EBS 289K

Homework 5

Part 1

The main purpose of this assignment is to write a function to implement the occupancy grid and test it. Function **updateLaserBeamGrid.m** is modified based on **updateLaserBeamBitmap.m**. There is a global variable, pOcc which is the probability related to measuring. Currently, pOcc sets to be 0.9. In this assignment I will test my function several times by add some lasers to different locations.

Original Laser Location:

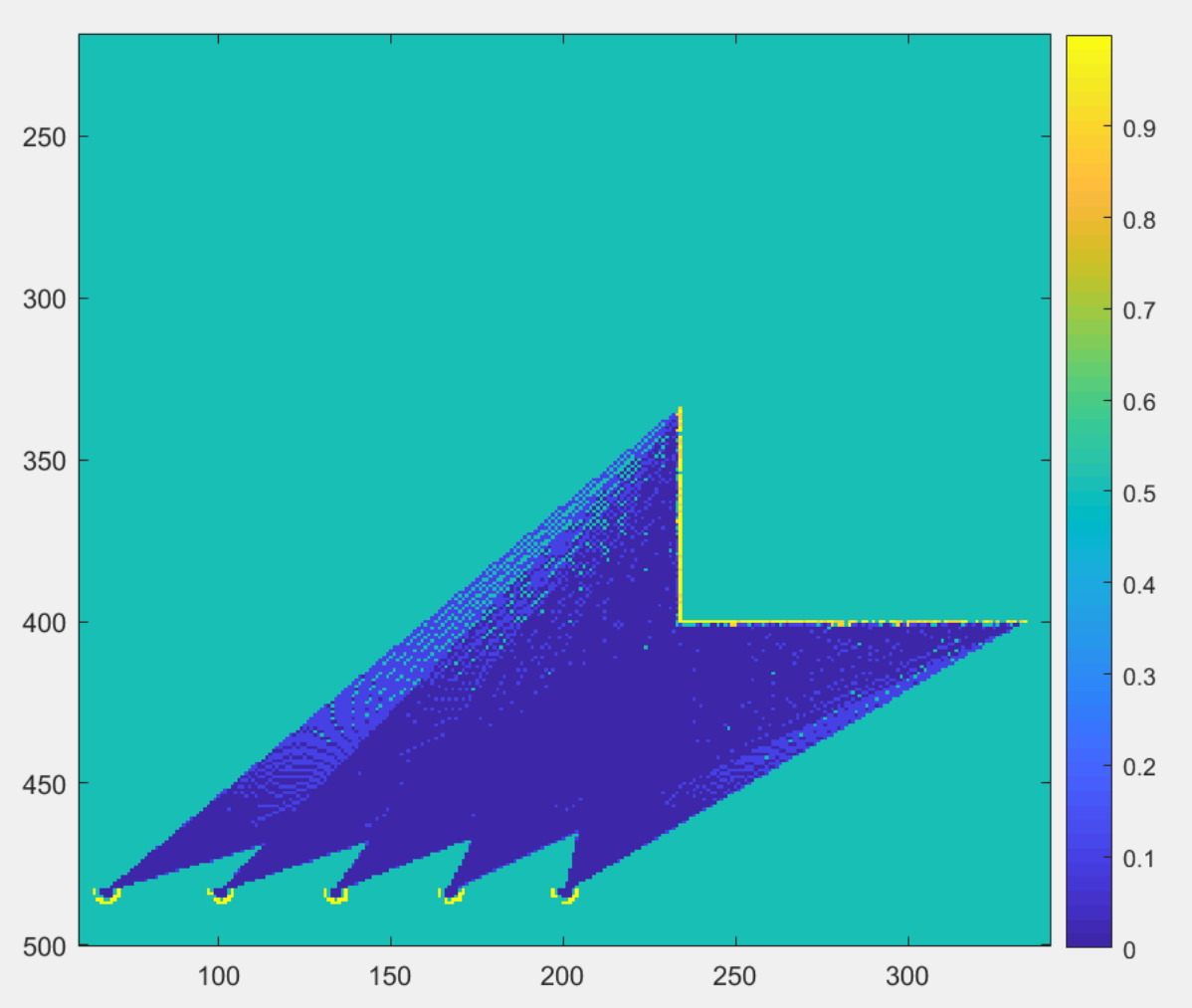

Figure 1 Occupancy grid of original lasers

In figure 1, 5 lasers detected an obstacle in the middle area. The boundary of this obstacle is yellow since the occupancy of those grids are close to 1. The yellow boundary means we are pretty sure there is an obstacle has two vertical edges. What about the other sides of the boundary? Let's add more laser.

Add five more lasers to the right

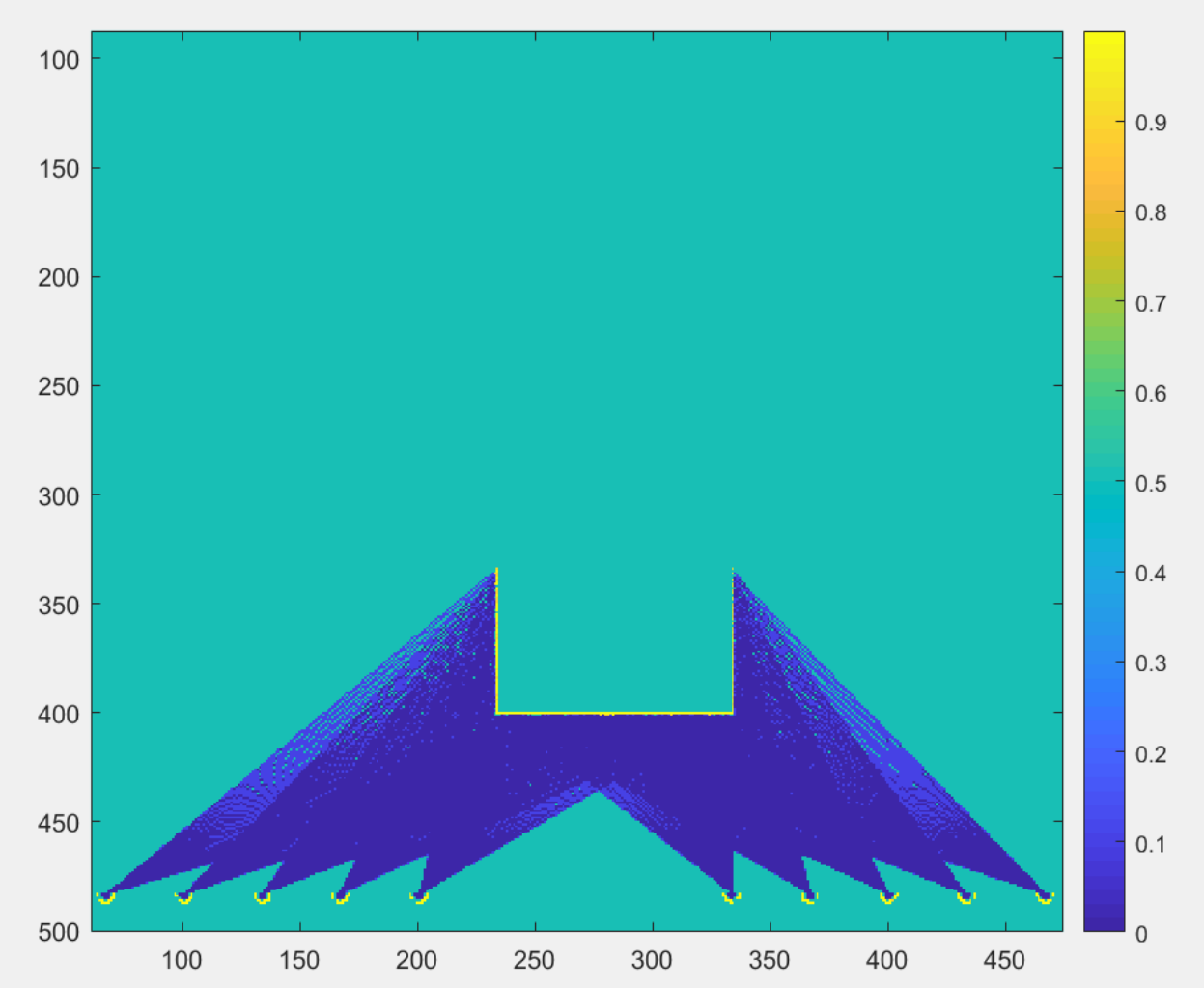

Figure 2 10 lasers at the bottom

If we add five more laser to the right, we can see the right boundary of the obstacle. But we still have no knowledge about the top of this obstacle. So let's 5 lasers on the top of the grid and set them to look down.

Add five more lasers on the top of the grid map

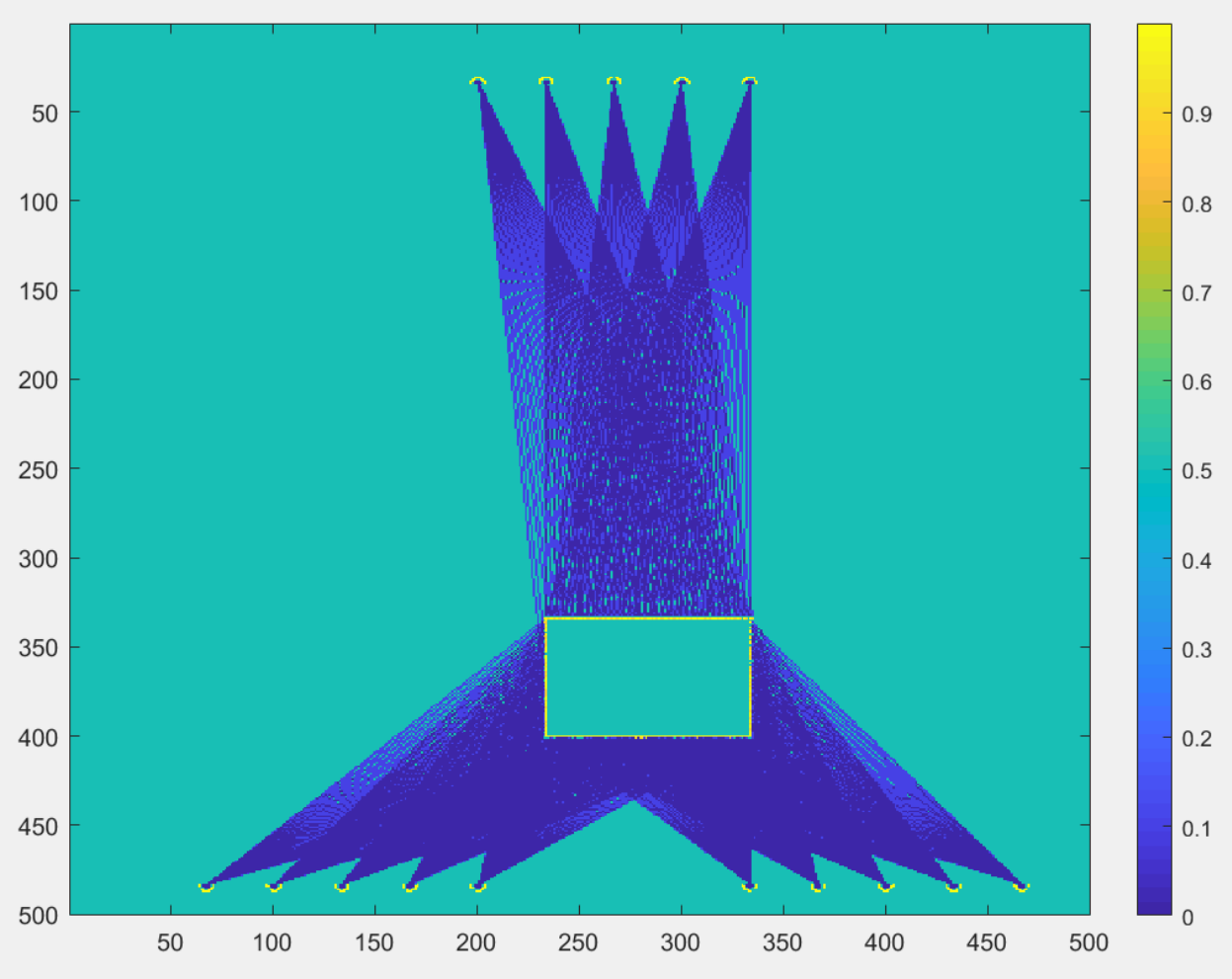

Figure 3 Occupancy grid of three groups of lasers

Added five lasers on the top of the grid, we can see the all boundaries of this obstacle. For all the experiments, the **updateLaserBeamGrid.m** function works well.

Part 2

Brief descriptions of functions that I added to homework 5.

hw5.m	main function used to initialize variables, create maps and start main loop
repulesPoints.m	function used to list all repulsive points in current window
virtualForceField.m	based on the repulsive points and its occupancy, compute repulsive and attractive forces.
updateOgrid.m	used a laser on the robot to update the occupancy grid map
ackermann.m	ackermann model
updateLaserBeamGrid.m	laser beam grid map function

Table 1 functions of my codes

ABOUT MY CODE:

1. Hw5.m
   Two situations:
   situation 1 have one obstacle for hw5 part 3;
   situation 2 have eight obstacles for hw5 part 4.
   I have tuned all parameters. No parameter needs to be changed to run my code for both situation 1 and situation 2.
   All you need is to change situations to see how the robot runs.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% There are two situations:
% choose situation 1: 1 obstacle,
% choose situation 2: 8 obstacles.
% situation = 1;
situation = 2;
```

2. repulesPoints.m
   There is a trick here. Since the number of repulsive points is influenced by the resolutions of occupancy map and display map. The number of repulsive points may change dramatically with different resolution which have a great impact on the repulsive force. The variables related to repulsive force would be really hard to tune and may only specified to certain condition. In order to generalize variables, in this experiment, I map the occupancy points to display map rounded to one decimal. Thus, no mater how big the resolution of the occupancy grid map is, the number of repulsive points would not change rapidly. In other words, repulsive points in one area may be considered as one point while computing the repulsive force.
3. virtualForceField.m
   Nothing special in this function but the repulsive force function. While tuning, I found it would be better to increase the repulsive force when the robot is running too close to the obstacle. Instead of divided be distance square, I define the repulsive force to be a function divided by distance over 4. With the new definition, the robot seems more sensitive when it is too close to obstacles.

```
f_r_x = occupancy*fcr*(start_x - repules(i,1))/d^4;
f_r_y = occupancy*fcr*(start_y - repules(i,2))/d^4;
```

Part 3

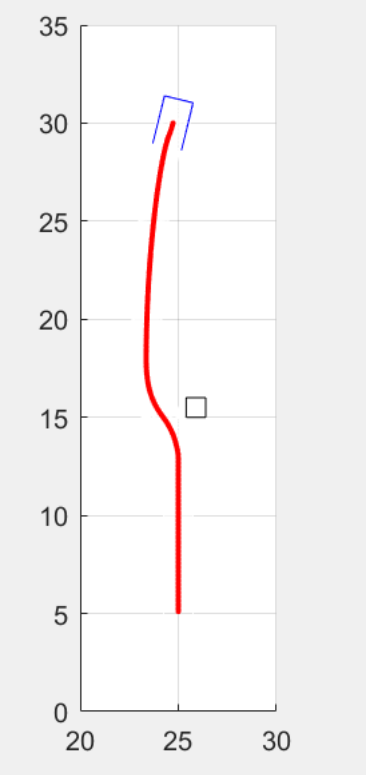

Figure 4 The trace for one obstacle

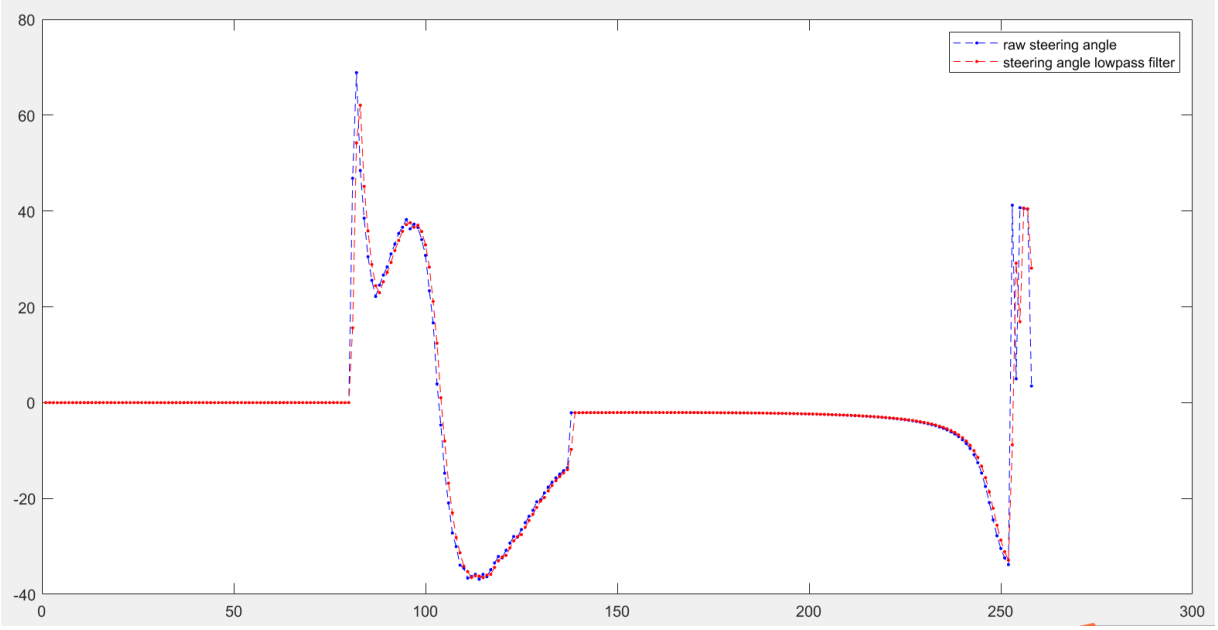

Figure 5 steering angle with and without low pass filter

Part 4

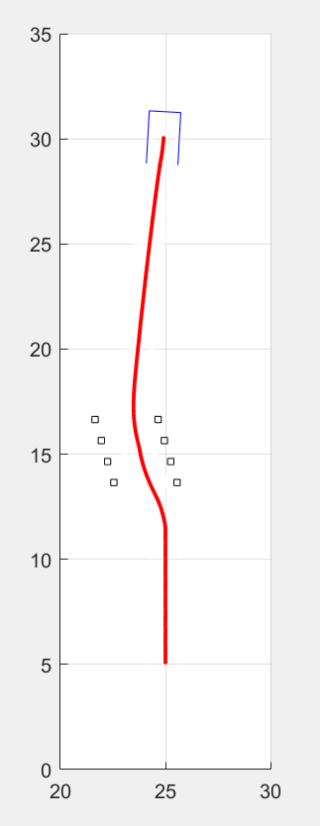

Figure 6 The trace for eight obstacle

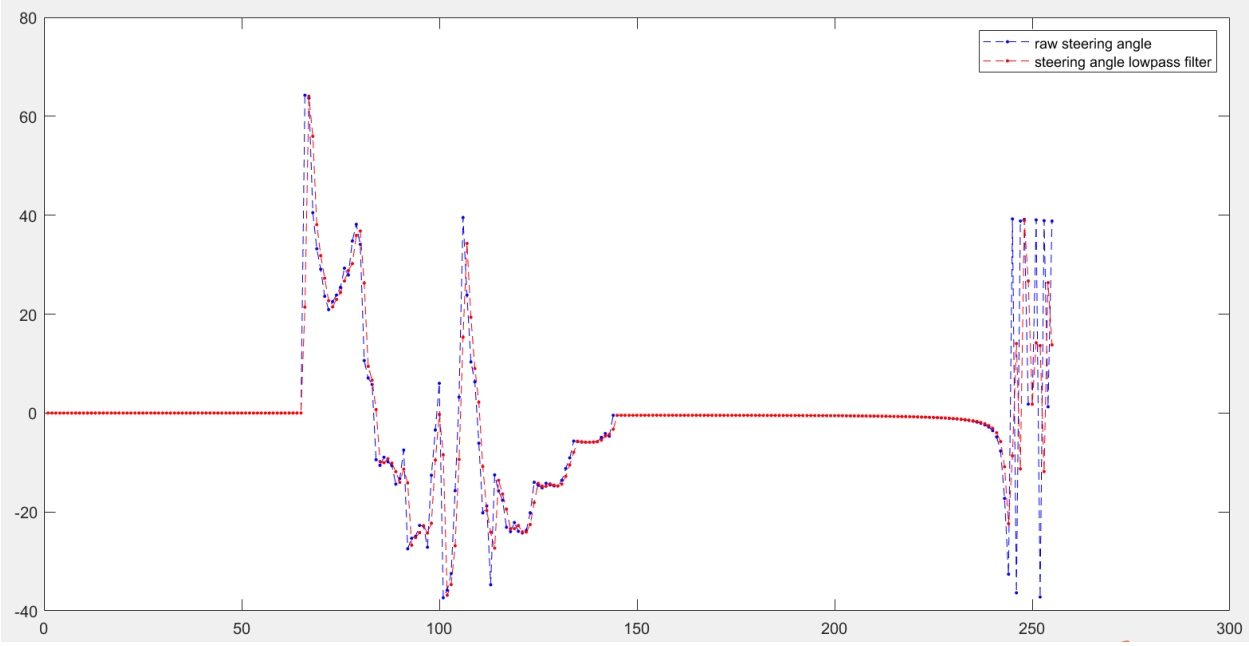

Figure 7 steering angle with and without low pass filter

Comparison and Analysis

1. Original and new repulsive force function:
   Instead of using the original repulsive force function, I define a new repulsive force function by divided distance power of 4 to make robots more sensitive to obstacles which are too close to the robot. In figure 8, the left one shows the trace using original forces, the right one shows the trace using my forces.
   Here is the comparison:

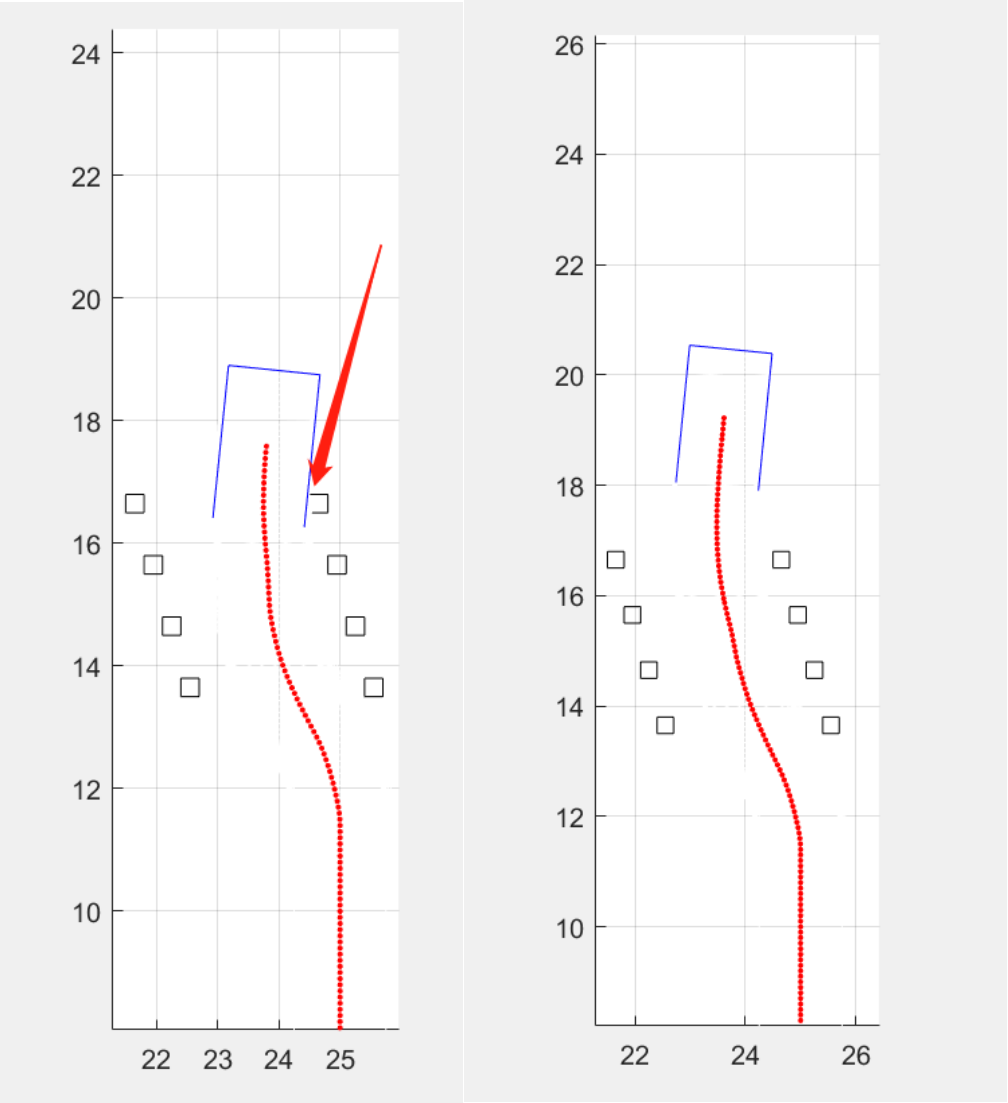

Figure 8 Comparison between two repulsive force function

   In figure 8, the left robot used the origin force function and there is a crash in the end, the right robot used my new repulsive force function and it works well.

2. With and without speed control
   In this experiment, speed control plays an important role for avoiding obstacles. Speed would decrease when facing obstacle. The slowdown of velocity gives the robot more change to turning itself. In figure 9, the left one shows the trace without speed control and there is a crash, the right one shows the trace with speed control and it works well.

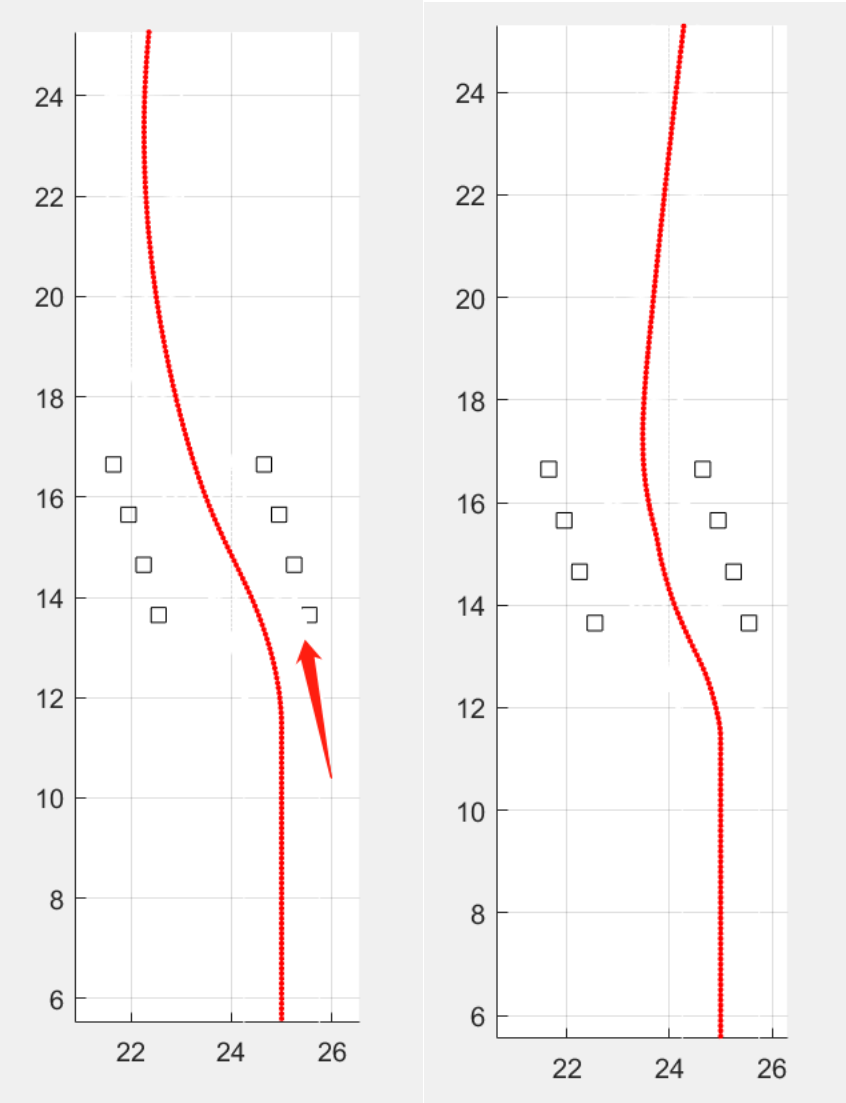

Figure 9 with and without speed control

3. With and without velocity low pass filter
   To avoid all obstacles, the low pass filter was adjusted to have a small amount of affected on the whole system. In figure 10, the left one shows the trace without low pass filter, the right one shows the trace with low pass filter. Here is the comparison:

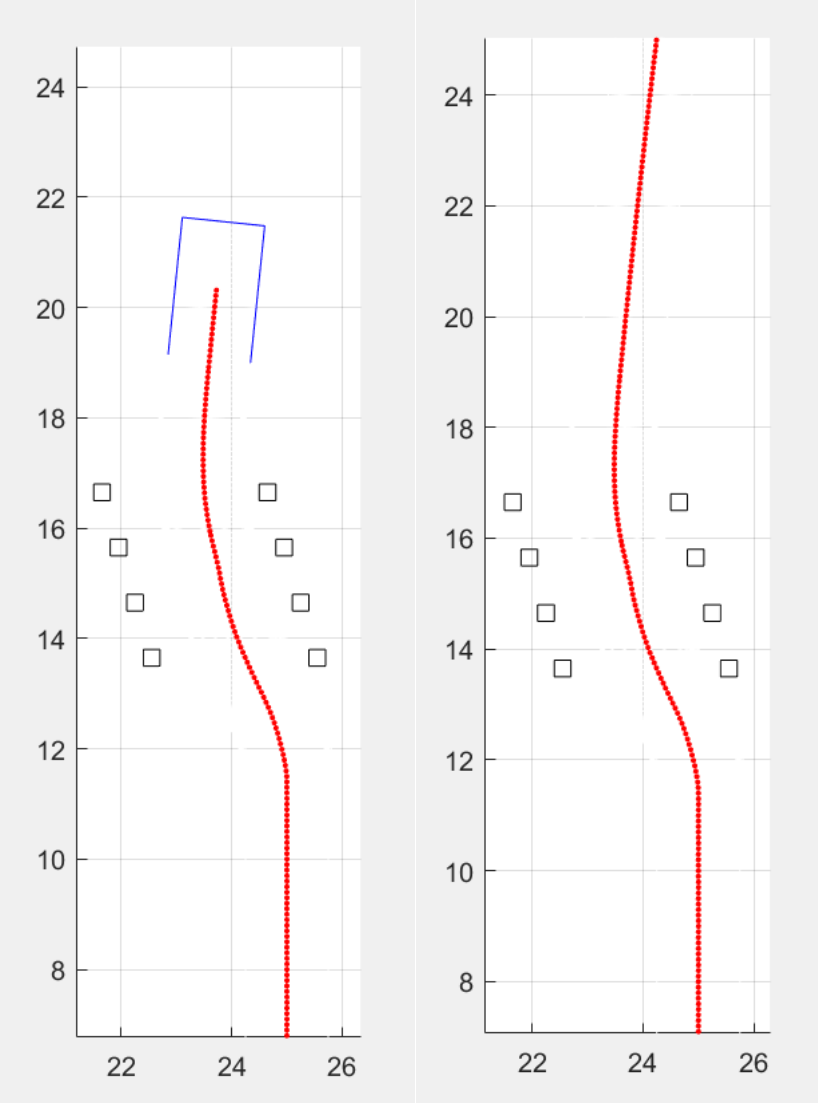

Figure 10 with and without low pass filter for steering angle