UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

Facultad de Ingeniería de Producción y Servicios

Escuela Profesional de Ingeniería Electrónica (EPIE)

Proyecto de Curso: Sistemas de Control Avanzado 2025

TIF:
ROV BLUEROV (Matlab + ROS + Gazebo)

Docente:
Prof. Juan C. Cutipa Luque (PhD)

Grupo G9_R2_SCA2025_EPIE_UNSA

Integrantes:

Nolasco Calla, Anyelo Jesús
Torres Turumpire, Octavio Isaac

Fecha: 3 de noviembre de 2025

Arequipa - Perú

Índice

1. Introducción

El presente informe corresponde al desarrollo del proyecto final del curso Sistemas de Control Avanzado (SCA 2025), enfocado en el diseño, simulación e implementación de controladores no lineales aplicados al ROV BlueROV, un vehículo submarino operado remotamente con seis grados de libertad (6 DOF). El objetivo principal del proyecto es implementar, simular y comparar distintas estrategias de control avanzado sobre un modelo dinámico no lineal del ROV, evaluando su desempeño frente a perturbaciones externas, incertidumbres paramétricas y condiciones reales de operación submarina. En particular, se desarrollan las siguientes estrategias de control: Control por Modos Deslizantes (SMC), implementado directamente en el entorno ROS + Gazebo, permitiendo la interacción en tiempo real con el modelo físico simulado del ROV y la evaluación de su robustez ante perturbaciones hidrodinámicas. Control Backstepping, implementado inicialmente en Matlab/Simulink para el diseño y validación teórica del controlador, y posteriormente integrado con ROS + Gazebo, permitiendo su ejecución sobre el modelo dinámico no lineal del BlueROV en un entorno de simulación realista. Control Backstepping implementado directamente en ROS + Gazebo, sin intermediación de Simulink, con el objetivo de evaluar su desempeño computacional y su aplicabilidad en arquitecturas de control embebidas.

2. Objetivos

2.1. Objetivo General

Diseñar, implementar, simular y comparar controladores no lineales avanzados aplicados al ROV BlueROV, basado en un modelo dinámico no lineal de seis grados de libertad (6 DOF), evaluando su desempeño, robustez y viabilidad de implementación frente a perturbaciones externas, incertidumbres paramétricas y condiciones reales de operación submarina mediante entornos Matlab/Simulink, ROS y Gazebo.

2.2. Objetivos Específicos

- Modelar dinámicamente el ROV BlueROV considerando su comportamiento no lineal, acoplamientos dinámicos y efectos hidrodinámicos relevantes para el diseño de controladores avanzados.
- Diseñar e implementar un controlador por Modos Deslizantes (SMC) en el entorno ROS + Gazebo, evaluando su capacidad de rechazo a perturbaciones y su robustez ante incertidumbres del modelo.
- Diseñar e implementar un controlador Backstepping en Matlab/Simulink, validando su estabilidad y desempeño mediante simulaciones numéricas sobre el modelo dinámico no lineal del ROV.

- Integrar el controlador Backstepping diseñado en Simulink con el entorno ROS + Gazebo, permitiendo su ejecución sobre el modelo físico simulado del BlueROV en un entorno de simulación realista.
- Implementar el controlador Backstepping directamente en ROS + Gazebo, sin intermediación de Simulink, con el fin de evaluar su desempeño computacional y su aplicabilidad en arquitecturas de control en tiempo real.
- Comparar el desempeño de los controladores implementados mediante métricas como error de seguimiento, tiempo de establecimiento, rechazo de perturbaciones y esfuerzo de control.
- Analizar la viabilidad práctica de cada estrategia de control para su futura implementación en plataformas ROV reales.

3. Articulos presentados

El estudio y control de vehículos submarinos no tripulados, como el **BlueROV2**, ha sido objeto de diversas investigaciones recientes que aportan bases sólidas para el desarrollo del presente proyecto.

3.1. An Open-Source Benchmark Simulator: Control of a BlueROV2 Underwater Robot

Autores: Malte von Benzon et al. (2022) – *Journal of Marine Science and Engineering.*

Este artículo presenta un simulador de código abierto desarrollado en MATLAB/-Simulink para el BlueROV2, basado en las ecuaciones dinámicas de Fossen. El modelo incorpora efectos físicos fundamentales:

- Cinemática y dinámica del vehículo.
- Hidrodinámica e interacción con el agua.
- Modelo detallado de propulsores.
- Fuerzas de flotabilidad y gravedad.
- Efectos de corrientes oceánicas.
- Modelado del cable (*tether*) mediante el método de masas concentradas.

El simulador fue validado experimentalmente en un tanque de pruebas, empleando un controlador por modos deslizantes para la inspección de estructuras submarinas. **Contribución principal:** Plataforma modular validada experimentalmente para comparar algoritmos de control de ROVs bajo perturbaciones reales.

3.2. Experimental Force Data of a Restrained ROV under Waves and Current

Autores: Roman Gabl et al. (2020) – Revista *Data*.

Este trabajo presenta un conjunto de datos experimentales obtenidos en el tanque de olas y corrientes FloWave (Universidad de Edimburgo). Se midieron las fuerzas hidrodinámicas sobre un BlueROV2 sujeto mediante cables instrumentados con celdas de carga, bajo:

- Corrientes de hasta 1 m/s.
- Olas regulares e irregulares.
- Efectos de sombra de un obstáculo cilíndrico a diferentes distancias.

Se utilizaron sistemas de captura de movimiento submarino y sensores de fuerza triaxiales. **Contribución principal:** Conjunto de datos público de referencia para validar modelos hidrodinámicos y estrategias de control en entornos realistas.

3.3. Model-Free High-Order Sliding Mode Controller for Station-Keeping of an AUV

Autores: Josué González-García et al. (2022) – Revista *Sensors*.

El artículo propone un controlador por modos deslizantes de alto orden, libre de modelo, aplicado al mantenimiento de posición (*station-keeping*) de un BlueROV2. Las principales características son:

- No requiere conocimiento del modelo hidrodinámico.
- Convergencia en tiempo finito ajustable por el usuario.
- Alta robustez frente a perturbaciones externas desconocidas.

Las pruebas experimentales demostraron un error cuadrático medio (RMSE) entre 1–4 cm incluso con perturbaciones de 50 N. **Contribución principal:** Controlador robusto sin modelo, validado experimentalmente, ideal para aplicaciones de mantenimiento de posición en ROVs autónomos.

4. Modelo del Sistema ROV BLUEROV

El ROV BLUEROV es un vehículo submarino con 6 grados de libertad (movimiento en surge, sway, heave, roll, pitch, yaw). Su modelo dinámico no lineal se basa en las ecuaciones de movimiento de vehículos submarinos:

$$M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) = \tau$$

donde:

- M: matriz de inercia y masa añadida.
- $C(\nu)$: matriz de Coriolis y centrífuga.
- $D(\nu)$: matriz de arrastre hidrodinámico.
- $g(\eta)$: vector de fuerzas de flotabilidad y peso.
- τ: vector de fuerzas/torques de los actuadores (hélices).

El sistema se simula considerando las constantes hidrodinámicas obtenidas del modelo BlueROV2 de *Blue Robotics*, ajustadas en Matlab/Simulink y ROS.

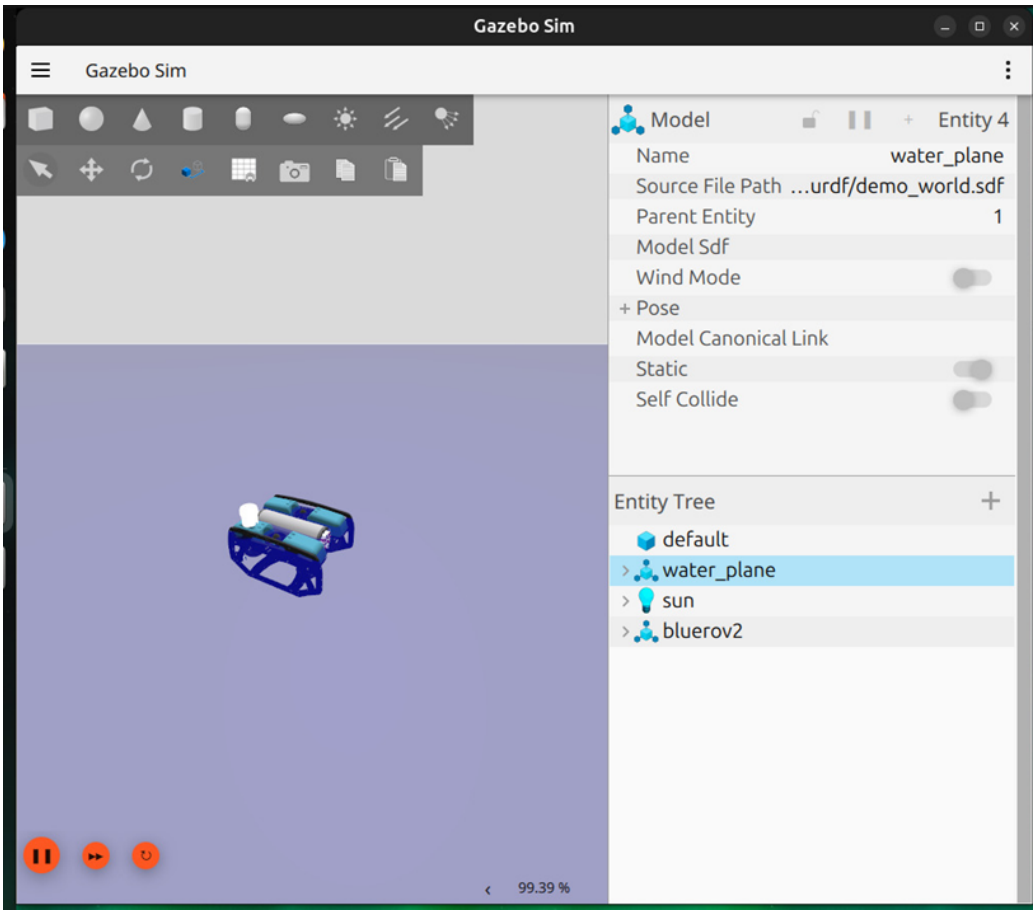

Figura 1: Simulación del entorno 3D ROV BlueROV en Matlab/Simulink y ROS.

5. Controlador Backstepping

5.1. Fundamento Teórico

Tenemos el sistema dinámico:

$$\dot{\eta} = J(\eta)\,\nu$$

$$M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) = \tau$$

$$\Rightarrow \dot{\nu} = M^{-1}\left[-(C+D)\nu - g + \tau\right]$$

Definimos el error de tracking:

$$e = \eta - \eta_d$$

Derivando:

$$\dot{e} = \dot{\eta} - \dot{\eta}_d$$

$$\dot{e} = J(\eta)\nu - \dot{\eta}_d$$

Ahora imponemos:

$$\dot{e} = -K_1 e$$

$$J(\eta)\nu - \dot{\eta}_d = -K_1 e$$

Por tanto, definimos la velocidad estabilizadora:

$$\hat{\nu} = J^{-1}(\eta)\left(\dot{\eta}_d - K_1 e\right)$$

Error extendido y Lyapunov

Definimos:

$$z = \nu - \hat{\nu}$$

Función de Lyapunov extendida:

$$V = \frac{1}{2}e^T e + \frac{1}{2}z^T z$$

Derivando:

$$\dot{V} = e^T\dot{e} + z^T\dot{z}$$

$$\dot{z} = \dot{\nu} - \dot{\hat{\nu}}$$

Reemplazando datos:

$$\dot{V} = e^T\left[J\nu - \dot{\eta}_d\right] + z^T\left[\dot{\nu} - \dot{\hat{\nu}}\right]$$

$$= e^T \left[J\nu - \dot{\eta}_d \right] + z^T \left[M^{-1} \left(-(C+D)\nu - g + \tau \right) - \dot{\hat{\nu}} \right]$$

Buscamos un w tal que $\dot{V}$ sea definida negativa:

$$w = \dot{\hat{\nu}} - Je - K_2 z$$

Pero w contiene τ. Igualamos y despejamos:

$$M^{-1} \left(-(C+D)\nu - g + \tau \right) = \dot{\hat{\nu}} - Je - K_2 z$$

$$\tau = M \left(\dot{\hat{\nu}} - Je - K_2 z \right) + (C+D)\nu + g$$

Ley de control

$$\boxed{\tau = M \left(\dot{\hat{\nu}} - Je - K_2 z \right) + (C+D)\nu + g}$$

5.2. Resultados de Simulación

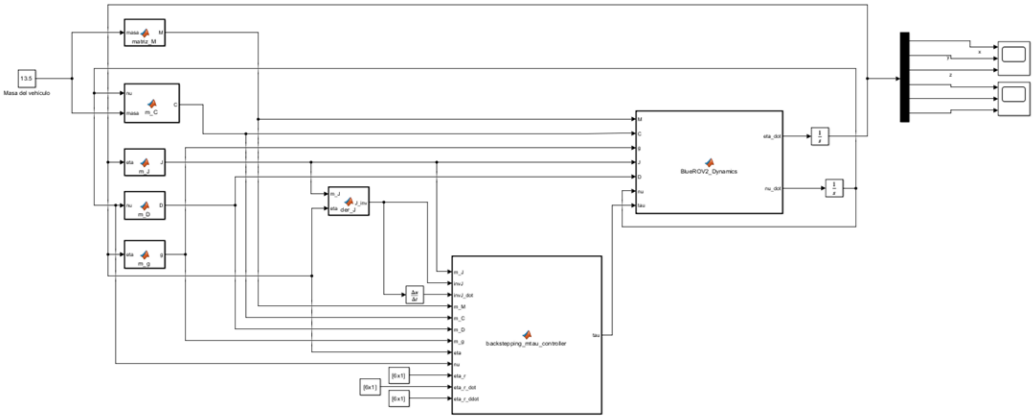

Figura 2: Controlador del sistema por Backstepping

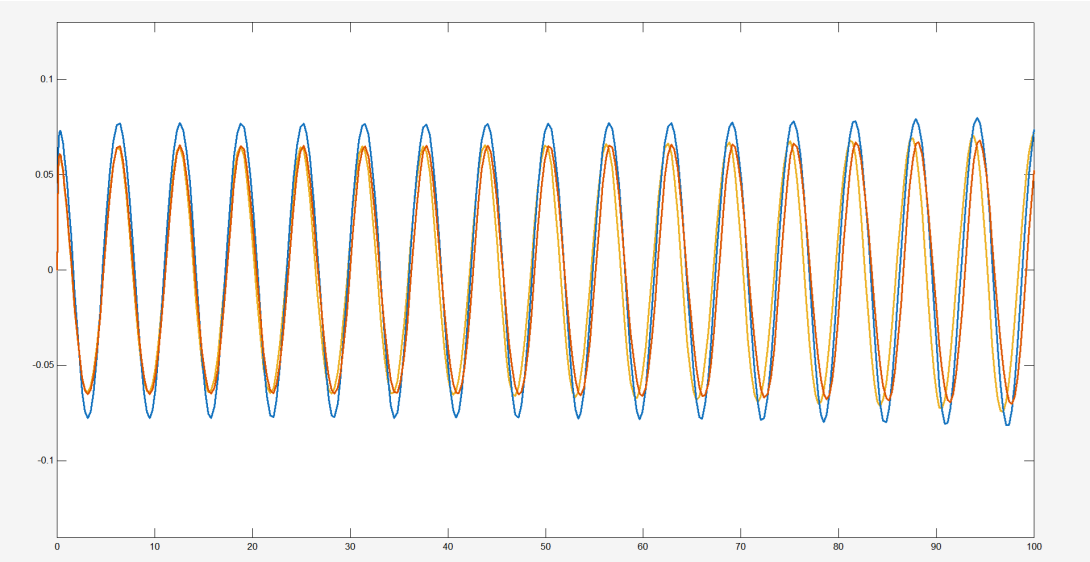

Figura 3: Posiciones lineales de la respuesta al controlador por Backstepping

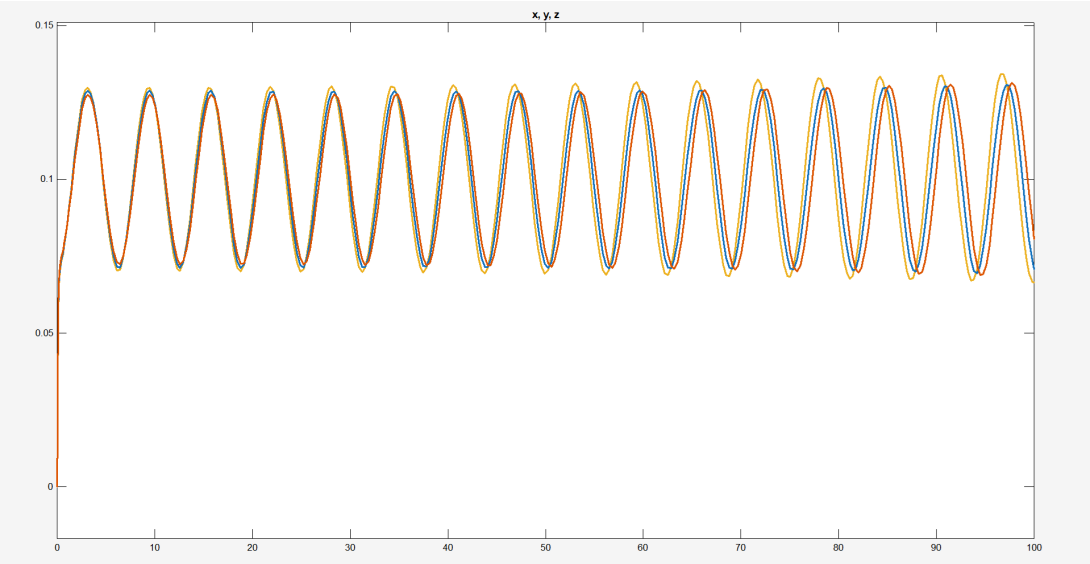

Figura 4: Posiciones angulares de la respuesta al controlador por Backstepping

5.3. Implementación del Controlador Backstepping en Matlab/Simulink + ROS 2 + Gazebo

Con el fin de validar el desempeño del controlador Backstepping bajo condiciones cercanas a la operación real, se implementó el esquema de control desarrollado en el entorno **Matlab/Simulink**, integrándolo con **ROS 2** y el simulador físico **Gazebo**. Esta integración permite ejecutar el controlador sobre el modelo dinámico no lineal del ROV BlueROV en un entorno tridimensional realista, considerando efectos hidrodinámicos y perturbaciones externas.

El controlador Backstepping se implementó en Simulink mediante bloques matemáticos que representan las ecuaciones dinámicas del sistema, el cálculo del error de seguimiento e, el error extendido z, y la ley de control:

$$\tau = M\left(\dot{\hat{\nu}} - J(\eta)e - K_2 z\right) + (C(\nu) + D(\nu))\nu + g(\eta)$$

Los valores de las matrices M, $C(\nu)$, $D(\nu)$ y $g(\eta)$ fueron definidos a partir de los parámetros hidrodinámicos del modelo BlueROV2 proporcionados por *Blue Robotics*, ajustados para su uso en Simulink y compatibles con el modelo utilizado en Gazebo.

La comunicación entre Simulink y ROS 2 se realizó mediante los bloques de *ROS Toolbox*, permitiendo:

- La suscripción a los estados del ROV (η, ν) publicados desde Gazebo a través de tópicos ROS.
- El cálculo de la señal de control τ en Simulink.
- La publicación de las fuerzas y torques de control hacia Gazebo mediante mensajes ROS, actuando directamente sobre las hélices del ROV.

El simulador Gazebo se empleó para modelar el entorno submarino tridimensional, incluyendo gravedad, flotabilidad, arrastre hidrodinámico y perturbaciones externas, mientras que ROS 2 actúa como middleware de comunicación entre el controlador y el modelo físico.

Este enfoque permitió evaluar el desempeño del controlador Backstepping en términos de seguimiento de trayectoria, estabilidad, esfuerzo de control y rechazo a perturbaciones, así como analizar su viabilidad para una futura implementación en sistemas ROV reales.

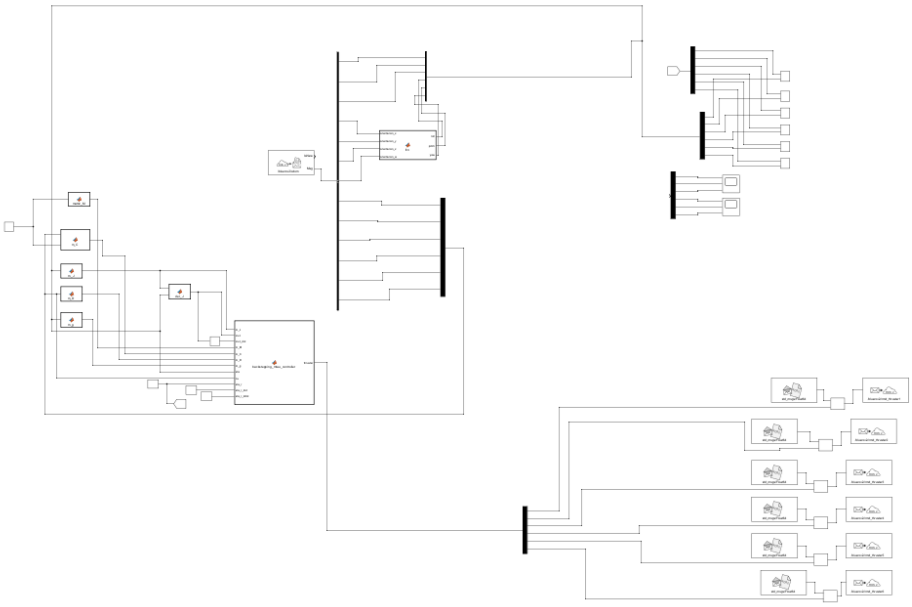

Figura 5: Esquema de integración del controlador Backstepping en Matlab/Simulink con ROS 2 y Gazebo.

5.4. Implementación del Controlador Backstepping en ROS 2 + Gazebo

```python
#!/usr/bin/env python3
"""
Backstepping 6-DOF controller para BlueROV2 (corregido para ROS2 Jazzy)
"""

import rclpy
from rclpy.node import Node
from nav_msgs.msg import Odometry
from std_msgs.msg import Float64
import numpy as np
import math
import csv
import matplotlib.pyplot as plt
import pandas as pd
import time

def quat_to_euler(q):
    x, y, z, w = q.x, q.y, q.z, q.w
    t0 = 2.0 * (w * x + y * z)
    t1 = 1.0 - 2.0 * (x * x + y * y)
    roll = math.atan2(t0, t1)
    t2 = 2.0 * (w * y - z * x)
    t2 = np.clip(t2, -1.0, 1.0)
    pitch = math.asin(t2)
    t3 = 2.0 * (w * z + x * y)
    t4 = 1.0 - 2.0 * (y * y + z * z)
    yaw = math.atan2(t3, t4)
    return roll, pitch, yaw

class BacksteppingController(Node):

    def __init__(self):
        super().__init__("backstepping_controller_backstepping")

        odom_topic = "/bluerov2/odom"
        thruster_base = "/bluerov2/cmd_thruster"

        self.thruster_pubs = [
            self.create_publisher(Float64, f"{thruster_base}{i}", 10)
            for i in range(1, 7)
        ]

        self.odom_sub = self.create_subscription(
            Odometry, odom_topic, self.odom_cb, 10
        )

        self.eta = np.zeros(6)
        self.nu = np.zeros(6)
        self.odom_received = False

```

```
        self.eta_r = np.array([5.0, 5.0, 0.0, 0.0, 0.0, 0.0])
        self.eta_r_dot = np.zeros(6)
        self.eta_r_ddot = np.zeros(6)

        self.m = 25.0
        self.Ix = 0.26
        self.Iy = 0.23
        self.Iz = 0.37

        self.Xu_dot = 5.5
        self.Yv_dot = 12.7
        self.Zw_dot = 14.57
        self.Kp_dot = 0.12
        self.Mq_dot = 0.12
        self.Nr_dot = 0.12

        self.Xu = 25.15
        self.Yv = 7.364
        self.Zw = 17.955
        self.Kp = 10.888
        self.Mq = 20.761
        self.Nr = 3.744

        self.Lambda = np.diag([10.0, 10.0, 1.0, 10.0, 1.0, 15.0])
        self.Kd = np.diag([15.0, 15.0, 4.0, 8.0, 4.0, 10.0])

        self.tmin = np.array([-40]*6)
        self.tmax = np.array([40]*6)

        L = 0.18
        c = np.cos(np.deg2rad(45))
        s = np.sin(np.deg2rad(45))
        self.B = np.array([
            [ c,  c,  c,  c,   0,  0],
            [ s, -s, -s,  s,   0,  0],
            [ 0,  0,  0,  0,   1, -1],
            [ 0,  0,  0,  0,   0,  0],
            [ 0,  0,  0,  0,   0,  0],
            [ L, -L,  L, -L,   0,  0],
        ])
        self.B_pinv = np.linalg.pinv(self.B)

        self.csv_file = open("backstepping_log.csv", "w", newline="")
        self.csv_writer = csv.writer(self.csv_file)
        self.csv_writer.writerow([
            "t","x","y","z","phi","theta","psi",
            "xd","yd","zd","phid","thetad","psid",
            "Fx","Fy","Fz","Mx","My","Mz","u0","u1","u2","u3","u4","u5"
        ])
        self.csv_file.flush()

```

```
        self.dt = 0.02
        self.t = 0.0
        self.log_counter = 0
        self.timer = self.create_timer(self.dt, self.control_loop)

    def odom_cb(self, msg):
        x = msg.pose.pose.position.x
        y = msg.pose.pose.position.y
        z = msg.pose.pose.position.z
        phi, theta, psi = quat_to_euler(msg.pose.pose.orientation)
        self.eta = np.array([x, y, z, phi, theta, psi])

        u = msg.twist.twist.linear.x
        v = msg.twist.twist.linear.y
        w = msg.twist.twist.linear.z
        p = msg.twist.twist.angular.x
        q = msg.twist.twist.angular.y
        r = msg.twist.twist.angular.z
        self.nu = np.array([u, v, w, p, q, r])
        self.odom_received = True

    def inertia_matrix(self):
        MRB = np.zeros((6,6))
        MRB[0:3,0:3] = self.m * np.eye(3)
        MRB[3:6,3:6] = np.diag([self.Ix, self.Iy, self.Iz])
        MA = np.diag([
            self.Xu_dot, self.Yv_dot, self.Zw_dot,
            self.Kp_dot, self.Mq_dot, self.Nr_dot
        ])
        return MRB + MA

    def coriolis_matrix(self, nu):
        u,v,w,p,q,r = nu
        CRB = np.zeros((6,6))
        CRB[0:3,3:6] = np.array([
            [0, self.m*w, -self.m*v],
            [-self.m*w, 0, self.m*u],
            [self.m*v, -self.m*u, 0]
        ])
        CRB[3:6,0:3] = -CRB[0:3,3:6].T

        CA = np.zeros((6,6))
        CA[0,4] = -self.Zw_dot * w
        CA[0,5] =  self.Yv_dot * v
        CA[1,3] =  self.Zw_dot * w
        CA[1,5] = -self.Xu_dot * u
        CA[2,3] = -self.Yv_dot * v
        CA[2,4] =  self.Xu_dot * u
        return CRB + CA

    def damping_matrix(self, nu):
```

```
153         return np.diag([self.Xu, self.Yv, self.Zw, self.Kp, self.Mq,
                self.Nr])
154
155     def gravity_vector(self, eta):
156         phi, theta = eta[3], eta[4]
157         W = self.m * 9.81
158         B = self.m * 9.81 * 1.02
159         z_B = 0.06
160         cphi = math.cos(phi); sphi = math.sin(phi)
161         cth = math.cos(theta); sth = math.sin(theta)
162         g = np.zeros(6)
163         g[2] = (W - B) * cphi * cth
164         g[3] = - z_B * B * sth
165         g[4] =  z_B * B * sphi * cth
166         return g
167
168     def J_matrix(self, eta):
169         phi, theta, psi = eta[3], eta[4], eta[5]
170         cphi = math.cos(phi); sphi = math.sin(phi)
171         cth = math.cos(theta); sth = math.sin(theta)
172         cps = math.cos(psi); sps = math.sin(psi)
173         R = np.array([
174             [cps*cth, cps*sth*sphi - sps*cphi, cps*sth*cphi + sps*sphi],
175             [sps*cth, sps*sth*sphi + cps*cphi, sps*sth*cphi - cps*sphi],
176             [-sth,    cth*sphi,               cth*cphi]
177         ])
178         cth_safe = max(1e-6, cth)
179         J2 = np.array([
180             [1.0, sphi * math.tan(theta),  cphi * math.tan(theta)],
181             [0.0, cphi,                   -sphi],
182             [0.0, sphi/cth_safe,          cphi/cth_safe]
183         ])
184         J = np.zeros((6,6))
185         J[0:3,0:3] = R
186         J[3:6,3:6] = J2
187         return J
188
189     def J_inv_matrix(self, eta):
190         J = self.J_matrix(eta)
191         R = J[0:3,0:3]
192         J2 = J[3:6,3:6]
193         R_inv = R.T
194         try:
195             J2_inv = np.linalg.inv(J2)
196         except:
197             J2_inv = np.linalg.pinv(J2)
198         Jinv = np.zeros((6,6))
199         Jinv[0:3,0:3] = R_inv
200         Jinv[3:6,3:6] = J2_inv
201         return Jinv
202
```

```
    def J_dot_matrix(self, eta, nu):
        eps = 1e-6
        J = self.J_matrix(eta)
        J2 = J[3:6,3:6]
        euler_rates = J2 @ nu[3:6]
        Jdot = np.zeros_like(J)
        for i in range(3):
            eta_pert = eta.copy()
            eta_pert[3+i] += eps
            J_pert = self.J_matrix(eta_pert)
            Jdot += (J_pert - J) * (euler_rates[i] / eps)
        return Jdot

    def J_inv_dot_matrix(self, eta, nu):
        J = self.J_matrix(eta)
        Jdot = self.J_dot_matrix(eta, nu)
        Jinv = self.J_inv_matrix(eta)
        return - Jinv @ Jdot @ Jinv

    def control_loop(self):
        if not self.odom_received:
            return

        eta = self.eta.copy()
        nu = self.nu.copy()

        M = self.inertia_matrix()
        C = self.coriolis_matrix(nu)
        D = self.damping_matrix(nu)
        g = self.gravity_vector(eta)

        J = self.J_matrix(eta)
        Jinv = self.J_inv_matrix(eta)
        Jinv_dot = self.J_inv_dot_matrix(eta, nu)

        e1 = eta - self.eta_r
        e1_dot = J @ nu - self.eta_r_dot

        nu_r = Jinv @ (self.eta_r_dot - self.Lambda @ e1)

        nu_r_dot = Jinv_dot @ (self.eta_r_dot - self.Lambda @ e1) + \
                   Jinv @ (self.eta_r_ddot - self.Lambda @ e1_dot)

        e2 = nu - nu_r

        tau = M @ (nu_r_dot - J @ e1 - self.Kd @ e2) + C @ nu + D @ nu +
            g

        u = self.B_pinv @ tau
        u_sat = np.minimum(np.maximum(u, self.tmin), self.tmax)

```

```
253         try:
254             row = [self.t] + list(eta) + list(self.eta_r) + \
255                   list(tau) + list(u_sat)
256             self.csv_writer.writerow(row)
257             self.csv_file.flush()
258         except:
259             pass
260 
261         self.t += self.dt
262 
263         for i, pub in enumerate(self.thruster_pubs):
264             msg = Float64()
265             msg.data = float(u_sat[i])
266             pub.publish(msg)
267 
268     def plot_results(self):
269         try:
270             df = pd.read_csv("backstepping_log.csv")
271         except:
272             return
273         if df.empty:
274             return
275 
276         t = df["t"]
277         x = df["x"]; xd = df["xd"]
278         y = df["y"]; yd = df["yd"]
279         z = df["z"]; zd = df["zd"]
280         yaw = df["psi"]; yawd = df["psid"]
281 
282         u0 = df["u0"]; u1 = df["u1"]
283         u2 = df["u2"]; u5 = df["u5"]
284 
285         plt.figure(figsize=(10,10))
286         plt.subplot(5,1,1); plt.plot(t,x); plt.plot(t,xd,"--")
287         plt.subplot(5,1,2); plt.plot(t,y); plt.plot(t,yd,"--")
288         plt.subplot(5,1,3); plt.plot(t,z); plt.plot(t,zd,"--")
289         plt.subplot(5,1,4); plt.plot(t,yaw); plt.plot(t,yawd,"--")
290         plt.subplot(5,1,5);
291         plt.plot(t,u0); plt.plot(t,u1); plt.plot(t,u2); plt.plot(t,u5)
292         plt.tight_layout(); plt.show()
293 
294 def main(args=None):
295     rclpy.init(args=args)
296     node = BacksteppingController()
297     try:
298         rclpy.spin(node)
299     except KeyboardInterrupt:
300         node.csv_file.close()
301         node.plot_results()
302     node.destroy_node()
303     rclpy.shutdown()
```

```
304
305 if __name__ == "__main__":
306     main()
```

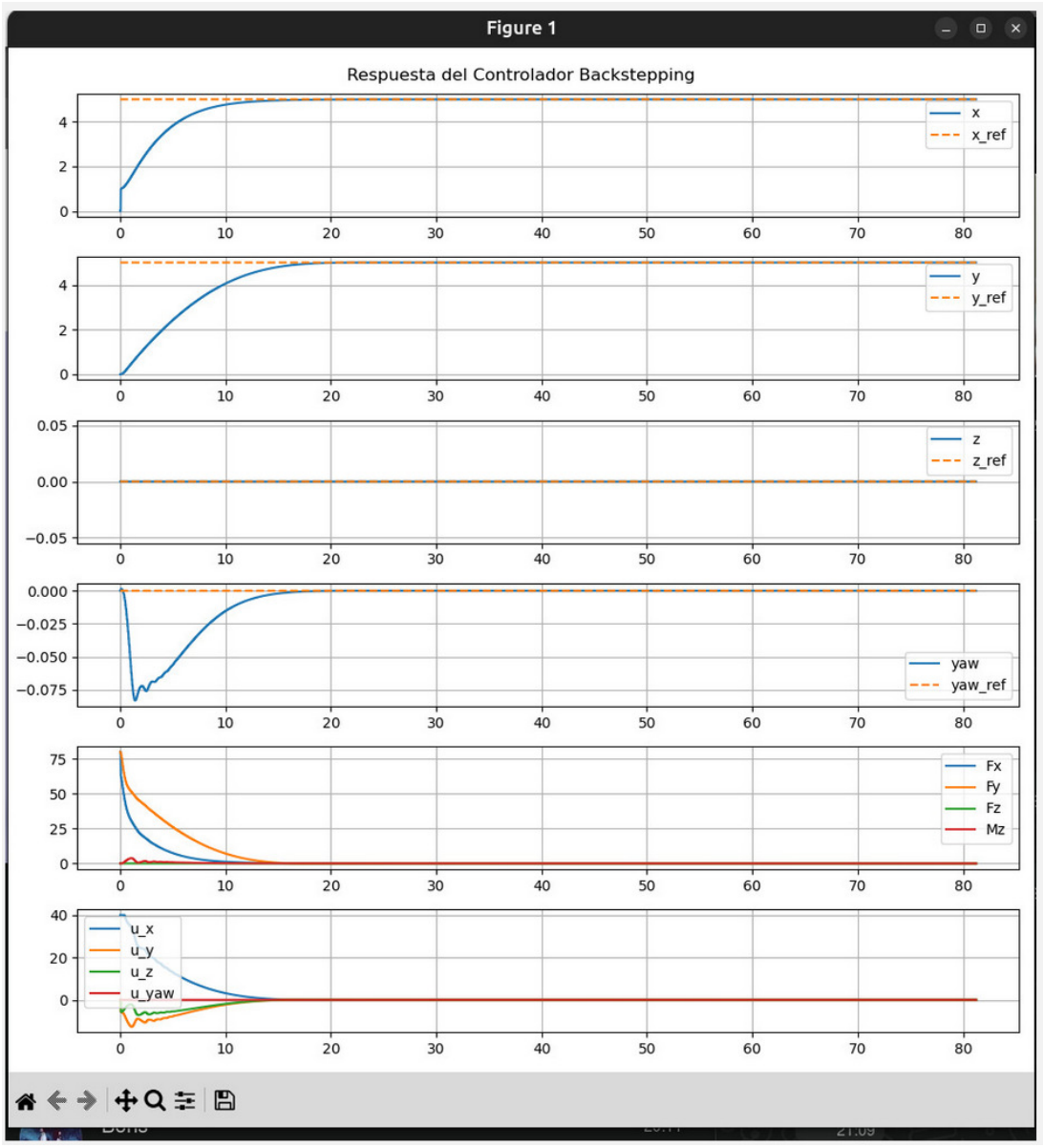

Figura 6: Vistas del eje x, y, z, yaw, Fuerzas/ torques y entrada a los trhusters

Cuadro 1: Cuadro comparativo del controlador Backstepping

Variable	**Referencia**	**Respuesta Observada**	**Evaluación**
Posición X	5 m	Convergencia suave con pequeño undershoot; estabiliza en ∼15 s.	Bueno
Posición Y	5 m	Comportamiento similar a X, un poco más lenta pero estable.	Bueno
Posición Z	0 m	Se mantiene estable sin desviaciones apreciables.	Excelente
Yaw (ψ)	0 rad	Sobreimpulso inicial evidente; amortigua y estabiliza en ∼20 s.	Mejorable
Fuerzas / Momentos	—	Picos iniciales altos (acción correctiva fuerte), luego descienden a 0.	Adecuado
Comandos del Control (u)	—	Magnitudes elevadas al inicio que se atenúan rápidamente.	Adecuado

6. Controlador por Modos Deslizantes (SMC)

6.1. Fundamento Teórico

El control por **Modos Deslizantes** (SMC) proporciona robustez frente a incertidumbres y perturbaciones. Se define la superficie de deslizamiento:

$$s = \dot{e} + \lambda e$$

La ley de control propuesta es:

$$\tau_z = m(\dot{z}_d - \lambda e - k\,\mathrm{sign}(s)) + D_z v_z + g_z(\eta)$$

6.2. Implementación del Controlador Modos Deslizantes en ROS 2 + Gazebo

```
#!/usr/bin/env python3
"""
Controlador SMC (modo deslizante) para BlueROV2
Adaptado para ROS2 Jazzy + gz-sim.

Incluye:
 - Logging autom tico en smc_log.csv (t, x,y,z,yaw, refs, Fx,Fy,Fz,Mz, u0..u5)
 - Gr ficas autom ticas al presionar Ctrl+C
"""

import rclpy
```

```python
from rclpy.node import Node
from nav_msgs.msg import Odometry
from std_msgs.msg import Float64
import numpy as np
import math
import csv
import matplotlib.pyplot as plt
import pandas as pd


def quat_to_yaw(q):
    """Convierte quaternion     yaw."""
    siny_cosp = 2.0 * (q.w * q.z + q.x * q.y)
    cosy_cosp = 1.0 - 2.0 * (q.y * q.y + q.z * q.z)
    return math.atan2(siny_cosp, cosy_cosp)


class SlidingModeController(Node):

    def __init__(self):
        super().__init__("sliding_mode_controller")

        odom_topic = "/bluerov2/odom"
        thruster_base = "/bluerov2/cmd_thruster"

        self.thruster_pubs = [
            self.create_publisher(Float64, f"{thruster_base}{i}", 10)
            for i in range(1, 7)
        ]

        self.odom_sub = self.create_subscription(
            Odometry, odom_topic, self.odom_cb, 10
        )

        self.get_logger().info(f"    Suscrito a: {odom_topic}")
        self.get_logger().info(f"    Publicando a: {thruster_base}X")

        self.pos = np.zeros(3)
        self.vel = np.zeros(3)
        self.yaw = 0.0
        self.yaw_rate = 0.0

        # Setpoints (puedes modificarlos din micamente)
        self.xd = 5.0
        self.yd = 5.0
        self.zd = 0.0
        self.yawd = 1.0

        # Modelo (m, Izz)
        self.m = 25.0
        self.Izz = 13.0
```

```
63
64          # Ganancias PD nominales (componente equivalente)
65          self.kp_x = 15.0
66          self.kd_x = 11.0
67
68          self.kp_y = 15.0
69          self.kd_y = 11.0
70
71          self.kp_z = 0.4
72          self.kd_z = 60.0
73
74          self.kp_yaw = 15.0
75          self.kd_yaw = 11.0
76
77          # ===== Par metros SMC (ajustables) =====
78          self.lambda_x = 1.5
79          self.lambda_y = 1.5
80          self.lambda_z = 2.0
81          self.lambda_yaw = 2.0
82
83          self.eta_x = 60.0
84          self.eta_y = 60.0
85          self.eta_z = 200.0
86          self.eta_yaw = 10.0
87
88          self.phi_x = 0.2
89          self.phi_y = 0.2
90          self.phi_z = 0.5
91          self.phi_yaw = 1.0
92
93          # l mites thrusters
94          self.tmin = np.array([-40, -40, -40, -40, -40, -40], float)
95          self.tmax = np.array([ 40,  40,  40,  40,  40,  40], float)
96
97          # Matriz de asignaci n thrusters
98          L = 0.18
99          c = np.cos(np.deg2rad(45))
100         s = np.sin(np.deg2rad(45))
101
102         self.B = np.array([
103             [ c,  c,  c,  c, 0, 0],    # Fx
104             [ s, -s, -s,  s, 0, 0],    # Fy
105             [ 0,  0,  0,  0, -1, 1],   # Fz
106             [ 0,  0,  0,  0, 0, 0],    # Mx
107             [ 0,  0,  0,  0, 0, 0],    # My
108             [ L, -L,  L, -L, 0, 0],    # Mz
109         ], float)
110
111         self.B_pinv = np.linalg.pinv(self.B)
112
113         # contador para limitar logs
```

```
114         self.log_counter = 0
115
116         # ============================
117         #      CSV LOG AUTOM TICO
118         # ============================
119         # Abrimos el archivo desde el inicio y escribimos cabecera (
                coincide con columnas usadas)
120         self.csv_file = open("smc_log.csv", "w", newline="")
121         self.csv_writer = csv.writer(self.csv_file)
122         self.csv_writer.writerow([
123             "t",
124             "x", "y", "z", "yaw",
125             "xd", "yd", "zd", "yawd",
126             "Fx", "Fy", "Fz", "Mz",
127             "u0", "u1", "u2", "u3", "u4", "u5"
128         ])
129         self.csv_file.flush()
130
131         # tiempo simulado / muestreo
132         self.t = 0.0
133         self.dt = 0.02  # 50 Hz
134
135         self.timer = self.create_timer(self.dt, self.control_loop)  # 50
                Hz
136
137
138     def odom_cb(self, msg: Odometry):
139         self.pos[0] = msg.pose.pose.position.x
140         self.pos[1] = msg.pose.pose.position.y
141         self.pos[2] = msg.pose.pose.position.z
142
143         self.vel[0] = msg.twist.twist.linear.x
144         self.vel[1] = msg.twist.twist.linear.y
145         self.vel[2] = msg.twist.twist.linear.z
146
147         q = msg.pose.pose.orientation
148         self.yaw = quat_to_yaw(q)
149         self.yaw_rate = msg.twist.twist.angular.z
150
151     def wrap(self, a):
152         return (a + math.pi) % (2 * math.pi) - math.pi
153
154     def sat(self, x, phi):
155         """Saturaci n suave: tanh(x/phi). phi>0"""
156         if phi <= 0.0:
157             return np.sign(x)  # fallback (no smoothing)
158         return np.tanh(x / phi)
159
160     def control_loop(self):
161
162         # Errores de posici n
```

```
        ex = self.xd - self.pos[0]
        ey = self.yd - self.pos[1]
        ez = self.zd - self.pos[2]

        # Errores en velocidad (setpoint vel = 0)
        evx = -self.vel[0]
        evy = -self.vel[1]
        evz = -self.vel[2]

        eyaw = self.wrap(self.yawd - self.yaw)
        eyaw_rate = -self.yaw_rate

        # Superficies de deslizamiento
        sx = evx + self.lambda_x * ex
        sy = evy + self.lambda_y * ey
        sz = evz + self.lambda_z * ez
        syaw = eyaw_rate + self.lambda_yaw * eyaw

        # Componente equivalente (PD nominal)
        Fx_eq = self.m * (self.kp_x * ex + self.kd_x * evx)
        Fy_eq = self.m * (self.kp_y * ey + self.kd_y * evy)
        Fz_eq = self.m * (self.kp_z * ez + self.kd_z * evz)
        Mz_eq = self.Izz * (self.kp_yaw * eyaw + self.kd_yaw * eyaw_rate
            )

        # T rmino robusto SMC (suavizado con tanh)
        Fx_robust = - self.eta_x * self.sat(sx, self.phi_x)
        Fy_robust = - self.eta_y * self.sat(sy, self.phi_y)
        Fz_robust = - self.eta_z * self.sat(sz, self.phi_z)
        Mz_robust = - self.eta_yaw * self.sat(syaw, self.phi_yaw)

        # Ley total
        Fx = Fx_eq + Fx_robust
        Fy = Fy_eq + Fy_robust
        Fz = Fz_eq + Fz_robust
        Mz = Mz_eq + Mz_robust

        tau = np.array([Fx, Fy, Fz, 0.0, 0.0, Mz])
        u = self.B_pinv @ tau
        u = np.minimum(np.maximum(u, self.tmin), self.tmax)

        # ==============================
        #   LOG DE POSICI N, FUERZAS Y COMANDOS (cada ciclo)
        # ==============================
        try:
            self.csv_writer.writerow([
                round(self.t, 4),
                float(self.pos[0]), float(self.pos[1]), float(self.pos
                    [2]), float(self.yaw),
                float(self.xd), float(self.yd), float(self.zd), float(
                    self.yawd),
```

```
                float(Fx), float(Fy), float(Fz), float(Mz),
                float(u[0]), float(u[1]), float(u[2]), float(u[3]),
                    float(u[4]), float(u[5])
            ])
            # Forzamos a disco (evita archivo vac o si cortas r pido)
            self.csv_file.flush()
        except Exception as e:
            self.get_logger().warning(f"CSV write error: {e}")

        # incrementamos tiempo nominal
        self.t += self.dt

        # ============================
        #   LOGS por consola (limitados)
        # ============================
        if self.log_counter % 10 == 0:  # cada 10 ciclos (~5 Hz)
            self.get_logger().info(
                "\n"
                f"POS    = [{self.pos[0]:.2f}, {self.pos[1]:.2f}, {self.
                    pos[2]:.2f}, {self.yaw:.2f}]\n"
                f"SP     = [{self.xd:.2f}, {self.yd:.2f}, {self.zd:.2f},
                    {self.yawd:.2f}]\n"
                f"ERR    = [{ex:.2f}, {ey:.2f}, {ez:.2f}, {eyaw:.2f}]\n"
                f"S      = [{sx:.3f}, {sy:.3f}, {sz:.3f}, {syaw:.3f}]\n"
                f"TAU    = [{Fx:.2f}, {Fy:.2f}, {Fz:.2f}, {Mz:.2f}]\n"
                f"THR    = [{u[0]:.2f}, {u[1]:.2f}, {u[2]:.2f}, {u[3]:.2f
                    }, {u[4]:.2f}, {u[5]:.2f}]"
            )

        self.log_counter += 1

        # Publicar en cada thruster
        for i, pub in enumerate(self.thruster_pubs):
            msg = Float64()
            msg.data = float(u[i])
            pub.publish(msg)

    # ============================================================
    #          GRAFICADOR AUTOM TICO AL DETENER CONTROLADOR
    # ============================================================
    def plot_results(self):
        # Usamos pandas para conveniencia
        try:
            df = pd.read_csv("smc_log.csv")
        except Exception as e:
            self.get_logger().error(f"No se pudo leer smc_log.csv: {e}")
            return

        if df.empty:
            self.get_logger().warning("smc_log.csv est   vac o      no
                hay datos para graficar.")
```

```
            return

        t = df["t"]
        x = df["x"]
        y = df["y"]
        z = df["z"]
        yaw = df["yaw"]
        xd = df["xd"]
        yd = df["yd"]
        zd = df["zd"]
        yawd = df["yawd"]

        Fx = df["Fx"]
        Fy = df["Fy"]
        Fz = df["Fz"]
        Mz = df["Mz"]

        u0 = df["u0"]
        u1 = df["u1"]
        u2 = df["u2"]
        u3 = df["u3"]
        u4 = df["u4"]
        u5 = df["u5"]

        plt.figure(figsize=(10, 12))

        plt.subplot(5,1,1)
        plt.plot(t, x, label="x")
        plt.plot(t, xd, "--", label="x_ref")
        plt.ylabel("X [m]")
        plt.grid(True)
        plt.legend()

        plt.subplot(5,1,2)
        plt.plot(t, y, label="y")
        plt.plot(t, yd, "--", label="y_ref")
        plt.ylabel("Y [m]")
        plt.grid(True)
        plt.legend()

        plt.subplot(5,1,3)
        plt.plot(t, z, label="z")
        plt.plot(t, zd, "--", label="z_ref")
        plt.ylabel("Z [m]")
        plt.grid(True)
        plt.legend()

        plt.subplot(5,1,4)
        plt.plot(t, yaw, label="yaw")
        plt.plot(t, yawd, "--", label="yaw_ref")
        plt.ylabel("Yaw [rad]")
```

```
308         plt.grid(True)
309         plt.legend()
310 
311         plt.subplot(5,1,5)
312         plt.plot(t, u0, label="u0")
313         plt.plot(t, u1, label="u1")
314         plt.plot(t, u2, label="u2")
315         plt.plot(t, u3, label="u3")
316         plt.plot(t, u4, label="u4")
317         plt.plot(t, u5, label="u5")
318         plt.ylabel("Thruster cmds")
319         plt.xlabel("Tiempo [s]")
320         plt.grid(True)
321         plt.legend()
322 
323         plt.suptitle("Respuesta del SMC")
324         plt.tight_layout()
325         plt.show()
326 
327 
328 def main(args=None):
329     rclpy.init(args=args)
330     node = SlidingModeController()
331 
332     try:
333         rclpy.spin(node)
334 
335     except KeyboardInterrupt:
336         print("\n\n>>> CONTROLADOR DETENIDO     Generando gr ficas...\n
            ")
337         # cerramos CSV y luego graficamos
338         try:
339             node.csv_file.close()
340         except Exception:
341             pass
342         node.plot_results()
343 
344     node.destroy_node()
345     rclpy.shutdown()
346 
347 
348 if __name__ == "__main__":
349     main()
```

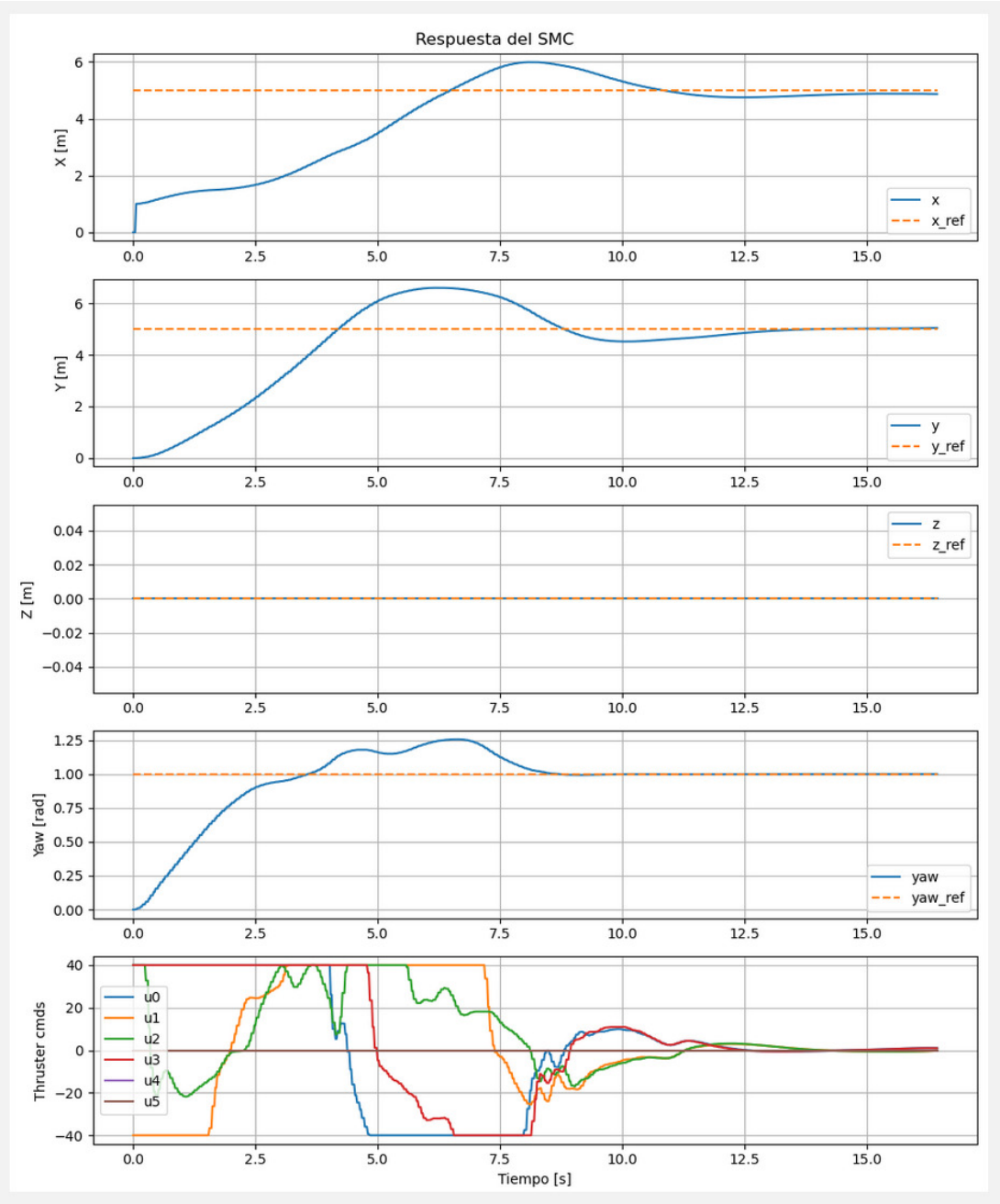

Figura 7: Vistas del eje x, y, z, yaw, Fuerzas/ torques y entrada a los trhusters

Cuadro 2: Cuadro comparativo del Controlador SMC

Variable	**Referencia**	**Respuesta Observada**	**Evaluación**
Posición X	5 m	Sobreimpulso significativo (6 m), luego converge lentamente hacia la referencia.	Mejorable
Posición Y	5 m	También presenta sobreimpulso (6 m) y caída posterior antes de estabilizar.	Mejorable
Posición Z	0 m	Se mantiene prácticamente constante sin error; excelente estabilidad.	Excelente
Yaw (ψ)	1.08 rad	Sobreimpulso inicial, oscilación leve y estabilización posterior.	Adecuado
Thruster cmds	—	Variaciones bruscas típicas del SMC; alta actividad y discontinuidades visibles.	Control robusto pero alto chattering

7. Conclusiones

- El controlador Backstepping proporcionó una respuesta más suave y estable, con menor esfuerzo de los thrusters, siendo adecuado para movimientos precisos y condiciones sin perturbaciones fuertes.
- El controlador por Modos Deslizantes (SMC) mostró mayor robustez frente a incertidumbres y perturbaciones, logrando estabilización rápida, aunque con mayor actividad y oscilación en los actuadores.
- Ambos controladores pudieron implementarse correctamente como nodos ROS2 Jazzy, demostrando modularidad, facilidad de integración y buen desempeño en el simulador gz-sim.
- La comunicación MATLAB/Simulink ROS2 permitió validar y comparar los controladores en tiempo real, brindando un entorno de pruebas flexible y confiable.
- La arquitectura desarrollada es escalable y lista para su futura implementación en un BlueROV2 real, con mínimos cambios al sistema de control.

7.1. Enlace a GitHub

Repositorio del proyecto: `https://github.com/otorrest18/BLUEROV2_UNSA_2025B_SCA`

8. Bibliografía

Referencias

[1] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. Hoboken, NJ, USA: Wiley, 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/9781119994138

[2] T. I. Fossen, *Guidance and Control of Ocean Vehicles*. Chichester, U.K.: John Wiley & Sons, 1994. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/9780470015585

[3] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*. Englewood Cliffs, NJ, USA: Prentice Hall, 1991. [Online]. Available: https://www.pearson.com/en-us/subject-catalog/p/applied-nonlinear-control/P200000006274

[4] V. I. Utkin, *Sliding Modes in Control and Optimization*. Berlin, Germany: Springer, 1992. [Online]. Available: https://link.springer.com/book/10.1007/978-3-642-84379-2

[5] M. Krstić, I. Kanellakopoulos, and P. Kokotović, *Nonlinear and Adaptive Control Design*. New York, NY, USA: Wiley, 1995. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/9780470610803

[6] MathWorks, "MATLAB and Simulink Documentation." [Online]. Available: https://www.mathworks.com/help. Accessed: 2025.

[7] Open Robotics, "ROS 2 Documentation." [Online]. Available: https://docs.ros.org. Accessed: 2025.

[8] Open Robotics, "Gazebo Simulator Documentation." [Online]. Available: https://gazebosim.org. Accessed: 2025.

[9] Blue Robotics, "BlueROV2 Technical Documentation." [Online]. Available: https://bluerobotics.com/store/rov/bluerov2/. Accessed: 2025.