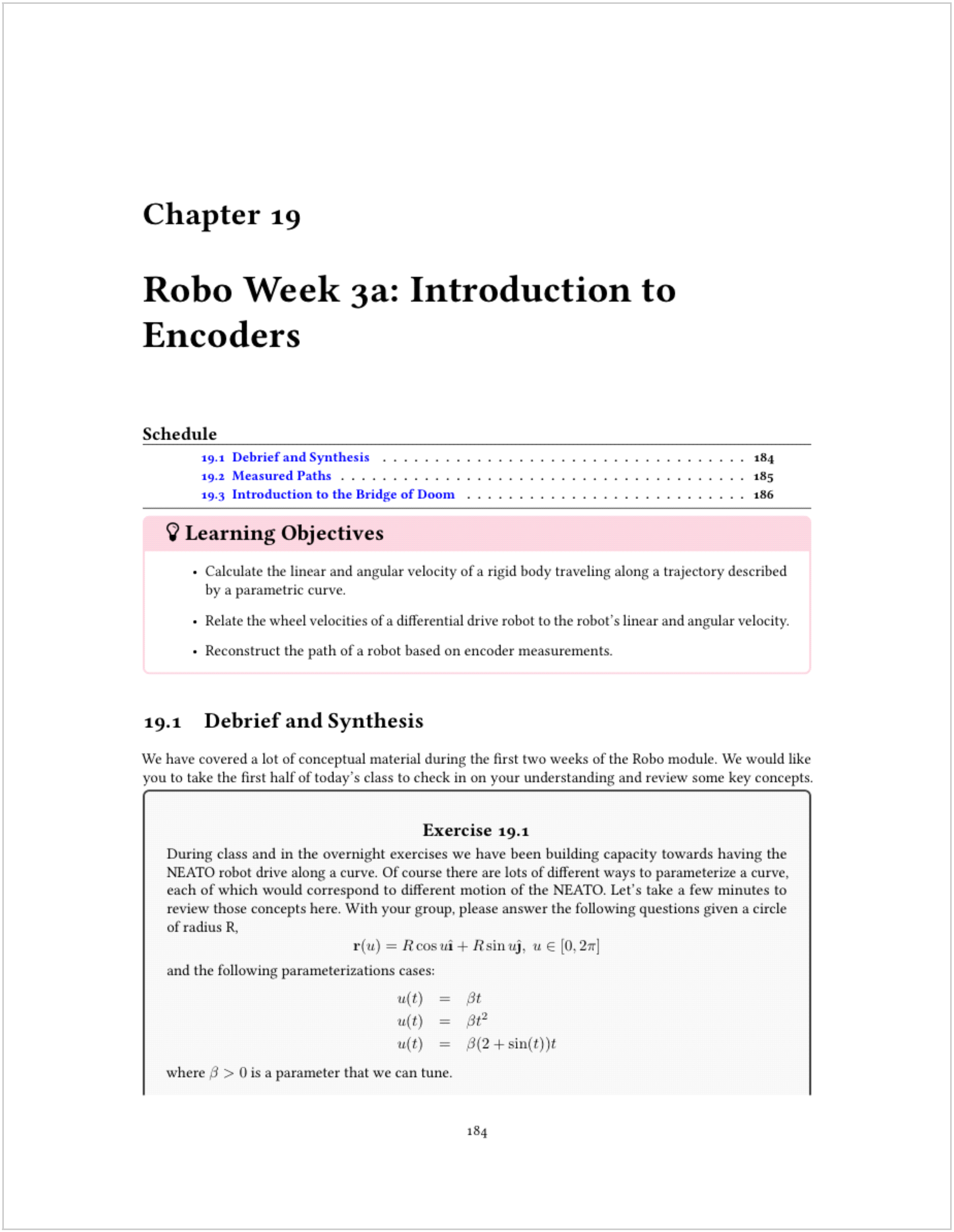
Week 3

Monday, March 29, 2021 10:20 AM

RoboWeek 3



Chapter 19

Robo Week 3a: Introduction to Encoders

Schedule

Learning Objectives

- Calculate the linear and angular velocity of a rigid body traveling along a trajectory described by a parametric curve.
- Relate the wheel velocities of a differential drive robot to the robot's linear and angular velocity.
- Reconstruct the path of a robot based on encoder measurements.

19.1 Debrief and Synthesis

We have covered a lot of conceptual material during the first two weeks of the Robo module. We would like you to take the first half of today's class to check in on your understanding and review some key concepts.

Exercise 19.1

During class and in the overnight exercises we have been building capacity towards having the NEATO robot drive along a curve. Of course there are lots of different ways to parameterize a curve, each of which would correspond to different motion of the NEATO. Let's take a few minutes to review those concepts here. With your group, please answer the following questions given a circle of radius R,

$$\mathbf{r}(u) = R\cos u\hat{\mathbf{i}} + R\sin u\hat{\mathbf{j}},\ u \in [0, 2\pi]$$

and the following parameterizations cases:

$$\begin{aligned} u(t) &= \beta t \\ u(t) &= \beta t^2 \\ u(t) &= \beta(2+\sin(t))t \end{aligned}$$

where $\beta > 0$ is a parameter that we can tune.

2) a) $u = \beta t = 2\pi \Rightarrow t = \frac{2\pi}{\beta}$

b) $t = \sqrt{\frac{2\pi}{b}}$

c) $\beta(2+\sin(t))t = 2\pi$

solve numerically w/ MATLAB

[illegible]

solve numerically w/ MATLAB

3) $r'(v) = v(v) = -R\sin(v)v'\hat{i} + R\cos(v)v'\hat{j}$

1. Qualitatively describe the motion of the NEATO in each case. A sketch could be useful.

a) circle const speed
b) circle inc speed
c) circ w/ oscilating speed

2. How long does it take to traverse the curve once? (Hints: Your answer should depend on β. If the answer cannot be determined symbolically, indicate how you could go about solving it.)
3. How would you find the linear velocity of a robot traveling along the curve? What is the direction of this velocity? (It would be really great if you employed the chain rule here and kept your work as general as possible.)
4. How would you find the angular velocity? What is the direction of the angular velocity? (It would be really great if you employed the chain rule here and kept your work as general as possible.)

$v'(t)\hat{k}$

5. Having found the linear and angular velocity, how would you find the left and right wheel speeds for the differential drive?

$V_L = |V| - |\omega|\frac{d}{2}$
$V_R = |V| + |\omega|\frac{d}{2}$

6. The robot has a maximum wheel speed of 0.3 m/s. How would you choose β to ensure your robot never exceeds this speed limit? $dist = 2\pi R$ per $u = 2\pi$ $\frac{u(t)}{2\pi} \cdot 2\pi R$

$R\,u(t) = 0.3\frac{m}{s}$
Solve for β

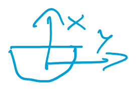
19.2 Measured Paths

One potential source of error when working with a real robot, is that the 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, the robot instead 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 Neatos (both real and virtual) are outfitted with sensors called *wheel encoders*, which provide accurate estimates of the linear travel of each wheel over time. For a nice introduction to encoders and how they work, check out this video from Sparkfun. 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.

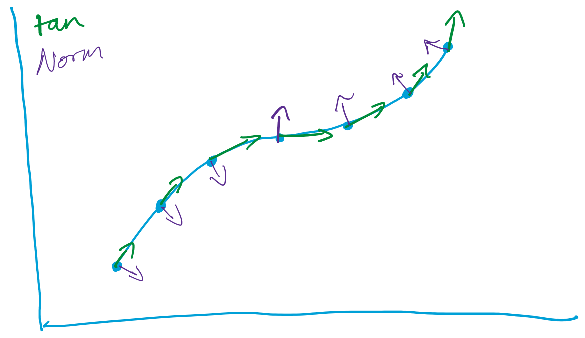
Exercise 19.2

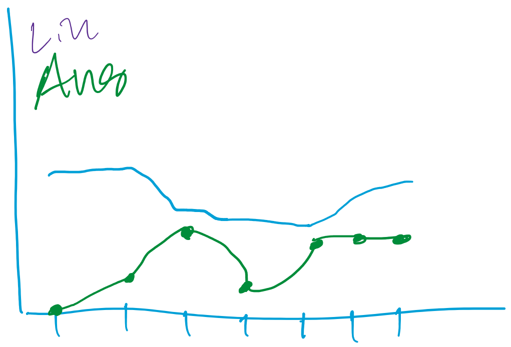
Suppose that at time t your robot is at position $\mathbf{r}(t)$, with a heading of $\theta(t)$. Let's further assume that at $t = 0$ the robot is stationary and pointing along $\theta = 0$, which corresponds to the robot facing along the positive x-axis of the room.

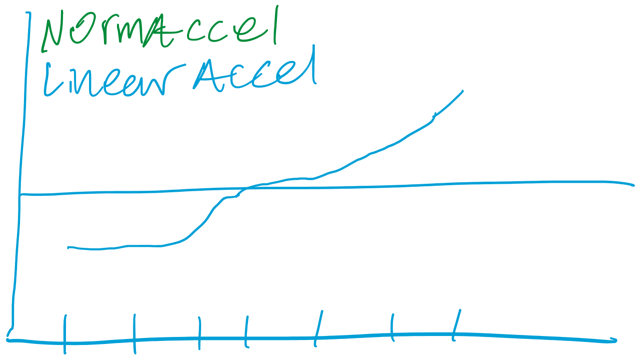
1. Draw a picture to make sure that you are clear as to the definition of the coordinate system.



2. Sketch a (fairly smooth) potential trajectory that your robot will be driving. Choose several points along the curve separated by Δt. Sketch in the unit tangent and unit normal vector at each point.
3. Sketch an estimated plot of your robot's linear and angular speeds as a function of time as it traverses your curve.
4. Sketch an estimated plot of your robot's tangential and normal components of acceleration as a function of time as it traverses your curve.
5. Sketch a qualitative version of your estimate of right and left wheel velocity as a function of time.


tan
Norm


Lin
Ang


NormAccel
Linear Accel

Exercise 19.3

For a discretized path (expressed in terms of short time increments rather than continuously) 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 speed, ω is the angular velocity in the $\hat{\mathbf{k}}$ direction, and since the robot is always oriented along the path we can define $\hat{\mathbf{T}}$ in relation to the global (classroom fixed) coordinate system as

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

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 $t + \Delta t$.

19.3 Introduction to the Bridge of Doom

Challenge one for the robotics module is the Bridge of Doom. Successfully complete this challenge, you will need to command your robot to traverse a bridge described by a parametric curve. The Bridge of Doom challenge can be found in Chapter 21, and we encourage you to start working on the path planning portions of the challenge after today's class.

Solution 19.1

1. Qualitatively describe the motion of the NEATO in each case.

   (a) For the parameterization $u(t) = \beta t$, the NEATO drives around the circle described by $\mathbf{r}(u)$ at a constant velocity. The velocity of the NEATO will be impacted by the choice of β.

   (b) For the parameterization $u(t) = \beta t^2$, the NEATO drives around the circle described by $\mathbf{r}(u)$ with an increasing velocity due to the t^2 term in the parameterization.

   (c) For the parameterization $u(t) = \beta(2 + \sin(t))t$, the NEATO will travel around the circle described by $\mathbf{r}(u)$, but the velocity will oscillate between positive and negative, causing the robot to actually backup along the path at times.

2. How long does it take to traverse the curve once? (Your answer should depend on β If the answer cannot be determined symbolically, indicate how you could go about solving it.) To traverse the circle once, the parameter u must travel from 0 to 2π. Therefore, to determine the time to complete a traverse, we must solve for t in each of the parameterizations, and plug in $u = 2\pi$.

   (a) For the parameterization $u(t) = \beta t$, $t = \frac{u(T)}{\beta}$, so $t_{final} = \frac{2\pi}{\beta}$.

   (b) For the parameterization $u(t) = \beta t^2$, $t = \sqrt{\frac{u(T)}{\beta}}$, so $t_{final} = \sqrt{\frac{2\pi}{\beta}}$. Note that this time is less than the previous parameterization. This makes sense since the robot is speeding up throughout the traverse of the curve.

   (c) For the parameterization $u(t) = \beta(2 + \sin(t))t$, $\frac{u(t)}{\beta t_{final}} = 2 + \sin(t)$, so $\frac{2\pi)}{\beta t} = 2 + \sin(t_{final})$. This can be solved using Matlab or any other numerical solver.

3. How would you find the linear velocity of a robot traveling along the curve? What is the direction of this velocity? (It would be really great if you employed the chain rule here and kept your work as general as possible.) The linear velocity would be found by

$$\mathbf{v}(t) = \frac{d}{dt}[\mathbf{r}(u(t))] = \mathbf{r}'(u(t))u'(t)$$

   after applying the chain rule. For the function

$$\mathbf{r}(u(t)) = R\cos(u(t))\hat{\mathbf{i}} + R\sin(u(t))\hat{\mathbf{j}},$$

   we can write

$$\frac{d}{dt}\mathbf{r}(u(t)) = -R\sin(u(t))u'(t)\hat{\mathbf{i}} + R\cos(u(t))u'(t)\hat{\mathbf{j}}$$

   where

$$u'(t) = \frac{d}{dt}u(t).$$

   The velocity is in the direction of the unit tangent vector

$$\hat{\mathbf{T}} = \frac{\mathbf{r}'}{|\mathbf{r}'|} = \frac{-R\sin(u(t))u'(t)\hat{\mathbf{i}} + R\cos(u(t))u'(t)\hat{\mathbf{j}}}{Ru'(t)} = -\sin(u(t))\hat{i} + \cos(u(t))\hat{j}$$

   .

4. How would you find the angular velocity? What is the direction of the angular velocity? (It would be really great if you employed the chain rule here and kept your work as general as possible.) The angular velocity can be found as $\omega = \hat{\mathbf{T}} \times \frac{d\hat{\mathbf{T}}}{dt}|_{room}$ which simplifies to $u'(t)\hat{k}$. The angular velocity will always be in the $\pm\hat{k}$ direction. Positive $\hat{k}$ indicates a counterclockwise rotation, while negative $\hat{k}$ indicates a clockwise rotation.

5. Having found the linear and angular velocity, how would you find the left and right wheel speeds for the differential drive?

$$V_L = V - \omega\frac{d}{2} \quad (19.1)$$

$$V_R = V + \omega\frac{d}{2} \quad (19.2)$$

6. The robot has a maximum wheel speed of 0.3 m/s. How would you choose β to ensure your robot never exceeds this speed limit?

   We know from above that $\frac{d}{dt}\mathbf{r}(u(t)) = -R\sin(u(t))u'(t)\mathbf{\hat{i}} + R\cos(u(t))u'(t)\mathbf{\hat{j}}$, and the magnitude of the linear velocity is $|r'(u(t))| = R|u'(t)|$. We also know that $|\omega| = |u'(t)|$. Combining this with the equations for left and right wheel velocity above, we would choose β so:

$$0.3 \geq Ru'(t) - |u'(t)|\frac{d}{2} \quad (19.3)$$

$$0.3 \geq Ru'(t) + |u'(t)|\frac{d}{2} \quad (19.4)$$

Solution 19.2

Solutions to this exercise will vary by breakout room based on your robot's trajectory.

Solution 19.3

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 $t+\Delta t$.
Conceptually, we can think about the robot making discrete moves in space over the period Δt. Each of these moves would consist of a translation and a rotation. From V_L and V_R, we can find the linear and angular velocities as:

$$V = \frac{V_L + V_R}{2}$$

$$\omega = \frac{V_R - V_L}{d}$$

With these velocities, for a robot initially at position $\mathbf{r}(t) = x(t)\hat{i} + y(t)\hat{j}$ in the global frame, we can then say:

$$\begin{aligned}
\mathbf{r}(t+\Delta t) &= \mathbf{r}(t) + \frac{dr}{dt}\Delta t \\
\mathbf{r}(t+\Delta t) &= \mathbf{r}(t) + v\hat{\mathbf{T}}\Delta t \\
\mathbf{r}(t+\Delta t) &= (x(t) + V\cos(\theta(t))\Delta t)\hat{i} + (y(t) + V\sin(\theta(t))\Delta t)\hat{j}
\end{aligned}$$

To fully define the position of the robot, we must know its orientation $\theta(t)$ with respect to the global reference frame, in addition to its position $\mathbf{r}(t)$:

$$\begin{aligned}
\theta(t+\Delta t) &= \theta(t) + \frac{d\theta}{dt}\Delta t \\
\theta(t+\Delta t) &= \theta(t) + \omega\Delta t
\end{aligned}$$

Chapter 20

Robo Week 3b: Measured Paths

Schedule

Learning Objectives

- Relate the wheel velocities of a differential drive robot to the robot's linear and angular velocity.
- Reconstruct the path of a robot based on encoder measurements.
- Explore sources of error (e.g. wheel slippage) between encoder output and the motion of a robot.

20.1 Encoders in Action

During class today you will complete a series of simple experiments with the NEATO while simultaneously collecting motion data. In order to do this, we have provided you a web-based tool for collecting the wheel position encoder data while you are running your experiment.

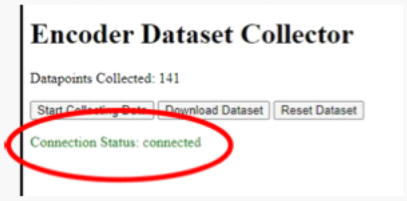
Exercise 20.1

Watch the video demonstrating how to collect a simulated encoder dataset here. Once you have watched the video, try collecting a sample dataset by completing the following steps:

1. Open the robo simulator by running the command:

```
qeasim start empty_no_spawn
```

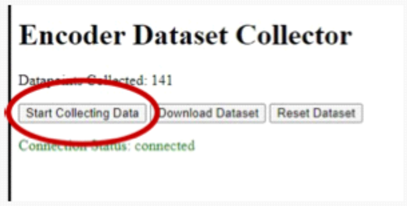
2. With the simulator running visit the following link encoder_dataset_collector. This will open the web-based data collection tool in your local browser. Make sure you see "Connection Status: connected."


Encoder Dataset Collector

Datapoints Collected: 141

Connection Status: connected

3. Click the "start collecting data" button to begin collecting wheel encoder data. You should see the number of "Datapoints Collected" begin to increase.


Encoder Dataset Collector

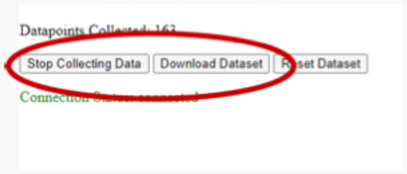
Datapoints Collected: 141

Connection Status: connected

4. Run a basic robot experiment by running the following command in your browser-based Matlab:

```
runBasicWheelVelocityExperiment
```

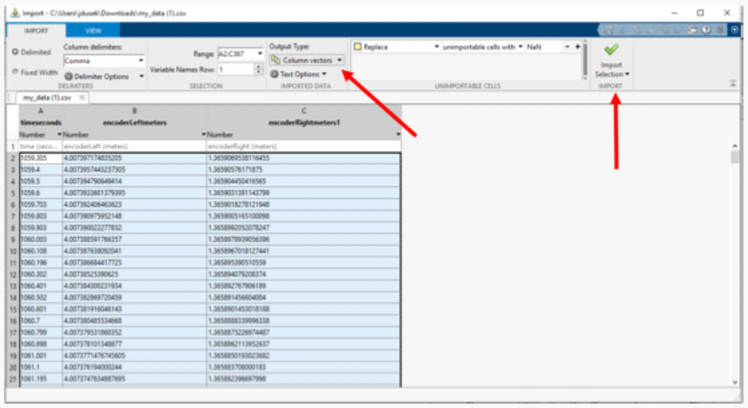
5. When your robot has stopped moving, stop data collection using the "Stop Data Collection" button, then download the dataset using the "Download Dataset" button.


Datapoints Collected: 163

Connection Status: connected

6. After you complete your experiment and download your data, you can reset the dataset using the "Reset Dataset" button.

7. Load the collected data into your local Matlab by drag and dropping the "my_data.csv" file into the command window. This will bring up the Matlab import GUI. You will want to import your data as "Column Vectors" as shown in the image below. The default variables names will be *encoderLeftmeters*, *encoderRightmeters*, and *timeseconds*. You may choose to rename the variables for convenience.



Delimited · Column delimiters: Comma · Delimiter Options · Fixed Width

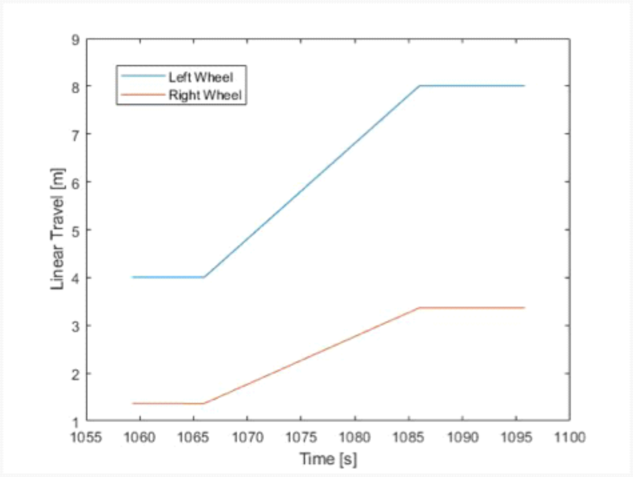
Range: A2:C367 · Variable Names Row: 1

Output Type: Column vectors · Text Options

Replace unimportable cells with NaN

	A	B	C
	timeseconds	encoderLeftmeters	encoderRightmeters1
	Number	Number	Number
1	time (seco...	encoderLeft (meters)	encoderRight (meters)
2	1059.305	4.007397174835205	1.3659069538116455
3	1059.4	4.0073957443237305	1.36590576171875
4	1059.5	4.007394790649414	1.365904450416565
5	1059.6	4.0073933601379395	1.3659031391143799
6	1059.703	4.007392406463623	1.3659018278121948
7	1059.803	4.007390975952148	1.3659005165100098
8	1059.903	4.007390022277832	1.3658992052078247
9	1060.003	4.007388591766357	1.3658978939056396
10	1060.108	4.007387638092041	1.3658967018127441
11	1060.196	4.007386684417725	1.365895390510559
12	1060.302	4.00738525390625	1.365894079208374
13	1060.401	4.007384300231934	1.365892767906189
14	1060.502	4.007382869720459	1.365891456604004
15	1060.601	4.007381916046143	1.3658901453018188
16	1060.7	4.007380485534668	1.3658888339996338
17	1060.799	4.007379531860352	1.3658875226974487
18	1060.898	4.007378101348877	1.3658862113952637
19	1061.001	4.0073771476745605	1.3658850193023682
20	1061.1	4.007376194000244	1.365883708000183
21	1061.195	4.0073747634887695	1.365882396697998

8. Plot the left and right wheel encoder linear travel values versus time. You should get a plot that looks something like the one below.



20.1.1 Finding Wheel Velocities

You can compute the left and right wheel velocities by taking the difference between two consecutive distances (e.g. $encoderLeftmeters(4) - encoderLeftmeters(3)$) and dividing it by the time difference ($timeseconds(4) - timeseconds(3)$ in this example). The Matlab function *diff* will likely be useful in this process. Note that the robot's initial e*encoderLeftmeters* and *encoderRightmeters* values are arbitrary and unrelated to x or y coordinates in space. The encoder values represent the linear distance of travel for each wheel.

20.2 Encoder Experiments

Complete the following simple experiments. For each experiment, record the encoder data for analysis using the encoder_dataset_collector.

Exercise 20.2

Using the provided DriveCircles.m function (it is already installed in the simulator, so the link is only for reading through the code), have the robot spin counterclockwise for ten seconds, then spin clockwise for ten seconds around around the robot's center (i.e V_R=-V_L or the opposite). The function takes inputs of left and right wheel velocity as *DriveCircles(vL,vR)*, and the times are already set. Collect and plot the left and right wheel encoder data.

1. Does the plot of the wheel linear travel look as you would expect?

2. Find the left and right wheel velocity at each time step and plot them. Do they match your expectations? If you know d, you can also find and plot the linear and angular velocity.

Exercise 20.3

Using encoder data to reconstruct the position of your robot has limitations- we assume there is a "no slip" condition between the wheel and the ground. In other words, the linear travel output from the encoder assumes that the wheel moves forward a distance of πd, where d is the wheel diameter, for each revolution. This assumption falls apart if there is *slipping* between the wheel and the surface. Launch the robot simulator using the command:

```
qeasim start ice_rink
```

Using the driveforward.m function introduced in the overnight, conduct three experiments where the robot drives a specified distance at increasing speeds. Collect and plot the encoder data. You might want to do "edit -> reset world" in the simulator between experiments.

1. Does the linear distance traveled collected by the encoders match the distance input to the function? How about the distance travelled by the robot in the simulated world?

2. If the values do not match, why not?

3. Plot the linear and angular velocity as a function of time. Do they match your expectations?

Exercise 20.4

If you have time in class: Calculate the left and right wheel velocities needed to drive a circle of radius 0.5m in 20s. Use the code runBasicWheelVelocityExperiment.m drive the circle while simultaneously collecting a wheel encoder dataset using encoder_dataset_collector.

1. Plot the anticipated trajectory of your robot in Matlab. You may want to use the Matlab command *axis equal* to make sure your circle looks nice and circular.

2. In the same figure, plot the actual trajectory of your robot. How closely do they match?
3. Quantifying error is an important part of any experiment. For the circular path, how would you calculate error between the anticipated trajectory and the actual path of the robot? Are you interested in total accumulated error? Average error and each time step? Distance away from the "ideal" curve at each time stop (e.g. are you stying within a lane)? Does something else make sense? As a group, draw sketches of what each of these types of error would represent visually, and come up with mathematical expressions.

20.2.1 Optional Extension: Encoders in Real-Time

You can also read the robot's wheel positions real-time from the */encoders* ROS 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!

Exercise 20.5

Modify the driveforward.m function to use encoder feedback instead of time to determine when to stop the robot.

Exercise 20.6

Try writing a function that commands the NEATO to drive a square with the side length as an input variable. Think about using the encoder feedback to determine when to turn, and whether the robot has turned 90 degrees. The *'/cmd_vel'* Ros topic might be a good option here because it takes the linear and angular velocity as inputs. If the square is too easy, how about driving a star?

Exercise 20.7

Feedback from the encoders can be used to correct the robot's position if it strays from the planned trajectory. Think about how you could calculate error between the anticipated and actual path, and what actions would need to be taken to reduce that error.

20.3 Crossing the Bridge of Doom- Robot Code and Mapping

The second half of the Bridge of Doom challenge, available in Chapter 21, is to write a piece of software to drive a robot across the bridge. While the robot is completing its perilous journey, you will collect encoder data, which will be used to reconstruct the robot's actual path. We will take a few minutes at the end of class to discuss your robot code and some recommendations for how to approach the problem.

Solution 20.2

1. Does the plot of the wheel linear travel look as you would expect? Example plot below for the case of $V_L = -0.3$ and $V_R = -0.3$ initially.

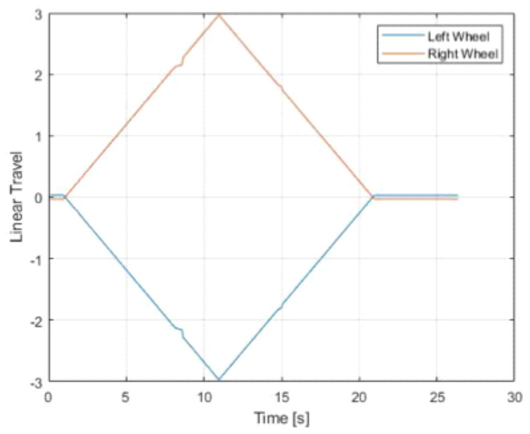


Figure 20.1: Linear travel of left and right wheel while spinning in circles around center.

2. Find the left and right wheel velocity at each time step and plot them. Do they match your expectations? Example plot below for the case of $V_L = -0.3$ and $V_R = -0.3$ initially.

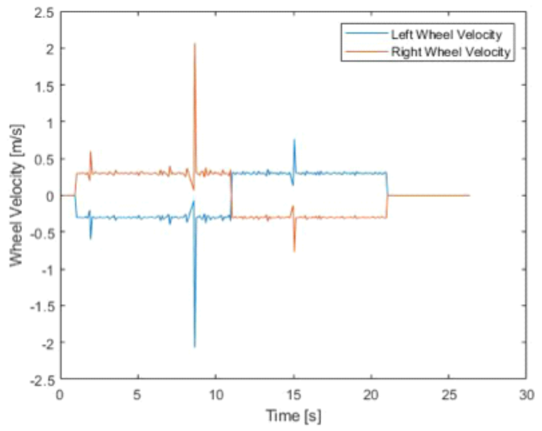


Figure 20.2: Velocity of left and right wheel while spinning in circles around center.

Solution 20.3

1. Does the linear distance traveled collected by the encoders match the distance input to the function? How about the distance travelled by the robot in the simulated world?

   See below.

2. If the values do not match, why not?

   The linear travel outputted from the encoders is a close match to the input distance to the function. The encoder distance measured is based off of wheel revolutions, so the fact the robot is slipping on ice doesn't impact this measurement. However, the slippage will impact the actual distance travelled. If we think back to the free body diagrams for rolling objects like cars and robots in the Boats module, we'll remember that the friction force (specifically static friction between the tires and the ground) moves the car or robot forward. This same principle applies to our Neato robots. To move, the motors apply a torque to the wheels and the friction that acts on the wheels as a result will move robot forward. If the torque from the motor is greater than the resultant torque from the friction force around the center of the wheel, then the wheel will slip. This generally occurs at higher velocities, as the wheels accelerate. For low velocities, there is minimal slip between the wheels and the ice rink surface, and the robot travels approximately the distance expected (possibly with some sliding at the end). For higher velocities, the wheels slip, and the distance travelled will be less than expected.

3. Plot the linear and angular velocity as a function of time. Do they match your expectations?

   Linear and angular velocity calculated from the encoders will closely match expectations. The linear and angular velocity of the actual body are different, but, in a real robot, without a position sensor like a GPS or Vicon, you won't know this from encoder data alone! Hopefully, this illustrates the perils of even a simple model like reconstructing position from wheel encoders.

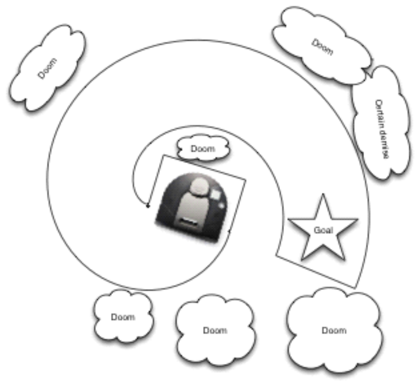
Chapter 21

Robo: Bridge of Doom Challenge

Contents

21.1 Overview

Welcome to Robo Ninja Warrior. Your first challenge, should you choose to accept it (and you should!), will be crossing the *Bridge of Doom*. 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.



The Bridge of Doom.

Learning Objectives

By the end of this challenge, you should be able to:

1. Compute tangent vectors and normal vectors to parametric curves, and connect these vectors to motion.
2. Derive a motion model of a robot.
3. Validate a motion model of a robot empirically.
4. Control a robot using an open-loop control strategy.
5. Map the path of a robot based on encoder values.

21.2 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 Doom. The shape of the centerline of the Bridge of Doom is defined by the following parametric curve:

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

21.3 (Re)Meet your Neato

You've already achieved passing familiarity with your Neato, however, in this challenge you two will really get acquainted! The Neato moves via differential drive, which we worked with in class this week. In this challenge you are tasked with piloting your robot across the Bridge of Doom. 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 Doom. 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.

For this challenge, we are considering the maximum wheel speed of the simulated NEATO to be 2.0 m/s. Your commanded wheel speeds should not exceed this value. The wheelbase for the simulated NEATO is d=0.235m.

21.4 Crossing the Bridge of Doom

We've already come a long way 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! We'll go step by step as follows:

Exercise 21.1

1. For the Bridge of Doom, plot the parametric curve that defines the centerline of the bridge.
2. On the same figure, plot the unit tangent and unit normal vectors at several points along the curve. You should have starter code to help with this in the Robo Homework 1 assignment.

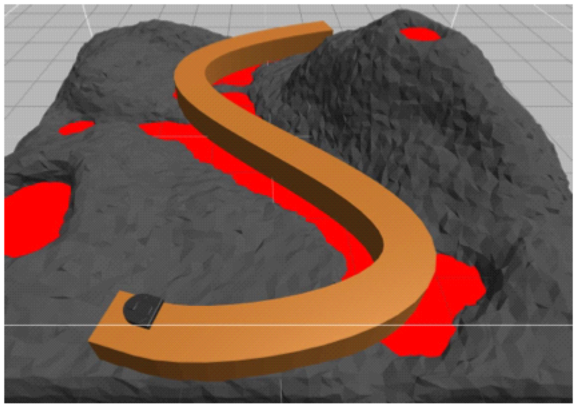


Figure 21.1: The fearless (simulated) NEATO traversing the treacherous Bridge of Doom.

Exercise 21.2

1. Given the parametric curve, compute and plot your robot's planned linear speed and angular velocity at its COM as a function of time for the Bridge of Doom.
2. *After successfully traversing the bridge* (i.e. come back to this part later), use your encoder data measurements to compute the linear speed and angular velocity of your robot and add this to your plot (**Note:** The wheelbase for the simulated NEATO is d=0.235m). The planned (theoretical) speed and angular velocity should be plotted with a solid line, while the experimental result should be plotted with a dashed line. Make sure your plots include appropriate units, labels, and legends.

Exercise 21.3

1. Compute and plot your robot's left and right wheel velocities as a function of time for the Bridge of Doom.
2. *After successfully traversing the bridge* (again, come back to this), use your encoder data to compute the measured wheel velocities and add them to your plot. The planned (theoretical) velocities should be plotted with a solid line, while the experimental result should be plotted with a dashed line. Make sure your plots include appropriate units, labels, and legends. Do the theoretical and measured values match?

Exercise 21.4

Write a program to drive your robot across the Bridge of Doom. To do this, you will need to send the appropriate control signal to your robot's wheels based on the time elapsed since the start of the path. You can choose to command the left and right wheel velocities directly, or the robot's linear and angular velocity. Both are available through ROS commands. **Take a video of your robot crossing the Bridge of Doom.**

Note: Be careful about handling the case when $|\mathbf{r}'(t)| = 0$. In this case $\hat{\mathbf{N}}(t)$ is not defined and $\omega(t) = 0$. You can always slow down or speed up your robot by multiplying t by a constant (we have used the constants α or β in previous assignments for this purpose). Be careful since **the maximum speed of each of the robot's wheels is 2.0 m/s.**

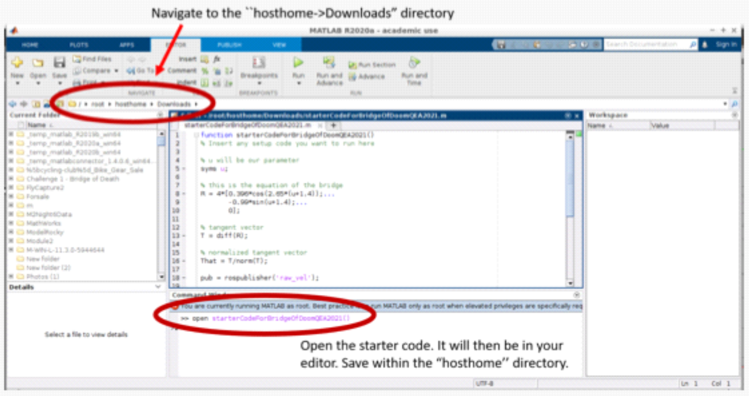
To get you started:

1. Start the Bridge of Doom environment in the simulator, use the following MATLAB command.

   ```
   >> qeasim start bod_volcano
   ```

2. Download the starter code here. The link will take you to the Robo Ninja Warrior webpage where you can click the "Download Source" link.

3. You will be modifying and running your code in the browser version of Matlab. After downloading the started code to you local machine, navigate to the "Downloads" folder in the browser version of Matlab to open the starter code.



Navigate to the ``hosthome->Downloads" directory

```
function starterCodeForBridgeOfDoomQEA2021()
% Insert any setup code you want to run here

% u will be our parameter
syms u;

% this is the equation of the bridge
R = 4*[0.396*cos(2.65*(u+1.4));...
       -0.99*sin(u+1.4);...
       0];

% tangent vector
T = diff(R);

% normalized tangent vector
That = T/norm(T);

pub = rospublisher('raw_vel');
```

```
>> open starterCodeForBridgeOfDoomQEA2021()
```

Open the starter code. It will then be in your editor. Save within the "hosthome" directory.

4. It is **crucial** that you save your code within the "hosthome" file tree when working in the browser version of Matlab. If you do not, you will lose your work when the broswer is closed. The starter code is a MATLAB function; make sure that you save it with a '.m' file extension.

5. The starter code will initialize the position of the simulated NEATO on the Bridge of Doom. You can build the rest of your robot control program from this starting point.

6. In the browser version of MATLAB, ensure the file is in your path (or folder). At the command window, type `starterCodeForBridgeOfDoomQEA2021()` to run the function.

7. To be successful in the simulated NEATO, you may need to base calculations involving time on 'simulation time'. The solution to Exercise 18.3, on page 173 has critical information for using ROS time (we recommend you use `rostic` and `rostoc`).

Exercise 21.5

Map your robot's predicted and actual path crossing the Bridge of Doom by using encoder_dataset_collector to capture encoder values as your robot traverses the bridge, convert that to coordinates and headings for the robot throughout its perilous journey (if you need a refresher on how to use the dataset collector, refer back to Section 20.1). Use the Matlab `quiver` command to plot the predicted and experimental unit tangent vectors at various points along the curve (note: do not include an arrow for every time step or your plot will be too cluttered). The planned (theoretical) path should be plotted with a solid line, while the experimental result should be plotted with a dashed line. Make sure your plots include appropriate units, labels, and legends.

21.5 Writing up Your Work

Exercise 21.6

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

1. A brief introduction explaining the challenge and including the functional definition of the bridge.
2. A description of your methodology explicitly identifying and explaining the key equations and parameters needed to produce the plots identified above.
3. Each of the plots detailed above with appropriate captions.
4. A link to a youtube video of your robot in action (include a link in your writeup). Bonus points (not really) for high production value like this.

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

21.6 Optional Extensions

Exercise 21.7

1. 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 class 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.
2. If you'd like, you can traverse the Icy Bridge of Doom. Change the NEATO_WORLD from bod_volcano to bod_ice_bridge. In order to navigate this, you will likely need to use the actual position as feedback to modify your control strategy. The actual position of your Neato is

given by the simulator. For an example of how to do this see the motion model experiment code from earlier in the module (basically e-mail us if you want to do this!).