DTU Electrical Engineering
Department of Electrical Engineering

31388 Advanced Autonomous Robots

# Final project report

s192230 Manxi Lin
s192146 Caining Liu
s192279 Ruchir Sharma
s192161 Xinyuan Liu

April 18, 2022

# Contents

# 1 Introduction

This report documents the work and results obtained in the course *31388 Advanced Autonomous Robots* at the Technical University of Denmark.

## 1.1 Project Background

As every year DTU held competition for this course in which robots try to complete an obstacle avoidance and drive through all the guided marks. But this year due to the COVID-19 pandemic challenges are replaced with other requirements. This year, all the task has to be performed on the simulation.

## 1.2 Problem Description

In this project, our motivation is to make a dedicated program so that robot can autonomously navigate through all the guided mark while avoiding the obstacle and detects the shape, size and pose of the objects. In addition, to perform the task, the robot also needs to demonstrate its robustness.

We are required to do two tasks. One is to drive the robot to an unknown position according to the guide marks in a maze and the other task is to detect the shape, size and pose of an object in a given region. All the tasks are done in SMR-CL and C++20. To automate some process, python 2.7.17 and shell 4.4.20 are used. Our simulation runs on Ubuntu 18.04.3.

The structure of the report is as follows. Firstly, preparations needed to be done before driving the robot are described in detail. Secondly, driving the robot through out the maze while reading the guide marks is discussed. Thirdly, how laser data is collected is introduced. Lastly, object detection is mentioned in detail.

# 2 Preparations

According to the description provided, our robot should start from the left bottom corner of the map and then move to the position of the first guide mark. We set the starting pose of robot $(0.25, 0.45, 0)$ in our assignment. After that, it reads the guide mark which contains the position of the next guide mark and so on, until it finds number 98. The stopping criteria is whether it reads the number 98 from a certain guide mark.

In order to achieve this goal, the strategy used is graph path planning, which is to find the route from current position to a given goal position. After acquiring the final route returned by the path planner, robot follows the path. So, this task is divided into two chapters, in this chapter only planning part of the route is described. In the next part, how robot drives throughout the route by reading guide marks in the simulator is discussed.

## 2.1 Creating a node map

For planning a path, the first step is to create an environment and then select the optimal path by means of certain searching algorithm. The planner used in this project is graph planner, which is implemented with the help of a plugin module called "*auplan.so.0*". This path planner is based on node-map path planning method and the map construction relies on nodes and connections. Therefore, map construction with nodes and connections is the first step for applying the graph planner.

Adding the points in the maze and establishing a connections between them are done by passing the graph planner plugin to the laser server. This way robot calculates the shortest distance between two points.

Firstly, planner is reset for the new graph. Secondly, nodes are defined in the maze which is black in color, total 25 nodes are used. To add the nodes, *addpoint* function is used. For instance, *laser addpoint pno=1 x=0.25 y=0.45*. The map containing all the nodes is given below:

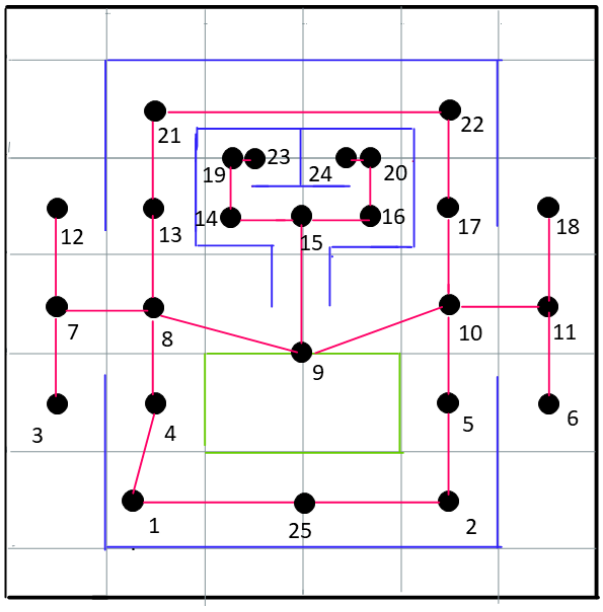
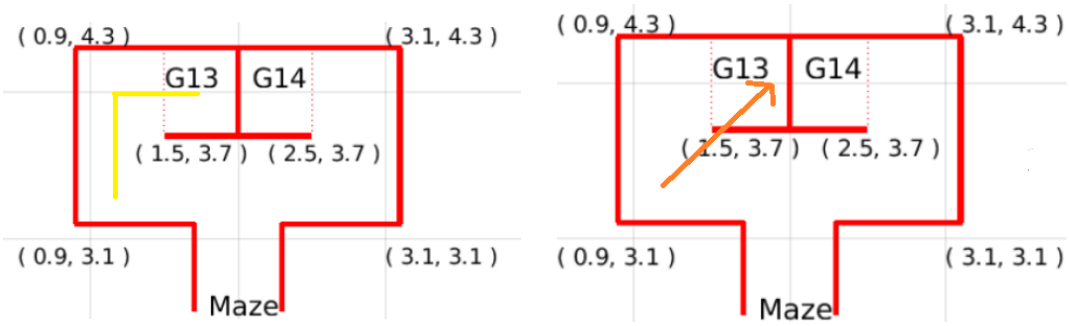



Figure 1: Graph Map for task 1

As it can be seen from the figure that nodes are added in front of each guide mark so that robot can move to the goal position with marginal error which depends on the Kalman filter. Apart from that, two nodes are added in front of both the guide marks 13 and 14 shown in figure 1. During the implementation we found that if we don't add the nodes, the robot will collide with the wall, as there is a wall in between the path which the robot has to follow to reach the guide mark. For this reason, the path is in L shape as shown in figure 2a.




(a) Map showing guide marks L path (b) Map showing collision

Figure 2: Maps showing the L-shape path and collision

Once all the nodes are defined, connections between nodes are generated using *addcon* function. Red color is used to represent the connections. The connections are bi-direction in our node map. The image showing the route from node 4 to 17 is given in figure 3. To define the pose of the guide marks three arrays are created each containing x-value, y-value and $\theta$ value. After that cost of the connections are calculated by using *calculatecost* function. Then, the localizer is called with push command so that we can keep track of the robot in the environment.

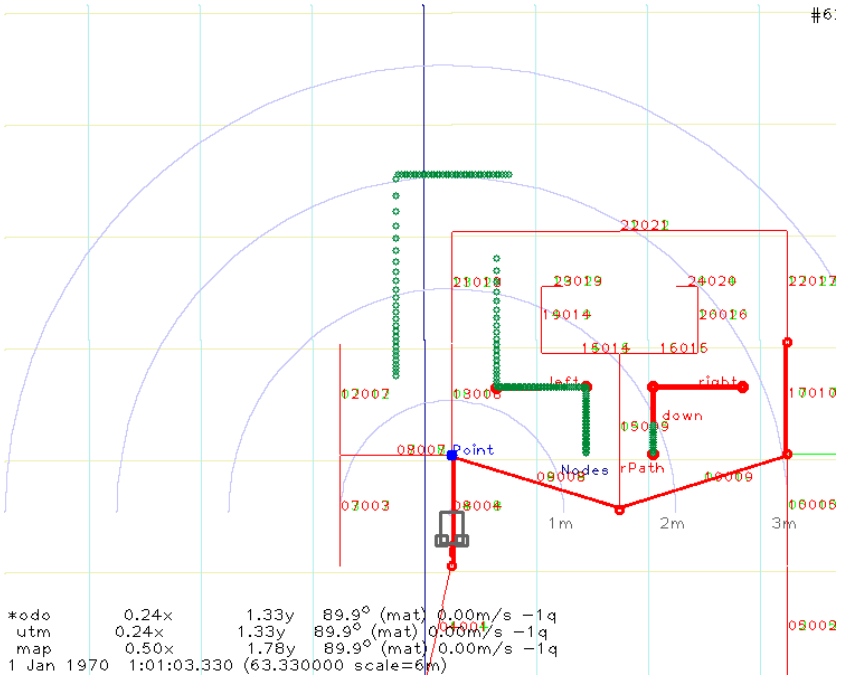



Figure 3: Connection between nodes

## 2.2 Kalman Filter calibration

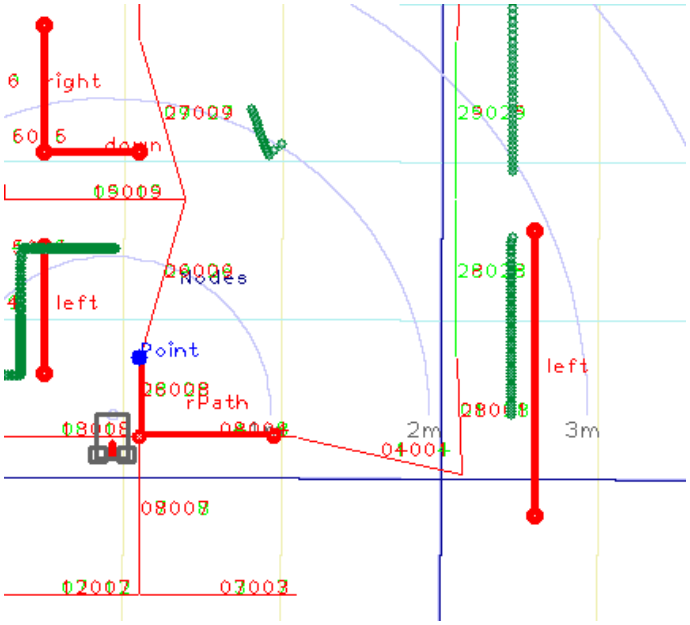



Figure 4: Map showing walls(red) and laser points(green)

Since the odometry error accumulates with time goes by. To localize the robot accurately, the Kalman Filter is needed. For the Kalman filter, *aulocalize* plugin is used. For that we set starting pose *(0.25, 0.45, 0)* and initial covariance *0.05,0.05,0.05* for x, y and $\theta$ respectively.

From figure 4, the red lines show the walls created in the simulator and dark green lines represent the laser points.

As it can be seen from the figure 4 there is a shift between the red lines and green lines. It is because that the data from laser scanner is not accurate. For that, calibration is needed to decrease the offset. Before driving the robot through the maze, we controled the robot to move to node 4, $(0.5, 1.5)$. After arriving there, we called odometry to get the relative pose of the robot to the origin, denoted as $x_{odo}$, $y_{odo}$ and $\theta_{odo}$. As stated before, the starting pose of the robot is $(0.25, 0.45, 0)$. The Equation used for current pose of the robot is given below:

$$x_c = x_{odo} + 0.25 \tag{1}$$

$$y_c = y_{odo} + 0.45 \tag{2}$$

$$\theta_c = \theta_{odo} \tag{3}$$

After that, we called Kalman Filter to make measurement update, resulting in an estimated pose, $x_l$, $y_l$ and $\theta_l$ of the robot. And then using that, we calculated the shift(offset) using equations given below:

$$x_{shift} = x_c - x_l \tag{4}$$

$$y_{shift} = y_c - y_l \tag{5}$$

$$\theta_{shift} = \theta_c - \theta_l \tag{6}$$

Once we get the offset, corrections are made for every target poses which are used later in the project.

$$x_{newtarget} = x_{target} + x_{shift} \tag{7}$$

$$y_{newtarget} = y_{target} + y_{shift} \tag{8}$$

$$\theta_{newtarget} = \theta_{target} + \theta_{shift} \tag{9}$$

# 3 Reading the the guide marks

As connections and nodes are initialized and discussed in the previous chapter. In this part, how the robot drives from one guide mark to another is explained. This includes two important steps: the first one is to drive the robot to a guide mark and the second one is to read the guide marks.

## 3.1 Drive to a guide mark

An important aspect of following the guide marks is to drive the robot from one node to another. In order to achieve this goal, "*drivew*" command is called. However, the "*drivew*" command works better when driving a robot in a straight-line rather than along a curve. Therefore, to drive the robot from one point on the map to another, making specific strategies is necessary. The solution is clarified in table 1 and figure 5.

Table 1: Drive the robot from pose A to pose B

| |
|---|
| let $A$ to be the current pose of the robot, $A = (x_1, y_1, a)$ in world frame |
| let $B$ to be the target pose of the robot, $B = (x_2, y_2, b)$ in world frame |
| let $dx$ to be the distance between A and B along x direction |
| let $dy$ to be the distance between A and B along y direction |
| let $c$ to be the angle from x axis to B, $c = atan(dy, dx)$ |
| control the robot to rotate angle $c - a$, then the robot should towards B |
| use "drivew" to drive the robot forward $\sqrt{dx^2 + dy^2}$ |
| rotate the robot angle $d$ to correct the orientation of the robot to be $b$ |

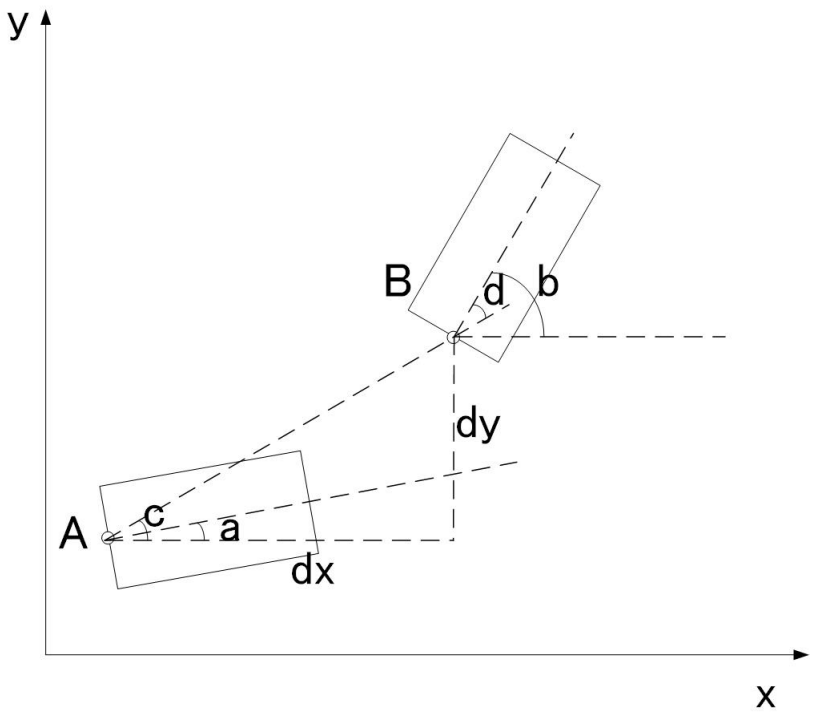



Figure 5: Drive from a point to another

Obviously, the angle **a** and **b** is the orientation in **A** and **B** respectively. So, odometry coor-

dinates are ought to be transformed to world coordinates in advance. This can be performed by using command "*invtrans $l0 $l1 $l2 $odox $odoy $odoth*". The results are stored in variables $res0, $res1 and $res2, representing the current x, y and $\theta$ of the robot respectively. Regarding the laser offset, the target pose, that is **B** in the figure, should consider shifting values. It is wise to normalize the angle we calculated. Without any doubt, turning **d** degrees to make sure the robot heading at the target node and commanding it move to that position are necessary.

We can use the above solution to drive the robot from one node to another. To achieve this goal, one variable is used to store the number of the target guide mark. Once we get the pose of the guidemark, we pass the pose to the $findroute$ function which returns us the route to the guide mark. After that, graph planner plugin knows which nodes lie in between the current guide mark and the target guide mark. To retrieve that information, *stringcat getpointp* command is called. This command returns a vector $v$ from the current robot position, pointing to the position of node $p$. Take the current pose of the robot and the vector $v$ into **A** and **B** described in table 1. When the robot arrives at node p, the counter that starts from the number stored in $l4 should decrease by one and this can be used to check if the robot has arrived at the final node in the route or not. Once the robot reached to the target node, it should correct its orientation so that the robot can be situated at the correct position with the orientation we want. With the process described above, we can drive the robot to an arbitrary node in the map.

In order to illustrate the following guide marks task in a clear way, the image showing the map with guide marks is given below:

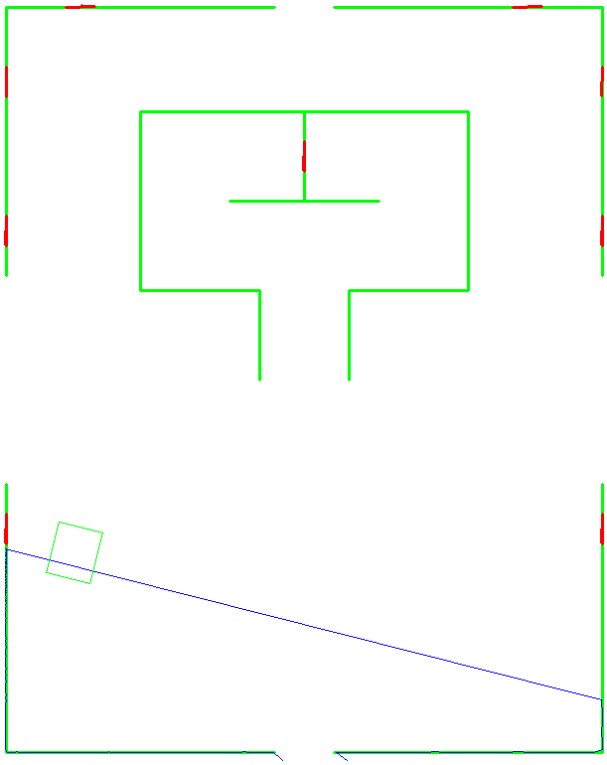
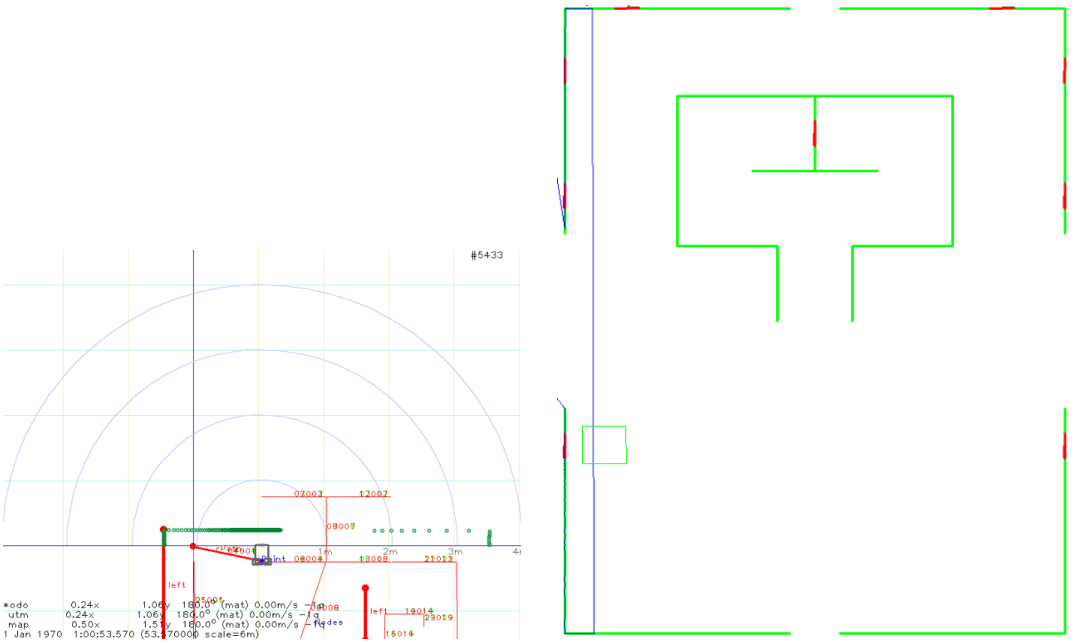


Figure 6: Guide marks in the map

Our prior knowledge about the guide marks is their poses. However, in real application, the robot is not able to drive to the pose directly, considering that the robot can not be regarded as a point, but as a polygon. That's why we need to make some changes to the coordinates of the obtained guide mark, and then take that as the target position. According to the figure, it is easy to find that most guide marks, which are shown in red lines, are attached on vertical

walls and there are only two guide marks attaching on the top wall of the maze, so that we can separate these three conditions to deal with: $gmth = 0$, $gmth = \pi$ and $gmth = \pi/2$, where gmth is a variable we defined to store the orientation of the target guide mark.



(a) Reading the guide mark 2 in qclient (b) Reading the guide mark 2 shown in the map

Figure 7: Reading guide mark

### 3.2 Read the guide marks

The last process during each movement is to read the guide mark and then get the guide mark number with which we know the pose of the next guide mark. In order to read the guide marks, *fiducialid* command is used. Normally, the data may not be the final value 98. When the guide mark returns a number greater than zero, then the number can be used to find the next guide mark. However, the robot may receive a bad result due to its orientation that is not correct enough so that it is necessary to handle this situation using a trick that the robot can turn a small angle like only ten degrees and then fetch the guide mark information again. After adjusting the orientation slightly, the robot can finally head at the guide mark and read the data correctly. Once the robot reads the guide mark with number 98, the first mission stops and the next mission carried away. The robot reading the first guide mark is shown in figure 7.

### 3.3 Route in application

The robot's trajectory is shown in figure 8. This is based on the odometer data that we got during the experiment. A $0.35 \times 0.3$-rectangle is used to represent the robot. As we can see from the figure that, the robots scanned guide mark 2, 3, 6, 7, 11, 12, 13. We set the guide mark 2 to be the first guide mark. According to our experiment recording, the robot was driven to guide mark 2, 7, 11, 3, 6, 12 successively, and it finally went to guide mark 13 and stopped.

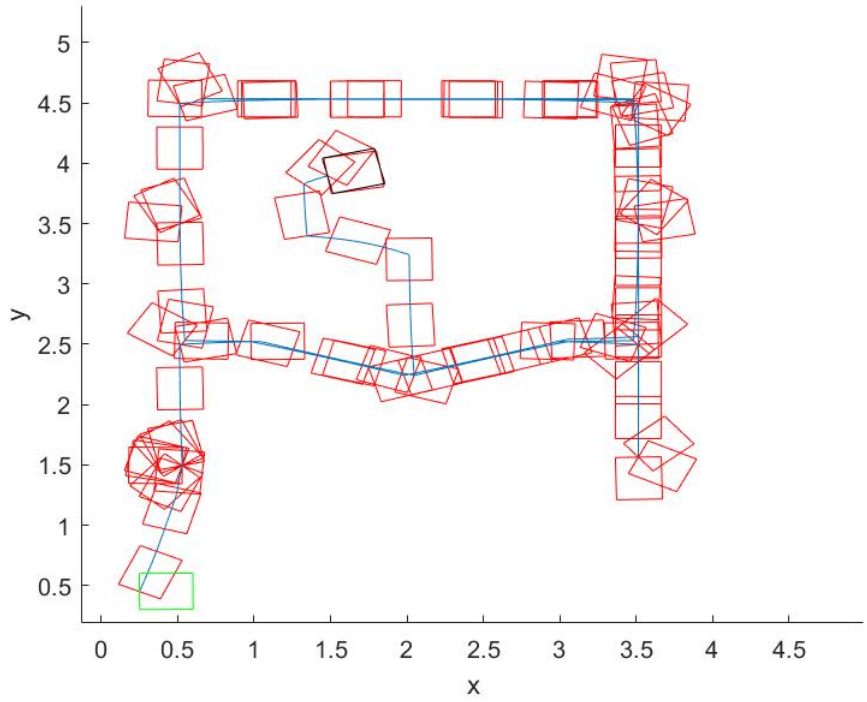


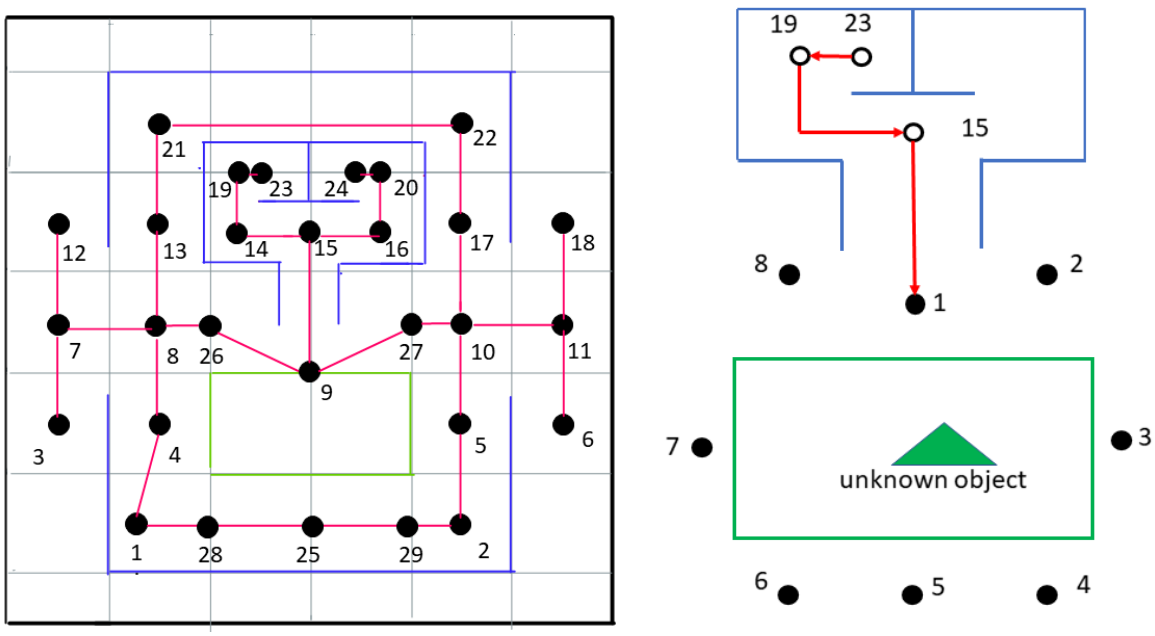

Figure 8: Plot shows poses of the robot

# 4 Data collection for object detection

Since, the first mission is discussed in the previous two chapters, in the next two chapters, we mentioned about the second task. The requirement is to fetch the general information: shape, size and pose of an unknown object which is located somewhere within the marked green square in the map. We have already known that the possible shape of the object is either triangle or rectangle. The most convenient way for a mobile robot to scan an area is to use its laser scanner so that the object can be identified with the coordinates of point o and the orientation of the object.

## 4.1 Sequence of object scanning

In this part, we focused on the data collection for an object detection. Take the given case for example, as stated before, the robot stops at node 23 after the first mission is finished. We manually control the robot move backwards for a distance, from node 23 to node 19. It is because there is not enough space for the robot to turn around automatically. For sure since there is an abundant allowance left when designing the node map, even if the termination of task 1 is neither node 23 nor 24, going backward for a distance will not result in any collision. Afterwards, the robot can move with the help of graph planner from node 19 to node 15. To ensure the robot doesn't enter the green region, we drive the robot in the forward direction about 0.8 meters. That is, node 9 is moved a little upward in task2, from $(2, 2)$ to $(2, 2.4)$. The red arrows in figure 9b illustrate the process.




(a) Graph map having 29 nodes

(b) Sequence of nodes in scanning

Figure 9

For the green square, we design a route containing 8 nodes near the boundary, 4 of which are added newly. Their series number are 26, 27, 28 and 29. The new node map is shown in figure 9a.

The numbers in figure 9b refer to the sequence of nodes used in scanning.

Table 2 shows the coordinates of the nodes and poses of the robot. For example, node $1(2, 2.4, -0.5\pi)$

means that the X coordinate of the node is 2 and the Y coordinate of the node is 2.4 and the robot orientation is -90 degree relative to the world coordinate system.

Firstly, robot moves to point 1 in figure 9b, heading towards the bottom of the maze. Then, robot moves to point $2-8$ in sequence. At each position, robot performs an object scanning and collects the laser points. What the laser scanner collects and returns to the server is discussed in the next section more specifically.

Table 2: Coordinates of nodes

| nodes | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| X coordinate | 2 | 3 | 3.5 | 3 | 2 | 1.01 | 0.5 | 1 |
| Y coordinate | 2.4 | 2.5 | 1.5 | 0.5 | 0.5 | 0.5 | 1.5 | 2.5 |
| pose of robot | $-0.5\pi$ | $-0.75\pi$ | $\pi$ | $0.75\pi$ | $0.5\pi$ | $0.25\pi$ | 0 | $-0.25\pi$ |

## 4.2 Selecting the position of robot for object scanning

It is important to explain why we choose these 8 nodes as object scanning position for the robot. From the assignment instruction, we have known that the length of robot is $0.35m$ and the width of the robot is $0.3m$. There exists a case where the object happens to be spawned near the edge of the green square. Therefore, if we set the 8 nodes too close to the border of the green square, when the robot approaches the square, there is some probability that robot collides with object. To figure out a suitable distance between the stop position for robot and the green square to avoid a potential collision, we need to analyze the size of the robot, as shown in figure 10. The robot size is given as:

$$length = 0.35m$$

$$width = 0.3m$$

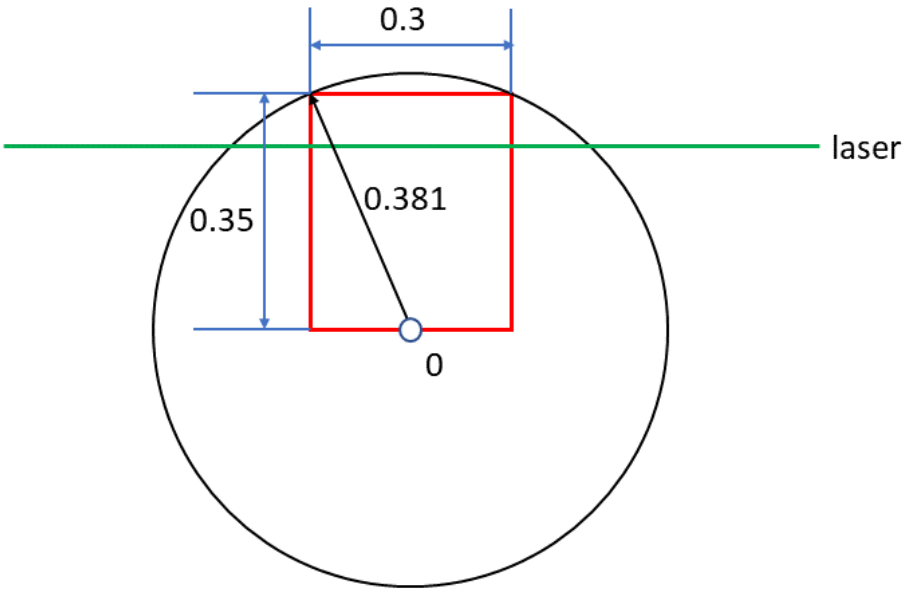



Figure 10: Distance between scanning position to the square

To simplify the calculation, in our assignment, we put the origin of our robot coordinate system on the midpoint of the robot's two wheels. When the robot turns around, the path of

its motion can be described with a circle. The centre of the circle is the robot origin, and the radius of the circle can be calculated by the equation given below:

$$r^2 = l^2 + (0.5w)^2 \tag{10}$$

where l and w are the length and width of the robot, respectively. The radius is calculated to be $r \approx 0.381m$. Considering allowance, let's round it up to $0.5m$. Thus, the maximum range of the robot motion is a 0.5m-radius circle with the robot origin as the centre. The minimum distance between the obstacle and the robot should be $0.5m$. According to this principle, we defined the coordinates of the 8 points.

### 4.3 Route in application

Figure 11 shows the route of the robot when doing object scanning, which is based on the odometer data we got during the experiment. To visualize the simulation result, We used a $0.35 \times 0.3$-rectangle to represent the robot. The blue line stands for the path of the robot. The sampling frequency is set to 150 Hz, which means that we picks and plots a robot pose at every 150 sampling time. In particular, the green rectangle illustrates the start pose of robot and the black one is the final pose of it.

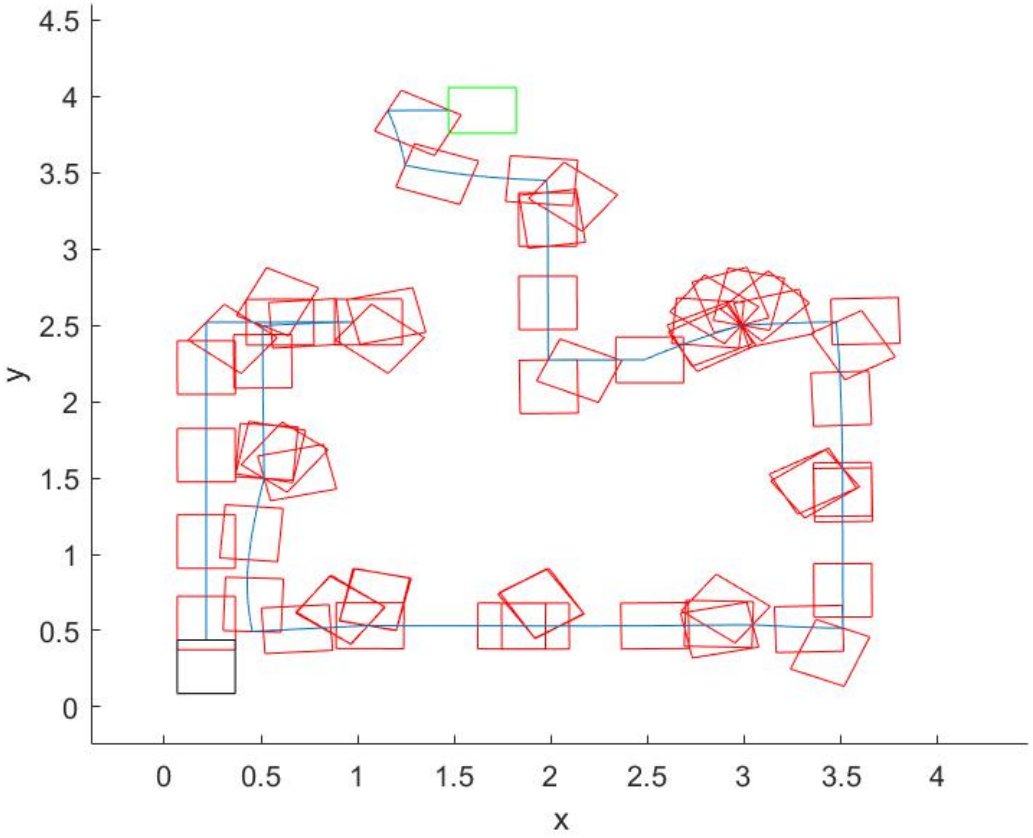



Figure 11: object-scanning route

After finishing the first mission, the robot stops at the position of the green rectangle in figure 11, that is node 23 in figure 9a, which is also the starting point of the second task. We manually make the robot go backwards to node 19. Then, the robot moves automatically to the first object scanning position and so on. After robot moves to the last scanning position and finishes object detection, the robot goes backward and finally moves to the origin. As it

can be seen from the figure, the black rectangle is inside the $0.6 \times 0.6$ square mentioned in the assignment instruction.

So far, the robot has collected all data for object detection, the process of recognizing the object will be discussed in the next chapter.

# 5 Object detection from laser data

As basic steps to finish the second mission have already been clarified in the previous chapters, in this chapter, the process of recognizing the object from laser data is set to the focus.

## 5.1 Transformation from the laser to the world

Figure 12 shows the three frames created for detection: the laser coordinate system, the robot coordinate system and the world coordinate system, which is in green, red and black respectively. The purple point, $p$, represents part of an object in the map.

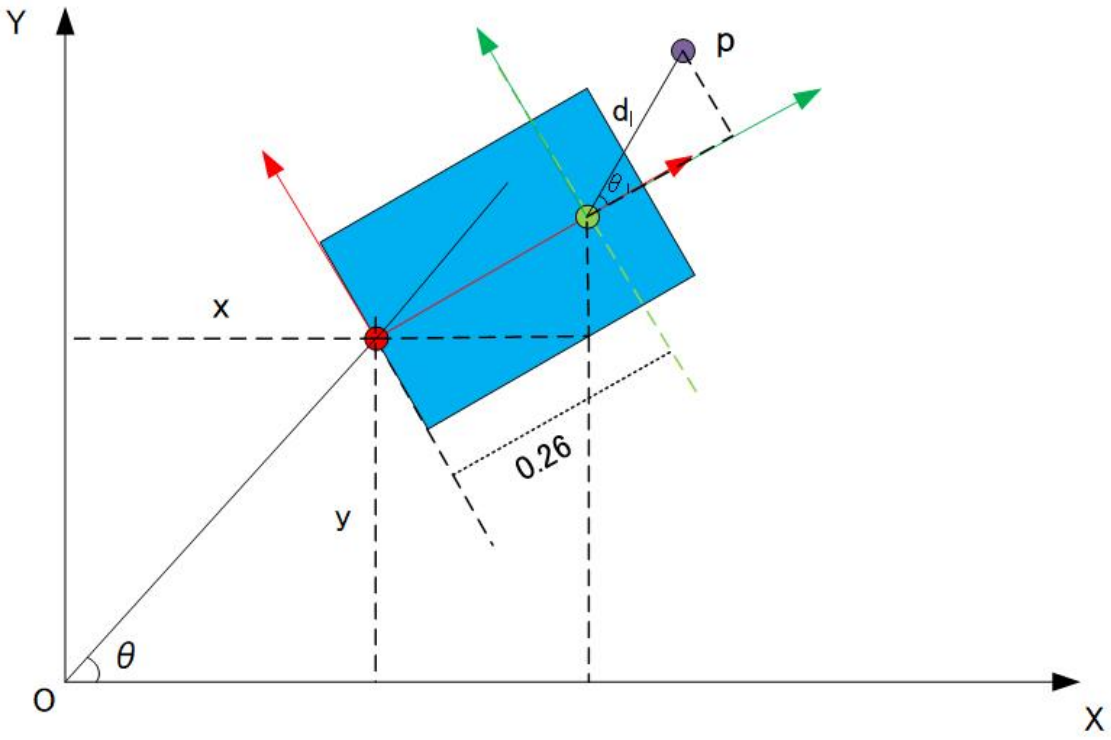



Figure 12: Coordinate Systems

In order to unify the coordinates of the object scanned in 8 different positions, we need to transform them into the world coordinate system. What the laser scanner returns to the server are the polar coordinates of $p$, denoted as $(\theta_l, d_l)$ in the figure 12. To make the pose of the point more clear, one can convert the polar coordinates to Cartesian coordinates according to the equations below:

$$x_l = d_l \cdot cos(\theta_l) \tag{11}$$

$$y_l = d_l \cdot sin(\theta_l) \tag{12}$$

where $x_l$ and $y_l$ are the coordinates of $p$ in the laser coordinate system.
According to the instruction for the assignment, the pose of the laser scanner in the robot coordinate system is $(0.26, 0, 0)$, indicating the transformation between the two frames:

$$x_r = x_l + 0.26 \tag{13}$$

$$y_r = y_l \tag{14}$$

where, $(x_r, y_r)$ is $p$'s position in the robot coordinate system.
When it comes to the world coordinate system, the position of $p$ depends on the robot pose $(x, y, \theta)$, given by:

$$x_w = x_r cos(\theta) - y_r sin(\theta) + x \tag{15}$$

$$y_w = x_r sin(\theta) + y_r cos(\theta) + y \tag{16}$$

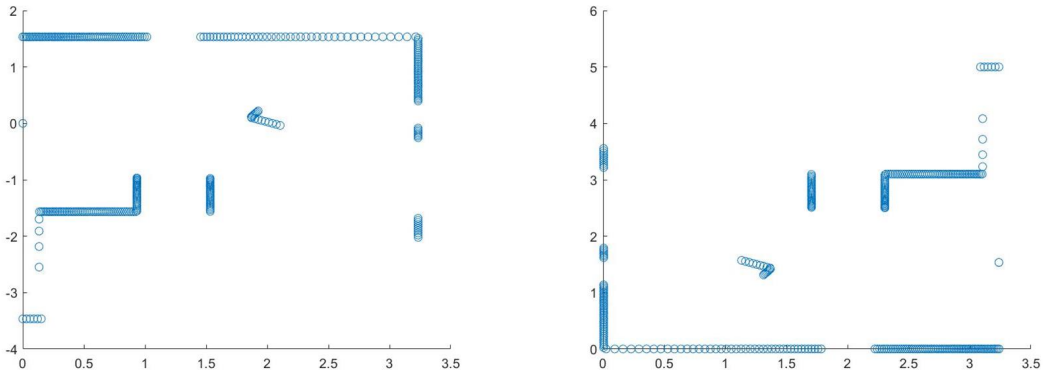



Figure 13: Laser points before and after transformation

Figure 13 illustrates the transformation on the laser points scanned at robot pose $(2, 2.4, -\pi/2)$. Since the object lies in a certain region, we can remove unnecessary points. The result of fusing laser points from all 8 positions is presented in figure 14.

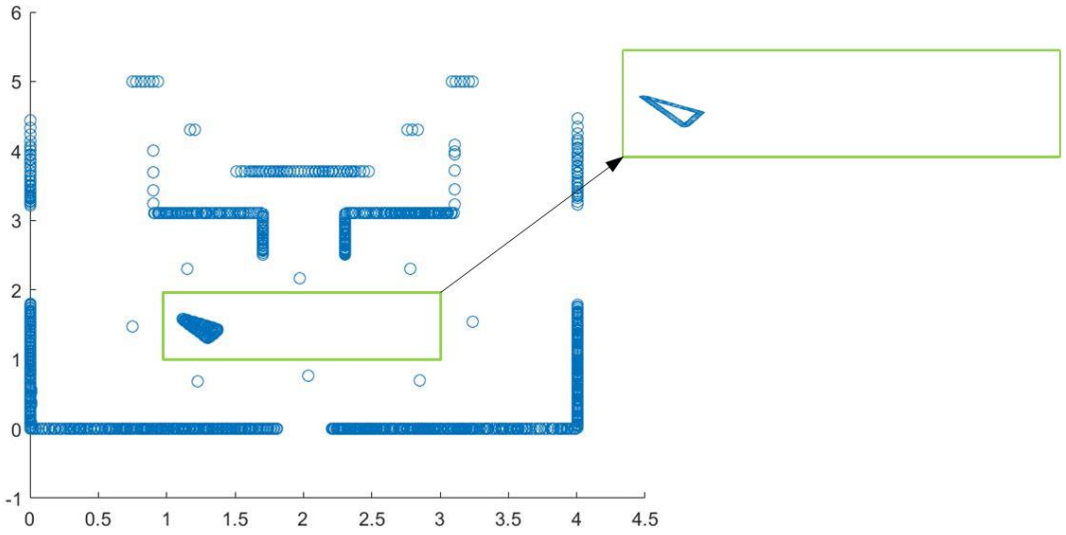


Figure 14: Laser fusion

## 5.2 Finding shape, size and pose

From figure 14, one can easily give the shape, size and pose of the object by observation while it's difficult for a program to do the same thing. The RANdom SAmple Consensus algorithm (RANSAC) maybe a proper solution.

Algorithm 1 illustrates the process of the classical RANSAC, which can only detect one line at a time, whereas there are at least 3 lines to detect in our case. Therefore, algorithm 2 is proposed.

**Algorithm 1** RANSAC

let $P$ to be a set of points
let $i$ to be maximum number of iteration
let $\sigma$ to be the threshold for determining the inliers
**repeat**
  randomly sample 2 points from $P$
  calculate line parameters
  computes the distance from all points in $P$ to the line
  score the line by the fraction of inliers whose distance to the line is given below $\sigma$
**until** $i$ reached
the line with the highest score is chosen as the solution

**Algorithm 2** RANSAC for multi-lines

let $P$ to be a set of points
let $i$ to be maximum number of iteration
let $\sigma$ to be the threshold for determining the inliers
let $\gamma$ to be threshold to select lines
**repeat**
  randomly sample 2 points from $P$
  calculate line parameters: slope $a$ and intercept $b$
  computes the distance from all points in $P$ to the line
  score the line by the fraction of inliers whose distance to the line is given below $\sigma$
**until** $i$ reached
put the lines whose scores are over $\gamma$ into a set $L$
project the lines in $L$ to points in $a$-$b$ space
combining the lines using adaptive k-means, k is selected from {3,4}

However, as it can be seen from the algorithms before, there are many parameters to determine in RANSAC. What's worse, the result of RANSAC changes everytime because it chooses points randomly to fit the line. In RANSAC, the probability of getting optimal result depends on the number of iteration. That is, RANSAC takes a longer time to get the better result.

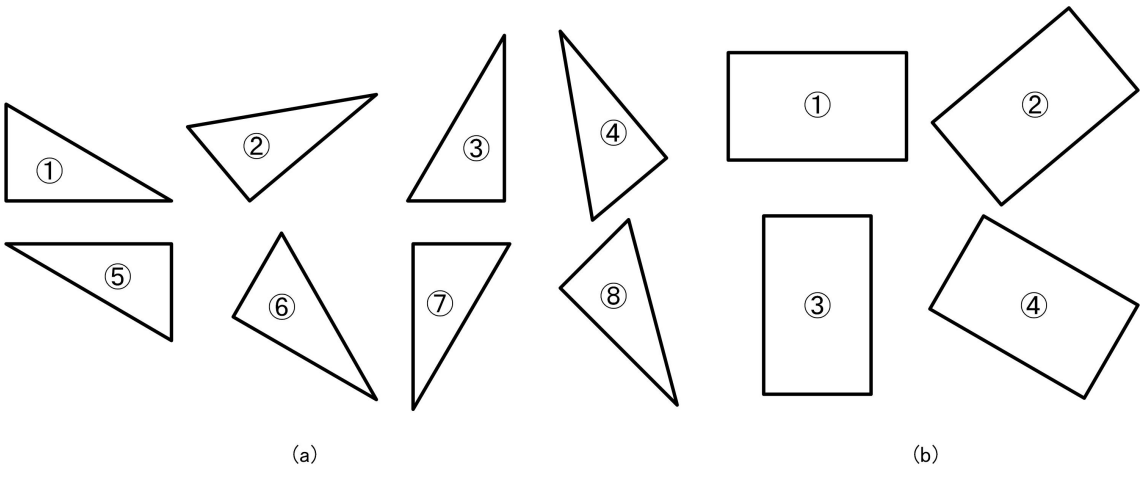



Figure 15: Potential poses

Therefore, in our assignment, another method is chosen. Figure 15 illustrates the potential pose of the objects. The figure suggests that since triangle and rectangle are convex, the leftmost/rightmost/bottom/top points in the laser points should be the vertices of the object. Afterwards, we only have to find the points with the maximum and minimum x or y coordinates, instead of looking for 3 or 4 lines via RANSAC. We considered some special cases, shown in figure 15 (a) ① ③ ⑤ ⑦, and (b) ① ③, there are more than 4 points of the object lies on its bounding box and thus special treatments are needed. The whole process is presented in algorithm 3.

**Algorithm 3** Object detection with vertices

let $P$ to be a set of laser points
let $E$ to be the set including the leftmost, rightmost, top and bottom points in $P$
let $\sigma$ to be the threshold of the distance between points
**if** the number of points in $E$ is more than 4 **then**
  find the corresponding situation in figure 15 and get the vertices $V$
**else**
  for every two points in $E$, if the distance of the two points is less than $\sigma$, replace them with their average
  let $V$ to be the output of the last step
**end if**
**if** the number of the points in $V$ is 3 **then**
  the shape of the object is TRIANGLE
**else if** the number of the points in $V$ is 4 **then**
  the shape of the object is RECTANGLE
**end if**
use $V$ to calculate the pose and size of the object

Evidence suggests that the algorithm shows a formidable performance in application. The result is illustrated in the table below.

Table 3: The result of detection

| Type | Shape | Position(m) | Width(cm) | Height(cm) | Orientation(rad) |
|---|---|---|---|---|---|
| Truth | Rectangle | (2.511, 1.431) | 40 | 15 | 2.4352 |
| Detected | Rectangle | (2.515, 1.429) | 39.46 | 15.75 | 2.4293 |
| Truth | Rectangle | (2.525, 1.727) | 30 | 20 | 2.5775 |
| Detected | Rectangle | (2.512, 1.693) | 29.79 | 19.56 | 2.5650 |
| Truth | Triangle | (2.102, 1.562) | 40 | 10 | 2.1204 |
| Detected | Triangle | (2.089, 1.530) | 34.4 | 9.50 | 2.1446 |
| Truth | Triangle | (2.295, 1.625) | 30 | 15 | -1.9076 |
| Detected | Triangle | (2.293, 1.545) | 30.63 | 14.60 | -1.9559 |

Considering the shifting problem mentioned in section 2.2, necessary corrections are made to adjust the detected position of the object. The result demonstrated that, apart from the position, the measured size and orientation of the object have little error.

# 6 Conclusion

## 6.1 Contribution

According to the assignment description, each of us should be responsible for a chapter. Our contribution is shown in the table below.

| Student number | Name | Chapter |
|---|---|---|
| s192279 | Ruchir Sharma | 2, Prior works |
| s192146 | Caining Liu | 3, Reading the guide marks |
| s192161 | Xinyuan Liu | 4, Data collection for object detection |
| s192230 | Manxi Lin | 5, Object detection from laser data |

## 6.2 Summary

In our report, two tasks are performed successively. The first one is to move the robot to a previously unknown target position by reading the guide marks. The second one is to detect an object in a given region. Figure 8, figure 11 and table 3 demonstrate the validity of our solution. However, we found some problems in experiments. As stated in the section 2.2, although we used Kalman Filter calibration to decrease the laser shifting, the offset did exist after calibration, which can be reflected on the detected positions in table 3. More work is needed to be done in the future to solve this problem. What's worse, in the beginning of task 2, prior knowledge is introduced in order to avoid potential collision. That is, we control the robot to move backward for a distance. By using two unidirectional node maps, of which one is designed for entering the maze and the other is for leaving the maze, the problem can be solved.