Department of Mechanical, Aerospace and Industrial Engineering

FINAL PROJECT REPORT

MECH 472/6631

INDUSTRIAL AUTOMATION

SUBMITTED TO:
Prof. Brandon Gordon

SUBMITTED BY**:**

STUDENT NAME	STUDENT ID
Harsh Sanmotra	40191913
Jagsir Singh	40171231
Deval Suthar	40204778
Parth Desai	40191092

Contribution to the project:

Harsh Sanmotra (40191913): Contributed towards image processing and centroid tracking and overall code compilation.

Jagsir Singh (40171231): Planning and development of complete motion. Planning and development of attack and defence sequence with designing of motion functions along with overall code compilation.

Deval Suthar (40204778): Development and design of complete motion. Attack and defence sequence planning along with developing supporting motion functions and overall report documentation.

Parth Desai (40191092): General logic development for various scenarios.

Table of Contents

Introduction

We aimed at developing fully autonomous robots capable of executing both attack and defense sequences. Our C++ code is developed such that it can perform vision and motion control task at the same time. We focused on precision with our image processing and motion control programs, as we were aware that our robot might get evaluated against the robot prepared by the other teams. In order to achieve precision, we tried various approaches for both image vision as well as attack & defence stratergies. Our robot has two salient features, first is it can identify the obstacles regardless of its color and second one is that it will never leave the screeen in ideal conditions.

Image Processing

In order to handle image processing we made few functions. First and foremost, RGB image is processed using image_prep() function, which applies a sequence of point wise processing function. After this the processed image is labelled using label_image() function and finally the centroids are traced using centroid() function, which not only identifies that obstacle and robot based on the colour but also based on the size for the labelled image. In order to find the size of the labelled image we made a seprate function called label_size() that is called inside the centroid() function.

Attack Sequence

After obtaining centroids information of opponent and obstacles, the robot checks for closest path to the opponent (defence) along with the required angle calculation that are done using triangle_angles(). Afterwards, attacking robot aligns itself towards the opponent that proceed to move towards opponent, however it also checks for the condition wheather there is an obstacle between both robots, and if the condition satisfies attacking robot avoids the obstacle using obstacle_avoid() function.

Defence Sequence

After obtaining centroids information of opponent and obstacles, the defending robot searches for the closest obstacle to hide behind. For this it makes use of traingle_angles_d() function which calculates some required angles and distances. After this make use of complete_defence() function which provides a point for defending robot to follow. This point is obtained by using following parametric equation.

$$px = ((cos(theta\_obs\_a)) * (d\_obs\_b + 150)) + y\_ic;$$

$$py = ((sin(theta\_obs\_a)) * (d\_obs\_b + 150)) + y\_jc;$$

After aligning and moving towards the point, the defending robot keeps a safe distance from the obstacle and it maintains and orientation which prevent it from being in the line of attack of attacking robot.

Obstacle Avoidance

Similar to defence, our entire obstacle avoidance functionality is based on following 4 parametric equiation. We prepared two function obstacle_avoid() and obstacle_avoid_defence() which make use of these parametric equation to obtain two separate point in real time, and one point is selected by robot to continously follow according to suitable condition.

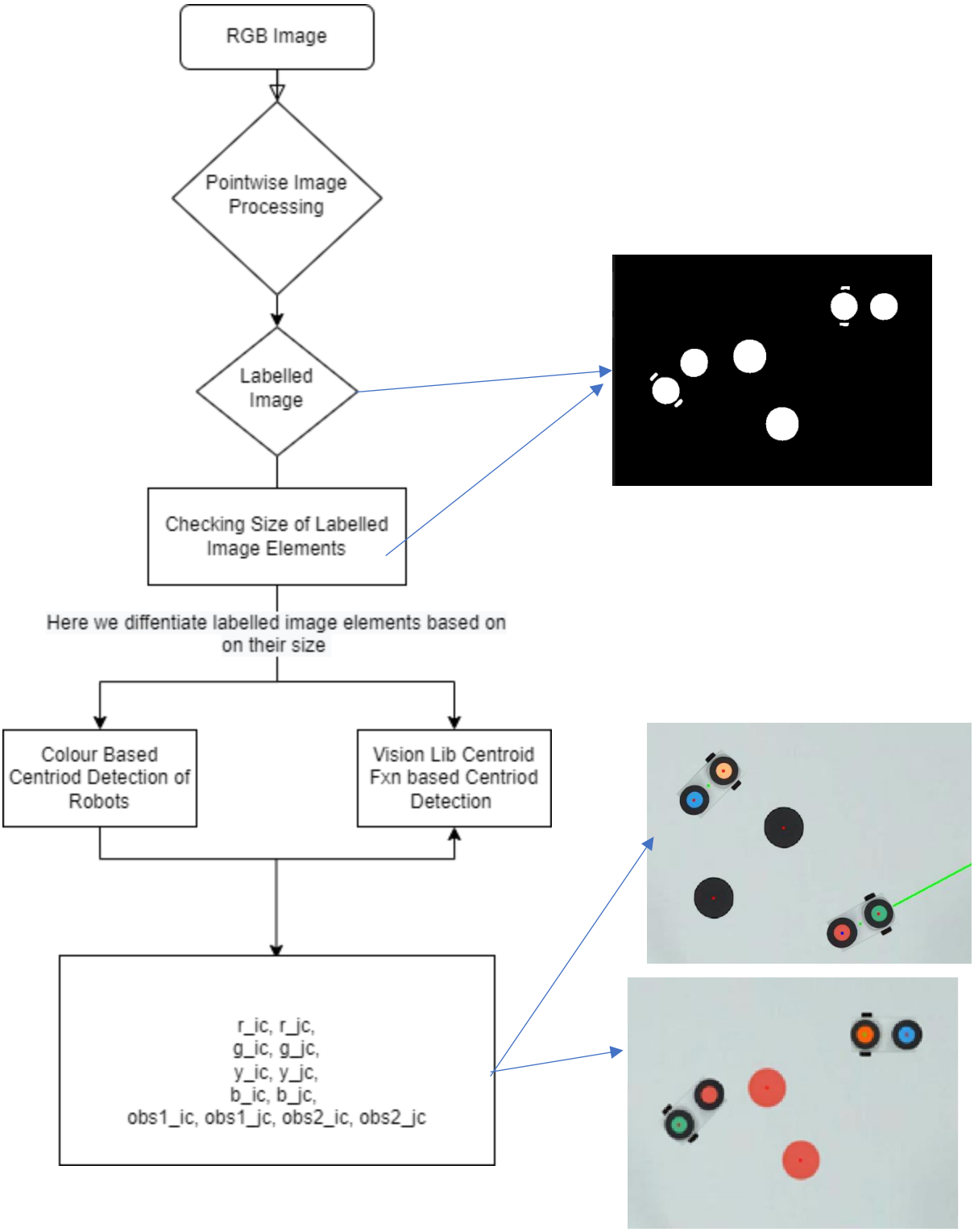
$$p1x = (d\_obs\_a / 6) * cos(theta\_obs) + c * cos((theta\_obs + 1.5708)) + g\_ic;$$

$$p1y = (d\_obs\_a / 6) * sin(theta\_obs) + c * sin((theta\_obs + 1.5708)) + g\_jc;$$

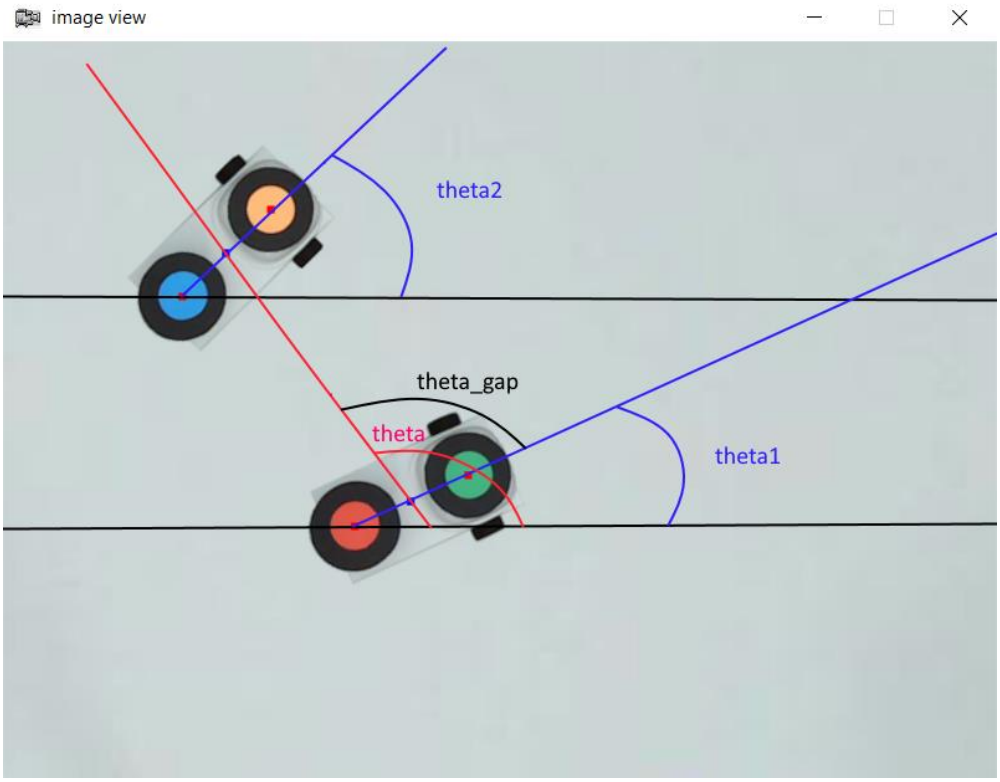
$$p2x = (d\_obs\_a / 6) * cos(theta\_obs) + c * cos(theta\_obs + 4.71239) + g\_ic;$$

$$p2y = (d\_obs\_a / 6) * sin(theta\_obs) + c * sin(theta\_obs + 4.71239) + g\_jc;$$

Flow Chart of vision



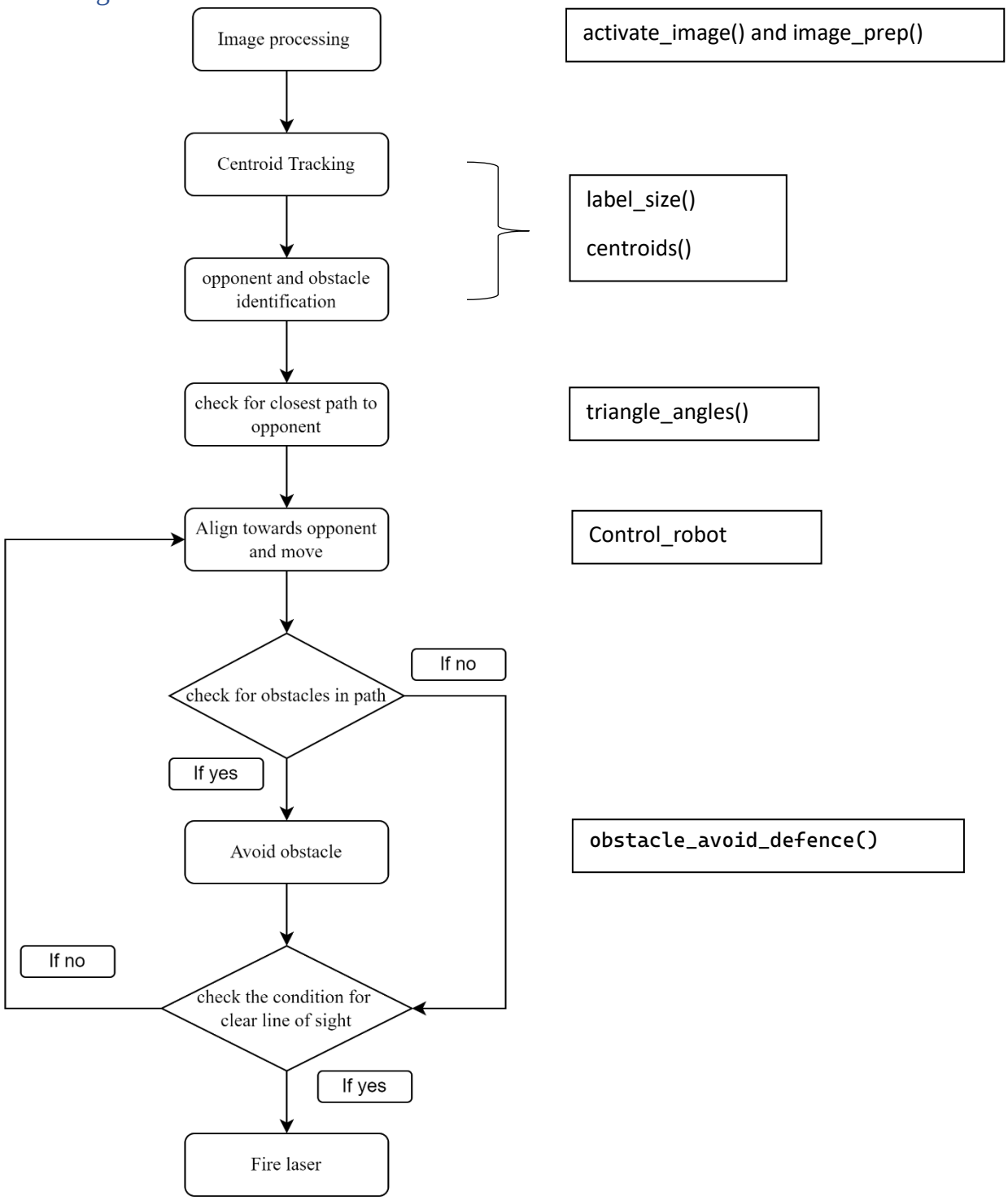
Angle Calculation




```
Select C:\Users\15145\Desktop\6631 Report\Attack_vs_Deffence\Attack_vs_Deffence\Debug\program.exe
Robot A angle = 308.436 Robot B angle = 230.098 Angle to opponent = -170.614
Robot A angle = 305.868 Robot B angle = 233.272 Angle to opponent = -167.764
Robot A angle = 302.69  Robot B angle = 236.703 Angle to opponent = -164.195
Robot A angle = 300.1   Robot B angle = 239.278 Angle to opponent = -161.056
Robot A angle = 296.885 Robot B angle = 242.471 Angle to opponent = -157.625
Robot A angle = 293.669 Robot B angle = 245.671 Angle to opponent = -153.982
Robot A angle = 291.138 Robot B angle = 247.929 Angle to opponent = -150.867
Robot A angle = 287.985 Robot B angle = 250.877 Angle to opponent = -147.366
Robot A angle = 284.379 Robot B angle = 254.051 Angle to opponent = -143.446
Robot A angle = 281.447 Robot B angle = 256.839 Angle to opponent = -140.177
Robot A angle = 278.635 Robot B angle = 259.948 Angle to opponent = -136.972
Robot A angle = 276.416 Robot B angle = 262.783 Angle to opponent = -134.26
Robot A angle = 272.86  Robot B angle = 265.654 Angle to opponent = -130.279
Robot A angle = 269.997 Robot B angle = 268.565 Angle to opponent = -127.046
Robot A angle = 267.135 Robot B angle = 271.429 Angle to opponent = -123.759
Robot A angle = 263.578 Robot B angle = 274.341 Angle to opponent = -124.814
Robot A angle = 260.652 Robot B angle = 277.212 Angle to opponent = -121.701
Robot A angle = 257.853 Robot B angle = 280.173 Angle to opponent = -118.664
Robot A angle = 255.437 Robot B angle = 280.173 Angle to opponent = -116.018
Robot A angle = 252.009 Robot B angle = 280.173 Angle to opponent = -112.359
Robot A angle = 248.857 Robot B angle = 280.173 Angle to opponent = -108.879
Robot A angle = 245.962 Robot B angle = 277.212 Angle to opponent = -106.224
Robot A angle = 243.432 Robot B angle = 274.341 Angle to opponent = -103.786
Robot A angle = 240.252 Robot B angle = 271.429 Angle to opponent = -100.833
Robot A angle = 237.091 Robot B angle = 268.565 Angle to opponent = -97.862
Robot A angle = 234.126 Robot B angle = 265.654 Angle to opponent = -95.0478
Robot A angle = 231.005 Robot B angle = 262.783 Angle to opponent = -92.2146
Robot A angle = 228.541 Robot B angle = 259.948 Angle to opponent = -89.8471
Robot A angle = 225.503 Robot B angle = 256.839 Angle to opponent = -86.7958
Robot A angle = 221.983 Robot B angle = 254.051 Angle to opponent = -83.5695
```

Attack sequence

Attack algorithm



The description of attack sequence

1. position_angle()

   After receiving locations of centroids from the centroids() function the information is passed to position_angle() function which calculates overall centroids of Robot A (x1, y1) and Robot B (x2, y2) with their respective angles (theta1) and (theta2) from the x-axis. This information is then passed to the triangle_angles() function.

2. triangle_angles()

   This function calculates angles between attacking robot A and defending robot B with respect to the closest centroid of obstacle available to the robots. The angles checked are used to determine if there is an obstacle present between attacking and defending robot. Information from this function is supplied to obstacle_avoid() function.

3. obstacle_avoid()

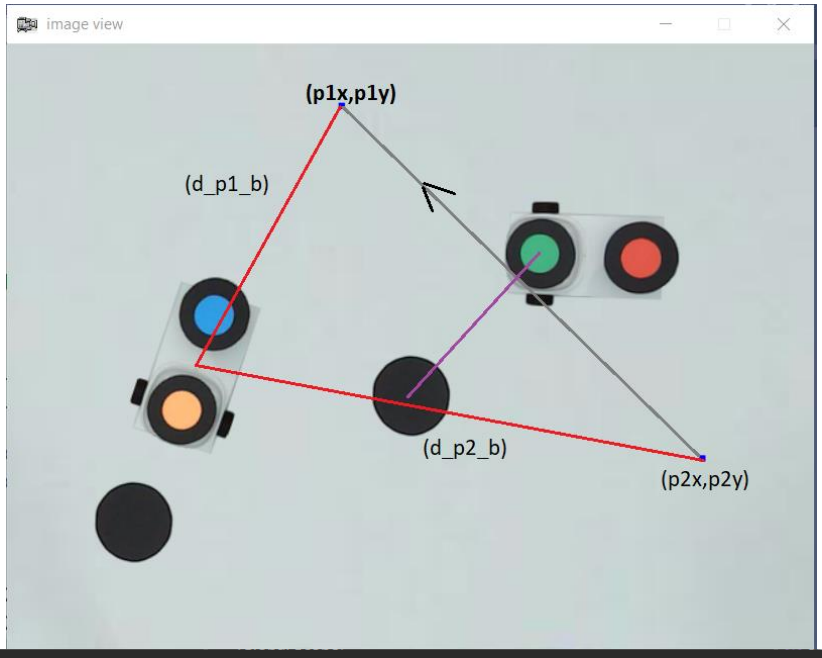
   If there an obstacle present between attacking and defending robot, this function servers the purpose of guiding the attacking robot around the obstacle without collision making it possible for attacking robot to have clear line of sight to fire laser. For the function parametric equations were set which provide 2 points (p1 and p2). The if else statements in the function decides which point the attacking robot needs to follow in order to move closer to the defending robot and fire laser. (c is the distance obtained by using trial and error method)

```
p1x = (d_obs_a / 6) * cos(theta_obs) + c * cos((theta_obs + 1.5708)) + g_ic;
p1y = (d_obs_a / 6) * sin(theta_obs) + c * sin((theta_obs + 1.5708)) + g_jc;
p2x = (d_obs_a / 6) * cos(theta_obs) + c * cos(theta_obs + 4.71239) + g_ic;
p2y = (d_obs_a / 6) * sin(theta_obs) + c * sin(theta_obs + 4.71239) + g_jc;
```

4. control_robot()

Once the Robot A reaches to the point provided by obstacle_avoid(), this function activates which mainly aligns the attacking robot A towards the opponent robot B and follows until the robot A reaches to a distance from where the laser can be fired accurately.

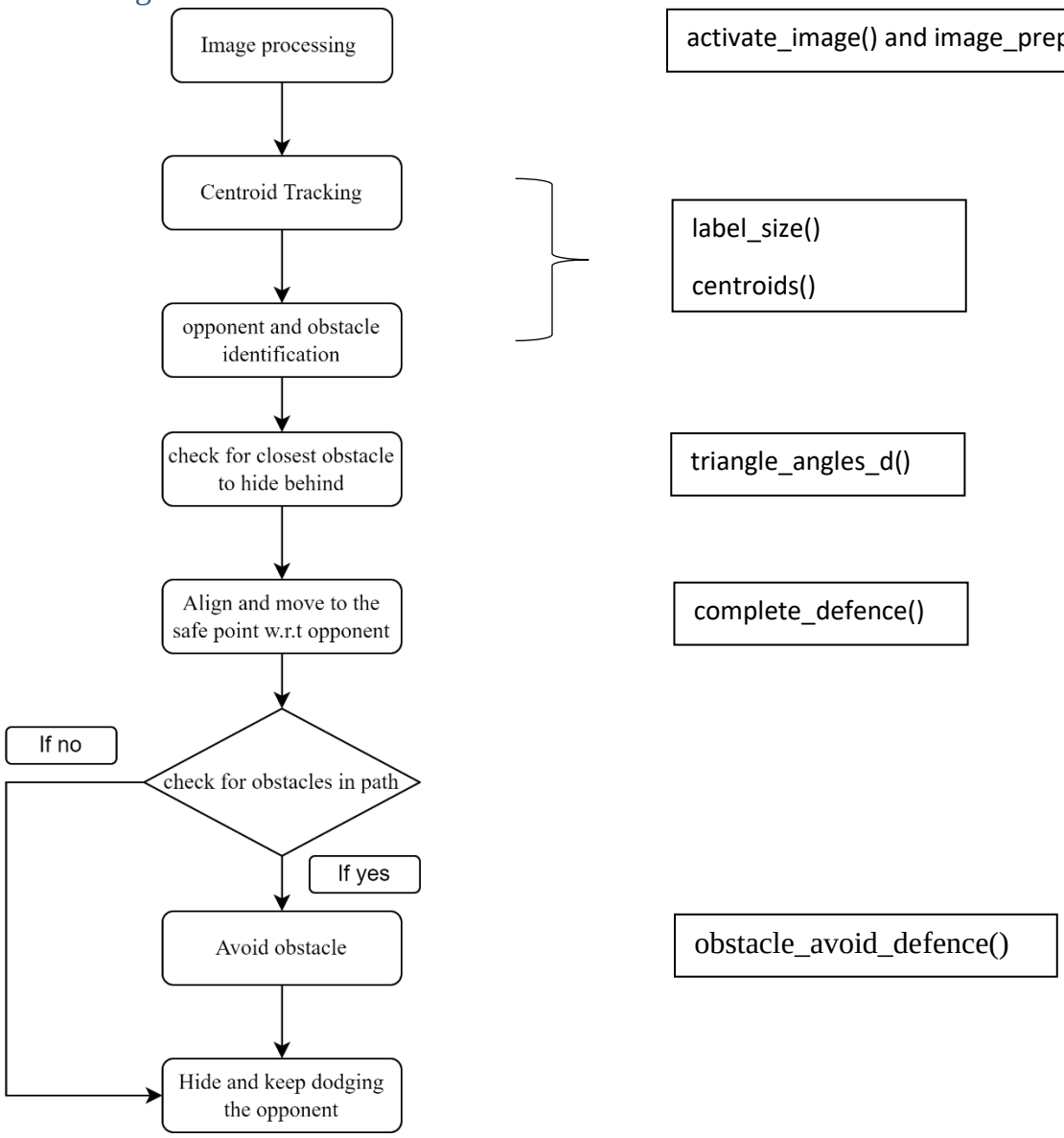
Output results




```
Select C:\Users\15145\Desktop\6631 Report\Attack_vs_Deffence\Attack_vs_Deffence\Debug\program.exe
================================================================================
1st point to avoid obstacle =   p1x = 93.8898   p1y = 72.0217
2nd point to avoid obstacle =   p2x = 93.8898   p2y = 72.0217
distance from 1st point to opponent = 281.609
distance from 2nd point to opponent = 470.806
================================================================================
================================================================================
1st point to avoid obstacle =   p1x = 93.8384   p1y = 72.3175
2nd point to avoid obstacle =   p2x = 93.8384   p2y = 72.3175
distance from 1st point to opponent = 282.306
distance from 2nd point to opponent = 471.01
================================================================================
================================================================================
1st point to avoid obstacle =   p1x = 90.6712   p1y = 76.8589
2nd point to avoid obstacle =   p2x = 90.6712   p2y = 76.8589
distance from 1st point to opponent = 277.679
distance from 2nd point to opponent = 470.991
================================================================================
================================================================================
1st point to avoid obstacle =   p1x = 87.5846   p1y = 81.4377
2nd point to avoid obstacle =   p2x = 87.5846   p2y = 81.4377
distance from 1st point to opponent = 275.259
distance from 2nd point to opponent = 470.912
================================================================================
================================================================================
1st point to avoid obstacle =   p1x = 87.5471   p1y = 81.7484
2nd point to avoid obstacle =   p2x = 87.5471   p2y = 81.7484
distance from 1st point to opponent = 273.742
distance from 2nd point to opponent = 471.214
================================================================================
```

Defence sequence

Defence algorithm



The description of defence sequence

1. position_angle():

   Similar to the attack sequence, defending robot B receives the overall centroids of robot A and robot B with their respective angles theta1 and theta2. This information is then passed to the triangle_angle_d() function.

2. triangle_angle_d():

   This function searches for the closest obstacle for defending robot to hide behind. Additionally, it also performs required calculations to measure angles which determines whether an obstacle is in between the path of robot B.

3. obstacle_avoid_defence():

   This function checks for the conditions in which defending robot B might collide to the obstacles. To avoid the obstacles according to the situation, using same parametric equation robot B selects suitable point to align itself and move towards it. The point is set up in such a way that robot does not collide with obstacle while following it.

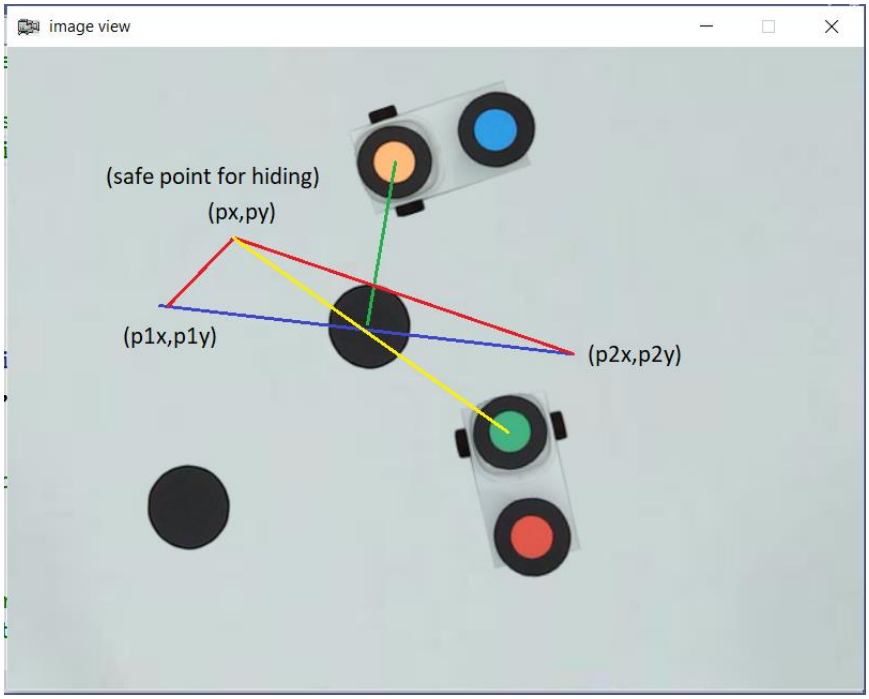
4. complete_defence():

   The defending robot selects a point behind the obstacle and allows robot B to position itself such a way that both the robots are in a straight line with an obstacle between them. This point is selected by using parametric equations as follows.

   ```
   px = ((cos(theta_obs_a)) * (d_obs_b + 150)) + y_ic;
   py = ((sin(theta_obs_a)) * (d_obs_b + 150)) + y_jc;
   ```

   the point basically moves according to the attacking robot location and hence allows defending robot to locate which direction is suitable to go. The defending robot will continuously follow the point allowing it to dodge the laser of enemy robot.

Output results

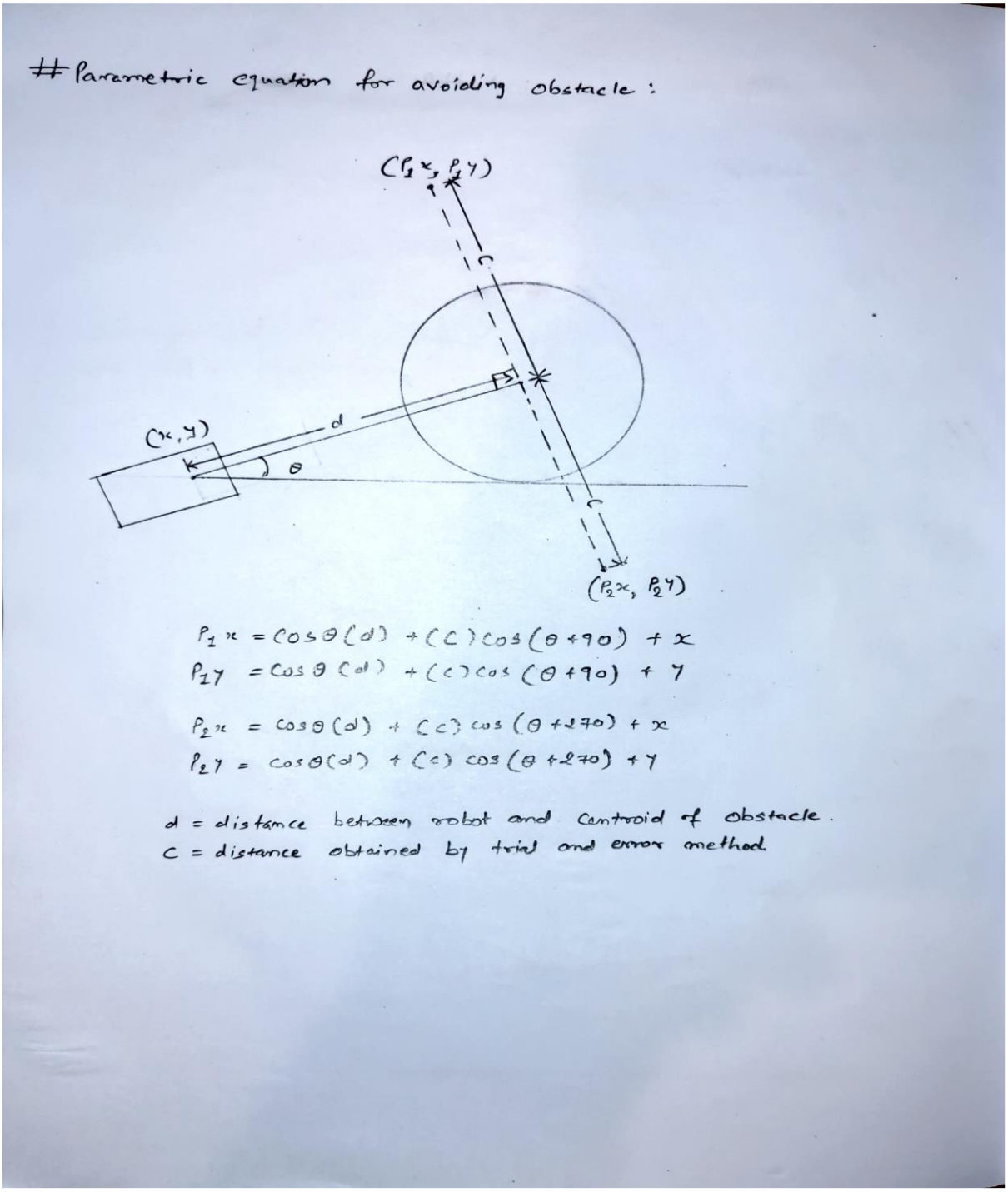



```
Select C:\Users\15145\Desktop\6631 Report\Attack_vs_Deffence\Attack_vs_Deffence\Debug\program.exe
================================================================================================================================
safe point to hide:     px = 220.178    py = 120.933
1st point to avoid obstacle:    p1x = 592.997   p1y = 177.511
2nd point to avoid obstacle:    p2x = 209.098   p2y = 497.857
================================================================================================================================
================================================================================================================================
safe point to hide:     px = 223.583    py = 119.33
1st point to avoid obstacle:    p1x = 594.32    p1y = 183.837
2nd point to avoid obstacle:    p2x = 202.999   p2y = 495.073
================================================================================================================================
================================================================================================================================
safe point to hide:     px = 226.275    py = 118.162
1st point to avoid obstacle:    p1x = 595.567   p1y = 189.214
2nd point to avoid obstacle:    p2x = 198.591   p2y = 493.205
================================================================================================================================
================================================================================================================================
safe point to hide:     px = 224.445    py = 116.8
1st point to avoid obstacle:    p1x = 594.732   p1y = 187.3
2nd point to avoid obstacle:    p2x = 198.975   p2y = 492.876
================================================================================================================================
================================================================================================================================
safe point to hide:     px = 227.94     py = 115.302
1st point to avoid obstacle:    p1x = 595.904   p1y = 193.802
2nd point to avoid obstacle:    p2x = 193.022   p2y = 489.921
================================================================================================================================
================================================================================================================================
safe point to hide:     px = 230.675    py = 114.226
1st point to avoid obstacle:    p1x = 597.004   p1y = 199.279
2nd point to avoid obstacle:    p2x = 188.756   p2y = 487.955
================================================================================================================================
```

Parametric equations

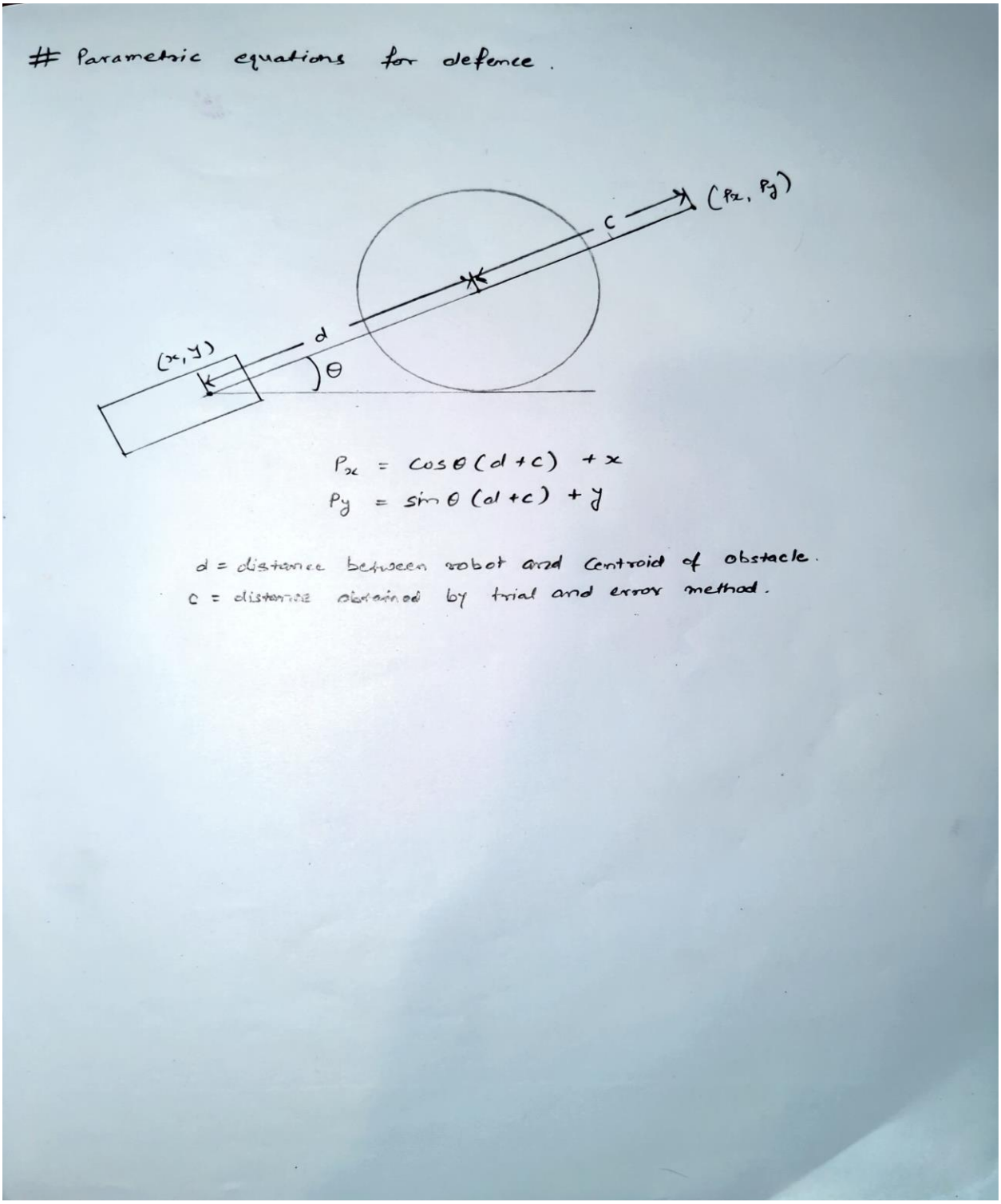
for obstacle_avoid():

The following picture describes calculation of parametric equation performed to find two points that robot can use to move around the obstacle without colliding it.



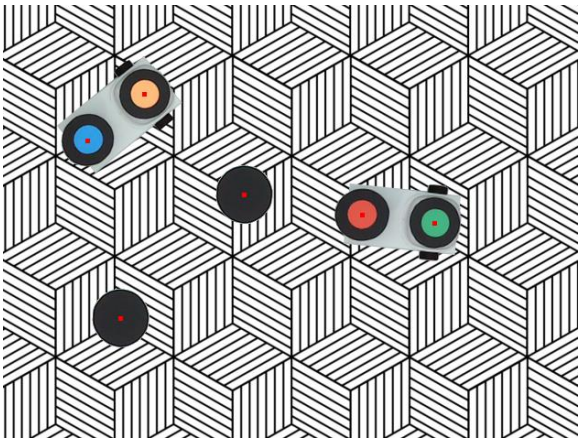
for complete defence:

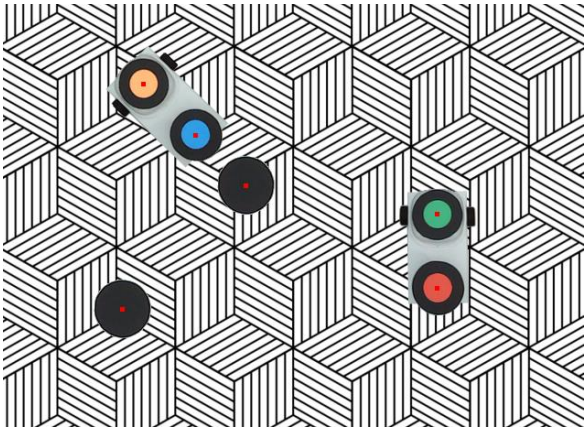
Following diagram represents the parametric equation calculated for the complete_defence() function.

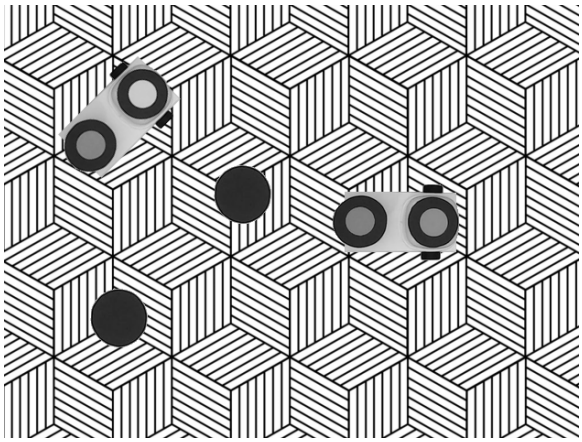


Optional part

1. For the optional past we changed the background of the simulator. Following are the output images. We were able to track centroids properly.







2. The way we have used the parametric approach for our defence and attack, we are confident that it will be able to track moving obstacles.
3. The label_size() function we have developed, is able to identify the obstacles of unknown shapes and sizes given that the size is not less than and equal to the size of the robot coloured circles.