Department of Mechanical and Industrial Engineering

MECH 472

Platoon Type-11

Project Report

Muneeb Rehman	40058834
Gurvir Grewal	40062915
Alexandro Tomassini	40029074

Responsibilities

Muneeb Rehman: [Page 3-7]

- Built the real robot with 3D printed designed parts, hardware, and Bluetooth communication.
- Built feature that allows acquisition of two or more real camera views.[Sim + Real Top View + Real First Person View]
- Built evolutionary type neural network from scratch.
- Built feature that allows robot centroid location regardless of obstacle colors.
- Built four side collision detection features.
- Built eight distance sensors feature.
- Built Virtual Force Field feature.
- Built attack and evasion strategy system, with PID controlled yaw.

Gurvir Grewal: [Page 7-10]

- Built the search method to access every pixel in the image relative to the moving robot.
- Built the image processing that creates an RGB mask of the safe locations created by stationary objects, distinguishing between objects and robots.
- Built the method to distinguish safe zone locations as labelled objects by building a custom safe zone thresholding feature which creates a binary image of the RGB safe zone mask.
- Built the feature to determine the centroids of each individual safe zone location, filtering out unfavorable safe zones and creating points of interest for evasion purposes.
- Built an initial evasion system to control the robots to the safe zone centroids, replaced by Virtual Force Field feature implemented for attacking.

Alexandro Tomassini [Pages 8,10]

- Assisted team in developing a defensive strategy for the robot during the defending phase.
- Assisted team member to create a method to determine the safe zones for the robot to hide during the defending phase.
- I worked on creating a bare bone (Plan B solution/Backup solution) method for controlling the robot such that it can navigate to a safe zone without colliding or driving off the screen.

The Real Robot

Before the vision robot simulator was announced, I set out to make the actual robot using two modified HS-422 servo motors, a Ping Sensor, an HC-05 Bluetooth module, one MPU6050 Gyroscope Accelerometer, and lots of manufactured parts. The robot was designed to be as compact as possible but also fast, which explains the big wheels. Using an old project of mines, the semi-spherical Surveillance Camera, it was possible to turn the Raspberry Pi camera module into a wireless webcam that the simulator could access, displaying a first-person view. However, since the new project instructions rules were changed and announced very late, the real robot is discontinued. It can still be setup to run like the simulator since there is an RGB light and a Serial program inside our project.

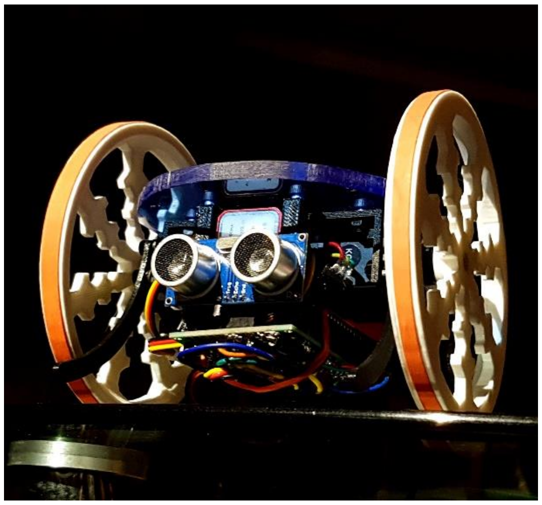


Figure 4- Platoon Type-11 Front Side

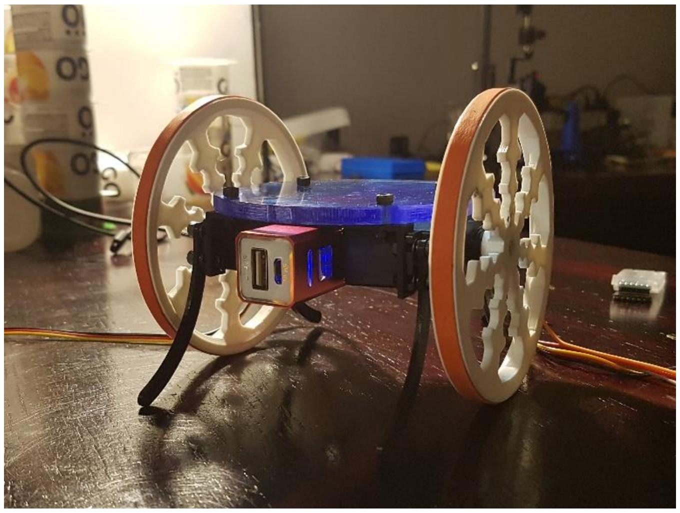


Figure 1-Platoon Type-11 Back Side

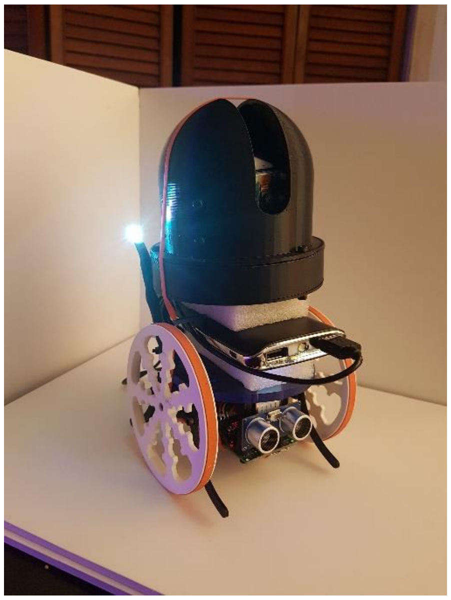


Figure 3- First Person Surveillance Camera

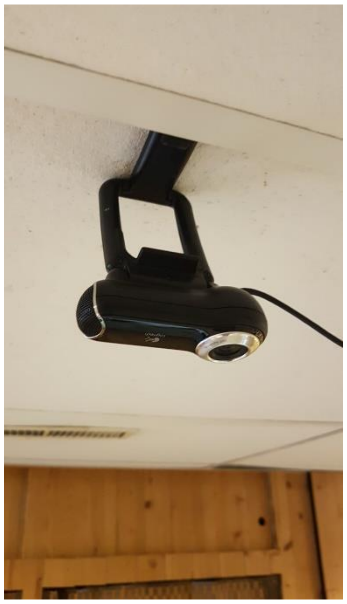


Figure 2- Top View Camera

Collision Sensors

The collision sensor was initially designed to reset the neural network and proceed to the next trial, it was then used to command the bot to move in a certain direction using an if else statement ladder, but this proved to be a failure since the high likelihood of executions interfering against each other causes the robot to move inside obstacles. Now, it's only used a safety precaution to stop the main player from hitting obstacles and borders, while also disable the laser system and attack system radar if the enemy robot leaves the map or hits an obstacle (which would cause it to share labels and confusing the main player). The collision points are a series of points that can be adjusted for different relative positions, lengths, and amount. Collision state is trigger if a white pixel is detected and is kept that way for a duration until the trigger is reset.

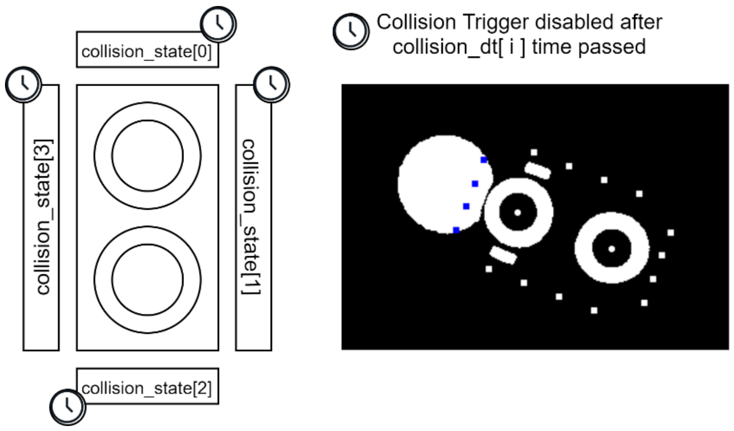


Figure 5- Collision Detection Feature

Distance Sensors

Same concept and reasoning as above, now used as an extra safety measure to avoid accidentally hitting obstacles and borders.

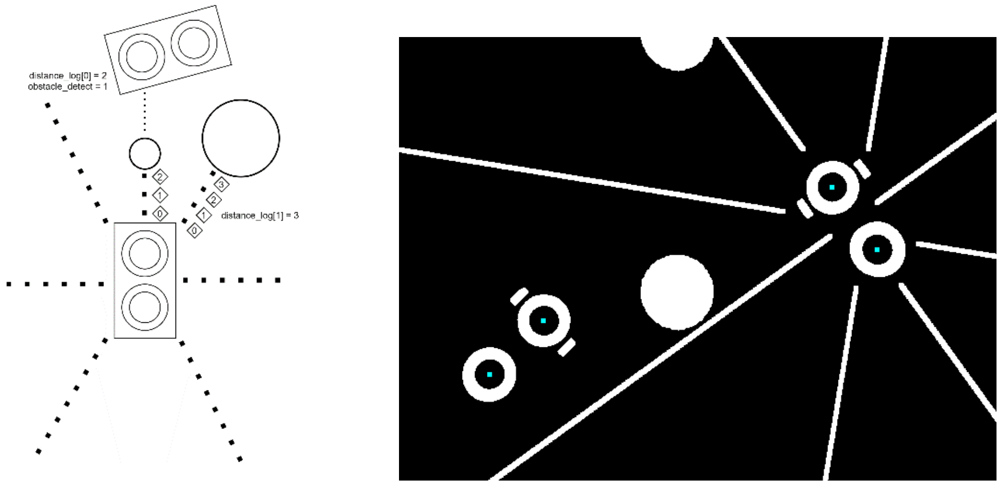


Figure 6- Distance Sensor

Evolutionary Neural Network

Both the collision sensor and distance sensors were made so that they can be used for my artificial intelligence system. The neural network is built from scratch and follows an evolutionary style system. The robot would go through any number of adjustable iterations between 2-100 with different weights, if it hits an obstacle or the simulation runs for too long, the weightings are saved, and the simulation is reset. The weights are mutated relative to the best weightings, and the weight configuration that can keep its eyes on the enemy for longer is kept. The hidden layer and output layer are calculated by the sigmoid function, and depending on the real time input values, the output adjusts its behavior using the weights that had a higher success of not hitting obstacles, surviving for longer and finding the enemy.

Unfortunately, the system had to be discarded completely, since it would take hours to keep training the robot, despite having increase the robot speed by ten times. Even if it successfully completes its tasks, it would take even more time to train it to avoid obstacles while looking for the enemy, especially on a small map. One possible solution would be to run the same robot a hundred times on the same map at the same time, but that's simply not possible.

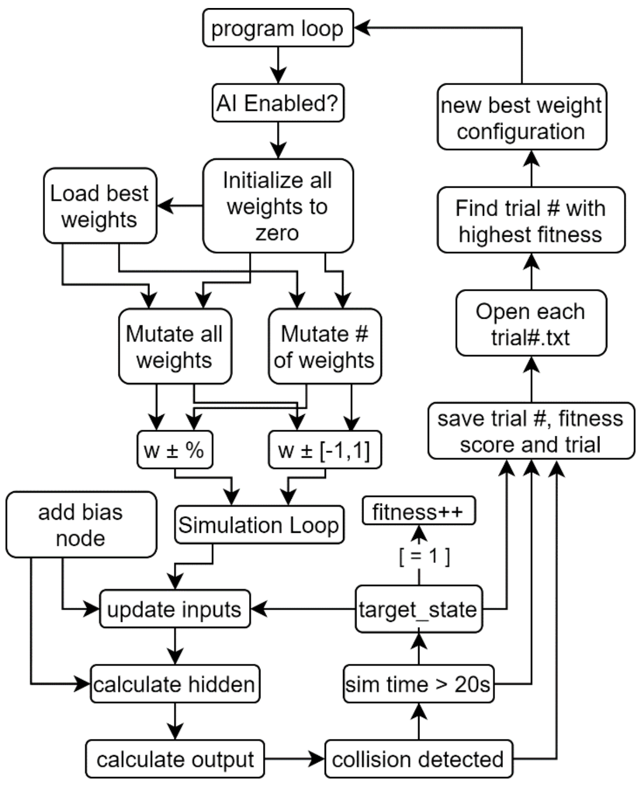


Figure 8 -NN Flowchart

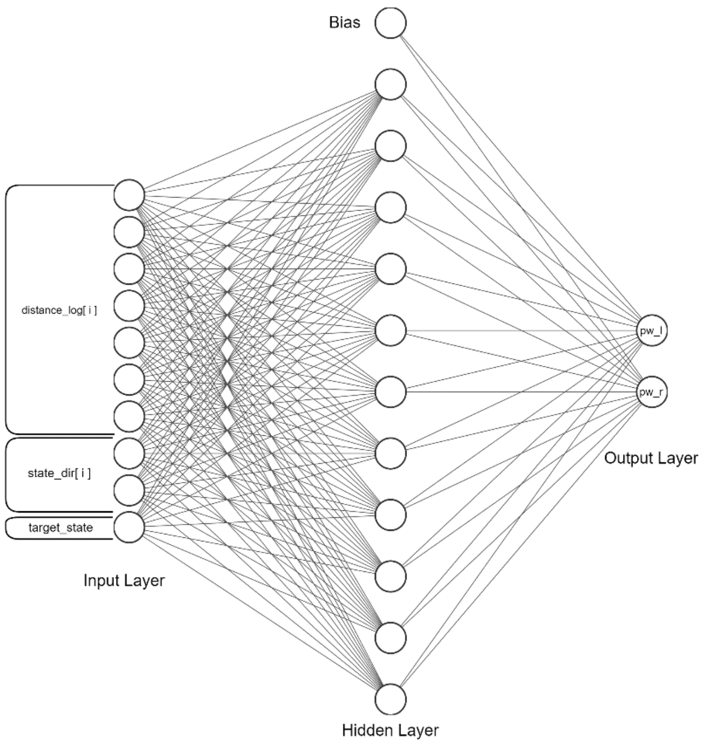


Figure 7- NN Design

Virtual Force Field System

The virtual force field is a concept that gives the robot a sense of autonomous direction based on the resultant vector. The robot goes straight if the magnitude is too small and can avoid obstacles as the resultant vector points in the opposite direction. If enemy labels are detected, then a large negative multiplier causes the resultant vector to point to that direction. If the robot is meant to evade the enemy, then the vectors ignore the enemy, but the colored safe zone becomes highlighted, with a large negative multiplier this time.

The VFF is designed to be very modifiable according to any specified theta range. The minimum and maximum range of each vector and their multiplier can be adjusted, and a counter vector would take care of the difference such the resultant vector is zero if there are no obstacles. However, these adjustments tend to cause the robot to display unexpected behaviors, such as turning left and right continuously because the range of vector combination isn't diverse enough, so the robot is stuck.

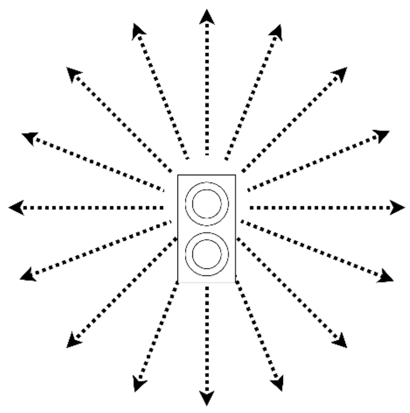


Figure 10- No object

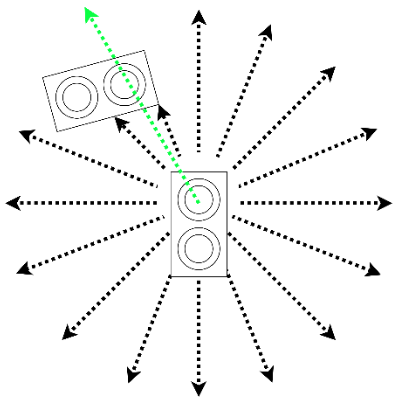


Figure 9- Enemy Detected

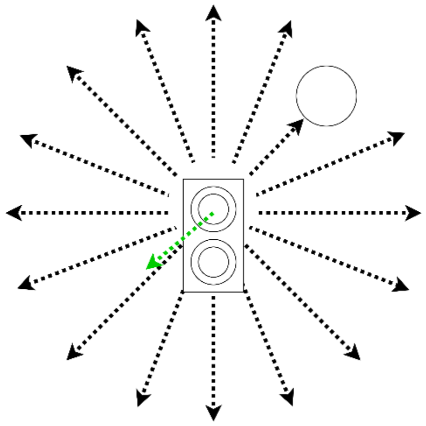


Figure 11- Obstacle Detected

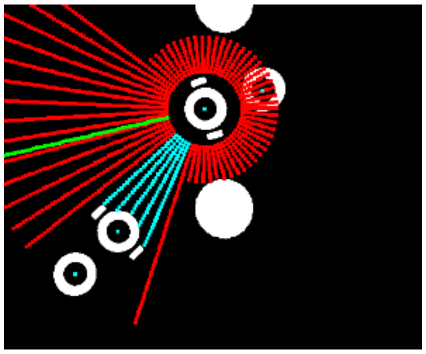


Figure 14- Modifiable VFF

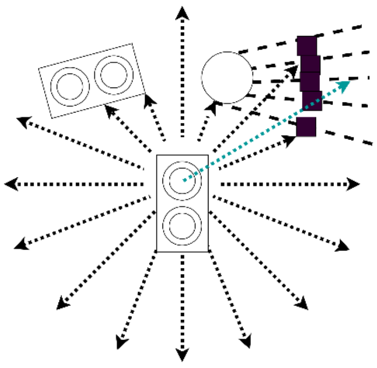


Figure 13- Safe Zone Detection Logic

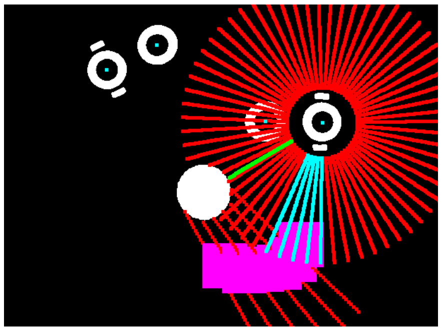


Figure 12- Safe Zone Detected

Offense & Evasion Strategy

The overall system, and the result of two months of work, is compressed here:

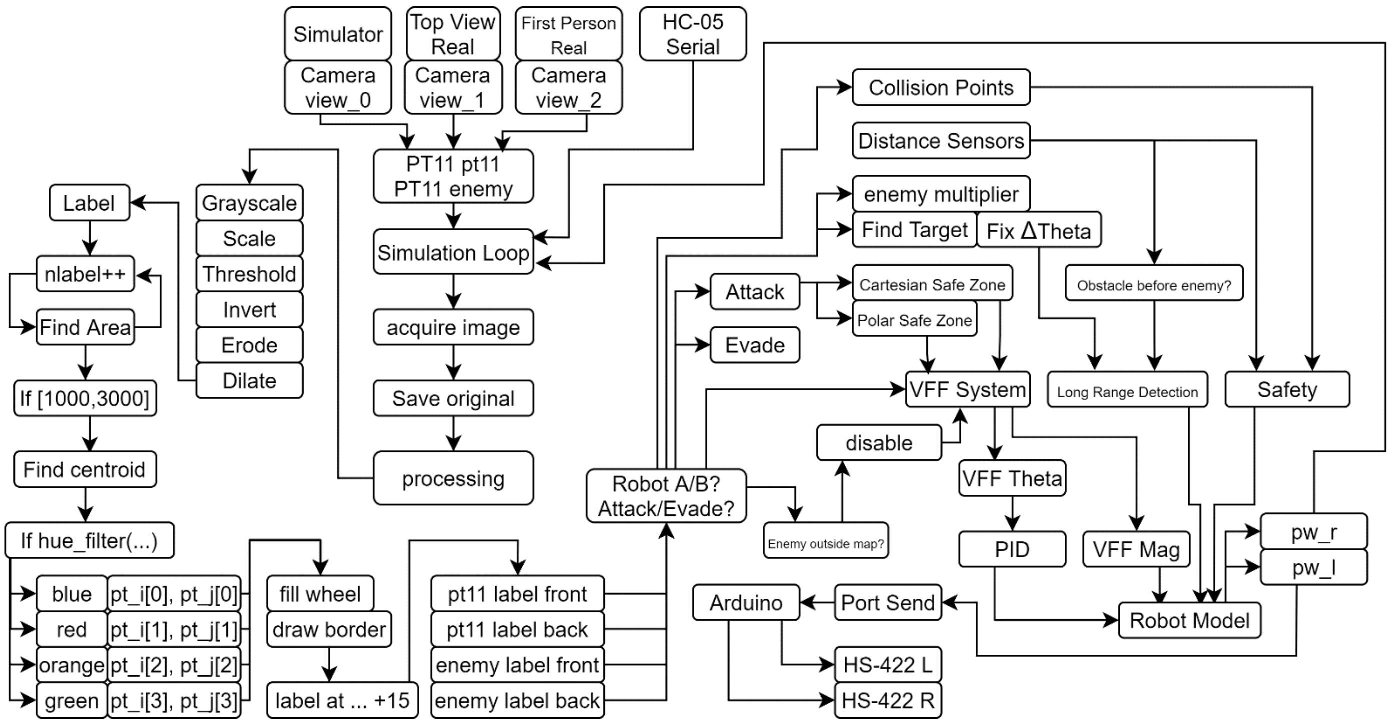


Figure 15- Overall System Design

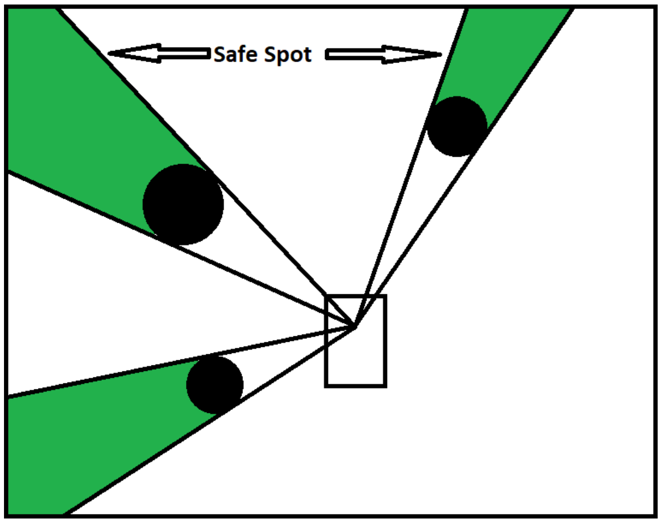
Adaptive robot centroid locator

Finally, after the project announcement that the obstacles would have similar colors to the robots, it was no longer possible to determine the centroid with just the robot colors. Two separate processes were made:

1. Before any processing is done, after the rgb image was copied to an original image file, every colored pixel within an HSV range were turned black, so the threshold process wouldn't remove the obstacles.
2. After labelling, only the labels that were within an Area range were kept. The Area of the robot's front and back side were always less than of those obstacles. The centroid of those locations are logged, and sorted depending on the colors that correspond to either Robot A or B, as well as the Front and Back side.

Defending Phase

During the defending phase, the defending robot needs to evade the attacker. To do so, the defender must avoid being in the line of sight of the attacker. If the evading robot were to be visible the attacking robot, the attacker could fire their laser and win the round. Obstacles have been placed in the combat zone which will help the create areas which block the attacker's line of sight. To determine these safe locations, it is known that lasers cannot pass though the obstacles. This means the evading robot needs to place itself on the opposing side of the obstacle with respect to the attacking robot.



Thus, the defending algorithm of the robot relies on tracking the position of the enemy robot and creating a radar that indicates what areas of the map are safe from the lazar due to obstacles breaking line of sight.

The methodology involves accessing every pixel in the image by storing their coordinates in an array. A line equation is used, where the first element in the array is the pixel located at the centroid of the robot, and the last element in the array is the pixel located near the border of the image. Processing is performed by incrementing through the array, accessing each pixel coordinate in the corresponding binary image, and assessing whether the pixel is white or black. Once a white pixel is detected, this indicates that an object has been detected along that specific line expanding from the robot, and thus every following black pixel in the array should be colored green in a RGB mask. This method creates a radar-like feature where every pixel coordinate is accessed using a linear, radar-like sweep, and the pixels in the array are colored as green once a stationary object is detected. A safety mechanism is in place to compare the pixel coordinate with a label image, to check the label number of that object, to assess whether the white pixel is a robot or a stationary object. Pixels are only drawn green if the object is determined to be stationary:

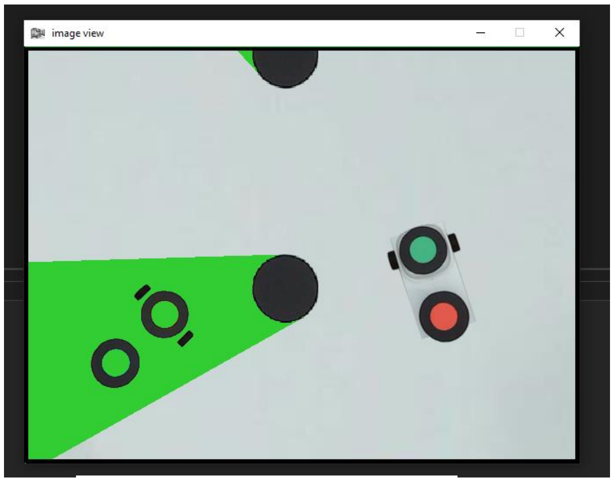


Figure 16- Safe Zone RGB Mask

Once the safe zone RGB mask has been created, further processing of this safety zone can be performed to determine its centroid. First, the safe zone RGB image is thresholded, where the white pixels in the binary object is exclusively the safe zone. Safety checks needed to be added to exclude the robot from appearing in the binary image, this is done by checking the same pixel coordinate in the label image, to determine if it is labelled as a robot object. If it is, then that pixel must be processed in such a way that it is not detected in the label image:

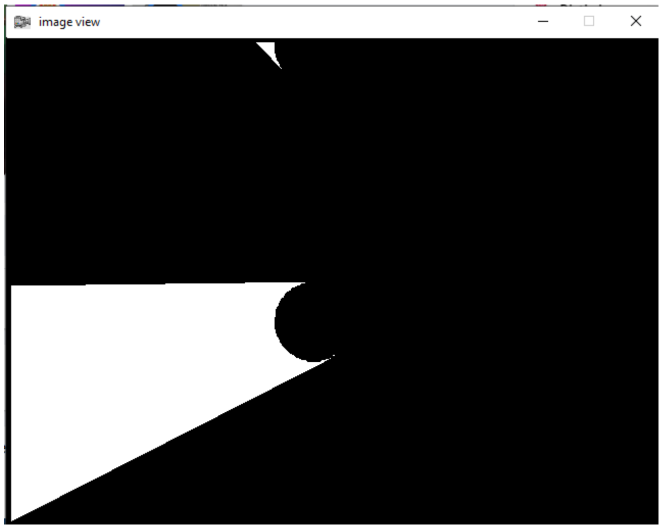


Figure 17- Binary Image of the Mask

We can then identify all safe zones as labelled objects in a label image. Safety mechanisms can now be put in place to filter out inappropriate safe zones, such as those that are too small (i.e.: composed of a small number of pixels). With this method, we can perform centroid calculations of each individual safe zone and assess which safe zone is closer or more suitable to hide behind:

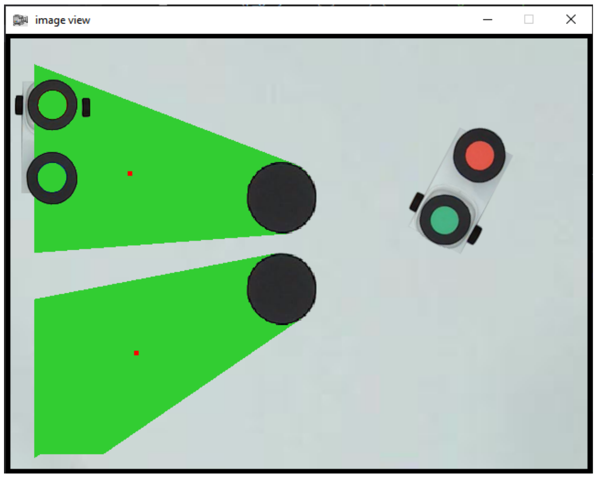


Figure 18- Centroids of Individual Safe Zones

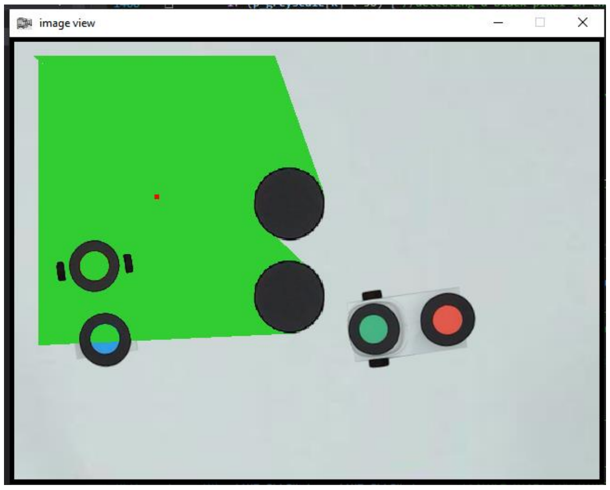


Figure 19- Safe Zones Combining into one Centroid using Label Image Method

Evasive strategies can now be implemented to avoid obstacle collision, in addition to hiding behind safe zones. However, we decided to implement the VFF feature utilized in attacking, to be used evasively to hunt for safe zones. To enable this vector-based detection, the centroids of the safe zones need to be marked in the RGB mask for the vectors to be able to detect the location of the safe zones and adjust position accordingly:

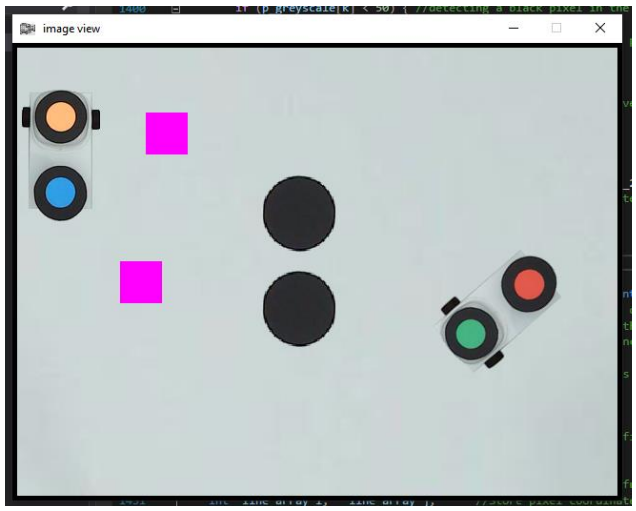


Figure 20- Centroids in Figure 18 Marked for VFF Detection

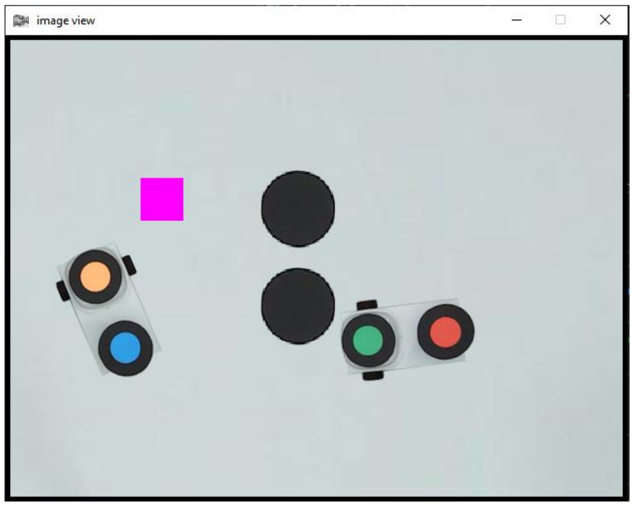


Figure 21- Centroids in Figure 19 Marked for VFF Detection

Backup Evading Solution

The backup solution for the evading maneuver used a brute force method. The goal of the method was to get the robot to drive to a desired location following the shortest path possible. This is achieved by allowing the robot to face the desired point and drive in a straight line to the location. This method would ignore obstacles meaning the robot would inevitably collide with objects. To avoid collisions and allow the robot to maneuver around obstacles, collision points around the robot have been used to trigger different processes. Whenever a collision sensor is triggered, distance sensors would activate and measure the robot's surroundings. The collision would trigger a reset flag and depending on what type of collision is present, the appropriate actions would be taken to avoid further collisions and reset the robot to a state where collisions are no longer present. Since frontal collisions are the most likely collision type, the robot will reverse away from the obstacle and use range data to determine which direction it needs to follow in order to circumvent an object. During the reset state, the robot will ignore the desired location point in order to not cause further unwanted collision. Once the bot has completed a number of maneuvers, the reset state will be removed, and the robot will attempt to align itself to the desired location to drive to the point. The process will continue until the robot reaches the desired point. Due to the brute force nature of this process, many if statements and several logic operations have been developed to successfully allow the robot to complete the evading maneuver.