relevancia.

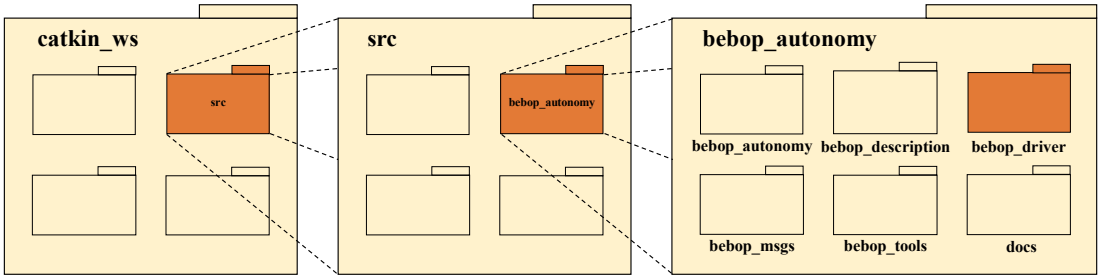


Figura 5.11: Paquete bebop_autonomy.

5.2.3.2. Ejecución del driver

Antes de trabajar con el Bebop 2 se recomienda colocar las hélices de manera correcta[15], verificar que la batería esté cargada, además de contar con un área despejada en donde se pueda llevar a cabo el vuelo del dron. Una vez considerado lo anterior, comenzamos con establecer la comunicación entre la PC remota y el Bebop 2, para ello encendemos el Bebop 2 presionando el botón ubicado en su parte trasera. Después de un par de minutos aparecerá una red Wifi la cual tendrá como nombre `Bebop2-xxxxxx`, los dígitos al final indican el número de identificación del dron.

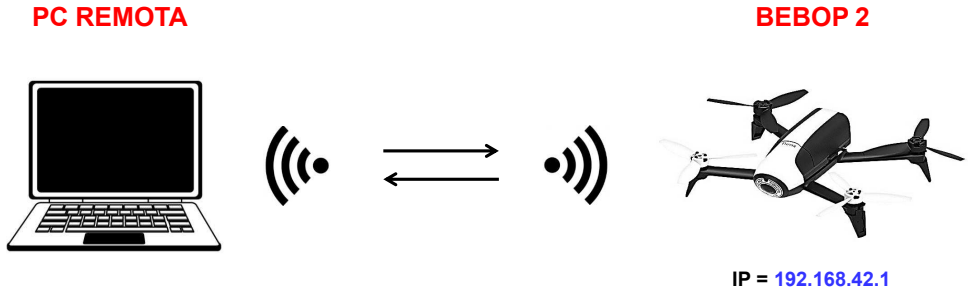


Figura 5.12: Red inalámbrica.

A continuación, nos conectamos a dicha red. Para verificar que la conexión se haya establecido de manera exitosa, abrimos la terminal e ingresamos el siguiente comando:

```
$ ping 198.168.42.1
```

[15] https://www.youtube.com/watch?v=vk4M8ZS4Ido

Lo cual devolverá un mensaje de manera repetida sobre el tiempo de envío y recepción de datos. Una vez verificado lo anterior procedemos a abrir una nueva terminal con la combinación de teclas `ctrl+alt+t` dentro de la cual ejecutamos el siguiente comando:

```
$ roslaunch bebop_driver bebop_node.launch
```

Después de ejecutar el comando anterior, el dron comenzará a publicar información sobre el estado de sensores tales como: cámara, unidad de medición inercial IMU, sensor de altitud y nivel de voltaje. De la misma forma, el dron estará preparado para recibir comandos que controlen su movimiento.

5.2.3.3. Envío de comandos al Bebop

Una forma rápida de verificar el correcto funcionamiento del dron mediante ROS es a través de la ejecución de un simple despegue y aterrizaje. Para efectuar el despegue publicamos un mensaje del tipo `std_msgs::Empty` al tópico `/bebop/takeoff`. Con el driver corriendo en una terminal abrimos una nueva y ejecutamos lo siguiente:

```
$ rostopic pub --once /bebop/takeoff std_msgs/Empty
```

Con lo anterior el dron se elevará aproximadamente a 1 metro del suelo y se mantendrá en vuelo estacionario de manera indefinida. Para el aterrizaje publicamos un mensaje del tipo `std_msgs::Empty` al topico `/bebop/land` con lo cual el vehículo volverá a posarse sobre suelo.

```
$ rostopic pub --once /bebop/land std_msgs/Empty
```

Para desplazar al Bebop es necesario publicar mensajes del tipo `geometry_msgs::Twist` al tópico `/bebop/cmd_vel`. Estos mensajes deben ser publicados mientras el vehículo se encuentre en vuelo estacionario. El efecto de como este mensaje influye en el movimiento del dron se describe en la Tabla 5.2:

Componente	**Descripción**
`linear.x` = k_x	Desplazamiento hacia adelante
`linear.x` = -k_x	Desplazamiento hacia atrás
`linear.y` = k_y	Desplazamiento a la izquierda
`linear.y` = -k_y	Desplazamiento a la derecha
`linear.z` = k_z	Ascenso
`linear.z` = -k_z	Descenso
`angular.z` = k_ψ	Rotación CCW
`angular.z` = -k_ψ	Rotación CW

Tabla 5.2: Componentes del mensaje `geometry_msgs::Twist` para el pilotaje del Bebop 2.

Donde $k_x, k_y, k_z, k_\psi \in [0 \dots 1]$. El dron ejecutará el último comando recibido mientras el driver se encuentra activo. Estos valores se vuelven cero cuando el dron recibe

un comando de despegue o aterrizaje. Para mantener al vehículo en vuelo estacionario en su posición actual es necesario asignar ceros a todas las componentes del mensaje `geometry_msgs::Twist`.

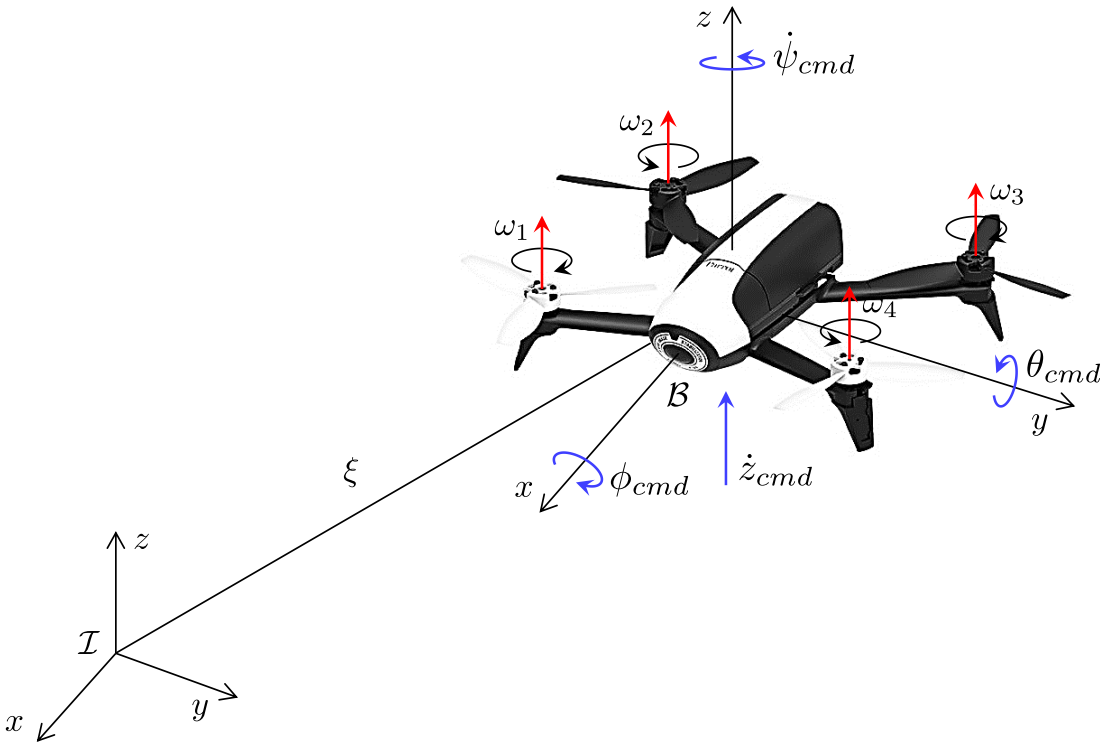


Figura 5.13: Diagrama de cuerpo libre del dron Bebop 2.

Las componentes `linear.x` y `linear.y` del mensaje `geometry_msgs::Twist` controlan los ángulos *pitch* y *roll* del dron y eventualmente los desplazamientos en el plano x-y. Estos ángulos dependen directamente del parámetro `PilotingSettingsMaxTiltCurrent` (α_{max}) el cual es especificado en grados y puede ser modificado, el valor por defecto es 20°. De este modo:

$$\theta_{cmd} = k_x \alpha_{max} \tag{5.1}$$

$$\phi_{cmd} = k_y \alpha_{max} \tag{5.2}$$

La componente `linear.z` controla la velocidad vertical del dron. Este valor depende del parámetro `SpeedSettingsMaxVerticalSpeedCurrent` (β_{max}) el cual también puede ser modificado, el valor por defecto es 1 m/seg. Por lo tanto:

$$\dot{z}_{cmd} = k_z \beta_{max} \tag{5.3}$$

La componente `angular.z` controla la velocidad angular del vehículo (alrededor del eje z), dicho valor depende del parámetro `SpeedSettingsMaxRotationSpeedCurrent` (γ_{max}), el cual también puede ser modificado cuyo valor por defecto es $100°/seg$.

$$\dot{\psi}_{cmd} = k_\psi \gamma_{max} \tag{5.4}$$

5.2.3.4. Lectura de datos del Bebop

El driver transmite la lectura de los sensores a través de tópicos. El vídeo de la cámara frontal del dron es publicado en el tópico `/bebop/image_raw` mediante un mensaje del tipo `sensor_msgs::Image` tal como se muestra en la Figura 5.14.

Figura 5.14: Cámara frontal del Bebop 2.

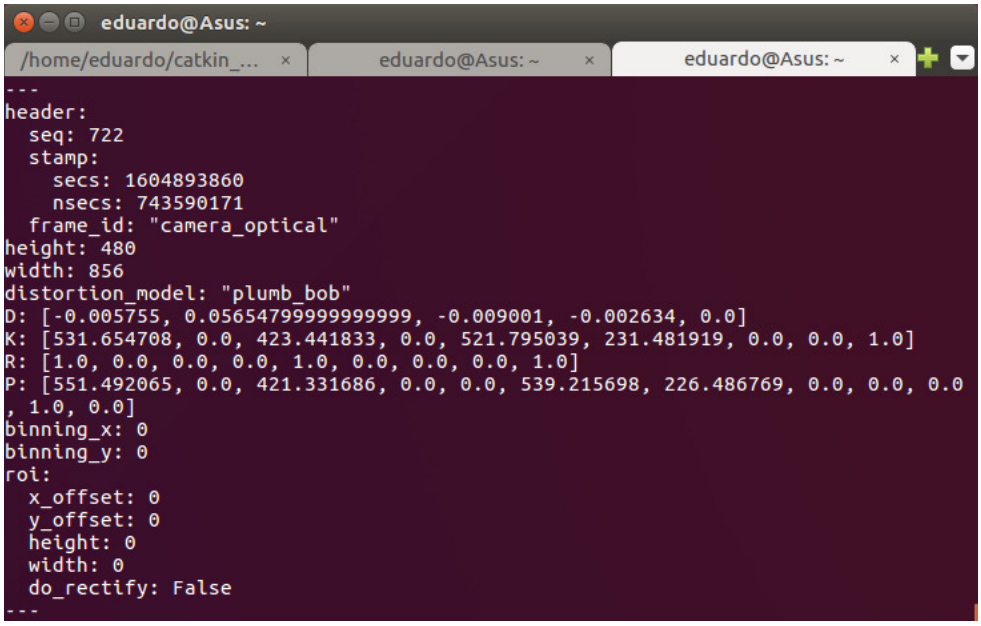


Figura 5.15: Datos de calibración de la cámara.

La información de calibración se publica en el tópico `/bebop/camera_info` (Figura

5.15), esta información se obtiene tras el proceso de calibración de la cámara, el archivo generado debe ubicarse en: `/bebop_driver/data/bebop_camera_calib.yaml` el cual se carga cada vez que se ejecuta el driver.

Por medio de un mensaje `bebop_msgs::Ardrone3PilotingStateAltitudeChanged` el driver publica información de la altitud del vehículo (en metros) con respecto al punto de despegue. En el tópico `/bebop/states/ardrone3/PilotingState/AltitudeChanged` se puede acceder a dicha información (Figura 5.16).

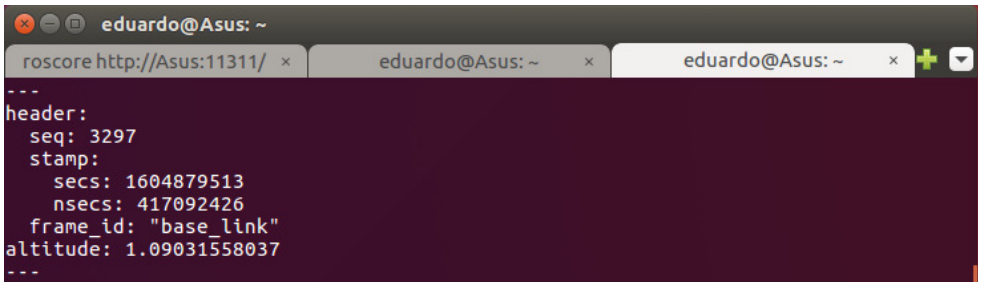


Figura 5.16: Altitud del vehículo.

En el tópico `/bebop/states/ardrone3/PilotingState/AttitudeChanged` se publican mensajes del tipo `bebop_msgs::Ardrone3PilotingStateAttitudeChanged` a una velocidad de 5 Hz los cuales incluyen datos sobre la orientación del dron (Figura 5.17), mismos que están dados en radianes.

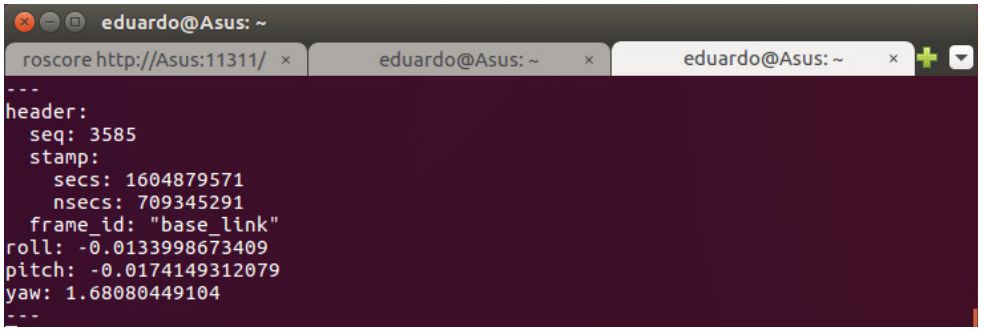


Figura 5.17: Datos de la orientación del vehículo.

El driver incorpora un sistema visual-inercial el cual efectúa la estimación de la odometría del vehículo la cual se publica en el tópico `/bebop/odom` a través de un mensaje del tipo `nav_msgs::Odometry` (Figura 5.18), a una velocidad de transmisión de 5 Hz. Este mensaje contiene la posición y orientación, así como la velocidad del Bebop la cual está representada en el sistema de referencia ESD. Esta información puede ser de gran utilidad en el diseño de controladores.

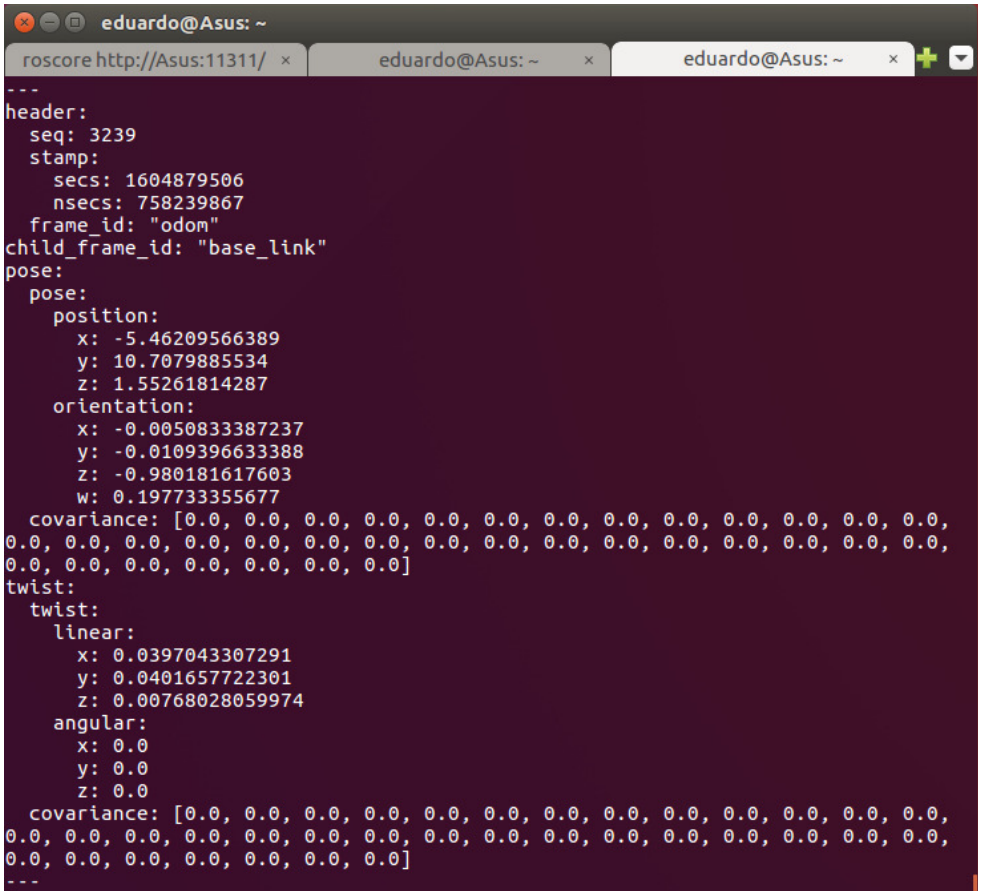


Figura 5.18: Datos de odometría del Bebop 2.

De igual forma se puede acceder a la información del GPS por medio del tópico `/bebop/fix` el cual publica información de la latitud, longitud y altitud del vehículo en un mensaje del tipo `sensor_msgs::NavSatFix` tal como se muestra en la Figura 5.19.

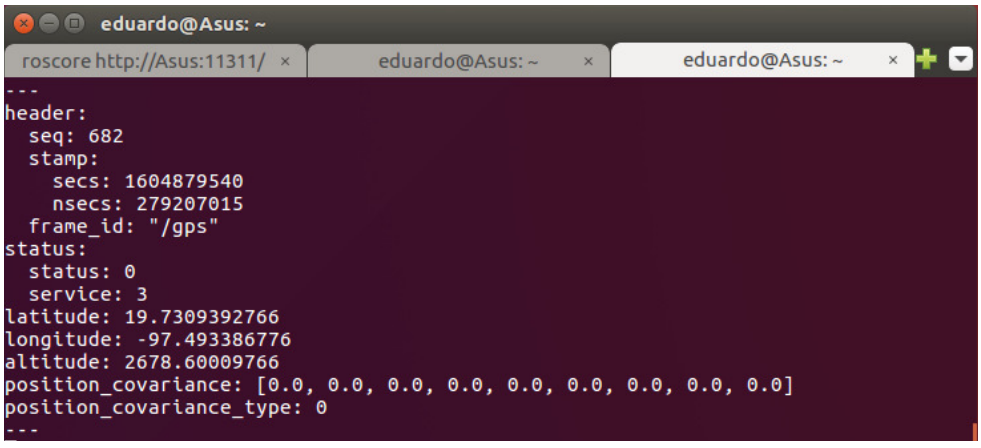


Figura 5.19: Datos del GPS del dron.

Por último, el tópico `/bebop/states/common/CommonState/BatteryStateChanged` publica información sobre el porcentaje de carga de la batería (5.20), el cual emplea un

mensaje del tipo `bebop_msgs::CommonCommonStateBatteryStateChanged`. Esta información es importante y se debe verificar antes de efectuar cada vuelo.

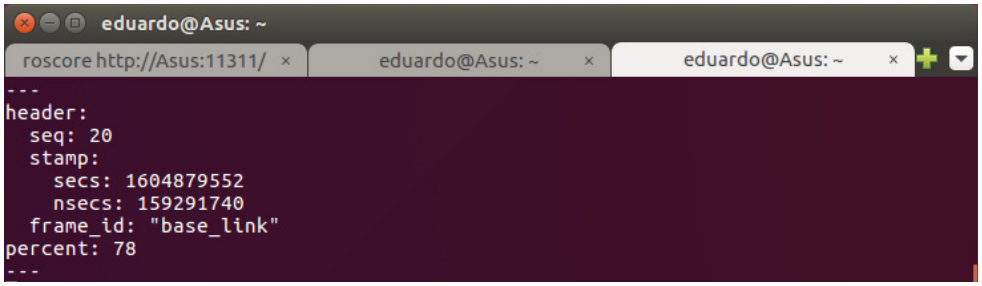


Figura 5.20: Datos sobre el estado de la batería del Bebop 2.

Para facilitar el uso del Bebop a través de ROS se ha creado un repositorio[16] el cual incluye distintos ejemplos para el pilotaje del dron, así como para efectuar la lectura de los datos proporcionados por el driver.

[16]https://github.com/EDU4RDO-SH/bebop_ros_examples.git

6 Desarrollo

6.1. Estimación de estados por medio de odometría visual

El sistema de odometría visual fue diseñado de acuerdo a los pasos descritos en el Capítulo 4, las pruebas experimentales se llevaron a cabo utilizando la secuencia de imágenes `2011_10_03_drive_0027`[1] del *dataset* KITTI [57], el cual es un conjunto de imágenes muy conocido en la comunidad científica para realizar pruebas de algoritmos de visión por computadora. Para usar este *dataset* dentro de ROS fue necesario convertir las imágenes a un archivo `.bag` de ROS. `kitti2bag` es un paquete [2] que nos permite efectuar esta tarea. El archivo `.bag` generado incluye la secuencia de imágenes a color y en escala de grises del *dataset* original, las transformaciones entre los diferentes sistemas de referencia del vehículo que capturó los datos, información de la IMU, así como información del LIDAR. Los datos anteriores son publicados a través de tópicos.

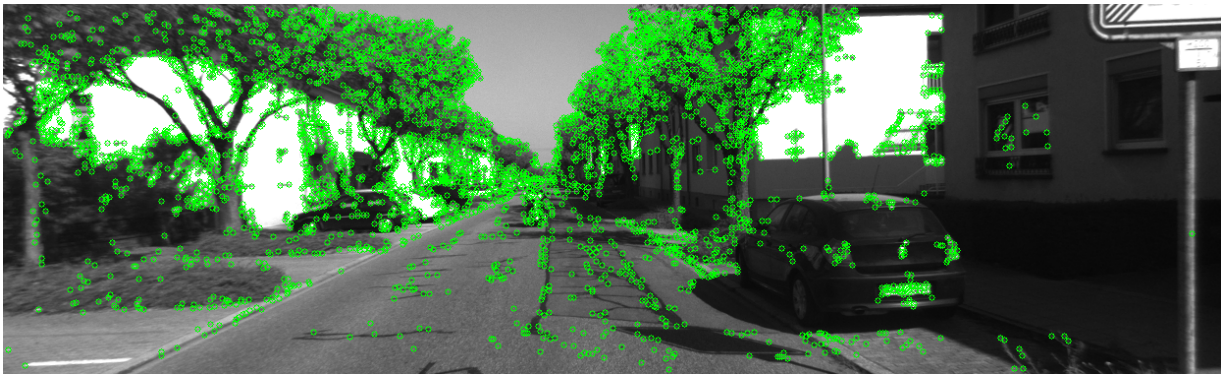


Figura 6.1: Características detectadas en la primera imagen.

Una vez realizada la configuración inicial, se procede a realizar la implementación del algoritmo de odometría visual con la ayuda de las librerías de OpenCV[3], el cual actual-

[1] http://www.cvlibs.net/datasets/kitti/raw_data.php?type=residential
[2] https://github.com/tomas789/kitti2bag
[3] https://opencv.org/

mente es considerado como el software más importante para el desarrollo de algoritmos de visión por computadora. Como primer paso, se procede a ejecutar el archivo `.bag` generado previamente y nos subscribimos al tópico que publica la secuencia de imágenes. A continuación, detectamos los puntos de interés con la función `cv::FAST()` en la primera imagen tal como se muestra en la Figura 6.1. A continuación, las posiciones de dichas características son estimadas en la segunda imagen con ayuda de la función `cv::calcOpticalFlowPyrLK()` como se ilustra en la Figura 6.2.

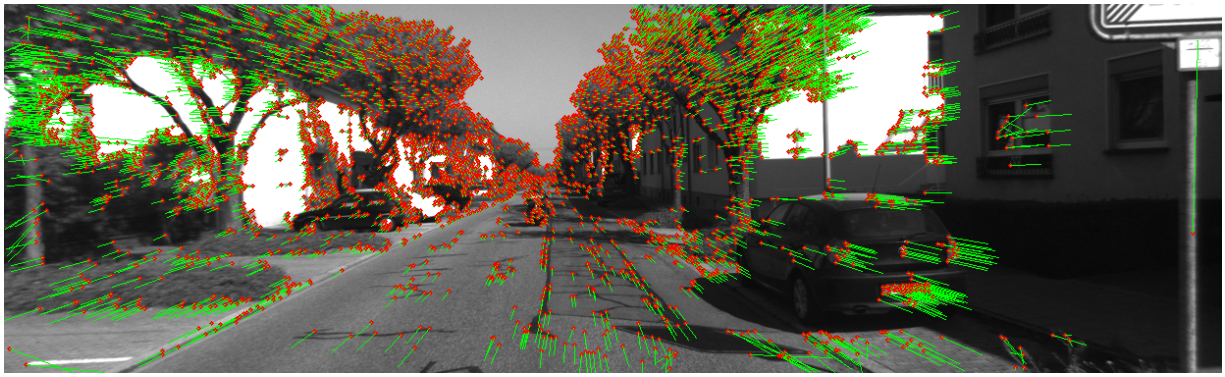


Figura 6.2: Flujo óptico de las dos primeras imágenes del *dataset*.

Una vez calculadas las posiciones de los puntos de interés de la segunda imagen es posible calcular la matriz esencial con la función `cv::findEssentialMat()`. Esta función requiere principalmente de los vectores de las posiciones de los puntos de interés en la primera y segunda imagen y de la matriz de calibración (6.1).

$$K = \begin{bmatrix} 718.8560 & 0 & 607.1928 \\ 0 & 718.8560 & 185.2157 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

Las componentes de la posición $t \in \mathbb{R}^3$ y orientación $R \in \mathbb{R}^{3\times3}$ se pueden extraer directamente de la matriz esencial tal como lo indica la ecuación (4.26). La función `cv::recoverPose()` permite recuperar la posición y orientación eligiendo la solución más apropiada de (4.26). Esta función requiere principalmente de la matriz esencial previamente calculada y de la matriz de calibración (6.1). La trayectoria completa $t_f \in \mathbb{R}^3$ y $R_f \in \mathbb{R}^{3\times3}$ se obtiene tras concatenar las poses consecutivas empleando las expresiones (6.2) cada vez que una nueva imagen es procesada.

$$\begin{aligned} t_f &= t_f + R_f t \\ R_f &= R R_f \end{aligned} \tag{6.2}$$

Parte del resultado se puede apreciar en la Figura 6.3, o bien en el siguiente vídeo[4].

[4]https://youtu.be/A9sfLDSboQg

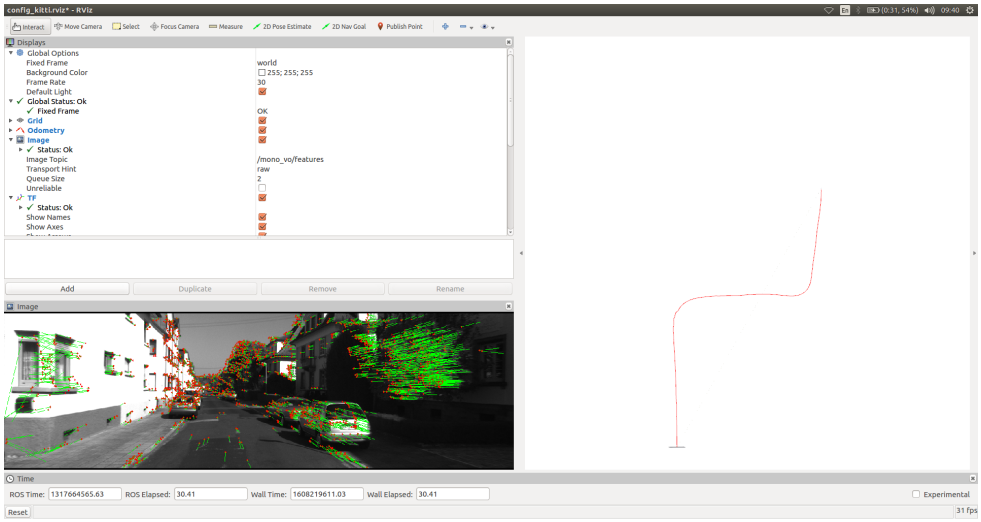


Figura 6.3: Trayectoria generada tras concatenar poses consecutivas.

A pesar de haber obtenido buenos resultados con el algoritmo anterior está claro que hay varias mejoras que se deben realizar, como la teoría lo establece la trayectoria obtenida mediante un sistema de odometría visual monocular solo es válida a un factor de escala, por lo que es necesario conocer el tamaño real de un objeto en la escena o la altura de la cámara con respecto al suelo para recuperar la trayectoria real. También es posible fusionar los datos de odometría con otros sensores para obtener mejores resultados. Esta es un área activa de investigación y cada vez se proponen métodos más sofisticados para resolver el problema de odometría visual monocular de una manera eficiente. Debido a lo anterior la estrategia de control MPC será evaluada utilizando los datos de odometría proporcionados por el driver del Bebop, como se mencionó en el Capítulo 5 los datos se obtienen del tópico `/bebop/odom`, estos datos son publicados por el driver el cual por medio de un sistema de odometría visual-inercial efectúa la estimación de los estados del robot. En la Figura 6.4 se muestra la odometría del dron dentro del entorno de RViz.

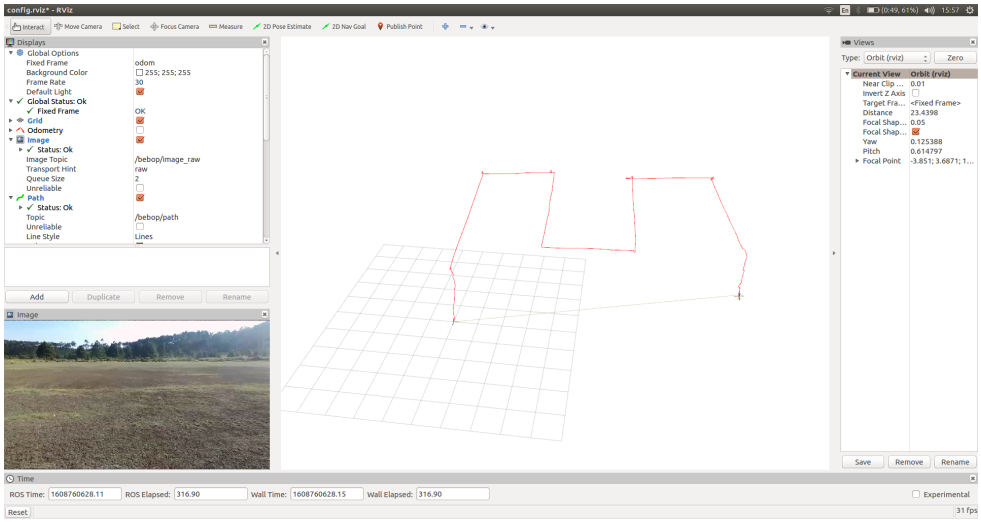


Figura 6.4: Odometría del Bebop en RViz.

En la Figura 6.5 se muestran las componentes de la posición estimadas por la odometría del dron.

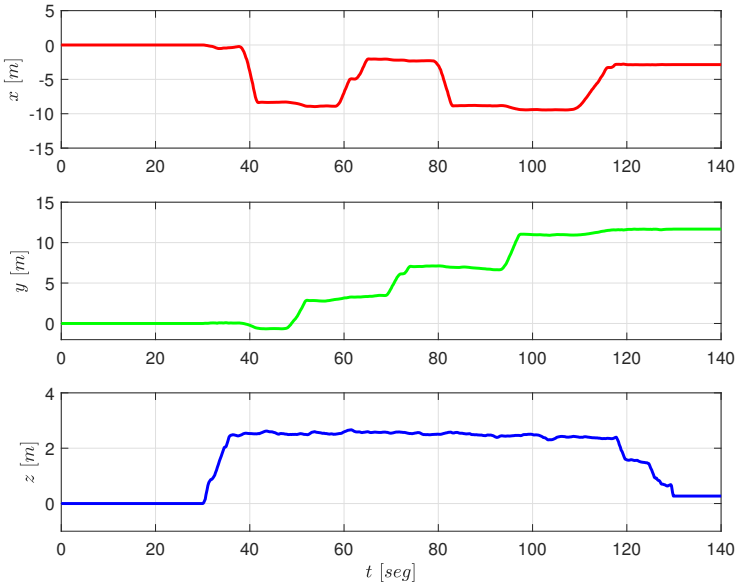


Figura 6.5: Posiciones estimadas por el Bebop.

En la Figura 6.6 se muestran las mismas componentes graficadas en el espacio tridimensional.

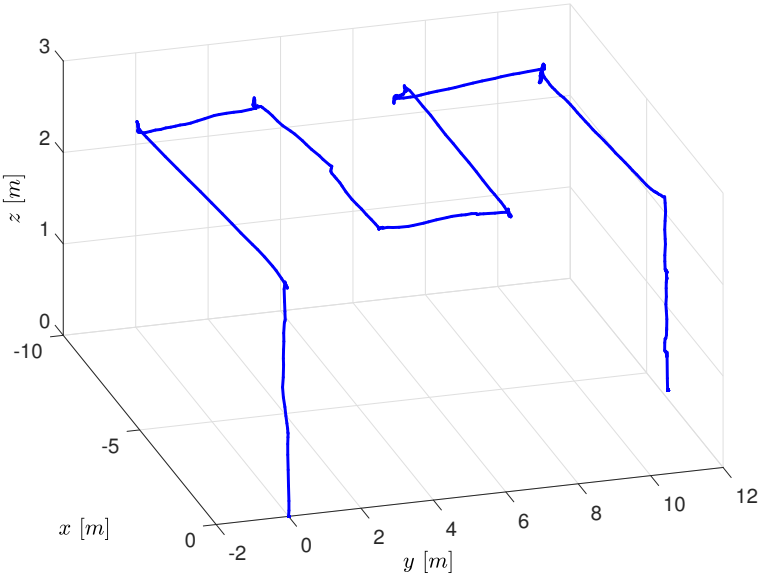


Figura 6.6: Posiciones estimadas en el espacio 3D.

De forma similar en la Figura 6.7 se muestran las componentes de la orientación.

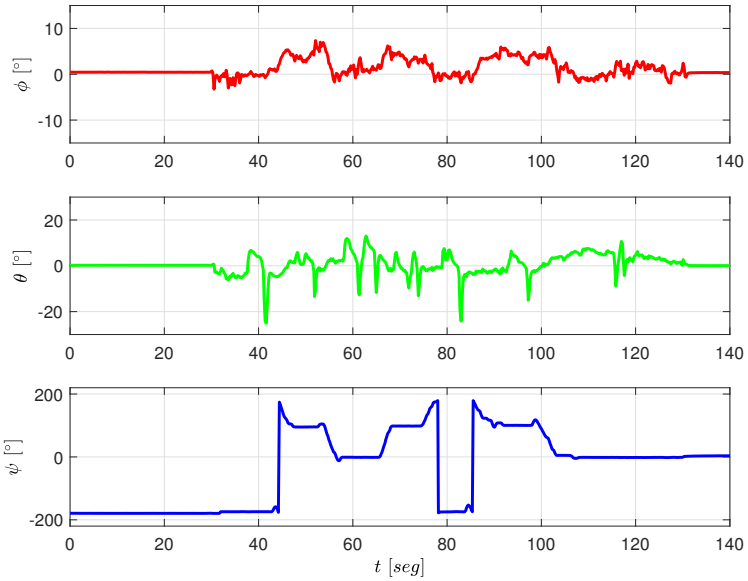


Figura 6.7: Componentes de la orientación estimadas.

Finalmente, en la Figura 6.8 se presentan las componentes de la velocidad lineal.

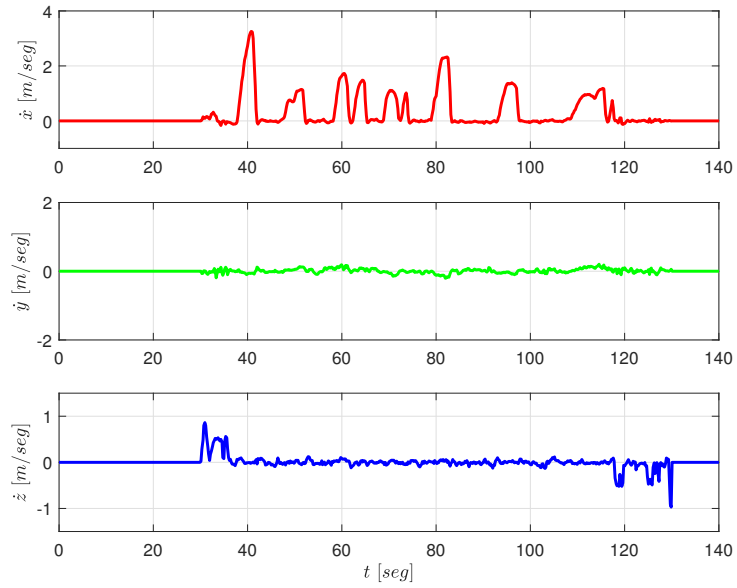


Figura 6.8: Velocidades lineales estimadas.

Distintas pruebas realizadas para la evaluación de la odometría del Bebop se pueden apreciar de mejor forma en el siguiente enlace[5].

[5]https://youtu.be/jZMR8T6uhwY

6.2. Implementación del control MPC para el seguimiento de trayectorias

6.2.1. Modelo dinámico

La pose de un vehículo aéreo se puede definir estableciendo un sistema de referencia inercial $\mathcal{I}$ y un sistema de referencia en el centro de gravedad del robot $\mathcal{B}$. La posición del sistema de referencia del vehículo con respecto al plano inercial se denota por $\boldsymbol{\xi} = \begin{bmatrix} x & y & z \end{bmatrix}^T$. De manera similar, la orientación del vehículo se representa mediante $\boldsymbol{\eta} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$. Donde ϕ, θ y ψ son conocidos como los ángulos *roll*, *pitch* y *yaw* respectivamente.

6.2.1.1. Modelo dinámico lineal

El modelo lineal se obtiene al considerar que el vehículo se encuentra en vuelo estacionario, además de tomar en cuenta la aproximación de ángulos pequeños y que el ángulo $\psi = 0$ se encuentra alineado con el eje x del sistema de coordenadas inercial $\mathcal{I}$ [58]. De esta manera:

$$\begin{aligned}
\dot{x} &= \dot{x} \\
\dot{y} &= \dot{y} \\
\dot{z} &= \dot{z} \\
\ddot{x} &= g\,{}^{\mathcal{I}}\theta - A_x\dot{x} + d_x \\
\ddot{y} &= -g\,{}^{\mathcal{I}}\phi - A_y\dot{y} + d_y \\
\ddot{z} &= \frac{1}{\tau_{\dot{z}}}(k_{\dot{z}}\dot{z}_{cmd} - \dot{z}) - A_z\dot{z} + d_z \\
\dot{\phi} &= \frac{1}{\tau_\phi}(k_\phi\,{}^{\mathcal{I}}\phi_{cmd} - {}^{\mathcal{I}}\phi) \\
\dot{\theta} &= \frac{1}{\tau_\theta}(k_\theta\,{}^{\mathcal{I}}\theta_{cmd} - {}^{\mathcal{I}}\theta)
\end{aligned} \tag{6.3}$$

Como se puede notar, en el modelo dinámico se han considerado tanto el efecto de arrastre así como posibles perturbaciones. En su representación matricial se tiene:

$$\dot{\boldsymbol{x}}(t) = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -A_x & 0 & 0 & 0 & g \\ 0 & 0 & 0 & 0 & -A_y & 0 & -g & 0 \\ 0 & 0 & 0 & 0 & 0 & -A_z - \frac{1}{\tau_{\dot{z}}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\phi} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\theta} \end{bmatrix}}_{A_c} \boldsymbol{x}(t) + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{K_{\dot{z}}}{\tau_{\dot{z}}} \\ \frac{K_\phi}{\tau_\phi} & 0 & 0 \\ 0 & \frac{K_\theta}{\tau_\theta} & 0 \end{bmatrix}}_{B_c} \boldsymbol{u}(t) + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{B_{d,c}} \boldsymbol{d}(t) \tag{6.4}$$

Donde $\boldsymbol{x} = \begin{bmatrix} x, y, z, \dot{x}, & \dot{y}, & \dot{z}, & {}^{\mathcal{I}}\phi, & {}^{\mathcal{I}}\theta \end{bmatrix}^T$ es el vector de estados, $\boldsymbol{u} = \begin{bmatrix} {}^{\mathcal{I}}\phi_{cmd}, & {}^{\mathcal{I}}\theta_{cmd}, & \dot{z}_{cmd} \end{bmatrix}^T$ el vector de entradas y $\boldsymbol{d} = \begin{bmatrix} d_x, & d_y, & d_z \end{bmatrix}^T$ el vector de perturbaciones externas. En este

modelo la orientación se ha representado dentro del sistema de coordenadas inercial $\mathcal{I}$ con el objetivo de eliminar al ángulo ψ del modelo. De este modo, las señales ${}^{\mathcal{I}}\phi_{cmd}$ y ${}^{\mathcal{I}}\theta_{cmd}$ son calculadas dentro del plano inercial $\mathcal{I}$ y a continuación convertidas al sistema de coordenadas del vehículo $\mathcal{B}$ realizando una rotación al rededor del eje z.

$$\begin{bmatrix}\phi_{cmd}\\ \theta_{cmd}\end{bmatrix} = \begin{bmatrix}\cos\psi & \operatorname{sen}\psi\\ -\operatorname{sen}\psi & \cos\psi\end{bmatrix}\begin{bmatrix}{}^{\mathcal{I}}\phi_{cmd}\\ {}^{\mathcal{I}}\theta_{cmd}\end{bmatrix} \tag{6.5}$$

6.2.1.2. Sistema de control de bajo nivel

Para lograr que el vehículo siga una trayectoria de referencia de manera precisa es necesario tomar en consideración la dinámica interna del dron, la cual es desconocida cuando se trata de drones comerciales. Por lo que es necesario efectuar la identificación del sistema en lazo cerrado del controlador para cada una de las entradas de control, las cuales se puede aproximar por medio de un sistema de primer orden:

$$\dot{\phi} = \frac{1}{\tau_\phi}(k_\phi\phi_{cmd} - \phi) \tag{6.6}$$

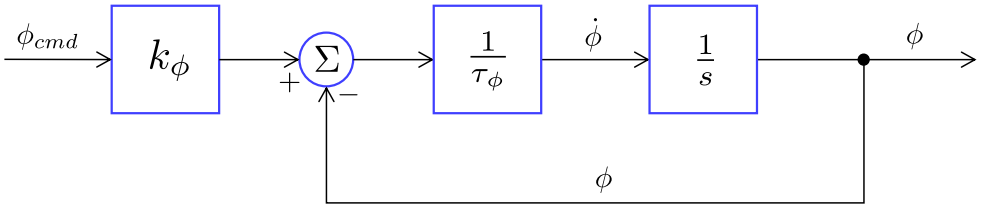


Figura 6.9: Lazo de control del ángulo ϕ.

$$\dot{\theta} = \frac{1}{\tau_\theta}(k_\theta\theta_{cmd} - \theta) \tag{6.7}$$

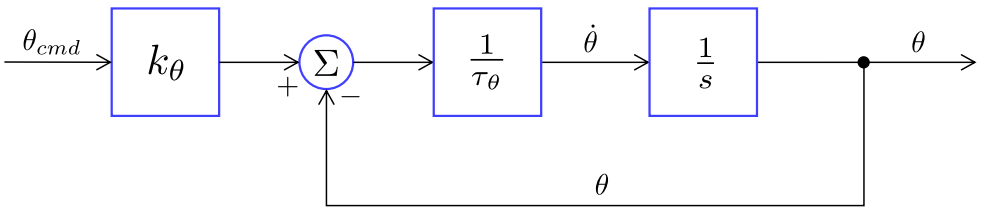


Figura 6.10: Lazo de control del ángulo θ.

$$\ddot{z} = \frac{1}{\tau_{\dot{z}}}(k_{\dot{z}}\dot{z}_{cmd} - \dot{z}) \tag{6.8}$$

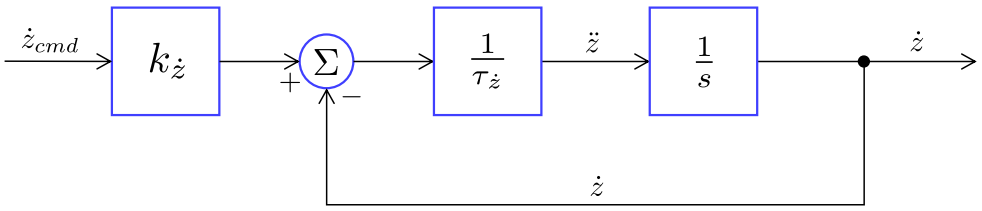


Figura 6.11: Lazo de control para el empuje vertical $\dot{z}$.

$$\dot{\psi} = \dot{\psi}_{cmd} \tag{6.9}$$

Donde k_ϕ, k_θ y $k_{\dot{z}}$ son las ganancias de los sistemas de primer orden; τ_ϕ, τ_θ y $\tau_{\dot{z}}$ las constantes de tiempo, los parámetros anteriores se obtienen tras el proceso de identificación. ϕ_{cmd}, θ_{cmd} y $\dot{z}_{cmd}$ son las entradas de control comandadas. $\dot{\psi}_{cmd}$ es la velocidad angular comandada alrededor del eje z. En este caso se considera que la velocidad angular $\dot{\psi}$ alcanza la referencia de manera instantánea, lo anterior es válido debido a que el movimiento en este eje no afecta en la traslación del vehículo. La identificación de los parámetros anteriores se lleva a cabo siguiendo el procedimiento descrito en[6].

6.2.2. Discretización

Debido a que el controlador a implementar se trata de un control MPC, es necesario efectuar la discretización del modelo dinámico (6.4) con la ayuda de las siguientes expresiones [59]:

$$\begin{aligned} A &= e^{A_c T_s} \\ B &= \int_0^{T_s} e^{A_c s} ds B_c \\ B_d &= \int_0^{T_s} e^{A_c s} ds B_{d,c} \end{aligned} \tag{6.10}$$

Donde T_s es el periodo de muestreo. El cálculo del costo terminal P se obtiene tras resolver la ecuación algebraica de Riccati de manera iterativa.

6.2.3. Control MPC

La estructura de control empleada consiste en un controlador de bajo nivel en conjunto con un controlador MPC externo el cual se encarga de controlar la posición del robot.

[6] https://github.com/ethz-asl/dji_onboard_sdk_ros/wiki/Dynamic-System-Identification

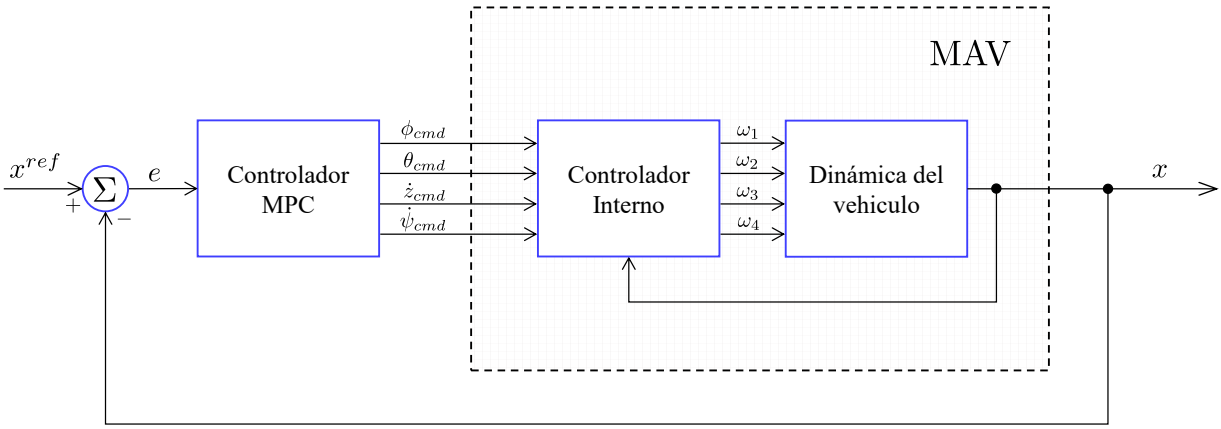


Figura 6.12: Lazo de control.

El problema de optimización para el control MPC se formula de la siguiente manera:

$$\begin{aligned}\min_{\boldsymbol{U},\boldsymbol{X}} \quad & \sum_{k=0}^{N-1}(\boldsymbol{x}_k-\boldsymbol{x}_k^{ref})^T Q_x(\boldsymbol{x}_k-\boldsymbol{x}_k^{ref})+(\boldsymbol{u}_k-\boldsymbol{u}_k^{ref})^T R_u(\boldsymbol{u}_k-\boldsymbol{u}_k^{ref}) \\ & +(\boldsymbol{u}_k-\boldsymbol{u}_{k-1})^T R_\Delta(\boldsymbol{u}_k-\boldsymbol{u}_{k-1})+(\boldsymbol{x}_N-\boldsymbol{x}_N^{ref})^T P(\boldsymbol{x}_N-\boldsymbol{x}_N^{ref}) \\ \text{s.a.} \quad & \boldsymbol{x}_{k+1}=A\boldsymbol{x}_k+B\boldsymbol{u}_k+B_d\boldsymbol{d}_k; \\ & \boldsymbol{d}_{k+1}=\boldsymbol{d}_k, \quad k=0,\ldots,N-1 \\ & \boldsymbol{u}_k\in\mathbb{U} \\ & \boldsymbol{x}_0=\boldsymbol{x}(t_0), \quad \boldsymbol{d}_0=\boldsymbol{d}(t_0)\end{aligned} \tag{6.11}$$

Donde $Q_x \succeq 0$ es el peso dado a la minimización del error de estados en el proceso de optimización.

$$Q_x=\begin{bmatrix} q_x^x & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & q_y^x & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_z^x & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_{\dot{x}}^x & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & q_{\dot{y}}^x & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & q_{\dot{z}}^x & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & q_\phi^x & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & q_\theta^x \end{bmatrix} \tag{6.12}$$

$R_u \succ 0$ indica el peso dado a la minimización de la secuencia de control, de forma similar $R_\Delta \succeq 0$ representa el peso dado a la taza de cambio de la secuencia de control.

$$R_u=\begin{bmatrix} r^u_{\phi_{cmd}} & 0 & 0 \\ 0 & r^u_{\theta_{cmd}} & 0 \\ 0 & 0 & r^u_{\dot{z}_{cmd}} \end{bmatrix} \qquad R_\Delta=\begin{bmatrix} r^\Delta_{\phi_{cmd}} & 0 & 0 \\ 0 & r^\Delta_{\theta_{cmd}} & 0 \\ 0 & 0 & r^\Delta_{\dot{z}_{cmd}} \end{bmatrix} \tag{6.13}$$

Por último, $P \in \mathbb{R}^{8\times 8}$ representa el peso dado al error de estado terminal. El problema de optimización fue resuelto con la ayuda de CVXGEN[7], el cual permite describir el

[7]https://cvxgen.com/docs/index.html

problema de optimización a través de un lenguaje de alto nivel dando como resultado un conjunto de librerías en C especificas para el problema en cuestión [60]. Con esta herramienta resulta relativamente sencillo formular problemas para controladores MPC[8].

6.2.4. Simulación del controlador en el vehículo AR.drone

Para evaluar el desempeño del controlador previamente descrito se procede a implementar dicho algoritmo en el vehículo AR.drone de la compañía francesa Parrot. Actualmente el modelo de este vehículo se encuentra disponible dentro del entorno de Gazebo, por lo que en conjunto con su driver[9] es posible implementar algoritmos mediante ROS como si se tratara del vehículo real. El modelo para la versión de ROS Kinetic se puede descargar directamente desde Github[10].

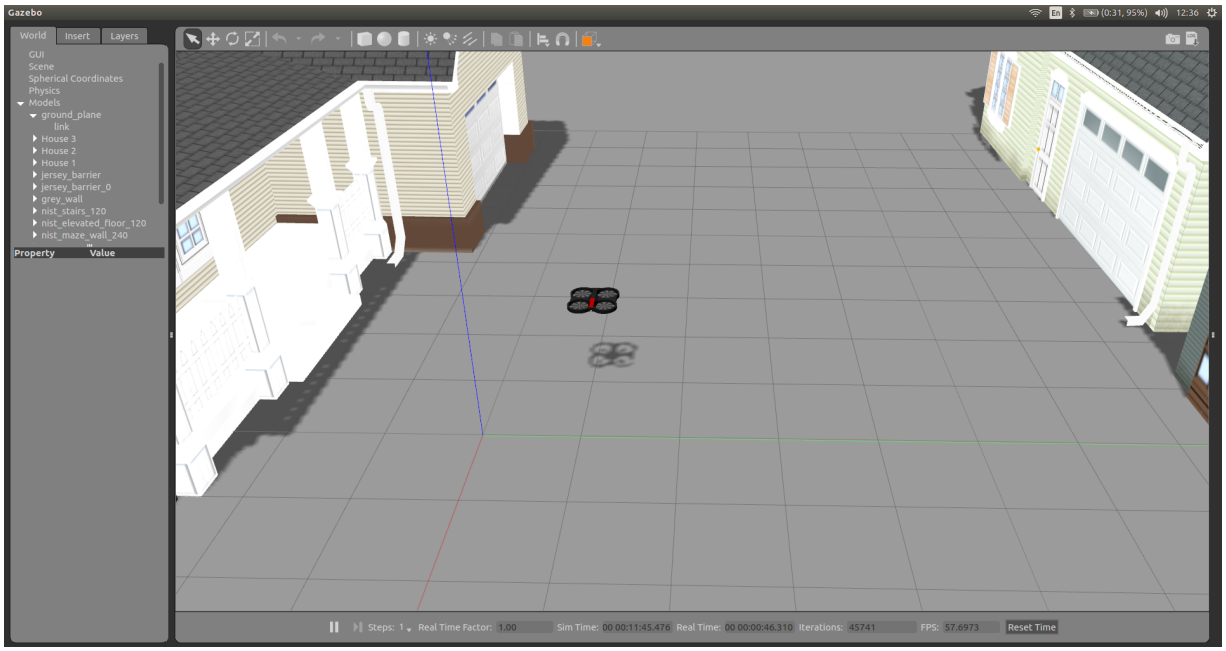


Figura 6.13: Vehículo AR.drone dentro del entorno de Gazebo.

Para hacer uso del modelo dinámico (6.4) es necesario conocer los parámetros para este vehículo en particular. De entre los más importantes aparecen las constantes de tiempo y ganancias de los lazos de control internos, por lo que resulta fundamental realizar la identificación de los sistemas de primer orden para cada entrada de control. El proceso de identificación se realizó de acuerdo al método descrito en [61] en donde la identificación de las funciones de transferencia se efectúa con la ayuda de la función `tfest()` de MATLAB, para ello es necesario colectar los datos de las entradas y las salidas en dos experimentos, el primero para estimar el sistema y el segundo para validar el modelo obtenido. Para el caso del empuje vertical se tienen los datos de la Figura 6.14.

[8] https://cvxgen.com/docs/mpc.html
[9] https://ardrone-autonomy.readthedocs.io/en/latest/
[10] https://github.com/angelsantamaria/tum_simulator

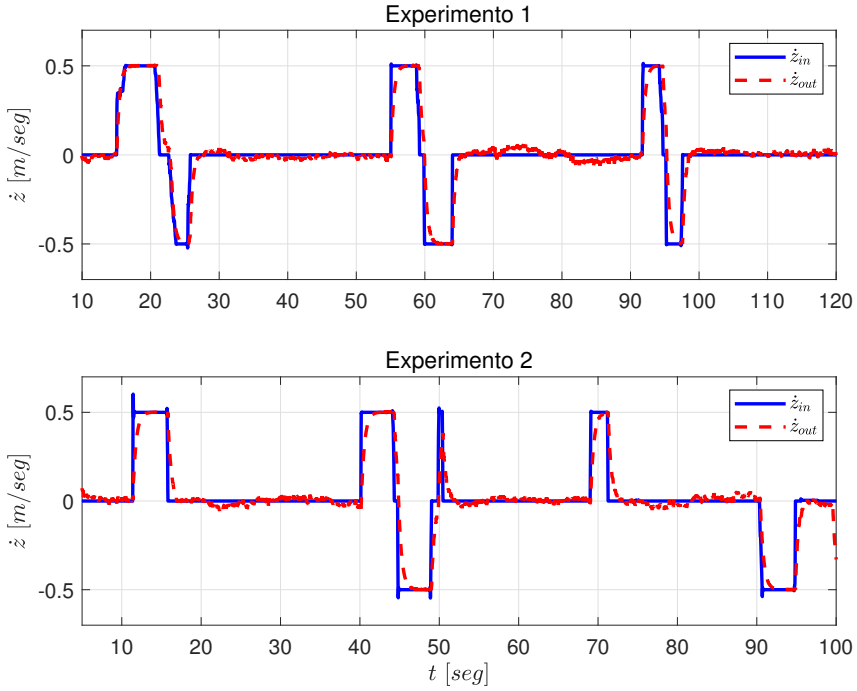


Figura 6.14: Datos de las entradas y salidas del empuje vertical $\dot{z}$.

La función de transferencia obtenida es la siguiente:

$$\frac{y_{\dot{z}}(s)}{u_{\dot{z}}(s)} = \frac{1.007}{0.415s + 1} \tag{6.14}$$

De este modo $\tau_{\dot{z}} = 0.415$ y $K_{\dot{z}} = 1.007$. La comparación entre la respuesta del sistema real con la respuesta del modelo obtenido aparece en la Figura 6.15. Como se puede apreciar el modelo estimado captura de manera precisa la dinámica del subsistema para el empuje vertical.

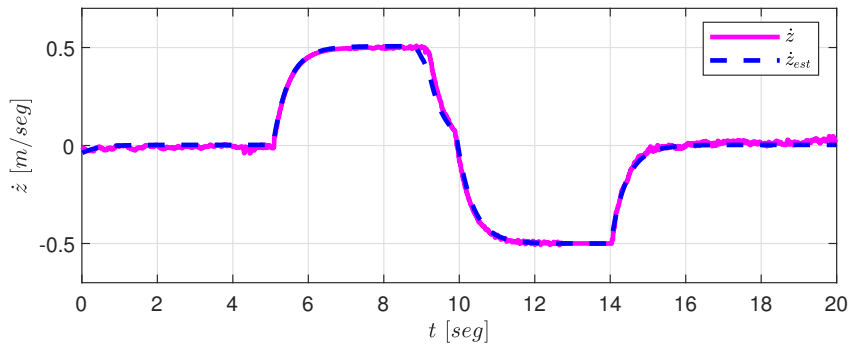


Figura 6.15: Respuesta del subsistema para el empuje vertical.

Para el caso de la velocidad angular se tienen los siguientes datos registrados en dos experimentos.

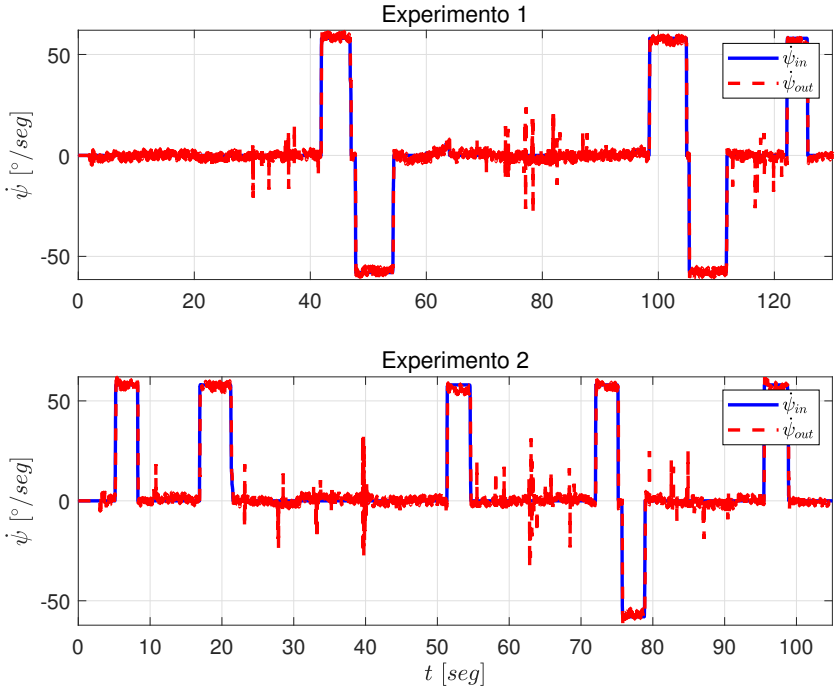


Figura 6.16: Datos de las entradas y salidas de la velocidad angular $\dot{\psi}$.

Estos datos nos permiten estimar la siguiente función de transferencia, la cual corresponde a un sistema de primer orden.

$$\frac{y_{\dot{\psi}}(s)}{u_{\dot{\psi}}(s)} = \frac{0.990}{0.040s + 1} \tag{6.15}$$

De esta manera se tiene que $\tau_{\dot{\psi}} = 0.040$ y $K_{\dot{\psi}} = 0.990$. La respuesta del modelo estimado con respecto a la respuesta real del sistema se puede apreciar en la Figura 6.17, la cual representa de manera precisa la respuesta de este subsistema.

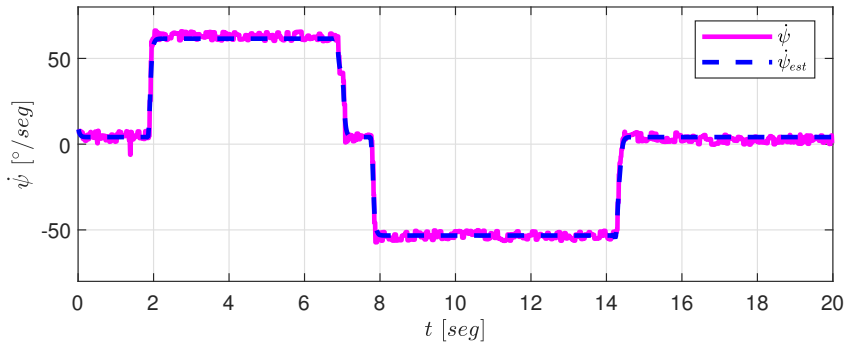


Figura 6.17: Respuesta del subsistema para la velocidad angular.

Los parámetros dinámicos para los subsistemas ϕ y θ son tomados de [61] debido a que el modelo de AR.drone en Gazebo entrega mediciones de los ángulos un tanto imprecisos

por lo cual no es posible llevar a cabo la identificación de estos subsistemas como se hizo con los anteriores.

$$\frac{y_\phi(s)}{u_\phi(s)} = \frac{1.673}{0.472s+1} \tag{6.16}$$

$$\frac{y_\theta(s)}{u_\theta(s)} = \frac{1.575}{0.472s+1} \tag{6.17}$$

El resumen de los parámetros dinámicos para los sistemas de primer orden se muestran en la Tabla 6.1.

Parámetro	ϕ	θ	$\dot{z}$	$\dot{\psi}$
τ	0.472	0.472	0.415	0.040
K	1.673	1.575	1.007	0.990

Tabla 6.1: Parámetros de la dinámica interna del AR.drone.

Considerando a los coeficientes de arrastre $A = \begin{bmatrix}0.01, & 0.01, & 0\end{bmatrix}^T$ y a $g = 9.81$ el modelo dinámico 6.4 para el AR.drone queda definido de la siguiente manera:

$$\dot{\boldsymbol{x}}(t) = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -0.01 & 0 & 0 & 0 & 9.81 \\ 0 & 0 & 0 & 0 & -0.01 & 0 & -9.81 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2.41 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2.11 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2.11 \end{bmatrix}}_{A_c} \boldsymbol{x}(t) + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2.42 \\ 3.54 & 0 & 0 \\ 0 & 3.33 & 0 \end{bmatrix}}_{B_c} \boldsymbol{u}(t) + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{B_{d,c}} \boldsymbol{d}(t) \tag{6.18}$$

Este modelo se utiliza para estimar la secuencia de control dentro del problema de optimización (6.11), para este caso se eligieron las siguientes matrices de ponderación. El peso dado a la minimización del error de estados en el proceso de optimización está representado por la siguiente matriz.

$$Q_x = \begin{bmatrix} 90 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 90 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 30 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 35 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 \end{bmatrix} \tag{6.19}$$

El peso dado a la minimización de la secuencia de control, así como el peso dado a la taza de cambio de la secuencia de control están dados por:

$$R_u = \begin{bmatrix} 35 & 0 & 0 \\ 0 & 35 & 0 \\ 0 & 0 & 2 \end{bmatrix} \qquad R_\Delta = \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 0.0025 \end{bmatrix} \tag{6.20}$$

Las restricciones se establecieron únicamente en las entradas de control con el objetivo de evitar la saturación de los actuadores.

$$\mathbb{U} = \begin{bmatrix} \phi_{min} \\ \theta_{min} \\ \dot{z}_{min} \end{bmatrix} \leq \boldsymbol{u} \leq \begin{bmatrix} \phi_{max} \\ \theta_{max} \\ \dot{z}_{max} \end{bmatrix} \tag{6.21}$$

Los límites se establecieron tomando en cuenta los valores máximos que pueden alcanzar las entradas de control. De esta manera:

$$\mathbb{U} = \begin{bmatrix} -12\ [^\circ] \\ -12\ [^\circ] \\ -0.5\ [m/s] \end{bmatrix} \leq \boldsymbol{u} \leq \begin{bmatrix} 12\ [^\circ] \\ 12\ [^\circ] \\ 0.5\ [m/s] \end{bmatrix} \tag{6.22}$$

La descripción del problema de optimización se llevó dentro del entorno CVXGEN considerando $N_p = N_c = 20$, las librerías generadas fueron posteriormente utilizadas dentro de ROS para el desarrollo del controlador.

6.2.4.1. Implementación en ROS

El algoritmo de control fue programado como un nodo dentro de ROS. El programa principal se subscribe a los datos de odometría provenientes del tópico `/ardrone/odometry`, así como los estados de referencia del tópico `/ardrone/command/trajectory` para calcular la secuencia de control óptima. Las señales de control calculadas son publicadas en el tópico `/ardrone/command/roll_pitch_yawrate_thrust` mediante un mensaje del tipo `mav_msgs::RollPitchYawrateThrust`, debido a que él AR.drone recibe comandos a través de mensajes del tipo `geometry_msgs::Twist` es necesario normalizar los comandos de control, dicha tarea se lleva a cabo en el nodo `/ardrone/control_inputs_normalizer_node` y publicados en el tópico `/cmd_vel`.

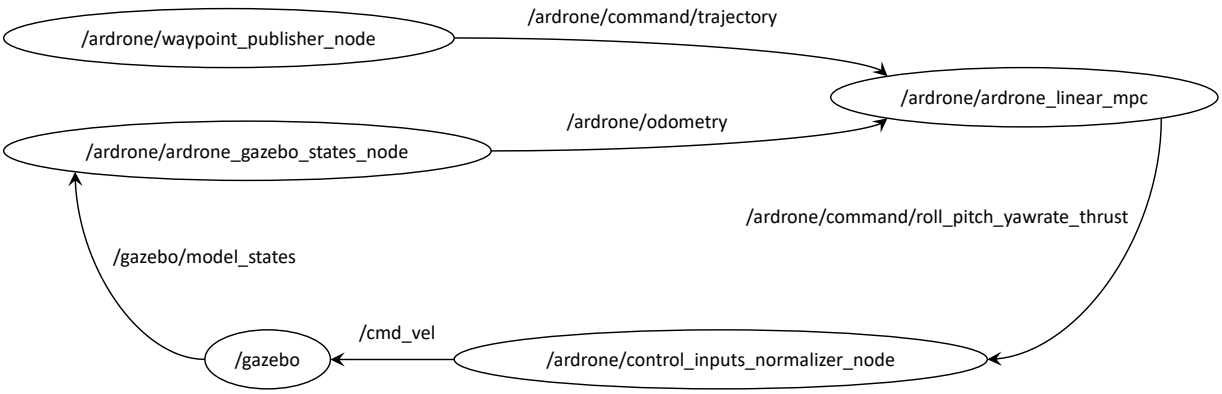


Figura 6.18: Lazo de control del vehículo AR.drone en ROS.

6.2.5. Implementación del controlador en el vehículo Bebop 2

En esta sección se procede a realizar la implementación del mismo controlador en el vehículo Bebop 2, en este caso se utilizan los parámetros dinámicos de la Tabla 6.1, así como las matrices de ponderación (6.19) y (6.20). Para este experimento de igual forma se consideraron restricciones en las entradas de control de acuerdo a la siguiente expresión:

$$\mathbb{U} = \begin{bmatrix} -10\ [^\circ] \\ -10\ [^\circ] \\ -0.5\ [m/s] \end{bmatrix} \leq \boldsymbol{u} \leq \begin{bmatrix} 10\ [^\circ] \\ 10\ [^\circ] \\ 0.5\ [m/s] \end{bmatrix} \tag{6.23}$$

La cuales corresponden al 50 % del valor máximo que puede alcanzar cada una de las entradas de control, mismas que fueron descritas en el Capítulo 5. Se han elegido valores conservativos para evitar la saturación de los actuadores, también para lograr transiciones entre puntos de referencia con una velocidad baja.

6.2.5.1. Implementación en ROS

Como se puede apreciar en la Figura 6.19 el nodo `/bebop/bebop_driver` se subscribe a los comandos de pilotaje a través del tópico `/bebop/cmd_vel` y a su vez este nodo publica los datos de odometría del robot por medio del tópico `/bebop/odom`. Debido a que los datos de odometría del Bebop son publicados a una frecuencia de 5 Hz (la cual se considera una velocidad relativamente baja en el diseño de controladores) es necesario volver a publicar dichos datos a una frecuencia mayor, en este caso se ha creado un nodo el cual simplemente se subscribe al tópico `/bebop/odom` y vuelven a publicar estos datos a una frecuencia de 100 Hz en el tópico `/bebop/odometry`, estos datos en conjunto con los comandos de referencia provenientes del tópico `/bebop/command/trajectory` ingresan al nodo `/bebop/bebop_linear_mpc_node` el cual se encarga de calcular las señales de control adecuadas para conducir al robot hacia los valores de referencia. Las señales de control calculadas son publicadas en el tópico `/bebop/command/roll_pitch_yawrate_thrust` mediante un mensaje del tipo `mav_msgs::RollPitchYawrateThrust`, debido a que el Bebop recibe únicamente comandos mediante mensajes del tipo `geometry_msgs::Twist` es necesario normalizar los comandos de control, dicha tarea se lleva a cabo en el nodo `/bebop/control_inputs_normalizer_node` los valores resultantes son publicados en el tópico `/bebop/cmd_vel` lo cual a su vez permite cerrar el lazo de control.

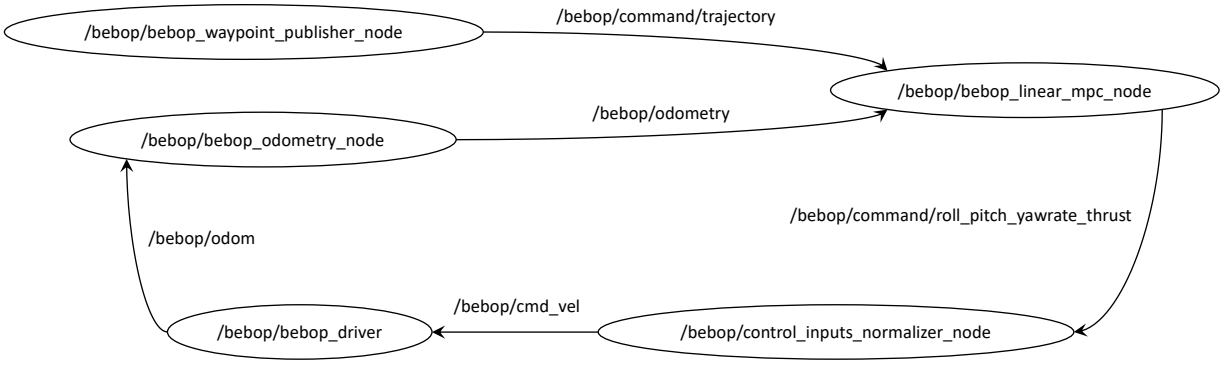


Figura 6.19: Lazo de control del vehículo Bebop 2 en ROS.

7 Análisis de resultados

7.1. Simulación del MPC lineal con el vehículo AR.drone

Los resultados obtenidos tras la implementación del controlador MPC en el modelo del AR.drone se muestran a continuación. La trayectoria de referencia consiste en trazar un cuadrado de 10 metros de longitud en cada lado a 4 metros de altura, los resultados obtenidos se muestran en la Figura 7.1.

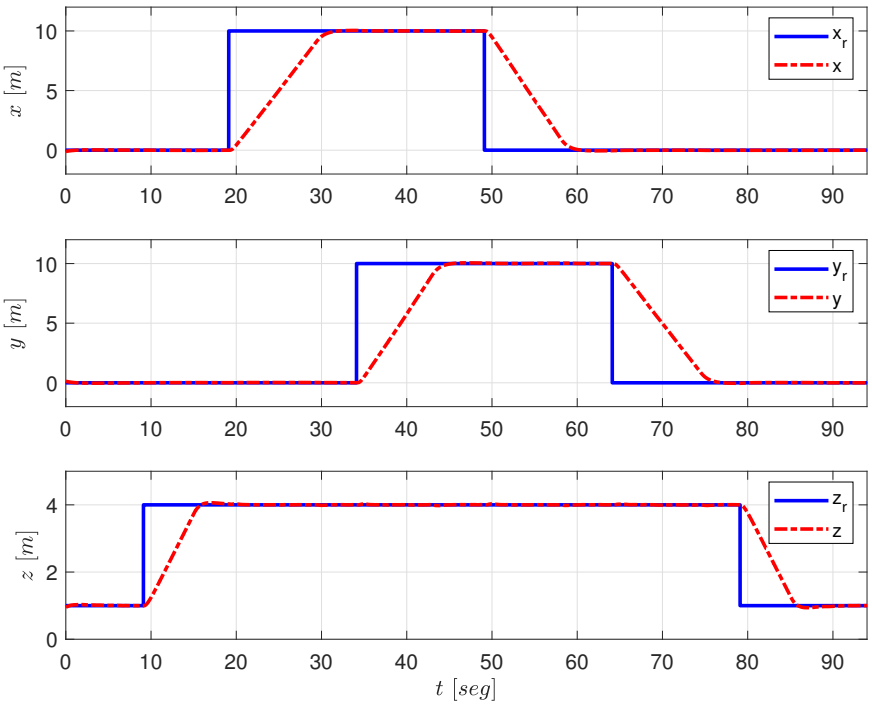


Figura 7.1: Variables de estado de la posición del AR.drone.

Las líneas sólidas indican los valores de referencia y las líneas punteadas representan a los estados actuales del robot.

Como se puede apreciar el robot alcanza los valores de referencia de manera satisfactoria. Para las componentes x e y el robot se desplaza de 0 a 10 metros con el objetivo de trazar la trayectoria en forma de cuadrado y después de alrededor de 30 segundos el robot vuelve a su posición inicial. De manera similar el dron se eleva a una altura de 4 metros y al final del recorrido el vehículo desciende a 1 metro de altura en donde se mantiene en vuelo estacionario de manera indefinida a la espera de nuevos puntos de referencia.

En la Figura 7.2 se aprecia la trayectoria trazada por el vehículo. La línea punteada indica la trayectoria del robot y la línea sólida el trayecto de referencia. Como se puede notar, el algoritmo de control cumple de manera eficiente con su tarea al mantener al robot muy cercano a la trayectoria de referencia.

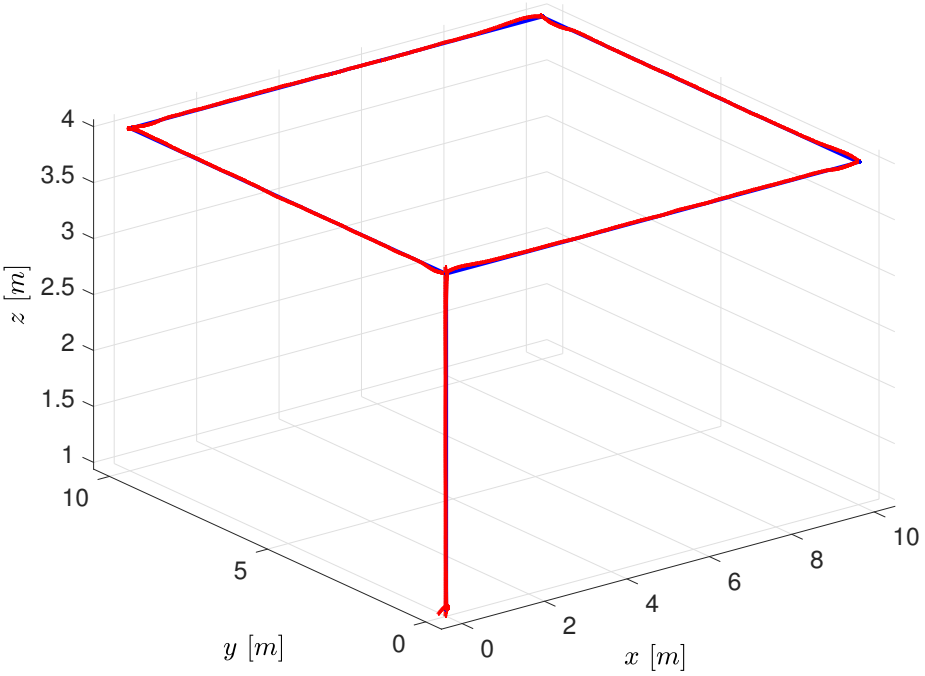


Figura 7.2: Trayectoria en el espacio 3D trazada por el AR.drone.

Una forma más clara de observar el desempeño del controlador es a través de los errores de estado. En la Figura 7.3 se muestran los errores de estado para la posición. Las protuberancias ocurren cada vez que el robot se desplaza de un punto hacia otro, en este caso, cada vez que se efectúa un desplazamiento en el plano $x - y$. Se inicia con un error de 10 metros y, a medida que el tiempo evoluciona la magnitud de los errores convergen a cero. De manera similar ocurre con la altitud del vehículo, inicialmente se cuenta con un error de 3 metros, sin embargo, durante el recorrido el controlador tiene la capacidad de mantener el robot a 4 metros de altura y al final conducirlo a 1 metro de altitud en donde se mantendrá en vuelo estacionario de manera indefinida.

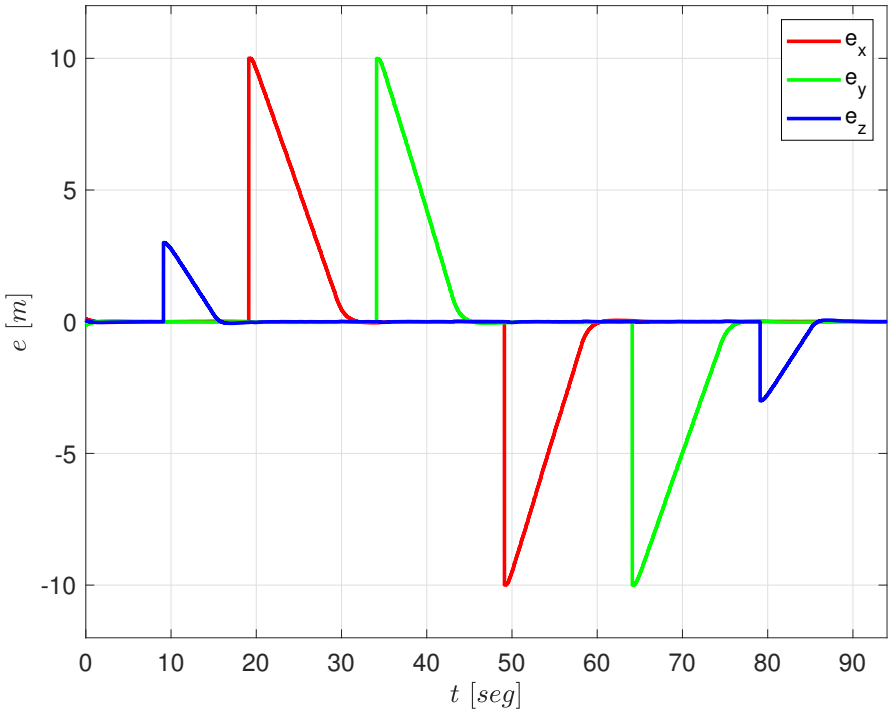


Figura 7.3: Errores de posición.

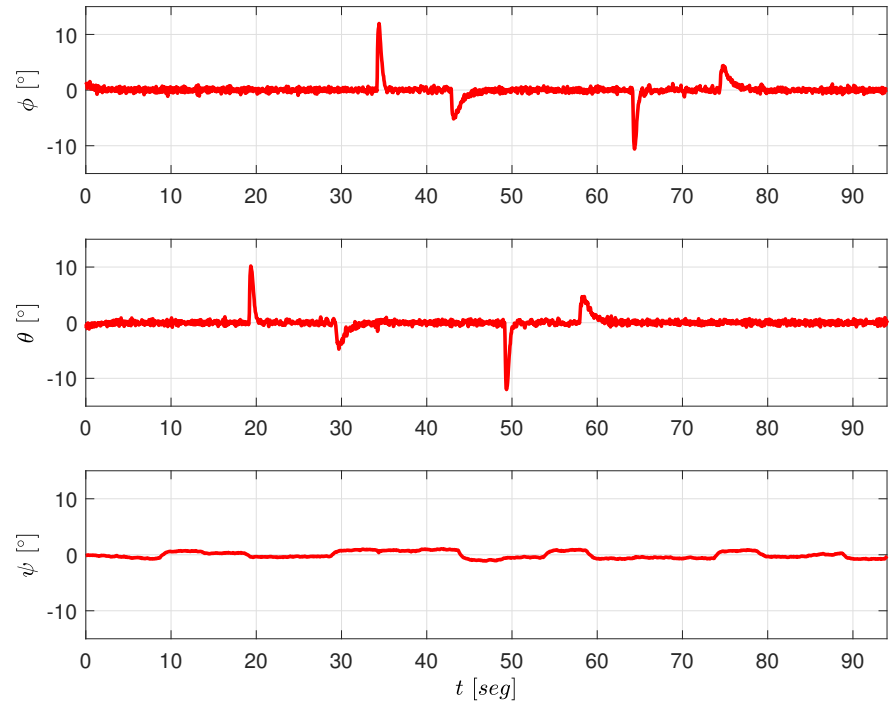


Figura 7.4: Variables de estado de la orientación del AR.drone.

En la Figura 7.4 se muestran las variables de estado de la orientación. Como se puede apreciar, en la mayoría del recorrido los ángulos de orientación se mantienen oscilando alrededor de cero, únicamente se tienen picos en los ángulos *pitch* y *roll* cada vez que el robot efectúa un desplazamiento en el plano $x-y$, es importante recordar que para inducir un desplazamiento dentro del plano 2D es necesaria una inclinación del robot. Debido a que no existe control para la orientación, ángulo en *yaw* se mantiene en cero a lo largo del trayecto.

En la Figura 7.5 se presentan las velocidades lineales para cada uno de los ejes. Como los resultados lo demuestran las velocidades en el plano $x-y$ alcanzan valores máximos de alrededor 1 m/seg, para el caso de la velocidad vertical se alcanza un valor máximo de 0.5 m/seg. Lo anterior permite que el robot se desplace a una velocidad relativamente baja.

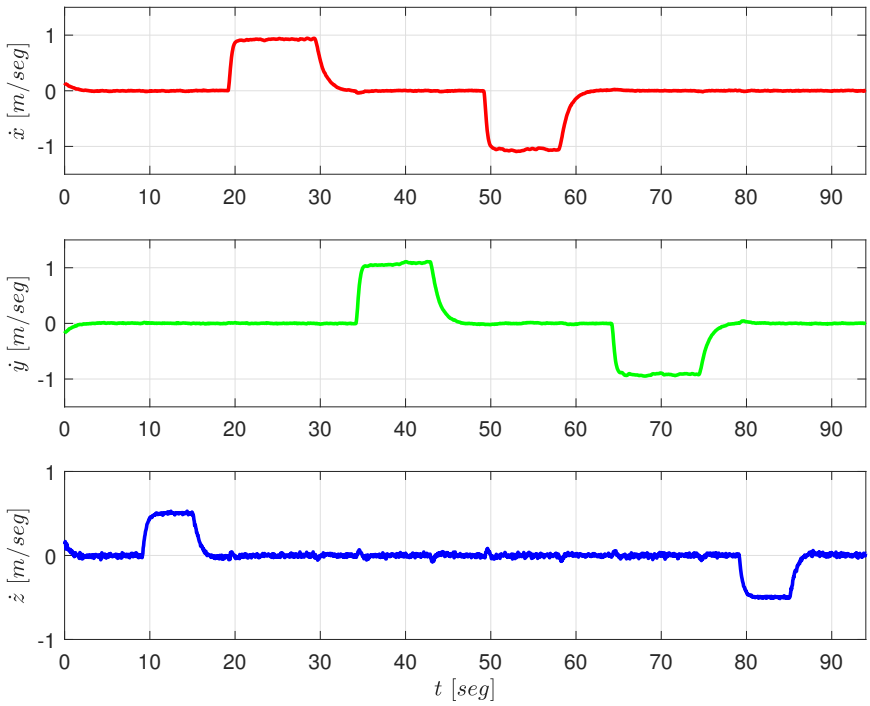


Figura 7.5: Velocidades lineales del AR.drone.

Las señales de control calculadas y aplicadas al vehículo para el seguimiento de la trayectoria de referencia aparecen en la Figura 7.6. Como se puede notar los valores calculados se encuentran dentro de los límites establecidos en la ecuación (6.22). El algoritmo de control es capaz de alcanzar el objetivo de control a pesar de las restricciones impuestas en las entradas de control, esta característica resulta de gran importancia cuando se trata de la implementación en plataformas reales, debido a que puede darse el caso en que los actuadores entren en zona de saturación y consecuentemente puedan ocurrir daños de hardware lo cual se puede traducir en perdidas económicas.

El entorno de Gazebo nos permite simular y probar algoritmos de control en modelos de robots los cuales se comportan como si se tratase de plataformas reales, esto permite realizar experimentos sin poner en riesgo la integridad del robot real.

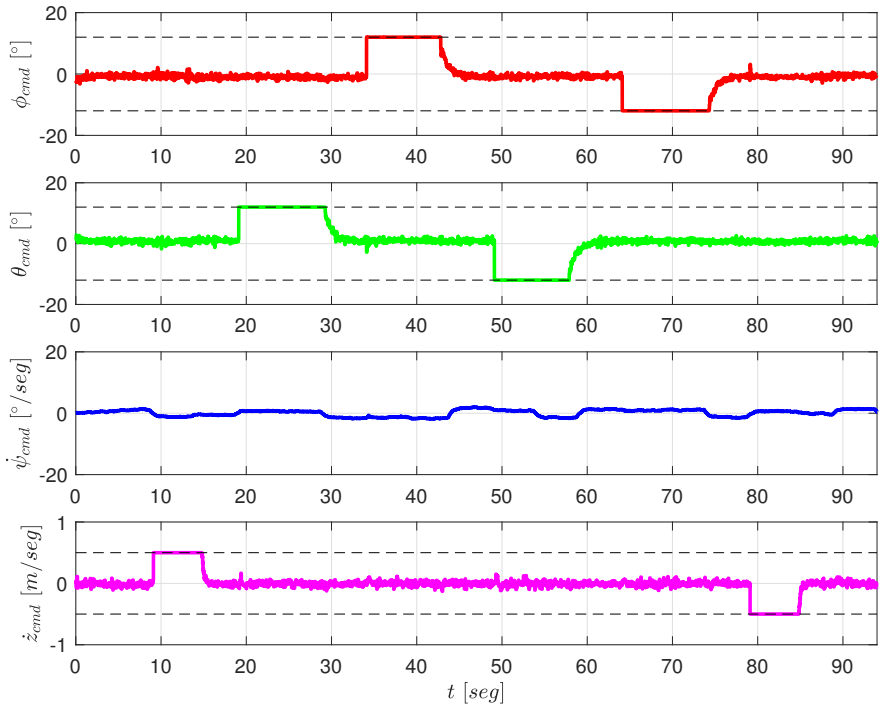


Figura 7.6: Señales de control calculadas por el controlador.

En este caso las restricciones se impusieron de acuerdo a los valores máximos que puede alcanzar él AR.drone, sin embargo, se pudieron haber elegido valores más conservativos para que el vehículo lograra transiciones suaves. El manejo de restricciones es quizás la característica más importante de los controladores MPC, aunque se podría pensar que bastaría simplemente con implementar funciones de saturación a cada entrada de control, pero no es el caso, en la literatura se discute esta situación y se hace la comparación entre el manejo de restricciones usando funciones de saturación y mediante la incorporación de las mismas dentro del problema de optimización en un controlador MPC, los resultados arrojan que las funciones de saturación pueden ocasionar que el sistema se vuelva inestable produciendo oscilaciones en las secuencias de control.

7.2. Implementación del MPC lineal en el Bebop 2

En este apartado se muestran los resultados obtenidos tras la implementación del control MPC para el control de posición. Para este caso se han elegido dos trayectorias en el espacio 3D con el objetivo de evaluar el desempeño del controlador.

7.2.1. Trayectoria 1

Esta trayectoria consiste en un cuadrado de 10×10 metros, misma que debe ser trazada a una altura de 4 metros. Tras ejecutar el controlador se obtuvieron los siguientes resultados: En la Figura 7.7 se aprecian las variables de estado correspondientes a la posición del robot, en la cual se incluyen los valores de referencia así como los estados actuales del robot. Como se puede notar el controlador cumple con su tarea al mantener los estados del vehículo lo más cercano a los valores de referencia. Una forma más clara de observar el recorrido del vehículo es mediante la representación de las variables de estado de la posición en el espacio 3D tal como se muestra en la Figura 7.8. A pesar de la existencia de pequeños desvíos durante el recorrido del robot, el algoritmo tiene la capacidad de corregir dichos errores para dirigir al vehículo hacia la referencia deseada. El desempeño del controlador en la plataforma experimental se puede apreciar de mejor forma en el vídeo del siguiente enlace[1]. Los errores de posición se muestran en la Figura 7.9.

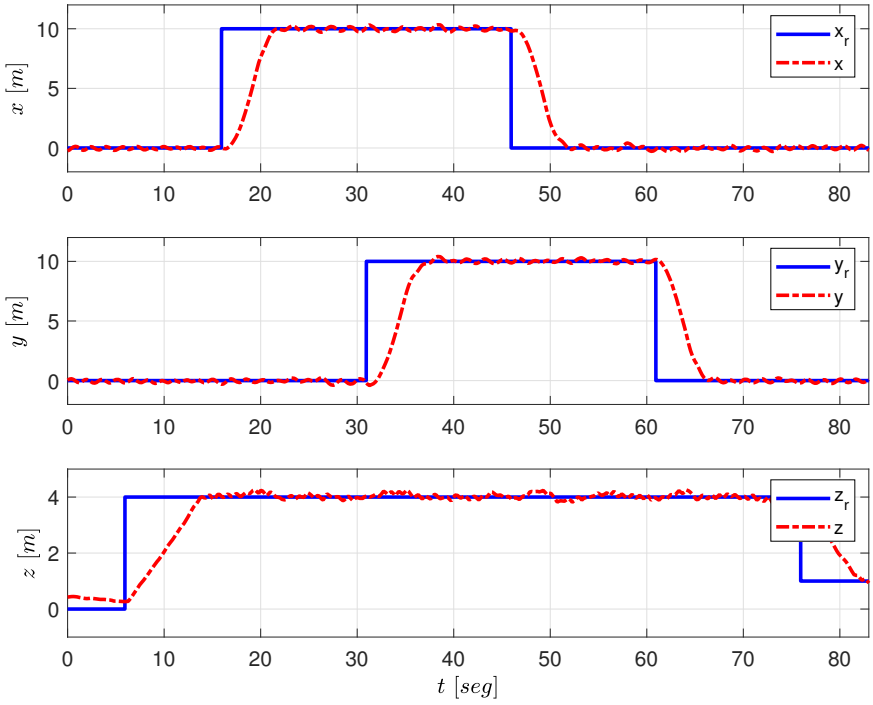


Figura 7.7: Variables de estado de la posición del Bebop.

[1]https://youtu.be/NI9R8ZSqui8

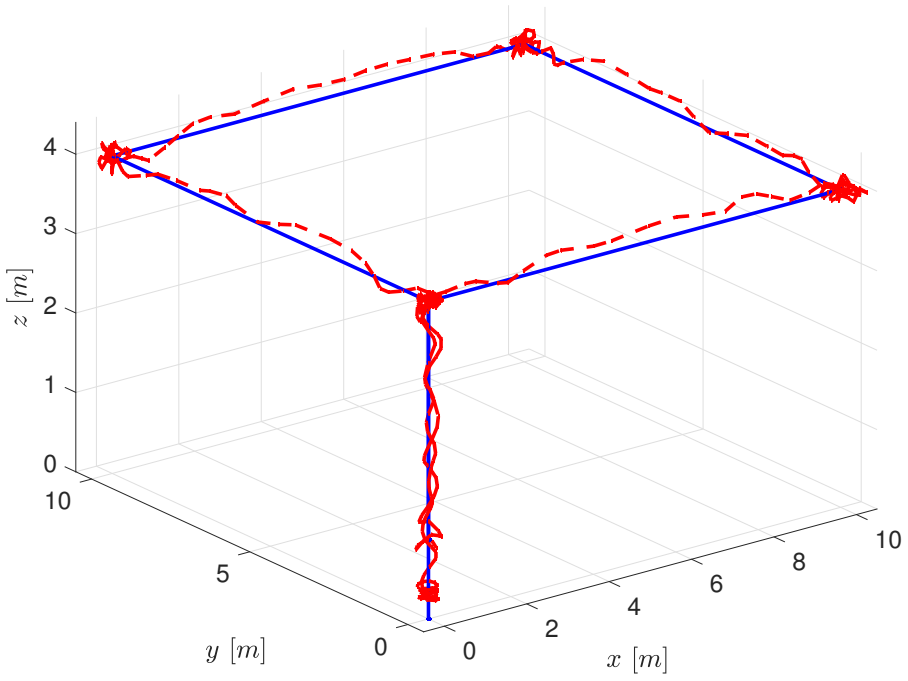


Figura 7.8: Trayectoria en el espacio 3D trazada por el Bebop.

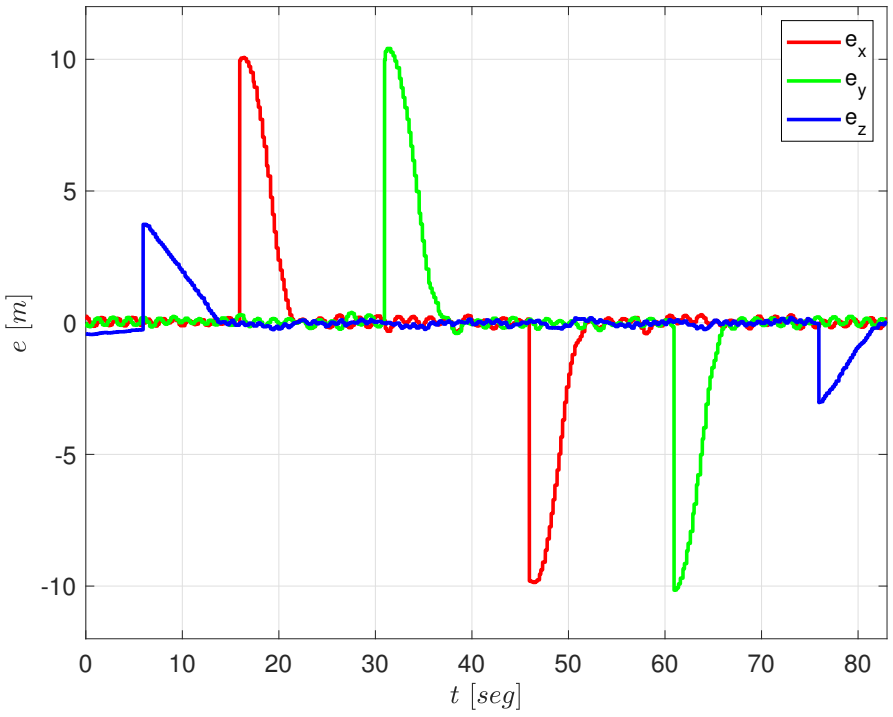


Figura 7.9: Errores de posición.

Los estados correspondientes a la orientación del robot se muestran en la Figura 7.10, en la cual se puede notar que los ángulos *roll* y *pitch* presentan oscilaciones las cuales inducen un balanceo en el vehículo. Lo anterior se debe principalmente a la baja taza de transmisión de los datos de odometría por parte del driver del robot. Por tal motivo es necesario un sistema de estimación de estados más robusto para mejorar los resultados obtenidos y de esta forma reducir la magnitud de las oscilaciones.

Las velocidades lineales para cada uno de los ejes cartesianos se presentan en la Figura 7.11, en la cual se puede notar que el robot se desplaza a una velocidad de aproximadamente 2.5 m/seg para movimientos en el plano $x-y$. Para desplazamientos verticales, el Bebop alcanza una velocidad de alrededor 0.5 m/seg.

El esfuerzo de control que entrega cada una de las entradas se muestra en la Figura 7.12 en donde se puede ver que cada señal de control obedece las restricciones establecidas en la ecuación (6.23). Para el caso de los ángulos *roll* y *pitch* ambas señales se encuentran restringidas entre -10 [°], 10 [°] lo cual permite al Bebop efectuar transiciones suaves en el plano $x-y$. En este caso no se realizó el control del ángulo *yaw* por tal motivo la entrada $\dot{\psi}_{in}$ se mantiene en cero a lo largo del recorrido, es decir el vehículo se mantiene en todo momento orientado hacia el polo norte magnético $\psi = 0$. El empuje vertical se mantuvo dentro de los valores -0.5 $[m/seg]$, 0.5 $[m/seg]$ de igual manera obedeciendo las restricciones impuestas durante la etapa de diseño. Es importante recordar que las restricciones se impusieron considerando únicamente el 50 % de total que cada una de las entradas de control puede proporcionar.

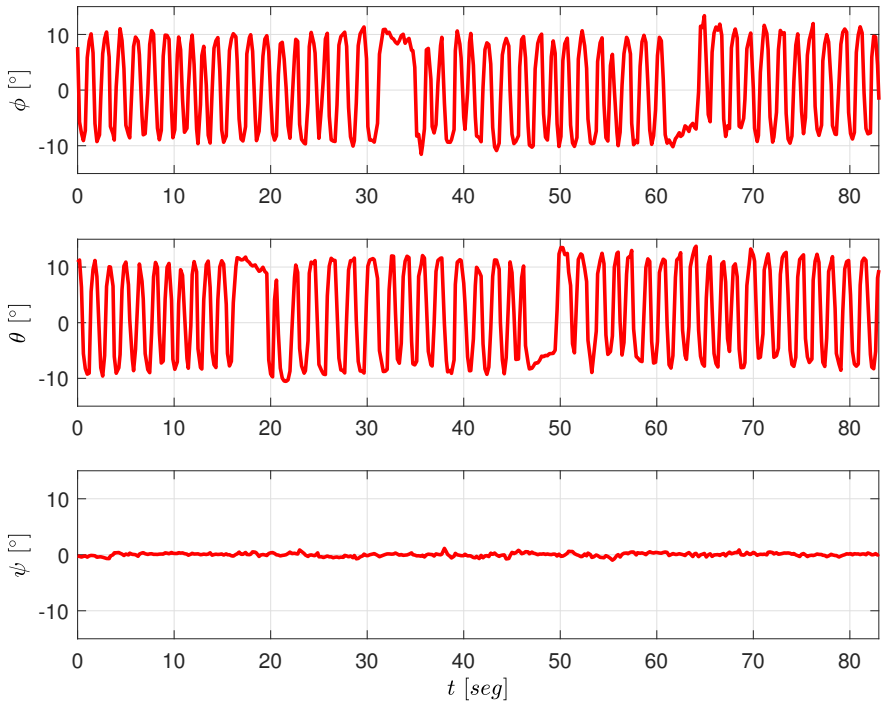


Figura 7.10: Variables de estado de la orientación del Bebop.

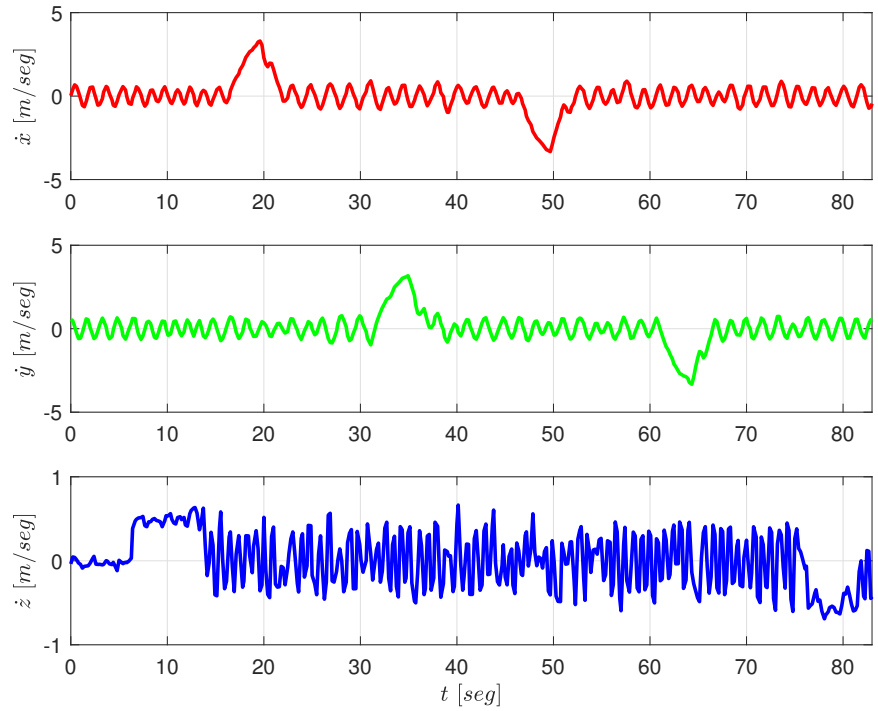


Figura 7.11: Velocidades lineales del Bebop.

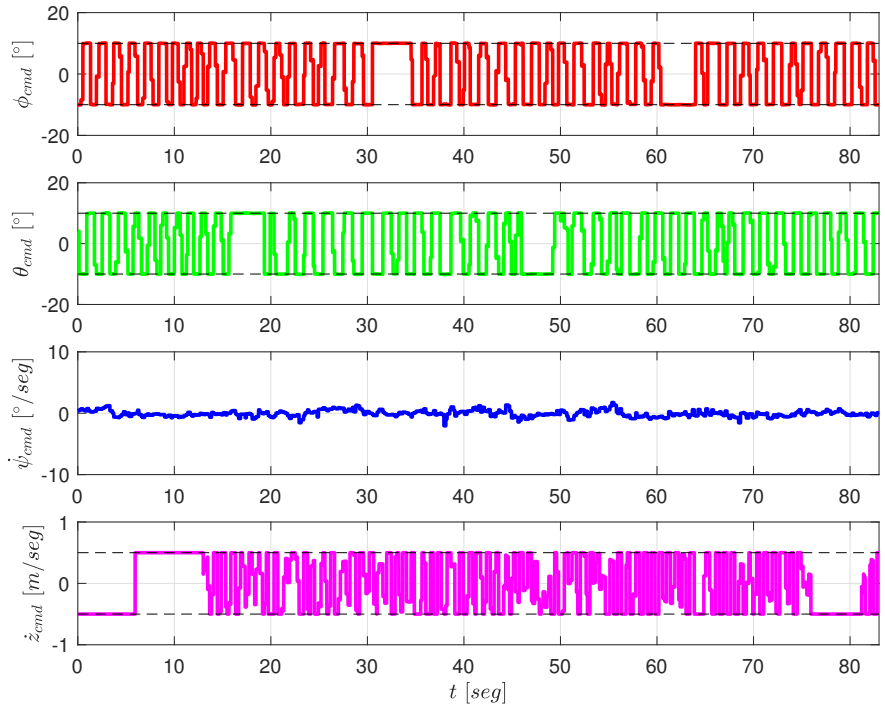


Figura 7.12: Señales de control calculadas por el controlador.

7.2.2. Trayectoria 2

En este caso se decidió implementar una trayectoria con un grado de complejidad mayor con respecto a la trayectoria anterior. La referencia deseada se trata de una trayectoria tipo escalera en el espacio tridimensional. El robot realiza movimientos a lo largo del eje x de una longitud de 8 metros, y a medida que el vehículo se eleva igualmente se desplaza de manera progresiva a lo largo del eje y, la altura máxima que alcanza el robot es de 10 metros, la cual una vez alcanzada el robot desciende hasta 1 metro de altura en donde se mantiene en vuelo estacionario hasta que finalmente recibe un comando de aterrizaje.

Los estados del robot así como los valores de referencia se muestran en la Figura 7.13 en donde se puede notar que el controlador cumple con su tarea de conducir al Bebop hacia los valores deseados. En el caso de la componente en x el vehículo efectúa un desplazamiento de manera alterna de 0-8 metros, la referencia establecida se alcanza de manera correcta sin presentar sobreimpulsos en cada transición. Para la componente en y el vehículo se desplaza desde 0 metros hasta alcanzar un recorrido de 16 metros en donde se mantiene de manera indefinida una vez finalizado el trayecto. En la misma figura se muestra como el robot asciende de forma vertical progresivamente hasta alcanzar una altitud máxima de 10 metros, a partir de la cual comienza a descender. En la etapa de descenso el vehículo traza un trayecto de comportamiento lineal, esto se debe principalmente a que el robot alcanza la velocidad máxima de descenso la cual fue establecida como una restricción dentro del problema de optimización, si se deseara tener asensos y descensos más rápidos bastaría con relajar la restricción antes mencionada, obviamente sin superar el valor nominal del robot.

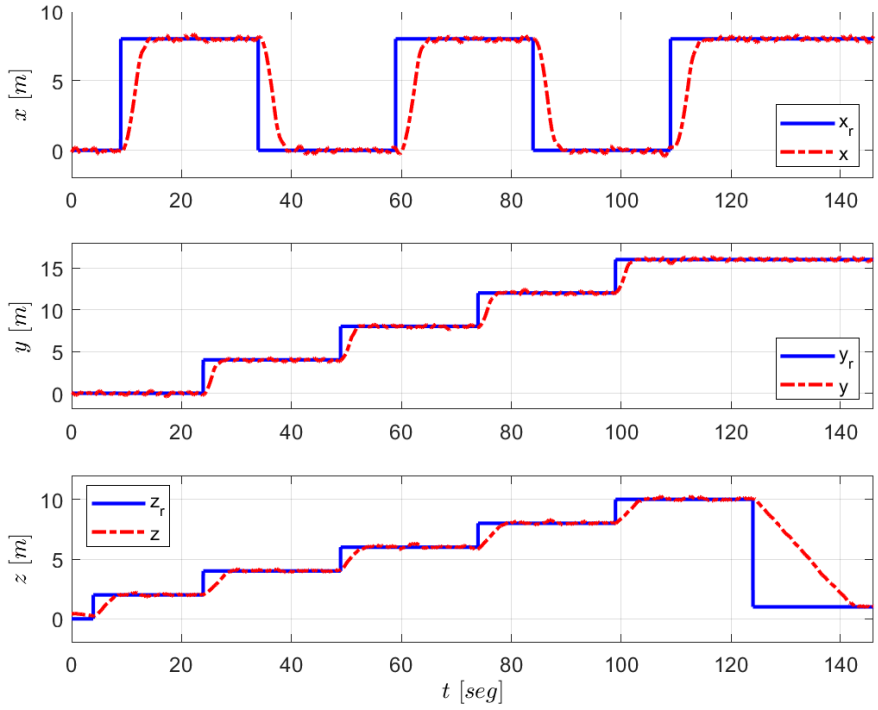


Figura 7.13: Variables de estado de la posición del Bebop.

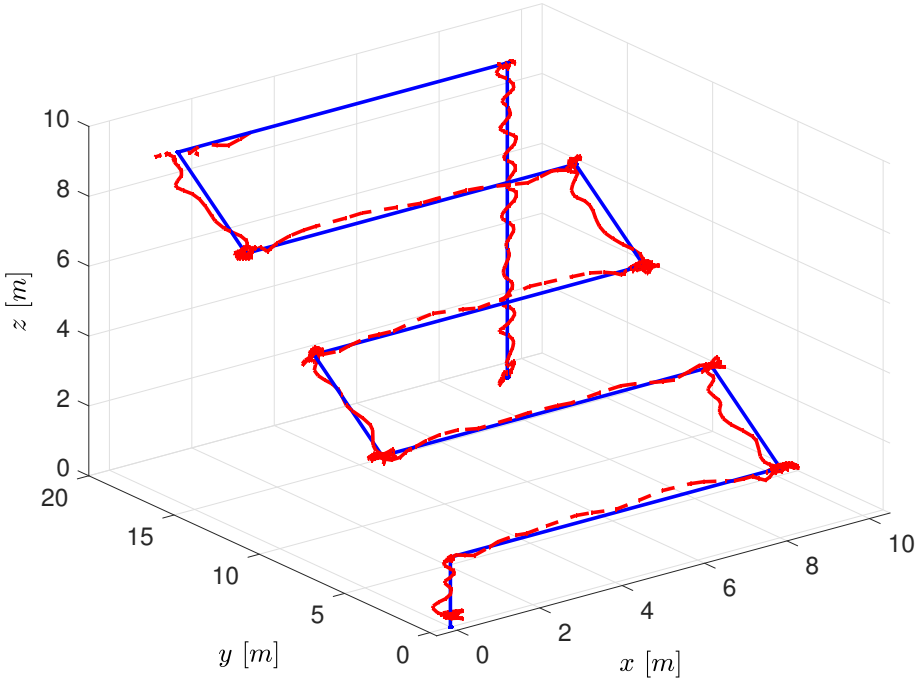


Figura 7.14: Trayectoria en el espacio 3D trazada por el Bebop.

Una manera más clara de apreciar el trayecto del robot es mediante la representación de las variables de estado de la posición en el espacio tridimensional tal como se muestra en la Figura 7.14. El desempeño del controlador en la plataforma experimental para el seguimiento de la trayectoria de referencia se puede apreciar en el siguiente enlace[2].

Los errores de estado se ilustran en la Figura 7.15, en donde los picos y valles ocurren cada que el robot se le asigna un nuevo punto de referencia. El error en x alterna entre valores positivos y negativos debido a que el trayecto a lo largo de este eje consiste en un desplazamiento de 0-8 metros y viceversa. El error en y se compone de valores positivos, ya que el robot se desplaza únicamente en dirección positiva sobre este eje. Finalmente, el error en z inicia con picos positivos debido a que es el momento en que el robot efectúa el asenso progresivo, al final del recorrido se tiene un pico negativo el cual ocurre cuando el dron comienza a descender. Es importante mencionar que todos los errores tienden asintóticamente a cero una vez alcanzados los valores de referencia.

Las variables de estado correspondientes a la orientación aparecen en la Figura 7.16, en donde se tienen los ángulos *roll* y *pitch* los cuales presentan oscilaciones, dicho fenómeno se atribuye principalmente a la baja taza de publicación de los datos de odometría por parte del driver. El ángulo en *yaw* se mantiene en cero a lo largo de todo el recorrido debido a que en este caso no se cuenta con control de orientación.

[2]https://youtu.be/rTkFIRUJPKw

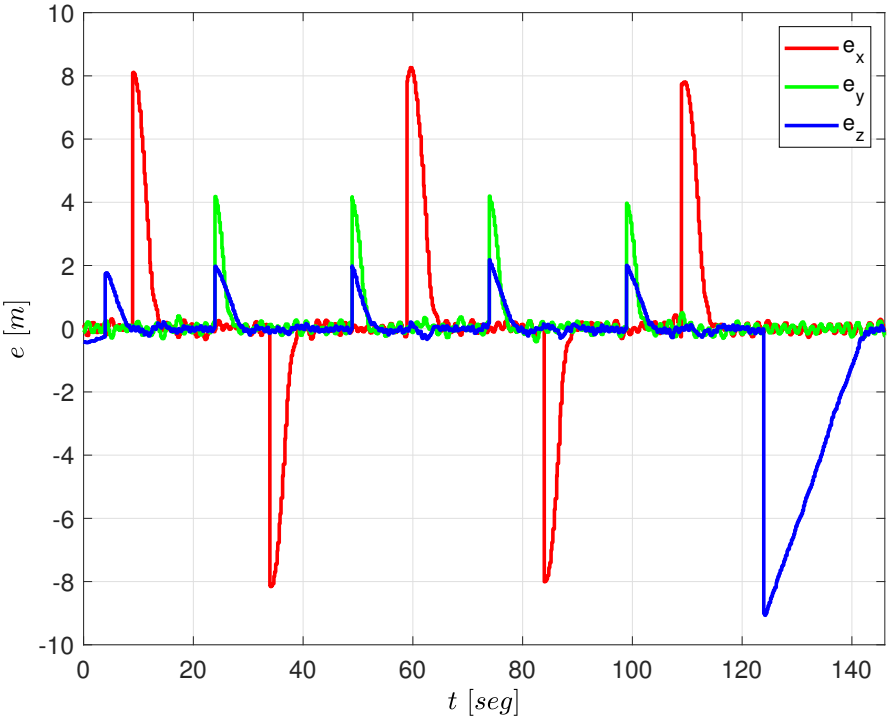


Figura 7.15: Errores de posición.

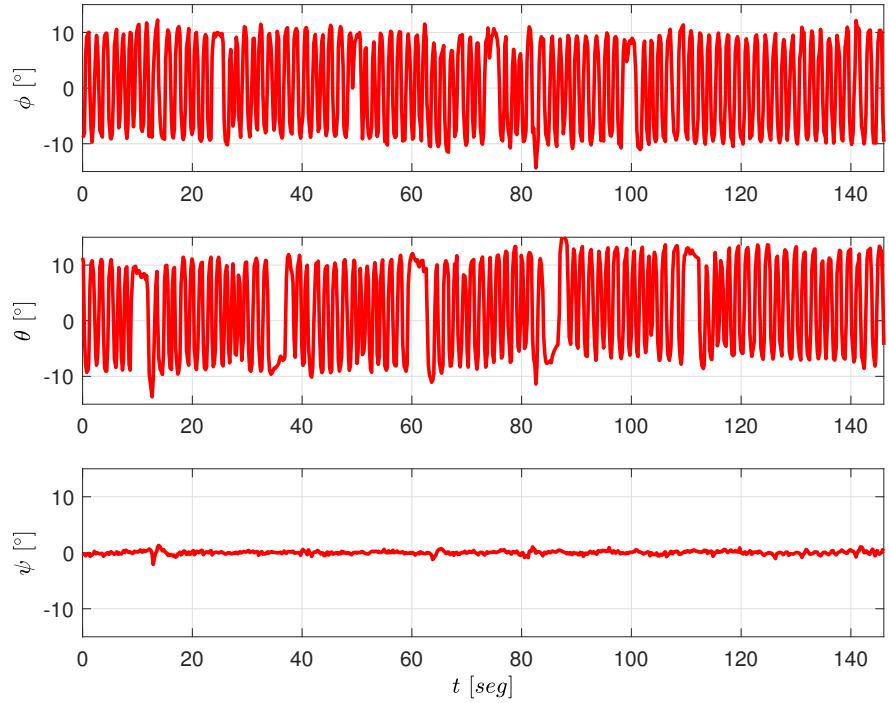


Figura 7.16: Variables de estado de la orientación del Bebop.

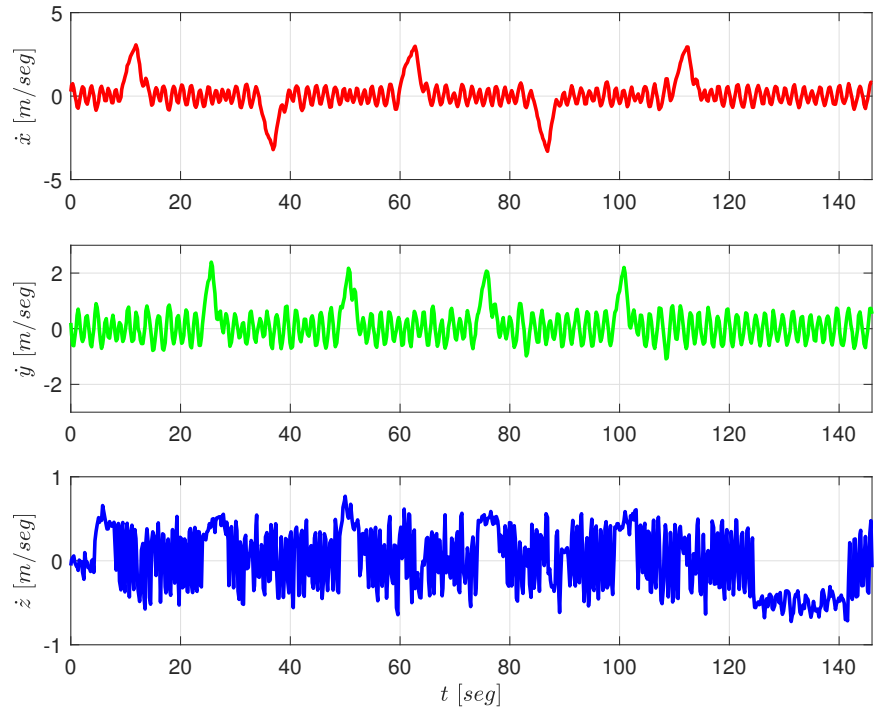


Figura 7.17: Velocidades lineales del Bebop.

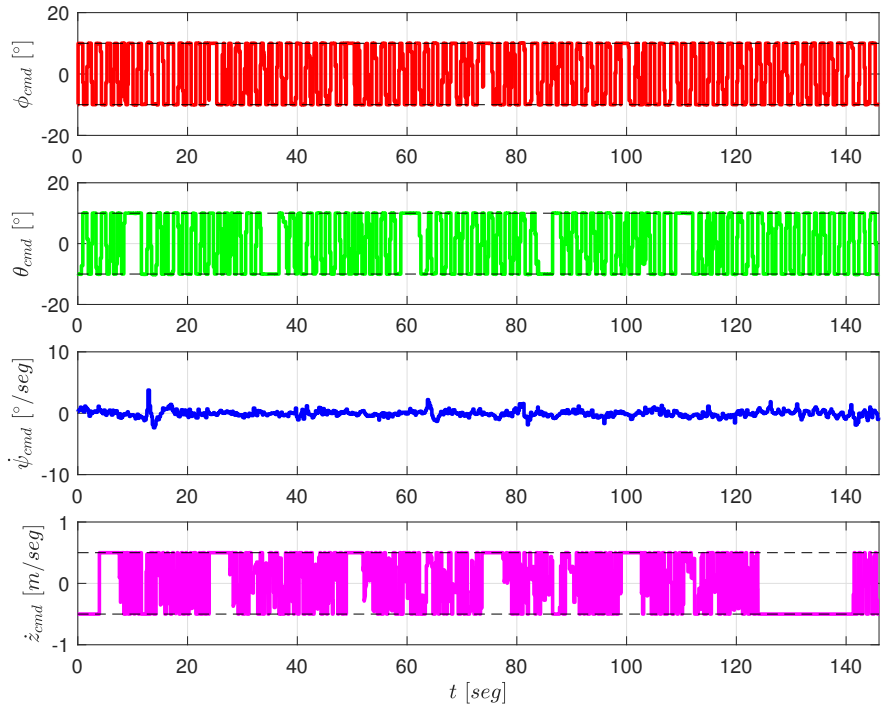


Figura 7.18: Señales de control calculadas por el controlador.

Las velocidades lineales se incluyen en la Figura 7.17, la velocidad a lo largo del eje x presenta picos y valles los cuales se producen cada vez que el robot se desplaza hacia adelante y hacia atrás de manera repetida. La componente de velocidad en y presenta valores positivos, ya que para esta trayectoria el robot únicamente se desplaza hacia la izquierda. La velocidad $\dot{z}$ presenta cambios cada que se asigna una nueva referencia para el ascenso vertical. Resulta interesante observar que esta componente de velocidad se mantiene dentro de los límites que se establecieron en la entrada de control para el empuje vertical, esta velocidad no corresponde a la señal de control sino a una variable de estado del robot. Es importante mencionar que una vez alcanzado un punto de referencia las velocidades se mantienen oscilando alrededor de cero, ya que teóricamente el robot alcanza una posición constante.

Las entradas de control se muestran en la Figura 7.18 en la cual se puede notar que, de forma similar al experimento anterior, los valores aplicados a la planta obedecen las restricciones impuestas. Los ángulos ϕ_{cmd} y θ_{cmd} se mantienen acotados dentro de los límites $-10°, 10°$. Para el caso de $\dot{\psi}_{cmd}$ el valor se mantiene oscilando alrededor de cero durante todo el trayecto, ya que en este caso no se efectúa el control de orientación. Finalmente, la entrada de control $\dot{z}_{cmd}$ de igual manera se mantiene dentro de los límites establecidos, en este caso $-0.5\ m/seg, 0.5\ m/seg$. Las restricciones se establecieron al $50\,\%$ de valor máximo de cada una de las entradas de control, si se desearan transiciones más rápidas bastaría con modificar el valor de las restricciones en el proceso de optimización, siempre y cuando sin superar el valor nominal de cada entrada.

8 Conclusiones Generales

8.1. Conclusiones

En este trabajo se llevó a cabo el diseño de una estrategia de control MPC lineal para resolver el problema de seguimiento de trayectorias de un cuadricóptero. Como los resultados lo demuestran la estructura de control propuesta efectúa el seguimiento de trayectorias de forma satisfactoria. El algoritmo es capaz de calcular la secuencia de control óptima para que el robot siga una trayectoria deseada. En este caso fueron consideradas restricciones en las señales de control las cuales nunca fueron violadas y con ello se evitó que los actuadores entraran en zona de saturación.

El controlador MPC lineal fue evaluado primeramente en el simulador Gazebo haciendo uso del modelo AR.drone, los estados del robot fueron tomados directamente del simulador, por lo que tanto las posiciones como velocidades usadas fueron prácticamente las ideales. De esta manera los resultados obtenidos en la simulación fueron bastante buenos y sirvieron como punto de partida para la implementación en una plataforma real. El modelo dinámico se obtuvo tras colectar datos de las entradas y salidas del vehículo, los parámetros obtenidos fueron utilizados tanto para el modelo dinámico del vehículo en la simulación, así como en la plataforma real.

El mismo controlador fue implementado en el vehículo Bebop, en este caso se consideraron 2 trayectorias de referencia. Los estados del robot se tomaron de la odometría que el driver del vehículo calcula de manera interna, la desventaja que se presentó es que los datos son publicados a una frecuencia de 5 Hz por lo que de alguna manera afectó en el desempeño del controlador. Sin embargo, los resultados obtenidos son aceptables. Las trayectorias de referencia son seguidas por el robot de manera precisa. En este caso se creó un nodo para visualizar las trayectorias de referencia así como la trayectoria trazada por el robot.

La inclusión de restricciones durante el diseño de la ley de control permitió diseñar una estrategia de control más robusta. En este caso únicamente se establecieron restricciones en las entradas de control, sin embargo, de igual forma se pudieron haber impuesto restricciones en los estados del sistema, por ejemplo se puede dar el caso en el que se requiera que el vehículo no supere una altura determinada, o que sobrepase algún límite de velocidad de desplazamiento. El control predictivo basado en modelo es una estrategia que había sido

empleado únicamente en procesos industriales de dinámicas lentas, sin embargo debido a sus características ya descritas y a los avances en la tecnología de procesadores, este tipo de controladores han cobrado gran interés para su implementación en la robótica móvil.

Como su nombre lo indica los controladores MPC dependen principalmente del modelo del sistema, por tal motivo resulta fundamental conocer el modelo dinámico de la planta a controlar. Por lo que, para este caso es de vital importancia conocer las distintas técnicas de identificación. Puesto que en la mayoría de los casos el sistema a controlar en primera instancia carecerá de un modelo matemático.

8.2. Trabajo futuro

La etapa de control solo se trata de una parte en el proceso de navegación autónoma. Para lograr que un robot móvil sea completamente autónomo es necesario dotarlo de la habilidad de estimar los estados actuales de manera precisa, además de integrar un sistema que sea capaz de calcular trayectorias libres de colisiones.

A pesar de haber obtenido buenos resultados, el controlador se le pueden realizar varias mejoras. Como trabajo futuro se propone diseñar un sistema para la estimación de estados mucho más robusto, como se sabe el sistema actual está basado en la cámara del robot. El sistema puede mejorar si se integra una cámara más y de esta manera utilizar algoritmos de visión estereoscópica los cuales en teoría resultan más ser precisos. Por otra parte el modelo dinámico puede ser estimado de mejor manera si se conocen los estados de manera precisa, en este caso se puede hacer uso de sistemas de captura de movimiento para tal propósito.

Debido a distintas circunstancias el controlador no pudo ser evaluado en laboratorio, sin embargo, en un futuro la estrategia de control puede ser evaluada con la ayuda de un sistema de captura de movimiento para verificar el verdadero desempeño del controlador exponiendo al vehículo a perturbaciones externas, tales como ráfagas de viento.

A lo largo del desarrollo de este trabajo se ha ganado experiencia en el control de cuadricópteros, desde la etapa de identificación, estimación de estados, hasta la etapa de control. El siguiente paso lógico consiste en mejorar la estrategia de control mediante el diseño de un control MPC no lineal, así como el desarrollo de un estimador de estados más robusto e integrar todo en un solo sistema.

9

Apéndice A: Vídeos demostrativos

En esta sección se incluyen los vídeos demostrativos de los experimentos efectuados, así como los resultados obtenidos. Cada imagen contiene un enlace que direcciona hacia un sitio externo del vídeo en cuestión. En el primer vídeo se muestran los resultados obtenidos del cálculo de la odometría haciendo uso del *dataset* KITTI, en el segundo vídeo se muestran los experimentos realizados relacionados con la odometría del Bebop, los dos últimos vídeos muestran el desempeño del controlador MPC lineal implementado en el Bebop 2.

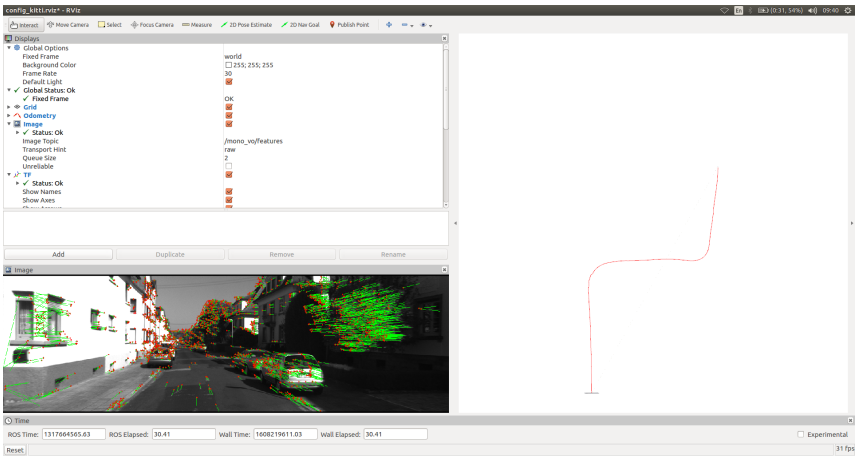


Figura 9.1: `https://youtu.be/A9sfLDSboQg`

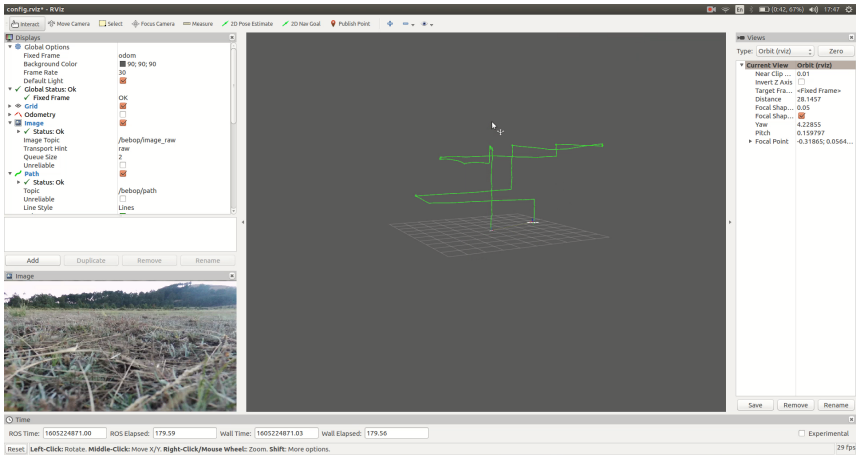


Figura 9.2: https://youtu.be/jZMR8T6uhwY

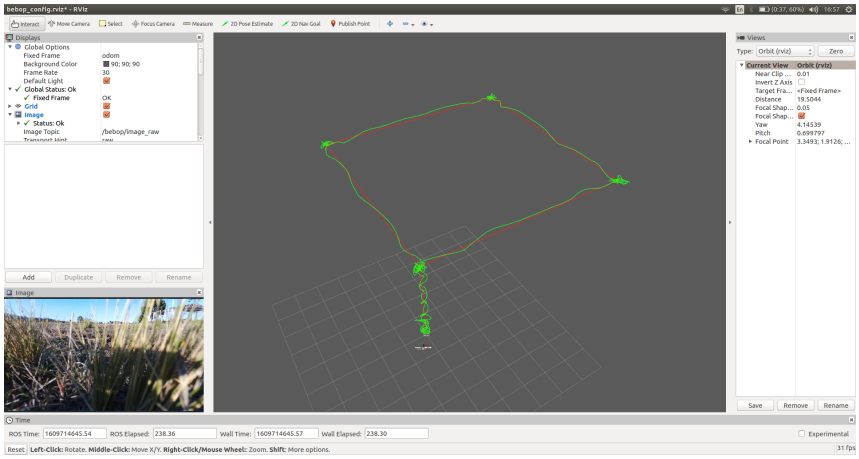


Figura 9.3: https://youtu.be/NI9R8ZSqui8

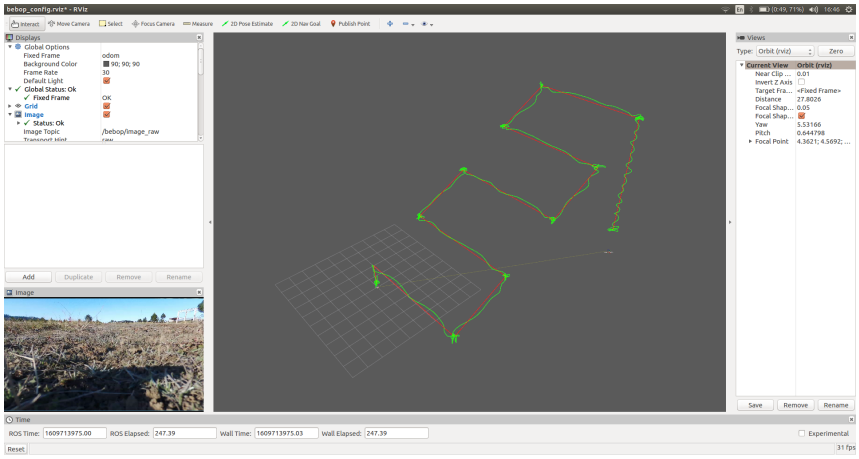


Figura 9.4: https://youtu.be/rTkFIRUJPKw

Bibliografía

[1] Norouzi Ghazbi Somayeh y col. «Quadrotors unmanned aerial vehicles: A review». En: *International Journal on Smart Sensing and Intelligent Systems* 9 (2016), págs. 309-333. ISSN: 1178-5608. DOI: 10.21307/ijssis-2017-872.

[2] Kim Jinho, Gadsden S. Andrew y A. Wilkerson Stephen. «A Comprehensive Survey of Control Strategies for Autonomous Quadrotors». En: 43 (2020), págs. 3-16. ISSN: 0840-8688. DOI: 10.1109/CJECE.2019.2920938.

[3] Sigalos Athanasios y col. «Design of a flight controller and periphelas for a quadcopter». En: *International Journal of Engineering Applied Sciences and Technology* 4 (2019), págs. 463-470. ISSN: 2455-2143. DOI: 10.33564/IJEAST.2019.v04i05.067.

[4] Sabatino Francesco. «Quadrotor control: modeling, nonlinear control design, and simulation». Estocolmo, Suecia: Royal Institute of Technology, jun. de 2015.

[5] Bouabdallah Samir, Noth André y Siegwart Roland. «PID vs LQ control techniques applied to an indoor micro quadrotor». En: 3 (oct. de 2004), págs. 2451-2456. DOI: 10.1109/IROS.2004.1389776.

[6] Miranda Colorado Roger. *DRONES - Modelado y control de cuadrotores*. 1.ª ed. Alfaomega, 2018, pág. 208. ISBN: 978 -607-538-314-9.

[7] Madani Tarek y Benallegue Abdelaziz. «Backstepping Control for a Quadrotor Helicopter». En: (oct. de 2006), págs. 3255-3260. DOI: 10.1109/IROS.2006.282433.

[8] Raffo Guilherme, Ortega Manuel y Rubio Francisco. «An integral predictive/nonlinear H-inf control structure for a quadrotor helicopter». En: *Automatica* 46 (oct. de 2010), págs. 29-39. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2009.10.018.

[9] Kurak Sevkuthan y Hodzic Migdat. «Control and Estimation of a Quadcopter Dynamical Model». En: 6 (mar. de 2018), págs. 63-75. ISSN: 2303-4521. DOI: http://dx.doi.org/10.21533/pen.v6i1.164.

[10] Castillo P. y col. «Modelado y estabilización de un helicóptero con cuatro rotores». En: *Revista Iberoamericana de Automática e Informática Industrial RIAI* 4 (ene. de 2007), págs. 41-57. ISSN: 1697-7912. DOI: 10.1016/S1697-7912(07)70191-7.

[11] Luukkonen Teppo. *Modelling and control of quadcopter*. Inf. téc. Espoo, Finlandia: Aalto University, 2011.

[12] Balasubramanian E. y Vasantharaj R. «Dynamic Modeling and Control of Quad Rotor». En: *International Journal of Engineering and Technology (IJET)* 5 (feb. de 2013), págs. 63-69. ISSN: 0975-4024.

[13] Li Lebao, Sun Lingling y Jin Jie. «Survey of advances in control algorithms of quadrotor unmanned aerial vehicle». En: (feb. de 2015), págs. 107-111. DOI: 10.1109/ICCT.2015.7399803.

[14] Lopes Renato V. y col. «Model Predictive Control applied to tracking and attitude stabilization of a VTOL quadrotor aircraft». En: *ABCM Symposium Series in Mechatronics* 5 (2012), págs. 176-185.

[15] Chen Xi y Wang Liuping. «Cascaded Model Predictive Control of a Quadrotor UAV». En: *2013 3rd Australian Control Conference* (nov. de 2013), págs. 354-359. DOI: 10.1109/AUCC.2013.6697298.

[16] Eskandarpour Abolfazl y Sharf Inna. «A constrained error-based MPC for path following of quadrotor with stability analysis». En: *Nonlinear Dynamics* 99 (abr. de 2019), págs. 899-918. DOI: 10.1007/s11071-019-04859-0.

[17] Chipofya Mapopa, Lee Deok Jin y Chong Kil To. «Trajectory Tracking and Stabilization of a Quadrotor Using Model Predictive Control of Laguerre Functions». En: *Control, Stability, and Qualitative Theory of Dynamical Systems 2014* 2015 (feb. de 2015). DOI: 10.1155/2015/916864.

[18] Raffo Guilherme V., Ortega Manuel G. y Rubio Francisco R. «MPC with Nonlinear H-inf Control for Path Tracking of a Quad-Rotor Helicopter». En: *IFAC Proceedings Volumes* 41 (jul. de 2008), págs. 8564-8569. ISSN: 1474-6670. DOI: 10.3182/20080706-5-KR-1001.01448.

[19] Silvagni Mario y col. «Multipurpose UAV for search and rescue operations in mountain avalanche events». En: *Geomatics, Natural Hazards and Risk* 8 (oct. de 2016), págs. 18-33. ISSN: 1947-5705. DOI: 10.1080/19475705.2016.1238852.

[20] Vazquez-Nicolas J.M. y col. «Towards automatic inspection: crack recognition based on Quadrotor UAV-taken images». En: *International Conference on Unmanned Aircraft Systems (ICUAS)* (jun. de 2018), págs. 654-659. ISSN: 2575-7296. DOI: 10.1109/ICUAS.2018.8453390.

[21] Sa Inkyu y col. «weedNet: Dense Semantic Weed Classification Using Multispectral Images and MAV for Smart Farming». En: *IEEE Robotics and Automation Letters* 3 (ene. de 2018), págs. 588-595. ISSN: 2377-3766. DOI: 10.1109/LRA.2017.2774979.

[22] ArsalanKhan Muhammad y col. «Unmanned Aerial Vehicle-based Traffic Analysis: A Case Study to Analyze Traffic Streams at Urban Roundabouts». En: *The 9th International Conference on Ambient Systems, Networks and Technologies (ANT)* 130 (2018), págs. 636-643. ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.04.114.

[23] Varentsov Mikhail y col. «Experience in the quadcopter-based meteorological observations in the atmospheric boundary layer». En: *IOP Conference Series: Earth and Environmental Science* 231 (feb. de 2019). DOI: 10.1088/1755-1315/231/1/012053.

[24] Alexis Kostas y col. «Coordination of Helicopter UAVs for Aerial Forest-Fire Surveillance». En: *Applications of Intelligent Control to Engineering Systems* (ene. de 2009), págs. 169-193. DOI: 10.1007/978-90-481-3018-4_7.

[25] Pagliari Diana y Pinto Livio. «Use of fisheye Parrot Bebop 2 images for 3D modelling using commercial photogrammetric software». En: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2 (mayo de 2018), págs. 813-820. DOI: 10.5194/isprs-archives-XLII-2-813-2018.

[26] García Luis Rodolfo y col. «Modeling the Quad-Rotor Mini-Rotorcraft». En: (ene. de 2013), págs. 23-34. DOI: 10.1007/978-1-4471-4399-4_2.

[27] Diebel James. «Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors». En: *Matrix* 58 (ene. de 2006).

[28] Reyes Córtes Fernando. *ROBÓTICA - Control de Robots Manipuladores*. 1.ª ed. Alfaomega, 2011, pág. 600. ISBN: 978-607-7071-90-7.

[29] Camacho Eduardo y Bordons Carlos. *Model Predictive Control*. 2.ª ed. Springer-Verlag London, 2007, pág. 405. ISBN: 978-1-85233-694-3.

[30] Maciejowski J.M. *Predictive Control With Constraints*. 1.ª ed. Pearson Education, 2001, pág. 256. ISBN: 978-0-20139-823-6.

[31] Wang Liuping. *Model Predictive Control System Design and Implementation Using MATLAB*. 1.ª ed. Springer-Verlag London, 2009, pág. 378. ISBN: 978-1-84882-330-3.

[32] Camacho F. Eduardo y Bordons Carlos. «Control Predictivo: Pasado, Presente y Futuro». En: *Revista Iberoamericana de Automática e Informática Industrial* 1.3 (2004), págs. 5-28.

[33] Kamel Mina, Burri Michael y Siegwart Roland. «Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles». En: *The 20th World Congress of the International Federation of Automatic Control* 50 (jul. de 2017), págs. 3463-3469. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2017.08.849.

[34] Siegwart Roland, Nourbakhsh Illah y Scaramuzza Davide. *Introduction to Autonomous Mobile Robots*. 2.ª ed. The MIT Press, 2011. ISBN: 978-0-26201-535-6.

[35] Corke Peter. *Robotics, Vision and Control: Fundamental Algorithms In MATLAB*. 2.ª ed. Springer, 2017. ISBN: 978-3-31954-412-0.

[36] Cheng Yang, Maimone Mark y Matthies Larry. «Visual odometry on the Mars Exploration Rovers». En: 1 (oct. de 2005), págs. 903-910. ISSN: 1062-922X. DOI: 10.1109/ICSMC.2005.1571261.

[37] Maimone Mark, Cheng Yang y Matthies Matthies Larry. «Two years of visual odometry on the mars exploration rovers: Field reports». En: *Journal of Field Robotics* 24.3 (2007), págs. 169-186. DOI: 10.1002/rob.20184.

[38] Nister David, Naroditsky Oleg y Bergen James. «Visual Odometry». En: *Computer Society Conference on Computer Vision and Pattern Recognition* (2004), págs. 652-659. ISSN: 1063-6919. DOI: 10.1109/CVPR.2004.1315094.

[39] Scaramuzza Davide y Fraundorfer Friedrich. «Visual Odometry Part I». En: *IEEE Robotics and Automation Magazine* 18.4 (2011), págs. 80-92. ISSN: 1070-9932. DOI: 10.1109/MRA.2011.943233.

[40] Scaramuzza Davide y Fraundorfer Friedrich. «Visual Odometry Part II». En: *IEEE Robotics and Automation Magazine* 19.2 (2012), págs. 78-90. ISSN: 1070-9932. DOI: 10.1109/MRA.2012.2182810.

[41] Harris C. y Stephens M. «A combined corner and edge detector». En: (1988), págs. 147-151. DOI: 10.5244/C.2.23.

[42] Shi Jianbo y Tomasi Carlo. «Good features to track». En: (jun. de 1994), págs. 593-600. ISSN: 1063-6919. DOI: 10.1109/CVPR.1994.323794.

[43] Rosten Edward y Drummond Tom. «Machine learning for high-speed corner detection». En: *9th European Conference on Computer Vision* (jul. de 2006), págs. 430-443. DOI: 10.1007/11744023_34.

[44] Lowe David. «Distinctive image features from scale-invariant keypoints». En: *International Journal of Computer Vision* (nov. de 2004), págs. 91-110. DOI: 10.1023/B:VISI.0000029664.99615.94.

[45] Herbert Bay y col. «Speeded-up robust features (SURF)». En: *Computer Vision and Image Understanding* 110.3 (2008), págs. 346-359. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2007.09.014.

[46] Leutenegger S., Chli M. y Siegwart R. «BRISK: Binary robust invariant scalable keypoints». En: *2011 International Conference on Computer Vision* (2011), págs. 2548-2555. ISSN: 1550-5499. DOI: 10.1109/ICCV.2011.6126542.

[47] Rublee E. y col. «ORB: An efficient alternative to SIFT or SURF». En: *2011 International Conference on Computer Vision* (2011), págs. 2564-2571. ISSN: 1550-5499. DOI: 10.1109/ICCV.2011.6126544.

[48] Lucas Bruce y Kanade Takeo. «An Iterative Image Registration Technique with an Application to Stereo Vision». En: *Proceedings of the 7th International Joint Conference on Artificial Intelligence* (1981), págs. 121-130.

[49] Longuet-Higgins H.C. «A computer algorithm for reconstructing a scene from two projections». En: *Nature* 293.10 (1981), págs. 133-135. DOI: 10.1038/293133a0.

[50] Parrot. *Parrot Bebop 2*. 2019. URL: https://www.parrot.com/es/drones/parrot-bebop-2 (visitado 17-05-2019).

[51] ROS. *Documentation*. 2020. URL: http://wiki.ros.org/ (visitado 27-02-2020).

[52] Pyo Yoonseok y col. *ROS Robot Programming (English)*. ROBOTIS, dic. de 2017. ISBN: 9791196230715. URL: http://community.robotsource.org/t/download-the-ros-robot-programming-book-for-free/51.

[53] O'Kane Jason M. *A Gentle Introduction to ROS*. CreateSpace Independent Publishing Platform, oct. de 2013. ISBN: 9781492143239. URL: http://www.cse.sc.edu/~jokane/agitr/.

[54] W.S. Newman. *A systematic approach to learning robot programming with ROS*. 1.ª ed. 2017. ISBN: 978-1-49877-782-7. DOI: 10.1201/9781315152691.

[55] Enrique Fernández y col. *Effective Robotics Programming with ROS*. 3.ª ed. Packt Publishing Ltd, 2016. ISBN: 978-1-78646-365-4.

[56] Lentin Joseph y Jonathan Cacace. *Mastering ROS for Robotics Programming*. 2.ª ed. Packt Publishing Ltd, 2018. ISBN: 978-1-78847-895-3.

[57] Geiger Andreas y col. «Vision meets Robotics: The KITTI Dataset». En: *International Journal of Robotics Research* 32 (sep. de 2013), págs. 1231-1237. DOI: 10.1177/0278364913491297.

[58] Kamel Mina y col. «Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System». En: *Robot Operating System (ROS) The Complete Reference, Volume 2* (mayo de 2017), págs. 3-39. DOI: 10.1007/978-3-319-54927-9_1.

[59] Aström Karl J. y Wittenmark Björn. *Computer-controlled systems, theory and design*. 3.ª ed. Prentice Hall, 1997, pág. 557. ISBN: 0-13-314899-8.

[60] Mattingley J. y Boyd S. «CVXGEN: A code generator for embedded convex optimization». En: *Optimization and Engineering* 13 (mar. de 2012), págs. 1-27. DOI: 10.1007/s11081-011-9176-9.

[61] Sa Inkyu y col. «Dynamic System Identification, and Control for a cost effective open-source VTOL MAV». En: (ene. de 2017). arXiv: 1701.08623 [cs.RO].

Anexos

En este apartado se incluyen artículos publicados relacionados con el trabajo de tesis, así como las diferentes actividades realizadas a lo largo de este periodo.

Primeramente se muestra el artículo titulado *Model-based predictive control for trajectory tracking of a quadrotor* el cual fue incluido dentro del número especial 2020 de la revista digital "Memorias del Congreso Nacional de Control Automático" con ISSN: 2594-2492 publicada el día 18 de noviembre del 2020. Esta revista se encuentra disponible en línea[1] en el sitio web de la AMCA.

De igual forma se anexa el artículo *Seguimiento de trayectorias de un robot móvil diferencial a través del sistema operativo robótico ROS* publicado en el libro "Desarrollos con Enfoque Mecatrónico 2020" con ISBN: 978-607-9394-22-6. Este libro se puede descargar desde su sitio web[2], el cual incluye los trabajos presentados en el 19° Congreso Nacional de Mecatrónica que se llevó a cabo del 3 al 4 de diciembre del 2020 en formato virtual.

Adicionalmente se adjunta la constancia de participación en el curso *Reconstrucción 3D y localización con sensores RGB-D* impartido por el Dr. Fernando Israel Ireta Muñoz del 21 de septiembre al 30 de octubre del 2020 con una duración de 25 horas en formato virtual.

Finalmente se integra el certificado que avala el dominio del idioma Inglés.

[1] http://www.amca.mx/RevistaDigital/cnca2020/Index.htm

[2] http://www.mecamex.net/Libros/2020-Libro-DesarrolloconEnfoqueMecatronico.pdf

Model-based predictive control for trajectory tracking of a quadrotor

E. Salazar-Hidalgo * **J. Castañeda-Camacho** *
C. Martínez-Torres ** **J. Martínez-Carranza** ***

* *Benemérita Universidad Autónoma de Puebla, Facultad de Ciencias de la Electrónica, Puebla, México.*
** *Universidad de las Américas Puebla, Puebla, México*
*** *Instituto Nacional de Astrofísica, Óptica y Electrónica, Puebla, México.*

Abstract: This paper presents the design of a model-based predictive controller (MPC) for trajectory tracking of a quadrotor. The dynamic model is obtained using the Euler-Lagrange equations of motion; the resulting model is linearized around the hovering condition. The optimal control sequence is applied by following the receding horizon principle. The optimization problem is based on a quadratic cost function, which incorporates the state errors and the vector of control inputs. This process is repeated at each sampling instant, considering the plant model as well as possible constraints. Numerical simulations validate the controller's performance, which makes the vehicle follow the desired trajectory precisely.

Keywords: Euler-Lagrange, linearization, predictive control, MPC, trajectory tracking

1. INTRODUCCIÓN

Recientes avances en la tecnología de microprocesadores, miniaturización de sensores, así como un incremento en la capacidad de almacenamiento en las baterías, han hecho de los vehículos aéreos no tripulados una realidad. Específicamente los cuadricópteros han cobrado interés debido a sus particulares características que lo diferencian de otros tipos de vehículos aéreos, tales como: su estructura simple, la habilidad de mantenerse en vuelo estacionario y la capacidad de despegar y aterrizar de forma vertical. Un cuadricóptero como su nombre lo indica es un vehículo con cuatro rotores los cuales se encuentran distribuidos de manera estratégica para formar una estructura en forma de cruz.

Este tipo de plataformas han cobrado especial interés en el área de control automático debido a la naturaleza no lineal, multivariable y subactuada del modelo dinámico. Se han propuesto una gran variedad de estrategias de control para este tipo de plataformas tales como controladores PID: Abdulsalam (2019); Backstepping: Glida (2020); LQR: Kuantama et al. (2018), Kurak (2018); PD: Luukkonen (2011); par calculado: Balasubramanian (2013); entre otros.

En la ultima década ha habido un incremento en el diseño de controladores predictivos para vehículos aéreos, especialmente controladores MPC. Este tipo de controladores tienen la habilidad de incluir restricciones en el proceso de optimización, y con ello obtener una secuencia de control

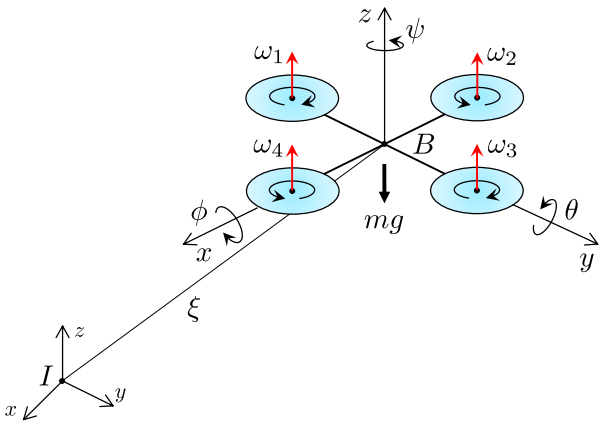


Figura 1. Diagrama de cuerpo libre de un cuadricóptero.

capaz de cumplir con el objetivo de control, Camacho (2007). Comparado con otro tipo de controladores la sintonización resulta un tarea fácil incluso para sistemas de múltiples entradas y múltiples salidas.

El problema del seguimiento de trayectorias empleando controladores MPC ya ha sido estudiado por varios autores. Por ejemplo: En Lopes et al. (2012) se emplea únicamente un controlador MPC para la posición y orientación. En Chen (2013) y Eskandarpour (2018) se emplean controladores MPC en cascada que utilizan funciones ortonormales para el cálculo de la secuencia de control, esta estructura divide la dinámica del cuadricóptero en dos subsistemas: traslación y rotación. Incluso exis-

ten implementaciones de este tipo de controladores en plataformas reales: Kamel (2017), registrando un buen desempeño.

En este trabajo se diseña un control MPC para el seguimiento de trayectorias de un cuadricóptero similar al de los trabajos anteriores, pero con la particularidad de dividir la estructura de control en 3 subsistemas: Altitud, Rotación y Traslación, cada uno con su propio controlador MPC propiamente sintonizado. Por simplicidad, las restricciones no son consideradas.

El resto del trabajo se encuentra organizado de la siguiente manera: En la Sección 2 se presenta el modelo dinámico utilizado y la linealización del mismo. En la Sección 3 se presenta la estructura de control propuesta. El diseño del control MPC se incluye en la Sección 4. Los resultados de las simulaciones numéricas aparecen en la Sección 5. Finalmente, las conclusiones se integran en la Sección 6.

2. MODELO DINÁMICO DE UN CUADRICÓPTERO

Para este trabajo se ha decidido seguir el formalismo de Euler-Lagrange: García et al. (2013), Luukkonen (2011) debido a que este método describe de manera explícita el balance de energías de un cuerpo en movimiento.

La pose relativa del sistema de referencia del vehículo $\{B\}$ con respecto al sistema de referencia inercial $\{I\}$ se encuentra definida por el vector de posición $\boldsymbol{\xi} \in \mathbb{R}^3$ y por el vector de orientación $\boldsymbol{\eta} \in \mathbb{R}^3$ (Figura 1).

$$\boldsymbol{\xi} = [x \ y \ z]^T, \qquad \boldsymbol{\eta} = [\phi \ \theta \ \psi]^T$$

2.1 Ecuaciones de movimiento traslacional

Las ecuaciones que describen la dinámica traslacional de un cuadricóptero están dadas por: García et al. (2013).

$$m\ddot{\boldsymbol{\xi}} + mg = \boldsymbol{F}_\xi$$

donde m representa la masa del vehículo y g la constante de aceleración gravitacional. $\boldsymbol{F}_\xi \in \mathbb{R}^3$ representa la fuerza aplicada al cuadricóptero generada por el empuje total de los rotores.

$$\boldsymbol{F}_\xi = \begin{bmatrix} C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\phi S_\theta S_\psi - S_\phi C_\psi \\ C_\phi C_\theta \end{bmatrix} F$$

Aquí, $F = \sum_{i=1}^{4} f_i$, donde $f_i = k_T \omega_i^2$. $k_T > 0$ es la constante de empuje y ω_i es la velocidad angular del i-ésimo rotor.

2.2 Ecuaciones de movimiento rotacional

Así mismo la dinámica rotacional del cuadricóptero esta descrita por la siguiente expresión: García et al. (2013).

$$M(\boldsymbol{\eta})\ddot{\boldsymbol{\eta}} + C(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})\dot{\boldsymbol{\eta}} = \boldsymbol{\tau}$$

donde $M(\boldsymbol{\eta}) \in \mathbb{R}^{3\times3}$ se conoce como la matriz de inercia la cual es definida positiva y por lo tanto simétrica. $C(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) \in \mathbb{R}^{3\times3}$ es la matriz de fuerzas centrípetas y de Coriolis. $\boldsymbol{\tau}$ es el vector de pares externos aplicados.

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} (f_3 - f_1)l \\ (f_2 - f_4)l \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix}$$

l es la distancia entre el eje de los motores y el centro de masa del vehículo. $M_i = k_D \omega_i^2$, donde k_D es conocido como coeficiente de arrastre.

2.3 Linealización

Este proceso se lleva a cabo considerando la situación en la que el vehículo se encuentra en vuelo estacionario, en donde se satisfacen las siguientes condiciones de operación:

$$\boldsymbol{\xi} = \boldsymbol{\xi}_0 \qquad \phi = 0 \qquad \theta = 0 \qquad \psi = \psi_d$$

Donde $\boldsymbol{\xi}_0$ es un vector de posición constante y ψ_d el ángulo en *yaw* constante deseado. Empleando la aproximación para ángulos pequeños,

$$\text{sen} \approx \phi \approx 0, \quad \text{sen} \approx \theta \approx 0, \quad \cos\phi \approx 1, \quad \cos\theta \approx 1$$

la matriz de inercia $M(\boldsymbol{\eta})$ se reduce a una matriz diagonal y la matriz de fuerzas centrípetas y de Coriolis $C(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}})$ se vuelve cero. Con lo anterior es posible obtener el modelo dinámico linealizado.

$$\begin{aligned} \ddot{x} &= g\theta & \ddot{\phi} &= \tau_\phi / I_{xx} \\ \ddot{y} &= -g\phi & \ddot{\theta} &= \tau_\theta / I_{yy} \\ \ddot{z} &= F/m - g & \ddot{\psi} &= \tau_\psi / I_{zz} \end{aligned} \tag{1}$$

donde I_{xx}, I_{yy} e I_{yy} son los momentos de inercia.

3. DISEÑO DEL CONTROLADOR

La estrategia de control utilizada se ilustra en la Figura 2. El controlador de altitud genera el empuje necesario para mantener al vehículo a una altura determinada. El controlador de traslación controla las coordenadas x e y generando las señales de referencia ϕ_d y θ_d que ingresan directamente al controlador de orientación. Estas señales en conjunto con ψ_d ingresan a dicho controlador el cual genera los pares τ_ϕ, τ_θ y τ_ψ los cuales se aplican directamente a la planta.

3.1 Controlador de Altitud

Se toma la tercera ecuación de (1) y se hace la representación en variables de estado.

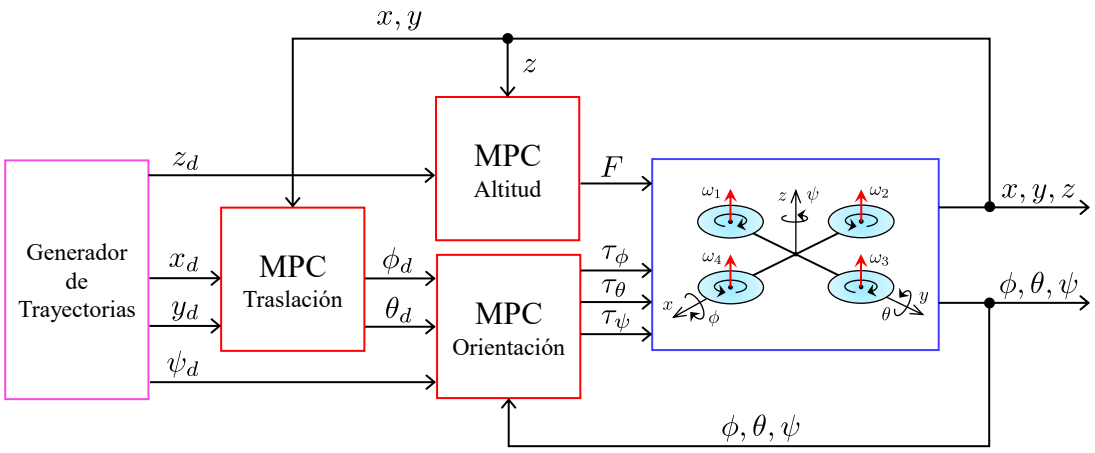


Figura 2. Estructura de control MPC.

$$\frac{d}{dt}\begin{bmatrix} z \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} z \\ \dot{z} \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} G \tag{2}$$

donde $G = F - mg$. La salida del subsistema será la altura del vehículo z.

3.2 Controlador de Traslación

De forma similar se toman las dos primeras ecuaciones de (1) y se efectúa la representación en el espacio de estados.

$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ g & 0 \\ 0 & 0 \\ 0 & -g \end{bmatrix}\begin{bmatrix} \theta_d \\ \phi_d \end{bmatrix} \tag{3}$$

Este subsistema tiene como entradas los ángulos θ_d y ϕ_d, y como salidas las posiciones en x e y.

3.3 Controlador de Orientación

Se toman las últimas tres ecuaciones de (1) para efectuar la representación en el espacio de estados tal como se muestra a continuación:

$$\frac{d}{dt}\begin{bmatrix} \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \\ \psi \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \\ \psi \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1/I_{xx} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1/I_{yy} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1/I_{zz} \end{bmatrix}\begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{4}$$

Como entrada se tiene a $\boldsymbol{\tau} = [\tau_\phi \ \tau_\theta \ \tau_\psi]^T$ y como salida a $\boldsymbol{\eta} = [\phi \ \theta \ \psi]^T$.

4. CONTROL PREDICTIVO DE MODELO

El término Control Predictivo no designa a una estrategia de control particular sino a un conjunto de métodos de control que hacen uso explícito de un modelo de la planta para obtener la señal de control minimizando una función de costo, Camacho (2007). La minimización de la función de costo puede obtenerse de manera analítica si no existen restricciones, de otra forma es necesaria la implementación de algún método numérico.

4.1 Modelo aumentado en el espacio de estados

El modelo en el espacio de estados puede ser usado tanto en el caso monovariable así como en el caso multivariable y se puede extender al caso no lineal, Camacho (2007). Considere que la planta cuenta con m entradas, q salidas y s estados.

$$\begin{aligned} \boldsymbol{x}_m(k+1) &= A_m\boldsymbol{x}_m(k) + B_m\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= C_m\boldsymbol{x}_m(k) \end{aligned}$$

Donde $A_m \in \mathbb{R}^{s\times s}$, $B_m \in \mathbb{R}^{s\times m}$, $C_m \in \mathbb{R}^{q\times s}$. $\boldsymbol{x} \in \mathbb{R}^n$ representa el vector de estados, $\boldsymbol{u} \in \mathbb{R}^m$ el vector de entradas y $\boldsymbol{y} \in \mathbb{R}^q$ el vector de salidas. Se definen la variación del vector de estados y la variación del vector de entradas de control de la siguiente manera:

$$\begin{aligned} \Delta\boldsymbol{x}_m(k+1) &= \boldsymbol{x}_m(k+1) - \boldsymbol{x}_m(k) \\ \Delta\boldsymbol{u}(k) &= \boldsymbol{u}(k) - \boldsymbol{u}(k-1) \end{aligned}$$

Para dar lugar a la versión aumentada del modelo original, Wang (2009).

$$\begin{aligned} \begin{bmatrix} \Delta\boldsymbol{x}_m(k+1) \\ \boldsymbol{y}(k+1) \end{bmatrix} &= \begin{bmatrix} A_m & 0_m^T \\ C_mA_m & I_{q\times q} \end{bmatrix}\begin{bmatrix} \Delta\boldsymbol{x}_m(k) \\ \boldsymbol{y}(k) \end{bmatrix} \\ &\quad + \begin{bmatrix} B_m \\ C_mB_m \end{bmatrix}\Delta\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= [0_m \ I_{q\times q}]\begin{bmatrix} \Delta\boldsymbol{x}_m(k) \\ \boldsymbol{y}(k) \end{bmatrix} \end{aligned} \tag{5}$$

Donde $I_{q\times q} \in \mathbb{R}^{q\times q}$ representa a la matriz identidad y $0_m \in \mathbb{R}^{s\times q}$ es una matriz cero. Por simplicidad el modelo aterior suele representarse de la siguiente manera:

$$\begin{aligned} \boldsymbol{x}(k+1) &= A\boldsymbol{x}(k) + B\Delta\boldsymbol{u}(k) \\ \boldsymbol{y}(k) &= C\boldsymbol{x}(k) \end{aligned} \tag{6}$$

Las dimensiones del modelo aumentado están determinadas por $n = s + q$.

4.2 Modelo de predicción

En base al modelo en el espacio de estados (6), las variables de salida futuras se calculan mediante la siguiente expresión, Camacho (2007):

$$\hat{\boldsymbol{y}}(k+j|k) = CA^j\boldsymbol{x}(k) + \sum_{i=0}^{j-1} CA^{j-i-1}B\Delta\boldsymbol{u}(k+i)$$

para $j = 1, \ldots, N_p$, donde N_p representa el horizonte de predicción. Además $\hat{\boldsymbol{y}}(k+j|k)$ indica la salida predicha en $k+j$ a partir de la información proporcionada por la planta en el instante k. De la expresión anterior se obtiene el modelo de predicción.

$$\hat{\boldsymbol{Y}} = F\boldsymbol{x}(k) + \Phi\Delta\boldsymbol{U} \tag{7}$$

donde,

$$F = \begin{bmatrix} CA & CA^2 & CA^3 & \cdots & CA^{N_p} \end{bmatrix}^T$$

$$\Phi = \begin{bmatrix} CB & 0 & 0 & \ldots & 0 \\ CAB & CB & 0 & \ldots & 0 \\ CA^2B & CAB & CB & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ CA^{N_p-1}B & CA^{N_p-2}B & CA^{N_p-3}B & \ldots & CA^{N_p-N_c}B \end{bmatrix}$$

Cuyas dimensiones son: $F \in \mathbb{R}^{qN_p\times n}$ y $\Phi \in \mathbb{R}^{qN_p\times mN_c}$. Además N_c representa el horizonte de control, generalmente se elige $N_c \leq N_p$.

4.3 Ley de control

El objetivo del MPC consiste en llevar la señal de salida predicha los mas cercana posible al valor de referencia la cual se considera constante dentro de la ventana de optimización. Lo anterior se puede traducir como la búsqueda de la mejor secuencia de control $\Delta\boldsymbol{u}$ de tal forma que una función del error entre el valor de referencia y el valor predicho sea minimizada. Considere el siguiente vector el cual contiene los valores de referencia deseados.

$$\boldsymbol{R}_s^T = \underbrace{[1\ 1\ \ldots\ 1]}_{N_p} r(k)$$

Se define la siguiente función de costo:

$$J = (\boldsymbol{R}_s - \hat{\boldsymbol{Y}})^T\bar{Q}(\boldsymbol{R}_s - \hat{\boldsymbol{Y}}) + \Delta\boldsymbol{U}^T\bar{R}\Delta\boldsymbol{U} \tag{8}$$

donde el primer término está relacionado con los errores entre el valor de referencia y la salida predicha, mientras que el segundo término refleja el peso dado al tamaño de $\Delta\boldsymbol{U}$ cuando la función de costo J es minimizada. Además,

$$\bar{Q} = \begin{bmatrix} Q & 0 & \ldots & 0 \\ 0 & Q & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & Q \end{bmatrix}, \qquad \bar{R} = \begin{bmatrix} R & 0 & \ldots & 0 \\ 0 & R & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & R \end{bmatrix}$$

Aquí $\bar{Q} \in \mathbb{R}^{qN_p\times qN_p}$ y $\bar{R} \in \mathbb{R}^{mN_c\times mN_c}$. $Q \in \mathbb{R}^{q\times q}$ y $R \in \mathbb{R}^{m\times m}$ son matrices diagonales, las cuales son usadas como parámetros de sintonización. Sustituyendo el modelo de predicción (7) dentro de la función de costo (8), derivando e igualando a cero se obtiene la secuencia de control óptima.

$$\Delta\boldsymbol{U} = \left[\Phi^T\bar{Q}\Phi + \bar{R}\right]^{-1}\Phi^T\bar{Q}\left[\boldsymbol{R}_s - F\boldsymbol{x}(k)\right]$$

Considerando que

$$\boldsymbol{R}_s = \underbrace{[1\ 1\ \ldots\ 1]}_{N_p} r(k) = \bar{\boldsymbol{R}}_s r(k)$$

se tiene

$$\Delta\boldsymbol{U} = \left[\Phi^T\bar{Q}\Phi + \bar{R}\right]^{-1}\Phi^T\bar{Q}\left[\bar{\boldsymbol{R}}_s r(k) - F\boldsymbol{x}(k)\right] \tag{9}$$

Empleando el principio del control de horizonte deslizante, únicamente los primeros m elementos del vector $\Delta\boldsymbol{U}$ son aplicados realmente a la planta, dicho procedimiento se repite en cada instante de muestreo.

5. SIMULACIONES NUMÉRICAS

El proceso consiste en obtener la versión aumentada de cada subsistema utilizando (5). A continuación el modelo de predicción se obtiene a partir de (7). Finalmente la ley de control se calcula con ayuda de (9). La simulación se lleva a cabo contemplando los parámetros del cuadricóptero que aparecen en la Tabla 1, los cuales son tomados de Silano et al. (2019).

Para obtener la versión aumentada de cada subsistema es necesario llevar a cabo la discretización de cada modelo. En este caso por simplicidad, se emplea el método de Euler.

$$A_d = I + A_cT_s, \qquad B_d = B_cT_s, \qquad C_d = C_c$$

Aquí T_s representa el periodo de muestreo, $I \in \mathbb{R}^{s\times s}$ la matriz identidad cuyas dimensiones dependen del número

Tabla 1. Parámetros del sistema

Parámetro	Valor	Unidades
m	0.5	kg
g	9.81	m/s^2
l	0.12905	m
I_{xx}	0.00389	$kg \cdot m^2$
I_{yy}	0.00389	$kg \cdot m^2$
I_{zz}	0.00780	$kg \cdot m^2$
k_T	8.54858×10^{-6}	$kg \cdot m$
k_D	0.016	m

de estados de cada subsistema. Los parámetros de sintonización para cada controlador se muestran en la Tabla 2 los cuales se obtuvieron de forma heurística.

Tabla 2. Parámetros MPC

Parámetro	Altitud	Posición	Orientación
T_s	0.05	0.05	0.005
N_p	10	10	12
N_c	10	10	12
Q	diag(20)	diag(20)	diag(25)
R	diag(0.01)	diag(0.01)	diag(0.1)

La trayectoria de referencia consiste en una espiral, la cual es descrita por las siguientes ecuaciones:

$$x_d = 2\cos(0.2t), \quad y_d = 2\,\mathrm{sen}(0.2t), \quad z_d = 0.5t, \quad \psi_d = \pi$$

Los resultados obtenidos se presentan a continuación: En la Figura 3 se ilustra la trayectoria trazada por el vehículo en el espacio tridimensional.

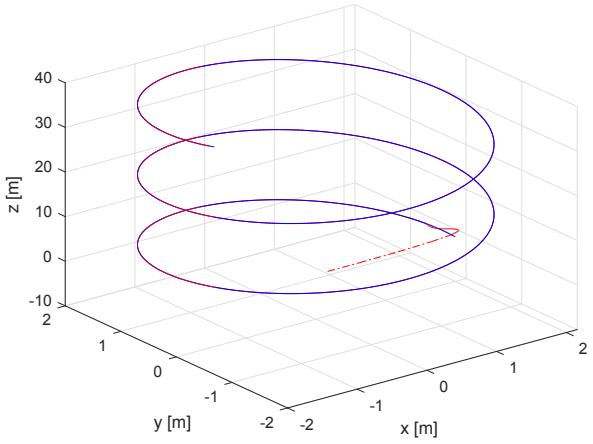


Figura 3. Trayectoria del vehículo en el espacio tridimensional.

Como se puede apreciar en la Figura 4 y Figura 5 a medida que el tiempo evoluciona los estados convergen a hacia los valores de referencia.

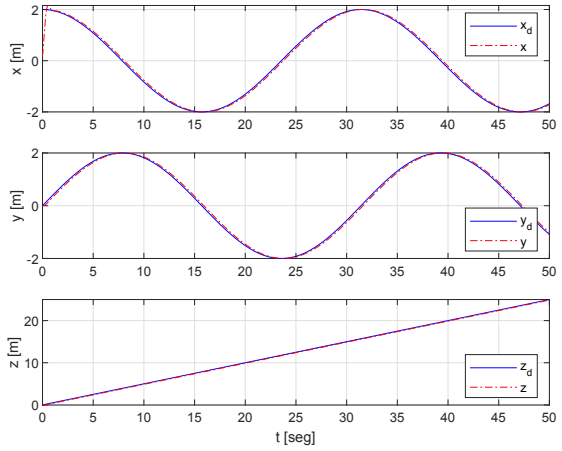


Figura 4. Variables de estado de la posición.

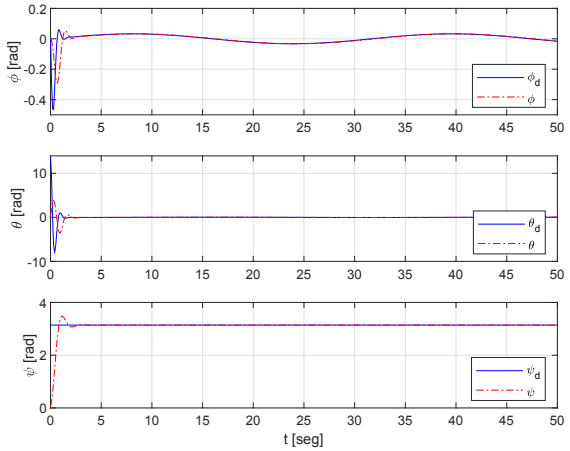


Figura 5. Variables de estado de la orientación.

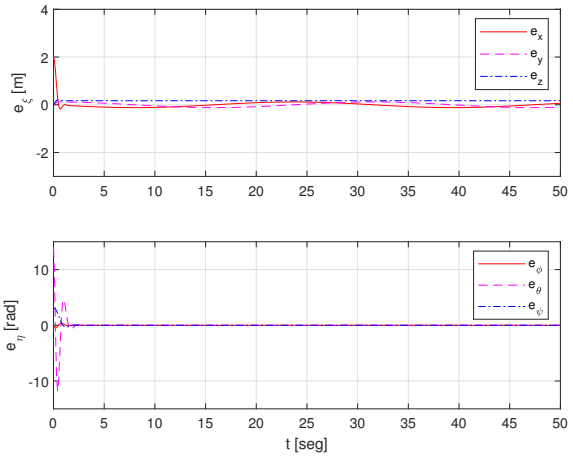


Figura 6. Errores de estado.

En la Figura 6 aparecen los errores de posición y orientación. Para el caso de la posición, los errores se mantienen oscilando alrededor de cero, dichos valores nunca llegan a ser cero, sin embargo, la magnitud de los mismos se encuentra acotada. Los errores de orientación, por otra parte tienden asintóticamente a cero a medida que el tiempo tiende a infinito. Lo anterior garantiza que la trayectoria deseada sea trazada de forma precisa. Empleando la siguiente expresión es posible calcular el indice de desempeño de cada controlador.

$$\mathcal{L}_2[\boldsymbol{e}(t)] = \sqrt{\frac{1}{T_s}\int_0^{Ts} \left\|\boldsymbol{e}(t)\right\|^2 dt}$$

Considerando todo el tiempo de simulación se obtuvieron los siguientes valores numéricos para cada controlador: altitud: `0.1698`, traslación: `0.1485`, y orientación: `0.8094`. Lo anterior representa un buen comportamiento debido a que valores lo mas cercanos a cero se traducen en un buen desempeño. Siendo el controlador de orientación el que presenta un desempeño menor en comparación con los demás controladores.

Por otro lado, en la Figura 7 se muestran las velocidades angulares elevadas al cuadrado de cada rotor. En el estado transitorio los cuatro rotores alcanzan su velocidad máxima debido a que el vehículo se encuentra lejos de la trayectoria de referencia (Figura 3), por lo tanto, se necesita un mayor esfuerzo de control para conducir el cuadricóptero hacia la referencia deseada.

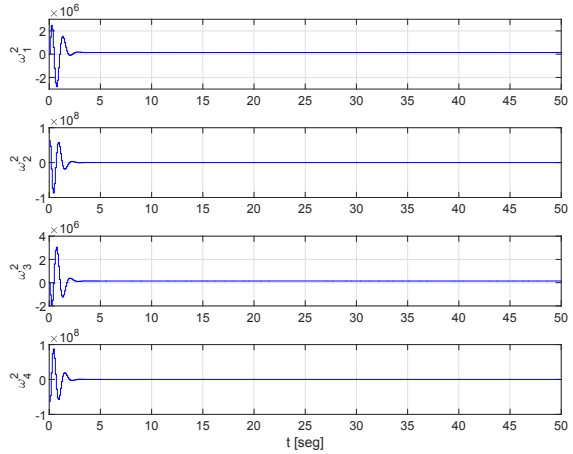


Figura 7. Velocidades angulares de los rotores.

6. CONCLUSIONES

En este trabajo se llevó a cabo la descripción y simulación de una estrategia de control MPC para resolver el problema de seguimiento de trayectorias de un cuadricóptero. La estructura empleada tiene la particularidad de dividir al modelo dinámico en tres subsistemas: altitud, posición y orientación, lo cual reduce la complejidad del análisis así como el diseño de la ley de control. Dentro de la función de costo se considera el vector de errores así como el vector de entradas de control. En este caso, por simplicidad las restricciones no fueron consideradas. Como las gráficas lo demuestran la estructura de control propuesta efectúa el seguimiento de trayectorias de forma satisfactoria.

REFERENCIAS

E. Kuantama, I. Tarca, y R. Tarca. *Feedback Linearization LQR Control for Quadcopter Position Tracking*. En: 2018 5th International Conference on Control, Decision and Information Technologies. págs. 204-209, Abr. 2018.

H.E. Glida y Col. *Optimal model-free backstepping control for a quadrotor helicopter*. En: Nonlinear Dynamics. págs. 3449–3468, May. 2020.

E. Balasubramanian y R. Vasantharaj. *Dynamic modeling and control of quad rotor*. En: International Journal of Engineering and Technology (IJET). págs. 63–69, Feb.-Mar. 2013.

S. Kurak y M. Hodzic. *Control and estimation of a quadcopter dynamical model*. En: Periodicals of Engineering and Natural Sciences. págs. 63–75, Mar. 2018.

A. Eskandarpour e I. Sharf. *A constrained error-based MPC for path following of quadrotor with stability analysis*. En: Nonlinear Dynamics. págs. 899-–918, Abr. 2019.

R.V. Lopes, P.H. De Rodrigues, G.A. Borges y J.Y. Ishihara. *Model predictive control applied to tracking and attitude stabilization of a VTOL quadrotor aircraft*. En: ABCM Symposium Series in Mechatronics. págs. 176–185, 2012.

X. Chen y L. Wang. *Cascaded model predictive control of a quadrotor UAV*. En: 2013 3rd Australian Control Conference. págs. 354–359, Nov. 2013.

E. Camacho y C. Bordons. *Model predictive control*. Springer-Verlag London, 2nd ed., 2007.

L. Wang. *Model predictive control system design and implementation using Matlab*. Springer-Verlag London, 2009.

L.R. García, A.E. Dzul, R. Lozano y C. Pégard. *Modeling the quad-rotor mini-rotorcraft*. En: Quad Rotorcraft Control. págs. 23–34, Ene. 2013.

T. Luukkonen. *Modelling and control of quadcopter*. Espoo, Finlandia, 2011.

A. Abdulsalam y I. Kasim. *Nonlinear PID controller design for a 6-DOF UAV quadrotor system*. En: Engineering Science and Technology, an International Journal. págs. 1087–1097, Ago. 2019.

G. Silano, P. Oppido y L. Iannelli. *Software-in-the-loop simulation for improving flight control system design: a quadrotor case study*. En: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC). págs. 466–471, Oct. 2019.

M. Kamel y Col. *Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles*. En: IFAC 2017. págs. 3463-3469, Jul. 2017.

ASOCIACIÓN DE MÉXICO DE CONTROL AUTOMÁTICO
MEMORIAS DEL CONGRESO NACIONAL DE CONTROL AUTOMÁTICO 2020

Se otorga el presente

Certificado

a

Eduardo Salazar Hidalgo
Josefina Castañeda Camacho
César Martínez Torres
José Martínez Carranza

Por su valiosa contribución en el número especial de las **Memorias del Congreso Nacional de Control Automático 2020 (ISSN: 2594-2492)** con el artículo

"*Model-based predictive control for trajectory tracking of a quadrotor*".

Dra. Flor Lizeth Torres Ortiz
COORDINADORA EDITORIAL

Desarrollos con enfoque mecatrónico, Capítulo 3, pp. 27 - 38
ISBN: 978-607-9394-22-6, año 2020

Seguimiento de trayectorias de un robot móvil diferencial a través del sistema operativo robótico ROS

Salazar-Hidalgo Eduardo[1], Castañeda-Camacho Josefina[1], Martínez-Torres Cesar[2], Martínez-Carranza José[3]

[1]Benemérita Universidad Autónoma de Puebla
[2]Universidad de las Américas Puebla
[3]Instituto Nacional de Astrofísica, Óptica y Electrónica

Contacto: eduardo.salazarh@alumno.buap.mx

Resumen

En el presente trabajo se presenta el diseño de un sistema de control lineal para el seguimiento de trayectorias de un robot móvil diferencial haciendo uso del sistema operativo robótico ROS en este caso se trabajó con la versión Kinetic Kame. Básicamente el procedimiento se divide en tres etapas: La primera de ellas consiste en la generación de una secuencia de estados deseados, los cuales permitan guiar al robot hacia un punto arbitrario, tomando en consideración restricciones dinámicas y físicas del vehículo. La segunda etapa se encarga de estimar los estados actuales del robot con la ayuda de los sensores a bordo. Por último, en la etapa de control se evalúan los estados actuales del vehículo con respecto a los estados deseados y en base al error se toman decisiones para llevar a cabo acciones correctivas. El controlador propuesto consiste en un control lineal que hace uso del modelo dinámico para calcular las velocidades lineal y angular las cuales se consideran como entradas de control. Las pruebas experimentales se llevan a cabo sobre el robot móvil diferencial Turtlebot3 Waffle PI.

Palabras clave: robot móvil diferencial, control lineal, seguimiento de trayectorias, ROS, Gazebo, Turtlebot3 Waffle PI

1. Introducción

En los últimos años ha habido un aumento considerable en los trabajos publicados relacionados con robots móviles. Los robots móviles han sido exitosamente empleados en la industria, como robots de servicio, en tareas domésticas, exploración de áreas de difícil acceso o peligrosas para el ser humano, educación, así como en el sector del entretenimiento [1], [2], [3]. Especialmente los robots móviles diferenciales despiertan el interés de la comunidad científica debido a su estructura simple. La estructura de este tipo de vehículos consta de un par de ruedas actuadas fijadas al chasis principal, en combinación con una o dos ruedas pasivas.

Uno de los objetivos de la robótica consiste en dotar a robots de habilidades para la ejecución de tareas de manera autónoma. Actualmente existe un enfoque que busca crear robots capaces de efectuar navegación autónoma. Ésta es un área extensa y generalmente se divide en dos partes: Planeación de trayectorias y control. La etapa de planeación se enfoca en la generación de una trayectoria libre de colisiones para conducir al robot de un punto A hacia un punto B. El algoritmo de control es responsable de mantener al robot lo más cercano posible a la trayectoria previamente generada al calcular la velocidad lineal y velocidad angular adecuadas. El algoritmo de control depende directamente de los errores de estado por lo que resulta crucial conocer los estados actuales del robot, para ello existen diferentes métodos de localización tales como: odometría, SLAM o de sistemas de captura de movimiento.

Como se mencionó anteriormente la navegación autónoma depende de un conjunto de subsistemas los cuales deben trabajar de manera coordinada para lograr un objetivo común. Por tal motivo es necesaria una plataforma lo suficientemente robusta para el desarrollo e implementación de dichos algoritmos. ROS es un entorno consolidado en donde resulta relativamente sencillo implementar algoritmos para la generación de trayectorias, estimación de estados y control. Además, este sistema cuenta con robustos simuladores que permiten evaluar dichos algoritmos sin la necesidad de contar con una plataforma real. Recientemente ha habido trabajos en los que se hace uso de este sistema operativo para llevar a cabo el control de robots móviles diferenciales [4], [5].

El resto del trabajo se organiza de la siguiente manera: En la sección 2 se presenta el modelo dinámico del robot el cual está basado en el modelo cinemático. La plataforma de experimentación se describe detalladamente en la sección 3. La sección 4 describe el procedimiento llevado a cabo para el diseño del sistema de control. Los resultados experimentales obtenidos se presentan en la sección 5. Finalmente, las conclusiones se añaden en la sección 6.

2. Modelo dinámico de un robot móvil diferencial

Existen distintas formulaciones para la obtención del modelo dinámico de un robot móvil diferencial, fundamentalmente se pueden encontrar las que emplean las ecuaciones de movimiento de Euler-Lagrange (balance de energías) [6], [7] y las que parten del modelo cinemático para obtener modelos dinámicos basados en errores [1], [8], [9].

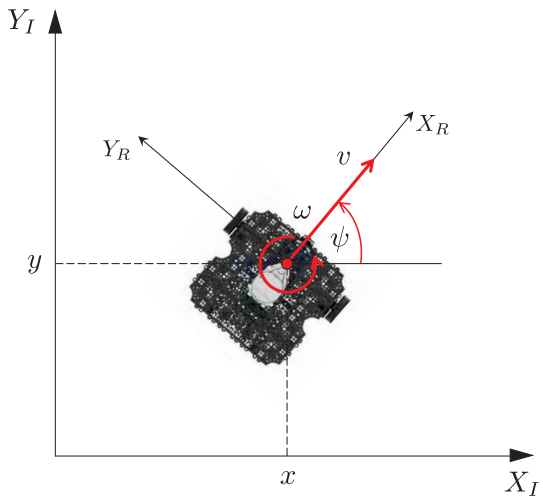


Figura 1. Sistemas de referencia global y local de un robot móvil diferencial.

La pose del robot consta de dos componentes para la posición (x, y), y una componente para la orientación en ψ la cual indica el giro del robot alrededor del eje z.

$$\boldsymbol{q} = \begin{bmatrix} x \\ y \\ \psi \end{bmatrix} \tag{1}$$

Estas variables se encuentran definidas con respecto al sistema de coordenadas global. Para el caso de la orientación, el ángulo del vehículo resulta de la desviación existente entre el eje X_R del sistema de coordenadas del robot con respecto al eje X_I del sistema de coordenadas global, tomando como ángulos positivos los que siguen el sentido contrario del giro de las manecillas del reloj. El modelo cinemático de un robot con restricciones holonómicas como el de la Figura 1 está representado por las siguientes ecuaciones diferenciales [1], [8], [9]:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} cos\,\psi & 0 \\ sin\,\psi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (2)$$

Donde v y ω representan a la velocidad lineal y velocidad angular respectivamente, consideradas como entradas de control.

El problema del seguimiento de trayectorias consiste en hacer que el robot siga una trayectoria de referencia, la cual generalmente es conocida de antemano. La trayectoria debe ser generada considerando límites en velocidades y aceleraciones, así como las restricciones físicas del robot y del entorno. Es común que la etapa de generación de trayectorias sea efectuada de forma independiente por otro algoritmo o en tiempo real para un robot autónomo.

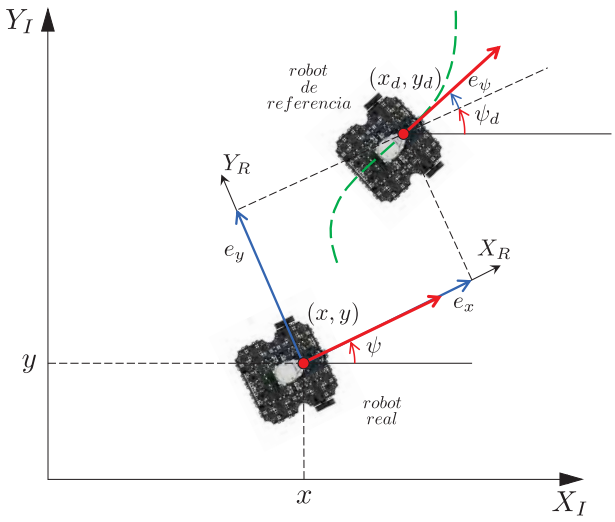


Figura 2. Seguimiento de trayectorias.

Básicamente el robot debe seguir la pose de referencia tal como se muestra en la Figura 2. El control debe ser capaz de mantener al robot lo más cercano posible a la referencia haciendo que los errores de estado tiendan a cero cuando el tiempo tienda a infinito.

3. Plataforma de experimentación

3.1 ROS

ROS (Robot Operating System) o sistema operativo robótico es un entorno para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo. ROS es considerado más bien como un meta-sistema operativo ya que es instalado sobre un sistema operativo ya existente. ROS se desarrolló originalmente en 2007 bajo el nombre de switchyard por el laboratorio de Inteligencia Artificial de Stanford. ROS provee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros.

ROS está orientado para el sistema operativo Ubuntu Linux, a pesar de que existen adaptaciones para otros S.O. no está garantizada su estabilidad. Debido a su éxito, distintas instituciones y compañías han comenzado a adaptar sus productos para ser usados en este entorno. Existe una larga lista de

robots compatibles, mismos que se pueden encontrar en [10]. En dicho sitio se pueden encontrar una gran variedad de robots los cuales se subdividen en 4 categorías principales: Aéreos, Terrestres, Marinos y Manipuladores.

Existen varias herramientas para facilitar la programación de robots en ROS entre ellas se encuentra *Gazebo* el cual es un simulador el cual permite programar robots como si estos se trataran de plataformas reales y *Rviz* en donde se pueden visualizar los datos del conjunto de sensores que incorpora un robot.

Recientemente se lanzó una nueva versión de ROS denominada ROS2 la cual resulta ser la versión mejorada de su predecesora con varios cambios significativos.

3.2 Turtlebot3 Waffle PI

TurtleBot3 es una plataforma de hardware estándar de ROS enfocado para su uso en las áreas de investigación y educación (Figura 3). Este robot fue concebido para facilitar la programación de robots. El éxito de esta plataforma se debe principalmente a su tamaño, y a su relativo bajo costo. El objetivo principal del proyecto TurtleBot3 consiste básicamente en reducir el tamaño y precio de la plataforma sin sacrificar la calidad y prestaciones que ofrecían sus predecesores TurtleBot1 y Turtlebot2. Actualmente existen tres modelos del TurtleBot3: Burger, Waffle (descontinuado) y Waffle PI. En este trabajo se hará uso del modelo Waffle PI por lo que a continuación se presenta una descripción de sus características principales [11], [13].

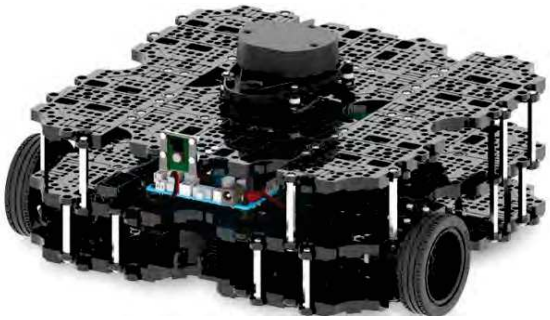


Figura 3. TurtleBot3 Waffle Pi (tomada de [11]).

3.2.1 Componentes

Las partes principales que integran al robot TurtleBot3 Waffle PI consisten en una Raspberry PI cuyo modelo puede ser el B ó B+. Este robot incorpora un par de sensores que le permiten ejecutar tareas de localización, mapeo y navegación. El primero se trata de un sensor de distancia laser LIDAR modelo LDS-01. El segundo sensor se trata de una cámara la cual se conecta directamente a la Raspberry PI. Además, este robot cuenta con la tarjeta open CR 1.0, la cual sirve como interfaz en el caso de que se deseara añadir más sensores al robot, incluso ésta misma tarjeta puede ser usada como un Arduino Uno debido a que cuenta con los pines de expansión característicos de dicha tarjeta. El movimiento del robot se lleva a cabo gracias a la incorporación de un par de motores Dynamixel modelo XM430.

3.2.2 Especificaciones

En la Tabla 1 se enlistan las características más importantes del robot Turtlebot3 Waffle PI, más detalles sobre el robot se pueden encontrar en su página oficial [11].

Tabla 1. Características del robot TurtleBot3 Waffle PI.

Características	Valor
Máxima velocidad traslacional	0.26 m/seg
Máxima velocidad rotacional	1.82 rad/seg
Máxima aceleración traslacional	2.5 m/seg^2
Máxima aceleración rotacional	3.2 rad/seg^2
Carga máxima	30 kg
Tiempo de operación	2 horas
Tiempo de recarga	2.5 horas
Alimentación	12V DC, 5 A

3.2.3 Estructura mecánica

La estructura principal del robot se compone de dos ruedas acopladas a la parte frontal del chasis principal y de un par de ruedas pasivas en la parte trasera. Por la configuración anterior, se considera que este modelo pertenece al grupo de robots móviles diferenciales. Las dimensiones del robot son de aproximadamente 30.6cm x 28.1cm x 14.1cm. El peso total del robot es de alrededor de 1.8 kg. Debido a la naturaleza *Open Source* del proyecto Turtlebot3, los diseños CAD de la estructura se encuentran disponibles, lo cual permite a los diseñadores acceder a dichos archivos para su descarga[1], incluso para hacer modificaciones propias.

3.2.4 Configuración del robot para su uso con ROS

Para efectuar este procedimiento, es necesario disponer de una PC remota desde la cual se efectué el control del robot. El proceso consiste en instalar una distribución de Linux en ambas computadoras, a continuación, instalar ROS y finalmente establecer comunicación entre ambos dispositivos. Para acceder a la computadora del robot es necesario el empleo de comandos *ssh* desde la PC remota. La configuración empleada se ilustra en la Figura 4.

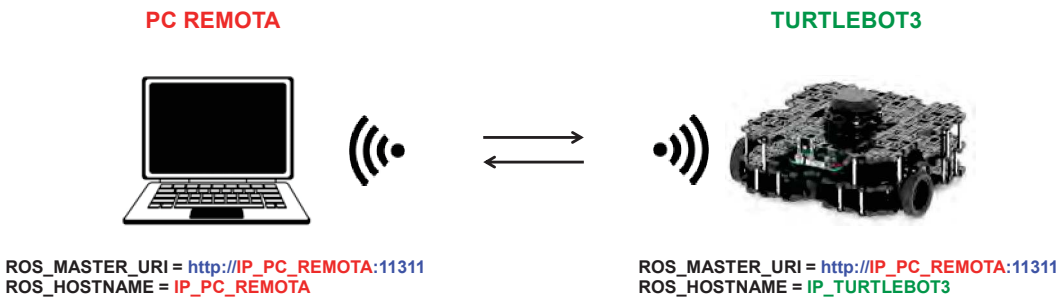


Figura 4. Configuración de red.

Una vez establecida la comunicación lo único que resta es ejecutar el *driver*[2] del robot (el conjunto de librerías encargadas de controlar los dispositivos de bajo nivel) desde la PC remota. Lo anterior abrirá paso a la transmisión de información sobre el estado de los sensores, de entre los más importantes se encuentran: el estado de la batería, la unidad de medición inercial IMU, el sensor laser,

[1] http://www.robotis.com/service/download.php?no=678
[2] https://github.com/ROBOTIS-GIT/turtlebot3.git

así como un sistema que se encarga de calcular la odometría del vehículo (posiciones y velocidades). El mismo *driver* es capaz de recibir comandos de velocidad lineal v y velocidad angular ω para el pilotaje del robot.

4. Desarrollo

El procedimiento para efectuar el control del robot TurtleBot3 Waffle PI se divide en tres etapas las cuales se describen detalladamente en las siguientes subsecciones.

4.1 Generación de estados deseados

El propósito de un generador de estados deseados consiste en calcular una secuencia de estados los cuales conduzcan al robot para que éste siga una trayectoria predefinida tomando en consideración las posibles restricciones físicas y limites en los actuadores [12].

En este trabajo se optó por dividir a la trayectoria completa en segmentos más pequeños, generar los estados deseados en cada segmento y al final concatenarlos para obtener la trayectoria completa. El procedimiento consiste en calcular la distancia entre dos punto del plano 2D, a continuación almacenar los estados intermedios en un mensaje del tipo *nav_msgs::Odometry* para su posterior publicación.

Los estados calculados se componen de la posición en x, la posición en y, la orientación en ψ, así como las velocidades v y ω. Las velocidades se calculan de tal modo que sigan perfiles trapezoidales o triangulares como los de la Figura 5 con el objetivo de lograr que el robot efectué transiciones suaves entre segmentos.

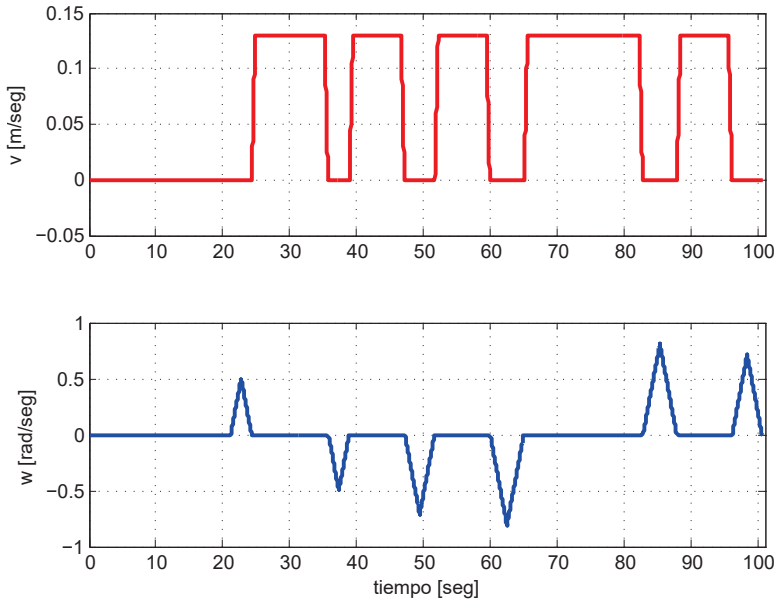


Figura 5. Perfiles de velocidad.

4.2 Estimación de estados del robot

Los estados del Turtlebot3 Waffle PI se obtienen directamente del *driver*. El nodo principal del robot incorpora un sistema que calcula la odometría del vehículo a partir de la información proporcionada por los encoders y de la unidad de medición inercial IMU. La información se publica en el tópico */odom* por medio de un mensaje *nav_msgs::Odometry*.

En la Figura 6 se muestra el trayecto generado por el robot tras recibir comandos de teleoperación. La trayectoria corresponde a la posición en el espacio 2D a partir de los datos de odometría calculados por el robot.

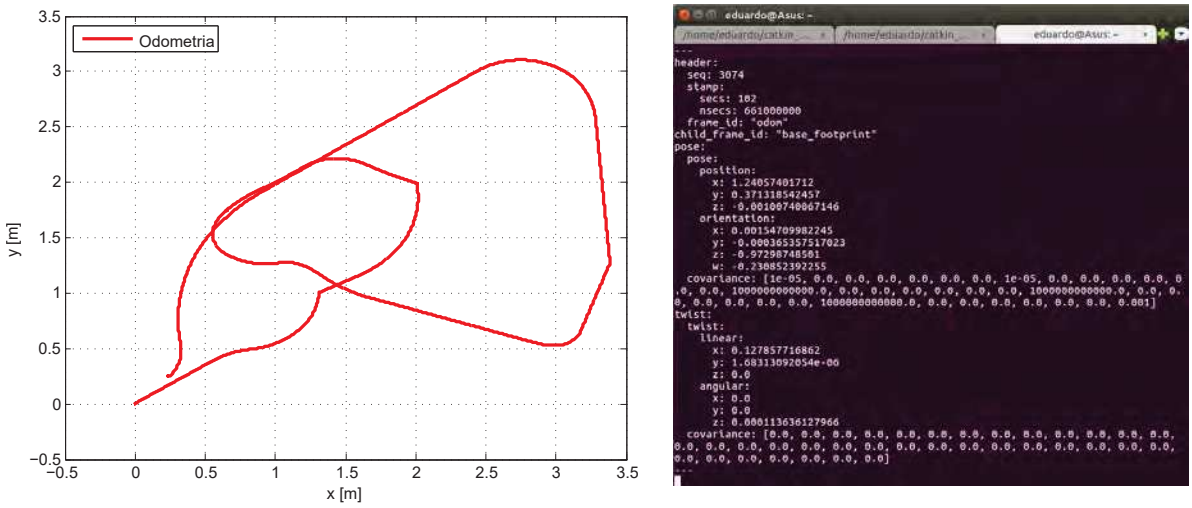


Figura 6. Odometría del Turtlebot3 Waffle PI.

4.3 Control

En esta sección se efectúa el diseño del sistema de control el cual tiene como tarea principal comandar al robot para que éste siga una trayectoria de referencia.

Como primer paso se procede a definir al vector de errores laterales $\boldsymbol{d}_{err}$, y al error en yaw ψ_{err} relativos a un segmento de la trayectoria. Un ejemplo de un segmento de una trayectoria deseada es una línea que inicia en (x_0, y_0) tangente al vector $\boldsymbol{t}$.

$$\boldsymbol{t} = \begin{bmatrix} cos(\psi_d) \\ sen(\psi_d) \end{bmatrix} \tag{3}$$

Y normal al vector $\boldsymbol{n}$.

$$\boldsymbol{n} = \begin{bmatrix} -sen(\psi_d) \\ cos(\psi_d) \end{bmatrix} \tag{4}$$

A continuación, se calcula el error lateral del robot en la posición (x, y) con ayuda de la siguiente expresión:

$$\boldsymbol{d}_{err} = \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \right) \boldsymbol{n} \tag{5}$$

El error ψ_{err} se calcula como la diferencia entre el ángulo en yaw actual y el ángulo en yaw deseado.

$$\psi_{err} = \psi - \psi_d \tag{6}$$

De esta forma el vector de errores queda definido de la siguiente manera:

$$\boldsymbol{e} = \begin{bmatrix} \boldsymbol{d}_{err} \\ \psi_{err}) \end{bmatrix} \tag{7}$$

El objetivo del algoritmo consiste en calcular las velocidades v y ω necesarias para hacer que el vector de errores tienda a asintóticamente a cero.

De esta manera:

$$v = v_d + K_d e_2 \tag{8}$$

$$\omega = \psi_d + K_\psi e_3 + K_l e_1 \tag{9}$$

Donde K_d, K_l y K_ψ son las ganancias de control. Las ecuaciones (8) y (9) son las encargadas de calcular las velocidades v y ω aplicadas al robot y con ello cerrar el lazo de control tal como se muestra en la Figura 7.

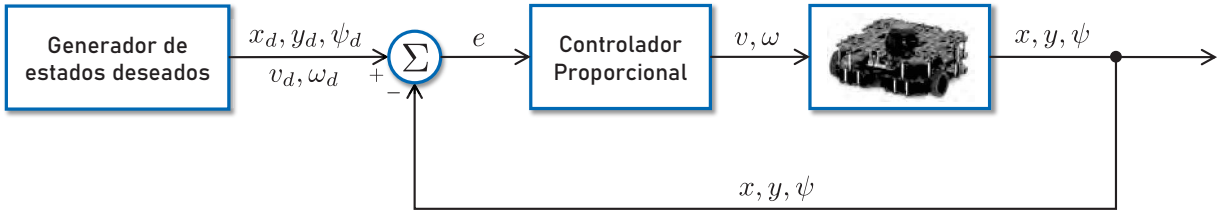


Figura 7. Lazo de control.

El controlador fue evaluado dentro del simulador *Gazebo* (Figura 8), considerando las siguientes ganancias de control (Tabla 2):

Tabla 2. Ganancias de control.

Parámetro	Valor
K_l	1.0
K_d	3.0
K_ψ	10.0

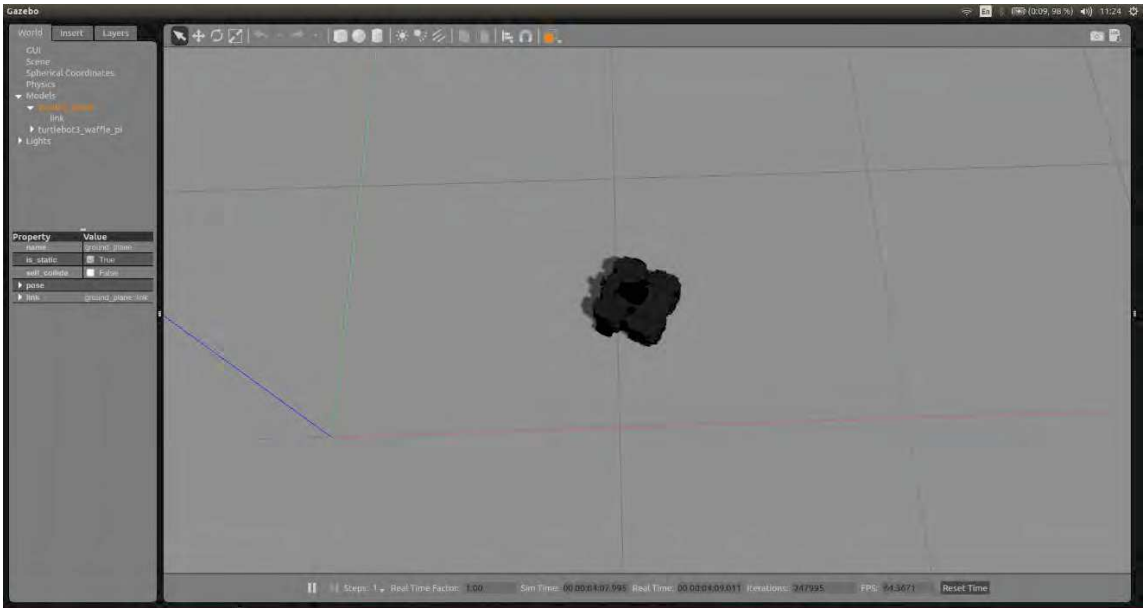


Figura 8. Robot Turtlebot3 Waffle PI dentro del entorno de Gazebo.

Desarrollos con enfoque mecatrónico, Capítulo 3, pp. 27 - 38
ISBN: 978-607-9394-22-6, año 2020

Tras ejecutar el programa principal el robot comenzó a seguir la trayectoria de referencia tal y como se esperaba. La Figura 9 se muestran los errores de estado durante el seguimiento de la trayectoria empleando la herramienta de ROS *rqt_plot*.

Figura 9. Errores de estado en rqt_plot.

La manera en que los nodos se encuentran interconectados se muestra en la Figura 10. Los nodos están representados por elipses y es donde se lleva a cabo el procesamiento de información. Los tópicos se representan por rectángulos y son los buses por los cuales se transmiten los datos. Por último, las flechas indican la dirección de flujo.

Como se puede observar la información fluye de acuerdo con la estructura del lazo de control de la Figura 7. El nodo */turtlebot_linear_control_node* se subscribe a los datos de odometría provenientes del tópico */odom*, así como los datos de los estados deseados provenientes del tópico */desired_states* para el cálculo las señales de control, las cuales se mandan directamente el robot a través del tópico *cmd_vel*.

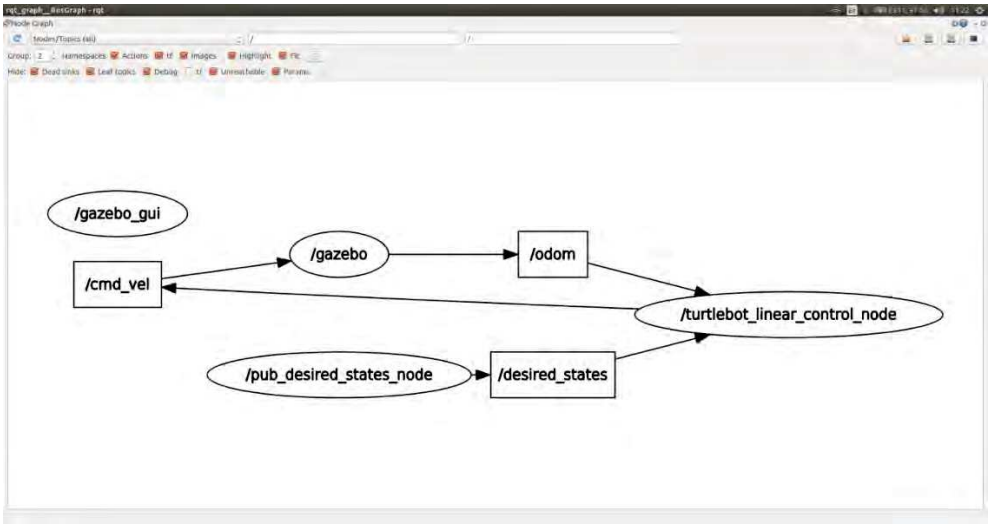


Figura 10. RQT Graph.

5. Resultados

El algoritmo de control fue programado como un nodo dentro de ROS. El programa principal se subscribe a los datos de odometría, así como los estados deseados y con esta información se obtiene el vector de errores. Lo errores de estado se usan en las ecuaciones (8) y (9) para el cálculo de las señales de control.

En la Figura 11 se aprecia la trayectoria trazada por el vehículo. La línea punteada indica la trayectoria deseada y la línea sólida el trayecto seguido por el robot. Como se puede notar, el algoritmo de control cumple de manera eficiente con su tarea al mantener al robot muy cercano a la trayectoria de referencia. Las flechas indican la dirección de avance, comenzando en el origen $(0, 0)$ y finalizando nuevamente en el origen $(0, 0)$.

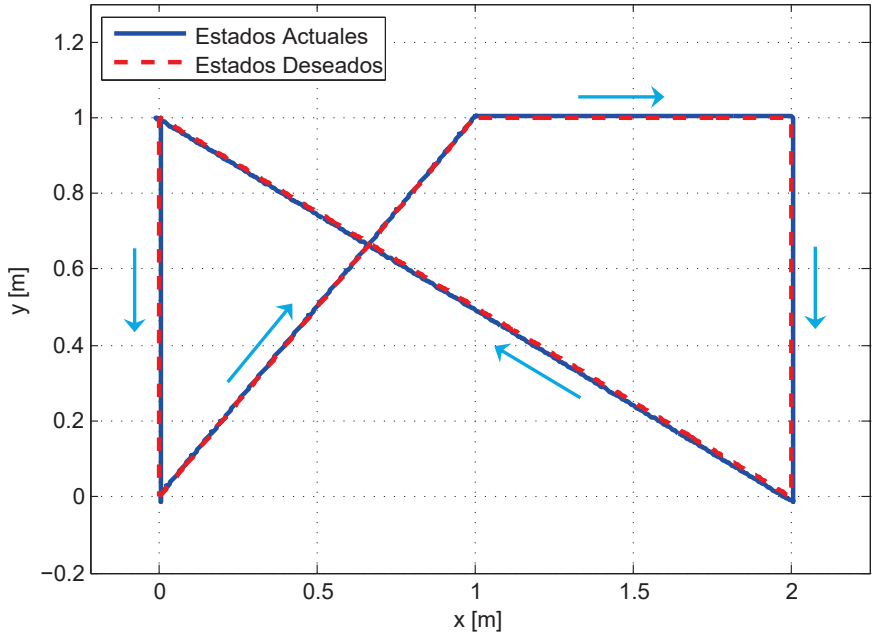


Figura 11. Trayectoria trazada por el robot.

Una forma más clara de observar el desempeño del controlador es a través de los errores de estado. Como se muestra en la Figura 12 la magnitud de los errores es muy pequeña y se mantienen acotados a medida que el tiempo evoluciona.

En la misma figura se pueden notar pequeñas protuberancias cada vez que el robot cambia de posición, sin embargo, al tratarse de sobreimpulsos de menor magnitud no afecta en el comportamiento final del sistema.

La implementación del controlador en la plataforma real y el desempeño de este mismo se puede ver en[3].

El comportamiento de las entradas de control al trazar la trayectoria de la Figura 11 se muestra en la Figura 13. La gráfica de la parte superior indica la velocidad lineal del robot. Como se aprecia el perfil de velocidad obtenido es de tipo trapezoidal.

[3] https://youtu.be/gjtTbT0YgIY

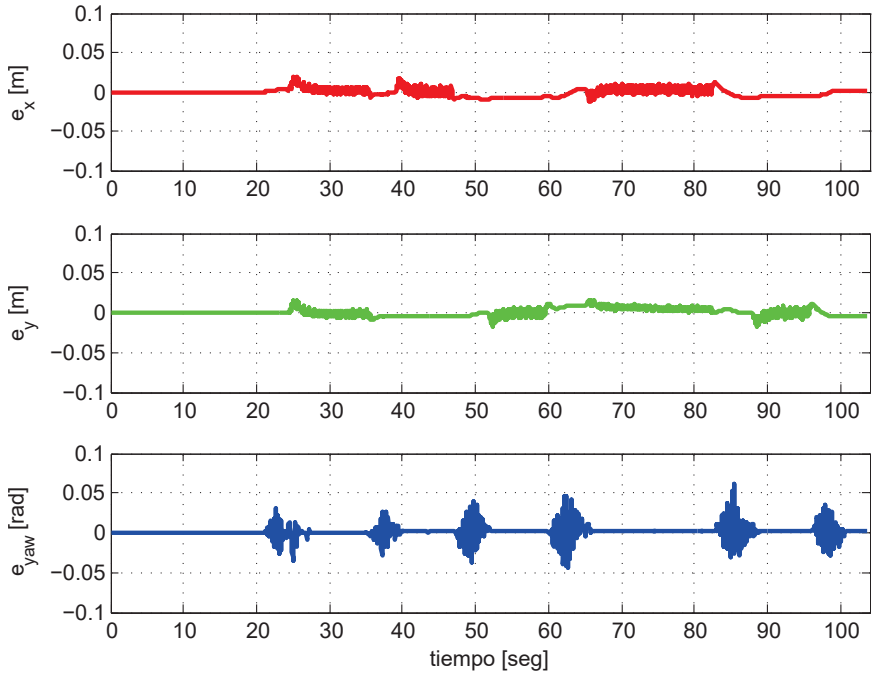


Figura 12. Errores de estado.

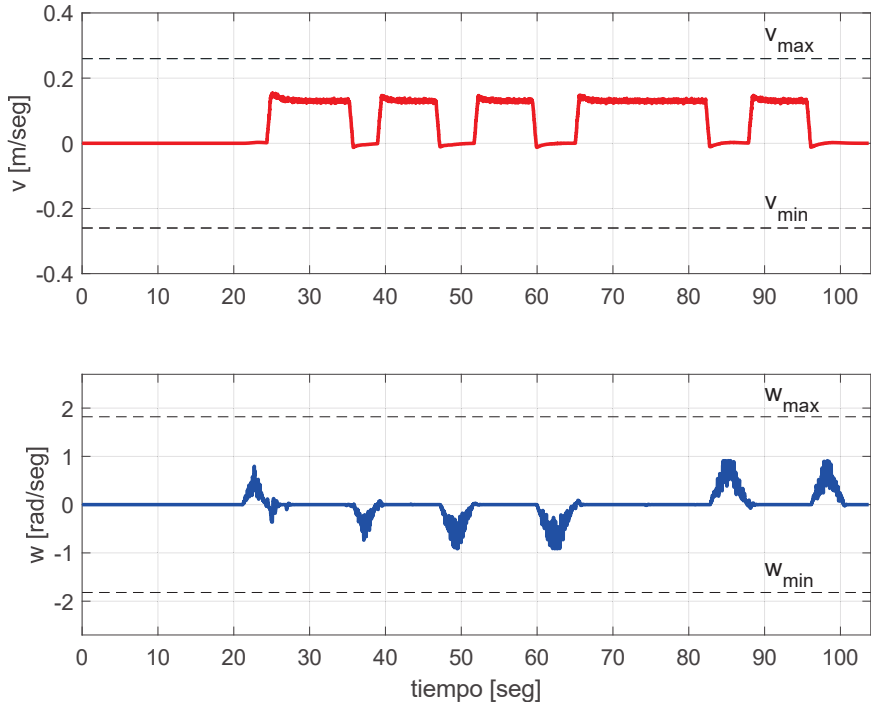


Figura 13. Entradas de control.

En la segunda grafica se muestra la velocidad angular del robot alrededor del eje z. En este caso el perfil de velocidad obtenido es de tipo triangular. Este tipo de perfiles permiten al robot efectuar transiciones suaves entre segmentos con la finalidad de evitar posibles colisiones. El plan consiste en iniciar con una velocidad baja y lentamente acelerar hasta alcanzar la velocidad máxima, cuando se esté a punto de llegar al final del recorrido del segmento nuevamente se reduce la velocidad de manera paulatina hasta llegar al estado de reposo. Lo anterior garantiza transiciones suaves entre segmentos.

Además, en la misma figura se aprecia que los límites en los actuadores (Tabla 1) nunca son violados, por lo que se puede tener la tranquilidad de que los motores nunca alzanzarán el límite de saturación.

6. Conclusiones

Los resultados obtenidos ilustran y validan el desempeño del controlador. El algoritmo es capaz de generar los comandos de velocidad adecuados para que el robot siga una trayectoria deseada. A pesar de que el controlador presentado se trata de un controlador lineal los resultados presentados son satisfactorios y cumplen con la tarea de seguimiento de trayectorias. Además, el algoritmo de control puede ser modificado para implementar leyes de control más robustos en la misma plataforma experimental.

En esta ocasión se hizo uso de los datos de odometría para estimar los estados del robot, sin embargo, se pueden utilizar otros métodos para mejorar la precisión en la estimación o incluso fusionar los datos proporcionados por los distintos sensores.

Como se mencionó en un inicio, la etapa de control solo se trata de una parte en el proceso de navegación autónoma, por lo que como trabajo futuro se puede proponer el diseño de un algoritmo inteligente capaz de generar trayectorias libres de colisiones, para posteriormente integrar ambas partes y así poder evaluar el sistema completo.

Referencias

[1] Siegwart R. y col. *“Introduction to autonomous mobile robots”*, 2 ed., MIT Press, E.U. 2011.
[2] Rubio F. y col. *“A review of mobile robots: Concepts, methods, theoretical framework, and applications”, International Journal of Advanced Robotic System*, págs. 1-22, 2019.
[3] Crnokić B. y col. *“Different applications of mobile robots in education”, International Journal on Integrating Technology in Education*. Vol. 6, Num. 3, págs. 15-28, 2017.
[4] Besseghieur K. L. y col. *“Trajectory tracking control for a nonholonomic mobile robot under ROS”, Journal of Physics*. págs. 1-5, 2018.
[5] Bensaci C. y col. *“Nonlinear control of a differential wheeled mobile robot in real time-Turtlebot2”, Third International Conference on Technological Advances in Electrical Engineering*, 2019.
[6] Elsayed M. y col. *“Real time trajectory tracking controller based on Lyapunov function for mobile robot”*, *International Journal of Computer Applications.* Vol. 168, págs. 1-6, 2017.
[7] Dhaouadi R. y Abu H. A. *“Dynamic modelling of differential-drive mobile robots using Lagrange and Newton-Euler methodologies: A unified framework”, Advances in Robotics and Automation,* 2013.
[8] Kuhne F. y Fetter L. W. *“Model predictive control of a mobile robot using linearization”, Proceedings of the IEEE Mechatronics and Robotics*. págs. 525-530, 2004.
[9] Klancar G. y Skrjanc I. *“Tracking-error model-based predictive control for mobile robots in real time”, Robotics and Autonomous Systems.* Vol. 55, Num. 6, págs. 460-469, 2007.
[10] https://robots.ros.org/ consulta: 5 oct. 2020.
[11] https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ consulta: 3 de oct. 2020.
[12] Newman W. S. *“A systematic approach to learning robot programming with ROS”*, CRC Press, E.U. 2018.
[13] Pyo Y. y col. “ROS robot programming”, ROBOTIS, Corea, 2017.

La Asociación Mexicana de Mecatrónica A.C. otorga el presente

RECONOCIMIENTO

a:

Salazar-Hidalgo Eduardo,
Castañeda-Camacho Josefina,
Martínez-Torres Cesar
y Martínez-Carranza José

Por su participación con el trabajo titulado:

“Seguimiento de trayectorias de un robot móvil diferencial a través del sistema operativo robótico ROS”, en el marco del 19° Congreso Nacional de Mecatrónica.

Lic. Jorge Enrique Orozco Ramírez
Presidente

3 y 4 de Diciembre
2020, México

Dr. J. Emilio Vargas Soto
Fundador

Dr. Fernando Israel Ireta Muñoz
Ingeniero R&D Nivel 2
INRIA
2004 Route des Lucioles
06902, Valbonne, Francia
fernando.ireta-munoz@inria.fr

INVENTEURS DU MONDE NUMÉRIQUE

01/12/2020

Constancia de asistencia al curso Reconstrucción 3D y Localización usando sensores RGB-D

A QUIEN CORREPONDA :

Por medio de la presente hago constar que el asistente **Eduardo Salazar Hidalgo** perteneciente a **Benemérita Universidad Autónoma de Puebla** ha participado y asistido al curso introductorio « *Reconstrucción 3D y localización con sensores RGB-D* », impartido por el Dr. Fernando Israel Ireta Muñoz del 21 de Septiembre al 30 de Octubre del 2020, con una duración de 25 horas. En el cual participaron 40 asistentes de 9 diferentes instituciones mexicanas a través de video conferencias.

El asistente participó activamente en el desarrollo práctico y teórico de los siguientes temas:

Sesión 1. Introducción a SLAM.
Sesión 2. Estimación de la posición (1ra parte): Formación de la imagen y calibración de cámaras.
Sesión 3. Estimación de la posición (2da parte) : Posición 6 grados de libertad, lie algebra y transformaciones homogéneas.
Sesión 4. Estimación de la posición (3ra parte) : Método IRLS (Iteratively Re-Least Squares).
Sesión 5. RGB-D SLAM (1ra parte): Métodos basados en profundidad y color.
Sesión 6. RGB-D SLAM (2da parte): Métodos híbridos y técnicas de mapeo.

El curso ha permitido al asistente ampliar su campos de investigación en el área de robotica y visión por computadora.

Cordialmente,
Dr. Fernando Israel Ireta Muñoz

TOEFL ITP Score Report

Name of Institution: UPAEP PUEBLA

Name: SALAZAR EDUARDO

Student Number: 9

DOB: 01/15/1994

Sex: M

Degree:

Times Taken TOEFL: None

Native Country: Mexico
Native Language: Spanish

Scaled Scores:

Listening Comprehension:	58
Structure & Written Expression:	57
Reading Comprehension:	60
Total Score:	583

Test Date: 01/25/2020
Form: TOEFL ITP

ETS TOEFL ITP

Student's File Copy
Do Not Copy

The face of this document has a security background. The back contains a watermark. Hold at an angle to view.

The *TOEFL® ITP* Assessment Series is designed to be used for placement, progress monitoring, and exit purposes. *TOEFL® ITP* scores can also be used for admissions to programs and institutions where English is not the dominant language of instruction for content courses. Learn more at **www.ets.org/toefl_itp/use.**

137012-16573 • FB619R100 • Printed in U.S.A. I.N. 770462