Bridge of Doom Challenge

Berwin Lan and Cara Mulrooney QEA 2 Spring 2021, Olin College of Engineering

The Bridge of Doom Challenge aims to apply our knowledge of parametric curves, robot motion models, validation, and debugging techniques by writing a program to autonomously pilot a simulated NEATO robot from the starting platform to the goal using MATLAB, ROS, and Gazebo. The shape of the centerline of the Bridge of Doom follows a parametric curve that is defined in Equation 1 as $\vec{\boldsymbol{r}}(u) = 4[0.3960cos(2.65(u+1.4)\hat{\imath} - 0.99sin(u+1.4)\hat{\jmath}], (u \in [0, 3.2])$*. As the robot traversed the bridge, we created several plots, including the parametric curve that defines the centerline of the bridge, the unit tangent and unit normal vectors along the theoretical path, the theoretical and actual linear speed and angular velocity at the robots center of mass as a function of time, the theoretical and actual left and right wheel velocities of the robot, and the actual path the robot travels on the Bridge of Doom along with predicted and experimental unit tangent vectors along the curve. This paper documents our methodology.*

In Equation 1, the parameter u represents bt, where b is a dilation factor used for conversion from the raw parameterization of the curve to time so speeds can be achieved by the robot and t is time, in seconds. Thus, the outputs produced by the encoders are scaled by a factor of b, which needs to be taken into account when comparing theoretical and experimental values. In our code, $b = 0.1$.

$$\begin{aligned}\vec{\mathbf{r}}(u) &= 4[0.3960cos(2.65(u+1.4)\hat{\imath} \\ &-0.99sin(u+1.4)\hat{\jmath}], (u \in [0, 3.2])\end{aligned} \tag{1}$$

The unit tangent, which represents the $+\hat{x}$ direction of the NEATO's local coordinate system, is described by Equation 2.

$$\hat{\mathbf{T}} = \frac{\mathbf{r}'}{|\mathbf{r}'|} \tag{2}$$

The unit normal is described by Equation 3

$$\hat{\mathbf{N}} = \frac{\hat{\mathbf{T}}'}{|\hat{\mathbf{T}}'|} \tag{3}$$

The MATLAB Symbolic Toolbox was used to calculate theoretical plots. In Equations 1, 2, and 3, u is a real, positive parameter. Thus, numerical values were computed by substituting values in for u in the equations, and those numerical values were plotted in Figure 1 to create the theoretical path of the NEATO along with the units tangent and normal at evenly spaced points in time. The NEATO follows the path of the parametric equation shown in Equation 1. As it travels, the unit tangent matches the heading of the robot, always pointing in the local $+\hat{y}$ direction. The unit normal vector is perpendicular to the unit tangent vector, and it points in the direction the robot will turn.

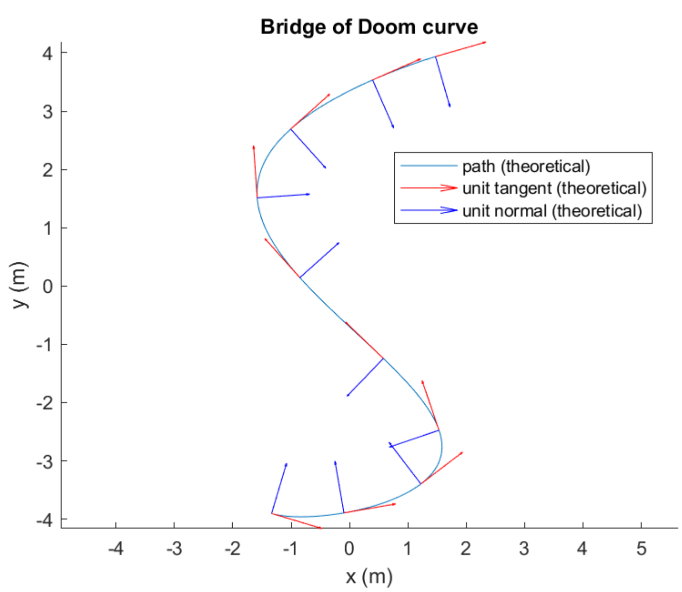

Figure 1: NEATO's Theoretical Path, Unit Tangent Vectors, and Unit Normal Vectors

$$\vec{\omega} = \hat{\mathbf{T}} \times \frac{d\hat{\mathbf{T}}}{dt} \tag{4}$$

$$V = \left|\frac{d\vec{r}}{dt}\right| \tag{5}$$

The angular velocity is the cross product of $\hat{\mathbf{T}}$ and $\frac{d\hat{\mathbf{T}}}{dt}$, as described in Equation 4, and the linear speed is the normalized first derivative of position with respect to time, as shown in Equation 5. The MATLAB Symbolic Toolbox was used with the resulting equations and numerical substitutions to create Figure 2, which shows the theoretical linear speed and angular velocity over time. As the figure shows, the angular velocity maximum at t = 10s represents the NEATO turning left and its minimum at t = 22s represents the NEATO turning right. In addition, the NEATOs linear speed has minima when turning (ex. t = 10s, 22s) and maxima when traveling between turns (ex. t = 4s, 15s, 26s)

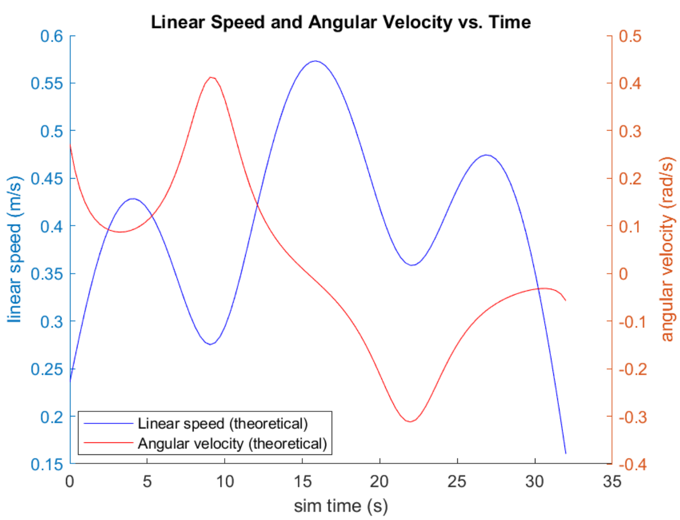

Figure 2: NEATO's Theoretical Linear Speed and Angular Velocity

$$V_L = V - \omega\frac{d}{2} \tag{6}$$

$$V_R = V + \omega\frac{d}{2} \tag{7}$$

The theoretical left and right wheel velocities as a function of time were calculated using Equations 6 and 7, which use linear speed V, angular velocity $\vec{\omega}$, and the distance between the wheels d. In this case, d = 0.235 m. Figure 3 shows the theoretical left and right wheel velocities as calculated using Equations 6 and 7. When $V_L < V_R$, the NEATO is turning right, which occurs between t = [0, 15s). When $V_L > V_R$, the NEATO is turning left, which occurs between t = (15s, 30s). A greater difference in wheel velocities indicates a sharper turn. At t = 15s and $t > 30$s, the left and right wheels are traveling at equal velocities; thus, the NEATO is driving in a straight line.

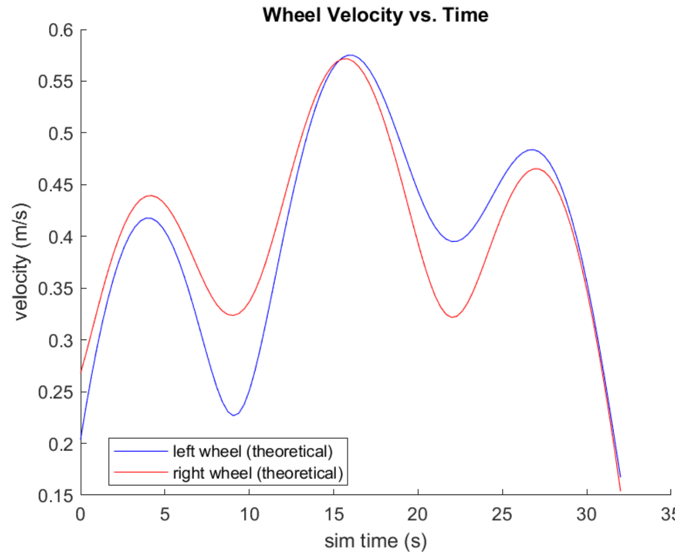

Figure 3: NEATO's Theoretical Left and Right Wheel Velocities

The following Figures 4, 5, and 6 show the theoretical plots in Figures 1, 2, and 3 overlaid with reconstructions of the experimental values obtained from the encoders of the simulated NEATO.

Figure 4, which shows the reconstructed wheel velocities, was created by dividing the change in raw encoded distance over the change in time to find each velocity, defining velocity as the change in distance divided by the change in time. Although there is significant noise present in the reconstruction of the left and right wheel velocities, the overall experimental values closely match the theoretical values.

Using the reconstructed left and right wheel velocities, ω_{actual} and V_{actual} were calculated using Equations 8 and 9, which describe the relationship between

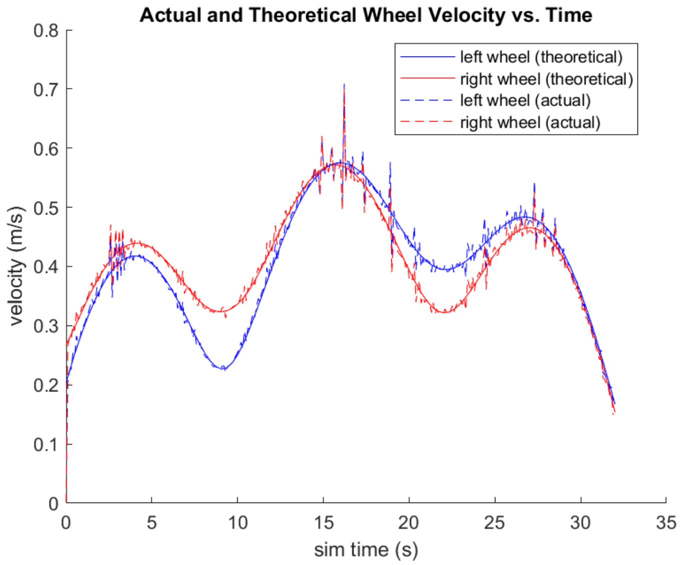

Figure 4: NEATO's Theoretical vs. Actual Left and Right Wheel Velocities

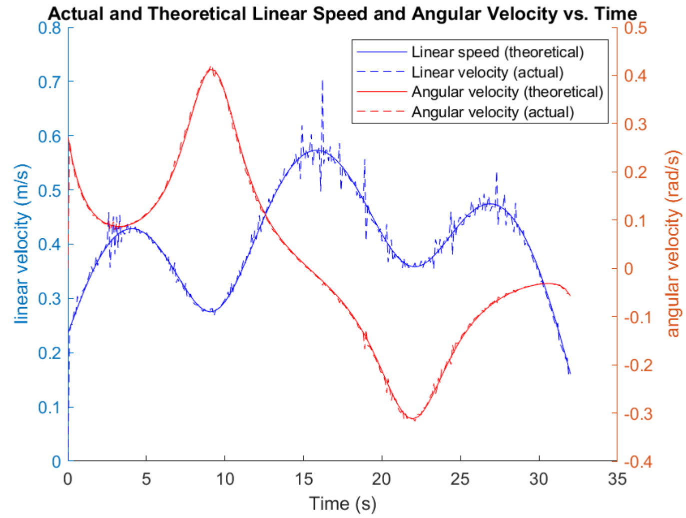

Figure 5: NEATO's Theoretical vs. Actual Linear Speed and Angular Velocity

wheel velocities and ω and V. The resulting reconstruction based on experimental data is plotted in Figure 5. Similarly to the path plotted in Figure 5, there is significant noise present in the reconstruction of linear and angular velocity, but the overall experimental values closely align with the theoretical values.

$$\omega = \frac{V_R - V_L}{d} \tag{8}$$

$$V = \frac{V_L + V_R}{2} \tag{9}$$

In order to reconstruct the experimental path of the NEATO shown in Figure 6, the angular velocity ω was numerically integrated with respect to time to calculate θ, the NEATO heading. The linear speed V was also numerically integrated with respect to time to get the x and y positions of the robot. The unit tangent vector was calculated using Equation 10 and normalized for plotting. There is a small degree of separation between the actual and theoretical paths over time; by examining the reconstructed values and comparing them to the theoretical values, we can conclude that this drift off the nominal path is due to noise in the encoder data, which impacted the calculations throughout.

$$\hat{\mathbf{T}} = cos\theta\hat{\mathbf{i}} + sin\theta\hat{\mathbf{j}} \tag{10}$$

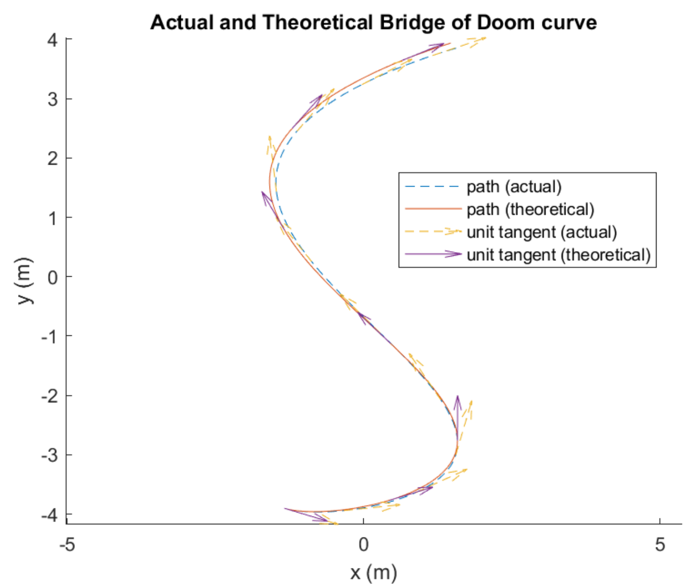

Figure 6: NEATO's Theoretical vs. Actual Path and Theoretical vs. Actual Unit Tangent Vectors

A video of the NEATO successfully traversing the Bridge of Doom is available at https://youtu.be/z9KYcnpo-5o. A MATLAB Drive folder containing our code is available at https://drive.matlab.com/sharing/e9e8a016-aee1-4200-a862-4c200a550dfa/.

Appendix: Simulator Code

```
function starterCodeForBridgeOfDoomQEA2021()

% Explicitly defining u
u = [];
% u will be our parameter
syms u;

% Assumptions
assume(u,{'real', 'positive'});

% Dilation - conversion from the raw parameterization of the curve to time.
% As the dilation increases, the time that the robot will take to travel
% along the curve increases.
dilation = 10;

% Equation of the bridge (scaled by dilation)
R = 4*[0.396*cos(2.65*((u/dilation)+1.4));...
       -0.99*sin((u/dilation)+1.4); 0];

% Bounds of u (with respect to dilation)
timeBounds = [0 (3.2*dilation)];

% Tangent vector
T = diff(R);

% Normalized tangent vector
That = T/norm(T);

% Linear speed vector
linearSpeed = simplify(norm(T));

% Angular velocity vector
omega = cross(That, diff(That));

% Left and Right Wheel Velocities
d = 0.235;
velocity_left = linearSpeed - ((d/2)*omega(3));
velocity_right = linearSpeed + ((d/2)*omega(3));

pub = rospublisher('raw_vel');

% Pause 1 second
pause(1);

% Stop the robot if it's going right now
stopMsg = rosmessage(pub);
stopMsg.Data = [0 0];
send(pub, stopMsg);

```

```
% Pause 1 second
pause(1);

bridgeStart = double(subs(R,u,timeBounds(1)));
startingThat = double(subs(That,u,timeBounds(1)));
placeNeato(bridgeStart(1),  bridgeStart(2), startingThat(1), ...
startingThat(2));

% Wait a bit for robot to fall onto the bridge
pause(1);

% Driving
rostic;
u_val = 0;
while u_val < timeBounds(2)
    t = rostoc;
    u_val = t;
    msg = rosmessage(pub);
    % Substitute current time (u value) into expressions above
    velocity_L = double(subs(velocity_left, u, u_val));
    velocity_R = double(subs(velocity_right, u, u_val));
    msg.Data = [velocity_L,velocity_R];
    % send messages to the robot
    send(pub,msg);
end

% Stop the robot
msg = rosmessage(pub);
msg.Data = [0 0];
send(pub,msg);
clear pub;

end
```

Appendix: Calculations and Plotting Code

```
Robo: Bridge of Doom Challenge
Authors: Berwin Lan and Cara Mulrooney

Exercise 21.1
1. For the Bridge of Doom, plot the parametric curve that defines the centerline of the bridge.
2. On the same figure, plot the unit tangent and unit normal vectors at several points along the
curve. You should have starter code to help with this in the Robo Homework 1 assignment.

clc;

% Load data
encoder_data = table2array(readtable("encoder_data.csv"));
timeseconds = encoder_data(9:end,1);
timeseconds = timeseconds - timeseconds(1);
encoderLeftmeters = encoder_data(9:end,2);
encoderRightmeters = encoder_data(9:end,3);

% The position equations
dilation = 0.10;

% Equations
syms t
ri = 4 * 0.396*cos(2.65*((dilation*t) + 1.4)); % x-component of vector
rj = 4 * -0.99*sin((dilation*t) + 1.4);         % y-component of vector
rk = 0 * (dilation*t);                          % z-component of vector
r = [ri, rj, rk];

% Find tangent vector
dr = diff(r, t);
T_hat=simplify(dr ./ norm(dr));

% Find normal vector
dT_hat=diff(T_hat,t);
N_hat=simplify(dT_hat ./ norm(dT_hat));

% Plotting the position curve
figure(1); clf; hold on;
fplot(r(1),r(2),[0,3.2/dilation])
title("Bridge of Doom curve"); xlabel("x (m)"); ylabel("y (m)"); axis equal;
% define a set of evenly spaced points between 0 and 3.2/dilation
t_num = linspace(0, 3.2/dilation, 10);

% Substitute values into equations
for n=1:length(t_num)
     r_num(n,:) = double(subs(r, t, t_num(:,n)));
     T_hat_num(n,:) = double(subs(T_hat, t, t_num(:,n)));
     N_hat_num(n,:) = double(subs(N_hat, t, t_num(:,n)));

    % Plot vectors
```

```
    hold on
    quiver3(r_num(n,1), r_num(n,2), r_num(n,3), T_hat_num(n,1), T_hat_num(n,2), T_hat_num(n,3),
       'r') % plot the unit tangent
    quiver3(r_num(n,1), r_num(n,2), r_num(n,3), N_hat_num(n,1), N_hat_num(n,2), T_hat_num(n,3),
       'b') % plot the unit normal

end
legend({"path (theoretical)", "unit tangent (theoretical)", "unit normal (theoretical)"},
    "Location", "best"); axis equal; hold off;
```

Exercise 21.2

1. Given the parametric curve, compute and plot the linear speed and angular velocity at its COM as a function of time for the Bridge of Doom. (Reference code: Ex 18.2)

```
% Find angular velocity
omega = simplify(cross(T_hat, dT_hat));

% Find linear speed
speed = simplify(norm(dr)); % m/s, first derivative of position vector

% Plotting
t_num = linspace(0, 3.2/dilation, 100); % define a set of evenly spaced points between 0 and 3.2

% Initialize time, speed, and omega vectors
% t_values = [];
speed_values = [];
omega_values = [];

% Calculate values for speed and omega
for n=1:length(t_num)
    omega_plot = double(subs(omega, t, t_num(:,n)));

    % t_values(1,n) = n;
    speed_values(1,n) = double(subs(speed, t, t_num(:,n)));
    omega_values(1,n) = double(subs(omega(3), t, t_num(:,n)));
end

% Plot vectors
figure(3); clf; hold on
yyaxis left;
plot(t_num, speed_values, "b", "DisplayName", "Linear speed (theoretical)");
yyaxis right;
plot(t_num, omega_values, "r", "DisplayName", "Angular velocity (theoretical)");
legend("Location", "best");
xlabel("sim time (s)"); title("Linear Speed and Angular Velocity vs. Time");
yyaxis left; ylabel("linear speed (m/s)");
yyaxis right; ylabel("angular velocity (rad/s)");
```

2. After successfully traversing the bridge, use your encoder data measurements to compute the linear speed and angular velocity of your robot and add this to your plot.

```
% Set parameter for distance between wheels
```

```
d = 0.235;      % m

% Find wheel velocities
velocityLeft = diff(encoderLeftmeters) ./ diff(timeseconds);
velocityRight = diff(encoderRightmeters) ./ diff(timeseconds);

% Find angular velocity
omega_actual = (velocityRight - velocityLeft) ./ d;  % rad/s

% Find linear speed
speed_actual = (velocityLeft + velocityRight) ./ 2; % m/s

% Plot vectors
figure(3); hold on;
yyaxis left;
plot(timeseconds(1:end-1,:), speed_actual, "b--", "DisplayName", "Linear velocity (actual)");
yyaxis right;
plot(timeseconds(1:end-1,:), omega_actual, "r--", "DisplayName", "Angular velocity (actual)");
legend("Location", "best");
xlabel("Time (s)"); title("Actual and Theoretical Linear Speed and Angular Velocity vs. Time");
yyaxis left; ylabel("linear velocity (m/s)");
yyaxis right; ylabel("angular velocity (rad/s)");
```

Exercise 21.3

1. Compute and plot your robot's left and right wheel velocities as a function of time for the Bridge of Doom.

```
d_num = 0.235;  % m
V_left = simplify(speed - omega(1,3) * d_num / 2);
V_right = simplify(speed + omega(1,3) * d_num / 2);

% Initialize time, speed, and omega vectors
clf; figure();
left_values = [];
right_values = [];

% Calculate values for speed and omega
for n=1:length(t_num)
    left_values(1,n) = double(subs(V_left, t, t_num(:,n)));
    right_values(1,n) = double(subs(V_right, t, t_num(:,n)));
end

figure(2); clf; hold on;
plot(t_num, left_values, "b", "DisplayName", "left wheel (theoretical)");
plot(t_num, right_values, "r", "DisplayName", "right wheel (theoretical)");
title("Wheel Velocity vs. Time"); xlabel("sim time (s)"); ylabel("velocity (m/s)");
    legend("Location", "best")
```

2. After successfully traversing the bridge, use your encoder data to compute the measured wheel velocities and add them to your plot.

```
figure(2); hold on; legend("Location", "best")
```

```
plot(timeseconds(1:end-1,:), velocityLeft, "b--", "DisplayName", "left wheel (actual)");
plot(timeseconds(1:end-1,:), velocityRight, "r--", "DisplayName", "right wheel (actual)");
title("Actual and Theoretical Wheel Velocity vs. Time")
```

Exercise 21.4

See starterCodeForBridgeOfDoomQEA2021.m

Exercise 21.5

Map your robots predicted and actual path crossing the Bridge of Doom using encoder values as your robot traverses the bridge, convert that to coordinates and headings for the robot throughout its perilous journey. Use the Matlab quiver command to plot the predicted and experimental unit tangent vectors at various points along the curve (note: do not include an arrow for every time step or your plot will be too cluttered). The planned (theoretical) path should be plotted with a solid line, while the experimental result should be plotted with a dashed line. Make sure your plots include appropriate units, labels, and legends.

```
% Integrate the angular velocity to get theta
theta_integrated = cumtrapz(timeseconds(1:end-1,:), omega_actual);

% Calculate and adjust starting theta
shift_That = double(subs(T_hat, t, timeseconds(1)));
shift_theta = atan(shift_That(2)/shift_That(1));
theta_integrated = theta_integrated + shift_theta;

% Get speed components from actual speed
speed_x = speed_actual .* cos(theta_integrated);
speed_y = speed_actual .* sin(theta_integrated);

% Integrate speed components to get location coordinates
pos_x = cumtrapz(timeseconds(1:end-1,:), speed_x);
pos_y = cumtrapz(timeseconds(1:end-1,:), speed_y);
location_actual = [pos_x pos_y]';

% Calculate the starting position
shift_r = double(subs(r, t, timeseconds(1)));

% Adjusting for starting position
location_actual(1,:) = location_actual(1,:) + shift_r(1);
location_actual(2,:) = location_actual(2,:) + shift_r(2);

% Calculate tangents
T_hat_u = cos(theta_integrated) ./ sqrt(cos(theta_integrated) .^ 2 + sin(theta_integrated) .^ 2);
T_hat_v = sin(theta_integrated) ./ sqrt(cos(theta_integrated) .^ 2 + sin(theta_integrated) .^ 2);
x_vector = [];
y_vector = [];
u_vector = [];
v_vector = [];

% Plotting the position curve
figure(11); clf; hold on; legend("Location", "best")
title("Actual and Theoretical Bridge of Doom curve"); xlabel("x (m)"); ylabel("y (m)");
figure(11); hold on; plot(location_actual(1,:), location_actual(2,:), "--", "DisplayName",
```

```
    "path (actual)"); axis equal;

% Plot theoretical path
fplot(r(1),r(2),[0,3.2/dilation],"DisplayName","path (theoretical)")

for i=1:30:328
    x_vector(i,1) = location_actual(1,i);
    y_vector(i,1) = location_actual(2,i);
    u_vector(i,1) = T_hat_u(i,:);
    v_vector(i,1) = T_hat_v(i,:);
end

scale = 2;

% Plot actual unit tangent
quiver(x_vector(:,1), y_vector(:,1),u_vector(:,1),v_vector(:,1), scale, "--", "DisplayName",
    "unit tangent (actual)"); hold on;

r_num = [];
T_hat_num = [];

for n=1:15:length(t_num)
     r_num(n,:) = double(subs(r, t, t_num(:,n)));
     T_hat_num(n,:) = double(subs(T_hat, t, t_num(:,n)));
end

quiver(r_num(:,1), r_num(:,2), T_hat_num(:,1), T_hat_num(:,2), "DisplayName",
    "unit tangent (theoretical)") % plot the unit tangent
```