[1]

Exploración y despliegue de algoritmos interactivos en Docker: simulaciones con seguidor de línea, videojuego 2D y entorno ROS

Dikersson Alexis Cañon Vanegas, Dikerssoncanon@usantotomas.edu.co
Miguel Angel Jimenez Morales,miguel.jimenezm@usantotomas.edu.co

Este proyecto integra tres enfoques aplicados a la simulación computacional interactiva: (1) un algoritmo PID aplicado a un vehículo seguidor de línea con interfaz gráfica en Tkinter, (2) un videojuego tipo "Space Invaders" programado en Pygame, y (3) una demostración de nodos ROS en contenedores Docker, utilizando la herramienta turtlesim como entorno visual. Todos los algoritmos fueron personalizados, ejecutados de forma contenida y documentados conforme a lineamientos de despliegue reproducible con integración a GitHub y Docker Hub.

Índice de Términos - ROS, Docker, github, Videojuegos.

I. INTRODUCCIÓN

En la actualidad, el uso de contenedores Docker y repositorios distribuidos como GitHub representa una práctica esencial para el desarrollo, la validación y la portabilidad de aplicaciones tecnológicas. En este contexto, se propone una actividad de integración de conocimientos adquiridos en programación, control de procesos y sistemas distribuidos, a través de la implementación y personalización de tres algoritmos visuales interactivos.

La actividad se divide en tres partes complementarias: un sistema de control PID para seguimiento de línea, un videojuego 2D de naves espaciales, y una simulación de comunicación entre nodos en ROS (Robot Operating System). Cada uno de estos escenarios representa un paradigma distinto de interacción y ejecución de procesos: control en lazo cerrado, interacción usuario-juego y sistemas distribuidos basados en mensajes.

El trabajo no solo busca evaluar el entendimiento técnico de los algoritmos y su estructura, sino también la capacidad del estudiante para personalizarlos, desplegarlos correctamente en contenedores y documentar su funcionamiento mediante buenas prácticas de ingeniería de software.

II. OBJETIVO GENERAL

Desplegar algoritmos interactivos mediante el uso de Python, Docker y GitHub para aplicar conceptos de control, visualización y ejecución contenida en entornos simulados.

III. OBJETIVOS ESPECÍFICOS

- Implementar un algoritmo PID para seguimiento de línea con interfaz gráfica en Tkinter.
- Personalizar un videojuego 2D en Pygame utilizando sprites y lógica de colisiones.
- Ejecutar un ejemplo básico de ROS en Docker verificando el funcionamiento de nodos publisher y subscriber.
- Documentar cada algoritmo en un archivo README.md con estructura clara y estilos consistentes.
- Subir los proyectos a GitHub y Docker Hub con enlaces funcionales y organización estructurada.

.

IV. MARCOS TEORICO

El presente proyecto integra conceptos fundamentales en el área de ingeniería de sistemas, control y robótica, mediante el desarrollo de tres escenarios computacionales que representan diferentes arquitecturas de ejecución y simulación.

Control PID:
El controlador Proporcional–Integral–Derivativo (PID) es uno de los más utilizados en sistemas de control debido a su simplicidad y eficacia. Permite corregir el error entre una variable de referencia y una señal de salida, sumando tres acciones: una proporcional al error, una integral que acumula errores pasados y una derivativa que anticipa la tendencia del error. Este controlador se aplica en el simulador de seguimiento de línea, permitiendo que el vehículo ajuste su trayectoria en tiempo real con base en la lectura de sensores virtuales.

Interfaz gráfica con Tkinter:
Tkinter es una librería estándar de Python para crear interfaces gráficas. Se emplea en el simulador de seguimiento de línea para visualizar el movimiento del carro y la pista, actualizando dinámicamente la posición y rotación del

[1]

vehículo con cada iteración del algoritmo.

Programación de videojuegos con Pygame:
Pygame es una biblioteca orientada a la creación de videojuegos 2D en Python. Su estructura basada en sprites, grupos, eventos y bucles permite construir juegos interactivos que manejan múltiples entidades y colisiones. En este proyecto, se desarrolla un juego tipo "Space Invaders" con múltiples niveles, un jefe final, power-ups y gestión de vidas, lo que permite aplicar conocimientos de programación orientada a objetos y lógica condicional.

ROS (Robot Operating System):
ROS es un middleware que facilita la comunicación entre diferentes nodos o programas en sistemas robóticos distribuidos. Su arquitectura basada en nodos, topics y mensajes promueve la modularidad, permitiendo que múltiples procesos se comuniquen de manera asincrónica. En el presente proyecto se emplea el ejemplo clásico `talker-listener` y la herramienta `turtlesim` para mostrar visualmente la interacción entre nodos dentro de un contenedor Docker, validando el correcto funcionamiento de la red ROS.

Docker:
Docker permite ejecutar aplicaciones en contenedores aislados, asegurando la portabilidad del software y evitando conflictos entre dependencias. Todos los algoritmos del proyecto fueron desplegados en entornos Docker, lo que garantiza la replicabilidad del experimento en cualquier equipo que soporte contenedores.

V. Procedimiento

A. Seguidor de línea.

Para desarrollar el simulador del vehículo seguidor de línea se implementó una interfaz gráfica en Tkinter y se aplicó un algoritmo de control PID para ajustar la dirección del vehículo en función de la lectura de dos sensores virtuales.

1. **Configuración del entorno gráfico:**
   Se utilizó la librería `tkinter` para crear una ventana de 800x600 píxeles con una pista rectangular y un checkpoint circular. El entorno fue diseñado con colores contrastantes para facilitar la detección de la línea por parte de los sensores.

2. **Diseño del vehículo:**
   El vehículo fue representado como un polígono con dos ruedas y dos sensores delanteros (izquierdo y derecho), cuya posición se actualiza en función del ángulo y la posición general del carro. La estructura se compone de objetos gráficos (`create_polygon`, `create_oval`) que se transforman en cada ciclo de actualización.

3. **Lectura de sensores:**
   Cada sensor virtual consulta si está sobre la pista (rectángulo) o sobre un checkpoint mediante la función `canvas.find_overlapping`, lo que permite identificar si se detecta la línea de seguimiento.

4. **Cálculo del error y control PID:**
   La diferencia entre la activación de los sensores se interpreta como un error. Este error es procesado por un controlador PID personalizado con ganancias ajustadas manualmente (`Kp=1.9`, `Ki=0.001`, `Kd=0.8`). El resultado se usa para modificar el ángulo del vehículo, simulando un comportamiento realista de giro.

5. **Actualización del movimiento:**
   En cada iteración, se calcula el nuevo ángulo del carro y su desplazamiento en el plano. La función `move()` es ejecutada periódicamente por `after(30, game_loop)`, lo que permite una animación fluida y continua del seguimiento de la pista.

6. **Visualización del rastro:**
   Se dibujan pequeñas marcas en cada posición del vehículo para visualizar el recorrido. Esto permite analizar el comportamiento del control PID y ajustar parámetros si es necesario.

Este algoritmo está contenido en un archivo Python y puede ejecutarse de forma independiente o dentro de un contenedor Docker. El código fuente se encuentra debidamente comentado y será anexado como parte del informe.

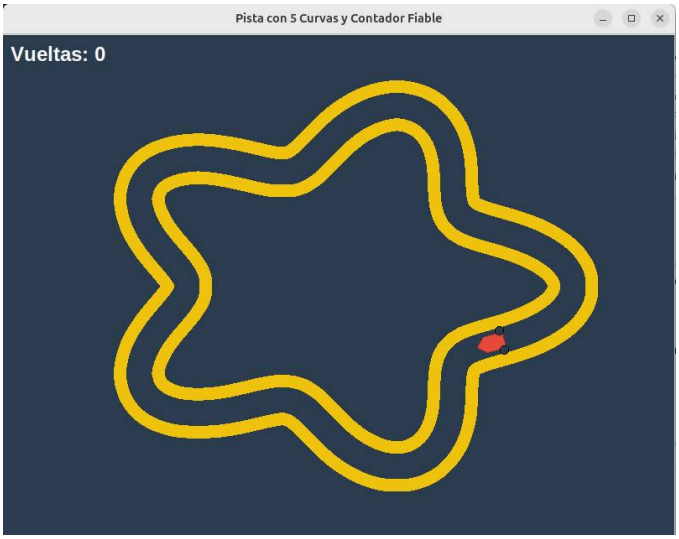

Fig 1. Seguidor de linea en forma de estrella.

B. Naves espaciales

El videojuego "SpaceMax Defender" fue desarrollado utilizando la librería `pygame`, con una estructura modular basada en clases, eventos, grupos de sprites y bucles de juego. El objetivo del juego es eliminar oleadas de enemigos y enfrentar un jefe final, incorporando mecánicas como escudos, disparos, power-ups y control de vidas.

1. **Inicialización del entorno y verificación de recursos:**
   Se utilizó `os` y `sys` para definir la carpeta base del script y verificar la existencia de la carpeta `images`. Esta verificación es fundamental para asegurar la carga correcta de los sprites personalizados. Si una imagen no se encuentra, el sistema genera un reemplazo visual con color de respaldo (fallback).

2. **Carga de sprites y assets:**
   Las imágenes del jugador, enemigos y jefe fueron cargadas mediante una función `load_img()` que escala y convierte las imágenes según el tamaño deseado. También se creó un fondo dinámico de estrellas con `pygame.draw.circle()` para dar efecto espacial.

3. **Definición de clases:**
   - `Player`: controla el movimiento lateral del jugador, la activación del escudo (tecla `S`), y el sistema de vidas y score.
   - `Enemy`: representa a los enemigos comunes, con desplazamiento horizontal, colisiones y disparos.
   - `Boss`: se activa en el nivel 4, tiene más vida y dispara en abanico.
   - `Bullet` y `EnemyBullet`: proyectiles del jugador y enemigos, respectivamente.
   - `Explosion` y `PowerUp`: efectos visuales de explosión y mejoras que otorgan vidas adicionales.

4. **Manejo de eventos:**
   Se controlan eventos como movimiento (`LEFT`, `RIGHT`) y disparo (`SPACE`). También se manejan eventos de colisión entre balas y enemigos, entre balas y el jefe, y entre enemigos y el jugador (si no tiene escudo activo).

5. **Sistema de niveles y dificultad:**
   A medida que se eliminan los enemigos, se avanza a un nuevo nivel hasta llegar al jefe final (nivel 4). Cada nivel aumenta la velocidad y la agresividad de los enemigos. Si se derrota al jefe, el juego finaliza con éxito.

6. **HUD y visualización:**
   Se muestra en pantalla el nivel actual, la puntuación y las vidas restantes mediante íconos de corazones. En el nivel final se muestra una barra de vida del jefe. También se dibuja un escudo circular azul cuando el jugador activa su defensa.

7. **Control de ejecución:**
   Todo el juego corre dentro de un bucle principal (`while running:`), con un `clock.tick(60)` para asegurar 60 FPS estables. Al final del juego, se cierra Pygame y se sale del programa con `pygame.quit()` y `sys.exit()`.

El código fuente del juego se encuentra documentado y será anexado al informe. Las imágenes utilizadas deben ubicarse en la carpeta `images/` para que el juego funcione correctamente.

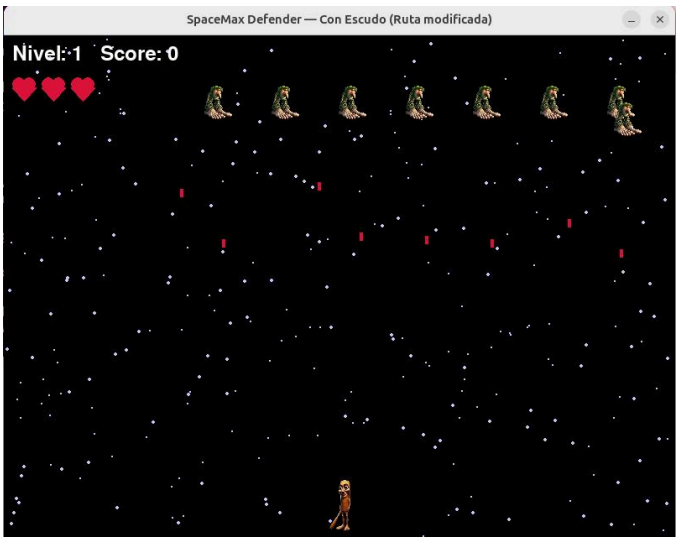

Fig 2. Resultado final del juego.

Fig 3. Organización de las carpetas.

C. ROS

La tercera parte del proyecto consistió en la ejecución de un entorno básico de ROS (Robot Operating System) dentro de un contenedor Docker. El objetivo fue validar la arquitectura distribuida de ROS mediante el ejemplo clásico `talker-listener` y una simulación visual utilizando `turtlesim`.

Primero, se utilizó una imagen de ROS Noetic para crear un contenedor aislado llamado `ros_noetic_exercise`. Este contenedor permite ejecutar nodos ROS sin afectar el sistema operativo anfitrión. Para acceder al entorno, se usó el comando `docker exec -it ros_noetic_exercise /bin/bash`.

Una vez dentro del contenedor, se inicializó el entorno con `source /opt/ros/noetic/setup.bash`.

En una consola, se ejecutó el nodo publisher (`talker`) con el comando `rosrun rospy_tutorials talker`. Este nodo publica mensajes en el tópico `/chatter` cada segundo. En otra consola, se accedió al contenedor y se ejecutó el nodo subscriber (`listener`) con `rosrun rospy_tutorials listener`, que se suscribe al mismo tópico y muestra los mensajes recibidos.

Adicionalmente, se ejecutó `rosrun turtlesim turtlesim_node` para abrir una interfaz visual con la tortuga, y en otra consola se lanzó `rosrun turtlesim turtle_teleop_key` para controlarla con el teclado. Esto permitió validar gráficamente el funcionamiento de ROS dentro del contenedor.

Este ejemplo demostró que ROS permite la comunicación entre nodos de forma desacoplada: el `talker` no necesita saber quién lo escucha, y el `listener` no necesita saber quién publica. El ROS Master gestiona el descubrimiento dinámico entre nodos.

La correcta ejecución de los nodos, el flujo de mensajes y la interacción visual confirmaron que el entorno ROS en Docker funciona de manera aislada y eficaz.

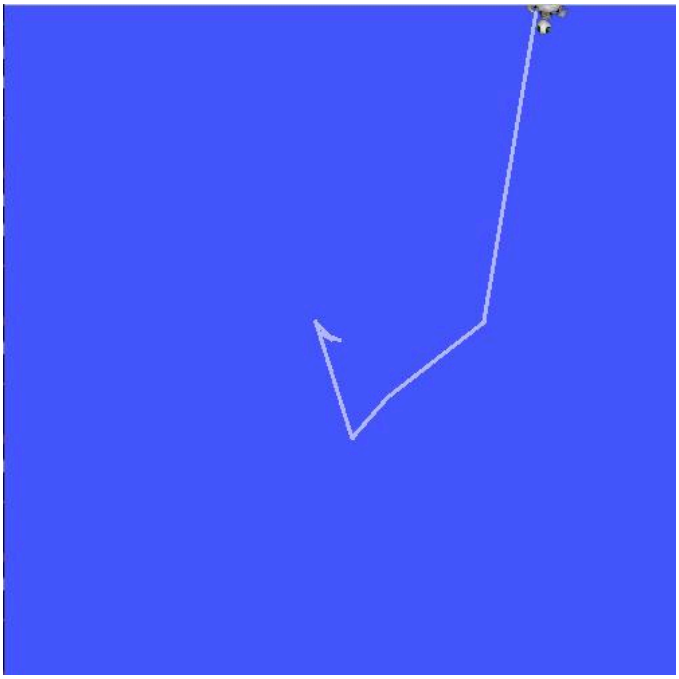

Fig 4. Uso de ROS.

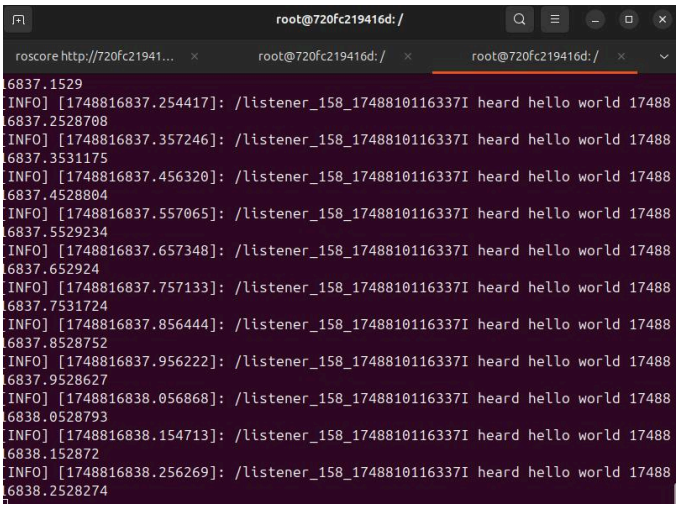

FIg 5. Visualización de procedimientos.

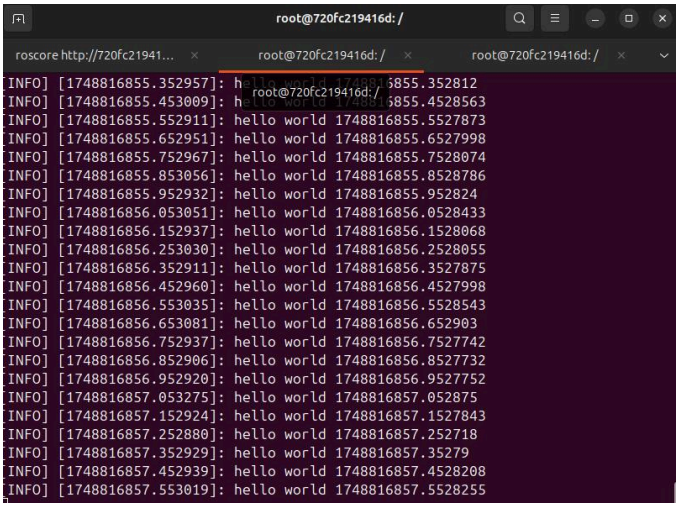

Fig 6. Visualización de procedimiento.

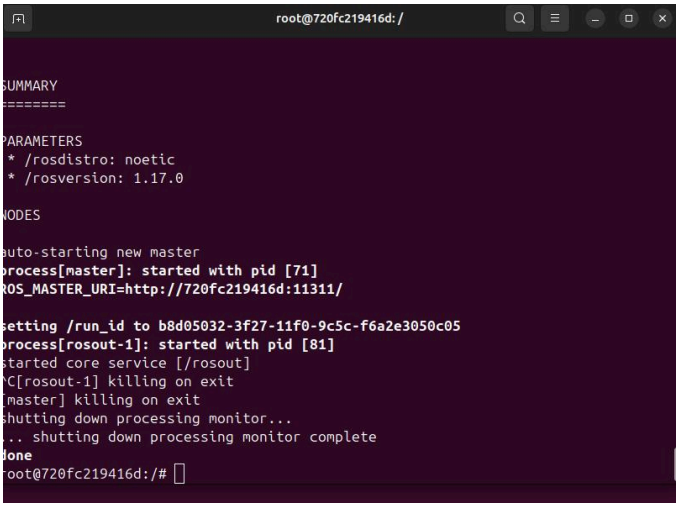

Fig 7. Visualización de procedimiento.

VI. Resultados

El desarrollo e implementación de los tres algoritmos permitió validar distintos aspectos técnicos y operativos relacionados con simulación, control e integración en entornos contenibles.

En el primer caso, el simulador del seguidor de línea logró ejecutar correctamente un algoritmo PID personalizado que permitió al vehículo seguir una trayectoria predefinida con estabilidad. Se observó cómo pequeñas variaciones en los valores de las constantes del PID afectaban directamente la suavidad del seguimiento y la respuesta a los desvíos. Visualmente, el rastro generado por el carro en la interfaz gráfica permitió evaluar el desempeño del control en tiempo real.

En el segundo caso, el videojuego "SpaceMax Defender" integró correctamente múltiples niveles, enemigos dinámicos, efectos visuales, sistema de vidas, power-ups y un jefe final. Las colisiones entre sprites, la activación del escudo y la variación en la dificultad por niveles fueron validadas durante las sesiones de prueba. El juego respondió de manera fluida a 60 cuadros por segundo, y las imágenes personalizadas fueron cargadas correctamente desde la carpeta `images`.

Finalmente, en el entorno ROS, la ejecución del ejemplo `talker-listener` dentro de un contenedor Docker evidenció que los nodos podían comunicarse correctamente a través del tópico `/chatter`, sin necesidad de que se conocieran mutuamente. La simulación con `turtlesim` confirmó la funcionalidad del sistema distribuido, y la interacción del usuario con el teclado se reflejó en tiempo real en el movimiento de la tortuga.

En conjunto, los tres experimentos demostraron que es posible construir, ejecutar y documentar sistemas interactivos utilizando herramientas modernas como Python, Docker, Tkinter, Pygame y ROS, fortaleciendo así las habilidades prácticas en control, programación y despliegue de software.

VII. Conclusiones

- La ejecución de este proyecto permitió alcanzar de manera satisfactoria cada uno de los objetivos propuestos, reforzando competencias técnicas en programación, control y despliegue de sistemas interactivos.
- Se logró **implementar el algoritmo PID** en un entorno gráfico mediante Tkinter, lo que facilitó la comprensión visual del comportamiento de un sistema en lazo cerrado. La simulación evidenció cómo el ajuste de los parámetros influye directamente en la estabilidad y precisión del seguimiento de trayectorias, cumpliendo con el objetivo de aplicar principios de control en una situación concreta.
- Se pudo **personalizar un videojuego 2D en Pygame** con niveles progresivos, sprites gráficos propios, detección de colisiones y efectos visuales. Esta implementación fortaleció las habilidades en programación orientada a objetos, gestión de eventos y diseño modular, cumpliendo con el objetivo de desarrollar entornos gráficos interactivos bajo estructuras complejas.
- La actividad de ROS permitió **ejecutar un entorno distribuido en Docker**, validando la arquitectura publisher–subscriber mediante los nodos `talker` y `listener`, así como una interacción visual por medio de `turtlesim`. Este procedimiento permitió cumplir el objetivo de explorar la comunicación entre procesos aislados bajo un sistema operativo de propósito robótico, dentro de un contenedor reproducible.
- Adicionalmente, se logró **documentar técnicamente cada desarrollo** en archivos README con explicaciones funcionales, y **publicar los proyectos** en plataformas como GitHub y Docker Hub, garantizando trazabilidad y acceso controlado. Esto cumplió con los objetivos relacionados con documentación y despliegue reproducible.
- En conjunto, la experiencia demostró la aplicabilidad real de las herramientas aprendidas en semestres anteriores, consolidando habilidades clave para enfrentar proyectos más complejos en campos como robótica, automatización, simulación y desarrollo de software distribuido.

References

[1] The ROS Wiki. "rospy_tutorials/Tutorials," [En línea]. Disponible en: https://wiki.ros.org/rospy_tutorials/Tutorials
[2] Open Robotics. "turtlesim Package," [En línea]. Disponible en: https://wiki.ros.org/turtlesim

[3] Docker Inc. “Get Started with Docker,” [En línea]. Disponible en: https://docs.docker.com/get-started/
[4] Python Software Foundation. “Tkinter — Python interface to Tcl/Tk,” [En línea]. Disponible en: https://docs.python.org/3/library/tkinter.html
[5] Pygame Community. “Pygame Documentation,” [En línea]. Disponible en: https://www.pygame.org/docs/
[6] Åström, K. J., & Hägglund, T. (1995). “PID Controllers: Theory, Design, and Tuning.” Instrument Society of America.
[7] IEEE. “Guidelines for Author Supplied Electronic Text and Graphics,” [En línea]. Disponible en: https://www.ieee.org/publications/authors/author-guide-interactive.pdf
[8] GitHub Docs. “Introduction to GitHub,” [En línea]. Disponible en: https://docs.github.com/en/get-started