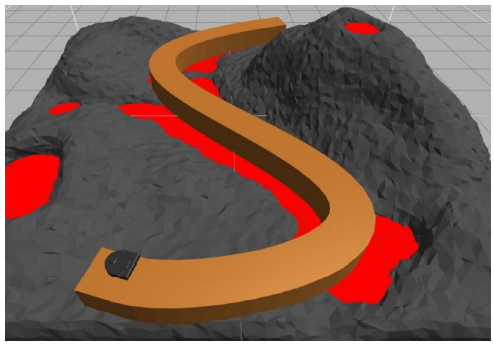
Tolulope Oshinowo **Quantitative Engineering Analysis**
Bridge Of Doom

INTRODUCTION:
During the third session of QEA I we learned about robots; both in terms of understanding the surface level philosophy and the basics of controlling a robot. In learning about controlling a robot we learned about the sensory-motor loop, and how it defines the basic mechanics of any type of robot. During the past few weeks we have delved into controlling Neato robots with commands written in MATLAB sent using the ROS Toolbox.

In this project we were tasked with using the knowledge that we have accumulated from multivariable calculus and the ROS experience from previous assignments to successfully drive the Neato robot across an 'S'-shaped bridge. The bridge was defined by the parametric equation below:

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

In order for the Neato to cross the bridge successfully the appropriate wheel velocities will need to be passed to the respective motors on the Neato. These velocities will be derived from the same equation that defines the bridge.



METHODOLOGY:
In order to successfully traverse the bridge the necessary wheel velocities must be found. Here was my methodology implemented in MATLAB.

First I initialized the symbolic variables that would be used in my calculations.

```
syms R u r t;
```

Then I defined some other constants that would be used in my calculations.

```
d=.235; %wheelbase distance
beta=.334; %multiplying constant
steps=200; %number of parametric steps
```

Because the Neato can only put out a maximum linear speed of 2m/s, beta would be necessary as a step multiplier to keep things within range.

```
u=beta*t; %define u in terms of t (time)
```

Next I recreated the parametric equation in MATLAB (R will later be defined as 4).

```
ri=R*0.3960*cos(2.65*(u + 1.4));
rj=-R*0.99*sin(u + 1.4);
rk=0*u;
r=[ri,rj,rk];
```

After that I set up assumptions for the symbolic variables so calculations would be done correctly. From here on out I iterate across the curve with 't', since 'u' is a multiple of 't'.

```
assume(R,{'real','positive'});
assume(t,{'real','positive'});
```

Now I defined the unit tangent vectors along the parametric curve.

```
dr=diff(r,t);
T_hat_ugly=dr./norm(dr);
T_hat=simplify(T_hat_ugly);
```

And then I defined the unit normal vectors along the parametric curve (most importantly dT_hat).

```
dT_hat=diff(T_hat,t);
N_hat=dT_hat/norm(dT_hat);
N_hat=simplify(N_hat);
```

With the necessary unit vectors along the curve defined, I defined the omegas (angular velocities) and LS's (linear speeds) at each point along the curve.

```
Omega=cross(T_hat,dT_hat); %define the angular velocity
LS = norm(dr); %define the linear speed
```

Then I defined vectors corresponding to the symbolic variables.

```
R_num = 4; %define a specific radius
t_num = linspace(0,3.2/beta,steps); %define a set of evenly spaced
points
```

I then used a for loop to fill out vectors corresponding to the other variables that I would need to find the wheel velocities at each point of the equation.

```
for n=1:length(t_num)
      r_num(n,:)=double(subs(r,[R, t],[R_num, t_num(n)]));
      %calculate instantaneous position
      T_hat_num(n,:)=double(subs(T_hat,[R, t],[R_num, t_num(n)]));
      %calculate instantaneous unit tangent vector
      N_hat_num(n,:)=double(subs(N_hat,[R, t],[R_num, t_num(n)]));
      %calculate instantaneous unit normal vector
      Omega_num(n,:)=double(subs(Omega,[R, t],[R_num, t_num(n)]));
      %calculate instantaneous angular velocity
      LS_num(n,:)=double(subs(LS,[R, t],[R_num, t_num(n)]));
      %calculate instantaneous linear speed
end
```

Finally I was able to obtain the theoretical velocities for each wheel of the Neato.

```
vL_T=LS_num-Omega_num(:,3).*(.235/2); %theoretical left wheel velocity
vR_T=LS_num+Omega_num(:,3).*(.235/2); %theoretical right wheel velocity
```

Using these last two vectors I was able to create a function that passed in the index of each wheel velocity to the Neato, and successfully drove across the Bridge of Doom. In the following section I will describe how I plotted each of the necessary plots.

THE PLOTTING:

First I loaded the collected data.

```
load BODtrial.mat % load collected data
```

For the first post-plot I needed to find the experimental left & right wheel velocities with encoder data.

```
%setting up variables for left and right vel
s=dataset(:,1),s=s(55:end,:); %s is used as a replacement for t to not
overwrite the previous sym with the name 't'
rL_E=dataset(:,2),rL_E=rL_E(55:end,:);
rR_E=dataset(:,3),rR_E=rR_E(55:end,:);
dt=diff(s);
dL_E=diff(rL_E);
dR_E=diff(rR_E);
vL_E=dL_E./dt;
vR_E=dR_E./dt;
```

Then I plotted the chart.

```
plot(vL_T(2:2:end,:),'-'),hold on; %(2:2:end,:) due to doubling of data
at each step from linspace
plot(vR_T(2:2:end,:),'-'),hold on;
plot(vL_E,'--'),hold on;
plot(vR_E,'--');
xlabel('Index (Normalized Timesteps)');
ylabel('Velocity (m/s)');
legend('Theo. Left Vel','Theo. Right Vel','Exp. Left Vel','Exp. Right
Vel');
title('Theoretical & Experimental Left and Right Wheel Velocities');
hold off;
```

For the second post-plot I needed to find linear speeds and angular velocities.

```
%setting up variables for linear speed and angular velocity
v_E=(vL_E+vR_E)./2;
v_T=(vL_T+vR_T)./2;
omega_E=(vR_E-vL_E)./d;
omega_T=(vR_T-vL_T)./d;
```

Then I plotted the chart.

```
yyaxis left;
plot(s(1:end-6),v_T(2:2:end,:),'-'),hold on;
plot(s(1:end-1),v_E,'--'),hold on;
ylim([-2,2]); %limits of wheel velocity
```

```
xlabel('Time');
ylabel('Linear Speed (m/s)');
yyaxis right;
plot(s(1:end-6),'-'),omega_T(2:2:end,:),hold on;
plot(s(1:end-1),'--'),omega_E,hold on;
ylabel('Angular Velocity (rad/s)');
legend('Theo. Lin Speed','Exp. Lin Speed','Theo. Ang Vel','Exp. Ang
Vel');
title('Theoretical & Experimental Linear Speed and Angular Velocity');
hold off;
```

For the final post-plot I needed to find the instantaneous positions with the lin speeds & angular vels.

```
r_E=zeros(length(s),2);
theta_E=zeros(length(s),1);
for n=1:length(dt)
r_E(n+1,1)=r_E(n,1)+v_E(n)*cos(theta_E(n))*dt(n);
r_E(n+1,2)=r_E(n,2)+v_E(n)*sin(theta_E(n))*dt(n);
theta_E(n+1)=theta_E(n)+omega_E(n)*dt(n);
end
```

(the theoretical vectors are 200x1, so I accounted for twice as many points with half the step size)

```
r_T=zeros(200,2);
theta_T=zeros(200,1);
for n=1:199
r_T(n+1,1)=r_T(n,1)+v_T(n)*cos(theta_T(n))*.05;
r_T(n+1,2)=r_T(n,2)+v_T(n)*sin(theta_T(n))*.05;
theta_T(n+1)=theta_T(n)+omega_T(n)*.05;
end
```

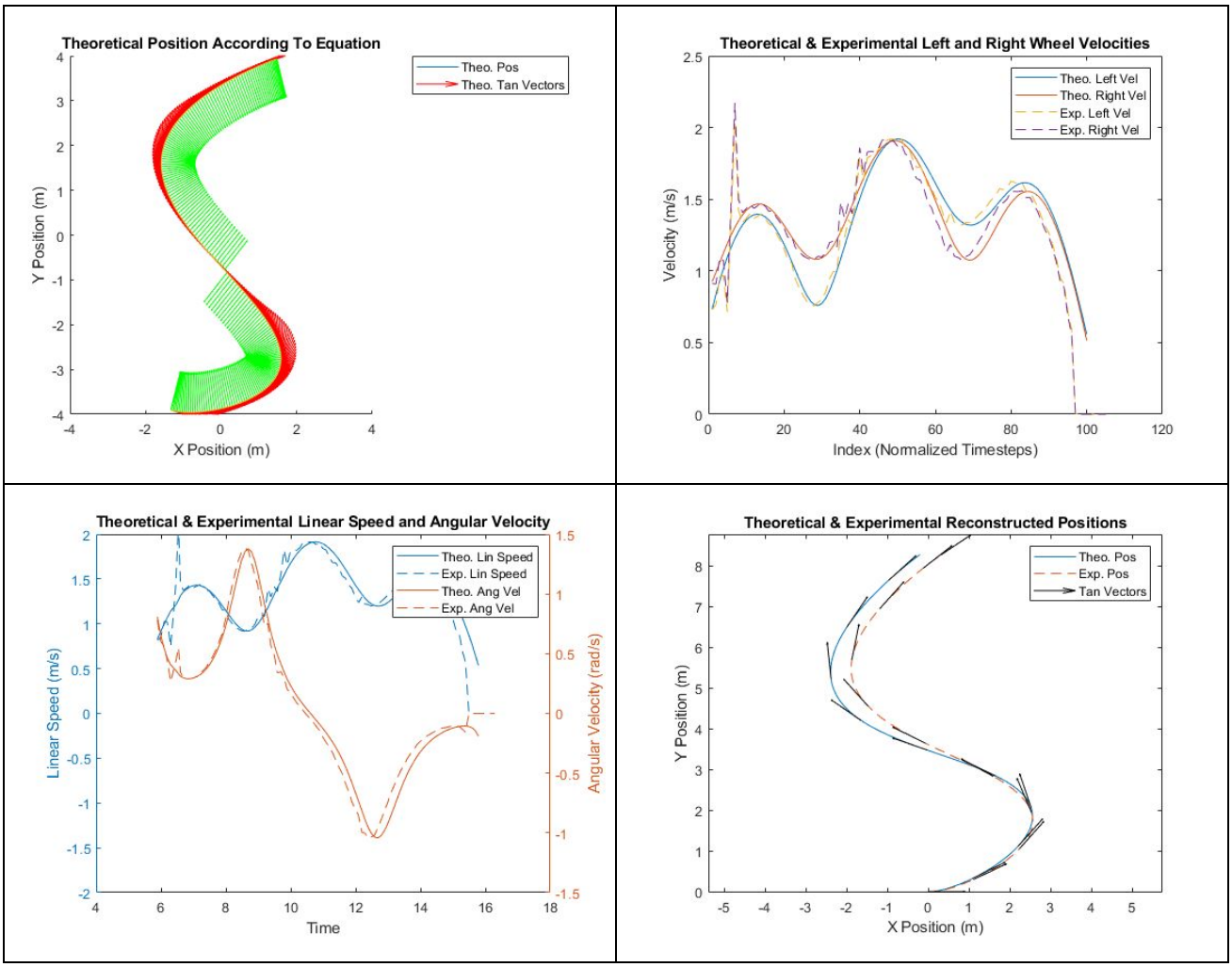
Then I plotted the chart.

```
plot(r_T(:,1),r_T(:,2),'-'),hold on;
plot(r_E(:,1),r_E(:,2),'--'),hold on;
for k=1:20:length(r_T)
    % take theoretical xy coordinate and xy vector components

quiver(r_T(k,1),r_T(k,2),cos(theta_T(k)),sin(theta_T(k)),'color','k');
end
for k=1:10:length(r_E)
    % take experimental xy coordinate and xy vector components

quiver(r_E(k,1),r_E(k,2),cos(theta_E(k)),sin(theta_E(k)),'color','k');
end
axis equal;
legend('Theo. Pos','Exp. Pos','Tan Vectors');
xlabel('X Position (m)');
ylabel('Y Position (m)');
title('Theoretical & Experimental Reconstructed Positions');
hold off;
```

THE PLOTS:



YOUTUBE LINK:
https://youtu.be/CVbBhb7Jsbk