QEA Night 3: Crossing the Bridge of Doom

Pranavi Boyalakuntla
4/18/2019

Introduction

The goal of this challenge is to command a NEATO to cross a bridge whose shape is predefined by a parametric curve function. The NEATOS that are used are repurposed in such a way that allows us to send commands to them using ROS. The bridge in this paper is defined by equation 1.

$$0.396\cos(2.65(u+1.4))\hat{i} - 0.99\sin(u+1.4)\hat{j} + 0\hat{k}, u \in [0, 3.2]$$

Equation 1: This is the parametric equation that describes the shape of the bridge.

General Process

The NEATO takes in left and right wheel velocities. As a result, the only way to make the NEATO travel in a path defined by a function is to calculate the according wheel velocities. In addition, the fastest the NEATO can move is 0.3 m/s. To control this, we introduce a new variable: alpha. 'U' is multiplied by this value as shown in equation 2. To simplify things in the future, alpha will be substituted by 0.2.

$$r(u) = 0.396\cos(2.65(\alpha * u + 1.4))\hat{i} - 0.99\sin(\alpha * u + 1.4)\hat{j} + 0\hat{k}, u \in [0, 3.2]$$

Equation 2: This equation shows how alpha is used in defining the path.

The equation to calculate the left and right wheel velocities is $V_L = V - \omega\frac{d}{2}$ and $V_R = V + \omega\frac{d}{2}$, where 'V' is the linear velocity of the NEATO, $'\omega'$ is the angular velocity, and 'd' is the distance between the wheels. 'd' can be found by simply measuring the distance. It was found to be 0.254 m. 'V' can be found by applying the principle that the derivative of position is velocity. Therefore, if r(u) defines position, its derivative defines velocity, as shown in equation 3. The magnitude of this vector will provide the equation for 'V'. To solve for ω, the definition in equations 4 and 5 are utilized.

$$V = |r'(u)|$$

Equation 3: This equation shows how 'V' is solved for. It is simply the magnitude of the derivative of r(u).

$$\hat{T} = \frac{r'(u)}{|r'(u)|}$$

Equation 4: This is the equation needed to solve for $\hat{T}$. It is the tangent vector to the curve divided by its own magnitude which normalizes it.

$$\omega = \hat{T} \times \frac{d\hat{T}}{dt}$$

Equation 5: This equation is the definition needed in order to calculate what omega is. Let $\hat{T}$ be the normalized tangent vector to the curve.

Equation 4 is used to solve for $\hat{T}$, the normalized tangent vector to the curve. Once this is solved, it can be used in equation 5 to solve for ω. ω is found by taking the cross product of $\hat{T}$ with its derivative. Now, V_L and V_R can be solved for and sent as commands to the NEATO.

The way that worked the best for sending this information to the NEATO was by solving the equations for V_L and V_R at discrete points rather than continually updating the wheel velocities. A set of evenly spaced values from 0 to 3.2, as those are the conditions for this function, can be created and the wheel velocities can be calculated at each of those points and sent to the NEATO. Since they are discrete points, they cannot all be sent to the NEATO at once as the NEATO wouldn't follow the intended path. There needs to be a delay in between sending velocities that corresponds to the number of discrete points and alpha value chosen. An alpha value of 0.2 and 100 discrete points worked. The delay that corresponds to this is .16 seconds.

This value can be calculated by dividing 3.2 by the alpha value multiplied by the number of discrete points. The fact that the alpha value, in this scenario, is 0.2 means that the robot will take 16 seconds to complete the path (3.2/0.2 = 16). The fact that there are 100 evenly spaced discrete points means that there is a new wheel velocity every 0.16 seconds (16/100=0.16).

Asking the computer to continually calculate the wheel velocities dependent on whenever the computer chooses to update slows things down and bugs may be experienced when the NEATO is asked to act on the information sent. The NEATOs are also incredibly finicky and may not travel the exact same path every time and small changes in initial heading can greatly affect the path as well.

Plots

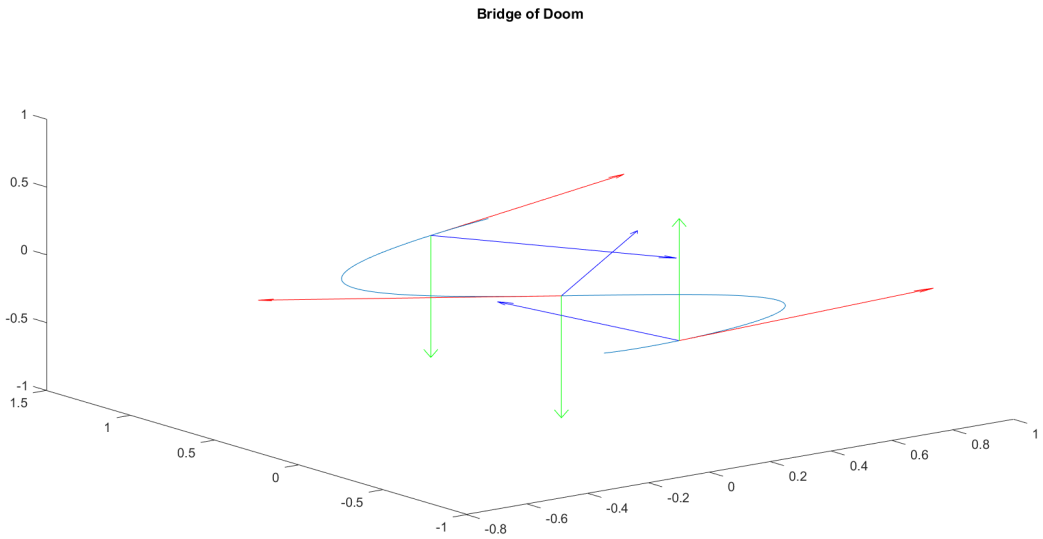


Figure 1: This image shows the intended path defined by equation 1. The normalized tangent vector (red), normalized normal vector (blue), and normalized binormal vector (green) are shown at three different points (u=10, u = 50, u = 90).

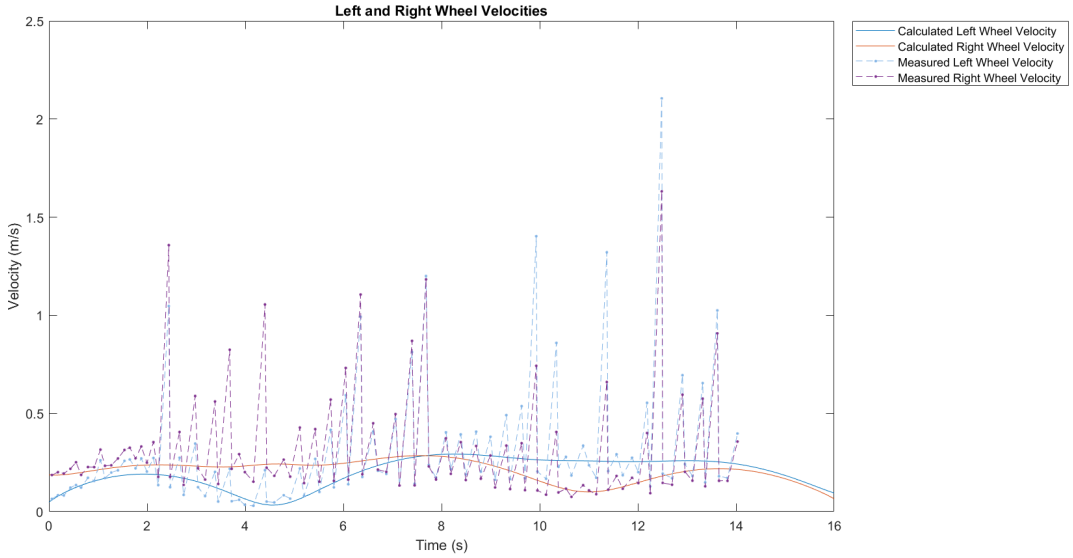


Figure 2: This graph shows the calculated and measured wheel velocities. The general trend of the measured left and right wheel velocities line up with the calculated ones which validates the theoretical velocities. The encoder does not output wheel velocity data, so it was found by taking the numerical derivative of the left and right wheel positions discretely. There are a lot of upper outliers which indicate errors in the encoder data collection. The maximum wheel speed of a NEATO is 0.3 m/s which further proves that the upper outliers are due to faulty encoder readings.

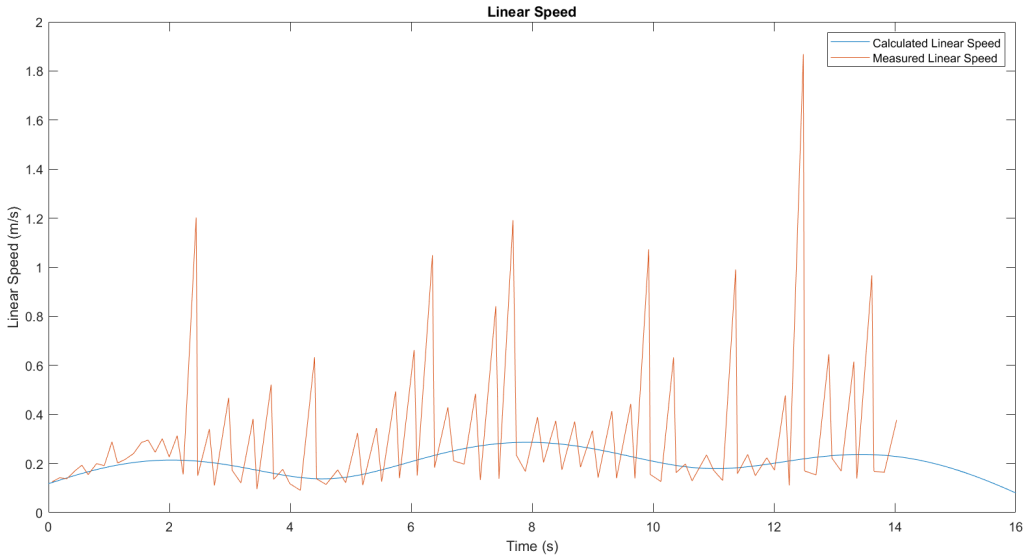


Figure 3: This graph shows the calculated and measured linear speeds of the NEATO. The encoder does not output this data, so the measured linear speed was found using the following equation: $V = \frac{V_L + V_R}{2}$. The general trend of the calculated linear speed can be found in the graph of the measured linear speed. There are a lot of upper outliers and speed values that are faster than the NEATOs can physically move. This insinuates that there was more error in encoder data collection.

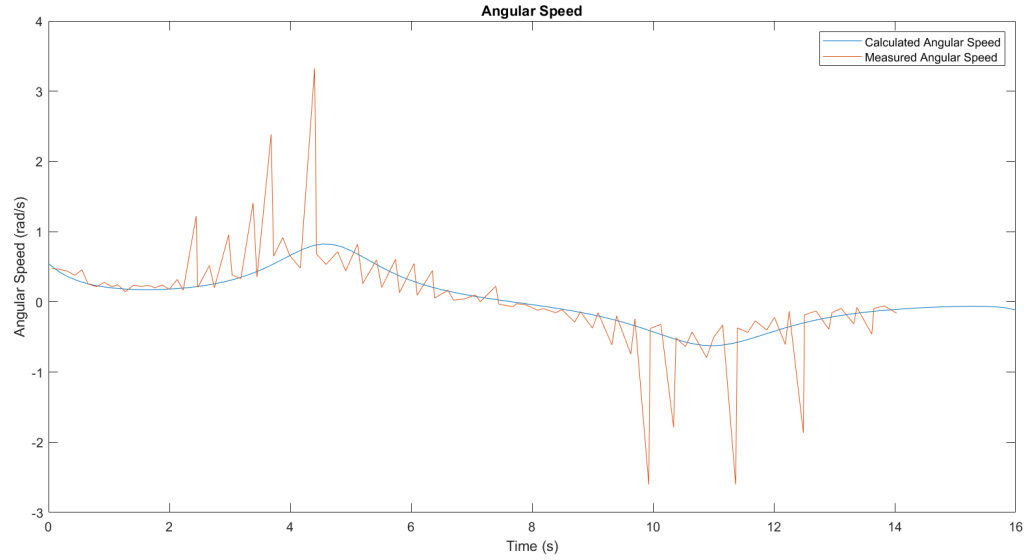


Figure 4: This graph shows the measured and calculated angular speed for this path. It can be very clearly seen that both the measured and calculated speeds follow the same trend. This data is not outputted by the encoder, so it was calculated using the measured wheel velocities with the following equation: $\omega = \frac{V_R - V_L}{d}$, where 'd' is the distance between the wheels.

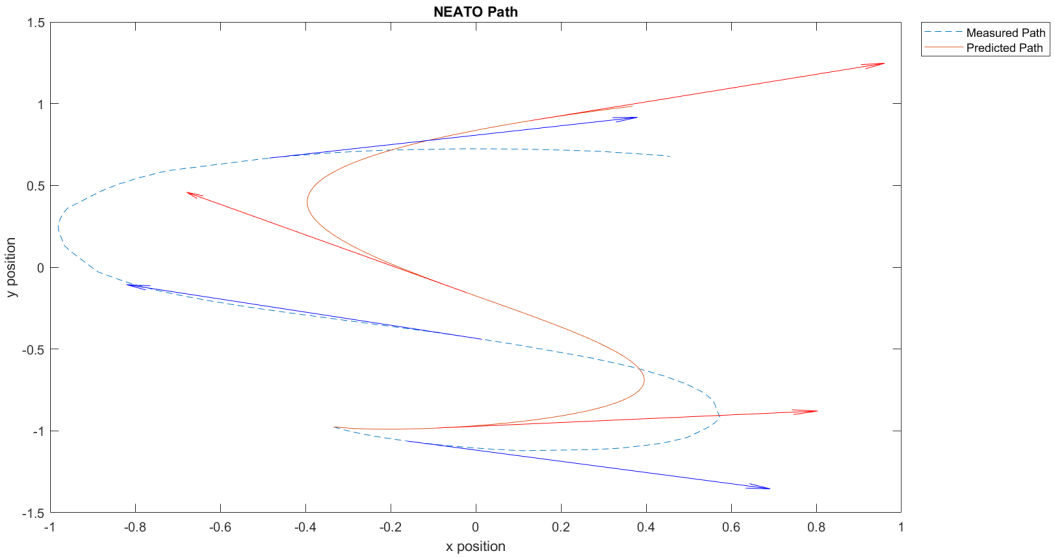


Figure 5: This is the plot of the measured and predicted paths. The arrows tangent to the two curves are the $\hat{T}$ vectors at discrete points (u = 10, 50, 90). This information is not given by the encoder, so the derivative of the measured angular velocity is used to vectorize the linear speed of the NEATO. The derivatives of the velocity vectors are the coordinates of the NEATO at each discrete point.

Error

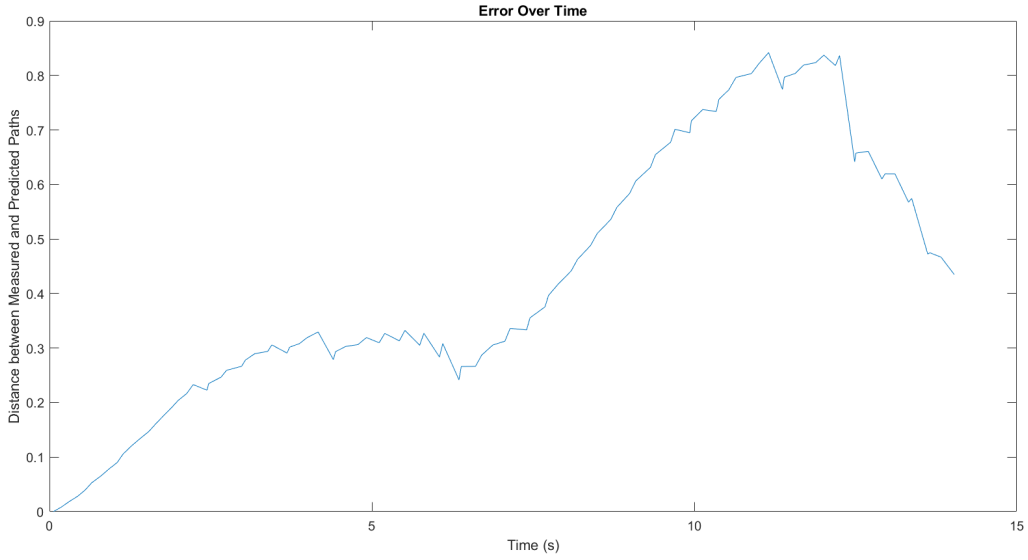


Figure 6: This graph illustrates the physical distance between the measured and predicted paths over time.

The graph in figure 6 clearly illustrates how the distance between the measured and calculated paths grows larger as time passes by. This is because the error accumulates over time.

In addition, the initial heading which is hugely impacted by human error greatly affects how closely the NEATO will follow the prescribed path. This means that, in this error graph as well, the distance from the actual path is affected heavily by the arbitrary initial heading.

The error is also much greater than what is visually seen when the NEATO drives over the bridge of doom. The NEATO stays on the bridge (for the most part)! The calculations predict that it will drive very far off of the curve. This could be because of the incorrect encoder data. The spikes in velocity cause the calculations to show that the NEATO travelled farther than it actually did.

This form of error calculation was chosen because it is the most comprehensive way of conveying that error accumulates. The general upward trend in error over time shows this idea very visually.

Bridge of Doom Mathematica

In[128]:=
```
SetDirectory[NotebookDirectory[]];
<< ToMatlab.m
```

Defining r

alpha is defined as 0.2 here for simplicity later on

In[222]:=
```
r[u_] = {0.396 * Cos[2.65 * ((u * 0.2) + 1.4)], -0.99 * Sin[(u * 0.2) + 1.4], 0}
Expand[r[u]] // ToMatlab
```

Out[222]=
$$\{0.396 \cos[2.65\,(1.4 + 0.2\,u)],\ -0.99 \sin[1.4 + 0.2\,u],\ 0\}$$

Out[223]=
```
[0.396E0.*cos(0.265E1.*(0.14E1+0.2E0.*u)),(-0.99E0).*sin(0.14E1+ ...
  0.2E0.*u),0];
```

Defining THat

In[224]:=
```
THat[u_] = Simplify[r'[u] / Sqrt[r'[u].r'[u]], {u ∈ Reals}]
Expand[THat[u]] // ToMatlab
```

Out[224]=
$$\left\{-\frac{0.20988 \sin[3.71 + 0.53\,u]}{\sqrt{0.039204 \cos[1.4 + 0.2\,u]^2 + 0.0440496 \sin[3.71 + 0.53\,u]^2}},\right.$$
$$\left.-\frac{0.198 \cos[1.4 + 0.2\,u]}{\sqrt{0.039204 \cos[1.4 + 0.2\,u]^2 + 0.0440496 \sin[3.71 + 0.53\,u]^2}},\ 0\right\}$$

Out[225]=
```
[(-0.20988E0).*sin(0.371E1+0.53E0.*u).*(0.39204E-1.*cos(0.14E1+ ...
  0.2E0.*u).^2+0.440496E-1.*sin(0.371E1+0.53E0.*u).^2).^(-1/2),( ...
  -0.198E0).*cos(0.14E1+0.2E0.*u).*(0.39204E-1.*cos(0.14E1+0.2E0.*u) ...
  .^2+0.440496E-1.*sin(0.371E1+0.53E0.*u).^2).^(-1/2),0];
```

Defining NHat

In[226]:=
```
NHat[u_] = Simplify[THat'[u] / Sqrt[THat'[u].THat'[u]], {u ∈ Reals}]
Simplify[NHat[u], {u ∈ Reals}] // ToMatlab
```

Out[226]=
$$\left\{\left(\cos[1.4+0.2\,u]\left(-0.111236\cos[1.4+0.2\,u]\cos[3.71+0.53\,u]-0.041976\sin[1.4+0.2\,u]\sin[3.71+0.53\,u]\right)\right)\Big/\left(\sqrt{\left(\cos[1.4+0.2\,u]^4\left(0.0123735\cos[3.71+0.53\,u]^2+3.48201\times10^{-19}\sin[1.4+0.2\,u]^2\right)+0.0139029\cos[1.4+0.2\,u]^2\cos[3.71+0.53\,u]^2\sin[3.71+0.53\,u]^2+0.000440496\sin[2.8+0.4\,u]^2\sin[3.71+0.53\,u]^2+\left(2.78561\times10^{-18}\cos[3.71+0.53\,u]^2+0.00197977\sin[1.4+0.2\,u]^2\right)\sin[3.71+0.53\,u]^4+0.000583657\csc[1.4+0.2\,u]^2\sin[2.8+0.4\,u]^3\sin[7.42+1.06\,u]+0.00262319\sin[2.8+0.4\,u]\sin[3.71+0.53\,u]^2\sin[7.42+1.06\,u]\right)}\right),\left(7.00898\times10^{-18}\cos[1.4+0.2\,u]^2\sin[1.4+0.2\,u]+0.117911\cos[1.4+0.2\,u]\cos[3.71+0.53\,u]\sin[3.71+0.53\,u]+0.0444946\sin[1.4+0.2\,u]\sin[3.71+0.53\,u]^2\right)\Big/\left(\sqrt{\left(\cos[1.4+0.2\,u]^4\left(0.0123735\cos[3.71+0.53\,u]^2+3.48201\times10^{-19}\sin[1.4+0.2\,u]^2\right)+0.0139029\cos[1.4+0.2\,u]^2\cos[3.71+0.53\,u]^2\sin[3.71+0.53\,u]^2+0.000440496\sin[2.8+0.4\,u]^2\sin[3.71+0.53\,u]^2+\left(2.78561\times10^{-18}\cos[3.71+0.53\,u]^2+0.00197977\sin[1.4+0.2\,u]^2\right)\sin[3.71+0.53\,u]^4+0.000583657\csc[1.4+0.2\,u]^2\sin[2.8+0.4\,u]^3\sin[7.42+1.06\,u]+0.00262319\sin[2.8+0.4\,u]\sin[3.71+0.53\,u]^2\sin[7.42+1.06\,u]\right)},0\right\}$$

Out[227]=
```
[cos(0.14E1+0.2E0.*u).*((-0.111236E0).*cos(0.14E1+0.2E0.*u).*cos( ...
 0.371E1+0.53E0.*u)+(-0.41976E-1).*sin(0.14E1+0.2E0.*u).*sin( ...
 0.371E1+0.53E0.*u)).*(cos(0.14E1+0.2E0.*u).^4.*(0.123735E-1.*cos( ...
 0.371E1+0.53E0.*u).^2+0.348201E-18.*sin(0.14E1+0.2E0.*u).^2)+ ...
 0.139029E-1.*cos(0.14E1+0.2E0.*u).^2.*cos(0.371E1+0.53E0.*u).^2.* ...
 sin(0.371E1+0.53E0.*u).^2+0.440496E-3.*sin(0.28E1+0.4E0.*u).^2.* ...
 sin(0.371E1+0.53E0.*u).^2+(0.278561E-17.*cos(0.371E1+0.53E0.*u) ...
 .^2+0.197977E-2.*sin(0.14E1+0.2E0.*u).^2).*sin(0.371E1+0.53E0.*u) ...
 .^4+0.583657E-3.*csc(0.14E1+0.2E0.*u).^2.*sin(0.28E1+0.4E0.*u) ...
 .^3.*sin(0.742E1+0.106E1.*u)+0.262319E-2.*sin(0.28E1+0.4E0.*u).* ...
 sin(0.371E1+0.53E0.*u).^2.*sin(0.742E1+0.106E1.*u)).^(-1/2),( ...
 0.700898E-17.*cos(0.14E1+0.2E0.*u).^2.*sin(0.14E1+0.2E0.*u)+ ...
 0.117911E0.*cos(0.14E1+0.2E0.*u).*cos(0.371E1+0.53E0.*u).*sin( ...
 0.371E1+0.53E0.*u)+0.444946E-1.*sin(0.14E1+0.2E0.*u).*sin(0.371E1+ ...
 0.53E0.*u).^2).*(cos(0.14E1+0.2E0.*u).^4.*(0.123735E-1.*cos( ...
 0.371E1+0.53E0.*u).^2+0.348201E-18.*sin(0.14E1+0.2E0.*u).^2)+ ...
 0.139029E-1.*cos(0.14E1+0.2E0.*u).^2.*cos(0.371E1+0.53E0.*u).^2.* ...
 sin(0.371E1+0.53E0.*u).^2+0.440496E-3.*sin(0.28E1+0.4E0.*u).^2.* ...
 sin(0.371E1+0.53E0.*u).^2+(0.278561E-17.*cos(0.371E1+0.53E0.*u) ...
 .^2+0.197977E-2.*sin(0.14E1+0.2E0.*u).^2).*sin(0.371E1+0.53E0.*u) ...
 .^4+0.583657E-3.*csc(0.14E1+0.2E0.*u).^2.*sin(0.28E1+0.4E0.*u) ...
 .^3.*sin(0.742E1+0.106E1.*u)+0.262319E-2.*sin(0.28E1+0.4E0.*u).* ...
 sin(0.371E1+0.53E0.*u).^2.*sin(0.742E1+0.106E1.*u)).^(-1/2),0];
```

Defining BHat

In[228]:=
```
BHat = Simplify[THat[u] × NHat[u], {u ∈ Reals}]
ToMatlab[BHat]
```

Out[228]=
$$\{0.,\ 0.,\ \left(-0.0220248\,\text{Cos}[1.4+0.2\,u]^3\,\text{Cos}[3.71+0.53\,u]-0.00831125\,\text{Cos}[1.4+0.2\,u]^2\,\text{Sin}[1.4+0.2\,u]\,\text{Sin}[3.71+0.53\,u]-0.0247471\,\text{Cos}[1.4+0.2\,u]\,\text{Cos}[3.71+0.53\,u]\,\text{Sin}[3.71+0.53\,u]^2-0.00933852\,\text{Sin}[1.4+0.2\,u]\,\text{Sin}[3.71+0.53\,u]^3\right)\Big/\left(\sqrt{\left(0.039204\,\text{Cos}[1.4+0.2\,u]^2+0.0440496\,\text{Sin}[3.71+0.53\,u]^2\right)\left(\text{Cos}[1.4+0.2\,u]^4\left(0.0123735\,\text{Cos}[3.71+0.53\,u]^2+3.48201\times10^{-19}\,\text{Sin}[1.4+0.2\,u]^2\right)+0.0139029\,\text{Cos}[1.4+0.2\,u]^2\,\text{Cos}[3.71+0.53\,u]^2\,\text{Sin}[3.71+0.53\,u]^2+0.000440496\,\text{Sin}[2.8+0.4\,u]^2\,\text{Sin}[3.71+0.53\,u]^2+\left(2.78561\times10^{-18}\,\text{Cos}[3.71+0.53\,u]^2+0.00197977\,\text{Sin}[1.4+0.2\,u]^2\right)\text{Sin}[3.71+0.53\,u]^4+0.000583657\,\text{Csc}[1.4+0.2\,u]^2\,\text{Sin}[2.8+0.4\,u]^3\,\text{Sin}[7.42+1.06\,u]+0.00262319\,\text{Sin}[2.8+0.4\,u]\,\text{Sin}[3.71+0.53\,u]^2\,\text{Sin}[7.42+1.06\,u]\right)\right)\right)\}$$

Out[229]=
```
[0.E-323,0.E-323,((-0.220248E-1).*cos(0.14E1+0.2E0.*u).^3.*cos( ...
  0.371E1+0.53E0.*u)+(-0.831125E-2).*cos(0.14E1+0.2E0.*u).^2.*sin( ...
  0.14E1+0.2E0.*u).*sin(0.371E1+0.53E0.*u)+(-0.247471E-1).*cos( ...
  0.14E1+0.2E0.*u).*cos(0.371E1+0.53E0.*u).*sin(0.371E1+0.53E0.*u) ...
  .^2+(-0.933852E-2).*sin(0.14E1+0.2E0.*u).*sin(0.371E1+0.53E0.*u) ...
  .^3).*((0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+0.440496E-1.*sin( ...
  0.371E1+0.53E0.*u).^2).*(cos(0.14E1+0.2E0.*u).^4.*(0.123735E-1.* ...
  cos(0.371E1+0.53E0.*u).^2+0.348201E-18.*sin(0.14E1+0.2E0.*u).^2)+ ...
  0.139029E-1.*cos(0.14E1+0.2E0.*u).^2.*cos(0.371E1+0.53E0.*u).^2.* ...
  sin(0.371E1+0.53E0.*u).^2+0.440496E-3.*sin(0.28E1+0.4E0.*u).^2.* ...
  sin(0.371E1+0.53E0.*u).^2+(0.278561E-17.*cos(0.371E1+0.53E0.*u) ...
  .^2+0.197977E-2.*sin(0.14E1+0.2E0.*u).^2).*sin(0.371E1+0.53E0.*u) ...
  .^4+0.583657E-3.*csc(0.14E1+0.2E0.*u).^2.*sin(0.28E1+0.4E0.*u) ...
  .^3.*sin(0.742E1+0.106E1.*u)+0.262319E-2.*sin(0.28E1+0.4E0.*u).* ...
  sin(0.371E1+0.53E0.*u).^2.*sin(0.742E1+0.106E1.*u))).^(-1/2)];
```

Calculating VL and VR

finding the magnitude of v(t)

In[241]:=
```
v[u_] = Sqrt[r'[u].r'[u]]
ToMatlab[v[u]]
```

Out[241]=
$$\sqrt{0.039204\,\text{Cos}[1.4+0.2\,u]^2+0.0440496\,\text{Sin}\left[2.65\,\left(1.4+0.2\,u\right)\right]^2}$$

Out[242]=
```
(0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+0.440496E-1.*sin(0.265E1.*( ...
  0.14E1+0.2E0.*u)).^2).^(1/2);
```

finding the magnitude of ω

In[243]:=
```
ω[u_] = (Cross[THat[u], THat'[u]])[[3]]
ToMatlab[ω[u]]
```

Out[243]=
$$0. - \frac{0.0000338511 \cos[1.4 + 0.2\,u]^5 \cos[3.71 + 0.53\,u]}{\left(0.039204 \cos[1.4 + 0.2\,u]^2 + 0.0440496 \sin[3.71 + 0.53\,u]^2\right)^3} - \left(0.000012774 \cos[1.4 + 0.2\,u]^4 \sin[1.4 + 0.2\,u] \sin[3.71 + 0.53\,u]\right) / \left(0.039204 \cos[1.4 + 0.2\,u]^2 + 0.0440496 \sin[3.71 + 0.53\,u]^2\right)^3 - \left(0.0000760702 \cos[1.4 + 0.2\,u]^3 \cos[3.71 + 0.53\,u] \sin[3.71 + 0.53\,u]^2\right) / \left(0.039204 \cos[1.4 + 0.2\,u]^2 + 0.0440496 \sin[3.71 + 0.53\,u]^2\right)^3 - \left(0.0000287057 \cos[1.4 + 0.2\,u]^2 \sin[1.4 + 0.2\,u] \sin[3.71 + 0.53\,u]^3\right) / \left(0.039204 \cos[1.4 + 0.2\,u]^2 + 0.0440496 \sin[3.71 + 0.53\,u]^2\right)^3 - \left(0.0000427362 \cos[1.4 + 0.2\,u] \cos[3.71 + 0.53\,u] \sin[3.71 + 0.53\,u]^4\right) / \left(0.039204 \cos[1.4 + 0.2\,u]^2 + 0.0440496 \sin[3.71 + 0.53\,u]^2\right)^3 - \frac{0.0000161269 \sin[1.4 + 0.2\,u] \sin[3.71 + 0.53\,u]^5}{\left(0.039204 \cos[1.4 + 0.2\,u]^2 + 0.0440496 \sin[3.71 + 0.53\,u]^2\right)^3}$$

Out[244]=
```
0.E-323+(-0.338511E-4).*cos(0.14E1+0.2E0.*u).^5.*cos(0.371E1+ ...
0.53E0.*u).*(0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+0.440496E-1.*sin( ...
0.371E1+0.53E0.*u).^2).^(-3)+(-0.12774E-4).*cos(0.14E1+0.2E0.*u) ...
.^4.*sin(0.14E1+0.2E0.*u).*sin(0.371E1+0.53E0.*u).*(0.39204E-1.* ...
cos(0.14E1+0.2E0.*u).^2+0.440496E-1.*sin(0.371E1+0.53E0.*u).^2).^( ...
-3)+(-0.760702E-4).*cos(0.14E1+0.2E0.*u).^3.*cos(0.371E1+0.53E0.* ...
u).*sin(0.371E1+0.53E0.*u).^2.*(0.39204E-1.*cos(0.14E1+0.2E0.*u) ...
.^2+0.440496E-1.*sin(0.371E1+0.53E0.*u).^2).^(-3)+(-0.287057E-4).* ...
cos(0.14E1+0.2E0.*u).^2.*sin(0.14E1+0.2E0.*u).*sin(0.371E1+ ...
0.53E0.*u).^3.*(0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+0.440496E-1.* ...
sin(0.371E1+0.53E0.*u).^2).^(-3)+(-0.427362E-4).*cos(0.14E1+ ...
0.2E0.*u).*cos(0.371E1+0.53E0.*u).*sin(0.371E1+0.53E0.*u).^4.*( ...
0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+0.440496E-1.*sin(0.371E1+ ...
0.53E0.*u).^2).^(-3)+(-0.161269E-4).*sin(0.14E1+0.2E0.*u).*sin( ...
0.371E1+0.53E0.*u).^5.*(0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+ ...
0.440496E-1.*sin(0.371E1+0.53E0.*u).^2).^(-3);
```

Rest of vl and vr is calculated in MATLAB

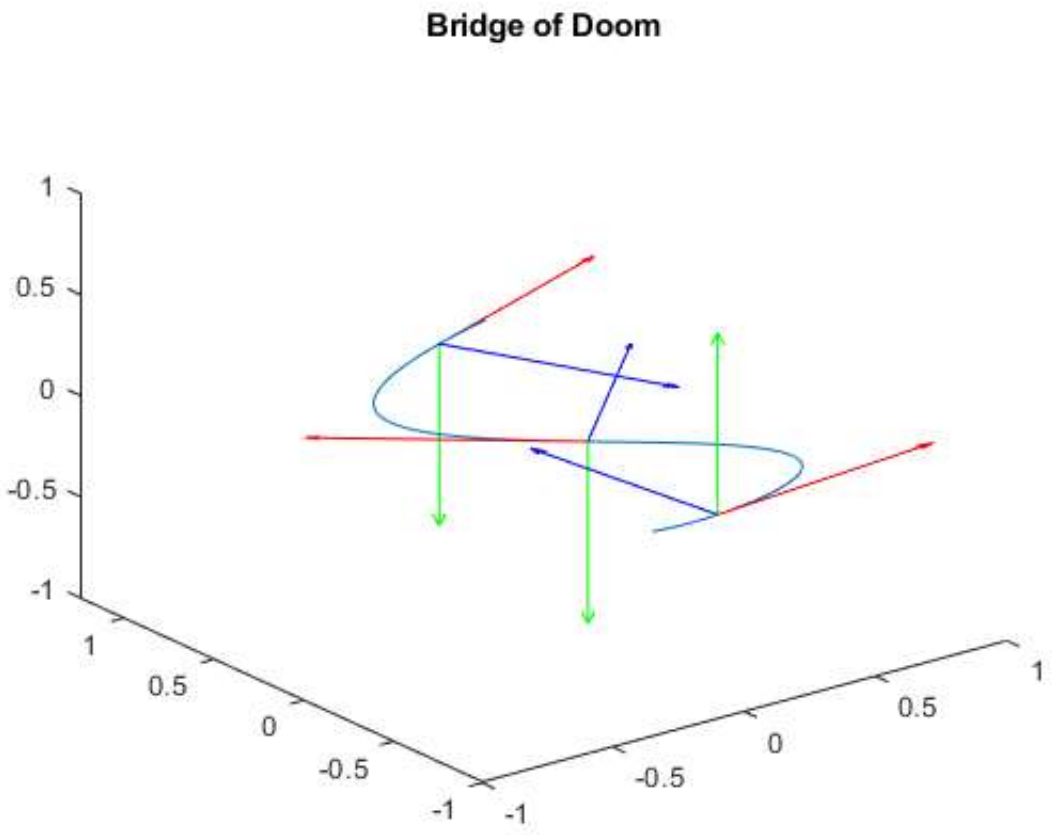
Contents


```
clear all
load('bridgeofdeathencoder.mat')

%defining a set of points to input in equations
u = linspace(0,3.2/0.2,100);
%I defined alpha as 0.2 in Mahematica.

%defining the predicted path as a vector
r=[0.396E0.*cos(0.265E1.*(0.14E1+0.2E0.*u));(-0.99E0).*sin(0.14E1+ ...
  0.2E0.*u);0*u];

%defining the normalized tangent vector
THat=[(-0.20988E0).*sin(0.371E1+0.53E0.*u).*(0.39204E-1.*cos(0.14E1+ ...
  0.2E0.*u).^2+0.440496E-1.*sin(0.371E1+0.53E0.*u).^2).^(-1/2);( ...
  -0.198E0).*cos(0.14E1+0.2E0.*u).*(0.39204E-1.*cos(0.14E1+0.2E0.*u) ...
  .^2+0.440496E-1.*sin(0.371E1+0.53E0.*u).^2).^(-1/2);0*u];

%defining the normalized normal vector
NHat=[cos(0.14E1+0.2E0.*u).*((-0.111236E0).*cos(0.14E1+0.2E0.*u).*cos( ...
  0.371E1+0.53E0.*u)+(-0.41976E-1).*sin(0.14E1+0.2E0.*u).*sin( ...
  0.371E1+0.53E0.*u)).*(cos(0.14E1+0.2E0.*u).^4.*(0.123735E-1.*cos( ...
  0.371E1+0.53E0.*u).^2+0.348201E-18.*sin(0.14E1+0.2E0.*u).^2)+ ...
  0.139029E-1.*cos(0.14E1+0.2E0.*u).^2.*cos(0.371E1+0.53E0.*u).^2.* ...
  sin(0.371E1+0.53E0.*u).^2+0.440496E-3.*sin(0.28E1+0.4E0.*u).^2.* ...
  sin(0.371E1+0.53E0.*u).^2+(0.278561E-17.*cos(0.371E1+0.53E0.*u) ...
  .^2+0.197977E-2.*sin(0.14E1+0.2E0.*u).^2).*sin(0.371E1+0.53E0.*u) ...
  .^4+0.583657E-3.*csc(0.14E1+0.2E0.*u).^2.*sin(0.28E1+0.4E0.*u) ...
  .^3.*sin(0.742E1+0.106E1.*u)+0.262319E-2.*sin(0.28E1+0.4E0.*u).* ...
  sin(0.371E1+0.53E0.*u).^2.*sin(0.742E1+0.106E1.*u)).^(-1/2);( ...
  0.700898E-17.*cos(0.14E1+0.2E0.*u).^2.*sin(0.14E1+0.2E0.*u)+ ...
  0.117911E0.*cos(0.14E1+0.2E0.*u).*cos(0.371E1+0.53E0.*u).*sin( ...
  0.371E1+0.53E0.*u)+0.444946E-1.*sin(0.14E1+0.2E0.*u).*sin(0.371E1+ ...
  0.53E0.*u).^2).*(cos(0.14E1+0.2E0.*u).^4.*(0.123735E-1.*cos( ...
  0.371E1+0.53E0.*u).^2+0.348201E-18.*sin(0.14E1+0.2E0.*u).^2)+ ...
  0.139029E-1.*cos(0.14E1+0.2E0.*u).^2.*cos(0.371E1+0.53E0.*u).^2.* ...
  sin(0.371E1+0.53E0.*u).^2+0.440496E-3.*sin(0.28E1+0.4E0.*u).^2.* ...
  sin(0.371E1+0.53E0.*u).^2+(0.278561E-17.*cos(0.371E1+0.53E0.*u) ...
  .^2+0.197977E-2.*sin(0.14E1+0.2E0.*u).^2).*sin(0.371E1+0.53E0.*u) ...
  .^4+0.583657E-3.*csc(0.14E1+0.2E0.*u).^2.*sin(0.28E1+0.4E0.*u) ...
  .^3.*sin(0.742E1+0.106E1.*u)+0.262319E-2.*sin(0.28E1+0.4E0.*u).* ...
  sin(0.371E1+0.53E0.*u).^2.*sin(0.742E1+0.106E1.*u)).^(-1/2);0*u];
```

```
%defining the normalized binormal vector
BHat=[0*u;0*u;((-0.220248E-1).*cos(0.14E1+0.2E0.*u).^3.*cos( ...
  0.371E1+0.53E0.*u)+(-0.831125E-2).*cos(0.14E1+0.2E0.*u).^2.*sin( ...
  0.14E1+0.2E0.*u).*sin(0.371E1+0.53E0.*u)+(-0.247471E-1).*cos( ...
  0.14E1+0.2E0.*u).*cos(0.371E1+0.53E0.*u).*sin(0.371E1+0.53E0.*u) ...
  .^2+(-0.933852E-2).*sin(0.14E1+0.2E0.*u).*sin(0.371E1+0.53E0.*u) ...
  .^3).*((0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+0.440496E-1.*sin( ...
  0.371E1+0.53E0.*u).^2).*(cos(0.14E1+0.2E0.*u).^4.*(0.123735E-1.* ...
  cos(0.371E1+0.53E0.*u).^2+0.348201E-18.*sin(0.14E1+0.2E0.*u).^2)+ ...
  0.139029E-1.*cos(0.14E1+0.2E0.*u).^2.*cos(0.371E1+0.53E0.*u).^2.* ...
  sin(0.371E1+0.53E0.*u).^2+0.440496E-3.*sin(0.28E1+0.4E0.*u).^2.* ...
  sin(0.371E1+0.53E0.*u).^2+(0.278561E-17.*cos(0.371E1+0.53E0.*u) ...
  .^2+0.197977E-2.*sin(0.14E1+0.2E0.*u).^2).*sin(0.371E1+0.53E0.*u) ...
  .^4+0.583657E-3.*csc(0.14E1+0.2E0.*u).^2.*sin(0.28E1+0.4E0.*u) ...
  .^3.*sin(0.742E1+0.106E1.*u)+0.262319E-2.*sin(0.28E1+0.4E0.*u).* ...
  sin(0.371E1+0.53E0.*u).^2.*sin(0.742E1+0.106E1.*u))).^(-1/2)];
```

Deliverable 1

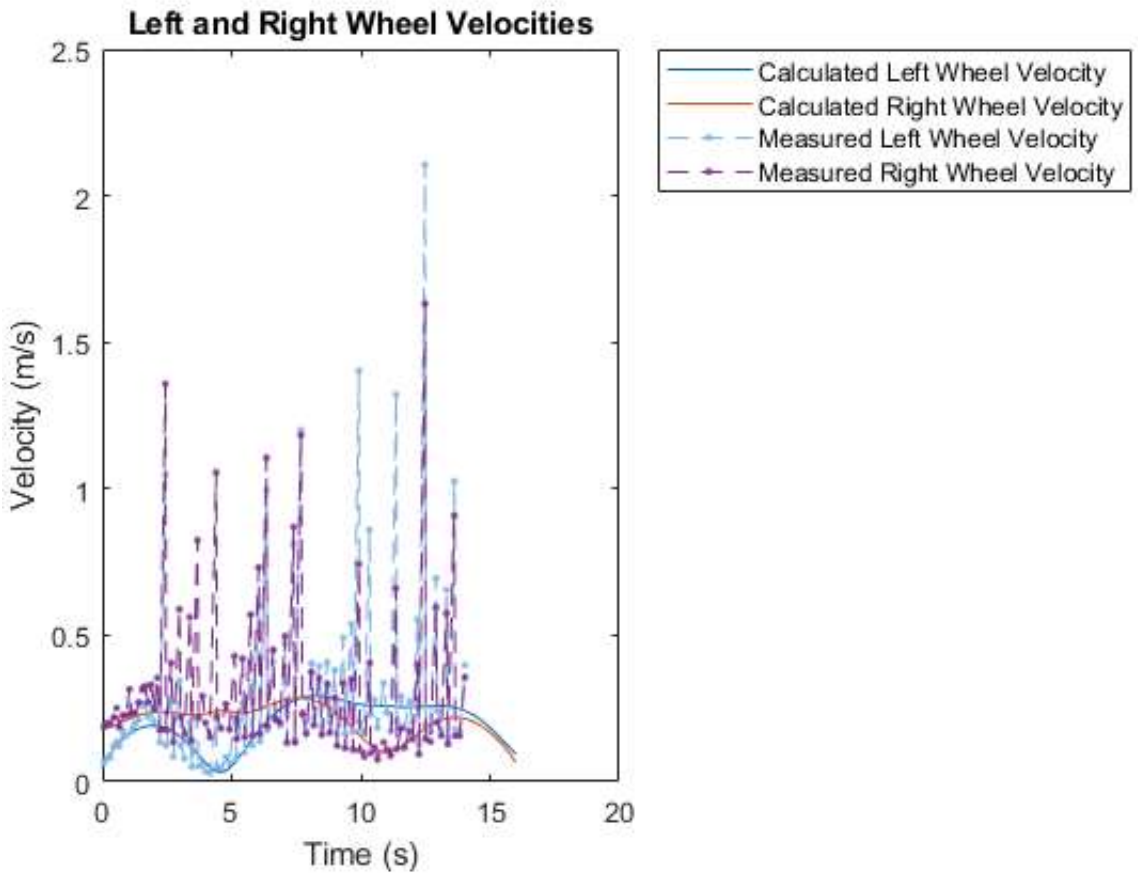
```
%plotting the predicted path and the tangent, normal, and binormal vectors
figure(1)
plot3(r(1,1:100),r(2,1:100),r(3,1:100)), hold on
quiver3(r(1,10),r(2,10),r(3,10),THat(1,10),THat(2,10),THat(3,10),'r')
quiver3(r(1,10),r(2,10),r(3,10),NHat(1,10),NHat(2,10),NHat(3,10),'b')
quiver3(r(1,10),r(2,10),r(3,10),BHat(1,10),BHat(2,10),BHat(3,10),'g')
quiver3(r(1,50),r(2,50),r(3,50),THat(1,50),THat(2,50),THat(3,50),'r')
quiver3(r(1,50),r(2,50),r(3,50),NHat(1,50),NHat(2,50),NHat(3,50),'b')
quiver3(r(1,50),r(2,50),r(3,50),BHat(1,50),BHat(2,50),BHat(3,50),'g')
quiver3(r(1,90),r(2,90),r(3,90),THat(1,90),THat(2,90),THat(3,90),'r')
quiver3(r(1,90),r(2,90),r(3,90),NHat(1,90),NHat(2,90),NHat(3,90),'b')
quiver3(r(1,90),r(2,90),r(3,90),BHat(1,90),BHat(2,90),BHat(3,90),'g')
title('Bridge of Doom')
```



Deliverable 2

```
%defining the distance between the wheels
d=0.254;
%defining the linear velocity
%this was solved in mathematica but is the derivative of the path
v=(0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+0.440496E-1.*sin(0.265E1.*( ...
  0.14E1+0.2E0.*u)).^2).^(1/2);

%defining the angular velocity
%this was solved in mathematica using the following equation
%omega = THat x dTHat/dt
omega=0.E-323+(-0.338511E-4).*cos(0.14E1+0.2E0.*u).^5.*cos(0.371E1+ ...
  0.53E0.*u).*(0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+0.440496E-1.*sin( ...
  0.371E1+0.53E0.*u).^2).^(-3)+(-0.12774E-4).*cos(0.14E1+0.2E0.*u) ...
  .^4.*sin(0.14E1+0.2E0.*u).*sin(0.371E1+0.53E0.*u).*(0.39204E-1.* ...
  cos(0.14E1+0.2E0.*u).^2+0.440496E-1.*sin(0.371E1+0.53E0.*u).^2).^( ...
  -3)+(-0.760702E-4).*cos(0.14E1+0.2E0.*u).^3.*cos(0.371E1+0.53E0.* ...
  u).*sin(0.371E1+0.53E0.*u).^2.*(0.39204E-1.*cos(0.14E1+0.2E0.*u) ...
  .^2+0.440496E-1.*sin(0.371E1+0.53E0.*u).^2).^(-3)+(-0.287057E-4).* ...
  cos(0.14E1+0.2E0.*u).^2.*sin(0.14E1+0.2E0.*u).*sin(0.371E1+ ...
  0.53E0.*u).^3.*(0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+0.440496E-1.* ...
  sin(0.371E1+0.53E0.*u).^2).^(-3)+(-0.427362E-4).*cos(0.14E1+ ...
  0.2E0.*u).*cos(0.371E1+0.53E0.*u).*sin(0.371E1+0.53E0.*u).^4.*( ...
  0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+0.440496E-1.*sin(0.371E1+ ...
  0.53E0.*u).^2).^(-3)+(-0.161269E-4).*sin(0.14E1+0.2E0.*u).*sin( ...
  0.371E1+0.53E0.*u).^5.*(0.39204E-1.*cos(0.14E1+0.2E0.*u).^2+ ...
  0.440496E-1.*sin(0.371E1+0.53E0.*u).^2).^(-3);
```

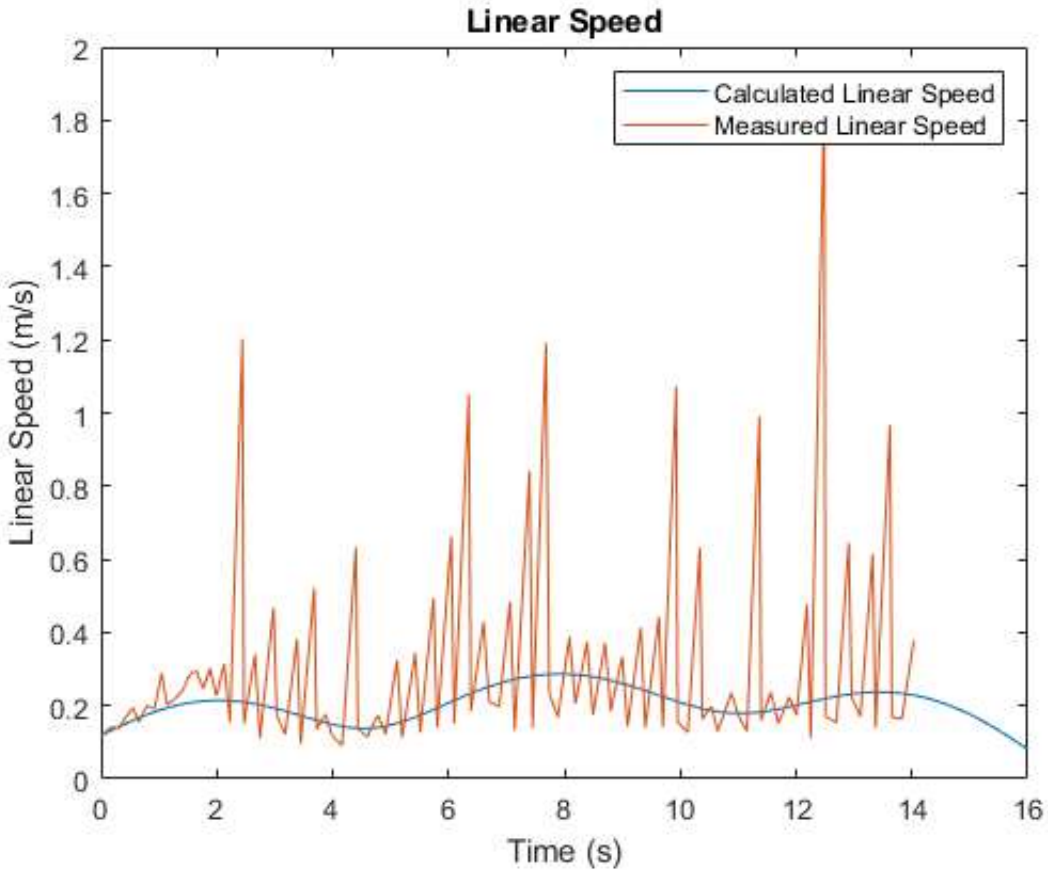
```
%solving for the left and right wheel velocities
vl=v-(omega*(d/2));
vr=v+(omega*(d/2));

%taking the step by step derivative of the discrete points
for l = 1:122
    expvl(1,l)=dataset(l,1);
    expvl(2,l)=(dataset(l+1,2)-dataset(l,2))/(dataset(l+1,1)-dataset(l,1));
    expvr(1,l)=dataset(l,1);
    expvr(2,l)=(dataset(l+1,3)-dataset(l,3))/(dataset(l+1,1)-dataset(l,1));
end

%plotting the predicted and measured left and right wheel velocities
figure(2)
plot(u,vl), hold on
plot(u,vr)
plot(expvl(1,1:100),expvl(2,1:100),'--.','Color',[0.5,0.7,0.9])
plot(expvr(1,1:100),expvr(2,1:100),'--.')
title('Left and Right Wheel Velocities')
xlabel('Time (s)')
ylabel('Velocity (m/s)')
legend('Calculated Left Wheel Velocity','Calculated Right Wheel Velocity','Measured Left W
heel Velocity','Measured Right Wheel Velocity','Location','northeastoutside')
```

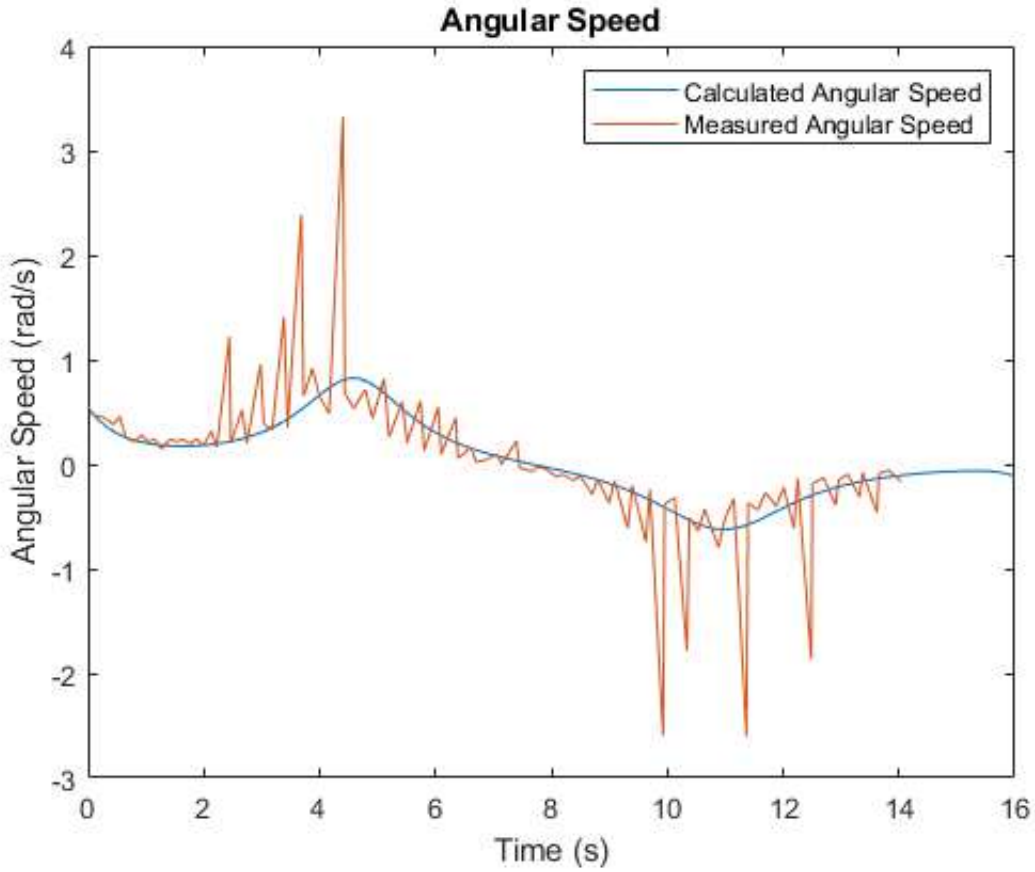


Deliverable 3

```
%calculating the linear speed of the NEATO
linvel=(vl+vr)/2;
```

```
%calculating the angular speed of the NEATO
angvel=(vr-vl)/d;

%calculating the "measured" linear speed using measured wheel velocities
explinvel = (expvl(2,:)+expvr(2,:))/2;

%calculating the "measured" angular speed using measured wheel velocities
expangvel = (expvr(2,:)-expvl(2,:))/d;

%plotting the calculated and measured linear speeds
figure(3)
plot(u,linvel),hold on
plot(expvl(1,1:100),explinvel(1,1:100))
title('Linear Speed')
xlabel('Time (s)')
ylabel('Linear Speed (m/s)')
legend('Calculated Linear Speed','Measured Linear Speed')

%plotting the calculated and measured angular speeds
figure(4)
plot(u,angvel),hold on
plot(expvr(1,1:100),expangvel(1,1:100))
title('Angular Speed')
xlabel('Time (s)')
ylabel('Angular Speed (rad/s)')
legend('Calculated Angular Speed','Measured Angular Speed')
```





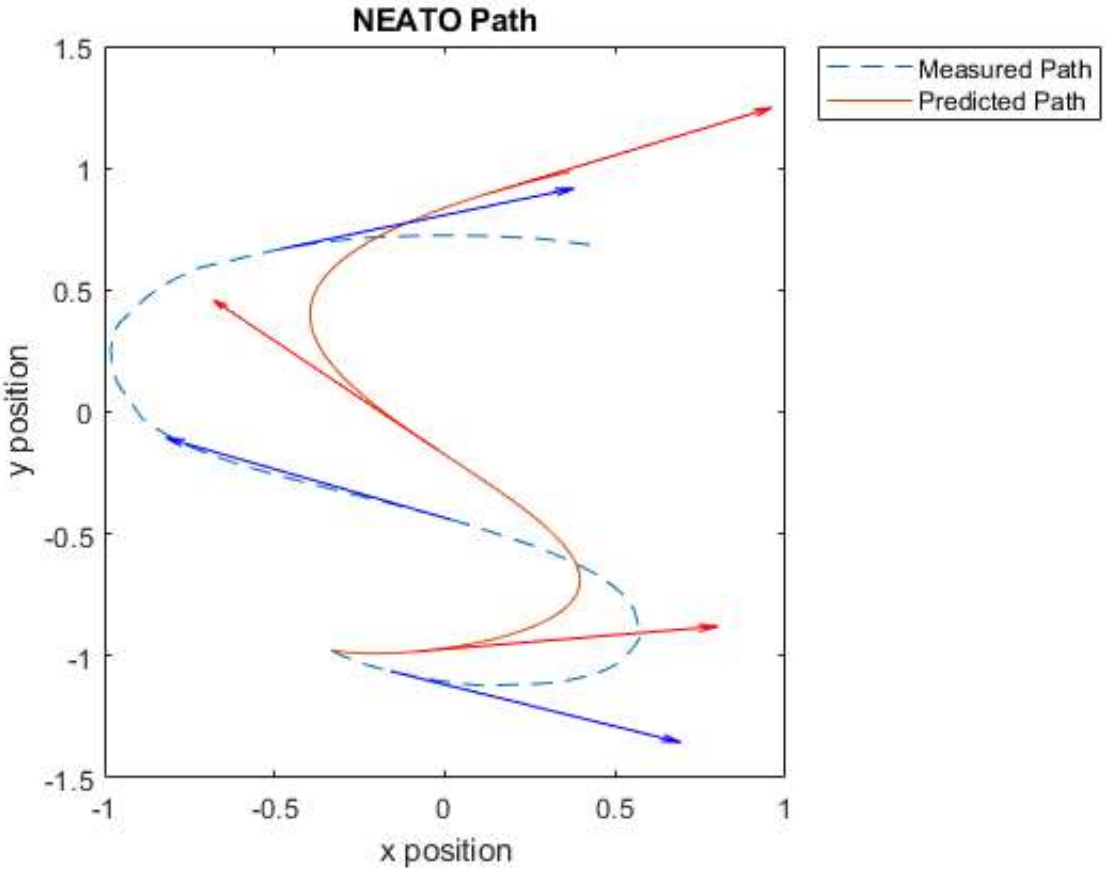
Deliverable 5

```
%N/A
```

Deliverable 6

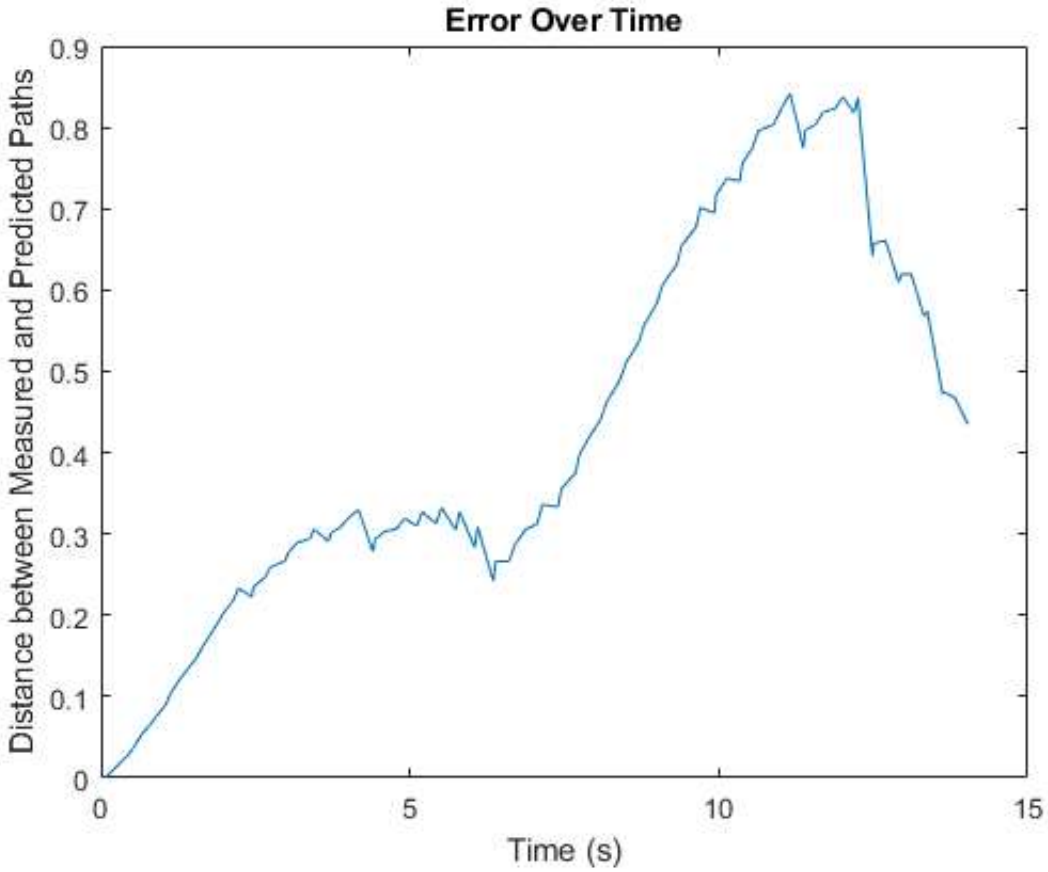
```
%integrating the experimental angular velocity to find theta
exptheta = cumtrapz(expvr(1,:),expangvel(1,:));

%using theta to vectorize the linear velocity of the NEATO
for y = 1:122
    vectv(y,1) = explinvel(1,y)*cos(exptheta(1,y));
    vectv(y,2) = explinvel(1,y)*sin(exptheta(1,y));
end

%numerically integrating the vectorized velocity to find coordinates
exppos(:,1) = cumtrapz(dataset(1:122,1),vectv(:,1));
exppos(:,2) = cumtrapz(dataset(1:122,1),vectv(:,2));

%taking the transpose of the position matrix to make dimensions work
exppostrans = transpose(exppos);

%the initial heading is versatile so changing this so it visually
%fits the initial heading of the expected path
deg = 40;
rot = [cosd(deg) sind(deg);-sind(deg) cosd(deg)];
expposrot = rot*exppostrans;
```

```
%shifting the starting point to match that of the expected path
shiftpos(1,:) = expposrot(1,:)-0.3337;
shiftpos(2,:) = expposrot(2,:)-0.9756;


figure(5)
%plotting the measured path
plot(shiftpos(1,:),shiftpos(2,:),'--'),hold on

%plotting the predicted path
plot(r(1,1:100),r(2,1:100))

%creating a matrix of not normalized tangent vectors
%rotating the not normalized tangent vectors the same amount as the path
quivvectv=rot*[vectv(10,1) vectv(50,1) vectv(90,1);vectv(10,2) vectv(50,2) vectv(90,2)];

%plotting THats for u = 10 and normalizing if needed
quiver(r(1,10),r(2,10),THat(1,10),THat(2,10),'r')
quiver(shiftpos(1,10),shiftpos(2,10),quivvectv(1,1)/norm([0.1911 -0.0655]),quivvectv(2,1)/
norm([0.1911 -0.0655]),'b')

%plotting THats for u = 50 and normalizing if needed
quiver(r(1,50),r(2,50),THat(1,50),THat(2,50),'r')
quiver(shiftpos(1,50),shiftpos(2,50),quivvectv(1,2)/norm([-0.1959 0.0784]),quivvectv(2,2)/
norm([-0.1959 0.0784]),'b')

%plotting THats for u = 90 and normalizing if needed
quiver(r(1,90),r(2,90),THat(1,90),THat(2,90),'r')
quiver(shiftpos(1,90),shiftpos(2,90),quivvectv(1,3)/norm([0.1634 0.0471]),quivvectv(2,3)/n
orm([0.1634 0.0471]),'b')
title('NEATO Path')
xlabel('x position')
ylabel('y position')
legend('Measured Path','Predicted Path','Location','northeastoutside')
```



Deliverable 7

```
%calculating the distance between the two paths
distance=abs(shiftpos(:,1:100)-r(1:2,:));

%finding the magnitude of each of these differences
dist=transpose(sqrt(distance(1,:).^2+distance(2,:).^2));

%plotting the distance between the two paths over time
figure(6)
plot(dataset(1:100,1),dist(:,1))
title('Error Over Time')
xlabel('Time (s)')
ylabel('Distance between Measured and Predicted Paths')
```



Deliverable 4

```
pub = rospublisher('/raw_vel');
msg = rosmessage(pub);

%0.2 is my alpha
%3.2/alpha gives a time of 16 seconds
ticend=3.2/0.2;
tic

%initializing a loop counter
count=1;

while count<101
   %while loop stops when 16 seconds happens (based on alpha)
   while toc<count*0.16
    %picking values from the precalculated matrix of velocities
    msg.Data=[vl(1,count),vr(1,count)];
    send(pub,msg);

    %this should be 0.16 but the NEATO arbitrarily worked better with 0.2
    %wait a certain amount of time until the next calculated vr and vl
    pause(0.20); %randomly changing this made it stay on the bridge better
   end
   %counting how many times through the loop
   %there are only 100 vr and vl values so stops loop then
   count=count+1;
end
msg.Data=[0,0];
```

```
send(pub,msg);
```

```
Error using BridgeofDoom (line 223)
Cannot determine the message type of the /raw_vel topic. Either the topic is not registere
d with the ROS master, or the ROS master is not reachable.
```

Published with MATLAB® R2018a

https://youtu.be/sGa4hseD9pU