Labs & Project Overview

There are roughly nine labs, followed by roughly 4 weeks for a semester project (depending on the semester. Each week has one lab exercise and one learning objective that supports a successful robotics project for a mobile robot.

Lab Topics & Learning

The labs include hardware, instruction, and hands-on tasks. Each team performs a lab exercise and submits results. Results may be a software, a set of measurements, or a working hardware result. For example, the students learn to power & control the motor driver in-lab, demonstrate forward/reverse control, and the instructor verifies the result for lab credit.

Week	Lab Topic	Learning Topic
1	Raspberry Pi, Linux	How to program in Linux
2	Python, Onboard Sensors, NodeRed	How to collect & display sensor data
3	Motor drivers, inverse kinematics	How to calculate movement & command motors
4	Encoders & Kinematics	How to plan movement & measure wheels
5	Compass & Calibration	How to calibrate a sensor
6	PID control	How to perform closed-loop control
7	Machine vision	How to use machine vision
8	Lidar, Obstacle Detection	How to detect obstacles
9	actuator integration	How to add actuators to my system

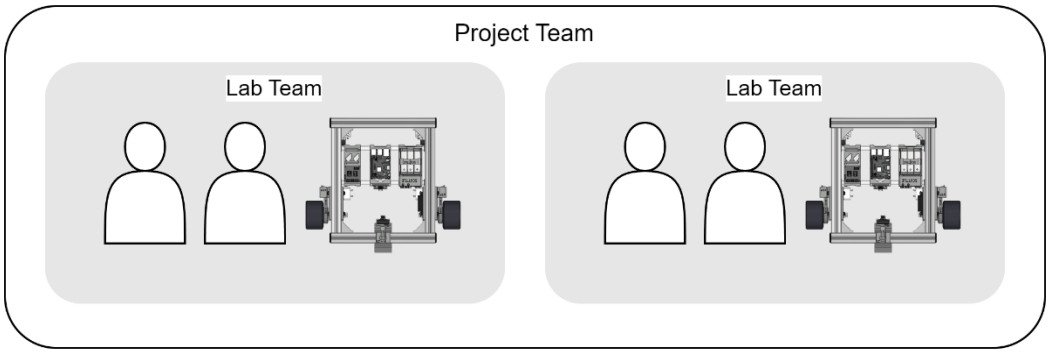
Project Actions & Learning

After lab exercises, weeks are dedicated to the teams project. The project selection is open-ended but the teams must take action to keep progress over four weeks.

Week	Project Action	Learning Topic
1	Raspberry Pi, Linux	Plan a robot function with sensors, actuators
2	Python, Onboard Sensors, NodeRed	Convert plans into computations
3	Motor drivers, inverse kinematics	Test & adjust a system for functionality
4	Encoders & Kinematics	Demonstrate outcomes & barriers.

Teams:

There are generally two students per robot, and two robots per team. For the labs, there are two teams combined together. Then, one robot is available for testing while the other robot is available for demonstration. Or, the teams may demonstrate both robots working together.



Resources

This is a resource-intensive lab. The following list describes resources and where the come from

Robot kits

kits include the SCUTTLE components and a few extra parts such as power supply and USB cables. They are checked out to students by the instructor and checked back in at the end of the course. Kit inventory is provided as a separate document. Students may bring kits home or to other lab spaces as instructed.

Laptops

Students use their personal computers to learn and perform lab exercises. The Laptop displays the robot software, by command-line-interface with the robot CPU. The students are directed to install some simple programs such as VSCode or PuTTy for performing lab exercises. Other academic software is also available for CAD modeling when it comes to building a project, or "slicer" software for designing a 3D print.

Technical Diagrams

Diagrams cover proper wiring configuration, coordinate frames for robot navigation, assembly of mechanical parts, etc. They are distributed in PDF documents found online, such as "Kinematics Guide" stored at scuttlerobot.org.

Datasheets

Datasheets are not needed for the labs directly, but they are necessary for adjusting sensors, changing the wiring of your robot, or discovering operating limits. For example, does my power adapter have sufficient power to add 3 servos? Datasheets are provided by links in the lab content or other technical guides.

Software

Template software is used for some labs. Usually, it is a python program. For example, one program may be downloaded by the student, added to the SCUTTLE CPU (raspberry pi or beaglebone) and perform a task. One such program commands the robot to drive in a straight line for a specified time. Software is made available to download from the web, or is included in the pre-existing image for the SCUTTLE CPU. The image is stored online, and students learn to download and install a fresh image on their CPU in case the software becomes broken.

Videos

Some learning material is provided in video format during the lab instruction. The instructor may use a pre-recorded video to explain a concept, such as "lithium battery safety." These videos are reused each semester and are noted in the lab content or simply shown during labs. Other videos may be offered for technical efforts such as wiring motors, or soldering a PCB. Videos are typically stored on YouTube and are openly accessible.

Tools

SCUTTLE is designed to require very few tools. Students may make modifications, repairs, or build additional circuitry. Hand tools are available in the lab (not to be checked out) such as soldering iron, crimpers, and screwdrivers. Your instructional lab offers these tools for use any times during your lab hours. Further work may be performed inside the lab during business hours or using the tools in your common labs. Students are encouraged to utilize engineering labs for 3D Printing, Electronics, and Fabrication as needed. There is a "tools guide" at scuttlerobot.org indicating necessary tools to support the mobile robotics lab.

Supplies

The instructional lab has consumable supplies such as solder, connectors, wires, and fasteners that are found on the robot. Students may use these supplies with permission, and are encouraged to build their projects by making designs with these components. This will save time in your design, improve repeatability, and increase the instructor's ability to help troubleshoot.

Project Resources

For project ideas, there is an Applications Guide on the scuttle website. There are also example projects made by previous students. The example projects are discoverable in the scuttle website videos, and sometimes further documented on Hackster.io, and with 3D printable models often shared on grabCAD.com. Students are encouraged to redesign past projects for improved results.

Lab 1: Linux and Beaglebone (v1.3)

Overview

TOPIC: Getting started with BeagleBone, Shell, and Linux environment.

This lab is an introduction to several of the tools used in the course. Keep these instructions for reference in future labs.

Resources required:

Team Parts	References
• Beaglebone Blue (to be called "Blue") • 3D printed bracket for blue, & screws • 12v 2.0 A DC power supply with barrel connector • Micro USB cable • Individual laptops • 16 GB microSD card (check out)	VIDEO: Flash a Debian Image VIDEO: Linux Root, Home, directories VIDEO: Connect by USB & find SSID PLAYLIST: all SCUTTLE videos Web Page: SCUTTLE home page

Prelab: Command Line on Your Own

This prelab introduces you to BASH scripting, and sets up your PC with necessary software for the course. We can test our robot faster and with fewer steps on the beaglebone if we make commands to the Debian operating system directly through Linux Command Line. We use bash scripting when we make these commands so we need a basic familiarity with commands and navigating the file system with a text-based, lightweight terminal program like Moba Xterm. Codecademy is a free site with interactive tutorials that we will use.

STEPS

1. Go to codecademy.com and create an account. Save your credentials.
    a. Navigate to the catalog and find the Bash/Shell subject
    b. Perform the course "Learn the Command Line"
        i. Complete each tutorial available with the free version. This includes Navigating the File System, Viewing and Changing the File System, Redirecting Input and Output, Configuring the Environment, Bash Scripting.
    c. Save your progress (or make sure it autosaves) to show the lab instructor at the start of the coming lab.
2. Download and install Etcher software from the web, onto your laptop. This will prepare you to flash your own debian image to your micro SD card.
3. (optional & highly recommended) Download and install Moba Xterm for fast and reliable terminal sessions.
4. (optional & highly recommended) Download and install notepad++ for offline editing of software files.

Task 1: Install the latest 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 Blue can run Debian directly from it's onboard hard drive but it's easiest to install the OS to an SD card by flashing the card and then booting the BBB from the micro SD card.

STEPS:

1. Your beaglebone must be secured in the 3D printed bracket before powering on. Use the black M2 plastic screws to secure your computer into the bracket.
2. Access the website "Beagleboard.org gettting started" which includes steps for flashing a new image, and browsing to the blue in the web browser
    a. Grab the latest Stretch IoT (without graphical desktop). Choose "recommended," and not the "Flasher."
    b. Flash your SD card with your PC (requires 10 minutes), and then insert into your Blue (while powered off).
    c. Boot from the SD card using this process: Hold the SD button on your Blue, connect your beagle micro-usb to your PC USB port to power on, and release the SD button. The device will boot in under 1 minute.
        i. Follow the steps in this video to open a terminal session with your blue and retrieve your beagle's wifi SSID. (youtube video)
            1. Host IP: 192.168.7.2, port 22 (using usb)
            2. Username: debian
            3. Password: temppwd
            4. Once you have been granted access, type "help" to see a list of available commands. To learn about these commands, type help then space then the command, and hit enter, such as "help times".
    d. Power off the blue by pressing POW on the board, wait for the power light to turn off, and then disconnect the usb cable.

Task 2: Flash BeagleBone Blue (BBB) and Access Cloud9, Wifi

You have successfully flashed the latest Debian image onto your blue. Now let's get the blue connected to your PC, and also setup wifi access for the board. These steps are supported by a youtube video here.

STEPS:

1. First use your laptop to retrieve a wifi setup program from the web.
    a. Navigate to the SCUTTLE github, then go to software/python/basics and find setup_wpa_enterprise.py
    b. Click the "raw" button to view just the program, and ctrl-A, then Ctrl-C to copy the program.
    c. Paste this into a temporary location such as notepad++ or a new text file, and DO NOT alter the indentation & spacing of the text.
2. Also retrieve a two programs for installing important packages on your blue, located at software/scripts
    a. Locate the self installer part 1 (link here)
    b. Locate the self installer part 2 (link here)

   c. Copy the text in the same manner as step 1 and store it on your PC in two files.
3. Boot up your blue again **but this time power it with a 12.0v power supply**, and after 1.5 minutes look for an access point with your SSID such as "BeagleBone-DC5A" on your PC.
   a. Connect your PC to this hot spot using the default password: BeagleBone
   b. You will no longer have internet access on your PC unless you have two wifi adapters
4. Navigate to Cloud9 IDE hosted by your Blue
   a. Enter in your web browser: 192.168.8.1:3000 to access Cloud9
   b. Create a new file in the home directory by right clicking in the workspace tree panel.
      i. In cloud9 settings, you can "show home folder" in the workspace tree.
      ii. The home directory is indicated by a folder with the "~" symbol.
      iii. Name the file setup_wpa_enterprise.wpa, paste your software into it and save it.
      iv. In your terminal, navigate to home (cd ~) then run the file by typing "sudo python3 setup_wpa_enterprise.wpa" and hit enter. Follow the steps in the program to connect to wifi.
         1. Password for sudo permission: "temppwd"
      v. Verify that you have successfully connected by running "ping google.com."
5. Install the necessary Linux packages for the course.
   a. The shell commands for installing packages are in task 2 step 2.
   b. Create a new file in the home directory of your blue and paste the text from step 2, and save the file.
   c. Run the shell scripts
      i. type "sudo bash self_installer_part1.sh"
      ii. press enter. This script ends in a reboot of your device.
      iii. After reboot, initiate a new terminal session
      iv. Type "sudo bash self_installer_part2.sh". Expect this to take 30 minutes!

Lab Checkoff

Each open circle below is worth equal points for the lab total score. (ie, with 12 circles and a 10-point lab, each circle is worth 10/12 or 0.83 points)

- Prelab:
  - Nav the file system
  - View and change the file system
  - Redirecting i/o
  - Configuring the environment
  - Bash scripting
  - Etcher is installed
- Task1
  - Beaglebone is secured with 2 screws (using latest 3d print)
  - Your SSID is recorded in a note on your computer hard drive
  - Image is downloaded to a safe location on your computer
  - Ping google.com to show connectivity
- Task2
  - Show setup_wpa_enterprise.py is created in the home directory
  - Show self_installer.sh (both parts) is created in home directory

- L1_bmp.py runs (indicating successful package installation).

Lab 2: Onboard Sensors, Graphical data display

This lab walks you through accessing your onboard sensors such as accelerometer, analog-to-digital converter, and temperature sensor. You will write a program that sends data to a file on the beaglebone, and the file will be read by a prebuilt graphical user interface called NodeRed. Before the lab you'll need familiarity with Python. During this lab, consider what sensors will be useful for your robotics project and familarize yourself with them.

Prelab: Python on Your Own

STEPS:

1. Log onto Linkedin Learning with your university email address. Access the course titled "Learning Python." Tip: you can increase the speed of the playback.
    a. Watch the 6 videos in chapter 2 (39 minutes total). These include:
        i. Building Hello World
        ii. Variables & Expressions
        iii. Python Functions
        iv. Conditional Structures
        v. Loops
        vi. Classes
    b. Note that the IDE in this course (VScode IDE) shown by the instructor has similar operation to our IDE (cloud9).
    c. Note that we will use Python3, which has slightly different syntax from Python2.
    d. At the start of the lab, you'll show the videos are viewed in your Linkedin Learning account.
2. Install Notepad++ on your laptop. (REQUIRED).

Hardware required:

Team Parts	Videos
Beaglebone blue (1 per team) 12v dc power supply with barrell connector (from scuttle kit) Individual laptops microSD cards (1 per team)	VIDEO: NodeRed on Blue

Task 1: Access sensor data

Level 1 programs are intended to access data from a sensor or send commands to an actuator. In this tasks you'll run level1 files in standalone configuration and become familiar with some available sensors on the Blue. Note that the sample loop at the end of each L1 file does not print out every parameter available from that sensor. *Read the comments in these programs – you will need to understand them and manipulate them in your project!*

STEPS:

1. Boot your beaglebone by barrell plug power and connect to your wifi.
    a. Create a folder in your home directory (cd ~) named "basics"

b. Copy these files from the github and save them in your basics folder
    i. L1_mpu.py
    ii. L1_adc.py
    iii. L1_rssi.py
    iv. L1_bmp.py (requires library installation – see comments in the program)
    v. L2_log.py and L3_telemetry.py
c. Prepare the first program for running on your Blue (L1_mpu.py)
    i. Open the program for editing by double-clicking it in cloud9.
    ii. Uncomment the section below the line that reads: "UNCOMMENT THE SECTION BELOW TO RUN AS A STANDALONE PROGRAM". Toggle comments by highlighting the text and typing "ctrl" + "/"
d. Open a terminal on your Blue and run the mpu program
    i. Change directories to "basics," containing the program
    ii. Run by using command "python3 L1_mpu.py"
    iii. Observe the outputs, the magnitudes of values, the units of the data.
e. Run the remaining L1 programs in the same manner.
    i. Answer the questions for the lab checkoff
    ii. Return the programs to the original condition, with the while-loop commented.

Task 2: Output data to the NodeRed GUI

This task uses 3 programming levels. 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 of the action. Once the L3_telemetry.py is running, you can configure NodeRed to access these variables in realtime. 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 realtime data in your web browser.

STEPS:

1. Boot your beaglebone and connect
    a. Run the L3_telemetry.py file from inside your basics folder.
        i. Use command: sudo python3 L3_telemetry.py
        ii. Verify that the program runs, and some values are being printed in your command line interface
    b. Access NodeRed web page, which is published by your Blue by default
        i. In browser, enter 192.168.8.1:1880 (this indicates port 1880)
    c. Create a "flow" which will access values from a file on your beagle:
        i. Refer to the video: NodeRed on Blue for building your own flow.
        ii. Create a chart for x and y axes of accelerometer.
            1. Include chart titles that name the axis and units
            2. Scale the charts y axis, from -10 to 10
        iii. 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.

Lab Checkoff:

To receive points for this lab, the lab instructor will checkoff items corresponding to the tasks. Items will be checked at the start of the next lab. If there is a report to be written, report format will be distributed on eCampus.

PRELAB:

- Notepad++ is installed on your computer

LAB: FORMAT THE FOLLOWING INFORMATION INTO A SINGLE PAGE WITH NAMES, DATE, LAB NUMBER,

- Checkoff: You have the python files in your /basics folder:
  - L1_mpu.py
  - L1_adc.py
  - L1_rssi.py
  - L1_bmp.py
  - L2_log.py
  - L3_Telemetry.py
- What is output by mpu.py?
  - How many values?
  - What are the units?
- What is output by adc.py?
  - How many values?
  - What are the units?
- What is output by rssi.py?
  - What are the units?
  - What two devices are in communication over wlan0?
- What is output by bmp.py?
  - What are the units?
- Show your plot of x and y accelerometers in nodered
  - Chart titles include proper axis
  - Chart titles include proper units

Lab 3: Motor Drivers, Inverse Kinematics V2.1

This lab walks you through operating your motor drivers. This consists of powering the motor driver, verifying signal wires from the Blue to the motor driver (four connections), and understanding the PWM output from the motor drivers. You will also see how the command is converted from robot speed to wheel speed to pulse width command to voltage to an actual wheel movement.

Prelab: Understand wires for motor drivers

STEPS: Be prepared to answer the following for lab:

1. Watch the video explaining duPont Connectors
    a. What are the large wings designed to grasp?
    b. What are the small wings designed to grasp?
    c. Is solder added before or after the crimp, or not at all?
2. How much max current is typically carried by our wires? (search web if needed)
    a. Signal wires: assuming 28 american wire gauge (awg).
    b. Power wires: assuming 18 awg.

Resources required:

Team Parts	Videos
Beaglebone blue with SD flashed (1 per team) 12v dc power supply with barrell connector (from scuttle kit) Connector: 12v power to motor driver Connector: 12v power to beagle Motor driver circuit board Motor/pulley/wheel assemblies Motor signal wire harness	VIDEO: Anatomy of DuPont Connector VIDEO: Root, Home, & Workspace VIDEO: Build a Power Splitter MODELS: Latest CAD files for 3D printing • Motor driver bracket • Wheel assy uni-bracket

Task 1: Run the motor driver

In this task you'll run and understand the level 1 program that operates your motors. You will connect all of the wiring to operate the motors, and verify your motors are connected in the proper configuration.

STEPS:

1. Build your **motor signal wire harness** which connects Blue to the motor driver.
    a. Use the video on duPont connectors for proper crimping.
    b. Follow the latest wiring diagram, linked from the scuttle web page.
        i. M1 and M2 on the Blue's silkscreen will connect to input pairs (1,2) and (3,4) respectively.
        ii. Ground on the motor driver signals has no connection but *the motor driver power source must share a ground with the beagle power source*.
        iii. The wire colors, pin sequence, Left/right assignment, and positive/negative assignments must match the instructions precisely.
2. Prep the L1_motors.py file.
    a. Boot your beaglebone by barrell plug power and connect to your wifi.

   b. Add the L1_motors.py and L2_inverse_kinematics.py files to your basics folder from the web.
      i. You should already have a basics folder in your home directory.
   c. Tip: show /home/debian in Cloud9 workspace tree under settings.
   d. Uncomment the bottom section of L1 code to execute the while-loop.
3. Run the L1_motors.py program with wheels lifted up.
   a. Prop the rear of the robot so it can't drive away.
   b. Verify that the wheels move forward for a period, then reverse.
      i. Both wheels should move in the same direction.

Task 2: Perform kinematics, navigation

All driving commands for SCUTTLE will come in the form of $\dot{x}$ and $\dot{\theta}$, 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 commands are converted by the L2 program to generate individual wheel speed commands. The L1 will be driving the motors and the L2 will be converting commands to the form necessary for L1 to execute them.

STEPS:

1. Run the L2_inverse_kinematics.py file.
   a. Use command: python3 L2_inverse_kinematics.py
   b. The corresponding L1 file must be together in your "basics" directory.
   c. Test various input pairs of xd and td, and observe the output wheelspeeds printed to the screen, to prepare for the next steps.
2. *Simulate* a navigation program for your robot by driving your wheels unloaded.
   a. Achieve a path on the ground which is geometrically similar to the drawing in Figure 1. Please note!: there is no feedback control in this lab so actual driving is not recommended.
      i. Note that SCUTTLE's max speed is approximately 0.4m/s forward and 0.99 radian/s turning. Constrain your speeds to within this max.
      ii. Tip: the arc length of a semicircle is pi*r, and the forward velocity integrated over the time of execution gives your arc length in a driving motion.
      iii. Your robot should end in the same position and heading as starting.
         1. The star in the figure marks the start.
      iv. You may choose d1 and d2 lengths.
   b. Write your program into the loop and save your python file to submit on eCampus as Lab_3.0_TeamX.py
      i. Start with the lab3 template under scuttle/software/python/labs

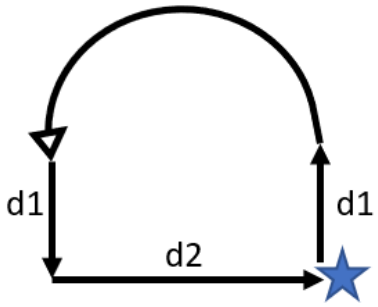


Figure 1. Driving Pattern

Table 1. Your Navigation Sequence

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

Lab Checkoff:

Each white circle is worth 1/16 of the lab, totaling 10 points for the lab. Task 2 and the submitted program will be graded outside of the lab. Task1 will be graded at the start of the following lab.

TASK1

- Power splitter is made properly
  - Red/Black housings are secured together. red is on the right when facing text
  - Connectors are fully seated (tug to test)
- Motor Driver signal harness made properly
  - M1 is connected to pins 1,2 on motor driver
  - M2 is connected to pins 3,4 on motor driver
  - Dupont crimps are fully seated (see an off-the shelf wire for reference. Metal is not showing near the back of the housing where wire enters)
- Wheels pass test with L1_motors.py running (send 0.6 pwm if needed)
  - Both wheels move forward for 4 seconds
  - Both wheels move reverse for 2 seconds

TASK 2

- Proposed theta-dot and x-dot will achieve the path
  - d1 and d1 are equal
  - motion 2 (arc) turns 180 positive (or pi or 3.14)
  - d2 and diameter of arc are equal
  - motion 4 and 6 turns 90 positive (or pi/2, or 1.57 rad)

SUBMITTED PROGRAM

- Program is formatted
  - team number, names, date are included
  - code purpose is included
  - d1 and d2 distances are calculated in meters.
- Driving pattern is updated
  - Speeds are added in 6 sections
  - Sleep timing is added in 6 sections

Lab 4: Encoders, Kinematics V2

This lab walks you through operating your encoders, building the i2c communication cables, and utilizing the forward kinematics functions. The encoders provide information about wheel speeds and the navigation strategies make decisions about chassis speeds. So, a Level-2 program is utilized to convert the wheelspeed data into chassis speed data.

Task 0: Understand i2C bus

STEPS: You should understand these questions and their answers. Responses will not be collected.

1. Watch the video explaining encoder Connectors
    a. What are the A1 and A2 pin condition for Left Encoder?
    b. What is the left encoder address?
    c. What are the A1 and A2 pin condition for Right Encoder?
    d. What is the right encoder i2c address?
2. Describe the encoder
    a. What is the resolution?
    b. What is the maximum value returned?
    c. In 1 full rotation of encoder, how many full rotations of the wheel are there?

Resources required:

Team Parts	Videos
Beaglebone blue with SD flashed (1 per team) Assembled SCUTTLE with Encoder cables (left and right) I2C cable (JST-ZH on Blue to dupont on bus board) I2C bus board soldered and covered on bottom with insulating material I2C board bracket	Video: Battery Safety & Handling Video: Build your SCUTTLE i2c board Video: Mounting Encoders Video: Exporting Node Red Flow MODELS: Latest CAD files for 3D printing • Motor driver bracket • Wheel assy uni-bracket

Task 1: Run the encoders

In this task you'll run and understand the level 1 program that operates your encoders. You will connect all of the wiring to operate the encoders as well as the wire that connects i2c to the beaglebone jst connector.

STEPS:

1. Build your **encoder wire harness (left and right)** which connects Blue to the motor driver.
2. Build your **i2C bus board**.
    a. For how to solder successfully, follow the video (i2c video)
    b. For how to lay out the board, follow the diagram (wiring diagram)
3. Build your connector from **Beagle (JST-SH 4-pin) to I2C bus**.
    a. Use the 4-pin JST-SH connector and crimp on the 4-pin dupont connectors
4. Download the L1_encoder.py program and run it from your basics folder.
    a. Uncomment the loop as required

   b. Verify that the encoder values are representative of the wheel movements
      i. Move the wheels by hand while the chassis is lifted
      ii. Turning a wheel forward should produce a climbing value for the corresponding wheel
      iii. Turning a wheel backwards produces a falling value for the corresponding wheel
      iv. A stopped wheel produces a non-changing value

Task 2: Perform kinematics

The encoders L1 program returns only static readings. The Level 2 program, kinematics, 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, thetaDot]. In this task you'll verify these functions and send them to a GUI.

STEPS:

1. Download and run the L2_kinematics.py program
   a. Output the xDot and thetaDot values to the command line by uncommenting the loop as required.
   b. Prepare a new Level 3 file to print PDL and PDR, xDot and thetaDot to the command line.
      i. Create a new Level 3 code. You may use the template under software/python/labs
      ii. import the L2_kinematics.py program so you can access getPdCurrent() and getMotion() functions.
      iii. Call the getPdCurrent() function & print out the values
         1. Confirm that the values don't change when the wheels are static.
         2. Confirm that the magnitude and direction are correct when the wheels are moved
      iv. Call the getMotion() function & print out these values as well
         1. Confirm the values make sense.
2. Set up a nodeRed GUI for both phi dots, xDot, and thetaDot
   a. Import and use the function uniqueFile() from the L2_log.py program for each element you will plot
   b. Export your nodered flow to be paired with your code for submission.
3. Verify your code and your flow work together and submit on eCampus as two files.

Lab Checkoff

TASK 1: THESE ITEMS WILL BE CHECKED OFF IN LAB

- I2C Board fabricated properly:
  - Continuity across horizontal row
  - No connection across vertical rows
- I2C Cables fabricated properly:
  - Left Encoder end matches drawing
  - Right Encoder end matches drawing

- L1Movements respond properly
    - Phi dot Left responds to left wheel movement, correct direction
    - Phi dot right responds to right wheel movement, correct direction
    - X-dot increases from zero when either wheel moves forward
    - T-dot indicator responds (+,-) to wheel advancement (R,L)

TASK 2: THIS SECTION WILL BE GRADED BASED ON YOUR SUBMISSION

- 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. (your level 3 software should be compatible with the provided L1 and L2 softwares – do not modify their functionality)

Lab 5: Compass, Calibration v2

This lab will prepare you to determine your absolute orientation of your robot. Since the encoders only offer information on change in direction, it's necessary to use the compass to find the global direction.

Resources required:

Team Parts	Resources
Assembled SCUTTLE Robot with beaglebone oriented properly.	No special resources

Task 1: Calibrate your compass

There are two parts to calibrating your compass. The first part 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.

STEPS:

1. Calibrate through the rcpy library using sudo rc_calibrate_mag command.
    a. Your beagle must be mounted on the robot, so that the magnetic effects of the motors are considered in the calibration.
    b. Follow the instructions in rc_calibrate_mag which takes 15 seconds of data collection.
        i. The onscreen instruction says to move the board "in all directions" but you only need to reach all directions your beagle will experience in operation.
        ii. The z-axis may remain vertical, and you can try to push the robot through a 360 degrees circle twice.
2. Perform manual calibration.
    a. Run rc_test_mpu -m to show the magnetometer axes printed to screen.
        i. There should be no warning that calibration has not been performed for magnetometer
    b. Next, download and run the loop for L2_heading.py to discover the x and y range
        i. With raw values being printed, rotate your robot and record the max and min achievable magnetometer values for x and y.
        ii. Overwrite the existing xRange and yRange variables in the L2_heading code and rerun.
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).
    b. Use a new L3 file to log your headings (or add this to a previous file which logs data). Create indicators in NodeRed to output the heading.

Lab Checkoff:

- Beagle is oriented correctly on your robot, and in the center (between motor driver and battery pack)

- Your x minimum and maximum are entered into your L2 code (overwrite the existing values)
- Your y minimum and maximum are entered into your L2 code (overwrite the existing values)
- Pointing the robot North gives a value of 0 +/- 8 degrees
- Pointing the robot East gives a value of 90 +/- 8 degrees
- Pointing the robot West gives a value of -90 +/- 8 degrees
- Pointing the robot South gives an absolute value larger than 172 degrees
- Nodered has indicators for scaled X, scaled Y, and Heading
- Nodered has units indicated on the indicators
- Log files are created in your basics directory with meaningful names (ie “magX.txt”)

Lab 6: Closed-loop Speed Control

Overview

TOPIC: The objective of this lab is to tune the gain or K values of a PID control system and understand how each gain directly impacts the control signal and subsequently SCUTTLE's movement.

NOTE: For this lab we will only be using the ONE wheel in the SCUTTLE system. Use your left wheel unless your RH hardware has a problem.

Hardware required:

Team Parts	Resources:
• 12 V wall connector power source • Beagle Bone • Proper i2c bus connections to encoders and beagle • Power and signal wires connected to motor driver • Wheel stand for SCUTTLE • Gamepad (extra credit task)	• Video: Understanding Proportional Control • **Lab Template:** Lab6Template.py

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 values will be represented by PdTargets, the target wheel speed we want to reach in rad/s, and the measured values, represented by PdCurrents which are read by the encoders.

Steps:

1. Prepare Lab6Template.py for running and logging in Closed Loop mode, within Cloud9.
    a. Select Closed Loop operation: uncomment the 'sc.DriveClosedLoop()' function and comment 'sc.DriveOpenLoop()' function.
    b. Set static PD Targets of 9.7 rad/s target phis and comment the line which generates targets from L2_inverse_kinematics.py
        i. *9.7 rad/s is the maximum speed when the wheels are loaded and readily achieved when free spinning.*
        ii. *Note: if your hardware has a problem on one wheel, set that wheel target to zero!*
2. Open L2_speed_control.py and set your kp value.
    a. find where 'driveClosedLoop()' function is defined
    b. Change the kp control value to 0.04
    c. Make sure ki and kd are set to 0.0

    d. Make sure excel_data.csv is in your basics folder
3. Make sure to save your code changes and run L3_PID_Lab.py in your terminal.
    a. Let the program run for about 15 seconds and terminate using ctrl^C
4. Go back to your text editor and open your .csv file and make sure you have at least 100 data samples. (Find 100 lines in the csv file)
    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. NOTE:
        i. If using Excel 2016 or earlier follow go to the next step.
        ii. If using Excel 2017 or later, skip the next step.
5. Open a new Excel Workbook (2016 and prior)
    a. Click on Data→FromText
    b. Find the .csv file you downloaded and import it
    c. Once the Text Import Menu pops up choose:
        i. Delimited→Next
        ii. Make sure only the Comma delimiter selected→Next
        iii. General→Finish
        iv. Choose where you want to paste your data
    d. Skip the next step.
6. Open a new Excel Workbook (2017 and later)
    a. Click on Data→ FromText/CSV
    b. Find the .csv file
    c. Click on Import→Load
7. Graph both the Current Phi's and the Target Phi's. The result should resemble something close to the graph below.
    a. NOTE: Columns in data are in the order described in log.csv_write() function
    b. Make sure to include appropriate Title, Axis, and Legend labels

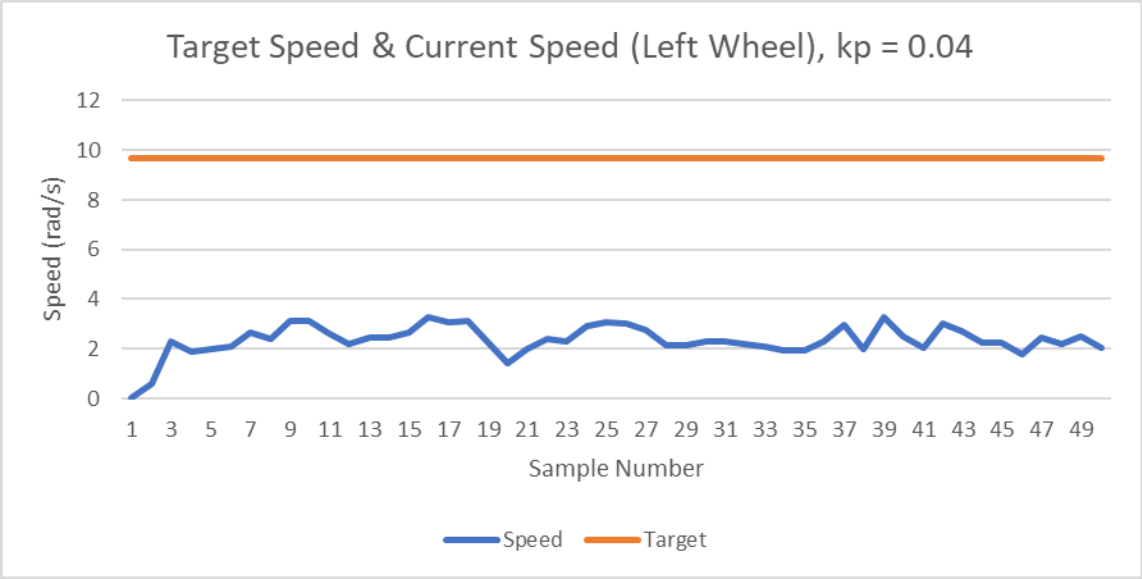


Figure 1. Target & Current Speeds using proportional control

8. Go back to and redo steps 2 through 7, but this time change the kp value to 0.09
9. Questions to answer before moving forward:
    a. Explain why the controller reached an oscillating steady state error but not near our target value when we used a kp value of 0.4
        i. (your answer)
    b. What is causing the oscillations in the Current Phi's after reaching steady state error using these kp values?
        i. (your answer)
    c. Is this the behavior you were expecting from a proportional only controller?
        i. (Your answer)
    d. Were the oscillations in the second graph supposed to decrease? If so why and if not why not?
        i. (Your answer)
    e. What is a potential reason as to why the 0.09 kp value still did not reach the target values?
        i. (Your Answer)
    f. Which additional control term do you think would be most beneficial to reach the target values?

Task 2: Proportional + Integral Control

This section will focus on adding the Integral term and analyzing how it affects our SCUTTLE system

Steps:

1. Utilize L2_speed_control.py while implementing just integral control.
    a. Go back to your text editor and open L2_speed_control.py
    b. Review the driveClosedLoop function & see how the "ki" term will be utilized.
    c. Set the ki value to 0.01 and kp value to 0.0
    d. Run the Lab6Template.py to call the control system into action.
2. Repeat graphing steps depicted in steps 3-8 of task 1.
    a. Keep target phi-dots of 9.7 rad/s.
    b. Make sure to graph the Target Phi's and the Current Phi's
3. Once you observe output of the wheel, go back and change kp to 0.04 and ki = 0.04. Make sure to graph these results. Record observations.
4. Questions to answer before moving forward:
    a. Explain why this tuning of just using the integral term should or should not reach the target value.
        i. (your answer)
    b. Which tuning took longer to reach the target values, using only the integral of ki = 0.01 or the tuning with kp and ki gains. Why?
        i. (your answer)
    c. Out of all the kp tuning in this lab, which kp settings gave the best results. Why?
        i. (your answer)

Task 3: Derivative

In a PID control system, the derivative term is the control signal portion acting on the derivative of the error. This 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'll make a hand calculation of a derivative control signal instead of adding a d-term and running it.

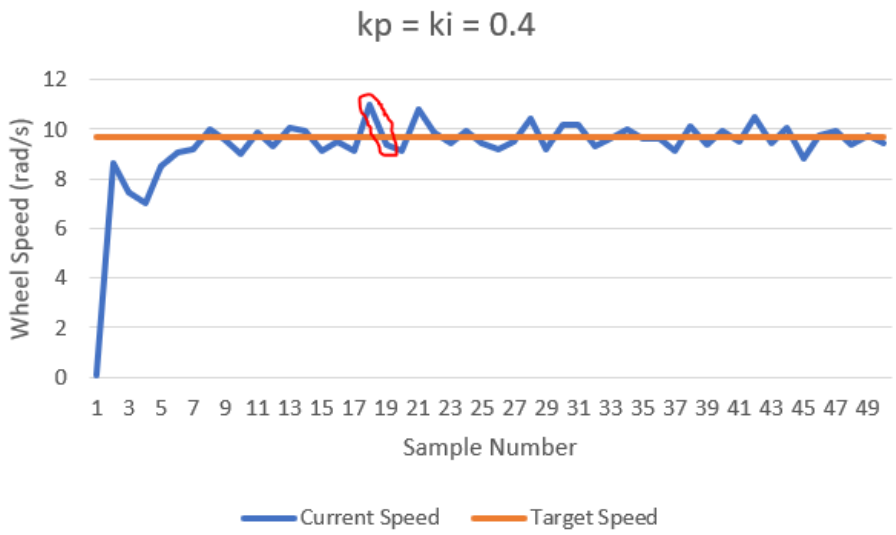


Figure 2. Observation on Derivative of error

Steps:

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 2, 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. Do not use any samples in the first 10 samples from 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 samples.

Questions:

1. The units of the control signal u are PWM percentage in our system, directly sent to the motor. If the value U comes out to 1, you're sending 100% forward power to the motors, and -1 is 100% reverse power.
    a. Given this info, what percent of power will be added to the control signal?
        i. Is it positive or negative?
    b. Looking at the speed and the target during the second sample, is it going to help you reach your target to add the derivative term for this sample?
    c. Describe a situation where the derivative term would be most helpful. Do some research to give a thoughtful answer to this question.

Task 4: Integral Wind – Up [Extra Credit]

1. Navigate to the inverse_kinematics.py file→def getPdTargets(): and make the following changes.
    a. Comment out the code in Figure 1.

```
C = np.clip(C,-9.7, 9.7)
```

Figure 3

    b. Navigate to Lab6Template.py → def_loop_drive(ID): and make sure you are receiving PdTargets from the gamepad. This can be done by making sure your code looks like Figure 2 below.

```
# THIS CODE IS FOR OPEN AND CLOSED LOOP control
#pdTargets = np.array([9.7, 9.7]) #Fixed requested PhiDots; SPECIFICALLY FOR PID LAB
pdTargets = inv.getPdTargets() # populates target phi dots from GamePad
```

Figure 4

    c. Run L3_drive_mt.py and move your joystick to make your SCUTTLE only move one wheel and hold it for about 8 – 10 seconds.
    d. Let go of the joystick. The wheel should still be spinning for a couple seconds after you let go of the joystick.
        i. Accurately explain why this is happening. (HINT: Think of how the Phi Dots are being generated using x-dot and theta-dot)

Lab Checkoff:

TASK 1: PROPORTIONAL CONTROL

Items:

- Lab Report is generated in PDF form.
    - Having heading with Team name, date, and team members
    - Having clean, readable format
- Proportional Graph displaying Target Phi's and Current Phi's (two series)

    - Graph points using kp = 0.04 is presented
    - Graph points using kp = 0.09 is presented
    - Graphs have Axes labeled correctly (x and y)
    - Graphs are clear & readable with good resolution
    - Graphs show precisely 50 data samples
- Meaningful answers
    - Questions 1 & 2
    - Questions 3 & 4
    - Questions 5 & 6

TASK 2: INTEGRAL CONTROL

Items:

- Lab Report is generated in PDF form.
    - Having heading with Team name, date, and team members
    - Having clean, readable format
- Two PI graphs displaying Target Phis and Current Phis (two series on each)
    - Graph for condition: kp = 0, ki = 0.1
    - Graph for condition: kp = 0.04, ki = 0.04
    - Graphs have Axes labeled correctly (x and y)
    - Graphs are clear & readable with good resolution
    - Graphs show precisely 50 data samples
- Meaningful answers to questions
    - 4A
    - 4B
    - 4C

TASK 3: DERIVATIVE CONTROL

Items:

- Hand Calculation for derivative term
    - Graphs have Axes labeled correctly (x and y)
    - Graphs are clear & readable with good resolution
    - Graphs show precisely 50 data samples
- Meaningful answers to questions
    - 1A
    - 1B
    - 1C

Lab 7: Computer Vision & HSV Filter

Overview

TOPIC: Capture images with a 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. Then, generate driving speed targets using the visual target information.

Resources required:

Team Parts	References
• Beaglebone Blue secured in bracket. • Power source for Blue. • Individual laptops • Microsoft USB Camera	VIDEO: Flash a Debian Image VIDEO: View live video using mjpeg streamer VIDEO: Tuning Color Tracking using NodeRED (Short) VIDEO: Tuning Color Tracking using NodeRED (Full Length) PLAYLIST: all SCUTTLE videos Web Page: SCUTTLE home page

Code needed:

- Lab7Template.py
- L2_color_target
- L2_kinematics as kin
- L2_speed_control as sc
- L2_inverse_kinematics as inv
- L2_log as log
- L2_joint as joint
- Machine_vision directory
-

Prelab: Understanding HSV

This prelab introduces you to the HSV color space which will allow you to understand how HSV is different than RGB and why it is used for color tracking.

STEPS

1. Review the "Basic Principle" section of the following wikipedia page to ensure a minimum understanding of the HSV color space. Answer these questions about HSV:
    a. If I view a red apple and I turn off half the lights in the room, I will keep the same:
        i. YOUR ANSWER:_____
    b. If I compare a red apple in a black and white vs color photograph, the only difference should be the:
        i. YOUR ANSWER:_____

Task 1: Run MJPG Streamer and NodeRED Flow

First, we will start up the code that will give you access to the view of the camera. Then we will set up NodeRed which will allow us to begin finding the color range values to track our object. Download the latest versions of the files and dependent files for this lab.

STEPS:

1. Verify that your camera plugged in and recognized by the 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”.
    b. Grab the start_mjpg_streamer.sh and L3_image_filter.py files from the Github and save them into the computer_vision folder you created.
    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. Navigate to nodered_color_tracking.json in the GitHub 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” then “Clipboard”.
        iii. 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
        i. If you have other flows, double click their titles and select “disable.” By default, the imported flow is enabled.
        ii. Click “Deploy”.
        iii. Click on the “Dashboard” (this has a bar chart icon)
        iv. Access the deployed page by clicking on the box with an arrow pointing out of it.
4. Verify you can see a video stream with 3 images, and 6 sliders which you can control.

Task 2: Tuning your HSV filter values.

Firstly we need to find the HSV color range that we want to capture in the image stream. This means 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.

STEPS:

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 object you want to track is being tracked reliably, record the HSV range you have selected.

Task 3: Using your HSV filter values.

Now that you have your minimum and maximum HSV values we can use these values in our color tracking code. This code ultimately generates phi-dots for the wheels to track the target by driving. It’s currently set to perform steering only (no chasing).

STEPS:

1. Open Lab7Template.py and modify the values in the variable “color_range” using [H,S,V] minimums followed by [H,S,V] maximums.
2. Run the program and view just the target capturing information:
    1. Comment the sections for Build speed targets, update vars for control system, and call control system to action.
    2. Run and verify that all the variables make sense
    3. Then, run again with driving activated.

Lab 8: Lidar & Obstacle Detection

Overview

TOPIC:

Measure the surroundings using the Lidar sensor (SICK Model TiM561), discover the nearest obstacle using a Level-2 code, and pass the vector describing this obstacle to the NodeRed GUI.

BACKGROUND:

You won't be successful in using the lidar without understanding some theory and constraints. Review the Lidar slides to understand the functionality, the basics of the code, and the limitations. These are found on the Software section of the SCUTTLE webpage.

Resources required:

Team Parts	References
• Beaglebone Blue secured in bracket. • Power source for Blue. • SICK lidar sensor • Lidar power cable • Power splitter to give 12v to the lidar. • Micro USB cable to connect lidar to the Beaglebone.	VIDEO: Lidar implementation displaying nearest obstacle PLAYLIST: all SCUTTLE videos Web Page: SCUTTLE home page NodeRed Flow: (from github) Lidar dimensions: dimensions on github VIDEO: Fasten your Lidar to the Chassis

Task 1: Prepare software, connections, and bracket

This software includes a level 1, level 2, and level 3 (lab template) python file. To run the lidar, you also need a usb driver and the pysicktim library installed on your linux machine.

STEPS:

1. Install the necessary items to run the Lidar.
    a. Install the python usb library by going to your terminal, & enter the command "sudo pip3 install pyusb."
    b. Install the pysicktim library by entering "sudo pip3 install pysicktim."
    c. Confirm the installations succeeded
2. Download the files for the lab.
    a. Get "L1_lidar.py", "L2_vector.py", and "Lab8Template.py."
    b. Get the nodeRed code from the github and import into your flow.
3. Secure your lidar unit and connect it to USB and Power
    a. Use M3 screws to attach the lidar to the bracket
    b. Use M6x10mm screws and t-slot fasteners to secure the lidar to the chassis.
        i. Follow the video for hardware instructions.

Task 2: Collect measurements and output relevant vector data

Use the level 2 code in this task to run one kind of scenario that may be useful: find the nearest vector to the lidar. This could be used for obstacle detection. The L1 will get all values, the L2 will remove erroneous values and output the nearest vector, and the L3 (lab template) will capture those values and pass them to a log file for NodeRed to display.

STEPS:

1. Verify the lidar is connected to USB:
    a. Run the command "lsusb,"
    b. Verify you see "Bus 001 Device 002: ID 1a40:0101 Terminus Technology Inc. Hub."
    c. This indicates the SICK lidar device ID.
2. Verify the L1_lidar.py program works.
    a. Run the program with the "while" loop uncommented.
    b. Verify that 54 data points [distance(m),angle(deg)] are printed
3. Verify the L2_vector.py program works.
    a. Recomment the Level 1 loop.
    b. Uncomment the level2 loop.
    c. Run the code and verify that the nearest vector is indicated
4. Create NodeRed flow nodes which indicate the distance and angle
    a. Indicate the distance with a bar graph
    b. Indicate the angle, (alpha) with a gauge.
    c. Use appropriate range, titles, and units.
5. Verify that your lidar is accurately detecting the nearest obstacle!

Semester Project Outline

The mobile robotics course project has an open-ended mission but specific requirements. These requirements are designed to:
1) help you in the process of making a robust robot mission
2) make sure that the learning outcomes of the course are being executed in your project scope and
3) create a fair grading criteria while allowing flexibility in your mission.

Overview

The project will use sensors and actuators to perform a novel task.

Overall Requirements

Your project proposal (in powerpoint format accompanied by a presentation) should include the following items:

- MISSION
  - Present the mission of the project.
  - What is the target?
  - What sub-objectives are expected to be met to achieve the mission?
  - How will the sub-objectives help you to perform incremental success and achieve parts of the mission? (not requiring a 100% perfection to achieve some subobjecives)
  - What mathematical conversion, aggregation, or processing will be made to convert the inputs to the outputs or decisions?
- SENSING
  - What sensors will be utilized for the task?
  - What are the raw outputs of the sensor? (units, range, accuracy, error)
  - What are the outputs required to make a decision?
    - What is the gap between raw inputs and required outputs?
  - What software will be written to convert data into decisions?
  - How does your plan guarantee successful decision making?
    - Is the sensor resolution sufficient?
    - Is the sensor accuracy sufficient?
    - Is the sensor range sufficient?
    - What evidence validates these items?
- ACTUATING
  - What actuators will be used for the mission?
  - What calculations must be made to give succesful commands to the actuator?
  - What evidence supports that the actuator/calculation is sufficient?
    - Include units, ranges.
  - How does your plan guarantee successful actuating?
    - Is the actuator resolution sufficient?
    - Is the actuator accuracy sufficient?
    - Is the actuator range sufficient?

        - What evidence validates these items?
- DATA COLLECTION
    - Data is collected in the processes of the mission
    - The data is saved in .csv format, .txt format, or received on a cloud platform that retains the data, or a combination of these.
    - Data type, quantity, range, units, and format are planned.
- REPORTING
    - Useful information is presented to the user.
    - The information that the robot uses to make decisions
    - The information collected as objectives of the mission, if applicable
- ROUTINES
    - Present a routine that will be followed
    - What condition will indicate the start of your program?
    - What condition will mark each decision point?
- MISSION STATUS
    - How will your robot indicate incremental success or failure?
    - Why is your choice of increments, or stages, compatible with the robot system?
    - Why is your choice of stages
- FAILURE MODES
    - What condition will result in a successful demo?
    - What is the boundary condition, or that which will not yield success?
    - How will you provide the required conditions for success?

Description of Requirements

PRESENT THE MISSION:

Offer an overview of the mission. Describe what is included in the mission and describe what is not included as part of the mission (ie, cups of water will be carried but not if the cup exceeds 500g). Explain how the demonstration should start and finish. Explain if this will be a human input, an environmental condition, or a timing-based condition.

SENSING:

Explain what sensor will be used. The existing sensors may be used, but if you do not add any sensor you must at least add some computation of new knowledge from existing sensors. For example, if you will build a map of a room, you may combine encoders and compass data to generate points that describe the floor plan.

ACTUATION:

This has the same idea as the sensors. You may use the **existing actuators** or add **new actuators**. If you use the existing ones, explain how you'll command them to perform new actions. You must explain specific criteria including the range of motion or speed or force, the accuracy required to be successful, and why you predict that your selected actuator will be sufficient in meeting the criteria.

DATA COLLECTION

By default, the robot collects data in order to complete the mission. However, the data collection requirement is one step more. This may take the form of simply logging the values collected to go about the mission, or it may be additional measurements taken to report back to the user, if such data is relevant to the mission. Explain how the data will be used if the mission is unsuccessful, and how the data will be used if the mission is succesful. In the real world, data logs are used to troubleshoot a process and to make improvements. Explain why you have made your choice of data collection, what form will the log take, how many values are expected, their units and their range.

DESCRIPTION OF OVERALL ROUTINE

The robot may perform a circuit routine (lasting forever) or a start and finish routine (driven by some condition) You must describe the whole process that the robot will follow, and how the choice is made to move to each subsequent step or remain in the current step.

FAILURE MODES

Even if your robot fails, your demo should succeed. Address the conditions that will lead to failure for your mission. If your robot fails to complete it's mission, there are two redeeming items. 1) The robot should be prepared to report the conditions (ie, "color target not found," "rfid value has no match,"). 2) the parameters you estimate to lead to success (resolution of a sensor, accuracy of an actuator) simply may not have been met. Because of your planning ahead, you should be able to explain quantitatively why any given routine in your mission failed. Your report following the demo should indicate what prevented the mission completion, quantitatively.

Additionally, your proposal presentation should show what you'll do to set up the environment for success. For example, if you identified that the LIDAR will detect walls nicely but become confused by legs of tables and chairs, you may need to build borders of cardboard. Fully explain based on the limitations of your hardware and software what conditions will need to be controlled and how you will do so.

Mission Ideas

- Ride an elevator
- Collect tennis balls
- Make a map of the room
- Take measurement samples of multiple objects and report info audibly.
- Multimode driving:
    - Set up a program that drives using the gamepad with user control, and autonomously perform a function when a button is pressed on the gamepad.
- System diagnostics:
    - Autonomously drive patterns and have SCUTTLE produce outputs describing it's limitations of speed, turning rate, or accuracy. Store the value as a
- Active obstacle avoidance:
    - Use the lidar to detect the nearest object and continuously distance the robot from the object.

SCUTTLE Kit Check-in

	Complete?	Notes
3 Battery cells above 3.8 volts		
Battery Charger with cable (coiled neatly)		
Box 1: Camera coiled neatly, micro-usb cable, (coiled neatly), and robot labels		
Box 2: All fabricated cables (encoders x 2, i2c jumper, power splitter, beaglebone power, AND I2C BOARD!)		
Box 3: all tools (3,4,6mm allens, phillips screwdriver, 3D tool, battery remover)		
Box 4: 2x 12v PSU in your kit (coiled neatly)		
Foreign items removed from kit		
Micro SD card returned as checked-out		
Undamaged beagle in box:		
Robot in clean configuration (no broken brackets, all chassis hardware installed.)		
scuttle has 3 brackets onboard: (battery, motor driver, beagle)		
All nonstandard parts removed from robot		
Team's Checked-out parts are returned (see sheet)