Fahad hussain

Mobile robotics

Assignment 2

Contents

[Introduction: 2](#_Toc99013953)

[Code development 3](#_Toc99013954)

[Flowchart 4](#_Toc99013955)

# Introduction:

Mobile robotics is an industry that has grown quite a lot in the last few years. Robots are used in several different industries for varying tasks and will carry on being an integral part of the present and future of technology

There are several different uses of mobile robots but for this experiment the primary focus will be to place and locate a mobile robot on a given environment map and allow the robot to safely collect data of the robot’s surrounding environment and let it safely reach its desired destination without colliding with any objects. There are several real-life applications of robotics that already effect our day-to-day life, one being a self-cleaning robot that travels throughout a house and clean and avoids any obstacles that may come in its path, another example could be Surveillance/exploring a region that humans find it naturally difficult to be in e.g., Underwater/Radiated environments.

In mobile robotics path planning and navigation are crucial elements when managing and controlling robots, it allows the user to guide and assist any robotic machine in reaching a target destination. There are Four requirements that a user must consider when planning a mobile robotic path navigation:

-Task planning

-Map building

-Path planning

-Robot Guidance

Task planning requires the user to carefully plan out the use of the robot, for instance a user can design the Start position of the robot and the specified target destination, the controller can also consider the fact that there are several different types of robotics, each one will have path planning methods that are more suitable to one than the other.

If a robot needs to traverse through a give environment it will need a map to do this, this is due to the fact that the mobile robot will need to know its location and the objects that are surrounding its location and will then accordingly plan a path to its desired destination, in some instances there will be no maps that are available for the robot to use, In that case, the robot will be required to create its own dynamic map.

The role of path planning method is to allow the robot to find the correct path in an environment so it can safely reach its desired destination, there are usually four different types of path planners that are commonly used for robotics,

Teaching and fixed paths method-

This method uses Landmarks to guide the robot, for instance all over the map there will variables that are marked as landmarks, the robot can then be configured to follow the path according to a planned path, for example the robot can go to the landmarks P1-P2-P3.

Another method that can be implemented is the Graph search path planner, it essentially makes the map into a graph and splits all the landmarks into nodes, each landmark/object can be represented through a variable, Since it is a graph based algorithm there will be costs assigned to a landmark, Since it is a graph based algorithm the use of famous path finding algorithms such as A\* search and Dijkstra's shortest path  can then be used on the graph to navigate the robot.

In this assignment a Laser sensor approach was used to navigate and control the Mobile robot, The purpose of these lasers is to help the robot collect data about the environment, so it can safely get to its desired destination. Another feature of a scanner mobile robot is that it can also be used to detect any objects/obstacles that may be in the environment, this is beneficial for any person manipulating mobile robots as it is a way to collect data without actually physically being there, This can be setup in a large scale operation where several different mobile robots can be assigned to different locations where they can gather information about the environment they are in and then use the same data for future uses. For this task a SLAM (Simultaneous localisation and mapping) Based navigation method was used. The code was compiled in C++ and simulated in gazebo.

A red and black sewing machine

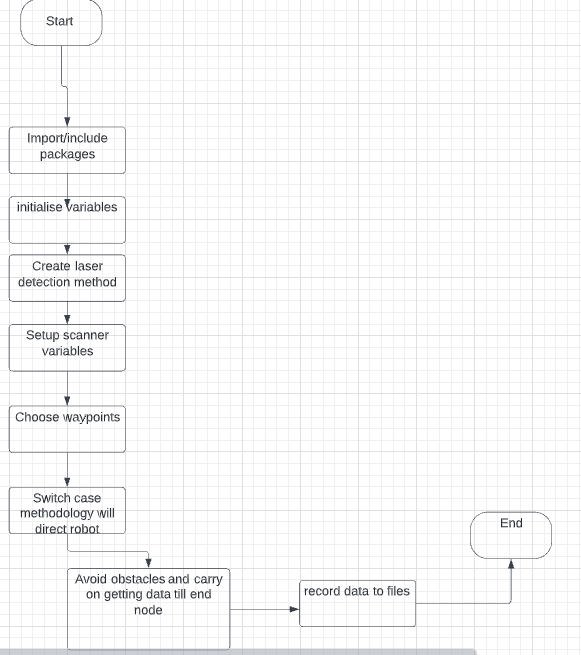
Description automatically generated with low confidence

Figure 2: ROS Model used in simulation

# Code development

At first the ROS package had to be installed into the directory, After the package had been built There was a need to create a tutorial node which was used in our experiment, The node was a C++ file which contained predefined variables and functions that were used to manipulate the robot, At first a odometry based algorithm was used to manoeuvre the robot, there was predefined pillars that were configured so that the robot will go to a specific landmark before the others, but for the experiment the use of a laser based approach was more appropriate. After the node was compiled a simulation of a robot moving horizontally to the end of the graph appeared. The robot still needed to be configured so that it reaches its desired charger destination without crashing into a pillar, this was done using the scanCallback class, this method helps the robot scan in a specific range reading at a given angle, this was used to gather the necessary information to input the necessary variables into the scan call back method by using a switch case method.

# Flowchart

  
Figure 1: Flowchart  
  
As you can see in figure 1 there is quite a simple flow of information in this diagram, the program starts by importing all the necessary packages, then there is a need to initialise the robots starting position and some basic variables that will allow us to control the robot, The environment map must also be configured, Once the robot and the map have correctly loaded in there is a need to collect the necessary coordination data that will be inputted into a csv file, this information can then help the user map out the correct path that the robot can take, This part of the assignment took a lot trial and error as finding out the most appropriate values that will take your robot to its destination the quickest.  
  
  
Appendix

This is tutorial\_pkg\_node.cpp file, it is assuming that you have made the necessary changes for the CMakelist.txt and package.xml files and any other necessary files,

#include <chrono>

#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;

//ofstream odomTrajFile;

//ofstream odomVelFile;

ofstream laserFile; // Declare a file object for recording your laser data

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;

}

class Stopper : public rclcpp::Node {

public:

double laser\_landmark1 = 1.3, laser\_landmark2 = 1.4;

double laser\_landmark3 = 0.7, laser\_landmark4 = 0.6, laser\_landmark5 = 0.4;

int stage = 1;

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);

double robVelocity;

int numberOfCycle = 0;

/\* 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") {

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

                                                                         std::bind(&Stopper::scanCallback, this,

                                                                                   std::placeholders::\_1));

     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));

};

/\* moving function \*/

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 odom\_landmark1 = 1.20, odom\_landmark1a = 0.38, odom\_landmark2 = 0.80;

double odom\_landmark3 = 1.20, odom\_landmark4 = 1.80, odom\_landmark5 = 2.25;

//int stage=0;

//ofstream odomTrajFile;

private:

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

// 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\_;

//

};

void Stopper::moveForward(double forwardSpeed) {

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

msg.linear.x = forwardSpeed;

publisher\_->publish(msg);

}

void Stopper::moveStop() {

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

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

publisher\_->publish(msg);

}

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

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

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

publisher\_->publish(msg);

}

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);

}

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);

//odomTrajFile<< PositionX <<" "<< PositionY<<endl;

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;

   /\* if (PositionY < odom\_landmark1 && PositionX < odom\_landmark1a) {

     stage = 1;

     moveForward(FORWARD\_SPEED\_MIDDLE);

} else if (PositionX < odom\_landmark2) {

     stage = 2;

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

} else if (PositionX < odom\_landmark3) {

     stage = 3;

     moveForward(FORWARD\_SPEED\_HIGH);

} else if (PositionX < odom\_landmark4) {

     stage = 4;

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

} else if (PositionX < odom\_landmark5) {

     stage = 5;

     moveForward(FORWARD\_SPEED\_MIDDLE);

} else {

     stage = 6;

     moveStop();

}\*/

//odomTrajFile<< PositionX <<" "<< PositionY<<endl;

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

//odomVelFile << numberOfCycle++ << " " << robVelocity << endl;

}

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;

switch(stage){

case 1:

     if (frontRange > laser\_landmark1)

         moveForward(FORWARD\_SPEED\_MIDDLE);

     else stage = 2;

     break;

     case 2:

         if (mleftRange < laser\_landmark2)

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

         else stage = 3;

         break;

  case 3:

         if (mrightRange < laser\_landmark3)

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

         else stage = 4;

         break;

    case 4:

         if (rightRange < laser\_landmark4)

             moveForward(FORWARD\_SPEED\_HIGH);

         else stage = 4;

         break;

  case 5:

         if (mleftRange < laser\_landmark5)

             moveForward(FORWARD\_SPEED\_MIDDLE);

         else stage = 5;

         break;

  case 6:

            moveStop();

            break;

}

transformMapPoint(laserMapFile, frontRange, frontAngle, robotHeadAngle, PositionX, PositionY);

transformMapPoint(laserMapFile, mleftRange, mleftAngle, robotHeadAngle, PositionX, PositionY);

transformMapPoint(laserMapFile, leftRange, leftAngle, robotHeadAngle, PositionX, PositionY);

transformMapPoint(laserMapFile, rightRange, rightAngle, robotHeadAngle, PositionX, PositionY);

transformMapPoint(laserMapFile, mrightRange, mrightAngle, robotHeadAngle, PositionX, PositionY);

}

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;

}

void Stopper::startMoving() {

laserFile.open("/ufs/servh01/users/fh17602/M-Drive/ros\_workspace/src/tutorial\_pkg/laserData.csv",ios::trunc); //note you should modify hhu to your userna

laserMapFile.open("/ufs/servh01/users/fh17602/M-Drive/ros\_workspace/src/tutorial\_pkg/laserMapData.csv",ios::trunc); //note you should modify hhu to your userna

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

rclcpp::WallRate loop\_rate(10);

while (rclcpp::ok()) { // keep spinning loop until you press Ctrl+C

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

     rclcpp::spin(node); // update

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

}

  laserFile.close();

  laserMapFile.close();

}

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

rclcpp::init(argc, argv);

Stopper stopper;

stopper.startMoving();

return 0;

}