Chapter 10

In-Class Activity Day 6: NeatoKart, Back to the Starting Line

Today, we are going to apply what we've learned about parametric curves to build an odometry model for the Neato. Specifically, we are going to use math to give the Neato the ability to keep track of its position and orientation. At the end of this assignment, we will write a homing routine that returns the Neato back to its original position and orientation (after having gone for a short drive).

Learning Objectives

- Further investigate the concept of velocity, especially as it relates to position and orientation
- Introduce the unit circle
- Write a "return home" function for a Neato after it travels an arbitrary path

10.1 Odometry Model: Encoder Feedback

In order to build a mathematical model that can keep track of the Neato's position, we must first understand the feedback information that is available to us (by feedback, we mean measurements of the environment and the state of the Neato). The Neato has access to a few different sources of feedback, including LIDAR data, a bump sensor, and encoder information. For the purposes of odometry (i.e. position and orientation tracking), we are particularly interested in the encoder measurements.

A rotary encoder is an electromechanical device that enables the measurement of rotation. By attaching an encoder to the shaft of a wheel, we are able to measure the net rotation of the wheel. In the case of the Neato, this means that our controller/state-estimator receives the angles $\phi_l(t)$ and $\phi_r(t)$ as feedback,

$$\phi_l(t) = \text{ rotation of the left wheel}, \qquad \phi_r(t) = \text{ rotation of the right wheel} \tag{10.1}$$

which correspond to the rotation of the left and right wheels respectively. We can then scale this rotation by the radius of the wheel to find the net "forward distance" traveled by each wheel (see fig 10.1):

$$x_l = (r_{wheel})\phi_l(t), \qquad x_r = (r_{wheel})\phi_r(t) \tag{10.2}$$

For the purposes of today's exercise (and Rainbow Road), you won't need to keep track of this scaling; since it's already built into the Neato interface (the MATLAB interface we provide you returns $x_l(t)$ and $x_r(t)$ instead of $\phi_l(t)$ and $\phi_r(t)$). For example, the following block of code connects to the Neato, sends a query for the encoder measurements, and stores it into two local variables:

```
%connect to the neato at IP address 192.168.16.01
neatov3.connect('192.168.16.01');
%query the neato for sensor data (including encoders)
sensors = neatov3.receive();
%unpack the encoder data into the variables xl and xr
%where xl, xr are the net forward distance traveled by the
```

```
%left and right wheels respectively
xl = sensors.encoders(1);
xr = sensors.encoders(2);
%disconnect from the neato
neatov3.disconnect();
```

Alternatively, the teleoperation (remote control) interface returns the encoder measurements:

```
%set the amount of time to run tele-op mode
drivetime=20;
%set the IP address to connect
IP_str = '192.168.16.01';
%run the neato in tele-op mode
%(use the game controller to drive Neato around)
%this function returns the encoder measurements as output
measured_data = neato_tele_op(IP_str,drivetime);

%unpack the measured data
%list of times for each of the measurements
tlist =  measured_data(:,1);
%list of encoder measurement of xl
left_wheel_encoder_list = measured_data(:,2);
%list of encoder measurements of xr
right_wheel_encoder_list = measured_data(:,3);
%commanded left wheel velocity
left_wheel_vel_input_list = measured_data(:,4);
%commanded right wheel velocity
right_wheel_vel_input_list = measured_data(:,5);
```

For today's exercise, we will be using this teleoperation interface (the second block of code) to generate the data for our odometry model. If you are working from your dorm and don't have access to a Neato, we have also provided you some example data on Canvas ('neato_data_day_assignmento6.mat') to help you test your code. You can use the code below to load this data into your MATLAB environment:

```
%path where the .mat file is stored
my_path = 'C:\Users\otaylor\Documents\MATLAB\';
%name of the .mat file we want to load
fname = 'neato_data_day_assignment06.mat';
%load the variable recorded_data from the .mat file
%and save it into the variable measured_data
measured_data =load([my_path,fname],'recorded_data').recorded_data;

%unpack measured_data into 5 separate vectors
tlist = measured_data(:,1);
left_wheel_encoder_list = measured_data(:,2);
right_wheel_encoder_list = measured_data(:,3);
left_wheel_vel_input_list = measured_data(:,4);
right_wheel_vel_input_list = measured_data(:,5);
```

Exercise 10.1

Please review the last exercise of the day 3 assignment (exercise 4.3) to make sure you understand how to plot and manipulate the encoder data.

10.1.1 Wheel Velocity

The scalar wheel velocities $v_l(t), v_r(t)$ are the time derivative of the net "forward distance" traveled by the wheels:

$$v_l(t) = \frac{dx_l(t)}{dt}, \qquad v_r(t) = \frac{dx_r(t)}{dt} \tag{10.3}$$

Note that, though we are referring to v_l and v_r as "velocities", they are scalar quantities, not vector quantities. In this case, we call them velocities because they can take on both positive or negative values (as opposed to speed, which is always nonnegative).

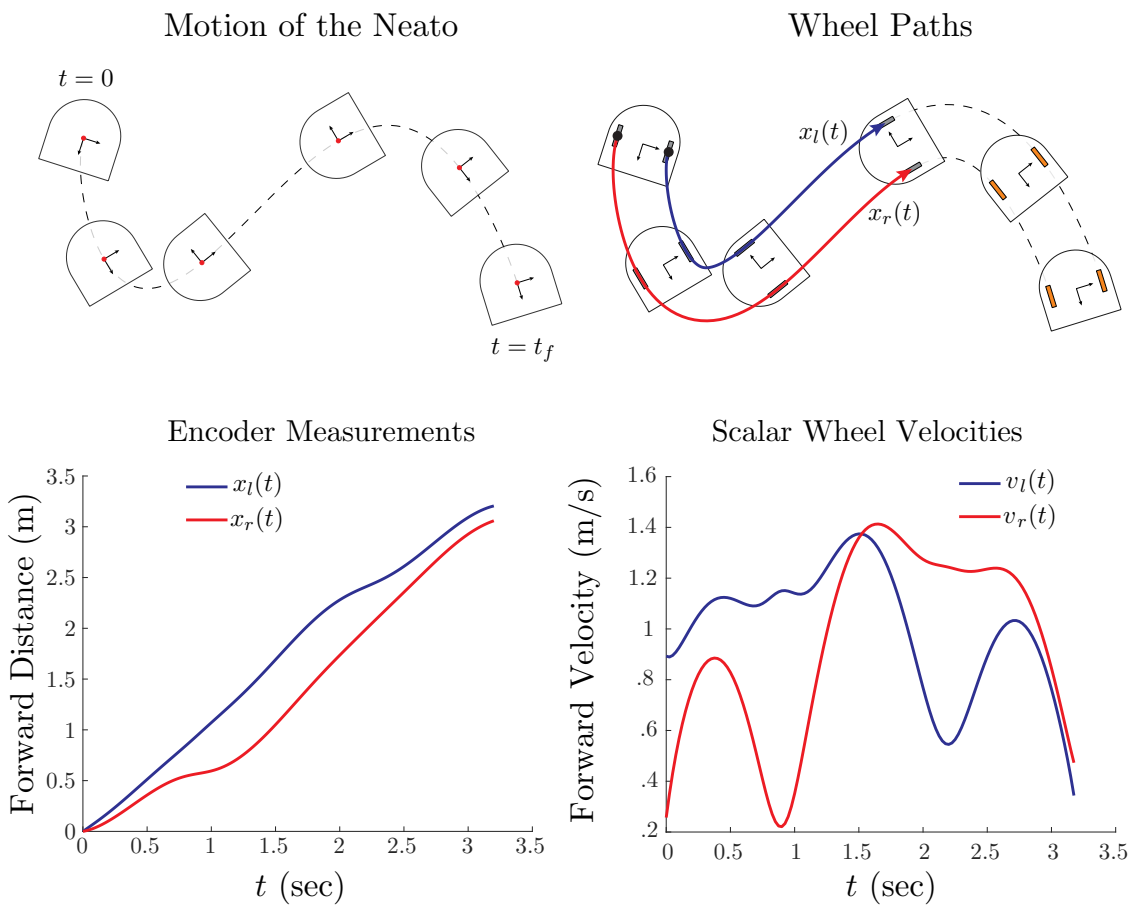


Figure 10.1: Visualization of the Neato traveling along some path. The scalar values $x_l(t)$ and $x_r(t)$ track the motion of each wheel. Differentiating $x_l(t)$ and $x_r(t)$ gives us the scalar wheel velocities, $v_l(t)$ and $v_r(t)$.

Previously, we used the finite difference approximation to estimate $v_l(t)$ and $v_r(t)$:

$$v_l(t_i) \approx \frac{x_l(t_{i+1}) - x_l(t_i)}{t_{i+1} - t_i}, \qquad v_r(t_i) \approx \frac{x_r(t_{i+1}) - x_r(t_i)}{t_{i+1} - t_i} \tag{10.4}$$

Exercise 10.2

Please review the last exercise of the day 3 assignment (exercise 4.3) to make sure you understand how to approximate the scalar wheel velocities from the encoder data in MATLAB.

10.2 Orientation

The rotation rate of the Neato, $\dot{\theta} = \frac{d\theta}{dt}$ depends on the velocity of the left and right wheels (v_l and v_r) as well as the distance between the two wheel, d (which is around 24 cm, or .24 m):

$$\dot{\theta} = \frac{d\theta(t)}{dt} = \frac{(v_r - v_l)}{d} \tag{10.5}$$

We still haven't discussed why the model behind this equation (we are leaving that for next week). For the moment, please accept it as the truth, though we understand if you doubt it. Given some initial orientation $\theta(t = 0) = \theta_0$, we can find the current orientation by integrating the rotation rate:

$$\theta(t_f) = \left(\int_0^{t_f} \dot{\theta} dt \right) + \theta_0 \tag{10.6}$$

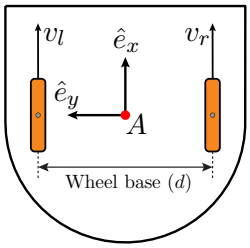
We can use the following Riemann sum to approximate θ:

$$\theta(t_n) \;=\; \theta(t_0) + \int_{t_0}^{t_n} \dot{\theta}(t)dt \;\approx\; \theta(t_0) + \sum_{i=0}^{n-1} \underbrace{(t_{i+1} - t_i)}_{\Delta t} \dot{\theta}(t_i) \tag{10.7}$$

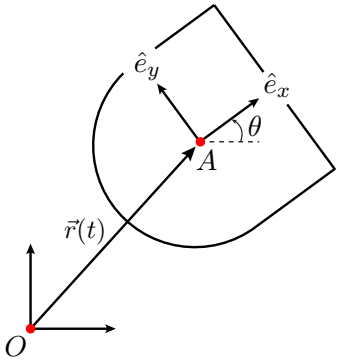
Exercise 10.3

Please review the last exercise of the day 3 assignment (exercise 4.3) to make sure you understand how to approximate the Neato's orientation from the encoder data.

Body-Fixed Unit Vectors



Neato in the World



At this point, we should be caught up to where we were at the end of day 3. Let's now apply what we've learned about parametric curves to approximate the position of the Neato as a function of time, $\vec{r}(t)$. Our process is as follows: we will use the Neato's orientation to compute its forward heading, $\hat{e}_x = \pm\hat{T}$, which is parallel to the unit tangent vector. We can also extract the Neato's speed, $||\vec{v}(t)||$ from the encoder data. The velocity can then be approximated by combining the Neato's heading and speed. Finally, we can integrate the velocity to figure out the Neato's position (relative to its starting location).

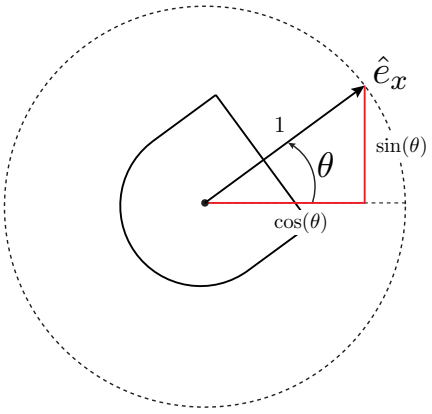
10.2.1 Heading Vector

The Neato's heading vector, $\hat{e}_x$, is a function of its angle, $\theta(t)$:

$$\hat{e}_x = (\cos(\theta(t)), \sin(\theta(t))) \tag{10.8}$$

This is because, as a two dimensional unit vector, $\hat{e}_x$ must live on the unit-circle (visualized below).

Neato Heading Vector



10.3 Velocity

The Neato's speed, $|\vec{v}(t)|$ can be expressed as the average of the left and right scalar wheel velocities:

$$||\vec{v}(t)|| = \left|\frac{v_l(t) + v_r(t)}{2}\right| \tag{10.9}$$

This is another equation that we will be deriving next week, please accept it as truth for now. Previously, we showed that since the Neato can only move forwards or backwards (and not side-to-side), the unit tangent vector of the parametric curve describing the Neato's motion, $\hat{T}$, is either equal or opposite the forward heading vector, $\hat{e}_x$:

$$\hat{T} = \pm\hat{e}_x, \quad \hat{T} = \hat{e}_x \text{ (moving forwards)}, \ \hat{T} = -\hat{e}_x \text{ (moving backwards)} \tag{10.10}$$

The velocity of the Neato is the product of it's magnitude and the tangent vector, $\vec{v} = ||\vec{v}||\hat{T}$. This means that we can combine our expressions for $||\vec{v}||$ and $\hat{T}$ to find the velocity vector:

$$\vec{v}(t) = \frac{v_l(t) + v_r(t)}{2}\hat{e}_x = \left(\frac{v_l(t) + v_r(t)}{2}\cos(\theta(t)), \frac{v_l(t) + v_r(t)}{2}\sin(\theta(t))\right) \tag{10.11}$$

Note that the formulation shown above takes into account the possibility that the Neato might be driving backwards, which is why there are no absolute values in the expression for $\vec{v}(t)$.

Exercise 10.4

1. Write a function or script that uses the encoder measurements to track the Neato's **velocity** as a function of time. You are encouraged to use your code from the previous assignment that keeps track of the Neato's heading as a starting point.
2. Generate plots of the x and y components of the Neato's **velocity** as a function of time. Use data that you collect from running the Neato in teleoperation mode (if you are working from home, you can use the data set that we provided you).

10.4 Position

Previously, we established that velocity is the derivative of position:

$$\vec{v}(t) = \frac{d\vec{r}(t)}{dt} \tag{10.12}$$

It follows from the fundamental theorem of calculus that position must be the integral of velocity:

$$\vec{r}(t_f) = \left(\int_0^{t_f} \vec{v}(t)dt\right) + \vec{r}(0) \tag{10.13}$$

As with $\theta(t)$, we can approximate position from velocity by using a Riemann sum:

$$\vec{r}(t_n) \ = \ \vec{r}(t_0) + \int_{t_0}^{t_n} \vec{v}(t)dt \ \approx \ \vec{r}(t_0) + \sum_{i=0}^{n-1} \underbrace{(t_{i+1} - t_i)}_{\Delta t}\vec{v}(t_i) \tag{10.14}$$

Exercise 10.5

1. Write a function or script that uses the encoder measurements to track the Neato's **position** as a function of time.
2. Generate a plot of the x and y components of the Neato's **position** as a function of time. Use data that you collect from running the Neato in teleoperation mode (if you are working from home, you can use the data set that we provided you). **Note:** If you are testing your estimator

while driving the Neato on carpet, you may run into some accuracy issues. This is likely due to the wheels slipping on the carpet, which means that the amount the Neato actually translates/rotates is less than what is predicted by our odometry model. As such, it is recommended that you drive the Neato on linoleum when testing your estimation script/function.

3. Generate a plot of the x-y path that the Neato traveled.

10.5 Return to the Starting Line

Your task for the rest of this exercise is to write a routine that returns the Neato back to its original starting position/orientation, after driving it around in teleoperation mode. To do this, you will need to figure out how to use the position and orientation estimate to compute a sequence of motor commands that will drive the Neato home. We recommend that you use the 'setVelocities' function to send commands to the wheels:

```
%sets the left/right wheels to drive at vl & vr respectively
%units are in m/s
%vl and vr must be between in the range [-.3 m/s, .3 m/s]
neatov3.setVelocities(vl,vr);
```

Using the setVelocities interface, we can construct two simple motion primitives: move forward and rotate.

10.5.1 Move Forward

If we command the left and right wheels to have the same scalar velocity,

$$v_l = v_0, \qquad v_r = v_0 \tag{10.15}$$

then we can expect the Neato to move forward at a constant rate without rotating:

$$\vec{v} = v_0 \hat{e}_x, \qquad \dot{\theta} = 0 \tag{10.16}$$

10.5.2 Rotate

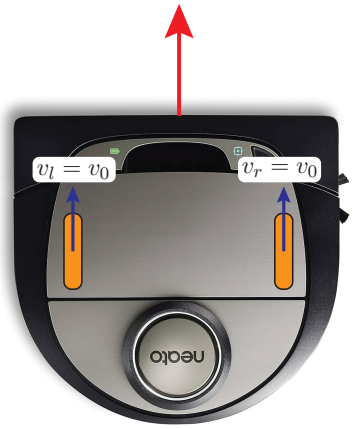
If we command the left and right wheels to have opposite velocities,

$$v_l = -v_0, \qquad v_r = v_0 \tag{10.17}$$

then we can expect the Neato to rotate at a constant rate without any translation:

$$\vec{v} = 0, \qquad \dot{\theta} = \frac{v_r - v_l}{d} = 2\frac{v_0}{d} \tag{10.18}$$

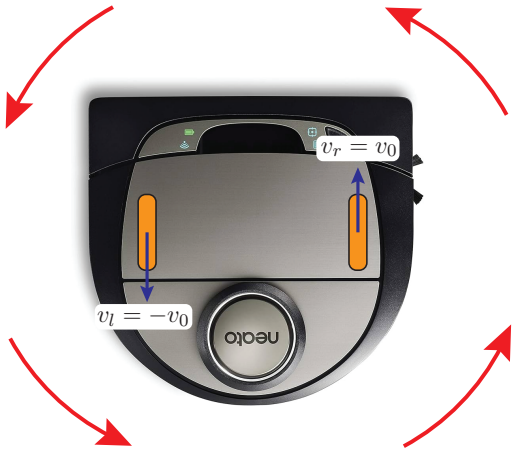
Move Forward



neatov3.setVelocities(v0,v0);

$$v = v_0, \quad \dot{\theta} = 0$$

Rotate



neatov3.setVelocities(-v0,v0);

$$v = 0, \quad \dot{\theta} = 2\frac{v_0}{d}$$

10.5.3 Putting it all Together

Now that we have some basic motion primitives, we can apply what we know to drive the Neato along simple motions. Specifically, if we command the Neato to translate or rotate at a certain rate for a predetermined amount of time, we can get it to move forwards/backwards or rotate clockwise/counterclockwise by a desired amount. Here, we can rely on MATLAB's tic and toc commands to keep track of the amount of time that has elapsed as the program runs. The following template demonstrates how to command the Neato to move forward by some distance l_0 via a combination of the 'setVelocities' interface and tic/toc:

```
%connect to the neato at IP address 192.168.16.01
neatov3.connect('192.168.16.01');

%set the distance we want to travel forward (in the case, 1 meter)
l0 = 1;
%choose a value for the forward velocity
%in this case, .1 m/s for both wheels (highest value you can choose is .3 m/s!)
v0 = .1
%set the left and right wheel velocities ot be v0
vl=v0; vr=v0;
%figure out how long to move the Neato by dividing the length by the forward velocity
move_time = l0/v0;

%set the timer to zero
tic;

%loop until move_time amount of time has passed
while toc<=move_time
    %command the left and right wheel velocities (in meters/second)
    neatov3.setVelocities(vl, vr);

    %query the neato for sensor data (including encoders)
    %doing this adds a delay so the neato won't get overloaded
    %with velocity commands
    sensors = neatov3.receive();
end

%set the wheel velocities to zero
%make sure to do this, otherwise, the Neato may continue moving
```

```
%when you don't want it to!
neatov3.setVelocities(0,0);
%make sure the Neato heard your command by waiting until you get sensor data
sensors = neatov3.receive();

%disconnect from the Neato
neatov3.disconnect();
```

At this point, we should have all the necessary pieces to write the homing routine.

Exercise 10.6

Work together with your table to write a homing routine that drives the Neato back to its starting position and orientation after running in teleoperation mode. Record a video of the Neato executing this routine, and submit it as a YouTube link as part of your homework deliverable (do not upload the video directly to Canvas!). If you are unable to complete this exercise in class, make sure finish it as the last part of your homework assignment. Each table group only needs to create a single video (though if you want to make and submit your own, separate from the group, feel free to do so)

Note: You may run into trouble if you are driving your Neato on carpet. For best results, drive it on linoleum (or some other surface that the wheels don't slip on).