# EECS 4421 LAB2

## Name: Mahfuz Rahman

## Student No.: 217847518

1. **A screenshot of a computer

   Description automatically generatedGazebo View of the world**

**Code that generates the map (collect\_lidar.py) :**

import math

import numpy as np

import rclpy

from rclpy.node import Node

from nav\_msgs.msg import Odometry

from sensor\_msgs.msg import LaserScan

import cv2

def euler\_from\_quaternion(quaternion):

"""

Converts quaternion (w in last place) to euler roll, pitch, yaw

quaternion = [x, y, z, w]

"""

x = quaternion.x

y = quaternion.y

z = quaternion.z

w = quaternion.w

sinr\_cosp = 2 \* (w \* x + y \* z)

cosr\_cosp = 1 - 2 \* (x \* x + y \* y)

roll = np.arctan2(sinr\_cosp, cosr\_cosp)

sinp = 2 \* (w \* y - z \* x)

pitch = np.arcsin(sinp)

siny\_cosp = 2 \* (w \* z + x \* y)

cosy\_cosp = 1 - 2 \* (y \* y + z \* z)

yaw = np.arctan2(siny\_cosp, cosy\_cosp)

return roll, pitch, yaw

class CollectLidar(Node):

\_WIDTH = 513

\_HEIGHT = 513

\_M\_PER\_PIXEL = 0.05

def \_\_init\_\_(self):

super().\_\_init\_\_('collect\_lidar')

self.get\_logger().info(f'{self.get\_name()} created')

self.\_map = np.zeros((CollectLidar.\_HEIGHT, CollectLidar.\_WIDTH), dtype=np.uint8)

self.\_cur\_x = 0.0

self.\_cur\_y = 0.0

self.\_cur\_yaw = 0.0

self.create\_subscription(Odometry, "/odom", self.\_odom\_callback, 1)

self.create\_subscription(LaserScan, "/scan", self.\_scan\_callback, 1)

def \_scan\_callback(self, msg):

angle\_min = msg.angle\_min

angle\_increment = msg.angle\_increment

ranges = msg.ranges

for i, r in enumerate(ranges):

if r < msg.range\_max:

angle = angle\_min + i \* angle\_increment

# Converting from LIDAR polar coordinates to Cartesian coordinates

x\_car = r \* math.cos(angle)

y\_car = r \* math.sin(angle)

# Transform from robot-relative cartesian coordinates to world coordinates

x\_world = self.\_cur\_x + x\_car \* math.cos(self.\_cur\_yaw) - y\_car \* math.sin(self.\_cur\_yaw)

y\_world = self.\_cur\_y + x\_car \* math.sin(self.\_cur\_yaw) + y\_car \* math.cos(self.\_cur\_yaw)

# Convert world coordinates to grid coordinates

grid\_x = int(x\_world / CollectLidar.\_M\_PER\_PIXEL + CollectLidar.\_WIDTH // 2)

grid\_y = int(y\_world / CollectLidar.\_M\_PER\_PIXEL + CollectLidar.\_HEIGHT // 2)

# Mark the grid cell as occupied if within bounds

if 0 <= grid\_x < CollectLidar.\_WIDTH and 0 <= grid\_y < CollectLidar.\_HEIGHT:

self.\_map[grid\_y, grid\_x] = 255 # Occupied

cv2.imshow('map', self.\_map)

cv2.waitKey(10)

def \_odom\_callback(self, msg):

pose = msg.pose.pose

self.\_cur\_x = pose.position.x

self.\_cur\_y = pose.position.y

\_, \_, self.\_cur\_yaw = euler\_from\_quaternion(pose.orientation)

self.get\_logger().info(f"Robot at ({self.\_cur\_x}, {self.\_cur\_y}, {self.\_cur\_yaw})")

def main(args=None):

rclpy.init(args=args)

node = CollectLidar()

try:

rclpy.spin(node)

except KeyboardInterrupt:

pass

rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

main()

1. **Mapping on a 2D map using LIDAR**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

1. **How did you decide to represent the map and update it when sensor measurements are obtained?**

Since we are using a robot equipped with a LIDAR scanner, we get polar coordinates or basically a list of distances and angles that represents the distance to an object at a specific angle from the robot’s position at the time.

Thus, we can convert the polar coordinates to cartesian coordinates and plot onto a 2D map. For this, I used the following conversion:

r is the range (distance) from the LIDAR.

𝜃 is the angle of the LIDAR beam. This is computed using *angle\_min + i \* angle\_increment,* where *i*  is the index of the range.

However, the LIDAR data is relative to the robot’s current position and orientation. We thus use the robot’s global position which is x, y, and yaw angle from odometry to transform the LIDAR points and plot onto the map. We calculate this using the following:

Now that we have the coordinates, we need to plot them onto our 2D map. To achieve this, we use the following method:

We then mark these grids onto the map as occupied.

1. **How did you decide where to drive the robot?**

We can drive around the robot using a number of ways. I explored two methods, where one approach was simply to use the *teleop\_twist\_keyboard* node. The other approach was to drive the robot automatically utilizing the sensor data and a path planning algorithm to navigate around obstacles. The robot can leverage Dijkstra’s algorithm to decide on a path to a goal location. If an obstacle is detected within a certain range, the robot adjusts its trajectory by steering away from the detected object.

1. **Was the model consistent over all measurement obtained?**

Considering the simplicity of our environment and robot, the model was generally consistent over time. Since our environment was static, the readings from the robot’s sensors were always same. However, I observed some factors that affected consistency due to the robot’s movements. For instance sudden movements of the robot at high speed or sudden stopping. The robot’s drifts over time (due to wheel slip or other factors), introduced inconsistencies in how sensor readings were integrated into the map.