2.12: Introduction to Robotics
Lab 2: ODrive Motor Control*

Spring 2020

Instructions:

1. **When your group is done with each task, call a TA to do your check-off.**
2. **No need to turn in your code or answers to the questions in the handout.**

1 Objectives

In this lab, you will learn how to control **Harmonic Drive motors** using **ODrive motor driver boards**. You will be using the terminal on an Ubuntu 16.04 machine.

1.1 Interacting with the Odrive via Python Class

For this lab, you'll need the serial number of your ODrive. To do so, connect the ODrive to your computer via USB. Then, open a terminal and running the following command:

```
odrivetool shell
```

Take note of the serial number, then you can quit the odrivetool.

Either in a new terminal or the same, clone the lab2 folder from the github

```
git clone https://github.com/mit212/lab2odrive.git
```

Then via terminal using the cd command navigate to be inside the lab2odrive folder. Using a text editor, open the file OdrivePythonClass.py. In this file, we created a ODrive python class for you called OdrivePython. There are several methods which will make configuring, connecting and moving your motor much easier. Please check out the script `OdrivePythonClass.py`.

First, open a python shell within the folder you have your `robot212_OdrivePythonClass.py` file saved, by running:

```
cd lab2odrive
python
```

Begin by importing the dependencies and ODrive class from the ODrive. You can also copy/-paste these directly if you open the OdrivePythonClass.py file in gedit or your favorite python text editor.

*

1. Version 1 - 2020:Rachel Hoffman-Bice, Jerry Ng, Steven Yeung and Kamal Youcef-Toumi

```
import odrive
from odrive.enums import *
import time
import math
import fibre
import serial
import struct
import signal
import sys
import pdb
import matplotlib.pyplot as plt
import numpy
from OdrivePythonClass import OdrivePython
```

Next, we can initialize an OdrivePython instance. Remember the serial number of the motor that we said was important? This is where you need it! To initialize a class you need two numbers. The first is the serial number. The second is the axis that your motor is connected to. **Note:** When assigning your variable to be of the OdrivePython class, it will immediately begin connecting to the assigned serial number and run a full initialization of the motor.

```
#The syntax to be used is as follows:
#mymot=OdrivePython('SERIAL_NUMBER_HERE',AXIS_NUMBER_HERE)
#an example:
mymot=OdrivePython('2086378C3548',1)
```

After a few moments, the ODrive should be found and the initialization process should begin. You should hear a beep and a click like before. A series of statements will print.

```
Found odrive!
(axis error:, '0x0')
(motor error:, '0x0')
(encoder error:, '0x0')
```

These error statements come from using the mymot.printErrorStates() method. This method will be useful for debugging. Once the initialization has started up without any errors, we can begin the tuning process!

2 Tuning the Motor

In this section, you will primarily be using three different functions: `set_gains(Kpp,Kvp,Kvi)`, `PosMoveTuning(position)`, and `VelMoveTuning(velocity)`. For the set gains method, the inputs are a three element array `(Kpp, Kvp, Kvi)`, specifying the proportional gain for the position loop, the proportional and integral gains for the velocity loop, respectively. It should be noted that there is no integral gain for the position loop on the ODrives.

```
mymot.set_gains(1,.0001,0)
```

After setting the gains, the other two commands can be used, such as:

```
mymot.PosMoveTuning(100000)
```

This would bring up the live plotter and display the estimated position of the motor as it moves towards 100000 counts. We can also set the velocity with the following command:

```
mymot.VelMoveTuning(100000)
```

This would set the reference velocity to 100000 [counts/s]. By using the `Tuning` version of these methods, you should see live plots of the position and velocity of the motor. You can change the gains and run the tuning commands to see the step response.

For the controller tuning process, we will first need to set an objective of the controller performance. In this lab, our objective is to **reach a velocity of 1/4 [rev/s] in 2 seconds with no steady state error, and move a relative position of a quarter revolution with no overshoot**.

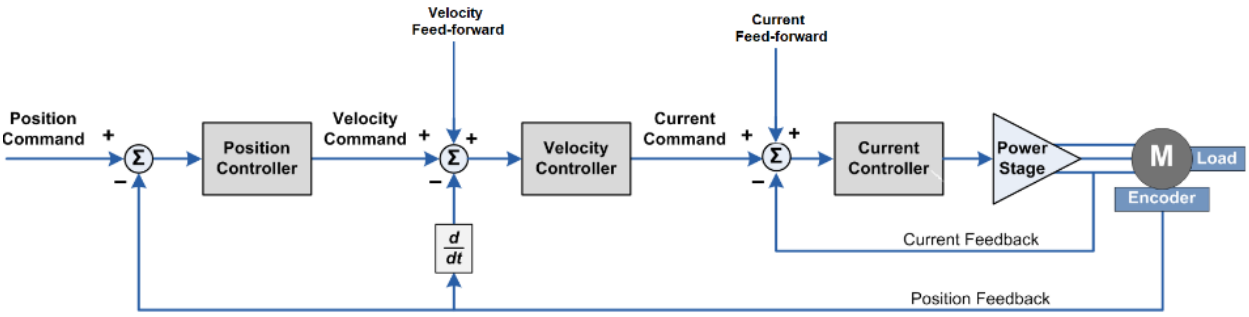

Figure 1: The ODrive Block Diagram. The Position Controller is a P loop, the velocity controller is a PI loop and the current controller is a PI loop.

As you can see from the diagram, there are three loops of control, we will need to tune both the inner velocity control loop and the outer position control loop. We will try different orders to tune the two loops.

2.1 Outer Loop then Inner Loop

2.1.1 Position Tuning

First, let's determine the step response for a small motion. Set the gains to be appropriate value and see what the step response is. Since the motor is incremental, you'll need to first move the motor to it's zero position. As of now, tune the motor so that it **settles to 1/8 of a revolution in 2 seconds**.

Now set gains to be appropriate values using the following command and run `PosMoveTuning` to observe the set response. Replace Kpp, Kvp, and Kvi with the respective gains you are testing

```
mymot.set_gains(Kpp,Kvp,Kvi)

mymot.PosMoveTuning(12500)
```

You'll want to begin with your integral gain set to zero and your derivative gain set to some nonzero small value, and increase your proportional gain first. After achieving some amount of overshoot, you'll want to increase your derivative gain to compensate. For position control for the ODrive, there is no integral gain so you will not have to worry about that for the position controller.

Take note of the parameters of the step response (estimations from the graph are fine) such as rise time and steady state error. After testing your gains thoroughly to achieve the step response you want, try testing if they were for a larger step, like one full revolution.

```
mymot.PosMoveTuning(400000)
```

Question 1 *What is the difference between the two step responses? What might cause this difference?*

2.1.2 Velocity Tuning

Now that you've gotten a feel for tuning a position controller, we will move onto velocity control. Similar to before, you'll be using the `set_gains` function to tune your gains. However, since it is the velocity controller, the proportional position gain is irrelevant, and you'll be trying to tune the velocity gains. Again, you can use this command to see the response.

```
mymot.VelMoveTuning(400000)
```

As mentioned earlier, the objective for the velocity controller is to control the motor with a **step response with minimum steady state error and reaches a velocity of 1/4 [rev/s] in 2 seconds.**

2.2 Inner Loop then Outer Loop

Now that we have tuned the system for both of these two different controllers, it is important to note that the order of tuning matters. As such we wish to tune the inner loop and then the outer loop so that the gains should work for both types of control, position and velocity. This matters a lot for when trajectory control is used as the order in which the different dynamics of the system converges is important.

To that end, the following tuning procedure can be done such that you have gains that work in a variety of circumstances. This was adopted from the ODrive recommended tuning procedure [1]. It should be noted that the original tuning guide **assumes you are tuning to track a sinusoidal trajectory for position**.

Rather than tracking a sinusoidal trajectory, recall the task for this lab is to create a controller that can do two things: **Reach a velocity of 1/4 [Rev/Sec] in 2 seconds with no steady state error, and move a relative position of a quarter revolution with no overshoot**.

2.2.1 Velocity Tuning

1. Set vel_integrator_gain to 0 and pos_gain to 1
2. Make sure you have a stable system. If it is not, decrease all gains until you have one.
3. Increase vel_gain by around 30% per iteration until the motor exhibits some vibration.
4. Back down vel_gain to 50% of the vibrating value.
5. The vel_integrator_gain can be set to 0.5 * bandwidth * vel_gain, where bandwidth is the overall resulting tracking bandwidth of your system. Say your tuning made it track commands with a settling time of 100ms (the time from when the setpoint changes to when the system arrives at the new setpoint); this means the bandwidth was 1/(100ms) = 1/(0.1s) = 10hz. In this case you should set the vel_integrator_gain = 0.5 * 10 * vel_gain.

2.2.2 Position Tuning

1. Increase pos_gain by around 30% per iteration until you see some overshoot.
2. Back down pos_gain until you do not have overshoot anymore.

It is worth noting that in general, gains are tuned to gain specific responses as the system is inherently nonlinear due to friction and a variety of other factors. Make sure you take note of these final values so that you can use them for future labs (or at least use them as a starting point for tuning in future labs).

Question 2 *How could you compensate for the nonlinear effects of friction (Coulomb friction, and the transition from static to kinetic friction)? Will it matter for this project?*

Question 3 *How would you tune the gains on your controller after you've constructed your robot? What would you pick as your initial gains?*

References

[1] Getting started with odrive. [Online]. Available: https://docs.odriverobotics.com