Gina Cody School of Engineering and Computer Science

Concordia University

MECH6631 Project Report

Qiaomeng Qin(40207375)

Design and implement an Image Processing algorithm to locate and identify all objects in the map. Design and implement an Hiding Strategy to find the closest safe point.

Xiaobo Wu(40216033)

Design and implement a path planning algorithm (ASIA) by myself to find a collision-free sequence of motions (path point) between an initial(current) position and a final(expected) position and complete this part report. Design a hiding of robot algorithm with Qiaomeng Qin and write the report of this part.

Mario Araujo(40059376)

Yuelong Wu(xxxxxx)

2022.04.01

Abstract

technology

Contents

List of Tables

List of Figures

Listings

List of Abbreviations

Chapter 1

Introduction

As shown in Figure 1.3, two robots perform a competition in this project. One robot chase the other and try to hit the opponent with laser. And the other robot tries to hide from the laser. Both robots are controlled via intelligent algorithms. This report mainly introduces the algorithms for image processing, hiding and attacking strategies, path planning and robot control.

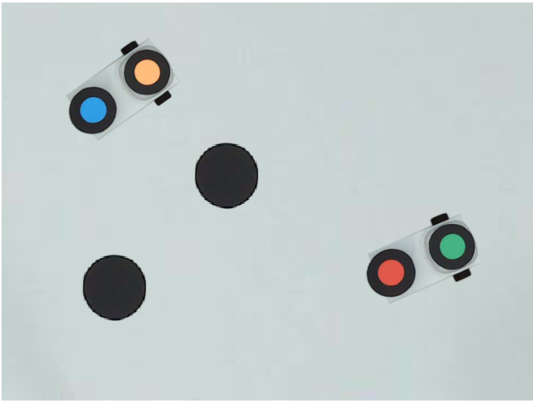


Figure 1.1: Overview of the project.

1.1 Modelling of the Robot

In order to simplify the robot model, we assume that there is no wheel slipping. And the basic structure of robot is shown in Figure 1.2.

$$v_r = \omega_r R$$

Where v_r is the linear velocity, R is the radius of wheel, and ω_r is the angular velocity.

x_c and y_c is the coordinate of the vehicle centre. θ is the direction of vehicle. D is the distance bewteen two wheels. The geometry model of this vehicle is shown below:

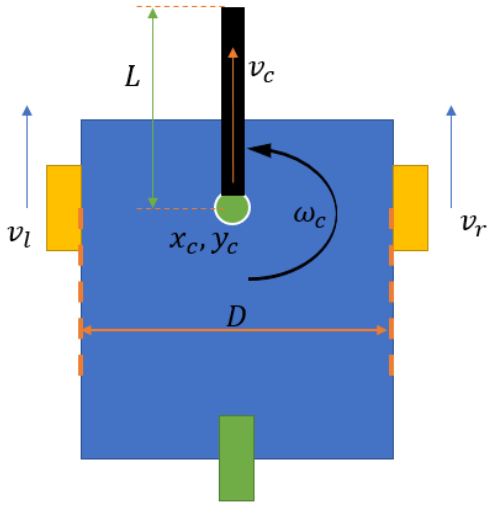


Figure 1.2: vehicle model.

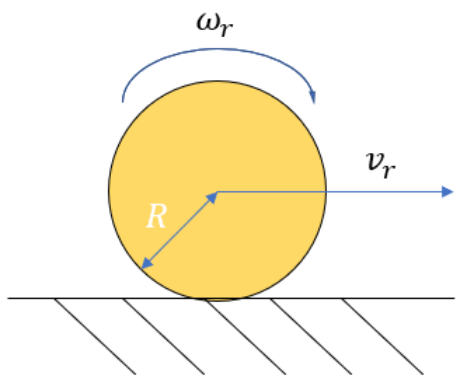


Figure 1.3: wheel model.

$$v_c = (v_r + v_l)/2$$

$$\omega_c = (v_r - v_l)/D$$

$$\dot{\theta}_c = \omega_c = (v_r - v_l)/D$$

1.2 System Structure

The flowchart of the whole system is shown in Figure 1.4

The input and output for each part is shown in Table 1.1. The input of subsequent part is the output of last part.

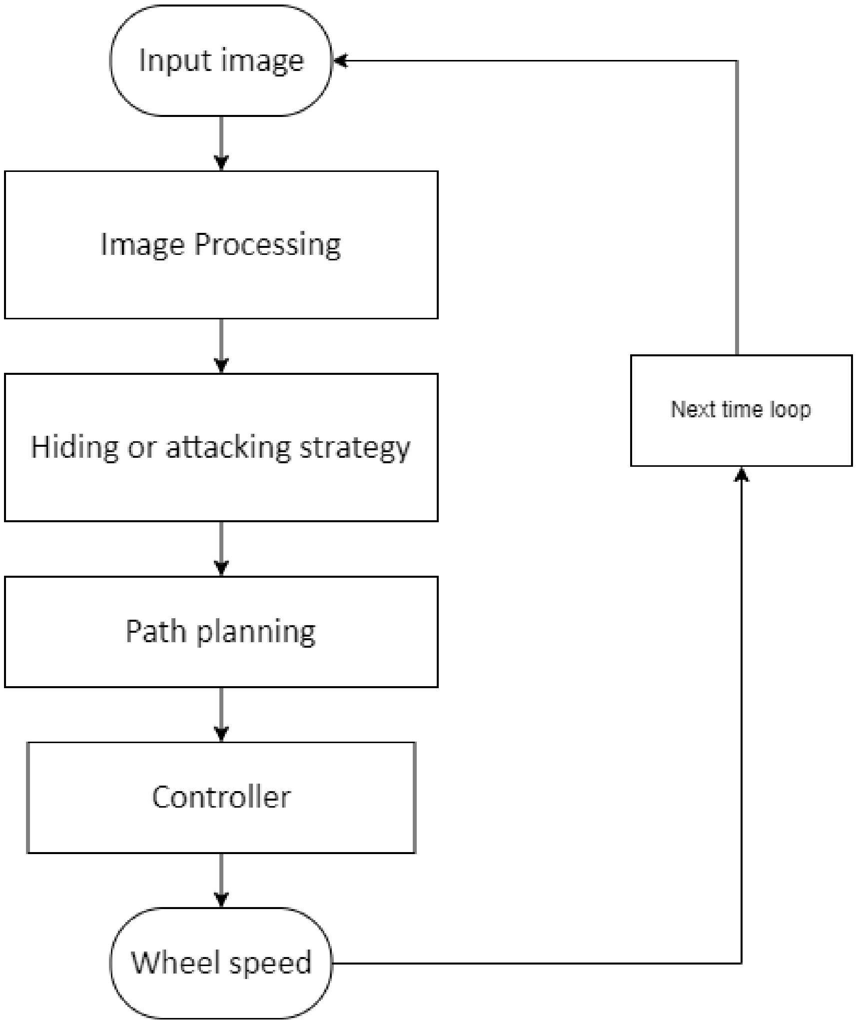


Figure 1.4: the flowchart of the whole system.

1.3 Teamwork

The job of every team member is shown in table 1.2.

Part	Input	Output
Image processing	RGB image of the whole map	Positions of every object with identification
Hiding Strategy	output from image processing	expected hiding position
Attacking strategy	output from image processing	expected attacking position and fire order
Path planning	current position and expected position	expected trajectory avoiding the obstacles and boundary of map
Controller	next point of expected trajectory	speed of two wheels

Table 1.1: the input and output for each part in system.

Name	Qiaomeng Qin	Xiaobo Wu	Yuelong Wu	Mario
Project Management	System design			
Image Processing	Methodology Implementation			
Hiding Strategy	Methodology Implementation	Methodology		Methodology
Attacking strategy				Methodology Implementation
Path planning		Methodology Implementation		
Robot Controller				Methodology Implementation
Report Writing	Introduction Image Processing Hiding Strategy	Hiding Strategy Path Planning	Controller Designing Attacking Strategy	Controller Designing Attacking Strategy

Table 1.2: the work for each team member.

Chapter 2

Image Processing

2.1 Locating Objects

The flowchart of whole image processing algorithm is shown in figure 2.1.

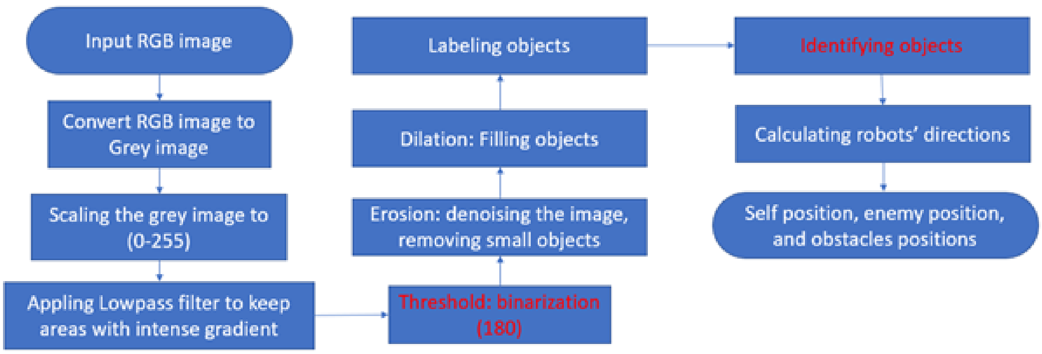


Figure 2.1: Image processing flowchart.

While testing the code, we found that if color of obstacle is changed to blue, the threshold of binarization, with original value of 80, would be not suitable anymore. After testing, the threshold is determined as 180, which is suitable for all colors of obstacles. The comparison of different thresholds is shown in figure 2.2.

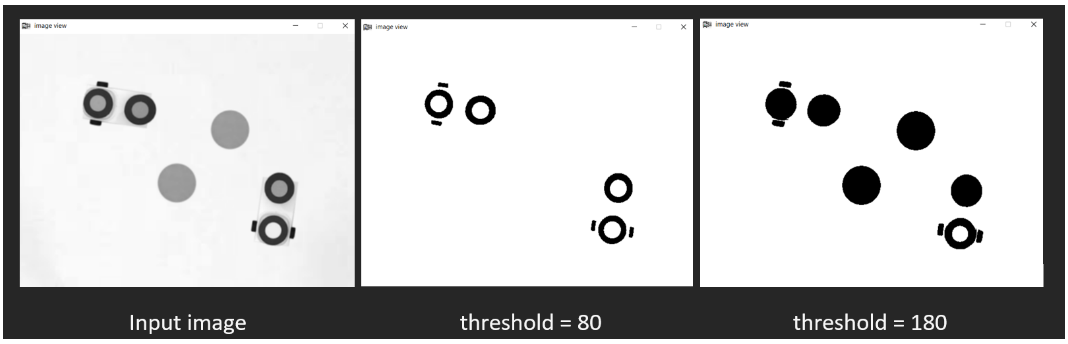


Figure 2.2: Different thresholds in binarization.

2.2 Identifying Objects

Identifying objects is the core part of it, the flowchart is shown in figure 2.3 (assume that self-robot is robot A, red and green)

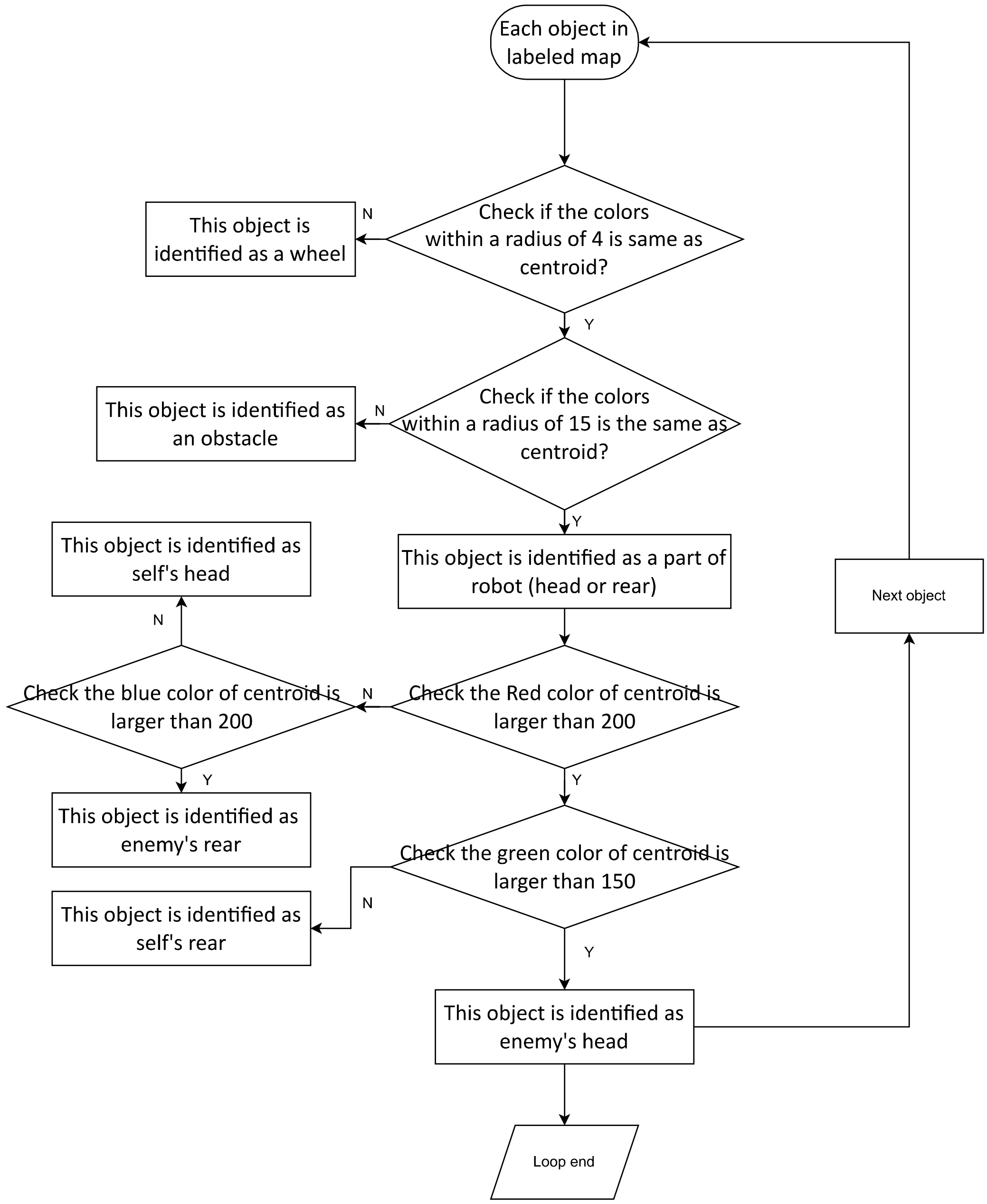


Figure 2.3: Identifying flowchart.

To differentiate self and enemy, we analyzed the RGB values of four different colors as shown in table 2.1

The final result of image processing is shown in figure 2.4 and 2.5.

Colour	Red Value	Green Value	Blue Value
Green(A1)	67	180	131
Red(A2)	226	90	77
Orange(B1)	255	189	124
Blue(B2)	48	158	228

Table 2.1: RGB values of four colors.

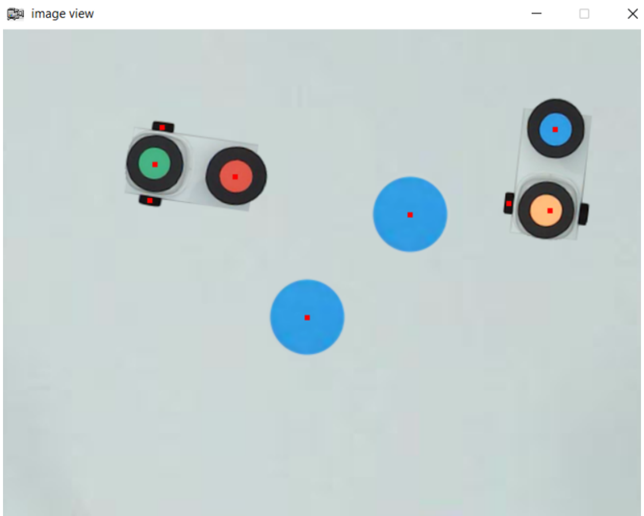


Figure 2.4: Locating all objects.

Microsoft Visual Studio Debug Console

```
found 10 objects
id: x,y,label: 3: 449.282 147.644 1
id: x,y,label: 6: 485.213 142.89 2
id: x,y,label: 6: 413.338 155.278 3
id: x,y,label: 5: 300.037 200.604 4
id: x,y,label: 4: 460.987 227.095 5
id: x,y,label: 2: 105.345 280.958 6
id: x,y,label: 5: 400.037 300.604 7
id: x,y,label: 6: 178.083 325.71 8
id: x,y,label: 1: 148.391 347.422 9
id: x,y,label: 5: 118.24 118.24 0
self_position: 148.391, 347.422, 0.996064,
enemy_position: 449.282, 147.644, -1.71706,
is 100, 100 free? 1
is 300, 200 free? 0
found 10 objects
id: x,y,label: 3: 449.282 147.644 1
id: x,y,label: 6: 485.213 142.89 2
id: x,y,label: 6: 413.338 155.278 3
id: x,y,label: 5: 300.037 200.604 4
id: x,y,label: 4: 460.987 227.095 5
id: x,y,label: 2: 105.345 280.958 6
id: x,y,label: 5: 400.037 300.604 7
id: x,y,label: 6: 178.083 325.71 8
id: x,y,label: 1: 148.391 347.422 9
id: x,y,label: 5: 118.24 118.24 0
self_position: 148.391, 347.422, 0.996064,
enemy_position: 449.282, 147.644, -1.71706,
is 100, 100 free? 1
is 300, 200 free? 0
```

Figure 2.5: Locating all objects.

Chapter 3

Path planning

After we get the position information of car, obstacles, and boundary by image processing, if we want to hide and attack using laser, we really need to control car move from current position to expected position in a special space seeing Figure 3.1, For example, when we want to attack the opponent car behind the obstacles ,we need to catch up opponent car as soon as possible on the one hand, and on the another hand we should avoid the obstacles and boundary. But how to move car effectively in a special space is hard question. Robot path planning is used to find a collision-free sequence of motions (way point) between and an initial(current) position and a final(expected) position within a specified environment.

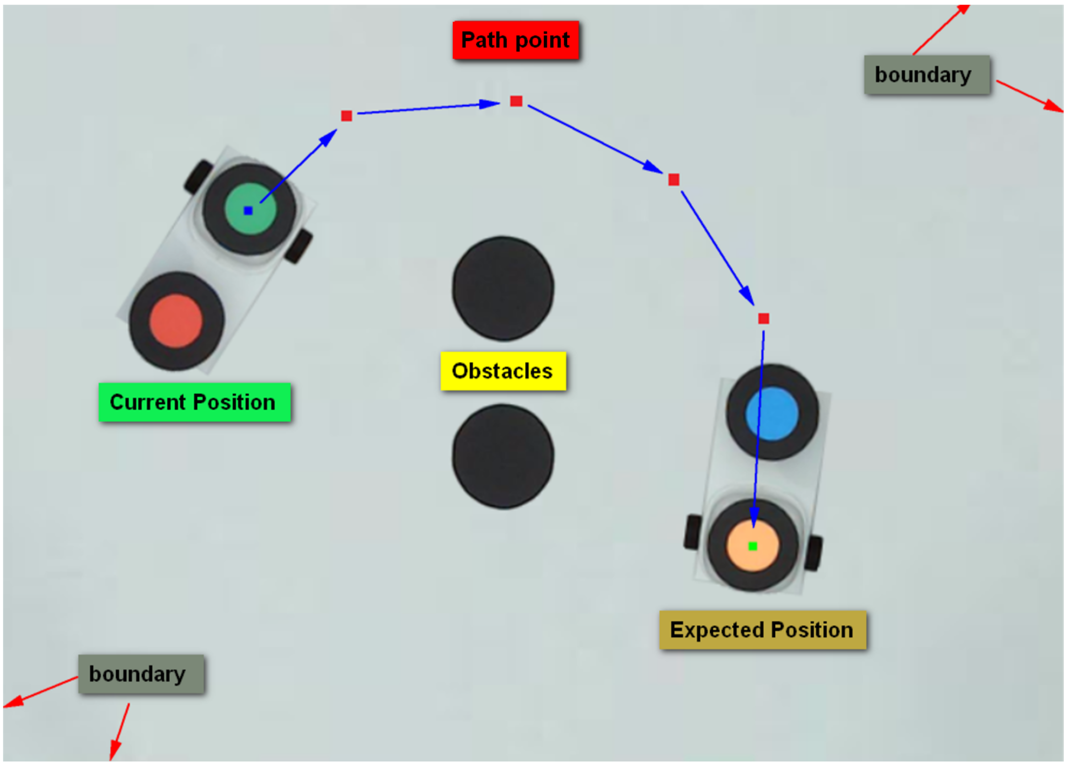


Figure 3.1: How to find the path point.

According to the special tasks and requirements of this project of MECH 6631 and after reading all kinds of path planning algorithms, a novel sampling and iterative based path planning algorithm (called Arc Sampling and Iterative Algorithm(ASIA)) is designed to used in this project.

3.1 The principle and process of ASIA

ASIA uses four steps to find and renew a new path point from current position to expected position.

1. **Step 1: Get sampling points**
   As shown in Figure 3.2, First, we need to build a body coordinate system. The body coordinate origin is current position , and axis X points the the expected position, and axis Y and axis X intersect at right angles. Second, initialize the value of sampling distance, sampling radian interval and total sampling radian. And get sampling sequence around current position.

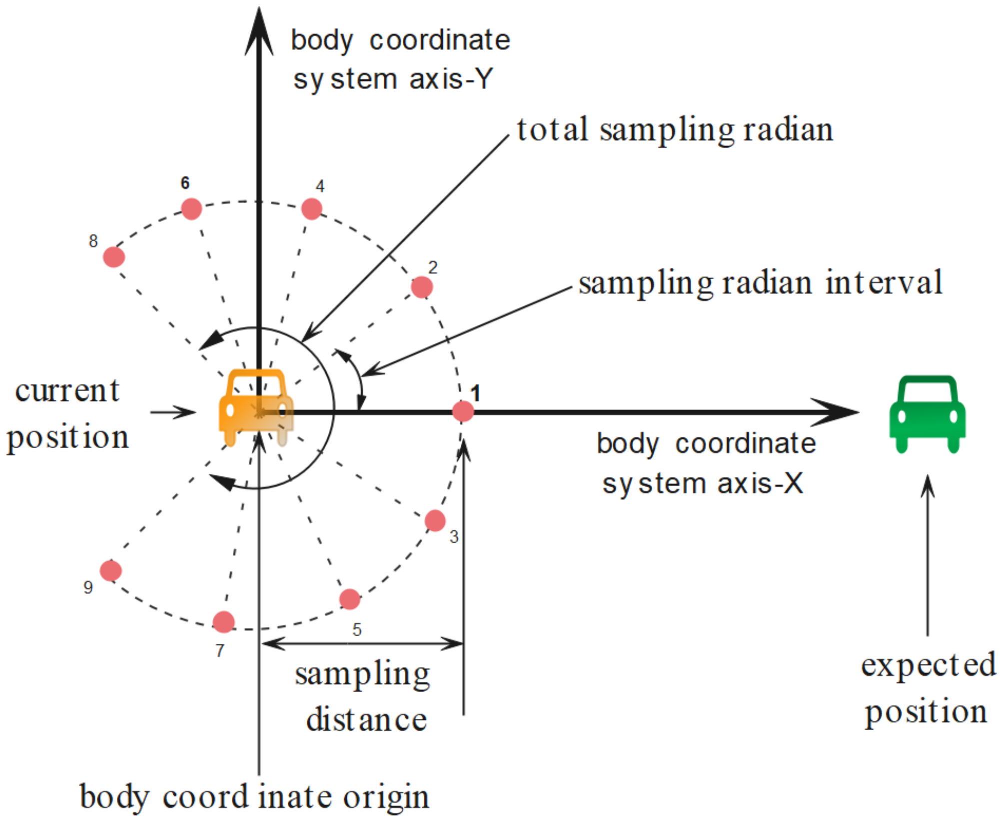


Figure 3.2: How to get sampling points.

2. **Step 2: Coordinate transformation**
   As seen in Figure 3.3,We need to translate sampling points from body

coordinate system to image coordinate system using rotation matrix after getting sampling points because we need to create path point sequence in image coordinate system.

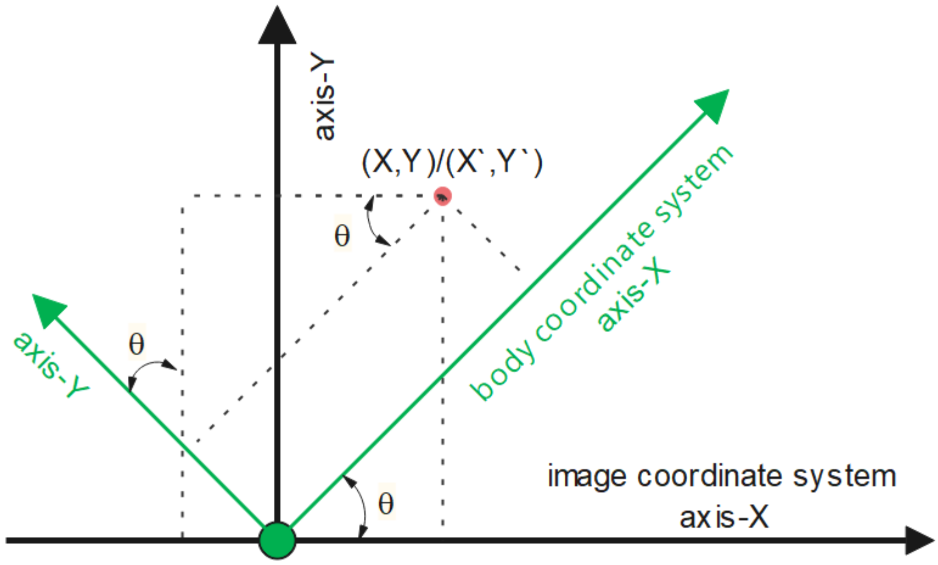


Figure 3.3: coordinate transformation.

According to the simple geometric knowledge in figure 3.3, we will get the transformation equation

$$\begin{cases} X = X' * cos\theta - Y' * sin\theta \\ Y = X' * sin\theta + Y' * cos\theta \end{cases} \tag{3.1}$$

If the origin between body coordinate system and image coordinate system is not in the same position——in other words the origin position of body coordinate system is (X_0, Y_0) in the mage coordinate system, a translation transform need to be added.

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} X' \\ Y' \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix} \tag{3.2}$$

3. **Step 3: Get path point**
   As seen in Figure 3.4,Judging if the sampling points(here it is represented by sampling points 1 9) is located in the free space sequentially (not coincide with the obstacle and is located within the image boundary). Once the conditions are met , the sampling point will be taken as the path point and renew current position, at the same time stop the judgement.

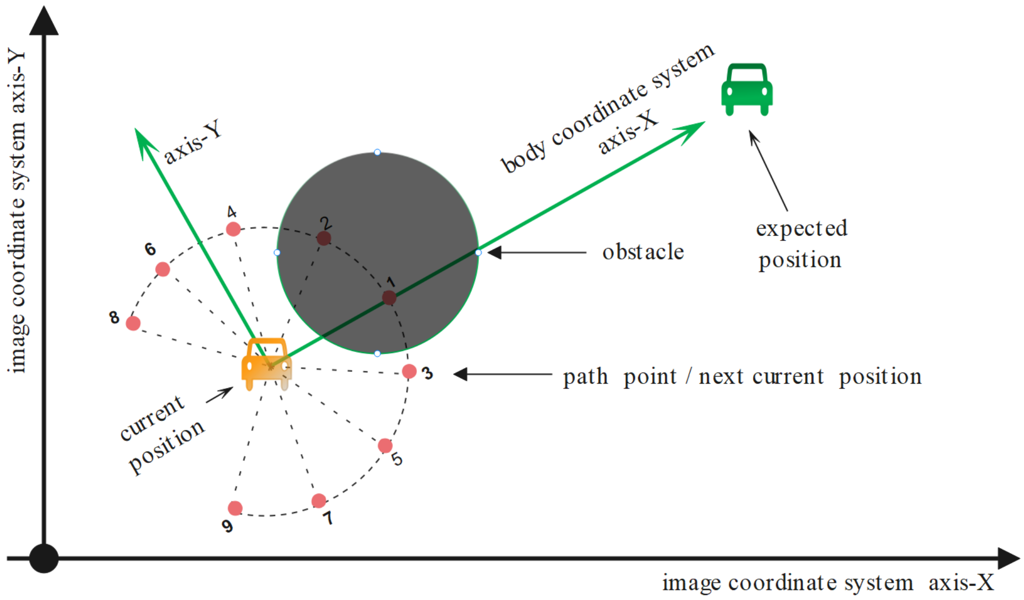
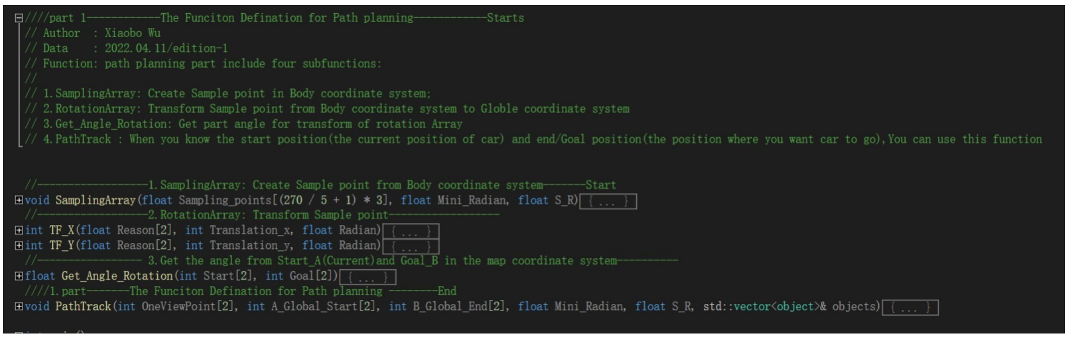


Figure 3.4: how to get path point and renew current position.

4. **Step 4: Iterate to get all of the path point**
   Iterate steps 1 3, until the distance between the current position and the expected position is less the the sampling distance. Up to now , we get all of the path point of path planing.

3.2 The implement of ASIA

There are two parts codes to implement the ASIA. As seen in Figure 3.5, there are four Sub functions Sampling _ Array, Rotation _ Array(including TF _ X and TF _ Y), Get _ Angle _ Rotation and PathTrack to support ASIA.



```
////part 1------------The Funciton Defination for Path planning------------Starts
// Author  : Xiaobo Wu
// Data    : 2022.04.11/edition-1
// Function: path planning part include four subfunctions:
//
// 1.SamplingArray: Create Sample point in Body coordinate system;
// 2.RotationArray: Transform Sample point from Body coordinate system to Globle coordinate system
// 3.Get_Angle_Rotation: Get part angle for transform of rotation Array
// 4.PathTrack : When you know the start position(the current position of car) and end/Goal position(the position where you want car to go),You can use this function


//------------------1.SamplingArray: Create Sample point from Body coordinate system-------Start
void SamplingArray(float Sampling_points[(270 / 5 + 1) * 3], float Mini_Radian, float S_R){ ... }
//------------------2.RotationArray: Transform Sample point------------------
int TF_X(float Reason[2], int Translation_x, float Radian){ ... }
int TF_Y(float Reason[2], int Translation_y, float Radian){ ... }
//----------------- 3.Get the angle from Start_A(Current)and Goal_B in the map coordinate system----------
float Get_Angle_Rotation(int Start[2], int Goal[2]){ ... }
////1.part-------The Funciton Defination for Path planning --------End
void PathTrack(int OneViewPoint[2], int A_Global_Start[2], int B_Global_End[2], float Mini_Radian, float S_R, std::vector<object>& objects){ ... }
```

Figure 3.5: implement of ASIA Using four sub functions.

As seen in Figure 3.6, Call the path planning / PathTrack, sub function in the main function.

```
std::cout << "is 100, 100 free? "
    << check_space(objects, 320, 400) << std::endl;
std::cout << "is 300, 200 free? "
    << check_space(objects, 300, 200) << std::endl;


/// Part 2.---------------------Track object for Path planning---------------------Start
/// Author  : Xiaobo Wu
/// Data    : 2022.04.10
int A_Global_Start[2] = { self_position_x, self_position_y };                 //initiallze a dynamic input parameter
int B_Global_End[2] = { enemy_position_x, enemy_position_y };
int* OneView = new int[2];                                                     // initialize a dynamic Array
PathTrack(OneView, A_Global_Start, B_Global_End, 3.1415926 / 19, 20, objects);  // realize the one-step viewpoint path track
draw_point_rgb(rgb, OneView[0], OneView[1], 225, 0, 0);                        // draw the view point to track/draw the goal point
////part 2.------------------Track object for Path planning----------------END


//free_image(map);
//   Image processing done ---------------------------------------------
```

Figure 3.6: Call the path planning algorithm.

Chapter 4

Robot Control

Chapter 5

Hiding and Attacking Strategies

As we all know , there are two tasks everyone robot / program should be able to complete on the vision simulator. First, Your robot chases the opponent robot and tries to hit it with the laser while the other robot tries to avoid getting hit. Second, The opponent robot chases your robot and tries to hit it with the laser while your robot tries to avoid getting hit. We find that obstacles are considered to block the laser so we give the following hiding and attacking strategies.

5.1 Hiding

Always goushi hide self robot behind a obstacle and keep two robots and obstacle at a straight line, as shown in figure 5.1:

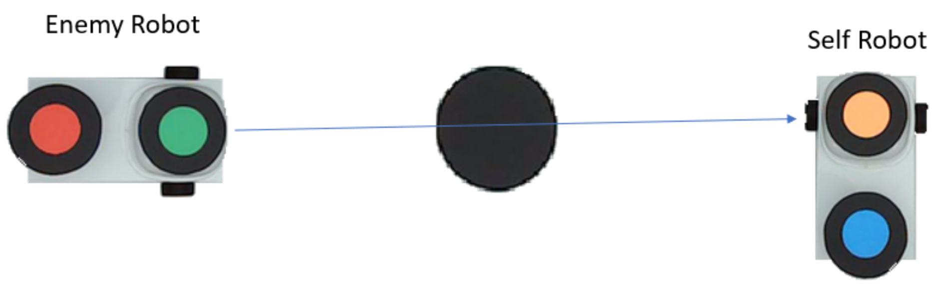


Figure 5.1: an example of positions that satisfies the hiding strategy.

5.2 The tactics of hiding

In order to implement the strategies above, detail tactics will be introduced. As we can see in the figure 5.2. Here we assume that there are three obstacles (1 3 obstacles in the environment randomly placed in the central part of the environment) in workspace. The red points are the centre of obstacles and robot, O point is the centre of opponent robot and P point is the centre of my robot. In hiding model, my robot need to move to hiding point behind the obstacles quickly. But how to pick the specific hiding point is a difficult problem to be solved. There is a feasible solution to be introduced. Three straight lines (auxiliary lines) pass through the center of the opponent robot and the center of the obstacle, they are line OD, line OE and line OF. The vertical feet are point A, point B and point C, when make vertical lines from point P to these three lines— line OD, line OE, line OF. My robot will regard C point as the best hiding point. Compared with other hiding points (A position and B position) C is the nearest point where to the current my robot position. According to the position of obstacles and robot (my robot and opponent robot), the algorithm of hiding model will update the dynamic hiding point to avoid the hit and attack from opponent robot.

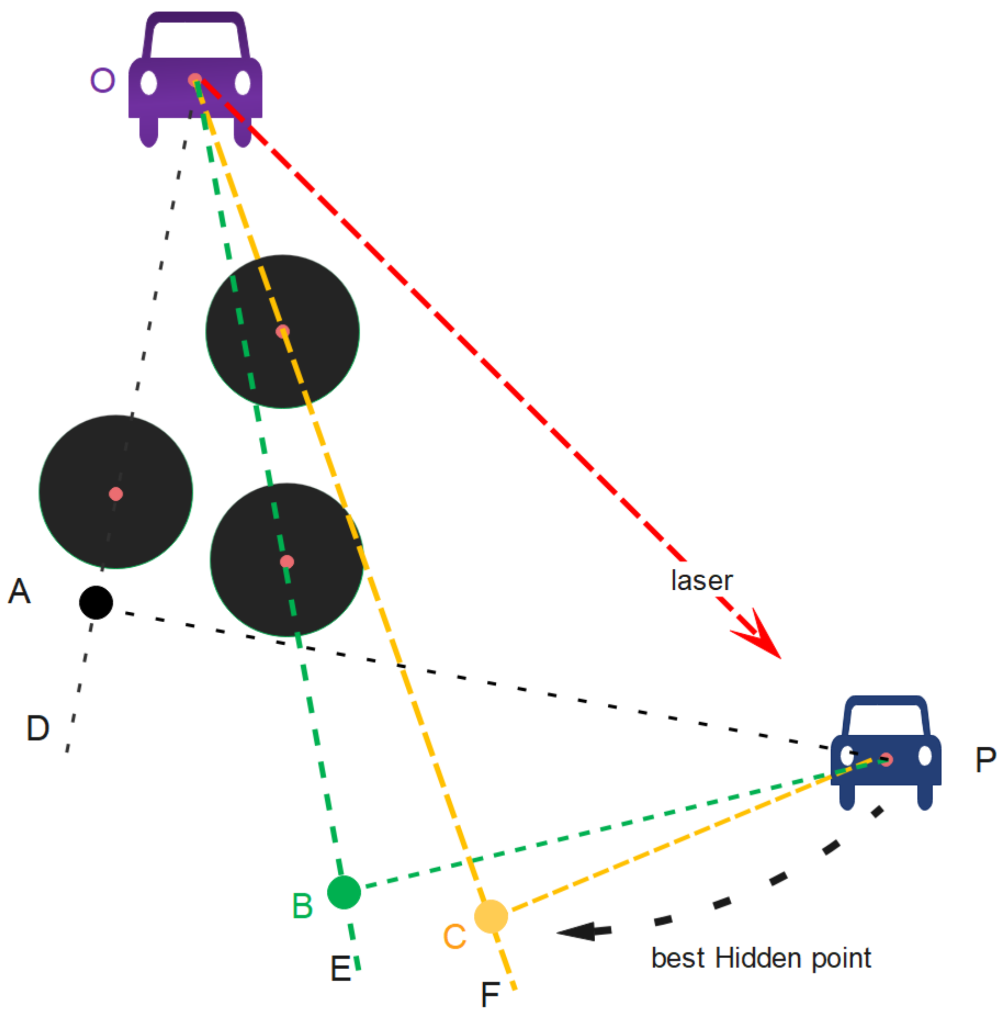


Figure 5.2: an tactics for hiding strategy.

5.3 The implementation of hiding

In order to find the nearest point that satisfies the condition, an obstacle between the self and enemy, the foot of perpendicular from current position to the straight-line connecting enemy and obstacles, as shown in figure 5.3

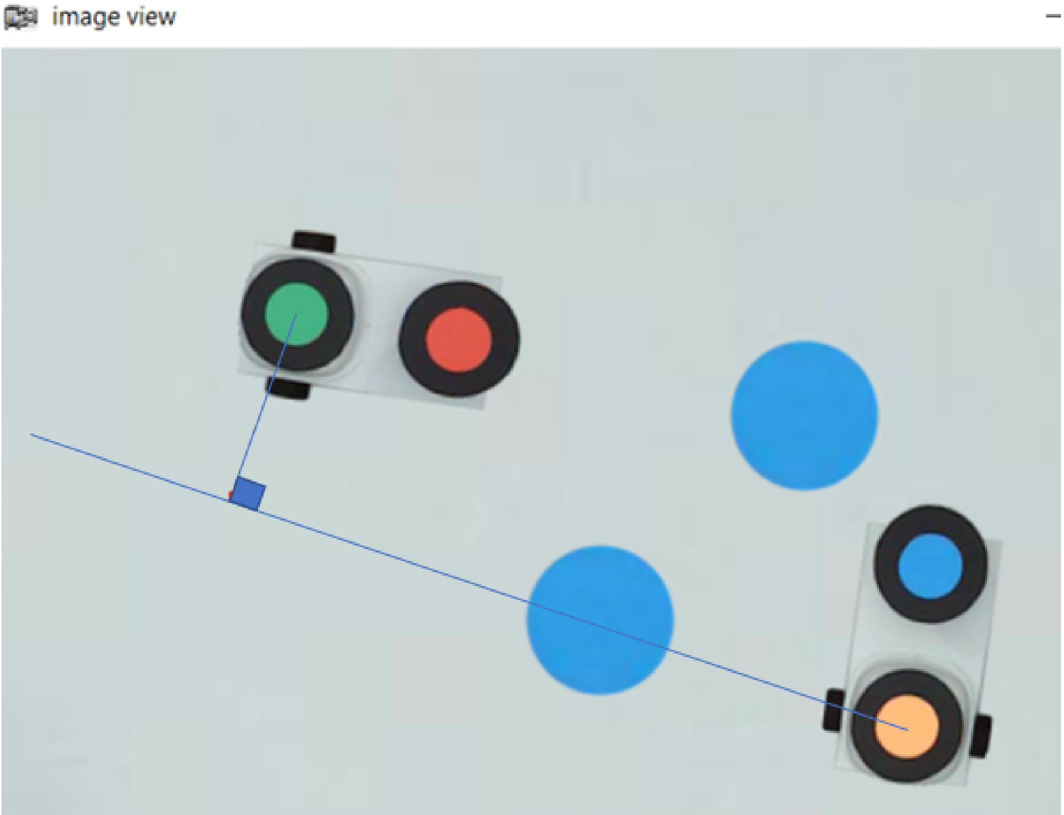


Figure 5.3: he foot of perpendicular from current position to the straight-line connecting enemy and obstacles.

The calculation of the foot of the perpendicular is as follows: We assume that $A = (x_a, y_a)$, $B = (x_b, y_b)$ are position of enemy and obstacle as shown in figure 5.4. $C = (x_c, y_c)$ is the position of self-robot, $O = (x_o, y_o)$ is the foot of perpendicular from point C to line AB.

Since $\overrightarrow{AB} \perp \overrightarrow{CO}$, we have:

$$(x_b - x_a)(x_o - x_c) + (y_b - y_a)(y_o - y_c) = 0 \tag{5.1}$$

Since $\overrightarrow{AB}$ and $\overrightarrow{AO}$ has same direction, we have:

$$\begin{cases} x_o = k(x_b - x_a) + x_a \\ y_o = k(y_b - y_a) + y_a \end{cases} \tag{5.2}$$

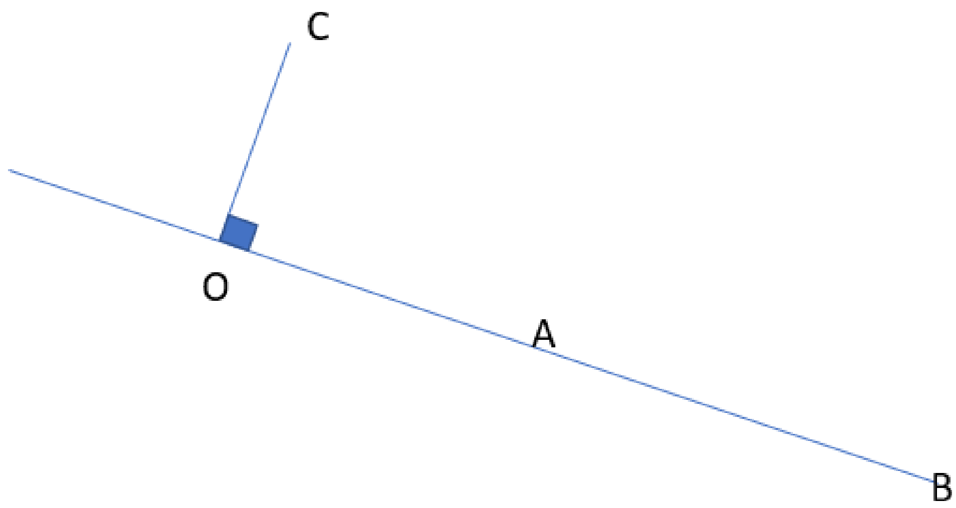


Figure 5.4: The foot of perpendicular.

Substitute x_o and y_o in the second equation to find out k. Then substitute k in the third equation to find out x_o and y_o.

$$k = -\frac{(x_a - x_c)(x_b - x_a) + (y_a - y_c)(y_b - y_a)}{(x_b - x_a)^2 + (x_b - x_a)^2} \tag{5.3}$$

Once we have arrived at the expected position, direction of the self-robot is also important. As shown in figure 5.5 a, if the direction of self-robot is not opposed to the enemy-robot, to run away from the enemy, the self-robot should rotate first, and then move around the obstacle. However, if we keep the direction of self-robot as opposed to the enemy, as shown in figure 5.5 b, we do not need to rotate robot first, we can move robot around the obstacle directly instead.

Figure 5.5: The importance of self-robot direction

Therefore, to keep the direction as opposed to the enemy, the expected position also requests a specific theta. For the convenience of calculation, we should stipulate that the theta of robot is between pi and -pi as shown in figure 5.6.

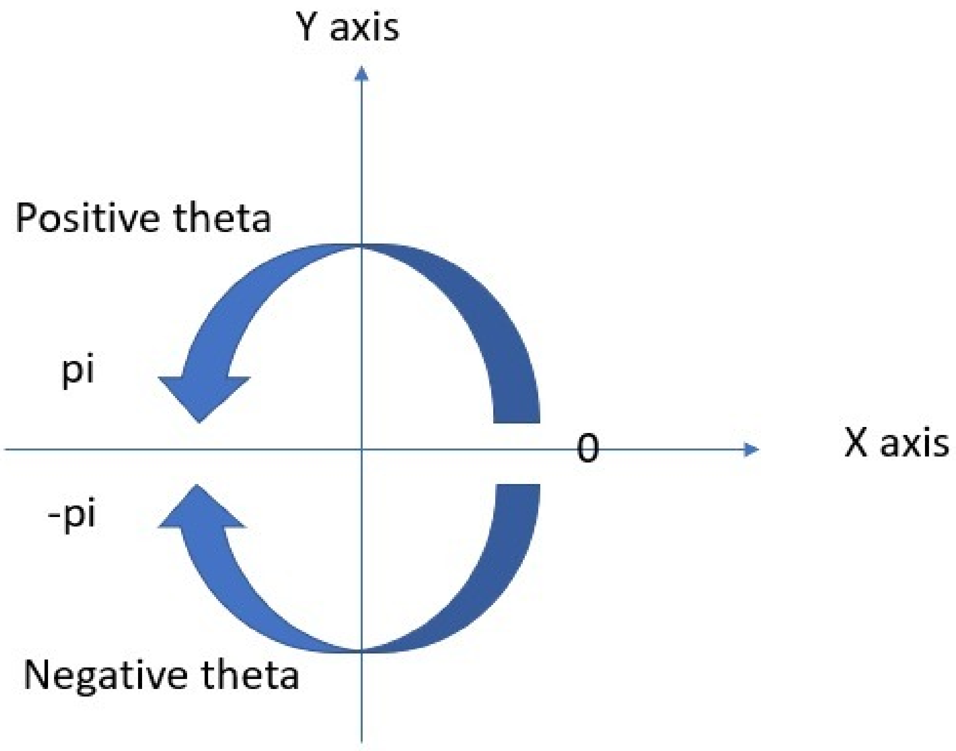


Figure 5.6: The representation of theta.

5.4 Attacking

Since there is only one chance to fire the laser, it should fire until there is no obstacle between two robots. In order to do so, attacking robot should chase another robot.

Method 1: Move the robot to the position of another robot as soon as possible.

Method 2: Searching the nearest point that is at the straight line with another robot without an obstacle as shown in figure 5.7:

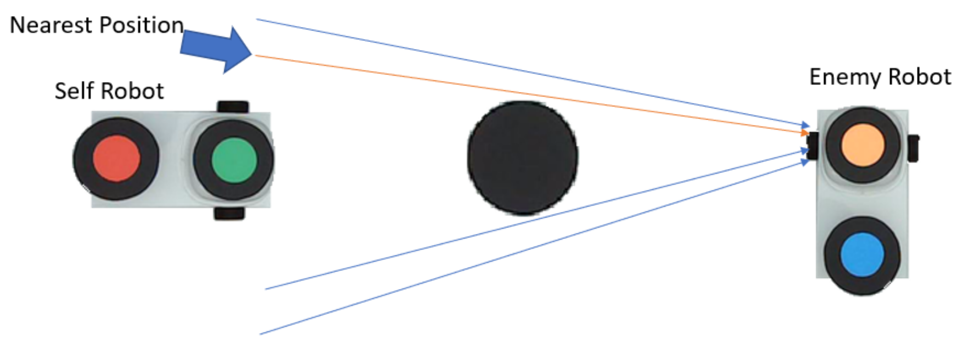


Figure 5.7: an example of positions that satisfies the attacking strategy.