Lab 6 - LIDAR and Obstacle Detection

Overview

The objective of this lab is to measure the SCUTTLE's surroundings using a LIDAR sensor to: detect the nearest obstacles, determine a vector describing the distance and angle of the nearest obstacle, and finally output the information to a Node-RED GUI. LIDAR is short for Light Detection and Ranging, which is a method for precision range finding based on time-of-flight measurements of light pulses in the ultraviolet, visible, or near infrared spectrum. LIDAR is common in robotics, specifically in: object detection, object classification, and simultaneous localization and mapping (SLAM) at the cutting edge.

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

- SCUTTLE Homepage
- SCUTTLE GitHub
- Guides and resources
- SCUTTLE LIDAR Slides
- SCUTTLE LIDAR Demonstration

Software Needed:

- Python scripts:
  - L1_lidar.py
  - L2_vector.py
  - lidar_driving.py
- NodeRED flows:
  - flow_lidar_driving.json

Preparing the LIDAR Module

The LIDAR module used in this lab is the TiM561 LIDAR from SICK. It covers a 270 degree sector at a maximum of ten meters using an opto-electronic laser scanner. The device functions by emitting pulses of light at regular intervals in different directions. Light emitted hits a surface and reflects back to a photodiode in the sensor. The precise time elapsed between emission and detection can be used, in combination with a precise value for the speed of light, to calculate the distance the light traveled.

In this lab, the sensor pulses at 15 Hz to obtain an angular resolution of 0.33 degrees at 1 meter. This means that for a surface 1 meter away, measurements will be 0.33 degrees apart. This angle increases with distance, making it more difficult to measure narrow objects the further away they are. LIDAR sensors also perform poorly on surfaces that are reflective, dark, or at a shallow angle because the light does not bounce directly back to the sensor.

1. Once the TA has handed out the sensors and given instructions, begin attaching the sensor to your team's SCUTTLE. The sensor should mount to the back of the scuttle facing forward. Ensure no wires or other components are in the sensor's range to avoid self-detection.

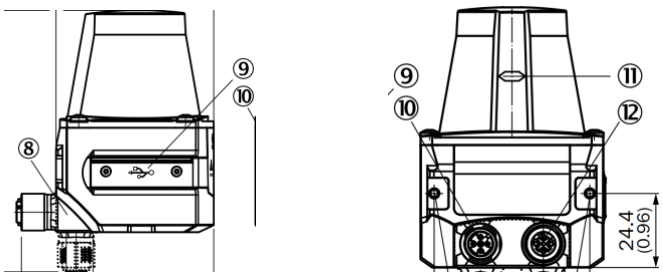

Figure 1: LIDAR Sensor

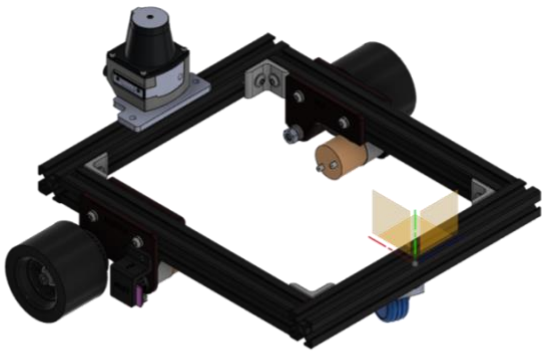

Figure 2: SCUTTLE frame with LIDAR mounted

Take pic of SCUTTLE with LIDAR mounted

2. Connect the LIDAR power cable and micro-USB.
    a. The sensor receives power from the battery pack to run its electronics and a small motor for spinning the photodiode. You will need an adapter to extend the number of ports on the battery for powering motors and LIDAR simultaneously.
    b. The micro-USB is for communication with the host device and transmitting measurements. It should connect to one of the Pi's USB ports.
3. Power up the SCUTTLE, if you have not already done so, and in terminal enter the command: *lsusb*
   Take pic of Lidar in list
   This lists all USB devices detected by the Pi. You should see the LIDAR in this list.
    a. If the module does not appear, listen for the motor inside running. If it is not on, verify that power is connected.
    b. If the LED on the LIDAR module is not green after ten seconds, press and hold the small button on the front of the device for three seconds. If the LED is still not green, ask for a replacement LIDAR.
    c. The LIDAR device might show up with a blank name. Compare the number of USB devices when LIDAR is unplugged and plugged in the USB ports.

Take pic of SCUTTLE with green light

Collect and Process LIDAR Data

With the LIDAR sensor operational, you can now use the *L1_lidar.py* script to obtain measurements from the device. These are output as an array of 54 distance (m) and angle (deg) measurements. Your task will be to test the sensor's output, obtain the angle and distance of the closest measurement, and determine that measurement's local cartesian (x,y) coordinates. *L2_vector.py* provides useful functions for performing the last two steps; carefully read its contents before writing your own L3 script. Finally, this information will be displayed to a Node-RED GUI you create.

1. Download the *L1_lidar.py* and *L2_vector.py* scripts from GitHub.
    a. Run *L1_lidar.py* with sudo as `sudo python3 L1_lidar.py`. Verify that the output is an array of 54 datums, each with a distance and angle.
    Pic of output
    b. Run *L2_vector.py* with sudo as `sudo python3 L2_vector.py`. Verify that the new output is a distance and angle matching the closest obstacle.
    Pic of output
    c. We use sudo here to get root permission to use the USB interface on the Pi to retrieve the sensor's data. Also, all packages are installed in the *python3* environment instead of *python*. Use `python3` run all your python scripts.
2. Now you will create a Node-RED flow and L3 script to log closest obstacle measurements for display.
    a. As usual, import the L1 and L2 scripts needed for your code to run and implement them in L3. Do not modify the L1 or L2 scripts.
    b. Use a bar graph for distance and a compass for angle. The bar graph is a type of chart in Node-RED and the compass is a type of gauge.
    c. Make sure you use the correct units and titles for the data.
    Node red flow pic and dashboard pic

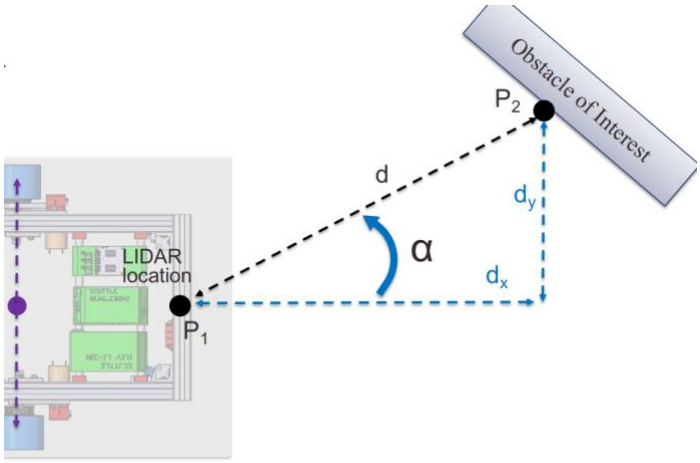

Figure 3: Relationship of (d, theta) with (x, y)

3. Lastly, add code to your L3 script to also log the x and y components of the closest obstacle vector. Refer to figure 3 above.
    a. The distance and angle vector can be easily converted to x and y cartesian coordinates. Look at *L2_vector.py* if you are unsure how to do this.
    b. Add two line charts to your GUI to display the x and y coordinates with the rest of the data. Verify that the signs are correct such that left is positive for y and forward is positive for x. Set the range of the x and y coordinates to a reasonable distance. Set the range of the gauge from -135 to 135.
4. Ask the TA to check your work, then commit and push your Node-RED flows and code to GitHub.

Node red flow pic and dashboard pic

Driving with LIDAR

In this final exercise, you will use an existing NodeRED flow and Python script to visualize the full LIDAR point cloud for manually driving your SCUTTLE robot. This demonstrates how a fully autonomous robot might perceive its environment and use LIDAR for navigation. While driving, pay attention to how the point cloud changes for close and distant obstacles.

1. Download the *lidar_driving.py* script and *flow_lidar_driving.json* flow from GitHub.
    a. Ensure the Python script is in the same directory as your other L1 and L2 scripts, since it depends on some of them for actuating the motors.
    b. The script is written in an "object oriented" format and uses a class to contain its functions and variables. This is a common and useful way to structure programs with multiple modules that interact with each other and is especially helpful for robots, which typically have multiple interdependent modules. The use of L1 and L2 can mimic this without Python syntax, but lacks many of the features of objects in Python. If you would like to learn more about object oriented programming in Python, you can watch this video.
    c. Import the flow to NodeRED and deploy it.

Node red flow pic and dashboard pic

2. Run the script using: `sudo python3 lidar_driving.py`
    a. Sudo is still needed because the script uses LIDAR, which requires sudo to interact with the USB interface on the Pi.
3. Change the theme style in your NodeRED editor to dark mode because the white obstacles would only be visible in the dark theme instead of the light theme.

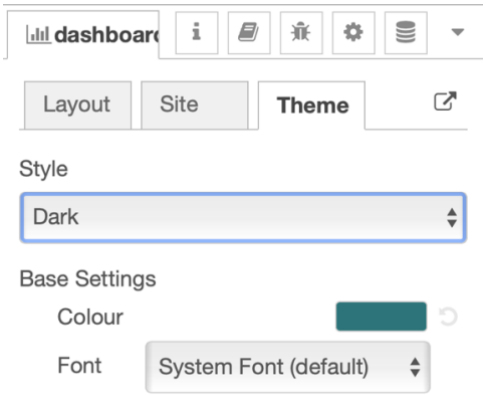

Figure 4: Change NodeRED theme style from light to dark

4. Open your NodeRED dashboard and find the tab containing the LIDAR visualizer, which should look similar to figure 5.
    a. This flow will function on your mobile device; however the chart does not scale properly to a small screen. It is recommended to test this on a laptop or tablet.
    b. The red points are a rough estimate of the robot's frame, and the white points indicate the LIDAR readings.

Node red dashboard pic

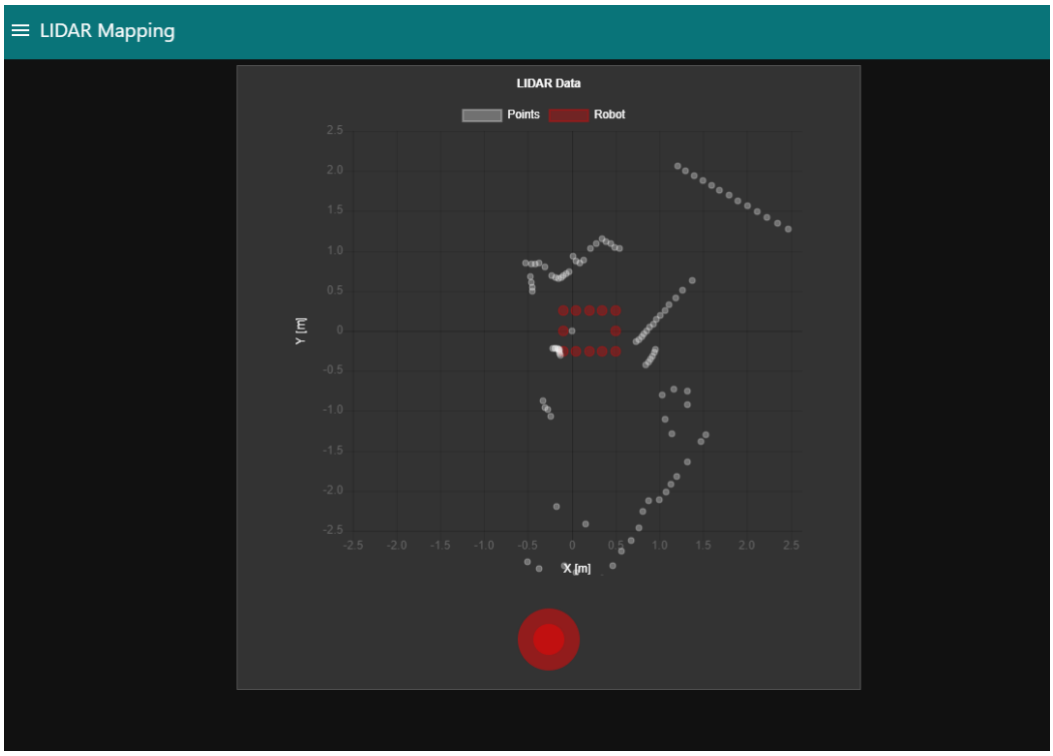

Figure 5: LIDAR Visualizer Example

5. You should now be able to move the joystick to drive the SCUTTLE and see the point cloud change as you navigate.
    a. Notice how distant points are more spread out than nearer points.
6. This concludes the lab. Make sure your work has been saved and pushed to GitHub before leaving.

Take SS of push in terminal and SS of github update

Post-Lab Questions

1. What type of LIDAR sensor are we using?

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

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

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.

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.

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