Lab 4 - Motor Drivers and Inverse Kinematics

Overview

This lab walks you through operating your motor drivers and the basics of inverse kinematics. Driving the SCUTTLE requires powering your Raspberry Pi and your motor driver using the battery pack, verifying signal wires from the Pi to the motor driver, and understanding the PWM output from the motor drivers. You will see how commands get converted from robot speed to wheel speed then to PWM commands to voltage which translates to actual wheel movement.

Resources:
The following are links to useful resources for this lab:

- MXET300-SCUTTLE GitHub
- MXET-300 Wire Guide
- Kinematics guide
- SCUTTLE homepage
- SCUTTLE - Assembly & Wiring Guide
- Guides and resources

Software Needed:

- Python scripts:
  - L1_motor.py
  - L2_inverse_kinematics.py
  - L2_speed_control.py
  - L3_noderedControl.py
- Templates:
  - L3_path_template.py
- Node-RED flows:
  - flow_joystickControl.json

Wiring and Running the Motors

Recall that L1 programs receive or send information from devices and are typically used to build more complex programs. In this case, an L1 motor program allows you to control the power of a motor by varying the PWM duty cycle it receives. Since GPIO on the Pi cannot support driving motors due to their high current demand and noisy electrical characteristics, a device called an H-bridge is used to drive the motors based on an input signal using current sourced by the battery pack.

1. In this lab, you will complete the assembly of your SCUTTLE. After some assembly instructions have been given, begin assembling the components onto the chassis with **the battery pack powered off**.
   a. It is also advised you disconnect your wiring, then place the battery pack, Pi, and motor driver bracket on the DIN rail **BEFORE** you do any wiring. This will prevent potential damage to connectors.

NOTE: The following link will open a YouTube video that you can watch for an in depth start to finish assembly guide. SCUTTLE - Assembly & Wiring Guide

2. Connect the L298N, which is the H-bridge used to drive the SCUTTLE motors.
   a. Two of the motor cables will be used to connect motors to the H-bridge and the third cable connects to the battery pack. Connect the motor cables to the two blue outer terminals of the H-bridge and the battery cable to the center terminal with three inputs. These are shown on the bottom of the H-bridge in figure 1. Make sure to match the color of your cables with those shown. Also ensure you are using the correct two inputs for the power terminal; one will have no connection.
   b. Then, connect the PWM signal cable to the pins on the opposite end of the H-bridge from the power supply. Open the Pi's enclosure and connect the pins to the GPIO header as shown in figure 1. The physical pin numbers are 11, 12, 15, and 16. They should each connect to a green, blue, violet, and grey cable respectively. Double check this step since it's easy to mix up the small pins.

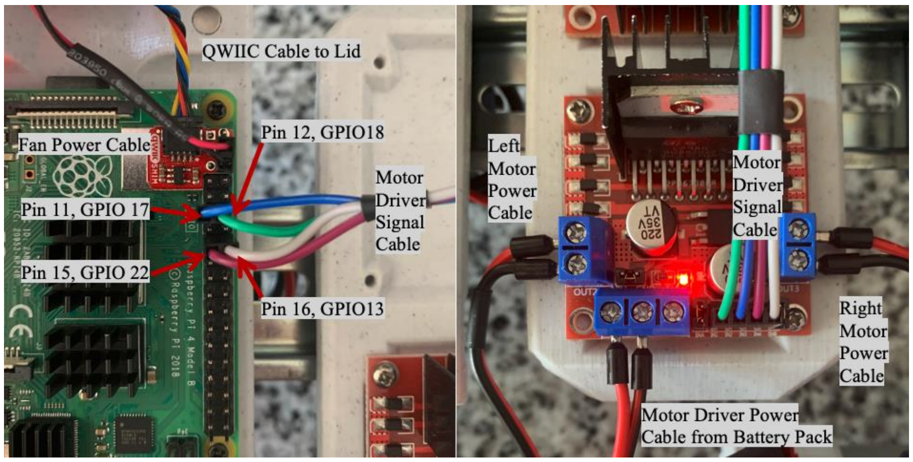


Figure 1: GPIO and H-Bridge pin connections

3. Once your motors and H-bridge are set up, ask the TA to check your wiring **before** powering your Pi back on. Your completely assembled SCUTTLE should look like it does in figure 2.

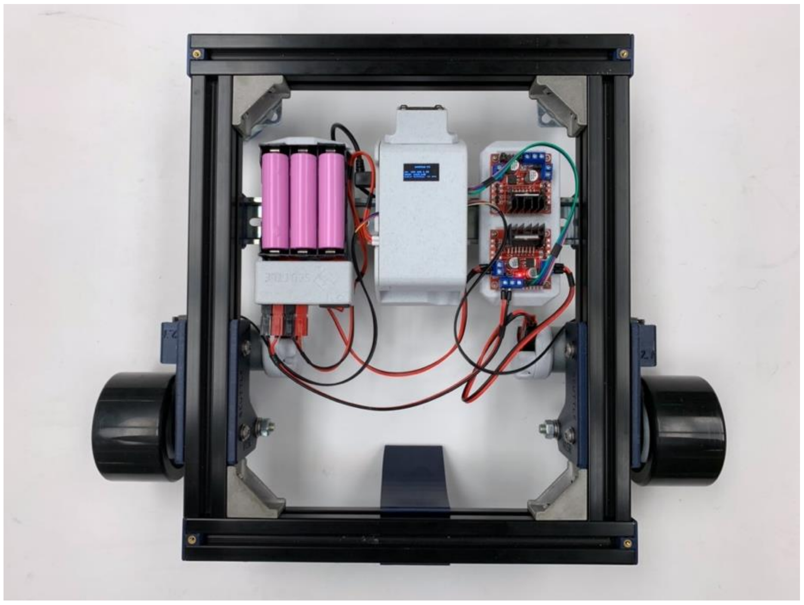


Figure 2: Complete SCUTTLE

4. After confirming the setup and booting the Pi, use the `wget` command in the MXET300-SCUTTLE *basics* directory to download the following files into your basics folder within your Pi:
    a. L1_motor.py
    b. L2_inverse_kinematics.py
    c. L2_speed_control.py
    d. L3_noderedControl.py

    Remember to click “Raw” to get the plain-text version of the code when using `wget`.

5. **Ensure your SCUTTLE is mounted on its stand and the back wheels are lifted to avoid driving off the table.**
6. Now to test your motors, while within your basics folder, run the command `python3 L1_motor.py`
    a. The terminal should show some output that the motors are spinning.
    b. Ensure the wheels turn in the correct direction indicated by the terminal output. If they do not you need to check your wiring. Either the polarity of the motor power output is reversed or the PWM signal cable for that motor needs to be flipped.
    c. For future reference, it is not recommended to change the direction of your motors (inverting the sign of the duty cycle) in software to correct issues like this. Functional software should remain as-is, when possible, to simplify troubleshooting.
7. Ask the TA to check the completion of this step to ensure everything works before proceeding.

Perform Inverse Kinematics

Now that the motors spin, the SCUTTLE can be driven using inverse kinematics. This is the process where target speeds for the chassis are converted to necessary speeds for the motors. You will input a series of forward and rotational velocities for the SCUTTLE chassis to follow and the program will calculate the necessary wheel speeds to reach those targets.

Keep in mind that without a device to measure the wheel speeds your SCUTTLE will not actually follow the exact path given. This is called open-loop control, since there's no feedback on whether or not your actuators actually reach their targets. In the next lab we will explore using encoders and, in the future, implementing them for closed-loop control.

1. Use the command `python3 L2_inverse_kinematics.py`
   a. You will be prompted to enter x dot and theta dot values, which are the forward and angular velocities of the chassis.
   b. The program returns the left and right wheel speeds as pdl and pdr, or phi dot left and right.
   c. This demonstrates the primary purpose of this script: to convert desired chassis speeds to wheel speeds.

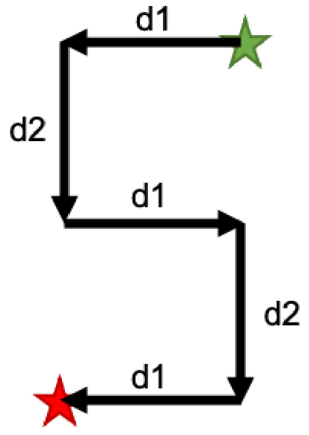


Figure 3: "S" for SCUTTLE Path

2. Now you will prepare a series of motion instructions for the SCUTTLE to follow to create the S path shown above. Your SCUTTLE will begin with the green star and end at the red star.
   a. First, define lengths for d1 and d2. Something less than a meter is reasonable.
   b. Then, fill in table 1 below with the chassis speeds and step durations necessary to complete each motion of the S path. You should start by selecting a duration for the motion and then determine the speed necessary to displace in that time, or vice-versa. The **top speed** of the SCUTTLE is approximately **0.4 m/s**, so constrain your forward velocity to this range. Be sure to include your d1 and d2 values as well as your completed table in your lab report.
   c. Keep in mind that we are assuming the robot reaches these target speeds, so our path may not exactly reflect the planned one.

   d. Also, understand that a right turn in relation to SCUTTLE's 1st person point of view is a negative $\dot{\theta}$. This would be a clock-wise turn from a bird's eye view. The same as you are looking at the path in figure 2.

d1 = d2 =

Motion	$\dot{x}\ (m/s)$	$\dot{\theta}\ (rad/s)$	Duration (s)
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Table 1: Velocity Data

3. Once you have completed table 1, navigate to the templates directory in the MXET300-SCUTTLE GitHub and `wget` the *L3_path_template.py* file into your basics folder.
   a. You will now use the template file to have SCUTTLE drive the path. Pay attention to how the code is written and note the comments. You may need to add some additional lines of code in order to create the full path. Ask your TA if you need some guidance.
   b. Ensure your SCUTTLE is still propped up on its stand when completing this task otherwise it could drive off and hit something. You can observe how the wheels turn freely in mid-air to determine how it is making the path.
   c. When you are ready to try out your code, use `python3` to run it from the terminal.
4. When you have completed making SCUTTLE drive the path with the *L3_path_template.py* file and ask the TA to check your work. Move on to the next task once the TA has verified that SCUTTLE can drive the path.
   a. You may demo this program on the floor or a stand, since the robot will not follow the path exactly without feedback.

Node-RED Control

Now for some fun! You will drive the SCUTTLE manually using a joystick accessible on the Node-RED dashboard. You should have already downloaded *L3_noderedControl.py* from GitHub, which receives joystick inputs from Node-RED to drive the motors. Now you will need to download and import the Node-RED flow called *flow_joystickControl.json*. Follow these instructions to prepare the SCUTTLE for Node-RED control.

1. Navigate to the inside of your "Node-RED_Flows" folder you created in Lab 2 and then `wget` the *flow_joystickControl.json* file from the nodered directory within the MXET300-SCUTTLE GitHub.
2. In the VS Code terminal, run the *L3_noderedControl.py* script using python3.

3. Now, navigate to Node-RED and import the flow as a new flow, then deploy and open the flow within the dashboard. You should see a red circle within a red circle, this is the GUI joystick that you can use to move your SCUTTLE.
4. You can now set SCUTTLE down on the ground and have some fun driving it around.
5. If you want to drive SCUTTLE using your mobile device, connect it to the lab's network. Remember the network is hidden, so you will need to enter the SSID and password.
    a. SSID: mxet_Lab
    b. Password: mxetlab006
    c. Alternatively, you can drive by accessing the Node-RED flow on your laptop or connect the Pi to your mobile device's hotspot.
6. Now connect your phone to the same network as the SCUTTLE, go to the dashboard webpage (*192.168.1.1##:1880/ui/*), and try driving.
7. If all of these steps work correctly, show the TA to get your final checkoff.

In lab 2, you wrote voltage measurements to a file for Node-RED to read in and display. This is a simple method, but it is relatively slow since a write/read to memory is much quicker than write/read to storage. In this case the roles are also reversed: Node-RED must send some data out and a Python script needs to read it in. There is a node for writing a file out from a Node-RED flow, but we want our driving to be low latency so this option isn't ideal. Therefore, we will use a "UDP out" node to send the joystick values from the dashboard to a specific port and IP address.

Node-RED runs on the Pi and so does the L3 driving script, so the node will send data to 127.0.0.1, which addresses the local host and allows the device to "talk to itself" through its own network interface. The joystick values, sent as UDP packets to a specific port, are then received by the L3 Python program, which is written to receive data on that port. The data is then unpacked to extract the joystick position for driving.

Commit and Push to GitHub

Don't forget to commit your changes and push them to GitHub after every lab. Add the new Python and Node-RED files to the commit before pushing so everything is backed up.

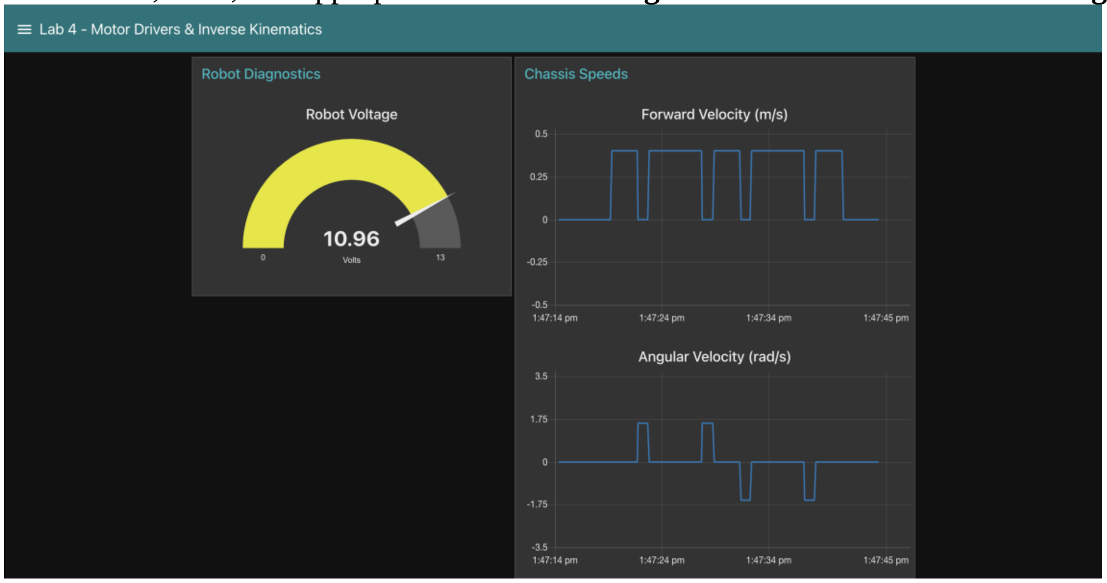
1. Use the *git status* command to see the current state of your local repository.
2. **Ensure you are in the mxet300_lab directory folder.** Use the command `git add .` to track the changes made to files in the directory and add them to the staging area.
    a. Make sure you include a period after `git add`, which indicates that the command applies to all contents in the directory.
    b. If you have files you do not want to commit, such as logs or your own work separate from the lab, you can just use `git add <name>` where name is the file you want to stage for commit. This way you can add each file you want individually and ignore the ones you don't.
3. Commit the changes using: `git commit -m "YOUR COMMIT NOTE HERE"`
4. Finally, use `git push` to push all commits to your remote repository on GitHub.

Post-Lab Questions

1. Why won't the SCUTTLE, as configured in this lab, follow the planned path exactly?

2. Go through your *L3_path_template.py* file script for completing the S path and briefly explain how the code is working.

3. Modify the L3_path_template script to log $\dot{x}$ and $\dot{\theta}$. Then create a flow to display these velocities in a chart while the program runs.
   a) Attach a screenshot of the flow with data included. Make sure to configure the charts with titles, units, and appropriate bounds. **The figure would look like the following.**



   b) Optionally, you could log $\dot{x}$ and $\dot{\theta}$ from Node-RED Joystick. You can use "file in" nodes to read this information or optionally try using "UDP in" nodes. For the former, you can log values from L3_noderedControl script. For the latter you can refer to the joystick flow and L3 driving script for examples on using sockets to communicate via network interface.