

Lab 3 – Compass Calibration

Overview

This lab will prepare you to determine your absolute orientation of your robot. Since the encoders only offer information on change in direction, it is necessary to use the compass to find the heading angle of the SCUTTLE robot. The compass is a 3-axis magnetometer, used to measure the strength and direction of the magnetic field in the vicinity of the Raspberry Pi. The compass is embedded in the IMU (Inertial Measurement Unit) of the board. The IMU is an BNO055, a 9-axis Motion Tracking Device that combines a 3-axis gyroscope, 3-axis accelerometer and 3-axis magnetometer.

Resources:

The following are links to useful resources for this lab:

- [MXET300-SCUTTLE GitHub](#)
- [SCUTTLE homepage](#)
- [Guides and resources](#)

Software Needed:

- [Templates:](#)
 - Magnetometer_Compass_Calibration.ipynb
- [Python scripts:](#)
 - Magnetometer_Compass_Calibration.ipynb
 - L2_compass_heading.py
 - L1_log.py

Calibrate Magnetometer

In this lab, you will receive your team's SCUTTLE chassis, which is made of 3030 aluminum t-slot extrusions, wheel assemblies, and a DIN rail for mounting the control boards to. All the other necessary components for assembly have already been provided to you within your clear container kit you received in Lab 1.

1. After receiving some setup instructions from the TA. Mount the raspberry pi and battery pack to the DIN rail on the SCUTTLE chassis. Be sure the raspberry pi and battery back are positioned as they are shown in figure 1.



Figure 1: Pi and Battery Pack configuration (left: front, right: rear)

NOTE: For this lab, you will only need to mount the raspberry pi and battery pack to the SCUTTLE chassis. The motor driver bracket and wiring will be added in the next lab.


2. Once your raspberry pi and battery pack are mounted and connected, power on the pi.
3. When the pi has completed powering on and connecting to the network, open the VS Code and using the terminal, `cd` in to your *basics* directory/folder.
4. In another tab within your web browser, navigate to the “[templates](#)” directory in the MXET300-SCUTTLE GitHub repo.
5. Use `wget` to retrieve the “Magnetometer_Compass_Calibration.ipynb” jupyter notebook file into your *basics* folder.
6. Next you will open the jupyter notebook file. Create a new web browser tab and like going to going to VS Code and Node-RED, navigate to the address in your browser: `scuttle-##:8888` or `192.168.1.1##:8888`

Remember, replace `##` with your lab kit number, which is the two-digit value labeled on the side of the pi's ethernet port.

This will open Jupyter Lab which is a web-based IDE. Though it is better supportive to the jupyter notebook file type and their capabilities.

7. Using the Jupyter lab's file explorer, navigate to and open the "Magnetometer_Compass_Calibration.ipynb" notebook.
8. Now, move your SCUTTLE to an open area which does not have electronics around the SCUTTLE. In FERM 006 lab, the carpet area at the entrance would be great.
9. Here you will run each Jupyter notebook cell one by one to calibrate the magnetometer. Click in each cell and hit "Shift-Enter" on your keyboard to execute the cell. Run the first and second cells under the "Reading the Raw Magnetometer Values" section and observe the output. You will see plots of the raw magnetometer readings over time.

NOTE: If the animation does not start after running the cell, try to execute the cell again. If the problem persists, restart the Jupyter Notebook kernel. Click the "kernel" tab on the top of the notebook. In the drop-down menu, click "restart kernel and clear output".

NOTE: For all animation in the Jupyter Notebook, the animation would continue until you hit the power icon  on the top right corner. If all animations are not stopped, it would continue and consumes a lot of power. Refer to the figure 2 for detail.

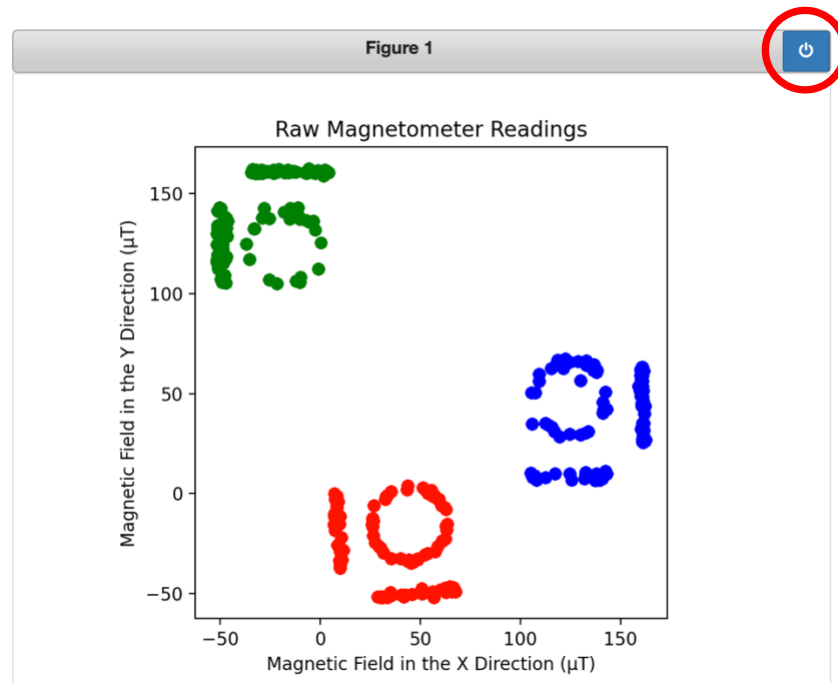


Figure 2: Stop the animation by clicking the top right power icon.

10. Now you will need to calibrate the magnetometer this will be done while running the next cell of code. Run the next cell of code. This is the first cell under the “Using Magnetometer Reading To Get A Compass Heading”. While it is running you will use your hands to gently and at a moderate pace rotate the entire SCUTTLE 10 times clockwise **AND** 10 times counterclockwise about **ALL three** of its axes (x,y,z). Place your SCUTTLE in a swivel chair like it is shown in figure 3 and rotate the chair so that the SCUTTLE rotates about as if it were a compass.

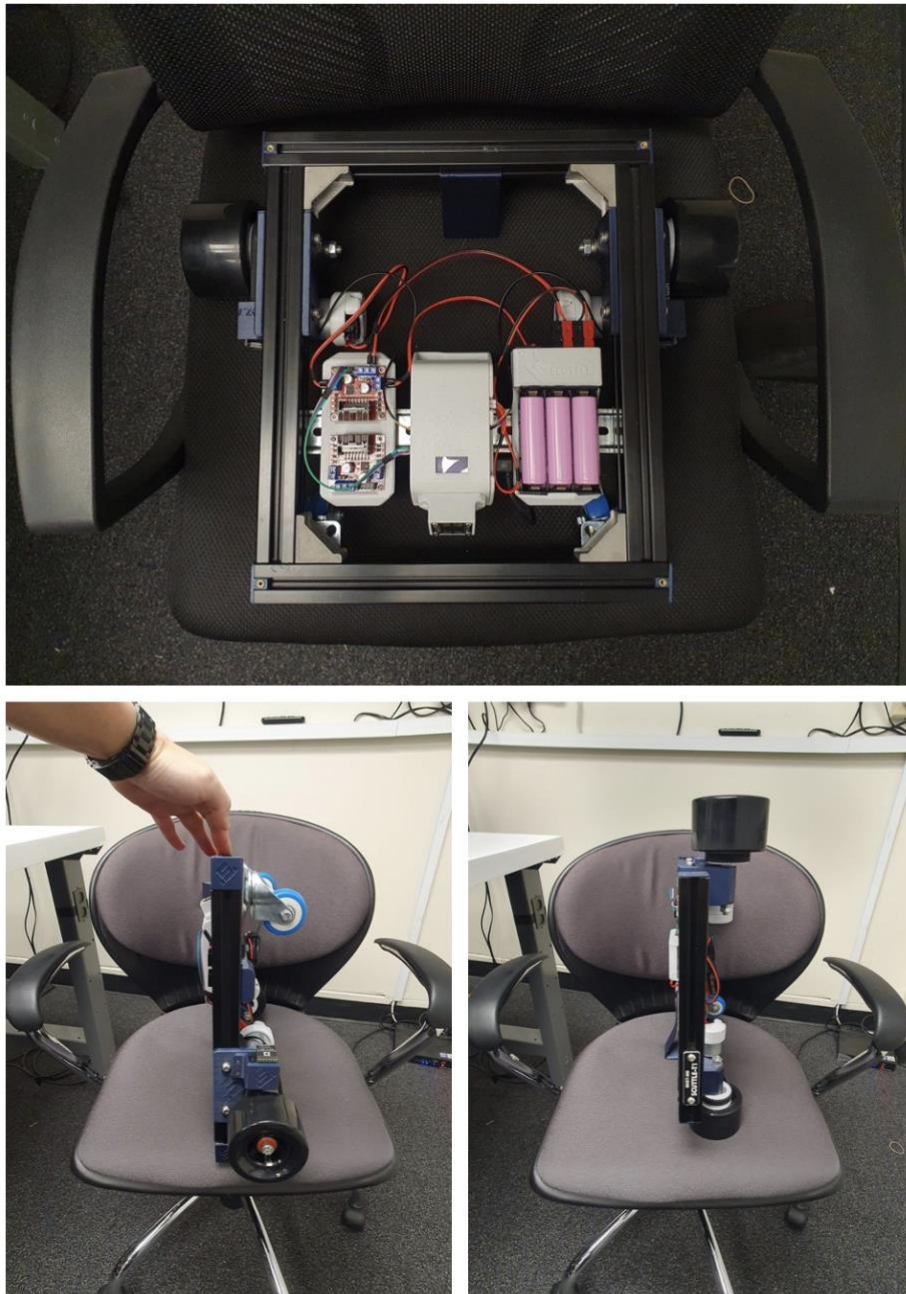


Figure 3: Put your SCUTTLE on a swivel chair and gently
Top: Rotate the IMU Z axis by placing **SCUTTLE flat** on the chair
Left: Rotate the IMU X axis by placing **shorter** extrusion on the chair
Right: Rotate the IMU Y axis by placing the **longer** extrusion on the chair

11. Run the next cell of code under the “Calibrate the Magnetometer” section. Observe the output, both the plot and print statement. In the plot, the calibrated readings are normalized between -1 to 1.

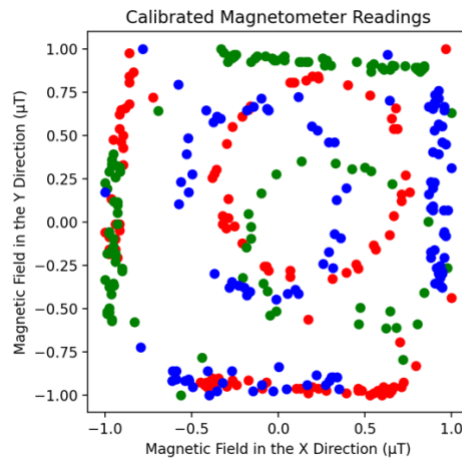


Figure 4: Calibrated compass readings are normalized between -1 to 1

NOTE: The “Final calibration in uTesla:” is the calibrated magnetometer values. Make note of this as you will need to use these values in “L2_compass_heading” python file line 24 to calculate and display the compass heading in degrees.

12. Run the next cell under the “Validate Calibrated Magnetometer Readings” and repeat the same rotation process you did in step 10. The output figure should be similar as the figure calibrated magnetometer readings above. If we see three axis circles overlapped and all the new measurements are normalized within -1 and 1, the validation is completed.
13. Run the final cell under “Calculate A Compass Heading”. While the final cell is running, place your SCUTTLE in a swivel chair like it is shown in figure 3 (top) and rotate the chair so that the SCUTTLE rotates at Z axis about as if it were a compass. You will be able to see the plot change and see how the compass heading changes. The second subplot shows the X and Y readings intersect respectively. It should be like figure 5.

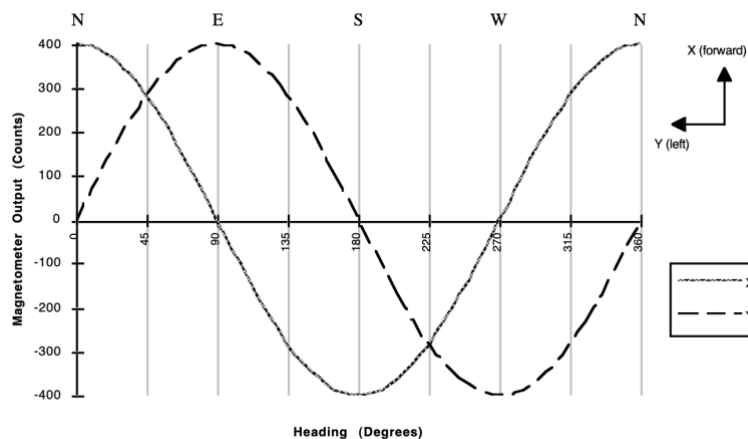


Figure 5: X and Y Magnetometer Readings for Different Compass Headings

Node-RED Compass

With the compass now calibrated, you will now take the calibration data and use it in the compass heading python file to continuously print out the compass heading in degrees. Then create an L3 code and Node-RED dashboard that will display a compass and its heading.

1. Navigate back to VS Code and within the terminal make sure you are within your *basics* directory.
2. Use `wget` to retrieve the “[L2_compass_heading.py](#)” file into your *basics* folder from the *basics* directory in the MXET300-SCUTTLE GitHub.
3. Now open the `L2_compass_heading` script and read through it to understand the comments and code.
4. In line 24, enter the calibrated magnetometer values from the “Calibrate the Magnetometer” output in the jupyter notebook.
5. Save then execute the script using the terminal and `python3`.
6. Repeat the rotation of SCUTTLE while it is in the swivel chair and observe the compass heading print out within the terminal. You will see the compass heading output in degrees

NOTE: You may notice that the readings might sporadically jump or be a little off. This could be due to noise in the environment and/or a poor calibration. You can try moving to a new area and re-calibrating. If the readings are within 10 degrees of accuracy, then it is fine. The point to this lab is to demonstrate the concepts of compass calibration and provide some new understandings. It is ok if the heading is not completely accurate.

7. After verifying the working of the “`L2_compass_heading`”, end the program with “`Ctrl + C`”.
8. Now create your own L3 code that get the compass heading from the “`L2_compass_heading`” and log the values so that you can have Node-RED read and display the compass heading. Create a new python file, “`L3_Compass.py`”
9. In the file, create two functions, one for the lab and one for the post-lab question. Create the first function to retrieve the compass heading from the “`L2_compass_heading`” python file. Create the second function that divide -180 to 180 heading degrees to 8 cardinal directions (North, North West, West, South West, South, South East, East, and North East). Assuming North is centered at zero degrees. Finally, log compass heading using “`tmpFile`” function and log the cardinal directions using “`stringTmpFile`” function from `L1_log` within a while loop.

10. Finally, create a Node-RED dashboard that reads and displays the compass heading using a compass gauge and cardinal directions using a text node. Make sure the gauge has the correct unit and title.

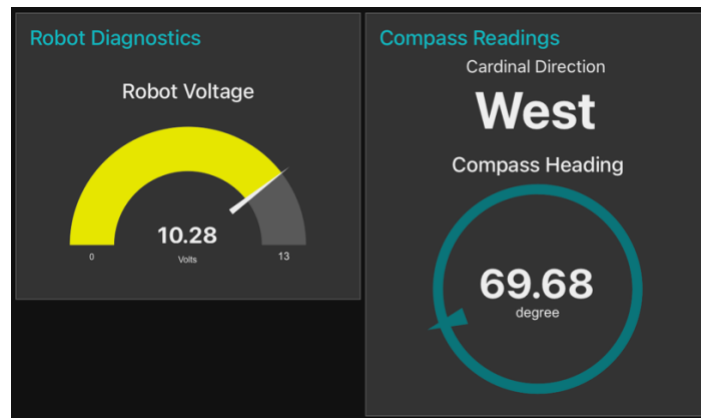


Figure 6: Example Node-RED gauge on dashboard

11. Deploy your flow and run the L3 script. Verify that your program works and ask the TA to check you off.
12. Once checked off, grab screenshots of the dashboard data for your report, export your Lab3 Node-RED flow and save it to your local repository, commit the changes, and push them to your team's remote GitHub repository.

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. Transform the heading number into cardinal directions (N, NE, E, SE, S, SW, W, and NW)
 - a. Create a **function** in your L3 code to decide in which cardinal direction the Raspberry Pi is heading. A function that takes heading as input and returns the heading as N, NE, E, SE, S, SW, W, and NW. (Hint: North is 0 degrees and south is both 180 and -180 degrees. West and east should have positive and negative degrees respectively.)
 - b. Modify your while loop to log the cardinal direction as a .txt file.
 - c. Display cardinal direction and heading value in Node-RED. Be sure to include proper labeling and units for your gauges.
 - d. Be sure to push all your updated work to your team's repo. This includes your updated python file(s) and Node-RED flows.
2. For what orientation (local/global) is a compass required?
3. Before calibration, the raw data from the compass shows the following minimum and maximum values:

Before Calibration	Min (microtesla)	max (microtesla)
x_c	-20	50
y_c	-32	25

If the raw data reading from the compass (before calibration) shows that $x_c = 15$; $y_c = -10$; what is resulting scaled x_c and y_c values? Please be sure to show the general equation and your work.