Taller 9: Percepción con LiDAR e IMU

Robótica Móvil

2^{do} cuatrimestre 2025

1 Introducción

En este taller se desarrollarán las funcionalidades para trabajar con dos sensores con los que no trabajamos hasta ahora: LiDAR (*Light Detection and Ranging*) y unidad inercial (*Inertial Measurement Unit*, IMU). De la IMU, utilizaremos únicamente la información provista por el giróscopo. El taller consiste en implementar diversos algoritmos utilizando el código base indicado en cada ejercicio. Lea atentamente el enunciado y los comentarios del código base.

1.1 Material requerido

Para resolver el taller, se deberá descargar del campus de la materia el paquete `imu_laser`. En éste se incluye el código, archivos `.launch.py`, configuraciones de RViz2 y escenas de CoppeliaSim.

Este taller necesita que el simulador CoppeliaSim sea capaz de entender los mensajes del tipo `sensor_msgs/msg/Imu`, por lo que es necesario agregar este tipo de mensajes al paquete de interfaz `sim_ros2_interface`, tal como fue explicado en el enunciado del Taller 6.

1.2 Configuraciones de RViz2

Para facilitar la configuración del entorno de visualización, se proveen archivos `.rviz` los cuales pueden ser cargados desde la interfaz gráfica de RViz2:

File → Open Config → Selección del archivo .rviz correspondiente

Estos archivos de configuración añaden la visualización de tópicos predefinidos a la interfaz.

Ejercicio 1: Compensación de sesgo (*bias*) del giróscopo

En este ejercicio se busca realizar un preprocesamiento de la información obtenida por un sensor giroscópico (velocidad angular) montado sobre un robot móvil, con el objetivo de eliminar su sesgo (*bias*). Para ello, se debe primero adquirir una serie de muestras de los valores arrojados por el sensor y, a partir de las mismas, estimar el *bias*. Una vez completada la calibración (que tomará un tiempo determinado por el usuario), los posteriores sensados recibidos deberán ser corregidos para remover el *bias* correspondiente.

Deberán completar `/imu_laser/src/imu_calibrator.cpp` y `/imu_laser/src/imu_calibrator.h`. Se utilizará el archivo `/imu_laser/launch/imu.launch.py` para la ejecución de los nodos requeridos:

```
ros2 launch imu_laser imu.launch.py
```

Tienen la posibilidad de configurar el tiempo de calibración modificando el parámetro:

```
parameters=[{"calibrate": 0}]
```

el cual determina la cantidad de segundos que se deberán acumular mediciones. Pasado el tiempo de calibración, el nodo llama al método:

```
void calculate_bias()
```

el cual deberán completar para calcular efectivamente el *bias* con las mediciones acumuladas durante el tiempo de calibración.

Las mediciones de la IMU son recibidas de manera continua por medio del método:

```
void on_imu_measurement(const sensor_msgs::msg::Imu& msg)
```

el cual acumula mediciones durante el tiempo de calibración y luego realiza la corrección de las mediciones y la estimación de la orientación mediante la integración de estos valores.

Entonces, se pide implementar:

a) La estimación del *bias* y la posterior compensación de los valores recibidos.

b) El cálculo de la orientación del robot a partir de integrar en el tiempo las velocidades ya compensadas. Verificar que se elimina el *bias* y no se observa *drift* (deriva) significativo.

Nota: debido a que estamos trabajando con un robot que se mueve en el plano, solo necesitaremos la velocidad angular sobre el eje Z, es decir, el ángulo *yaw*.

Utilice la configuración de RViz2 ubicada en `/imu_laser/launch/imu.rviz` para verificar el correcto funcionamiento, comparando la orientación estimada con la *ground-truth*. La escena a utilizar de CoppeliaSim es `/imu_laser/coppeliaSim/imu.ttt`.

En RViz2 podrán observar un marco de referencia que representa la orientación estimada del robot. Responder:

a) ¿Qué sucede si se integran las mediciones sin calibrar la IMU?

b) ¿Al calibrar la IMU, el problema que observamos desaparece por completo?

Pueden enviar comandos de velocidad al robot para corroborar que las estimaciones se condicen con la simulación:

```
ros2 topic pub  /robot/cmd_vel geometry_msgs/msg/Twist
                    "{linear: {x: 0.0, y: 0.0, z: 0.0},
                      angular: {x: 0.0, y: 0.0, z: -0.1}}"
```

Ejercicio 2: Detección de landmarks

En este ejercicio se utilizará un LiDAR para detectar objetos especialmente colocados en la escena, que cumplirán el rol de *landmarks* o referencias del entorno. Se pide procesar los datos crudos de dicho sensor y realizar la detección de estos *landmarks*.
En el escenario a resolver, se sabe que existen "postes" (objetos cilíndricos que pueden ser detectados por el LiDAR) de 10 cm de diámetro y que están fijos. Debido a que el LiDAR devuelve muchas mediciones de distancia sobre el plano por cada sensado, un poste puede ser detectado por uno o más rayos del sensor. Al mismo tiempo, queremos reconocer estos *landmarks*, mediante una única posición relativa al robot.

Deberán trabajar con el archivo `/imu_laser/src/landmark_detector.cpp`, que recibe mediciones de sensado del LiDAR a través del método:

```
void on_laser_scan(const sensor_msgs::msg::LaserScan::SharedPtr msg)
```

el cual deberán completar para establecer centroides que representen la posición de los postes. Se pide, entonces:

a) Filtrar mediciones inválidas utilizando la información provista en el mensaje (ángulos y rangos válidos). Pasar la información de los mensajes a coordenadas cartesianas.

b) Agrupar las mediciones por cercanía teniendo en cuenta las dimensiones conocidas de los postes a detectar (utilizar coordenadas cartesianas y distancia euclídea).

c) Establecer centroides que representen los postes en relación al robot.

d) Se debe publicar un mensaje de tipo `robmovil_msgs::msg::LandmkarkArray`, que consiste de un arreglo de mensajes de tipo `robmovil_msgs::msg::Landmark`. Estos elementos describen la posición de un landmark en forma relativa al robot en coordenadas polares.

Para resolver el ejercicio, utilizar la escena `/imu_laser/coppeliaSim/laser_landmarks.ttt` y la configuración de RViz2 de `/imu_laser/launch/landmark_detector.rviz`. Para lanzar el nodo de detección con LiDAR, ejecutar:

```
ros2 launch imu_laser landmark_detector.launch.py
```

Para corroborar la correcta implementación del ejercicio, verificar en RViz2 que la estimación de la pose del *landmark* coincida con el centro geométrico de los rayos que impactan en cada poste.