

MXET 300
Mechatronics I



**Multidisciplinary
Engineering Technology**
COLLEGE OF ENGINEERING

LABORATORY # 6
LIDAR and Obstacle Detection

Submission Date: 03/24/2025

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

Introduction

This lab focused on the implementation of a LIDAR-based obstacle detection system using the SICK TiM561 sensor mounted on the SCUTTLE robotic platform. The objective was to measure the surrounding environment, identify the closest obstacle, and determine its position using distance and angle measurements. The LIDAR sensor operates by emitting laser pulses across a 270-degree field of view and measuring the time of flight of reflected signals to calculate object distances. The collected data was processed using Python scripts, including *L1_lidar.py* for raw data acquisition and *L2_vector.py* for extracting the nearest obstacle's distance and angle. This information was then converted into Cartesian coordinates and visualized using a Node-RED graphical interface. Additionally, a real-time LIDAR point cloud was generated to analyze obstacle distribution while manually driving the SCUTTLE robot. The experiment provided hands-on experience in sensor integration, data processing, and real-time visualization, demonstrating how LIDAR technology supports autonomous navigation and object detection. Various challenges associated with LIDAR, including measurement accuracy, interference from reflective or dark surfaces, and angular resolution limitations, were also explored.

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 procedure that is described in the Lab 4 manual. Two more connections are required between the absolute rotary encoders and the Raspberry Pi and can be done following the procedure described in the Lab 5 manual. These connection schemes can be seen in Figure 1, Figure 2, and Figure 3.

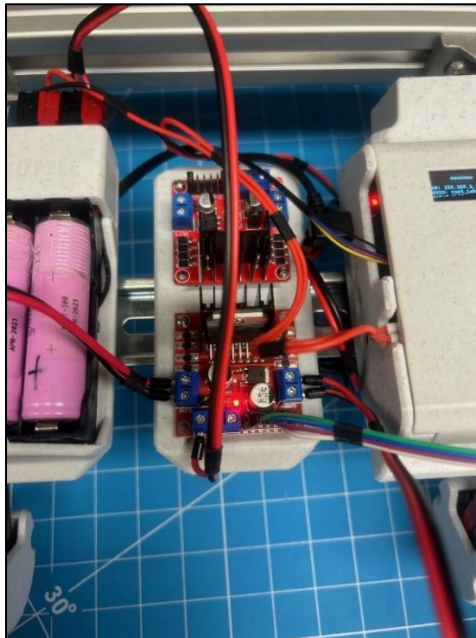


Figure 1: H-Bridge Connections

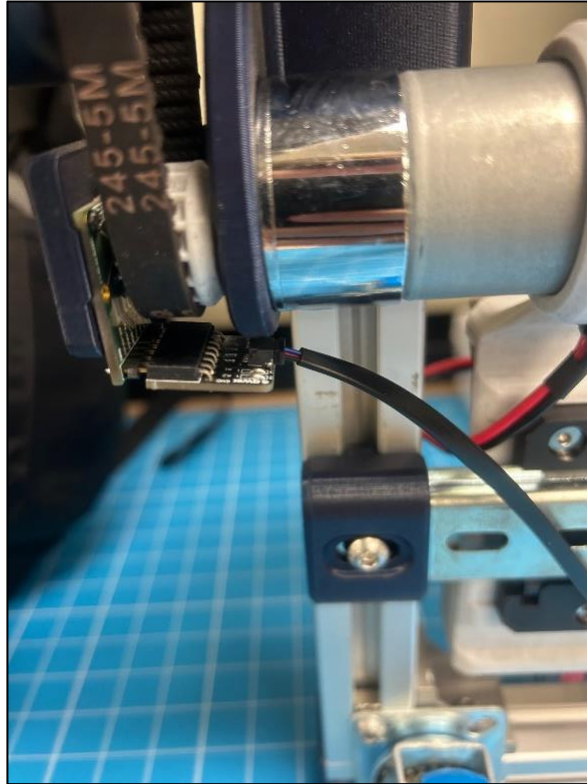


Figure 2: Encoder Connection (Same on Both Sides)

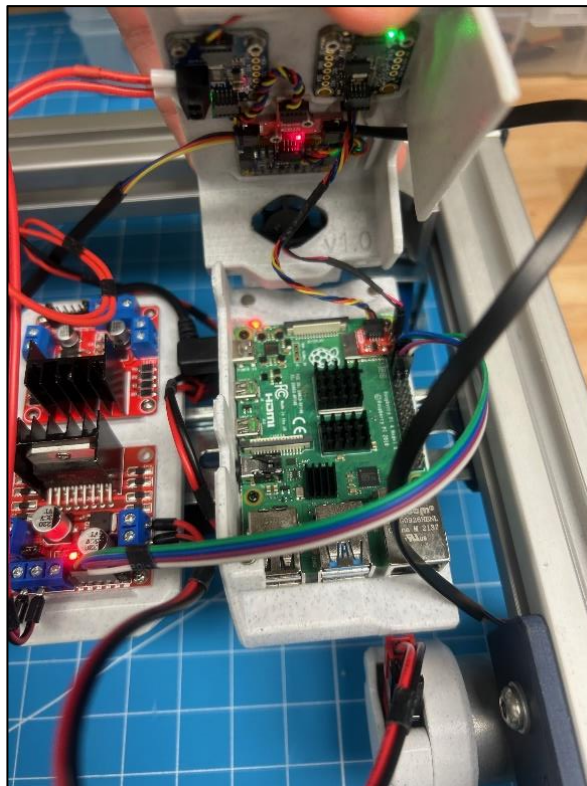


Figure 3: Raspberry Pi Enclosure Connections

For this lab there is an additional sensor that must be mounted and connected to the SCUTTLE and Raspberry Pi. The sensor being used is a TiM561 LIDAR module from SICK which covers a 270 degree sector at a maximum of ten meters using an opto-electronic laser scanner. A LIDAR sensor operates by emitting light pulses in multiple directions at regular intervals. When these pulses hit a surface, they reflect back to a photodiode in the sensor. By measuring the time between emission and detection, and using the speed of light, the system calculates the distance traveled. The sensor pulses at 15 Hz, achieving an angular resolution of 0.33 degrees at a 1-meter distance. This resolution decreases with distance, making it harder to detect narrow objects further away. LIDAR sensors also struggle with reflective, dark, or shallow-angled surfaces since the reflected light may not return directly to the sensor.

To mount the sensor to the rear of the SCUTTLE a screwdriver must be used to loosen the screws and allow the connectors to slide into a slot in the SCUTTLE body ensuring that the sensor itself is facing forward. Once in the slot the screws can be tightened until the Lidar Sensor is securely connected to the SCUTTLE body. Ensure that none of the wires or components of the SCUTTLE are in the sensors range to avoid the sensor detecting the SCUTTLE itself. Once complete the Lidar Sensor must be connected to the Raspberry Pi and battery pack. This requires a LIDAR power cable, micro-USB cable, and an adapter to extend the number of ports on the battery for powering motors and LIDAR simultaneously. The LIDAR power cable is connected to the LIDAR itself and the adapter which is connected to the battery pack. This power is used to run its electronics and a small motor for spinning the photodiode. The micro-USB cable is connected to the LIDAR and the Raspberry Pi allowing for communication with the host device and transmitting measurements. To prevent wiring errors, connections must be verified before proceeding. This connection scheme can be seen in Figure 4. The final setup for the SCUTTLE can be seen in Figure 5.

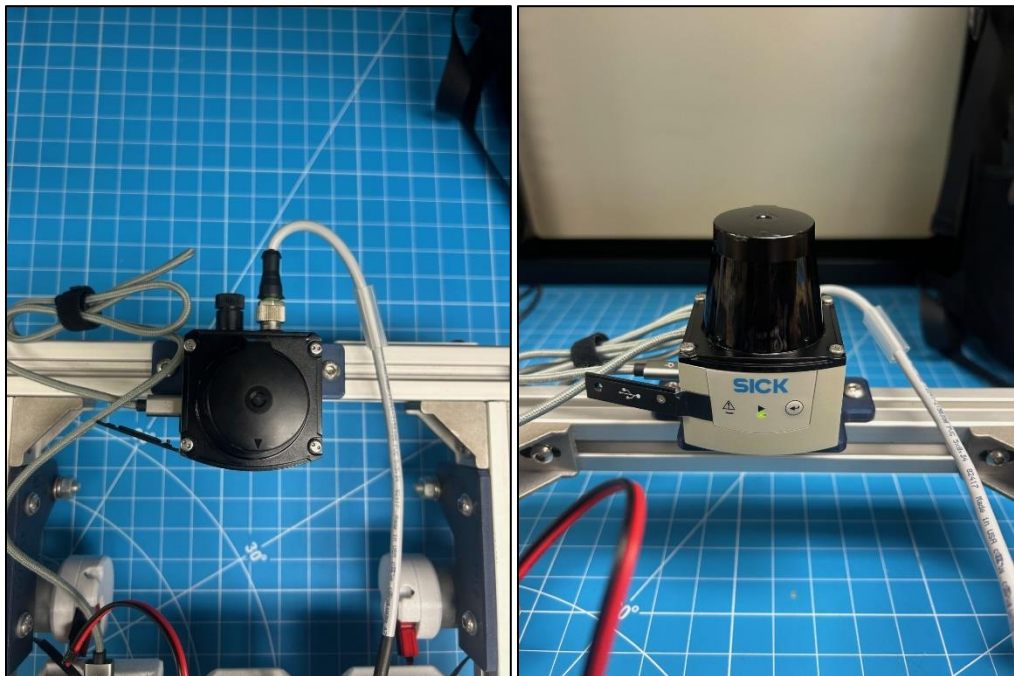


Figure 4: Lidar Sensor Connections

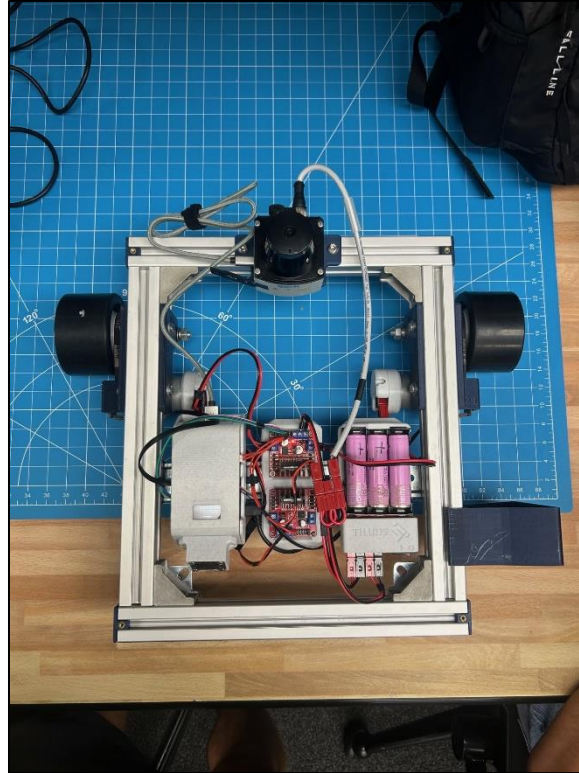


Figure 5: Complete SCUTTLE Physical Setup with All Connections

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 "lsusb" command. This command lists all USB devices detected by the Raspberry Pi. While the LIDAR is connected it will display the terminal output that can be seen in Figure 6. While it is disconnected it will display the terminal output that can be seen in Figure 7. The LIDAR must show up on this list to continue with the next parts of the lab.

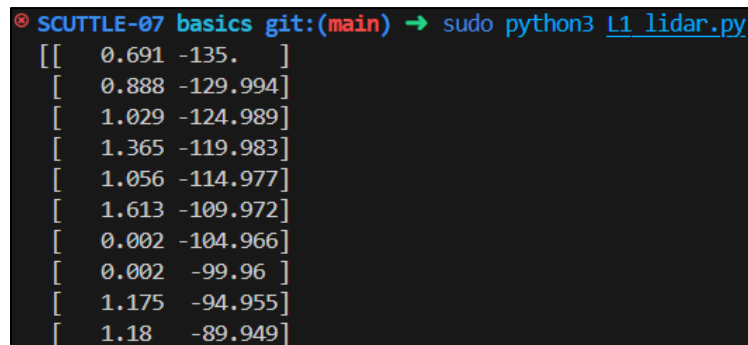
```
● SCUTTLE-07 basics git:(main) → lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 19a2:5001
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Figure 6: USB devices detected by the Pi with LIDAR Connected

```
● SCUTTLE-07 basics git:(main) → lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Figure 7: USB devices detected by the Pi with LIDAR Disconnected

Based on Figure 6, it can be inferred that device 003 is the LIDAR sensor even though it does not have a display name like the rest of the devices. With an operational LIDAR sensor, the next step involves obtaining measurements to test the sensor's output, find the angle and distance of the closest measurement, and determine that measurement's local cartesian (x,y) coordinates. To achieve this a few files must be imported. To do this 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 GitHub directory, to download and import L1_lidar.py, which obtains measurements from the device and outputs them as 54 distance (m) and angle (deg) measurements, and L2_vector.py, which provides other useful functions. Follow this same procedure to import lidar_driving.py and flow_lidar_driving.json that will be used later in the lab. Running the L1_lidar.py with sudo as "sudo python3 L1_lidar.py" and L2_vector.py with sudo as "sudo python3 L2_vector.py" will result in the terminal outputs seen in Figure 8 and Figure 9 respectively, where L1 outputs array of 54 datums, each with a distance and angle and L2 outputs a distance and angle matching the closest obstacle.

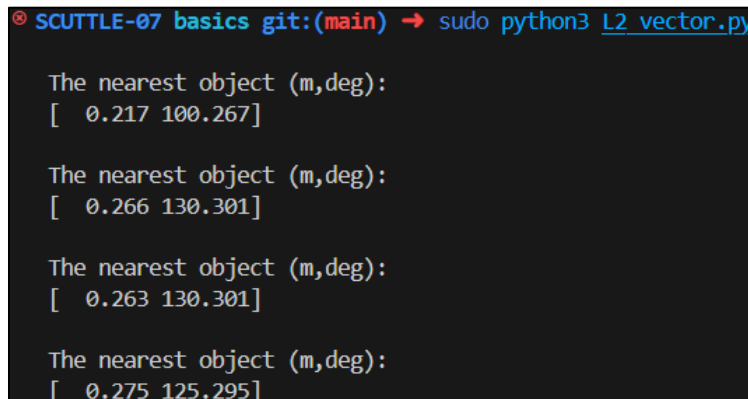


```

SCUTTLE-07 basics git:(main) → sudo python3 L1_lidar.py
[[ 0.691 -135. ]
 [ 0.888 -129.994]
 [ 1.029 -124.989]
 [ 1.365 -119.983]
 [ 1.056 -114.977]
 [ 1.613 -109.972]
 [ 0.002 -104.966]
 [ 0.002 -99.96 ]
 [ 1.175 -94.955]
 [ 1.18 -89.949]

```

Figure 8: L1_lidar.py Terminal Output



```

SCUTTLE-07 basics git:(main) → sudo python3 L2_vector.py

The nearest object (m,deg):
[ 0.217 100.267]

The nearest object (m,deg):
[ 0.266 130.301]

The nearest object (m,deg):
[ 0.263 130.301]

The nearest object (m,deg):
[ 0.275 125.295]

```

Figure 9: L2_vector.py Terminal Output

Sudo is used while running these scripts to get root permission to use the USB interface on the Raspberry Pi to retrieve the sensor's data. With these working scripts, the next step is to create an L3 script in order to create a Node-RED flow to log the closest measurement obstacle for display. The L1 and L2 scripts must be imported into this L3 script and used to log the readings as seen in the complete L3 script in Figure 10.

```

1  import L2_vector as vec
2  import L1_log as log
3  import L1_ina as ina
4  import time
5
6
7  while True:
8      voltage = ina.readVolts()
9
10     vec_dist = vec.getNearest()
11     vec_dist2 = vec.polar2cart(vec_dist[0], vec_dist[1])
12
13     log.tmpFile(vec_dist[0], "L2_dist.txt")
14     log.tmpFile(vec_dist[1], "L2_angle.txt")
15
16     log.tmpFile(vec_dist2[0], "L2_x.txt")
17     log.tmpFile(vec_dist2[1], "L2_y.txt")
18
19     log.tmpFile(voltage, "voltage_log.txt")
20     time.sleep(0.1)

```

Figure 10: Complete L3 Script

In Node-RED a bar graph will be used to display the distance of the nearest obstacle while a compass will be used to display the angle of the nearest obstacle. The Node-RED flow and dashboard can be seen in Figure 11 and 12 respectively.

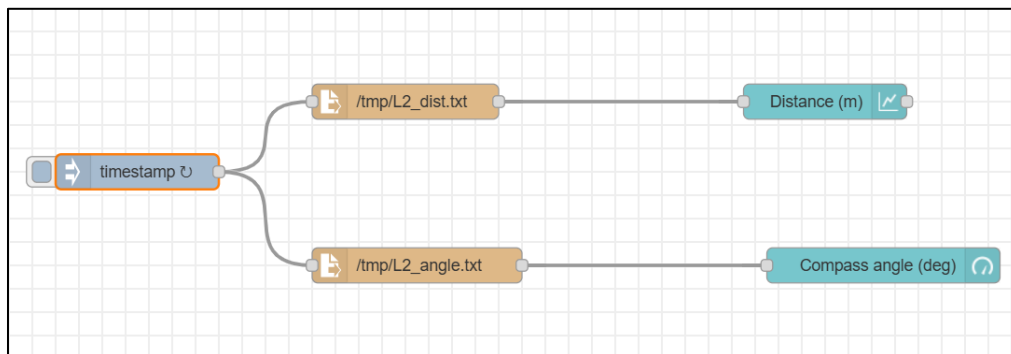


Figure 11: L3 Closest Obstacle Measurements Node-RED Flow

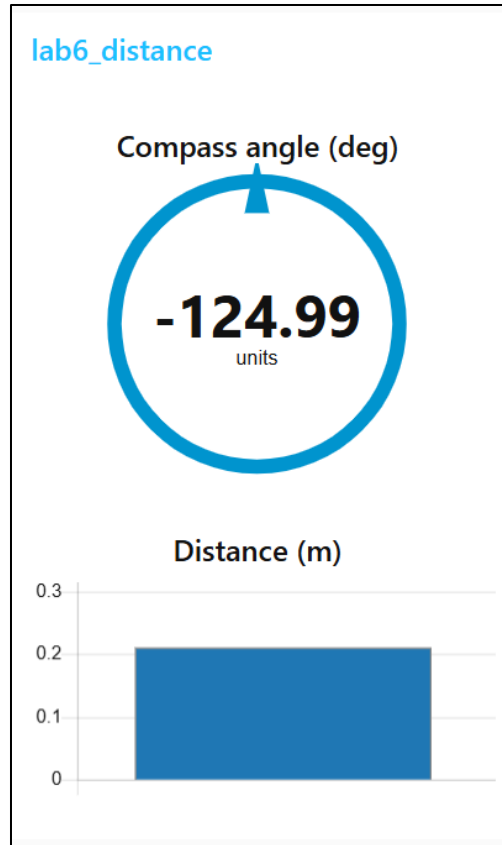


Figure 12: L3 Closest Obstacle Measurements Node-RED Dashboard

With this part complete there are a few more additions that are required to be made on the L3 script and the Node-RED. These additions include logging the x and y cartesian coordinate components of the closest obstacle vector that are found by converting the distance and angle vector of the L2_vector.py script. This is done by altering the L3 script to what can be seen in Figure 13.

```

1  import L2_vector as vec
2  import L1_log as log
3  import L1_ina as ina
4  import time
5
6
7  while True:
8      voltage = ina.readVolts()
9
10     vec_dist = vec.getNearest()
11     vec_dist2 = vec.polar2cart(vec_dist[0], vec_dist[1])
12
13     log.tmpFile(vec_dist[0], "L2_dist.txt")
14     log.tmpFile(vec_dist[1], "L2_angle.txt")
15
16     log.tmpFile(vec_dist2[0], "L2_x.txt")
17     log.tmpFile(vec_dist2[1], "L2_y.txt")
18
19     log.tmpFile(voltage, "voltage_log.txt")
20     time.sleep(0.1)

```

Figure 13: Complete L3 Script

With these additions to the L3 code, the Node-RED flow can be updated to display the voltage of the SCUTTLE in the form of a gauge and x and y coordinates in the form of two line charts to the dashboard GUI. Left should be positive for y, forward should be positive for x, and the range of the x and y coordinates must be set to a reasonable distance. The gauge range is set from -135 to 135. The final Node-RED flow and dashboard can be seen in Figure 14 and 15 respectively.

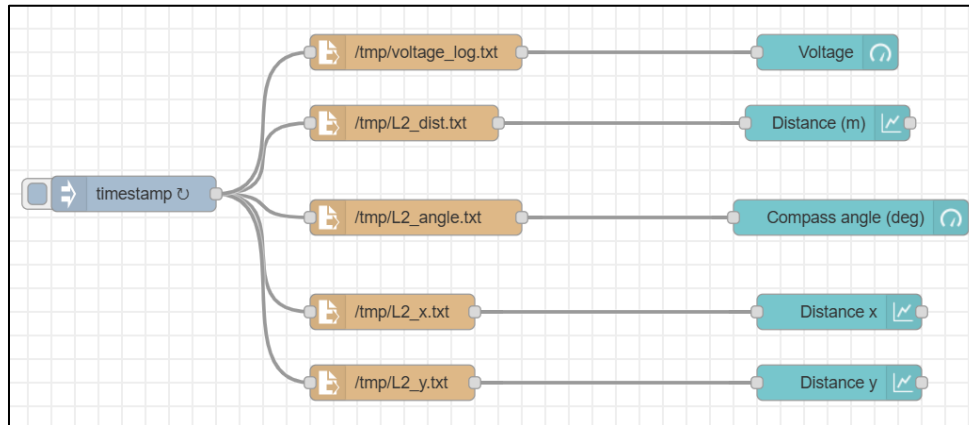


Figure 14: L3 Closest Obstacle Measurements with X & Y Coordinates Node-RED Flow

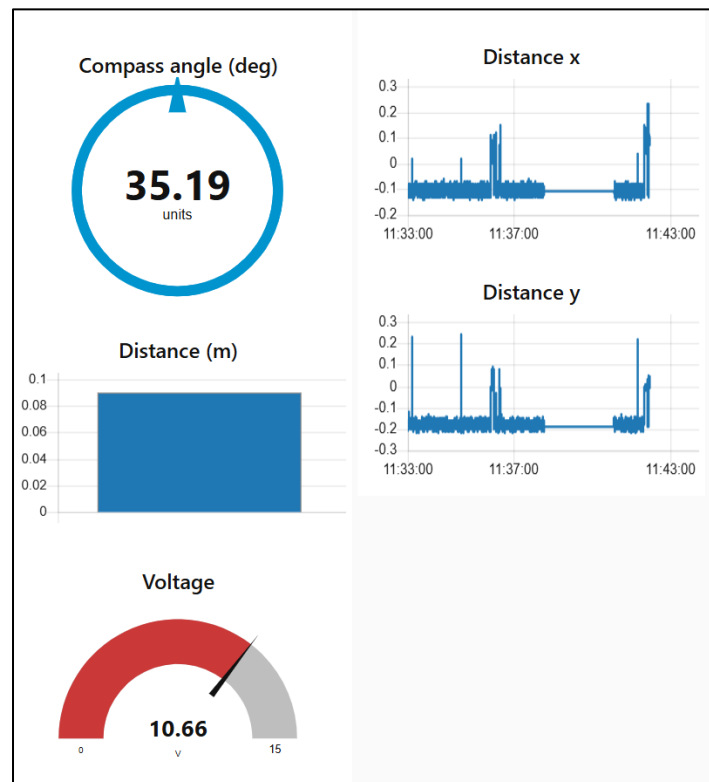


Figure 15: L3 Closest Obstacle Measurements with X & Y Coordinates Node-RED Dashboard

With this part of the lab now complete the final exercise involves driving the SCUTTLE with LIDAR utilizing an existing Node-RED flow and Python script to visualize the complete LIDAR point cloud while manually operating the SCUTTLE robot. This provides insight into how an autonomous system could

perceive its surroundings and use LIDAR for navigation. By observing the point cloud, changes in obstacle proximity can be analyzed in real-time. The process requires utilizing the previously downloaded `lidar_driving.py` script and `flow_lidar_driving.json` from GitHub, ensuring proper directory placement alongside previously used scripts. The script employs an object-oriented structure to manage motor actuation and sensor data, demonstrating an organized approach to multi-module robot programming. Once the flow is imported into Node-RED and deployed, the script is executed with root permissions to allow USB interface communication with the LIDAR module. For better visualization, the Node-RED theme is switched to dark mode, ensuring clear contrast of the point cloud data. The dashboard displays red points representing the robot's frame and white points indicating LIDAR readings. While driving, variations in the spread of data points highlight the impact of distance on sensor accuracy. The Node-RED flow and dashboard can be seen in Figure 16 and 17 respectively.

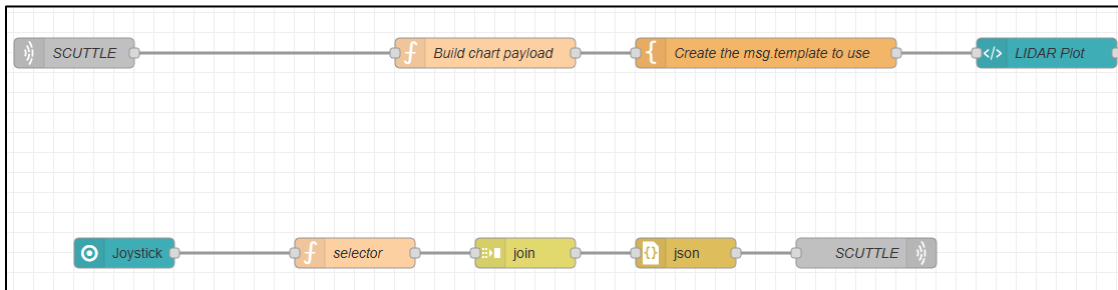


Figure 16: Driving with LIDAR Node-RED Flow

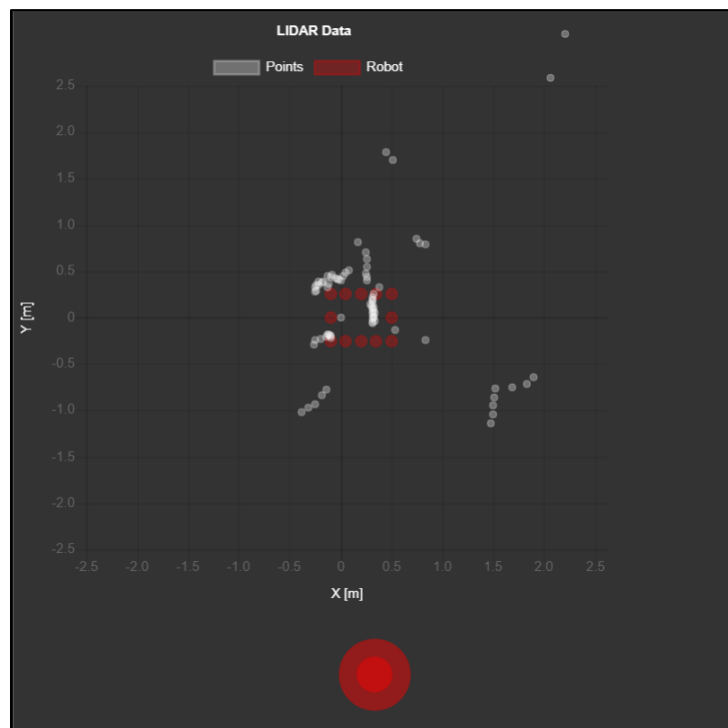


Figure 17: Driving with LIDAR Node-RED Dashboard

Conclusion

The implementation of LIDAR-based obstacle detection on the SCUTTLE robotic platform demonstrated the capabilities and limitations of time-of-flight laser sensing in mobile robotics. The system successfully acquired and processed distance and angle data, enabling real-time identification and tracking of obstacles. By converting polar measurements into Cartesian coordinates, the system provided a spatial representation of the environment, which was then visualized using Node-RED. The LIDAR data proved effective in detecting nearby obstacles; however, performance was affected by surface reflectivity, object geometry, and angular resolution constraints. The integration of Python scripts for data acquisition and processing emphasized the importance of structured programming in robotic perception systems. The experiment highlighted the role of LIDAR in Simultaneous Localization and Mapping (SLAM) and autonomous navigation, reinforcing its application in real-world robotic systems.

Post-Lab Questions

1. What type of LIDAR sensor are we using?

The TiM561-2050101 from SICK calculates the distance of objects by emitting light in a 270-degree range and recording the time of flight of the laser pulse.

2. How does LIDAR work? Include reference pictures and technical data to support your explanation.

LIDAR (Light Detection And Ranging) measures the distance of objects from the sensor, within a detection range dependent on the specific sensor in use, by emitting short laser pulses and using the laser pulse's time of flight to determine the object's distance. The distance is determined by multiplying the speed of light (3×10^8 m/s) by the time of flight and halving the product to find the one-way distance of the detected object. The SICK TiM561-2050101 emits an infrared (850 nm) laser and utilizes a 15 Hz scanning frequency over a 270-degree aperture angle. The scanning range is typically 8 meters with a maximum of 10 meters. This process is repeated over the 270-degree field of view (horizontally) and measures the distance and angle of detected objects relative to the sensor. An array of distance and angle measurements is produced from this operation. The following figures support this description as seen in Figure 18 and Figure 19.

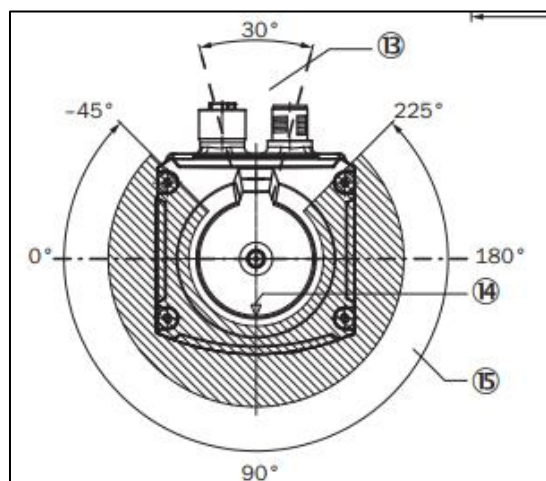


Figure 18: LIDAR Schematic Showing 270 Degree Field of View

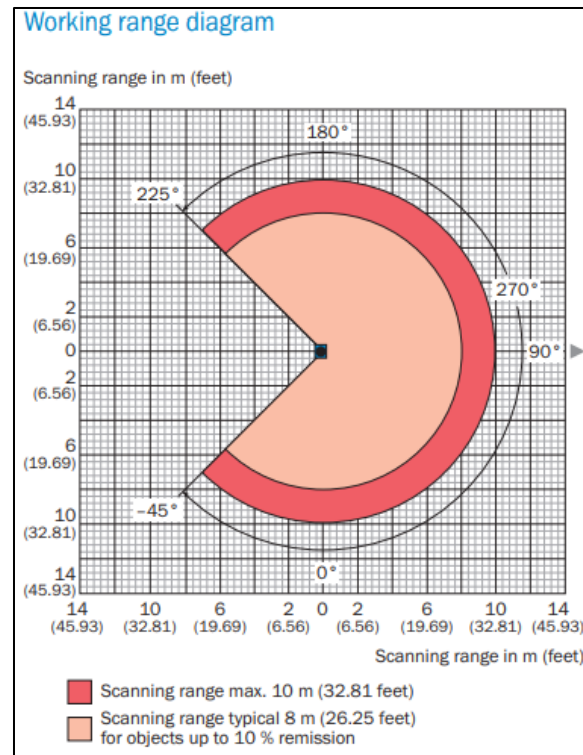


Figure 19: Working Range Diagram of LIDAR Showing 270 Degree Field of View

3. What type of objects influence the reliability of your LIDAR readings?

LIDAR can be influenced by an object's geometry and physical properties such as reflectivity, surface texture/angle, and shape. An object with a highly reflective surface can return a deceptively strong signal, however, one with a very dark surface can absorb portions of the signal, leading to inaccurate readings or none at all. An object's surface texture/angle, especially objects with shallow incident angles or irregular textures, influences the LIDAR readings by dispersing the laser pulses and decreasing the measurement's accuracy. Objects with holes facing the LIDAR sensor influence the LIDAR readings by recording an increased time of flight and subsequently larger distance measurements.

4. When visualizing the LIDAR point cloud, the robot frame remained static while points around the robot moved. Explain this briefly using terminology from class.

The robot frame remained static while the points around the robot moved because the robot frame was referenced as the point of origin from which the distance of detected objects was calculated. Every object's distance measurement is done relative to the sensor which is fixed to the robot.

5. What does LIDAR resolution mean? How many points does the LIDAR return in this lab for each reading? Find the SICK TiM561's datasheet online and determine the maximum number of points it can output.

LIDAR resolution is the level of identifiable detail within scanned data. The LIDAR returned 54 points for each reading. The SICK TiM561 can output a maximum of 811 points.

6. What is SLAM? Why is LIDAR a useful technology for this technique?

SLAM refers to a process of Simultaneous Localization And Mapping, where a robot creates and updates a map of the surrounding environment while tracking its global position within the environment. LIDAR is useful for implementations of SLAM for many reasons; it can provide precise distance measurements in real-time, quickly scan large fields of view for accurate object detection outside the robot's driving path and produce highly accurate/useful point cloud data for further analysis.

Appendix

Lab 6 Commands Used:

- wget L1_lidar.py
- sudo python3 L1_lidar.py
- wget L2_vector.py
- sudo python3 L2_vector.py
- touch L3_logs
- log.tmpFile(voltage, "voltage_log.txt")
- log.tmpFile(B[0], "L2_dist.txt")
- log.tmpFile(B[1], "L2_angle.txt")
- log.tmpFile(C[0], "L2_x.txt")
- log.tmpFile(C[1], "L2_y.txt")
- git add .
- git commit -m "lab6"
- git push

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:

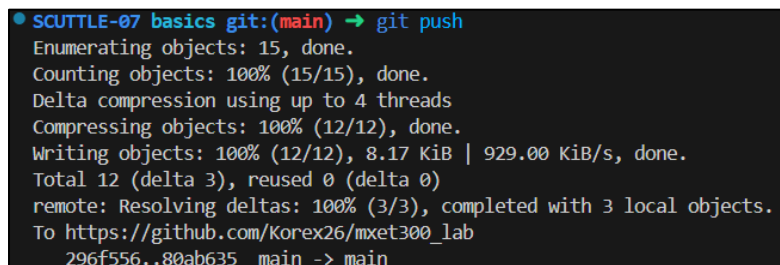
- cd basics/
- wget
- python3 L2_compass_heading.py
- python3 L3_Compass.py
- tmpFile
- stringTmpFile
- git add .
- git commit -m "Lab 3"
- git push

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.

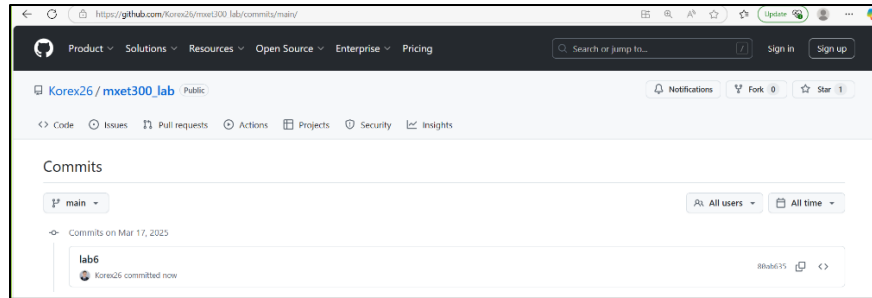


```

SCUTTLE-07 basics git:(main) → git push
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 4 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 8.17 KiB | 929.00 KiB/s, done.
Total 12 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/Korex26/mxet300_lab
296f556..80ab635  main -> main

```

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



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

References

- K. Rex, R. Worley, *MXET 300 Lab Report 5*, 2025.
- 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 6 Manual - LIDAR and Obstacle Detection*, 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.