

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 $\vec{r}(u) = 4[0.3960\cos(2.65(u + 1.4)\hat{i} - 0.99\sin(u + 1.4)\hat{j}]$, ($u \in [0, 3.2]$). 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, $b = 10$.

$$\vec{r}(u) = 4[0.3960\cos(2.65(u + 1.4)\hat{i} - 0.99\sin(u + 1.4)\hat{j}], (u \in [0, 3.2]) \quad (1)$$

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

$$\hat{T} = \frac{\mathbf{r}'}{|\mathbf{r}'|} \quad (2)$$

The unit normal is described by Equation 3

$$\hat{N} = \frac{\hat{T}'}{|\hat{T}'|} \quad (3)$$

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 equa-

tions, 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.

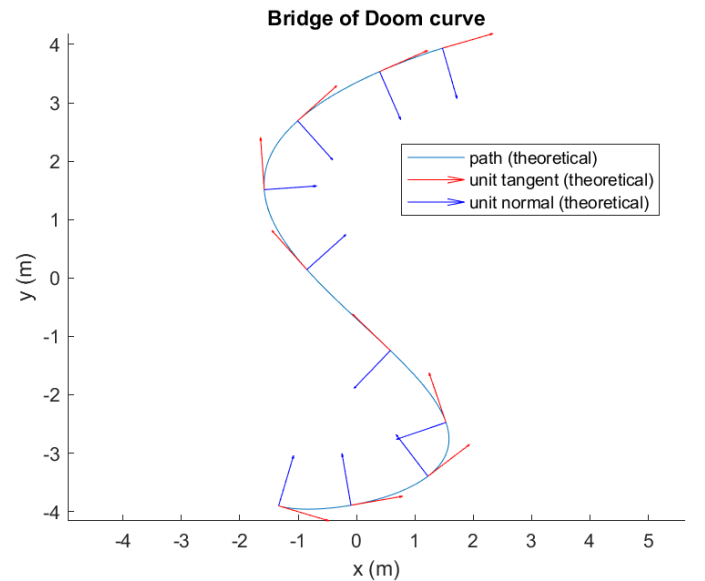


Figure 1: NEATO's Theoretical Path, Unit Tangent Vectors, and Unit Normal Vectors

$$\vec{\omega} = (\hat{\mathbf{T}} \times \frac{d\hat{\mathbf{T}}}{dt})\hat{k} \quad (4)$$

$$V = \left| \frac{d\vec{r}}{dt} \right| \quad (5)$$

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 $t = 10$ s represents the NEATO turning left and its minimum at $t = 22$ s represents the NEATO turning right. In addition, the NEATO's linear speed has minima when turning (ex. $t = 10$ s, 22 s) and maxima when traveling between turns (ex. $t = 4$ s, 15 s, 26 s)

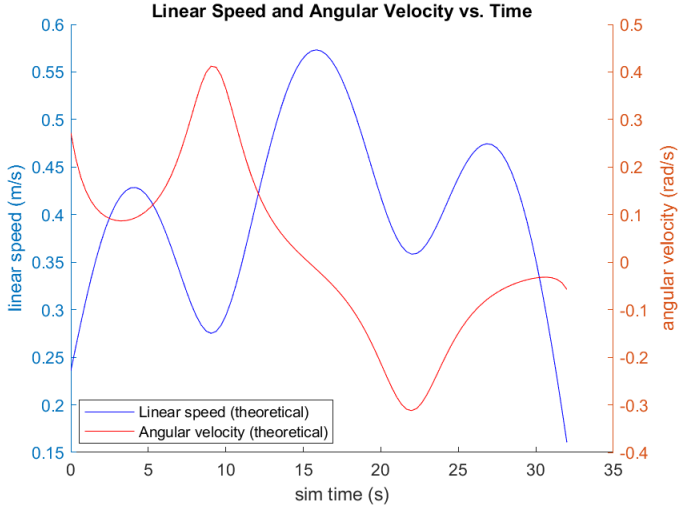


Figure 2: NEATO's Theoretical Linear Speed and Angular Velocity

$$V_L = V - \omega \frac{d}{2} \quad (6)$$

$$V_R = V + \omega \frac{d}{2} \quad (7)$$

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

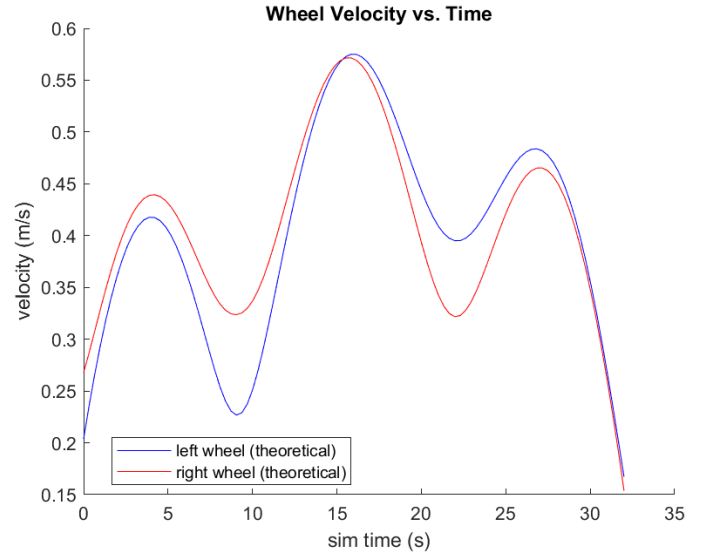


Figure 3: NEATO's Theoretical Left and Right Wheel Velocities

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 $t = [0, 15)$ s. When $V_L > V_R$, the NEATO is turning left, which occurs between $t = (15, 30)$ s. A greater difference in wheel velocities indicates a sharper turn. At $t = 15$ s and $t > 30$ s, the left and right wheels are traveling at equal velocities; thus, the NEATO is driving in a straight line.

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

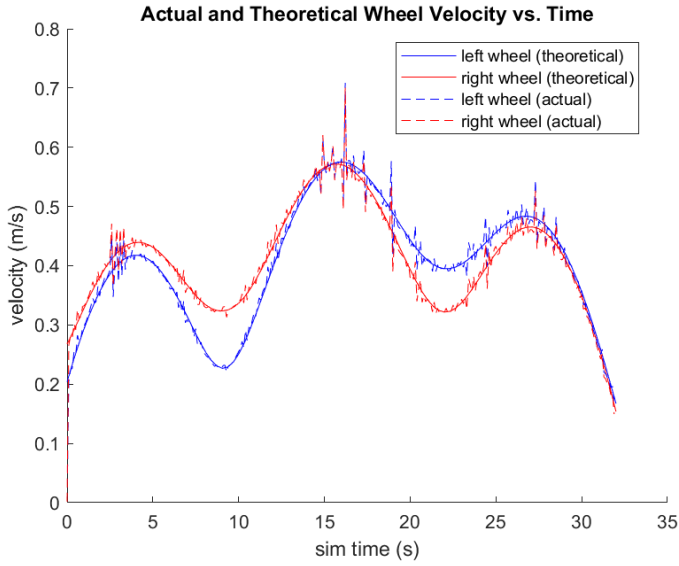


Figure 4: NEATO's Theoretical vs. Actual Left and Right Wheel Velocities

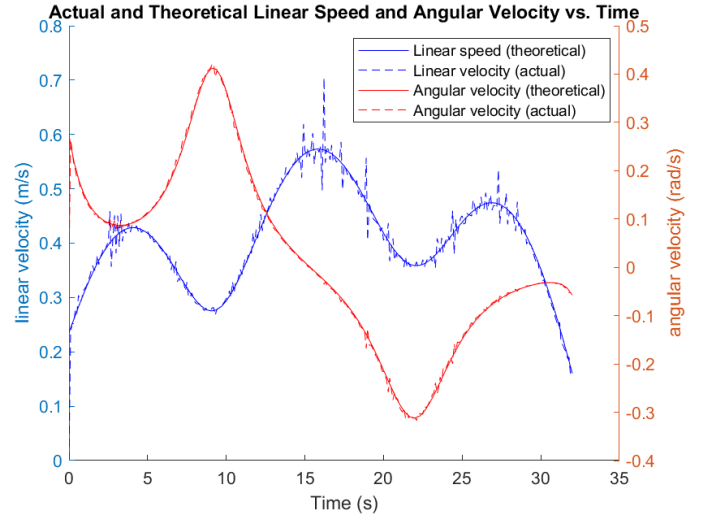


Figure 5: NEATO's Theoretical vs. Actual Linear Speed and Angular Velocity

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.

$$\omega = \frac{V_R - V_L}{d} \quad (8)$$

$$V = \frac{V_L + V_R}{2} \quad (9)$$

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.

$$\hat{\mathbf{T}} = \cos\theta\hat{\mathbf{i}} + \sin\theta\hat{\mathbf{j}} \quad (10)$$

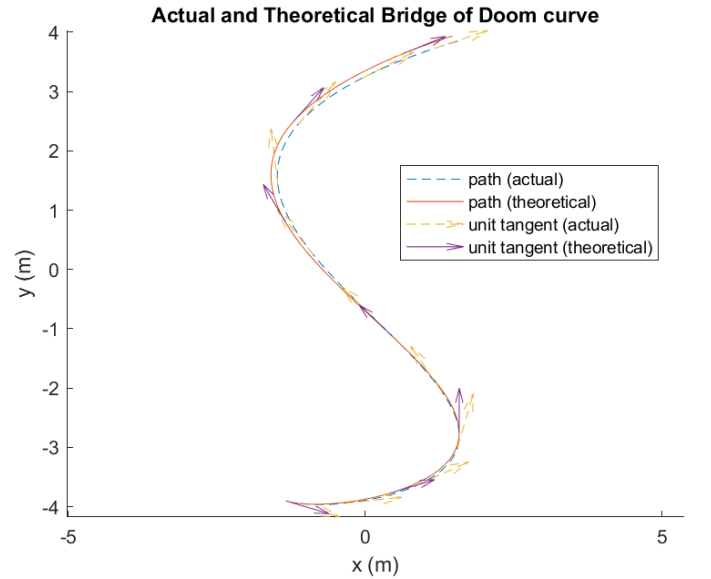


Figure 6: NEATO's Theoretical vs. Actual Path and Theoretical vs. Actual Unit Tangent Vectors

A video of the NEATO successfully traversing the Bridge of Doom is available at <https://youtu.be/z9KYcnp0-5o>. A MATLAB Drive folder containing our code is available at <https://drive.matlab.com/sharing/e9e8a016-ae1-4200-a862-4c200a550dfa/>.

Appendix: Simulator Code

```
1  function starterCodeForBridgeOfDoomQEA2021()
2
3  % Explicitly defining u
4  u = [];
5  % u will be our parameter
6  syms u;
7
8  % Assumptions
9  assume(u,{'real', 'positive'});
10
11 % Dilation - conversion from the raw parameterization of the curve to time.
12 % As the dilation increases, the time that the robot will take to travel
13 % along the curve increases.
14 dilation = 10;
15
16 % Equation of the bridge (scaled by dilation)
17 R = 4*[0.396*cos(2.65*((u/dilation)+1.4));...
18        -0.99*sin((u/dilation)+1.4); 0];
19
20 % Bounds of u (with respect to dilation)
21 timeBounds = [0 (3.2*dilation)];
22
23 % Tangent vector
24 T = diff(R);
25
26 % Normalized tangent vector
27 That = T/norm(T);
28
29 % Linear speed vector
30 linearSpeed = simplify(norm(T));
31
32 % Angular velocity vector
33 omega = cross(That, diff(That));
34
35 % Left and Right Wheel Velocities
36 d = 0.235;
37 velocity_left = linearSpeed - ((d/2)*omega(3));
38 velocity_right = linearSpeed + ((d/2)*omega(3));
39
40 pub = rospublisher('raw_vel');
41
42 % Pause 1 second
43 pause(1);
44
45 % Stop the robot if it's going right now
46 stopMsg = rosmessage(pub);
47 stopMsg.Data = [0 0];
48 send(pub, stopMsg);
49
```

```

50  % Pause 1 second
51  pause(1);
52
53  bridgeStart = double(subs(R,u,timeBounds(1)));
54  startingThat = double(subs(That,u,timeBounds(1)));
55  placeNeato(bridgeStart(1), bridgeStart(2), startingThat(1), ...
56  startingThat(2));
57
58  % Wait a bit for robot to fall onto the bridge
59  pause(1);
60
61  % Driving
62  rostopic;
63  u_val = 0;
64  while u_val < timeBounds(2)
65      t = rostopic;
66      u_val = t;
67      msg = rostopic(pub);
68      % Substitute current time (u value) into expressions above
69      velocity_L = double(subs(velocity_left, u, u_val));
70      velocity_R = double(subs(velocity_right, u, u_val));
71      msg.Data = [velocity_L,velocity_R];
72      % send messages to the robot
73      send(pub,msg);
74  end
75
76  % Stop the robot
77  msg = rostopic(pub);
78  msg.Data = [0 0];
79  send(pub,msg);
80  clear pub;
81
82  end

```

Appendix: Calculations and Plotting Code

```
1 Robo: Bridge of Doom Challenge
2 Authors: Berwin Ian and Cara Mulrooney
3
4 Exercise 21.1
5 1. For the Bridge of Doom, plot the parametric curve that defines the centerline of the bridge.
6 2. On the same figure, plot the unit tangent and unit normal vectors at several points along the
7 curve. You should have starter code to help with this in the Robo Homework 1 assignment.
8
9 clc;
10
11 % Load data
12 encoder_data = table2array(readtable("encoder_data.csv"));
13 timesecseconds = encoder_data(9:end,1);
14 timesecseconds = timesecseconds - timesecseconds(1);
15 encoderLeftmeters = encoder_data(9:end,2);
16 encoderRightmeters = encoder_data(9:end,3);
17
18 % The position equations
19 dilation = 0.10;
20
21 % Equations
22 syms t
23 ri = 4 * 0.396*cos(2.65*((dilation*t) + 1.4)); % x-component of vector
24 rj = 4 * -0.99*sin((dilation*t) + 1.4); % y-component of vector
25 rk = 0 * (dilation*t); % z-component of vector
26 r = [ri, rj, rk];
27
28 % Find tangent vector
29 dr = diff(r, t);
30 T_hat=simplify(dr ./ norm(dr));
31
32 % Find normal vector
33 dT_hat=diff(T_hat,t);
34 N_hat=simplify(dT_hat ./ norm(dT_hat));
35
36 % Plotting the position curve
37 figure(1); clf; hold on;
38 fplot(r(1),r(2),[0,3.2/dilation])
39 title("Bridge of Doom curve"); xlabel("x (m)"); ylabel("y (m)"); axis equal;
40 % define a set of evenly spaced points between 0 and 3.2/dilation
41 t_num = linspace(0, 3.2/dilation, 10);
42
43 % Substitute values into equations
44 for n=1:length(t_num)
45     r_num(n,:) = double(subs(r, t, t_num(:,n)));
46     T_hat_num(n,:) = double(subs(T_hat, t, t_num(:,n)));
47     N_hat_num(n,:) = double(subs(N_hat, t, t_num(:,n)));
48
49     % Plot vectors
```

```

50     hold on
51     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),
52         'r') % plot the unit tangent
53     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),
54         'b') % plot the unit normal
55
56 end
57 legend({"path (theoretical)", "unit tangent (theoretical)", "unit normal (theoretical)"},
58     "Location", "best"); axis equal; hold off;
59
60 Exercise 21.2
61 1. Given the parametric curve, compute and plot the linear speed and angular velocity at its COM as
62 a function of time for the Bridge of Doom. (Reference code: Ex 18.2)
63
64 % Find angular velocity
65 omega = simplify(cross(T_hat, dT_hat));
66
67 % Find linear speed
68 speed = simplify(norm(dr)); % m/s, first derivative of position vector
69
70 % Plotting
71 t_num = linspace(0, 3.2/dilation, 100); % define a set of evenly spaced points between 0 and 3.2
72
73 % Initialize time, speed, and omega vectors
74 % t_values = [];
75 speed_values = [];
76 omega_values = [];
77
78 % Calculate values for speed and omega
79 for n=1:length(t_num)
80     omega_plot = double(subs(omega, t, t_num(:,n)));
81
82     % t_values(1,n) = n;
83     speed_values(1,n) = double(subs(speed, t, t_num(:,n)));
84     omega_values(1,n) = double(subs(omega(3), t, t_num(:,n)));
85 end
86
87 % Plot vectors
88 figure(3); clf; hold on
89 yyaxis left;
90 plot(t_num, speed_values, "b", "DisplayName", "Linear speed (theoretical)");
91 yyaxis right;
92 plot(t_num, omega_values, "r", "DisplayName", "Angular velocity (theoretical)");
93 legend("Location", "best");
94 xlabel("sim time (s)"); title("Linear Speed and Angular Velocity vs. Time");
95 yyaxis left; ylabel("linear speed (m/s)");
96 yyaxis right; ylabel("angular velocity (rad/s)");
97
98 2. After successfully traversing the bridge, use your encoder data measurements to compute the
99 linear speed and angular velocity of your robot and add this to your plot.
100
101 % Set parameter for distance between wheels

```

```

102 d = 0.235;          % m
103
104 % Find wheel velocities
105 velocityLeft = diff(encoderLeftmeters) ./ diff(timeseconds);
106 velocityRight = diff(encoderRightmeters) ./ diff(timeseconds);
107
108 % Find angular velocity
109 omega_actual = (velocityRight - velocityLeft) ./ d; % rad/s
110
111 % Find linear speed
112 speed_actual = (velocityLeft + velocityRight) ./ 2; % m/s
113
114 % Plot vectors
115 figure(3); hold on;
116 yyaxis left;
117 plot(timeseconds(1:end-1,:), speed_actual, "b--", "DisplayName", "Linear velocity (actual)");
118 yyaxis right;
119 plot(timeseconds(1:end-1,:), omega_actual, "r--", "DisplayName", "Angular velocity (actual)");
120 legend("Location", "best");
121 xlabel("Time (s)"); title("Actual and Theoretical Linear Speed and Angular Velocity vs. Time");
122 yyaxis left; ylabel("linear velocity (m/s)");
123 yyaxis right; ylabel("angular velocity (rad/s)");
124
125 Exercise 21.3
126 1. Compute and plot your robot's left and right wheel velocities as a function of time for the
127 Bridge of Doom.
128
129 d_num = 0.235; % m
130 V_left = simplify(speed - omega(1,3) * d_num / 2);
131 V_right = simplify(speed + omega(1,3) * d_num / 2);
132
133 % Initialize time, speed, and omega vectors
134 clf; figure();
135 left_values = [];
136 right_values = [];
137
138 % Calculate values for speed and omega
139 for n=1:length(t_num)
140     left_values(1,n) = double(subs(V_left, t, t_num(:,n)));
141     right_values(1,n) = double(subs(V_right, t, t_num(:,n)));
142 end
143
144 figure(2); clf; hold on;
145 plot(t_num, left_values, "b", "DisplayName", "left wheel (theoretical)");
146 plot(t_num, right_values, "r", "DisplayName", "right wheel (theoretical)");
147 title("Wheel Velocity vs. Time"); xlabel("sim time (s)"); ylabel("velocity (m/s)");
148 legend("Location", "best")
149
150 2. After successfully traversing the bridge, use your encoder data to compute the measured wheel
151 velocities and add them to your plot.
152
153 figure(2); hold on; legend("Location", "best")

```



```

154 plot(timesseconds(1:end-1,:), velocityLeft, "b--", "DisplayName", "left wheel (actual)");
155 plot(timesseconds(1:end-1,:), velocityRight, "r--", "DisplayName", "right wheel (actual)");
156 title("Actual and Theoretical Wheel Velocity vs. Time")
157
158 Exercise 21.4
159 See starterCodeForBridgeOfDoomQEA2021.m
160
161 Exercise 21.5
162 Map your robots predicted and actual path crossing the Bridge of Doom using encoder values as your
163 robot traverses the bridge, convert that to coordinates and headings for the robot throughout its
164 perilous journey. Use the Matlab quiver command to plot the predicted and experimental unit tangent
165 vectors at various points along the curve (note: do not include an arrow for every time step or your
166 plot will be too cluttered). The planned (theoretical) path should be plotted with a solid line,
167 while the experimental result should be plotted with a dashed line. Make sure your plots include
168 appropriate units, labels, and legends.
169
170 % Integrate the angular velocity to get theta
171 theta_integrated = cumtrapz(timesseconds(1:end-1,:), omega_actual);
172
173 % Calculate and adjust starting theta
174 shift_That = double(subs(T_hat, t, timesseconds(1)));
175 shift_theta = atan(shift_That(2)/shift_That(1));
176 theta_integrated = theta_integrated + shift_theta;
177
178 % Get speed components from actual speed
179 speed_x = speed_actual .* cos(theta_integrated);
180 speed_y = speed_actual .* sin(theta_integrated);
181
182 % Integrate speed components to get location coordinates
183 pos_x = cumtrapz(timesseconds(1:end-1,:), speed_x);
184 pos_y = cumtrapz(timesseconds(1:end-1,:), speed_y);
185 location_actual = [pos_x pos_y]';
186
187 % Calculate the starting position
188 shift_r = double(subs(r, t, timesseconds(1)));
189
190 % Adjusting for starting position
191 location_actual(1,:) = location_actual(1,:) + shift_r(1);
192 location_actual(2,:) = location_actual(2,:) + shift_r(2);
193
194 % Calculate tangents
195 T_hat_u = cos(theta_integrated) ./ sqrt(cos(theta_integrated) .^ 2 + sin(theta_integrated) .^ 2);
196 T_hat_v = sin(theta_integrated) ./ sqrt(cos(theta_integrated) .^ 2 + sin(theta_integrated) .^ 2);
197 x_vector = [];
198 y_vector = [];
199 u_vector = [];
200 v_vector = [];
201
202 % Plotting the position curve
203 figure(11); clf; hold on; legend("Location", "best")
204 title("Actual and Theoretical Bridge of Doom curve"); xlabel("x (m)"); ylabel("y (m)");
205 figure(11); hold on; plot(location_actual(1,:), location_actual(2,:), "--", "DisplayName",

```

```

206     "path (actual)"); axis equal;
207
208     % Plot theoretical path
209     fplot(r(1),r(2),[0,3.2/dilation],"DisplayName","path (theoretical)")
210
211     for i=1:30:328
212         x_vector(i,1) = location_actual(1,i);
213         y_vector(i,1) = location_actual(2,i);
214         u_vector(i,1) = T_hat_u(i,:);
215         v_vector(i,1) = T_hat_v(i,:);
216     end
217
218     scale = 2;
219
220     % Plot actual unit tangent
221     quiver(x_vector(:,1), y_vector(:,1),u_vector(:,1),v_vector(:,1), scale, "--", "DisplayName",
222         "unit tangent (actual)"); hold on;
223
224     r_num = [];
225     T_hat_num = [];
226
227     for n=1:15:length(t_num)
228         r_num(n,:) = double(subs(r, t, t_num(:,n)));
229         T_hat_num(n,:) = double(subs(T_hat, t, t_num(:,n)));
230     end
231
232     quiver(r_num(:,1), r_num(:,2), T_hat_num(:,1), T_hat_num(:,2), "DisplayName",
233         "unit tangent (theoretical)") % plot the unit tangent

```
