Lab 1 Pre-Lab Activity

Prelab: Command Line and Python on your own

Task 1

Log onto LinkedIn Learning using your TAMU credentials. Access the course titled *"Learning Linux Command Line".* Watch the following chapters. Take the chapter quiz for each chapter and save progress to show lab instructors at the beginning of Lab 1.

1. Chapter 2: Command Line Basics
2. Chapter 3: File, Folder, and Permission
3. Chapter 4: Common Command-Line Task and Tools

Log onto LinkedIn Learning again and access the course titled *"Learning Python*". Watch the following topics under Python Basics chapters. Take the chapter quiz and save progress to show lab instructors at the beginning of Lab 1.

1. Variables and expressions
2. Python Functions
3. Conditional Structures
4. Loops
5. Classes
6. Importing and using modules

Python, Linux, and MORE

Overview

Welcome to the beginning of this adventure, my name is Jorge Roa and yes you can call me George. By the end of the semester, I expect you to know how a mobile robot works, understanding not only their hardware but also software. This lab illustrates the meaning of being a mechatronics engineer, mastering both the mechanical and electronics fields. There is no doubt that Lab 1 will be the most difficult, because you are required to learn about python language and the Linux environment, which will be fundamental not only for your success in this class but also as a mechatronics engineer.

The objective of this lab is to introduce python, Linux, and Cloud9 IDE. We will be talking about the BeagleBone Blue during the next lab. These tools will help build a solid foundation on how mobile robots are programmed and used.

Resources

Video

- Flash a Debian Image
- Linux Root, Home, directories
- Connect by USB & find SSID
- wget
- SCUTTLE videos

Web Page

- SCUTTLE home page

Resources Required

- MXET 300 Lab Kit #1 (TA will give it to you)
- Personal Laptop (at least one per team)

Prelab: Command Line and Python on your own

1. Activity attached on a different document.
2. Please come to class with the following downloaded
    a. Download the following Debian Image(15GB)
    b. SD card Formatter
    c. Belena Etcher
    d. Putty Download Link

Lab

Task 1: Install the MXET 300 Spring 2021 Debian Image

A computer "image" contains the full operating system and file system for a standalone computer like your Blue. Debian is an operating system based on the Linux kernel. The BeagleBone Blue can run Debian directly from its onboard hard drive but it is easiest to install the OS to an SD card by flashing the card and then booting the BBB (BeagleBone Blue) from the micro-SD card.

1. Your BeagleBone must be secured in the 3D printed bracket using the black M2 plastic screws as shown in Figure 1.

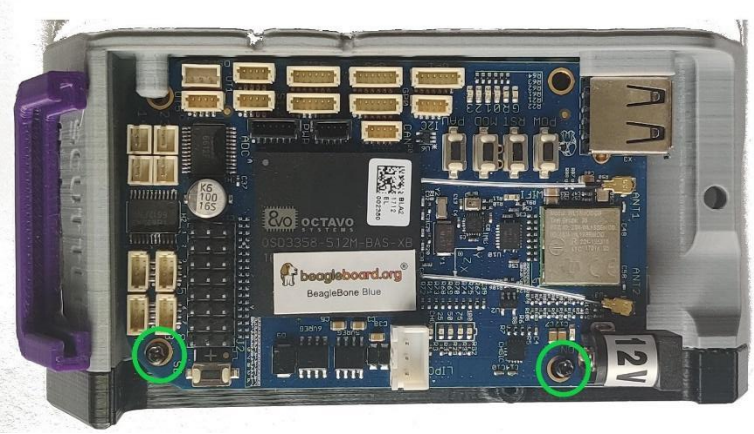


Figure 1

2. Flash Debian image:
    a. Download the Image. (same link as in the prelab activity section above)
    b. Format your SD card before using it to flash image using SD card Formatter or any software of your preference.
    c. Flash your SD card.
    d. Insert into your BeagleBone (**while powered off**).
    e. Boot from the SD card using the following process:
        i. Hold the SD button on your BeagleBone, connect your beagle micro-USB to your PC USB port to power on, and release the SD button as shown in Figure 2.

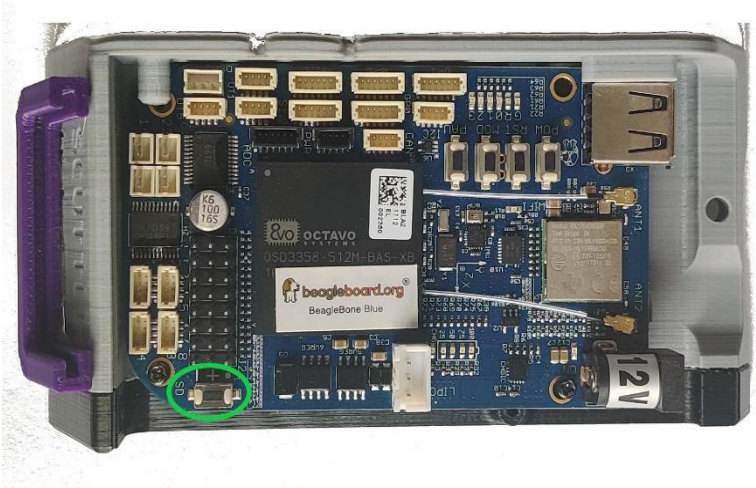


Figure 2

f. Follow the steps in the Video to open a terminal session with your BeagleBone and retrieve your BeagleBone's Wi-Fi SSID.
   i. Host IP: 192.168.7.2 Port: 22
   ii. Username: **debian**
   iii. Password for Debian: **temppwd** **(Password will not show)**

g. Memorize your BeagleBone's SSID, you will be using it every time you interact with your BeagleBone.

h. Once you have obtained the SSID, disconnect the BeagleBone from your computer.

Task 2: Access Cloud9 & Connect BeagleBone to Wi-Fi

You have successfully flashed the latest Debian image onto your blue. Now let us connect to the board, and setup the Wi-Fi connection.

1. Boot up your blue again but this time use a 12.0v DC power supply. After 1-minute look for an access point with your SSID such as "*BeagleBone-###]"* on your PC. Connect your PC to this hot spot using the default password: **BeagleBone**
2. Navigate in your browser to 192.168.8.1:3000 to find the Cloud9 IDE.
3. Inside the terminal as shown in Figure 3, type *"cd /opt/scripts/network"* to change directory to the optional software available within Debian. Then type *"sudo bash wifi_enterprise.sh"* to execute as root a bash program that will assist you to connect to an enterprise Wi-Fi network such as Texas A&M Wi-Fi.

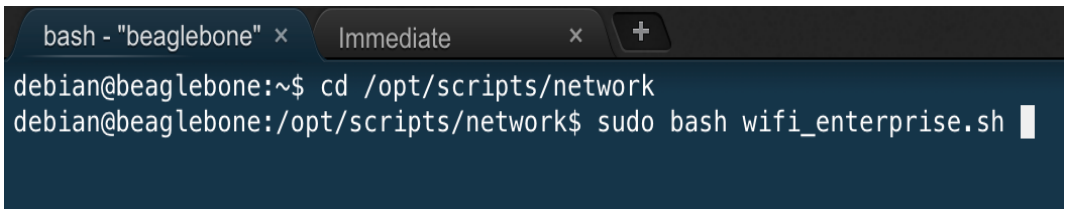


Figure 3

4. Once the program is executed it will show you a list of Wi-Fi networks available such as in Figure 4. Please enter the number assigned to the Texas A&M student network in the terminal to continue.

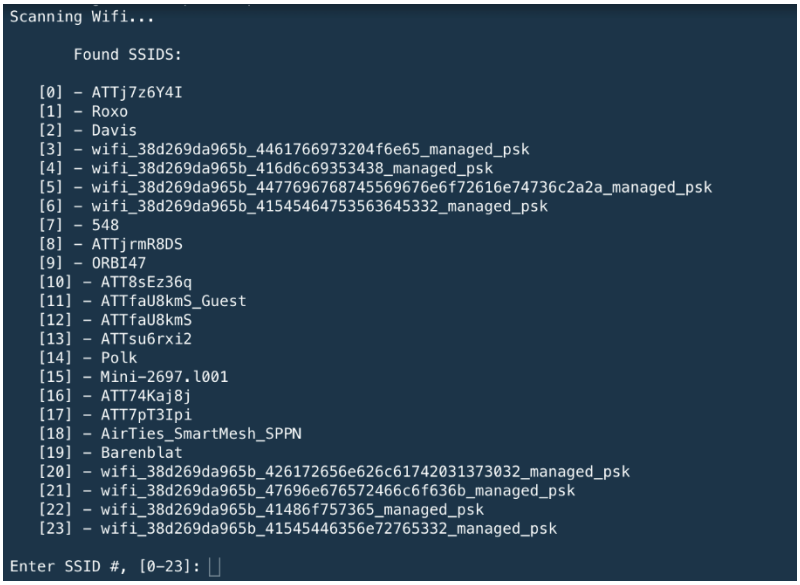


Figure 4

5. Then program you will be asked for your student credentials. Please input your username and password (keep in mind that the password is invisible). Once you have finish, your BeagleBone should have connection to internet. Verify you have successfully connected by typing “ping google.com” if your connection was successful you should see something like Figure 5.

Figure 5

Post-Lab Activity

No post lab activity or lab report. *Prelab activity will count as the Lab 1 grade!*

Lab 2: Onboard Sensor & GUI

Overview

This lab will walk you through accessing the onboard sensors of the BeagleBone Blue. This board has an **accelerometer, temperature sensor, analog-to-digital converter, gyroscope, magnetometer, barometer, and an altitude sensor**. You will learn how to access all the onboard sensors and display its data using a prebuilt graphical user interface called NodeRed. By learning what data comes out of each sensor, you will have more tools to use when developing your mobile robot.

References

Video

- SCUTTLE Software Architecture
- How to NodeRed with BeagleBone
- SCUTTLE Videos

Web Page

- SCUTTLE Software Guide
- SCUTTLE Home Page
- SCUTTLE_GITHUB

Resources Required

- MXET 300 Lab Kit #1
- PC

Prelab

Task 1: Get familiar with the SCUTTLE software architecture

Go over the SCUTTLE Software Architecture slides found within the SCUTTLE Software Guide and watch the SCUTTLE Software Architecture video explaining the different levels of software used in the SCUTTLE robot. The material will also be discussed in lab. Become familiar with the software architecture and format as you will be required to create similar slides to explain how your final project software will be structured.

Lab

Task 1: Access Sensor Data

Level 1 programs are intended to access data from a sensor or send commands to an actuator. In this task you will be looking at how to access data coming from different onboard sensors and print it on the terminal. As mentioned before, the BeagleBone Blue board has some extremely useful sensors that eliminate the need of having to add external sensors to your project.

1. Turn on your BeagleBone using the 12V DC power supply and connect to Wi-Fi
2. Make a folder in your home (~) directory named "*basics*."
3. Using the *wget* program, download the following file to the *basics* folder created in step 2. You can find the files in the SCUTTLE GitHub
    a. L1_mpu.py
    b. L1_adc.py
    c. L1_bmp.py
    d. L2_log.py
    e. L3_telemetry.py
4. Prepare the first script called L1_mpu.py for running in the BeagleBone
    a. Open the program using Cloud9
5. Open a terminal on Cloud9 if there is not one already opened
6. In the terminal, execute L1_mpu.py using Python3
    a. Using terminal, change directory to the basics folder
    b. Run the program using the command "*python3 mpu.py*."
    c. Observe the output on the terminal, understand the units and magnitudes of the values
7. Repeat with the rest of the L1 programs in the same manner
    a. Before running, please read the comments at the beginning of the scripts, you might be required to install some libraries before executing the code. In addition, the comments might include useful information to help you understand the code

Task 2: Output Data to NodeRed GUI

This task will demonstrate how the three different programing levels interact with each other. The L1 will be accessing data from the sensors, the L2 will be passing data to log files saved on the hard disc of your Blue, and the L3 program coordinates all the action. Once the L3_telemetry.py is running, you can configure NodeRed to access these variables in real time.

MXET 300 Mobile Robotics **Lab 2**

NodeRed is a graphical programming interface like LabView, and it is running by default on your Blue's Debian image. You "deploy" your NodeRed flow and lastly, view the dashboard for real time data in your web browser.

1. Run the L3_telemetry.py file from inside your basics folder
    a. Use the command *"python3 telemetry"*
    b. Verify the program runs, some values should begin to print in your terminal These are your x and y values from your BeagleBone's accelerometer
2. While L3_telemetry runs, access NodeRed on using the same browser as you use with Cloud9
    a. In your browser, go to 192.168.8.1:1880
3. Create a flow, which will access the data that is being logged by *L3_telemetry.py* from your BeagleBone
    a. L3_telemetry.py overwrites a text file every 20ms with new values from the accelerometer. NodeRed will access the text file, grab the value, and then display it using its dashboard
    b. Refer to the How to NodeRed with BeagleBone video for building your own flow.
    c. Create a chart for the x and y axes of accelerometer
    d. Include chart titles that name the axis and units
    e. Scale the charts y axis, from -10 to 10
    f. Verify that the x and y printed on the silkscreen of your Blue match the outputs of your charts, and the magnitudes correspond to true gravitational force
4. For lab **check off**, you need to show the lab TA two charts for the x and y accelerometer values.

Post-Lab Activity (Due Before Next Lab)

Lab Report Format:

- Introduction
- Procedure Summary/Objective
- Hardware/Software used
- Analysis/Discussion
- Conclusion
- List and short description of Linux commands used during lab (appendix/reference list)

Please include the following information in the lab report:

1. What is output by mpu.py?
    a. How many values?
    b. What are the units?
2. What is output by adc.py?
    a. How many values?
    b. What are the units?
3. What is output by bmp.py?
    a. How many values?
    b. What are the units?
4. Give an example on how you can use at least two of the sensor data covered in a mobile robot application.
5. Show your graph or gauge for the following data, make sure to include chart tittle and proper units.
    a. Voltage at barrel plug connection.
    b. Temperature, Pressure, and Altitude.
    c. Board Acceleration on x, y, and z axis.

Lab 2: 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 BeagleBone Blue. The compass is embedded in the IMU(Inertial Measurement Unit) of the board. The IMU is an MPU-9250, a 9-axis Motion Tracking Device that combines a 3-axis gyroscope, 3-axis accelerometer and 3-axis magnetometer.

References

Video

- SCUTTLE videos
- Absolute Orientation & Magnetometer explained

Web Page

- Supplemental Documents
- Absolute Orientation & Magnetometer (pg15 – 17)
- SCUTTLE home page
- SCUTTLE GITHUB

Resources Required

- MXET 300 LAB KIT #1
- MXET 300 LAB KIT #2

Prelab

No Prelab Assigned

Lab

Magnetometers are affected by disturbances on magnetic fields. Magnetic disturbances can be caused by hard-iron or soft-iron sources. Hard-iron sources are objects that produce magnetics fields such as motors and magnets, soft-iron sources are objects that are magnetic such as nails and metals. If magnetic disturbances are part of the robot frame and rotates with magnetometer sensor then it can be calibrated out. By calibrating our compass, we find the disturbances ahead of time and correct every sample.

There are two parts to calibrating your compass, the first is built into the *rcpy* library and stores calibration values to files in the library. The second part offers us better control, helps us check accuracy, and requires understanding of the magnetometer functionality.

Task 1: Calibrate through the *rcpy* library

1. Carefully Place your BeagleBone Blue and battery pack on a desk chair
2. Run L1_mpu.py from Lab 2 for a couple of seconds before stopping it by typing ctlr+c
3. Run the command ***sudo rc_calibrate_mag***
    a. Follow the instructions as shown in the terminal window, this will sample the magnetometer for 15 seconds. Rotate the SCUTTLE around the Z-axis as many times as possible to collect sufficient data for calibration.

Task 2: Perform manual calibration

1. Run the command ***rc_test_mpu -m*** to show the magnetometer axes printed on screen.
    a. There should be no warning that calibration has not been performed for **magnetometer.**
2. Next, download and run the loop for ***L2_heading.py*** to discover the x and y range.
    a. Comment some lines from the main while loop to make it look as Figure 1.

```
if __name__ == "__main__":
    while True:
        axes = getXY()                          # call xy function
        print("raw values:", axes)
        # axesScaled = scale(axes)              # perform scale function
        # # print("scaled values:", axesScaled)   # print it out
        # h = getHeading(axesScaled)            # compute the heading
        # headingDegrees = round(h*180/np.pi, 2)
        # print("heading:", headingDegrees)
        # time.sleep(0.25)                      # delay 0.25 sec
```

Figure 1

b. With raw values being printed as shown in Figure 2, rotate your robot and record the max and min achievable magnetometer values for x and y.

```
raw values: [ 66.5     2.927]
raw values: [ 68.273   2.073]
raw values: [ 67.564   0.927]
raw values: [ 67.236   1.209]
raw values: [ 66.536   1.691]
raw values: [ 68.1     3.164]
raw values: [ 66.864   0.973]
raw values: [ 67.173   1.9  ]
raw values: [ 66.755   3.673]
raw values: [ 65.909   2.318]
raw values: [ 65.282  -0.482]
```

Figure 2

c. Overwrite the existing xRange and yRange variables in the L2_heading code and rerun. Figure 3 shows the values that need to be updated for your recorded values. Range varies between BeagleBones.

```
xRange = np.array([-68.182, 68.800])                    # range must be updated for your device
yRange = np.array([-67.836, 68.891])                    # range must be updated for your device
```

Figure 3

3. Verify success of your calibration & output to GUI
   a. While running the heading program, observe that your heading function can range nearly from -180 all the way to 180, and that the cardinal directions (NSWE) correspond approximately to (0, +/-180, 90, -90). Your while loop should look as shown in Figure 4

```
47 if __name__ == "__main__":
48     while True:
49         axes = getXY()                               # call xy function
50         # print("raw values:", axes)
51         axesScaled = scale(axes)                     # perform scale function
52         # print("scaled values:", axesScaled)          # print it out
53         h = getHeading(axesScaled)                   # compute the heading
54         headingDegrees = round(h*180/np.pi, 2)
55         print("heading:", headingDegrees)
56         time.sleep(0.25)                             # delay 0.25 sec
57
```

Figure 4

   b. Your printed value should look like the one shown in Figure 5 when heading North and as shown in Figure 6 when heading South.

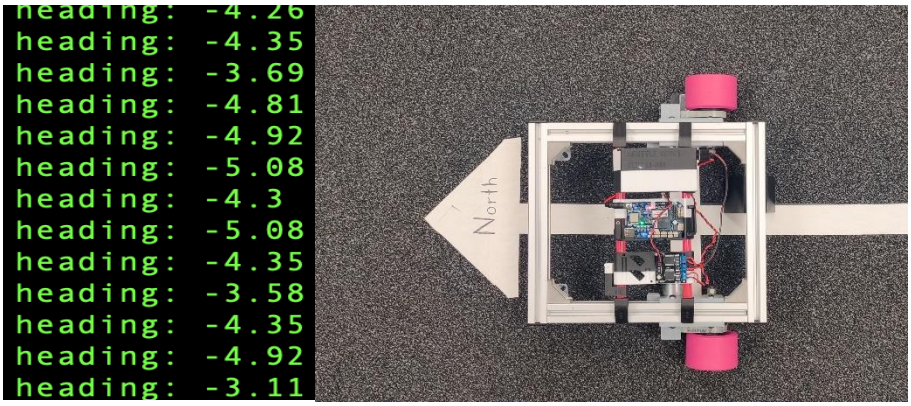


Figure 5

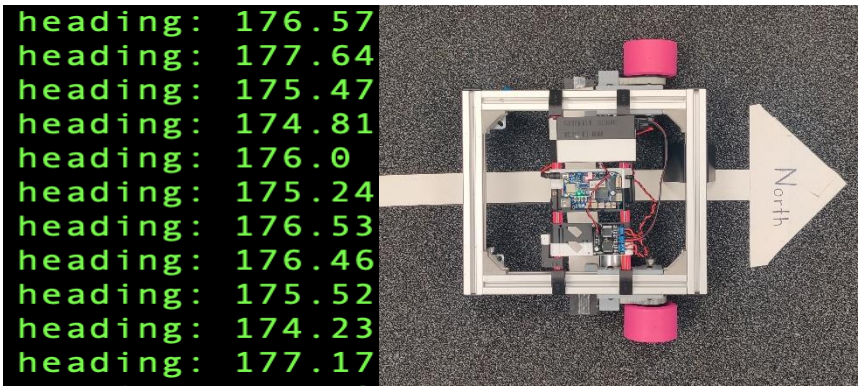


Figure 6

c. Create your own L3 file to log your heading and create an indicator in NodeRed to display the heading number.

Post-Lab Activity (Due Before Next Lab)

1. For your lab report you must transform the heading number into cardinal directions (N, E, S and W)
    a. Create a function in your L3 code to decide in which cardinal direction the BeagleBone is heading. A function that takes heading as input and returns the heading as N, E, S or W.
    b. Modify your while loop to log the cardinal direction as a txt file.
    c. Display cardinal direction and heading value in NodeRed.
2. Rubric for python code and NodeRed flow:
    - Proper software (python code and flow) is submitted on eCampus:
        - Code contains names, date, team number, and code purpose.
        - Code for flow is provided & imports properly.
        - Python software creates 2 log files:
            - Heading
            - Cardinal direction
        - Python software runs without errors.
        - Flow displays heading and cardinal direction.

Lab Report Format:

- Introduction
- Procedure Summary/Objective
- Hardware/Software used
- Analysis/Discussion
- Conclusion
- List and short description of Linux commands used during lab (appendix/reference list)

Motor Drivers & Inverse Kinematics

Overview

This lab walks you through operating your motor drivers and the basics of inverse kinematics. Driving the SCUTTLE requires powering your BeagleBone and your motor driver using the battery pack, verifying signal wires from the BeagleBone to the motor driver, and understanding the PWM output from the motor drivers.

You will see how commands get converted from robot speed to wheel speed then to PWM commands to voltage which translates to actual wheel movement.

References

- **Wiring Diagram**
- **SCUTTLE Kinematics**
- **SCUTTLE GitHub**
- **SCUTTLE Home Page**
- **SCUTTLE YouTube**

Resources Required

- MXET 300 LAB KIT #1
- MXET 300 LAB KIT #2
- MXET 300 LAB KIT #3

Lab

Task 1: Wire and Run SCUTTLE's Motor

During this task you will run and understand the level 1 program that operates your motors.

1. Figure 1 shows how the driver and motors connect to the BeagleBone.
2. Following figure 1, carefully connect all your power and signal cables, make sure power is OFF to avoid burned components.
3. Ground on the motor driver signal has no connection, the motor driver power source must share a ground with the BeagleBone power source.
4. The wire colors, pin sequence, Left/right assignment, and positive/negative assignments must match the instructions precisely.

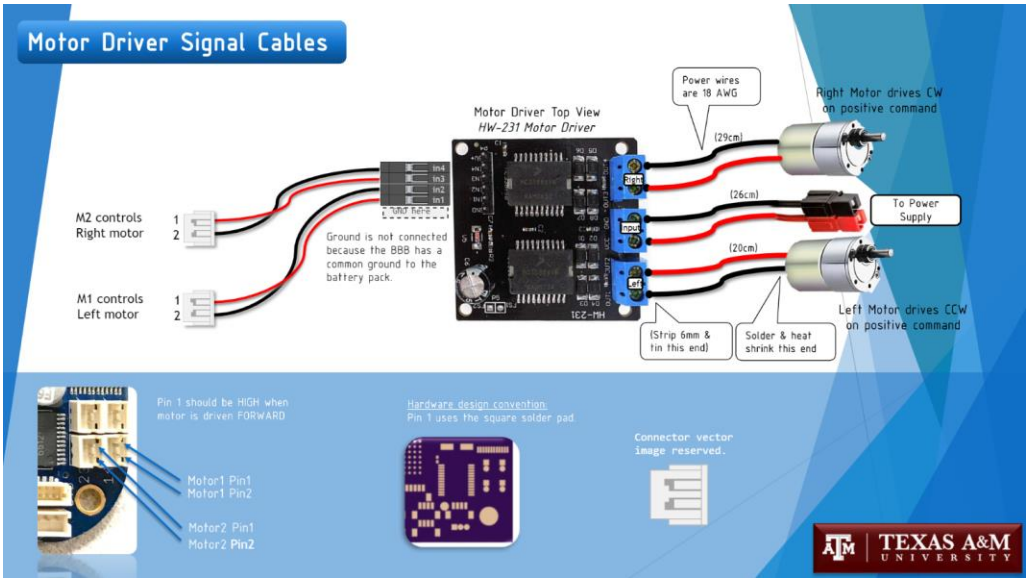


Figure 1

5. Prepare the L1_motors.py file to be executed.
    a. Boot your BeagleBone using the battery pack and connect to its Wi-Fi signal.
    b. **Add the L1_motors.py, L2_speed_control.py, L2_inverse_kinematics.py, and L1_gamepad.py** files to your basics folder from the web using the *wget* command within the BeagleBone terminal.
6. Run the L1_motors.py program with the wheels lifted.
    a. Lift the SCUTTLE frame using a 3D printed stand.
    b. Type *sudo python3 L1_motors.py* in your terminal
    c. Verify that both wheels move forward for 4 seconds, then reverse for 4 seconds.
    d. Show the TA that both wheels move forward and backwards before continuing to task 2.

Task 2: Perform Inverse Kinematics

All driving commands for SCUTTLE will come in the form of $\dot{x}$**_dot (xd) and theta_dot(td)**, the forward and angular velocities of the vehicle. In code the command will appear as **[xd, td]** which is a column matrix.

This task demonstrates how **xd** and **td** values are converted by the L2 program into individual wheel speed values. Before continuing forward, please review the **SCUTTLE kinematics(pg 9)**.

1. Prepare to run the L2_inverse_kinematics.py file.
    a. Use command*: python3 L2_inverse_kinematics.py*
    b. Input various input pairs of **xd** and **td** and observe the output wheel speed in rad/s printed to screen.
2. ***Simulate*** a navigation path as shown in Figure 2 for your robot by driving your wheels unloaded. ***"Please note there is no feedback control in this lab so actual driving is not recommended."***
    a. Note that SCUTTLE's max speed is approximately 0.4 m/s forward. Constrain your speed within this max speed.
        i. You will need to choose d1 and d2 lengths.
        ii. Your SCUTTLE will begin with the green star and end at the red star.
    b. Complete Table 1, by determining x_dot, theta_dot, and duration of each motion. **Include table in lab report.**
    c. Write your own script using the following script as a template: **SCUTTLE GitHub Lab Templates.**
    d. Since open loop control is implemented in this lab, the robots might not execute the expected path as planned.

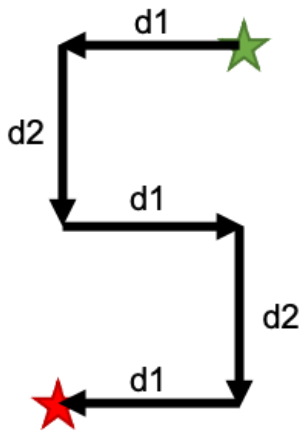


Figure 2

Motion	$\dot{x}\ (m/s)$	$\dot{\theta}\ (rad/s)$	Duration (s)
1			
2			
3			
4			
5			
6			
7			
8			
9			

Table 1

Task 3: Using Gamepad to Drive SCUTTLE

In this task you will use the gamepad to drive your scuttle around. Once you are ready, please ask the TA for a gamepad and a USB dongle.

1. Run the command *lsusb* to check the USB devices currently connected to the board.
    a. Connect the USB Dongle to the BeagleBone's USB port.
    b. Run the command *lsusb* again to check if the gamepad's USB dongle was detected.
2. Download to your basics folder the file named **Lab3_gamepadControl.py.**
3. Turn on your controller pressing the middle button on the gamepad.
    a. Run the program using the command *python3 Lab3_gamepadControl.py*
    b. SCUTTLE should react to the left joystick motion.
    c. Comment to the code, explaining what each line of code does. Explain the flow of data from the joystick to the motor duty cycle. **Make sure to include this information in the lab report.**

Post-Lab Activity (Due Before Next Lab)

Make sure to include:

- Your Lab3_template.py with your information
- Completed table 1 (explain how this was calculated)
- Lab3_gamepadcontrol.py with comments

Task 1: Learn about the NumPy library

- In this and all the future labs we will be using the famous NumPy python library. This library allows for easy manipulation of matrices in python. Data scientists are among the people that use this library the most, due to its speed and scalability. Please take a look at the following link and follow their examples on how to use the library and work with arrays. We are a going to have a quiz at the beginning of next lab **(Lab 5)**

- **https://www.w3schools.com/python/numpy_getting_started.asp**

Please submit your code with your lab report

Individual Lab Report Format:

- Introduction
- Procedure Summary/Objective
- Hardware/Software used
- Analysis/Discussion
    - Make sure explain difference between inverse vs forward kinematics, and open loop vs close loop.
- Conclusion

Encoders & Forward Kinematics

Overview

This lab walks you through operating your encoders, building the i2c communication cables, and utilizing the forward kinematics functions. The encoders provide information about the wheel, the IC measures the absolute position of the shaft's rotation angle, consists of Hall sensors, analog digital converter, and digital signal processing. You will learn about the I2C protocol and it is used to communicate with any I2C sensors, such as the encoder. You will also use a Level-2 program to convert the wheel speed data into chassis speed data **[x_dot, theta_dot].**

References

Video

- **How to export NoteRed Flow**
- **18650 Batteries**
- **How does I2C works**
- Unboxing Encoder Video
- How to plot gamepad values in NodeRed
- Intro to SCUTTLE software architecture

Web Page

- **Wiring Diagram**
- **As5048A Datasheet**
- SCUTTLE Kinematics
- SCUTTLE **home page**
- SCUTTLE GITHUB

Resources Required

- MXET 300 LAB KIT #1
- MXET 300 LAB KIT #2
- MXET 300 LAB KIT #3

Lab

Task 1: Connect and Run the encoders

During this task you will first connect the wires required to operate the encoders and then run the L1 software to read information from the As5048A chip. It's important to keep your robot **powered off** while connecting the wires.

1. Connect the encoder cables(left and right) shown in Figure 1, to the I2C bus board, shown in Figure 2. Make sure that you follow the wiring diagram below.

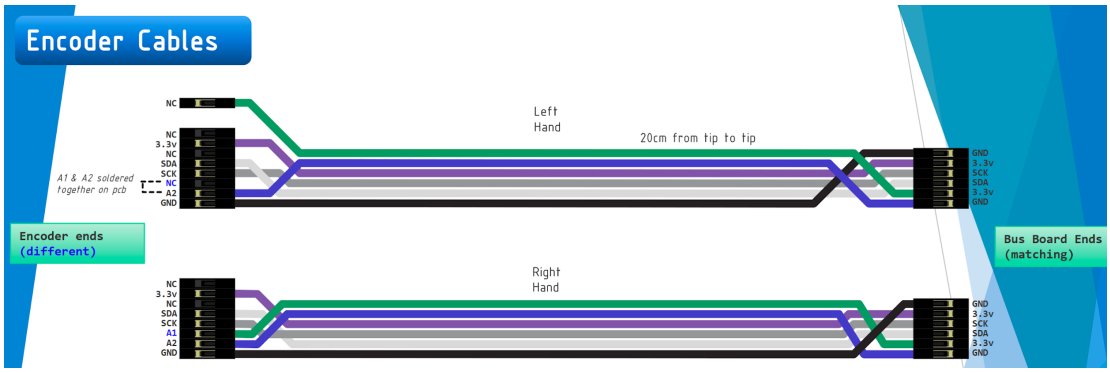


Figure 1 – Encoder Cables

2. Understand your i2C bus board. Shown in Figure 2.

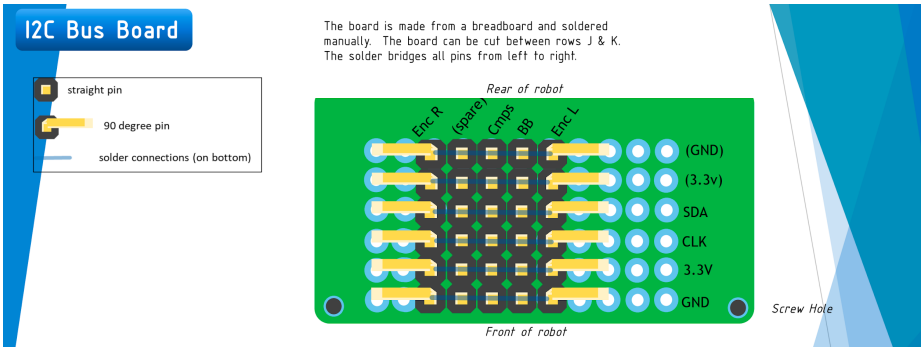


Figure 2 – I2C Bus Board

3. Connect your I2C bus cable from the BeagleBone to the I2C bus board.
    a. We use the 4-pin JST-SH connector as shown in Figure 3.

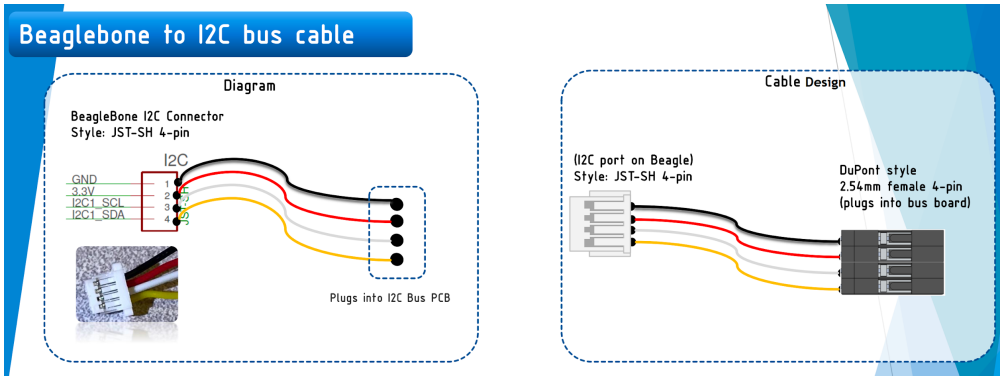
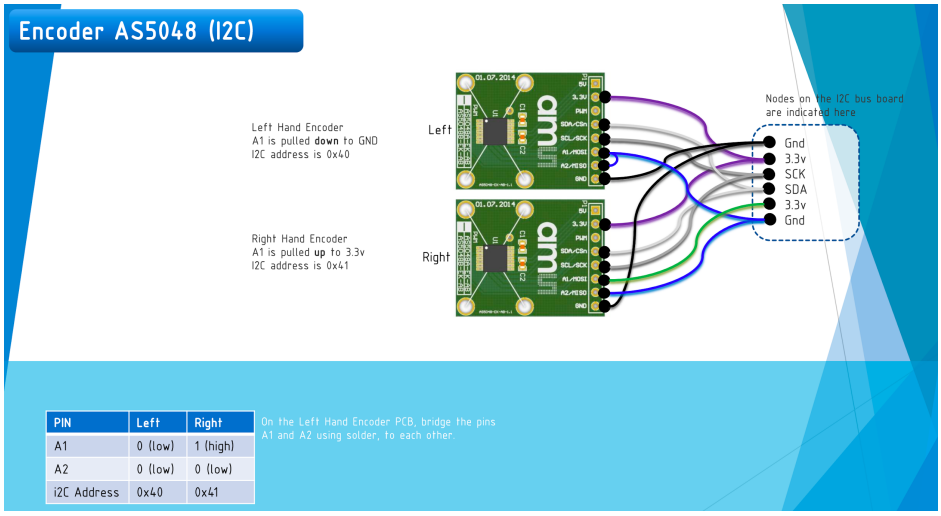


Figure 3 – JST 4 pin to I2C bus

4. Understand how the wheel encoders connect to the I2C Bus board using the encoder cables.



PIN	Left	Right
A1	0 (low)	1 (high)
A2	0 (low)	0 (low)
i2C Address	0x40	0x41

Figure 4 – Encoder AS5048

5. Connect the I2C bus board to the BeagleBone I2C port as shown in Figure 5.

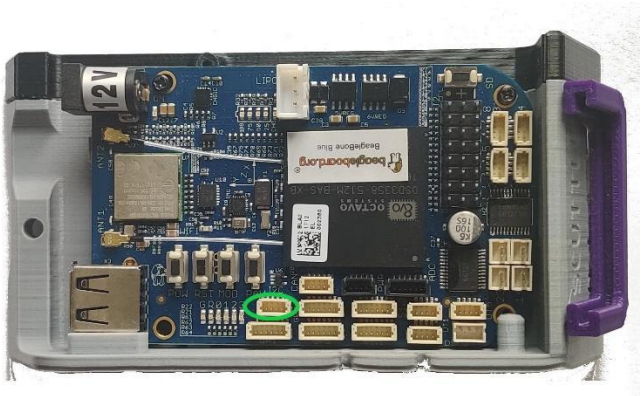


Figure 5 – I2C port location

6. Figure 6 shows the correct pinout for the I2C port. Make sure you refer to it during troubleshooting.

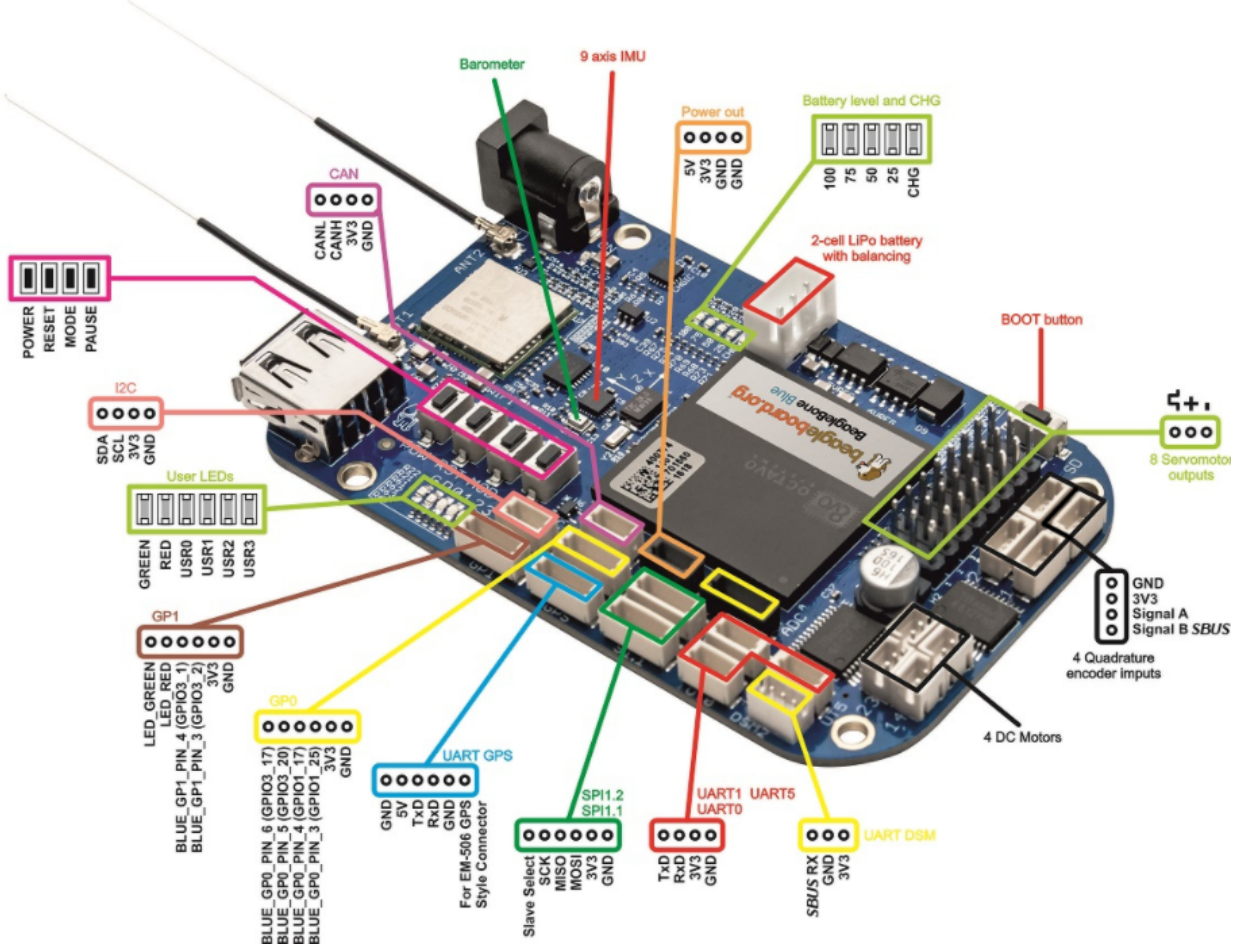


Figure 6 – BBB Pinout

7. **Get your wiring checked off by your TA** before powering the BeagleBone and Motor Driver with your battery pack
8. Run the following command on your terminal
    a. *"sudo pip3 install Adafruit_GPIO"*
9. Once you connect the I2C bus board to the BeagleBone, run the following command on the terminal " *sudo watch -n0.2 i2cdetect -y -r 1* "
    a. The output should look like Figure 7
    b. Connect and disconnect the left encoder, note how address 40 won't show anymore. This is how you check that the BeagleBone is detecting the correct I2C address on each encoder.

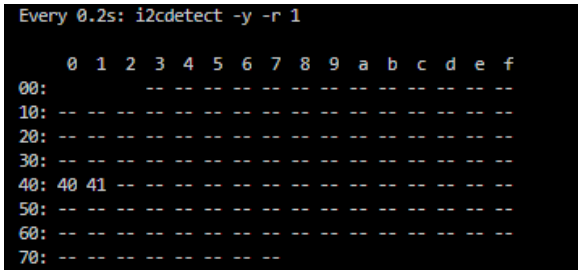


Figure 6 – Expected output

10. Download the **L1_encoder.py** program to your basics folder.
    a. Run the L1_encoder.py program using python3.
    b. Verify that the encoder values are representative of the wheel movements.
    c. Move the wheels by hand while the chassis is lifted
    d. Turning the right wheel forward should produce a climbing value for the corresponding wheel.
    e. Turning the right wheel backwards produces a falling value for the corresponding wheel.
    f. The left wheel is reversed, a forward motion of the wheel will produce a decrease of angle rather than an increase as in the right wheel. This is being taken care of during the calculations of forward kinematics, as we know L1 software just outputs raw data. DO NOT MODIFY L1 OR L2 SCUTTLE CODE. Every modification must be done on L3.
    g. A stopped wheel produces a non-changing value

Task 2: Perform Forward Kinematics

The encoders L1 program returns only static readings. The Level 2 program, L2_kinematics.py, derives two different information pairs from the Level 1 output. The function getPdCurrent() will return information in the form of [pdl pdr], a column matrix describing phi-dot left and phi-dot right. The function getMotion() will interpret the phi-dots and convert them to chassis movement, giving [xDot theta_Dot]. In this task, you will **verify these functions and send them to a GUI**.

1. Download and run the L2_kinematics.py program
    a. Output the xDot and thetaDot.
2. Make a new L_3 file to print [PDL , PDR] and [x_Dot , theta_Dot] to the terminal.
    a. Set up a NodeRed GUI for both phi dots, xDot, and thetaDot.
        i. Use the function tmpFile() from the L2_log.py program for each element you will plot.

   b. Export your NodeRed flow to submit. To learn how to export your NodeRed flow refer to the reference video named **How to export NodeRed flow**

3. Verify that your code and your flow work together

4. Submit code, and flow on eCampus on different files under the Lab5 submission box to be checked out from the lab. Code needs to be a python file and the flow a txt file.

Post-Lab Activity (Due Before Next Lab)

In addition to submitting your L3 code and flow to Ecampus, explain:

1. How the scuttle reads the encoder values and produces chassis linear and angular velocity.
2. How does the I2C protocol work with the encoders.

Rubric for python code and NodeRed flow:

- Explain the difference between log.UniqueFile() and log.tempFile()
- Proper software (python code and flow) are submitted on eCampus:
    - Code contains names, date, team number, and code purpose.
    - Code for flow is provided & imports properly.
    - Python software creates 4 log files (one for each variable)
    - Gauges indicate units in their titles (ie m/s, radian/s)
    - Python software runs without errors.

Lab Report Format:

- Introduction
- Procedure Summary/Objective
- Hardware/Software used
- Analysis/Discussion
- Conclusion

Lidar & Obstacle Detection

Overview

The objective of this lab is to measure the SCUTTLE surroundings using a Lidar sensor, discover the nearest obstacles, create a vector describing the angle and distance from the robot frame to the closest obstacle, and finally output the information of this vector to the NodeRed GUI. Lidar stands for Light Detection and Ranging, a remote sensing method for measuring distances with high precision. Lidar uses ultraviolet, visible or near infrared light to perform distance readings. In the field of robotics, Lidars are used for perception of the environment as well as object classification. The most common application for Lidar on mobile robotics application is for simultaneous localization and mapping (SLAM).

References

Video

- SCUTTLE videos
- SCUTTLE Robot - SICK Lidar Sensor Scans for Nearest Obstacle

Web Page

- SCUTTLE Lidar Lecture Slides
- SCUTTLE home page
- SCUTTLE GITHUB

Resources Required

- SCUTTLE
- BeagleBone Blue
- Battery Pack
- SICK Lidar Sensor
- Lidar Power Cable
- Micro USB Cable
- Lidar Bracket kit

Prelab

No Prelab Assigned

Lab

During this lab you will use the TiM561 Lidar from SICK, it is an opto-electronic laser scanner that scans the perimeter of its surrounding at a single plane with the aid of laser beams. Scanning is performed across a 270 ° sector. The maximum range is 10m, the TiM561 emits a laser beam using a laser diode to an object or person, this is reflected at its surface, then a reflection is detected by a photodiode. It uses the "time-of-flight measurement" to determine the distance to the object or person with a frequency of 15 Hz. The TiM561 uses 2.1 watts during operation, and an angular resolution of 0.33° at 1m.

Task 1: Securing and Connecting TiM561 Lidar Sensor

Before beginning using the Lidar with SCUTTLE, make sure you have all the resources required for the completion of this lab. Watch the following video to learn how the sensor its positioned on the frame and how to connect its signal and power cables.

1. Attach the Lidar to the Lidar Bracket using M3 screws.
2. Using M6x10mm screws and T-slot nuts, attach Lidar Bracket to SCUTTLE chassis.
3. Connect your Micro USB cable to the Micro USB port, located behind the black rubber plate (9) as seen in Figure 1.

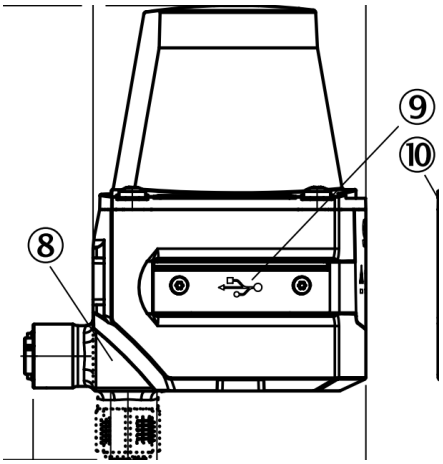
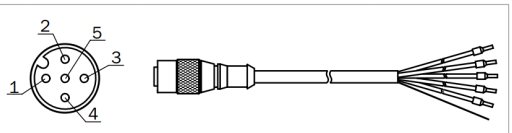


Figure 1-Lidar Side View

4. Modify your power splitter to accommodate for the Lidar power cable connection.

5. Based on the diagram from Figure 2, crimp only pin 1(Vin) and 3(GND) from Lidar power cable.



M12 - A / Socket *Illustration may differ*

Pin	Signal	Function	Wire color
1	24 V	Supply voltage IN (9 ... 28 V)	Brown
2	SYNC/Device Ready	Synchronization output	White
3	GND SYS	Ground	Blue or Yellow
4		Not assigned	
5		Not assigned	

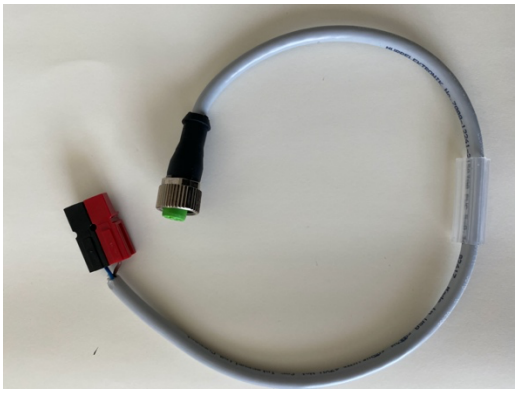


Figure 2-Power Cable Diagram

6. Connect the Lidar Power cable to the Power/Synchronization connection (10) as shown in Figure 3. Finally plug the Lidar to the SCUTTLE battery pack, you should hear the little DC motor inside the device spinning and a red light on the front panel of the Lidar, which after a couple of second should turn green to indicate the Lidar is ready. Figure 4 shows where should the Lidar be located on the frame of the SCUTTLE.

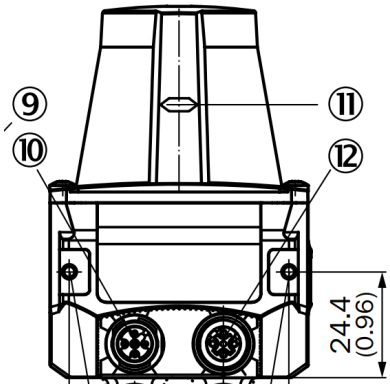


Figure 3-Lidar Back View

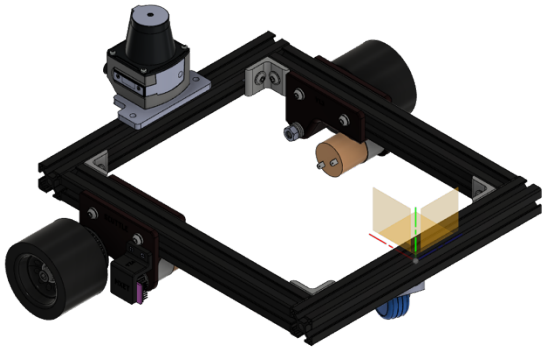


Figure 4-SCUTTLE

Task 2: Prepare Software

In order to run the our Tim561 SICK Lidar you will need to download and install some external libraries. For proper operation you will install on your BeagleBone the python USB library and the Lidar library required to operate the device using Python.

1. Install the pyusb library, which offers easy USB devices communication in Python. In the BeagleBone terminal please enter the following command: *"sudo pip3 install pyusb."*
2. Install the pysicktim, a library made to interact with the SICK TiM561 Lidar sensor over the USB connection. In the BeagleBone terminal please enter the following command: *"sudo pip3 install pysicktim."*

Task 3: Collect Measurements and Output Relevant Vector Data

Once all the required libraries are installed, you will use level 1, 2 and 3 software to run a task very common to robotics systems, find the nearest object to the robot. The level 1 script will grab raw data from the Lidar, assign an angle to each measurement, and create an array containing the distance and angle of each measurement. The level 2 script will use the measurement data from the level 1 program to find where is the closes object to the scuttle. Finally, create a level 3 script that uses the data from the level 2 software to log it for NodeRed to plot it.

1. Verify the Lidar is connected to USB
    a. Run the command *"lsusb"*
    b. Verify your BeagleBone recognizes the SICKLidar USB connection
2. Download to your *basics* folder the following files:
    a. L1_lidar.py
    b. L2_vector.py

3. Run L1_lidar.py
    a. Run the python script using the command *"sudo python3 L1_lidar.py"*
    b. Verify that 54 data points [distance(m), angle(deg)] are being printed in your terminal
4. Run L2_vector.py
    a. Run the python script using the command *"sudo python3 L2_vector.py"*
    b. Verify that the vector to the nearest object is being printed in your terminal
5. Create a new L3 script that plots the distance and angle of the closest object
    a. Indicate distance with a bar graph
    b. Indicate the angle alpha with a gauge
    c. Use appropriate units and tittles as shown in Reference Video

Post-Lab Activity (Due Before Next Lab)

Modify your L3 script to output dx and dy to your closest obstacle instead of d to your NodeRed GUI. Keep in mind that the L2_vector.py is returning the d and α. Refer to Figure 4. Submit your L3 script and NodeRed Flow to the ecampus submission boxes and include them at the end of your lab report. Make sure to write comments to your code.

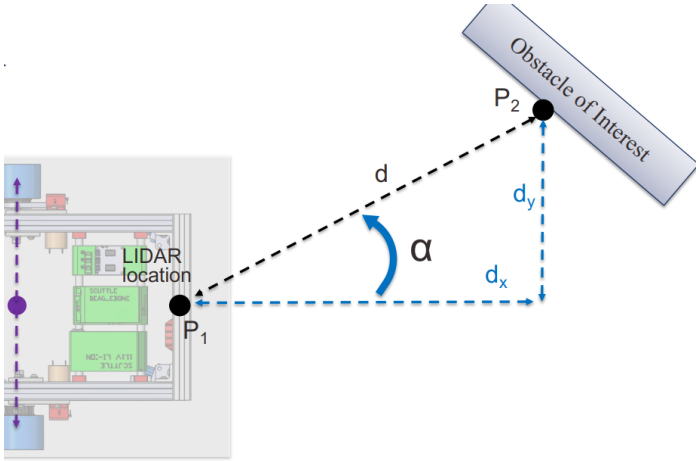


Figure 5-SCUTTLE Lidar

Make sure you include the following in your lab report:

1. What type of Lidar do we have available in the MXET lab?
2. How does the Lidar work? Include reference pictures and technical data to explain how does the Lidar work
3. What type of objects/surfaces affect your Lidar readings?

4. To what do we refer when we talk about Lidar resolution? What is the maximum number of points that the Lidar can return? Why would you use a small amount of points, instead of a large amount of points?
5. How is mapping done using the Lidar? What type of algorithm is used for mapping?
6. How can a Lidar be used in industry? Give 2 examples

Lab Report Format:

- Introduction
- Procedure Summary/Objective
- Hardware/Software used
- Analysis/Discussion
- Conclusion

Closed-Loop Control

Overview

During this lab you will get introduced to the topic of closed-loop controls, by now you should have a clear understanding of what **open-loop control** is, its advantages and disadvantages. Figure 1 shows the difference between the open and closed loop control-flow diagram(CFD). Closed-loop control takes feedback data from sensors as input and uses it to calculate the output, therefore making the output related to the input.

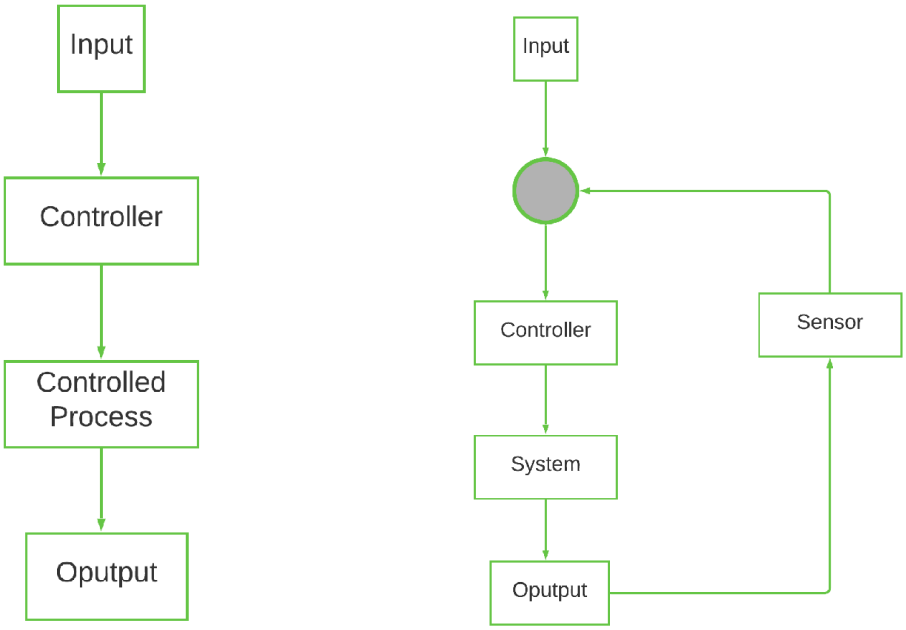


Figure 1 – Open-Loop (left) and Closed-loop (right)

In a closed-loop control system the actuating error signal, which is the difference between the target signal and the feedback signal is fed to the controller so as to reduce the error and bring the output of the system to a target value. These types of control are used in most automatic process control applications in industry today to regulate flow, temperature, pressure, level, and many other industrial process variables. We use closed loop controls every day, car cruise control uses closed loop to keep your car going at a constant speed, by adjusting the amount of throttle output according to the wheel speed input.

There are three basic feedback control actions: proportional(kp), integral(ki) and derivative(kd). Proportional control action is generated based on the current error, the integral control action is generated based on the past error, and the derivative control action is generated based on the anticipated future error. Integral of the error can be interpreted as the past information about it. Derivative of the error can be interpreted as a measure of future error. Based on these three control types, multiple other controllers can be created, such as proportional (P) controller, proportional-integral (PI) controller, proportional-derivative (PD) controller, and proportional-integral-derivative (PID) controller, to achieve desired control performance. At any given time, the control signal u(t) is defined as the following, note that e is the control error(target – actual). Figure 2 shows three types of controllers using different controller parameters.

$$u(t) = ke(t) + k_i \int_0^t e(\tau)d\tau + k_d \frac{de}{dt}$$

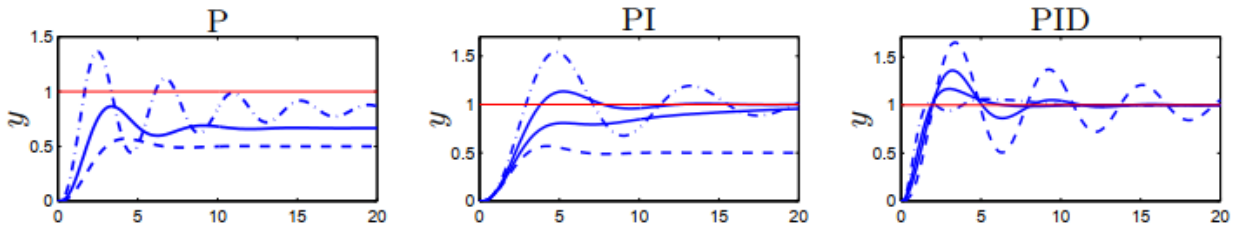


Figure 2- the controller parameters are k = 1 (dashed), 2 and 5 (dash-dotted) for the P controller, k = 1, ki = 0 (dashed), 0.2, 0.5 and 1 (dash-dotted) for the PI controller k = 2.5, ki = 1.5 and kd = 0 (dashed), 1, 2, 3 and 4 (dash-dotted) for the PID controller.

References

Video

- Understanding Proportional Control
- How PID affects performance
- PID Autonomous Vehicles
- Demo of a PID controller
- SCUTTLE videos

Web Page

- Society of Robots PID cool explanation
- SCUTTLE home page
- SCUTTLE GITHUB

Resources Required

- SCUTTLE
- BeagleBone Blue
- Battery Pack
- Motor Driver
- Encoders

Prelab

No Prelab Assigned

Lab

During this lab we will be creating a P, PI, and PID controller.

Task 1: Proportional Control

This section will introduce the use of the Proportional term in a PID control system. This term is defined as the *directly proportional to the error between your target values and your current values* in a closed feedback loop system. In this lab, our target wheel speed (rad/s) will be represented by *PdTargets*. The measured values, represented by *PdCurrents*,which are coming from the encoder.

1. Run L1_motors.py and L1_encoder.py to verify that everything is wired correctly, this will make our troubleshooting easier
2. Run in your terminal the following command to check the voltage level of your jack
    a. Sudo rc_battery_monitor
    b. If your Jack voltage is less than 10V, please ask for charged batteries to TA
    c. Remember to keep your battery pack charge!
3. Prepare Lab6Template.py (yes Lab6Template!) for running and logging in Closed Loop mode, within Cloud9.
    a. Set static PD Targets to **9.7 rad/s** target phis, this can be done in line 38
    b. 9.7 rad/s is the maximum speed when the wheels are loaded and readily achieved when free spinning.
4. Open L2_speed_control.py and set your kp value, this can be done in line 13.
    a. Change the kp control value to 0.06
    b. Make sure ki and kd are set to 0.0
5. Make sure to save your code changes and run Lab6Template.py in your terminal.
    a. Let the program run for about 10 seconds and terminate using ctrl^C
    b. If you do not see any motion on the wheels, increase the kp by increments of 0.01 until you see motion
    c. The following is a common error, and the expected output when running Lab6Template:

```
➜ basics git:(master) X sudo python3 Lab6Template.py
[sudo] password for debian:
/usr/local/lib/python3.7/dist-packages/rcpy-0.5.1-py3.7-linux-armv7l.egg/rcpy/__init__.py:116: UserWarning: > Robotics cape initialized
  warnings.warn('> Robotics cape initialized')
/usr/local/lib/python3.7/dist-packages/rcpy-0.5.1-py3.7-linux-armv7l.egg/rcpy/__init__.py:127: UserWarning: > Installing signal handlers
  warnings.warn('> Installing signal handlers')
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
ALSA lib pcm_dmix.c:1108:(snd_pcm_dmix_open) unable to open slave
```

6. Go back to your file system(folders), open your .csv file and make sure you have at least 60 data samples

a. Download the .csv file. If you are using Cloud9, find the file named 'excel_data.csv' in the basics folder, right click, and download it. (You can download the file in the same manner if you are using MobaXterm Windows terminal.

b. Open a new Excel Workbook (2016 and prior)

i. Click on Data □ FromText

ii. Find the .csv file you downloaded and import it

iii. Once the Text Import Menu pops up choose:

1. Delimited □ Next
2. Make sure only the Comma delimiter selected □ Next
3. General □ Finish

c. Graph both the Current Phi's and the Target Phi's. The result should resemble something close to the graph shown in Figure 3

i. Make sure to include appropriate Title, Axis, and Legend labels

ii. *Make sure to include your graph in Lab report, and explain what is happening*

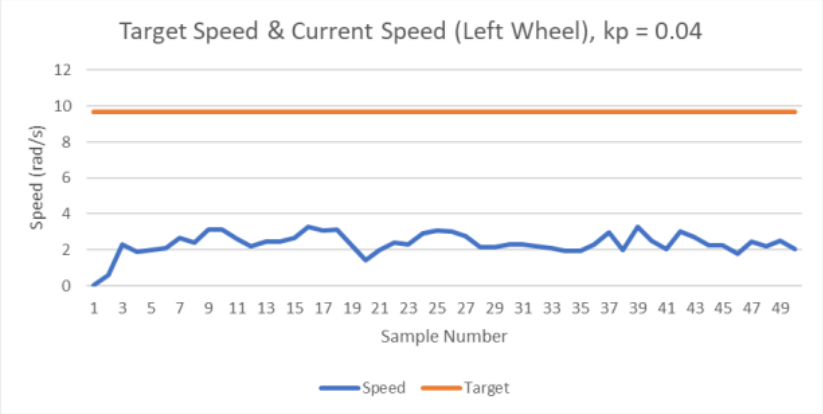


Figure 3

7. Go back and redo steps 3 through 5, but this time change the kp value to 0.09 and 0.5. Do you see any difference?

a. *include graphs in report and comment on the observations*

Task 2: Proportional + Integral Control

Now we will analyze the effects of adding the integral term to our SCUTTLE Controller. Watch the following video for reference video Demo of a PID controller

1. Run a I control(integral only)

a. Go back to your text editor and open L2_speed_control.py

b. Review the driveClosedLoop function and see how the ki term will be utilized.

   c. Set the ki value to 0.01, kd and kp set to 0.0
   d. Run the Lab6Template.py to call the control system into action.
   e. Play with multiple Ki values such as 0.02 and 0.03
2. Repeat the graphing steps from task 1. Keep all your data within one Excel sheet, just make a new tab for every test.
   a. *Make sure you add to lab report*
3. Once you observe the output of the wheel, go back and change kp to 0.04 and ki = 0.04. Make sure to graph these results. Record observations by graphing.
   a. *Make sure you add to lab report*
   b. *explain what is the difference between a P and a PI controller*

Task 3: Derivative

In a PID control system, the derivative term is the control signal portion acting on the derivative of the error. This Derivative error is defined as the change in error from one sample to the next, while the error is defined as the target minus the current value.

Since the error can change drastically from one sample to the next, implementing a d term can cause unstable performance. In this section you will make a hand calculation of a derivative control signal instead of adding a d-term and running it.

1. In Excel, make a copy of your data output from the trial where kp and ki are 0.04
   a. Just as in Figure 4, locate two adjacent samples with a large delta in speed
   b. Perform a hand calculation of the derivative term and find the derivative part of the control effort
      i. Don't use any samples in the first 10 samples from the startup.
      ii. Assume kd = 0.04.
      iii. Find the error for each term.
      iv. Find the difference in error.
      v. Instead of measuring the dt, just use dt = 1. We will use units of samples instead of units of time.
      vi. Multiply your de/dt by kd, and this is the control effort u_d.
      vii. For the second sample in your two-sample sequence, also calculate the
      control effort u_p, which designates with proportional control.
   c. Compare your results of kp and kd for the two sample
   d. Include your calculations in the lab report

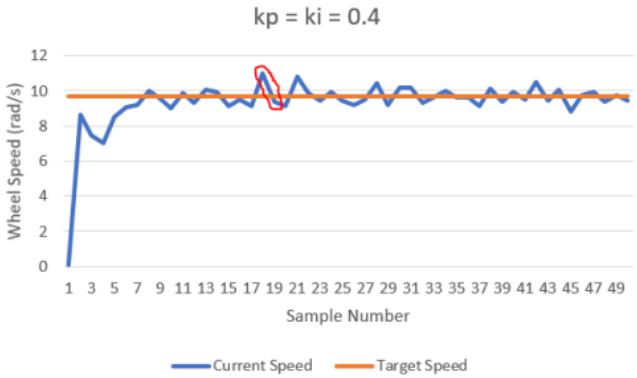


Figure 4

INDIVIDUAL Lab Report Format:

MAKE SURE TO INCLUDE YOUR GRAPHS! MAKE SURE TO ADD ALL THE CORRESPONDING LABELS AND A BRIEF EXPLANATION OF WHAT IS GOING ON

- Introduction
- Procedure Summary/Objective
- Hardware/Software used
- Analysis/Discussion
- Conclusion

Computer Vision & HSV Filter

Overview

During this lab you will get introduced to the topic of computer vision and image processing, during the lab you will capture images with an USB camera, create a “mask” based on a color range filter, and convert the masked area into a visual target having x,y coordinates and a radius. Finally generate driving speed targets using the visual target information.

You will learn the meaning of color spaces, specifically RGB and HSV color space. We will learn how to perform image processing using the OpenCV libraries, which allow us to have control over every single frame coming from the USB camera. Image processing enables computer vision, a field of computer science that focuses on replicating parts of the complexity of the human vision system and enabling computers to identify and process objects in images and videos in the same way that humans do. Until recently, computer vision only worked in limited capacity.

Thanks to advances in artificial intelligence and innovations in deep learning and neural networks, the field has been able to take great leaps in recent years and has been able to surpass humans in some tasks related to detecting and labeling objects.

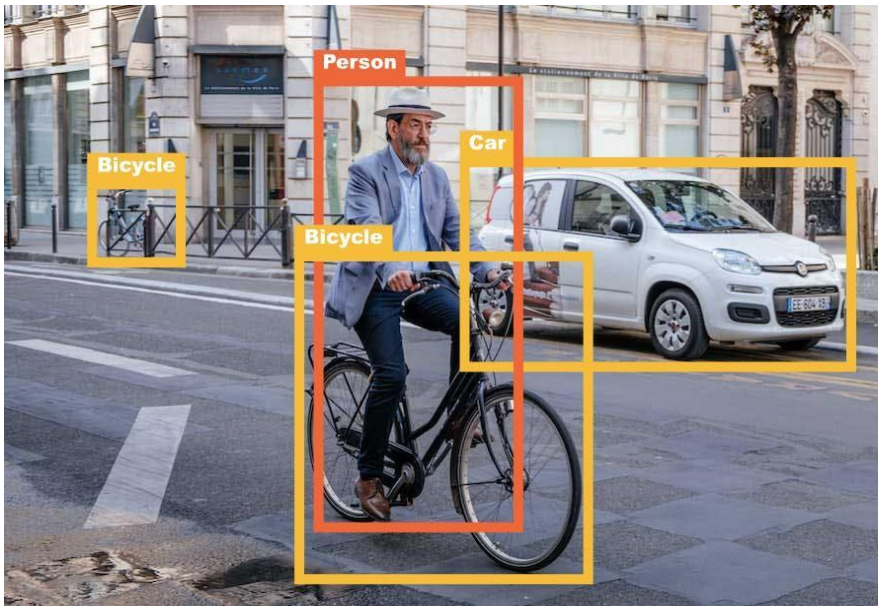


Figure 1 – Object Detection

References

LinkedIn Recommended Courses

- OpenCv for Python Developers

Video

- Computer Vision Demo
- View live video on web browser using MJPG streamer
- Computer Vision for Autonomous Vehicles
- Switch LED through Dashboard
- SCUTTLE videos

Web Page

- RGB
- HSV
- OpenCV
- SCUTTLE home page
- SCUTTLE GITHUB

Resources Required

- SCUTTLE
- BeagleBone Blue
- Battery Pack
- Motor Driver
- USB-Camera

Prelab

No Prelab Assigned

Lab

Task 1: Run MJPG Streamer and NodeRED Flow

To get a view of what the camera output looks like, we will use the MJPG-Streamer. MJPG-Streamer is a command line application that copies JPEG frames from one or more input plugins to multiple output plugins. It can be used to stream JPEG files over an IP-based network from a webcam to various types of viewers such as Chrome, Firefox, Cambozola, VLC, mplayer, and other software capable of receiving MJPG streams. **It was originally written for embedded devices with limited resources in terms of RAM and CPU.**

1. Verify that your camera is plugged in and is recognized by the BeagleBone Blue.
    a. Run the command *"lsusb"* in the terminal and verify a device containing the name "Microsoft Corp." exists in the output
2. Run the MJPG_Streamer Filter
    a. Navigate to your basics folder and create a folder named *"computer_vision"* using the command *"mkdir computer_vision"*
    b. Grab the start_mjpg_streamer.sh and L3_image_filter.py files from the Github and save them into your computer_vision folder.
    c. Run the command "*bash start_mjpg_streamer.sh L3_image_filter.py*".
3. In NodeRed, set up the pre-defined flow for machine vision calibration
    a. Navigate to your NodeRED Flows web page at 192.168.8.1:1880
    b. Go to NodeRed Flow Link in the GitHub addon section and copy its contents.
        i. Open your NodeRED page and click on the menu button on the top right of the page
        ii. Click on "Import"
        iii. Click on "Clipboard"
        iv. Paste the NodeRED flow into the text box and click "Import". Select the "new flow" button to avoid overwriting your existing one
    c. Deploy the NodeRED Flow
4. Verify that your dashboard shows a video stream with 3 images, and 6 sliders which you can control as shown in Figure 2

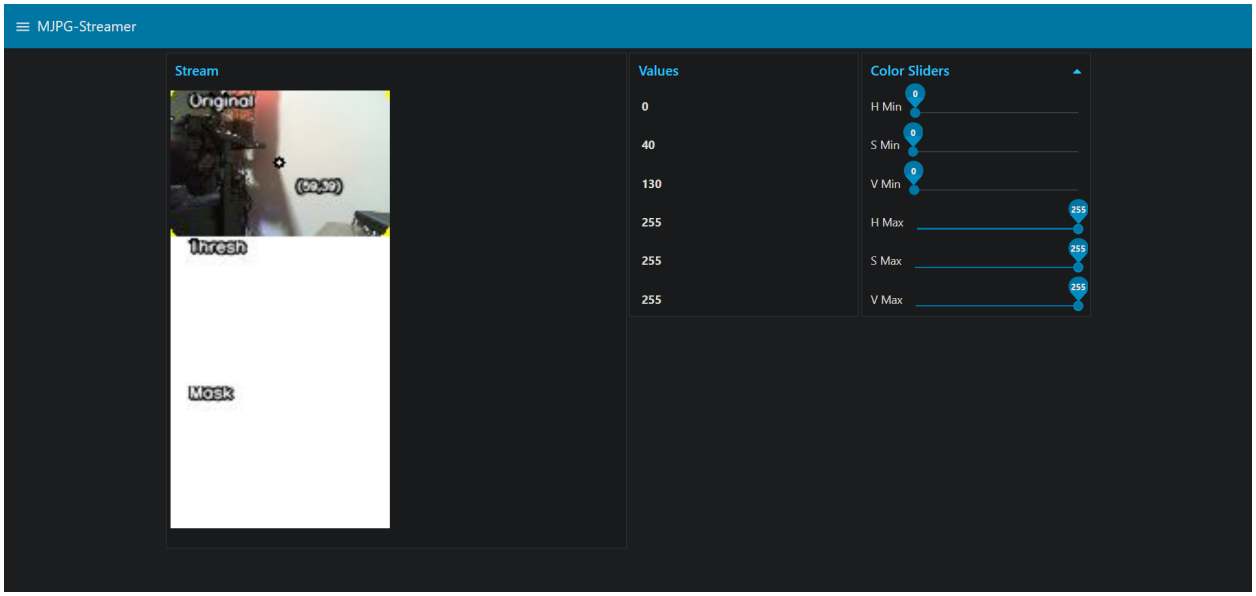


Figure 2 - Dashboard

Task 2: Tuning your HSV filter values

First, we need to find the HSV color range that we want to capture in the image stream. We need to find an HSV minimum value and an HSV maximum value**. Any pixels between our minimum and maximum HSV values will be considered a pixel that is part of the object we are trying to capture AND WILL BE SHOWN IN *WHITE* IN THE MASK AND THRESHOLD OUTPUT FEED.**

1. Calibrate minimum sliders
    a. Move to the right (increasing) as far as possible without blacking out your target
2. Calibrate maximum sliders
    a. Slide the maximum sliders to the left while making sure the object you want to track is always white
3. Once the output looks like the one in Figure 3, **record the HSV range you have selected and take screenshots for your lab report**

Figure 3 – Good Mask Result

Task 3: Chase the Basketball

Finally, we will make the SCUTTLE follow the basketball using only the camera. Note how the SCUTTLE determines its distance from the ball using only the camera feedback, in addition to determining where the ball is in the frame. During this task we will use the previously recorded min and max HSV values to help SCUTTLE see only the ball.

1. Download the following SCRIPT to your *"computer_vision"* folder.
2. Change the min and max values located in line 25-31 to your values obtained during Task 1. Refer to Figure 4
3. Run the script using "*sudo python3 color_tracking_v1.py"*

```
22
23  #    Color Range, described in HSV
24
25  v1_min = 0     # Minimum H value
26  v2_min = 0     # Minimum S value
27  v3_min = 0    # Minimum V value
28
29  v1_max = 255     # Maximum H value
30  v2_max = 255     # Maximum S value
31  v3_max = 255    # Maximum V value
```

Figure 4

Post-Lab Activity (Due Before Next Lab)

- Grab one more object with different color and perform task 1 to find their min and max values, include screenshots on your lab report

In addition to the mentioned information during lab instructions, make sure you include the following in your lab report:

1. How can computer vision be used in robotics? Give at least 3 examples, with at least one example of industrial application
2. Explain what color space is and why do we use HSV instead of RGB

Lab Report Format:

- Introduction
- Procedure Summary/Objective
- Hardware/Software used
- Analysis/Discussion
- Conclusion