Bridge of Doom

Lilo Heinrich

April 2020

1 Introduction

The Bridge of Doom is a simulated bridge whose centerline is defined by a parametric equation given below. The challenge is to make a neato robot autonomously traverse this bridge with a maximum wheel speed of 2 m/s.

$$R(u) = 4 * [0.396\cos(2.65(u + 1.4))\hat{\imath} - .99\sin(u + 1.4)\hat{\jmath}].(u \in [0, 3.2])$$

2 Calculations Program

The first step is to calculate the linear and angular velocity components. To do this, I added the speed coefficient β, reformatting the path equation to be a variable of t, representative of time. I calculated the linear and angular velocity vectors, and from that the left and right velocity using the following equations.

The parametric path:

$$R(t) = 4 * [0.396\cos(2.65(\beta t + 1.4))\hat{\imath} - .99\sin(\beta t + 1.4)\hat{\jmath}].(t \in [0, 3.2/\beta])$$

The linear velocity vector:

$$V = |dR|$$

The unit tangent vector and from that the angular velocity vector:

$$\hat{T} = R'/|dR|$$

$$\omega = |\hat{T} \times d\hat{T}|$$

The right and left wheel velocities (with wheelbase width d in meters):

$$vR = V + \omega d/2$$

$$vL = V - \omega d/2$$

To visualize the problem, I plotted the parametric curve that defines the centerline of the bridge and added the linear velocity vector tangent to the curve and the angular velocity vector in the z-axis direction, animating them to plot over time. Here is a snapshot of it running:

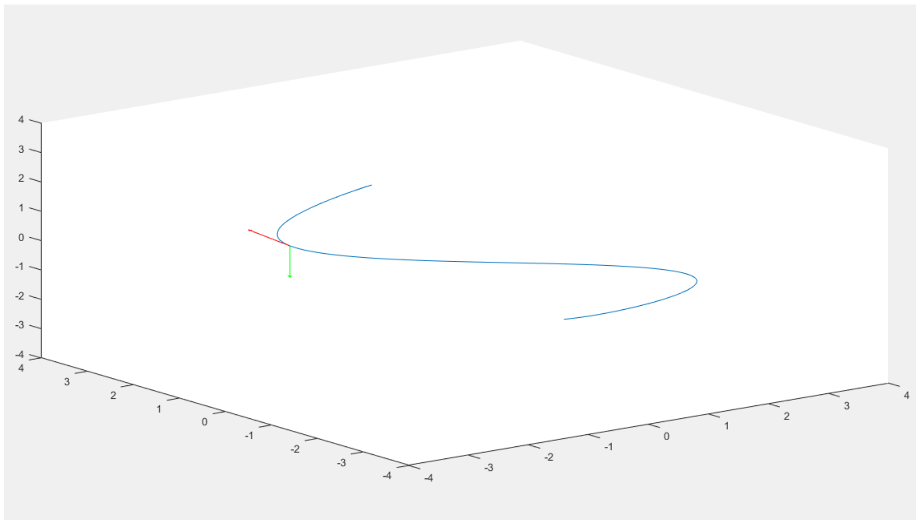


Figure 1: 3D graph of curve with linear and angular velocity vectors

The next problem to solve is figuring out what the highest value of β is that won't exceed the maximum wheelspeed limit of 2 m/s. To do this, I took the right and left velocity equations and plugged them into a graphing calculator program. I made β a slider and adjusted it up until a point on one of the functions was almost touching y=2.

I'm sure that there is a better and more programmatic way to do this, I just don't know the syntax for it. In pseudocode: I would find the relative maximum of both functions on the interval, set both equal to the maximum wheelspeed, solve for β, and then take the minimum out of the two resulting answers.

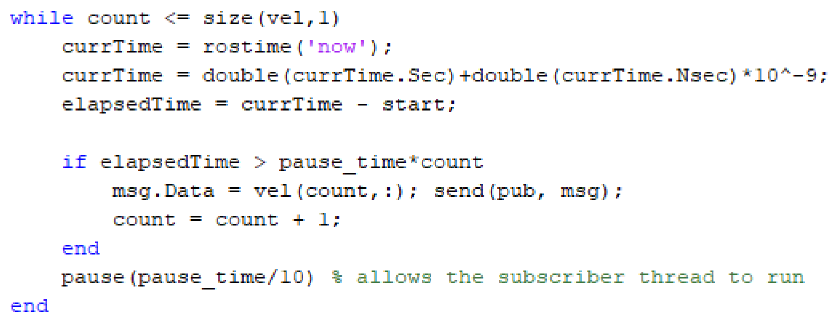
Finally, I discretized the path data by calculating n number of times and corresponding velocities between the start and the end of the path, storing them in an array called pathdata. In a .mat file I saved three variables: pathdata, the function R, and the function $\hat{T}$. This file will be called up later from the path execution program. The reason I included R and $\hat{T}$ was because they are necessary to calculate the initial position and heading of the robot on the bridge.

A small note: I decreased the endpoint of the function (βt) slightly because this prevented the robot from driving off the other end of the bridge.

3 Autonomous Program

The autonomous program contains a function called drive which takes in two filenames: the path file which it reads the times and velocities from and the data file which it writes the encoder and accelerometer data to. It starts data collection, places the neato and waits for it to settle, then begins driving.

The most important section of the drive method is the control loop to send the velocities, which is a while loop with a counter on how many velocities have been sent. It constantly checks the simulation time using $rostime('now')$ to see if the elapsed time has yet passed the point when it should send the next velocity. Inside the while loop, there is a short (0.01s) pause, allowing the rossubscriber thread to interrupt and execute the callback function to collect data.



```
while count <= size(vel,1)
    currTime = rostime('now');
    currTime = double(currTime.Sec)+double(currTime.Nsec)*10^-9;
    elapsedTime = currTime - start;

    if elapsedTime > pause_time*count
        msg.Data = vel(count,:); send(pub, msg);
        count = count + 1;
    end
    pause(pause_time/10) % allows the subscriber thread to run
end
```

Figure 2: Code snippet of control loop

4 Graphs and Analysis

Looking at the collected data, the accelerometer readings were very spiky and hard to understand, so I stuck to the encoders only. First of all I plotted the raw data from both of them over time.

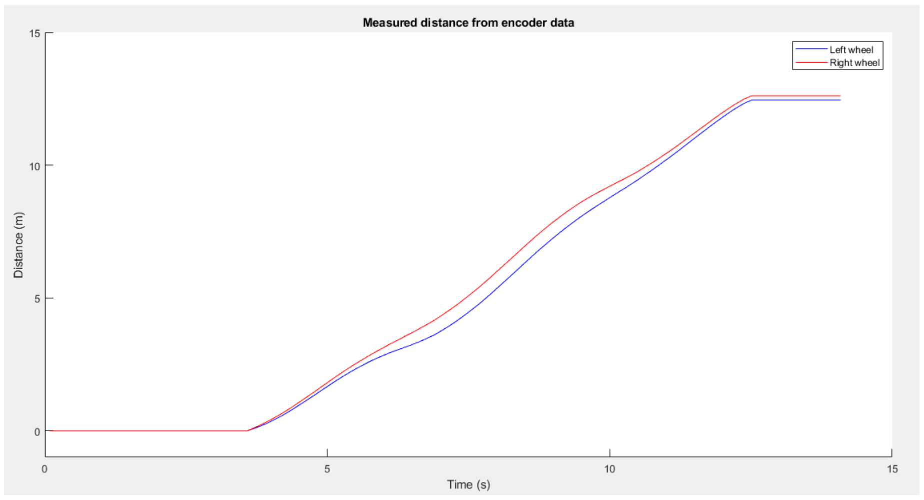


Figure 3: Graph of encoder distance over time

Next, I differentiated the encoder distances to get the measured right and left wheel velocities, adding them as dashed lines to the wheel velocity graph. There is one confusing part near the end where the red and blue swap.

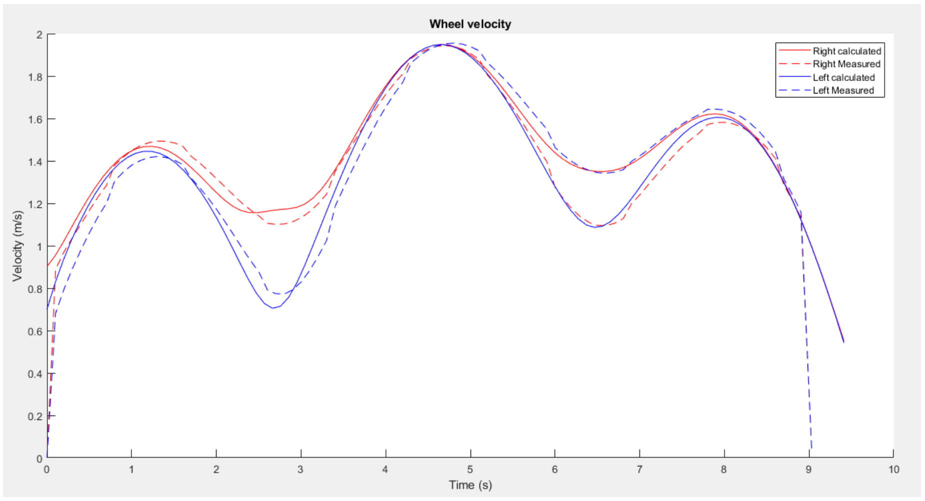


Figure 4: Graph of wheel velocity over time

Working backwards, I next calculated linear and angular velocity from right and left velocity. Formulas used for this (R, L as right, left encoder distance):

$$V = (R + L)'/(2 * dt)$$

$$\omega = (R - L)'/(d * dt)$$

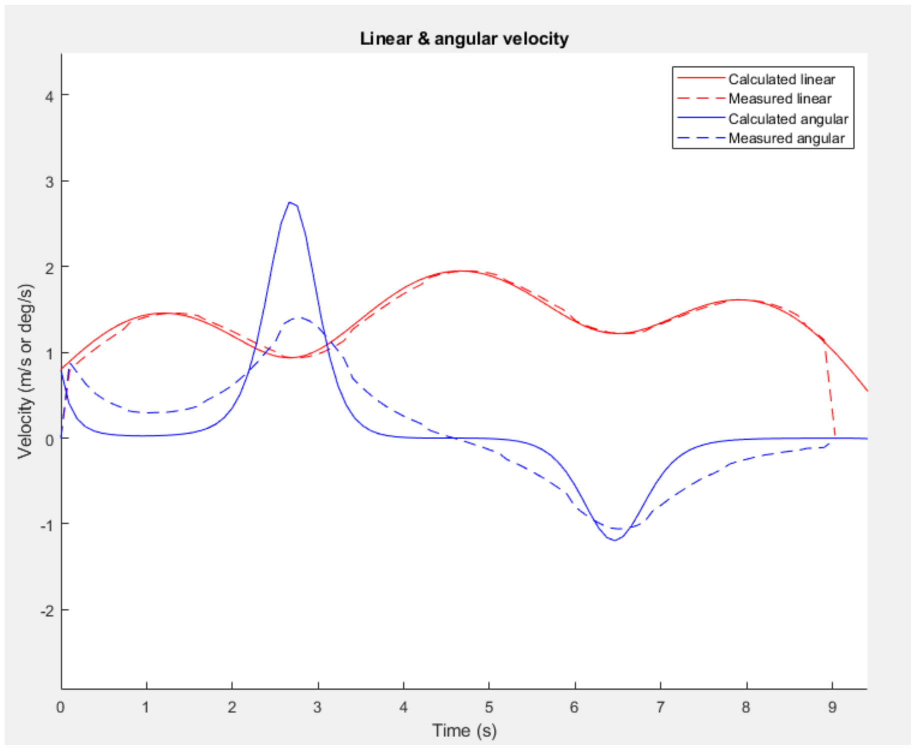
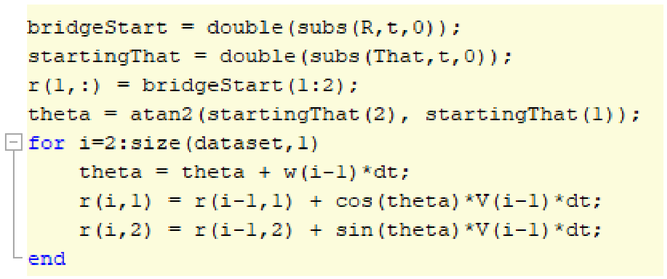


Figure 5: Graph of linear and angular velocity over time

Lastly, I integrated the linear and angular velocity back into the original parametric path. It ended up having a slight error which accumulated towards the end. When I adjusted the orignal heading to see how it impacted the error of the graph, I found that mostly corrected the problem. Such a seemingly small adjustment as two to three degrees made a huge difference. Yet it was still able to cross the bridge so I guess in this instance it doesn't matter so greatly. Here is a code snippet that I think best explains how I did this:



```
bridgeStart = double(subs(R,t,0));
startingThat = double(subs(That,t,0));
r(1,:) = bridgeStart(1:2);
theta = atan2(startingThat(2), startingThat(1));
for i=2:size(dataset,1)
    theta = theta + w(i-1)*dt;
    r(i,1) = r(i-1,1) + cos(theta)*V(i-1)*dt;
    r(i,2) = r(i-1,2) + sin(theta)*V(i-1)*dt;
end
```

Figure 6: Code snippet of integrating V and ω into parametric r

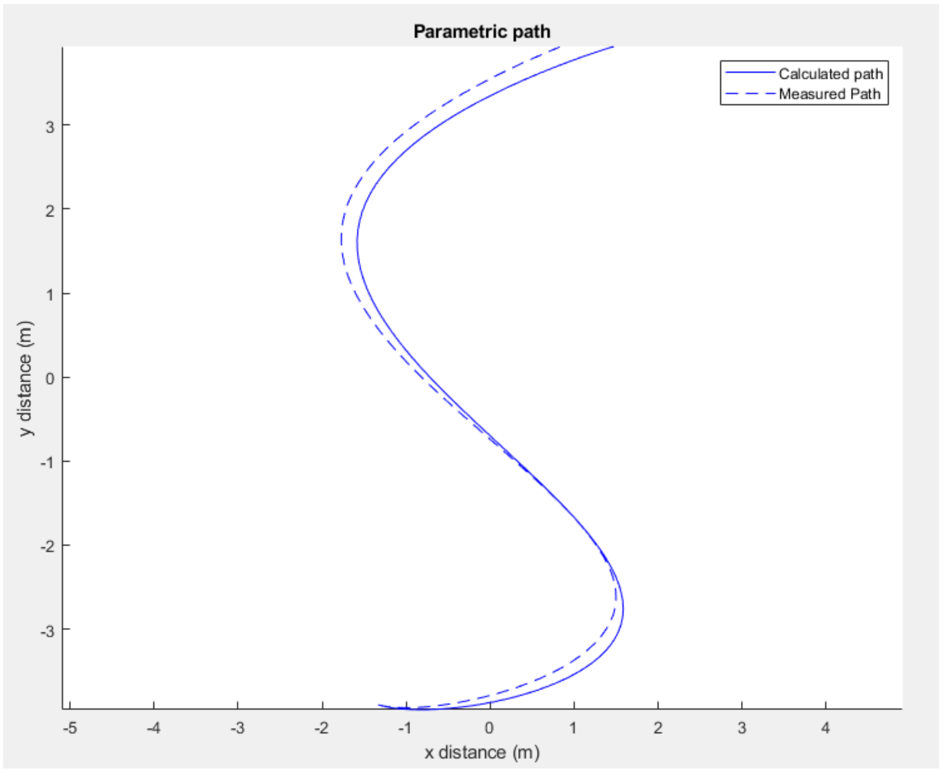


Figure 7: Graph of parametric path function

5 Results

My robot drove the Bridge of Doom successfully (most of the time). This was pretty good for not using any feedback, which I believe is called an open-loop system. If I had more time, I would have liked to add error-correction such as proportional response to the error in the position or velocity, making its' behaviour more accurate. Sadly, I did not have time to implement that feature.

Links to the video of my robot crossing the bridge:
- GitHub (requires download):
github.com/liloheinrich/BridgeOfDoom/blob/master/media/Bridge_of_Doom.mp4
- YouTube: youtu.be/l399SHLOJIs

6 Code

Here is a link to my GitHub repository containing this code:
https://github.com/liloheinrich/BridgeOfDoom