Robótica Inteligente

Práctica final: Mapeado láser

David Revillas
drevillas002@ikasle.ehu.eus

Mónica Rampérez
mramperez001@ikasle.ehu.eus

Abstract

El objetivo de esta práctica final es el de construir un mapa del entorno mientras se teleopera el robot a través del teclado utilizando información de odometría.

1. Introducción

En esta práctica final se pretende desarrollar un módulo que permita construir un mapa del entorno mientras se teleopera el robot a través del teclado. Para ello, se necesita proyectar las lecturas del láser en el espacio de coordenadas global en el que se mueve el robot, haciendo uso de la odometría.

1.1. Análisis del problema

La idea es dibujar un mapa de puntos del entorno utilizando la información recibida por la parte delantera del escáner láser del robot. Para ello, será necesario proyectar los puntos detectados por el lídar teniendo en cuenta la posición del robot en cada momento, utilizando su odometría. Además, para que el mapeado del entorno sea completo, se podrá mover el robot de manera teleoperada, permitiendo que llegue a todos los lugares necesarios para completar el mapa.

2. Estrategia utilizada

2.1. Descripción del *Turtlebot*

El robot utilizado en esta práctica es un *Turtlebot* con lídar. Este escáner permite obtener un nube de puntos en 360º mapeándolos de la forma indicada en los ejes de la Figura 1.

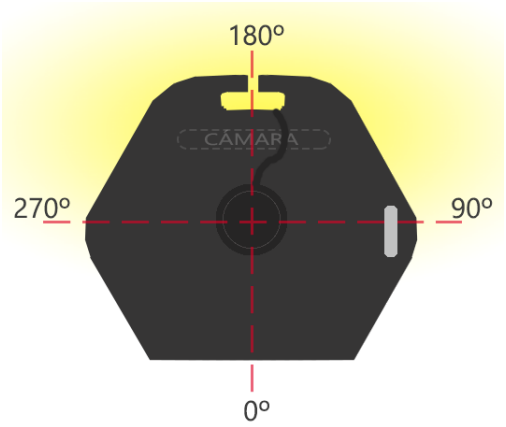


Figura 1: Esquema del robot *Turtlebot*.

2.2. *Topics* utilizados

Para resolver esta práctica, la primera tarea es obtener la información del entorno. Nuestro robot *Turtlebot* está provisto de un dispositivo lídar que permite obtener un nube de puntos en 360º, aunque en este caso sólo nos interesa utilizar los 180º que comprenden la parte delantera, como se indica en la Figura 1. Por tanto, utilizaremos la información recibida entre los 90º y los 270º. Esta nube de puntos la proporciona el *topic* `scan` correspondiente al lídar, el cuál implementa el formato de mensaje `sensor_msgs::LaserScan` con lo siguientes datos:

```
std_msgs/Header header
     uint32 seq
     time stamp

string frame_id

float32 angle_min

float32 angle_max

float32 angle_increment

float32 time_increment

float32 scan_time

float32 range_min

float32 range_max

float32[] ranges

float32[] intensities
```

Como vemos, el escáner proporciona gran cantidad de información, aunque los datos que realmente nos van a ser útiles en esta tarea se encuentran en el array `float32[] ranges`, que indica la distancia a la que se encuentra cada punto detectado para cada uno de los ángulos. De esta manera, la suscripción a este *topic* nos permite conocer la situación del robot con el entorno prácticamente en tiempo real.

Desde el *topic* de odometría obtenemos la siguiente información del mensaje `nav_msgs/Odometry`:

```
Header header

string child_frame_id

geometry_msgs/PoseWithCovariance pose

geometry_msgs/TwistWithCovariance twist
```

Utilizaremos la información de `pose` para conocer la posición y la rotación del robot.

Por último, en este caso, para aplicar la velocidad de movimiento al robot, se utiliza el modulo *teleop* previamente implementado.

2.3. Mapeado del entorno

Gracias a la odometría podemos conocer en cada momento las posiciones x_r e y_r del robot, así como su ángulo de orientación θ_r.

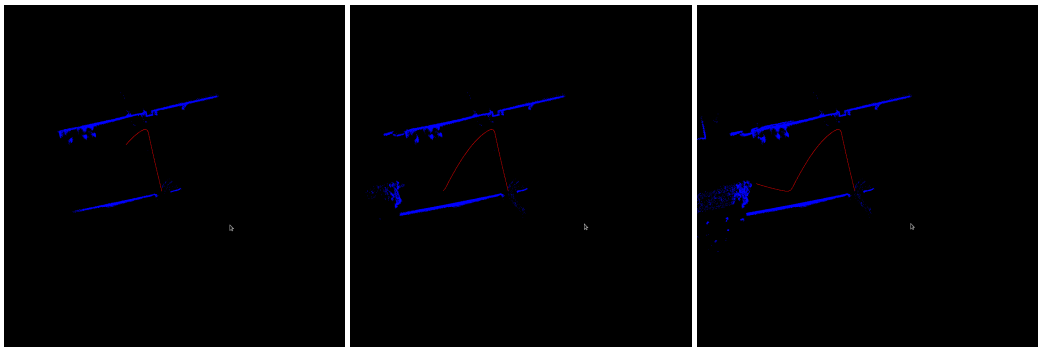
Además, para cada uno de los 180 puntos detectados por el láser, tenemos la información de la distancia I a la que se ha detectado y el ángulo θ_l al que corresponde.

Hemos establecido un umbral de 5 metros para dibujar sólo aquellos puntos que se detecten por debajo de esa distancia y así, con esta información es posible proyectar cada punto detectado en el mapa.

$$x_m = x_r + I * cos(\theta_r + \theta_l)$$

$$y_m = y_r + I * sin(\theta_r + \theta_l)$$

De esta manera, para cada instante de tiempo obtenemos las coordenadas de cada punto x_m, y_m a dibujar en el mapa, como se muestra en la Figura 2.



(a) (b) (c)

Figura 2: Construcción progresiva del mapa del pasillo. A la izquierda de cada imagen, se encontrarían la escaleras, a la derecha, el laboratorio de los otros compañeros y en el punto de partida, la salida de nuestro laboratorio.

3. Evaluación del comportamiento

Se ha probado a construir tanto el mapa del entorno de la clase como el del pasillo, con éxito en ambos casos. Sin embargo, es notable que los movimientos muy veloces y los giros rápidos provocan errores considerables que quedan reflejados en el dibujo del mapa.

En la Figura 3 se observa una de las pruebas realizadas para el construir el mapa de la clase.

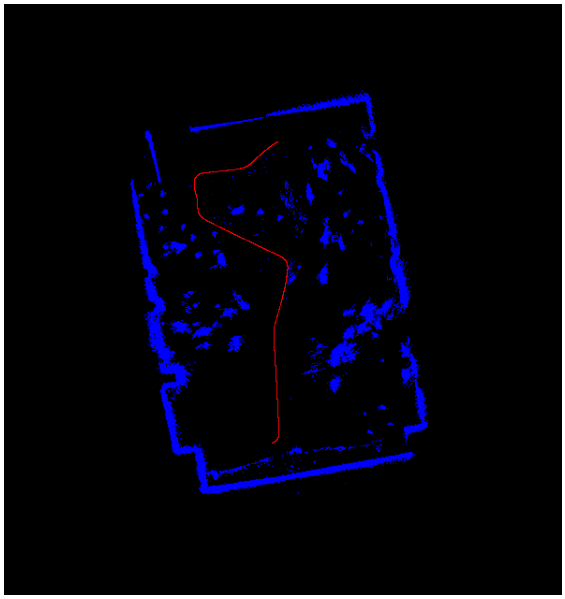


Figura 3: Mapa de la clase. La esquina superior izquierda correspondería a la entrada de la misma.

4. Extras

4.1. Sorteo de obstáculos

Aunque no se trate de una mejora en sí del proceso de mapeado, se ha conseguido unir el módulo de sorteo de obstáculos, desarrollado en la primera práctica, junto al módulo implementado. Es por ello que la trayectoria que se muestra en la Figura 3 resulta tan suave. De esta manera evitamos los saltos bruscos de velocidad y movimiento ocasionados por el módulo de *teleop*.

4.2. Difuminado

Sólo por diversión, implementamos un difuminado de la imagen entre las líneas 193 y 198 del fichero `mapping.cpp`. En la implementación original, el callback `processOdom` actualiza la variable global `cv::Mat mymap` coloreando la matriz con datos de tipo `cv::Vec3b`, es decir, coloreando un píxel en rojo en la posición actual del *turtlebot* y un píxel azul con el obstáculo localizado mediante el lídar. Sin embargo, esta implementación no hace más que ir añadiendo color indefinidamente a la matriz.

Nuestra implementación proponía reducir esos valores de manera constante mientras el escaneado del lídar no volviera a detectar el mismo obstáculo, concretamente, sustrayendo el vector `cv::Vec3b(12, 0, 1)` a cada elemento de la matriz, lo que supone que aproximadamente en $255/12 \approx 21$ llamadas consecutivas a la función, un píxel que en cierto momento se hubiera detectado como obstáculo (elemento establecido como `cv::Vec3b(255, 0, 0)` en la matriz), en la llamada 22 desaparecería de la imagen. De manera similar para la trayectoria seguida por el robot, pero esta vez durante 255 llamadas.

De esta manera, obtenemos un desvanecimiento de imágenes antiguas de forma progresiva, como se observa en la Figura 4. Se puede apreciar el desvanecimiento de la trayectoria roja, y más sutilmente, los obstáculos azules.

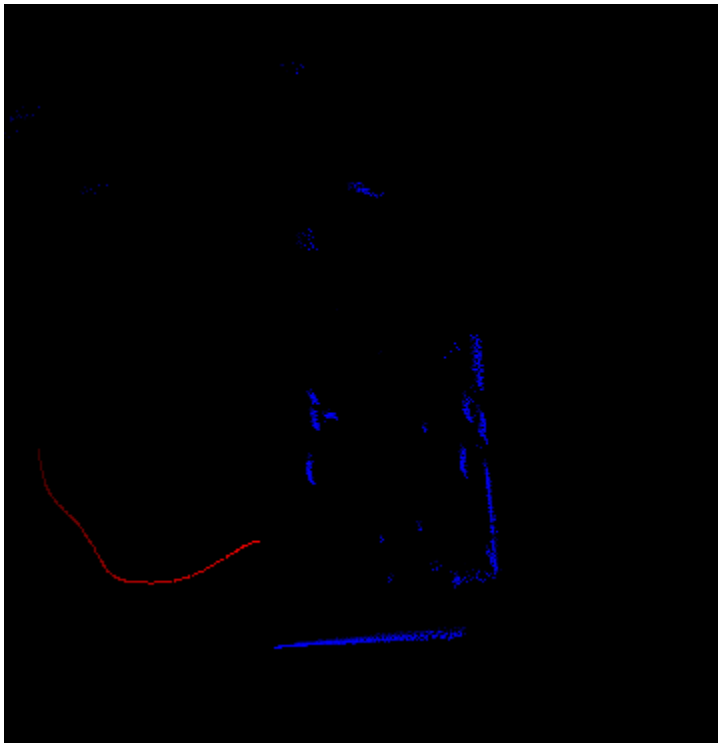


Figura 4: Resultado tras el difuminado aplicado en cada llamada a `processOdom`.