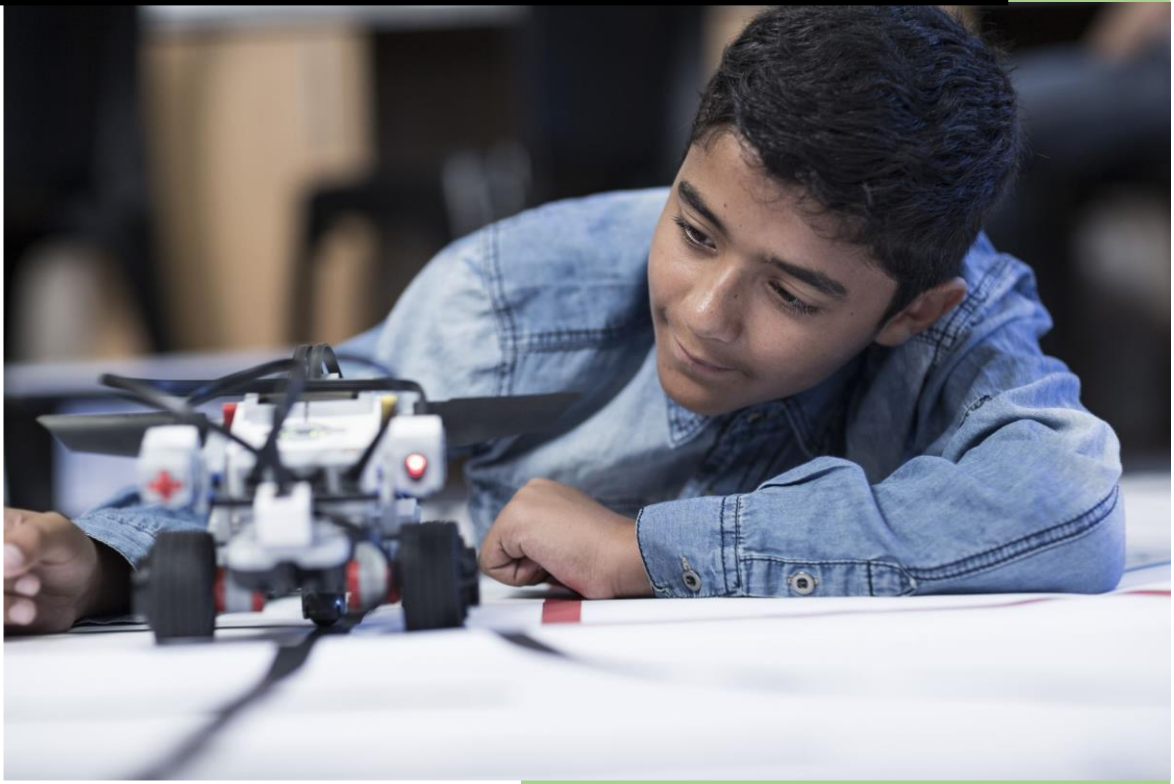
2022

Project #2 Report



Gabriel Mortensen

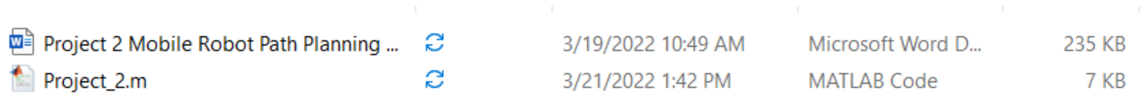
Introduction

In this project I was given the initial code that would allow me to “design and implement an Artificial Potential Controller to allow a mobile robot to follow/track a virtual moving target.” The main pieces missing from the provided code concerned the orientation and velocity of the robot. By using the coding instructions by the professor, I was able to implement the proper equations that allow the robot to follow the virtual target. The following paper will provide descriptions of each test case provided in the instruction pdf, explanations for the four graphs provided with the professor’s portion of the code, and several test cases that modify important variables while also describing their impact.

Prior Statement | How to Run | Overall Structure of Files (a)

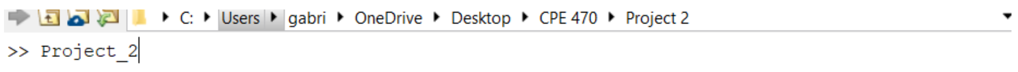
In the document provided by the professor it notes, “Write Matlab or C/Cpp code (or other languages you prefer) to implement your designed potential controller.”

This project was coded using MatLab and can be executed by typing ‘Project_2’ into the command prompt (or click the run button). During the run time of the code the user (TA) can select various options that will provide a series of different graphs and scenarios. The files associated with this code can be in Project_2_Code.zip. The following figures further detail how the files are organized and how the code can be executed.



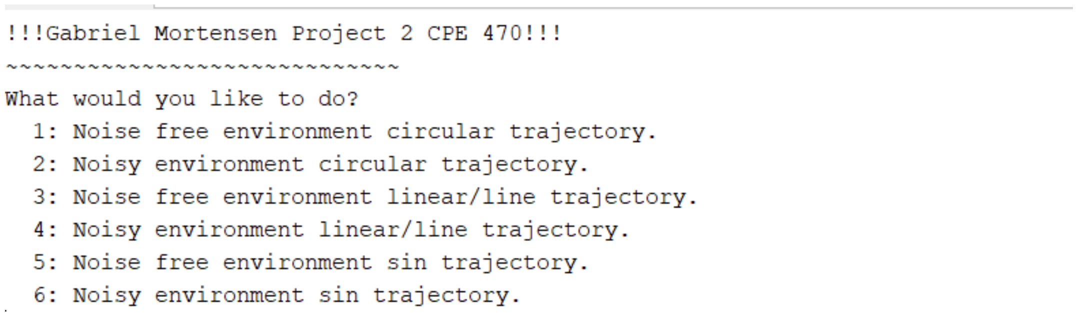
Project 2 Mobile Robot Path Planning ...	3/19/2022 10:49 AM	Microsoft Word D...	235 KB
Project_2.m	3/21/2022 1:42 PM	MATLAB Code	7 KB

Fig.a.1: Files within the zip file



```
C: ▸ Users ▸ gabri ▸ OneDrive ▸ Desktop ▸ CPE 470 ▸ Project 2
>> Project_2
```

Fig.a.2: Command line in prompt used to run the code.



```
!!!Gabriel Mortensen Project 2 CPE 470!!!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
What would you like to do?
  1: Noise free environment circular trajectory.
  2: Noisy environment circular trajectory.
  3: Noise free environment linear/line trajectory.
  4: Noisy environment linear/line trajectory.
  5: Noise free environment sin trajectory.
  6: Noisy environment sin trajectory.
```

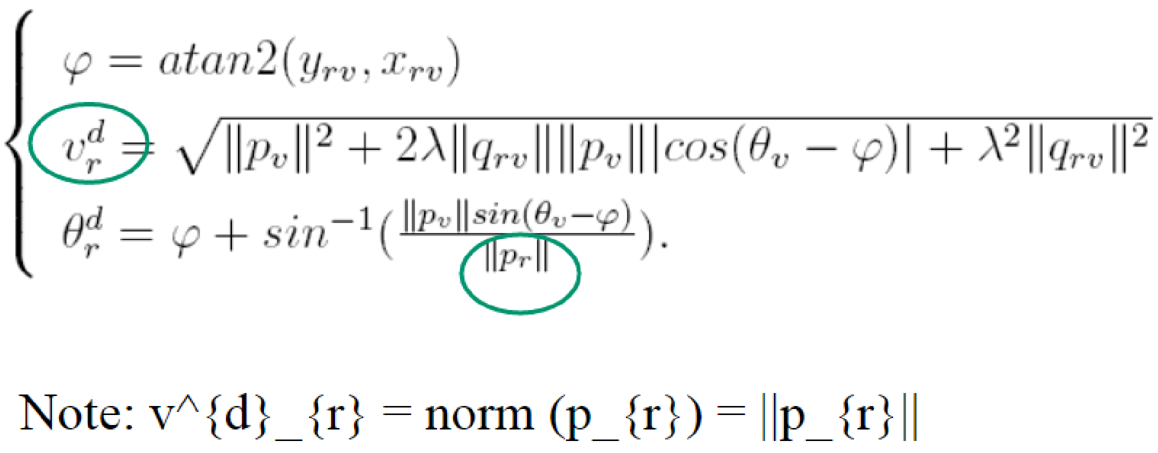
Fig.a.3: First instance of running the code where user is presented with menu.

Extra: Variable Significance and Information

Before examining the results of each case described by the instructions document it is important to establish the code and its meaning. Most of the code was moved over from the instructions power point provided to students in lecture. The provided code allowed for the virtual target to move in a circle with and without noise, both of which can be observed via the first two menu options. For this project students had to implement the following into the code:

1. Have Robot follow the virtual target
    a. Update Phi during the for loop
    b. Use phi and other variables to update the velocity of the robot
    c. Use Phi and the velocity to calculate the heading r_theta of the robot
2. Have virtual target move in linear direction
3. Have virtual target move in linear direction with noise
4. Have virtual target move in sin wave pattern
5. Have virtual target move in sin wave pattern with noise

Step one could easily be achieved by comprehending the qrv was the relative positions between the robot and the virtual target. After that point the equations could simply be copied over from the instruction's pdf slide 5 shown in Fig.a.4



$$\begin{cases} \varphi = atan2(y_{rv}, x_{rv}) \\ v_r^d = \sqrt{||p_v||^2 + 2\lambda||q_{rv}||\,||p_v||\,|cos(\theta_v - \varphi)| + \lambda^2||q_{rv}||^2} \\ \theta_r^d = \varphi + sin^{-1}\left(\frac{||p_v||sin(\theta_v - \varphi)}{||p_r||}\right). \end{cases}$$

Note: v^{d}_{r} = norm (p_{r}) = ||p_{r}||

Fig.a.4: Equations needed to find Phi, robot velocity, and robot heading.

Aside from this information extra coding was done to ensure that the user can manipulate the following variables:

1. Max_vdr.
   a. This variable limits the maximum velocity of the robot via a min function in the for loop.
2. Lambda.
   a. This variable determines the attractive force of the virtual target.
3. Mean.
   a. This variable establishes the factor of shift in the data, how “off” the data is from its actual position.
4. Standard deviation.
   a. This variable determines the noise around the path.

Noise Free Environment (b)

For the noise free environment, the code was primarily provided by the professor in terms of the circular motion. All one needed to do in this case was set the x and y positions of the virtual target inside the for loop to have them moved at the desired predetermined path (i.e., linear or sin wave).

Movement in a linear/line trajectory

In this subsection I will first plot the tracking results of the following information without any modifications and explain their purpose:

1. Trajectories of the target and robot
2. Tracking error between the target and robot
3. Robot's heading
4. Robot's velocity

To observe these four graphs, one must direct their attention to Fig.b.1.

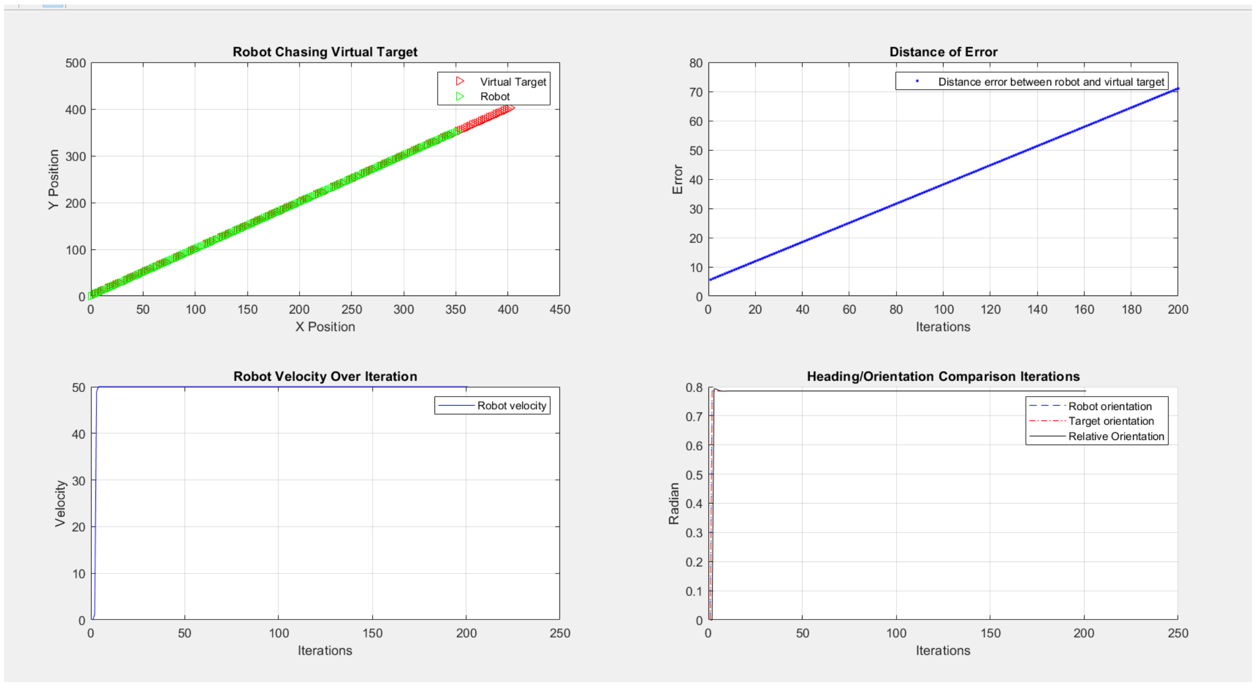


Fig.b.1: All four graphs outputted.

In this scenario the robot speeds up to meet with the virtual target then maintains a constant velocity as it is preprogrammed to not go beyond the virtual target. During this time the virtual target continues moving ahead of the robot and starts to steadily increase in speed such that the distance between the robot and virtual target becomes farther apart as shown by the end of the "chase" in the top left graph of Fig.b.1 where the green does not entirely overlap the red. Because of the increasing distance between the robot and the virtual target the error between the two entities also increases over the number of iterations as seen in top right graph of Fig.b.1. The robot's velocity is set at a max rate of 50, therefore in the robot velocity graph the robot rushes up to the virtual target and is then stuck at the limiter placed upon it as shown in bottom left graph of Fig.b.1.

In the bottom right graph of Fig.b.1 both the virtual target and the robot immediately spike up to a constant as they follow a fixed orientation the entire time of the program. Zooming into the bottom right graph (Fig.b.1.Zoom) one can observe that the robot has to slightly adjust itself by using the relative orientation.

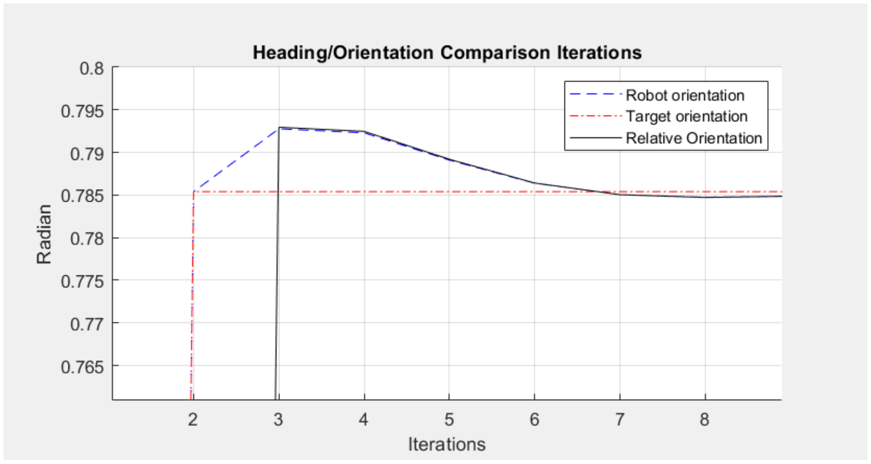


Fig.b.1.Zoom: The Heading/Orientation graph is zoomed in to show the robots adjustment to the sudden change of the virtual target.

Now consider a case where the velocity of the robot was increased along with the attraction force on the target. This scenario can be observed in Fig.b.1.2

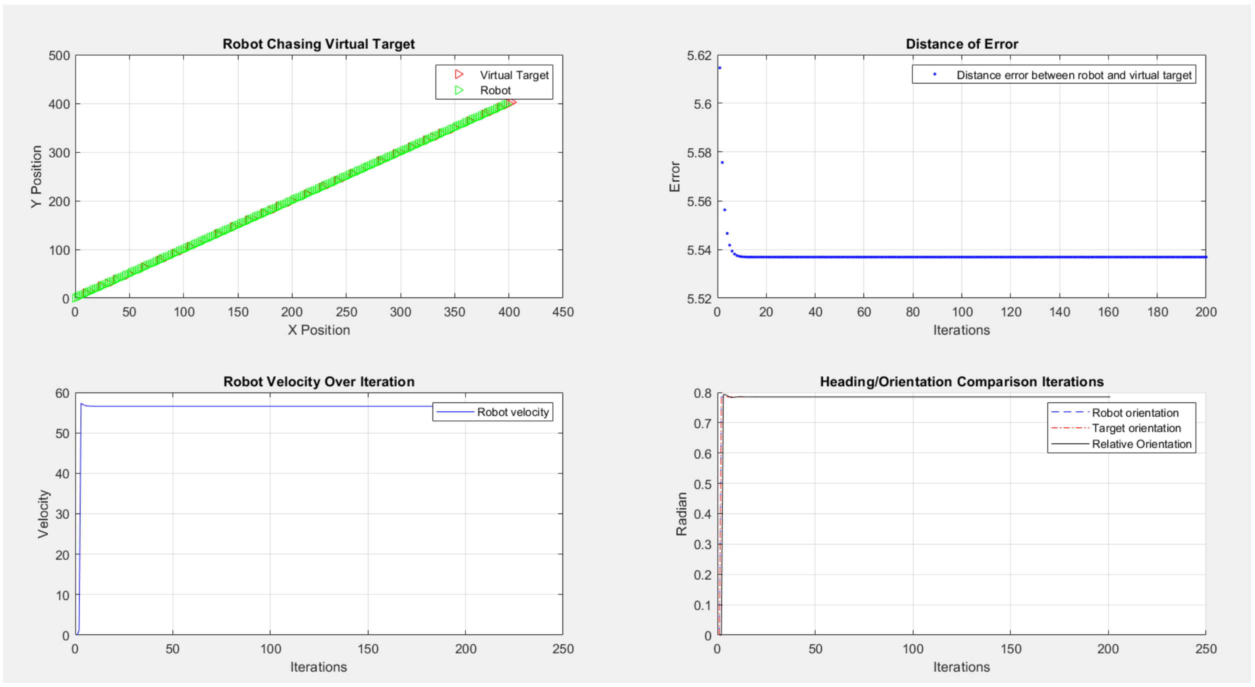


Fig.b.1.2: All four graphs outputted with max_vrd 120 (as opposed to 50) and lambda 10 (as opposed to 8.5).

Because the robot has an easier time matching the target's speed (velocity limiter) and position (lambda) there appears to be less error between the distance as noted in the top left graph where the error is decreasing over the number of iterations. Another area of interest concerns the heading and orientation comparison. If one were to zoom in for the Heading/Orientation graph of Fig.b.1.2 (noted as Fig.b.1.2.Zoom) they would find that the robot has to spend more iterations overcorrecting than in Fig.b.1.Zoom.

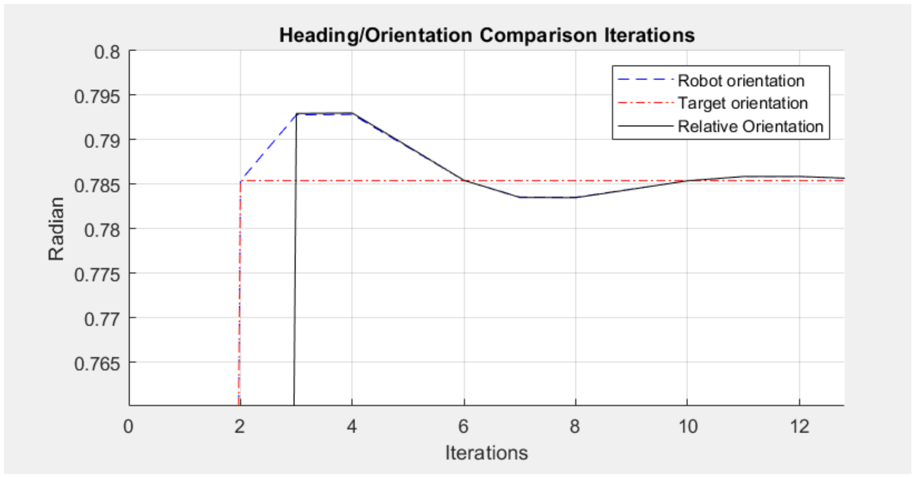


Fig.b.1.2.Zoom: The robot is more sensitive in orientation and has to correct itself again after iteration 6.

It should be noted that if the lambda is unmodified this overcorrection does not happen as shown in Fig.b.1.3.Zoom.

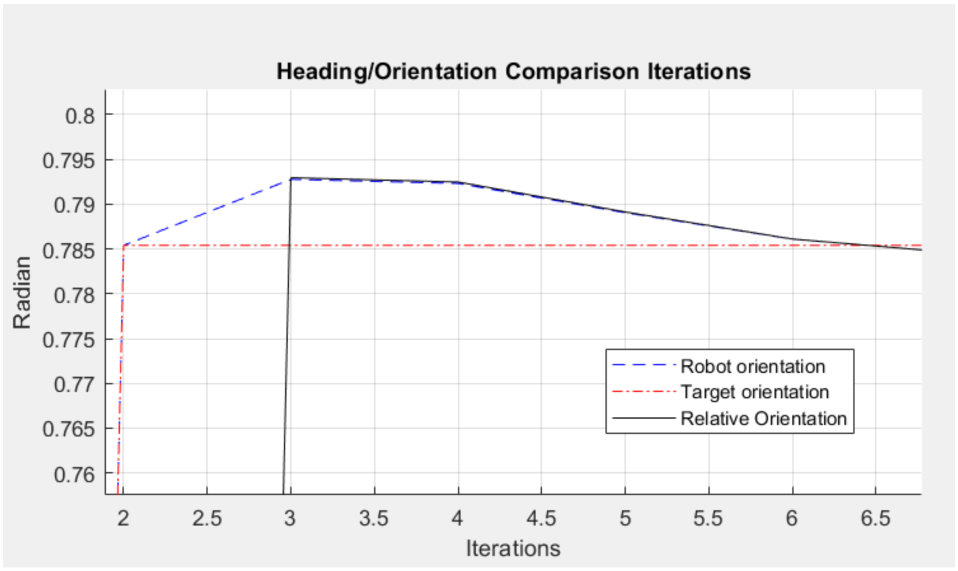


Fig.b.1.3.Zoom: Result of heading comparison graph with lambda = 8.5 and max_vrd = 120.

The reason for the overcorrection in Fig.b.1.2.Zoom is because the attractive force is so great that the robot corrects itself too quickly in response to the "tug" by the virtual target that is still moving.

The combination of the increased attractive force with the diagonal moving target causes the robot to require overcorrection as it is too sensitive to any slightly positional change in the virtual target. Overall, these figures represent an important note of how important each variable is to the contribution of the robot's ability to follow the virtual target.

Movement in a sin wave trajectory

In this subsection I will first plot the tracking results of the following information without any modifications and explain their purpose:

1. Trajectories of the target and robot
2. Tracking error between the target and robot
3. Robot's heading
4. Robot's velocity

To observe these four graphs, one must direct their attention to Fig.b.2.

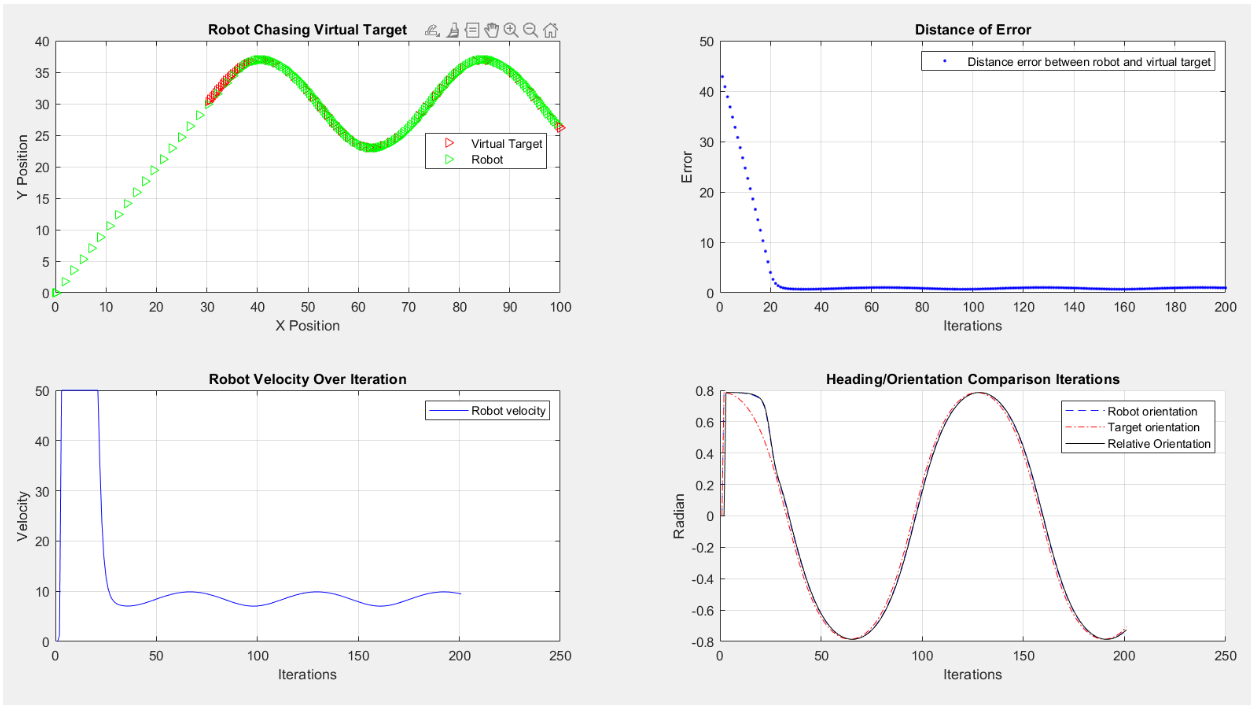


Fig.b.2: All four graphs outputted.

In this case the robot is able to catch up to the virtual target allowing less error to occur over the iterations as the robot closes the gap between itself and the virtual target. While following the virtual target on the wave pattern the robot makes frequent turns as noted by the heading and orientation graph while also having to modify its speed based on the position of the virtual target as shown in the robot velocity graph. The initial set up where the robot is trying to reach the target in a somewhat diagonal path causes the robot to turn a little later than the virtual target's initial orientation as shown in the top left section of the heading and orientation graph. The speed of the robot slightly affects the distance error as the virtual target can get slightly ahead of the robot at some points in the path making a smaller sin wave pattern for the error as shown in the error graph.

Let’s consider another case where the force of attraction is less potent yet the velocity of the robot is at its same limit. This case can be observed in Fig.b.2.1.

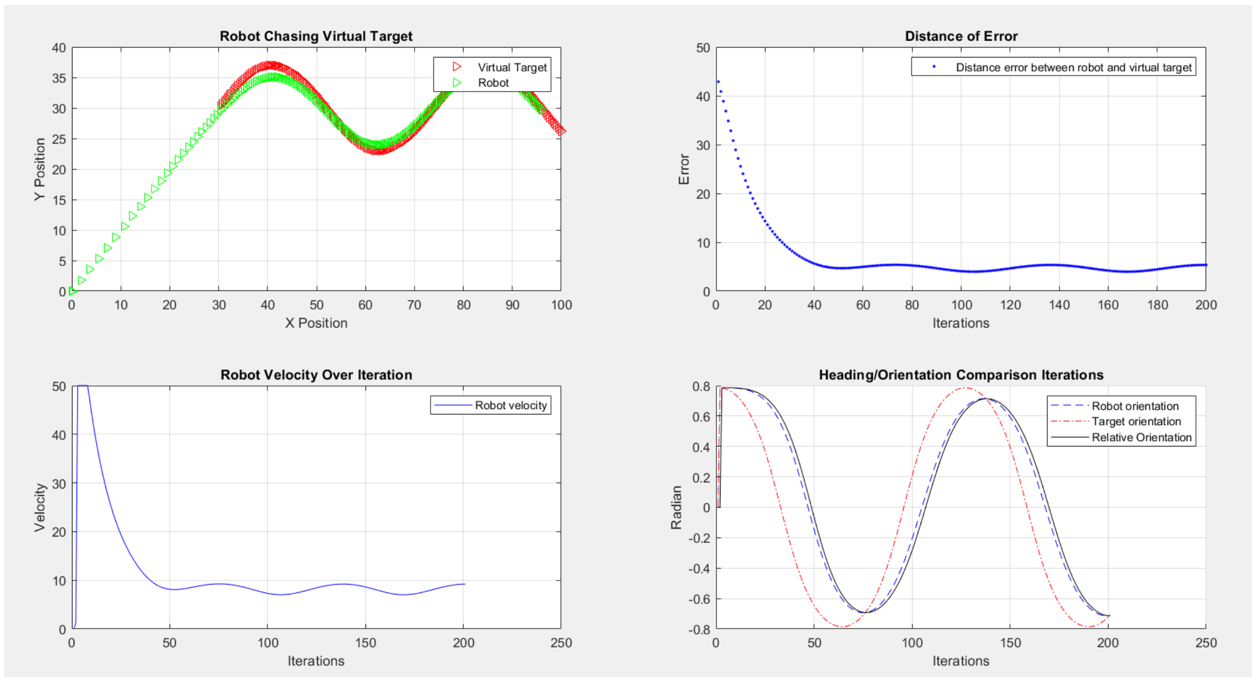


Fig.b.2.1: All four graphs outputted, max_vrd = 50 and Lambda = 1.5.

In this situation the robot finds it more difficult to track the robot as shown in the top left figure as the path of the robot (green) does not entirely line up with the virtual target (red). Because of the increased distance distortion between the two entities the error caused by the distance increases significantly as the previous range of the miniature sin wave in Fig.b.2 was roughly 1 to 2 whereas the range of this error graph is approximately 4 to 6. The comparison of the two graphs is observed in Fig.b.2.3.

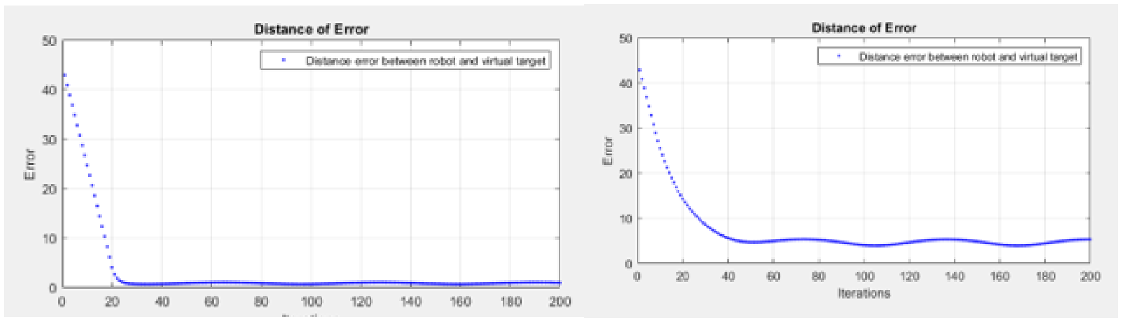


Fig.b.2.3: Distance Error comparison of Lambda = 8.5 (left) with Lambda = 1.5 (right)

The orientations of the two graphs are also different. In Fig.b.2.1 the robot is easily able to adjust itself so that it is in line with the virtual target. In Fig.b.2.2 however the robot has a difficult time latching onto the virtual target path because the “pull” on the robot is not as strong as in Fig.b.2.1.

The weaker attraction results in a situation where the robot does not directly follow the target thus creating an orientation that is similar but not exact to the virtual target. This approximation is more apparent when the two heading graphs are compared side by side in Fig.b.2.4.

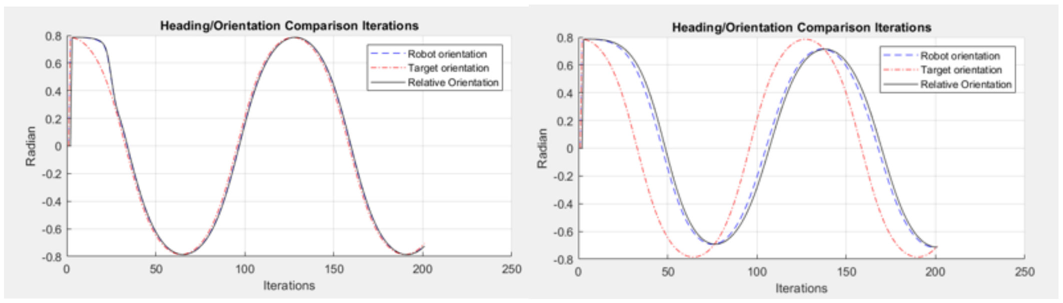


Fig.b.2.3: Orientation comparison of Lambda = 8.5 (left) with Lambda = 1.5 (right)

Noisy Environment (c)

For the noisy environment, the code was primarily provided by the professor in terms of the comment code containing circular motion. All one needed to do in this case was set the x and y positions of the virtual target inside the for loop and attach the prebuilt std and mean to have them move at the desired predetermined path (i.e., linear or sin wave).

Movement in a linear/line trajectory

In this subsection I will plot the tracking results of the following information:

1. Trajectories of the target and robot
2. Tracking error between the target and robot
3. Robot's heading
4. Robot's velocity

To observe these four graphs, one must direct their attention to Fig.c.1.

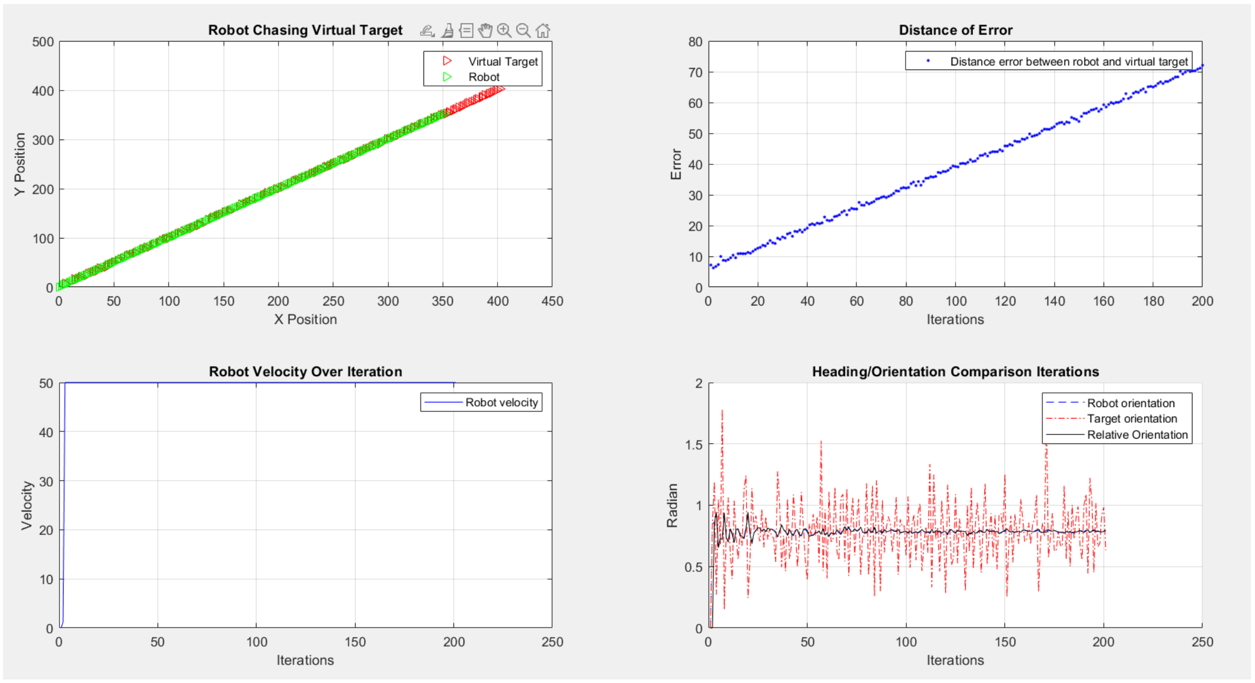


Fig.c.1: All four graphs outputted. std = 0.5, mean = 0.5, max_vdr = 50, lambda = 8.5.

This case is very similar to the no noise linear case with the exception that three of the graphs have changed in significant ways. When taking a closer look at the Robot Chase graph in Fig.c.1.2 it becomes apparent that the noise has caused the virtual target to have a somewhat "shaky" movement along its diagonal path. This shaky movement is confirmed by the Distance Error graph which is not entirely connected as it was in Fig.b.1 showing that the virtual target is not sticking to the straight diagonal position assigned to it.

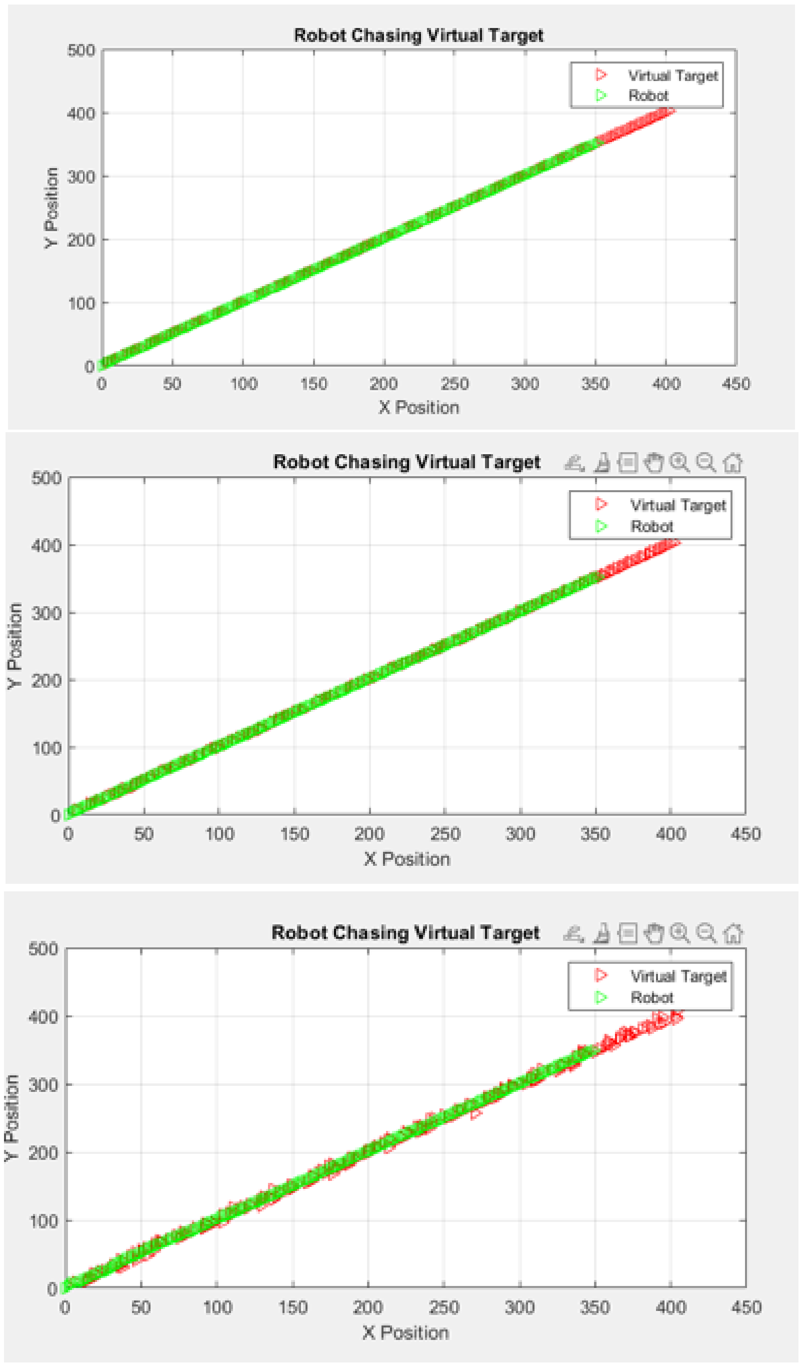


Fig.c.1.2: Robot Chase graph without noise (top), std = 0.5 (middle) and std = 3.0 (bottom)

The Heading and Orientation graph further demonstrates the virtual targets noisy behavior with sharp changes in orientation, luckily the equation in Fig.a.1 uses Phi in in its calculations for the robot orientation. Recall that Phi signifies the <u>relative orientations</u>, therefore by determining the approximate position of the previous state the robot can rely on off the Phi to maintain a more realistic result as observed in Fig.c.1.3.

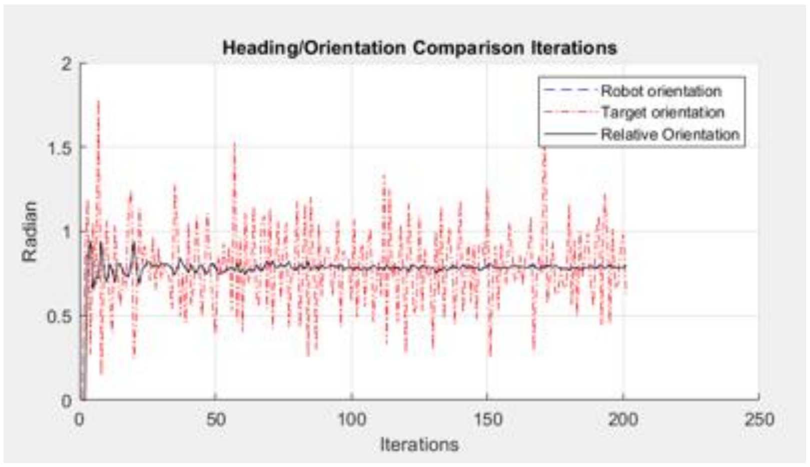


Fig.c.1.3: All four graphs outputted. Std = 3, mean = 0.5, max_vdr = 50, lambda = 8.5.

When testing for other cases it is important to consider the effect of mean and standard deviation on the overall system. In one case let only the standard deviation increase while in another case let the mean increase and standard deviation decrease. These cases can be observed in Fig.c.1.4-5.

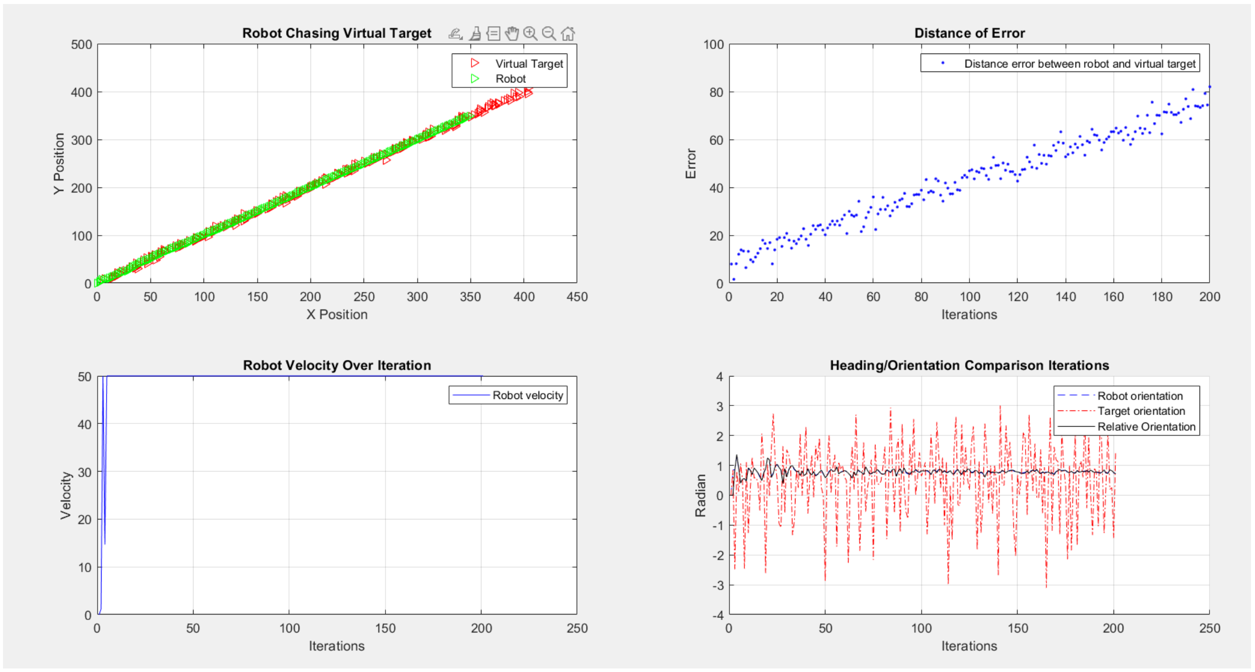


Fig.c.1.4: All four graphs outputted. std = 3, mean = 0.5, max_vdr = 50, lambda = 8.5.

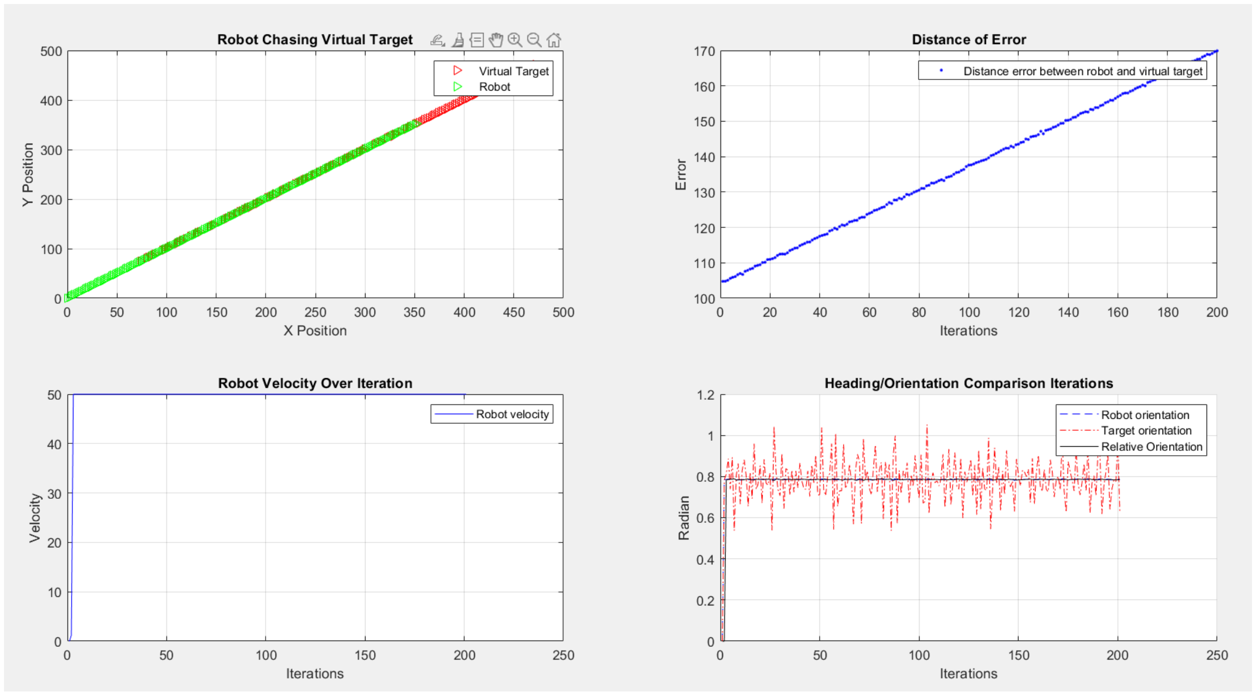


Fig.c.1.5: All four graphs outputted. std = 0.2, mean = 70, max_vdr = 50, lambda = 8.5.

When comparing the two figures it is apparent that the mean will extend the predetermined path set for the virtual target and the standard deviation determines the intensity of the random distribution of the data point. Therefore, the higher the standard deviation the more erroneous the data can become in the Distance of Error graph. The standard deviation also affects the erroneous data in that it can cause the distance to become more scattered and thus produce a scattered appearance. The heading and orientation graph appears to only be affected by the standard deviation. The smaller the magnitude of standard deviation the more accurate the relative orientation (Phi) becomes thus allowing the robot orientation to have a more accurate result as shown in Fig.c.1.5's Heading/Orientation Graph.

Movement in a sin wave trajectory

In this subsection I will plot the tracking results of the following information:

1. Trajectories of the target and robot
2. Tracking error between the target and robot
3. Robot's heading
4. Robot's velocity

To observe these four graphs, one must direct their attention to Fig.c.2.

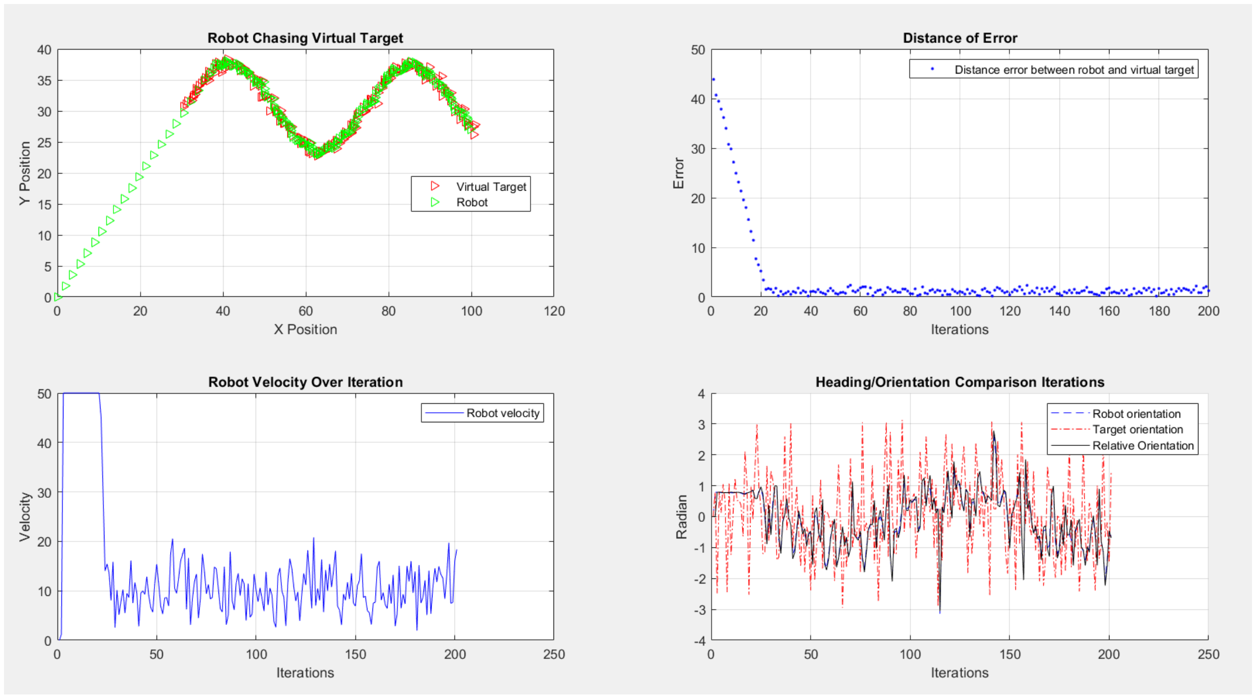


Fig.c.2: All four graphs outputted. Std = 0.5, mean = 0.5, max_vdr = 50, lambda = 8.5.

In this example it is apparent that the robot is having a difficult time trying to directly line up with the virtual target's movement. The Distance of Error graph reveals that the robot does get relatively close to the virtual target but due to the noise the robot cannot completely pinpoint the position (as shown by the scattered appearance from iteration 20 to 200). Because of the various movements of the virtual target around the sin wave pattern the robot has to constantly make sharp changes in its velocity as it tries to adjust itself to meet the smallest of changes in the virtual target's position. This chaotic attempt to follow the sign wave results in the Heading and Orientation that is very disfigured compared to the noiseless version in Fig.b.2. Although the relative orientation (Phi) helps maintain an average orientation it is evident that the robot made many sharp (and chaotic) turns when trying to chase the widespread data points representing the virtual target's position in the Chase Graph.

One method to ensure the robot minimizes error would be to simply lower the magnitude of the standard deviation. This method can be observed in Fig.c.2.1.

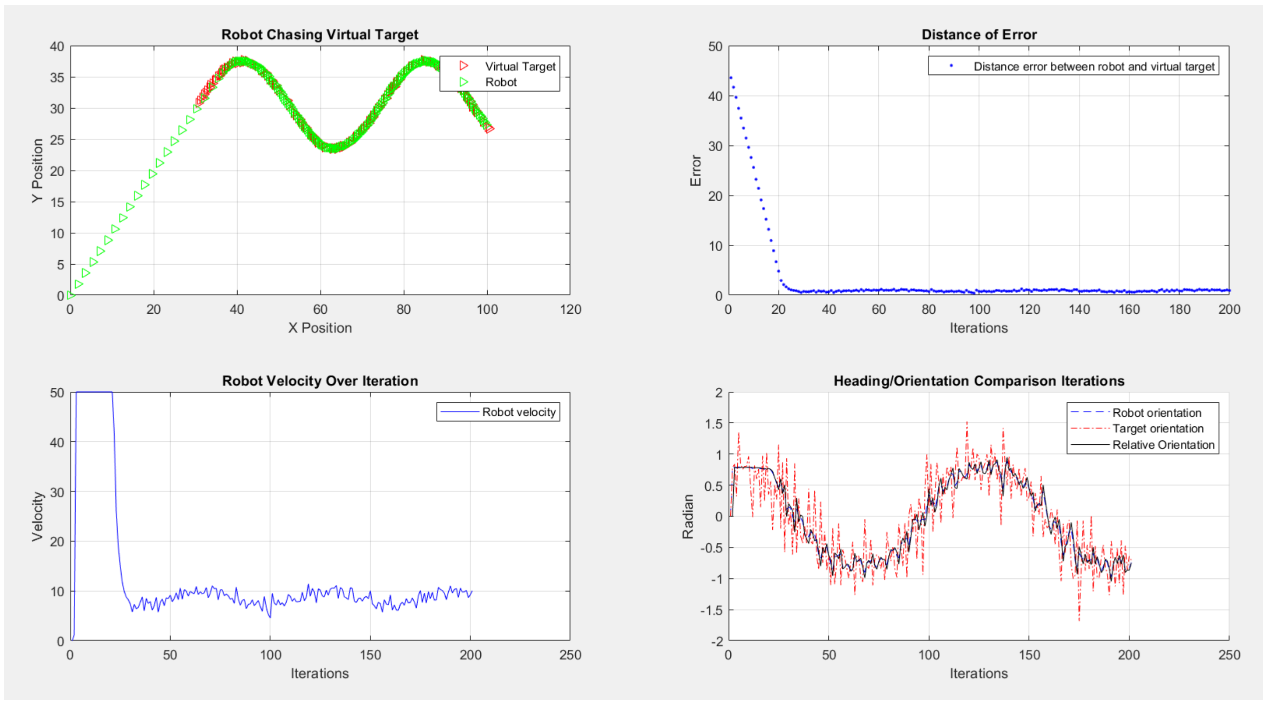


Fig.c.2: All four graphs outputted. Std = 0.1, mean = 0.5, max_vdr = 50, lambda = 8.5.

Notice that when the standard deviation is minimized all graphs appear less chaotic in that the sharp jagged changes soften. During this example it also becomes more apparent that the orientation and velocity of the robot is that of a sin wave. The smaller the magnitude in standard deviation the more apparent this characteristic becomes.

Consider a situation where the noise is unchangeable. In this case changing the velocity and lambda of the environment may help in reducing erroneous data. This method can be observed in Fig.c.2.3.

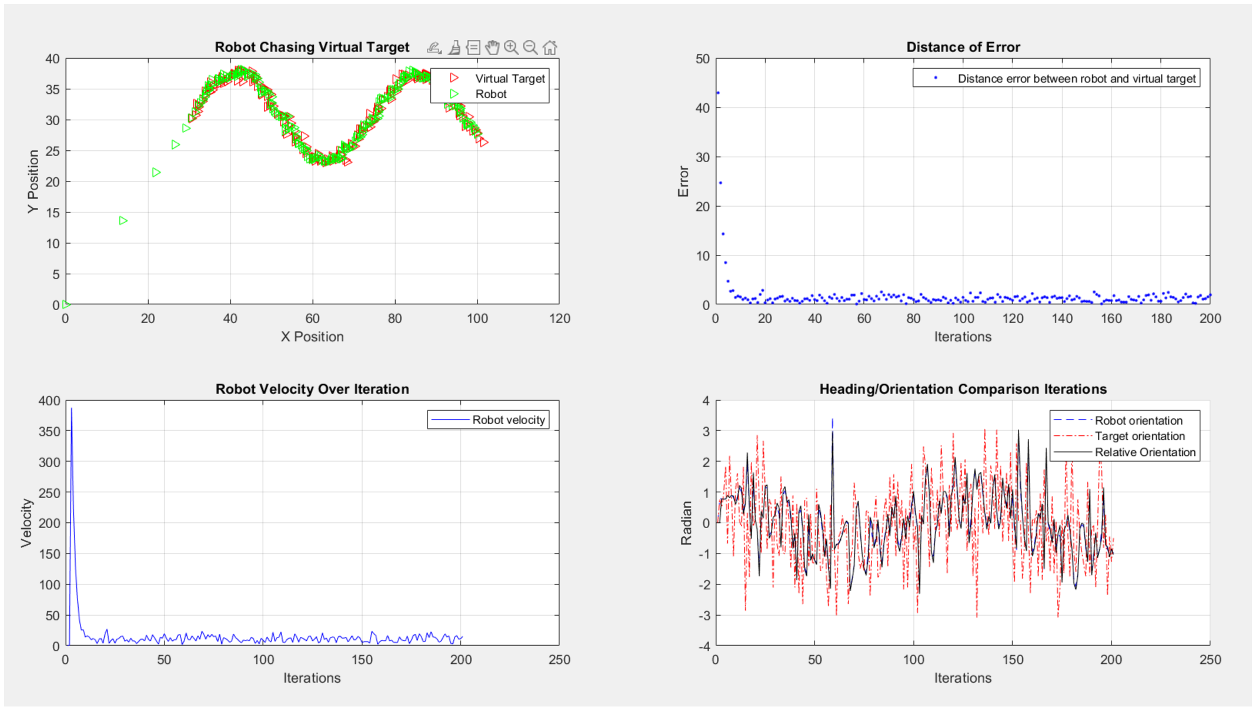


Fig.c.2: All four graphs outputted. Std = 0.5, mean = 0.5, max_vdr = 2000, lambda = 9.

In this situation the robot is easily able to catch up to the virtual target but it still making sharp turns. Despite the initial hypothesis, this scenario still causes a lot of error. The reason for this error is because the increased speed and attractive force of the virtual target cause the robot to become overly sensitive. In conclusion, having increased lambda and velocity will cause the robot to constantly overcorrect itself resulting in a very jagged result in the chase, heading, and velocity of the robot.

Code Portion (d)

```
%Gabriel Mortensen
%Mar. 2022
%CPE 470
%Project 2

function Project_2()

% *** Initialization for clearing all input ***
clc, close all
clear

% *** Obtain User Input ***
reply = 0;
while (reply < 1 || reply > 6  )
    disp('!!!Gabriel Mortensen Project 2 CPE 470!!!')
    disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
    disp('What would you like to do?')
    disp('  1: Noise free environment circular trajectory.')
    disp('  2: Noisy environment circular trajectory.')
    disp('  3: Noise free environment linear/line trajectory.')
    disp('  4: Noisy environment linear/line trajectory.')
    disp('  5: Noise free environment sin trajectory.')
    disp('  6: Noisy environment sin trajectory.')
    reply = input('');
    disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
end

% *** Set parameters for simulation ***
n = 2; % Number of dimensions
delta_t = 0.05; % Set time step
t = 0:delta_t:10;% Set total simulation time
lambda = 8.5; % Set scaling factor of attractive potential field
vr_max = 50; % Set maximum of robot velocity
Phi =  zeros (length(t),1); % Inital Phi of the robot

% *** Set VIRTUAL TARGET ***
qv = zeros (length(t),n); %Initial positions of virtual target
pv = 1.2; %Set velocity of virtual target
theta_t = zeros (length(t),1); % Initial heading of the virtual target

% *** Set ROBOT ***
%Set initial state of robot (robot)
qr = zeros (length(t),n); %initial position of robot
v_rd =  zeros (length(t),1); %Initial velocity of robot
theta_r = zeros (length(t),1); % Initial heading of the robot

% *** Obtain user input again on robot velocity ***
new_reply = 0;
while (new_reply < 1 || new_reply > 2  )
    disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
    fprintf("The current velocity of the VIRTUAL TARGET is: %.2f \n", pv)
    fprintf("The current limit on the velocity of the ROBOT is: %d \n", vr_max)
    disp('Do you want to change the limit on the ROBOT velocity?')
    disp('  1: Yes.')
```

```
    disp('  2: No.')
    new_reply = input('');
    disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
end
% *** react to user input if required ***
if(new_reply == 1)
      disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
      disp('What speed limit would you like to place on the robot?')
      vr_max = input('');
      disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
end

% *** Obtain user input again on robot velocity ***
extra_reply = 0;
while (extra_reply < 1 || extra_reply > 2  )
    disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
    fprintf("The current lambda is: %.2f \n", lambda)
    disp('Do you want to change the lambda?')
    disp('  1: Yes.')
    disp('  2: No.')
    extra_reply = input('');
    disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
end
% *** react to user input if required ***
if(extra_reply == 1)
      disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
      disp("What would you like lambda to be?")
      lambda = input('');
      disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
end

% *** Set relative states between ROBOT and VIRTUAL TARGET ***
qrv = zeros (length(t),n); %Save relative positions between robot and virtual target
prv = zeros(length(t),n); %Save relative velocities between robot and virtual target

% *** Compute initial relative states between robot and  VIRTUAL TARGET ***
qrv(1,:) = qv(1,:) - qr(1,:);%Compute the initial relative position
prv(1,:) = [pv*cos(theta_t(1))-v_rd(1)*cos(theta_r(1)), pv*sin(theta_t(1))-v_rd(1)*sin(theta_r(1))]; %Compute the
initial relative velocity

% *** Set noise mean and standard deviation ***
noise_mean = 0.5;
noise_std = 0.5; %try 0.2 also

% *** Get user input ***
if (mod(reply, 2) == 0)
    disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
    disp('Please select the mean (typical is 0.5):')
    noise_mean = input('');
    disp('Please select the standard deviation (typical is 0.5 or 0.2):')
    noise_std = input('');
    disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
end

% *** Make a figure and maxmize it for user experience ***
Fig = figure();
```

```
Fig.WindowState = 'maximized';


%=========MAIN PROGRAM==================
for i =2:length(t)

    %Store variable for previous state
    Previous_State = i -1;

    %++++++++++CIRCULAR TRAJECTORY+++++++++++
    if (reply == 1)
    %Set target trajectory moving in CIRCULAR trajectory WITHOUT noise
        qv_x = 60 - 15*cos(t(i));
        qv_y = 30 + 15*sin(t(i));
        qv(i,:) = [qv_x, qv_y]; %compute position of virtual target
    end
    if (reply == 2)
    %Set target trajectory moving in CIRCULAR trajectory WITH noise
        qv_x = 60 - 15*cos(t(i))+ noise_std * randn + noise_mean;
        qv_y = 30 + 15*sin(t(i)) + noise_std * randn + noise_mean;
        qv(i,:) = [qv_x, qv_y];  %compute position of target
    end
    if (reply == 3)
    %Set target trajectory moving in linear trajectory WITHOUT noise
        qv_x = i*2;
        qv_y = i*2;
        qv(i,:) = [qv_x, qv_y];  %compute position of target
    end
    if (reply == 4)
    %Set target trajectory moving in linear trajectory WITHOUT noise
        qv_x = i*2 + noise_std * randn + noise_mean;
        qv_y = i*2 + noise_std * randn + noise_mean;
        qv(i,:) = [qv_x, qv_y];  %compute position of target
    end
    if (reply == 5)
    %Set target trajectory moving in linear trajectory WITHOUT noise
        qv_x = 30 + t(i) * 7;
        qv_y = 30 + sin(t(i)) * 7;
        qv(i,:) = [qv_x, qv_y];  %compute position of target
    end
    if (reply == 6)
    %Set target trajectory moving in linear trajectory WITHOUT noise
        qv_x = 30 + 7 * t(i) + noise_std * randn + noise_mean;
        qv_y = 30 + 7 * sin(t(i)) + noise_std * randn + noise_mean;
        qv(i,:) = [qv_x, qv_y];  %compute position of target
    end
    %Compute the target heading
    qt_diff(i,:) =  qv(i,:)- qv(Previous_State,:);
    theta_t(i) = atan2(qt_diff(i,2),qt_diff(i,1));

    %======UPDATE position and velocity of robot=========
    %Phi calculation: Slide 2 from Project2_Instruction.ppt
    Phi(i) = atan2(qrv(Previous_State,2), qrv(Previous_State,1));

    %v_rd calculation: Slide 4 from Project2_Instruction.ppt
    v_rd_expression = (norm(pv)^2) + (2*lambda*norm(qrv(Previous_State, :))*norm(pv)*abs(cos(theta_t(i)-
Phi(i)))) + ((norm(qrv(Previous_State, :)))^2*(lambda)^2);
    v_rd(i) = sqrt(v_rd_expression);

    %Limit Robot Velocity if need be
    v_rd(i) = min(v_rd(i), vr_max);

    %theta_r calculation: Slide 4 from Project2_Instruction.ppt
    %Calculates orientation (heading)
    theta_r_expression_numerator = (norm(pv) * sin(theta_t(i) - Phi(i)));
    theta_r_expression_denominator = norm(v_rd(i));
    theta_r_expression = theta_r_expression_numerator / theta_r_expression_denominator;
    theta_r(i) = Phi(i) + asin(theta_r_expression);

    %Calculate position of the robot
    qr(i,:) = qr(Previous_State,:) + v_rd(i)*delta_t*[cos(theta_r(Previous_State)), sin(theta_r(Previous_State))];
    qrv(i,:) = qv(i,:) - qr(i,:);
    prv(i,:) = [pv*cos(theta_t(i)) - v_rd(i)*cos(theta_r(i)), pv*sin(theta_t(i)) - v_rd(i)*sin(theta_r(i))];

    error(i) = norm(qv(i,:)-qr(i,:));

    %plot postions qv of virtual target
    subplot(2,2,1);
    plot(qv(:,1),qv(:,2),'r>')
    hold on
    %plot postions qv of robot
    plot(qr(:,1),qr(:,2),'g>')

    M = getframe(gca);
end

%label graph
subplot(2,2,1);
grid on
xlabel("X Position")
ylabel("Y Position")
title('Robot Chasing Virtual Target')
legend('Virtual Target', 'Robot')

subplot(2,2,2);
plot(error(2:length(t)), 'b.')
grid on
title('Distance of Error')
xlabel("Iterations")
ylabel("Error")
legend('Distance error between robot and virtual target')

subplot(2,2,3);
plot(v_rd, 'b')
grid on
legend('Robot velocity')
xlabel("Iterations")
ylabel("Velocity")
title('Robot Velocity Over Iteration')

set(0,'CurrentFigure', Fig)
```

```
subplot(2,2,4);
hold on
plot(theta_r, '--b')
plot(theta_t, '-.r')
plot(Phi, 'k')
grid on
xlabel("Iterations")
ylabel("Radian")
title('Heading/Orientation Comparison Iterations')
legend('Robot orientation', 'Target orientation', 'Relative Orientation')

end
```