ENTORNOS OPERATIVOS PARA ROBÓTICA E INFORMÁTICA INDUSTRIAL (EOII)

TRABAJO 2 - ROS2
SISTEMA DE SEGUIMIENTO DE TORTUGAS EN TURTLESIM

Autores: Javier Veyrat Zaragozá y Julio Quesada Font

ÍNDICE

1. DESCRIPCIÓN DE LA IMPLEMENTACIÓN

1.1 E1 - GENERACIÓN DE TORTUGA PERSEGUIDORA (turtle_spawner.py)

Se ha implementado un nodo que genera automáticamente la tortuga "explorer" utilizando el servicio /spawn de TurtleSim.

El nodo:

- Lee parámetros: explorer_x, explorer_y, explorer_theta
- Valida que la posición esté dentro de los límites
- Crea cliente del servicio /spawn
- Envía request asíncrono para crear la tortuga
- Confirma creación exitosa mediante callback

1.2 E2 - SISTEMA DE SEGUIMIENTO (turtle_tracker.py)

Implementa un algoritmo de control proporcional para que la tortuga "explorer" persiga a "turtle1".

El nodo:

- Se suscribe a /turtle1/pose y /explorer/pose
- Publica comandos de velocidad en /explorer/cmd_vel
- Ejecuta bucle de control a 50 Hz

1.3 E3 - SERVICIO DE INFORMACIÓN (turtle_info_service.py)

Implementa el servidor del servicio personalizado "turtle_info".

Información proporcionada:

- Posición (x, y) de turtle1 y explorer
- Orientación (theta) de ambas tortugas
- Velocidades lineales y angulares actuales
- Distancia euclidiana entre ambas tortugas
- Timestamp de la consulta

1.4 E4 - CLIENTE DE CONSULTA (turtle_info_client.py)

Nodo cliente que invoca el servicio turtle_info periódicamente.

El nodo:

- Consulta el servicio cada segundo (configurable vía parámetro query_rate)
- Muestra información estructurada y formateada en consola
- Maneja errores si el servicio no está disponible
- Verifica disponibilidad del servicio al iniciar

1.5 E5 - FICHERO LAUNCH (launch.xml)

Fichero de lanzamiento XML que pone en marcha todo el sistema.

El fichero de lanzamiento:

- Lanza los nodos
- Permite el uso de parámetros personalizados

1.6 E6 - ACTION SERVER (turtle_info_action_server.py)

Convierte el servicio de información en un Action Server.

Funcionamiento:

- Recibe goal con update_rate y catch_distance
- Envía feedback periódicamente con información de las tortugas
- Detecta cuando explorer alcanza a turtle1 (distancia < catch_distance)
- Espera 3 ciclos con velocidad ≈ 0 para confirmar
- Envía resultado final con estadísticas

2. RESULTADOS DE PRUEBAS

2.1 PRUEBA DE LANZAMIENTO DEL SISTEMA

Comando: ros2 launch turtle_tracker launch.xml

Resultado: EXITOSO
- TurtleSim se inicia correctamente
- Tortuga explorer aparece en posición (2.0, 2.0)
- El tracker comienza a funcionar inmediatamente
- El servicio turtle_info está disponible
- El cliente muestra información cada segundo

2.2 PRUEBA DE SEGUIMIENTO

Comando: ros2 run turtlesim turtle_teleop_key

Resultado: EXITOSO
- Al mover turtle1 con las teclas, explorer la sigue
- El seguimiento es suave gracias al control proporcional
- Explorer se detiene cuando alcanza a turtle1
- La velocidad es proporcional a la distancia

2.3 PRUEBA DE PARÁMETROS PERSONALIZADOS

Comando: ros2 launch turtle_tracker launch.xml explorer_x:=8.0 explorer_y:=8.0

Resultado: EXITOSO
- Explorer aparece en la posición especificada (8.0, 8.0)
- El sistema valida que está dentro de límites

2.4 PRUEBA DE VALIDACIÓN DE LÍMITES

Comando: ros2 launch turtle_tracker launch.xml explorer_x:=15.0 explorer_y:=15.0

Resultado: EXITOSO
- El sistema detecta que está fuera de límites
- Muestra warning y ajusta automáticamente a (10.5, 10.5)

2.5 PRUEBA DEL ACTION SERVER

Terminal 1: ros2 launch turtle_tracker launch_with_action.xml
Terminal 2: ros2 run turtlesim turtle_teleop_key
Terminal 3: ros2 run turtle_tracker turtle_info_action_client.py

Resultado: EXITOSO
- El Action Client envía goal correctamente
- Recibe feedback periódico con información de tortugas
- Cuando explorer alcanza a turtle1, el action completa

3. PROBLEMAS ENCONTRADOS Y SOLUCIONES

PROBLEMA 1: Normalización de ángulos

Descripción: El error angular podía tener valores fuera de [-π, π] causando comportamiento errático en el seguimiento.

Solución: Se implementó función de normalización:

```
def normalize_angle(self, angle):
    while angle > math.pi:
        angle -= 2.0 * math.pi
    while angle < -math.pi:
        angle += 2.0 * math.pi
    return angle
```

PROBLEMA 2: Oscilaciones al acercarse al objetivo

Descripción: Explorer oscilaba al estar muy cerca de turtle1.

Solución:
- Se añadió stop_distance para detenerse a cierta distancia
- Se reduce velocidad lineal cuando hay mucho error angular usando cos(e_θ)

4. DESCRIPCIÓN DE INTERFACES IMPLEMENTADOS

4.1 SERVICIO: TurtleInfo.srv

REQUEST
(vacío - no requiere parámetros)

RESPONSE
Contiene campos float64 para las coordenadas y velocidades de ambas tortugas, además de la distance y un timestamp

4.2 ACTION: TurtleInfoAction.action

GOAL
update_rate (Hz) y catch_distance (umbral de parada).

RESULT
success (bool), final_distance, total_time y mensaje de estado.

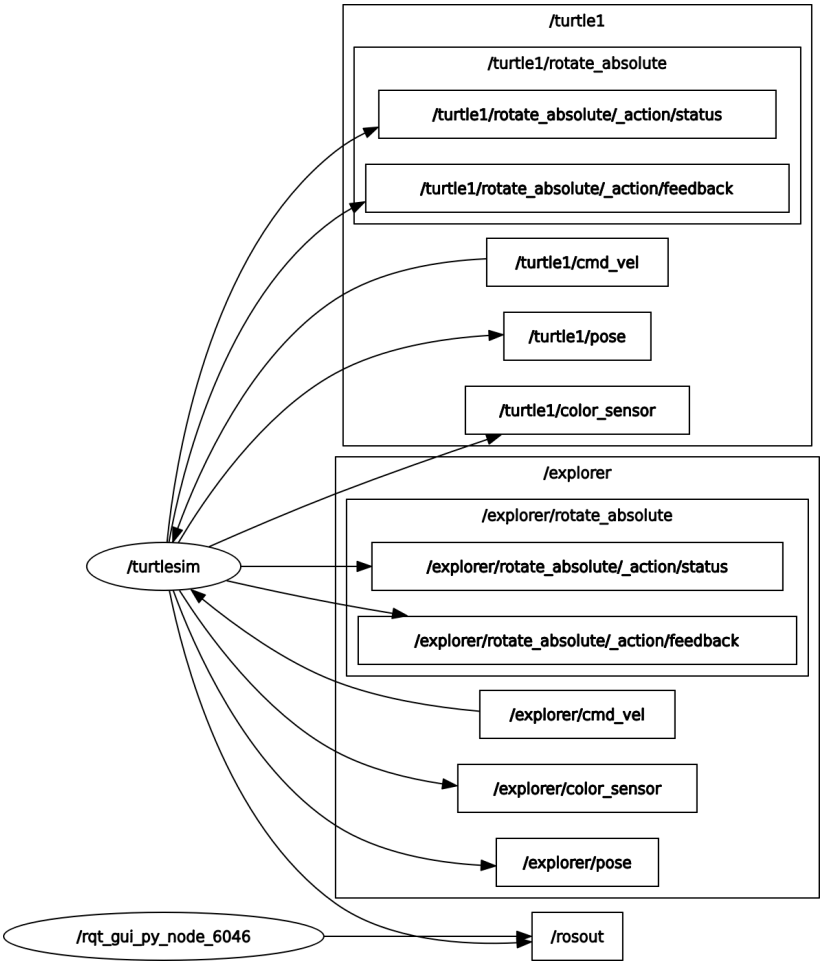
FEEDBACK
Replica la estructura de respuesta del servicio TurtleInfo para monitorización en tiempo real.

4.3 MENSAJES UTILIZADOS (de paquetes estándar)

Se utilizan:
turtlesim/msg/Pose para la lectura de sensores
geometry_msgs/msg/Twist para el envío de comandos de velocidad

5. MAPA DE NODOS, TOPICS Y SERVICIOS

El siguiente diagrama representa la arquitectura del sistema:



TOPICS:
  /turtle1/pose   [turtlesim/msg/Pose]   - Pose de turtle1
  /explorer/pose   [turtlesim/msg/Pose]   - Pose de explorer
  /turtle1/cmd_vel   [geometry_msgs/msg/Twist]- Velocidad turtle1 (teleop)
  /explorer/cmd_vel   [geometry_msgs/msg/Twist]- Velocidad explorer (tracker)

SERVICIOS:
  /spawn   [turtlesim/srv/Spawn]   - Crear tortugas
  turtle_info   [turtle_tracker/srv/TurtleInfo] - Info de tortugas

ACTIONS (con launch_with_action.xml):
  turtle_info_action   [turtle_tracker/action/TurtleInfoAction]