Universidad de San Andrés, Depto. de Ingeniería

Práctica 5 - Mapeo y FastSLAM

I-402 - Principios de la Robótica Autónoma

Prof. Ignacio Mas, Tadeo Casiraghi y Bautista Chasco

27 de octubre de 2025

Fecha límite de entrega: 14/11/25, 23:59hs.
Modo de entrega: Enviar por el Aula Virtual del Campus el código (.m) comentado y los gráficos (.jpg ó .pdf) y/o animaciones.

1. Mapeo con poses conocidas

Un robot debe construir un mapa de grilla de ocupación (celdas $c_0, \ldots, c_n$) de un entorno unidimensional usando una secuencia de mediciones de un sensor de distancia.
Asumir el siguiente modelo de sensor: cada celda de la grilla con una distancia menor que la distancia medida se asume ocupada con una probabilidad de $p = 0.3$. Cada celda más allá de la distancia medida se asume ocupada con una probabilidad de $p = 0.6$. Las celdas ubicadas a más de $20cm$ por detrás de la distancia medida no cambian su probabilidad de ocupación.

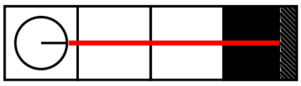

Figura 1.1: representación de grilla unidimensional

Calcular el mapa de grilla de ocupación resultante usando el modelo inverso del sensor usando Python. Asignar coordenadas a las celdas, desde 0 hasta 200 (incluyendo ambos valores), con incrementos de a 10 a un array `c` y los valores de *belief* a otro array `m`. Usar `matplotlib.pyplot.plot(c,m)` para visualizar el *belief*. Las mediciones y el *belief a priori* se detallan en la siguiente tabla:

Resolución de la grilla	$10cm$
Logitud del mapa (1-D)	$2m$
posición del robot	c_0
orientación del sensor	mirando hacia c_n (ver figura)
mediciones (en cm)	101, 82, 91, 112, 99, 151, 96, 85, 99, 105
prob. *a priori*	0.5

2. Implementación de algoritmo FASTSLAM

En este ejercicio se implementará el algoritmo FASTSLAM basado en landmarks. Se asume que los landmarks son identificables por lo que el problema de asociación de datos está resuelto.

2.1. Notas preliminares

Estaremos utilizando el código de publicación de landmarks del TP4. Solo tienen que agregar lo siguiente:

- Archivo launch en carpeta de launch: `launch_my_fastslam.launch.py`
- Agregado a setup.py: Alias `"fastslam = custom_code.my_fastslam:main",`
- Archivo pythyon junto con otros codigos: `my_fastslam.py`

El orden de lanzamiento sería entonces:

- Terminal 1: `ros2 launch custom_code launch_my_fastslam.launch.py`
- Terminal 2: `ros2 launch turtlebot3_custom_simulation custom_room.launch.py`

Notarán que esta vez no se abrirá Rviz2. Corriendo estos dos launch files se estará corriendo la simulación y se publicará la información necesaria para ejecutar un FastSlam con landmarks. Esto es:

- /delta : Tipo de dato DeltaOdom (custom_msgs). Contiene la data de deltas de odometría.
- /observed_landmarks : Tipo de dato PoseArray (geometry_msgs). Contiene la data de rango y ángulo relativo de los landmarks vistos. El valor de x es el rango, y el de z el ángulo.

Para cualquier cálculo de odometría puede asumir alphas:

$$[0.2,\ 0.2,\ 0.001,\ 0.001]$$

Puede asumir también que el desvío estándar del error del rango y el ángulo medido es de 0.05 (metros y radianes respectivamente)

2.2. Paso de corrección FASTSLAM

Deberán implementar un paquete de ROS2 en python que tome la data publicada y ejecute el algoritmo de FastSlam. Este paquete deberá tener un nodo que publique la posición estimada del robot, como también las posiciónes y covarianzas de cada landmark de la mejor partícula. Para publicar y visualizar la posición del robot puede elegir el método que deseen. Para publicar los landmarks puede hacer uso del siguiente código:

```
def quaternion_from_yaw(yaw):
    q = Quaternion()
    q.x = 0.0
    q.y = 0.0
    q.z = np.sin(yaw / 2.0)
    q.w = np.cos(yaw / 2.0)
    return q

def make_landmark_marker(self, idx, x, y):
    m = Marker()
    m.header.frame_id = "map"
    m.header.stamp = self.get_clock().now().to_msg()
    m.id = idx
    m.type = Marker.SPHERE  # point marker
    m.action = Marker.ADD
    m.pose.position.x = x
    m.pose.position.y = y
    m.pose.position.z = 0.0
    m.scale.x = 0.1  # sphere radius
    m.scale.y = 0.1
    m.scale.z = 0.1
    m.color.r = 1.0
    m.color.g = 0.0
    m.color.b = 0.0
    m.color.a = 1.0
    return m

def make_covariance_marker(self, idx, x, y, cov):
    # Compute ellipse parameters
    vals, vecs = np.linalg.eigh(cov)
    order = vals.argsort()[::-1]
    vals, vecs = vals[order], vecs[:,order]
    angle = np.arctan2(vecs[1,0], vecs[0,0])
    scale_x = 30*2*np.sqrt(vals[0])  # width
    scale_y = 30*2*np.sqrt(vals[1])  # height

    m = Marker()
    m.header.frame_id = "map"
    m.header.stamp = self.get_clock().now().to_msg()
    m.id = idx
    m.type = Marker.CYLINDER
    m.action = Marker.ADD
```

```
    m.pose.position.x = x
    m.pose.position.y = y
    m.pose.position.z = 0.0
    # rotation about z-axis

    q = quaternion_from_yaw(angle)
    m.pose.orientation = q

    m.scale.x = scale_x
    m.scale.y = scale_y
    m.scale.z = 0.01  # thin cylinder for 2D ellipse
    m.color.r = 0.0
    m.color.g = 0.0
    m.color.b = 1.0
    m.color.a = 0.3
    return m
```

Para usarlo tendrán que importar **Quaternion** de **geometry_msgs** y **Marker** de **visualization_msgs**. Luego, importando **MarkerArray** de **visualization_msgs** podrán hacer una lista de markers con los landmarks y sus covarianzas haciendo algo similar a esto:

```
best_particle = #Get best particle
ma = MarkerArray()
for lm_id, (lm_mu, lm_sigma) in best_particle.landmarks.items():
    ma.markers.append(self.make_landmark_marker(lm_id*2, lm_mu[0],
        lm_mu[1]))
    ma.markers.append(self.make_covariance_marker(lm_id*2+1, lm_mu
       [0], lm_mu[1], lm_sigma))

self.my_landmark_publisher.publish(ma)
```

Esto lo podrán visualizar en Rviz2. Se recomienda hacer un launch file que lanze tanto Rviz2 como su nodo de fastslam para facilitar el desarollo. Tendrán que guardar la configuración de su Rviz2 en el paquete para que podamos visualizar su trabajo una vez entregado.