Sistema de Seguimiento de Tortugas con ROS2

Trabajo 2 - EOII

Francisco Nortes Novikov
Vicente Burdeus Sánchez

16 de enero de 2026

1. Descripción de la Implementación

El sistema desarrollado consiste en un seguidor de tortugas en ROS2 que permite que una tortuga exploradora siga automáticamente a otra tortuga controlada manualmente. Se implementó una arquitectura modular con los siguientes componentes:

Paquetes creados:

- **follower**: Paquete principal con la lógica de seguimiento
- **follower_interfaces**: Definiciones de servicios y actions personalizados

Nodos principales:

- **pose_savers**: Suscriptor a `/turtle1/pose` y `/explorer/pose`
- **explorer_velocity**: Controlador proporcional que publica velocidades en `/explorer/cmd_vel`
- **turtle_info_service**: Servidor del servicio de información
- **catch_info_action**: Action server para el proceso de captura

Ecuaciones del controlador: $d = \sqrt{(x_t - x_e)^2 + (y_t - y_e)^2}$, $\alpha = \text{atan2}(y_t - y_e, x_t - x_e)$, $\theta_e = \text{atan2}(\sin(\alpha - \theta_{actual}), \cos(\alpha - \theta_{actual}))$, $v = 1{,}0 \times d \times \cos(\theta_e)$, $\omega = 4{,}0 \times \theta_e$.

2. Interfaces Implementados

2.1. Servicio TurtleInfo.srv

Proporciona información completa sobre ambas tortugas:

Request: Vacío

Response: `float64` para posiciones (`turtle_x/y`, `explorer_x/y`), orientaciones (`turtle_theta`, `explorer_theta`), velocidades (`turtle/explorer_linear/angular_velocity`) y `distance`.

2.2. Action CatchTurtle.action

Permite monitorizar el proceso de captura con feedback continuo:

Goal: Vacío

Result: `bool caught` (indica si la tortuga fue capturada)

Feedback: Los mismos campos que la respuesta del servicio TurtleInfo

3. Problemas Encontrados y Soluciones

3.1. Bloqueo del Action Server

Problema: Al implementar inicialmente el action server en el nodo principal, este bloqueaba la ejecución del resto de callbacks, impidiendo que el controlador funcionara correctamente.

Solución: Tras investigar la documentación de ROS2 sobre executors y callback groups, se optó por una arquitectura modular completamente separada. Se creó un nodo independiente para cada funcionalidad principal, utilizando `SharedPoses` con `threading.Lock` para compartir datos de forma thread-safe entre nodos.

3.2. Sincronización de Datos

Problema: Acceso concurrente a las poses desde múltiples nodos.

Solución: Implementación de la clase `SharedPoses` con locks y `deepcopy` para garantizar acceso thread-safe a los datos compartidos.

4. Resultados de Pruebas

Se realizaron las siguientes pruebas con resultados exitosos:

1. **Seguimiento básico:** La tortuga exploradora sigue correctamente a turtle1 controlada por teleop
2. **Servicio de información:** El cliente recibe y muestra correctamente todos los datos de posición y velocidad
3. **Action server:** Proporciona feedback continuo durante el proceso de captura hasta alcanzar el objetivo

El sistema funciona de manera fluida y precisa. Las siguientes capturas muestran los clientes funcionando:

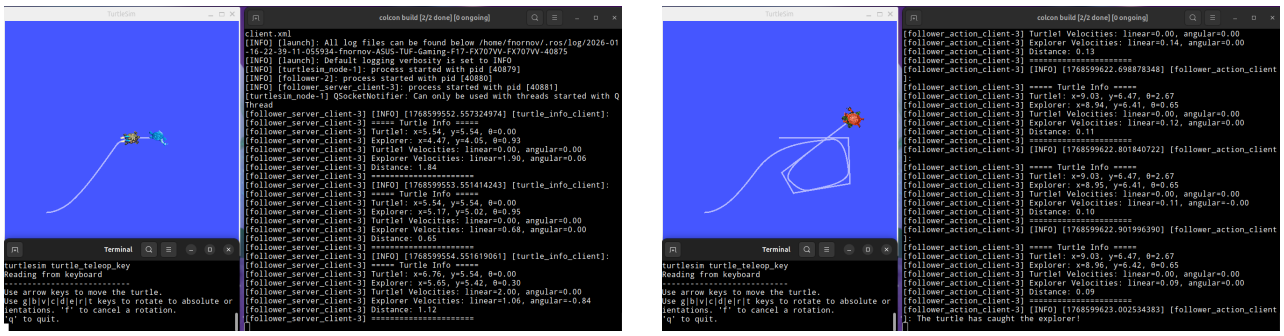

Figura 1: Cliente del servicio de información (izq.) y del action server (der.)

5. Mapa de Nodos, Topics y Servicios

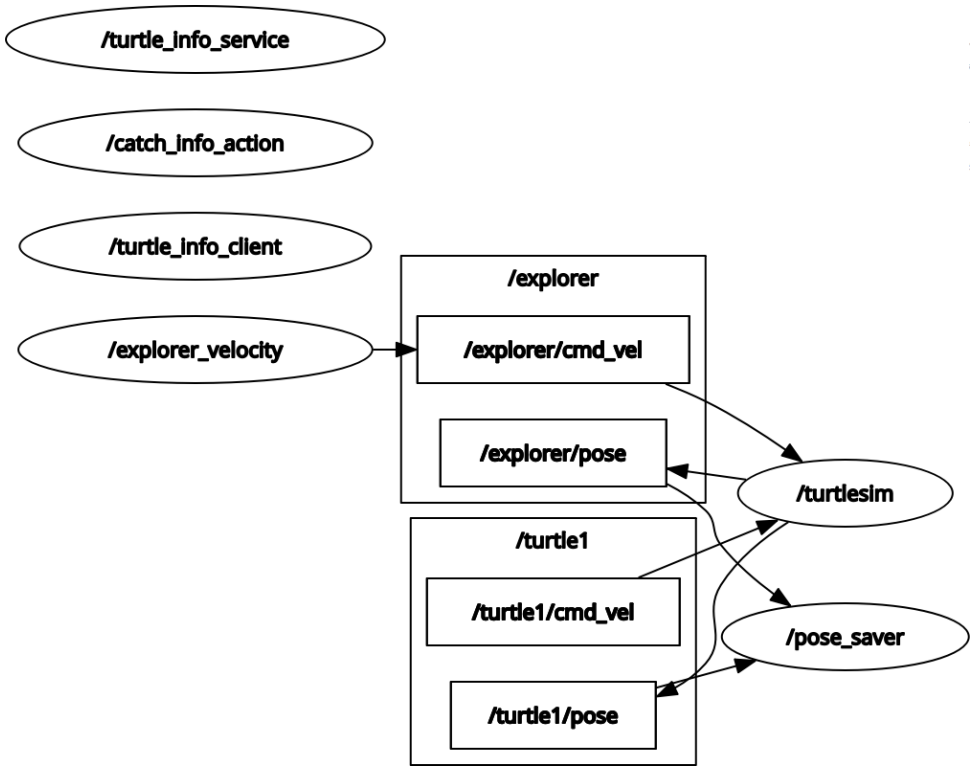

Figura 2: Grafo de nodos del sistema (rqt_graph)