Assignment Two

MEC4406

Joshua Garry

U1107131

https://youtu.be/J7Bub6kBJvM

Aim.

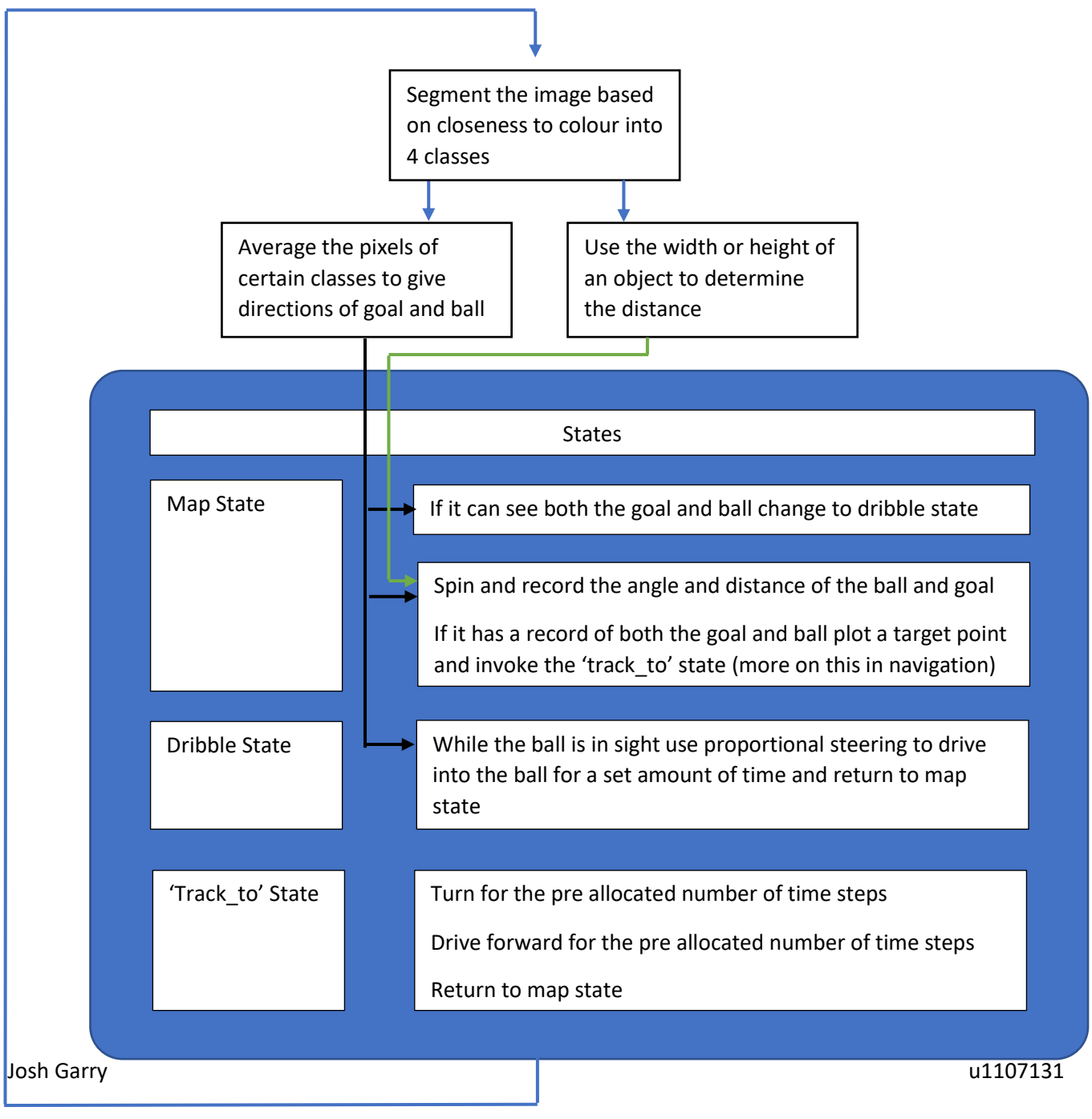
The aim was to build a simple navigation system for the robots. This would be an attack and goalie variant that would serve different purposes on the field. This was accomplished in part by designing an arena that was coloured in a way that makes image segmentation easier.

Hardware design choice and justification.

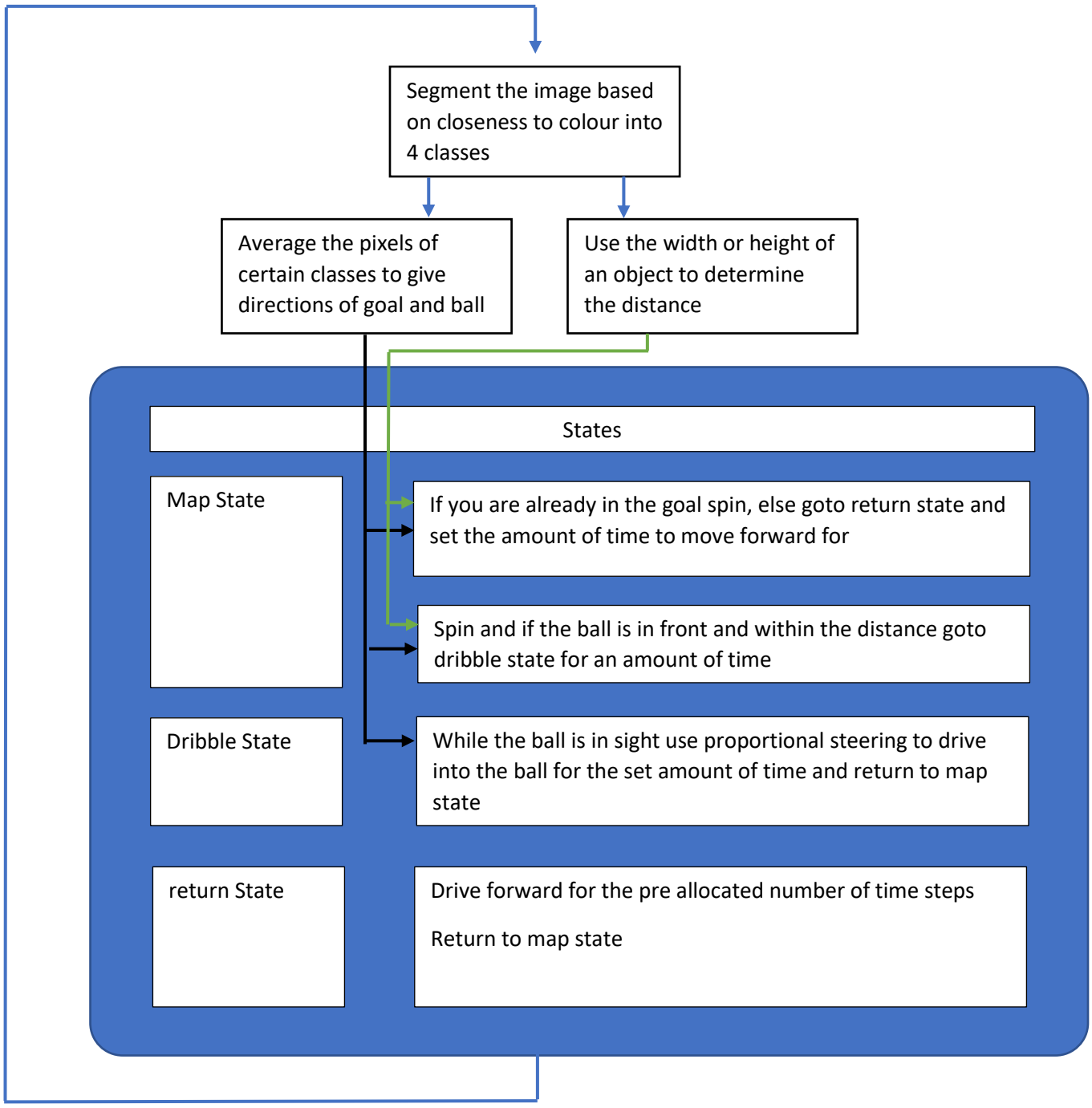
The e-puck robot was chosen as it has good documentation on the Webots website as well as a simple design. This was also useful as it has an array of sensors that are useful for tracking and object recognition including distance and gyroscopic sensors, three axis accelerometer and camera. Unfortunately this dose not have a compass and the control and navigation code had to be rethought of in order to work around this.

Flowchart of code and navigation system

The following is an iteration of the main control loop of the attacking robot:

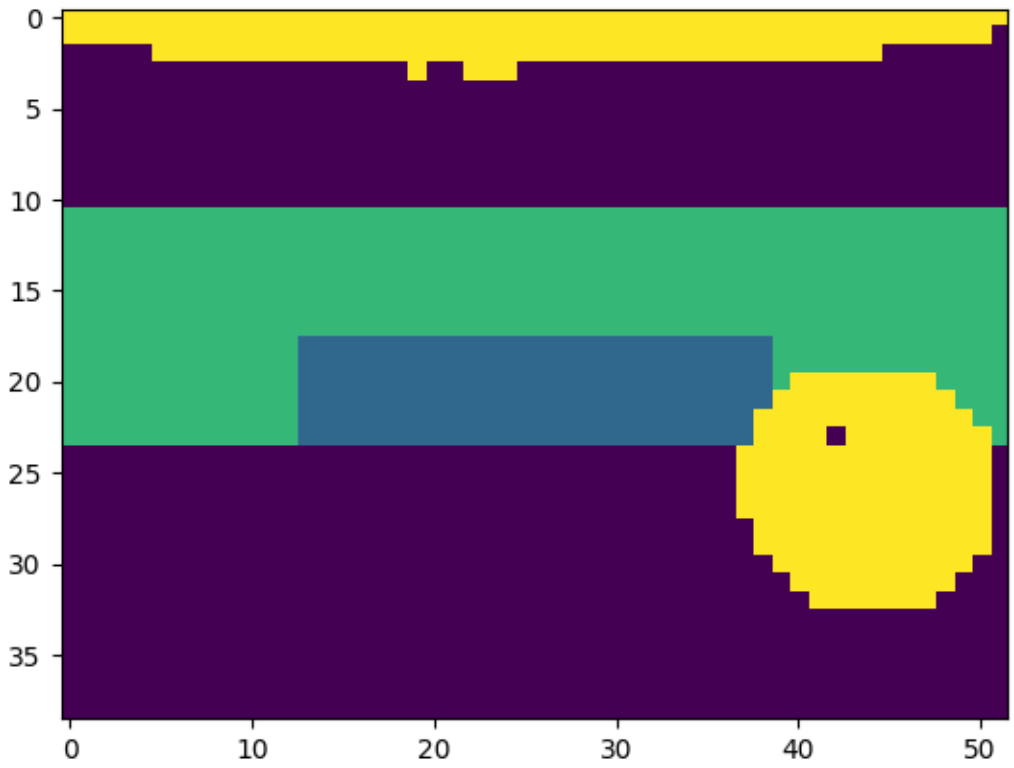


The control loop of the defending robot:

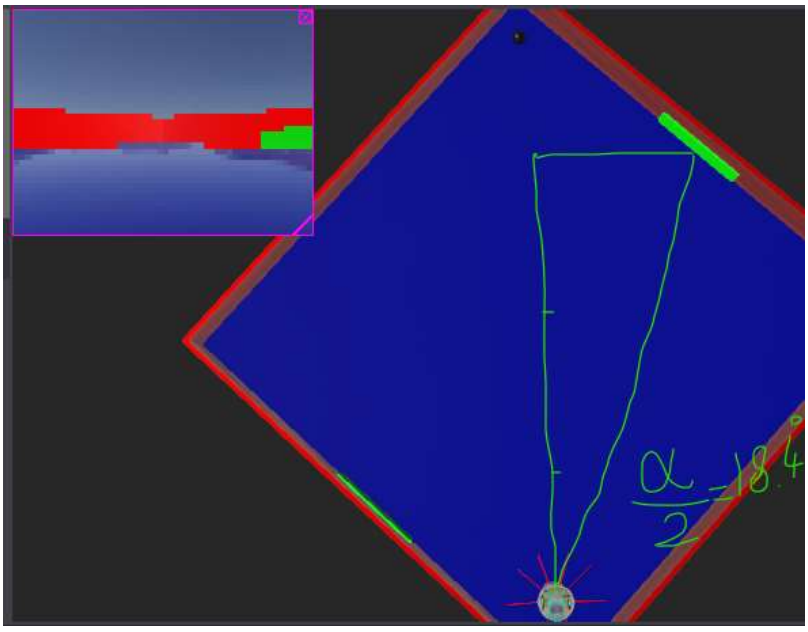


Navigation system

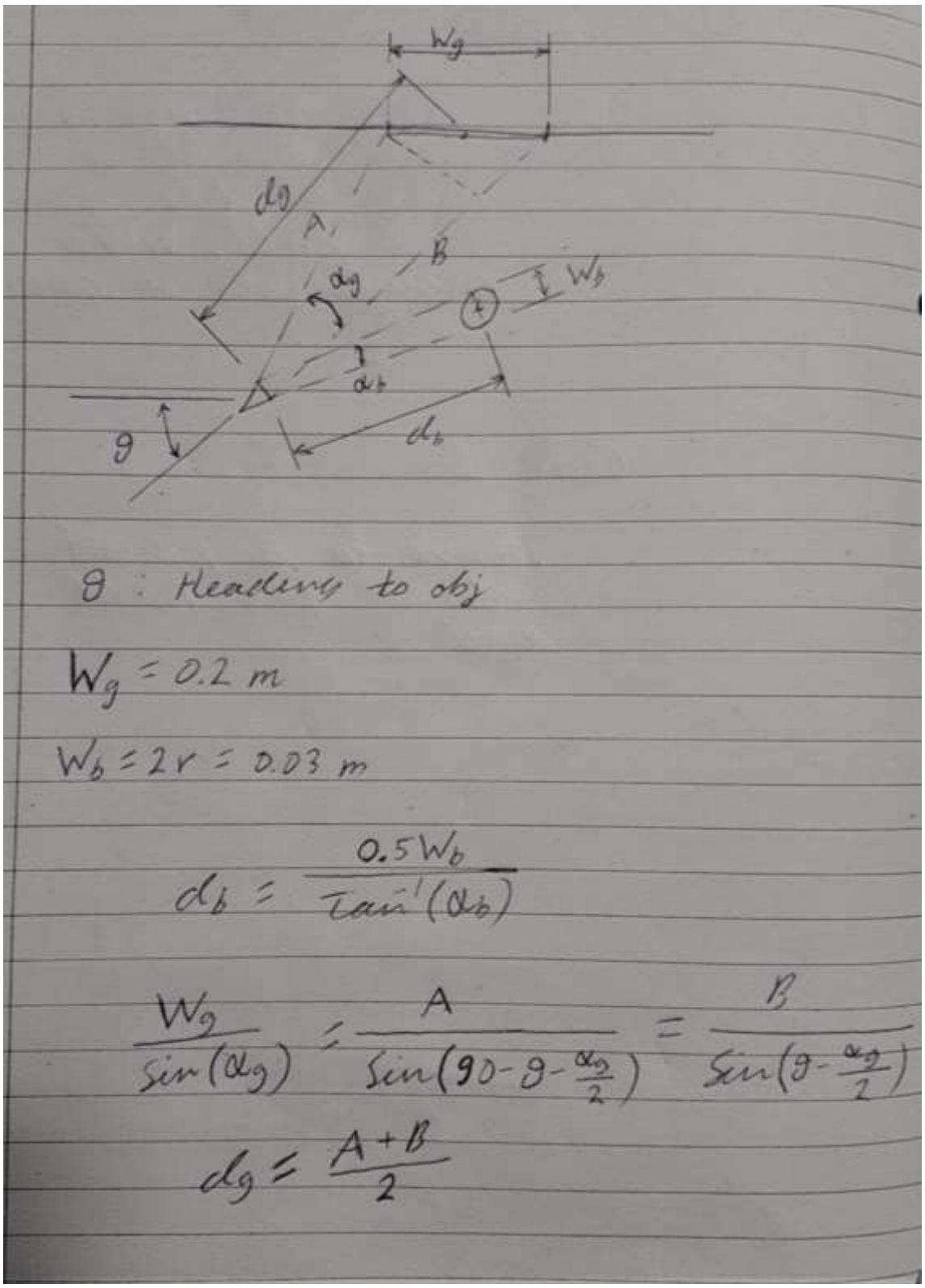
When the segmentation prosses is done the output image looks like this. This is made by subtracting 4 monotone images from the original image and seeing which produces the smallest value. This produces a high performance segmentation algorithm (about 10 times faster than when I tried with 4 nested for loops).



This image can then be used to find the angels of each of the objects in the frame by finding the average x values of each class and then using an approximation for conversion from pixel index to angle with the rough observation displayed below.



This can then be used to find the distances of the objects given there known width, for the ball, and height for the goal (Although initially I attempted to use the width for both witch yielded a bad result).



To then approximate a target position for the robot to move to, some geometry was done. Converting from the polar coordinates to cartesian with the robot as the origin and the goal as the y axis intercept the following could be deduced.

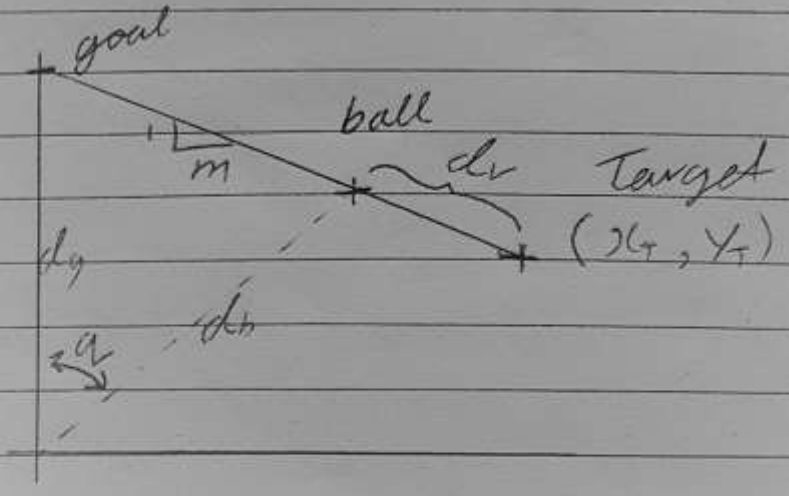
$$(x_g, y_g) = (0, d_g)$$

$$(x_b, y_b) = (\cos(q)\,d_b, \sin(q)\,d_b)$$

$$m = \frac{y_b - y_g}{x_b - y_g}$$

$$(x_T, y_T) = \left(x_b + \frac{d_v}{\sqrt{m^2+1}}, y_b + \frac{m d_v}{\sqrt{m^2+1}}\right)$$

$$d_T = \sqrt{x_T^2 + y_T^2}$$



Once the target was found the (xt, yt) point was converted back to polar coordinates for the robot to turn to and then drive forward to. For a Desmos plotting of the math:

https://www.desmos.com/calculator/cq4kl7zug0

Reflection – what went wrong and why, how might you look to improve the system overall.

One major hold up was the confusing for incorrect documentation in the Webots Camera module where the method GetImageArray() is claimed to return a set of python nested lists corresponding to the pixels of the image. However this function would only return an image where all rows were the first row. A Stack exchange post fortunately shared a workaround where the image buffer object could be interpreted as a NumPy matrix.

Another major hold up that made me rethink the entire control system was the lack of a compass on the e-puck robot.