Group31 Report

I. Design

First of all, after careful analysis and understanding of the project, we divided the project into three parts, which are:

1. Movement:

Drive the robot to walk to the specified position according to the specified coordinates.

2. Detecting the green room:

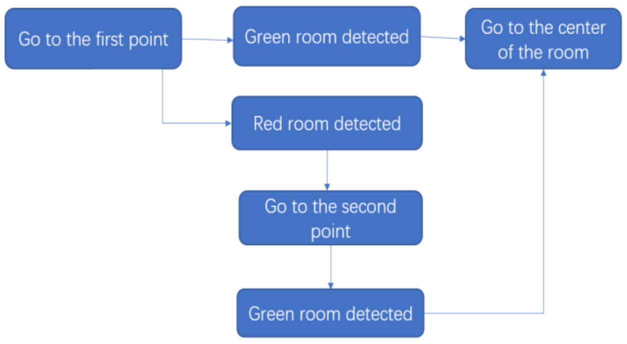
After the robot has reached the recognition positions of the entrances of the two rooms, it will recognise the red and green rooms respectively and detect the green room.

3. Character identification:

After the robot arrives at the centre point of the green room, the robot will follow a certain path strategy to detect the character image and save the identification image and the character name.

Movement

We use the coordinates provided in the “input_points.yaml” file and ‘move_base. send_goal’ to let the robot walk to a certain point. After reaching the specified position, the terminal will report "the robot has reached the specified position".

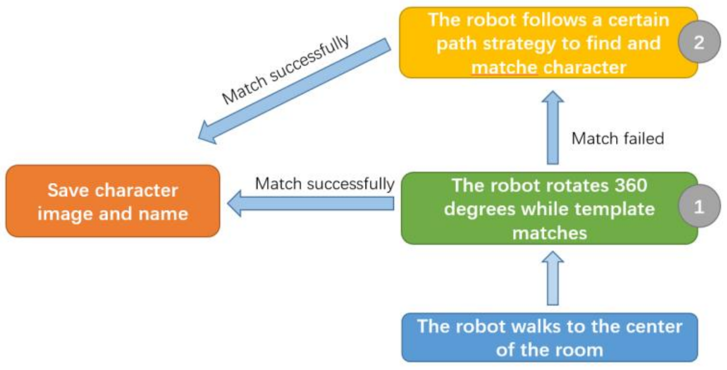


Detecting the green room

In the second part, we solve the problem mainly through the design of circle recognition. Firstly, we defines the boundaries of the green filter and the red filter, calibrated with green and red circle pictures in the simulated room. The image captured by the robot is then filtered and converted to greyscale images respectively for green colour and red colour. The areas of valid colours that appear in the image are calculated as red_max_area and green_max_area. The coordinate of the centre of green colour is calculated as cx and cy. If either the value of red_max_area or green_max_area exceeds the threshold (currently 1000, to be adjusted later in testing), and the x coordinate of the centre of the green colour is between 280 and 360 (approximately the centre area of the captured image),. The detecting will be successful.

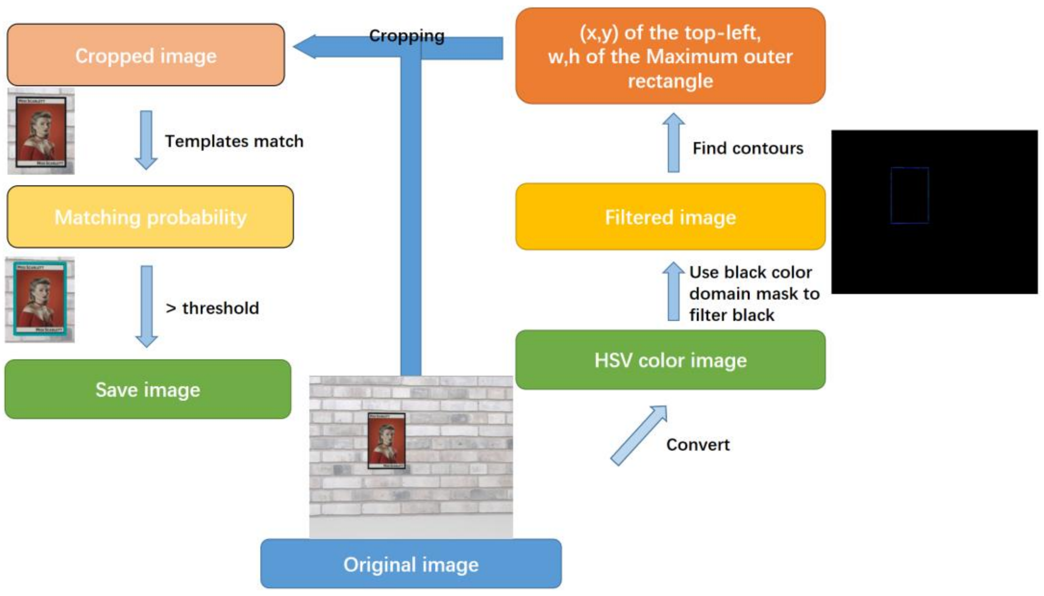
Character identification

The following figure shows the logic executed by the robot in the green room. After the robot reaches the centre point of the room, it will start the template matching strategy and rotate it in place for one round. If the matching is successful, the picture and character name will be saved, but if it failed, the path strategy would be turned on.


The robot follows a certain path strategy to find and matche character
2
Match successfully
Match failed
Save character image and name
Match successfully
The robot rotates 360 degrees while template matches
1
The robot walks to the center of the room

Matching strategy

Based on the observation of the four-character images, it is found that there is a black border around these images, So I designed a matching strategy based on this black border. The following figure is the logic flow chart of the matching strategy.



a) First of all, the photo was taken by the robot camera is the original image, which will be converted to the HSV colour gamut in the next step.

b) After that, a black colour gamut mask filters out the black part of the image.

c) Then, we use the ‘cv2.findcoutours’ function to find the contours on the filtered image, and use ‘cv2. boundingRect’ to obtain the coordinates of the top-left point of each black contour and the width and height of the rectangle surrounding the contour. Since the width and height of the character image are proportional, only a similar ratio width and height of black contours can be saved and passed to the next step.

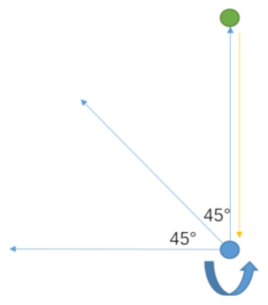
d) In the next step, to remove the interference items, the original image is cropped, according to the top-left point coordinates, w and h.

e) The cropped image is then matched with four templates by ‘cv2. matchTemplate’. If the maximum of the four matching results is greater than the threshold, the character is considered to be found. At the same time, save the image and output the character name of the template corresponding to the maximum matching rate

Path strategy

We designed two different path strategies to find character images. During the execution of these strategies, template matching is turn on. When the matching is successful, the strategy will stop.

Strategy 1:



a) The robot first walks forward a predetermined distance in one direction and rotates 360 degrees after reaching the endpoint.

b) After the rotation is over, the robot will return to the centre point of the room. Then increase the direction by 45 ° and repeat the above operation.

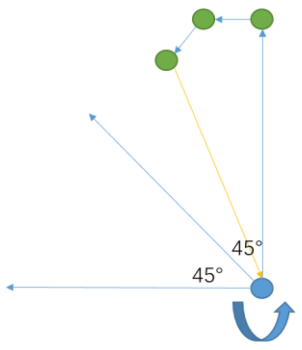
c) After the eighth execution or matching successfully, the strategy will stop.

d) Meanwhile, during walking, if it encounters an object, the robot will retreat, and then return to the centre point to operate in the next direction.

Advantage: Less time used.

Disadvantage: Can't walk to the back of the obstacle and the visible range is limited, the character may not be found.

Strategy 2:



a) The robot first walks forward a predetermined distance in one direction and rotates 360 degrees after reaching the endpoint.

b) The robot rotates 90 ° to the left, then walks forward a specified distance and stops to rotate 360 °.

c) Afterwards, The robot rotates 45 ° to the left, stops after walking a distance, and then rotates 360 °.

d) The robot returns to the centre of the room, rotates 45 ° to the left, and repeats a, b, c, and d until eight executions.

e) In operation step a, when encountering an obstacle, the robot will take a few steps back, and then execute b. Meanwhile, when encountering obstacles in b and c, the robot will take a few steps and execute d.

Advantage: Have a larger visible area. And it can bypass obstacles and see the wall behind them. It has a higher probability of finding the target.

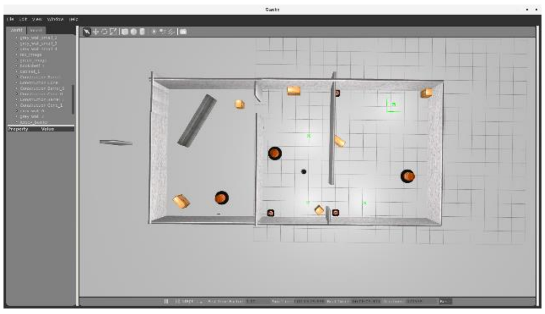
Disadvantage: Time-consuming, and there is a chance that it will get stuck in the dead end.

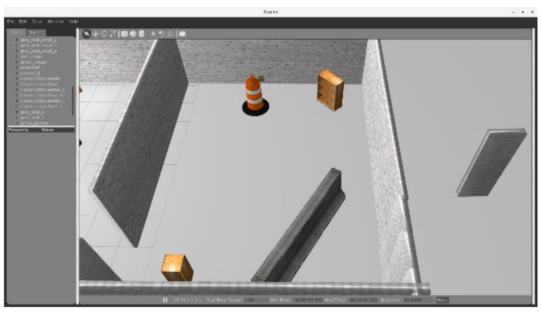
II. Implementation and Testing

Implementation of a new simulated environment

We implemented a new simulated environment to test the robustness of our programme. We designed the new environment by changing the location and orientation of the existing objects and adding new objects. The result is an environment where the starting room (where the default location of the robot is) is in the middle between two other rooms. The green room and red room are twice as large as the rooms provided in the example, which increases the difficulty for the programme to find the Cluedo image because the image is now further away from the centre point. We also deliberately placed the Cluedo image behind a cone barrier, making it out of the direct line of sight after the robot reaching the room. We specifically designed a path strategy which instructs the robot to go in a pattern that resembles an asterisk-shape. That way our robot can bypass the obstructions placed between the robot and the Cluedo image.

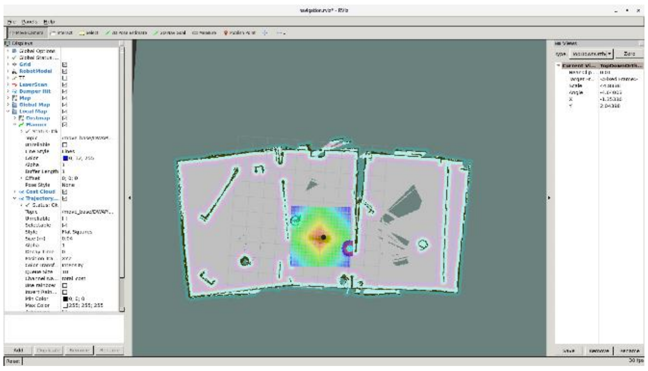
Following are a few screenshots showing the new environment.





To make the robot run in the simulated environment, we also need to create a map such that the robot can navigate itself. We ran the mapping function and manually walked the robot through all rooms, thus obtained a map of the environment.

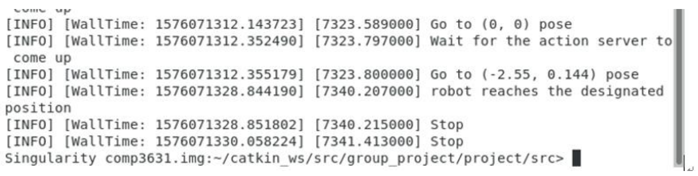
Sensor map of the new environment:



Movement testing

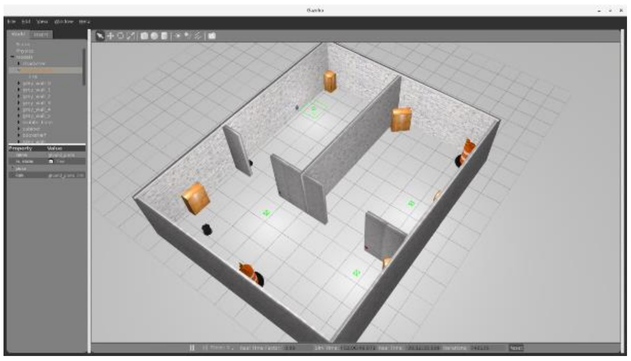
To complete the move test, we randomly selected a number of coordinates, and record the accuracy of the robot to the destination coordinates. During the test, the robot walked to the correct coordinates exactly as instructed. As shown in the figure below, I extracted two points from it as examples.

The coordinate of this point is <-2.55, 0.144>, and the robot reached the exact position.

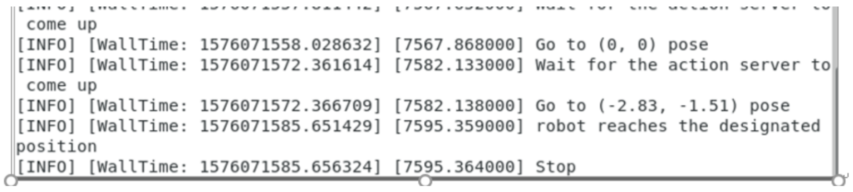


```
 come up
[INFO] [WallTime: 1576071312.143723] [7323.589000] Go to (0, 0) pose
[INFO] [WallTime: 1576071312.352490] [7323.797000] Wait for the action server to
 come up
[INFO] [WallTime: 1576071312.355179] [7323.800000] Go to (-2.55, 0.144) pose
[INFO] [WallTime: 1576071328.844190] [7340.207000] robot reaches the designated
position
[INFO] [WallTime: 1576071328.851802] [7340.215000] Stop
[INFO] [WallTime: 1576071330.058224] [7341.413000] Stop
Singularity comp3631.img:~/catkin_ws/src/group_project/project/src>
```

Run the move command, the robot first reaches the coordinates <0, 0>, then come to <-2.55, 0.144>, and finally the terminal reports that the robot reaches the designated position.



The coordinate of this point is (-2.83, -1.51), and the robot reached the exact position.



```
 come up
[INFO] [WallTime: 1576071558.028632] [7567.868000] Go to (0, 0) pose
[INFO] [WallTime: 1576071572.361614] [7582.133000] Wait for the action server to
 come up
[INFO] [WallTime: 1576071572.366709] [7582.138000] Go to (-2.83, -1.51) pose
[INFO] [WallTime: 1576071585.651429] [7595.359000] robot reaches the designated
position
[INFO] [WallTime: 1576071585.656324] [7595.364000] Stop
```

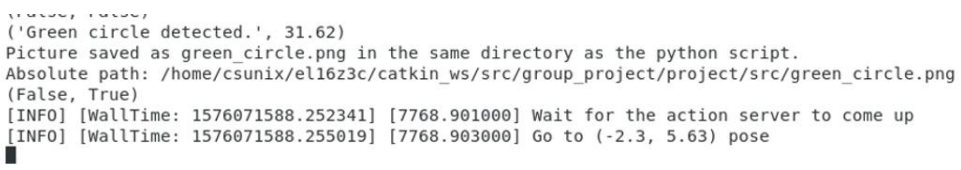
Run the move command, the robot first reaches the coordinates (0, 0), then come to (-2.83, -1.51), and finally, the terminal reports that robot reaches the designated position.

Green room detection testing

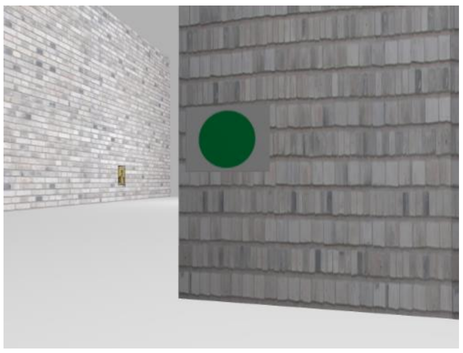
The main programme instructs the robot to go to the entrance of the first room, and slowly rotating itself while performing circle detection. As seen from the terminal output, after “robot reaches the designated position”, it started circle detection. The following output (False, False) means neither a red circle nor green circle is insight. When the robot reached the correct angle, it found the red circle and output (‘Red circle detected.’, 23.47), where the number 23.47 means the time elapsed since the script started execution.

The robot then decided to go to the entrance of the second room, which can be seen from the output “Go to (-1.43, 1.15) pose.”

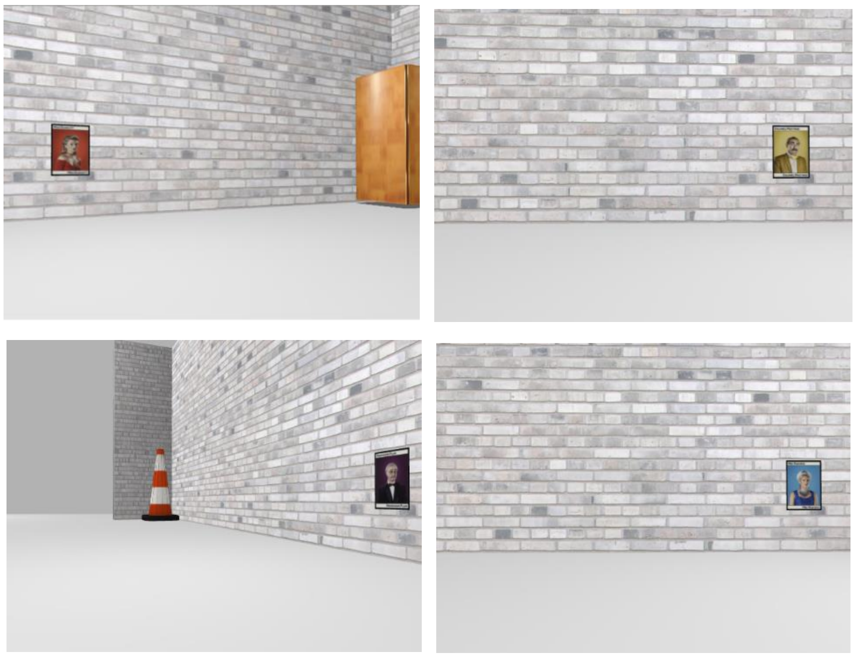
After reaching the entrance of the second room, the robot started rotating while performing circle detection. After a few (False, False), the robot detected the green circle and output (‘Green circle detected.’, 31.62), following is a message “Picture saved as green_circle.png in the same directory as the python script. Absolute path: ../green_circle.png”. The green circle picture is attached below. Then the robot decided to go into the centre point of the second room, which is seen from the following output “Go to (-2.3, 5.63) pose”



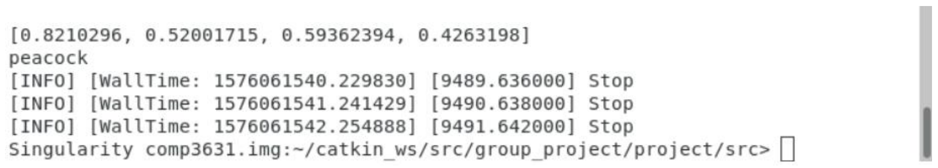
```
('Green circle detected.', 31.62)
Picture saved as green_circle.png in the same directory as the python script.
Absolute path: /home/csunix/el16z3c/catkin_ws/src/group_project/project/src/green_circle.png
(False, True)
[INFO] [WallTime: 1576071588.252341] [7768.901000] Wait for the action server to come up
[INFO] [WallTime: 1576071588.255019] [7768.903000] Go to (-2.3, 5.63) pose
```



Matching template testing



We changed the character images in the environment and adjusted the picture recognition threshold based on the results. After multiple test comparisons, we chose **0.65** as the matching threshold. When the suitability has exceeded the threshold, we consider that the character is found, and the character represented by the template with the highest match is the character name. As shown in the following figure, when the Cluedo character is detected, it will get four matching rates, corresponding to the template of the four characters. The first template has the highest matching degree, so the character name is considered 'peacock' here. After that, the program will stop and then automatically output image and character name.



```
[0.8210296, 0.52001715, 0.59362394, 0.4263198]
peacock
[INFO] [WallTime: 1576061540.229830] [9489.636000] Stop
[INFO] [WallTime: 1576061541.241429] [9490.638000] Stop
[INFO] [WallTime: 1576061542.254888] [9491.642000] Stop
Singularity comp3631.img:~/catkin_ws/src/group_project/project/src> 
```

Path Strategy testing

As mentioned above, after the robot enters the room, the robot will take a certain path strategy to find the character. Therefore, the following tests are compared for path strategy one and path strategy 2. We tested the performance ten times in two different environments, and the results are as follows.

Original Environment:

	Average time	Find character	Character accuracy
Strategy 1	1m36s	10/10	10/10
Strategy 2	2m28s	10/10	10/10

New Environment:

	Average time	Find character	Character accuracy
Strategy 1	5m7s	2/10	2/2
Strategy 2	3m42s	10/10	10/10

The results show that in a simple environment without more obstacles, both strategies can accurately find and identify the character, and strategy 1 takes less time. However, in a complex environment, strategy 1 seems difficult to find the location of the character, while strategy 2 is more efficient, it can accurately find the character in a short time. Therefore, in pursuit of the recognition rate, we adopted Strategy 2 as the path strategy in the final version.

Final version testing

video: https://youtu.be/zhTh0-xikAU

Real environment testing

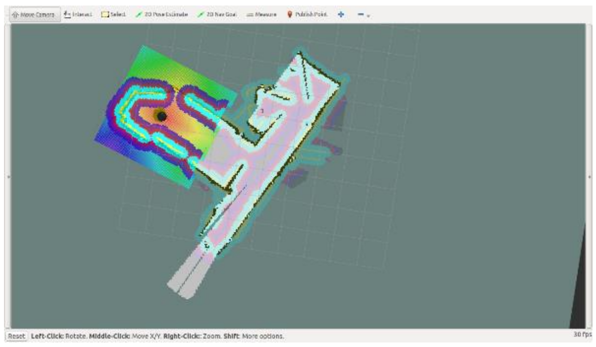
We tested our programme in the rooms provided in Logik space.

Before testing the programme on a real robot, we need to map the real environment and obtain the four coordinates (room1 entrance, room2 entrance, room1 centre, room2 centre). Meanwhile, we should make sure the robot can move to any given point on the map as instructed.

It's also important that our programme can identify green circle, red circle and Cluedo images in real environments without confusing them with other objects. Due to different lighting situation, sensor configuration and colour differences of an image on paper, etc., we need to re-adjust the filter boundaries by testing them against real images on paper.

After these preparations, our programme should be able to work in real environments.

The sensor map we created using the mapping function (robot is in the green room in the picture):

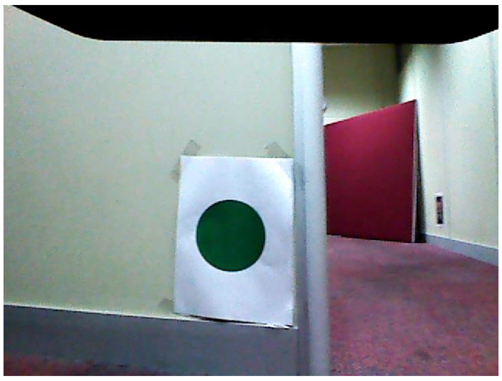

Move Camera
Interact
Select
2D Pose Estimate
2D Nav Goal
Measure
Publish Point
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click: Zoom. Shift: More options.
30 fps

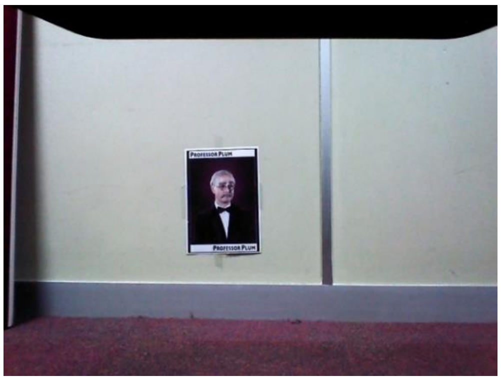
The four points were obtained by "publish points" function provided by Rviz, and validated by putting them in go_to_specific_point_on_map.py provided in Lab4

We ran the programme in the real environment and recorded the whole process. Please access our recording at https://youtu.be/hChltdCDT2w

The programme successfully identified the red circle at the entrance of the first room and then decided to go to the entrance of the second room. After spinning for two rounds, the programme found the location of the green circle and decided to go to the centre point of the second room. At the centre of the second room, the programme managed to identify the Cluedo image and terminated execution.

Green circle captured at the entrance of the green room and Cluedo image captured after reaching the centre point of the green room:





The robot successfully identified the circles at the entrances, went to the correct room and captured the images which completely contains the green circle and Cluedo image. The running time was within a reasonable range. Therefore, the overall performance and accuracy were up to expectation.

III. Summary

The move function we used in the programme is mainly based on scripts provided in Lab4. It can reliably navigate the robot to any given point but there are also constraints, such as that an accurate map needs to be provided beforehand.

We also reused a lot of code from Lab3 for circle detection. An advantage is that the robot can perform detection at roughly 10 frames per second, which means that images can be processed at a reasonable speed such that the robot will not miss out images that possibly contains useful information.

We did not implement the function for the robot to move its base at the circle detection stage, instead we tested it in a lot of situations and adjusted the threshold such that the robot can recongise the circle from a wide range of distances and angles. That way the robot will work in most situations without making the programme more complicated.

Our path strategy is designed to move the robot in all directions in order to bypass obstructions in the line of sight. This strategy is made to be generic although does not make use of the actual environment. This way our programme can calculate a search path in all kinds of situations whereas the drawback is that it is not guaranteed to find a solution.