Bridge of Doom$^{\text{dun}^{\text{dun}^{\text{dun}}}}$

By Ari Porad and Jacob Smilg

For Quantitative Engineering Analysis 2, April 6th 2021

Video

A video of our Neato using the code below to successfully traverse the Bridge of Doom$^{\text{dun}^{\text{dun}^{\text{dun}}}}$ is available on YouTube: https://youtu.be/6AWX_qQ3ueE

The Challenge

Program the simulated neato robot to autonomously cross the Bridge of Doom$^{\text{dun}^{\text{dun}^{\text{dun}}}}$, as defined by the curve:

$$r(u) = \lceil 0.3960\cos(2.65(u+1.4))\hat{\imath} - 0.99\sin(u+1.4$$

Robot Setup

Before we can graph the robot's behavior, we need to define some physical parameters:

```
% alpha is a linear scalar with tim
% It's been approximately calibrate
% below the limit of 2 m/s
alpha = 1/5;

% d is the wheelbase of the robot
d = 0.235; % m

% The range of u is specified by th
% dun dun)
U_RANGE = [0 3.2];

% We also convert the range of u in
% correlated) for convinence
T_RANGE = U_RANGE ./ alpha;

% N defines the number of points th
% Higher Ns result in higher resolu
```

```
% fairly minor.
N = 100;

% Define ranges of ts and us to use
us = linspace(U_RANGE(1), U_RANGE(2
ts = us ./ alpha;

% This loads the above equations ag
% file, which we use to ensure that
% drive the neato in the simulator 
% with the programming principle of
%
% We've commented this out for our 
% script by itself without it crash

% equations
```

Methodology

We begin by setting up the symbolic variables we'll use to process the $\text{Bridge of Doom}^{\text{dun}^{\text{dun}^{\text{dun}}}}$:

All code in this section should be in a file called `equations.m`*, which is referenced by both this live script and the program that actually drives the robot across the* $\text{Bridge of Doom}^{\text{dun}^{\text{dun}^{\text{dun}}}}$. *We use this pattern to ensure that both scripts use exactly the same equations, in line with the programming principle of DRY (Don't Repeat Yourself).*

```
% First we must define the symbolic
syms t omega v_l v_r

% Assumptions help MATLAB not go in
```

```
assume(t, {'real', 'positive'})
assume(omega, 'real')
assume(v_l, 'real')
assume(v_r, 'real')

% This isn't actually a symbolic va
% This allows us to modulate the sp
% adjusting alpha
u = t .* alpha;

% Define the curve of the Bridge of
ri=4 * (0.3960 * cos(2.65 * (u + 1.
rj=4 * (-0.99 * sin(u + 1.4));
rk=0*u;
r=[ri,rj,rk]
```

r =

$$\left(\frac{198 \cos\left(\frac{53\,t}{100} + \frac{371}{100}\right)}{125} \quad -\frac{99 \sin\left(\frac{t}{5} + \frac{7}{5}\right)}{25} \quad 0 \right)$$

```
% Calculate the robot's velocity (r
% speed (magnitude of velocity)
v=diff(r,t)
```

v =

$$\left(-\frac{5247\sin\left(\frac{53\,t}{100}+\frac{371}{100}\right)}{6250}\quad -\frac{99\cos\left(\frac{t}{5}+\frac{7}{5}\right)}{125}\quad 0\right.$$

```
speed = norm(v);

% Calculate the unit tangent and un
T_hat=simplify(v./norm(v))
```

T_hat =

$$\left(-\frac{53\sin\left(\frac{53\,t}{100}+\frac{371}{100}\right)}{\sigma 1}\quad -\frac{50\cos\left(\frac{t}{5}+\frac{7}{5}\right)}{\sigma 1}\quad 0\right)$$

where

$$\sigma 1=\sqrt{2500\cos\left(\frac{t}{5}+\frac{7}{5}\right)^2+2809\sin\left(\frac{53\,t}{100}\right.}$$

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

N_hat =

$$\left(-\frac{2\ (96725\,\sigma 6 + 140450\,\sigma 5 + 43725\,\sigma 4)}{\sigma 1}\ \frac{1}{}\right.$$

where

$$\sigma 1 = 53\ \sqrt{2500\ (73\,\sigma 6 + 106\,\sigma 5 + 33\,\sigma 4)}$$

$$\sigma 2 = \sin\left(\frac{63\,t}{50} + \frac{441}{50}\right)$$

$$\sigma 3 = \sin\left(\frac{43\,t}{50} + \frac{301}{50}\right)$$

$$\sigma 4 = \cos\left(\frac{93\,t}{100} + \frac{651}{100}\right)$$

$$\sigma 5 = \cos\left(\frac{53\,t}{100} + \frac{371}{100}\right)$$

$$\sigma 6 = \cos\left(\frac{13\,t}{100} + \frac{91}{100}\right)$$

```
% Calculate the rotational velocity
omega = simplify(cross(T_hat, dT_hat
```

```
omega =
```

$$\left(0\ \ 0\ -\frac{9672500\cos\left(\frac{7\,t}{100}+\frac{49}{100}\right)+4054060}{}\right.$$

```
rotation_speed = omega(3);

% Calculate the wheel velocities
v_l = simplify(speed - (rotation_spe
```

```
v_l =
```

$$\frac{47\ \left(9672500\cos\left(\frac{7\,t}{100}+\frac{49}{100}\right)+40540601\right.}{}$$

```
v_r = simplify(speed + (rotation_spe
```

```
v_r =
```

$$\frac{99\sqrt{2500\cos\left(\frac{t}{5}+\frac{7}{5}\right)^2+2809\sin\left(\frac{53\,t}{100}+\frac{3}{1}\right.}}{6250}$$

Encoders and Odometry

We load in the saved data from when we ran our code in the simulator, to plot the measured values side-by-side with the calculated ones here.

```
% We also load our saved encoder da
% simulator
load("encoder_data.mat");

%% The stored encoder data gives us
%% need the relative changes in eac
enc_dt = diff(enc_t);
enc_dl = diff(enc_l) ./ enc_dt;
enc_dr = diff(enc_r) ./ enc_dt;
```

```
%% We also need to derive the wheel
%% measured encoder values.

% Define our symbolic variables
syms vl vr vlin vrot;
assume(vl, 'real');
assume(vr, 'real');
assume(vlin, 'real');
assume(vrot, 'real');

% Define our equations
vl_eqn = vl == vlin - (vrot .* (d ./
vr_eqn = vr == vlin + (vrot .* (d ./

% Have MATLAB solve them
[vrot, vlin] = solve([vl_eqn, vr_eq

% Substitute in our data and calcul
vl = enc_dl;
vr = enc_dr;
vrots = double(subs(vrot));
```

```
vlins = double(subs(vlin));

%% Finally, we need to calculate the
%% encoder values

% Define our variables
enc_thetas = zeros(length(enc_t), 1
enc_rxs = zeros(length(enc_t), 1);
enc_rys = zeros(length(enc_t), 1);

% We need to calculate our initial 
% definition--cannot tell you your 
% calculated path. That's OK becaus
% actual program is to use the magi
% the starting position perscribed 
that_0 = double(subs(T_hat, T_RANGE
enc_thetas(1) = atan(that_0(2)/that
enc_rxs(1) = calculated_path(1, 1);
enc_rys(1) = calculated_path(1, 2);

% Then iteratively calculate each s
```

```
for i=1:length(enc_dt)
    enc_thetas(i + 1) = enc_thetas(
    enc_rxs(i + 1) = enc_rxs(i) + (
    enc_rys(i + 1) = enc_rys(i) + (
end

% Finally, calculate the Thats and 
% We could technically optimize thi
% inside the loop, but Keep-It-Simp
% root of all evil"/etc.
enc_Thats = [cos(enc_thetas), sin(e
enc_Nhats = [-cos(enc_thetas), -sin
```

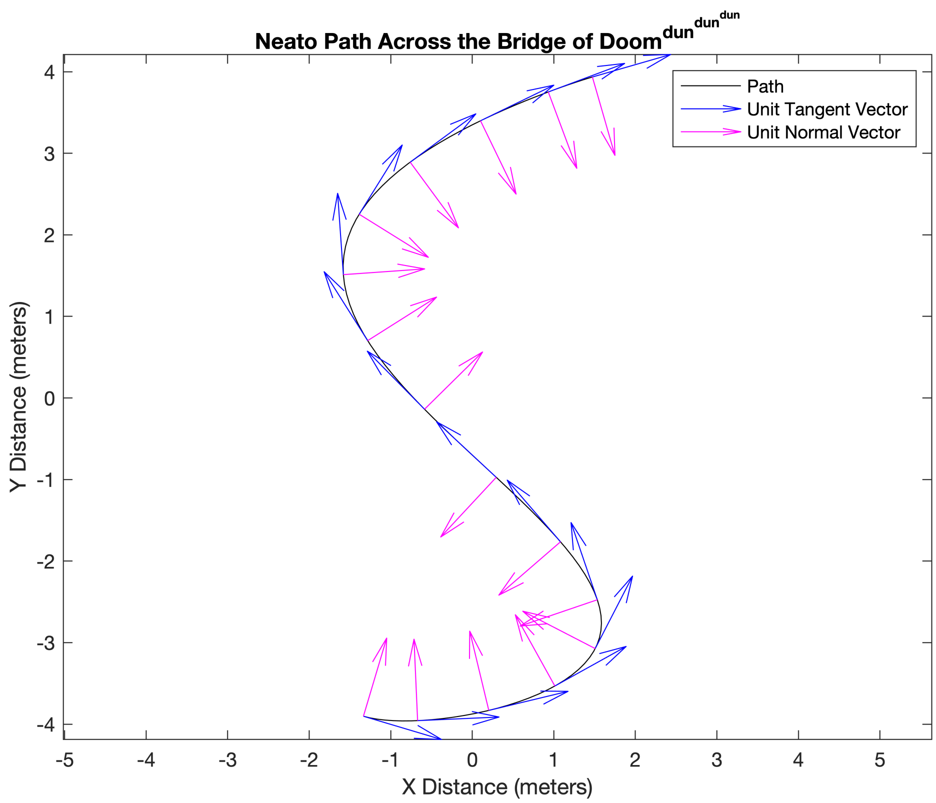
Neato Path Across the Bridge of Doom$^{\text{dun}^{\text{dun}^{\text{dun}}}}$ (Exercise 21.1)

Bridge of Doom$^{\text{dun}^{\text{dun}^{\text{dun}}}}$ *with the Neato on it*

```
figure(1); clf;

% Calculate and plot the Neato's pa
calculated_path = double(subs(r, t,
```

```
plot(calculated_path(:, 1), calcula

% Calculate and plot unit tangent v

% First, pick the points at which t
ts_for_arrows = linspace(T_RANGE(1)
start_points_for_calc_arrows = doub

% Then, calculate vector lengths
calc_that_end_points = double(subs(
calc_nhat_end_points = double(subs(

% Finally, plot the vectors
quiver( ...
    start_points_for_calc_arrows(:,
    calc_that_end_points(:, 1), cal
quiver( ...
    start_points_for_calc_arrows(:,
    calc_nhat_end_points(:, 1), cal

% Don't forget to label your graphs
```

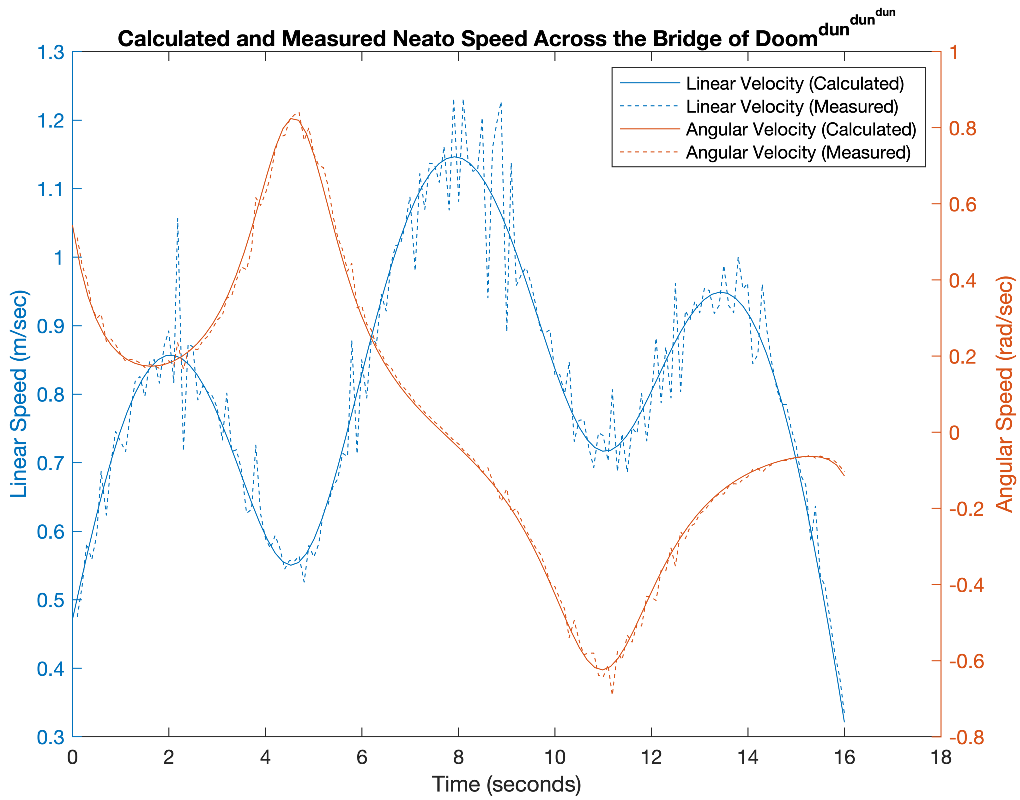
```
legend("Path", "Unit Tangent Vector
xlabel("X Distance (meters)");
ylabel("Y Distance (meters)");
title("Neato Path Across the Bridge
```



Neato Speed Across the Bridge of Doom$^{\text{dun}^{\text{dun}^{\text{dun}}}}$ (Exercise 21.2)

```
% Calculate the linear and rotation
speeds = double(subs(speed, t, ts))
rot_speeds = double(subs(rotation_s
```

```
figure(2); clf;

% Plot the calculated and measured
yyaxis left;
plot(ts, speeds, '-'); hold on;
plot(enc_t(2:end), vlins, '--'); ho
ylabel("Linear Speed (m/sec)");

% Plot the calculated and measured
yyaxis right;
plot(ts, rot_speeds, '-'); hold on;
plot(enc_t(2:end), vrots, '--'); ho

% Don't forget to label your graphs
xlabel("Time (seconds)");
ylabel("Angular Speed (rad/sec)");
title("Calculated and Measured Neat
legend( ...
```

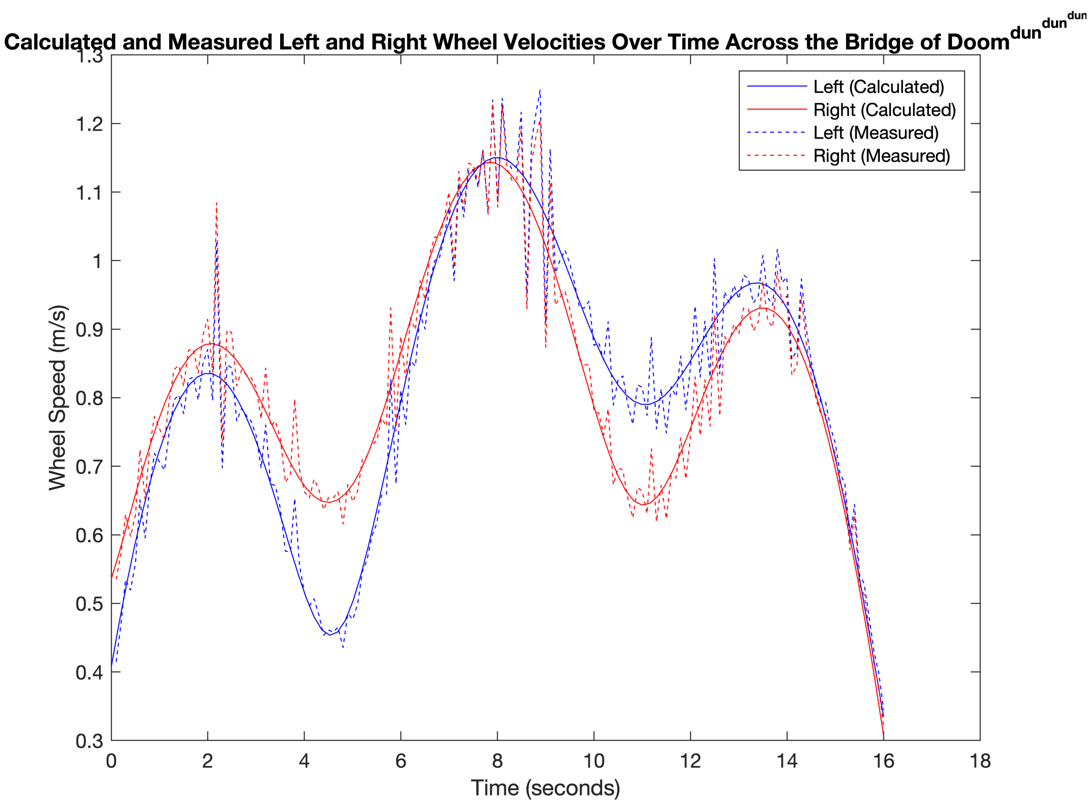



```
    "Linear Velocity (Calculated)",
    "Angular Velocity (Calculated)"
```

Neato Wheel Velocities Across the Bridge of Doom$^{\text{dun}^{\text{dun}^{\text{dun}}}}$ (Exercise 21.4)

```
% Calculate the left and right whee
v_ls = double(subs(v_l, t, ts));
v_rs = double(subs(v_r, t, ts));

% Plot them
figure(3); clf;
```

```
plot(ts, v_ls, 'b-'); hold on;
plot(ts, v_rs, 'r-');
plot(enc_t(2:end), enc_dl, 'b--');
plot(enc_t(2:end), enc_dr, 'r--');

% Don't forget to label your graphs
title("Calculated and Measured Left
xlabel("Time (seconds)");
ylabel("Wheel Speed (m/s)");
legend("Left (Calculated)", "Right
```



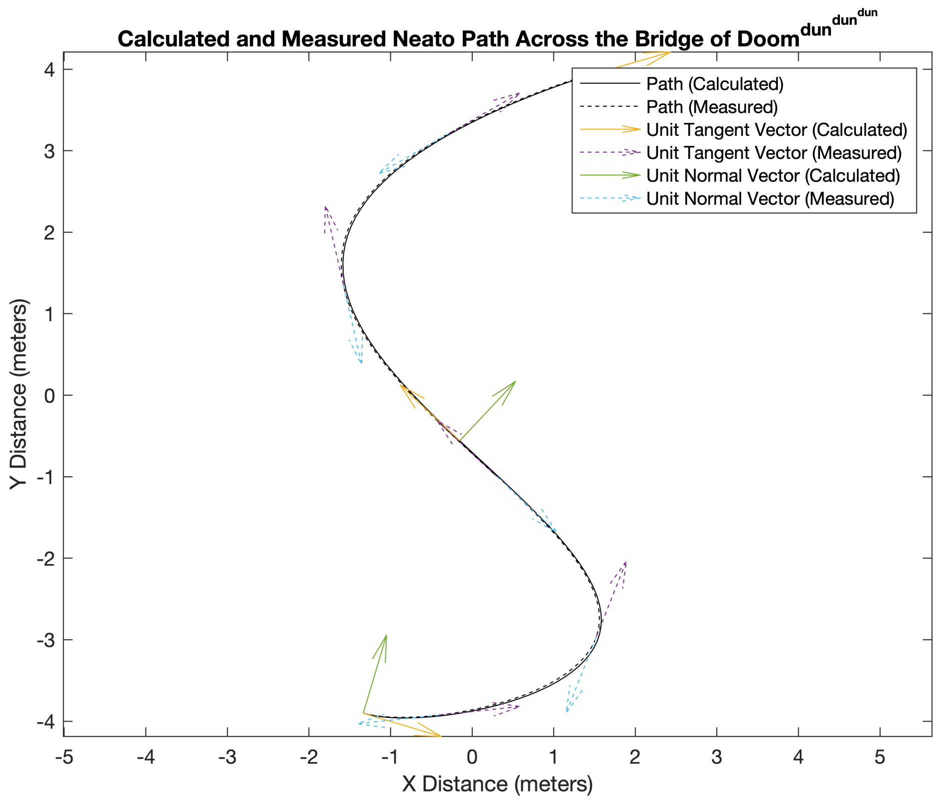
Calculated and Measured Neato Wheel Velocities Across the Bridge of Doom$^{\text{dun}^{\text{dun}^{\text{dun}}}}$ (Exercise 21.5)

Now that we've done our analysis, we can finally plot the calculated path against the plot our Neato actually took, along with the relevant unit tangent vectors.

```
figure(4); clf;

% Plot the paths themselves
plot(calculated_path(:, 1), calcula
plot(enc_rxs, enc_rys, 'k--');

%% Plot the unit tangent vectors

% This defines how many unit tangen
% this many samples. It will offset
% tangent vectors so they're both v
vector_spacing = 30;
```

```
% Calculate the predicted values fo
ts_for_arrows = linspace(T_RANGE(1)
start_points_for_calc_arrows = doub
calc_that_end_points = double(subs(
calc_nhat_end_points = double(subs(

% Plot the unit tangent vectors
quiver( ...
    start_points_for_calc_arrows(:,
    calc_that_end_points(:, 1), cal
quiver( ...
    enc_rxs(round(vector_spacing /
    enc_rys(round(vector_spacing /
    enc_Thats(round(vector_spacing
    enc_Thats(round(vector_spacing

% Plot the unit normal vectors
quiver( ...
    start_points_for_calc_arrows(:,
    start_points_for_calc_arrows(:,
```

```
    calc_nhat_end_points(:, 1), ...
    calc_nhat_end_points(:, 2), "of
quiver( ...
    enc_rxs(round(vector_spacing / 
    enc_rys(round(vector_spacing / 
    enc_Nhats(round(vector_spacing 
    enc_Nhats(round(vector_spacing 

% Don't forget to label your graphs
legend( ...
    "Path (Calculated)", "Path (Mea
    "Unit Tangent Vector (Calculate
    "Unit Normal Vector (Calculated
xlabel("X Distance (meters)");
ylabel("Y Distance (meters)");
title("Calculated and Measured Neat
```



Robot Code for Driving Across the Bridge of Doom$^{\text{dun}^{\text{dun}^{\text{dun}}}}$ (Exercise 21.4)

This code should be in a file called `drive_bod.m`*, and invoked from within the simulator. We've included it here for convinence*

```
function drive_bod()
    %% Configuration
    % alpha is a linear scalar with
```

```
    % It's been approximately calib
    % below the limit of 2 m/s
    alpha = 1/5;

    % d is the wheelbase of the rob
    d = 0.235; % m

    % The range of u is specified b
    % dun dun)
    U_RANGE = [0 3.2];

    % We also convert the range of
    % correlated) for convinence
    T_RANGE = U_RANGE ./ alpha;

    % This loads the equations from
    % use to ensure that this scrip
    % same equations, in line with t
    % Repeat Yourself).
    equations;
```

```
    % This is the format of the mes
    % into a vector here and conver
    % matlab function, it becomes m
    % Neato, which increases accura
    msgData = [v_l, v_r];

    % generateMessageData(t) now re
    % without using the Symbolic To
    generateMessageData = matlabFun

    %% Connect to the Neato
    disp("Connecting to Neato...")
    pub = rospublisher('raw_vel');

    %% Setup the Neato
    disp("Stopping Neato...")

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

```
    disp("Resetting Neato...")

    bridgeStart = double(subs(r,t,0
    startingThat = double(subs(T_ha
    placeNeato(bridgeStart(1),  bri

    % HACK: Wait a bit for robot to
    pause(2);

    %% Drive
    disp("Starting to Drive...")

    % Keep track of when we started
    rostic;

    while 1 % We'll break out of the
        % Get the current time
        cur_t = rostoc;

        % If we've hit the end, sto
```

```
        if cur_t >= T_RANGE(2)
            stopMsg = rosmessage(pu
            stopMsg.Data = [0 0];
            send(pub, stopMsg);
            break
        end

        % Otherwise, calculate the
        % go at that speed
        speedMsg = rosmessage(pub);
        speedMsg.Data = generateMes
        send(pub, speedMsg);
    end

    disp("Done!")
end
```