Mobile robotics assignment 2

Name: Hamze Hammami

Student code: 1906232

Module code: CE315

Table of Contents

[Introduction 2](#_Toc98769277)

[-Sensors 2](#_Toc98769278)

[1-Tactile sensors 2](#_Toc98769279)

[2-Proximity Sensors 2](#_Toc98769280)

[3-Structure sensing object 2](#_Toc98769281)

[4-Internal sensors 3](#_Toc98769282)

[- navigation and path planning 3](#_Toc98769283)

[1-Navigation 3](#_Toc98769284)

[2-Path planning 3](#_Toc98769285)

[-laser/trajectory implantation 4](#_Toc98769286)

[1-Laser control 4](#_Toc98769287)

[2-odometry control 5](#_Toc98769288)

[Software design 6](#_Toc98769289)

[-Control flowcharts 6](#_Toc98769290)

[Code development 6](#_Toc98769291)

[-main code 6](#_Toc98769292)

[-launch codes 6](#_Toc98769293)

[Experiential results 7](#_Toc98769294)

[-Velocity data 8](#_Toc98769295)

[-Laser map sketch 9](#_Toc98769296)

[-Laser data 9](#_Toc98769297)

[appendix 10](#_Toc98769298)

[-tutorial\_pkg\_node 10](#_Toc98769299)

[Citation 14](#_Toc98769300)

# Introduction

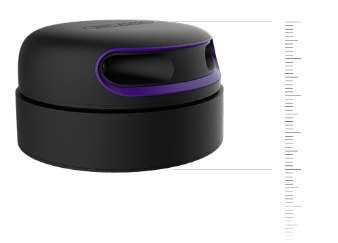
## -Sensors

One of the main things that are important with robot navigation is being able to sense the environment, just like humans we need eyes to navigate around, and sensors play a big part in the navigation of robot, some of the most important sensors robot use to avoid obstacles, detect landmarks…. Etc.

1-Tactile sensors: they are sensors that navigate the robot with touch, like grippers or hand sensing, these sensors help the robot with picking up objects or however tactile sensors are not used for environment navigation in our time as it is not as efficient at the other sensors.

[1] Robotic arm with “by coRo lab” with a “ultrasensitive multirotational tactile sensor” this sensor is able to measure presser, contact location and vibration.

2-Structure sensing sensor: these are laser sensors that find structure and can also find distances to objects, this is the sensor that our ros2 bot uses to map out with laser data and find objects, we use this sensor to find map and laser data in our code.



[2] RPLIDAR A3 Laser sensor which spins around at 360 degrees fast and map out environments with laser data, the ros2 bot uses this sensor.

3-Internal sensors: our last examples of sensors are internal sensors, these sensors are used to find the trajectory, velocity…etc of the robot, these can be used to find trajectory and velocity data in our code.

## - navigation and path planning

For the robot to plan its trajectory and navigation it uses many algorithms for both navigating around and planning its path, will be talking about some of those algorithms, why are these algorithms needed

### 1-Navigation

Odometry based algorithm: this algorithm uses x and y position on the map to navigate around with no knowledge of the surrounding environment.

Beacon based navigation: this algorithm uses beacons in order to navigate around, there are 3 types it uses to navigate, active beacons (RF signals, GPS, satellites. Etc), artificial land marks like QR codes or anything scannable, natural landmarks like furniture.

Map bases navigation: uses an environment that was pre mapped in order to find objects and navigate around safely

Kalman filtering / SLAM (EKF): Kalman algorithm is used to predict the next position of the robot by observing the environment and calculating its trajectory and velocity to reach that position, slam uses and extended Kalman filter to track features and landmarks for the robot to navigate, slam means simultaneous laicisation and mapping, which exactly like it sound is keeps track of data like laser data to navigate and predict the position of the next robot location , these 2 algorithms are used in our code for navigation.

### 2-Path planning

Teaching fixed path: this path planning methods uses landmarks in the environment to walk on a fixed path railway, road or even something as simple as coloured tape, this algorithm would always result in robot going at a fixed path that was given to it.

Graph search path planner: also known as Dijkstra’s algorithm, this algorithm uses map with nodes where the robot tries to find the lowest cost path through the nodes of the map.

A\* algorithm: this algorithm just like the graph search uses nodes on a path to traverse around however it deletes all paths but the lower cost path from the choices.

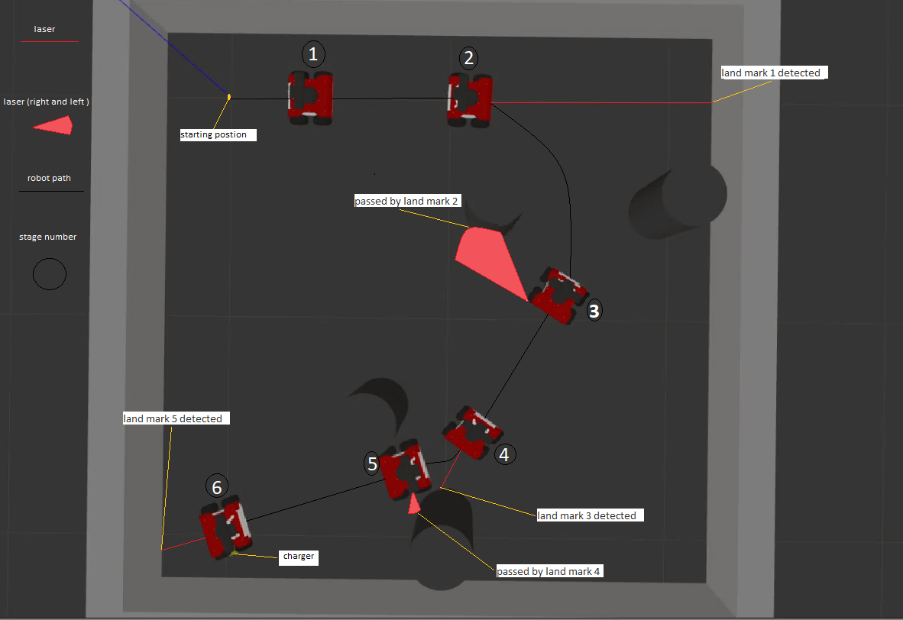
Composite-space path planner: this algorithm divides the map into grids where its either free space or obstacle and then uses the mapped-out data to find the shortest path instead of using nodes, this algorithm always improves as every time in find new free space and new obstacles it will update the faster route from point to point and it better at finding a direct path instead of node-to-node path where the robot has only the information of next position.

## -laser/trajectory implantation

In our code we have to control algorithms one for odometry and one for laser, both of these algorithms use fixed path planning with landmarks in order to navigate around, and in both control algorithms internal sensors and laser sensors work to get both odometry data, difference is navigating and which sensor it uses to navigate the environment.

### 1-Laser control

laser control uses the laser sensor in order to navigate around and uses slam algorithm to detect structures / landmarks, the navigation method It uses is slam which uses extended Kalman filter to detect land marks in a pre-observed environment , the robot path planning uses a fixed path and uses the laser landmarks around.



The robot doesn’t always change moving position when it detects a landmark, we can also set it to do that when it no longer sees a landmark, this shown in land mar 2 and 4.

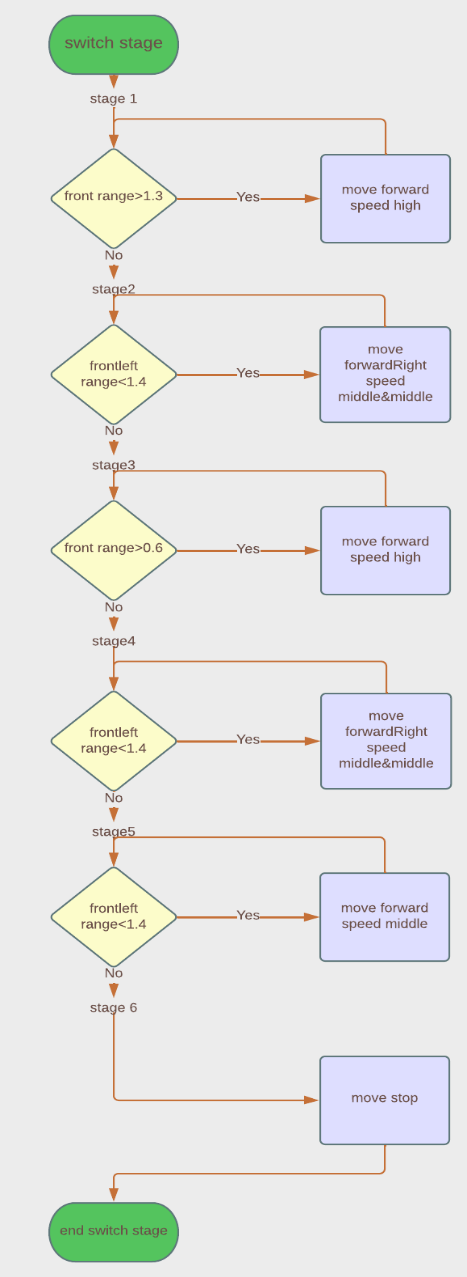
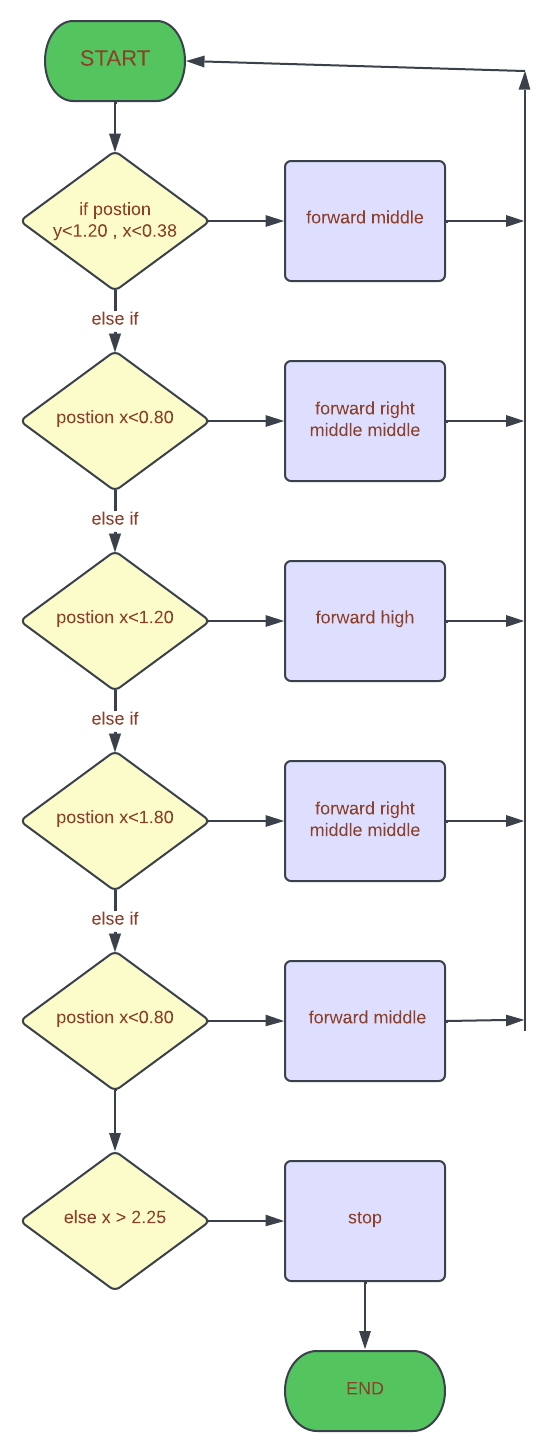
### 2-odometry control

On the other hand, odometry control uses the internal trajectory sensors, based on the position of the robot we use Kalman filtering algorithm to predict the next position of the robot, navigate the map before making the calculations for the robot to move, extended version does the same thing but instead uses laser data to detect land marks, so with odometry data we get a similar path to figure above but the path is predicted with the robot position.

# 

# Software design

## -Control flowcharts

Laser control flowchart trajectory control flowchart

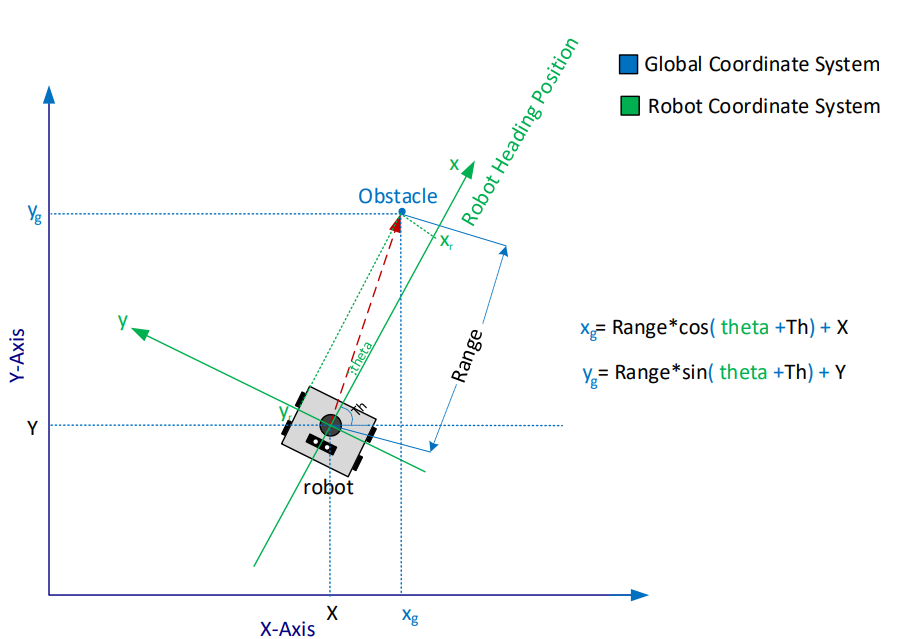
## -flowchart explanation

the flowcharts above have a similar control algorithm as shown in the flow chart where the robot’s movement is set into different stages that control the movement of the robot when reaching a landmark or In the laser control when seeing a landmark, the other important thing to mention here is that when laser is running we see that it is separated into stages, although technically the odometry is moving in stages as well, it doesn’t use the stages to control its movement rather it is on if statement with elseif and else where if the code is still running and its trajectory changes back into previous stage it will move as the previous stage algorithm asks, while wit laser once it has changed a stage disturbing the movement of the robot and putting it in a previous stage will affect that movement and it will move according to its current stage no matter the odometry, this is explained in the flow chart by showing the movement going back into its decision tap, while in odometry it goes back to the start of the if statement thus not having a stage based system like in laser.

# Code development

## -main code

The task at hand for our code is to implement, a path planning algorithm using the laser sensor on the ros2 bot, while getting data from laser sensor and internal sensors, to save the data we create the files for them first, then we got values to calculate our rotation using theta, we set our speeds for all movements and speeds, we create of our functions for moving, odometry data, velocity, laser call back……etc, then we start setting our functions, our movement control functions, and odomcallback and scancallback , we use those function to add data to the files we created before, these functions they also control the robot depending on which one is commented it controls with a set of stages and the way the robot moves from one stage to another is using trajectory landmarks or laser landmarks, stopper transform this functions transforms robot coordinates int global coordinates which uses the data gotten from call-back functions , shown in figure below.



Lastly we have the start moving function which opens the files we created before puts the laser data and odometry data into the files then closes them and after that we run the main function.

## -launch codes

1 tutorial\_rosobt.launch.py: this file is responsible for running the gazebo environment and spawning the rosbot2 at specific position and facing a specific angel, does that by getting the launch directions and generating

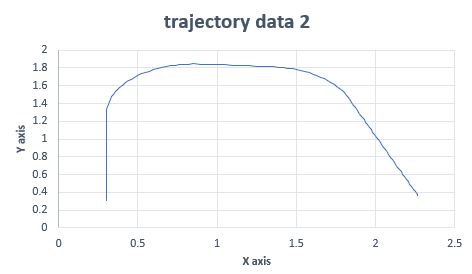
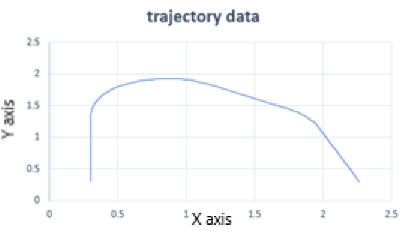
2 tutorial\_pkg\_node.launch.py: this file is responsible for running the code onto the robot

Ros2 launch tutorial\_pkg tutorial\_rosbot.launch.py command: runs the code together on the ros server in order to run them simultaneously where rosobot.launch runs the gazebo environment and spawns the robot at the desired location and pkg.launch run the code on the simulation.

# Experiential results

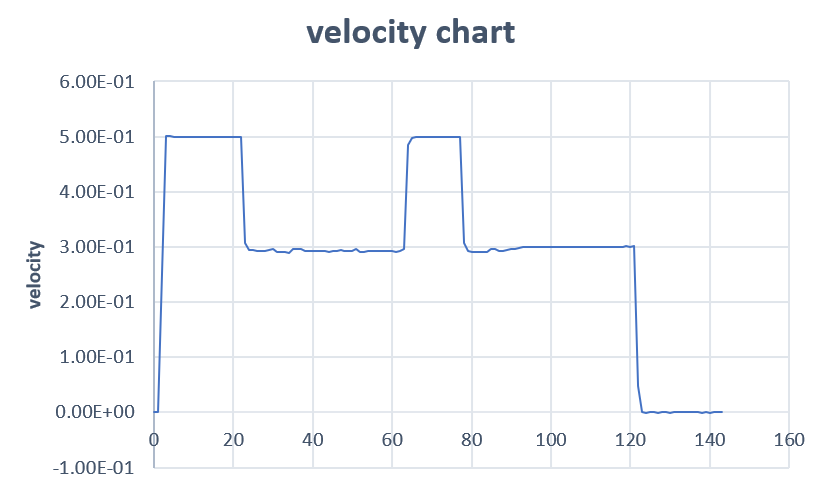
-trajectory data

1- Laser chart 2- odometry chart



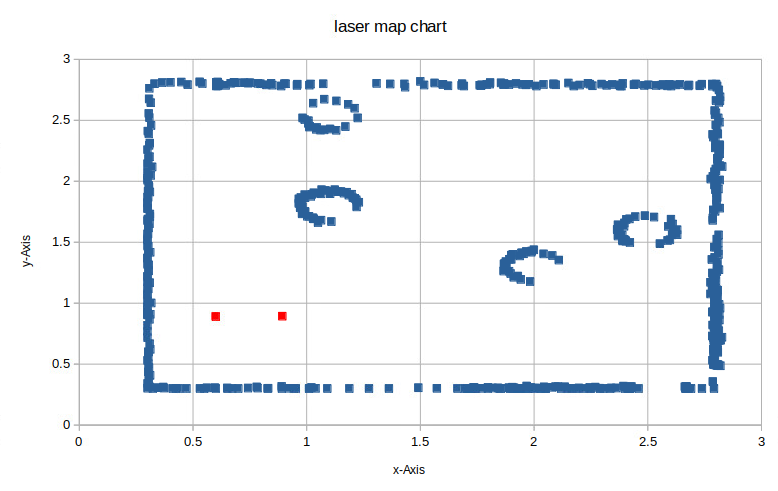
the 2 plots above are charts of robot trajectory, first one uses laser data to move and second uses odometry data to move, the as the code for odometry always reaches the charger, I am using it to compare the 2, although the path is different we can see that both have reached the location in fact the path taken in the first chart (laser) has taken a faster route to the charger, this proves our algorithms and landmarks to be efficient for this algorithm, I feel like this path is as accurate as we can get although with the ros2 environment the path can always change due to errors or delays with acceleration so we won’t always get the same path.

## -Velocity data



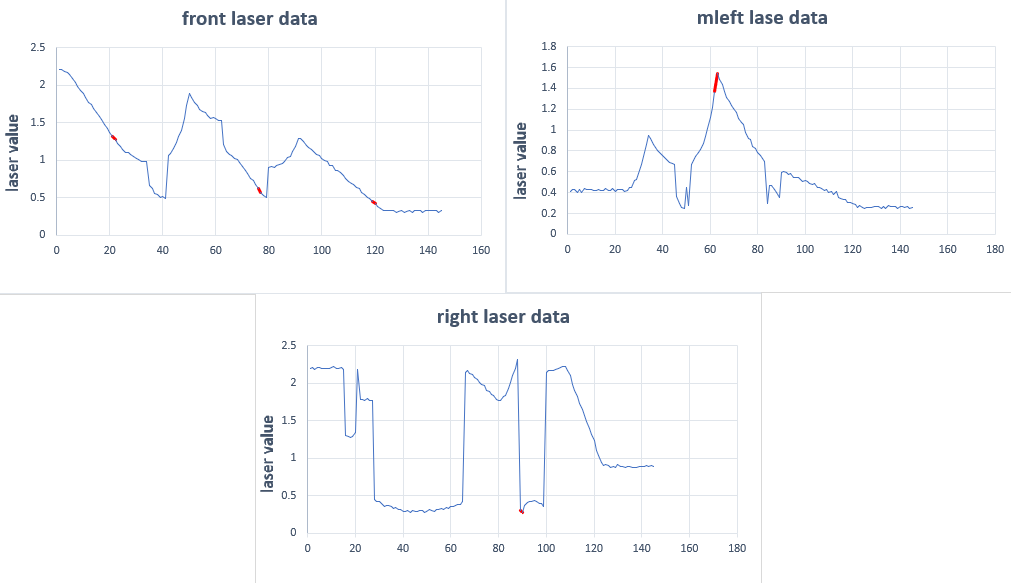
We can observer above the velocity of the robot, the speed I used for the robot was high, 2 X middle (for turning forward right) , high , 2 X middle and middle which seems to match with the chart, balancing between the velocity of the robot to make sure we don’t lose control in order to get to the charger as time efficient as possible, we can improve this by making the robot go faster at specific trajectories however this would require change in values in order to improve it.

## -Laser map sketch



Above we can see the map chart, they are pinpointed trajectories of where laser sensor hit an obstacle in order to draw the map. These trajectories however are inaccurate , the map itself is but the location is off, we can see that if we put the map and trajectory next to each other where starting position (red points) is not accurate to the graph, however if we want to improve the equation it just needs to be moved where the 0,0 point is where the left bottom corner on our graph is located thus moving every point the same as we did for that point.

## -Laser data



The data above shows us the lase range graph and the red points show where in the graph did laser cause the control to change its stage of movement, 3 times in front once in right and mleft.

# appendix

## -tutorial\_pkg\_node

//import our libraries.

#include <functional>

#include <memory>

#include <string>

#include "rclcpp/rclcpp.hpp"

#include "std\_msgs/msg/string.hpp"

#include "geometry\_msgs/msg/twist.hpp"

#include "geometry\_msgs/msg/pose.hpp"

#include "nav\_msgs/msg/odometry.hpp"

#include <fstream>

#include <time.h>

#include <iomanip>

#include "sensor\_msgs/msg/laser\_scan.hpp"

using namespace std::chrono\_literals;

using namespace std;

//include our files for odmetry data, velcoity data,data for each laser and---

//---laser mapping data.

ofstream odomTrajFile;

ofstream odomVelFile;

ofstream laserFile;

ofstream laserMapFile;

struct EulerAngles

{

double roll, pitch, yaw;

}; // yaw is what you want, i.e. Th

struct Quaternion

{

double w, x, y, z;

};

EulerAngles ToEulerAngles(Quaternion q)

{

// for calculating Th

EulerAngles angles;

// roll (x-axis rotation)

double sinr\_cosp = +2.0 \*(q.w \*q.x + q.y \*q.z);

double cosr\_cosp = +1.0 - 2.0 \*(q.x \*q.x + q.y \*q.y);

angles.roll = atan2(sinr\_cosp, cosr\_cosp);

// pitch (y-axis rotation)

double sinp = +2.0 \*(q.w \*q.y - q.z \*q.x);

if (fabs(sinp) >= 1)

angles.pitch = copysign(M\_PI / 2, sinp); //use 90 degrees if out of range

else

angles.pitch = asin(sinp);

// yaw (z-axis rotation)

double siny\_cosp = +2.0 \*(q.w \*q.z + q.x \*q.y);

double cosy\_cosp = +1.0 - 2.0 \*(q.y \*q.y + q.z \*q.z);

angles.yaw = atan2(siny\_cosp, cosy\_cosp);

return angles;

}

//set all our functions and other variables needed like robot velocity.....etc.

class Stopper: public rclcpp::Node

{

public:

/\*velocity control variables\*/

constexpr const static double FORWARD\_SPEED\_LOW = 0.1;

constexpr const static double FORWARD\_SPEED\_MIDDLE = 0.3;

constexpr const static double FORWARD\_SPEED\_HIGH = 0.5;

constexpr const static double FORWARD\_SPEED\_STOP = 0;

constexpr const static double TURN\_LEFT\_SPEED\_LOW = 0.3;

constexpr const static double TURN\_LEFT\_SPEED\_MIDDLE = 0.6;

constexpr const static double TURN\_LEFT\_SPEED\_HIGH = 1.0;

constexpr const static double TURN\_RIGHT\_SPEED\_LOW = -0.3;

constexpr const static double TURN\_RIGHT\_SPEED\_MIDDLE = -0.6;

constexpr const static double TURN\_RIGHT\_SPEED\_HIGH = -1.0;

/\*class constructor \*/

Stopper(): Node("Stopper"), count\_(0)

{

publisher\_ = this->create\_publisher<geometry\_msgs::msg::Twist > ("cmd\_vel", 10);

odomSub\_ = this->create\_subscription<nav\_msgs::msg::Odometry > ("odom", 10,

std::bind(&Stopper::odomCallback, this, std::placeholders::\_1));

laserScan\_ = this->create\_subscription<sensor\_msgs::msg::LaserScan > ("scan", 10,

std::bind(&Stopper::scanCallback, this, std::placeholders::\_1));

};

/\*stopper::functions \*/

void startMoving();

void moveStop();

void moveForward(double forwardSpeed);

void moveRight(double turn\_right\_speed);

void moveForwardRight(double forwardSpeed, double turn\_right\_speed);

void odomCallback(const nav\_msgs::msg::Odometry::SharedPtr odomMsg);

double PositionX = 0.3, PositionY = 0.3, homeX = 0.3, homeY = 0.3;

double robVelocity;

int numberOfCycle = 0;

void scanCallback(const sensor\_msgs::msg::LaserScan::SharedPtr scan);

double frontRange, mleftRange, leftRange, rightRange, mrightRange;

int laser\_index = 0; // index the laser scan data

Quaternion robotQuat;

EulerAngles robotAngles;

double robotHeadAngle;

double leftAngle = M\_PI / 2, mleftAngle = M\_PI / 4, frontAngle = 0;

double mrightAngle = -M\_PI / 4, rightAngle = -M\_PI / 2;

void transformMapPoint(ofstream &fp, double laserRange, double laserTh,

double robotTh, double robotX, double robotY);

int stage = 1;

private:

// Publisher to the robot's velocity command topic.

rclcpp::Publisher<geometry\_msgs::msg::Twist>::SharedPtr publisher\_;

rclcpp::TimerBase::SharedPtr timer\_;

size\_t count\_;

//Subscriber to robot’s odometry topic.

rclcpp::Subscription<nav\_msgs::msg::Odometry>::SharedPtr odomSub\_;

rclcpp::Subscription<sensor\_msgs::msg::LaserScan>::SharedPtr laserScan\_;

};

//function to move forward.

void Stopper::moveForward(double forwardSpeed)

{

auto msg = geometry\_msgs::msg::Twist();

msg.linear.x = forwardSpeed;

publisher\_->publish(msg);

}

//function to stop.

void Stopper::moveStop()

{

auto msg = geometry\_msgs::msg::Twist();

msg.linear.x = FORWARD\_SPEED\_STOP;

publisher\_->publish(msg);

}

//function to turn to the right.

void Stopper::moveRight(double turn\_right\_speed)

{

auto msg = geometry\_msgs::msg::Twist();

msg.angular.z = turn\_right\_speed;

publisher\_->publish(msg);

}

//function to move forward while turning to the right.

void Stopper::moveForwardRight(double forwardSpeed, double turn\_right\_speed)

{

auto msg = geometry\_msgs::msg::Twist();

msg.linear.x = forwardSpeed;

msg.angular.z = turn\_right\_speed;

publisher\_->publish(msg);

}

//function for odometry

void Stopper::odomCallback(const nav\_msgs::msg::Odometry::SharedPtr odomMsg)

{

PositionX = odomMsg->pose.pose.position.x + homeX;

PositionY = odomMsg->pose.pose.position.y + homeY;

RCLCPP\_INFO(this->get\_logger(), "RobotPostion: %.2f, %.2f", PositionX, PositionY);

RCLCPP\_INFO(this->get\_logger(), "Robot stage: %d ", stage);

//code for moving the robot usuing odmetry data

//if commented, uncomment to run robot movment with odometry and comment laser movment

/\*

if (PositionY < 1.20 && PositionX < 0.38)

{

stage = 1;

moveForward(FORWARD\_SPEED\_MIDDLE);

}

else if (PositionX < 0.80)

{

stage = 2;

moveForwardRight(FORWARD\_SPEED\_MIDDLE, TURN\_RIGHT\_SPEED\_MIDDLE);

}

else if (PositionX < 1.20)

{

stage = 3;

moveForward(FORWARD\_SPEED\_HIGH);

}

else if (PositionX < 1.80)

{

stage = 4;

moveForwardRight(FORWARD\_SPEED\_MIDDLE, TURN\_RIGHT\_SPEED\_MIDDLE);

}

else if (PositionX < 2.25)

{

stage = 5;

moveForward(FORWARD\_SPEED\_MIDDLE);

}

else

{

stage = 6;

moveStop();

}

\*/

robVelocity = odomMsg->twist.twist.linear.x; //get velocity

odomVelFile << numberOfCycle++ << " " << robVelocity << endl; //insert velocity into file

odomTrajFile << PositionX << " " << PositionY << endl; // insert trajectory into file

robotQuat.x = odomMsg->pose.pose.orientation.x;

robotQuat.y = odomMsg->pose.pose.orientation.y;

robotQuat.z = odomMsg->pose.pose.orientation.z;

robotQuat.w = odomMsg->pose.pose.orientation.w;

robotAngles = ToEulerAngles(robotQuat);

robotHeadAngle = robotAngles.yaw;

}

//function for laser

void Stopper::scanCallback(const sensor\_msgs::msg::LaserScan::SharedPtr scan)

{

leftRange = scan->ranges[300]; // get a range reading at the left angle

mleftRange = scan->ranges[250]; // get a range reading at the front-left angle

frontRange = scan->ranges[200]; // get a range reading at the front angle

mrightRange = scan->ranges[150]; // get a range reading at the front-right angle

rightRange = scan->ranges[100]; // get the range reading at the right angle

laserFile << leftRange << "," << mleftRange << "," << frontRange << "," <<

mrightRange << "," << rightRange << "," << laser\_index++ << endl; //insert laser ranges to fie

transformMapPoint(laserMapFile, frontRange, frontAngle, robotHeadAngle, PositionX, PositionY); //insert maplaser file front

transformMapPoint(laserMapFile, mleftRange, mleftAngle, robotHeadAngle,

PositionX, PositionY); //insert maplaser file mleft

transformMapPoint(laserMapFile, leftRange, leftAngle, robotHeadAngle,

PositionX, PositionY); //insert maplaser file left

transformMapPoint(laserMapFile, rightRange, rightAngle, robotHeadAngle,

PositionX, PositionY); //insert maplaser file right

transformMapPoint(laserMapFile, mrightRange, mrightAngle, robotHeadAngle,

PositionX, PositionY); //insert maplaser file mright

//function for controloing robot usuing laser (case3-5 = my code)

//if commented, uncomment to run robot movment with laser and comment odometry movment

switch (stage)

{

case 1:

if (frontRange > 1.3)

moveForward(FORWARD\_SPEED\_HIGH);

else stage = 2;

break;

case 2:

if (mleftRange < 1.4)

moveForwardRight(FORWARD\_SPEED\_MIDDLE, TURN\_RIGHT\_SPEED\_MIDDLE);

else stage = 3;

break;

case 3:

if (frontRange > 0.6)

moveForward(FORWARD\_SPEED\_HIGH);

else stage = 4;

break;

case 4:

if (rightRange > 0.3)

moveForwardRight(FORWARD\_SPEED\_MIDDLE, TURN\_RIGHT\_SPEED\_MIDDLE);

else stage = 5;

break;

case 5:

if (frontRange > 0.4)

moveForward(FORWARD\_SPEED\_MIDDLE);

else stage = 6;

break;

case 6:

moveStop();

break;

}

}

//transforms robot data into gloabal data

void Stopper::transformMapPoint(ofstream &fp, double laserRange, double laserTh,

double robotTh, double robotX, double robotY)

{

double transX, transY;

transX = laserRange\* cos(robotTh + laserTh) + robotX;

transY = laserRange\* sin(robotTh + laserTh) + robotY;

if (transX < 0) transX = homeX;

else transX += homeX;

if (transY < 0) transY = homeX;

else transY += homeY;

fp << transX << ", " << transY << endl;

}

//opens files and saves the data into the files while robot is moving

void Stopper::startMoving()

{

odomTrajFile.open("/ufs/servg00/users/hh18564/M-Drive/ros\_workspace/src/tutorial\_pkg/data/odomTrajData.csv", ios::trunc); //open TrajData

odomVelFile.open("/ufs/servg00/users/hh18564/M-Drive/ros\_workspace/src/tutorial\_pkg/data/odomVelData.csv", ios::trunc); //open VelData

laserFile.open("/ufs/servg00/users/hh18564/M-Drive/ros\_workspace/src/tutorial\_pkg/data/laserData.csv", ios::trunc); //open LaserData

laserMapFile.open("/ufs/servg00/users/hh18564/M-Drive/ros\_workspace/src/tutorial\_pkg/data/laserMapData.csv", ios::trunc); //open LaseMapData

RCLCPP\_INFO(this->get\_logger(), "Start moving");

rclcpp::WallRate loop\_rate(10);

while (rclcpp::ok())

{

auto node = std::make\_shared<Stopper> ();

rclcpp::spin(node); // update

loop\_rate.sleep(); // wait delta time

}

odomTrajFile.close(); //close TrajData

odomVelFile.close(); //close VelData

laserFile.close(); //close LaserData

laserMapFile.close(); //close LaseMapData

}

//main fucntion

int main(int argc, char \*argv[])

{

rclcpp::init(argc, argv);

Stopper stopper;

stopper.startMoving();

return 0;

}

# Citation

[1]"Why Tactile Intelligence Is the Future of Robotic Grasping", *IEEE Spectrum*, 2022 [Accessed: 20/3/2022].

[2] T. Huang, "RPLIDAR-A3 Laser Range Scanner\_ Robot Laser Range Scanner|SLAMTEC", *Slamtec.com*, 2022. [Accessed: 20/3/2022].

[3]"ROS: Home", *Ros.org*, 2022. [Online]. Available: https://www.ros.org/. [Accessed: 20/3/2022].