Lab 5 - Encoders and Forward Kinematics

Overview

This lab introduces encoders, the use of I2C for communicating with sensors, and forward kinematics. Encoders provide information about a component's rotation along an axis. The IC, using a Hall effect sensor and analog-to-digital processing, determines the absolute position of the shaft's rotation angle by measuring the changing magnetic field of a magnet embedded in the shaft. You will also be exposed to the I2C protocol for communicating with the encoders and other devices. Finally, you will write an L3 program to grab the wheel speed data and chassis speed data **[x_dot, theta_dot]** to display on a Node-RED dashboard.

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
- I2C - SparkFun Learn

Software Needed:

- Python scripts:
  - L1_encoder.py
  - L2_kinematics.py

Wiring and Running the Encoders

Absolute rotary encoders are used on the SCUTTLE to measure the position of the wheels. The sensor accomplishes this by measuring the change in the magnetic field of a magnet fixed to the wheel's axle. Software can then calculate angular velocity using time elapsed between two position measurements. The Pi communicates with its sensors using I2C, or inter-integrated circuit, which allows a single bus to chain devices together for communication.

1. Within your MXET-300 Lab Kit, there are two 300 mm long QWIIC cables. You will use these cables to connect your encoders to the Pi, if you have not done so already. **Ensure your robot is turned off for connecting these parts.**
2. Underneath the enclosure lid is a red component labeled QWIIC Multiport that looks like a plus sign. Connect the two encoder cables to this breakout board, as shown by the green circled ports in figure 1. The left encoder cable going to the left encoder, will connect to the left side and the right encoder cable going to the right encoder, will connect to the right side.
    a. **Insert the encoder cable straight into the port.** If you insert it at an angle, the pins inside the port would be crushed.
    b. The QWIIC Multiport acts as an I2C bridge to connect several devices to the bus in one place. This is one of the advantages of I2C.

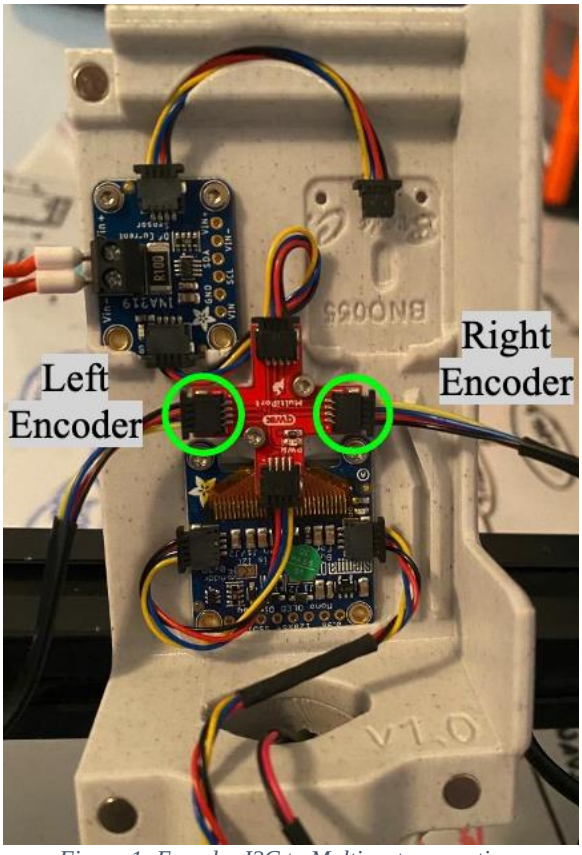

Figure 1: Encoder I2C to Multiport connections

3. Next you can connect the encoders. Two QWIIC adaptor boards on the encoders are used to convert the eight Dupont connectors on the AS5048 encoder to a much smaller JST connector for the I2C bus to connect to. The JST cables only go in one way, if you cannot insert the connector in, try flipping it around and try again.
    a. The I2C breakout boards should be oriented with the flat side facing away from the motors on either side.
4. Reattach the Pi case lid and power up the SCUTTLE after verifying the encoder connections.

5. Open the VS Code and in the terminal window, enter the following command:
   `sudo watch -n0.2 i2cdetect -y -r 1`
    a. This will display an output similar to figure 2 indicating the I2C addresses currently in use on the Pi's I2C bus 1.
    b. The left and right encoders are 0x40 and 0x41, respectively.
    c. The IMU is 0x28, the display is 0x3D, and the current sensor is 0x44.

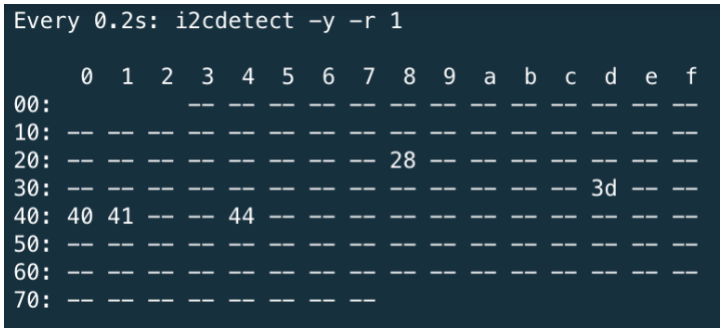

Figure 2: Example of i2cdetect output

6. Verify that your encoders are detected as addresses 0x40 and 0x41.
    a. If the encoders are not detected, check the wiring of your I2C components.
7. You are now ready to test the encoders. Download *L1_encoder.py* to your *basics* directory from the SCUTTLE GitHub then run it using Python 3.
    a. The *L1_encoder.py* output to the terminal will repeatedly print the encoder measurements, in degrees, for the left and right wheels.
    b. To check the encoders, manually spin the wheels. The encoder readings would produce climbing values when wheels are turning forward and falling values when they are reversing. The encoder is functional if your encoder reading behave in the same way.
    c. Leave the wheels alone for a moment and see if the measurement changes. Small changes are acceptable as noise, but the value should oscillate around a single point and not drift significantly over time.
8. Once you have printed the encoder values to console, ask your TA to check the encoder performance before proceeding.

Perform Forward Kinematics

In the previous lab, inverse kinematics was used to calculate the necessary wheel speeds to follow a trajectory given by a series of forward and rotational target velocities. Forward kinematics is the opposite: determining your forward and rotational velocities from the measured speeds of your wheels. This is useful for establishing the robot's position with respect to its origin and is a starting point for localization.

The program *L2_kinematics.py* uses *L1_encoder.py* to obtain wheel position measurements over time. It then determines the angular velocity of either wheel by the displacement and time elapsed between measurements. Using forward kinematics, these values are converted to chassis speeds in the form of forward velocity and rotational velocity.

```
xdot(m/s), thetadot (rad/s): [ 0.005 -0.218]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [ 0.011 -0.21 ]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [ 0.008 -0.057]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [ 0.001 -0.012]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [-0.001  0.004]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [-0.001  0.004]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [ 0.001 -0.004]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [ 0.001 -0.004]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [-0.001  0.004]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [-0.002  0.   ]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [0.001 0.004]       deltaT:  0.022
xdot(m/s), thetadot (rad/s): [0. 0.]     deltaT:  0.021
xdot(m/s), thetadot (rad/s): [ 0.001 -0.004]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [0. 0.]     deltaT:  0.021
```

Figure 3: Example output of L2_kinematics.py

1. Download the *L2_kinematics.py* to your *basics* directory from the SCUTTLE GitHub then run it using Python 3.
    a. The output should look like figure 3.
    b. The returned values indicate the chassis forward and rotational velocities as well as the time elapsed between position measurements.
2. While *L2_kinematics.py* runs, open a new terminal window and, with the SCUTTLE on its stand, launch *L1_motor.py*.
    a. Observe the values for xdot and thetadot. Do they match the motion carried out at each step of *L1_motor*?
3. Stop both scripts and make a new L3 script within your *basics* directory and call it *L3_log_speeds.py*.
    a. Using the L2 scripts already downloaded, write your L3 script to acquire wheel speed and chassis speed measurements as [PDL, PDR] and [xdot, thetadot].

    NOTE: Do not modify existing L1 and L2 scripts, only use them in your own L3 code.

    b. Print these values to the terminal and log them to a file for Node-RED to read. Make sure you add a short sleep after logging the data. This allows time for Node-RED to access the file's contents before the next datum is acquired. It also allows an ample delay for a keyboard interrupt (Ctrl+C) to register.

c. Create a Node-RED flow that reads the wheel speeds and chassis speeds, then displays them to the dashboard GUI. Use charts to plot the information over time. Make sure you include appropriate titles and units for all charts. You can copy the upper and lower bounds used in figure 4's example dashboard.
d. Finally, run your L3 script and *L3_path_template.py* simultaneously in two terminals to drive the wheels for moving forward, backward, and turning. This will provide some example data for the dashboard to display.

Figure 4: Example dashboard for wheel and chassis speeds

4. Deploy your flow and run the L3 script. Verify that your program works and ask the TA to check it off.
5. Once checked off, grab screenshots of the dashboard data for your report, export your Lab5 Node-RED flow and save it to your local repository, commit the changes, and push them to your team's remote GitHub repository.

Post-Lab Questions

1. How does SCUTTLE read the encoder values and produce its linear and angular velocities?

2. How does the I2C protocol work with the encoders?

3. Explain the difference between the *uniqueFile* function and *tmpFile* function in *L1_log.py*.