Night 2: Challenge 1: Crossing the Bridge of Death

Quantitative Engineering Analysis

1 Overview

Welcome to Robo Ninja Warrior. Your first challenge, should you choose to accept it (and you should!), will be crossing the *Bridge of Death*™. This challenge will push you and your robot literally to the brink, requiring you to be at the height of your analytical powers. Along the way you'll be building your knowledge of parametric curves, deriving robot motion models, and learning powerful validation and debugging techniques.

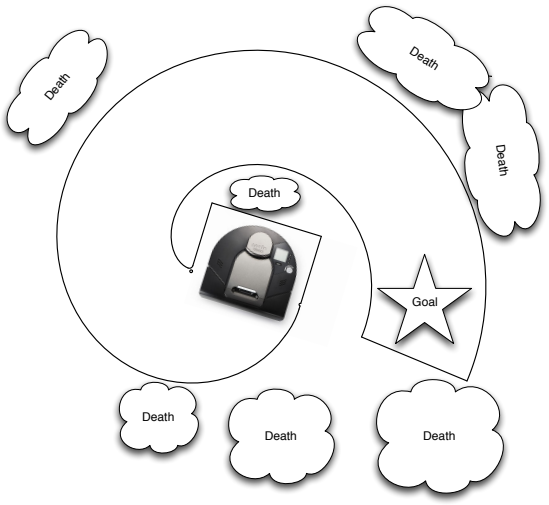

Figure 1: The Bridge of Death™.

1.1 Learning Goals

By the end of this challenge, you should be comfortable with the following:

1. Computing tangent vectors and normal vectors to parametric curves, and connecting these vectors to motion.
2. Deriving a motion model of a robot.
3. Validating a motion model of a robot empirically.
4. Controlling a robot using an open-loop control strategy.

2 *The Challenge*

You will write a program to autonomously pilot your robot from the starting platform to the goal. What lies between? Ahh, that is the harrowing Bridge of Death™. The shape of the centerline of the Bridge of Death™ is defined by one of the following parametric curves

$$\mathbf{r}(u) = -2a\left((l-\cos u)\cos u + (1-l)\right)\mathbf{i} + 2a(l-\cos u)\sin u\mathbf{j}\ (u \in [0, 2\pi], a = 0.4, l = 0.4).$$

or

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

3 *(Re)Meet your Neato*

You've already achieved passing familiarity with your Neato (see Figure 2), however, in this challenge you two will really get acquainted! The Neato moves via differential drive, which we worked with in class Monday. In this challenge you are tasked with piloting your robot across the Bridge of Death™. In order to control your robot, you will be using open-loop control. Open-loop control means that you will determine a sequence of motor commands (e.g., the velocities for each of the Neato's wheels) ahead of time. You will then write a program that sends these motor commands to the robot at the prescribed times irrespective of where the robot is along the path. Despite its simplicity, open-loop control can be quite powerful, and it is up to the task of crossing the Bridge of Death™. That being said, if you are looking for ways to take this challenge to the next level, you can use sensor feedback to modify your path midstream.

THE MAXIMUM SPEED OF YOUR NEATO IS ROUGHLY 0.3 METERS PER SECOND.

Figure 2: The Neato in all its glory. The Neato will be your bot for the duration of this module.

4 *Measured Paths*

One potential source of error that you may have identified in the in class exercises is that your robot is not able to instantaneously achieve a desired V_L and V_R when you send it a particular motor command. Given the pesky laws of physics, instead, the robot needs to accelerate to the desired velocity. In order to get a more accurate picture of what the robot actually did, we can use *measurements* of the wheel velocities to give us a more accurate estimate of the robot's

actual path in the world. Our Neato is outfitted with sensors called *wheel encoders*, which provide accurate estimates of the linear travel of each wheel over time. Knowing the linear travel and the time between measurements, the velocity of each wheel can be calculated. Next, you'll be determining formulas to update the robot's position and heading given measured values of V_L and V_R.

Exercise 1

Suppose that at time t your robot is at position $\mathbf{r}(t)$, with a heading of $\theta(t)$. Let's further assume that at $t = 0$ the robot is stationary and pointing along $\theta = 0$, which corresponds to the robot facing along the positive x-axis of the room(draw a picture to make sure that you are clear as to the definition of the coordinate system). Given measured values for V_L and V_R determine the values of $\mathbf{r}(t + \Delta t)$, and $\theta(t + \Delta t)$ which represent the position and heading of your robot at time $t + \Delta t$. For a discretized path (expressed in terms of short time increments rather than continuously):

1. You can, assuming that the time-step Δt is small, approximate a path by a series of movements in $\mathbf{r}$ and movements about the center of the robot in θ. The velocity of the robot is

$$\begin{aligned} \frac{d\mathbf{r}}{dt} &= v\hat{\mathbf{T}} \\ \frac{d\theta}{dt} &= \omega \end{aligned}$$

   where v is the linear velocity, ω is the angular velocity, and since the robot is always oriented along the path we can define $\hat{\mathbf{T}}$ as

$$\hat{\mathbf{T}} = \cos\theta\mathbf{i} + \sin\theta\mathbf{j}$$

   Using these definition, how would we approximate the position $\mathbf{r}(t + \Delta t)$ and heading $\theta(t + \Delta t)$ at the next point in time?

4.1 Validation Revisited

Exercise 2

Repeat at least one of your experiments from in class. This time, after the robot has completed its movement, plot the predicted path of your robot and the estimated path of your robot as determined by the measured wheel velocities. In order to do this, we have provided you a nice little script written by Paul Ruvolo (edited by Jeff Dusek) which you can use to collect the wheel position encoder data while you are running your experiment (which you can easily convert to velocities by taking the difference between two adjacent positions and dividing it by the timestep). Note that the robot's initial position is arbitrary. Hint: viscircles is a useful matlab function for plotting the predicted path of your robot if your predicted path is circular. What were the results? Comment on any perceived similarities or differences between the predicted and measured paths.

The script for collecting encoder data is called *collectDataset.m* and is linked to the Canvas assignment. It is also available from the sample code page. To collect encoder data, run the function *collectDataset('filename.mat')* from your command window. This will bring up a new figure window with the title "Dataset Collection Window". To start data collection , hit the space bar while focusing on the figure window. You will see the message "Starting Dataset Collection" in your command window if everything is working as intended. You can then run your personal script to control the robot, and encoder data will be collected in the background. When your robot motion has concluded, re-focus on the "Dataset Collection Window", and hit the space bar to stop data collection. You will see the message "Stopping Dataset Collection" in your command window.

After you stop the data collection, you will have a file *filename.mat* in your current directory. If you load this file, you will find a matrix "dataset" that contains the encoder and accelerometer data recorded from the robot. For this challenge you only care about the encoder data in columns 2 and 3, and the time stamps in column 1 (recall this data is linear travel of the wheel). The form of the data is:

$$dataset = [time, Pos_{left}, Pos_{right}, AccelX, AccelY, AccelZ] \quad (1)$$

Important: If you include a loop in your personal robot control script, make sure to include a pause of the form *pause(0.1)* within the loop. Otherwise, Matlab will try to execute that loop as fast as possible and will prevent the data collection script from recording encoder data.

Extension: you can also read the robot's wheel positions real time from the */encoders* topic and use this to plot the robot's motion real-time as it moves through the exercise...or even to correct for motion errors to bring it back closer to the desired path!

5 *Crossing the Bridge of Death*

Let's reflect on how far we've come towards completing our challenge. We have developed equations for V_L and V_R that achieve a desired linear and angular velocity, and we have validated this model empirically. All that remains is to program our robot to follow a parametric curve.

Exercise 3

Compute your robot's speed and angular velocity as a function of time for the following path.

$$\mathbf{r}(t) = \frac{1}{2}\cos(t)\hat{\mathbf{i}} + \frac{3}{4}\sin(t)\hat{\mathbf{j}}$$

Exercise 4

Compute your robot's speed and angular velocity as a function of time for the Bridge of Death™. Write a program to send the appropriate control signal based on the time elapsed since the start of the path. Be careful about handling the case when $\|\mathbf{T}(t)\| = 0$. In this case $\mathbf{N}(t)$ is not defined and $\omega(t) = 0$. Note, that you can always slow down or speed up your robot by multiplying t by a constant (be careful since the maximum speed of each of the robot's wheels is $0.3m/s$). Test your program thoroughly on the Bridge of Death™ while it is lying on the floor. When you are convinced your system is working properly, add some danger. Remember, always use a robot spotter when crossing the Bridge of Death™.

Exercise 5

Map your robot's predicted and actual path crossing the Bridge of Death™by using the provided code to collect the wheel encoder data and convert that to coordinates and headings for the robot throughout its perilous journey. You may want to use the Matlab quiver command for this.

6 *Writing up Your Work*

Prepare a writeup of your work on this challenge. Your writeup should contain the following components.

1. Your answers to all of the exercises in this document.
2. A description of your general process. What strategies did you try? What worked? What didn't? We are looking for something relatively comprehensive. A good length would be about a page.
3. A link to a youtube video of your robot in action (include a link in your writeup).

In addition to the writeup, you should also turn in your code (you could add a link to a Github repo in your writeup, or upload your MATLAB code files to your Canvas submission).

7 *Extension*

One weakness of the approach that you implemented is that it doesn't take into account the fact that the robot doesn't instantaneously do what you tell it to do. One possible way to remedy this is to monitor the robot's position over time using live readings of the wheel encoders. In the previous exercise, you derived a method for updating the robot's position and orientation given measurements of its wheel velocities. We call this estimate of the robot's position its odometry. By comparing the robot's position as determined by its odometry with the desired position (given by $\mathbf{r}(t)$) you can try to correct your robot's motion to more faithfully follow the path. How you accomplish this exactly is up to your own creativity and analysis skills.