Lab 1: Move the Mobile Robot Wheels: Motor Velocity Control
2.12: Introduction to Robotics
Fall 2016

Peter Yu, Ryan Fish, and Kamal Youcef-Toumi

Instructions:

1. **When your team is done with each task, call a TA to do your check-off.**

2. **Backup your files to reuse them in the future: email, flash drive, etc.**

3. **No need to turn in your code or answers to the questions in the handout.**

1   Introduction

The goal of today's lab is to learn about the hardware and software required to implement a wheel velocity controller. In this lab, you will:

1. Learn about the robot, motors, encoders, the motor drivers, the encoder buffers, the Arduino.

2. Implement both open-loop and closed-loop velocity control.

3. Use the controller to track a sinusoidal velocity profile.

2   The Robot Hardware

2.1   Mobile Robot

The mobile robot has two brushed DC motors with gearboxes that drive the wheels. The third wheel is a caster wheel. There is a **two-channel motor driver** to power the motors, which is controlled with PWM (Pulse-width modulation) signals from an Arduino microcontroller. This motor controller board can be found in the form of an Arduino Shield attached to the microcontroller. Each DC motor has an incremental magnetic quadrature encoder for angular position feedback. An estimate of angular velocity for feedback is obtained from this angular position. See Figure 1.

The encoder signal is too fast for the Arduino to read and track in real-time while also performing control and communication, so a separate **Encoder Buffer Board** is used to keep track of the encoder count. The Arduino can be programmed to receive the current encoder count from the Encoder Buffer Board, and use that information to calculate how much the wheel has rotated.

The **Arduino Uno** is plugged into an on-board computer via USB. The computer has a 4-thread 3.8 GHz i3 processor which runs Ubuntu and Windows 7. The PC can be connected to USB devices such as webcams, Kinect depth sensors, and Dynamixel smart servos.
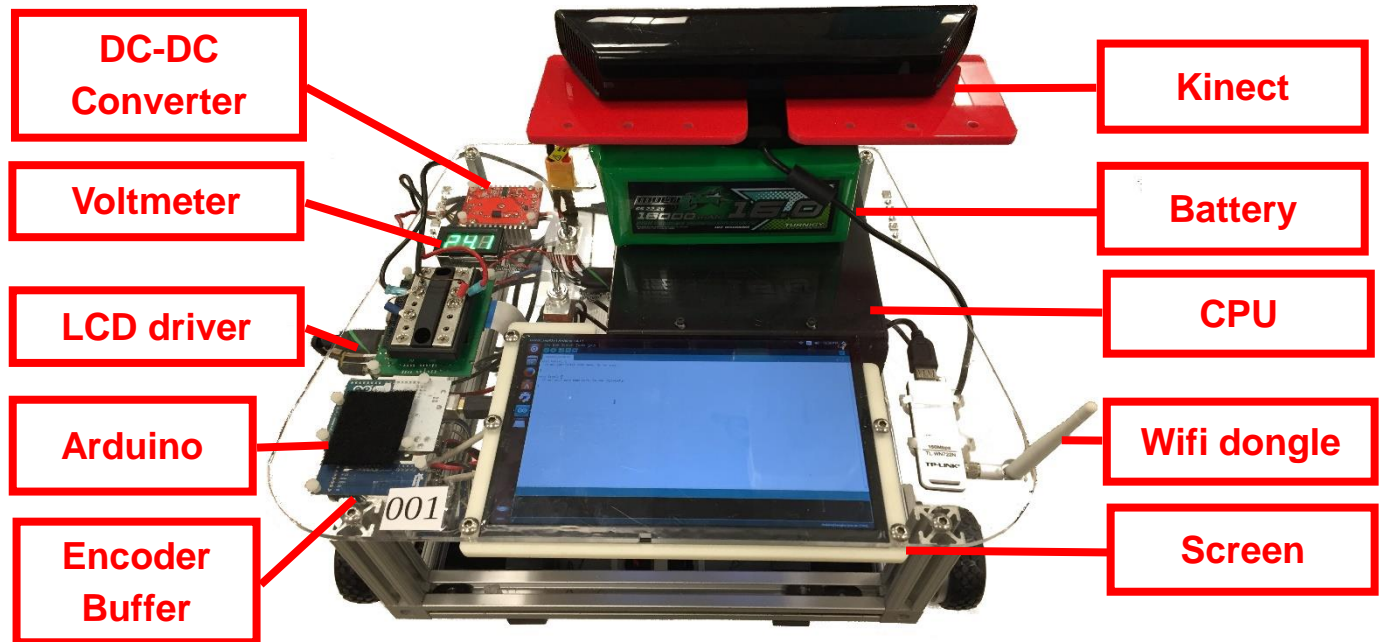
Figure 1: Our mobile robot platform.

2.2   Lithium Polymer (LiPo) Battery Safety

This battery is made up of 6 cells in series and has 16Ah of electrical charge (Figure 2). The average battery voltage is 22.20V, and can be charged up to 25.20V. Five safety rules must be followed:

- When the battery is close to 20V, please save files and turn off the onboard computer correctly, and ask TA to exchange for a charged battery.

- Keep the battery monitor alarm plugged into the battery at all times. It will warn you very loudly if the battery is not behaving as expected.

- If you feel that something is wrong with the battery ask a lab staff. In case of emergency try to unplug the battery and put the battery into a sand bucket in lab.

- Disconnect the battery from the robot when you're done with the lab.

- Turn off the battery switch on the robot when modifying electronic hardware.

Failure to obey the safety rules is considered failing the lab.

LiPo batteries are some of the most energy dense forms of electronic power storage, but they are also highly volatile and can catch fire, release toxic chemicals, and explode if you are careless with them. **LiPo will be damaged if you**

- short the + and − wires of it,

- overcharge it past 25.20 V,

- discharge it below 18.00 V,

- puncture it,

- use it with the cells unbalanced, or

- squeeze it too hard.

Additional rules:

- Use a safety pouch to charge the battery.

- To check voltages, use a small battery meter to check it. Ask a lab staff for details.

Figure 2: The LiPo battery on our mobile robots.

3   The Arduino Uno

The Arduino Uno is a microcontroller for prototyping. Low-level real-time control software is often written on a microcontroller because it ensures proper control loop timing and provides an immediate interface with electronics and sensors such as the Motor Driver and Encoder Buffer. Though a microcontroller typically processes data more slowly than a desktop computer and doesn't have as much memory, a micro can react to events more quickly since its software is less complex. The Arduino is a good choice for our robot, since the Encoder Buffers put out a digital signal, and the motor driver shield accepts PWM input.

The Arduino communicates with a computer through a USB port. We can use this port to upload programs to the Arduino, as well as send and receive data between the Arduino program and a computer program. Eventually, we will use this connection to send motion commands from the computer to the Arduino, and send position information from the Arduino to the computer.

4   Using the system

4.1   Turning on

Make sure that:

- your robot's wheels are off the ground using a textbook or chunk of 80/20,

- the **Main Power** switch and the **Motor Enable** switch are set to the off position, and

- your battery is plugged in and securely attached to the top platform of your robot with Velcro.

Turn on the **Main Power** switch and then various components should get powered. Check that the battery voltage is above 20V, and inform a lab staff if it is not.

Turn on your computer using the power button on computer, located on the right side of the robot in Figure 1. Then you can choose to use Ubuntu or Windows system. We will use Ubuntu as the main operating system for all the lab this semester.

Normally the system will login automatically on boot. Username and password are required when you need to install new software. You can find the login info near the battery charging area.

4.2   Using Arduino

Open the `controller.ino` file in the lab1 folder on the computer desktop. The file will be opened with the Arduino IDE (Integrated Developer Environment), as shown in Figure 3. This is the main control code that will be uploaded into the Arduino.

In order to upload code to the Arduino, make sure the Arduino is plugged into the computer and that the proper Board (Arduino Uno) and COM Port (The IDE typically tells you which port is connected to the Uno) are selected in the Tools tab of the IDE. Then, by clicking the icon ⬆, your code will be compiled and uploaded to the Arduino. If there is any error in your code, the code will not compile and you will have to fix them before continuing. The error line numbers are shown in the message display.
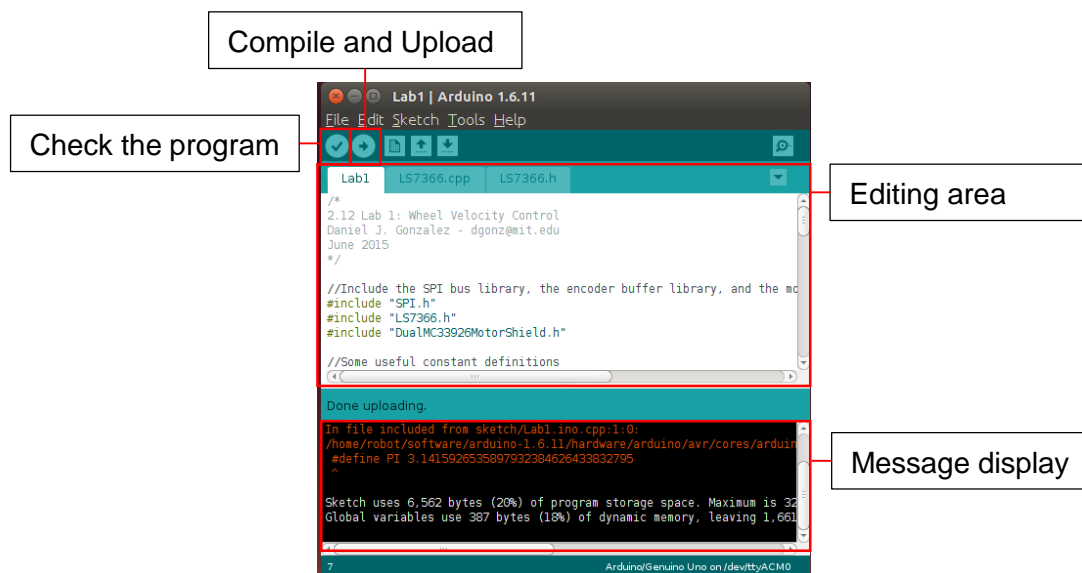

Figure 3: A screenshot of the Arduino IDE.

5   Task 1: Open-Loop Velocity Control

5.1   Downloading the Code

Download the `lab1.zip` file from course website: `http://robot2016.mit.edu/` to, for example, Desktop. Extract the lab1 folder in zip file on Desktop. From now on, we will denote the path to the lab1 folder `~/Desktop/lab1` as `LAB1`. `~` is an alias for user's home directory in Ubuntu, which

is `/home/robot/` in our case. The `lab1` folder contains all the codes required for this lab. It is structured as follows:

- `src` : source folder

  - `controller` : Arduino source code folder
    * `controller.ino` : top level controller code
    * `lab1_funcs.h` : declarations of the key functions
    * `lab1_funcs.cpp` : the skeleton of key functions you will be filling in today
    * `helper.h` : declaration of helper constants and functions. You should check the variable names it provides to you before programming anything.
    * `helper.cpp` : definition (values) of helper constants and functions
    * `LS7366.h` : encoder buffer communication API declaration
    * `LS7366.cpp` : encoder buffer communication API implementation

- `viz` :

  - `plot_controller.py` : visualization of controller signals

- `test`

  - `testEncoderSketch.ino` : for checking encoder buffer hardware and connections

You will find `controller.ino` has a section labelled STUDENT CODE starting at line 61. This file contains the high level structure of the controller. That includes calling functions to find the actual velocity, find the desired velocity, then compute the motor commands, and finally send the commands. The detailed implementation of those components resides in `lab1_func.cpp`.

In this lab, you will be mainly filling out `lab1_func.cpp`. Note that many helpful variables have been declared and defined in the supporting files `helper.h` and `helper.cpp`. This is to speed up the lab and keep code organized. So in general, you don't need to define any variables. For each function in `lab1_func.cpp`, you are asked to fill in code using relevant variables.

5.2 Motor specifications

The motor drivers provide some average voltage to the motors as a percentage of the total voltage applied to the motors; for us, this is the battery voltage ∼22.2V. This percentage is determined with the PWM duty cycle that we set in the Arduino using the `setM1Speed()` function of the `DualMC33926MotorShield` object. We name this `DualMC33926MotorShield` object as `md`, in the STUDENT CODE section of `controller.ino`. Function `setM1Speed()` takes a command within [-400, 400], corresponding to -100% and 100% voltage. The motors have a voltage constant of 26.94 rad/s/V and the wheel radius is 0.037 m. There are two types of gearboxes on the robots: ones labeled with 53 have a gear reduction ratio of 1/53, and others labeled with 26 have a 1/26 gear reduction ratio. The encoder is 20 ticks for model 53 and 80 ticks for model 26 per revolution inside the motor before the gear reduction. So then,

Question 1 *How many ticks per revolution of the wheel? What is the max speed of the robot? See the* `helper.h` *for* `enc2robotvel` *and* `maxMV` *conversions.*

5.3 Open Loop Controller

Fill out the `openLoopControl()` function in `lab1_funcs.cpp` that calculates and commands the motor to a desired forward velocity. Note that the range of this command input is [-400, 400] within the code. The maximum achievable velocity of the robots is about 0.41 m s^{-1} or 0.82 m s^{-1} for motor model 53 and motor model 26 respectively. This maximum speed is estimated as follows

$$V_{in} \times K_v \times G \times R = 22V \times 26.94 \text{rad/s/V} \times 1/53 \times 0.037\text{m} = 0.41\text{m s}^{-1},$$

where V_{in} is the input voltage, K_v is the rotation speed to voltage constant of the motor, G is the gear ratio, and R is the wheel radius. Upload your code to the Arduino and flip on the Motor Enable switch. Observe the motor turning as commanded.

5.4 Signal Visualization

A Python script `plot_controller.py` is provided to plot desired and actual wheel velocities, as well as the command input in realtime. This script is in LAB1/src/viz folder. Remember that `LAB1` is the path to the lab1 folder at `~/Desktop/lab1`. Open a terminal (Ctrl-Alt-T) in Ubuntu, and type the following commands (without the leading $).

```
$ cd LAB1/src/viz
$ python plot_controller.py
```

The Python script receives data from Arduino through serial connection. It plots the desired wheel velocity and the motor command input. In Section 6, we will calculate the wheel velocity and use this plotting script as well. **Make sure to close the plot by closing the plot window before reprogramming the Arduino.**

6 Task 2: Closed-Loop Velocity Control

You will notice that with only open-loop control, your velocity will be inaccurate if you apply some damping load to the wheel by either grabbing it or putting it on the ground. The battery's nonlinear behavior as it is being discharged will lead to inaccuracies in maintaining a voltage output. We can fix this by measuring the wheel's actual velocity and using it as part of the controller to track our desired velocity.

6.1 Timing

First, proper control loop timing must be achieved. The `loop()` function in `controller.ino` contains an `if` statement that checks whether or not it is the right time to execute a control loop. This time is determined by the constants defined in `helper.h`.

A constant named `PERIOD` is defined in the `helper.h`. The `dt` variable is defined as well based on this `PERIOD`. The period for the loop is set to 0.0005 seconds, so is `dt`.

6.2 Velocity Estimation

The encoder counts measure the instantaneous position of the motor. The reading from encoder 1 is done in `controller.ino`:

```
encoder1Count = readEncoder(1);
```

Now you must take this a step further and use it to estimate the motor velocity in `motor1_velocity()` in `lab1_funcs.cpp`. First, find the rate of change in encoder steps over time by discrete differentiation.

```
float stepVelocity = (encoder1count - encoder1countPrev) / dt;
```

Then you need to convert this to a robot velocity (m/s) using the given constants such as `enc2robotvel`. Note that you need to store `encoder1countPrev` by completing the first part of `storeOldVals()` in `lab1_funcs.cpp`.

Make sure to check the signedness of the encoder counting. If you are setting the motors to a positive value, you will expect the encoders to read back increasingly positive numbers. Use the visualizer script to determine if the wheel velocity matches the sign you expect, and change sign that multiplies `readEncoder()` in `controller.ino` if required.

After you are done with implementing velocity estimation, try setting different `desiredMV1` in `controller.ino`, and then use the visualizer to see whether or not the feedforward signal is enough to achieve this velocity.

6.3 Proportional Control (Denoted as P)

Now, we implement the proportional controller by generating a motor command within [-400, 400] that is proportional to our wheel velocity error. We subtract the wheel's actual velocity from our desired wheel velocity to get our error term, and multiply it by a proportional gain in the function `proportional_control()` which needs completion. A proper gain is at around 1000. We sum our control commands (only proportional for now) in our new `closedLoopController()`, and then feed the resulting command into `md.setM1Speed()` in `controller.ino`. Make sure to set `useClosedLoopControl` to `true` in `controller.ino`. This command should be limited within [-400, 400] using the `constrain()` function.

Question 2 *What happens to the motor speed response when you increase/decrease the proportional control gain? What happens to the command input?*

6.4 Adding Derivative Control (Denoted as D)

Adding derivative control can help improve our closed loop dynamic response. Store the previous error value by completing the second part of `storeOldVals()` so we can use it with the new wheel velocity error. We subtract the previous wheel velocity error from the current wheel velocity error to get the change in wheel velocity error, which we then divide by `dt` to approximate its discrete, multiply by the derivative gain, and add to the total motor velocity command. Complete these steps in `derivative_control()` and add it to `closedLoopController()` in `lab1_funcs.cpp`.

Question 3 *What happens to the motor response when the derivative control gain is increased/decreased? Comment on the rise time, overshoot, etc of the response.*

6.5 Adding Integral Control (Denoted as I)

Adding integral control can help eliminate any steady state error present from unmodeled damping or disturbances in the system. When the program starts, the integrated velocity error is 0. With each iteration of the control loop, the motor velocity error is multiplied by `dt` and then added to the integrated velocity error. This integrated velocity error is multiplied by the integral

gain, which is tuned manually, and then added to the total motor velocity command. Complete `integral_control()` and add it to `closedLoopController()` in `lab1_funcs.cpp`.

The command due to integral should be limited using the `constrain()` function.

Question 4 *What's the effect of integral control on the wheel velocity response? Why is it important to constrain the integration term? Include in your discussion the steady state error, and any other effects.*

7   Task 3: Sinusoidal Velocity Profile Tracking

One way to test the system's limit is to consider the desired wheel velocity change over time and see how the controller tracks it. A sinusoidal signal can serve this purpose.

In `controller.ino`, you can set `desiredMV1` to a sinusoid function `sinWave` by setting `useSineInput` to `true`.

Modify the variable `sinFreq` in `helper.cpp` in the range of 0.1 to 10 Hz and see how well the controller tracks the desired velocity signal. Modify the PID gains and try to achieve tighter tracking. If the controller becomes unstable, try modifying the gains. An example plotting of the controller for this task is shown in Figure 4.
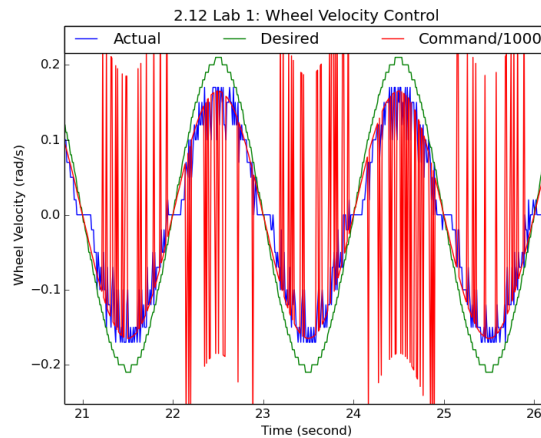

Figure 4: Example sine wave tracking result. Green: desired velocity, Blue: actual velocity, Red: motor commands.