Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Electrónica

Maestría en Ciencias de la Electrónica
Opción en Automatización

Monografía

CONTROL PREDICTIVO Y TOLERANCIA A FALLAS DE UN CUADRIRROTOR

Alumno: **Jesus Cristian Velázquez Carrasco**

Directores de Tesis: **Dra. Josefina Castañeda Camacho (FCE-BUAP)**
Dr. César Martínez Torres (EDEI-UDLAP)

Puebla, Puebla Septiembre de 2025.

Resumen

El presente trabajo tiene como objetivo integrar una estrategia de control de detección y reconfiguración de fallas basada en planitud, en conjunto con un controlador predictivo Basado en modelo (MPC) aplicado a odometría visual previamente desarrollado, para el control automático de trayectorias en un cuadrirrotor Bebop 2. La propuesta busca que los controladores sean capaces de adaptarse y reconfigurarse ante la aparición de fallas, garantizando la continuidad de la misión, minimizando riesgos de daño y asegurando la supervivencia del sistema aéreo.

Se presenta la estructura diseñada para el esquema de control tolerante a fallas (FTC, por sus siglas en inglés) y su integración con el MPC, validando su desempeño mediante simulaciones en el entorno GAZEBO y, finalmente, implementando la programación en ROS.

Se presentan los resultados obtenidos mediante la aplicación de las estrategias propuestas, evidenciando que el desempeño del esquema de control está directamente relacionado con la capacidad de cómputo de la máquina utilizada. Asimismo, se establecen los requisitos y lineamientos que servirán como base para el desarrollo de investigaciones futuras que den continuidad a este trabajo.

Índice general

Introducción

Los vehículos aéreos no tripulados (UAV, por sus siglas en inglés) han experimentado un crecimiento sostenido en popularidad durante los últimos años [1]. Su integración en diversas actividades obedece principalmente a dos factores: la reducción de costos frente a la mano de obra humana y la capacidad de ejecutar tareas que resultan complejas, riesgosas o poco deseables para las personas.

De manera paralela, el número de usuarios y aplicaciones de los UAV se ha expandido de forma considerable. Entre las aplicaciones más representativas destacan la entrega de paquetería, la inspección autónoma, la vigilancia y el sensado remoto. No obstante, estas tareas suelen involucrar la interacción con personas o el manejo de cargas de alto valor, lo cual incrementa los riesgos asociados. Un accidente en tales contextos puede ocasionar pérdidas económicas significativas o, en casos extremos, comprometer la integridad física de las personas.

En este tipo de aplicaciones resulta esencial que el vehículo sea capaz de estimar su posición aun en entornos sin disponibilidad de señal GPS, mantener un seguimiento de trayectoria preciso frente a perturbaciones externas y, en caso de falla [2], ejecutar una reconfiguración eficiente que minimice riesgos para el sistema y su entorno. Esta reconfiguración puede plantearse bajo dos objetivos principales: culminar la trayectoria establecida o garantizar un aterrizaje seguro.

El esquema de detección y reconfiguración de fallas descrito en [3] constituye un referente en esta línea de investigación. Por otro lado, el controlador predictivo basado en modelo (MPC) propuesto en [4] se estructuró en tres controladores independientes: altitud, orientación y traslación. Sin embargo, los estados de orientación y traslación se encuentran intrínsecamente acoplados. En consecuencia, una falla en alguno de ellos impacta de manera directa o indirecta a dos de los tres controladores.

Mediante la aplicación de las propiedades de la planitud diferencial es posible reconstruir los estados del sistema, permitiendo recuperar al menos uno de los controladores comprometidos. Esta estrategia asegura que el sistema pueda completar la trayectoria prevista o, en su defecto, mantener condiciones de operación seguras para el cuadrirrotor.

Tomando en consideración las herramientas que proporciona el control tolerante a fallas basada en planitud, junto con la estructura de control predictivo de modelo dividida en tres controladores, este trabajo propone la integración de ambas estrategias. El objetivo es demostrar que, ante la ocurrencia de una falla, la combinación de estos controladores permite que el cuadrirrotor cumpla con la trayectoria asignada y, en última instancia, garantice su supervivencia operativa.

En este trabajo se presentan, en primer lugar, los antecedentes relacionados con el estudio de los cuadrirrotores. Posteriormente, se desarrolla el marco teórico que sustenta la propuesta, para finalmente exponer los resultados obtenidos, con los cuales se valida la hipótesis planteada.

Objetivos

Los objetivos general y específicos de este trabajo de tesis se enuncian a continuación.

Objetivo general

Diseñar e implementar un controlador lineal predictivo junto con una estrategia de reconfiguración de falla para el funcionamiento autónomo de un cuadrirrotor.

Objetivos particulares

Los objetivos particulares de este trabajo se enumeran a continuación:

1. Estudiar la arquitectura de detección y reconfiguración de falla basada en planitud diferencial para el control de fallas de un cuadrirrotor.
2. Estudiar la implementación del MPC lineal(por sus siglas en inglés *model predictive control*) con la odometría visual para el control de un cuadrirrotor.
3. Acoplar la estrategia MPC con la estrategia FTC(por sus siglas en inglés *fault tolerant control*).
4. Simular la estrategia MPC con la estrategia FTC en ROS-Gazebo.
5. Publicación de resultados.

Estado del Arte

En la actualidad, los cuadrirrotores han cobrado una relevancia creciente, impulsada por los avances tecnológicos en el campo de la automatización. Se ha trabajado intensamente para dotar a estos vehículos de capacidades autónomas, lo cual ha permitido su integración tanto en el ámbito industrial como en el social, facilitando la ejecución de tareas en las que pueden superar el desempeño humano.

Entre las aplicaciones que han despertado el interés de diversas entidades en su uso y fabricación, se encuentran: operaciones de búsqueda y rescate, inspección de infraestructuras, agricultura de precisión, entrega de paquetería, entretenimiento, así como la recolección de información geográfica y meteorológica, entre otras.

De igual manera, la relevancia del control de fallas ha aumentado debido al crecimiento del número de vehículos que operan —o buscan operar— con un alto grado de autonomía. Para comprender con mayor claridad el estudio del control tolerante a fallas, es necesario identificar diversas líneas de investigación relevantes que sirven como referencia y respaldo a este trabajo. Dichas líneas permiten reconocer la pertinencia de integrar el control predictivo basado en modelo (MPC) con estrategias de control tolerante a fallas (FTC), lo cual justifica el planteamiento de su acoplamiento y resalta la importancia de la propuesta presentada en esta tesis.

Aplicación de inteligencia artificial al control de drones

Gracias a los avances recientes en inteligencia artificial, ha surgido un notable interés por el uso de herramientas que permitan predecir y optimizar trayectorias en tiempo real. Un ejemplo representativo es el artículo publicado en 2024 en [5], el cual presenta una revisión del estado del arte sobre la aplicación de técnicas de inteligencia artificial al control de drones. En este trabajo se destacan enfoques como el aprendizaje profundo, las redes neuronales y los algoritmos evolutivos, que contribuyen a mejorar la estabilidad, autonomía y adaptabilidad de los sistemas de control en entornos dinámicos. Asimismo, el estudio aborda la integración de estas tecnologías en plataformas de hardware específicas.

Optimización energética mediante MPC

Un estudio reciente aborda la gestión energética en UAVs mediante un modelo predictivo que considera parámetros como altitud y velocidad. Este enfoque permite optimizar el consumo de energía y mejorar la autonomía de los drones, especialmente en misiones prolongadas. Los resultados obtenidos sugieren que la implementación de este modelo puede reducir significativamente el consumo energético sin comprometer el rendimiento [6].

Este tipo de avances en el control de trayectorias evidencia una evolución constante, que busca mantenerse a la vanguardia frente al ritmo acelerado del desarrollo tecnológico actual. Como consecuencia, también ha aumentado el interés en investigaciones centradas en el control tolerante a fallas, orientadas a garantizar un funcionamiento seguro y fiable en condiciones adversas.

Robuztez en comunicacion de UAVs

Por ejemplo, en [7] se propone un marco de trabajo que integra técnicas de sensado, interferencia (*jamming*) y comunicaciones seguras para robustecer las redes de UAVs. Este enfoque permite detectar y mitigar amenazas en tiempo real, incrementando la seguridad y confiabilidad de las comunicaciones en entornos hostiles. Los resultados experimentales respaldan la viabilidad de esta propuesta tanto para aplicaciones militares como civiles.

Nuevas arquitecturas de control tolerante a fallas

Asimismo, trabajos recientes como el de Giral [8] presentan un enfoque innovador basado en transformadores para el control tolerante a fallas en UAVs de ala fija. Este enfoque está diseñado para adaptarse en tiempo real a cambios dinámicos ocasionados por daños estructurales o fallas en los actuadores. Utilizando mecanismos de atención y aprendizaje contextual propios de los modelos tipo transformador, el sistema mapea directamente los valores de referencia del lazo exterior —altitud, rumbo y velocidad del aire— en comandos de control, eliminando la necesidad de capas explícitas de detección de fallas. Los resultados experimentales demuestran que este controlador supera el rendimiento de sistemas de control tradicionales y de métodos de aprendizaje por refuerzo, tanto en condiciones nominales como en escenarios de fallas extremas.

Considerando estos avances tecnológicos, resulta natural que metodologías basadas en la planitud diferencial se vuelvan cada vez más accesibles y aplicables. Un caso representativo es el trabajo presentado en [9], donde se desarrolla un esquema de control basado en planitud que gestiona restricciones de entrada para un cuadricóptero en el espacio de salida plana, incorporando un componente de saturación. La estabilidad del sistema se demuestra mediante argumentos de Lyapunov, y la viabilidad del método se valida tanto en simulaciones como en pruebas experimentales utilizando una plataforma de nano-dron. Este controlador destaca por su capacidad para garantizar estabilidad y cumplimiento de restricciones, al tiempo que mantiene un bajo requerimiento computacional, lo que favorece su implementación en sistemas embebidos.

En la actualidad, es posible identificar artículos de investigación que guardan una mayor relación con la propuesta presentada en este trabajo, tales como en el trabajo de Mohammadi y Ramezani [10] se propone un esquema de control tolerante a fallas (FTC) basado en control predictivo de modelo (MPC) aplicado a un dron de tipo quadrotor. La principal contribución de este estudio radica en el diseño de una estrategia sin necesidad de un observador, utilizando en su lugar relaciones de redundancia analítica para la detección y compensación de fallas en los sensores.

El enfoque resulta particularmente relevante, ya que logra una respuesta robusta ante incertidumbres paramétricas y perturbaciones externas, garantizando estabilidad y desempeño aceptable incluso cuando se presentan fallas parciales o totales en los sensores.

Aunque el estudio de los drones ha avanzado significativamente, las investigaciones actuales muestran que aún existen áreas por explorar y mejorar. A diferencia de los enfoques previamente estudiados en el control de fallas de drones, la propuesta aquí planteada busca ir un paso más allá mediante el uso de un esquema de control más robusto: el control predictivo basado en modelo (MPC, por sus siglas en inglés). Lo destacable de este enfoque es su capacidad para desacoplar el control de la orientación, la posición y la altura del dron. Esta separación funcional permite que, ante una falla en alguno de estos componentes, los demás puedan seguir operando de forma precisa, lo que incrementa la confiabilidad y robustez del sistema. Este principio se alinea con el objetivo fundamental de este trabajo: garantizar la integridad y operatividad del sistema ante posibles fallas. Además, se busca implementar esta metodología dentro del entorno de desarrollo ROS, en el cual aún son escasas las aplicaciones prácticas de este tipo de controladores acoplados.

Capítulo 1

Marco teórico

En este capítulo se presentan las bases teóricas para el estudio del control de un cuadrirrotor.

1.1. Modelo Dinámico de un cuadrirrotor

Un cuadrirrotor es un vehículo aéreo conformado por cuatro rotores en los cuales puede tener una configuración de X o de +. En este caso en concreto se trabajará con la configuración "X", con la diferencia su contraparte que el frente se encuentra entre dos rotores. Los rotores cumplen la función de que al variar su velocidad angular, realiza un movimiento en específico, en cualquiera de los grados de libertad, presentados en la Figura 1.1.

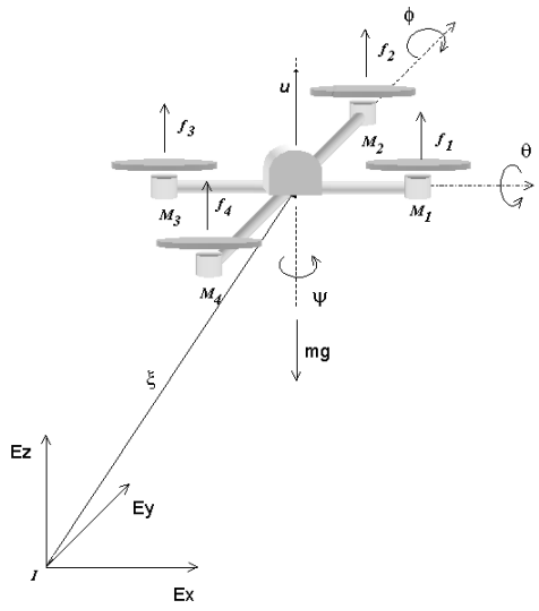

Figura 1.1: Diagrama de cuerpo libre de un cuadrirrotor y sus grados de libertad [11].

El movimiento sobre el eje z, se logra variando la velocidad angular de todos los rotores de manera simultanea, al aumentar la velocidad de los rotores 2 y 3 y disminuyendo los rotores de 1 y 4 se logra un movimiento frontal (movimiento en *pitch*).mientras que si se aumenta la velocidad de los rotores 1 y 2, y se disminuye 3 y 4, entonces se logra un desplazamiento lateral (movimiento en *roll*).

1.1.1. Ecuaciones del sistema dinámico

Para las ecuaciones de movimiento que describen la dinámica del cuadrirrotor, se usará el formalismo de Euler-Lagrange [12] que nos describe el balance de energía de un cuerpo en movimiento.

De ella se parte con que la posición relativa del sistema de referencia de nuestro vehículo β con respecto al sistema de referencia inercial I se encuentra definida por el vector $\boldsymbol{\xi} \in \mathbb{R}^3$, las ecuaciones fueron extraídas de [11].

$$\boldsymbol{\xi} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1.1}$$

En el caso de la orientación del vehículo esta quedará representada por el vector $\boldsymbol{\eta} \in \mathbb{R}^3$ el cual estará compuesto por los ángulos de Euler.

$$\boldsymbol{\eta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \tag{1.2}$$

Las ecuaciones generalizadas del vehículo aéreo descritas descritos por el vector $\boldsymbol{q} \in \mathbb{R}^6$:

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{\eta} \end{bmatrix} \tag{1.3}$$

Con estas ecuaciones se construye la ecuación de movimiento basado en Euler-Lagrange que viene dada por:

$$\frac{d}{dt}\left[\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{q}}}\right] - \frac{\partial \mathcal{L}}{\partial \boldsymbol{q}} = \begin{bmatrix} \boldsymbol{F}_{\xi} \\ \boldsymbol{\tau} \end{bmatrix} \tag{1.4}$$

De la ecuación anterior $\mathcal{L}$ representa el lagrangiano del sistema, el cual es la diferencia entre la energía cinética y la energía potencial del sistema.

$$\mathcal{L} = \mathcal{K} - \mathcal{U} \tag{1.5}$$

1.1.2. Ecuaciones del movimiento traslacional

La energía cinética del sistema está dado por la siguiente ecuación:

$$\mathcal{K} = \frac{1}{2} m \dot{\boldsymbol{\xi}}^T \dot{\boldsymbol{\xi}} \tag{1.6}$$

En donde m representa la masa del vehículo. Así mismo la energía potencial se expresa de la siguiente manera:

$$\mathcal{U} = mgz \tag{1.7}$$

En donde g representa la fuerza de gravedad. Con estas ecuaciones se construye el lagrangiano del sistema como se describe a continuación:

$$\mathcal{L} = \frac{1}{2} m \dot{\boldsymbol{\xi}}^T \dot{\boldsymbol{\xi}} - mgz \tag{1.8}$$

Una vez obtenido el lagrangiano, traducimos la ecuación 1.4 con el vector $\boldsymbol{\xi}$ como se muestra a continuación:

$$\frac{d}{dt}\left[\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{\xi}}}\right] - \frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}} = \boldsymbol{F}_\xi \tag{1.9}$$

Evaluando las operaciones de la ecuación anterior, finalmente obtendríamos la siguiente relación:

$$m\ddot{\boldsymbol{\xi}} + mg = \boldsymbol{F}_\xi \tag{1.10}$$

Donde $\boldsymbol{F}_\xi$ representa la fuerza aplicada al cuadrirrotor generada por el empuje total de los rotores misma que viene expresada de la siguiente forma:

$$\boldsymbol{F}_\xi = R\hat{\boldsymbol{F}} \tag{1.11}$$

Al mismo tiempo se tiene que:

$$\hat{\boldsymbol{F}} = \begin{bmatrix} 0 \\ 0 \\ F \end{bmatrix} \tag{1.12}$$

Siendo F la representación de la sumatoria de empujes ejercidos por cada rotor.

$$F = \sum_{i=1}^{4} f_i \tag{1.13}$$

El empuje de cada motor está dado por:

$$f_i = k_T \omega^2 \tag{1.14}$$

$R \in \mathbb{R}^3$ determina la orientación del vehículo al sistema de coordenadas inercial. El calculo de esta matrices viene desarrollado en abundancia en [4] y en [11].

Finalmente obtenemos que

$$\boldsymbol{F}_\xi = \begin{bmatrix} \cos\phi \text{sen}\theta \cos\psi + \text{sen}\phi\text{sen}\psi \\ \cos\phi \text{sen}\theta \text{sen}\psi - \text{sen}\phi\cos\psi \\ \cos\phi\cos\theta \end{bmatrix} F \tag{1.15}$$

Finalmente, sustituyendo estas ecuaciones, obtendríamos la dinámica traslacional de un cuadrirrotor tal como se muestra a continuación:

$$m\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \cos\phi \text{sen}\theta \cos\psi + \text{sen}\phi\text{sen}\psi \\ \cos\phi \text{sen}\theta \text{sen}\psi - \text{sen}\phi\cos\psi \\ \cos\phi\cos\theta \end{bmatrix} F - \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{1.16}$$

1.1.3. Ecuaciones del movimiento rotacional

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

$$\mathcal{K} = \frac{1}{2}\boldsymbol{\Omega}^T \boldsymbol{I}\boldsymbol{\Omega} \tag{1.17}$$

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

$$\boldsymbol{\Omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{1.18}$$

De igual forma, suponemos que la distribución de la masa del cuadrirrotor es simétrica, se considera el tensor de inercia $I \in \mathbb{R}^{3\times3}$ el cual es una matriz diagonal como se presenta a continuación:

$$\boldsymbol{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{1.19}$$

Las componentes de la velocidad angular del cuadrirrotor β se puede representar de la siguiente forma:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\text{sen}\theta \\ 0 & \cos\phi & \text{sen}\phi\cos\theta \\ 0 & -\text{sen}\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{1.20}$$

$$\boldsymbol{\Omega} = W_\eta \dot{\boldsymbol{\eta}} \tag{1.21}$$

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

$$W_\eta = \begin{bmatrix} 1 & 0 & -\text{sen}\theta \\ 0 & \cos\phi & \text{sen}\phi\cos\theta \\ 0 & -\text{sen}\phi & \cos\phi\cos\theta \end{bmatrix} \tag{1.22}$$

La matriz $\mathbb{J}$ actúa como matriz de inercia para la energía cinética total rotacional del cuadrirrotor, expresada directamente en términos de las coordenadas generalizadas η.

$$\mathbb{J} = W_\eta^T \boldsymbol{I} W_\eta \tag{1.23}$$

Sustituyendo 1.23 en 1.17 obtenemos lo siguiente:

$$\mathcal{K} = \frac{1}{2}\dot{\boldsymbol{\eta}}^T \mathbb{J} \dot{\boldsymbol{\eta}} \tag{1.24}$$

De esta forma, el Lagrangiano de la dinámica rotacional queda expresado como sigue, teniendo en consideración que aquí no toma parte la energía potencial al no haber variaciones en la altura.

$$\mathcal{L} = \frac{1}{2}\dot{\boldsymbol{\eta}}^T \mathbb{J} \dot{\boldsymbol{\eta}} \tag{1.25}$$

La ecuación del movimiento de Euler Lagrange para la dinámica de rotación queda expresada de la siguiente manera:

$$\frac{d}{dt}\left[\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{\eta}}}\right] - \frac{\partial \mathcal{L}}{\partial \boldsymbol{\eta}} = \boldsymbol{\tau} \tag{1.26}$$

Para finalmente obtener:

$$\mathbb{J}\ddot{\boldsymbol{\eta}} + \bar{V} = \boldsymbol{\tau} \tag{1.27}$$

En donde $\tau \in \mathbb{R}^3$ representa los momentos de $yaw, pitch$ y $roll$ y $\bar{V}$ está expresado de la siguiente manera:

$$\bar{V} = \left(\dot{\mathbb{J}} - \frac{1}{2}\frac{\partial}{\partial \eta}\left(\dot{\eta}^T \mathbb{J}\right)\right)\dot{\eta} = C\dot{\eta} \tag{1.28}$$

Donde C representa los términos de Coriolis, el cual contiene los efectos giroscópicos y centrífugos asociados con η.

De igual manera:

$$M(\boldsymbol{\eta}) = \mathbb{J} \tag{1.29}$$

Donde $M(\boldsymbol{\eta})$ es la matriz de inercia que queda representada de la siguiente forma:

$$M(\boldsymbol{\eta}) = \begin{bmatrix} I_{xx} & 0 & -I_{xx}S_\theta \\ 0 & I_{yy}\cos^2\phi + I_{zz}\text{sen}^2\phi & [I_{yy} - I_{zz}]\text{sen}\phi\cos\phi\cos\theta \\ -I_{xx}\text{sen}\theta & [I_{yy} - I_{zz}]\text{sen}\phi\cos\phi\cos\theta & m_{33} \end{bmatrix} \tag{1.30}$$

en donde m_{33} es igual a: $I_{xx}\text{sen}^2\theta + I_{yy}\text{sen}^2\phi\cos^2\theta + I_{zz}\cos^2\phi\cos^2\theta$. Finalmente de la ecuación 1.27 obtenemos:

$$M(\boldsymbol{\eta})\ddot{\boldsymbol{\eta}} + C\dot{\boldsymbol{\eta}} = \boldsymbol{\tau} \tag{1.31}$$

En donde $C\dot{\eta}$:

$$C\dot{\eta} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \tag{1.32}$$

En donde:

$C_{11} = 0$

$C_{12} = (I_{yy} - I_{zz})(\dot{\theta}\cos\phi\text{sen}\phi + \dot{\psi}\text{sen}^2\phi\cos\theta) + (I_{zz} - I_{yy})\dot{\psi}\cos^2\theta\cos\theta - Ixx\dot{\psi}\cos\theta$

$C_{13} = (I_{zz} - I_{yy})\dot{\psi}\cos\phi\text{sen}\phi\cos^2\theta$

$C_{21} = (I_{zz} - I_{yy})(\dot{\theta}\cos\phi\text{sen}\phi + \dot{\psi}\text{sen}^2\phi\cos\theta) + (I_{yy} - I_{zz})\dot{\psi}\cos^2\theta\cos\theta - Ixx\dot{\psi}\cos\theta$

$C_{22} = (I_{zz} - I_{yy})\dot{\theta}\cos\phi\text{sen}\phi$

$C_{23} = -I_{xx}\dot{\psi}\text{sen}\theta\cos\theta + Iyy\dot{\psi}\text{sen}^2\phi\text{sen}\theta\cos\theta + I_{zz}\dot{\psi}\cos^2\phi\text{sen}\theta\cos\theta$

$C_{31} = (I_{yy} - I_{zz})\dot{\psi}\cos^2\theta\text{sen}\phi\cos\phi - I_{xx}\dot{\theta}\cos\theta$

$C_{32} = (I_{zz} - I_{yy})(\dot{\theta}\cos\phi\text{sen}\phi\text{sen}\theta + \dot{\phi}\text{sen}^2\phi\cos\theta) + (I_{yy} - I_{zz})\dot{\phi}\cos^2\phi\cos\theta + I_{xx}\dot{\psi}\text{sen}\theta\cos\theta -$
$- I_{yy}\dot{\psi}\text{sen}^2\phi\text{sen}\theta\cos\theta - I_{zz}\dot{\psi}\cos^2\phi\text{sen}\theta\cos\theta$

$C_{33} = (I_{yy} - I_{zz})\dot{\phi}\cos\phi\text{sen}\phi\cos^2\theta - I_{yy}\dot{\theta}\text{sen}^2\phi\cos\theta\text{sen}\theta - I_{zz}\dot{\theta}\cos^2\phi\cos\theta\text{sen}\theta + Ixx\dot{\theta}\cos\theta\text{sen}\theta$

1.2. Planitud diferencial

La teoría diferencial de planitud busca determinar si un sistema de ecuaciones diferenciales puede ser parametrizado por funciones arbitrarias. Un sistema lineal o no lineal es plano si

existe un set de variables, diferencialmente independientes, llamadas salidas planas, las cuales son iguales a la cantidad de salidas de control. Un ejemplo seria que el vector estado y las entradas de control pueden ser expresadas como funciones de las salidas planas y un numero finito de sus derivadas de tiempo.

Como consecuencia, las trayectorias de control y de estado pueden ser obtenidas estudiando solo las trayectorias de salida planas.

Consideremos el sistema no linear $\dot{x} = f(x,u), x \in \mathbb{R}^n$ el vector estado, $u \in \mathbb{R}^m$ el vector control y f una C^∞ función de x y u. El sistema es diferencialmente plano si, y solo si, existe una salida vector plano $z \in \mathbb{R}^m$ tal que

- El vector salida plano esta expresado como la función del estado x y la entrada de control u y un numero finito de sus derivadas en el tiempo.

$$z = \phi_z\left(x, u, \dot{u}, ..., u^\gamma\right) \tag{1.33}$$

- El estado x y la entrada de control u están expresadas como la función del vector z y un numero finito de sus derivadas en el tiempo.

$$x = \phi_x\left(z, \dot{z}, ..., z^a\right) \tag{1.34}$$

$$u = \phi_u\left(z, \dot{z}, ..., z^{a+1}\right) \tag{1.35}$$

En donde z^a seria la a^{th} derivada con respecto al tiempo de z.

1.3. Odometría visual

Los sistemas de visión por computadora se dividen principalmente en dos etapas. La primera parte consiste en el desarrollo de los dispositivos capaces de convertir la luz en imágenes digitales. La segunda etapa se enfoca en el desarrollo de algoritmos con procesamiento de dichas imágenes.

1.3.1. Modelo de la cámara obscura

El diagrama que muestra el modelo de una lente delgada se presenta en la figura 1.2:

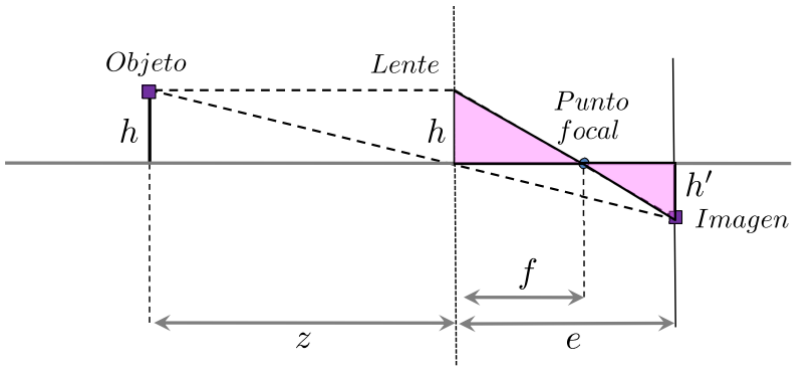

Figura 1.2: Diagrama de la lente delgada [4].

Empleando el principio de los triangulos semejantes, podemos obtener las siguientes relaciones:

$$\begin{aligned} \frac{h'}{h} &= \frac{e}{z} \\ \frac{h'}{h} = \frac{e-f}{f} &= \frac{e}{f} - 1 \\ \frac{e}{f} - 1 &= \frac{e}{z} \\ \frac{1}{f} &= \frac{1}{z} + \frac{1}{e} \end{aligned} \tag{1.36}$$

Donde z representa la distancia hacia el objeto, e la distancia hacia la imagen, y f la distancia focal del lente. Si hacemos las siguientes consideraciones:

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

Por lo tanto:

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

Y finalmente se obtiene la siguiente relación:

$$\frac{h'}{h} = \frac{f}{z} \tag{1.37}$$

La expresión 1.37 explica porque en una imagen digital, los objetos lejanos aparentan ser mas pequeños.

1.3.2. Modelo matemático de una cámara

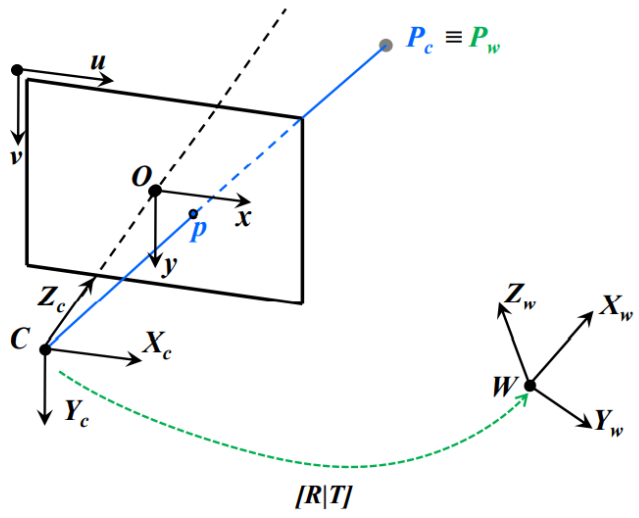

Figura 1.3: Geometría de una cámara que emplea el principio de la cámara obscura [4].

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. 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 pixeles. 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 pixeles.
3. Realizar una transformación para representar P_c en coordenadas con respecto al sistema de referencia global P_w.

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

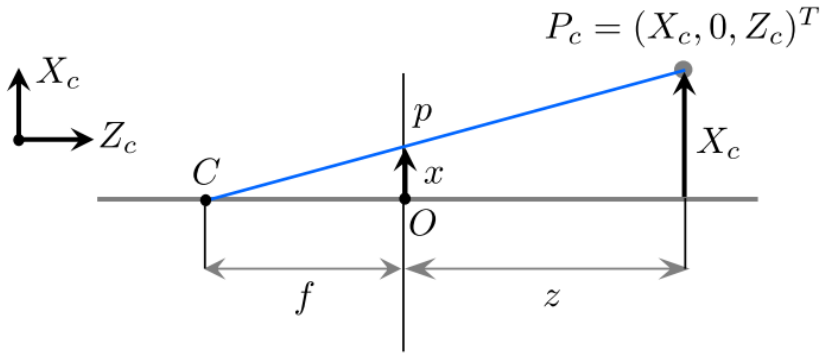

Figura 1.4: Proyección del punto P_c en el plano de la imagen [4].

$$\frac{x}{f} = \frac{X_c}{Z_c} \Longrightarrow x = \frac{fX_c}{Z_c} \tag{1.38}$$

$$\frac{y}{f} = \frac{Y_c}{Z_c} \Longrightarrow y = \frac{fY_c}{Z_c} \tag{1.39}$$

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

$$(X_c, Y_c, Z_c)^T \to \left(\frac{fX_c}{Z_c}, \frac{fY_c}{Z_c}\right)^T \tag{1.40}$$

Las ecuaciones 1.38 y 1.39 pueden ser representadas mediante coordenadas homogéneas. Considérese los siguientes vectores:

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \boldsymbol{P_c} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{1.41}$$

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

$$\begin{bmatrix} \lambda_x \\ \lambda_y \\ \lambda \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{1.42}$$

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 1.40 asume que el origen del sistema de coordenadas del plano de la imagen se ubica sobre el eje óptico, lo que en la practica no siempre resulta ser cierto.

Al tratarse de pixeles, 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 $offset$. Ademas, debido a que las coordenadas de un punto en la imagen se mide en pixeles se debe de considerar un factor de escala para las direcciones horizontal y vertical.

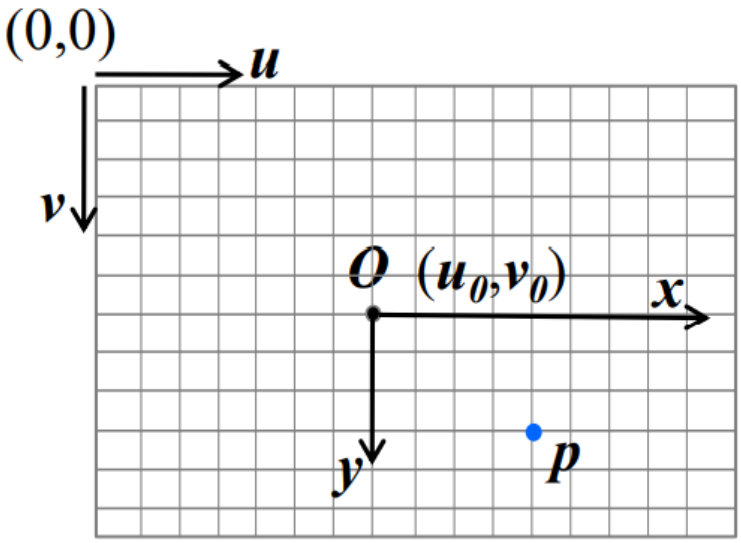

Figura 1.5: Plano pixeles [4].

De este modo se tiene que las siguientes ecuaciones:

$$u = u_0 + k_u x \Longrightarrow u = u_0 + \frac{k_u f X_c}{Z_c} \tag{1.43}$$

$$v = v_0 + k_v y \Longrightarrow v = v_0 + \frac{k_v f Y_c}{Z_c} \tag{1.44}$$

Para efectuar el siguiente mapeo.

$$(X, Y_c, Z_c)^T \longrightarrow \left(u_0 + \frac{k_u f X_c}{Z_c}, v_0 + \frac{k_v f Y_c}{Z_c} \right)^T \tag{1.45}$$

Lo anterior es expresado en coordenadas homogéneas.

$$\begin{bmatrix} \lambda_u \\ \lambda_v \\ \lambda \end{bmatrix} = \begin{bmatrix} k_u f & 0 & u_0 & 0 \\ 0 & k_v f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{1.46}$$

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. Dicha transformación está compuesta por una rotación R seguida de una traslación $\boldsymbol{t}$.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + t \tag{1.47}$$

Con esta transformación, la ecuación 1.48 se puede reescribir de la siguiente forma:

$$\begin{bmatrix} \lambda_u \\ \lambda_v \\ \lambda \end{bmatrix} = \begin{bmatrix} k_u f & 0 & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{1.48}$$

$$\lambda \boldsymbol{p} = K[R|t]\boldsymbol{P}_w \tag{1.49}$$

Donde K se conoce como matriz de parámetros intrínsecos y $[R|t]$ como matriz de parámetros extrínsecos. Estos parámetros se obtienen durante el proceso de calibración de la cámara [4].

1.3.3. Formulación de la Odometría visual

En este caso, el análisis se llevará a cabo considerando un sistema de odometría visual monocular. Para extender el 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 discretos k. En el caso de un sistema monocular, el conjunto de imágenes tomadas en los tiempo k estará denotado por $I_{0:n} = I_0, ..., I_n$. La figura nos muestra esta configuración:

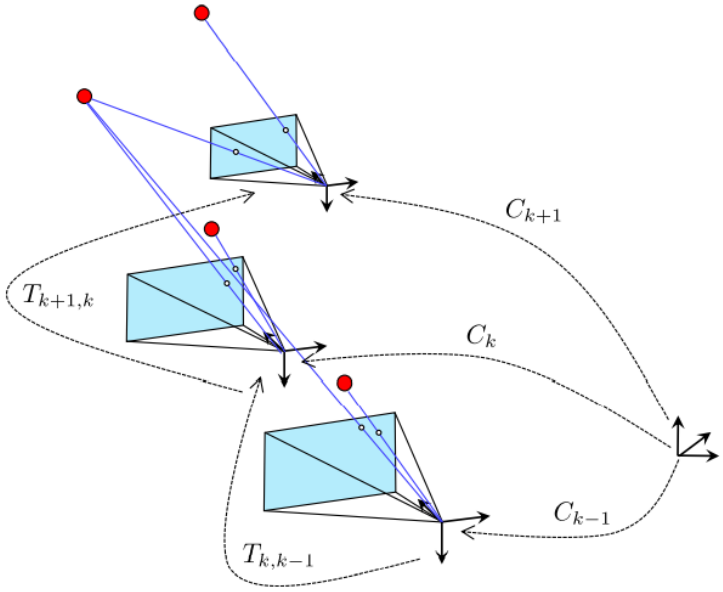

Figura 1.6: Ilustración del problema de Odometria visual. Las poses relativas $T_{k,k-1}$ de las posiciones adyacentes de la camara son calculadas a partir de características visuales y posteriormente concatenadas para obtener la pose absoluta C_k con respecto al sistema de coordenadas inicial en $k = 0$ [4].

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

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{1.50}$$

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

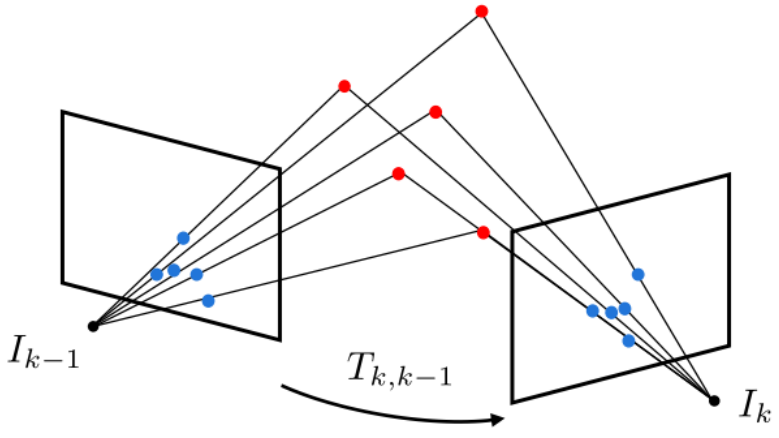

Figura 1.7: Transformación entre dos imágenes consecutivas [4].

Para simplificar la notación, desde ahora en adelante, se emplea T_k en lugar de $T_{k,k-1}$. El conjunto de poses en la cámara $C_{0:n} = C_0, ..., C_n$ 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 $k = 1, ..., n$. Por lo tanto:

$$C_n = C_{n-1}T_n \tag{1.51}$$

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 ultimas m poses para obtener una estimación mas precisa de la trayectoria local.

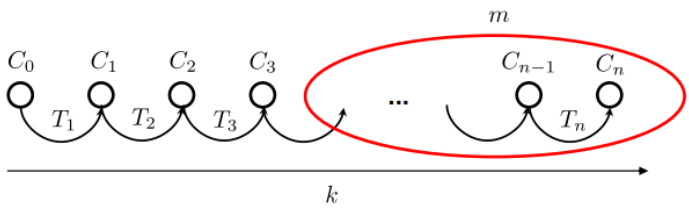

Figura 1.8: Refinamiento iterativo sobre las ultimas m poses [4].

Este refinamiento iterativo funciona al minimizar la suma de los cuadrados de los errores de reproyección de los puntos 3D re proyectados en las ultimas imágenes, conocido como $windowed - bundleadjustment$. Los puntos tridimensionales se obtienen por triangulación de los puntos detectados en cada imagen.

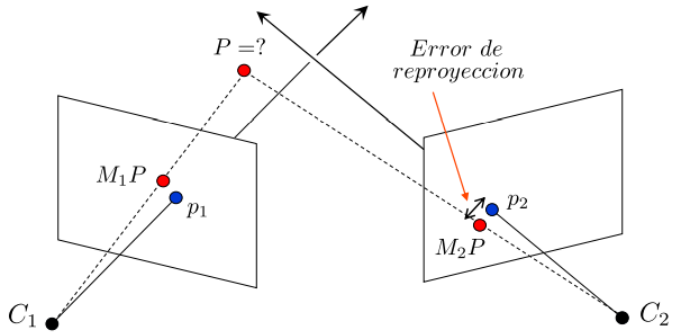

Figura 1.9: Refinamiento iterativo sobre las ultimas m poses [4].

El diagrama a bloques del algoritmo de odometría visual se resume en la figura tal. Para cada nueva imagen I_k, el primer paso consiste en detectar las características mas 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 1.7). 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 ultimas m imágenes para obtener una estimación mas precisa de la trayectoria.

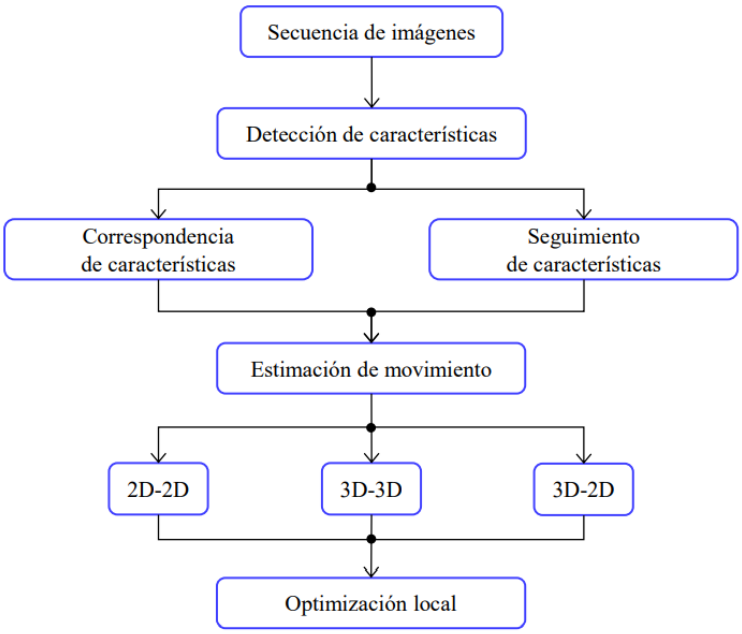

Figura 1.10: Error de reproyección visual [4].

1.4. Control Predictivo de Modelo (MPC)

El control predictivo de modelo (MPC por sus siglas en ingles *Model Predictive Control*) es un tipo de controlador que hace uso explícito de un tipo de planta, capaz de anticiparse a los eventos futuros que puede pasar el sistema. Esta capacidad de anticiparse a los eventos respeta las limitantes que pudiera tener el sistema y sobre todo, permite que las salidas y las entradas de una planta sean independientes.

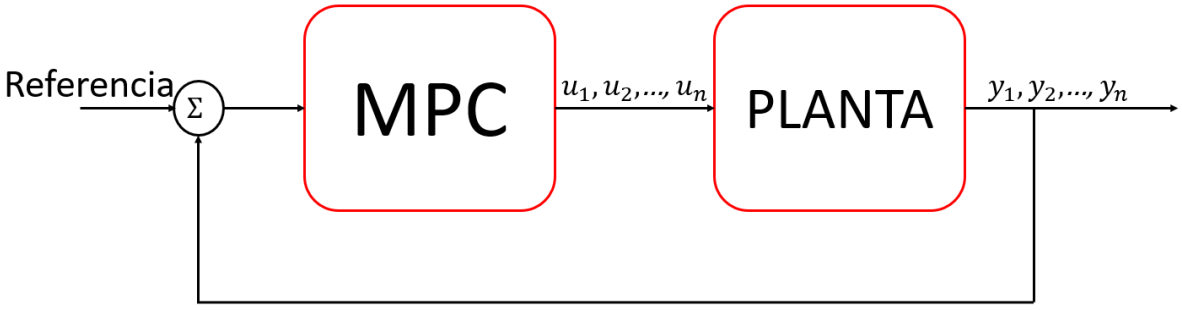

Figura 1.11: Diagrama de bloques de una planta con un MPC [13].

El control predictivo de modelo dado a todas estas características que posee, requiere una gran cantidad de memoria y de procesamiento computacional ya que resuelve el sistema en ciertos instantes de tiempo (horizonte de predicción), dando a entender que el MPC trabaja de forma discreta.

Una restricción más es que necesita un modelo de la planta en dónde simula las diferentes trayectorias que el sistema pudiera optar, haciendo que de esta manera pueda escoger la mejor opción con ayuda de un optimizador. Lo antes mencionado se puede percibir a través de las imagenes 1.12 y 1.13.

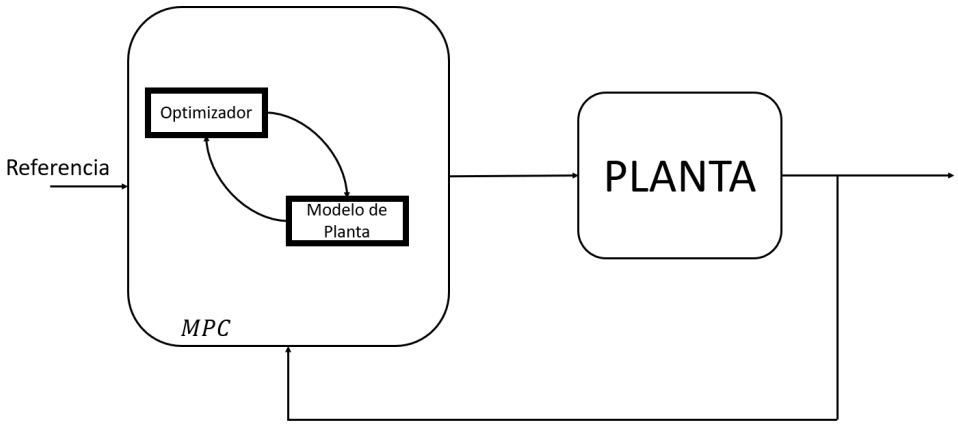

Figura 1.12: esquema de lo que conforma un controlador MPC [13].

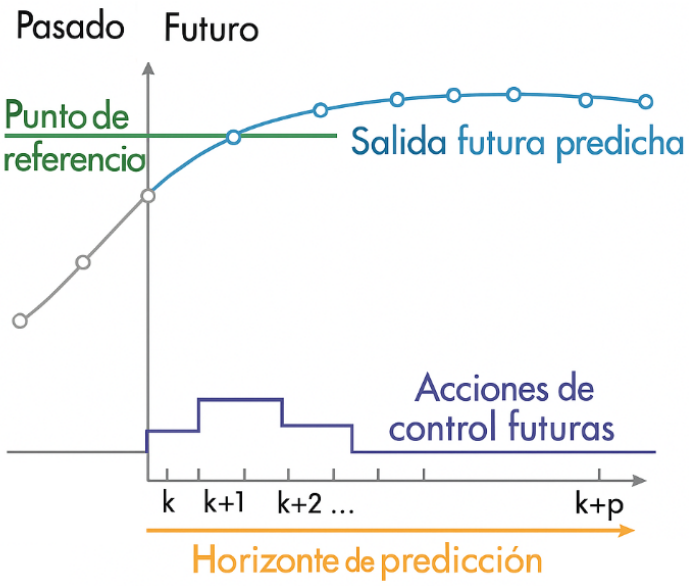

Figura 1.13: Horizonte de prediccion [13].

1.4.1. Control Predictivo de Modelo con odometría visual

El control predictivo de modelo que se usará como punto de partida en esta tesis es la propuesta por [4], el desarrollo realizado consiste en básicamente tomar la Odometría implementada en el algoritmo de Ross que nos proporciona la matriz de posición de los puntos de interés generado por la odometría visual. A su vez, se plantea un modelo dinámico del cuadrirrotor linealizado con el cual el controlador hará la predicción de la trayectoria y con ayuda del optimizador que se desarrolló durante este problema, que es básicamente el error de posición, ir trazando la trayectoria esperada.

1.5. Control tolerante a fallas (FTC)

El control tolerante a fallas (FTC por sus siglas en ingles *Fault Tolerant Control*) es un tipo de control que posee la habilidad de que cuando un componente de un sistema falla, esta se reconfigura automáticamente, permitiendo que termine la tarea que estaba desempeñando y así evitar posibles daños mayores.

Generalmente hablando, el FTC puede llegar a ser clasificado en dos tipos: los controladores pasivos PFTCS (por sus siglas en ingles *Passives Fault Tolerant Control System*) y en activos AFTCS (por sus siglas en ingles *Active Fault Tolerant Control System*). Los PFTCS son considerados como los controladores robustos. Su idea básica es hacer que el lazo cerrado del sistema sea robusto ante las incertidumbres y tiene limitación a ciertas fallas restrictivas, y por eso nos centraremos en las otra.

1.5.1. Control tolerante a fallas activo (AFTCS)

Los AFTCS (por sus siglas en ingles *Active Fault Tolerant Control System*) son controladores que consisten en ajustar o reconfigurar usando la informacion proveniente de un bloque de deteccion, teniendo la meta de mantener al menos el sistema estable. Para esto, ajustan los controladores a la falla detectada, teniendo la posibilidad de preservar el sistema en falla a el valor nominal.

Para fallas criticas, como perdida de un actuador, por ejemplo, el valor nominal no puede ser mantenido, en estos casos el sistema pasa a un modo en donde termine de realizar de manera segura, la ultima tarea que estaba realizando para evitar una perdida mayor.
Con estas consideraciones, el sistema FTC debe de cubrir las siguientes premisas:

- Considerar varios tipos de fallas (sensores, actuadores y el sistema mismo).
- dar información acerca de la falla y el desempeño deseado.
- decidir si el sistema puede seguir operando o no.

La estructura básica de los AFTCS esta compuesta de cuatro sub sistemas que se presentan a continuación seguido de un esquema en la figura 1.14:

- Un controlador retroalimentado reconfigurable, que pueda reaccionar ante las fallas cambiando algunos parámetros del controlador en el sistema entero.

- Un bloque de detección e identificación de falla. Este bloque tiene que actuar de manera rápida y precisa de acuerdo a la presencia de la falla.

- Un mecanismo de reconfiguracion, que se encarga de el mecanismo de identificación de falla con el controlador reconfigurado.

- Un planificador de trayectoria designado para evitar que el actuador se sature y ajustar las referencias después de la falla.

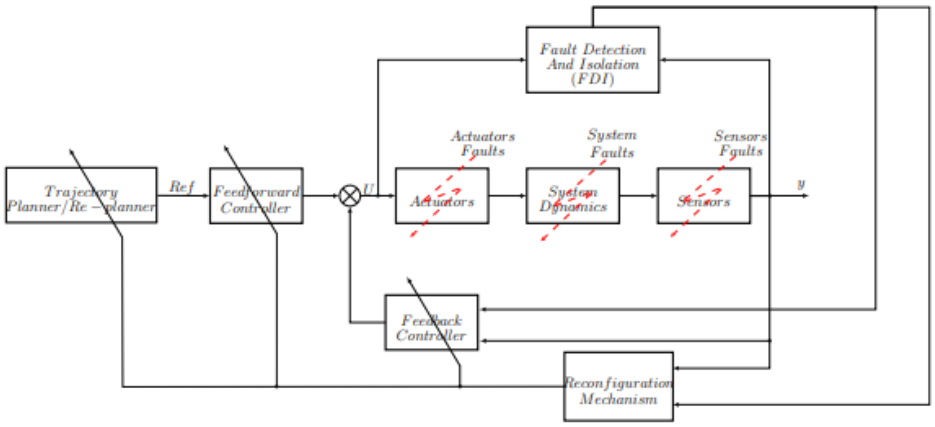

Figura 1.14: esquema del control tolerante a fallas activa [2].

1.6. Control tolerante a fallas basado en planitud diferencial

El método de control tolerante a fallas basado en planitud diferencial busca cubrir cada una de las fases necesarias para la construcción de un controlador.

Una de estas fases es la reconstrucción de una trayectoria basado en la teoría plana que se mostró con anterioridad. Sabemos que las salidas planas contienen información de varios elementos implícitos dentro de nuestra planta, el punto de usar esta teoría, es que cuando un sistema falla, encuentre las salidas planas, y como si se tratase de una ingeniería inversa, a partir de esas salidas tomar la señal que ha fallado de este sistema, y con ayuda de la matemática que implicaba el control de trayectoria con el cual la planta venia trabajando antes de su falla, reconstruir la señal que había fallado, para que el sistema pueda terminar su tarea.

Para entenderlo vamos a definir un sistema físico cualquiera que es controlado en varios parámetros, tal como se muestra en la figura 1.15:

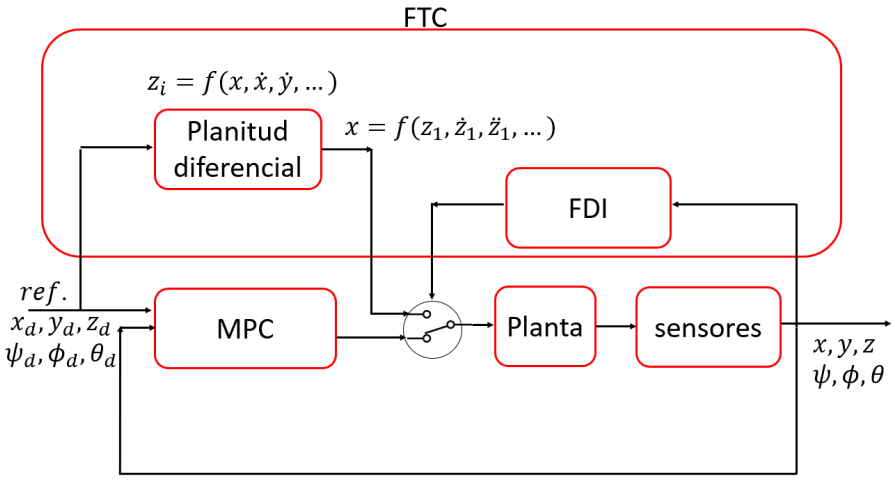

Figura 1.15: Esquema de un sistema controlado por un MPC acoplado a detección de fallas

Como se observa en la figura, el sistema cuenta con un módulo de sensado que permite obtener los parámetros correspondientes a la posición requerida. Una vez adquiridos, estos datos son enviados al sistema de control, el cual realiza las correcciones necesarias, en caso de ser requeridas.

Durante este proceso, si en algún momento se detecta una falla mediante un mecanismo de Detección y Aislamiento de Fallas (FDI, por sus siglas en inglés), se activa internamente un çonmutador"virtual. Este conmutador permite que los parámetros necesarios sean recalculados mediante el uso de planitud diferencial, lo que posibilita la continuidad de la tarea asignada, tal como se ilustra en la Figura 1.16.

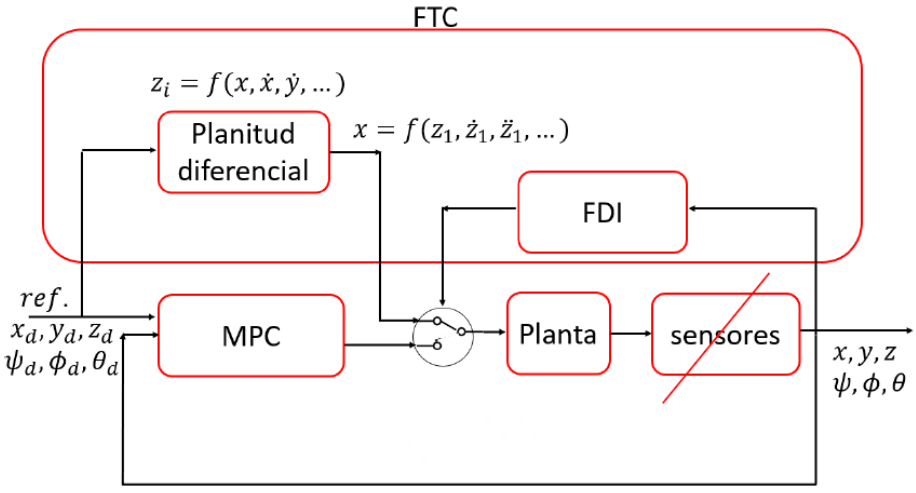

Figura 8: FTC acoplado a MPC.

Figura 1.16: Sistema con falla

Capítulo 2

Control FTC acoplado a control MPC

El interés por los sistemas de control tolerante a fallas ha aumentado considerablemente en los últimos años, lo que ha motivado su implementación en diversas áreas de aplicación. Este crecimiento se debe, en gran parte, al avance progresivo hacia la automatización completa de dispositivos, los cuales, en muchos casos, pueden verse expuestos a perturbaciones externas que comprometan su funcionamiento. En el presente capítulo, se propone una técnica que permite a un cuadrirrotor, ante la ocurrencia de una falla, reconfigurar un controlador predictivo basado en modelo, con el fin de mantener la estabilidad y seguridad del sistema, incluso bajo condiciones cambiantes.

2.1. Control Predictivo de Modelo

2.1.1. Modelo del sistema con control predictivo de modelo

El modelo del MPC (*Model Predictive Control* por sus siglas en ingles) empleado en este documento se saca de [14] y se presenta en la figura 2.1:

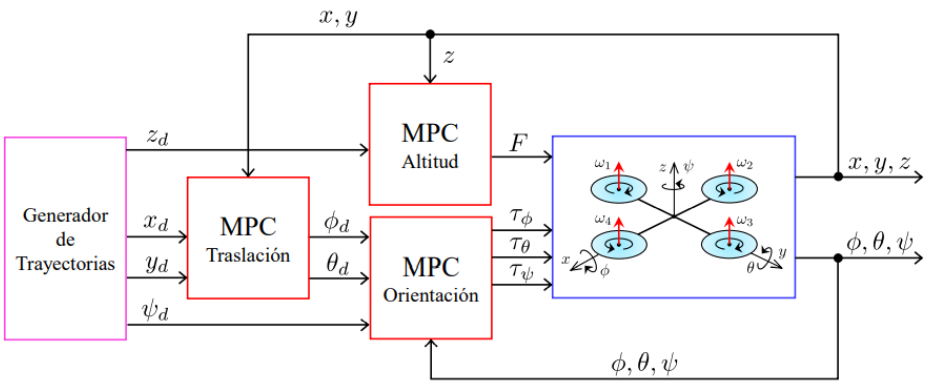

Figura 2.1: Diagrama de bloques del controlador MPC.

Para poder sacar las representaciones de los controladores, se opta por linealizar el modelo dinámico del cuadrirrotor.

2.1.2. Linealización del sistema dinámico

Para esta parte se toma en consideración que debido a que el sistema es subactuado, las coordenadas x e y se pueden calcular indirectamente con respecto a los angulos θ y ϕ respectivamente. Para la linealización consideramos que el dron se encuentra en vuelo estacionario en donde se satisfacen las siguientes condiciones:

$$\begin{gathered} \boldsymbol{\xi} = \boldsymbol{\xi_0} \\ \phi = 0 \\ \theta = 0 \\ \psi = \psi_d \end{gathered} \tag{2.1}$$

Donde ξ_0 es un vector de posición constante y ψ_d es el angulo constante respectivo deseado. Empleando la aproximación de ángulos pequeños:

$$\begin{gathered} \text{sen}\phi \approx \phi \approx 0 \\ \text{sen}\theta \approx \theta \approx 0 \\ \cos\phi \approx 1 \\ \cos\theta \approx 1 \end{gathered} \tag{2.2}$$

Con estas consideraciones la matriz de inercia$M(\boldsymbol{\eta})$ se reduce a una matriz diagonal y la matriz de fuerzas centrípetas y de Coriolis $C(\dot{\eta})$ se vuelve cero, con esto mismo se puede reescribir al siguiente modelo:

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

2.1.3. Diseño del controlador

A partir de estas ecuaciones se proponen los siguientes sistemas que representan a los controladores respectivos.

$$\frac{d}{dt}\begin{bmatrix} z \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} z \\ \dot{z} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} G \tag{2.3}$$

en donde $G = F - mg$. La salida del subsistema será la altura del vehículo z. El controlador de traslación en el espacio de estados queda definido como:

$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ g & 0 \\ 0 & 0 \\ 0 & -g \end{bmatrix}\begin{bmatrix} \theta_d \\ \phi_d \end{bmatrix} \tag{2.4}$$

En donde la entrada son los ángulos θ_d y ϕ_d y las salidas son la posición x y y. Finalmente el controlador de orientación queda definido como:

$$\frac{d}{dt}\begin{bmatrix}\phi\\ \dot{\phi}\\ \theta\\ \dot{\theta}\\ \psi\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}0&1&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&0&0\\ 0&0&0&0&0&1\\ 0&0&0&0&0&0\end{bmatrix}\begin{bmatrix}\phi\\ \dot{\phi}\\ \theta\\ \dot{\theta}\\ \psi\\ \dot{\psi}\end{bmatrix} + \begin{bmatrix}0&0&0\\ \frac{1}{I_{xx}}&0&0\\ 0&0&0\\ 0&\frac{1}{I_{yy}}&0\\ 0&0&0\\ 0&0&\frac{1}{I_{zz}}\end{bmatrix}\begin{bmatrix}\tau_\phi\\ \tau_\theta\\ \tau_\psi\end{bmatrix} \tag{2.5}$$

De donde se tiene como entrada a $\boldsymbol{\tau} = \begin{bmatrix}\tau_\phi & \tau_\theta & \tau_\psi\end{bmatrix}^T$ y como salida a $\eta = \begin{bmatrix}\phi & \theta & \psi\end{bmatrix}^T$.

2.1.4. Generación del control predictivo de modelo

El modelo en el espacio de estados puede ser realizado de diferentes maneras, tanto en el caso monovariable como en el caso multivariable y de igual manera si el sistema es no lineal. De acuerdo a [15] consideramos 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:

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

Esto da lugar a la versión aumentada del modelo original:

$$\begin{bmatrix}\Delta\boldsymbol{x}_m(k+1)\\ \boldsymbol{y}(k+1)\end{bmatrix} = \begin{bmatrix}A_m & 0_m^T\\ C_mA_m & I_q\times q\end{bmatrix}\begin{bmatrix}\Delta\boldsymbol{x}_m(k)\\ \boldsymbol{y}(k)\end{bmatrix} + \begin{bmatrix}B_m\\ C_mB_m\end{bmatrix}\Delta\boldsymbol{u}(k) \tag{2.6}$$

$$\boldsymbol{y}(k) = \begin{bmatrix}0_m & I_{q\times q}\end{bmatrix}\begin{bmatrix}\Delta\boldsymbol{x}_m(k)\\ \boldsymbol{y}(k)\end{bmatrix} \tag{2.7}$$

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

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

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

2.1.5. Modelo de predicción

En base al modelo en el espacio de estados, las variables de salida futura se calculan mediante la siguiente expresión [15]:

$$\hat{\boldsymbol{y}}(k+j|k) = CA^{j}\boldsymbol{x}(k) + \sum_{i=0}^{j-1} CA^{j-i-1}B\Delta\boldsymbol{u}(k+i) \tag{2.8}$$

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

$$\hat{\boldsymbol{Y}} = F\boldsymbol{x}(k) + \Phi\Delta\boldsymbol{U} \tag{2.9}$$

donde:

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

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

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 $N_c \leq N_p$.

2.1.6. Ley de control

El objetivo del MPC consiste en llevar la señal de salida predicha lo mas cercana posible al valor de referencia que se propone 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.

$$\boldsymbol{R}_s^T = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} r(k)$$

Se define la función de costo:

$$J = (\boldsymbol{R}_s - \hat{\boldsymbol{Y}})^T \bar{Q} (\boldsymbol{R}_s - \hat{\boldsymbol{Y}}) + \Delta\boldsymbol{U}^T \bar{R} \Delta\boldsymbol{U} \tag{2.10}$$

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

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

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 dentro de la función de costo, derivando e igualando a cero se obtiene la secuencia de control óptima.

$$\Delta \boldsymbol{U} = [\Phi^T \bar{Q} \Phi + \bar{R}]^{-1} \Phi^T \bar{Q} [\boldsymbol{R}_s - F\boldsymbol{x}(k)] \tag{2.11}$$

Considerando que:

$$\boldsymbol{R}_s^T = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} r(k) = \hat{\boldsymbol{R}}_s r(k)$$

se tiene

$$\Delta \boldsymbol{U} = [\Phi^T \bar{Q} \Phi + \bar{R}]^{-1} \Phi^T \bar{Q} [\hat{\boldsymbol{R}}_s r(k) - F\boldsymbol{x}(k)] \tag{2.12}$$

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

2.2. Control tolerante a fallas

2.2.1. Cálculo de salidas planas

Para el cálculo de las salidas planas que definirán las trayectorias en el caso de que el sistema entre en un estado de falla, se utilizará la metodología de salidas planas, las cuales se presenta en [16].Para ello tenemos que partir del sistema dinámico en su configuración no lineal, esto para hacer que la trayectoria sea lo mas cercana posible a la deseada, ya que en el modelo simplificado, hay consideraciones que en el proceso pasan omitidos, el modelo dinámico para esta parte es tomado de [17] ya que la solución del sistema dinámico que se propone aquí es mas cómodo para calcular las salidas planas. Partimos con que la matriz de Rotación obtenida en 1.15 cambia a la forma:

$$R = \begin{bmatrix} \cos\theta\cos\psi & \text{sen}\psi\cos\theta & -\text{sen}\theta \\ \cos\psi\text{sen}\theta\text{sen}\phi - \text{sen}\psi\cos\phi & \text{sen}\psi\text{sen}\theta\text{sen}\phi + \cos\psi\cos\phi & \cos\theta\text{sen}\phi \\ \cos\psi\text{sen}\theta\text{sen}\phi + \text{sen}\psi\cos\phi & \text{sen}\psi\text{sen}\theta\text{sen}\phi - \cos\psi\cos\phi & \cos\theta\cos\phi \end{bmatrix} \tag{2.13}$$

con esta sustitución obtenemos que el sistema dinámico de traslación se traduce en la siguiente forma:

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = F \begin{bmatrix} -\text{sen}\theta \\ \cos\theta\text{sen}\phi \\ \cos\theta\cos\phi \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \tag{2.14}$$

Considerando que las entradas de control son $u_1 = F, u_2 = \tau_\phi, u_3 = \tau_\theta$ y $u_4 = \tau_\psi$ obtenemos las siguientes igualdades:

$$\begin{aligned} m\ddot{x} &= -u_1\text{sen}\theta & u_2 &= \ddot{\phi} I_{xx} \\ m\ddot{y} &= u_1\cos\theta\text{sen}\phi & u_3 &= \ddot{\theta} I_{yy} \\ m\ddot{z} &= u_1\cos\theta\cos\phi - mg & u_4 &= \ddot{\psi} I_{zz} \end{aligned} \tag{2.15}$$

Con esto presente tenemos que obtener todos los estados y todas las señales de control en función de salidas planas y un numero finito de sus derivadas, de la ecuación anterior y definiendo nuestras salidas planas como: $z\alpha = \begin{bmatrix} x & y & z & \psi \end{bmatrix}^T$ obtenemos lo siguiente:

$$\begin{aligned} x &= z_1 & \dot{x} &= \dot{z}_1 \\ y &= z_2 & \dot{y} &= \dot{z}_2 \\ z &= z_3 & \dot{z} &= \dot{z}_3 \\ \psi &= z_4 & \dot{\psi} &= \dot{z}_4 \\ \theta &= \arcsin\left(\frac{m\ddot{z}_1}{-u_1}\right) & \dot{\theta} &= -m\left(\frac{\dddot{z}_1 u_1 - u_1^2 \ddot{z}_1}{u_1^2\sqrt{A}}\right) \\ \phi &= \arctan\left(\frac{\ddot{z}_2}{\ddot{z}_3 + g}\right) & \dot{\phi} &= \frac{\dddot{z}_2(\ddot{z}_3 + g) - \dddot{z}_3\ddot{z}_2}{(\ddot{z}_3 + g)^2 + {\ddot{z}_2}^2} \end{aligned}$$

Donde:

$$A = 1 - \left(\frac{m\ddot{z}_1}{u_1}\right)^2 \tag{2.16}$$

De aquí, para obtener u_1 partimos de sumar las igualdades de la ecuación 2.14 tal como se muestra a continuación:

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 = (-u_1 \text{sen}\theta)^2 + (u_1 \cos\theta \text{sen}\phi)^2 + (u_1 \cos\theta \cos\phi - mg)^2$$

Haciendo las operaciones necesarias que se muestran a detalle en el Anexo 1, obtenemos que u_1:

$$m\sqrt{(\ddot{x})^2 + (\ddot{y})^2 + (\ddot{z})^2 + 2g\ddot{z} + (g)^2} = u_1$$

Y finalmente dejando los términos en función de las salidas planas, obtendríamos:

$$u_1 = m\sqrt{(\ddot{z}_1)^2 + (\ddot{z}_2)^2 + (\ddot{z}_3)^2 + 2g\ddot{z}_3 + (g)^2} \tag{2.17}$$

De igual manera calculando los valores de u_2, u_3 y u_4 a través de realizar las derivadas necesarias, obtenemos que:

$$u_2 = \ddot{z}_4 \tag{2.18}$$

$$u_3 = -m\left(\frac{\left((\dddot{z}_1 u_1 - \ddot{u}_1\ddot{z}_1)\, u_1^2\sqrt{A}\right) - (\ddot{z}_1 u_1 - \ddot{u}_1\ddot{z}_1)C + 2u_1\dot{u}_1\sqrt{A}}{u_1^4 A}\right) \tag{2.19}$$

$$u_4 = \frac{(\dddot{z}_2(\ddot{z}_3 + g) + \ddot{z}_2\ddot{z}_3 - \dddot{z}_3\ddot{z}_2 - \ddot{z}_3\ddot{z}_2)\,(\ddot{z}_2(\ddot{z}_3 + g) - \ddot{z}\ddot{z}_2) - B}{(\ddot{z}_3 + g)^4 + (\ddot{z}_2)^4} \tag{2.20}$$

en donde:

$$B = (2(\ddot{z}_3 + g)\ddot{z}_3 + 2\ddot{z}_2\ddot{z}_2)\,(\ddot{z}_2(\ddot{z}_3 + g) - \ddot{z}_3\ddot{z}_2) \tag{2.21}$$

$$C = \left(\frac{-2m\ddot{z}_1\,(m\dddot{z}_1 u_1 - \dot{u}_1 m\ddot{z}_1)}{u_1\sqrt{A}}\right) \tag{2.22}$$

2.3. Simulaciones

Para simular el proceso se tiene que tomar la versión aumentado del sistema dinámico, seguido de su modelo de predicción y finalmente la ley de control. Para esta simulación realizada en el entorno computacional MATLAB se toman las especificaciones del cuadrirrotor obtenidas de [18] y que se presentan a continuación en la siguiente tabla:

Parámetro	Valor	Unidades
m	0.5	kg
g	9.81	m/s^2
l	0.12905	m
I_{xx}	0.00389	$kg \times m^2$
I_{yy}	0.00389	$kg \times m^2$
I_{zz}	0.00780	$kg \times m^2$
k_T	8.5485×10^{-6}	$kg \times m$
k_D	0.016	m

Tabla 2.1: Parámetros del sistema.

De igual manera tomamos los parámetros proporcionados por [14], que se presentan a continuación:

Parámetro	Altitud	Posición	Orientación
T_s	0.05	0.05	0.005
N_p	10	10	12
N_c	10	10	12
Q	20	20	15
R	0.01	0.01	0.1

Tabla 2.2: Parámetros del controlador MPC.

De ahí, la trayectoria para esta parte viene descrita por las ecuaciones:

$$x_d = 2\cos(0.2t) \qquad y_d = 2\text{sen}(0.2t) \qquad z_d = 0.5t \qquad \psi_d = \pi$$

La simulación de la trayectoria calculada, en comparación con la trayectoria real, queda de la siguiente manera:

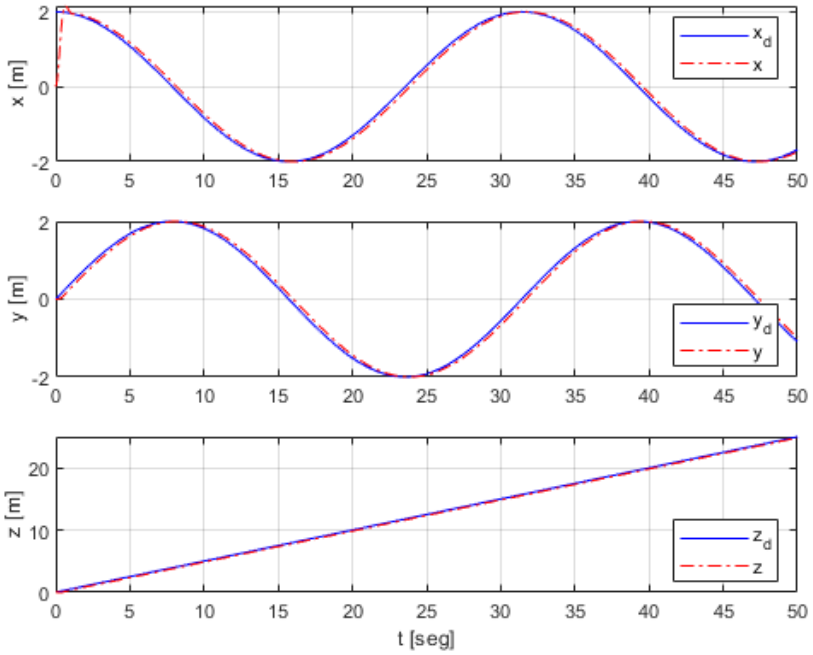

Figura 2.2: Trayectorias correspondientes a x, y, z.

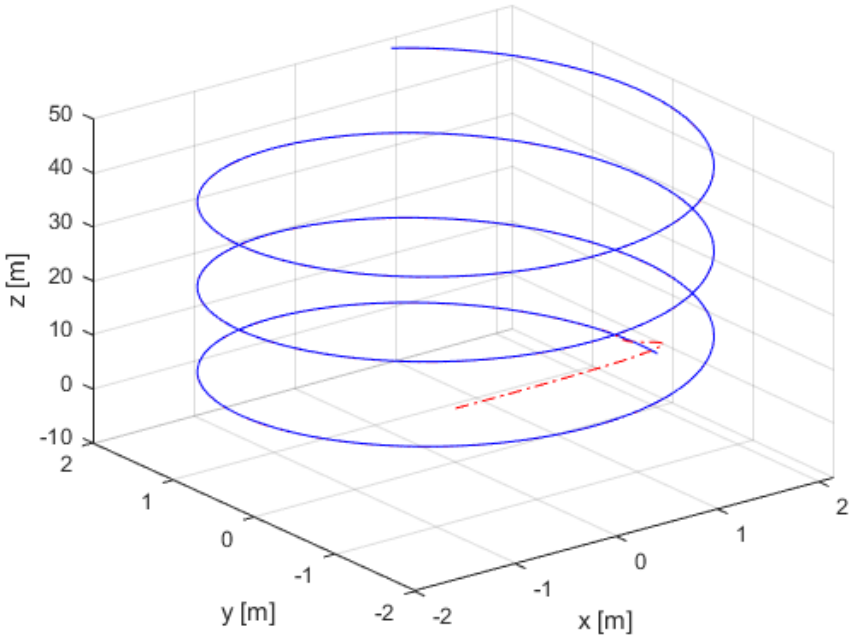

Figura 2.3: Trayectoria recorrida por el cuadrirrotor.

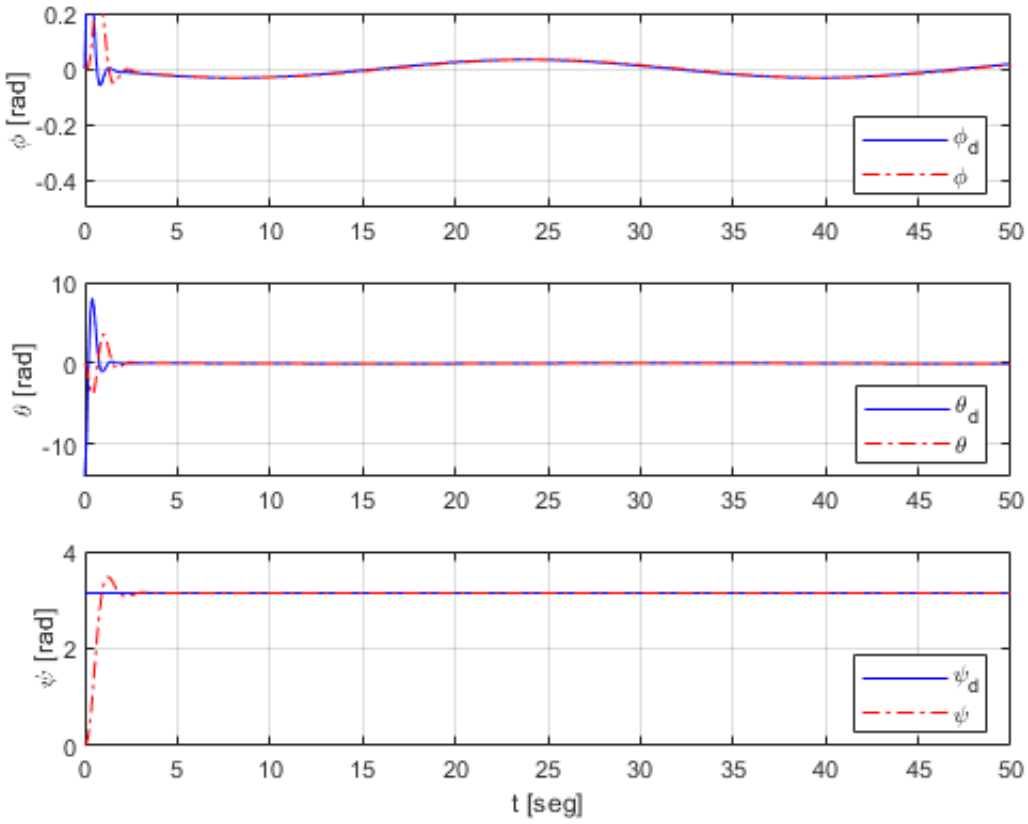

Figura 2.4: Trayectorias correspondientes a ϕ, θ, ψ.

En las gráficas anteriores se puede observar que, gracias a los datos proporcionados en la tabla precedente, la señal de salida del sistema presenta un comportamiento acorde con lo esperado, lo cual respalda la viabilidad del modelo propuesto.

Diversos estudios en la literatura han demostrado que, con una adecuada elección de los parámetros Q, R y N, el desempeño de un controlador predictivo basado en modelo (MPC) puede ser altamente satisfactorio. En este trabajo, la elección de dichos parámetros ha sido guiada tanto por estas investigaciones como por el análisis de la señal utilizada, la cual no presenta variaciones abruptas. Esta característica permite adoptar un enfoque más conservador en la selección de los valores, sin comprometer la estabilidad ni la precisión del sistema.

No obstante, para comprender a profundidad el funcionamiento interno del sistema, es necesario analizar el papel que desempeñan las variables Q, R y N dentro de la estructura del controlador.

Partimos con el ajuste del valor de R, si le damos un valor de 0.1, obtenemos:

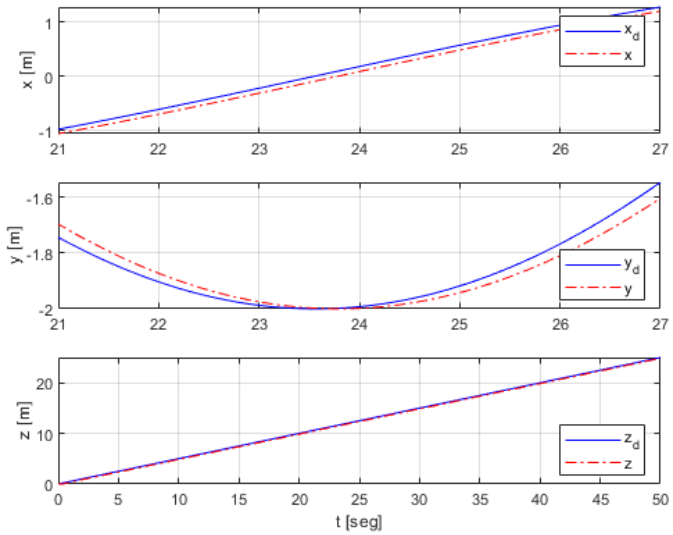

Figura 2.5: Trayectorias con $R = .1$.

igualmente damos el valor de R como 1, y obtenemos:

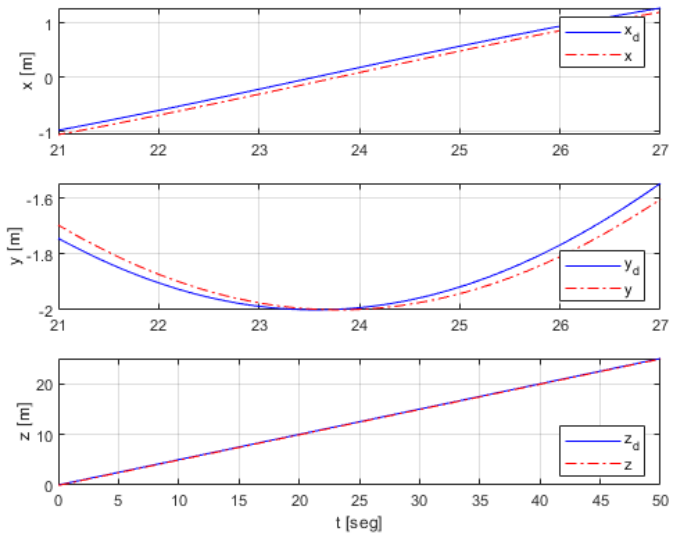

Figura 2.6: Trayectorias con $R = 1$.

Visualmente no se nota un cambio notable de un ejemplo a otro. esto es porque la variable R y esto es debido a que esta variable penaliza el tamaño o variacion de la señal de control, Esto indica que una R alta significa que el controlador es mas conservador con los cambios de entrada (menos agresivo) dicho de otra manera, con una R alta controlas que el proceso no tenga cambios bruscos en el sistema, y como la señal que se esta simulando no tiene cambios notables bruscos, podemos ser un poco conservadores en cuanto al valor de R.

También comprobamos el valor de Q con 20, y obtenemos:

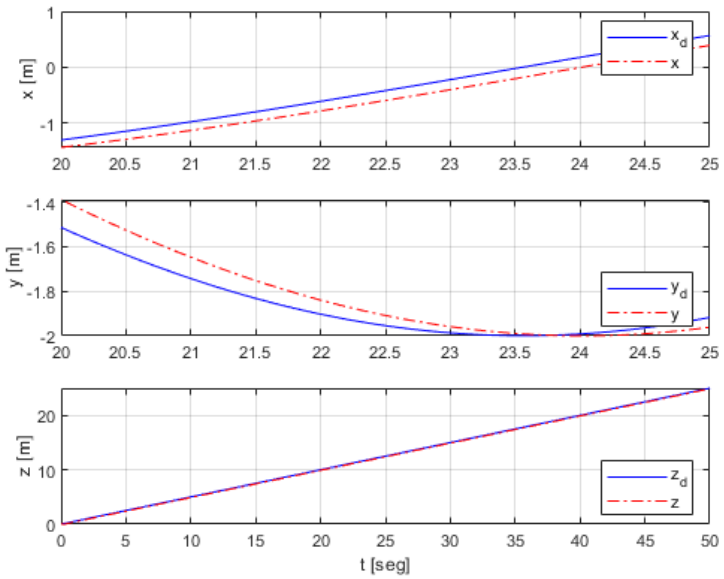

Figura 2.7: Trayectorias con $Q = 20$.

igualmente damos el valor de Q como 200, y obtenemos:

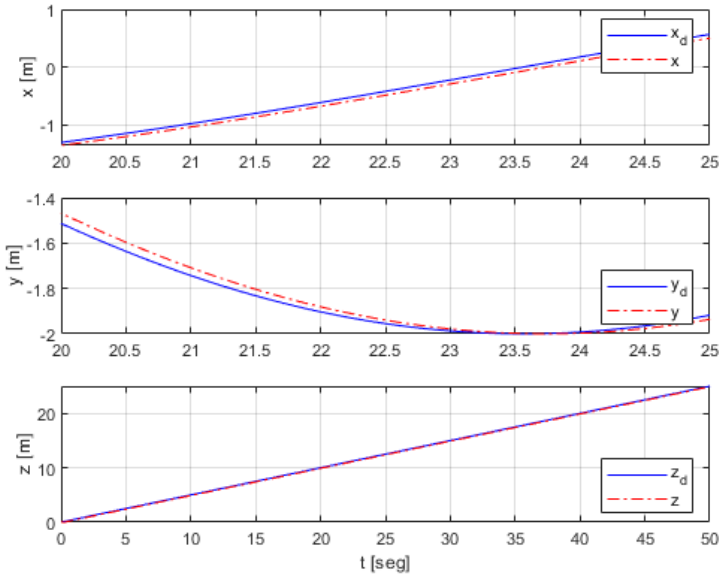

Figura 2.8: Trayectorias con $Q = 200$.

En este apartado es posible observar con mayor claridad la influencia del parámetro Q en el

comportamiento del controlador. Se aprecia que, al incrementar su valor, el controlador realiza un mayor esfuerzo por minimizar la diferencia entre la salida predicha y la referencia deseada. Esto se debe a que Q actúa como un factor de penalización sobre dicho error, incentivando un seguimiento más preciso de la trayectoria de referencia.

Finalmente, se observa la influencia del parámetro N en el desempeño del sistema. Para ello, se le asignó un valor de 1 y se efectuó una simulación, cuyos resultados se presentan en la figura siguiente.

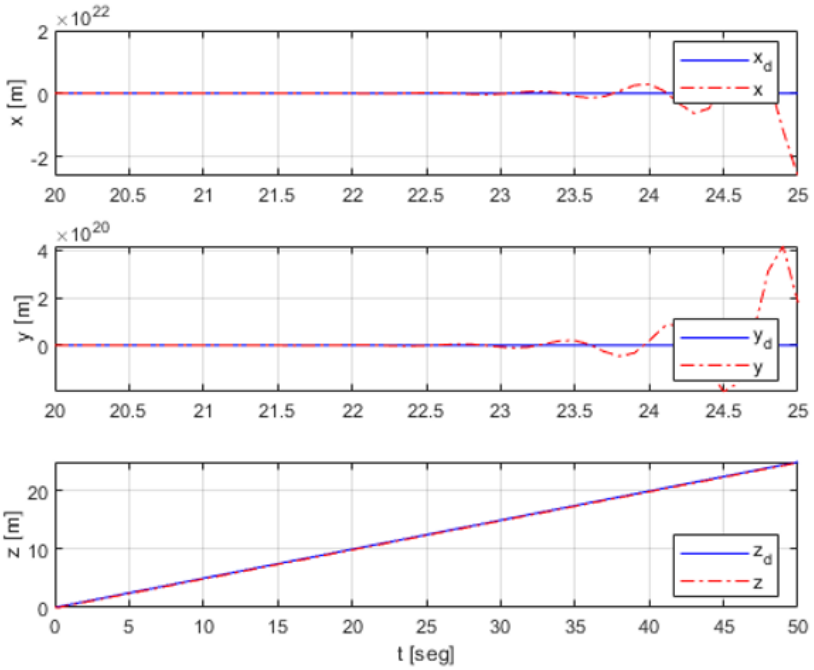

Figura 2.9: Trayectorias con $N = 1$.

En la imagen anterior se observa un cambio significativo en el comportamiento del sistema, lo que evidencia la influencia del parámetro N. Recordemos que N representa el número de pasos hacia adelante en el tiempo que se utilizan para predecir la salida del sistema. Esto implica que, a mayor valor de N, el controlador tiene una mayor capacidad de anticipar el comportamiento futuro del sistema.

Cuando se asigna un valor de $N = 1$, el horizonte de predicción es muy limitado, lo cual impide al controlador estimar adecuadamente las trayectorias futuras. Como resultado, el desempeño del sistema se ve comprometido. Por esta razón, es necesario utilizar valores de N mayores o iguales a 2 para que el controlador pueda anticiparse eficazmente a los cambios y generar una respuesta adecuada.

A partir de los resultados obtenidos en las simulaciones anteriores, se puede concluir que, en general, un incremento en los valores de los parámetros considerados tiende a mejorar el desempeño del controlador. No obstante, en este trabajo se ha optado por mantener los valores previamente seleccionados, adoptando un enfoque más conservador. Esta decisión busca alcanzar un equilibrio adecuado entre desempeño, estabilidad y robustez, garantizando así resultados satisfactorios sin comprometer la integridad del sistema.

2.3.1. Simulación con salidas planas

De igual manera, una vez obteniendo estos resultados en cuanto al control MPC, se simulan los resultados correspondientes a las salidas planas, para ello, de los valores de x_d, y_d, ψ_d y z_d se calculan los valores de θ y ϕ, para ello se tiene que derivar a través del entorno computacional MATLAB los valores requeridos para obtener el cálculo de los mismos, con todo esto, obtenemos los siguientes resultados:

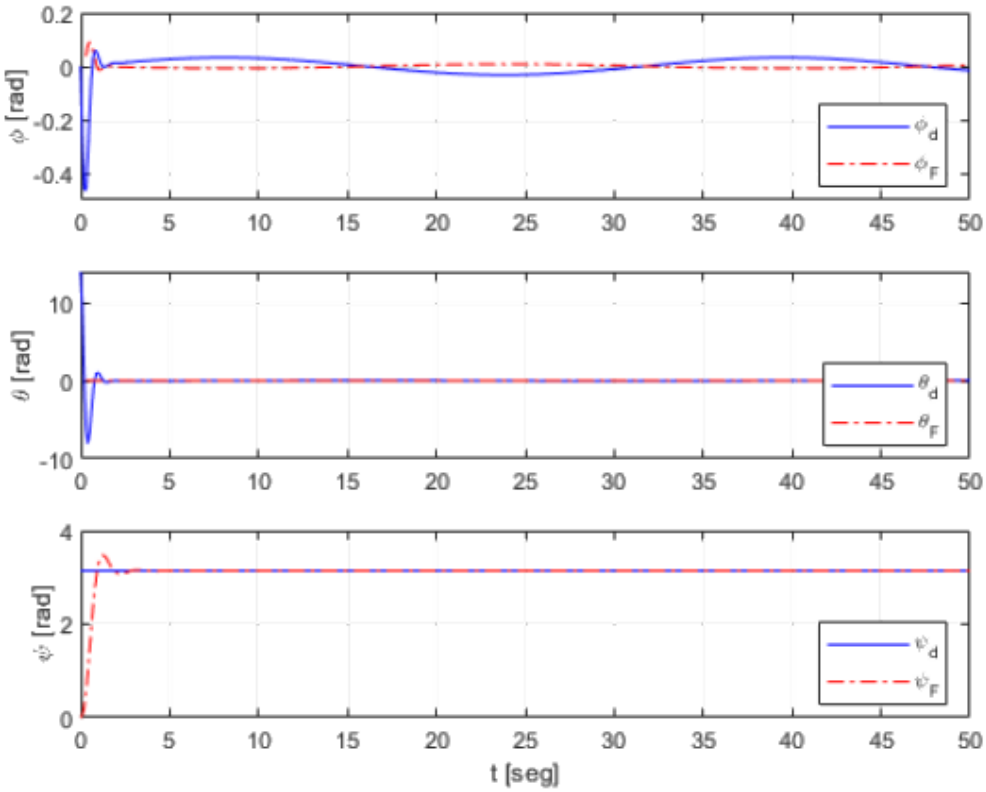

Figura 2.10: Calculo de posiciones angulares.

En las graficas podemos observar un desfase de 90 esto puede ser debido a que el cálculo de las salidas planas fueron realizadas y planteadas de forma diferente a como se desarrolló en un principio, sin embargo pareciera que los errores se mantienen mínimos, pero no son del todo satisfactorios, dado a que una variación mínima en algún ángulo del sistema, repercute de gran manera en la traslación del sistema. Otra razón por la cual pudiera presentarse este tipo de problemas, es que durante el cálculo de las derivadas, llegó a presentar un problema en cuanto la obtención de las soluciones, dado a que el método que proporciona MATLAB para obtener las derivadas, no está del todo optimizado. Al final que tanto pueda ser beneficioso para el sistema la sustitución de este control en el caso de Falla, dependerá a como se evalúe el sistema mas adelante en cuanto al nivel de rapidez que pueda llegar a requerir una respuesta para seguir realizando la tarea que uno ha planteado.

Parte de las soluciones fue tomar los sistemas dinámicos del sistema principal y del empleado en el control tolerante a fallas y buscar las similitudes y diferencias entre ellas. Finalmente después del analisis, se encontró que la matriz de rotación empleada en cada parte, impactaba en un signo en especifico. haciendo los cambios necesarios, finalmente obtenemos el resultado siguiente:

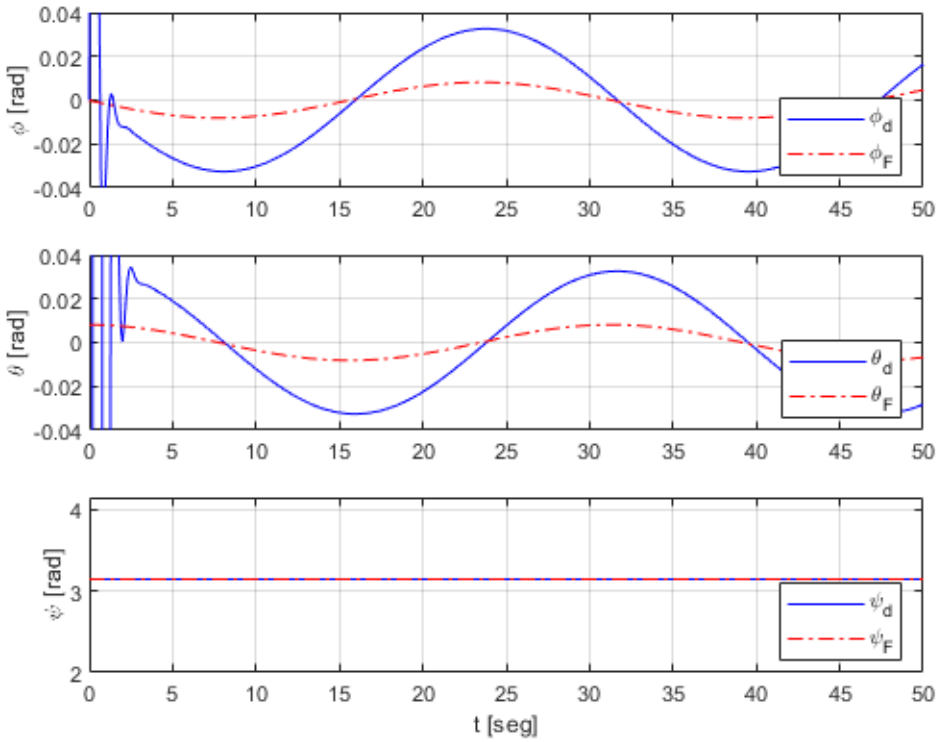

Figura 2.11: Posiciones angulares con corrección realizada.

En estas gráficas mostradas, el desfase ya está arreglado a excepción de los primeros instantes de tiempo, que es donde se descarta debido a que en las salidas planas no se toma en consideración las condiciones iniciales, sin embargo la amplitud es mas baja en los casos angulares de θ y de ϕ, en el caso de ψ, el resultado es similar a lo esperado, lo que nos puede garantizar que al menos en la altitud del sistema, este puede seguir funcionando de manera correcta ante un evento de falla, pero esto lo podremos visualizar simulando cada posible escenario.

2.4. Simulación de fallas en el cuadrirrotor

Una vez que se logran tener un resultado mas cercano a la trayectoria angular deseada, lo siguiente es observar como las salidas planas obtenidas, afectan a el sistema. para ello se hace una reestructuración del diagrama planteado en 2.1, quedando nuevamente como se muestra en la imagen siguiente:

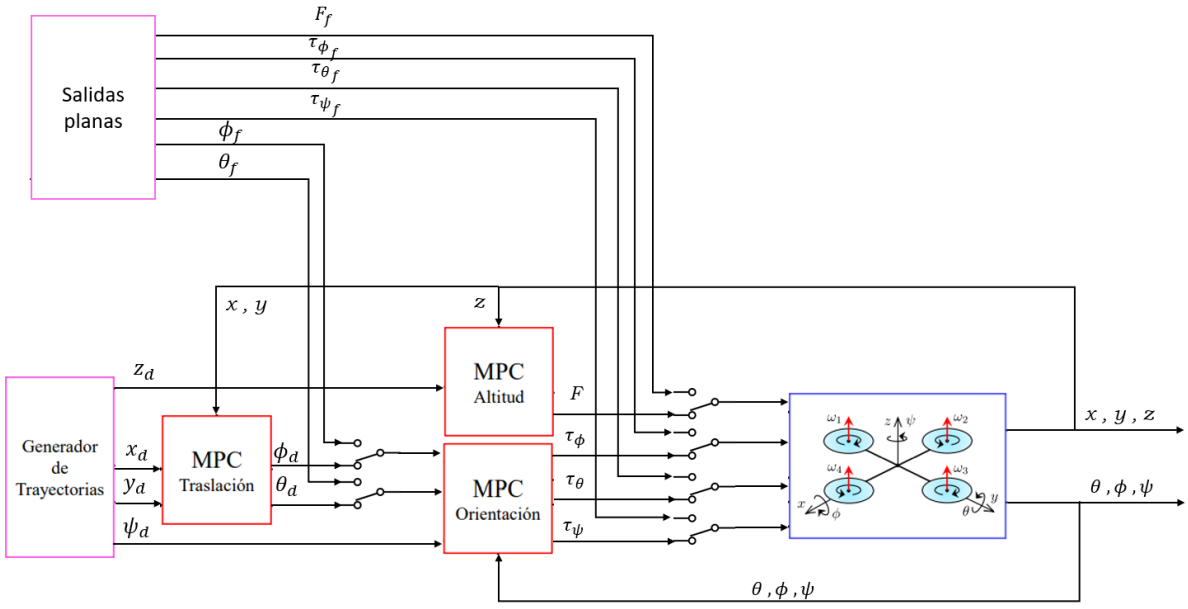

Figura 2.12: Acoplamiento del MPC con el FTC a través de las señales de operación del dron.

El enfoque principal de este trabajo se basa en la implementación de un sistema de identificación de fallas que permite realizar un *switch* virtual, conmutando de las señales generadas por el controlador predictivo basado en modelo (MPC) a aquellas obtenidas mediante las salidas planas.

Un aspecto notable e interesante de esta propuesta es que, como se mencionó previamente, los controladores están diseñados de forma desacoplada, al igual que sus respectivas salidas planas. Esta característica permite que, ante la falla de uno de los subsistemas, el impacto no se propague de manera directa a las demás variables controladas. La única excepción a esta independencia se da entre las variables de orientación y traslación, las cuales presentan una conexión directa debido a su relación física inherente en la dinámica del sistema.

2.4.1. Falla en MPC altitud

Este caso consiste en que el sensor de altitud z falle y por consecuencia el control tolerante a fallas pueda proporcionar a través de las salidas planas, la información para que el cuadrirrotor pueda terminar la tarea empleada.

Si el sensor de altitud z falla el MPC ya no podrá proporcionar la variable F, y por lo tanto las salidas planas proporcionan la variable F_f obteniendo las graficas que se presentan a continuación:

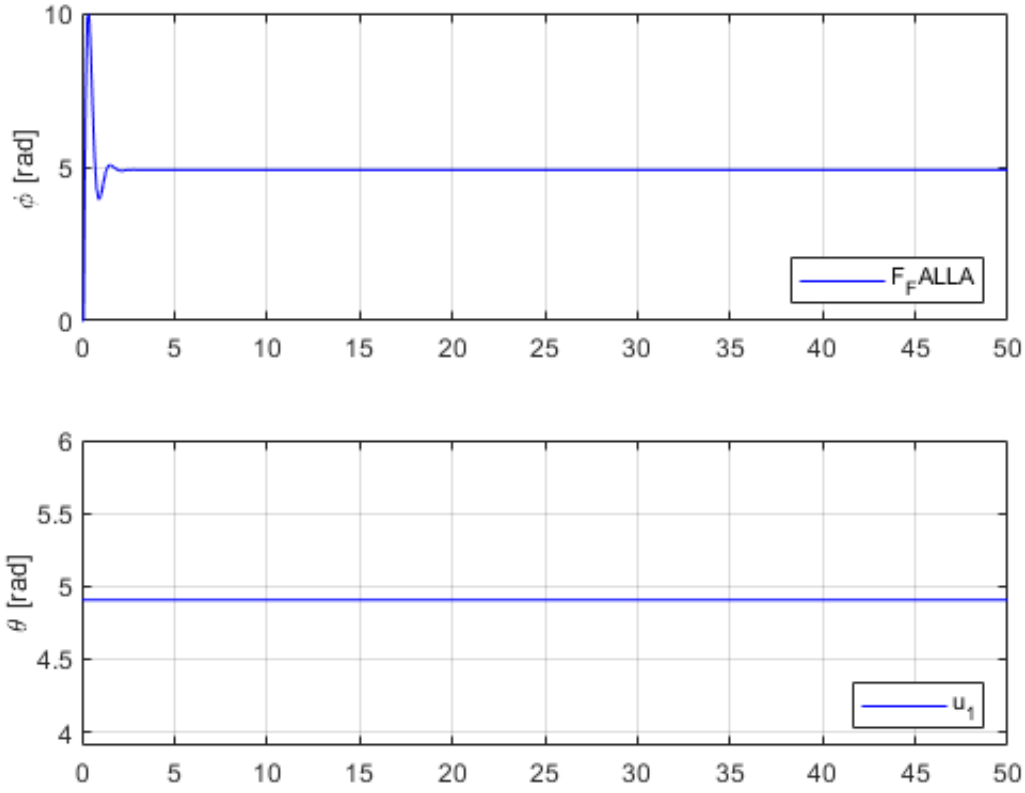

Figura 2.13: Comparación entre la variable F obtenida por salidas planas y del sistema MPC.

Se define que en un instante $k = 80$ se realice el *switch* virtual y con esto en consideración se obtiene las siguientes graficas que visualmente nos dan la trayectoria del dron ante este evento de falla en especifico:

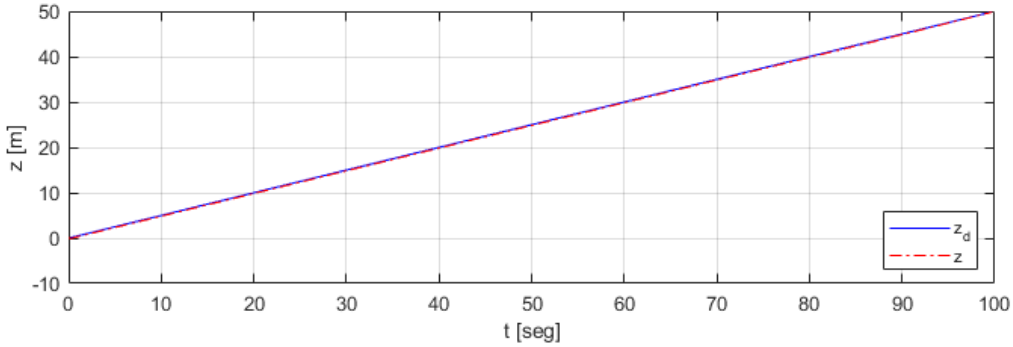

Figura 2.14: Altitud con falla en el instante 80.

Podemos observar en la grafica anterior, que los valores de la altitud no cambian en gran

medida, haciendo que las salidas planas funcionen como esperamos en este ámbito, con esto podemos hacer una visualización en el plano 3D como se muestra a continuación:

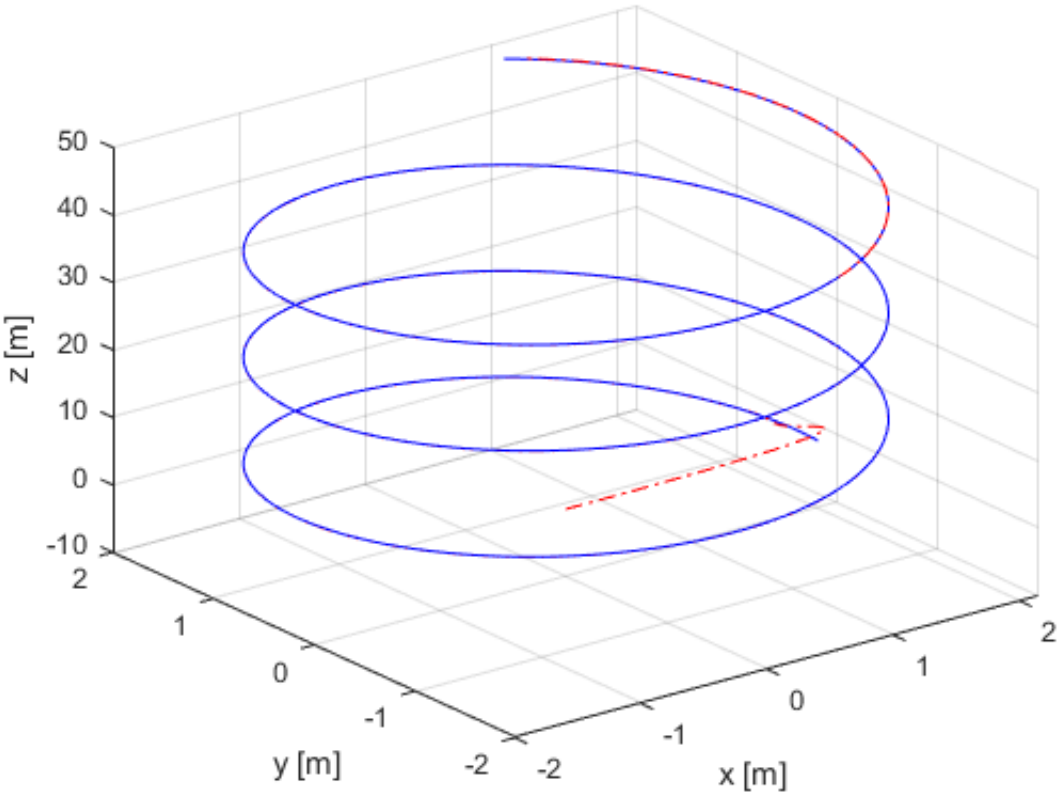

Figura 2.15: Altitud con falla mostrado en el plano 3D.

Al observar la figura anterior, se puede apreciar que este tipo de falla no genera un error significativo, lo cual indica un buen desempeño del sistema. Durante el proceso de obtención de estos resultados, surgió la posibilidad de que, si el sistema presentara una falla más grave —exceptuando el sensor en cuestión—, se podría considerar esta alternativa como una forma de proteger al dron. En ese caso, sería viable realizar un aterrizaje controlado utilizando únicamente las salidas planas relacionadas con la altitud. En principio, esta estrategia demuestra ser efectiva para el control de altura, y el siguiente paso será comprobar si también resulta útil para el control de la posición.

2.4.2. Falla en MPC posición

Este caso considera una falla en los sensores de posición x y y, lo que activa el funcionamiento del esquema de control tolerante a fallas. Al igual que en el caso anterior, dicho esquema utiliza las salidas planas para estimar y proporcionar la información necesaria que permita completar la trayectoria del vehículo.

En particular, si falla el sensor de posición en el eje x, el controlador predictivo basado en modelo (MPC) pierde la capacidad de generar la variable θ, por lo que esta es reemplazada por la estimación proporcionada mediante la salida plana, denotada como θ_f. De manera análoga, si falla el sensor de posición en el eje y, el MPC no puede calcular la variable ϕ, siendo esta suplida por la correspondiente salida plana, ϕ_f. Los resultados de esta compensación pueden observarse en las gráficas que se presentan a continuación.

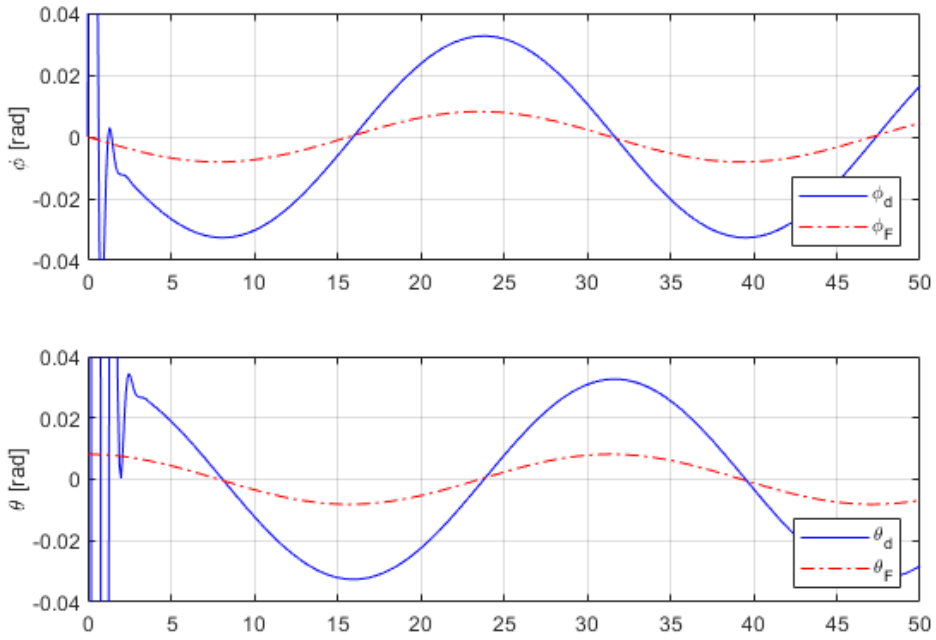

Figura 2.16: Comparación entre la variable θ y ϕ obtenida por salidas planas y del sistema MPC.

Se define que en un instante $k = 80$ se realice el *switch* virtual y se define que el sensor de posición falle en el ámbito x, obteniendo la siguiente grafica:

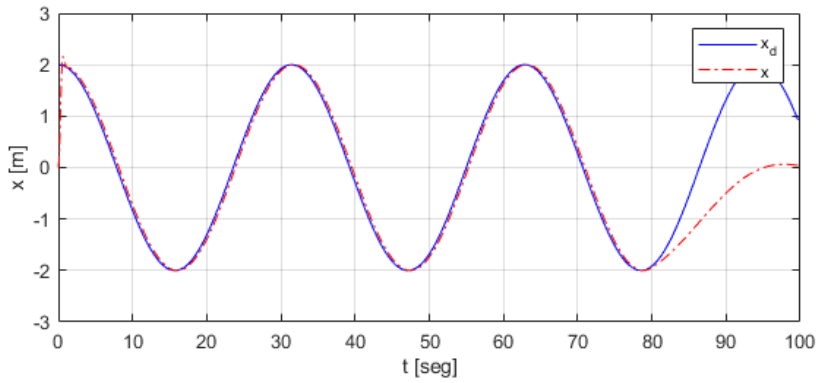

Figura 2.17: Posición x con falla en el instante 80.

En el caso de y donde simulamos los mismos parámetros, podemos observar lo siguiente:

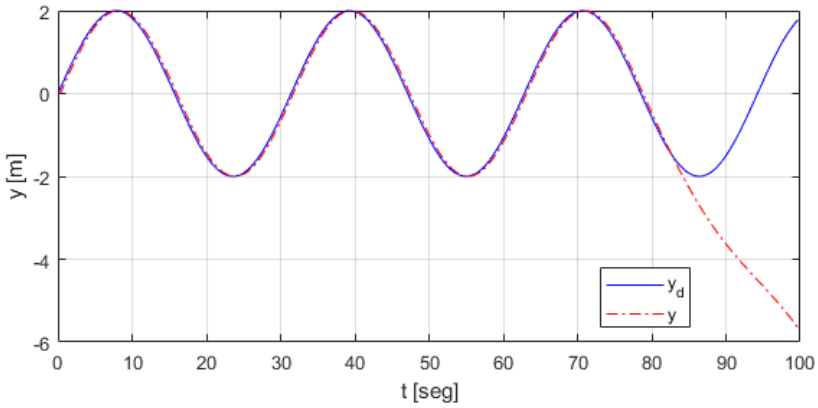

Figura 2.18: Posición y con falla en el instante 80.

En estos escenarios, se observa que las posiciones experimentan variaciones significativas, evidenciando errores apreciables en las trayectorias. Esto sugiere que, al menos de forma preliminar, el método no resulta completamente fiable para estos casos específicos. A continuación, se presenta el caso particular en el que ocurre una falla en el sensor de posición asociado al eje x visto en un plano tridimensional:

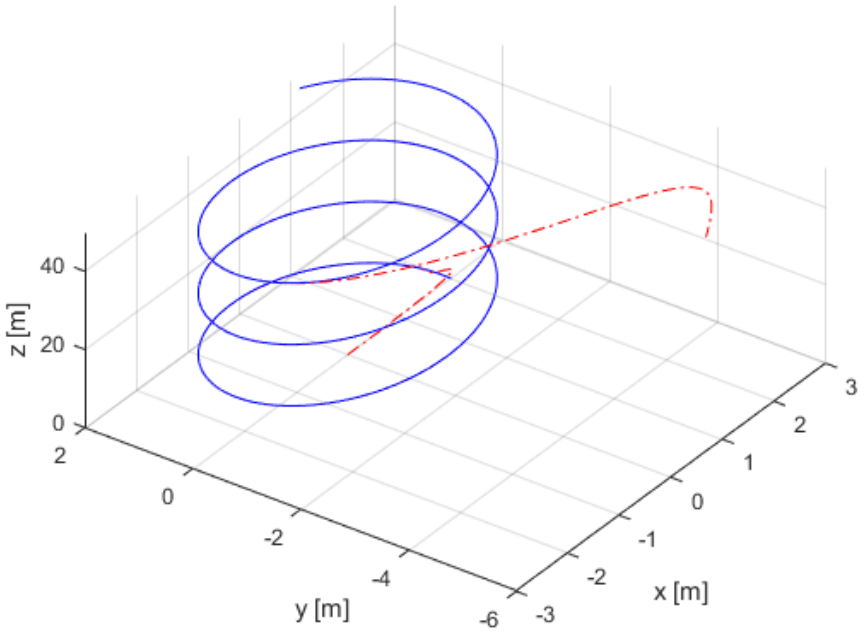

Figura 2.19: Posición x con falla en el instante 80. mostrado en el plano 3D.

de igual manera, observamos el momento en donde y falla:

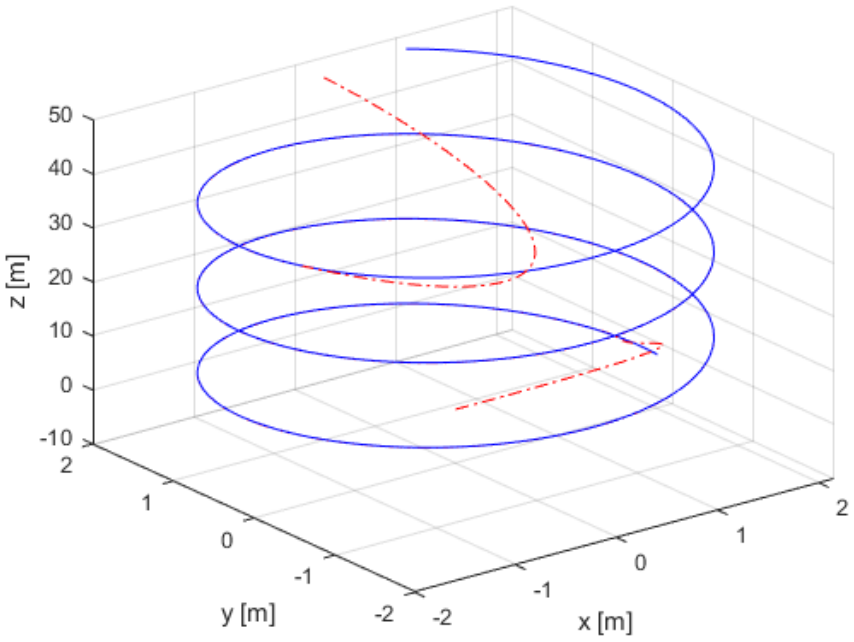

Figura 2.20: Posición y con falla en el instante 80. mostrado en el plano 3D.

En términos generales, el control tolerante a fallas aplicado a la posición no cumple con los requisitos necesarios para seguir adecuadamente la trayectoria deseada. Es importante destacar que los rangos de error observados son considerablemente altos, lo que, al menos por el momento, hace que este enfoque no sea viable. Visualmente, puede apreciarse que, en el caso del eje x, la trayectoria resultante ni siquiera mantiene la dirección esperada, mientras que en el eje y, aunque la trayectoria parece conservar una forma similar, el error sigue siendo significativo.

Un aspecto interesante observado durante estos experimentos es la alta sensibilidad de los drones a pequeñas variaciones en la orientación. Incluso desviaciones mínimas en el ángulo de orientación pueden provocar cambios sustanciales en la trayectoria, lo que resalta la necesidad de una alta precisión tanto en el control general como en los mecanismos de compensación ante fallas.

Finalmente, se analizará cómo el control tolerante a fallas influye directamente en la orientación del sistema, y cómo esta repercusión puede afectar el desempeño global del dron.

2.4.3. Falla en MPC orientación

Este caso contempla la falla de los sensores de orientación θ, ϕ y ψ. Ante esta condición, el sistema de control tolerante a fallas recurre a las salidas planas para generar la información necesaria que permita al cuadrirrotor completar la tarea asignada.

En particular, si falla el sensor asociado al ángulo θ, el controlador predictivo basado en modelo (MPC) pierde la capacidad de generar la señal de control τ_θ, siendo esta sustituida por

la correspondiente salida plana τ_{θ_f}. De forma análoga, en caso de falla del sensor ϕ, la señal τ_ϕ es reemplazada por τ_{ϕ_f}. Asimismo, si ocurre una falla en el sensor ψ, la señal de control τ_ψ es suplida por la salida plana τ_{ψ_f}.

Los resultados obtenidos bajo estas condiciones se muestran en las gráficas presentadas a continuación.

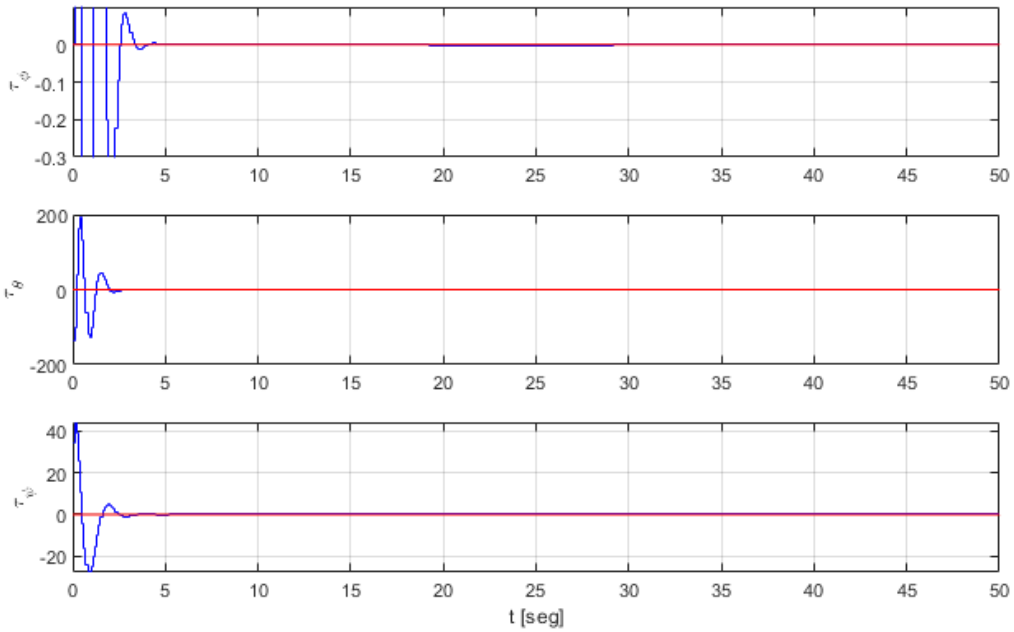

Figura 2.21: variables τ con fallas.

En este apartado, si bien los resultados obtenidos coinciden con lo esperado desde el punto de vista técnico, no se consideran plenamente viables para una implementación práctica. Esto se debe a que, al depender únicamente de las salidas planas, el sistema opera de manera equivalente a un lazo abierto, sin una retroalimentación efectiva que permita salvaguardar la integridad del dron ante perturbaciones o condiciones inesperadas.

Si bien este enfoque podría considerarse útil para generar una trayectoria de referencia, su uso como método de control directo no es recomendable. Al no proporcionar garantías suficientes de estabilidad ni de recuperación ante fallas, este escenario contradice los objetivos fundamentales de esta tesis, cuyo propósito es asegurar un control robusto y confiable que maximice la seguridad y operatividad del vehículo.

Todo el trabajo desarrollado hasta este punto permite concluir que el control tolerante a fallas basado en planitud diferencial ofrece un desempeño óptimo únicamente en dos de los tres casos evaluados: control de altitud y control de orientación. No obstante, el caso más relevante y, en cierto sentido, el de mayor utilidad práctica es el control de la posición.

Hasta el momento, se sostiene la hipótesis de que la razón por la cual el dron no logra seguir adecuadamente la trayectoria deseada radica en la falta de amplitud suficiente en las señales de control, lo cual impide alcanzar el punto objetivo. Tal como se explicó previamente, esto se

traduce en un error considerable en la ejecución de la trayectoria.

En la sección siguiente se propondrá una solución orientada a mitigar esta limitación y mejorar el desempeño del sistema en el control de posición.

2.5. Solución al desfase de trayectoria

En la sección anterior se presentó la propuesta de acoplamiento de ambas estrategias de control, aplicadas a tres casos distintos: orientación, posición y altitud. No obstante, se observó que, al producirse una falla en el caso de control de posición, el sistema no era capaz de seguir la trayectoria deseada de forma adecuada. En esta sección, se plantea una solución a dicho inconveniente con el objetivo de garantizar el seguimiento correcto de la trayectoria, mejorar el desempeño del sistema y asegurar que el dron cumpla con los requisitos mínimos exigidos por un esquema de control tolerante a fallas.

2.5.1. Analisis de trayectoria de falla

Retomando lo desarrollado hasta este punto, se identificó que al acoplar los controladores de posición dentro del esquema de control tolerante a fallas en el sistema del dron, este no lograba seguir la trayectoria deseada de manera precisa. El margen de error resultante era considerable, impidiendo que el dron se aproximara siquiera al recorrido esperado, como se ilustra en la siguiente figura.

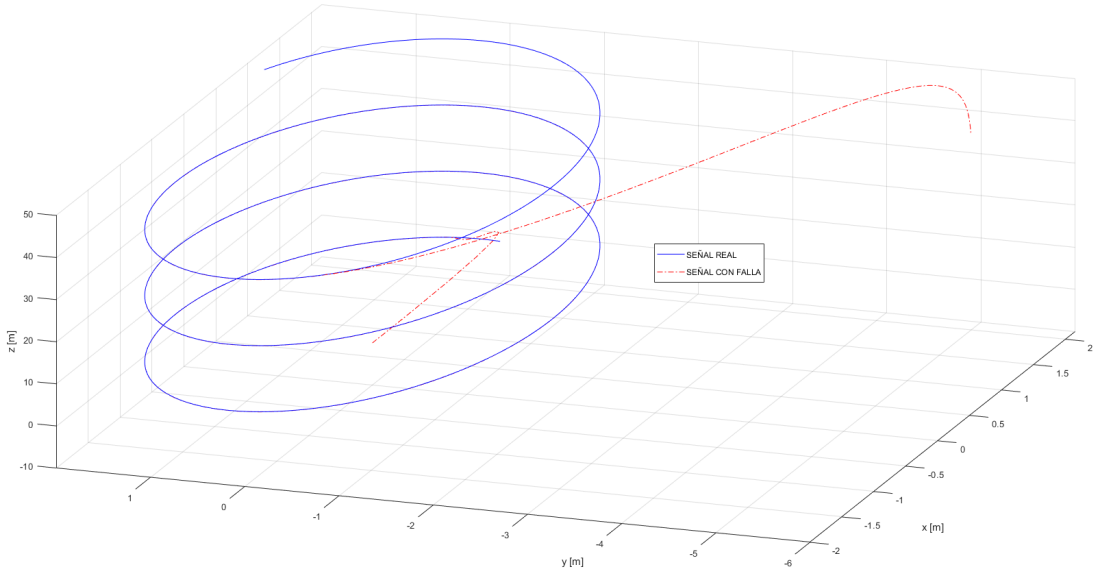

Figura 2.22: Error de trayectoria con Control Tolerante a Fallas.

Al analizar la imagen, se puede observar que el error en la trayectoria es considerablemente evidente. El comportamiento del sistema presenta una desviación significativa, al punto de que

el cuadrirrotor se desplaza en una dirección completamente opuesta a la esperada.

Con el objetivo de comprender la causa de esta desviación, se procedió a descartar posibles fuentes de error que pudieran estar afectando el desempeño del dron. Para ello, se analizan las señales generadas por el modelo *ODE* implementado en MATLAB, tal como se muestra en la figura a continuación:

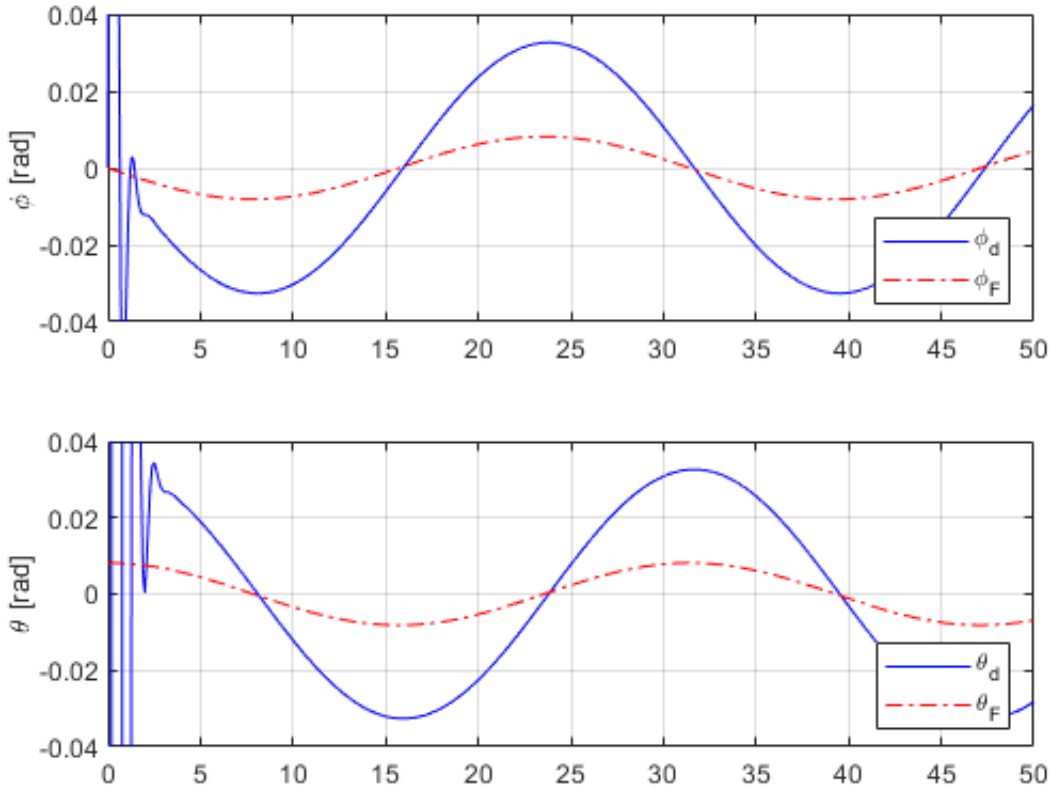

Figura 2.23: Señales reales de θ y ϕ y sus señales obtenidas con FTC.

Al comparar las señales generadas por el controlador *MPC*, se puede observar que el problema no está relacionado con un desfase temporal, sino con la amplitud. Incluso una mínima variación en la amplitud puede provocar un cambio significativo en la trayectoria de un dron. Esto se debe a que, debido a las altas velocidades a las que opera, alcanzar con precisión el punto deseado requiere una respuesta altamente exacta del sistema. Por esta razón, el control de drones exige soluciones robustas que minimicen este tipo de errores.

Antes de presentar la solución propuesta que permite al dron seguir la trayectoria deseada en presencia de una falla, es necesario discutir las posibles hipótesis que explican por qué la señal obtenida difiere de la que se espera durante su operación normal.

Una primera hipótesis sugiere que las capacidades de ejecución de MATLAB podrían no ser suficientes para resolver con precisión las operaciones requeridas para generar la señal esperada. Sin embargo, al comparar los resultados con los obtenidos en otros entornos de simulación, se

comprobó que las señales eran idénticas, lo que permitió descartar esta causa.

En segundo lugar, se procedió a revisar detalladamente todas las operaciones derivadas del enfoque de planitud diferencial. Dado que este método involucra ecuaciones de considerable complejidad, era razonable suponer que podrían existir errores en su formulación o implementación. No obstante, tras una verificación minuciosa, se confirmó que las operaciones eran correctas y que esta tampoco era la causa del problema.

Finalmente, se llegó a la conclusión de que, al tratarse de un sistema en lazo abierto, no se cuenta con un observador que permita estimar adecuadamente las variables internas del sistema. La ausencia de dicho observador impide la obtención de la señal necesaria para que el controlador pueda actuar de manera efectiva frente a situaciones de falla.

A partir de la hipótesis final planteada, el siguiente paso consiste en proponer un método que permita a la señal de control alcanzar la amplitud establecida por la señal de referencia.

2.5.2. Solución a la variación de amplitud de la trayectoria obtenida por planitud diferencial

Una solución inmediata al problema identificado con el controlador consiste en ajustar la señal de control para que alcance la amplitud requerida que permita replicar adecuadamente la trayectoria real. Para ello, se propone el uso de un controlador proporcional que actúe sobre la señal y compense la diferencia observada.

Con el fin de determinar la ganancia adecuada para dicho controlador, se procede inicialmente a graficar el error generado por el sistema de control tolerante a fallas respecto a la señal de referencia. Este comportamiento se muestra en la figura siguiente:

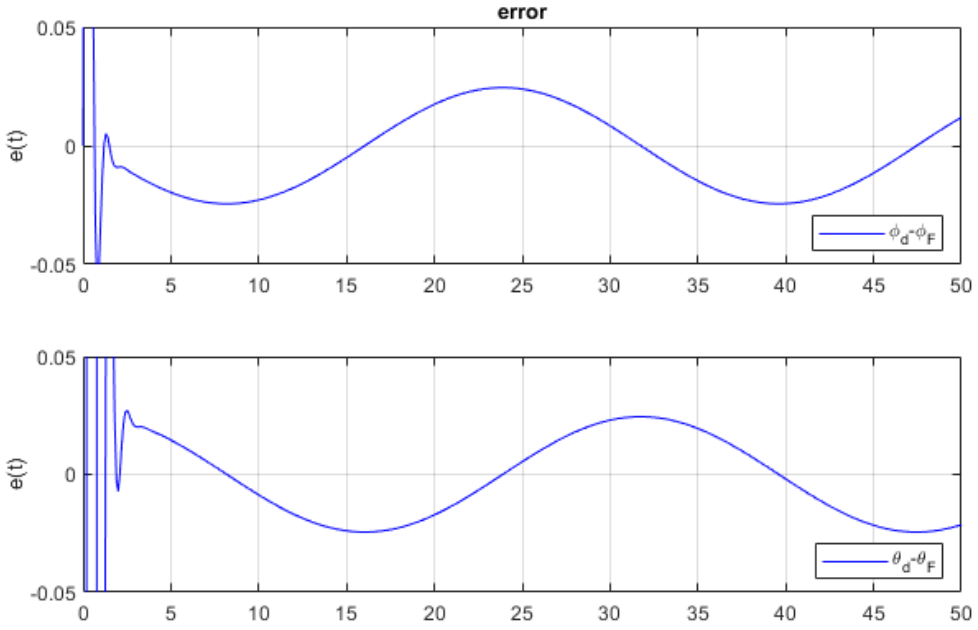

Figura 2.24: Señal de error de θ y ϕ.

A partir de la gráfica obtenida, se observa que, si bien el error es relativamente pequeño, incluso ligeras variaciones en la orientación pueden generar cambios significativos en la trayectoria del dron. Con base en este error, el siguiente paso consiste en calcular una ganancia proporcional que permita reducir dicha desviación y, de este modo, lograr que el sistema siga la trayectoria requerida.

Para este propósito, se emplea el método de ajuste mediante banda proporcional. Recordando que la ley de control proporcional se define como $u(t) = K_p e(t)$ y que la ganancia está relacionada con la banda proporcional a través de $K_p = \frac{1}{PB}$, donde PB representa la banda proporcional [19], se analizaron cuatro valores distintos de PB: 0 %, 25 %, 50 % y 100 %, los cuales se van a ir ajustando a el sistema de forma manual.

Los resultados mostraron que cuando la banda proporcional es del 25 %, el sistema presenta una respuesta cercana a la deseada. A partir de la pendiente observada en esta configuración, se determinó que el valor adecuado de la ganancia proporcional es $K_p = 4$.

Una vez establecida esta ganancia, se procedió a simular el sistema, obteniendo los siguientes resultados:

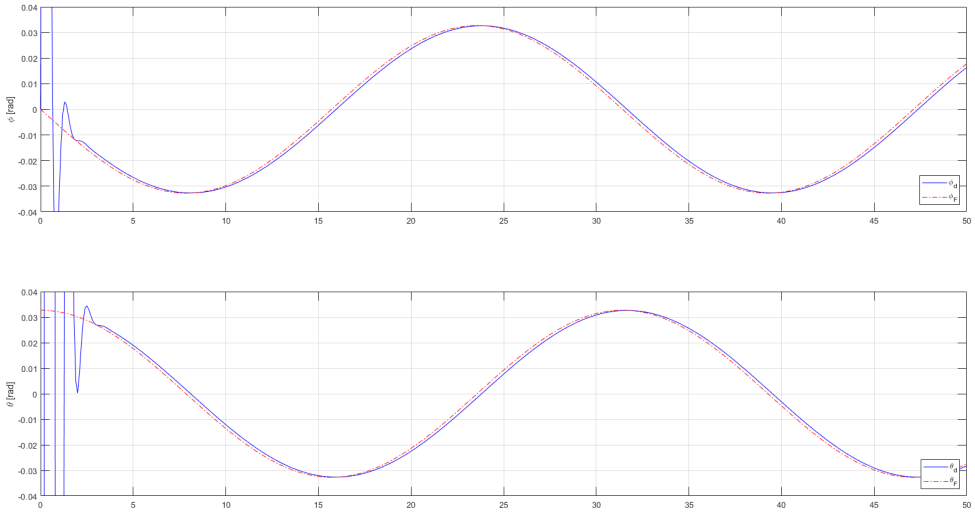

Figura 2.25: Señales reales de θ y ϕ y sus señales obtenidas con FTC.

Finalmente, se observa que el ajuste realizado permite obtener una trayectoria aceptable en lo que respecta a las orientaciones del sistema. Tal como se había anticipado, al no contar con un mecanismo de corrección inherente a un lazo de control cerrado, no se alcanza completamente la amplitud deseada.

Con este desarrollo, el siguiente paso consiste en analizar la trayectoria de posición del sistema utilizando los nuevos parámetros obtenidos, con el fin de evaluar si esta propuesta representa una alternativa viable para que el sistema opere de manera efectiva ante una falla en la posición. Esto se analiza con la siguiente imagen.

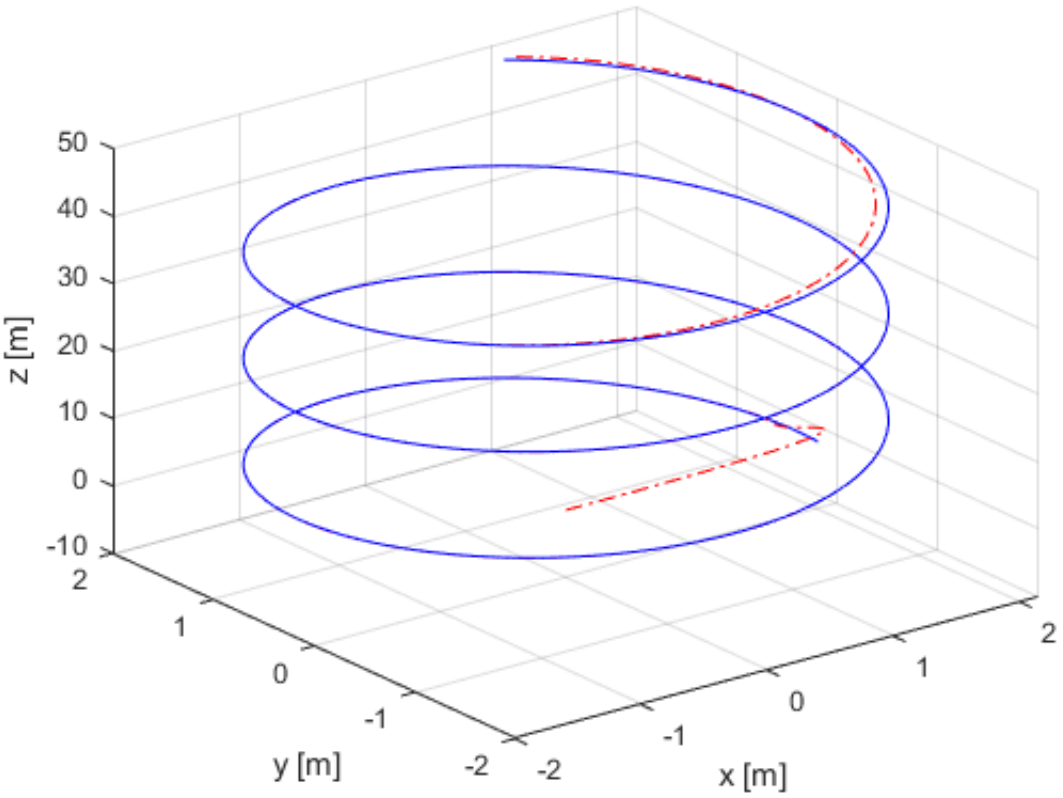

Figura 2.26: Trayectoria de posición corregida.

En este punto, se ha logrado obtener una trayectoria que permite al sistema cumplir con la tarea asignada incluso ante una falla en los sensores de posición. Si bien es posible optimizar la ganancia para reducir aún más el margen de error, lo demostrado hasta ahora confirma que, mediante una propuesta sencilla, las ecuaciones derivadas del enfoque de planitud diferencial son suficientes para alcanzar el desempeño deseado.

Con esta solución, se valida que el uso de planitud diferencial permite abordar eficazmente los problemas generados por fallas en el sensado, en los tres casos analizados a lo largo de esta tesis: orientación, posición y altitud.

Considerando los resultados obtenidos hasta este punto, se plantea proceder a la implementación del sistema. Para ello, se propone utilizar el entorno *Robot Operating System* (ROS), cuya elección se justificará en secciones posteriores. ROS ofrece múltiples beneficios al tratarse de una plataforma de código abierto ampliamente adoptada por la comunidad investigadora, lo que facilita el desarrollo de prototipos y constituye un punto de partida sólido para proyectos de investigación de mayor escala.

Capítulo 3

Implementación en ROS

Parte fundamental de esta tesis consiste en demostrar que la teoría propuesta puede aplicarse en un entorno real. Con este propósito, se ha planteado implementar los resultados obtenidos tanto del diseño del control tolerante a fallas como del control predictivo basado en modelo, utilizando un cuadrirrotor Parrot Bebop 2 como plataforma experimental.

Para el diseño del control predictivo basado en modelo, se retomará el trabajo presentado en [4], mientras que la implementación del control tolerante a fallas será desarrollada como un módulo adicional que se integrará a dicho diseño. Esto requiere un conocimiento sólido del sistema operativo Linux, así como del entorno de desarrollo ROS (Robot Operating System), donde se implementará todo el código.

Una vez verificado que el sistema responde adecuadamente frente a fallas bajo el enfoque de control tolerante, es indispensable validar su funcionamiento en un entorno de simulación cercano a la realidad. Para ello, se propone el uso del simulador Gazebo, el cual permite replicar condiciones reales de operación de manera controlada y segura.

3.1. Gazebo

Gazebo es un simulador 3D, cinemático, dinámico y multi-robot, capaz de permitirnos realizar distintos tipos de escenarios como robots articulados en entornos complejos, ya sea en interiores y exteriores, realistas y de forma tridimensional.

Varias de las ventajas que proporciona el uso de *Gazebo* es su uso, ya que se considera como software libre. De igual manera contiene una vasta cantidad de herramientas, que permite simular de forma realista la física de cuerpos rígidos, así como la simulación de elementos como el aire, la gravedad, etc.

Finalmente una de las cosas que también nos permite simular es el uso de diversos plugins que se pueden añadir, como sensores de odometría, fuerza de contacto, etc. Tomando en consideración todo esto, nos da la oportunidad de simular de forma completa lo que se esta proponiendo en el presente trabajo.

3.2. BebopS

BebopS es una extensión del paquete de ROS *RotorS* enfocado al modelado e integrando el cuadricoptero Parrot Bebop 2.

Este repositorio [18] fue hecho para diseñar sistemas de control complejos para el Parrot Bebop 2, pero puede ser usado también para la implementación de cualquier otro sistema aerodinámico.

El código que se emplea esta bajo la licencia Apache, haciendo que este sea usado para cualquier actividad educacional o científica.

Inicialmente el código fue desarrollado en Ubuntu 16.04 y la versión de Kinetic Kame en ROS, pero debido a que hoy en día esta distribución ha terminado con su soporte, se optó por migrar a su versión mas reciente, siendo la 20.04 la cual su soporte termina hasta mayo de 2025.

Es importante aclarar que, como parte del trabajo desarrollado en esta tesis, se migraron todos los repositorios a sus versiones más recientes. Sin embargo, muchas de las instrucciones contenidas en los programas originales resultaron obsoletas para las versiones actuales del software, lo cual generó diversos inconvenientes durante el proceso de instalación. Dado que gran parte de estos problemas no cuentan con soluciones documentadas en la red, los pasos detallados para llevar a cabo la instalación correctamente se incluyen en el Anexo 2.

Uno de los principales obstáculos fue la incompatibilidad con la versión de `Python` utilizada originalmente, ya que fue necesario actualizar el entorno para que funcionara con `Python 3`. Asimismo, se presentó un problema en la visualización del dron al ejecutar el simulador, lo cual se resolvió al volver a enlazar el modelo visual del *Parrot Bebop 2* con el código base del repositorio desarrollado por Silano.

Finalmente, se recomienda que, en caso de presentarse dificultades durante la instalación o ejecución del sistema, se consulte la sección de preguntas y respuestas del repositorio correspondiente alojado en `GitHub`, donde la comunidad de desarrolladores ofrece soporte activo.

3.3. Simulación de parrot bebop 2 en GAZEBO

Como se explicó previamente, *Gazebo* es un entorno de simulación que permite replicar las condiciones a las que estará sometido el robot en un entorno real. El uso de esta herramienta resulta fundamental, ya que posibilita evaluar el desempeño del dron bajo escenarios controlados que emulan una implementación práctica. De esta manera, es posible obtener datos representativos del comportamiento esperado del dron durante su operación, considerando tanto las tareas asignadas como los controladores desarrollados para su ejecución.

3.3.1. Creación del ambiente de simulación

Lo primero que vamos a hacer es usar un *.Launch* obtenido del repositorio de el *Parrotbebop*2, y definimos los valores de la velocidad del viento obtenidos de [20]

```
<arg name="wind_force" default="0.25"/>
<arg name="wind_start" default="5.0"/>
<arg name="wind_duration" default="40.0"/>
<arg name="wind_direction_x" default="1.0"/>
```

Una vez establecidos los parámetros, usaremos los argumentos de ROS para usar el *Parrot Bebop 2* y mostrarlo en la interfaz de Gazebo como se muestra a continuación:

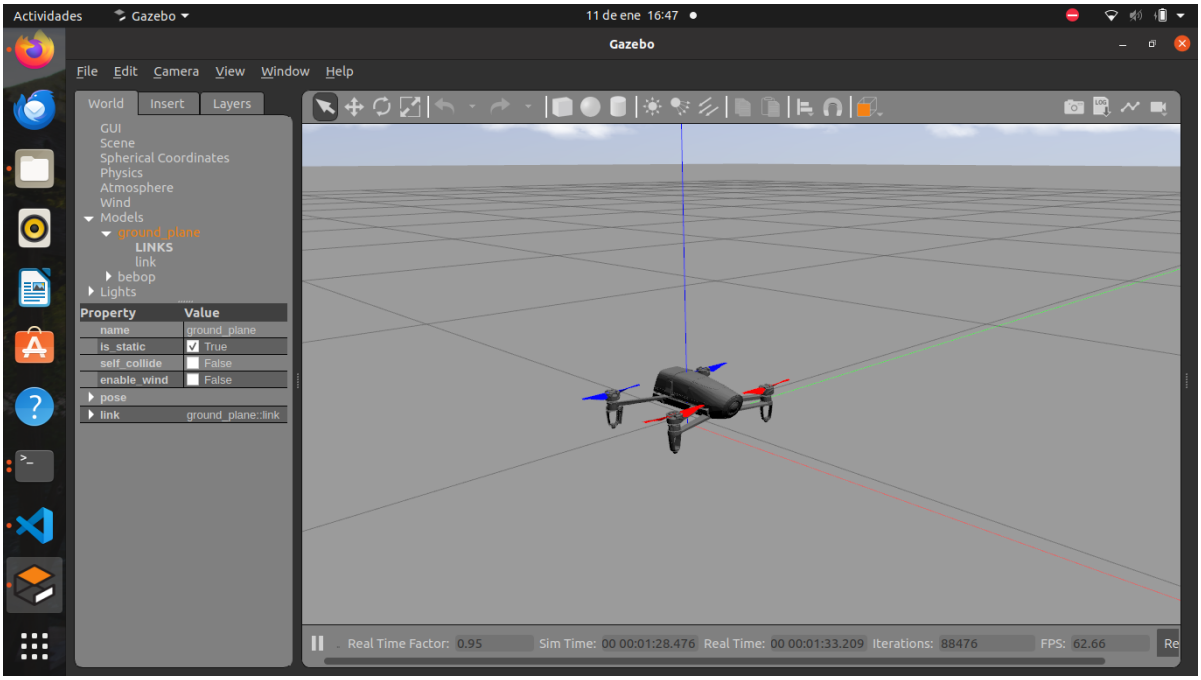

Figura 3.1: Dron Bebop 2 desplegado en Gazebo.

3.3.2. Generación de trayectorias

Una ventaja de *ROS* y el repositorio usado *Bebop*, es que se pueden exportar los vectores de trayectoria a través de un archivo de texto.

Vamos a buscar replicar la trayectoria de las simulaciones obtenidas a través de MATLAB y obtenemos un archivo con la trayectoria en vectores. esta misma tendrá el nombre de *waypoint.txt* y en ella estará alojada los puntos pertenecientes a la espiral.

Una vez obtenida la trayectoria a través del archivo de texto, lo siguiente es proceder a crear las señales de falla y observar si así como en MATLAB funcionan las señales, también en ROS.El codigo propuesto tiene la funcion de realizar las trayectorias conforme a la teoría de falla estudiada. Estas mismas son realizadas en *Python* y las lineas de código se presentan en el Anexo 3.

3.4. Resultados de la simulación

Antes de iniciar con la simulación, es necesario definir el rango de error máximo aceptable con el que se trabajará en el dron. Para ello, se toman como referencia estudios previos sobre control predictivo basado en modelos (MPC), como los presentados en [4], en los cuales se establece que:

$$\text{error} \leq 30\,\%$$

Este valor de error funcionará como punto de referencia para que, en caso de detectarse una falla, el sistema realice una conmutación virtual hacia el uso del controlador tolerante a fallas. El procedimiento consiste en que, si durante la ejecución de la trayectoria el dron se desvía fuera del rango de error permitido, el sistema de detección de fallas —en este caso, el FDI (Fault Detection and Isolation)— active automáticamente dicho cambio de forma virtual.

Este mecanismo se basa en la comparación en tiempo real entre la trayectoria esperada y la trayectoria ejecutada, lo que permite detectar desviaciones significativas atribuibles a posibles fallas y una vez que se salga del rango establecido, hacer ese *switch* virtual y tomar los valores proporcionados por planitud diferencial. Una vez definida esta lógica de actuación, el siguiente paso consiste en simular una falla de forma controlada, evitando comprometer físicamente la integridad del dron.

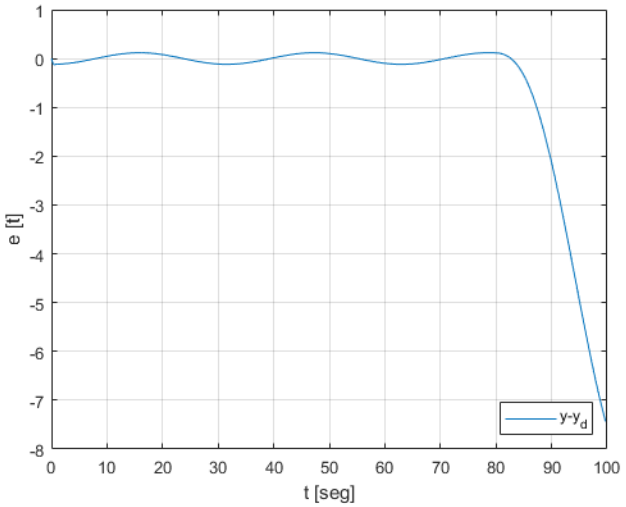

Figura 3.2: Grafica de error de posición de y.

A partir de la figura anterior se observa que el sistema mantiene una oscilación en torno al valor de referencia con un margen aproximado del 10 %. Sin embargo, en el instante de tiempo 80, cuando se introduce la falla, el sistema comienza a desviarse de su trayectoria, lo que se refleja en un incremento significativo del error, alcanzando un valor máximo cercano al 30 % en el instante 84. A partir de ese momento, entra en funcionamiento el esquema de control tolerante a fallas, el cual permite que el sistema continúe operando de manera estable bajo la nueva condición.

Cabe destacar que el sistema operativo de robots (ROS) permite experimentar con distintas configuraciones, tales como la desactivación del sistema de sensado de odometría. Para simular una falla en este componente, basta con asignar el valor `false` a la instrucción:

```
enable\_odometry\_sensor\_with\_noise
```

Con esta consideración, se introduce una falla en el instante 80, de manera análoga a la simulación realizada previamente en MATLAB. A partir de esta ejecución, se genera un video que muestra la trayectoria obtenida en el entorno de ROS, el cual puede visualizarse en [21].

A simple vista, se aprecia que, a pesar de la falla inducida, no se presenta una desviación significativa en la trayectoria del dron. Este comportamiento sugiere que la robustez del esquema de control validado en MATLAB también se mantiene en un entorno de simulación más complejo y realista.

Para validar con mayor rigurosidad los datos obtenidos en ROS, se emplea la herramienta *rqt_plot*, la cual permite representar gráficamente las señales generadas por el sistema durante la simulación.

Una limitación inherente a ROS es su capacidad restringida para la visualización tridimensional de trayectorias. Por esta razón, los datos registrados son exportados a MATLAB para su análisis detallado y su comparación con los resultados teóricos.

Para este propósito, se utiliza la herramienta *rosbag*, que permite almacenar en tiempo real los datos publicados por los tópicos del entorno *Gazebo*. Específicamente, se emplea el comando `rosbag record` para capturar la información correspondiente a la trayectoria del dron.

Una vez generado el archivo de registro, este se convierte al formato `.csv`, agrupando la información relevante de cada tópico en vectores compatibles con software de análisis como Excel o MATLAB. El comando utilizado para la conversión es el siguiente:

```
rostopic echo /topicname -b bagFileName.bag -p > file.csv
```

Posteriormente, estos datos se trasladan a MATLAB, donde se comparan con las señales generadas por la simulación en ese entorno, como se muestra en la referencia [22]. La siguiente imagen ilustra los resultados obtenidos.

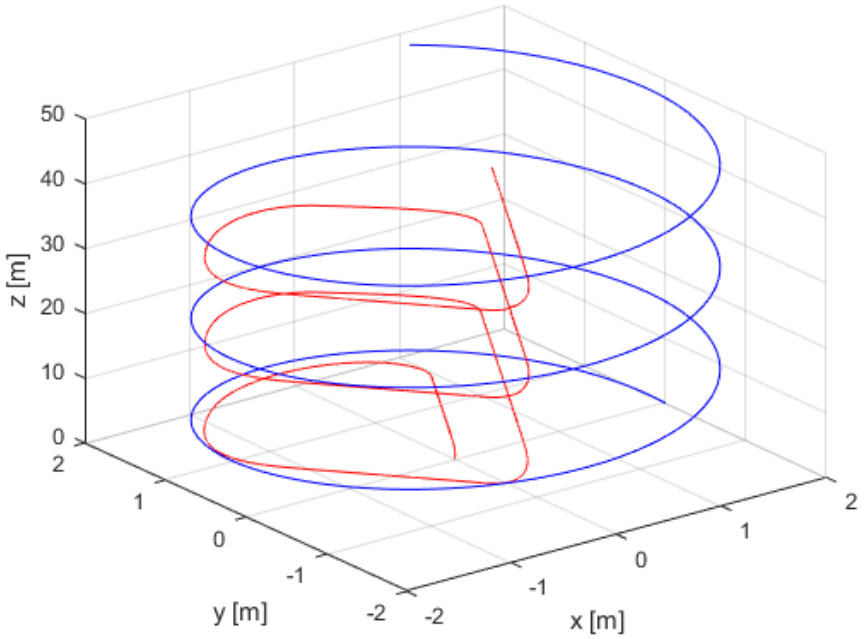

Figura 3.3: Señal dada por GAZEBO contra señal simulada en MATLAB

Al observar los primeros resultados, se aprecia que el error es considerablemente alto, ya que en ninguno de los grados de libertad del sistema se alcanza una proximidad aceptable respecto al valor de referencia requerido para el desempeño esperado.

Profundizando en el análisis de las posibles causas de este comportamiento, se identificó que el sistema no opera en tiempo real. Esto se debe a la elevada carga de procesamiento asociada a la simulación, al punto que el sistema reporta estar funcionando únicamente al 16

Con el objetivo de evaluar el funcionamiento del controlador propuesto, se optó por simplificar el entorno de simulación, eliminando algunos procesos y sintetizando otros. Esta estrategia permitió observar un desempeño más claro y cercano a un funcionamiento aceptable.

Los resultados obtenidos se encuentran en el video disponible en [23].

Asimismo, las gráficas correspondientes a esta simulación se presentan en la imagen siguiente:

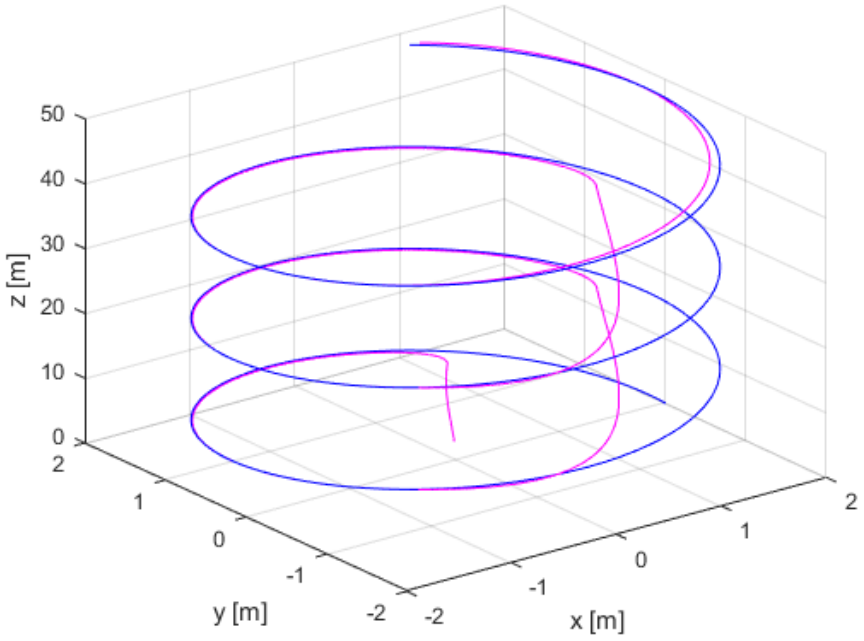

Figura 3.4: Señal de GAZEBO (Magenta) y señal de MATLAB(Azul)

En la figura anterior se puede observar que, a pesar de las mejoras realizadas en el código para simular la trayectoria, persisten indicios de una carga de procesamiento insuficiente. Esto se refleja en que la gráfica aún no cumple completamente con los requisitos establecidos para el desempeño esperado. Sin embargo, en ciertos puntos se aprecia una aproximación más cercana a la trayectoria deseada. Además, se puede constatar que el controlador tolerante a fallas actúa de forma oportuna y conforme a lo esperado. Este comportamiento se explica porque, al desactivarse el control predictivo basado en modelos, se reduce significativamente la carga de procesamiento, quedando solamente activa la lógica de control tolerante a fallas.

Es fundamental considerar las características del equipo de cómputo utilizado en esta simulación, con el fin de compararlas con las capacidades del dron Bebop 2 y estimar cómo se

comportaría el sistema en un entorno real.

Con el propósito de que el lector comprenda mejor las demandas de procesamiento que puede requerir el sistema, a continuación se detallan las especificaciones técnicas del equipo utilizado. Aunque esta computadora no es de última generación, supera ampliamente la capacidad de procesamiento integrada en el dron, lo que resalta la importancia de optimizar el código y los procesos involucrados para una implementación práctica.

Las características de la computadora utilizada son las siguientes:

Característica	Valor
Procesador	Intel Core i5-3210M (hasta 3.1 GHz) 3ra. generación
Memoria RAM	6 GB DDR3
Disco duro	750 GB
Tarjeta grafica	Nvidia Geforce 610M 2 GB
Red	802.11 b/g/n

Tabla 3.1: Caracteristicas de la computadora

Los datos mostrados en la tabla anterior respaldan lo planteado anteriormente: el dron no es capaz de replicar completamente las operaciones realizadas en la simulación, debido a que sus especificaciones, como se detallará en las secciones siguientes, no alcanzan el nivel de procesamiento de una computadora convencional. Aunque se podría intentar una implementación directa, es altamente probable que los resultados sean similares o incluso inferiores en términos de rendimiento.

Ante esta situación, se plantea como alternativa el uso de un sistema embebido externo que se encargue de ejecutar las operaciones asociadas al control tolerante a fallas. De esta manera, el sistema principal del Bebop 2 podría concentrarse exclusivamente en ejecutar el control predictivo basado en modelos, aligerando así la carga de procesamiento y mejorando la viabilidad de la implementación.

Una vez comprobado que la hipótesis de trabajo es válida, a pesar de las limitaciones impuestas por la simulación, el siguiente paso lógico sería implementar el sistema desarrollado en el dron Parrot Bebop 2. No obstante, debido a las restricciones derivadas de la pandemia, el dron dejó de funcionar adecuadamente. La batería del dispositivo dejó de ser operativa, y actualmente resulta difícil conseguir un reemplazo compatible.

Es importante señalar que las pruebas realizadas se llevaron a cabo únicamente en un entorno de simulación. No obstante, antes de que surgieran los problemas previamente mencionados —como la necesidad de actualizar la implementación del MPC en ROS a su versión más reciente— se alcanzaron avances significativos. A continuación, se presentan dichos avances, los cuales complementan y refuerzan la investigación desarrollada en este trabajo.

3.5. Parrot Bebop 2

A finales del 2014 Parrot presentó Bebop 1, un dron con un procesador interno doble núcleo ARM Cortex A9, que trabaja sobre Linux y aparte de ello una memoria de almacenamiento de

8 GB Y 1 GB de RAM. Mas tarde presentó la creación de *bebop autonomy* el cual nos permite interactuar con el dron a través de Linux, utilizando ROS.

A finales del 2015 Parrot presentó Bebop 2, un modelo mejorado con más autonomía de vuelo que, sin embargo, se rige por el mismo sistema operativo que su antecesor. El Bebop 2 presenta una autonomía de hasta 25 minutos y permite la realización de video en 1080p x 1920p y fotografías de hasta 14 megapixeles.

Figura 3.5: Cuadrirrotor Bebop 2 de Parrot.

Su velocidad máxima es de 60 km/h y se estabiliza gracias a la acción de un acelerómetro y un magnetómetro.

Las baterías que utiliza este dron son *Plugandplay* y su tiempo de carga es de hora y media.

Gracias a sus dos antenas internas el Bebop 2 emite $Wifi$ en 2,4 Ghz y en 5Ghz, y cuenta con una baliza de luz de color rojo en la parte trasera para localizarlo a distancia.

En rasgos generales, las características del Dron se presentan a continuación en la siguiente tabla [24]:

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

Tabla 3.2: Parámetros del sistema.

3.6. Plataforma ROS

ROS Robot Operating Systems 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. Este sistema fue diseñado para ser distribuido lo más modularmente posible, de modo que los usuarios pudiesen elegir qué partes utilizar en función de las necesidades de su aplicación. La librería está orientada para el sistema Ubuntu Linux, a pesar de que existe adaptaciones para otros S.O. esta no garantiza su estabilidad.

3.7. Nodos

Un nodo se piensa como un programa ejecutable. En cada nodo se propone realizar una tarea para que pueda ser un programa reutilizable. Un ejemplo es que en el nodo maestro se tenga el programa principal y en los nodos que se registran a el maestro, tener tareas como captura de imágenes, navegación, etc. Esto lo podemos visualizar mejor en la imagen siguiente:

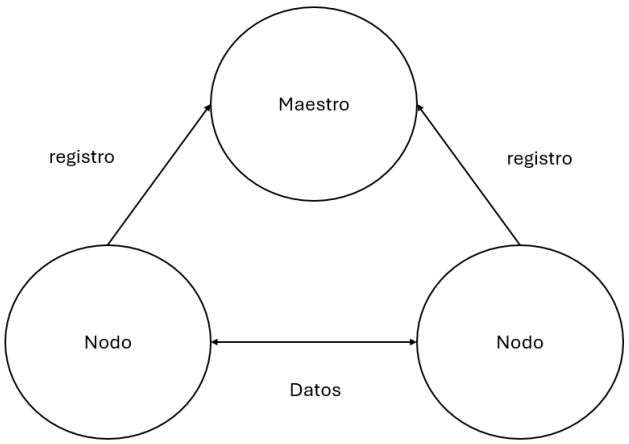

Figura 3.6: Esquema del funcionamiento de los nodos en ROS.

3.8. Instalación de ROS

Dado a que el avance ya hecho de la parte del control MPC se encuentra alojado en ROS kinetic Kame se necesita una computadora con un sistema operativo LINUX y UBUNTU 16.04 LTS, los pasos para una instalación correcta las podemos encontrar en [25]. Sin embargo, parte de este trabajo es actualizar lo que se tenia por parte de [4] y actualizarlo a UBUNTU 20.04, que es la ultima versión para trabajar el acoplamiento que fue desarrollado en esta versión, es por eso que describimos los pasos de como se realizó esta parte.

Bebop 2 puede ser controlado por medio del *driver bebop_autonomy* el cual fue desarrollado por Mani Monajjemi [26]. A continuación se presentan los pasos para la instalación para el trabajo de este driver, ademas los comandos que se presentan fueron probados.

Los pasos para la instalacion se encuentran en el Anexo 4.

Despues de los pasos de instalación y usando los tipos de mensaje que vienen en la tabla siguiente, podemos realizar un movimiento característico en el dron:

Componente	Descripción
$linear.x = kx$	Desplazamiento hacia delante
$linear.x = -kx$	Desplazamiento hacia atrás
$linear.y = ky$	Desplazamiento a la izquierda
$linear.y = -ky$	Desplazamiento a la derecha
$linear.z = kz$	Ascenso
$linear.z = -kz$	Descenso
$angular.z = k_\phi$	Rotación CCW
$angular.z = -k_\phi$	Rotación CW

Tabla 3.3: Componentes del Dron.

Una vez completado correctamente el proceso de instalación y establecido un enlace funcional con el Dron, el siguiente paso consiste en verificar el funcionamiento de las operaciones básicas descritas en la tabla anterior. Para ello, se llevarán a cabo pruebas de laboratorio en un entorno controlado, específicamente en un cubículo ubicado en el campus Puebla de la UDLAP.

Los resultados obtenidos de estas pruebas se presentan en la sección siguiente.

3.9. Pruebas en laboratorio

A continuación se anexan imágenes en donde se presenta el dron que se va a usar durante la implementación, como puntos a resaltar, dado a que en la pandemia los drones permanecieron guardados, estos se vieron comprometidos en cuanto a la duración de sus baterías, además se presenta un video demostrativo de el cuadrirrotor siendo probado en su estado *Hovering* en [27].

Figura 3.7: Cuadrirrotor en área de trabajo.

Figura 3.8: Cuadrirrotor en Hovering.

A simple vista, el dron muestra un buen nivel de estabilidad, lo que lo convierte en un candidato viable para la implementación del prototipo propuesto. Sin embargo, debido a la limitación de autonomía de aproximadamente cinco minutos, es necesario realizar pruebas con trayectorias breves, asegurando un punto de aterrizaje cercano que minimice el riesgo de daño al equipo. Además, es fundamental considerar un diseño adecuado para la integración de sistemas adicionales, procurando que el peso de estos no afecte significativamente la dinámica del dron.

Una vez verificado el funcionamiento básico del Bebop 2, el siguiente paso consiste en implementar el controlador predictivo basado en modelos (MPC). No obstante, antes de proceder con dicha implementación, se optó por seguir una serie de tutoriales con el fin de fortalecer y consolidar los conocimientos adquiridos sobre el entorno de desarrollo ROS.

3.10. Tutoriales de ROS

Debido a que el enfoque de esta tesis requiere que implementemos los controladores en el entorno computacional *ROS* es necesario aprender a usar cada uno de las herramientas básicas que nos proporcionan en el mismo, para ello se realizó cada uno de los tutoriales que se nos proporcionan en [28], para dominar las bases de este lenguaje de programación.

3.11. MPC implementado en Bebop 2

Para la implementación del dron Bebop 2, es necesario ejecutar una trayectoria en la que, en un momento determinado, se simule una falla y, al ocurrir esta, el sistema de control tolerante a fallas sea capaz de tomar el control del vehículo de manera autónoma.

En esta etapa del trabajo, se reutiliza el esquema de control predictivo basado en modelo (MPC) previamente desarrollado en [4], con la diferencia de que se actualiza su implementación para ser compatible con la versión de Ubuntu 20.04, dado que el desarrollo original fue realizado sobre Ubuntu 16.04. Cabe destacar que dicha actualización es estrictamente necesaria, ya que el código original se encuentra obsoleto y no puede ejecutarse directamente en entornos más recientes. Esta limitación impide su integración con el controlador tolerante a fallas implementado en ROS.

Una vez aclarado este punto, a continuación se describe el proceso de actualización y adaptación del controlador MPC, el cual permitirá su integración funcional en el entorno actual de desarrollo.

Lo primero es partir de desarrollar un sistema de control de bajo nivel ya que al ser un Dron comercial, no se tiene detalles claros de como esta definido su sistema dinámico.

Para la implementación del controlador MPC es necesario tomar en consideración la dinámica del dron interna. En este caso, dado a que trabajamos con una versión comercial, esta es desconocida, y por lo tanto se necesita aproximar con un sistema de primer orden el lazo cerrado de control de cada una de las entradas de control. Por ello se propone lo siguiente [4] :

$$\dot{\phi} = \frac{1}{\tau_\phi}(k_\phi \phi_{cmd} - \phi) \tag{3.1}$$

$$\dot{\theta} = \frac{1}{\tau_\theta}(k_\theta \theta_{cmd} - \theta) \tag{3.2}$$

$$\ddot{z} = \frac{1}{\tau_{\dot{z}}}(k_{\dot{z}} \dot{z}_{cmd} - \dot{z}) \tag{3.3}$$

$$\dot{\psi} = \dot{\psi}_{cmd} \tag{3.4}$$

Donde k_ϕ, k_θ y $k_{\dot{z}}$ son las ganancias de los sistemas de primer orden, de igual manera τ_θ, τ_ϕ y τ_ψ las constantes de tiempo, ϕ_{cmd}, θ_{cmd} y $\dot{z}_{cmd}$ son las entradas de control comandadas. $\dot{\psi}_{cmd}$ es la velocidad comandada alrededor del eje z.

En colaboración con compañeros de la UDLAP se implementó el MPC que hasta momento no se tenia registro de su funcionamiento como se presentaba en [4], se buscó migrar a la versión *Noetic*, correspondiente a ROS ya que el programa se encontraba alojado en *Kinetic*, y este era un tanto obsoleto.

Antes de correr el programa ya desarrollado, se necesita modificar el *.bashrc*, e insertar las siguientes lineas de código:

```
source ~/catkin_ws/devel/setup.bash
export export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/catkin_ws/devel/lib/
parrot_arsdk
export export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/catkin_ws/src/Master_Thesis/
bebop_mpc_control
export ROS_MASTER_URI=http://localhost:11311
export ROS_HOSTNAME=localhost
```

Después se necesita abrir la terminal de linux y escribir las siguientes lineas de código:

```
gedit ~/.bashrc
source ~/.bashrc
roscore
roslaunch bebop_driver bebop_node.launch
rostopic pub --once /bebop/takeoff std_msgs/Empty
```

Y además se necesita tener en consideración estas lineas de código para el aterrizaje:

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

Para determinar si el sistema está funcionando correctamente, se utilizará la odometría como herramienta para obtener la posición del dron en tiempo real. Para ello, se emplean las siguientes líneas de código, las cuales permiten visualizar cómo, a través de los datos proporcionados por la odometría, es posible conocer la ubicación exacta del dron durante su operación.

```
roslaunch bebop_odometry_example  bebop_odometry_example.launch
```

3.11.1. Comandos para control de MPC

Para correr el código del controlador MPC, se necesita primero designar la trayectoria que el dron va a realizar. Se utiliza un ejecutable que despliega la siguiente tabla en donde se designa el tiempo, la posición x, y , z y el angulo de cada punto por donde pasará el dron, en este caso se presenta la ejemplificación de un cuadrado.

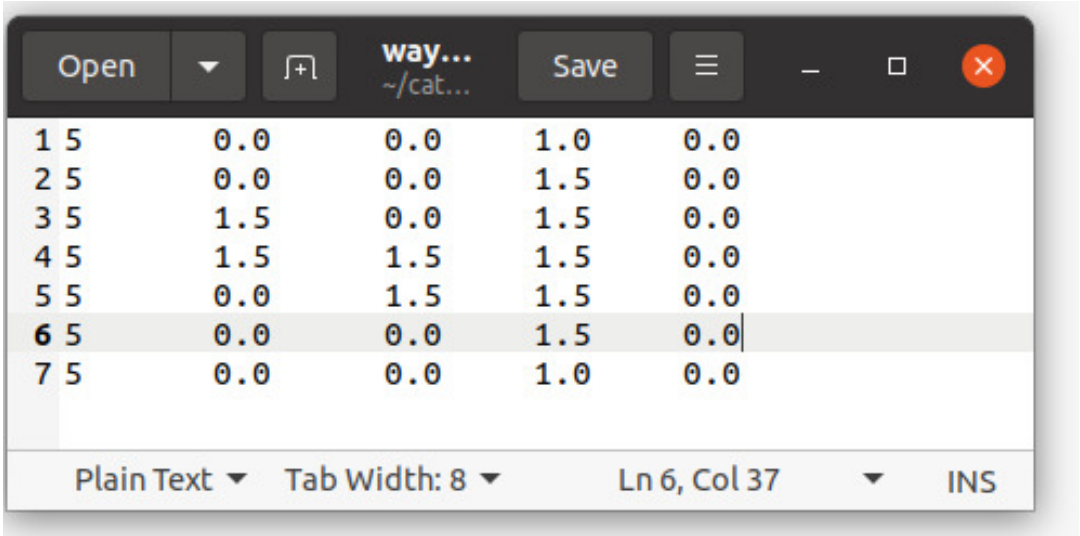

Figura 3.9: matriz para insertar la posición del dron.

Una vez que se insertan los valores, se corre el programa correspondiente para empezar a usar el MPC diseñado.

```
roslaunch mpc_traj_viz  bebop_mpc_viz.launch
roslaunch bebop_linear_mpc bebop_linear_mpc.launch
```

Una vez que se corre, se abre la ventana de odometría (*Rviz*) que muestra la trayectoria que el dron está realizando.

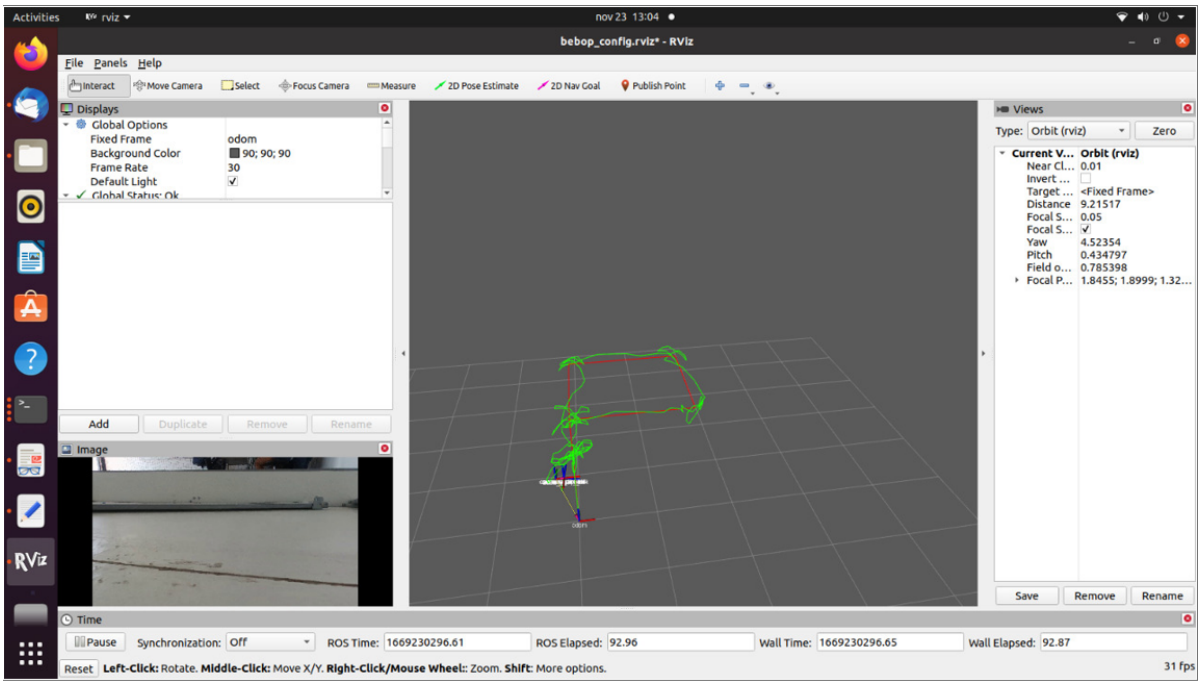

Figura 3.10: Dron realizando la trayectoria empleada con el MPC.

Hasta este punto, se logró comprobar satisfactoriamente que el sistema de control predictivo basado en modelos (MPC), propuesto en [4], funciona de manera correcta. La principal contribución de esta tesis consiste en la migración e implementación de dicho sistema en una versión más actual del sistema operativo, específicamente Ubuntu 20.04.

Sin embargo, tal como se anticipó al inicio de este trabajo, las limitaciones técnicas derivadas de la falla de la batería del dron impidieron completar la implementación en el nivel deseado. A pesar de ello, las simulaciones realizadas tanto en MATLAB como en el entorno virtual de Gazebo permitieron validar el comportamiento esperado del sistema bajo condiciones controladas.

Con base en estos resultados, se puede inferir que la propuesta tiene un alto potencial de funcionar correctamente en un entorno real, especialmente si se incorpora un sistema embebido externo que asista al dron en la ejecución de las tareas asociadas al control tolerante a fallas, reduciendo así la carga computacional total.

Conclusiones

El esquema de control tolerante a fallas propuesto en esta tesis, en conjunto con el controlador predictivo basado en modelo (MPC), ha demostrado un desempeño favorable en la mitigación de fallas, contribuyendo a la estabilidad y continuidad operativa del dron ante situaciones adversas y sobre todo garantizando su supervivencia. Los resultados obtenidos en los entornos de simulación validan la viabilidad del enfoque teórico, mostrando que es posible preservar el comportamiento del sistema incluso en presencia de perturbaciones o fallas simuladas.

En síntesis, este trabajo evidencia la viabilidad de integrar el control predictivo con estrategias de tolerancia a fallas en un cuadrirrotor comercial, evaluado dentro de un entorno de simulación realista. Esta contribución establece un precedente relevante para el desarrollo futuro de drones con mayores niveles de autonomía, resiliencia y seguridad operacional.

No obstante, uno de los principales desafíos identificados radica en la elevada demanda computacional de ambos controladores al ejecutarse de forma simultánea. Esta carga limita significativamente su implementación directa en plataformas con recursos restringidos, como el dron Bebop 2. De hecho,tomando en consideración que una computadora con capacidades superiores enfrenta dificultades para procesar ambos esquemas en tiempo real, es evidente que el dron en cuestión no puede hacerlo de forma autónoma.

Como solución potencial, se propone la incorporación de un sistema embebido adicional que permita ejecutar el controlador tolerante a fallas de manera paralela, liberando así parte de la carga computacional del sistema principal y posibilitando una implementación más eficiente y práctica.

Es importante señalar que, debido a limitaciones técnicas —en particular, la falla irreparable de la batería del dron— no fue posible realizar pruebas experimentales en un entorno físico. Por esta razón, se optó por validar la propuesta en un entorno de simulación robusto utilizando ROS y Gazebo, lo cual representa una mejora sustancial respecto a simulaciones únicamente en MATLAB.

A pesar de esta limitación, los resultados alcanzados ofrecen una base sólida que permite inferir el comportamiento esperado en condiciones reales, proporcionando así una guía útil para futuras implementaciones prácticas del sistema propuesto.

Bibliografía

[1] Rafael Yanushevsky. *Guidance of unmanned aerial vehicles*. CRC Press Taylor and Francis Group, 2011.

[2] Zhang Y. , Jiang J. Bibliographical review on reconfigurable fault-tolerant control systems. *IFAC symposium on Fault Detection, Supervision and Safety for Technical Processes*, 5:265–276, 2003.

[3] Martínez Torres César. Fault tolerant control by flatness approach. Tesis doctoral, 2025. Universidad Autónoma de Nuevo León.

[4] Salazar Hidalgo Eduardo. Control mpc de un cuadricóptero para el seguimiento de trayectorias basado en odometría visual. Tesis maestria, 2021. Benemérita Universidad Autónoma de Puebla.

[5] Daniel Caballero-Martin, Jose Manuel Lopez-Guede, Julian Estevez, and Manuel Graña. Artificial intelligence applied to drone control: A state of the art. *Drones*, 8(7), 2024.

[6] Nina Hendrarini and Muhammad Ikhsan Sani. Unmanned aerial vehicle (uav) predictive control energy management model. In *Proceedings of the 2024 7th International Conference on Software Engineering and Information Management*, ICSIM '24, page 147–152, New York, NY, USA, 2024. Association for Computing Machinery.

[7] Zhiqiang Wei, Fan Liu, Derrick Wing Kwan Ng, and Robert Schober. Safeguarding uav networks through integrated sensing, jamming, and communications. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8737–8741, 2022.

[8] Francisco Giral, Ignacio Gómez, Ricardo Vinuesa, and Soledad Le Clainche. Transformer-Based Fault-Tolerant Control for Fixed-Wing UAVs Using Knowledge Distillation and In-Context Adaptation. *arXiv e-prints*, page arXiv:2411.02975, November 2024.

[9] Huu-Thinh Do, Franco Blanchini, and Ionela Prodan. A flatness-based saturated controller design for a quadcopter with experimental validation. *arXiv e-prints*, page arXiv:2303.18021, March 2023.

[10] Arman Mohammadi and Amin Ramezani. A robust model predictive control-based method for fault detection and fault tolerant control of quadrotor uav. *Transactions of the Institute of Measurement and Control*, 45(1):37–48, 2023.

[11] García P. Castillo P. Modelado y estabilización de un helicoptero con cuatro rotores. *Revista iberoamericana de Automatica e informatica industrial*, 4:41–57, 2007.

[12] Brian C. Fabien. *Analitical System Dynamics, Modelling and simulation*. Springer, 2009.

[13] Matlab. Understanding model predictive control, part 1, why use mpc? https://www.youtube.com/watch?v=8U0xiOkDcmw, 2018.

[14] E Salazar-Hidalgo, J Castañeda-Camacho, César Martinez Torres, and Jose Martinez-Carranza. Model-based predictive control for trajectory tracking of a quadrotor. 10 2020.

[15] Eduardo Camacho and Carlos Bordons. *Model Predictive Control*, volume 13. 01 2004.

[16] César Martínez Torres, Loïc Lavigne, Franck Cazaurang, Efraín Alcorta García, and David Díaz Romero. Fault tolerant control of a three tank system: A flatness based approach. In *2013 Conference on Control and Fault-Tolerant Systems (SysTol)*, pages 529–534, 2013.

[17] S. Salazar-Cruz, A. Palomino, and R. Lozano. Trajectory tracking for a four rotor mini-aircraft. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 2505–2510, 2005.

[18] Giuseppe Silano, Pasquale Oppido, and Luigi Iannelli. Software-in-the-loop simulation for improving flight control system design: a quadrotor case study. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 466–471, 2019.

[19] Francisco Gato, Héctor Pardo, Esteban Pérez, José Roca, and Jose Calvo-Rolle. *Diseño de controladores PID*. 07 2020.

[20] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. *RotorS – A Modular Gazebo MAV Simulator Framework*, volume 625, pages 595–625. 01 2016.

[21] Ing. Chitan. Control predictivo y control tolerante a fallas en ros. https://www.youtube.com/watch?v=CZnU5AVFvzI, 2025.

[22] Morgan Quigley, Brian Gerkey, and William D. Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. O'Reilly Media, Inc., 1st edition, 2015.

[23] Ing. Chitan. Control predictivo y control tolerante a fallas en ros(corregido). https://www.youtube.com/watch?v=WTDV-Zwjb98, 2025.

[24] Parrot. Parrot bebop 2. url: https://www.parrot.com/es/drones/parrot/bebop-2, 2019.

[25] ROS.ORG. Ubuntu install of ros kinetic. http://wiki.ros.org/kinetic/Installation/Ubuntu.

[26] Bebop Autonomy. Ros driver for parrot bebop drone (quadrocopter) 1.0 & 2.0. https://bebop-autonomy.readthedocs.io/en/latest/.

[27] Ing. Chitan. Prueba dron bebop 2. www.youtube.com/shorts/l91_BdCSHnIu, 2022.

[28] Ros.org. Ros tutorials. https://wiki.ros.org/ROS/Tutorials, 2025.

Anexos

Anexo 1

Partimos de sumar las igualdades de la ecuación 2.14 como se muestra a continuación

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 = (-u_1 \text{sen}\theta)^2 + (u_1 \cos\theta \text{sen}\phi)^2 + (u_1 \cos\theta\cos\phi - mg)^2$$

Realizando los cálculos exponenciales correspondientes al lado derecho de la igualdad, obtenemos:

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 = u_1^2 \text{sen}^2\theta + u_1^2 \cos^2\theta \text{sen}^2\phi) + (u_1 \cos\theta\cos\phi - mg)^2$$

De igual manera, sabemos que:

$$(u_1 \cos\theta\cos\phi - mg)^2 = u_1^2 \cos^2\theta\cos^2\phi - 2mgu_1 \cos\theta\cos\phi + (mg)^2$$

Y obtenemos que:

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 = u_1^2 \text{sen}^2\theta + u_1^2 \cos^2\theta \text{sen}^2\phi + u_1^2 \cos^2\theta\cos^2\phi - 2mgu_1 \cos\theta\cos\phi + (mg)^2$$

Sabiendo que $\text{sen}^2\phi = 1 - \cos^2\phi$ entonces:

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 = u_1^2 \text{sen}^2\theta + u_1^2 \cos^2\theta(1 - \cos^2\phi) + u_1^2 \cos^2\theta\cos^2\phi - 2mgu_1 \cos\theta\cos\phi + (mg)^2$$

simplificando:

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 = u_1^2 \text{sen}^2\theta + u_1^2 \cos^2\theta - u_1^2 \cos^2\theta\cos^2\phi + u_1^2 \cos^2\theta\cos^2\phi - 2mgu_1 \cos\theta\cos\phi + (mg)^2$$

Realizando operaciones:

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 = u_1^2 \text{sen}^2\theta + u_1^2 \cos^2\theta - 2mgu_1 \cos\theta\cos\phi + (mg)^2$$

Como $\text{sen}^2\theta + \cos^2\theta = 1$ entonces:

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 = u_1^2 - 2mgu_1 \cos\theta\cos\phi + (mg)^2$$

Sabiendo que $\text{m}\ddot{z} + mg = u_1 \cos\theta\cos\phi$, entonces:

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 = u_1^2 - 2mg(m\ddot{z} + mg) + (mg)^2$$

realizando operaciones:

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 = u_1^2 - 2m^2g\ddot{z} - 2(mg)^2 + (mg)^2$$

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 = u_1^2 - 2m^2g\ddot{z} - (mg)^2$$

Despejando u_1

$$(m\ddot{x})^2 + (m\ddot{y})^2 + (m\ddot{z})^2 + 2m^2g\ddot{z} + (mg)^2 = u_1^2$$

$$m\sqrt{(\ddot{x})^2 + (\ddot{y})^2 + (\ddot{z})^2 + 2g\ddot{z} + (g)^2} = u_1$$

Anexo 2

En Ubuntu 20.04 abrimos una terminal y escribimos las siguientes lineas de código:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_
release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
$ sudo apt install curl # if you haven't already installed curl
$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros
.asc | sudo apt-key add -
$ sudo apt update
$ sudo apt install ros-noetic-desktop-full ros-noetic-joy ros-noetic
-octomap-ros ros-noetic-mavlink
$ sudo apt install ros-noetic-octomap-mapping ros-noetic-control
-toolbox
$ sudo apt install python3-vcstool python3-catkin-tools
protobuf-compiler libgoogle-glog-dev
$ sudo rosdep init
$ rosdep update
$ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
$ sudo apt-get install python3-rosdep python3-wstool ros-noetic-ros
libgoogle-glog-dev
```

Si no se ha creado un *ROS workspace* con anterioridad, se puede hacer las siguientes lineas de código:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace  # initialize your catkin workspace
$ cd ~/catkin_ws/
$ catkin init
$ cd ~/catkin_ws/src
$ git clone -b dev/ros-noetic https://github.com/gsilano/rotors_
simulator.git
$ git clone -b dev/ros-noetic https://github.com/gsilano/BebopS.git
$ git clone -b med18_gazebo9 https://github.com/gsilano/mav_comm.git
$ cd ~/catkin_ws
```

El workspace se tiene que hacer con *python_catkin_tools* , por lo tanto se tiene que hacer lo siguiente:

```
$ rosdep install --from-paths src -i
$ rosdep update
$ catkin config --cmake-args -DCMAKE_BUILD_TYPE=Release
-DCATKIN_ENABLE_TESTING=False
$ catkin build
```

Anexo 3

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Float64
import math
if __name__ == '__main__':
rospy.init_node("sinus_publisher")
pub = rospy.Publisher("/sinus", Float64, queue_size=10)
x = 0
rate = rospy.Rate(20)
while not rospy.is_shutdown():
# Increase the value by 0.1 rad
x += 0.05
# Multiply the sinus value by 10
# so it's nicer for the plot example
pub.publish(4.01*math.asin(0.5*9.81*(-2*math.
cos(x/5)/25)/-4.905))
#pub.publish(4.01*math.atan((-2*math.sin(x/5)/25)/
(0+9.8)))
rate.sleep()
```

Anexo 4

Como primer paso nos ubicamos en la carpeta */src* del espacio de trabajo.

```
cd catkin_ws/src
```

Después llevamos a cabo la descarga de un repositorio alojado en *Github*

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

regresamos a nuestro espacio de trabajo y compilamos.

```
cd ~/catkin_ws
catkin_make
```

Una vez que hemos completado los requisitos basicos para la instalación de ROS, lo siguiente a realizar es la ejecucion del driver en el Dron.

Para esta parte tenemos que proceder a encender el dron, para ello presionamos el boton que se encuentra en la parte trasera del dron. Una vez presionado esperamos a que el ordenador detecte la señal de $wifi$ de nuestro cuadrirrotor que tiene por nombre $Bebop2$ seguido de unos números de identificación.

Después realizada la conexión ejecutamos el siguiente comando para obtener un panorama del estado general del dron, como la cámara, sensores, etc. A su vez, en el momento que mandemos esta orden, el cuadrirrotor estará listo para realizar los comandos que se le ingresen.

```
roslaunch bebop_driver bebop_node.launch
```

De lo anterior, se procede a enviar los comandos correspondientes en el Bebop 2. Para esta parte se probó el código básico con el cual el cuadrirrotor puede realizar una tarea, en este caso se partió con el envío de comandos para despegue y aterrizaje del Bebop 2. Con la terminal ya abierta y los comandos anteriores ya corriendo, se procede a mandar lo siguiente:

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

Con la orden anterior, el cuadrirrotor procederá a elevarse a aproximadamente 1 metro de su estado inicial y se mantendrá en $Hovering$ o vuelo estacionario hasta que se le indique algo nuevo. Para esto procedemos a enviarle lo siguiente:

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

Con esta orden, estaríamos haciendo que si el dron se encuentra en $Hovering$, realice un descenso hasta el suelo. con el cual estaríamos probando los dos comandos básicos con el cual se controla el cuadrirrotor. Finalmente estando en vuelo $Hovering$ se puede publicar mensajes del tipo:

```
geometry msgs::Twist
```