Robots Exploradores - Práctica 3

Versión 1.0

Abel Belhaki Rivas

24 de noviembre de 2024

Contents

Bienvenido a la documentación del proyecto RobotsExploradores. Este proyecto está diseñado para trabajar con robótica móvil utilizando ROS y se divide en dos paquetes principales: **main** y **exploracion**.

Esta documentación detalla los componentes, configuraciones y pasos necesarios para poner en marcha este sistema robótico.

CHAPTER 1

Introducción

El objetivo del proyecto es coordinar diversas tareas en un robot móvil. Estas incluyen detección de objetos, aproximación a puntos específicos y navegación autónoma. A continuación, se presentan los principales paquetes y componentes del sistema:

Paquete Principal (main): - Implementa la máquina de estados para gestionar el flujo de trabajo del robot. - Proporciona una interfaz gráfica para interactuar con el robot.

Paquete de Exploración (exploracion): - Define los servidores de acción para tareas clave como detección de objetos y control autónomo.

Nota: Asegúrate de haber instalado todos los paquetes y dependencias necesarios antes de continuar.

1.1 Introducción

Este proyecto es parte de la asignatura de robots móviles en la Universidad de Alicante y tiene como objetivo explorar conceptos clave de la programación de robots móviles mediante el uso de ROS.

Características principales:

- Coordinación mediante máquinas de estados (SMACH).
- Procesamiento de imágenes RGBD para detección de objetos.
- Navegación autónoma utilizando LIDAR.
- Interfaz gráfica para interactuar con el robot.

1.2 Configuración del Entorno

1. **Crear el Workspace de ROS:** Si no tienes un workspace configurado, sigue estos pasos:

```
mkdir -p rob_mov_ws/src
cd rob_mov_ws/src
git clone https://github.com/ottocol/navigation_stage.git
cd ..
catkin_make
source devel/setup.bash
```

2. **Dependencias:** Asegúrate de instalar las dependencias necesarias para SMACH y otros paquetes ROS:

```
sudo apt-get install ros-noetic-smach ros-noetic-smach-ros ros-noetic-executive-
↪smach ros-noetic-smach-viewer
```

3. **Lanzar el Simulador:** Para iniciar el simulador con el stack de navegación:

```
roslaunch navigation_stage mi_navigation.launch
```

1.3 Uso del Sistema

Este sistema implementa una máquina de estados finitos (SMACH) para gestionar diferentes comportamientos del robot. Además, proporciona una interfaz gráfica basada en Tkinter para facilitar la interacción con el robot.

1.3.1 Pasos para Lanzar el Sistema

Para lanzar el sistema completo, ejecuta el archivo *main.launch* con el siguiente comando:

```
roslaunch squad_main main.launch
```

Este archivo lanzará todos los nodos necesarios, incluidos los siguientes:

- **Máquina de Estados (TurtleBot State Manager):** Nodo principal que gestiona los estados del robot y publica el estado actual en el tópico */current_state*.
- **Servidores de Acción:** - *object_detection* (detección de objetos). - *approach_object* (aproximación a objetos). - *autonomous_control* (navegación autónoma).
- **Interfaz Gráfica:** Proporciona una ventana para el control manual del robot y la visualización de imágenes.

1.3.2 Flujo del Sistema

1. **Estado Inicial (Reposo):** El robot comienza en reposo esperando instrucciones.
2. **Exploración:** Activa el servidor de detección de objetos y supervisa el entorno en busca de objetos.
3. **Aproximación:** Si se detecta un objeto durante la exploración, el robot se aproxima a él utilizando el servidor de acción correspondiente. Este se activa automáticametne desde el estado de exploración si lo hemos activado en la configuración.
4. **Navegación:** El robot se desplaza a una posición predefinida (por ejemplo, «HOME»). Este comando se activa automaticamente al acabar el estado de exploración si lo hemos activado en la configuración.

1.3.3 Detalles de la Interfaz Gráfica

La interfaz gráfica permite controlar y monitorizar el estado del robot de manera interactiva. Las funcionalidades principales incluyen:

- **Botones de Estado:** Cambia entre los diferentes estados del robot (reposo, exploración, navegación, aproximación).
- **Visualización de la Cámara:** Muestra en tiempo real las imágenes captadas por la cámara RGB.
- **Controles Manuales:** Permite mover el robot manualmente utilizando las teclas de flecha del teclado.

1.3.4 Configuración de la Máquina de Estados

La máquina de estados está compuesta por los siguientes estados:

1. **Reposo:** - Publica el estado actual en */current_state*. - Permite las transiciones a *Exploración* o *Navegación* mediante comandos en el tópico */command*.
2. **Exploración:** - Activa la detección de objetos mediante el servidor de acción *object_detection*. - Permite las transiciones a *Reposo*, *Aproximación*, o *Navegación* según los eventos detectados.
3. **Aproximación:** - Acerca el robot a un objeto detectado utilizando el servidor de acción *approach_object*. - Permite transiciones a *Reposo*, *Exploración*, o *Navegación*.
4. **Navegación:** - Mueve el robot a una posición predefinida utilizando el servidor de acción *move_base*. - Permite transiciones a *Reposo* o *Exploración*.

1.3.5 Comandos y Transiciones

Los comandos para cambiar de estado se publican en el tópico */command*. Algunos ejemplos incluyen:

- **`modo exploracion`**: Cambia al estado de exploración.
- **`ir a HOME`**: Cambia al estado de navegación hacia la posición «HOME».
- **`acercarse_objetivo`**: Cambia al estado de aproximación hacia un objeto detectado.
- **`parar`**: Regresa al estado de reposo.

1.3.6 Parámetros Configurables

Los parámetros del sistema se configuran en archivos YAML, que incluyen:

- **`configGeneral.yaml`**: Contiene configuraciones globales, como frecuencias de procesamiento, velocidades y distancias mínimas.
- **`config.yaml`**: Específico para cada estado, define parámetros como umbrales de detección y destinos predefinidos.

1.3.7 Tópicos Importantes

- **`/current_state`**: Publica el estado actual del robot.
- **`/command`**: Recibe comandos para cambiar el estado de la máquina de estados.
- **`/detected_objects`**: Publica la información de los objetos detectados.
- **`/cmd_vel`**: Envia comandos de movimiento al robot.
- **`/camera/rgb/image_raw`**: Proporciona imágenes RGB de la cámara.

—

Con esta guía, deberías tener todo lo necesario para ejecutar, configurar y controlar el sistema. Si necesitas más información, consulta la sección de configuración o los detalles específicos de los nodos en la documentación.

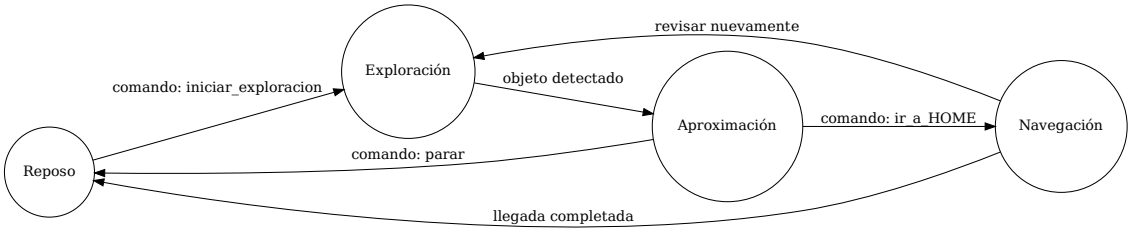
1.4 Máquina de Estados (squad_state_manager)

Descripción general del módulo *squad_state_manager*.

Estados principales:

- **Reposo:** Estado inicial de espera.
- **Exploración:** Activación del servidor de detección de objetos.
- **Aproximación:** Movimiento hacia un objeto detectado.
- **Navegación:** Desplazamiento a un punto específico.

Transiciones: - De *Reposo* a *Exploración* tras recibir el comando correspondiente. - De *Exploración* a *Aproximación* tras detectar un objeto. - De *Aproximación* a *Reposo* si se alcanza el objetivo o tras cierto tiempo.



TurtleBot State Manager

Este script gestiona el estado del TurtleBot utilizando SMACH, una biblioteca de Python para construir máquinas de estados finitas. Define y maneja diferentes estados como reposo, exploración, navegación y aproximación a objetos detectados. Además, integra una interfaz gráfica de usuario (GUI) construida con Tkinter para permitir la interacción manual y visualización del estado actual del robot y la imagen de la cámara.

Dependencias:

- rospy
- smach
- smach_ros
- actionlib
- std_msgs.msg
- squad_exploracion.msg

- sensor_msgs.msg
- cv_bridge
- PIL (Pillow)
- cv2 (OpenCV)
- move_base_msgs.msg
- geometry_msgs.msg
- threading
- tkinter

class squad_state_manager.**BaseState**(***kwargs*)

Bases: State

Clase base para todos los estados que publica el estado actual.

Esta clase sirve como clase base para los diferentes estados de la máquina de estados finita (SMACH). Publica el estado actual y suscribe a comandos para gestionar transiciones entre estados.

Hereda de:
smach.State: Clase base proporcionada por SMACH para definir estados.

Atributos:
state_pub (Publisher): Publicador al tópico "/current_state" para publicar el estado actual. command_sub (Subscriber): Suscriptor al tópico "/command" para recibir comandos de transición. comando (str): Comando actual recibido desde el tópico "/command". satate_name (str): Nombre del estado actual.

execute(*userdata*)

Publica el nombre del estado actual.

Este método debe ser implementado en las subclases para definir el comportamiento específico del estado.

Parámetros
userdata – Datos de usuario proporcionados por SMACH.

Devuelve
Resultado del estado que determina la transición siguiente.

Tipo del valor devuelto
str

state_callback(*msg*)

Callback para recibir comandos desde el tópico "/command".

Actualiza el atributo "comando" con el comando recibido en minúsculas.

Parámetros
msg (*String*) – Mensaje con el comando recibido.

class squad_state_manager.**EstadoApproach**

Bases: *BaseState*

Estado de Aproximación del robot que maneja el acercamiento a objetos detectados.

Este estado gestiona la lógica para que el robot se acerque a un objeto detectado utilizando un servidor de acción específico para el control de aproximación.

Hereda de:
BaseState: Clase base para estados en la máquina de estados finita (SMACH).

Atributos:

client_approach (SimpleActionClient): Cliente de acción para el control de aproximación. detectar_multiples_objetos (bool): Indicador para detectar múltiples objetos. acercarse_objetos (bool): Indicador para acercarse a objetos detectados. volver_casa (bool): Indicador para volver a la posición «HOME». last_object_detected (DetectedObject): Último objeto detectado. comando (str): Comando actual recibido.

`control_mode_callback`(*msg*)

Callback para el suscriptor del modo de control.

Cambia el modo de control del robot entre "manual" y "autonomous".

Parámetros

msg (*String*) – Mensaje con el modo de control.

`execute`(*userdata*)

Ejecuta el comportamiento principal del estado de Aproximación.

Este método gestiona la lógica para acercarse a un objeto detectado y las transiciones hacia otros estados basándose en el resultado de la aproximación.

Parámetros

userdata – Datos de usuario proporcionados por SMACH.

Devuelve

El resultado del estado, que determina la transición siguiente.

Tipo del valor devuelto

str

`feedback_cb`(*feedback*)

Callback para recibir feedback del servidor de aproximación.

Actualmente, este callback no realiza ninguna acción adicional.

Parámetros

feedback – Feedback recibido del servidor de acción de aproximación.

`class` squad_state_manager.**EstadoExploracion**

Bases: *BaseState*

Estado de Exploración del robot que gestiona la detección de objetos y el control autónomo.

Este estado coordina los comportamientos del robot durante la fase de exploración, incluyendo la detección de objetos, la aproximación a objetos detectados y la navegación autónoma.

Hereda de:

BaseState: Clase base para estados en la máquina de estados finita (SMACH).

Atributos:

client_object_detection (SimpleActionClient): Cliente de acción para la detección de objetos. client_autonomous_control (SimpleActionClient): Cliente de acción para el control autónomo. object_sub (Subscriber): Suscriptor al tópico de objetos detectados. state_sub (Subscriber): Suscriptor al tópico del estado actual. control_mode_sub (Subscriber): Suscriptor al tópico del modo de control. feedback_timeout (Duration): Tiempo máximo de espera para recibir feedback de detección. last_feedback_time (Time): Última vez que se recibió feedback. detectar_multiples_objetos (bool): Indicador para detectar múltiples objetos. acercarse_objetos (bool): Indicador para aproximarse a objetos detectados. volver_casa (bool): Indicador para volver a la posición «HOME». comando (str): Comando actual recibido. control_mode (str): Modo de control actual ("manual" o "autonomous"). autonomous_control_active (bool): Estado de la acción de control autónomo. objeto_detectado (bool): Indicador de detección de objeto. last_object_detected (DetectedObject): Último objeto detectado.

control_mode_callback(*msg*)

Callback para el suscriptor del modo de control.

Cambia el modo de control del robot entre “manual” y “autonomous”.

Parámetros
msg (*String*) – Mensaje con el modo de control.

execute(*userdata*)

Ejecuta el comportamiento principal del estado de Exploración.

Este método gestiona la lógica de detección de objetos, control autónomo, y las transiciones entre diferentes estados basándose en eventos y comandos.

Parámetros
userdata – Datos de usuario proporcionados por SMACH.

Devuelve
El resultado del estado, que determina la transición siguiente.

Tipo del valor devuelto
str

feedback_cb(*feedback*)

Callback para recibir feedback del servidor de detección de objetos.

Actualiza el tiempo del último feedback recibido.

Parámetros
feedback – Feedback recibido del servidor de acción de detección de objetos.

object_callback(*msg*)

Callback para el suscriptor de objetos detectados.

Actualiza el estado interno cuando se detecta un objeto.

Parámetros
msg (*DetectedObject*) – Mensaje con la información del objeto detectado.

class squad_state_manager.**EstadoNavegacion**

Bases: *BaseState*

Estado de Navegación del robot que maneja el desplazamiento a una posición predefinida.

Este estado utiliza el servidor de acción *move_base* para mover el robot a una ubicación específica definida en un diccionario de estaciones.

Hereda de:
BaseState: Clase base para estados en la máquina de estados finita (SMACH).

Atributos:
client (SimpleActionClient): Cliente de acción para *move_base*. satate_name (str): Nombre del estado actual. comando (str): Comando actual recibido.

control_mode_callback(*msg*)

Callback para el suscriptor del modo de control.

Cambia el modo de control del robot entre “manual” y “autonomous”.

Parámetros
msg (*String*) – Mensaje con el modo de control.

execute(*userdata*)

Ejecuta el comportamiento principal del estado de Navegación.

Este método envía un objetivo al servidor de acción *move_base* para mover el robot a las coordenadas especificadas en el destino y gestiona las transiciones basadas en el resultado de la navegación.

Parámetros
userdata – Datos de usuario proporcionados por SMACH.

Devuelve
El resultado del estado, que determina la transición siguiente.

Tipo del valor devuelto
str

moveTo(*x*, *y*)

Envía un objetivo al servidor de acción *move_base* para mover el robot a las coordenadas especificadas.

Parámetros

- **x** (*float*) – Coordenada X del destino.
- **y** (*float*) – Coordenada Y del destino.

class squad_state_manager.**EstadoReposo**

Bases: *BaseState*

execute(*userdata*)

Publica el nombre del estado actual.

Este método debe ser implementado en las subclases para definir el comportamiento específico del estado.

Parámetros
userdata – Datos de usuario proporcionados por SMACH.

Devuelve
Resultado del estado que determina la transición siguiente.

Tipo del valor devuelto
str

class squad_state_manager.**InterfazManager**(*state_machine*)

Bases: object

Gestor de la Interfaz Gráfica de Usuario (GUI) para la Máquina de Estados del TurtleBot.

Esta clase maneja la creación y actualización de la interfaz gráfica, la interacción con los tópicos de ROS para enviar comandos y recibir actualizaciones del estado del robot. También gestiona la visualización de imágenes de la cámara en tiempo real.

Atributos:
state_machine (StateMachine): Máquina de estados finita (SMACH) para gestionar los estados del robot. state_names (list): Lista de nombres de los estados definidos en la máquina de estados. current_state (str): Nombre del estado actual del robot. command_state_mapping (dict): Mapeo de comandos a nombres de estados. cmd_vel_pub (Publisher): Publicador al tópico "/cmd_vel" para enviar comandos de movimiento. state_pub (Publisher): Publicador al tópico "/command" para enviar comandos de estado. control_mode_pub (Publisher): Publicador al tópico "/control_mode" para cambiar el modo de control. control_mode_sub (Subscriber): Suscriptor al tópico "/control_mode" para recibir actualizaciones del modo de control. bridge (CvBridge): Objeto para convertir mensajes de ROS Image a OpenCV. camera_sub (Subscriber): Suscriptor al tópico de la cámara. normal_camera_topic (str): Tópico de la cámara normal. action_camera_topic (str): Tópico de la cámara procesada para acciones. current_camera_topic (str): Tópico de la cámara actualmente suscrito. camera_image (ImageTk.PhotoImage): Imagen actual de la cámara para visualizar en la GUI. control_mode (str): Modo de control actual ("manual" o "autonomous"). window (Tk): Ventana principal de la GUI. state_buttons (dict): Diccionario de botones de estado en la GUI. exploration_controls_frame (Frame): Marco para los controles de exploración en la GUI.

camera_callback(*msg*)

Callback para el suscriptor de la cámara.

Convierte la imagen recibida de ROS a un formato compatible con Tkinter y la muestra en la GUI.

Parámetros
msg (*ROSImage*) – Mensaje de imagen recibido del tópico de la cámara.

change_exploration_mode()

Cambia el modo de exploración entre “manual” y “autonomous”.

Actualiza la GUI y publica el nuevo modo de control al tópico “/control_mode”.

control_mode_callback(*msg*)

Callback para el suscriptor del modo de control.

Actualiza el modo de control actual y realiza acciones adicionales si es necesario.

Parámetros
msg (*String*) – Mensaje con el modo de control (“manual” o “autonomous”).

create_exploration_controls()

Crea los controles de exploración en la interfaz gráfica.

Incluye un botón para cambiar el modo de exploración y las instrucciones para el control manual mediante el teclado.

current_state_callback(*msg*)

Callback para el suscriptor del estado actual.

Actualiza el estado actual y resalta el botón correspondiente en la GUI.

Parámetros
msg (*String*) – Mensaje con el nombre del estado actual.

get_command_for_state(*state_name*)

Obtiene el comando correspondiente para un nombre de estado dado.

Parámetros
state_name (*str*) – Nombre del estado.

Devuelve
Comando asociado al estado o None si no hay mapeo.

Tipo del valor devuelto
str or None

on_arrow_down(*event*)

Maneja la pulsación de la tecla de flecha hacia abajo para mover el robot hacia atrás.

Parámetros
event – Evento de la tecla pulsada.

on_arrow_left(*event*)

Maneja la pulsación de la tecla de flecha hacia la izquierda para girar el robot a la izquierda.

Parámetros
event – Evento de la tecla pulsada.

on_arrow_right(*event*)

Maneja la pulsación de la tecla de flecha hacia la derecha para girar el robot a la derecha.

Parámetros
event – Evento de la tecla pulsada.

on_arrow_up(*event*)

Maneja la pulsación de la tecla de flecha hacia arriba para mover el robot hacia adelante.

Parámetros
event – Evento de la tecla pulsada.

on_key_release(*event*)

Maneja el evento de liberación de cualquier tecla para detener el movimiento del robot.

Parámetros
event – Evento de la tecla liberada.

on_state_button_click(*command*)

Maneja el evento de clic en un botón de estado.

Publica el comando correspondiente al tópico "/command" para cambiar el estado de la máquina de estados.

Parámetros
command (*str*) – Comando a publicar.

setup_gui()

Configura la interfaz gráfica de usuario utilizando Tkinter.

Crea la ventana principal, los botones de control, la visualización de la cámara y los controles de exploración.

subscribe_to_camera(*topic*)

Suscribe al tópico de la cámara especificado.

Desuscribe del tópico de cámara anterior si existe y suscribe al nuevo tópico.

Parámetros
topic (*str*) – Tópico de la cámara al que suscribirse.

switch_camera_topic(*new_topic*)

Cambia el tópico de la cámara a uno nuevo si es diferente al actual.

Parámetros
new_topic (*str*) – Nuevo tópico de la cámara al que suscribirse.

update_state_highlight()

Actualiza la interfaz gráfica para resaltar el botón correspondiente al estado actual y mostrar u ocultar los controles de exploración según sea necesario.

class squad_state_manager.**TurtleBotStateManager**

Bases: object

Gestor de Estados para el TurtleBot utilizando SMACH.

Esta clase inicializa la máquina de estados finita (SMACH) para gestionar los diferentes comportamientos del TurtleBot, incluyendo reposo, exploración, navegación y aproximación a objetivos. También configura la introspección para visualizar la máquina de estados y lanza la interfaz gráfica para la interacción del usuario.

Atributos:
sm (StateMachine): Máquina de estados finita que define los diferentes estados y sus transiciones. state_machine_thread (Thread): Hilo que ejecuta la máquina de estados. gui (InterfazManager): Gestor de la interfaz gráfica de usuario.

run_state_machine()

Ejecuta la máquina de estados finita (SMACH).

Este método es ejecutado en un hilo separado para no bloquear el hilo principal.

1.5 Paquete de Exploración

Este paquete contiene los servidores de acción necesarios para la detección y navegación.

Módulos:

- **squad_object_detection_action.py:** Detección de objetos en imágenes RGBD.
- **squad_approach_control_action.py:** Aproximación a objetos detectados.
- **squad_autonomous_control_action.py:** Navegación autónoma evitando obstáculos.

Nodos:

1.5.1 Detección de Objetos (squad_object_detection_action)

Este nodo procesa imágenes RGB y de profundidad para detectar objetos en el entorno.

Descripción del Nodo:

- Convierte imágenes de ROS a OpenCV usando *cv_bridge*.
- Detecta objetos por color (ejemplo: objetos rojos).
- Calcula coordenadas globales utilizando datos de la cámara y la odometría.
- Publica información en */detected_objects*.

Parámetros Importantes:

- **`color_threshold`**: Umbral de detección por color.
- **`topic_rgb`**: Tópico de entrada de imágenes RGB.
- **`topic_depth`**: Tópico de entrada de imágenes de profundidad.

Servidor de Acción para la Detección de Objetos en TurtleBot.

Este nodo implementa un servidor de acción que permite al TurtleBot detectar objetos en su entorno utilizando imágenes RGB y de profundidad. Coordina la conversión de imágenes ROS a OpenCV, la detección de objetos basados en color, el cálculo de coordenadas mundiales de los objetos detectados y la publicación de información relevante. Además, proporciona herramientas de depuración para visualizar imágenes procesadas y datos en tiempo real.

Funcionalidades Principales:

- Conversión de imágenes ROS a formatos utilizables por OpenCV.
- Detección de objetos basados en color (por ejemplo, objetos rojos).
- Cálculo de las coordenadas mundiales de los objetos detectados utilizando datos de profundidad y odometría.
- Publicación de información de objetos detectados en tópicos ROS.
- Proporciona una interfaz de retroalimentación para la frecuencia de procesamiento.
- Herramientas de depuración para visualizar imágenes procesadas y datos de profundidad.

Este servidor de acción está diseñado para ser parte integral de una máquina de estados finitos (SMACH) que gestiona el comportamiento del robot en diferentes estados, como exploración, aproximación a objetos y navegación.

class squad_object_detection_action.**TurtleBotObjectDetectionAction**

Bases: object

Servidor de Acción para la Detección de Objetos en TurtleBot.

Este servidor de acción procesa imágenes RGB y de profundidad para detectar objetos de interés en el entorno del TurtleBot. Coordina la conversión de imágenes, detección de objetos, cálculo de coordenadas mundiales y publicación de información detectada.

Hereda de:
objectlib.SimpleActionServer: Proporciona funcionalidades de servidor de acción.

Atributos:
server (SimpleActionServer): Servidor de acción para la detección de objetos. bridge (CvBridge): Instancia de CvBridge para convertir imágenes ROS a OpenCV. lock (threading.Lock): Bloqueo para controlar el acceso a datos compartidos. object_pub (Publisher): Publicador para la ubicación de objetos detectados. process_image_pub (Publisher): Publicador para imágenes procesadas. latest_image (Image): Última imagen RGB recibida. latest_depth (Image): Última imagen de profundidad recibida. latest_position (PoseStamped): Última posición del robot. latest_depth_cv (ndarray): Última imagen de profundidad convertida a OpenCV. image_count (int): Contador de imágenes procesadas. processing_times (deque): Cola para almacenar tiempos de procesamiento. fx (float): Parámetro intrínseco de la cámara (focal length x). fy (float): Parámetro intrínseco de la cámara (focal length y). cx (float): Parámetro intrínseco de la cámara (principal point x). cy (float): Parámetro intrínseco de la cámara (principal point y). camera_info_sub (Subscriber): Suscriptor al tópico de CameraInfo.

`calculate_world_coordinates`(*detected_objects*, *processed_image*)

Calcula las coordenadas mundiales de los objetos detectados a partir de sus coordenadas de píxel.

Utiliza la imagen de profundidad y los parámetros intrínsecos de la cámara para convertir las coordenadas de píxel a coordenadas mundiales.

Parámetros

- **detected_objects** (*list*) – Lista de objetos detectados con coordenadas de píxel.
- **processed_image** (*ndarray*) – Imagen procesada para dibujar información adicional.

Devuelve
Lista de objetos con coordenadas mundiales y la imagen procesada con anotaciones.

Tipo del valor devuelto
tuple

`callback`(*image_msg*, *depth_msg*, *odom_msg*)

Callback para procesar los mensajes sincronizados de imagen, profundidad y odometría.

Convierte las imágenes de ROS a formatos utilizables por OpenCV y almacena los datos recibidos.

Parámetros

- **image_msg** (*Image*) – Mensaje de imagen RGB.
- **depth_msg** (*Image*) – Mensaje de imagen de profundidad.
- **odom_msg** (*Odometry*) – Mensaje de odometría.

`camera_info_callback`(*msg*)

Callback para procesar el mensaje de CameraInfo y extraer parámetros intrínsecos.

Almacena los parámetros intrínsecos de la cámara y desuscribe del tópico una vez recibidos.

Parámetros
msg (*CameraInfo*) – Mensaje con información de la cámara.

detect_objects(*cv_image*)

Detecta objetos de color específico en una imagen RGB.

Utiliza la conversión a espacio de color HSV y segmentación por color para identificar objetos rojos, luego calcula sus coordenadas de píxeles.

Parámetros
cv_image (*ndarray*) – Imagen RGB en formato OpenCV.

Devuelve
Imagen procesada y lista de objetos detectados con sus coordenadas de píxel.

Tipo del valor devuelto
tuple

display_debug_info(*raw_image*, *processed_image*, *depth_image*)

Muestra información de depuración en las imágenes procesadas.

Dibuja timestamps, posición del robot y profundidad sobre las imágenes.

Parámetros

- **raw_image** (*ndarray*) – Imagen RGB original.
- **processed_image** (*ndarray*) – Imagen procesada con detecciones.
- **depth_image** (*ndarray*) – Imagen de profundidad procesada.

execute(*goal*)

Ejecuta la acción de detección de objetos.

Este método maneja el flujo principal de la acción, incluyendo la espera de CameraInfo, procesamiento de imágenes y publicación de resultados.

Parámetros
goal (*ObjectDetectionGoal*) – Objetivo de la acción recibido.

get_depth_at_pixel(*x*, *y*)

Obtiene la profundidad en un píxel específico de la imagen de profundidad.

Verifica que las coordenadas estén dentro de los límites de la imagen y maneja valores inválidos.

Parámetros

- **x** (*int*) – Coordenada x del píxel.
- **y** (*int*) – Coordenada y del píxel.

Devuelve
Valor de profundidad en metros o None si es inválido.

Tipo del valor devuelto
float or None

load_parameters()

Carga los parámetros esenciales del nodo desde los archivos de configuración.

Establece los tópicos de suscripción, frecuencia de procesamiento y otros parámetros clave.

process_latest_data(*feedback*)

Procesa los datos más recientes de imagen, profundidad y posición.

Detecta objetos en la imagen, calcula sus coordenadas mundiales y publica la información.

Parámetros
feedback (*ObjectDetectionFeedback*) – Feedback para actualizar la frecuencia de procesamiento.

subscribe_to_topics()

Suscribe a los tópicos necesarios y sincroniza los mensajes recibidos.

Utiliza message_filters para sincronizar las imágenes RGB, de profundidad y los datos de odometría.

transform_camera_to_global(*X_cam*, *Y_cam*, *Z_cam*)

Transforma las coordenadas desde el marco de la cámara al marco global.

Utiliza la orientación y posición actual del robot para realizar la transformación.

Parámetros

- **X_cam** (*float*) – Coordenada X en el marco de la cámara.
- **Y_cam** (*float*) – Coordenada Y en el marco de la cámara.
- **Z_cam** (*float*) – Coordenada Z en el marco de la cámara.

Devuelve
Coordenadas X y Y en el marco global o (None, None) si falla la transformación.

Tipo del valor devuelto
tuple

1.5.2 Aproximación a Objetos (squad_approach_control_action)

Este nodo mueve el robot hacia un objeto detectado.

Descripción del Nodo:

- Maneja una máquina de estados interna para controlar la aproximación.
- Utiliza *move_base* para realizar movimientos precisos.
- Configurable con parámetros como velocidades y distancia mínima.

Parámetros Importantes:

- **`linear_speed`**: Velocidad lineal del robot.
- **`angular_speed`**: Velocidad angular del robot.
- **`min_distance`**: Distancia mínima al objetivo.

class squad_approach_control_action.**ApproachObjectActionServer**(*name*)

Bases: object

Servidor de acción para la aproximación al objeto detectado.

Esta clase implementa un servidor de acción que coordina la aproximación del robot hacia un objeto detectado en el entorno. Utiliza una máquina de estados finitos (SMACH) para gestionar las diferentes etapas de la aproximación, incluyendo la verificación de la posición, movimiento hacia la posición objetivo, detección y centrado del objeto, aproximación final y espera después de la aproximación. Además, maneja la preempción de la acción para permitir la cancelación o interrupción del proceso en cualquier momento.

execute_cb(*goal*)

Callback de ejecución de la acción.

Coordina la máquina de estados para aproximarse al objeto detectado. Gestiona las transiciones entre estados como la verificación de la posición, movimiento hacia la posición objetivo, detección y centrado del objeto, aproximación final y espera después de la aproximación.

Parámetros
goal (*ApproachControlGoal*) – Objetivo de la acción (no utilizado en este caso).

object_callback(*msg*)

Callback para almacenar la información del objeto detectado.

Actualiza el objeto detectado con los datos recibidos del tópico de detección de objetos.

Parámetros
msg (*DetectedObject*) – Mensaje con la información del objeto detectado.

odom_callback(*msg*)

Callback para almacenar los datos de odometría.

Actualiza la posición actual del robot con los datos recibidos del tópico de odometría.

Parámetros
msg (*Odometry*) – Mensaje con la información de la posición del robot.

preempt_cb()

Callback para manejar la preempción de la acción.

Establece la bandera de preempción para indicar que se ha solicitado la interrupción de la acción en curso.

class squad_approach_control_action.**ApproachObjectState**(**args: Any*, ***kwargs: Any*)

Bases: State

Estado para aproximarse al objeto detectado.

Este estado mueve el robot hacia adelante en línea recta hasta que alcanza una distancia mínima especificada del objeto detectado. Monitorea continuamente la distancia al objeto para determinar cuándo detenerse.

execute(*userdata*)

Ejecuta la lógica del estado para aproximarse al objeto detectado.

Mueve el robot hacia adelante hasta que la distancia al objeto detectado sea menor o igual a la distancia mínima especificada. Si la distancia es adecuada, detiene el robot.

Parámetros
userdata (*smach.UserData*) – Datos de usuario proporcionados por SMACH.

Devuelve
Outcome del estado.

Tipo del valor devuelto
str

object_callback(*msg*)

Callback para almacenar la información del objeto detectado.

Actualiza el objeto detectado con los datos recibidos del tópico de detección de objetos.

Parámetros
msg (*DetectedObject*) – Mensaje con la información del objeto detectado.

class squad_approach_control_action.**CenterObjectState**(**args: Any*, ***kwargs: Any*)

Bases: State

Estado para centrar el objeto detectado en el campo de visión de la cámara.

Este estado ajusta la orientación del robot girando hacia el objeto detectado hasta que esté centrado en la imagen de la cámara. Utiliza las coordenadas del objeto para determinar la dirección del giro.

execute(*userdata*)

Ejecuta la lógica del estado para centrar el objeto en la cámara.

Ajusta la orientación del robot girando hacia el objeto detectado hasta que esté centrado en el campo de visión de la cámara. Si el objeto no está centrado dentro de una tolerancia definida, continúa girando en la dirección apropiada.

Parámetros
userdata (*smach.UserData*) – Datos de usuario proporcionados por SMACH.

Devuelve
Outcome del estado.

Tipo del valor devuelto
str

object_callback(*msg*)

Callback para almacenar la información del objeto detectado.

Actualiza el objeto detectado con los datos recibidos del tópico de detección de objetos.

Parámetros
msg (*DetectedObject*) – Mensaje con la información del objeto detectado.

class squad_approach_control_action.**CheckPositionState**(**args: Any*, ***kwargs: Any*)

Bases: State

Estado para verificar la posición actual del robot respecto al objetivo.

Este estado calcula la distancia y la diferencia de orientación entre la posición actual del robot y la posición objetivo. Determina si el robot ha alcanzado la posición deseada dentro de una tolerancia definida.

calculate_distance(*pos1*, *pos2*)

Calcula la distancia euclidiana entre dos posiciones.

Parámetros

- **pos1** (*geometry_msgs/Point*) – Primera posición.
- **pos2** (*geometry_msgs/Point*) – Segunda posición.

Devuelve
Distancia en metros.

Tipo del valor devuelto
float

calculate_orientation_diff(*ori1*, *ori2*)

Calcula la diferencia de orientación en yaw entre dos cuaterniones.

Parámetros

- **ori1** (*geometry_msgs/Quaternion*) – Primera orientación.
- **ori2** (*geometry_msgs/Quaternion*) – Segunda orientación.

Devuelve
Diferencia de orientación en radianes.

Tipo del valor devuelto
float

execute(*userdata*)

Ejecuta la lógica del estado para verificar la posición.

Calcula la distancia y la diferencia de orientación entre la posición actual y el objetivo. Determina si el robot está dentro de las tolerancias definidas para considerar que ha alcanzado la posición objetivo.

Parámetros
userdata (*smach.UserData*) – Datos de usuario proporcionados por SMACH.

Devuelve
Outcome del estado.

Tipo del valor devuelto
str

`quaternion_to_yaw`(*quat*)

Convierte un cuaternión a un ángulo yaw.

Parámetros
quat (*geometry_msgs/Quaternion*) – Cuaternión.

Devuelve
Ángulo yaw en radianes.

Tipo del valor devuelto
float

`class` squad_approach_control_action.**MoveToPositionState**(**args: Any*, ***kwargs: Any*)

Bases: State

Estado para mover el robot hacia la posición objetivo utilizando move_base.

Este estado envía un objetivo al servidor move_base para que el robot se desplace hacia la posición deseada. Monitorea el estado de la acción y maneja posibles resultados como éxito, aborto o preempción.

`execute`(*userdata*)

Ejecuta la lógica del estado para mover el robot hacia la posición objetivo.

Envía un objetivo al servidor move_base y monitorea su estado hasta que se complete la acción o se solicite una preempción.

Parámetros
userdata (*smach.UserData*) – Datos de usuario proporcionados por SMACH.

Devuelve
Outcome del estado.

Tipo del valor devuelto
str

`class` squad_approach_control_action.**StartObjectDetectionState**(**args: Any*, ***kwargs: Any*)

Bases: State

Estado para iniciar la detección de objetos mediante el servidor de acción de detección.

Este estado envía un objetivo al servidor de acción de detección de objetos para comenzar la detección. Monitorea el estado de la acción y maneja posibles resultados como éxito, aborto o preempción.

`execute`(*userdata*)

Ejecuta la lógica del estado para iniciar la detección de objetos.

Envía un objetivo vacío al servidor de acción de detección de objetos para comenzar la detección. Verifica si el servidor está disponible y maneja posibles preempciones.

Parámetros
userdata (*smach.UserData*) – Datos de usuario proporcionados por SMACH.

Devuelve
Outcome del estado.

Tipo del valor devuelto
str

class squad_approach_control_action.**StopObjectDetectionState**(**args: Any*, ***kwargs: Any*)

Bases: State

Estado para detener la detección de objetos mediante el servidor de acción de detección.

Este estado cancela cualquier objetivo pendiente o en curso en el servidor de acción de detección de objetos para detener la detección.

execute(*userdata*)

Ejecuta la lógica del estado para detener la detección de objetos.

Cancela cualquier objetivo activo o pendiente en el servidor de acción de detección de objetos y maneja posibles preempciones.

Parámetros
userdata (*smach.UserData*) – Datos de usuario proporcionados por SMACH.

Devuelve
Outcome del estado.

Tipo del valor devuelto
str

class squad_approach_control_action.**WaitState**(**args: Any*, ***kwargs: Any*)

Bases: State

Estado de espera tras aproximarse al objeto.

Este estado mantiene el robot detenido durante un tiempo definido, permitiendo que se realicen acciones posteriores o simplemente esperando antes de finalizar la acción.

execute(*userdata*)

Ejecuta la lógica del estado de espera.

Mantiene el robot detenido durante el tiempo especificado. Monitorea continuamente si se ha solicitado una preempción para poder interrumpir la espera si es necesario.

Parámetros
userdata (*smach.UserData*) – Datos de usuario proporcionados por SMACH.

Devuelve
Outcome del estado.

Tipo del valor devuelto
str

1.5.3 Control Autónomo (squad_autonomous_control_action)

Este nodo permite la navegación autónoma evitando obstáculos.

Descripción del Nodo:

- Segmenta el entorno en regiones (frente, izquierda, derecha, etc.) usando datos de LIDAR.
- Toma decisiones de movimiento en función de distancias detectadas.
- Configurable con parámetros para velocidades y distancias mínimas.

Parámetros Importantes:

- **`front_distance_threshold`**: Distancia mínima para avanzar.
- **`side_distance_threshold`**: Distancia mínima lateral.
- **`linear_speed`**: Velocidad lineal del robot.

Servidor de acción para el control autónomo del robot.

Este nodo implementa una acción que permite controlar el robot de manera autónoma, evitando obstáculos y colisiones con las paredes. La acción puede ser activada o cancelada mientras se cambia de modo. Está diseñada para ser extendida con algoritmos de control más avanzados en el futuro.

class squad_autonomous_control_action.**AutonomousControlActionServer**(*name*)

Bases: object

Servidor de acción para el control autónomo del robot.

Esta clase implementa la lógica para controlar el robot de manera autónoma, evitando colisiones con obstáculos detectados por el sensor LIDAR.

decide_motion(*regions*)

Decide el movimiento del robot basado en las regiones del LIDAR.

Parámetros
regions – Diccionario con las distancias mínimas en cada región.

Devuelve
Mensaje Twist con las velocidades lineal y angular.

execute_cb(*goal*)

Método de ejecución de la acción.

Controla el robot de manera autónoma, evitando obstáculos basados en los datos del LIDAR. La acción puede ser preempted si se recibe una solicitud de cancelación.

Parámetros
goal – Objetivo de la acción (no utilizado en este caso).

get_laser_regions()

Divide las lecturas del LIDAR en regiones.

Devuelve
Diccionario con las distancias mínimas en cada región.

laser_callback(*data*)

Callback para almacenar los datos del LIDAR.

Parámetros
data – Mensaje de tipo LaserScan con los datos del sensor.

stop_robot()

Detiene el robot publicando velocidades cero.

CHAPTER 2

Indices y tablas

- genindex

Índice de Módulos Python

S

Índice

F

G

I

L

M

O

P