

CE315 Mobile Robotics

Assignment 2

Jonathan Devaux

[1603905]

Table of Contents

Introduction	1
Path planners	1
Laser-based control strategy	1
Navigation methods	2
Software design	3
Flowchart (key)	3
Flowchart	4
Code development	5
Experimental results	6
Appendix	8

Introduction

Below path planners and navigation methods will be discussed as well as the laser-based control strategy used in the implementation

Path planners

There are a number of different types of path planners such as the teaching and fixed path planner, graph search path planner, free space path planner and the composite space path-planner. Additionally, there are requirements that are necessary for effective path planners such as finding a path for the robot to travel in the mapped environment, ensure that the path is not in the way of landmarks/objects to reduce the chances of collision as well as find solutions/alternatives in the case of errors in the execution stage and find the best path suitable to reduce overall travel time.

Teaching and Fixed path planner

It is possible for the user to teach a mobile robot to follow a specific path that are marked by wires/landmarks or even objects or barcodes if they are defined in the environment map. The control computer stores the planned paths and this controls the robot going from a to b or even to another place. However, through the use of operational research techniques path planned can be changed so that the best path is chosen.

Graph Search path planner

In the graph search path planner, the graph features nodes, and edges. Edges represent the differences/similarities between two nodes. Additionally, graphs can both either be undirected or directed.

Composite space path planner

In the composite space path planner, the space is divided into regular grids. This is to represent the division of space in the grid for either obstacles or free space in a cell. The aim of this is to find the shortest path between the location from start position and the location of the finish position.

Laser-based control strategy

In the laser based control strategy this will be controlled by the robot's sensors which data for local landmarks will be setup for the robot to determine position as well as motion.

Navigation methods

Navigation methods is used to guide the mobile from point a to b in order to reach the set destination without crashing into landmarks, features or any object and ensures smoothness of the course. The four popular navigation methods are odometry based navigation Odometry based navigation, Beacon based navigation, Map based navigation and SLAM based navigation.

Odometry based navigation

Odometry is used in navigation and can accumulate errors. The main errors are systematic errors and non-systematic errors. An example of a systematic error is the misalignment of the wheels and gears. In non-systematic errors an example will travelling over unexpected objects such as glass, rocks (in a rocky environment) and uneven ground.

Beacon based navigation

Beacons are used in navigation which come in different forms such as active beacons that can emit energy signals for example global positioning system(GPS) and radio frequency(RF) signals. Another form is the use of artificial landmarks which these are objects ranging in different sizes from small barcodes to name plates. One more form is the use of natural landmarks such as boxes, table, walls even a desk and many other physical objects can be used a natural landmark for the robot to detect.

Map based navigation

In map based navigation, a map information is stored into the robot or composed from data previously by sensors which this will be referred to as the local environment data. Then the data is compared to the map information already stored and if the data matches the robot will be able to detect its position with the use of its sensors. However, there are situations when the robot sensor data does not match the map information already stored which the robot will store sensor data for later use in the map information for any position calculation.

The map building has three stages which includes extraction of features from raw sensor data, then the robot will pull the data of various types from it sensors. Finally, the environment map will be generated with features included.

SLAM based navigation

Simultaneous localisation and mapping(SLAM) is used to for localisation and mapping in which odometry and other sensor data is pulled together to update the environment map. This reduces the uncertainty factor through the use of the extended Kalman filter(EKF) to track landmarks/features and objects in the local environment to locate the robot position.

Software design

Flowchart (key)

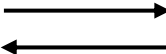
Key:

Start/Finish 

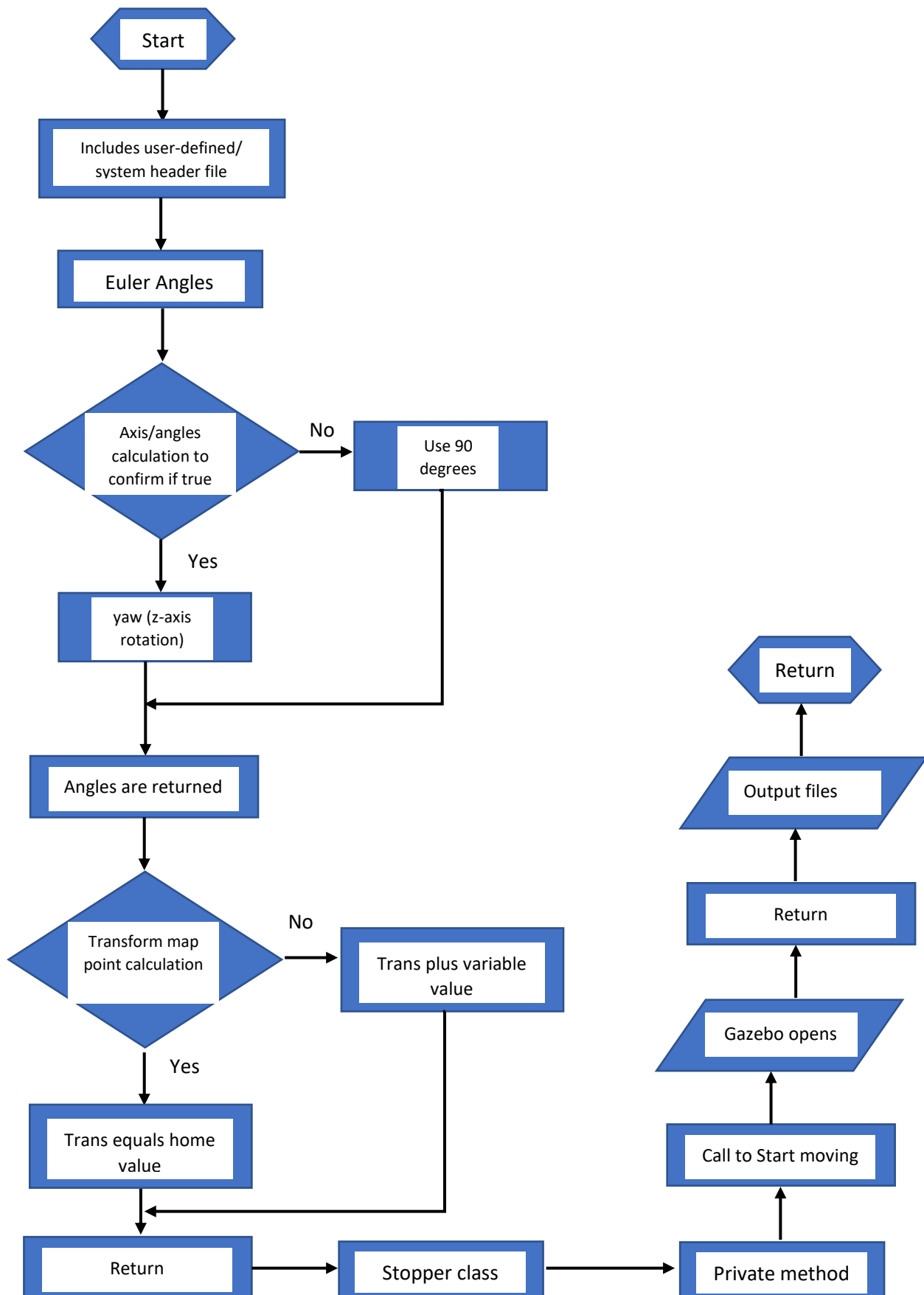
Computation/
processing 

Comparison 

Input/Output 

Arrows/
Connectors 

Flowchart



Code development

Class stopper is a callable function as the objects inside can be called in other parts of the code. This is due to making the class public.

```
class Stopper {
public:
// Tunable parameters
constexpr const static double FORWARD_SPEED_LOW = 0.1;
constexpr const static double FORWARD_SPEED_HIGH = 0.2;
constexpr const static double FORWARD_SPEED_SHIGH = 0.4;
constexpr const static double FORWARD_SPEED_STOP = 0;
constexpr const static double TURN_LEFT_SPEED_HIGH = 1.0;
constexpr const static double TURN_LEFT_SPEED_LOW = 0.3;
constexpr const static double TURN_RIGHT_SPEED_HIGH = -2.4;
constexpr const static double TURN_RIGHT_SPEED_LOW = -0.3;
constexpr const static double TURN_RIGHT_SPEED_MIDDLE = -0.6;
```

Below is the Stopper and the following void codes, this initialises the various codes given relating to movement, ranges and position that affect the odometry and EulerAngles.

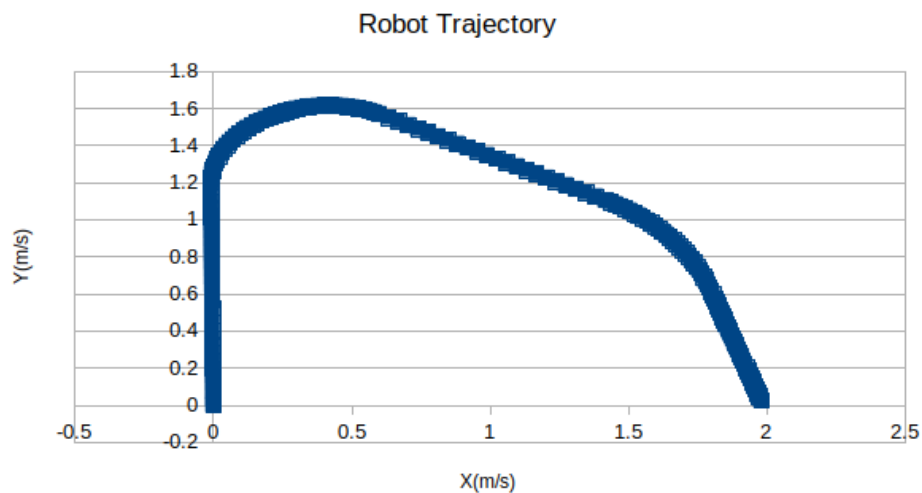
```
Stopper();
void startMoving();
void moveForward(double forwardSpeed);
void moveStop();
void moveRight(double turn_right_speed = TURN_RIGHT_SPEED_HIGH);
void moveForwardRight(double forwardSpeed, double turn_right_speed);
void odomCallback(const nav_msgs::Odometry::ConstPtr& odomMsg);
int stage;
double PositionX, PositionY;
double robVelocity;
double startTime; // start time
Quaternion robotQuat;
EulerAngles robotAngles;
double robotHeadAngle;
void transformMapPoint(ofstream& fp, double laserRange, double
laserTh, double robotTh, double robotX, double robotY);
void scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan);
double frontRange, mleftRange, leftRange, rightRange, mrightRange;
double backRange, leftbackRange, rightbackRange;
double landmark1 = 1.35, landmark2 = 1.5, landmark3 = 2.25, landmark4
= 1, landmark5 = 1.3;
```

The int main function below initialises the new ROS node named stopper whilst it also calls the node stopper to create the new stopper objects as well as ensures the starting of movement in which then returns everything.

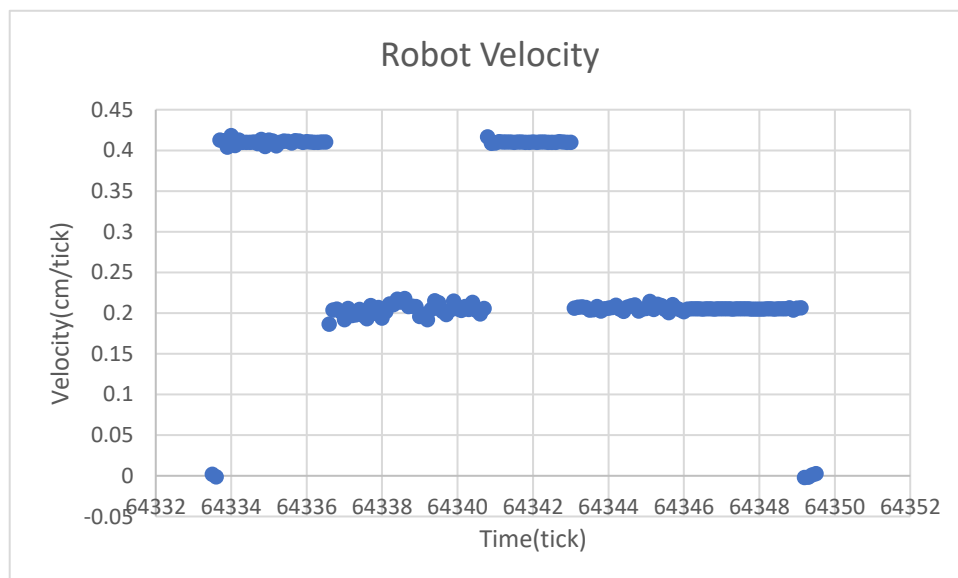
```
int main(int argc, char **argv) {
ros::init(argc, argv, "stopper"); // Initiate new ROS node named
"stopper"
Stopper stopper; // Create new stopper object
stopper.startMoving(); // Start the movement
return 0;
}
```

Experimental results

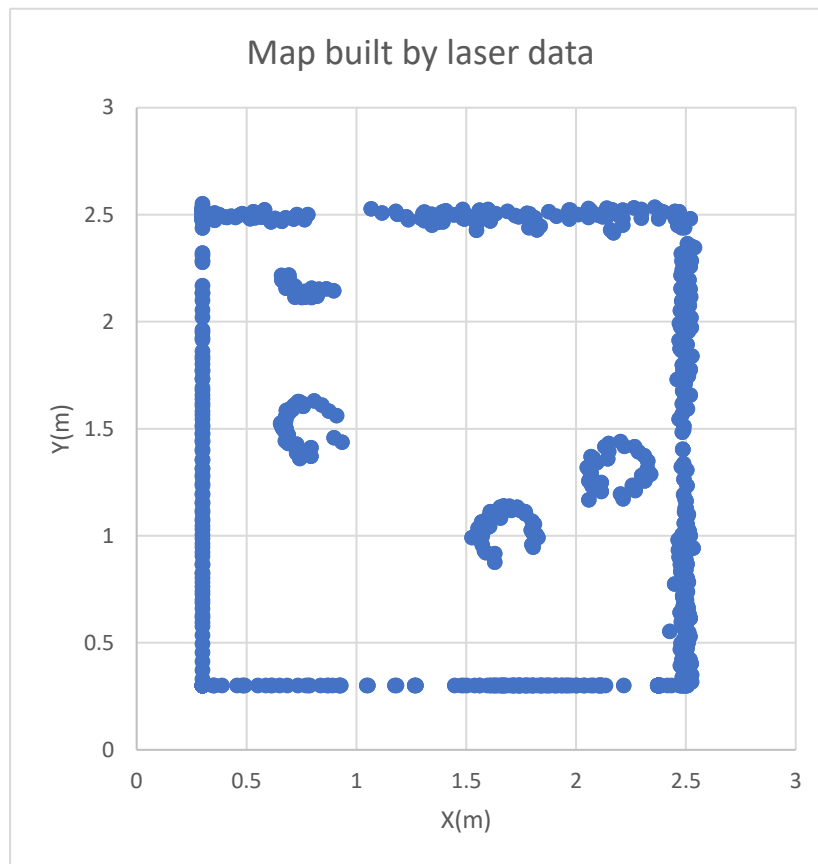
Here are the experimental results, below is the robot trajectory. As we can in the y axis the robot trajectory increases till it roughly reaches 1.65m/s and the x axis reach 0.5m/s and by then the robot trajectory starts to decrease over time as both the x axis and the y axis decrease once the x axis reaches 2m/s with y axis decreasing to 0m/s this reflects the movement of the robot as this shows that robot has reached the charging point.



Below is the robot velocity this here shows the velocity in the y axis and ranging in cm/s whilst the x axis is the time measured in seconds to record the time against velocity(cm/tick) taken to reach each checkpoint



Below is the map built by the laser as we can see it features details from the environment such as the obstacles. However, some parts are missing such as the starting point and the charging point.



Appendix

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "nav_msgs/Odometry.h"
#include <fstream>
#include <time.h>
#include <iomanip>
#include "sensor_msgs/LaserScan.h"

using namespace std;

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)
{
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;
}

ofstream odomVelFile;
class Stopper {
public:
// Tunable parameters and speed
constexpr const static double FORWARD_SPEED_LOW = 0.1;
constexpr const static double FORWARD_SPEED_HIGH = 0.2;
constexpr const static double FORWARD_SPEED_SHIGH= 0.4;
constexpr const static double FORWARD_SPEED_STOP = 0;
constexpr const static double TURN_LEFT_SPEED_HIGH = 1.0;
constexpr const static double TURN_LEFT_SPEED_LOW = 0.3;
constexpr const static double TURN_RIGHT_SPEED_HIGH = -2.4;
constexpr const static double TURN_RIGHT_SPEED_LOW = -0.3;
constexpr const static double TURN_RIGHT_SPEED_MIDDLE = -0.6;
Stopper();
void startMoving();
void moveForward(double forwardSpeed);
void moveStop();
void moveRight(double turn_right_speed = TURN_RIGHT_SPEED_HIGH);
void moveForwardRight(double forwardSpeed, double turn_right_speed);
void odomCallback(const nav_msgs::Odometry::ConstPtr& odomMsg);
int stage;
double PositionX, PositionY;
double robVelocity;
```

```

double startTime; // start time
Quaternion robotQuat;
EulerAngles robotAngles;
double robotHeadAngle;
void transformMapPoint(ofstream& fp, double laserRange, double laserTh,
double robotTh, double robotX, double robotY);
void scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan);
double frontRange, mleftRange, leftRange, rightRange, mrightRange;
double backRange, leftbackRange, rightbackRange;
double landmark1 = 1.1, landmark2 = 1.5, landmark3 = 1.6, landmark4 = 0.8,
landmark5 = 2;
int vel_timer = 0;

private:
ros::NodeHandle node;
ros::Publisher commandPub; // Publisher to the robot's velocity command
topic
ros::Subscriber odomSub; //Subscriber to robot's Odometry topic
ros::Subscriber laserSub; //Subscriber to robot's lasertopic
};

Stopper::Stopper(){
//Advertise a new publisher for the simulated robot's velocity command
topic at 10Hz
commandPub = node.advertise<geometry_msgs::Twist>("cmd_vel", 10);
odomSub = node.subscribe("odom", 20, &Stopper::odomCallback, this);
laserSub = node.subscribe("scan", 1, &Stopper::scanCallback, this);
}

//send a velocity command
void Stopper::moveForward(double forwardSpeed){
geometry_msgs::Twist msg; //The default constructor to set all commands to 0
msg.linear.x = forwardSpeed; //Drive forward at a given speed along the x-
axis.
commandPub.publish(msg);
}

void Stopper::moveStop(){
geometry_msgs::Twist msg;
msg.linear.x = FORWARD_SPEED_STOP;
commandPub.publish(msg);
}

void Stopper::moveRight(double turn_right_speed){
geometry_msgs::Twist msg;
msg.angular.z = turn_right_speed;
commandPub.publish(msg);
}

void Stopper::moveForwardRight(double forwardSpeed, double
turn_right_speed){
//move forward and right at the same time
geometry_msgs::Twist msg;
msg.linear.x = forwardSpeed;
msg.angular.z = turn_right_speed;
commandPub.publish(msg);
}

void Stopper::transformMapPoint(ofstream& fp, double laserRange, double
laserTh, double robotTh, double robotX, double robotY)

```

```

{
double transX, transY, homeX=0.3, homeY=0.3;
transX = laserRange * cos(robotTh + laserTh) + robotX;
transY = laserRange * sin(robotTh + laserTh) + robotY;
if (transX < 0) transX = homeX; else transX += homeX;
if (transY < 0) transY = homeY; else transY += homeY;
fp << transX << " " << transY << endl;
}

void Stopper::odomCallback(const nav_msgs::Odometry::ConstPtr& odomMsg) {
PositionX = odomMsg->pose.pose.position.x;
PositionY = odomMsg->pose.pose.position.y;
robVelocity = odomMsg->twist.twist.linear.x;
double currentSeconds = ros::Time::now().toSec();
double elapsedTime = currentSeconds - startTime;
odomVelFile << vel_timer++ << " " << robVelocity << 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;
}

void Stopper::scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan) {
frontRange = scan->ranges[0]; // get the range reading at 0 radians
mleftRange = scan->ranges[89]; // get the range reading at - $\pi/4$  radians
leftRange = scan->ranges[179]; // get the range reading at - $\pi/2$  radians
rightRange = scan->ranges[539]; // get the range reading at  $\pi/2$  radians
mrightRange = scan->ranges[629]; // get the range reading at  $\pi/4$  radians
backRange = scan->ranges[359]; // get the range reading at  $\pi$  radians
leftbackRange = scan->ranges[269]; // get the range reading at  $3\pi/4$  radians
rightbackRange = scan->ranges[449]; // get the range reading at  $-3\pi/4$ 
radians}
}

void Stopper::startMoving() {
ofstream odomTrajFile;
odomTrajFile.open("/home/jdl6264/ros_workspace/src/tutorial_pkg/odomTrajData.csv", ios::trunc);

odomVelFile.open("/home/jdl6264/ros_workspace/src/tutorial_pkg/odomVelData.csv", ios::trunc);
// data files
ofstream laserFile;
laserFile.open("/home/jdl6264/ros_workspace/src/tutorial_pkg/laserData.csv", ios::trunc);

int i = 0; // the index to record laser scan data
double frontAngle=0, mleftAngle=0.785, leftAngle=1.57;
double rightAngle=-1.57, mrightAngle=-0.785;
ofstream laserMapFile;
laserMapFile.open("/home/jdl6264/ros_workspace/src/tutorial_pkg/laserMapData.csv", ios::trunc);

startTime = ros::Time::now().toSec(); // obtain the start time
//double frontRange = 2.0;

ros::Rate rate(50); // Defined rate for repeatable operations.
ROS_INFO("Start moving");
ROS_INFO_STREAM("frontRange: " << frontRange);

```

```

ROS_INFO_STREAM("leftRange: " << leftRange);
ROS_INFO_STREAM("rightRange: " << rightRange); // Info stream
ROS_INFO_STREAM("mleftRange: " << mleftRange);
ROS_INFO_STREAM("mrightRange: " << mrightRange);
ROS_INFO_STREAM("stage: " << stage);
stage = 1;
while (ros::ok()){// keep spinning loop until user presses Ctrl+C
switch (stage){
case 1:
if (frontRange > 0){ // checks if the rosbot is reaching the 1st landmark
moveForward(FORWARD_SPEED_SHIGH);
if(frontRange < 1.1 and frontRange > 0.5){
stage = 2;
}
}
break;
case 2:
if (rightRange < 0.55 and rightRange > 0.32){ // 2nd landmark (turning)
moveForwardRight(FORWARD_SPEED_HIGH,TURN_RIGHT_SPEED_MIDDLE);
stage = 3;
}
break;
case 3:
if (frontRange > 1.15 and rightRange < 0.25){ // 3rd landmark through the
object
moveForwardRight(FORWARD_SPEED_HIGH, TURN_RIGHT_SPEED_MIDDLE);
stage = 4;
}
break;
case 4:
if (frontRange > 0.6 and frontRange < 1.5){ // 4th landmark straight
moveForward(FORWARD_SPEED_SHIGH);
stage = 5;
}
break;
case 5:
if (frontRange > 1 and rightRange < 0.25){
moveForwardRight(FORWARD_SPEED_SHIGH, TURN_RIGHT_SPEED_LOW); // 5th
landmark
stage = 6;
}
break;
// code for landmark 6
case 6:
if (frontRange > 0.25 and leftRange < 0.3){
moveForward(FORWARD_SPEED_SHIGH); // 0.55
stage = 7;
}
break;
// code for landmark 7
case 7:
if (frontRange < 0.2 and rightRange < 0.25){
moveForwardRight(FORWARD_SPEED_LOW, TURN_RIGHT_SPEED_MIDDLE); // 0.55
stage = 8;
}
break;
// code for landmark 8
case 8:
if (frontRange < 0.35){
moveForward(FORWARD_SPEED_SHIGH);
stage = 9;
}
}
}

```

```

}
break;
// code for landmark 9
case 9:
moveStop();
break;
}

odomTrajFile << PositionX << " " << PositionY << endl;
laserFile << i++ << frontRange<< " " << mleftRange << " " << leftRange << "
" << rightRange << " " << mrightRange << endl;
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);
ros::spinOnce(); // Allow ROS to process incoming messages
rate.sleep(); // Wait until defined time passes.
}
odomTrajFile.close();
odomVelFile.close(); //closes data files
laserFile.close();
laserMapFile.close();
}

int main(int argc, char **argv) {
ros::init(argc, argv, "stopper"); // Initiate new ROS node named "stopper"
Stopper stopper; // Create new stopper object
stopper.startMoving(); // Start the movement
return 0;
}

```