Industrial Automation (MECH 472/6631)

Instructor: Prof. Brandon W. Gordon

Submitted By		
Name	Student ID	Contribution
Shahariar Hossain	40183244	Size Calculation, Pattern tracking based on HSV outcome from the simulation, assigning variable & values of label & size based on pattern analysis, Theta calculation, Attack & Evade program development & general management of all project deliverables.
Pranab Kumar Saha	40185228	RGB & HSV Calculation, Object labelling, Angle (Theta) development, Attack part logic development & contribution to general quality check.
Mounia Abou-Bakr	40187002	Angle (Theta) calculation and pattern development to rotate robot based on angle, Evade program development, Opponent Tracking program development.
Dolapo Adeoya Emmanuel	40161456	General Logic development for various control scenarios.

May 2021

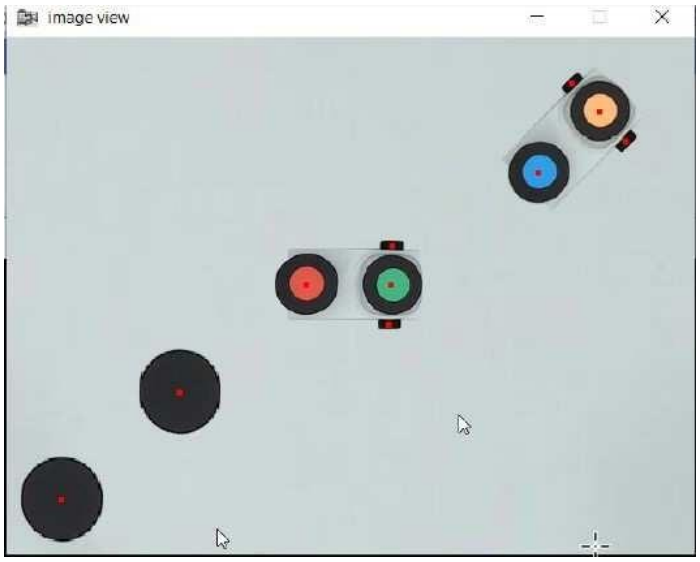
Index

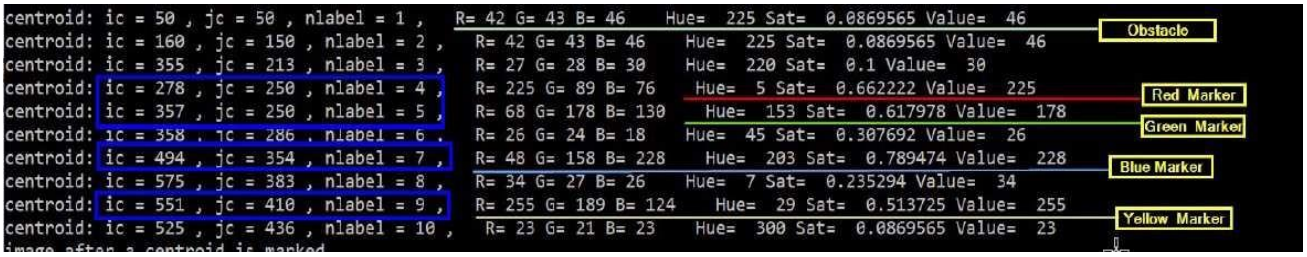
Introduction

Through this project we have tried to develop a fully autonomous RC car system and checked the logical parameters using a simulation program written in C++. The main objectives we have targeted to achieve through this effort is to identify a particular car based on the available parameters and assign all the control logics accordingly to operate the car to perform certain activities; in our project it is to hit the other car with the laser. As the preliminary step we have developed the program so that it can detect multiple objects in an image file based on color (RGB) and mark them in a sequence for the purpose of easy identification. Then we have extended our approach of object detection in a simulated environment and to increase the accuracy in object segregation we have introduced additional features like size (Total number of pixel present in a labelled object and HSV color co-ordinate. Then as the next step we were able to track all the objects moving in the simulation using above parameters and apply some control logics. As our simulation environment will have a certain boundary based the actual image size used as background and contains multiple obstacles our primary goal was to ensure that RC car will not go out of that boundary and will not collide any of those obstacles. Subsequently we have organised the scenario in two categories to check the control logics; first during the Evade Situation when an opponent car will attack and we have to successfully avoid that laser attack and second when we are in the tracing of the opponent and hit it successfully with a single laser shoot and we call it Attack Situation. Our program has successfully performed the above tasks in different scenarios. The following paragraphs will highlight the major milestones and important portions of the final program.

Object Detection in a Single Image :

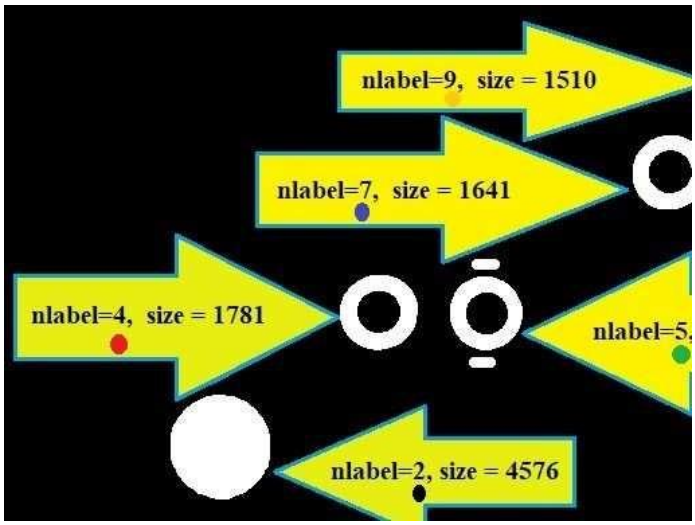
We have generated few images from the simulation and used them to check if our program can detect and label all the stationary objects present in those images. During this process we have to apply various convolution techniques like general filtering based on the threshold in a grey image, erosion and dialation. Below image showing the labelled objects and the output parameters we could trace from them. We could detect all the object's centroid which is an important aspect we have used in next level of our program. We could trace all the color using both RGB and HSV and analyzed their accuracy if the position of cars and obstacles are altered and if any particular lighting condition is applied.





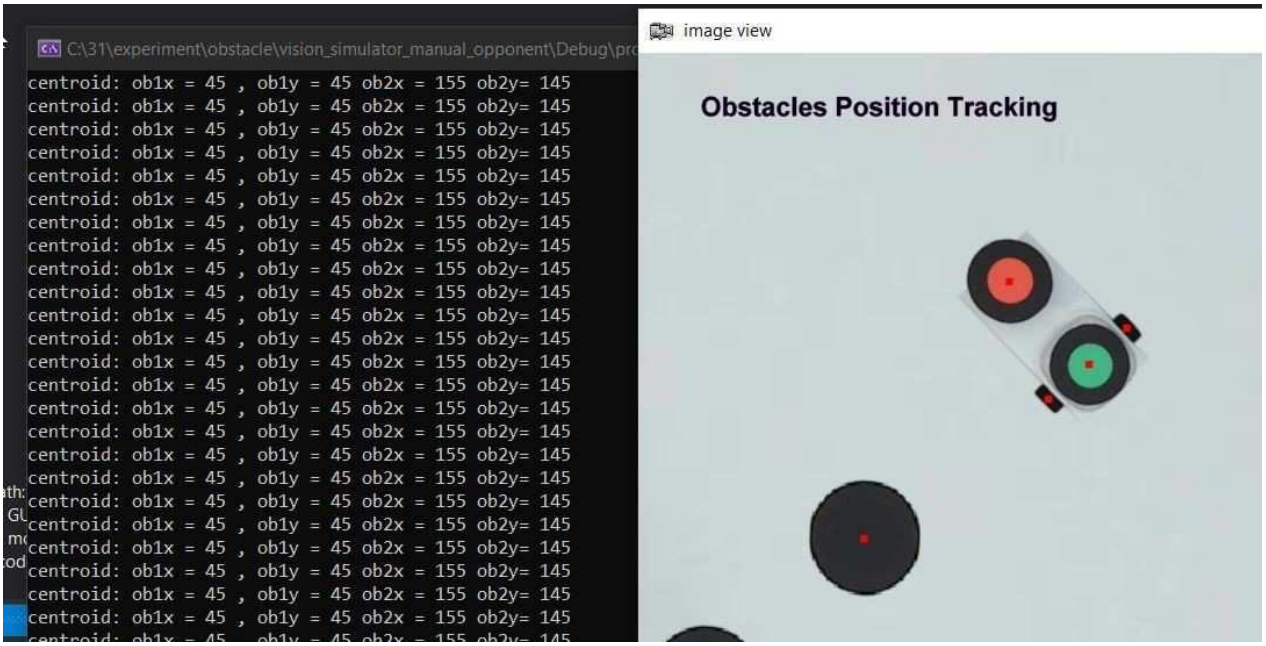
Object Detection & Pixel Size Calculation in Simulation :

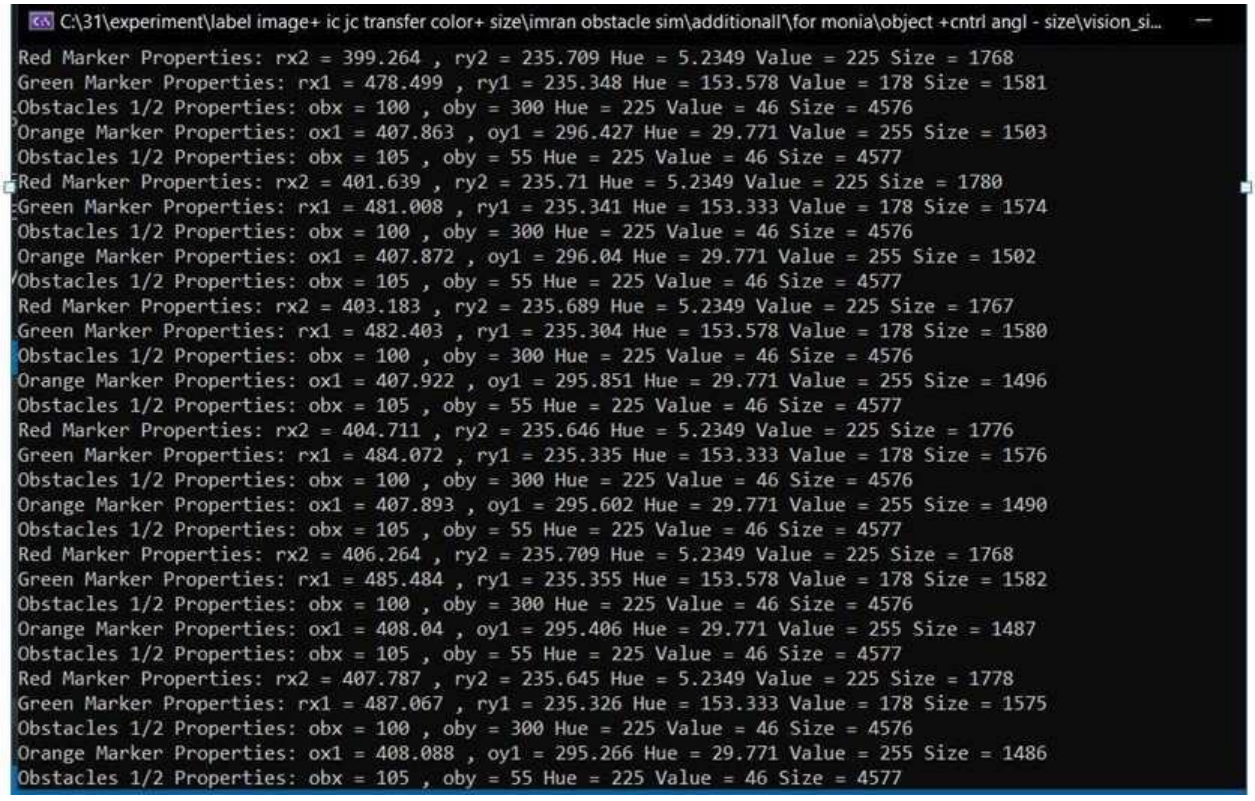
We have extended our approach in this stage to detect all the objects moving in the simulated environment using label, color parameters and the value of total pixel amount present in the particular labelled image stored in a variable size. We believe this is the right approach to increase the accuracy of the detection approach as using only the color parameters may get affected in various lighting and convolution situations. Based on these size and color parameters we can now decide which is our robot marker is and which is opponent robot marker, important for next step of tracking.



Tracking own robot, opponent & obstacles :

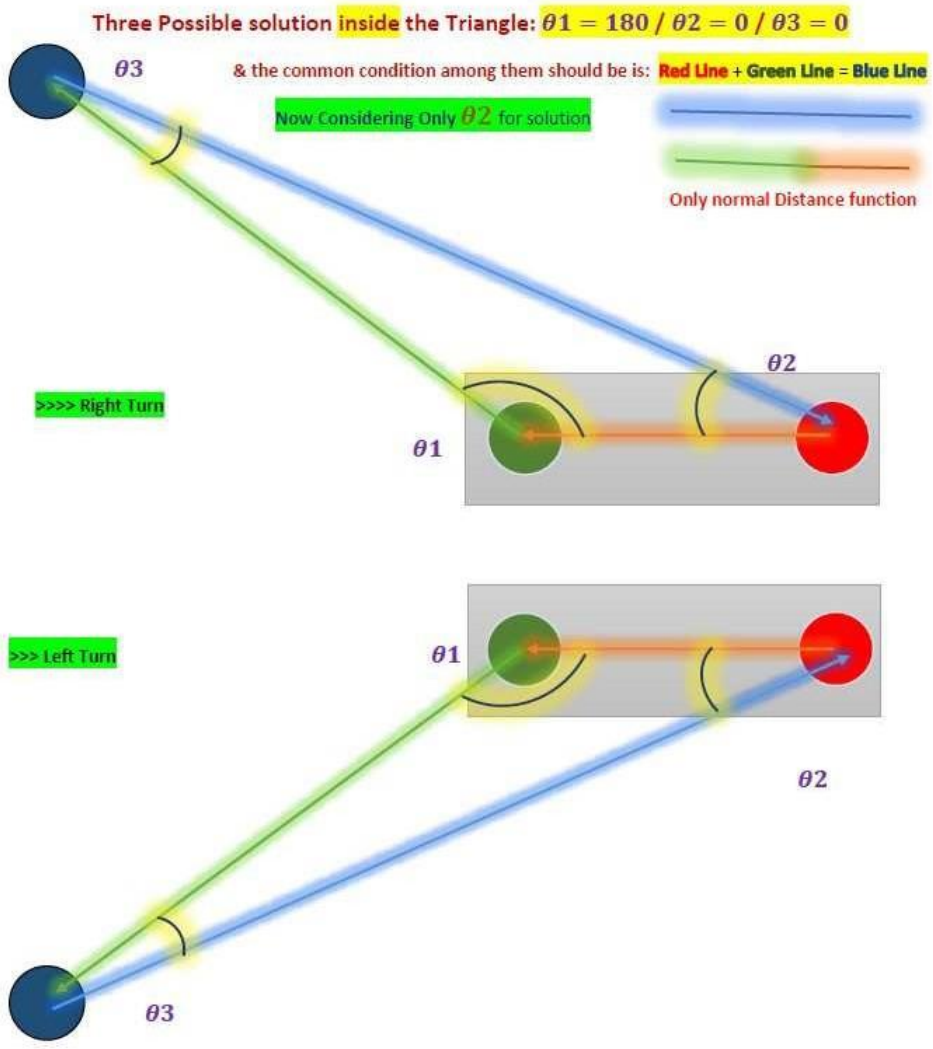
In this stage we have created few variables (Ex : rx,opx,obx etc) to save the centroid of particular markers which are changing their position continuously based on the motion we have assigned during the initialization stage. The following images taken during simulation demonstrate the changing values of marker's & obstacle's centroid values.





Theta (Angle detection) :

During this step we have developed functions to calculate the angle difference between our own robot and other objects (Opponent/Obstacles). Based on our experience this was the most difficult part of the project when we have to develop a pattern how our own robot will response to a certain angle difference and how we will assign control logic when a desired angle is reached. We have applied vector dot product technique to detect the desired angle and then apply certain pulse width which controls the car motion either moving in straight line or in a circular motion. As theta can be calculated in various approaches we have finally decided to use the analogy that the blue vector as shown below need to be aligned with the combined value of red and green vector. The location of the opponent in respect to our own robot will dictate the distance marked in blue, red and green line. To ensure that our own robot is facing towards the opponent the blue marked distance need to be greater than the green marked distance. Using this approach we have controlled the timing of laser shooting during the attack situation and ensured the desired gap between our robot and the opponent during evade scenario.



Control Logic Development :

Obstacle Avoidance:

During this step we have developed some functions to detect the obstacles based on their size, developed some if-else ladder to assign actions when an obstacle approaches close to our robot. As our program is based on the object size rather using it's color properties it should identify the number of obstacles present in the simulation, assign label on them and avoid them during the robot motion unless the predefined size of the obstacle is changed. It can be used even to detect the moving obstacles considering the same logic of having same pre-defined obstacle size. We have created a variable called range which will allow the robot to calculate the maximum distance it can be closer to an obstacle and then we have applied some time function to fridge the robot using pulse width value for certain time before it starts moving away from the obstacle.

```
double dis_og = distance(rx1, ry1, ox, oy); //// green to opponent centre
double dis_or = distance(rx2, ry2, ox, oy);//// red to opponent centre  //// dis_or > dis_og


                                          ////     if ((rx >= ob1x +40  || rx <= ob1x -40 ) || (ry >= ob1y +40 || ry <= ob1y - 40))

                                          ////     if ((rx >= ob2x +40  || rx <= ob2x -40 ) || (ry >= ob2y +40 || ry <= ob2y - 40))
if ((rx >= ob1x - obs_range && rx <= ob1x + obs_range) && (ry >= ob1y - obs_range && ry <= ob1y + obs_range))  // inner boundary 1
{

    // ------------------- turn around and run start --------------
    turn_around(tc1, tc2, turn_in_sec, pw_l, pw_r, go_forward_after_turn);
    // ------------------- turn around and run end --------------
}
else if ((rx >= ob2x - obs_range && rx <= ob2x + obs_range) && (ry >= ob2y - obs_range && ry <= ob2y + obs_range))  // inner boundary 2
{

    // ------------------- turn around and run start --------------
    turn_around(tc1, tc2, turn_in_sec, pw_l, pw_r, go_forward_after_turn);
    // ------------------- turn around and run end --------------
}
```

Sample Program for the Obstacle Avoidance

Boundary Condition (Avoiding going out of the screen) :

In this portion we have applied similar logical background we have used for avoiding obstacles. We have assigned a pre-defined value in all four direction of the background image until which our robot can travel. We have carefully considered these boundary values to avoid any scenario where some portion of the robot goes out of the screen and rest remains inside. Once our robot reaches near to the boundary it will stop the motion for a certain time period and after that period it will move according to the assigned circular or straight motion.

```
else  if ((rx >= 560 || rx <= 80) || (ry >= 400 || ry <= 80)) {  /// boundary
    turn_in_sec = 2;
    // ------------------- turn around and run start --------------
    turn_around(tc1, tc2, turn_in_sec, pw_l, pw_r, go_forward_after_turn);
    // ------------------- turn around and run end --------------
}
```

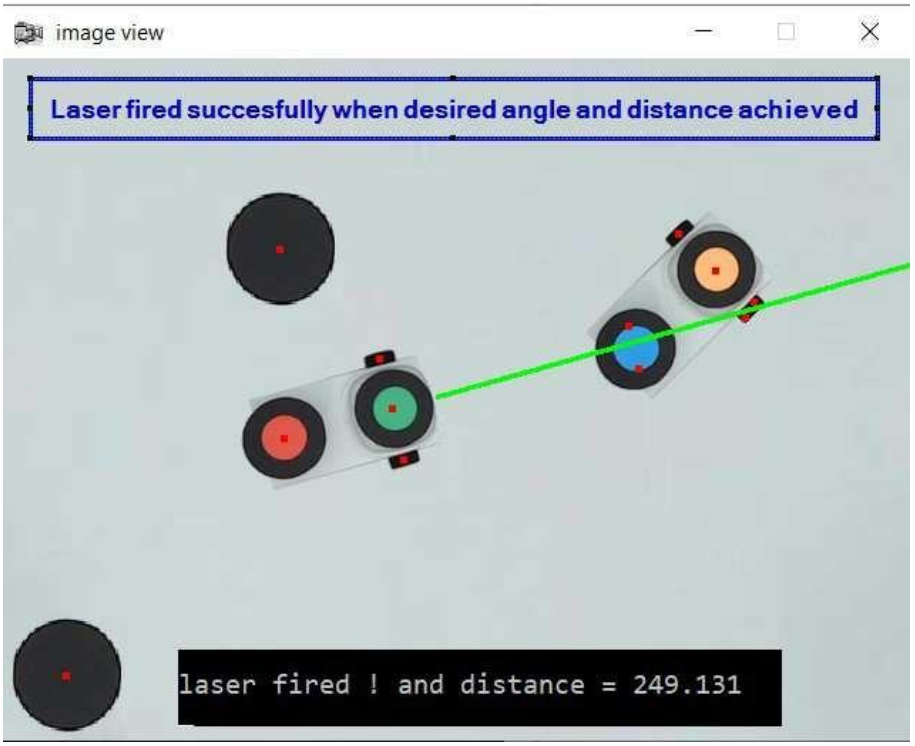
```
void turn_around(double& tc1, double& tc2, int turn_in_sec, int& pw_l, int& pw_r, int go_forward_after_turn)
{
    if (tc1 == 0)
    {
        tc1 = high_resolution_time();
    }
    tc2 = high_resolution_time() - tc1;

    if (tc2 > 0.01 && tc2 < turn_in_sec) {
        //go right
        pw_l = 1000;
        pw_r = 1000;
    }
    else if (tc2 < go_forward_after_turn) {
        pw_l = 1250;
        pw_r = 1750;
    }
}
```

Sample Function used for obstacle avoidance & in Boundary condition

Attack and Evade Scenario:

The Attack Scenario starts when our robot can successfully detect the angle difference with the opponent centroid and can align itself within the described range (In our case 1.5 degree on either side of the zero mark). We have assigned a forward motion when the above criteria is achieved and while reducing the distance it will consider the case when the distance between both the centroids reaches 250 pixel or less and then shoot the laser. Our approach here is to ensure precision as much possible before shooting the laser as only one chance is allowed per round.



```
    else {
        tc1 = 0; // set boundery timer tc1 to 0
                 //if (Diff1 > -range && Diff1 < range) /////
        if ((Diff1 > -range && Diff1 < range) && (dis_or > dis_og)) /////
        {
            // withing range go towards the opponent
            pw_l = 1250;
            pw_r = 1750;
            laser = 1;
            //laser = 1;
            attack_opponent(dis, dis_or, dis_og, pw_l, pw_r, laser);
            //laser = 1;
        }
        else if (Diff1 > range)
        {
            //go left
            pw_l = 2000;
            pw_r = 2000;
        }
        else if (Diff1 < -range) {
            //go rigt
            pw_l = 1000;
            pw_r = 1000;
        }
    }
}

void attack_opponent(double dis, double dis_or, double dis_og, int& pw_l, int& pw_r, int& laser)
{
    if ((Diff1 > -0.5 && Diff1 < 0.5) && (dis <150) && (dis_or > dis_og)) {
        pw_l = 1500; // stop
        pw_r = 1500;
        laser = 1;
    }
}
```

Sample Program for the Attack Scenario

For the evade portion we have applied the analogy to go further away from the opponent continuously while applying the function called "Turn Around" . During this step the above applied Boundary Condition and Obstacle Avoidance approaches still remains same.

```
if ((rx >= ob1x - obs_range && rx <= ob1x + obs_range) && (ry >= ob1y - obs_range && ry <= ob1y + obs_range))  // inner boundary 1
{
    // ------------------- turn around and run start --------------
    turn_around(tc1, tc2, turn_in_sec, pw_l, pw_r, go_forward_after_turn);
    // ------------------- turn around and run end --------------
}
else if ((rx >= ob2x - obs_range && rx <= ob2x + obs_range) && (ry >= ob2y - obs_range && ry <= ob2y + obs_range))  // inner boundary 2
{

    // ------------------- turn around and run start --------------
    turn_around(tc1, tc2, turn_in_sec, pw_l, pw_r, go_forward_after_turn);
    // ------------------- turn around and run end --------------
}
else  if ((rx >= 560 || rx <= 80) || (ry >= 400 || ry <= 80)) {  /// boundary
    // ------------------- turn around and run start --------------
    turn_around(tc1, tc2, turn_in_sec, pw_l, pw_r, go_forward_after_turn);
    // ------------------- turn around and run end --------------
}
else {

    if ((Diff1 > -range && Diff1 < range) && dis < 200)//150 /////&& (dis_or > dis_og)
    {
        // ------------------- turn around and run start --------------
        turn_around(tc1, tc2, turn_in_sec, pw_l, pw_r, go_forward_after_turn);
        // ------------------- turn around and run end --------------

    }
    else {
        tc1 = 0;
        // withing range go towards the opponent
        pw_l = 1250;
        pw_r = 1750;

    }
```

Sample Program for the Evade Scenario