Robotics report

1 - Line Following

1.1 - Approaching the problem

To create the line following algorithm we first looked at different potential approaches towards the problem. The first approach we considered was using a Convolutional neural network. This would involve using a neural network to identify the rope in order for the robot to follow it. This would be effective as it means that the colour of the rope wouldn't matter since the Artificial Intelligence (AI) could identify any colour of rope. However this could cause issues with the robot losing the rope on the ground misidentifying other objects as rope.

A second solution is to use an edge detection algorithm and then apply various filters to isolate the rope. However, this is a very complicated solution and would be very slow to run as well as being prone to the same misidentifying problems as the AI approach. This approach does have the benefit of being applicable to any colour rope since it would be detecting the shape of the rope and not the colour.

Another approach is using an image processing pipeline to identify the rope by colour. This approach would be beneficial as it is fast to run and doesn't require any training unlike the AI methods. However, the drawback of this approach is it would not work on other colour ropes and has to be set to detect each colour individually as well as being affected by noise in the image. As a result, for this technique to be successful in any path-following environment, it would be essential to program the robot with the ability to recognise a wide range of colours and select and appropriate mask to create the final, thresholded image. However, for this project, it is only required that the robot can recognise a blue and yellow line. This is the method we decided to go with.

One final approach is to combine these approaches together. This would consist of image processing on the image along with edge detection. This new image could then be used as input for the AI to identify the rope. This would be a very effective strategy since it would identify the rope reliably and be less effected by noise. However this would be very slow and would also be prone to all the problems associated with each solution.

1.2 - Image Processing Pipeline

The image is first blurred to remove any noise in the image and reduce the error that would arise through detection on a noisy image. The blurred image is then converted to a Hue, Saturation, Value (HSV) colour representation. We then create a mask for the specified colour range in HSV. The colour range for blue is (90, 80, 25) to (105, 255,255) and the colour range for yellow is (15, 25, 25) to (50, 255,255). This mask is then applied to the image which removes all colours that aren't in this range. We then convert this image to

grayscale and finally perform thresholding on the image to leave a final image as a series of white pixels.

These conversions are done in the get yellow and get blue functions. This functions take an image as input and returns the results of this process.

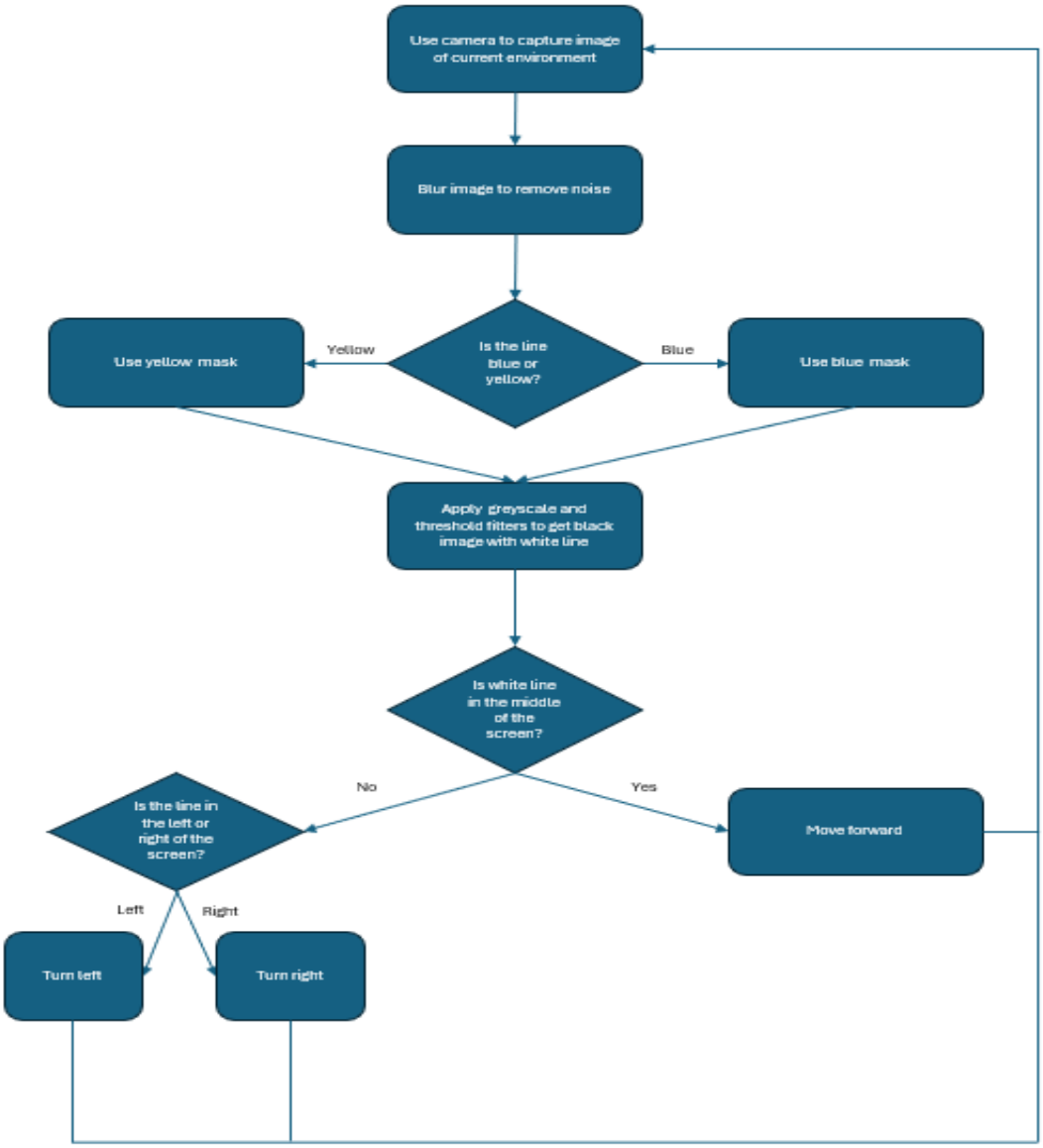
This image then has the top of it converted to black since the robot is only concerned with what is in front of it. And this removes any background objects that could influence the image and give the program an unnecessary amount of data to process.

1.3 - Moving the Robot

To have the robot follow the line we first need to identify what colour the line is. This is done by passing each frame into both of the image processors (yellow and blue) and then setting the current line colour to the one with the most white pixels. However, noise in the image can cause the robot to switch between thinking the rope is blue and thinning the rope is yellow. To reduce this issue a running poll of the last nine frames is taken and the majority vote for these frames is then used to set the line colour. This running poll is done within the camera class.

We then take the thresholded image from the output of the believed line colour and if there is more than a set number of white pixels in the middle of the image, then the robot moves forward. This can be viewed as the robot determining that there is a straight line of rope in front of it. If there are more than a set number of white pixels on the left of the image the robot will begin to turn left and the same is true for the white. This movement is done through a movement loop in which the state of the robot’s current view is re-evaluated at regular intervals to give an up-to-date model of the environment.

1.4 – Program Flowchart



1.5 – Testing the robot

To test the robot we first test to see if the robot is identifying the rope as the correct colour. This can be seen by looking at the image output of the image processing pipeline. If the image is a solid white line then the robot has correctly identified and converted the input image. One of the main challenges that arose from this section is getting the image processing system to ignore any noise or objects that weren't the line. For example, when detecting the blue line, the thresholding algorithm had to be very accurately tuned so that it didn't mistake the floor for a line. Additionally, in sunny environments, the image returned after processing contained numerous patches of white where the sun has been mistaken for the yellow line. For this case, the gaussian filter used for blurring had to be intensified,

followed by a change in the accepted values of the yellow mask. This ensured that the whiter light of the sun was ignored, leaving only the line in the final image.

We then test whether the robot can adapt to the change of line colour by picking it up and moving it to the other line. If it quickly changes the running poll is updating correctly as it should update within the 9 frame length of the poll.

We then test if the robot can follow and turn with the line. This is done by observing the movement. If the movement is too jittery tit means that the robot is trying to turn too rapidly or is trying to follow a path that is too narrow. Furthermore, the tests included picking the robot up to slightly or greatly reduce it's current positioning on the line. This was done to determine the robot's ability to adjust by taking in its current environment and turning so that it can travel forwards again. This also examined the ways in which the robot performs when traveling around the path in the other direction to ensure it can work well with a wide range of different turn angles.

1.6 – Results

For the first pipeline test the can reasonably reliably identify the rope in the image. This can be seen in the output to the right. This could be improved by narrowing down the exact colour needed for the mask. This would allow the robot to better identify the whole rope and would remove any excess noise.

When it comes to the colour changing test, the robot reliably changes colour in a short amount of time. This is a result of the running poll. Using a running poll allows the robot to update to view the line colour in real time. One way to improve the time it takes to transfer from one line to another is to reduce the size of the poll. This would allow the robot to look at less frames to change the colour and thus speed up changeover time. However the issue with this is while it would improve the changeover time, it would also introduce more uncertainty in the line identification process. This could be mitigate by improving the image processing pipeline to more accurately identify the colour of the rope.

The result of the final test of the robot's movement is that the robot can move around the track at a reasonably fast rate. This is good, however, the robot will often stop and turn left or right slightly while on the rope. This is because when the robot turns at the corners it does not give an exact turn. This leads to the robot being slightly misaligned to the rope. This could be mitigated by adding in a new option in the movement process where instead of just left right and forward, the robot could also move forward left and forward right where the turn is gradual. This would allow the robot to fix misalignment issues while it is moving instead of having to stop and adjust.

1.7 – Conclusion

Overall, the tests performed in this report successfully display the robot's ability to follow a path by utilising image processing and computer vision techniques. By creating functionality to transform an input image from the RGBD sensor, the robot was able to differentiate between the yellow and blue line and create a binary image in which the only white pixels were those of the line. As an improvement, this section could be extended so that the robot can recognise any colour line to create the thresholded image, making it a better general model in a real- world environment. However, this is beyond the scope of this project. Following, it was shown that the robot has the capability to recognise the state of it's environment in terms of where the line is in relation to its position. More specifically, it was able to determine whether to go forward to continue following the path, or if a change of direction was needed to ensure it was orientated correctly before proceeding.

In depth testing of this system proved that the task was successfully accomplished, providing evidence that all necessary functionality was implemented to a high degree of accuracy.

2. Human Following Task

2.1. Detection Selection

2.1.1. Design

Each Human 'target' is defined using the pre-trained model as discussed later, the result of these definitions are captured in an image using a bounding box. This bounding box is crucial for the design and implementation for the given task, as it relays information that the target has moved resulting in the robot to respond.

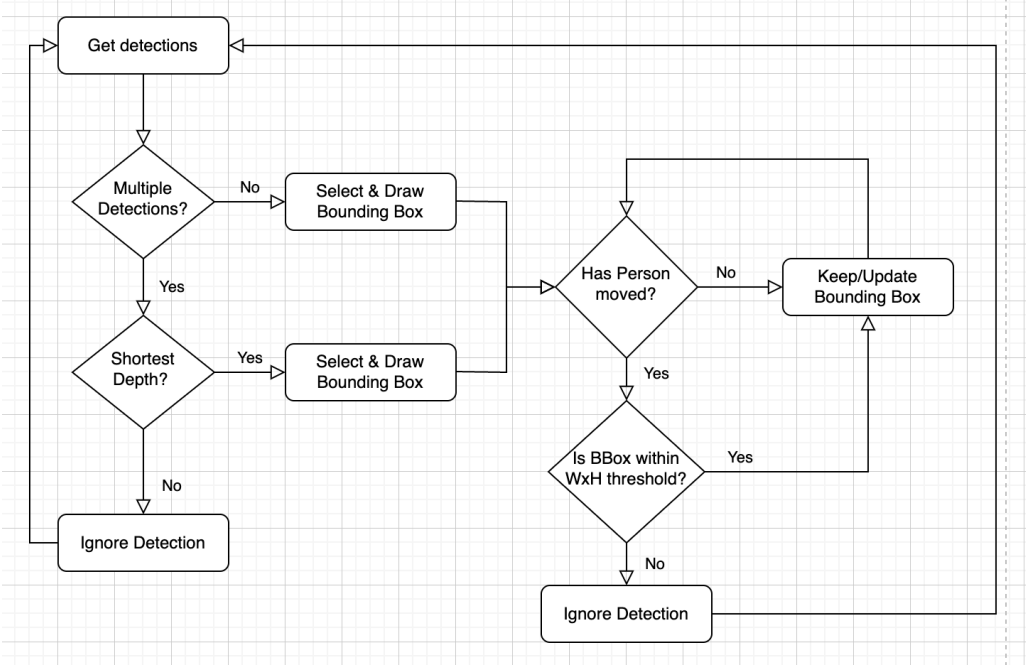
The main problem regarding target detection is the process of selecting the initial target. A solution to this issue would be to implement the robot's depth camera. This method includes choosing the 'target' that has the shortest (central depth inside the bounding box).

This central depth is calculated by restricting the bounding box to a percentage of the original one generated. Within this bounding box it then passes the x and y coordinates of the bounding box with the height and widths values. This then is used to splice all the depth values of the depth value array of the image. This splice is from the starting x and y coordinate to the width and height value. This should pull a 2d array which is then iterated through until the centre co-ordinate is found, returns the central depth value of the specified coordinates of the bounding box for a detection.

After this problem, the next issue is determining when the detected 'target' has moved either to the left, right, towards or away from the robot. As this information will allow the robot to move in its required direction. This could firstly be done by defining a threshold which determines whether the next detection is the same target or another Human. This threshold would take into account the width, height and depth of the bounding box. Then examine whether

the same target has moved by calculating whether the new detection is within the stored ‘target’ threshold.

The below Flow Chart describes the process of how each detection is defined and the results are given by drawing/ignoring or updating the bounding box for use in the movement section of the robot.



2.1.2. Functions

As touched upon above, each detection is defined using the pre-trained convolutional neural network model MobileNet. MobileNet is used because of its low latency processing which is perfect for use in an external device such as a robot.

The pretrained model takes an image as input along with an integer label which represents the type of object that the model should detect. In this case, a person would have the label ‘1’, and after processing would return a dictionary of arrays with both; positional axis of the detected person and the confidence that the detection is a person. These positional axis are then translated using the height and the width of the whole image for an easier representation when designing and implementing bounding boxes for each detection.

Initially each bounding box is drawn using the cv2 library’s draw() function but for the specified problem, we need only the selected target’s bounding box for further processing. However, during some testing, the most common error occurs when the bounding boxes are trying to be defined when there are no initial detections. By employing an initial boolean flag that triggers when a detection is found was the fix to this issue, and when this flag is true the main loop proceeds to be executed.

Using the bounding box as discussed above, most of the information inside the bounding box is not usable, i.e. either there is too much leeway on empty space (area that isn’t human). Hence, by resizing the bounding box to be more central around the human, gives us the opportunity to implement the central depth value of the “zoomed in” boundaries.

The resizing of the bounding box was done geometrically by scaling each diagonal bounding box coordinate by a factor of 0.5 (50% decrease). This can be seen in the following lines;

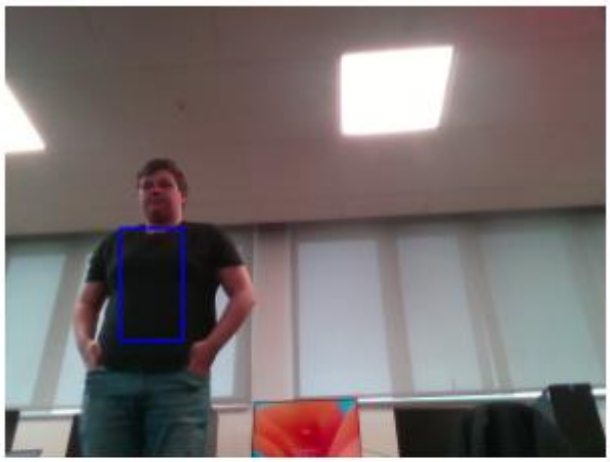
```
for det in matching_detections:
    new_bbox = det['bbox']
    for i in range (len(new_bbox)):
        new_bbox[i] = 0.5*new_bbox[i]
    new_bbox[0] = new_bbox[0] + 0.5*(new_bbox[0] + new_bbox[2])
    new_bbox[1] = new_bbox[1] + 0.5*(new_bbox[1] + new_bbox[3])
```

Further on from this scaling in the same “for det in matching_detections” loop, includes a conditional. This conditional compares each other detection with any new detections “new_bbox” to see whether the bounding box is within the width and height threshold (WxH Threshold). As discussed above, these lines of code stop any other people who are walking past to be the selected ‘target’.

After the scaled bounding box and selected target are defined. Using the depth camera’s depth values allowed us to determine how far away each detection was. Which is then used for processing the robot’s forward and backward movement.

2.1.3. Testing

As seen from the image below, the bounding box is scaled to a smaller square which covers all of the human. And the debug messages accompanying it shows the central depth of the bounding box and the next action of the robot when the bounding box (human) moves either to the left side or to the right.

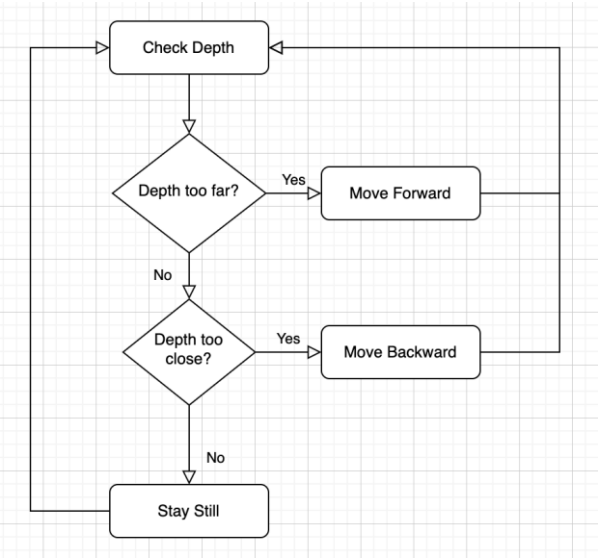


2.2. Robot Movement

2.2.1. Design

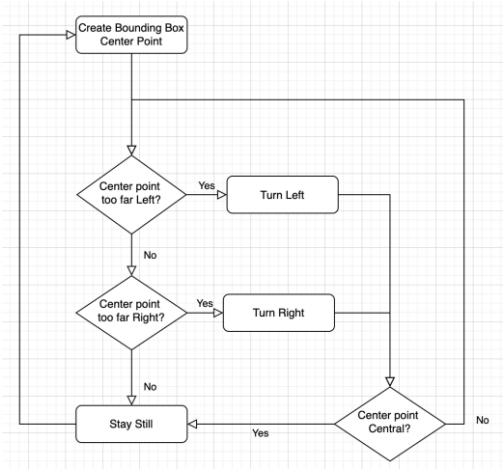
The specifications main priority regarding the robot's movement is that the robot follows the target at a 'safe distance', due to the vagueness in this we had decided that the robot's forward and backward movement would stabilise around a metre away from the target. Where, the robot being too close would result in its backward movement and vice versa.

Using the flowchart below, we can see how the robot processes whether the target is too close or too far. The starting box "Check Depth", takes the centre point of the target and compares this value with the "Too Far" and "Too Close" variables which will be discussed later. Then as seen with both of the decision boxes below, if any of these are true the robot will move forward or backwards depending on whether it is too far or too close to the target respectively.



For the robot to turn, we decided that the best was to detect which direction the bounding box was moving based on the midpoint of the horizontal lines (Centre Point) of the bounding box, then turn the robot in the same direction. The below flow chart shows how the robot would decide which direction to turn based on the parameters set in the code.

Turning Flowchart:



The terms “Too Far Left” and “Too Far Right” are based on thresholds that are set to 30% of either side of the image. The reason for not using a 50/50 threshold is because then the robot would keep jittering left and right as it would be highly unlikely that the ‘target’ would be perfectly in the centre.

2.2.2. Functions

As described above, below is the Turning function;

```
if ((bbox[0] + bbox[2]) * 0.5) > (width*0.7):
    robot.right(0.5)
elif ((bbox[0] + bbox[2]) * 0.5) < (width*0.3):
    robot.left(0.5)
```

This function is continuously executed in the main loop and flags the robot to turn depending on the ‘target’ horizontal movement. “((bbox[0] **+** bbox[2]) ***** 0.5)” takes the centre point of the horizontal bounding box line, this is because both bbox[0] and bbox[2] are defined to be the position of the minimum and maximum x coordinate respectively. And half of the addition of these values, give the centre point.

Getting the central depth value for use in the forward and backward movement are done as follows;

```
detections_depth = [camera.get_depth((0.1*int(width * (det["bbox"][0] + det["bbox"][2])) + 0.2*int(width
*  (det["bbox"][0] + det["bbox"][2]))) , (0.25*int(height * (det["bbox"][1] + det["bbox"][3])) +
0.125*int(height * (det["bbox"][1] + det["bbox"][3])))) for det in matching_detections]

if len(detections_depth) == 0:

    detections_depth = [0]

selected_index = detections_depth.index(min(detections_depth))
depth = detections_depth[selected_index]
```

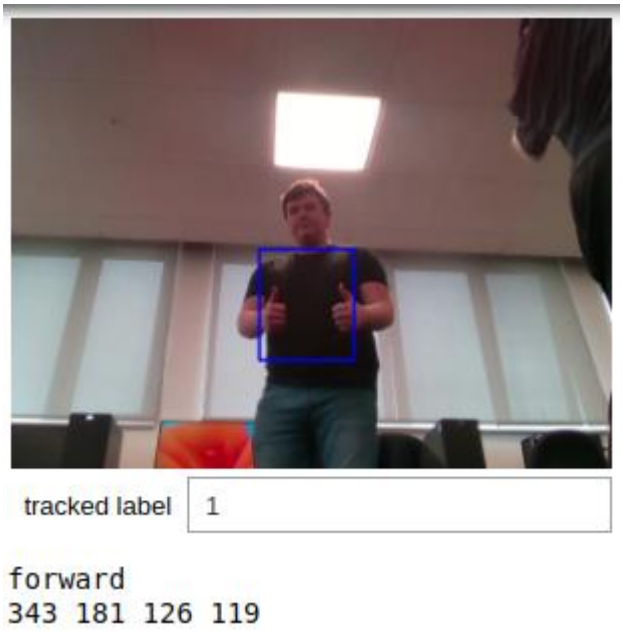
As explained earlier, the above block of code takes the central point of the bounding box's depth value inside the coordinates set by the resized bounding box. This value then can be used for the robot's movement.

After the robot is in line with the 'target' and the "target's" depth value has been set, the following function is then called, which is the Forward/Backward Function;

```
if depth < 700:
    robot.backward(0.4)
else:
    robot.forward(0.4)
```

Again, another threshold has been set on the depth to avoid the robot from jittering forwards and backwards if the target slightly moves in either direction. After this threshold the robot would move either forward toward the target or backwards away from the target.

2.2.3. Testing



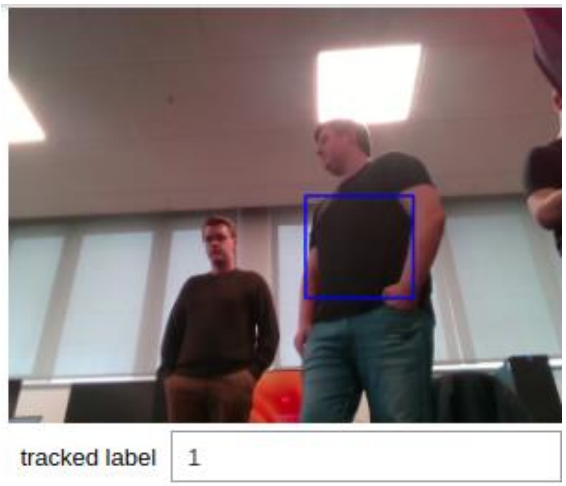
As seen from the image above we can see the resized bounding box along with the debug messages underneath that shows the robot's movement based on the predefined conditions. (e.g. too close – move backwards, too far - move forward, etc.)

In addition to these images, the robot performed well in both turning to follow the 'target' on the robot's x coordinates and moving forward and backward based on the central depth of the resized bounding box.

2.3. Human Following Test

2.3.1. Test variables

For this test, we have a screenshot of the bounding box's behaviour when faced with multiple background detections.



As seen above, when faced with multiple detections the target with the lowest central depth is detected. Due to this the robot was able to follow the 'target' defined with the bounding box above.

2.3.2. Conclusion

In conclusion, the robot is able to follow its target at a "safe distance". In accordance with this, the robot has also been tested when the target moved both to the side and backwards. This resulted in the robot turning to the human's position followed by a forward movement of the robot. In an ideal world, the robot should curve its forward movement to show fluidity in its actions.

2.3.3. Improvements

In order to improve this task, we would have liked to implement the earlier discussed function. This would be done using the robot.forward_left(x) and robot.forward_right(x) functions. In addition to this, we would have also liked to implement the speed of which the robot moves forward and backward. This would have been dependent on the robot's distance away from the target.