To: CPE 470
From: Brendan Aguiar
Date: April 12, 2021
Subject: Artificial Potential Controller

Report

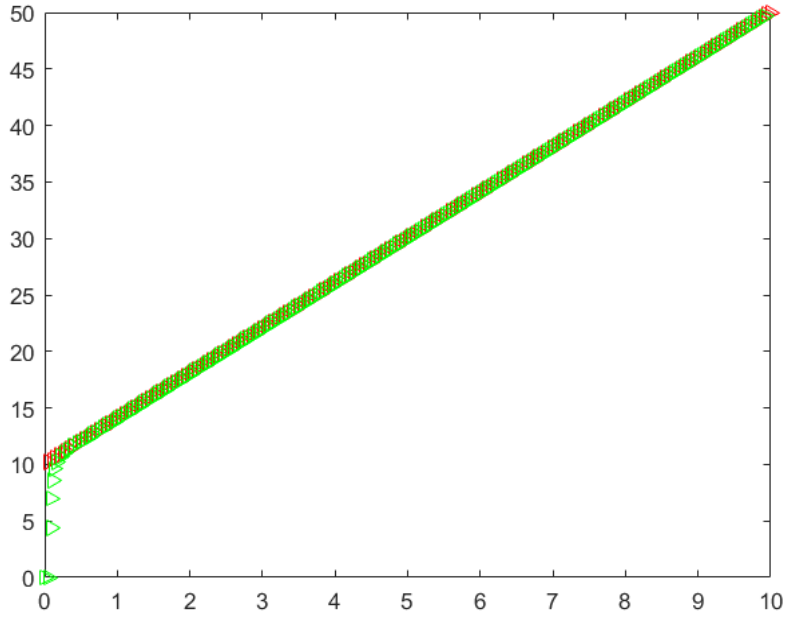
The purpose of this report is to create an Artificial Potential Controller in Matlab that will simulate a robot that follows the trajectory of a moving target. In order for the robot to follow the target, a model is constructed using their relative positions and the target's heading. The model will then update the robot's heading and velocity. In a real world application, the robot may be given velocity. To match this, an if condition is implemented to cap the robot's velocity.

Noise Free Environment

Linear Target

An important condition for the robot to catch up with the target is for their velocities to be at least equal to each other. In the case of the linear trajectory, the robot can never catch up to a target if their velocities are equivalent. They will maintain equidistance throughout the period of their traversals. Fig. 1 below has no constraints on the robot's velocity.

Fig. 1: The robot depicted in green quickly catches up to and matches the trajectory of the red target.



When the robot meets and closely follows the target, the distance error between the two matches their velocities. If the distance error between the two were to be zero, there would be a collision. As shown in Fig. 2, the robot maintains a velocity equivalent distance away as soon as it catches up to the target. Similarly in Fig. 3, the velocity of the robot approaches the value of the distance error as it closely trails the target.

Fig. 2: As the distance error between the robot and target decreases, the speed of the robot decreases.

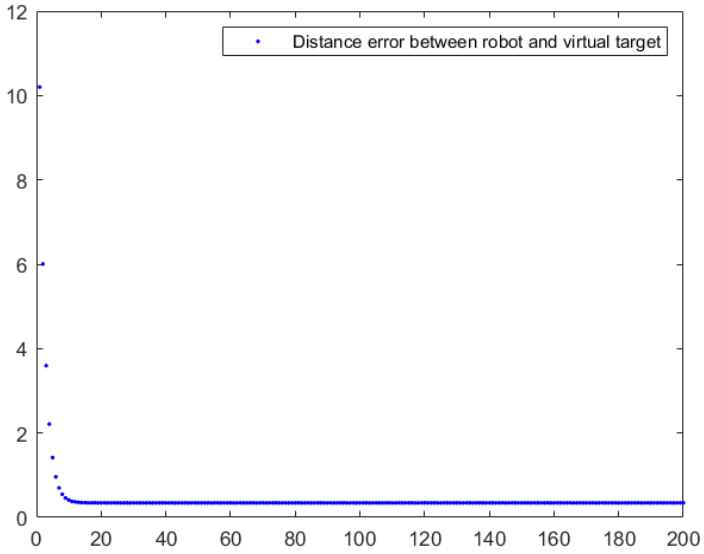
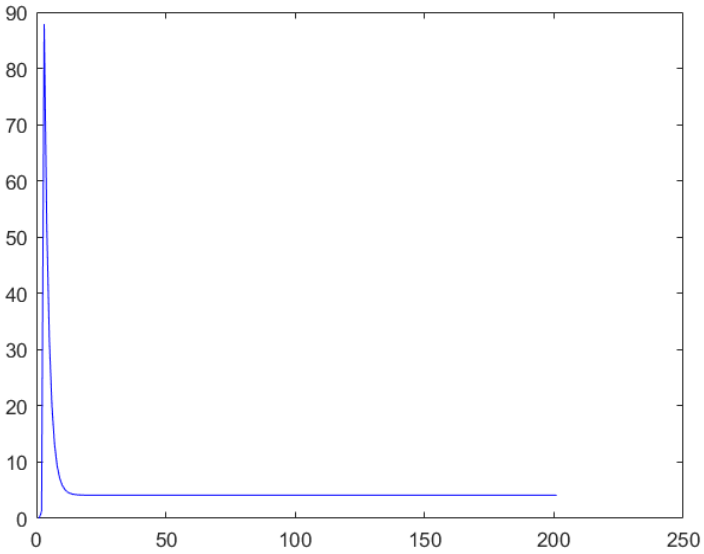
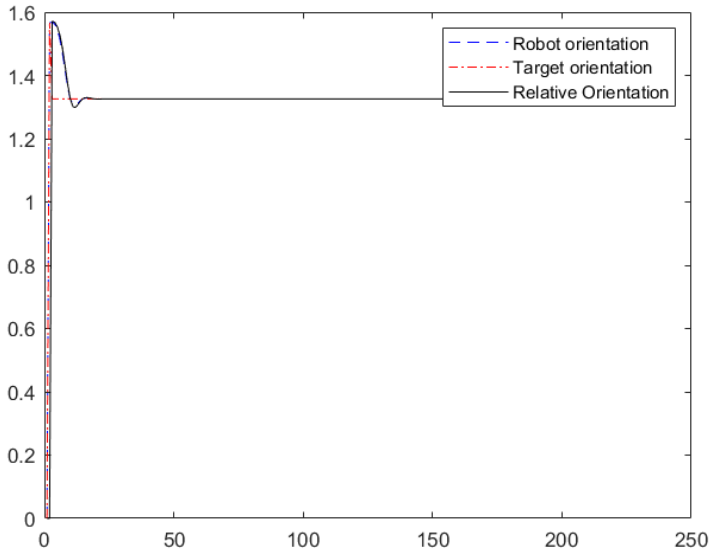


Fig. 3: When uncapped, the velocity of the robot reaches as 90 before matching the velocity of the target.



The orientation of the robot quickly adjusts as it aligns with the target. After the robot aligns with the target, both of their heading's remain constant as shown in Fig. 4.

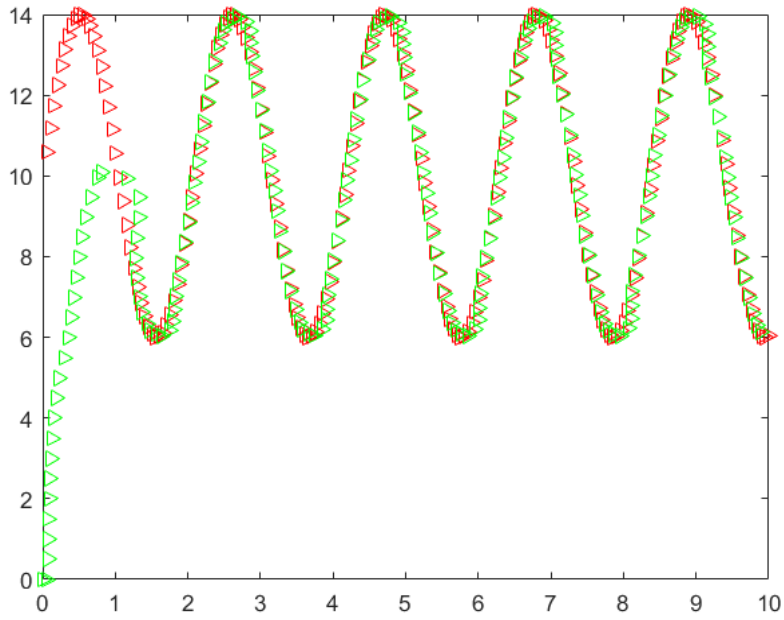
Fig. 4:The heading of the robot quickly matches up with the target's heading.



Sinusoidal Target

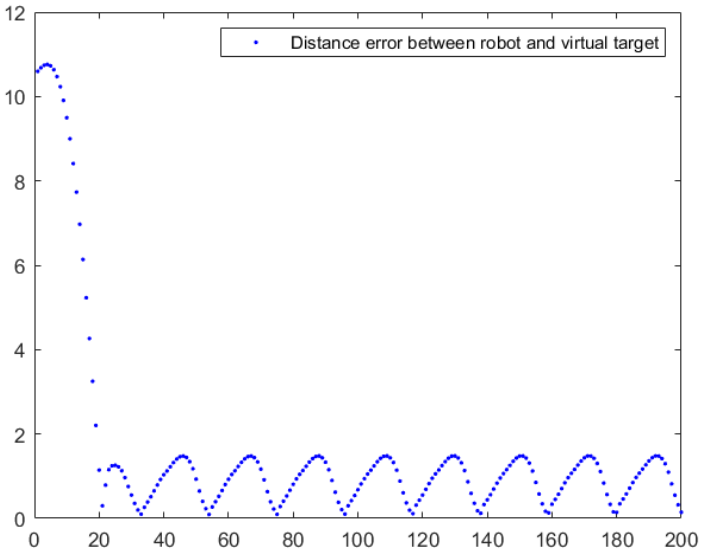
Unlike with the linear trajectory, the robot is able to catch up to the sinusoidal target trajectory when both of their velocities are equal. If the intention of the target is to avoid the robot, then the instinctual trajectory should not represent the sine wave. The target would be better off following a linear path. As demonstrated in Fig. 5., the robot is quickly able to catch up to the target after the two cross paths.

Fig. 5: The robot has zero velocity when it's heading is orthogonal to the target' as it turns to catch up to it.



One of the main disadvantages of the sinusoidal trajectory is that the target needs to decrease its speed every time it changes directions. This gives the robot more chances to catch up to the target. After catching up to the target, however, the two face the possibility of colliding every time they change directions as shown in Fig. 6 when their distance error nears zero.

Fig. 6: The distance errors that approach zero correspond to the sine peaks and valleys.



The sinusoidal test is the first implementation to incorporate a velocity cap for the robot. This cap creates a plateau effect when plotting the robot's velocity as shown in Fig. 7. As not to collide with the target, the robot velocity also slows down periodically. Fig. 8 is near identical to Fig. 7 except for an offset difference.

Fig. 7: The robot's velocity peak matches it's cap of ten and the valley nears the target's velocity.

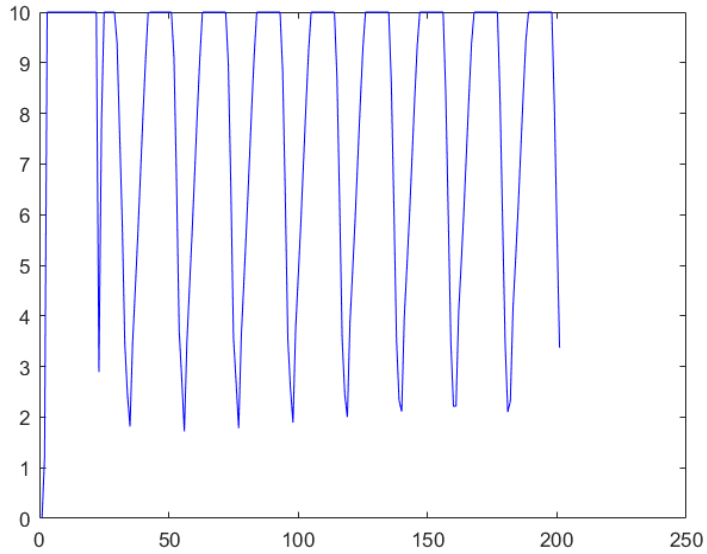
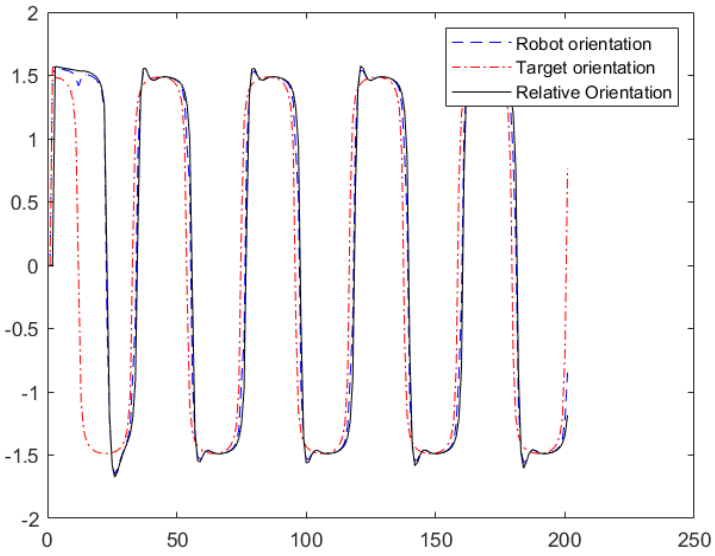


Fig. 8: The constantly changing headings closely match the constantly changing trajectories.

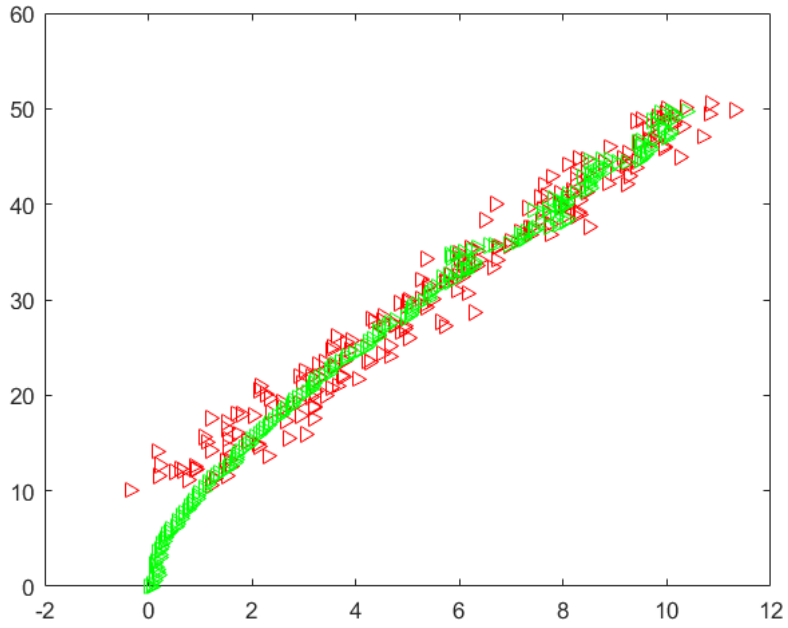


Noise Environment

Linear Target

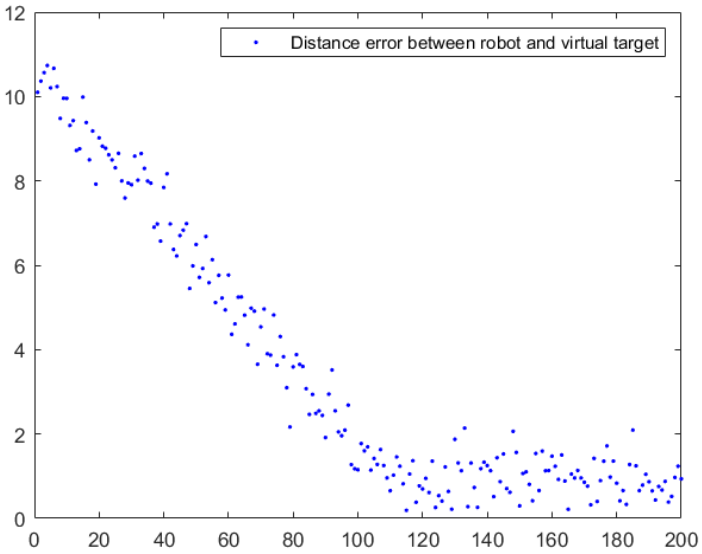
Noise was generated for the target using a gaussian distribution with a standard deviation and mean of one half. The noise was added to the target's updated position components. Fig. 9 shows the results when the robot tries to anticipate the target along with the noise.

Fig. 9: The noise causes the robot's trajectory to slow down as it triest predict the movement of the target.



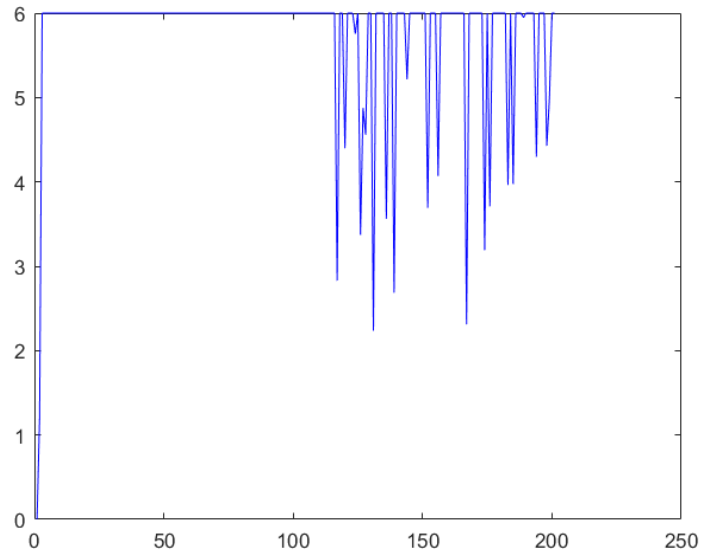
The distance errors between the robot and virtual target no longer represent a smooth path. Instead, the errors are dispersed matching the difficulty of the robot to anticipate the target. Although there are instances where the error between the two are near zero as shown in Fig. 10, this may have been out of pure coincidence.

Fig. 10: Unlike with the noise free environment, the robot can no longer quickly catch up to the target.



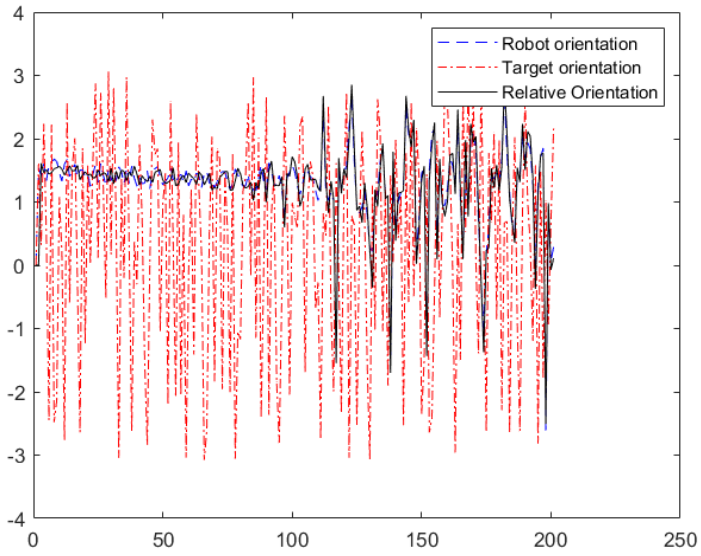
The noise environment greatly affects the speed the robot needs to keep in order to track the target. Unlike in Fig. 2 where the robot quickly honed in on the target and reduced its speed, Fig. 11 shows the robot mostly maintaining its max velocity to keep up with the hard-to-predict target.

Fig. 11: The few instances where the robot reduces its speed are quickly interrupted .



The robot's heading suffers most from the introduction of noise to the target. In the noise free environment, the robot was able to maintain a constant heading matching that of the target's. In the case of the noise environment, as shown in Fig. 12,the robot can no longer find a constant heading. In fact, the issue is compounded with time.

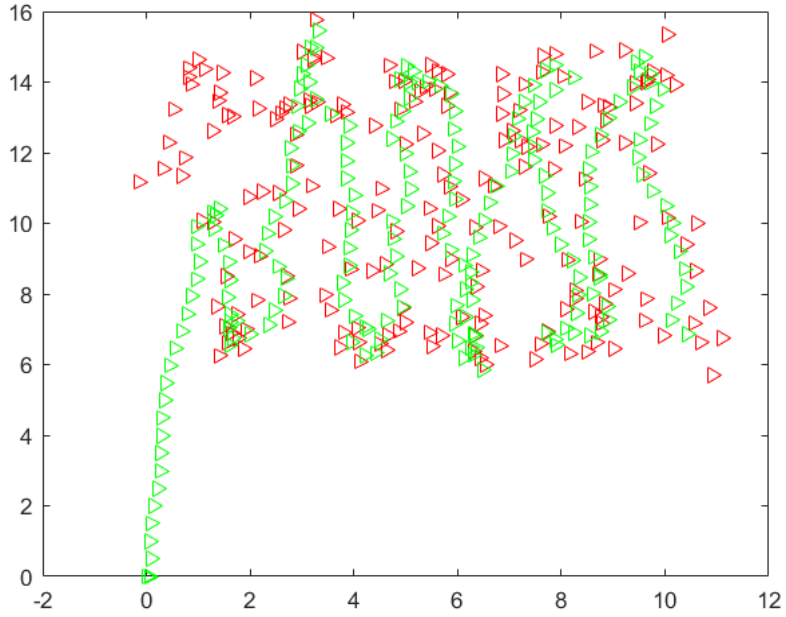
Fig. 12: The target heading has become unpredictable.



Sinusoidal Target

The noise environment test further shows the poor performance of the sinusoidal trajectory. Even with gaussian noise added to the target, the robot is able to predict its movement. Furthermore, without any a-priori knowledge of the target's trajectory or the robot's trajectory, it's unclear that the target's path is sinusoidal as shown in Fig. 13.

Fig. 13: The robot maintains a sinusoidal trajectory even through the noise.



The good performance of the robot against a sinusoidal trajectory is further shown in Fig. 14 in which the distance error quickly approaches zero before going through intervals. This contrasts from the linear trajectory where the robot took much longer to catch up to the target under noisy conditions. However, the robot still needs max velocity to track the target as shown in Fig. 15.

Fig. 14: These distance errors while not smooth closely match the errors of the noise free environment.

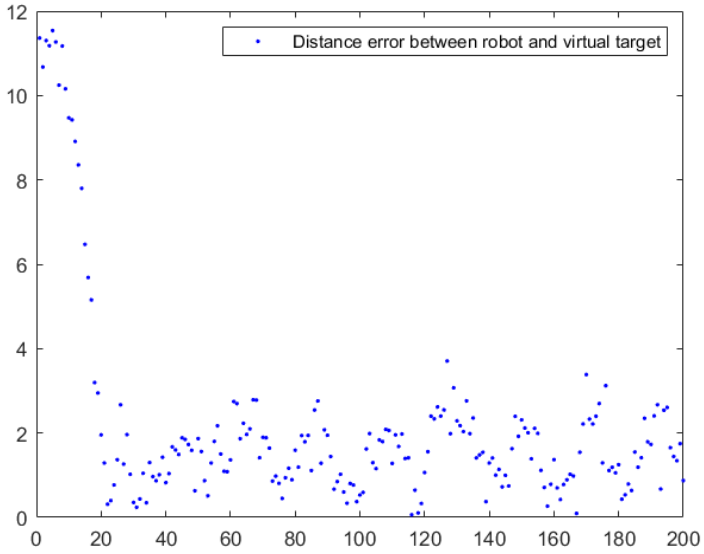
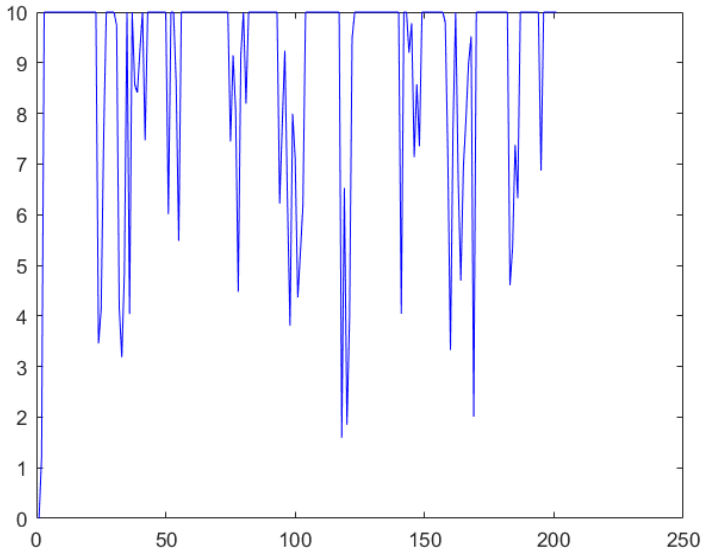
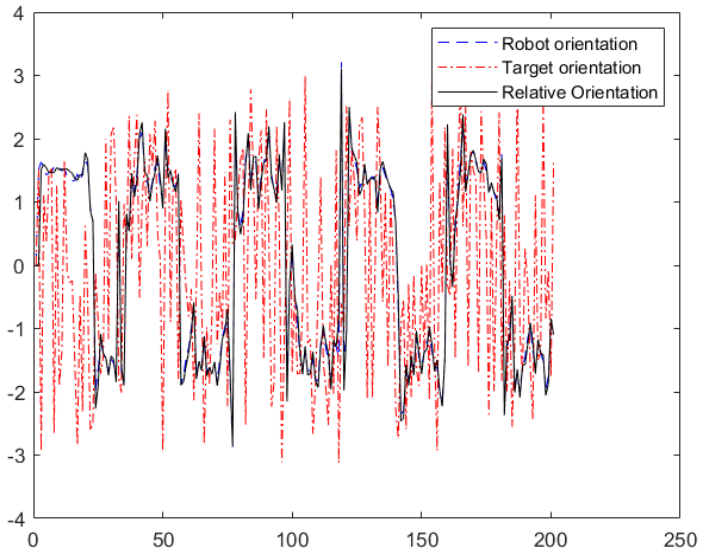


Fig. 15: The robot slows down in each interval to change directions.



The sinusoidal trajectory creates a predictable heading for the robot even under noisy conditions as shown in Fig. 16. Unlike with the linear trajectory, the relative orientation between the robot and the target is consistent. When considering these results for both a noisy and noise-free environment, the performance of the target is far better when given a linear trajectory.

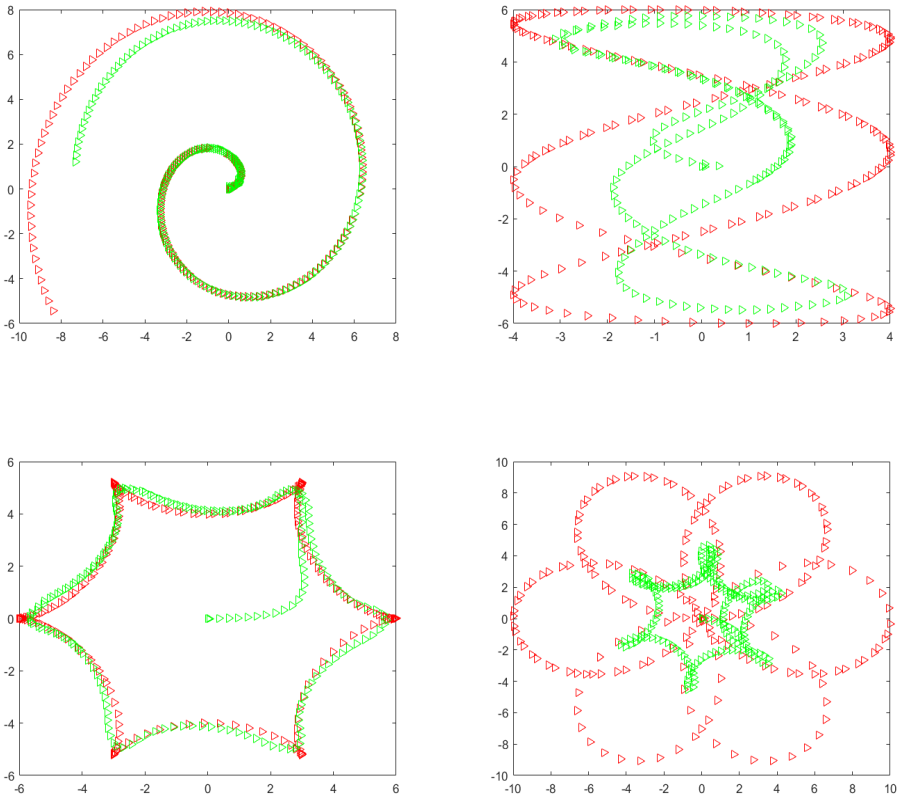
Fig. 16: The target's orientation, while noisy, maintains the characteristic of a sine wave.



More Trajectory Examples

Other parametric equations were included to compare the performance of different trajectories. In all cases the robot's max velocity matched the target's velocity. The worst performance of the trajectories came from the star shape, also known as the hypocycloid. The second worst performance comes from the spiral trajectory, but the target is able to outpace the robot over time. The other two trajectories used, which had good performance came from the Lissajous and hypotrochoid equations as shown in Fig. 17.

Fig. 17: The optimal trajectory for the target is the hypotrochoid trajectory in the bottom right.



Appendix

Source Code

```
%{In order to run the code, uncomment the parametric equation that matches the desired target trajectory. Each parametric equation has its own corresponding noise equation that can be uncommented and tested. The pr_max can be set to a higher value and the robot will more quickly catch up the target. The current parametric equation creates the Hypocycloid or star shaped trajectory. %}

% CPE470 Project 2: Artificial Potential Field Path Planning
%By Brendan Aguiar

%matlab notes
% x = y(i,:) assigns x the ith row of y
% x = y(:,i) assigns x the ith col of y

clc, clear
close all
n = 2; % Number of dimensions
delta_t = .05; % Set time step
t = 0:delta_t:10;% Set total simulation time
lambda = 8.5; % Set scaling factor of attractive potential field
pr_max = 6; % Set maximum of robot velocity. Experiment with different values.
error = zeros (length(t) - 1, 1); % Set tracking error

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

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

%=============Set relative states between robot and 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 TARGET=
qrv(1,:) = qv(1,:) - qr(1,:); % Compute the initial relative position
```

```
%Compute the initial relative velocity
a = pv*cos(theta_v(1)) - pr(1)*cos(theta_r(1));
b = pv*sin(theta_v(1)) - pr(1)*sin(theta_r(1));
prv(1,:) = [a, b];

%=============Loop Assignments=========================================
qt_diff = zeros(length(t), n);
phi = zeros(length(t), 1);
nested = 0;
%=============Set noise mean and std. dev.=============================
noise_mean = .5;
noise_std = 0.5;% = 0.2;

%=============MAIN
PROGRAM==============================================
for i = 2:length(t)
   %+++++++++SINUSOIDAL TRAJECTORY++++++++++++++++++++++++++++++
   %qv_x = t(i);%parametric equation 1
   %qv_y = 4*sin(t(i) * 3) + 10;%parametric equation 2

   %W/ noise
   %qv_x = t(i) + noise_std * randn + noise_mean;
   %qv_y = 4*sin(t(i) * 3) + 10 + noise_std * randn + noise_mean;
   %+++++++++LINEAR TRAJECTORY++++++++++++++++++++++++++++++++
   %qv_x = t(i);%parametric equation 1
   %qv_y = 4*t(i) + 10;%parametric equation 2

   %W/ noise
   %qv_x = t(i)+ noise_std * randn + noise_mean;
   %qv_y = 4*t(i) + 10 + noise_std * randn + noise_mean;
   %+++++++++CIRCULAR TRAJECTORY++++++++++++++++++++++++++++++++
   %W/O noise
   %qv_x = 60 - 15 * cos(t(i));%parametric equation 1
   %qv_y = 30 + 15 * sin(t(i));%parametric equation 2

   %W/ noise
   %qv_x = 60 - 15 * cos(t(i)) + noise_std * randn + noise_mean;
   %qv_y = 30 + sin(t(i))  + noise_std * randn + noise_mean;
   %+++++++++SPIRAL TRAJECTORY++++++++++++++++++++++++++++++++++
   %qv_x = t(i) * cos(t(i));
   %qv_y = t(i) * sin(t(i));
   %+++++++++Lissajous TRAJECTORY++++++++++++++++++++++++++++++
   %qv_x = 4* sin(3*t(i) + 5);
   %qv_y = 6*sin(t(i));
```

```
%+++++++++Hypotrochoid TRAJECTORY++++++++++++++++++++++++++++
%qv_x = 5 * cos(t(i)) + 5 * cos(5*t(i));
%qv_y = 5 * sin(t(i)) - 5 * sin(5*t(i));
%+++++++++Hypocycloid TRAJECTORY++++++++++++++++++++++++++++
qv_x = 5 * cos(t(i)) +  cos(5*t(i));
qv_y = 5 * sin(t(i)) -  sin(5*t(i));
qv(i,:) = [qv_x, qv_y]; % Compute position of target
qt_diff(i,:) = qv(i,:) - qv(i - 1,:);
theta_v(i) = atan2(qt_diff(i,2),qt_diff(i,1));


%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% Code by Brendan Aguiar
phi(i) = atan2(qrv(i - 1,2), qrv(i - 1,1));% Get phi
%modeling robot velocity
term1 = pv^2;
term2 = 2*lambda*norm(qrv(i - 1,:))*pv*abs(cos(theta_v(i)-phi(i)));
term3 = (norm(qrv(i - 1,:))*lambda)^2;
pr(i) = sqrt(term1 + term2 + term3);
if pr(i) >= pr_max
    pr(i) = pr_max;%robot velocity constraint
end
%modeling robot heading
nested = (pv*sin(theta_v(i) - phi(i))/pr(i));
theta_r(i) = phi(i) + asin(nested);

% Code by Dr. Jim La here%
c = cos(theta_r(i-1));
d = sin(theta_r(i-1));
qr(i,:) = qr(i - 1, :) + pr(i)*delta_t*[c, d];% Get next pos. of robot
qrv(i,:) = qv(i,:) - qr(i,:);% Update pos. between robot and target
e = pv*cos(theta_v(i) - pr(i)*cos(theta_r(i)));
f = pv*sin(theta_v(i) - pr(i)*sin(theta_r(i)));
prv(i,:) = [e,f];
error(i) = norm(qv(i,:) - qr(i,:)); % Get dist. between robot & target
% Plot positions qv of virtual target
plot (qv(:,1),qv(:,2), 'r>')
hold on
% Plot positions qv of robot
plot(qr(:,1),qr(:,2), 'g>')

M = getframe(gca); % Find out what this does.
%Getframe serves to capture the axes at each time in "t".
%In other words, the chart is reloading with each iteration similar to
```

```
    %how a movie reel would cycle through frames (Mathworks, website).
    %mov = addframe(mov, M); % Find out what this does
    %Addframe will add current frame to the avi file (Northwestern, website).
end

%============PLOT RESULTS==============================================
figure(2), plot(error(2:length(t)), 'b.')
legend('Distance error between robot and virtual target')
figure(3), plot(pr, 'b')
figure(4), plot(theta_r, '--b')
hold on
plot(theta_v, '-.r')
hold on
plot (phi, 'k')
legend('Robot orientation', 'Target orientation', 'Relative Orientation')
%https://www.mathworks.com/help/matlab/ref/getframe.html
%http://www.ece.northwestern.edu/local-apps/matlabhelp/techdoc/ref/addframe.html
```

References

Mathworks (website), retrieved from: https://www.mathworks.com/help/matlab/ref/getframe.html

Northwestern (website), retrieved from:
http://www.ece.northwestern.edu/local-apps/matlabhelp/techdoc/ref/addframe.html

Elepa (website), retrieved from:
https://elepa.files.wordpress.com/2013/11/fifty-famous-curves.pdf