Deusto
Universidad de Deusto

Hitos 1 y 2

Robótica y Automatización Inteligente

Grupo 7

Gonzalo Maldonado Arana
Vanessa Elizabeth Mejía Fajardo
Itsasne Presumido Martínez-Conde
Mishelle Quinchiguango

Gabriel Pulgar
Unai Laconcha
Jesús Sevillano
Gorka Elorriaga

Índice

Acceso a GitHub

Para mantener un seguimiento de las versiones del proyecto y unir el trabajo de ambos grupos, se ha creado un repositorio de github con el *workspace* de ROS y las imágenes de docker. En el siguiente enlace se encuentra el repositorio:

https://github.com/gonzalo002/robotica_g7_mucsi

Hito 1

Este primer hito se centra en sentar las bases del proyecto final de la asignatura, garantizando la correcta estructuración del entorno de desarrollo. Esto incluye la organización del workspace de ROS con paquetes básicos, la creación de imágenes Docker que integren todas las dependencias necesarias para trabajar con el brazo robótico UR3e, y la elaboración de una documentación concisa que describa los nodos, la comunicación entre ellos y los tipos de datos que conforman el sistema distribuido.

¿En qué consiste el Proyecto Final?

El proyecto consiste en el desarrollo de un sistema distribuido para el control de un brazo robótico colaborativo capaz de realizar tareas de detección y reconstrucción de figuras utilizando cubos de colores. Este sistema combina visión artificial, algoritmos de aprendizaje por refuerzo y control robótico, permitiendo al robot operar en modalidad automática o manual para completar la tarea asignada.

- **Funcionamiento automático:** El robot opera de manera autónoma sin intervención manual directa.
- **Funcionamiento manual:** La persona interactúa directamente sobre la posición del robot y el estado de la garra mediante gestos captados por las cámaras.

Despliegue del Sistema

El despliegue del proyecto contará con los siguientes elementos:

- **UR3:** Robot colaborativo de 6 ejes de la marca Universal Robots.
- **Cámaras:** Contaremos con un total de 4 cámaras USB, 2 para la detección de la figura y los cubos y otras 2 para la detección de la posición de la mano..
- **Cubos de colores:** Los utilizaremos para realizar la figura con un límite máximo de 25 cubos por figura.

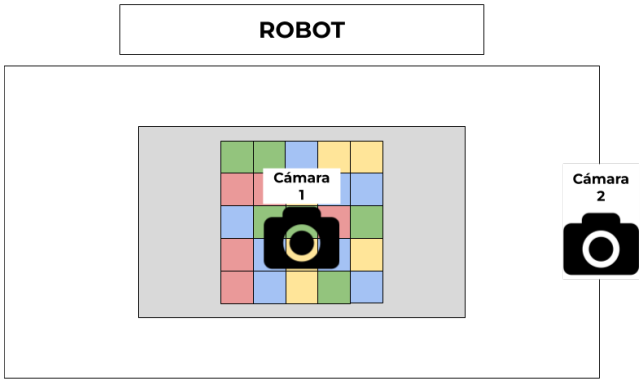

Imagen 1. Representación del primer espacio de trabajo del robot junto con las cámaras y cubos.

Imagen 2. Representación del segundo espacio de trabajo con las cámaras para la detección de la posición de la mano.

Imagen de Docker Personalizada

Para el desarrollo del proyecto se ha creado una imagen de docker personalizada con las librerías necesarias para la realización del proyecto. Esta imagen de docker se ha desarrollado utilizando como base la imagen aportada por el profesor en clase.

Las principales modificaciones realizadas sobre la imagen son la inclusión de las siguientes librerías de Python y paquetes de ROS Noetic.

- **Paquetes de ROS**
  - **ros-noetic-camera-calibration**
- **Librerías de Python**
  - **gymnasium==0.29.1**
  - **stable-baselines3[extra]==2.2.1**
  - **numpy --upgrade**
  - **scipy**
  - **opencv-contrib-python**
  - **mediapipe**
  - **tk**

Estructura del Sistema

Para alcanzar nuestro objetivo implementaremos el sistema distribuido aprovechando las capacidades de ROS, que proporciona una arquitectura modular basada en nodos.

- **Captura y Procesamiento de Imágenes**
  - **Nodo Cv2.1:** Recibe y procesa las imágenes captadas por las cámaras para controlar el robot mediante gestos manuales. La cámara frontal identifica la posición de la mano e interpola su ubicación en la imagen para determinar las coordenadas del robot en los ejes X e Y, además de detectar un gesto específico que indica el inicio y el final de la captura de la pose. Simultáneamente, la cámara superior localiza la mano y se encarga de definir exclusivamente la posición del robot en el eje Z, además de reconocer si la mano está abierta o cerrada, lo que permite controlar la apertura o cierre de la garra. Toda la información obtenida de ambas cámaras se guarda en un mensaje personalizado de ROS y se transmite mediante un topic al nodo encargado del control del robot.
  - **Nodo Cv2.2:** Recibe y procesa las imágenes de las cámaras para la detección de cubos y figuras. En este caso, a través de la cámara superior y frontal, se analiza la figura creada, detectando la posición y el color de los cubos visibles, almacenando esta información en matrices de 5x5. Posteriormente la figura es desarmada, esparciendo los cubos a través del espacio de trabajo del robot, detectando su posición, color y rotación a través de la cámara superior, almacenandose en un

mensaje personalizado de ROS. Una vez conocida la localización actual de los cubos, se envían los datos a través de un topic al nodo de *Reinforcement Learning*.

- **Toma de Decisiones con Aprendizaje por Refuerzo**
  - **Nodo RL:** Implementa un entorno y un agente de aprendizaje por refuerzo el cual tiene como objetivo maximizar la recompensa, en este caso negativa, la cual está asociada al tiempo necesario para la recogida de todos los cubos necesarios para la construcción de la figura descrita. Este nodo recibe a través de un topic los datos de la localización, rotación y color de los cubos, y devuelve tanto el orden de los cubos a coger como las trayectorias calculadas en el proceso.
- **Control y Ejecución del robot**
  - **Nodo Control_robot:** Recibe la información adquirida por los nodos previamente mencionados, y en base a esta ejecuta las acciones requeridas para que el robot cumpla con la tarea asignada.

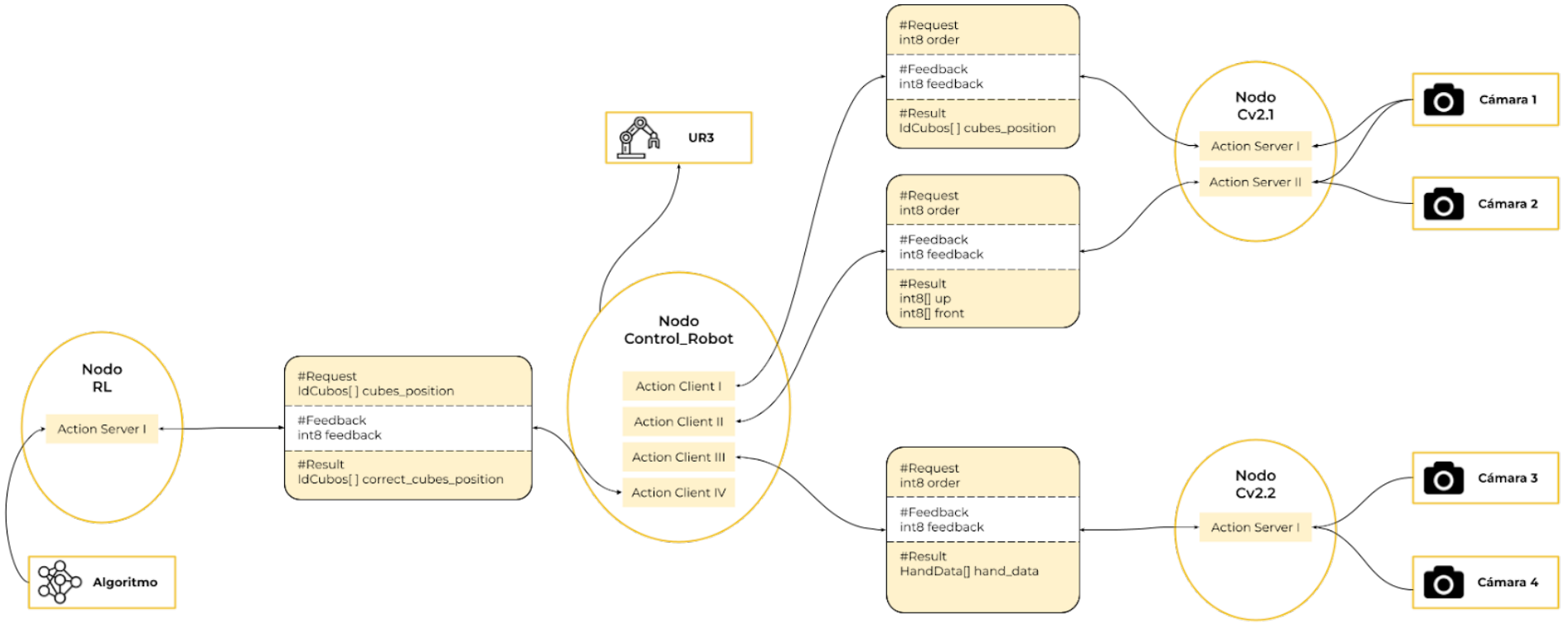

Imagen 3. Diagrama de la arquitectura de nodos de ROS del proyecto

Hito 2

En este segundo hito, el objetivo es desarrollar un nodo de ROS que permita capturar y procesar las variables clave de las trayectorias ejecutadas por el brazo robótico, como velocidad, configuración y esfuerzo. Estos datos serán registrados en tiempo real en una instancia de InfluxDB proporcionada por Gestamp, permitiendo su análisis posterior y la visualización de métricas mediante herramientas de monitoreo como Grafana.

Estructura del Sistema

El sistema para el registro de trayectorias se basa en dos nodos principales:

- **Nodo `control_robot`**
  - Genera un topic interno llamado `/control_movimiento`, que mediante datos tipo Int8 actúa como un disparador *(trigger)* para el registro de datos. Este topic pública un valor "1" cuando el robot está realizando una trayectoria y "0" cuando está detenido.
- **Nodo `influx_node`:**
  - Suscrito al topic `/joint_states`, desde donde recibe información del estado de los ejes del robot a una frecuencia de 500 Hz.
  - Suscrito al topic `/control_movimiento` donde si reside un "1" almacena los datos recibidos por el topic `/joint_states` y los almacena en la base de datos InfluxDB hasta que reciba un "0". El registro de datos de InfluxDB se realiza a una velocidad de 10 Hz.

Viendo la estructura del sistema se puede ver cómo nuestra captura de datos es síncrona, ya que dependemos de una señal externa para iniciar y detener el registro de la trayectoria. Esto implica que el nodo no está monitoreando continuamente el estado del robot, sino que actúa solo cuando recibe mensajes específicos a través del topic de `control_movimiento`.

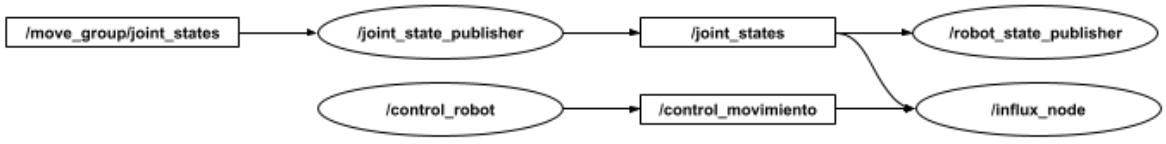

Imagen 4. Imagen obtenida a través de rqt_graph que representa gráficamente la estructura del sistema

Programa

Para la ejecución de los nodos y el registro de los datos se han desarrollado dos códigos que utilizan dos clases para gestionar todas las tareas:

- **`influx.py`: Gestión de Datos del Robot con InfluxDB**

  El primer código implementa un nodo ROS que interactúa con InfluxDb para registrar y consultar datos del estado de un robot, como posiciones, velocidades y esfuerzos de sus articulaciones a través de la clase `InfluxLoader()`

  - **Inicialización y Conexión:** Se inicializa el nodo ROS y se configura una conexión con una base de datos InfluxDB usando parámetros como el token, URL, organización y bucket.
  - **Suscripción a los topics:** El nodo se suscribe a los dos topics: `control_movimiento` y `joint_states`.
  - **Almacenamiento de Datos:** Cuando se activa el *flag*, los datos de las articulaciones se almacenan en InfuxDB. Los datos se escriben con marcas temporales precisas utilizando Point de la librería de cliente InfluxDB.

- **`control_robot.py`: Control del Robot**

  Este segundo código define una clase `ControlRobot()` que maneja operaciones de un robot manipulador en un entorno MoveIt, incluyendo el movimiento de articulaciones y la interacción con obstáculos.

  - **Inicialización y Configuración:** Se inicializan objetos MoveIt como RobotCommander, MoveGroupCommander y PlanningSceneInterface para gestionar el robot y su escena. Además, se define un publicador para enviar la señal de movimiento `control_movimiento`.
  - **Funciones para Control del Robot:** Esta clase consiste en la clase creada en clase con la única modificación de que al planificar y ejecutar trayectorias cartesianas se publica en el topic `/control_movimiento`.

  En el `main`, se ejecuta un programa interactivo que utilizando la clase anteriormente descrita permite al usuario ejecutar trayectorias concretas sobre el robot. El usuario selecciona una trayectoria predefinida (semicircular, rectangular o línea recta). El programa carga los puntos de la trayectoria desde archivos YAML y los ejecuta en el robot, primero moviéndolo al estado articular inicial, luego siguiendo la trayectoria cartesiana, y finalizando en el estado articular final.

Comprobación de la Ejecución

Para evaluar el sistema, se diseñan y ejecutan seis trayectorias diferentes:

- **Trayectorias sin colisión:**
  - **Curva o semicircular:** Permite observar el comportamiento de los ejes en un movimiento continuo y fluido.
  - **Rectangular:** Consiste en un desplazamiento por segmentos rectos, con cambios de dirección bien definidos.
  - **Línea recta:** Representa un movimiento simple, ideal para evaluar la estabilidad de los datos capturados.
- **Trayectorias con colisión:**

  Cada una de las trayectorias anteriores se repite, pero se introduce una colisión durante su ejecución. Esto permite analizar cómo varían las variables de esfuerzo y velocidad en situaciones de fallo mecánico o interrupciones en el movimiento.

Registro de Datos

Los datos se almacenan en InfluxDB siguiendo el siguiente formato:

- **Trayectory_<ID_unica_definida_por_timestamp>**
  - **effort**
    - joint name
  - **position**
    - joint_name
  - **velocity**
    - joint_name

El resultado final visto en InfluxDB es el siguiente:

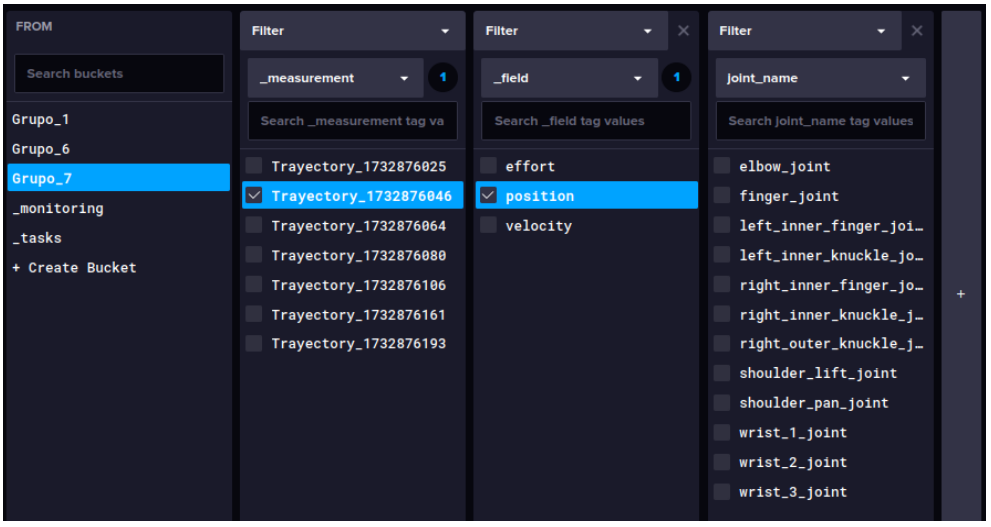

Imagen 5. Imagen donde se muestra la estructura de datos en InfluxDB

En las siguientes gráficas se pueden observar los resultados de las trayectorias realizadas:

- **Trayectoria 1: Semicírculo sin Colisión**

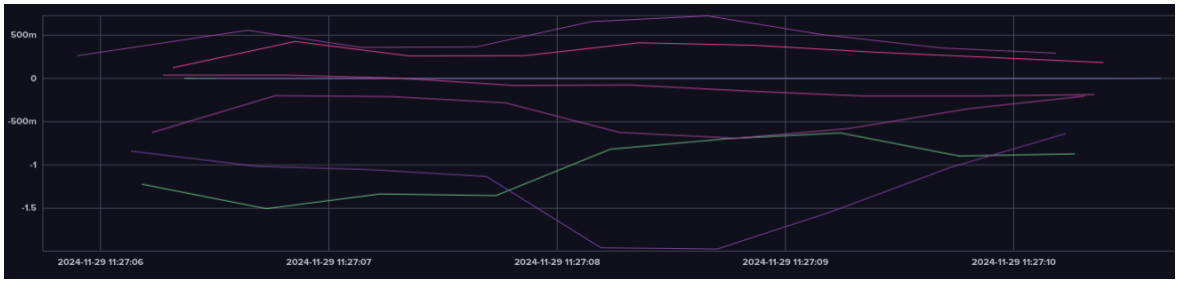

Imagen 6. Gráfico de esfuerzo de los ejes a lo largo la Trayectoria 1: Semicírculo sin Colisión

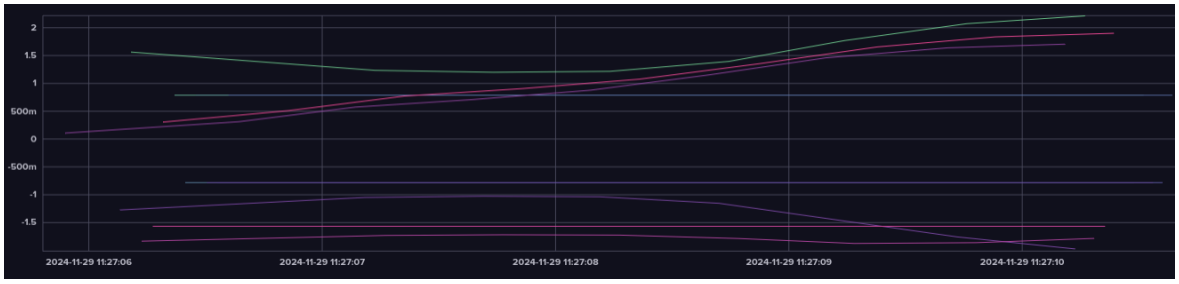

Imagen 7. Gráfico de posición de los ejes a lo largo la Trayectoria 1: Semicírculo sin Colisión

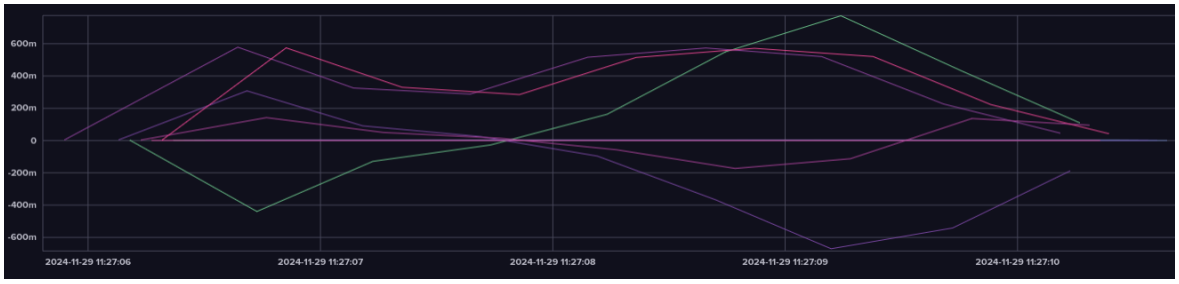

Imagen 8. Gráfico de velocidad de los ejes a lo largo la Trayectoria 1: Semicírculo sin Colisión

- **Trayectoria 2: Rectángulo sin Colisión**

Imagen 9. Gráfico de esfuerzo de los ejes a lo largo la Trayectoria 2: Rectángulo sin Colisión

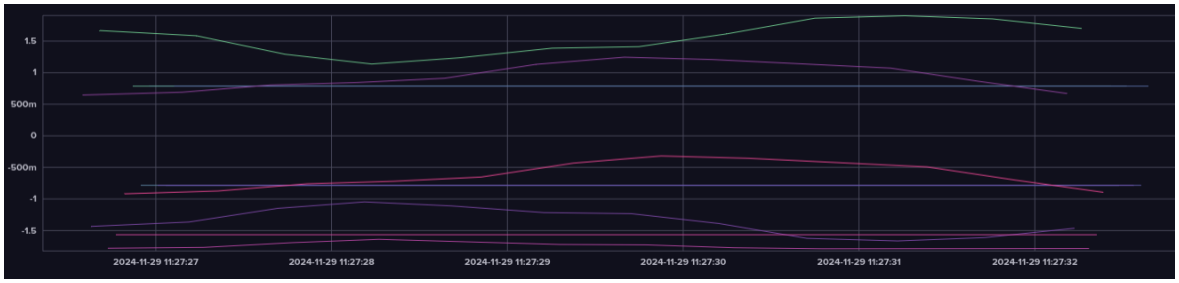

Imagen 10. Gráfico de posición de los ejes a lo largo la Trayectoria 2: Rectángulo sin Colisión

Imagen 11. Gráfico de velocidad de los ejes a lo largo la Trayectoria 2: Rectángulo sin Colisión

- **Trayectoria 3: Línea Recta sin Colisión**

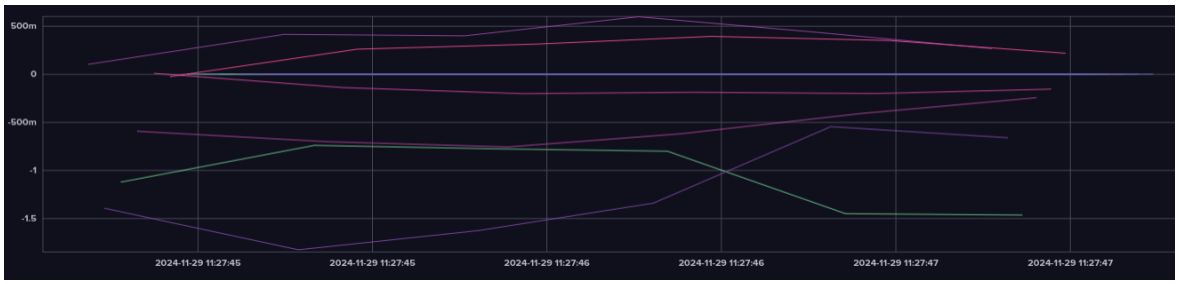

Imagen 12. Gráfico de esfuerzo de los ejes a lo largo la Trayectoria 3: Línea Recta sin Colisión

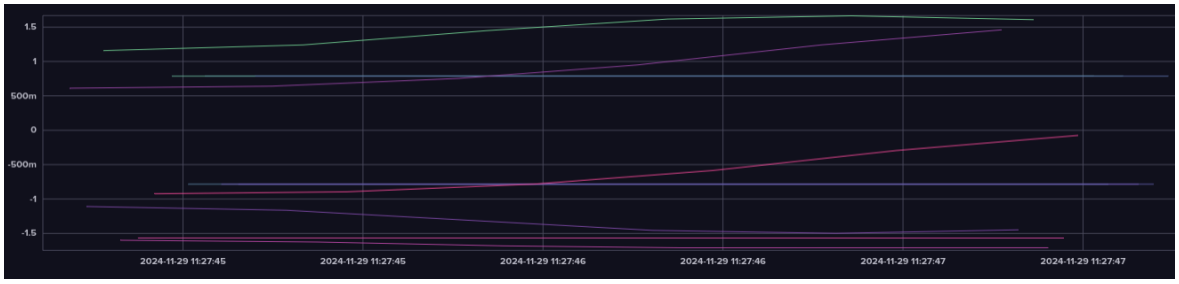

Imagen 13. Gráfico de posición de los ejes a lo largo la Trayectoria 3: Línea Recta sin Colisión

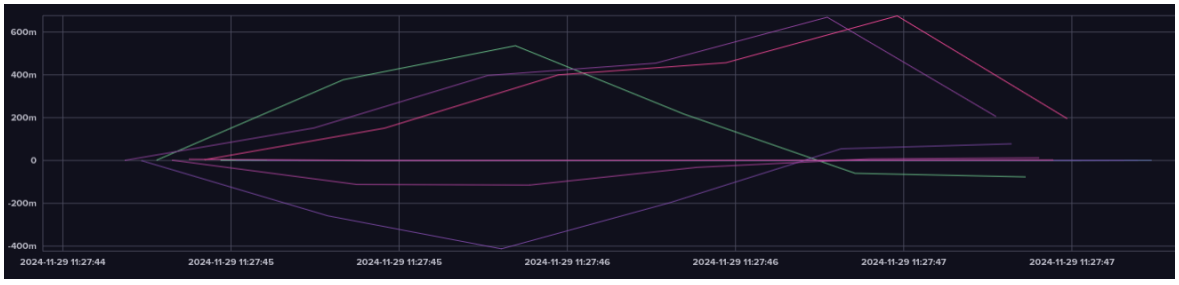

Imagen 14. Gráfico de velocidad de los ejes a lo largo la Trayectoria 3: Línea Recta sin Colisión

- ## Trayectoria 4: Semicírculo con Colisión

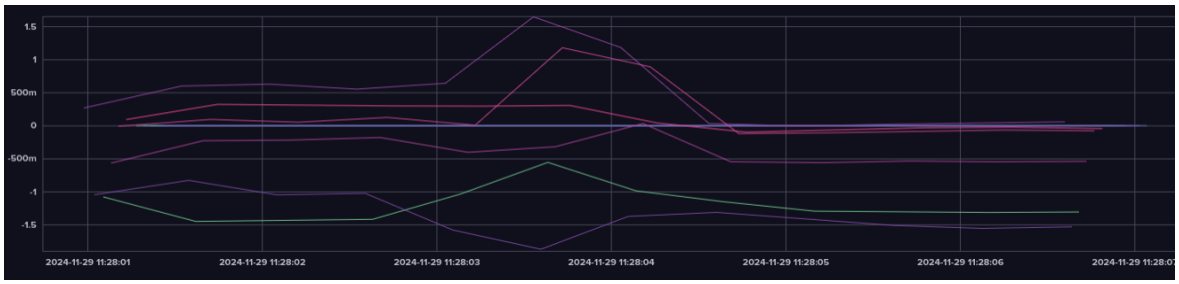

Imagen 15. Gráfico de esfuerzo de los ejes a lo largo la Trayectoria 4: Semicírculo con Colisión

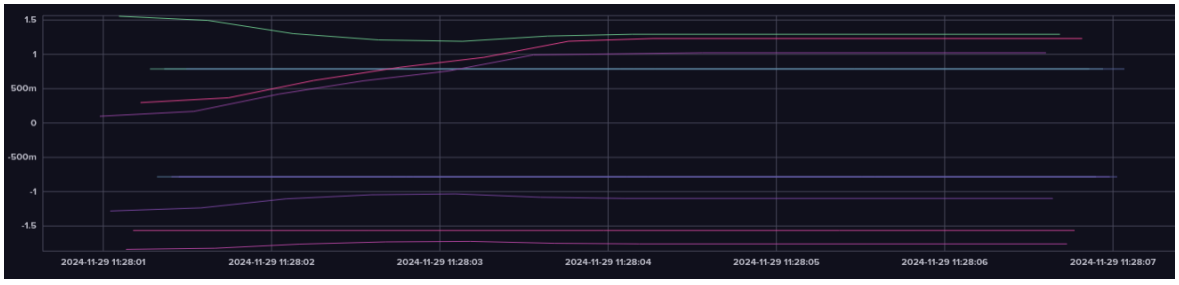

Imagen 16. Gráfico de posición de los ejes a lo largo la Trayectoria 4: Semicírculo con Colisión

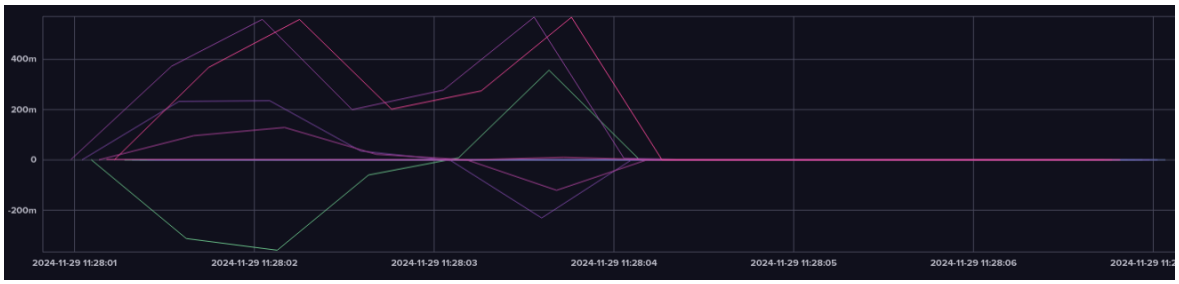

Imagen 17. Gráfico de velocidad de los ejes a lo largo la Trayectoria 4: Semicírculo con Colisión

- ## Trayectoria 5: Rectángulo con Colisión

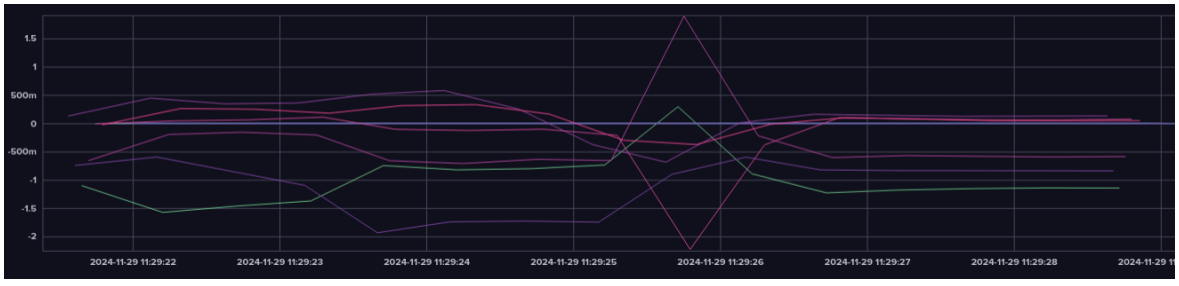

Imagen 18. Gráfico de esfuerzo de los ejes a lo largo la Trayectoria 5: Rectángulo con Colisión

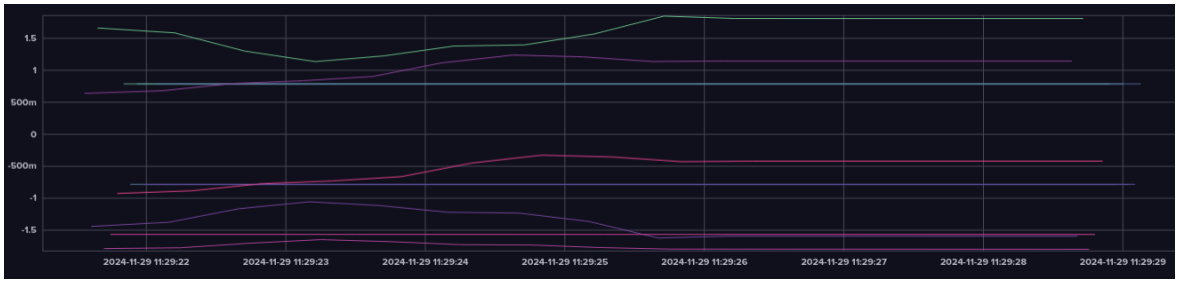

Imagen 19. Gráfico de posición de los ejes a lo largo la Trayectoria 5: Rectángulo con Colisión

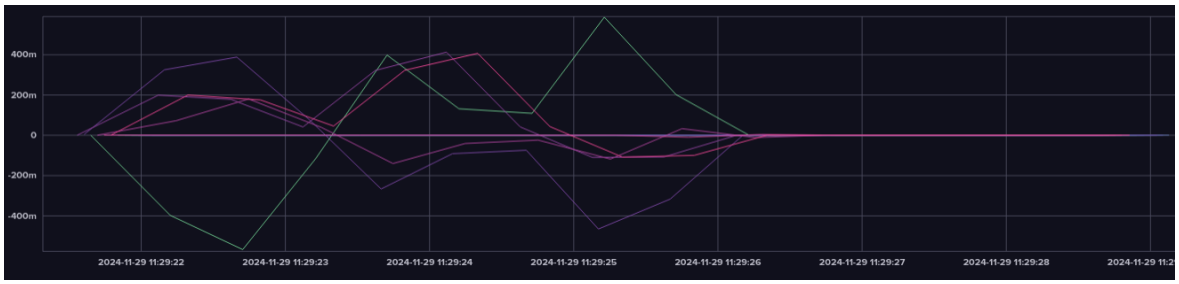

Imagen 20. Gráfico de velocidad de los ejes a lo largo la Trayectoria 5: Rectángulo con Colisión

- **Trayectoria 6: Línea Recta con Colisión**

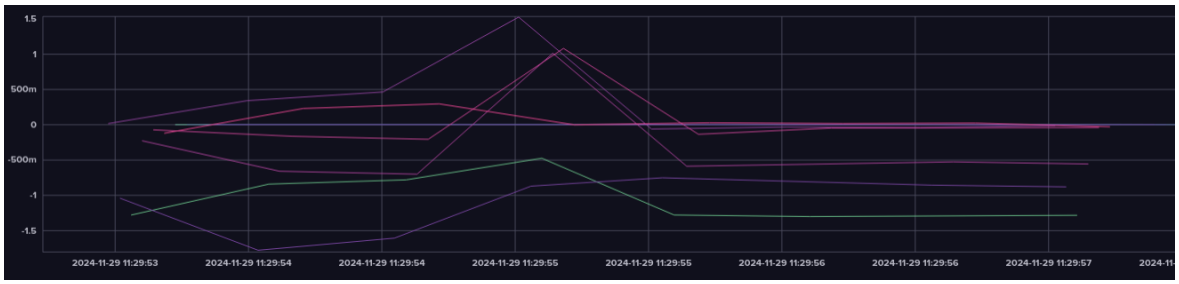

Imagen 21. Gráfico de esfuerzo de los ejes a lo largo la Trayectoria 6: Línea Recta con Colisión

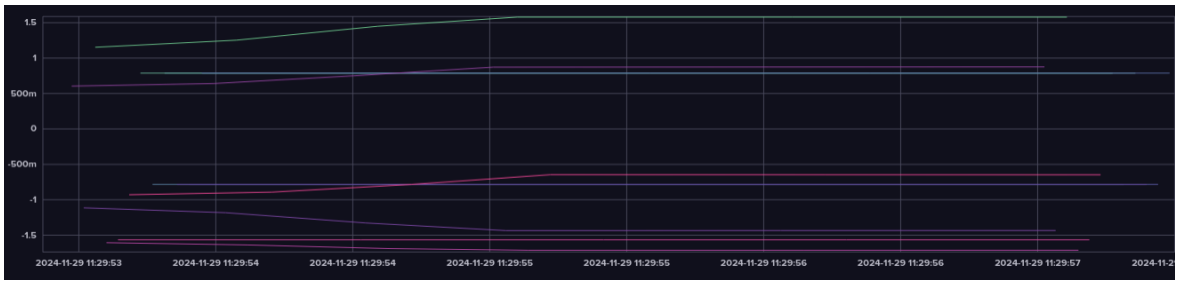

Imagen 22. Gráfico de posición de los ejes a lo largo la Trayectoria 6: Línea Recta con Colisión

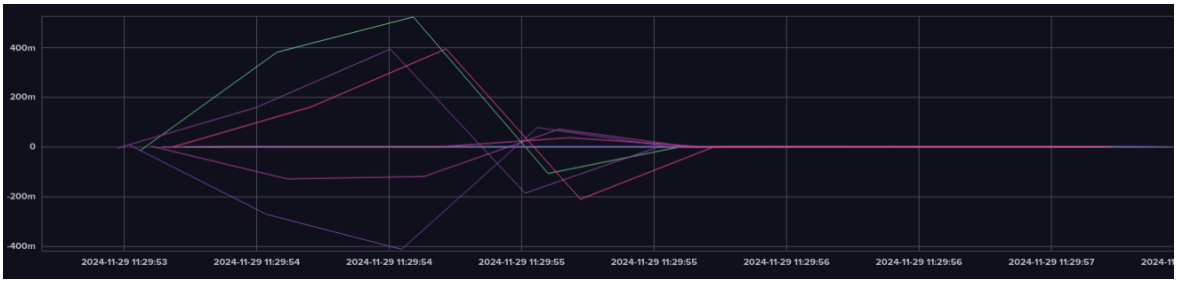

Imagen 23. Gráfico de velocidad de los ejes a lo largo la Trayectoria 6: Línea Recta con Colisión