MXET 300
Mechatronics I

Multidisciplinary
Engineering Technology
COLLEGE OF ENGINEERING

LABORATORY # 5

Encoders and Forward Kinematics

Submission Date: 03/03/2025

Name: Kyle Rex and Rex Worley
Section: 501
UIN: 932008894 and 432006442

Introduction

The objective of this lab is to explore the implementation of motor drivers and forward kinematics in a mobile robotic system using the SCUTTLE robot. The primary focus is on integrating absolute rotary encoders with the motor control system to accurately measure wheel position and velocity. This involves assembling and wiring the SCUTTLE robot, configuring the necessary hardware components, and utilizing the I2C communication protocol to establish reliable data transfer between the encoders and the Raspberry Pi. Through the execution of various scripts, the lab investigates how encoder readings can be processed to compute the robot's forward and rotational velocities using kinematic equations. The lab also emphasizes the importance of data logging and visualization, utilizing Node-RED to display real-time movement data. By running Python scripts to acquire, process, and visualize motion data, the lab provides hands-on experience in implementing and debugging motor control systems. Key concepts covered include the relationship between wheel motion and chassis movement, the mathematical principles behind forward and inverse kinematics, and the practical challenges involved in interfacing sensors with embedded systems.

Procedures and Lab Results

The lab begins with the assembly of the SCUTTLE robot, ensuring that all components, including the Raspberry Pi, battery pack, and motor driver bracket, are securely placed on the DIN rail, before wiring, following the same procedure for mounting these parts that is described in the Lab 3 manual. Placing the parts first, before wiring, prevents potential damage to connectors. Once all the parts are placed on the DIN rail the connections between the Raspberry Pi and Battery Pack can be completed following the same procedure for connecting these two parts described in the Lab 2 manual. Additional connections to the motors and L298N H-bridge are required on the SCUTTLE and can be completed following the Lab 3 manual. For this lab there are two additional connections that are required between the absolute rotary encoders and the Raspberry Pi. These connections are performed using two 300 mm long QWIIC cables which connect to the left and right sides of the red QWIIC Multiport component, inside of the enclosed lid of the Raspberry Pi, to the left and right encoders respectively. The purpose of the QWIIC Multiport is to act as an I2C bridge allowing for the communication between the Raspberry Pi and encoders. Additionally, each encoder has its own QWIIC adaptor board, which are what the other ends of the QWIIC cables will connect to, which provide conversion from the eight Dupont connectors on the AS5048 encoder to a much smaller JST connector for the I2C bus to connect to. To prevent wiring errors, connections must be verified before proceeding. These connection schemes can be seen in Figure 1, Figure 2, and Figure 3. The final setup for the SCUTTLE can be seen in Figure 4.

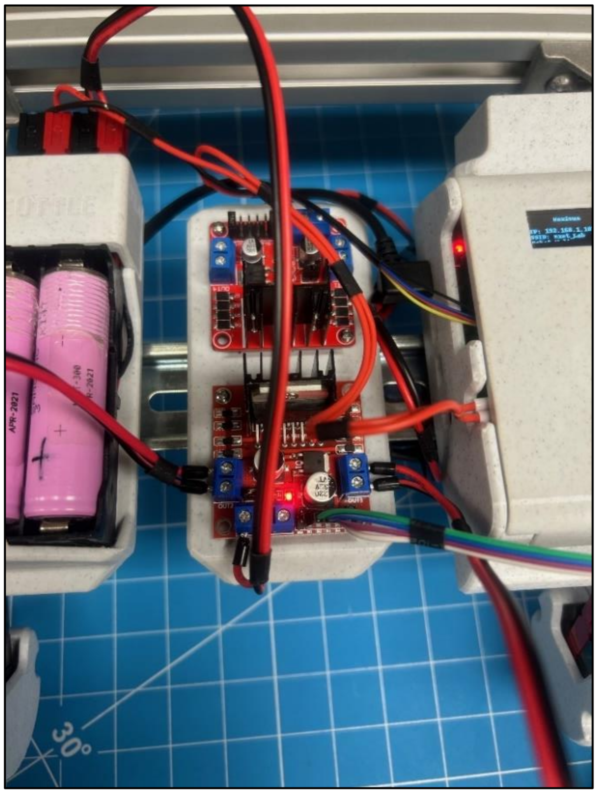


Figure 1: H-Bridge Connections

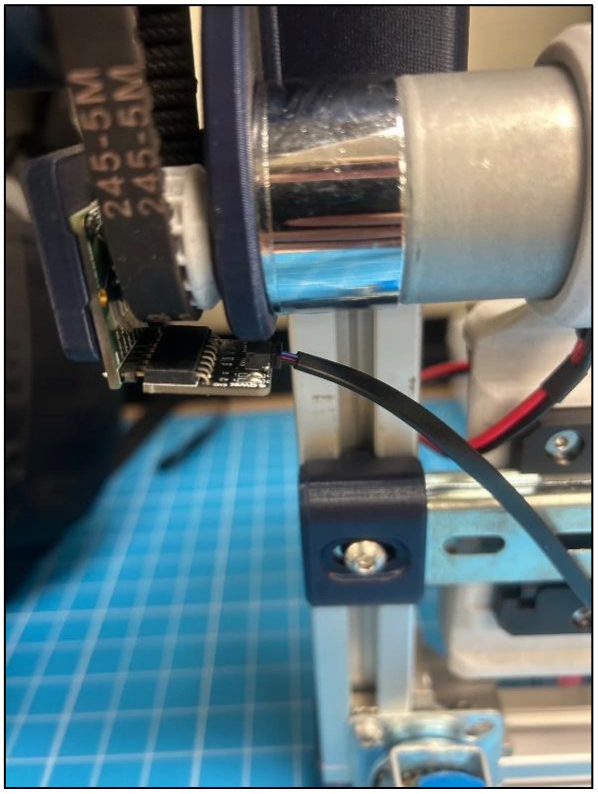


Figure 2: Encoder Connection (Same on Both Sides)

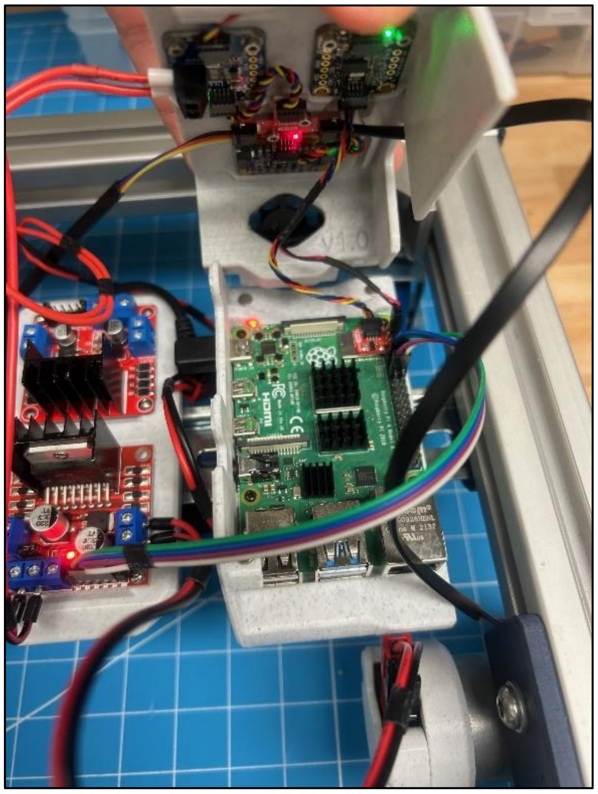


Figure 3: Raspberry Pi Enclosure Connections

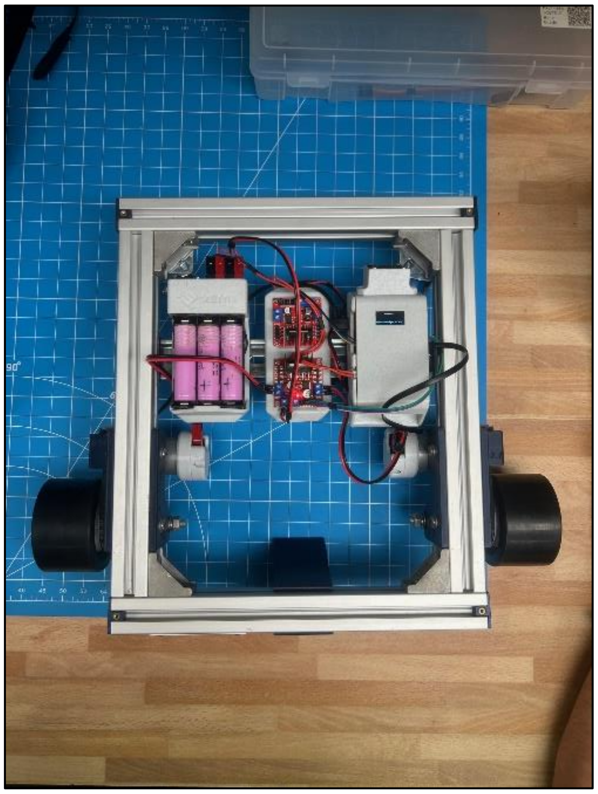
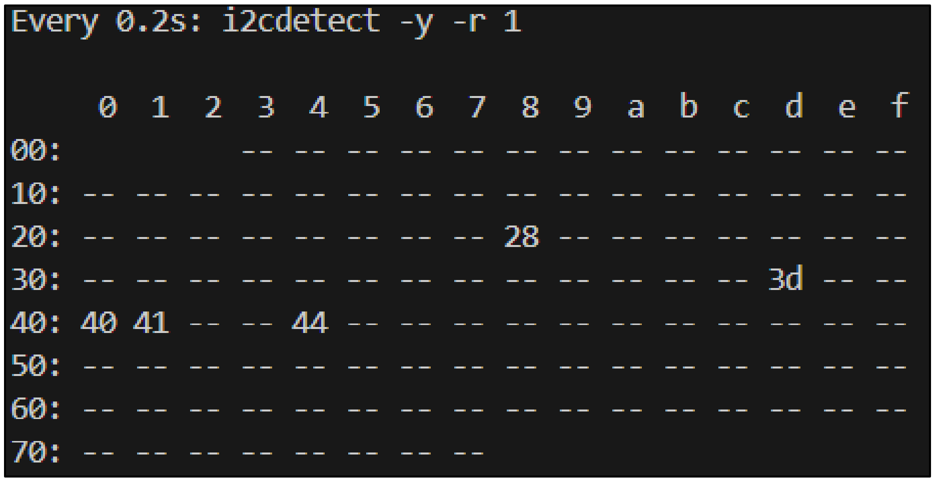


Figure 4: Complete SCUTTLE Physical Setup with All Connections

The additional QWIIC cables provide the SCUTTLE robot with the ability to use its absolute rotary encoders to measure the position of the wheels by measuring the change in the magnetic field of a magnet fixed to the wheel's axle. With these readings the angular velocity can be calculated using the time elapsed between two position measurements. I2C (inter integrated circuit) is used by the Raspberry Pi to communicate with the sensors allowing a single bus to chain devices together for communication.

After checking the wiring and setup are correct the next step is to complete the booting and connecting procedure that are described in the Lab 1 manual. This will allow the Raspberry Pi to be accessed through Visual Code Studio's terminal over a wireless network. When the pi powers on connect to the network and open the VS code. In the VS code terminal window enter the "sudo watch -n0.2 i2cdetect -y -r 1" command. It will display the output that can be seen in Figure 5.



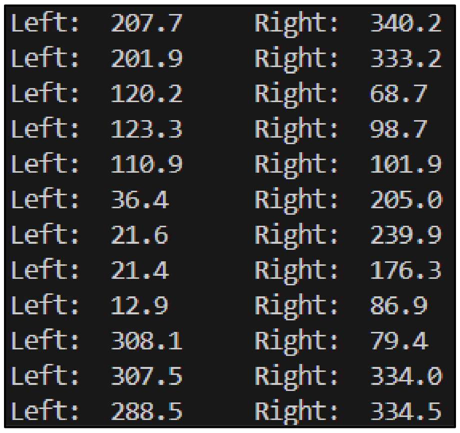
```
Every 0.2s: i2cdetect -y -r 1

     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- 28 -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- 3d -- --
40: 40 41 -- -- 44 -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 5: Correct i2cdetect output

This display shows the I2C addresses currently in use on the Pi's I2C bus 1. The left encoder is 0x40, the right encoder is 0x41, the IMU is 0x28, the display is 0x3D, and the current sensor is 0x44. The display should look exactly as it does in Figure 4. If it does not, then there is most likely a bad physical connection.

Next use the "cd" command to go into the basics directory. Once in this directory the "wget" command can be used, following the procedure in the Lab 2 manual, in the MXET300-SCUTTLE directory, to download and import L1_encoder.py. Do this same procedure to get L2_kinematics.py from the templates directory in in the MXET300- SCUTTLE GitHub that will be used later in the lab.

For this next part of the lab the encoders of the SCUTTLE are tested by running "python3 L1_encoder.py", which will result in terminal readings. The terminal output will print the encoder measurements, in degrees, for the left and right wheels. When the wheels are being turned forward the values should get larger. When the wheels are being turned backward the values should get smaller. When the wheels are not being turned the values should not change. An example of this output can be seen in Figure 6.

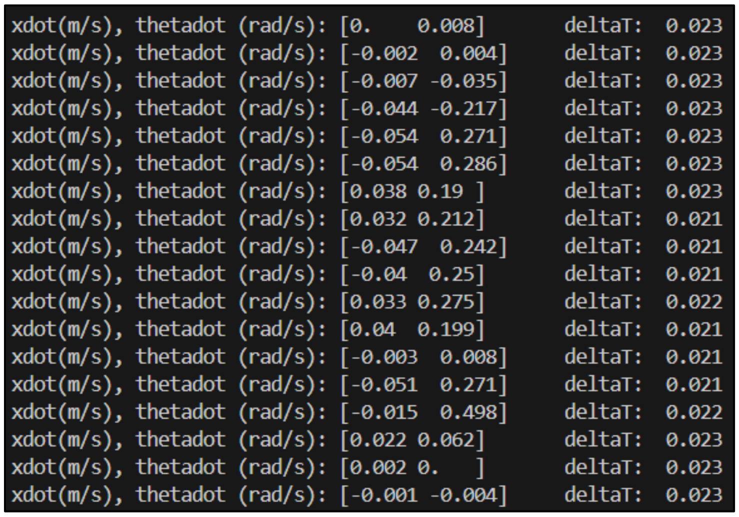


```
Left:  207.7    Right:  340.2
Left:  201.9    Right:  333.2
Left:  120.2    Right:  68.7
Left:  123.3    Right:  98.7
Left:  110.9    Right:  101.9
Left:  36.4     Right:  205.0
Left:  21.6     Right:  239.9
Left:  21.4     Right:  176.3
Left:  12.9     Right:  86.9
Left:  308.1    Right:  79.4
Left:  307.5    Right:  334.0
Left:  288.5    Right:  334.5
```

Figure 6: Example of Encoder Degree Test Values

In lab 4, inverse kinematics was utilized to determine the wheel speeds required to follow a specific path using a series of forward and rotational velocities. Forward kinematics, on the other hand, works in the opposite way. It calculates the robot’s forward, and rotational velocities based on measured wheel speeds. This approach is valuable for determining the robot’s position relative to its starting point and serves as a foundation for localization. The program L2_kinematics.py, that was imported previously, relies on the L1_encoder.py program to gather wheel position data over time. By analyzing the displacement and time difference between consecutive measurements, it computes the angular velocity of each wheel. These values are then converted into chassis velocities, represented by both forward velocity and rotational velocity, using forward kinematics.

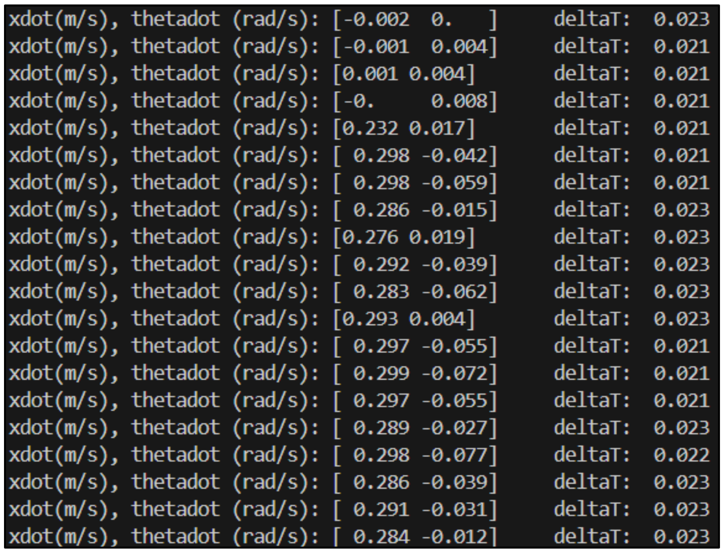
This next part of the lab begins by running “python3 L2_kinematics.py”, which will result in terminal readings similar to that of what can be seen in Figure 7 showing values for the linear velocity ($\dot{x}$) and angular velocity ($\dot{\theta}$) of the chassis based on the wheel speeds.



```
xdot(m/s), thetadot (rad/s): [0.    0.008]      deltaT:  0.023
xdot(m/s), thetadot (rad/s): [-0.002  0.004]    deltaT:  0.023
xdot(m/s), thetadot (rad/s): [-0.007 -0.035]    deltaT:  0.023
xdot(m/s), thetadot (rad/s): [-0.044 -0.217]    deltaT:  0.023
xdot(m/s), thetadot (rad/s): [-0.054  0.271]    deltaT:  0.023
xdot(m/s), thetadot (rad/s): [-0.054  0.286]    deltaT:  0.023
xdot(m/s), thetadot (rad/s): [0.038 0.19 ]      deltaT:  0.023
xdot(m/s), thetadot (rad/s): [0.032 0.212]      deltaT:  0.021
xdot(m/s), thetadot (rad/s): [-0.047  0.242]    deltaT:  0.021
xdot(m/s), thetadot (rad/s): [-0.04  0.25]      deltaT:  0.021
xdot(m/s), thetadot (rad/s): [0.033 0.275]      deltaT:  0.022
xdot(m/s), thetadot (rad/s): [0.04  0.199]      deltaT:  0.021
xdot(m/s), thetadot (rad/s): [-0.003  0.008]    deltaT:  0.021
xdot(m/s), thetadot (rad/s): [-0.051  0.271]    deltaT:  0.021
xdot(m/s), thetadot (rad/s): [-0.015  0.498]    deltaT:  0.022
xdot(m/s), thetadot (rad/s): [0.022 0.062]      deltaT:  0.023
xdot(m/s), thetadot (rad/s): [0.002 0.   ]      deltaT:  0.023
xdot(m/s), thetadot (rad/s): [-0.001 -0.004]    deltaT:  0.023
```

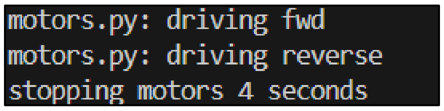
Figure 7: Example output of L2_kinematics.py Without Wheel Movement

This output shows the chassis forward and rotational velocities as well as the time elapsed between position measurements. While this continues to produce an output, place the SCUTTLE on its stand with the wheels off of the ground. With this step complete the L1_motor.py script can be run in another terminal using "python3 L1_motor.py". The wheels will begin to rotate and the output of the L2_kinematics.py script will reflect the motion of the motors. An example of what this may look like can be seen in Figure 8. The terminal output of L1_motor.py can be seen in Figure 9.



```
xdot(m/s), thetadot (rad/s): [-0.002  0.   ]     deltaT:  0.023
xdot(m/s), thetadot (rad/s): [-0.001  0.004]     deltaT:  0.021
xdot(m/s), thetadot (rad/s): [0.001 0.004]       deltaT:  0.021
xdot(m/s), thetadot (rad/s): [-0.     0.008]     deltaT:  0.021
xdot(m/s), thetadot (rad/s): [0.232 0.017]       deltaT:  0.021
xdot(m/s), thetadot (rad/s): [ 0.298 -0.042]     deltaT:  0.021
xdot(m/s), thetadot (rad/s): [ 0.298 -0.059]     deltaT:  0.021
xdot(m/s), thetadot (rad/s): [ 0.286 -0.015]     deltaT:  0.023
xdot(m/s), thetadot (rad/s): [0.276 0.019]       deltaT:  0.023
xdot(m/s), thetadot (rad/s): [ 0.292 -0.039]     deltaT:  0.023
xdot(m/s), thetadot (rad/s): [ 0.283 -0.062]     deltaT:  0.023
xdot(m/s), thetadot (rad/s): [0.293 0.004]       deltaT:  0.023
xdot(m/s), thetadot (rad/s): [ 0.297 -0.055]     deltaT:  0.021
xdot(m/s), thetadot (rad/s): [ 0.299 -0.072]     deltaT:  0.021
xdot(m/s), thetadot (rad/s): [ 0.297 -0.055]     deltaT:  0.021
xdot(m/s), thetadot (rad/s): [ 0.289 -0.027]     deltaT:  0.023
xdot(m/s), thetadot (rad/s): [ 0.298 -0.077]     deltaT:  0.022
xdot(m/s), thetadot (rad/s): [ 0.286 -0.039]     deltaT:  0.023
xdot(m/s), thetadot (rad/s): [ 0.291 -0.031]     deltaT:  0.023
xdot(m/s), thetadot (rad/s): [ 0.284 -0.012]     deltaT:  0.023
```

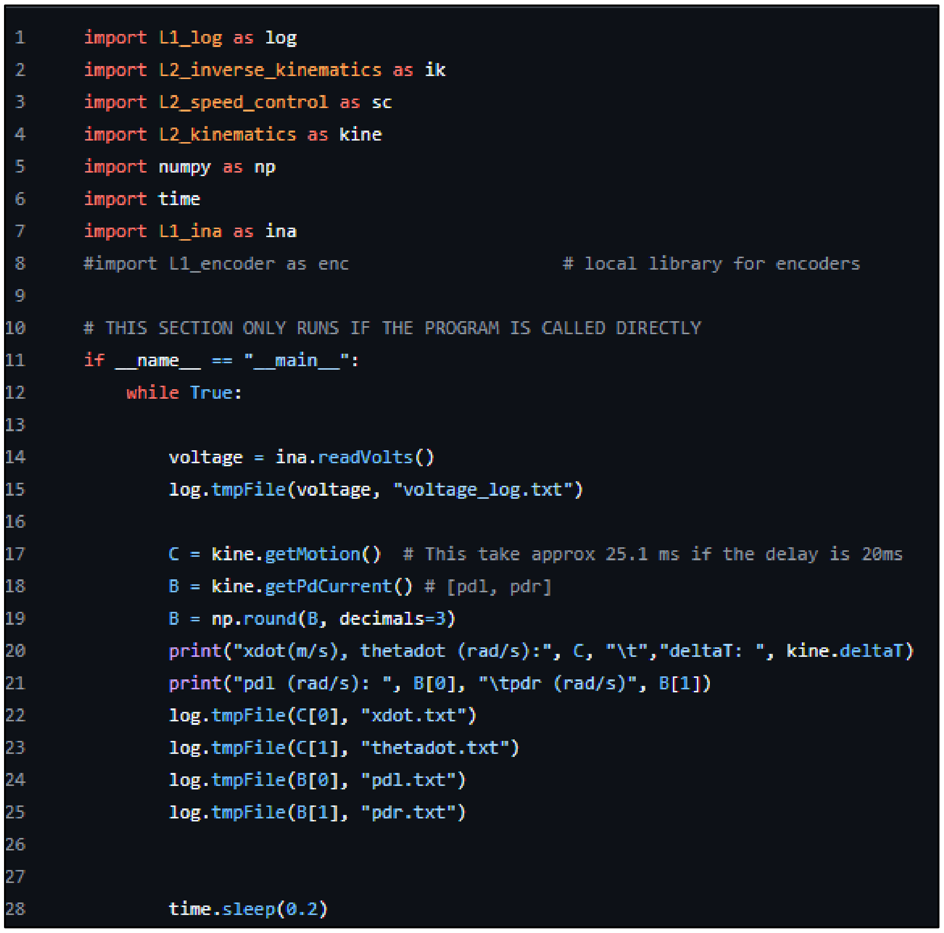
Figure 8: Example output of L2_kinematics.py With Wheel Movement



```
motors.py: driving fwd
motors.py: driving reverse
stopping motors 4 seconds
```

Figure 9: Example output of L1_motor.py

With this all complete, and all scripts terminated, the next step of this lab involves creating a new file called L3_log_speeds.py that will acquire the wheel speed and chassis speed measurements as [PDL, PDR] and [xdot, thetadot] from the L1 and L2 scripts previously imported. This file will additionally be used to print the values in the terminal while additionally logging them to files for Node-RED to read. When creating this new file L1_ina.py was imported to obtain a battery voltage measurement with L2_kinematics imported to use "getMotion" and "getPdCurrent" functions which produce the wheel speeds and chassis speeds each in their respective arrays. The chassis speed array is rounded to three decimal places and values are printed to the terminal. The imported L1_log file is used to create temporary text files for Node-RED to easily read and display. The newly created and complete script can be seen in Figure 10.



```
import L1_log as log
import L2_inverse_kinematics as ik
import L2_speed_control as sc
import L2_kinematics as kine
import numpy as np
import time
import L1_ina as ina
#import L1_encoder as enc                    # local library for encoders

# THIS SECTION ONLY RUNS IF THE PROGRAM IS CALLED DIRECTLY
if __name__ == "__main__":
    while True:

        voltage = ina.readVolts()
        log.tmpFile(voltage, "voltage_log.txt")

        C = kine.getMotion()  # This take approx 25.1 ms if the delay is 20ms
        B = kine.getPdCurrent() # [pdl, pdr]
        B = np.round(B, decimals=3)
        print("xdot(m/s), thetadot (rad/s):", C, "\t","deltaT: ", kine.deltaT)
        print("pdl (rad/s): ", B[0], "\tpdr (rad/s)", B[1])
        log.tmpFile(C[0], "xdot.txt")
        log.tmpFile(C[1], "thetadot.txt")
        log.tmpFile(B[0], "pdl.txt")
        log.tmpFile(B[1], "pdr.txt")


        time.sleep(0.2)
```

Figure 10: L3_log_speeds.py Script

With the complete L3_log_speeds.py script the next step involves creating a Node-RED flow that will read the wheel speeds and chassis speeds and display them to the dashboard GUI in the form of charts plotting the values over time making sure to include appropriate titles and units. The final Node-RED flow created to achieve this result can be seen in Figure 11.

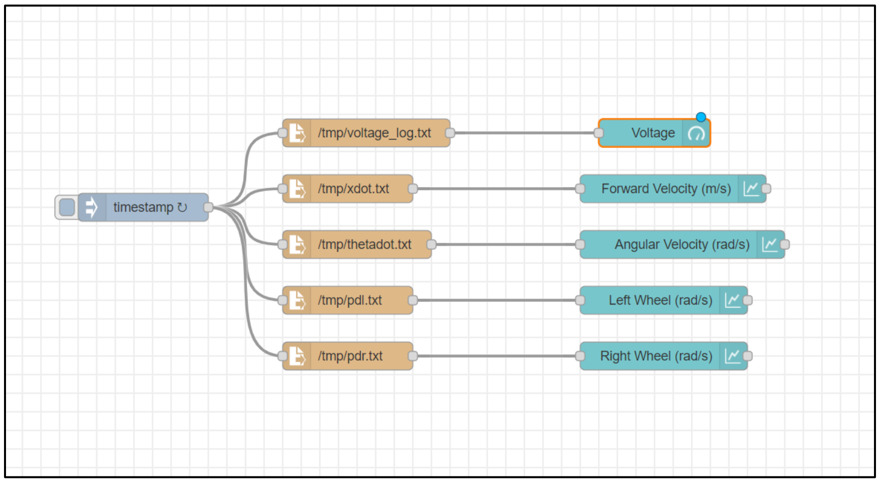


Figure 11: Node-RED Flow

With the flow complete the next step involves running the L3 script and L3_path_template.py simultaneously in two terminals. This will drive the wheels forward, backward, and turning, providing some example data for the dashboard to display. An example of the final Node-RED dashboard can be seen in Figure 12.

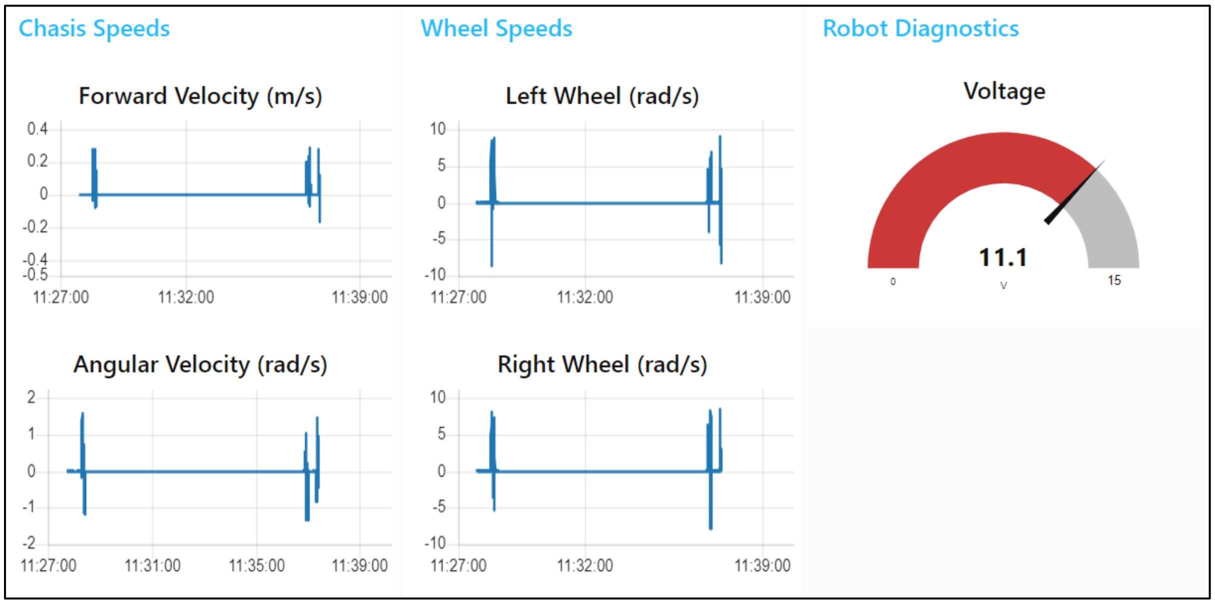


Figure 12: Node-RED Dashboard

Conclusion

The lab successfully demonstrated the integration of motor drivers and absolute rotary encoders with the SCUTTLE robot, allowing for precise tracking of wheel position and real-time velocity measurements. The use of I2C communication facilitated efficient data exchange between the encoders and the Raspberry Pi, enabling accurate computation of linear and angular velocities. By applying inverse kinematics in the previous lab, the system was able to determine the required wheel speeds to achieve specific movements, while using forward kinematics in this lab provided insights into the robot's actual trajectory based on measured encoder data. Throughout the lab, several challenges were encountered, including ensuring correct wiring connections, debugging communication errors between the Raspberry Pi and encoders, and creating a new python file to output specific values in order to produce accurate results. Troubleshooting these issues reinforced the importance of verifying hardware connections, validating software functionality, and systematically testing each component before full system integration. Additionally, the implementation of real-time data logging and visualization using Node-RED provided a clear representation of the SCUTTLE's movement data, aiding in performance analysis and system debugging. This lab provided valuable experience in integrating sensors with embedded systems, working with motor drivers, and applying mathematical models to real-world robotic applications. The skills and knowledge gained from this lab are fundamental to developing more advanced robotic systems that require precise motion control and localization capabilities.

Post-Lab Questions

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

The SCUTTLE reads the encoder values by using absolute rotary encoders that measure the change in the magnetic field of a magnet attached to the wheel axle. The encoders communicate with the Raspberry Pi via the I2C protocol, providing position data in degrees. Using the L1_encoder.py script, these values are read and converted into angular velocity by calculating the change in position over time. The L2_kinematics.py script then applies forward kinematics equations to determine the linear velocity ($\dot{x}$) and angular velocity ($\dot{\theta}$) of the chassis based on the wheel speeds. SCUTTLE combines python files to read encoder values and produce linear and angular velocities. The first file reads two bytes of data from one of two I2C addresses; this data is combined to create a 14-byte integer used to represent the motor shaft's position. The integer is converted from binary to degrees using a conversion factor of 360/2^14 and is subsequently rounded to 0.1 degrees. After adjusting the left motor's angle to account for orientation, the left and right encoder angles are printed in an array. The second code creates a matrix for calculating forward speed and turning rate. Two successive encoder angle readings are taken and computed to find the direct difference, difference plus 360, and difference minus 360. The smallest absolute of the three values calculated is used to ensure the true change value is used in future formulas. The angular change in degrees is divided by the elapsed time and converted to radians per second using the conversion factor of pi/180. The newly calculated wheel speeds are multiplied by a previously defined matrix with conversion formulas that return the final forward linear velocity in meters per second and angular velocity in radians per second.

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

The I2C (Inter-Integrated Circuit) protocol allows multiple devices to communicate with a single master device (the Raspberry Pi) using a shared two-wire bus (SDA for data, SCL for clock). Each encoder is assigned a unique address (0x40 for the left encoder, 0x41 for the right encoder), enabling the Pi to read position data from them. The QWIIC Multiport breakout board is used to connect multiple I2C devices to the same bus, facilitating communication between the Pi and the encoders. The I2C protocol creates a bus object that opens communication on the I2C bus number 1 and allows the SCUTTLE to select a specific encoder during communication. The SCUTTLE then requests data from a register address asking for 2 bytes of data. Once the data has been transmitted over the I2C bus, the two bytes are combined into a 14-bit integer value and scaled to represent the angle between 0 and 360 degrees. This value represents the motor shaft position in degrees.

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

The uniqueFile function creates a new, uniquely named file each time it is called, typically using timestamps or incrementing numbers. This ensures that each data logging session has a separate file for storage, preventing overwriting of previous logs. The tmpFile function, on the other hand, writes data to a temporary file that can be repeatedly overwritten. This is useful for real-time data access by Node-RED, as it continuously updates the latest recorded values without creating multiple files. The uniqueFile function uses an existing file directory which is more of a permanent storage path whereas the tmpFile function uses a temporary directory specifically for temporary files. In the context of L1_log.py, the uniqueFile function logs a key parameter for integration with NodeRed during long-term processes and other processes. The tmpFile logs data only needed for short-term calculations without reserving unnecessary memory and is useful in constantly updating data into a file for easy measurement on the NodeRed dashboard.

Appendix

Lab 5 Commands Used:

- sudo watch -n0.2 i2cdetect -y -r 1
- wget
- python3 L1_encoder.py
- wget
- python3 L2_kinematics.py
- python3 L1_motor.py
- ctrl + c
- log.tmpFile(voltage, "voltage_log.txt")
- log.tmpFile(C[0], "xdot.txt")
- log.tmpFile(C[1], "thetadot.txt")
- log.tmpFile(B[0], "pdl.txt")
- log.tmpFile(B[1], "pdr.txt")
- python3 L3_log_speeds.py
- ctrl + c
- python3 L3_log_speeds.py
- python3 L3_path_template.py

Lab 4 Commands Used:

- pwd
- python3 L3_path_template.py
- python3 L3_noderedControl.py
- git add .
- git commit -m “Lab 4”
- git push

Lab 3 Commands Used:

- cd basics/
- wget
- python3 L2_compass_heading.py
- python3 L3_Compass.py
- tmpFile
- stringTmpFile
- git add .
- git commit -m “Lab 3”
- git push

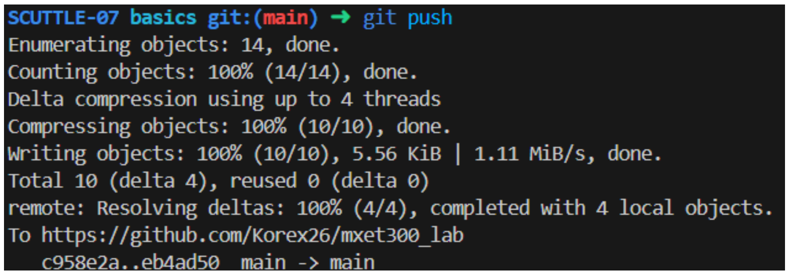
Lab 2 Commands Used:

- cd
- mkdir basics

- pwd
- wget
- python3 L1_ina.py
- sudo systemctl stop oled.service
- sudo mv oled.py/usr/share/pyshared/oled.py
- git status
- git add .
- git commit -m
- git push

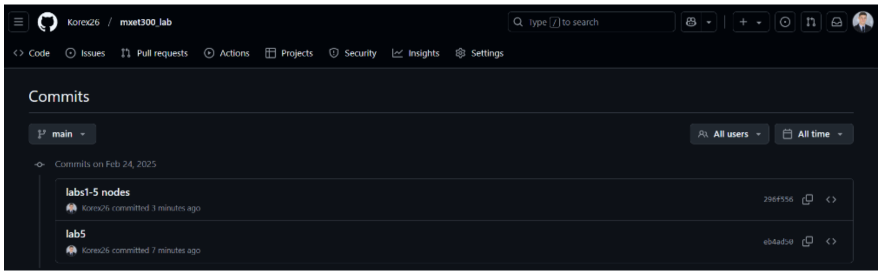
GitHub Submission

All required files were pushed to the team's repository for review and backup as seen in the GitHub Submission Figure 1 and GitHub Submission Figure 2. The repository can be found using the following link: https://github.com/Korex26/mxet300_lab.



```
SCUTTLE-07 basics git:(main) ➜ git push
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 4 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 5.56 KiB | 1.11 MiB/s, done.
Total 10 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/Korex26/mxet300_lab
   c958e2a..eb4ad50  main -> main
```

GitHub Submission Figure 1: Commit and Push to GitHub on Visual Studio Code



Korex26 / mxet300_lab

Commits

Commits on Feb 24, 2025

labs1-5 nodes
Korex26 committed 3 minutes ago — 296f556

lab5
Korex26 committed 7 minutes ago — eb4ad50

GitHub Submission Figure 2: Commit and Push to GitHub on GitHub

References

K. Rex, R. Worley, *MXET 300 Lab Report 4*, 2025.

K. Rex, R. Worley, *MXET 300 Lab Report 3*, 2025.

K. Rex, R. Worley, *MXET 300 Lab Report 2*, 2025.

X. Song, *Lab 5 Manual* - *Encoders _ Forward Kinematics*, 2025.

X. Song, *Lab 4 Manual* - *Motor Drivers _ Inverse Kinematics*, 2025.

X. Song, *Lab 3* - *Compass Calibration*, 2025.

X. Song, *Lab 2 Manual* - *Displays and GUI*, 2025.

X. Song, *Lab 1 Manual* - *Pi Setup*, 2025.