SCUTTLE BEGINNERS GUIDE

(version 2022.03 - Raspberry Pi)

TABLE OF CONTENTS

SECTION 1: STARTUP

This section is a comprehensive introduction to the software and programming languages that the SCUTTLE operates on. There is minimal hardware instruction as it is assumed that you already have an assembled SCUTTLE kit to use. This section will guide you through the installation, configuration, and operation of any required software. The two main software's that you will use when operating the SCUTTLE are Cloud9 and NodeRed. However, there are other softwares that are used exclusively in the initial setup phase that will also be introduced in this section. The two main programming languages used are Linux Command Line and Python. You will use Linux command line in the terminal of an IDE, such as Cloud9, to run Python scripts which for the most part have already been written for you.

There are also videos that follow along with these instructions on the scuttlerobot.org website under *Guides*.

MODULE 1: FLASHING THE PI

BACKGROUND

To flash the Raspberry Pi, we will flash an image to the micro-SD card that will be inserted in the Pi. An image is a full file and operating system for standalone computers like the Raspberry Pi you will be using. To flash the image to the micro-SD, we will use a software known as an Etcher. Once the OS is installed on the micro-SD we can insert it in the Raspberry Pi and boot from it.

MILESTONE 1: FORMATTING A SD CARD

Before the SD card can be flashed, you must check that it is formatted correctly. Etcher programs only work with SD cards that are formatted for FAT32 which is a specific type of file system.

1. To check whether your SD card is formatted correctly, open the file explorer.
2. If the SD card is plugged in (using the micro-SD/SD adapter), you should see a drive that has *SDHC* next to the name. Right click on that and select *format*.
3. Then in the small window that opens there will be a selection for *file system*.
    a. If it already has FAT32 selected, then the SD card is formatted correctly.
    b. If not, open the dropdown menu for *file system* and select FAT32.

Once the *file system* registers as FAT32 the SD card is ready for use. You are now ready to move to the next Milestone, but do not eject the SD card yet as it must remain connected to the computer to accomplish the next few Milestones.

MILESTONE 2: USING AN ETCHER

Etcher softwares allow us to easily flash an Operating System Image onto the SD cards. For our purposes we will use the Raspberry Pi Imager as they have the latest Pi OS images preloaded, but other etcher softwares such as Balena Etcher also work fine.

1. Download the latest version of Raspberry Pi Imager from the Raspberry Pi website to your computer by going to https://www.raspberrypi.org/software/ and selecting the option to *'Download for Windows'*.
2. Then open the program, allow access if it is requested. You will notice it opens a window with three steps.
3. First, choose the operating system that will be flashed to the SD card.
    a. Go to https://scuttlerobot.org/resources and scroll to the bottom of the page to find the section *SCUTTLE Image (Pi)* and click on *download* to download the latest SCUTTLE Image, *07-07-2021_RASPI_SCUTTLE_V0.21.img,* which can take a few minutes to download.
4. Using the Raspberry Pi Imager, scroll to the bottom of the *Operating System* menu and select *Use Custom.* Then select the *07-07-2021_RASPI_SCUTTLE_V0.21.img* image you just downloaded.
    a. Again, make sure you use the *07-07-2021_RASPI_SCUTTLE_V0.21.img* or else the next steps will not work correctly.
5. Second, choose the storage where the OS image will be flashed.
    a. Make sure you select the SD card for this option. Your Raspberry Pi Imager should now look like Figure 1 below.

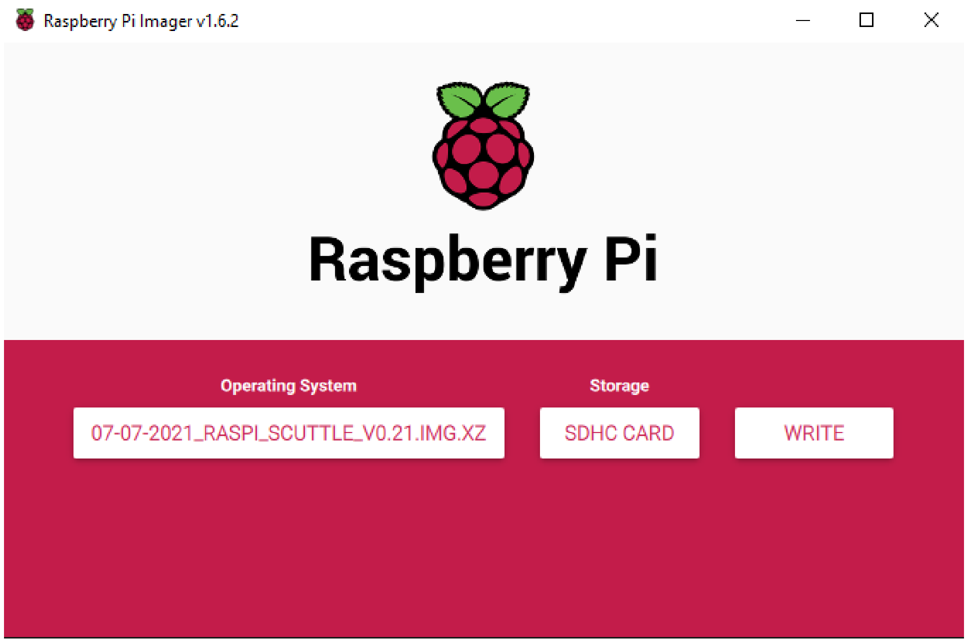

Raspberry Pi Imager v1.6.2
Raspberry Pi
Operating System
07-07-2021_RASPI_SCUTTLE_V0.21.IMG.XZ
Storage
SDHC CARD
WRITE

Figure 1

6. Third, click write and wait for the image to be flashed to the SD card.
    a. It will install and then verify the Image.

   b. When it is done it should say whether it is safe to remove the card or not.

If the SD card was flashed with the *07-07-2021_RASPI_SCUTTLE_V0.21.img* image successfully then you are ready to move to the next Milestone.

MILESTONE 3: BOOTING THE PI FROM THE SD CARD

Booting the Raspberry Pi from the SD card is a rather easy process. It involves plugging the SD card into the Pi while it is turned off and then turning it on. When the Raspberry Pi turns on it will be running on the OS that was flashed to the SD card.

1. Remove the Raspberry Pi from the 3D printed bracket and turn it over.
   a. There is a slot on the back end of the Pi for a micro-SD card.
      i. Insert the SD card you have flashed in that slot.
      ii. It will only go in one way so make sure that it is properly seated in the slot before trying to force it in incorrectly.
   b. Then re-secure the Raspberry Pi to the 3D printed bracket.
2. With the SD card inserted, power the Raspberry Pi using a USB connection to your computer.
   a. You should see a solid red light and a flashing green light.
   b. The green light will stop after a few seconds and only the red light will stay on.

The card is now successfully booted using the SD card. However, at this point you cannot connect wirelessly to control the Pi from your computer as we desire. Module 3 will explain how we can connect the Raspberry Pi to Wi-Fi so that we can interface over Secure Shell Protocol (SSH) wirelessly. Before that, you will need to complete Module 2 to get an understanding of what Linux Command Line is and how you can use it.

MODULE 2: INTRODUCTION TO LINUX COMMAND LINE

BACKGROUND

Linux Command Line is a programming language used to interface with a computer using text commands. It is used in specialized computer programs, known as terminals or shells, which interpret the commands and execute them. There are a wide variety of Linux commands for many different use cases, but we will focus on a few key commands that are prevalent to operating the SCUTTLE system.

It is important to have a good understanding of Linux Command Line as it is the most common language used when interfacing with the Raspberry Pi on the SCUTTLE. The Raspberry Pi is a small and simplified computer which we can communicate with by using Secure Shell Protocol (SSH), allowing us to remotely instruct the Raspberry Pi through a terminal using Linux commands. The different subsystems on the SCUTTLE are all connected to and controlled by the Raspberry Pi. By being able to control the Raspberry Pi, we can control the SCUTTLE.

MILESTONE 1: REVIEW VIDEO COURSE ABOUT LINUX COMMAND LINE

There are many things that we can learn about Linux Command Line but to start off on the right track you will begin by reviewing a few lessons that teach you the basics of the programming language.

1. Start by going to: https://www.codecademy.com/learn and creating a free account.
   a. Review the course *"Learn the Command Line"* at: https://www.codecademy.com/learn/learn-the-command-line
   b. Also review the course *"Learn Bash Scripting"* at: https://www.codecademy.com/learn/bash-scripting
   c. Make sure to save your progress as you review the lessons so that your instructor can verify your completion.

When you have completed watching those videos you should have a better understanding of what Linux Command Line is and be ready to move on to Milestone 2. Hopefully by beginning your learning process with this video course you will get a greater benefit from the next few Milestones which cover the way Linux Command Line is used in operating the SCUTTLE.

MILESTONE 2: USING BASIC COMMANDS IN A TERMINAL

Below is a Cheat Sheet of the common Linux commands you will use when working with the SCUTTLE.

Command	Definition
ls	List the files in the working directory.
mkdir	Creates a new file directory (folder). Ex: mkdir [name of directory]
touch	Creates a new file. Ex: touch [name of file]
rm	Removes a file. Ex: rm [name of file]
mv	Used to rename a file. Ex: mv [old file name] [new file name]
cp	Copies one file to another. Ex: cp [copied file] [modified file]
pwd	Returns the present working directory.
cd	Changes location to a specific directory. Ex: cd [~]/[target directory]
sudo	Execute a command as a "superuser". Ex: sudo [command to execute as superuser]
apt	Uses the APT package manager. Ex: apt install [name of package]
ifconfig	Configures network interfaces. Ex: ifconfig [option] [interface]
hostname -i	Shows the IP address of the system.
ssh	Connect using SSH. Ex: ssh pi@raspberrypi.local
wget	Download a file from a website. Ex: wget [link to GitHub repository file]
ping	Analyzes connection to a host. Ex: ping [website address]
lsusb	Lists any connected USB devices that are detected
ctrl-c	Ends the command running in the terminal.
ctrl-s	Saves the changes you have made to a file (specifically for use in Cloud9).
exit	Logs out of current session.

1. For reference, when you first open a terminal you will notice it says something like *pi@scuttle:/~$*.
   a. This is called a shell prompt and it will show up at the beginning of each new line.
   b. Although it may seem like a weird format, it can be simplified to *username@hostname:location$*.
   c. That simple prompt tells us useful information that can help us understand what system and file structure we are within.
   d. That means in the first example, the user pi at hostname scuttle is in the home file directory.
   e. The *$* simply denotes the end of the prompt and the beginning of where you can type commands.

MODULE 3: CONNECTING TO WIFI

BACKGROUND

As mentioned in Module 1, we will be configuring the Raspberry Pi as *headless* which simply means that there are no separate keyboard and monitor that will be plugged into the Raspberry Pi. We must connect the Raspberry Pi to our Wi-Fi to control it using SSH instead. SSH connections with the Pi are not enabled by default so we must manually create two files in the SD card boot drive to enable it.

MILESTONE 1: ADDING SSH AND WPA_SUPPLICANT FILES TO THE BOOT FOLDER

The two files we will need to create for enabling the Pi to be connected to your local wireless network are the *'ssh'* and *'wpa_supplicant'* files. To add a *'ssh'* and *'wpa_supplicant'* file to the SD cards' boot folder, we will use Notepad which should be included with Windows already. The boot folder can be found in the *File Explorer* under *This PC*.

1. Open Notepad and simply put one space on the document, no words or anything else.
   a. Then go to *file* and *save as 'ssh.txt'* to the boot folder.
   b. Check the boot folder to make sure it appeared.
2. Open a new document, again using Notepad.
   a. This time, copy the following text and paste it in the document:

```
country=US

ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev

update_config=1


network={

    ssid="NETWORK-NAME"

    psk="NETWORK-PASSWORD"

}
```

   b. Modify the SSID and PSK lines with your specific Wi-Fi network name and password.
   c. Please note, you should keep the "*Quotation Marks*" at the beginning and end of your entry.

   d. Also, it is recommended that the network you connect to should be 2.4GHz as the Pi may have problems connecting to 5GHz networks.
   e. Then again go to file and save it as '*wpa_supplicant.conf*'.
      i. You will need to change the file type to *'All files'* for it to allow you to save as .conf instead of .txt which it would default to.
3. Then check the boot folder again in File Explorer and make sure both the *'ssh.txt'* and '*wpa_supplicant.conf'* files were saved there.
   a. If they are, right click on the drive and select '*safely eject'* and then remove the SD card.

MILESTONE 2: BOOTING THE PI AGAIN AND VERIFYING IT IS CONNECTED TO THE INTERNET

With the *'ssh.txt'* and *'wpa_supplicant.conf'* files created, and the SD card safely ejected, you can boot the Raspberry Pi again and now it should be able to connect to the internet and allow for SSH.

1. Again, remove the Pi from its 3D printed bracket and plug the micro-SD into the slot at the bottom back of the Pi.
   a. Power the Pi with a USB connected to your computer.
   b. Allow it a minute or so to fully boot up.

**This next step can be done on Command Prompt, which is built into Windows, or also on free to download softwares such as Git Bash.*

2. Open the Command Prompt and execute the command:

```
ping scuttle.local
```

3. That should return information about how fast the connection was transmitted from the Pi over the internet.
4. It should also give you information such as what the IP address of the Pi is on your network.
   a. If you get this information then you are ready to move to the next milestone.
   b. If you do not get any of this feedback or it returns '*host not found'* or similar errors, check that your computer is on the same network that you entered login information for on the '*wpa_supplicant.conf*' file. Connect to the same network that the Pi is instructed to join and try again.

MILESTONE 3: CONNECTING TO THE PI OVER SSH

If you received feedback from using the '*ping*' command, then that means the Pi is connected to the internet. With the Pi connected to the internet, you are now able to control it wirelessly using SSH as previously mentioned. To connect to the Pi over SSH you will use the Command Prompt again.

1. Open Command Prompt and enter the following two commands:

```
ssh-keygen -R raspberrypi.local

ssh pi@scuttle.local
```

   a. The first command clears out any previous references to scuttle.local so you may get an error like '*host not found'* when running it but this is to be expected.
   b. The second command is an attempt to log in to the raspberry pi as the default user *pi*.
   c. It will most likely prompt you with a warning which you can accept.
   d. Then it should ask for a password for the user *pi*.
      i. The default password for the user *pi* is *raspberry*.
      ii. When you type in the password it hides the text so be extra careful you type correctly.
2. The shell prompt in the terminal should now be '*SCUTTLE ~ $*' meaning the user is now *pi* at the hostname *SCUTTLE*.

You have now successfully connected to the Raspberry Pi over SSH.

MILESTONE 4: CONFIGURING THE PI AND CHECKING FOR UPDATES

Now that you can control the Raspberry Pi over SSH, we will configure the Pi and update it so that it is ready to be implemented.

1. While still accessing over SSH as the user *pi*, type the command:

```
sudo raspi-config
```

   a. This command will open a menu to allow you to configure the Raspberry Pi.
   b. In this menu you will use the arrow keys and the enter button to navigate.
   c. The first things we will change are the hostname and the password in the *System Options*.
      i. Change the hostname from 'SCUTTLE' to 'scuttleXXX' where the XXX is the unit number of your SCUTTLE, such as 'scuttle601'.

ii. Then change the password from 'raspberry' to something unique.

iii. Write this new information in a secure spot so that you can refer back to it if you forget because you will need it each time you want to login as the user *pi* to access the Raspberry Pi.

d. The next thing to change in the *raspi-config* menu is to enable the I2C module

i. From the main *raspi-config* menu, go to *interfacing options > advanced options > I2C*

ii. Ensure that the I2C kernel module is enabled to automatically load by default.

2. With those changes made, try disconnecting the Pi and rebooting it to login using the new credentials you configured.

a. When you connect over SSH this time, you will use the command:

```
ssh pi@scuttleXXX.local
```

b. Again, changing the XXX to your specific SCUTTLE's unit number.

c. Then login as the user *pi* using the updated password you created to ensure the configuration changes you made were implemented.

3. Now to make sure the Pi is up to date; you can run the command:

```
sudo apt-get update -y
```

a. The Pi will most likely be up to date already but if it is not then this command will update it to the most recent versions of software.

Congratulations, your Raspberry Pi is now configured and updated.

MODULE 4: OPERATING CLOUD9

BACKGROUND

Cloud9 is the main software we use to operate the SCUTTLE. It is a cloud-based Integrated Development Environment (IDE) that allows us to write, run, and debug applications from within a browser. The robust Cloud9 IDE also gives us the ability to easily view and manage our file hierarchy and run commands in the bash terminal.

Additionally, if you wish to install Cloud9 on your own rather than using what is included in the *07-07-2021_RASPI_SCUTTLE_V0.21.img* you flashed then there is a step by step guide called *SCUTTLE Cloud9InstallGuidePi* in the supplemental folder of the SCUTTLE GitHub repository located at: https://github.com/scuttlerobot/SCUTTLE/blob/master/docs/supplemental/SCUTTLE%20Cloud9InstallGuidePi.pdf

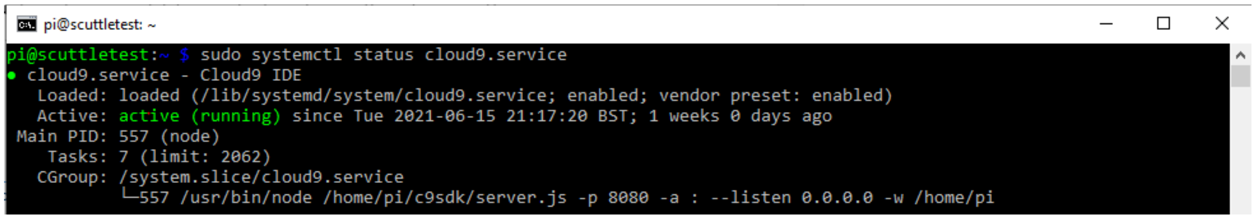
MILESTONE 1: VERIFY PROPER CLOUD9 INSTALLATION

Assuming that you flashed your SD card with the *07-07-2021_RASPI_SCUTTLE_V0.21.img* then you should already have Cloud9 installed and configured on your Raspberry Pi. To verify that it is installed and functioning correctly there are a few things we can do. We will begin with checking the status of the service file that was made to enable Cloud9 to start each time the Pi is booted up. Then we will also check a few configuration settings in the Cloud9 IDE.

1. Start by opening Command Prompt and logging in as the user Pi using SSH.
2. Then execute the following command to check the status of the Cloud9 service file.

```
sudo systemctl status cloud9.service
```

   a. This should return results similar to Figure 2 below showing that the Cloud9 IDE is *active (running).*



```
pi@scuttletest:~ $ sudo systemctl status cloud9.service
● cloud9.service - Cloud9 IDE
   Loaded: loaded (/lib/systemd/system/cloud9.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2021-06-15 21:17:20 BST; 1 weeks 0 days ago
 Main PID: 557 (node)
    Tasks: 7 (limit: 2062)
   CGroup: /system.slice/cloud9.service
           └─557 /usr/bin/node /home/pi/c9sdk/server.js -p 8080 -a : --listen 0.0.0.0 -w /home/pi
```

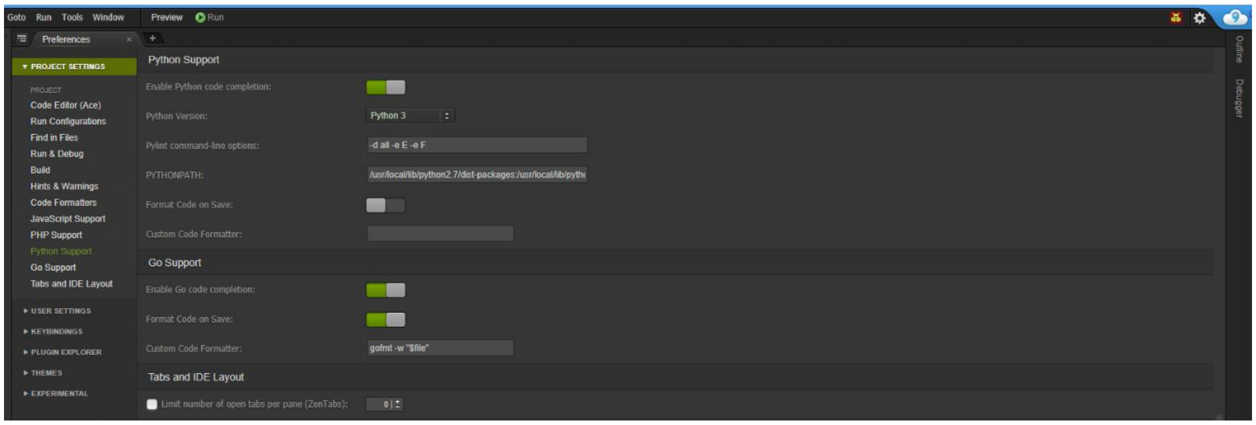
Figure 2

   b. If running that command returns that your Cloud9 is active and running then it is installed and enabled correctly.

3. Now that you have verified Cloud9 is running, try accessing it.
   a. Open a new tab in your browser.

   b. Search for *'scuttleXXX.local:8080'* where XXX is your SCUTTLE unit number that you modified in milestone 4 of module 3 in the first section, then the Cloud9 IDE should load.

4. We need to check which version of python is supported in Cloud9. The default is Python2, but we need it to be set to Python3 because all the python files we will run are based on Python 3.
   a. To check this, go to the Cloud9 settings by clicking the gear in the top right of the IDE.
   b. This will open a preferences menu.
   c. In the menu, navigate to Project Settings > Python Support > and make sure the Python version is Python 3 similar to Figure 3 below.



```
Python Support
Enable Python code completion:
Python Version:                  Python 3
Pylint command-line options:     -d all -e E -e F
PYTHONPATH:                      /usr/local/lib/python2.7/dist-packages:/usr/local/lib/pyth
Format Code on Save:
Custom Code Formatter:
Go Support
Enable Go code completion:
Format Code on Save:
Custom Code Formatter:           gofmt -w "$file"
Tabs and IDE Layout
Limit number of open tabs per pane (ZenTabs):   0
```

Figure 3

   d. If it is not already set to Python3 then modify it so that it is Python3.

5. In some images, there is an error that will cause problems when trying to use Pulse Width Modulation (PWM). To resolve this potential problem, make sure that *Jetson.GPIO* is uninstalled on your Raspberry Pi by using the following command.

```
sudo pip3 uninstall Jetson.GPIO
```

After this is done you have verified that Cloud9 is installed and configured properly and can move on to the next Milestone.

MILESTONE 2: MANAGING DIRECTORIES IN CLOUD9

Cloud9 has many useful features we will use, all within a single IDE. On the left side of the IDE is a file directory. An essential operation you will perform in the Cloud9 IDE will be moving files within folders and directories. With Cloud9, you can easily drag and drop files from one folder to another in the workspace file directory on the left side of the IDE. Let us begin by navigating to the home folder (~).

1. On the left side of the IDE is the workspace which shows our file directories.
    a. In the top right of the workspace is another gear, different from the settings at the top right of the IDE.
    b. Click on the gear and a drop-down menu will appear, select the option to *"Show Home in Favorites"* so that it is like Figure 4 shown below.

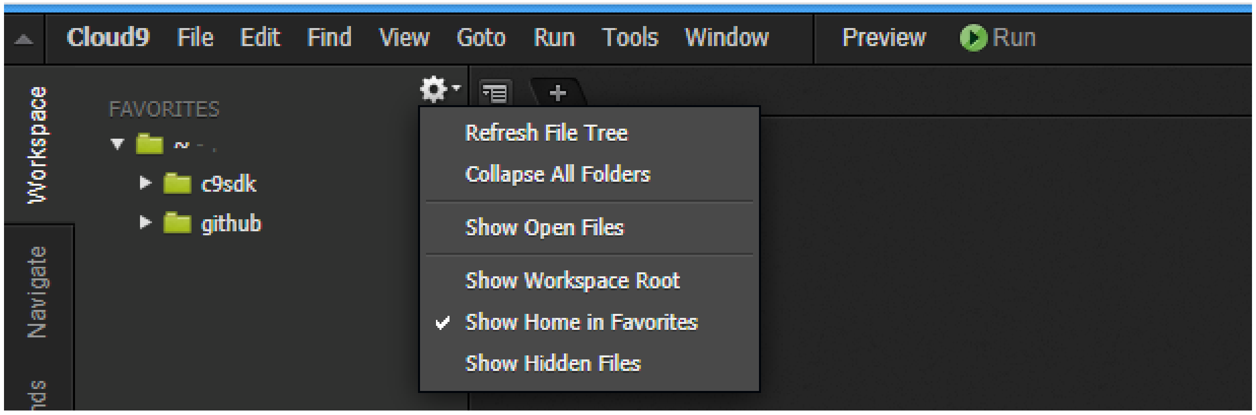
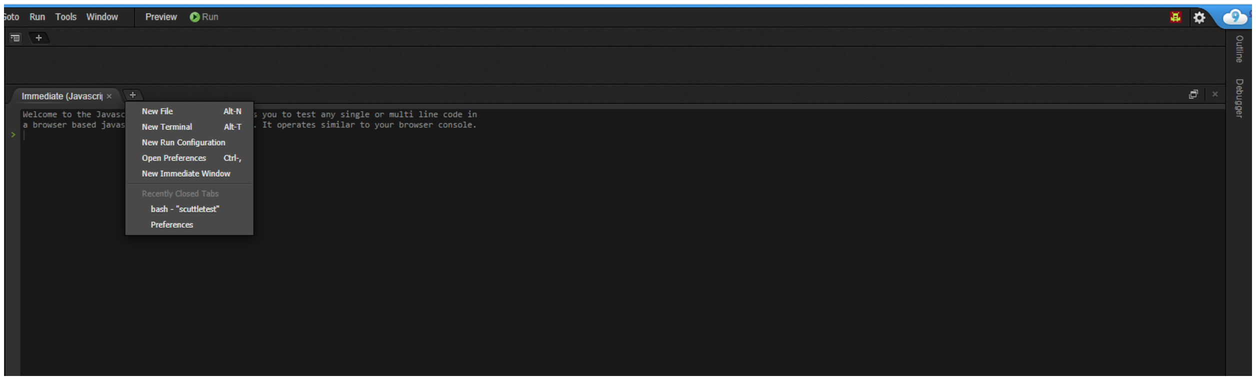


Figure 4

    c. You will notice that now the top folder in the workspace is now (~). That is the home folder.
2. Then start a new bash terminal by clicking the (+) and selecting *New Terminal* like Figure 5.



```
Welcome to the Javasc                          s you to test any single or multi line code in
a browser based javas                          . It operates similar to your browser console.
>
```

Figure 5

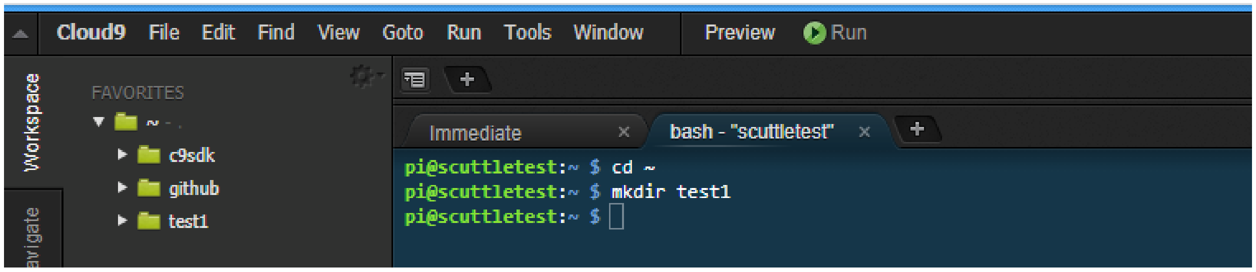
3. Now we will practice the two main ways that we can make folders (file directories) in Cloud9.
    a. The first way is by command line. In the bash terminal at the bottom of the IDE, execute the following command to make sure you are in the home directory

```
cd ~
```

        i. Then use the following command to make a folder that will be in the home directory, call it *test1:*

```
mkdir test1
```

ii. You should notice that the folder you just created appeared in the workspace under the home directory as shown in Figure 6 below:

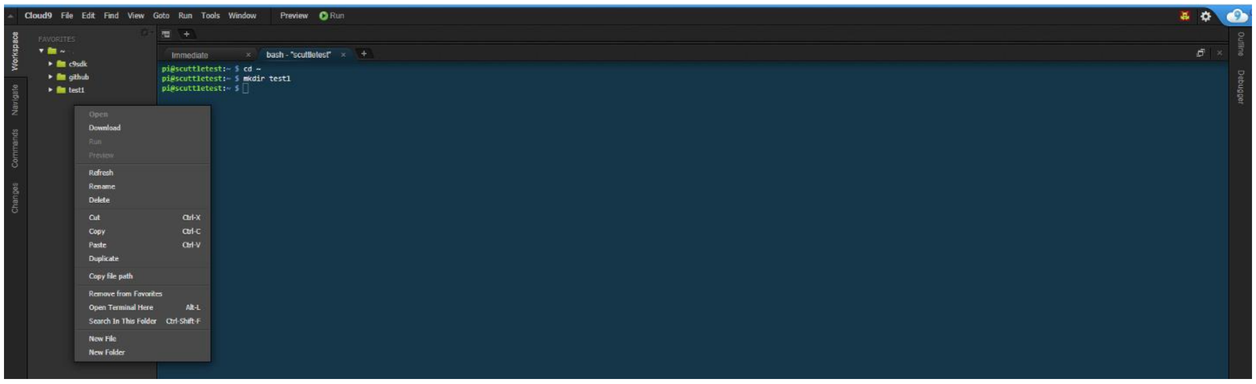


```
pi@scuttletest:~ $ cd ~
pi@scuttletest:~ $ mkdir test1
pi@scuttletest:~ $
```

Figure 6

b. The second way is to manually create a folder, which is done by right-clicking in the workspace.

i. This will open a drop-down menu as shown in Figure 7 below.

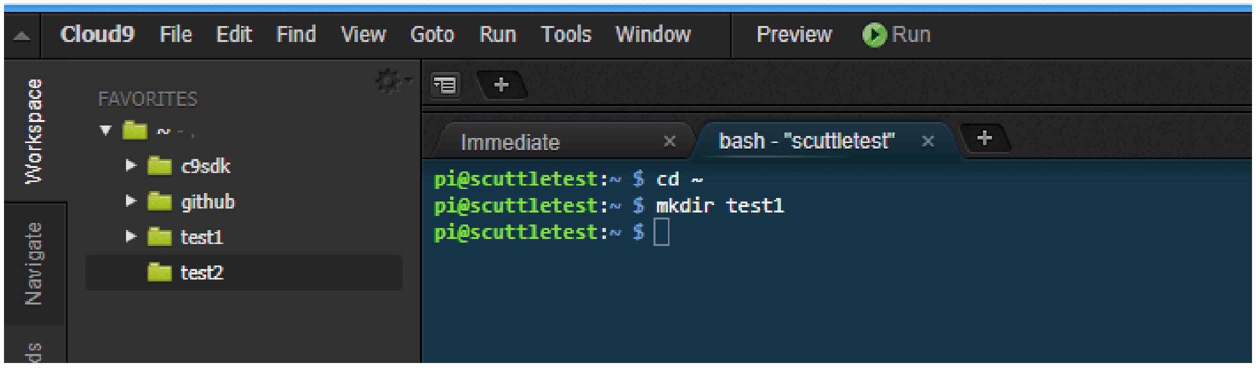


```
pi@scuttletest:~ $ cd ~
pi@scuttletest:~ $ mkdir test1
pi@scuttletest:~ $
```

Figure 7

ii. At the bottom of the drop-down menu is an option to create a "*New Folder*" which is what we want to click on.

iii. Then name the new folder *test2* like in Figure 8:



```
pi@scuttletest:~ $ cd ~
pi@scuttletest:~ $ mkdir test1
pi@scuttletest:~ $
```

Figure 8

Congratulations, now you have created two files using two different methods and learned about the file directory structure in the Cloud9 IDE.

MILESTONE 3: CREATING AND RUNNING PYTHON PROGRAMS IN PYTHON

Cloud9 also makes the process of creating and editing python scripts easy. Python is a programming language that is used for many general purposes and is easy to read, that is part of the reason why we use it for programming the SCUTTLE. Similar to the last milestone, a file can be made in Cloud9 using the command line or manually.

1. We will start by creating a python file by command line and saving it to the *test1* folder we made by using command line.
   a. Start by using the *cd* command in the bash terminal to navigate to the *test1* folder.

```
cd test1
```

   b. Then use the *touch* command in the bash terminal to create a file named *testscript1.py*.

```
touch testscript1.py
```

      i. The ".py" extension at the end of the file name makes it a python file.

2. You should now be able to see the file you created inside the *test1* folder in the workspace.
   a. If you double-click on the file in the workspace it will open the file where you can view or edit it.
      i. Since you just created the file it will be blank, but you can type in the following:

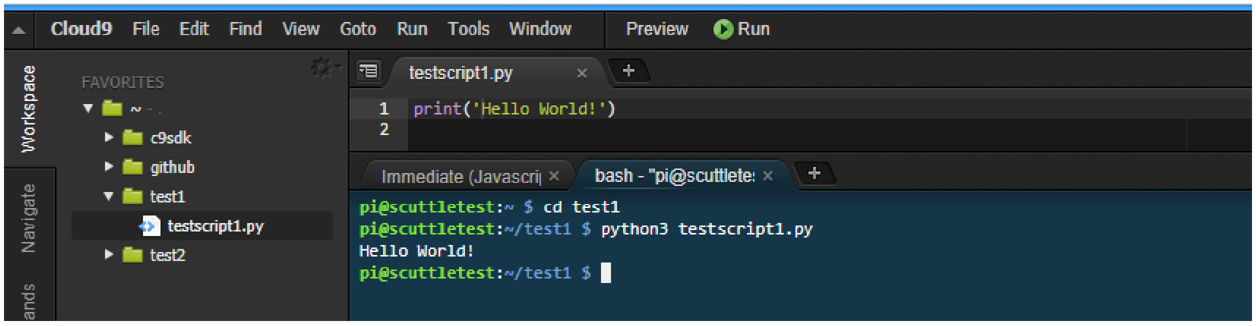
```
print('Hello World!')
```

      ii. Then use '*ctrl+s*' to save the changes to the file.

3. Now go to the bash terminal and execute the following command:

```
python3 testscript1.py
```

   a. This will then run the file and should print out '*Hello World!*' to the terminal like in Figure 9:



```
print('Hello World!')
```

```
pi@scuttletest:~ $ cd test1
pi@scuttletest:~/test1 $ python3 testscript1.py
Hello World!
pi@scuttletest:~/test1 $
```

Figure 9

4. That is how you create, edit, and run a python file using command line in Cloud9.
5. Now we will do the same but manually, which is done by right-clicking in the workspace.

   a. This will open a drop-down menu as shown in Figure 7.
      i. Near the bottom of the drop-down menu is an option to create a "*New File"* which is what we want to click on.
      ii. Then name the new file *testscript2.*
6. Next, double-click on the file in the workspace to open the file where you can view or edit it.
   a. Type in the following again:

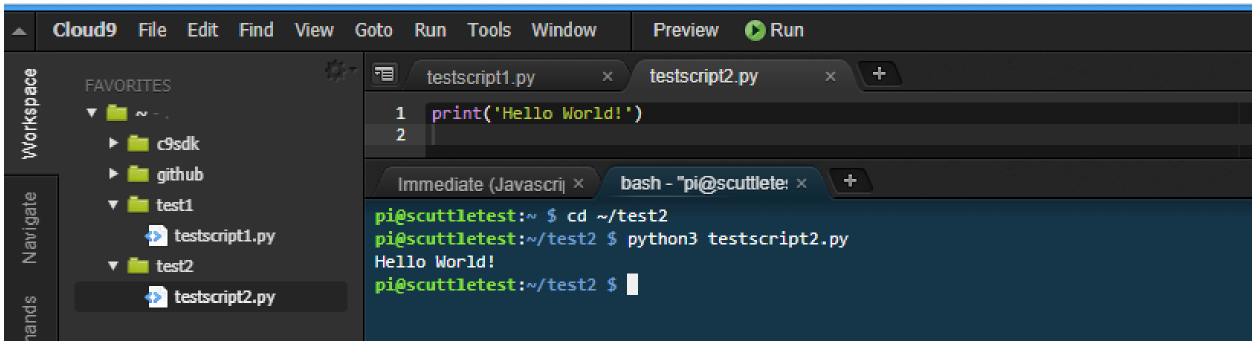
```
print('Hello World!')
```

   b. Then use '*ctrl+s'* to save the changes to the file.
7. Now go to the bash terminal and execute the following commands one at a time:

```
cd ~/test2

python3 testscript1.py
```

   a. This will navigate to the correct folder and run the file which should print out '*Hello World!*' to the terminal as you can see in Figure 10:



```
print('Hello World!')
```

```
pi@scuttletest:~ $ cd ~/test2
pi@scuttletest:~/test2 $ python3 testscript2.py
Hello World!
pi@scuttletest:~/test2 $ 
```

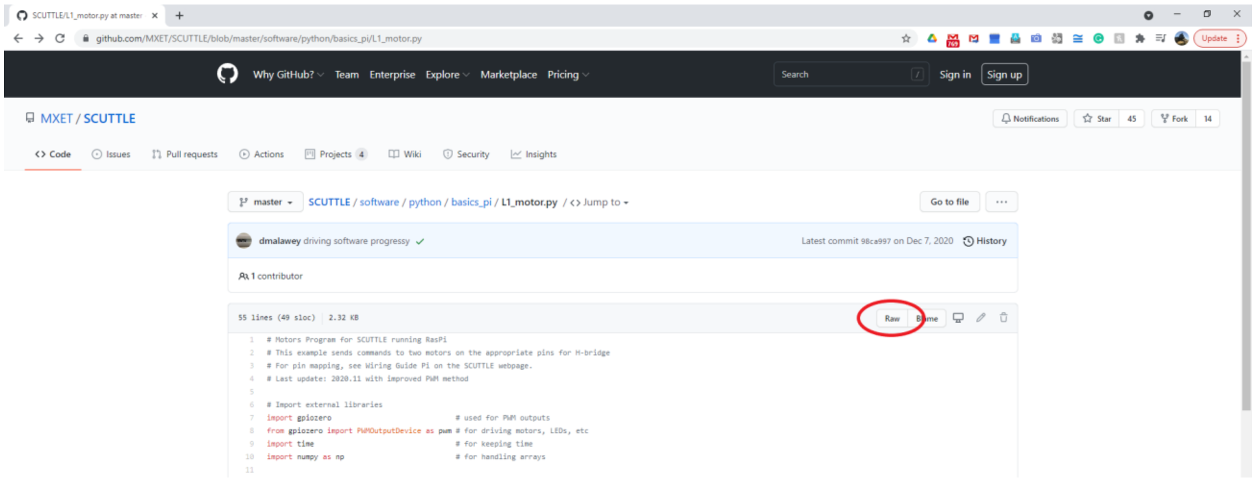
Figure 10

Congratulations, you have now created, edited, and run two python files using two different methods.

MILESTONE 4: GETTING SCUTTLE FILES FROM GITHUB TO CLOUD9

One common task you will perform in Cloud9 is copying Python files from the SCUTTLE GitHub repository which we can access at https://github.com/scuttlerobot/SCUTTLE/tree/master/software/python/basics_pi. The python files we get from the SCUTTLE GitHub have already been written and tested and allow us to easily run them in Cloud9.

1. First, simply find the file you want to copy from GitHub and select the option to view the raw code.
   a. Figure 11 below shows how you can view the raw code, using *L1_motors.py* as an example:


github.com/MXET/SCUTTLE/blob/master/software/python/basics_pi/L1_motor.py
MXET / SCUTTLE
master
SCUTTLE / software / python / basics_pi / L1_motor.py
Go to file
History
Raw
Blame

dmalawey driving software progressy ✓ Latest commit 98ca997 on Dec 7, 2020

1 contributor

55 lines (49 sloc) | 2.32 KB

```
# Motors Program for SCUTTLE running RasPi
# This example sends commands to two motors on the appropriate pins for H-bridge
# For pin mapping, see Wiring Guide Pi on the SCUTTLE webpage.
# Last update: 2020.11 with improved PWM method

# Import external libraries
import gpiozero                          # used for PWM outputs
from gpiozero import PWMOutputDevice as pwm # for driving motors, LEDs, etc
import time                              # for keeping time
import numpy as np                       # for handling arrays
```

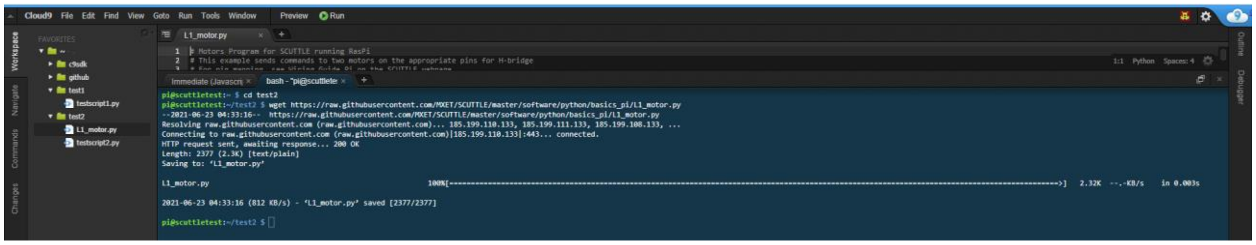
Figure 11

b. Clicking the *Raw* button will show the raw python code, that will change the URL for the page.

2. Then copy the URL for the Raw
3. Go back to Cloud9 and navigate to the folder you want to copy the file to.
4. We will use the *test2* folder, so *cd* into that:

```
cd test2
```

5. Then in the terminal use the '*wget*' command followed by pasting the Raw URL you copied like Figure 12:



```
pi@scuttletest:~ $ cd test2
pi@scuttletest:~/test2 $ wget https://raw.githubusercontent.com/MXET/SCUTTLE/master/software/python/basics_pi/L1_motor.py
--2021-06-23 04:33:16--  https://raw.githubusercontent.com/MXET/SCUTTLE/master/software/python/basics_pi/L1_motor.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.111.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2377 (2.3K) [text/plain]
Saving to: 'L1_motor.py'

L1_motor.py                    100%[------------------------------------------------->]   2.32K  --.-KB/s    in 0.003s

2021-06-23 04:33:16 (812 KB/s) - 'L1_motor.py' saved [2377/2377]

pi@scuttletest:~/test2 $
```

Figure 12

a. You should notice that the *L1_motors.py* file is now in the *test2* folder in the file workspace.

b. *wget* is used when we have individual files we want to copy.

6. When we want to copy the entire repository instead of individual files, there is a different method we use.

a. The *git clone* command is what we use to do this.

b. If you flashed your Pi with the *07-07-2021_RASPI_SCUTTLE_V0.21.img* then the entire SCUTTLE repository should already be in you file workspace.

c. If you were to try it on your own, you would first *cd* into the home directory ~.

d. Then use the *git clone* command to copy the entire SCUTTLE repository to your Cloud9:

```
git clone http://www.github.com/MXET/SCUTTLE
```

That concludes the process for copying files from the SCUTTLE GitHub. It also concludes this module on working with Cloud9. You are now ready to move to the next Module and learn about NodeRed, the second main software we use with the SCUTTLE.

MODULE 5: OPERATING NODERED

BACKGROUND

NodeRed is the second main software that is used in the operation of the SCUTTLE. We can use NodeRed to easily create an interactive Graphical User Interface (GUI) by dragging, dropping, and connecting prebuilt nodes. This gives us a superior method to view live information from the SCUTTLE in a more visual way than simply printing out information to the terminal.

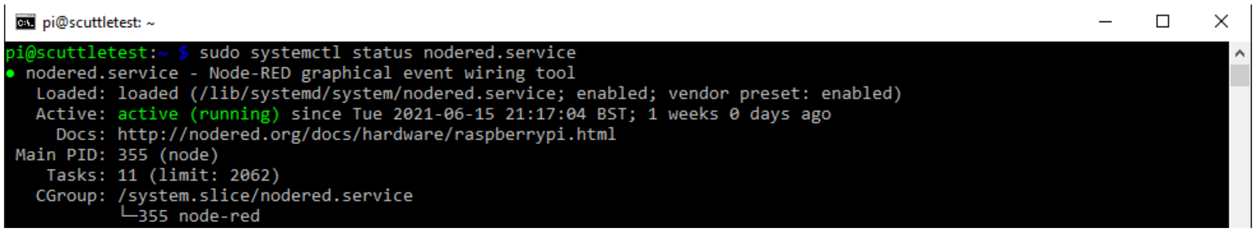
MILESTONE 1: VERIFY PROPER NODERED INSTALLATION

Assuming that you flashed your SD card with the *07-07-2021_RASPI_SCUTTLE_V0.21.img* then you should already have NodeRed installed and configured on your Raspberry Pi. To verify that it is installed and functioning correctly there are a few things we can do. We will begin with checking the status of the service file that was made to enable NodeRed to start each time the Pi is booted up. Then we will also check a few add-ons in NodeRed.

1. Start by opening Command Prompt and logging in as the user Pi using SSH.
2. Then execute the following command to check the status of the NodeRed service file.

```
sudo systemctl status nodered.service
```

   a. This should return results similar to Figure 13 below showing that NodeRed is *active (running).*



```
pi@scuttletest: ~
pi@scuttletest:~ $ sudo systemctl status nodered.service
● nodered.service - Node-RED graphical event wiring tool
   Loaded: loaded (/lib/systemd/system/nodered.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2021-06-15 21:17:04 BST; 1 weeks 0 days ago
     Docs: http://nodered.org/docs/hardware/raspberrypi.html
 Main PID: 355 (node)
    Tasks: 11 (limit: 2062)
   CGroup: /system.slice/nodered.service
           └─355 node-red
```

Figure 13

   b. If running that command returns that your NodeRed is active and running then it is installed and enabled correctly.

3. Now that you have verified NodeRed is running, try accessing it.
   a. Open a new tab in your browser.
   b. Search for *'scuttleXXX.local:1880'* where XXX is your SCUTTLE unit number that you modified in milestone 4 of module 3 in the first section, then NodeRed should load.
   c. When NodeRed loads, it should look like Figure 14 below.

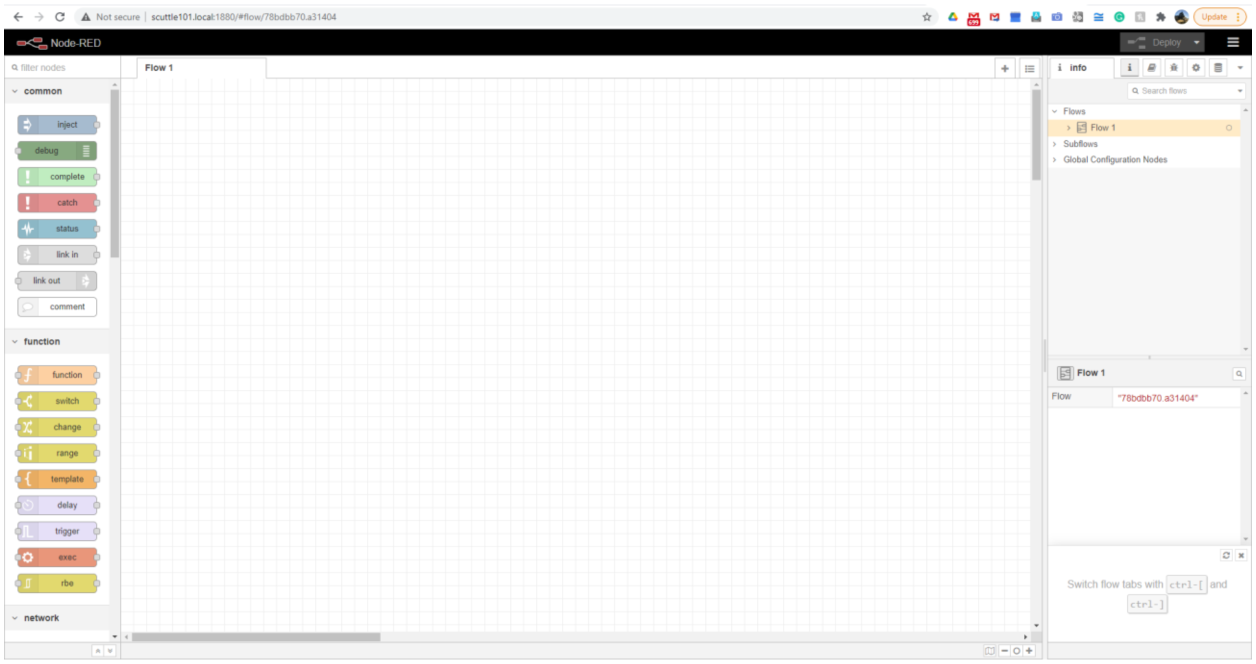


Figure 14

4. One important thing to remember when using NodeRed is that after you have modified a flow, in any way, you must click the *Deploy* button in the upper right to save your changes and for them to be reflected in the GUI.
    a. When you have made any changes since your last deploy, the *Deploy* button will turn red. This is another reminder to click it before you open the dashboard.

MILESTONE 2: VERIFY NODERED PACKAGE INSTALLATION

Again, if you flashed your SD card with the *07-07-2021_RASPI_SCUTTLE_V0.21.img* then you should already have the additional NodeRed packages installed on your Raspberry Pi. Each package that you can install on NodeRed consists of new nodes that will be added to the node palette. The nodes you have in the node palette can be dragged into the workspace and connected to configure the widgets you will have in your NodeRed GUI.

There is a dashboard package that features 21 additional nodes that add many prebuilt user interface nodes for us to use which makes it easy to build a clean and functional GUI. It includes many prebuilt GUI elements such as gauges, charts, and text notifications that we use to visually show the data from the sensors on the SCUTTLE. However, it does not come preinstalled on NodeRed so we must verify that it is installed through the image or install it ourselves.

1. In the node palette on the left side, scroll down to the bottom to make sure that the section of nodes called 'dashboard' is installed like in Figure 15 below.

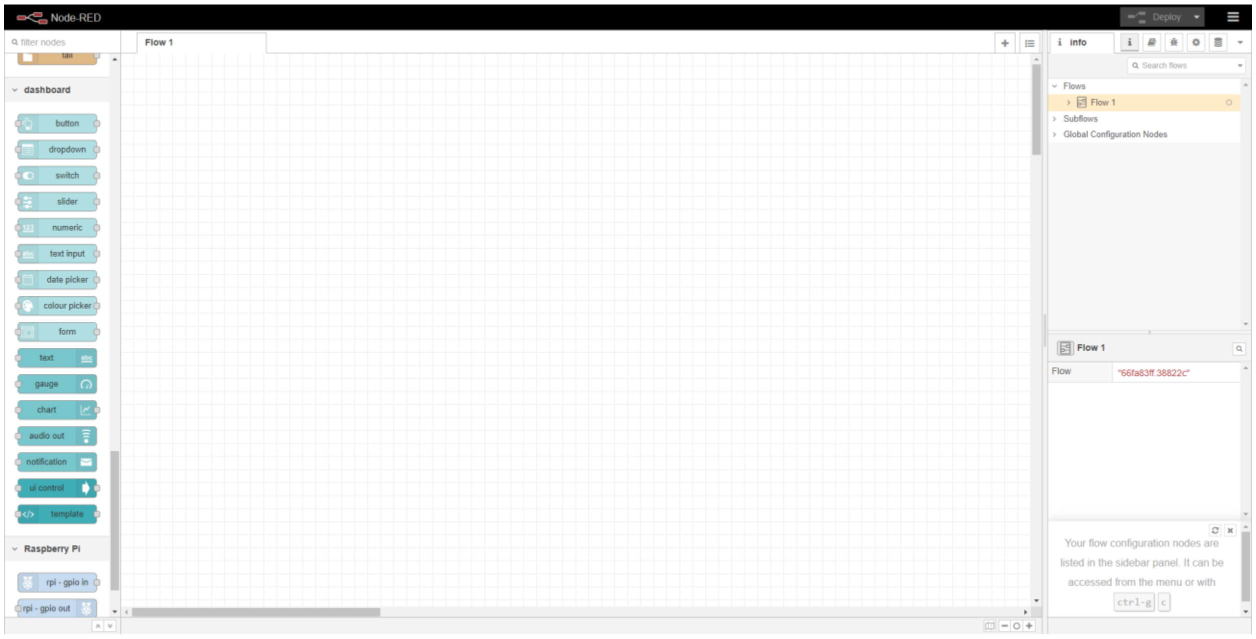


Figure 15

2. You can also check what packages of Node palettes have been installed by clicking on the three lines in the top right next to the '*Deploy*' button.
   a. That will open a dropdown menu where you should select '*manage palette*'.
      i. This will open a user settings window as shown in the Figure 16 that shows the dashboard package in the nodes that are installed.

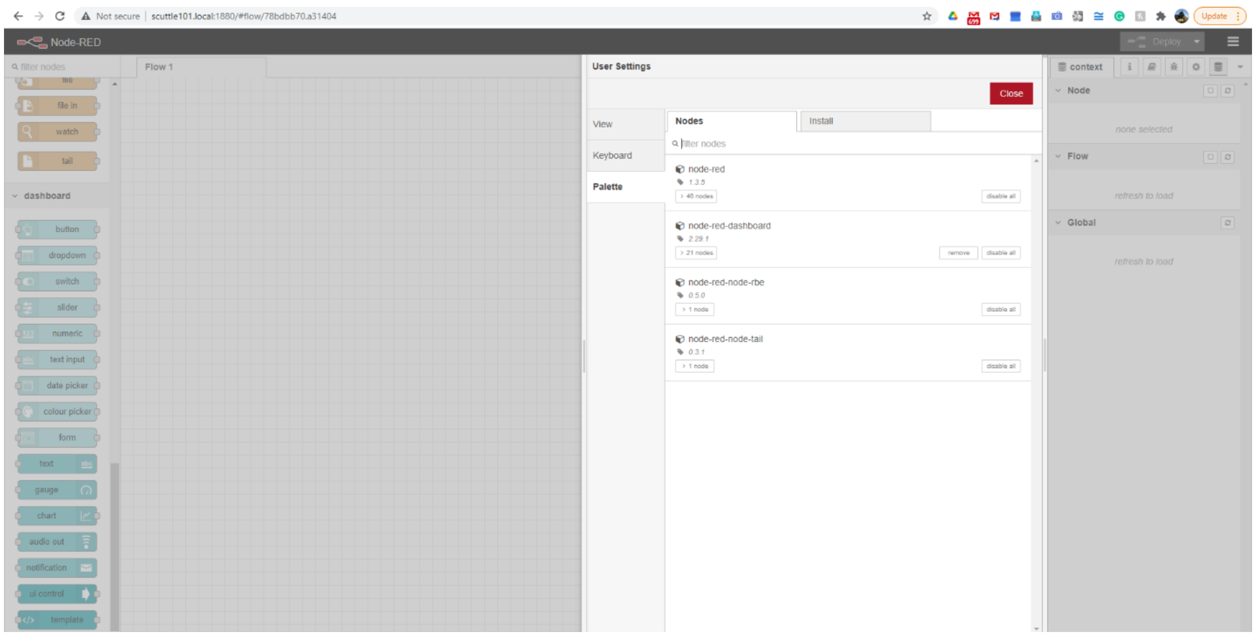


Figure 16

If that package is installed correctly then you have verified that NodeRed is installed and configured correctly.

SECTION 2: EXAMPLES

This section will include examples that you can try for yourself on your own SCUTTLE. Each Module in this section will walk you through the process of connecting a specific sensor and running a program that allows us to use that individual sensor to accomplish a task. This is an excellent way to learn about some common sensors and their applications in mobile robotics systems.

MODULE 1: CAMERA

BACKGROUND

This module will introduce you to the topics of image processing, computer vision, and HSV color spaces. We will use a USB camera mounted to the front of the SCUTTLE to capture images of the area in front of the SCUTTLE. We will stream those images to NodeRed where we will use a premade flow with sliders to filter the range of colors in order to create a "mask". A python script will also be run in Cloud9 that converts the masked area of the images into a visual target with a radius and (x,y) coordinates using OpenCV libraries. Finally, another python script will be used to have the SCUTTLE track the object using the visual target data.

MILESTONE 1: RUNNING THE MJPG STREAMER AND NODERED FLOW

To stream the images from the camera to NodeRed we will utilize MJPG-streamer which is a command line application that copies JPEG frames from input plugins to output plugins over IP-based networks. It was written for embedded devices with limited processing capabilities which makes it ideal to work with the Raspberry Pi. However, before we use MJPG-streamer we must mount the camera and verify it is recognized by the Raspberry Pi.

1. You will need the USB camera, 3D printed camera bracket, and an M5 screw and nut like Figure 17 below:

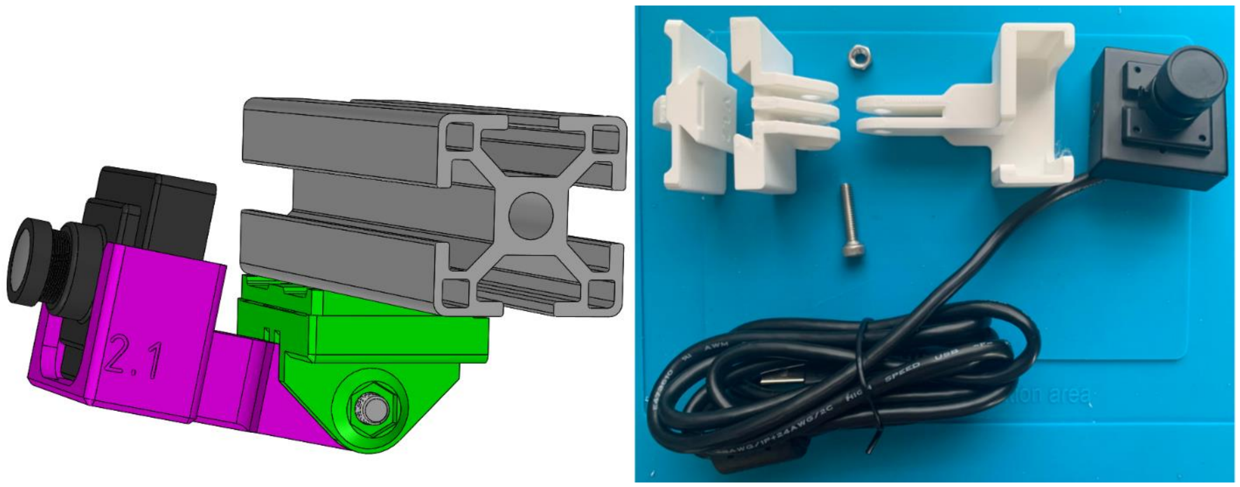


Figure 17

   a. The left side of Figure 17 shows how it should be assembled and mounted; the right side shows the individual parts.
   b. Mount the assembled bracket to the bottom of the SCUTTLE front chassis.
   c. Make sure the SCUTTLE is turned off before making any wiring changes.
      i. Then connect the USB end of the camera cable to a USB port on the Raspberry Pi.
      ii. Now that the wiring changes have been made it is time to power on the SCUTTLE.
2. Open Cloud9 by going to your browser and searching for 'scuttleXXX.local:8080' where XXX is your SCUTTLE unit number that you modified in milestone 4 of the module 3 in the first section, then the Cloud9 IDE should load.
   a. In the Cloud9 terminal, run this command to see USB devices the Raspberry Pi recognizes:

```
lsusb
```

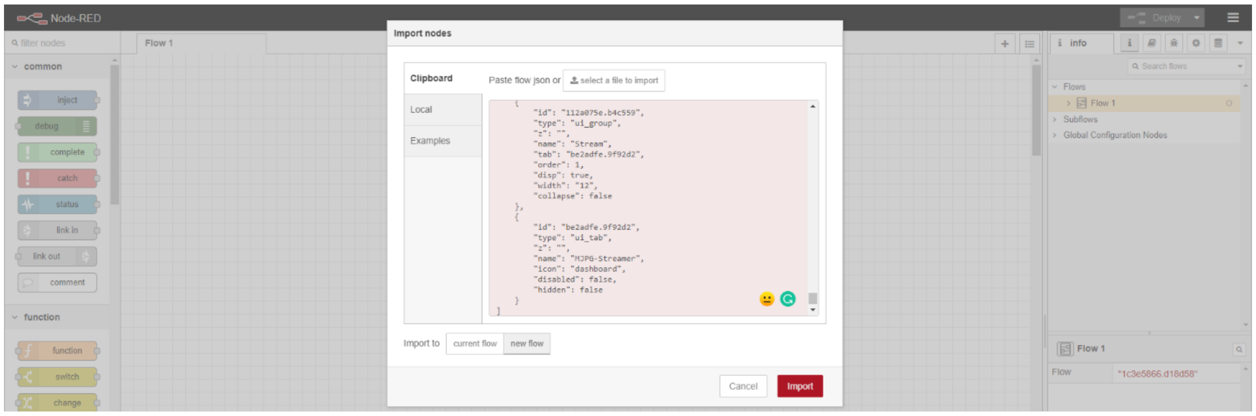
   b. You should notice that it lists an *"Alcor Micro Corp."* device, which is the USB camera.
      i. If nothing is listed after the *lsusb* then try plugging the USB into a different USB port on the Pi and make sure it is fully inserted in the correct orientation.
3. Now make a *Module1-Camera* folder in your home file directory using the command:

```
mkdir Module1-Camera
```

4. Then go to the python computer vision folder in the SCUTTLE GitHub or within the copied repository in Cloud9 to find the two files you want to copy to the new *Module1-Camera* folder you just created.
   a. View the raw code for both the *start_mjpg_streamer_pi.sh* and *L3_image_filter.py* files and use the *wget* command in the Cloud9 terminal to copy them to your *Module1-Camera* folder.
   b. Check the file directory to verify that the two files now appear in the *Module1-Camera* folder.
   c. If they both appear as they should, then use the following command to run both files at once:

```
bash start_mjpg_streamer_pi.sh L3_image_filter.py
```

5. Now go to SCUTTLE GitHub NodeRed color tracking and copy the entire script.
   a. Then open NodeRed by going to your browser and searching for 'scuttleXXX.local:1880' where XXX is your SCUTTLE unit number that you modified in milestone 4 of module 3 in the first section, then NodeRed should load.
   b. Open the menu by clicking on the button in the top right of the NodeRed environment.
      i. In the menu that appears, navigate to import>clipboard and paste the entire script you copied from the NodeRed color tracking folder of the GitHub.
      ii. Then select import>new flow to import the flow without overwriting the current flow.



```
    {
        "id": "112a075e.b4c559",
        "type": "ui_group",
        "z": "",
        "name": "Stream",
        "tab": "be2adfe.9f92d2",
        "order": 1,
        "disp": true,
        "width": "12",
        "collapse": false
    },
    {
        "id": "be2adfe.9f92d2",
        "type": "ui_tab",
        "z": "",
        "name": "MJPG-Streamer",
        "icon": "dashboard",
        "disabled": false,
        "hidden": false
    }
]
```

Figure 18

c. It should create a flow that looks like Figure 19 below:

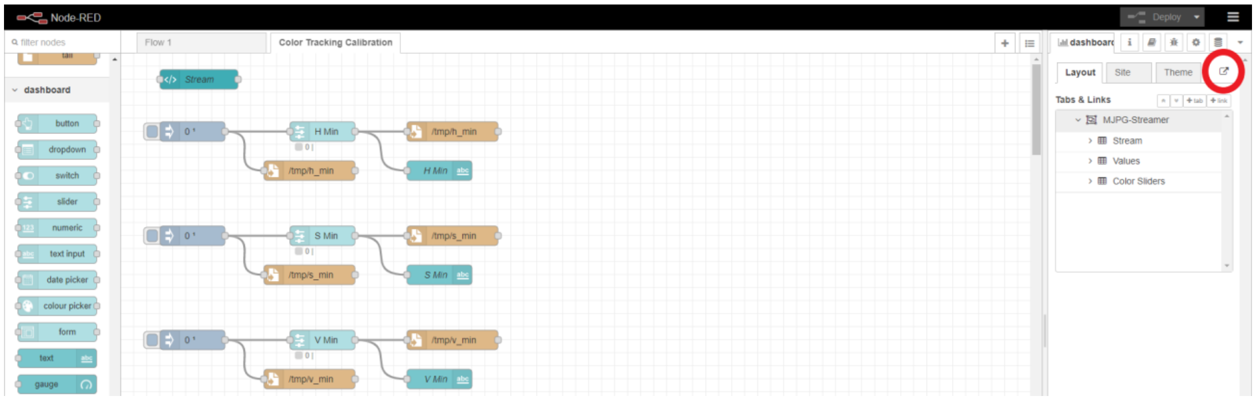


Figure 19

d. Now click the “Deploy” button in the upper right of NodeRed and then click the button that is circled in red within Figure 19, and it should open a new tab in your browser with six sliders like Figure 20 below.

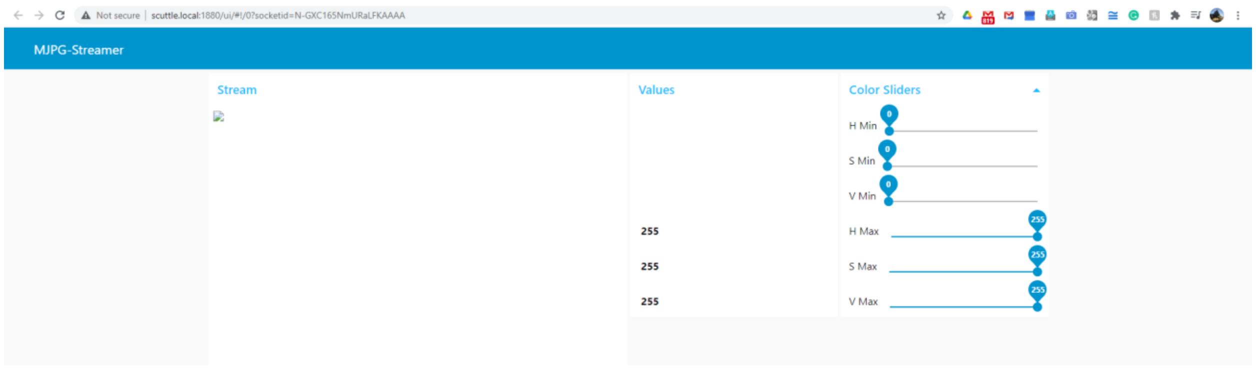
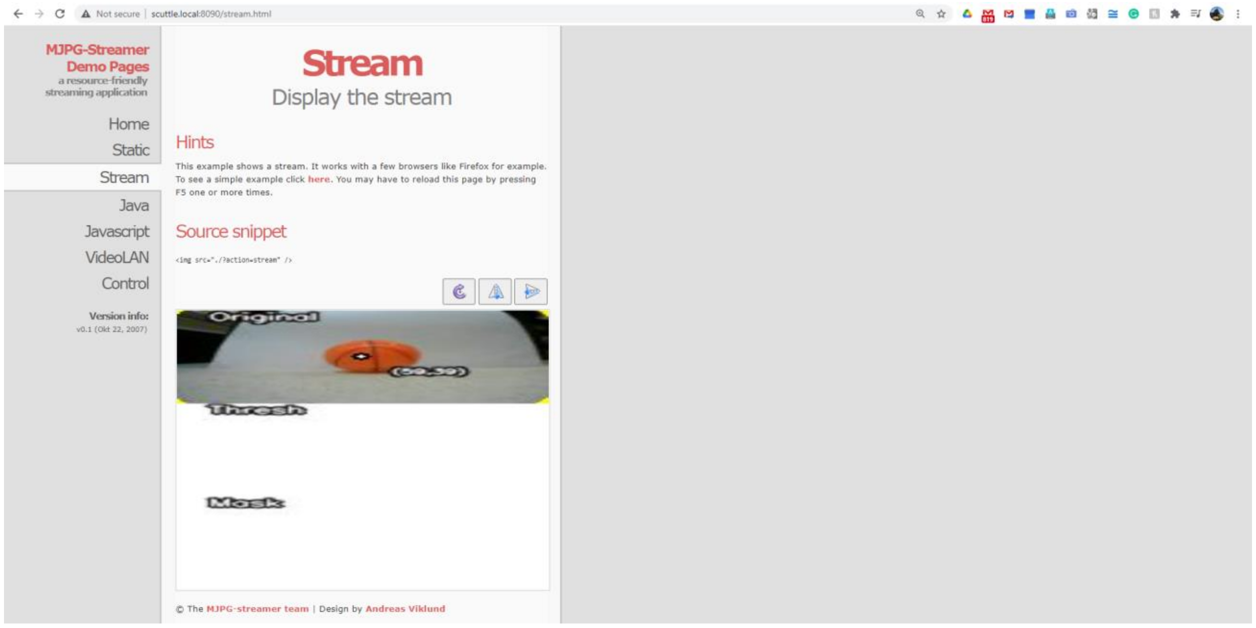


Figure 20

6. Next, open the MJPG Streamer by searching for ‘scuttleXXX.local:8090’ where XXX is your SCUTTLE unit number that you modified in milestone 4 of module 3 in the first section.

   a. This should open another tab in your browser that displays three live streams of the camera you have connected to the SCUTTLE which should look like Figure 21 below.



Stream

Display the stream

Hints

This example shows a stream. It works with a few browsers like Firefox for example. To see a simple example click here. You may have to reload this page by pressing F5 one or more times.

Source snippet

```
<img src="./?action=stream" />
```

© The MJPG-streamer team | Design by Andreas Viklund

Figure 21

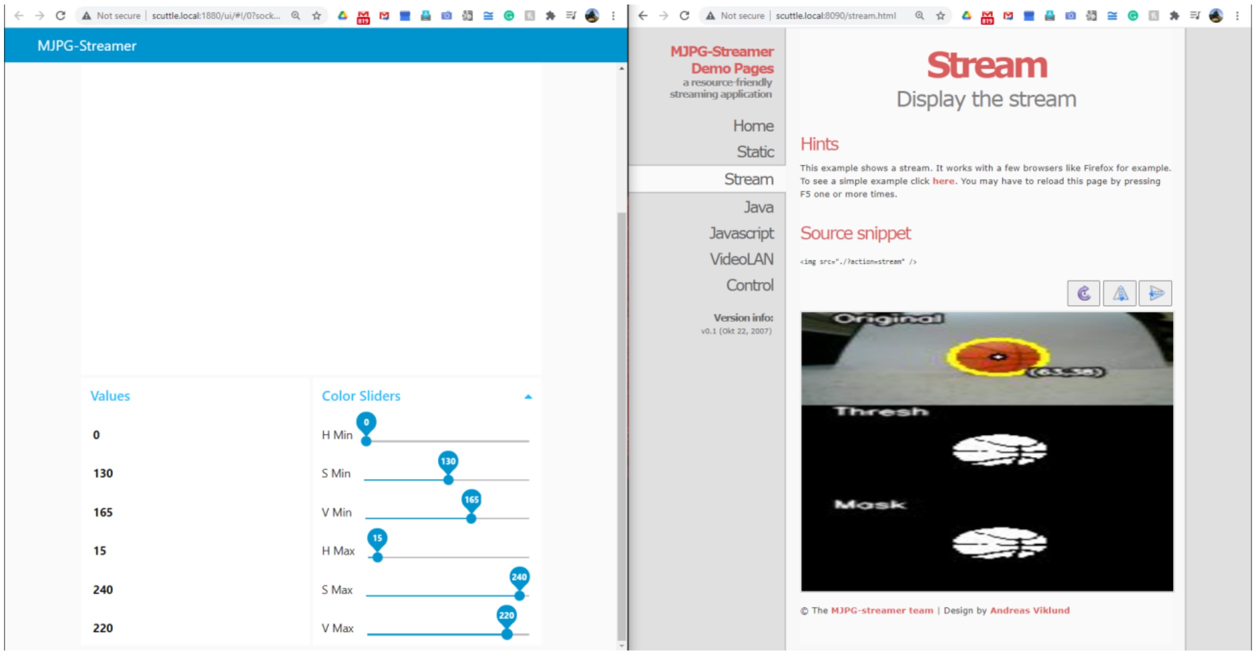
The six sliders in the NodeRed GUI and the MJPG Streamer with the three streams of the camera are what you will manipulate in the next milestone to filter the masked color range.

MILESTONE 2: CALIBRATING THE HSV VALUES

Hue, Saturation and Value (HSV) of an object is the color space associated with the object in OpenCV where Hue represents the color, Saturation represents the greyness and Value represents the brightness. An object's HSV range can change depending on the lighting in your testing area so try to perform this milestone in an area with constant and consistent lighting for the best HSV results. It is also best when the object you track is a different color from the rest of the surroundings. If the object is close in color to another object in the area then the HSV values will need to be very precise to differentiate the two objects in the python scripts. The python scripts work by filtering each pixel that the camera captures and assigning it a 0 or 1 value depending on if that pixel is within the calibrated HSV range. Then on the threshold and mask windows the pixels that were assigned a 0 appear black and the pixels assigned a 1 appear white. The script filters through all the pixels and identifies the area with most connected white pixels and determines that to be the object which it generates radius and coordinate values for.

1. To begin calibrating the HSV values, we will move each slider one by one until we have achieved the desired range for the object we are tracking.
2. Adjust the minimum sliders one by one, starting with hue, then saturation, and finally value.
    a. Start moving the sliders to the right (increasing) as far as they can go without causing the object to turn black in the threshold and mask windows.

3. Then adjust the maximum sliders one by one, starting with hue, then saturation, and finally value.
    a. Start moving the sliders to the left (decreasing) as far as they can go without causing the object to turn black in the threshold and mask windows.
4. When you are done you should have a result similar to Figure 22 below where the object is the only white part of the threshold and mask windows.



Values	Color Sliders
0	H Min
130	S Min
165	V Min
15	H Max
240	S Max
220	V Max

This example shows a stream. It works with a few browsers like Firefox for example. To see a simple example click here. You may have to reload this page by pressing F5 one or more times.

```
<img src="./?action=stream" />
```

© The MJPG-streamer team | Design by Andreas Viklund

Figure 22

    a. In this example, we are finding the threshold for the orange on the ball so that even the lines on the ball are not within the range.
    b. Make sure to note down what the six values are for referencing later on.
5. Try to move the object around, keeping it in the camera's frame of view, and make sure it is still detected.
    a. You should also notice that the program has drawn a yellow circle around the object in the original picture window, as well as given coordinates for the center point of the object.
    b. As you move the object closer and farther from the camera you should notice the circle and coordinate values change with it. The object and circle should get larger when the object is closer to the camera, and the coordinates should increase as the object moves closer and to the right because the origin of the coordinate axis is the top left of the window.

If the program accurately updates the three windows when the object is moved then you are ready to proceed to the next milestone where you will run a program that has the SCUTTLE track the object as it moves around the environment.

MILESTONE 3: TRACK A COLORED OBJECT

Now that you have completed the color filtering, you can use that information to implement a color tracking program so that the SCUTTLE tracks the object you calibrated an HSV color filter for. You should notice that when it is tracking the object, it also determines the distance to the object using only the camera feedback. This showcases how powerful of a sensor the camera alone can be.

You should also notice that as the program is running, the SCUTTLE has a particular method of tracking the object. It first turns until the object is centered and then it moves forwards or backwards accordingly. However, the object will not always be perfectly centered by the SCUTTLE because the use of thresholds which are explained in the Software Guide which can be found in the supplemental folder of the SCUTTLE GitHub.

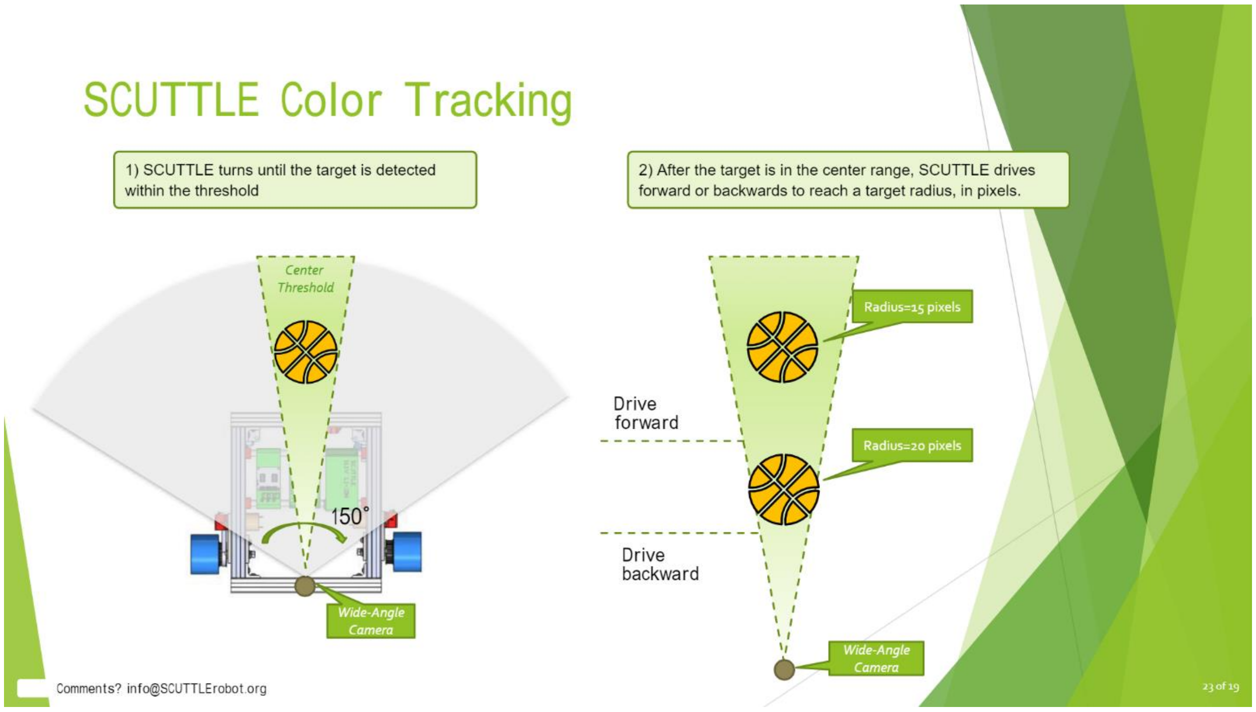

SCUTTLE Color Tracking
1) SCUTTLE turns until the target is detected within the threshold
2) After the target is in the center range, SCUTTLE drives forward or backwards to reach a target radius, in pixels.
Center Threshold
150°
Wide-Angle Camera
Radius=15 pixels
Radius=20 pixels
Drive forward
Drive backward
Wide-Angle Camera
Comments? info@SCUTTLErobot.org
23 of 19

Figure 23

Introducing thresholds like this allow for us to account for small imperfections in the sensors of the SCUTTLE and maintain a high-level of operational success.

1. Start by going to the SCUTTLE GitHub *basics_pi* folder.
2. Then copy the URL to the raw version of the following scripts:
    a. L3_follow.py
    b. L2_track_target.py
    c. L1_camera.py
    d. L2_speed_control.py
    e. L1_motor.py

   f. L2_inverse_kinematics.py
   g. L1_gamepad.py
3. Then in the Cloud9 terminal, use the *wget* command and the raw URL you found to copy the scripts to the *Module1-Camera* folder. Alternatively, you could also copy and paste them from the *basics_pi* folder already in your Cloud9 to the *Module1-Camera* folder.
   a. Verify that they all showed up in the file directory workspace in the *Module1-Camera* folder.
4. In Cloud9, double click on the *L2_track_target.py* script to open it for editing.
   a. In line 14 of the script, you will notice it has values for the HSV minimums and maximums.
   b. Enter the six calibrated HSV values you noted down in milestone 2 in the correct order such as ((h min, s min, v min), (h max, s max, v max)).
      i. Use *ctrl + s* to save the changes you made to the script.
5. Then in the Cloud9 terminal execute the following command to start the program:

```
sudo python3 L3_follow.py
```

   a. Now move the object around in the field of view of the camera, you should notice the SCUTTLE moves towards the object.

Congratulations, you have completed the Camera module! Hopefully now you have a better idea of a few ways that we can use the camera as a sensor on a mobile robot such as the SCUTTLE. It adds many possibilities for functions and tasks that we can assign the SCUTTLE. It is also neat how we can actively monitor what the camera is capturing to see what information the SCUTTLE is processing as it is executing a program.

MODULE 2: ENCODERS

BACKGROUND

The encoders are sensors that monitor the axle position to provide us information that we can convert to wheel speeds and subsequently into distance and angle data. This opens the door for us to be able to navigate the SCUTTLE around an area autonomously using calculations that determine where the SCUTTLE has moved in the local and global frames. There are two encoders on the SCUTTLE, one for each wheel. They connect to an I2C bus and then to the Raspberry Pi via Dupont connector cables.

MILESTONE 1: TESTING I2C CONNECTIONS

The assembly guides should have shown you how to connect the encoder wires to the I2C bus, but we will double check that the connections are correct. *Remember to make wiring changes before you turn on the SCUTTLE.*

1. Figure 24 below shows the way the left and right encoder cables should be wired and connected.

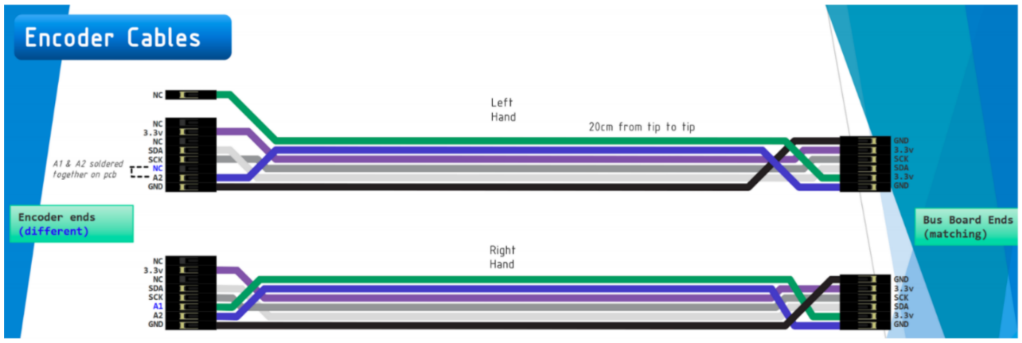


Figure 24

2. Take special note of the orientation of the wires for both the left and right as they are unique to their respective sides.
3. Figure 25 shows another view of how the wires should connect from the left and right encoders to the I2C bus, and from the I2C bus to the Raspberry Pi.

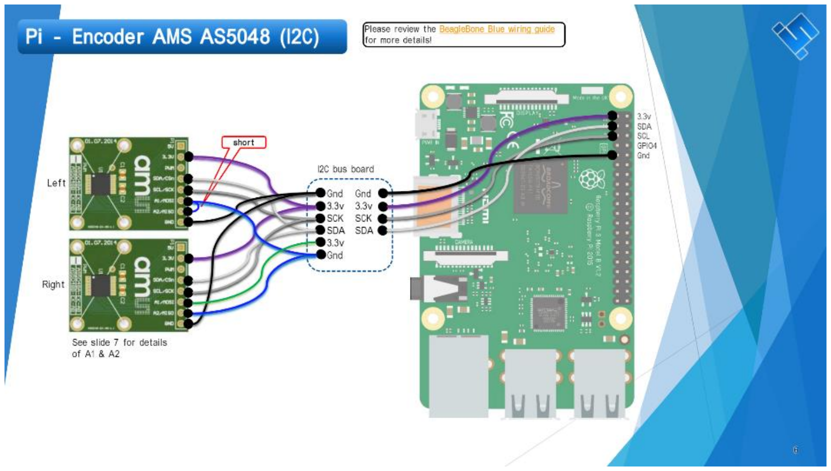

Pi - Encoder AMS AS5048 (I2C)
Please review the BeagleBone Blue wiring guide for more details!
short
Left
Right
I2C bus board
Gnd
3.3v
SCK
SDA
3.3v
Gnd
Gnd
3.3v
SCK
SDA
3.3v
SDA
SCL
GPIO4
Gnd
See slide 7 for details of A1 & A2
6

Figure 25

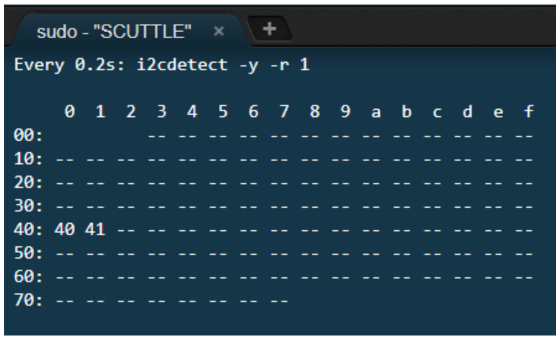
4. Verify again that your wiring is done as shown in Figures 24 and 25 before powering on the SCUTTLE.
5. Then power on the SCUTTLE and open up Cloud9 by searching for 'scuttleXXX.local:8080' where XXX is your SCUTTLE unit number that you modified in milestone 4 of module 3 in the first section, then the Cloud9 IDE should load.
6. In the Cloud9 terminal, use the following command to install the Adafruit GPIO needed for communicating with the I2C bus:

```
sudo pip3 install Adafruit_GPIO
```

   a. It may return that the GPIO is already installed which it should be if you flashed the correct image.
7. Then use the following command which will be used to determine if the Raspberry Pi is receiving the proper signals from the correct pins of the I2C board.

```
sudo watch -n0.2 i2cdetect -y -r 1
```

   a. This should print out a column matrix of information in the terminal that displays several columns and rows.



```
Every 0.2s: i2cdetect -y -r 1

     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 41 -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 26

   b. In the *40* row, you should see a 40 and 41.

8. To verify that the Raspberry Pi is detecting the correct I2C address from both encoders, unplug the left encoder.
   a. You should notice that the address 40 is no longer in the matrix.
   b. The same should occur for the 41 in the matrix when you unplug the right encoder.
   c. If these actions do not happen as stated, this most likely means you have the left and right encoder connections reversed.

MILESTONE 2: VERIFYING DIRECTIONS

Now that you have verified the Pi detects the correct I2C address from each encoder you can test and observe whether the encoder values are representative of wheel movements. By spinning the wheels on the SCUTTLE, we can verify that the readings the Raspberry Pi receives from the encoders are interpreted correctly. The change in encoder values should be reversed when comparing the right and left wheels. Meaning, turning the right wheel forward should result in a higher value and turning the right wheel backwards should result in a lower value. Then with the left wheel, turning it forward should result in lower values and turning it backwards should result in higher values. We can also use programs which perform forward kinematics to translate the encoder readings into chassis movement.

1. In Cloud9, start by creating a *Module2-Encoder* folder in your home directory using the following:

```
mkdir Module2-Encoder
```

2. Then go to the SCUTTLE GitHub *basics_pi* to find the file you want to copy to the new *Module2-Encoder* folder you just created. Again, copying and pasting the file that is already in the SCUTTLE folder in Cloud9 to the new *Module2-Encoder* folder is also acceptable.
   a. View the raw code for the L1_encoder.py file and use the *wget* command in the Cloud9 terminal to copy it to your *Module2-Encoder* folder.
   b. Check the Cloud9 file directory to verify that file now appears in the *Module2-Encoder* folder.
   c. If the file appears as it should, then use the following command to execute it:

```
 python3 L1_encoder.py
```

3. With the SCUTTLE propped up on its' stand, manually turn the wheels and verify that the encoder values that the program is printing in the terminal correlate with the table below based on the motion you cause.

Table 1

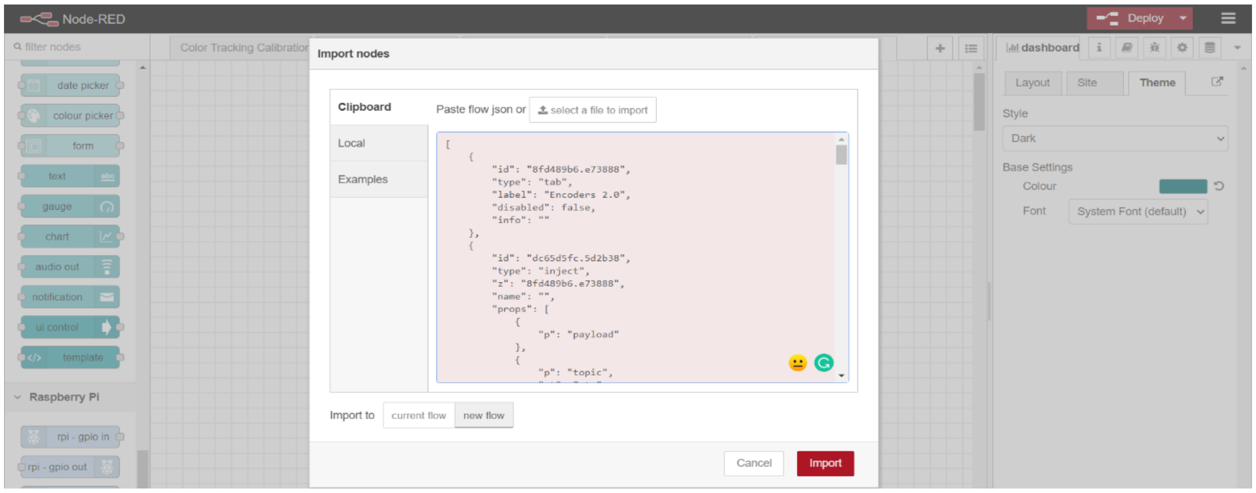
	Left Wheel	Right Wheel
Address	40	41
Turn Wheel FORWARD	Encoder Values DECREASE	Encoder Values INCREASE
Turn Wheel BACKWARD	Encoder Values INCREASE	Encoder Values DECREASE

The L1 program *L1_encoder.py* only returns static readings. To convert the data and send it to NodeRed to display as a GUI, you will need to use an L2 program.

4. Go to the SCUTTLE GitHub *basics_pi* to find the *L2_log_encoders.py* file that you want to copy to the *Module2-Encoder* folder you created. Again, copying and pasting the file that is already in the SCUTTLE folder in Cloud9 to the *Module2-Encoder* folder is also acceptable.
   a. View the raw code for the L2_log_encoders.py file and use the *wget* command in the Cloud9 terminal to copy it to your *Module2-Encoder* folder.
   b. You will also need the L1_log.py file so use the *wget* command in the Cloud9 terminal to copy it to your *Module2-Encoder* folder as well.
   c. Check the Cloud9 file directory to verify that the files now appear in the *Module2-Encoder* folder.
   d. If the files appear as they should, then use the following command to execute the L2 script:

```
python3 L2_log_encoders.py
```

5. Now go to SCUTTLE GitHub NodeRed encoders flow and copy the entire script.
   a. Then open NodeRed by going to your browser and searching for 'scuttleXXX.local:1880' where XXX is your SCUTTLE unit number that you modified in milestone 4 of module 3 in the first section, then NodeRed should load.
   b. Open the menu by clicking on the button in the top right of the NodeRed environment.
      i. In the menu that appears, navigate to import>clipboard and paste the entire script you copied from the NodeRed color tracking folder of the GitHub.
      ii. Then select import>new flow to import the flow without overwriting the current flow.



```
[
    {
        "id": "8fd489b6.e73888",
        "type": "tab",
        "label": "Encoders 2.0",
        "disabled": false,
        "info": ""
    },
    {
        "id": "dc65d5fc.5d2b38",
        "type": "inject",
        "z": "8fd489b6.e73888",
        "name": "",
        "props": [
            {
                "p": "payload"
            },
            {
                "p": "topic",
```

Figure 27

c. It should create a flow that looks like Figure 28 below:

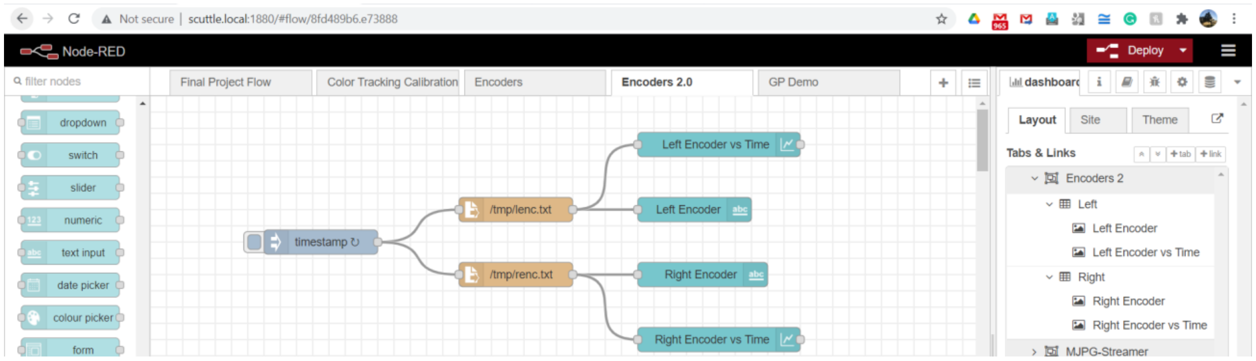
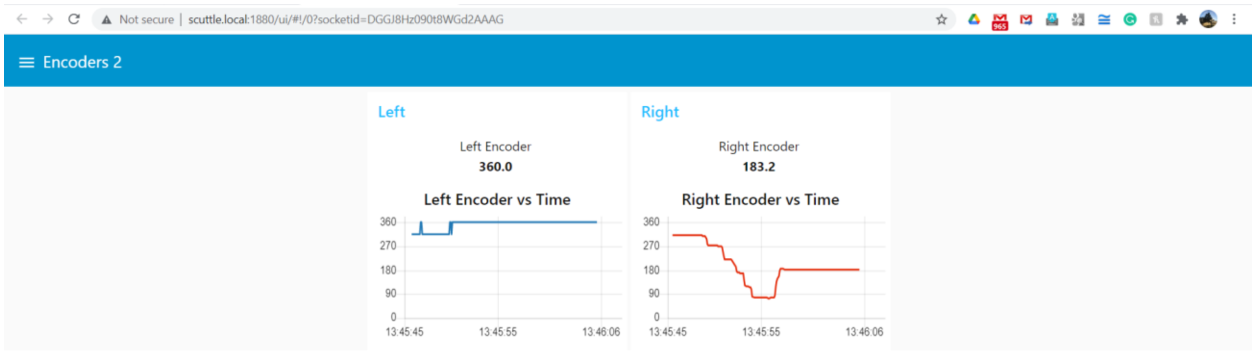


Figure 28

d. Now click the "Deploy" button in the upper right of NodeRed and then click the open dashboard button. It should open a new tab in your browser with two values which are each displayed on a respective chart with respect to time like Figure 29 below.



Left Encoder
360.0

Right Encoder
183.2

Figure 29

6. Then spin the wheels or move the SCUTTLE around the floor and take note of how the values change.

Congratulations, you have completed the Encoder module! You should now have a better idea of how to use encoders in mobile robotic systems and understand when it is good to implement them.

MODULE 3: GAMEPAD

BACKGROUND

The gamepad can be a useful tool to manually control several premade functions at the push of a button or move of a joystick. We will first test the operation of the gamepad by using the joystick to control the movement of the SCUTTLE. Then we will display readings on the changes in x-dot and theta-dot using NodeRed as the SCUTTLE is driven around. X-dot and theta-dot represent the forward and angular velocities of the SCUTTLE, respectively. The level two programs used in this example convert the x-dot and theta-dot values of the SCUTTLE chassis into individual wheel speeds using inverse kinematics.

MILESTONE 1: TESTING FUNCTIONS

The gamepad we will use with the SCUTTLE is the EasySMX controller pictured below in Figure 30.

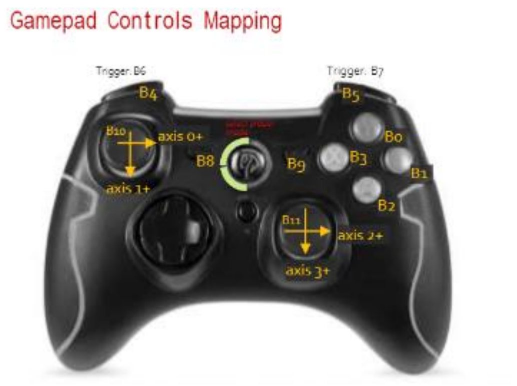


Figure 30

For the first milestone we will be driving the SCUTTLE which is done using the left joystick. The forward/backward axis of the joystick controls the speed the SCUTTLE moves forwards or backwards (X-dot), respectively. The left/right axis controls the speed which the SCUTTLE rotates left and right (Theta-dot), respectively. The SCUTTLE [Software Guide]() on GitHub provides a more specific explanation about the process involved in converting input from the joystick on the gamepad into output from the motors causing the SCUTTLE to move.

1. Start by connecting the gamepads' accompanying USB dongle to an open USB port on the Raspberry Pi.
2. Then open up Cloud9 by searching for 'scuttleXXX.local:8080' where XXX is your SCUTTLE unit number that you modified in milestone 4 of module 3 in the first section, then the Cloud9 IDE should load.
    a. Use the *lsusb* command to verify the device is connected.
3. Create a *Module3-Gamepad* folder in your home directory using the following:

```
mkdir Module3-Gamepad
```

4. Then go to the SCUTTLE GitHub *basics_pi* to find the files you want to copy to the new *Module3-Gamepad* folder you just created. Again, copying and pasting the files that are already in the SCUTTLE folder in Cloud9 to the new *Module3-Gamepad* folder is also acceptable.
   d. View the raw code for the following files and use the *wget* command in the Cloud9 terminal to copy them to your *Module3-Gamepad* folder.
      i. L3_gpDemo.py
      ii. L2_inverse_kinematics.py
      iii. L2_kinematics.py
      iv. L2_speed_control.py
      v. L1_log.py
      vi. L1_gamepad.py
      vii. L1_encoder.py
      viii. L1_motor.py
   e. Check the Cloud9 file directory to verify that the files now appear in the *Module3-Gamepad* folder.
   f. Then turn on the gamepad by pressing the middle button.
      i. Make sure the two blue LEDs on the Left half of the button are lit up. This means the controller is in the proper mode to work with the SCUTTLE.
      ii. If it is not in the correct mode, hold the middle button for six seconds and the mode will change. Do this until it is in the correct mode (left half of button lit up in Blue).
   g. If the files all appear as they should, use the following command to execute the main program:

```
python3 L3_gpDemo.py
```

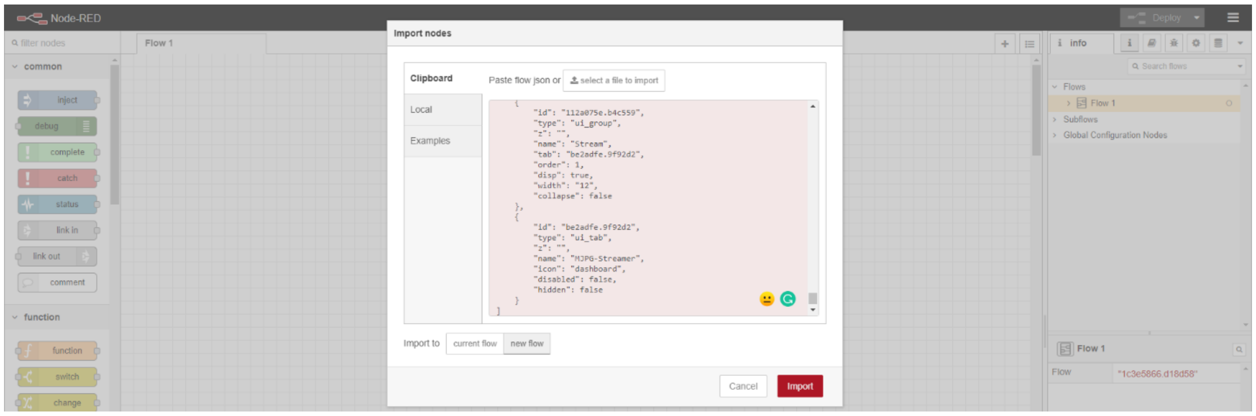
5. You should now be able to control the movement of the SCUTTLE with the left joystick on the gamepad.
   a. Try moving forward, backward, and spinning to the left and right.
   b. As you move the joystick further from its resting position, you should notice the movements become faster.

Congratulations, now you have connected the gamepad and used it to control the SCUTTLE. In the next milestone you will use a NodeRed GUI to view the change in values as you operate the SCUTTLE.

MILESTONE 2: VERIFYING DIRECTIONS

We can use NodeRed to visually display the live changes in values of x-dot and theta-dot as the SCUTTLE moves around an environment by showing the change in input from the gamepad itself such as joystick movement and button presses. This will show the change of the values between their respective maximum and minimum values.

1. Start by going to the NodeRed folder in GitHub and copy the entire script.
    a. Then open NodeRed by going to your browser and searching for 'scuttleXXX.local:1880' where XXX is your SCUTTLE unit number that you modified in milestone 4 of module 3 in the first section, then NodeRed should load.
    b. Open the menu by clicking on the button in the top right of the NodeRed environment.
        i. In the menu that appears, navigate to import>clipboard and paste the entire script you copied from the NodeRed color tracking folder of the GitHub.
        ii. Then select import>new flow to import the flow without overwriting the current flow.



```
    {
        "id": "112a075e.b4c559",
        "type": "ui_group",
        "z": "",
        "name": "Stream",
        "tab": "be2adfe.9f92d2",
        "order": 1,
        "disp": true,
        "width": "12",
        "collapse": false
    },
    {
        "id": "be2adfe.9f92d2",
        "type": "ui_tab",
        "z": "",
        "name": "MJPG-Streamer",
        "icon": "dashboard",
        "disabled": false,
        "hidden": false
    }
]
```

Figure 31

    c. It should create a flow that looks like Figure 32 below:

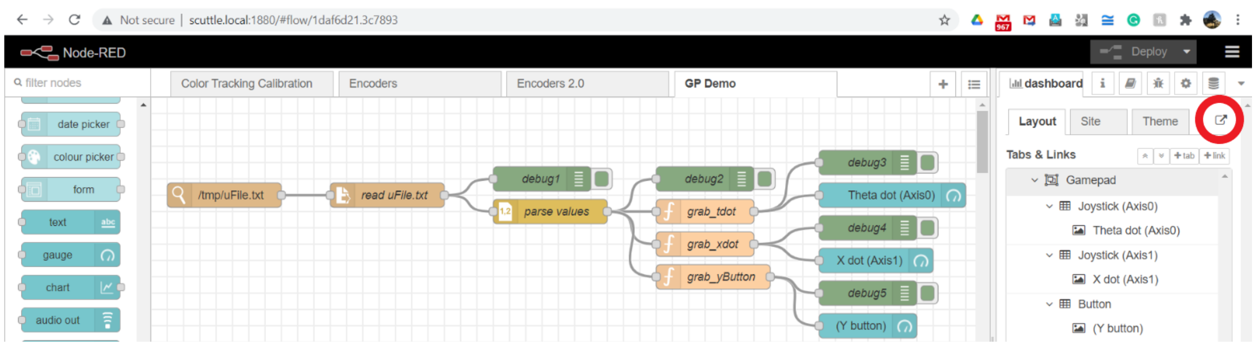


Figure 32

    d. Now click the "Deploy" button in the upper right of NodeRed and then click the button that is circled in red within Figure 32, and it should open a new tab in your browser with three gauges like Figure 33 below.

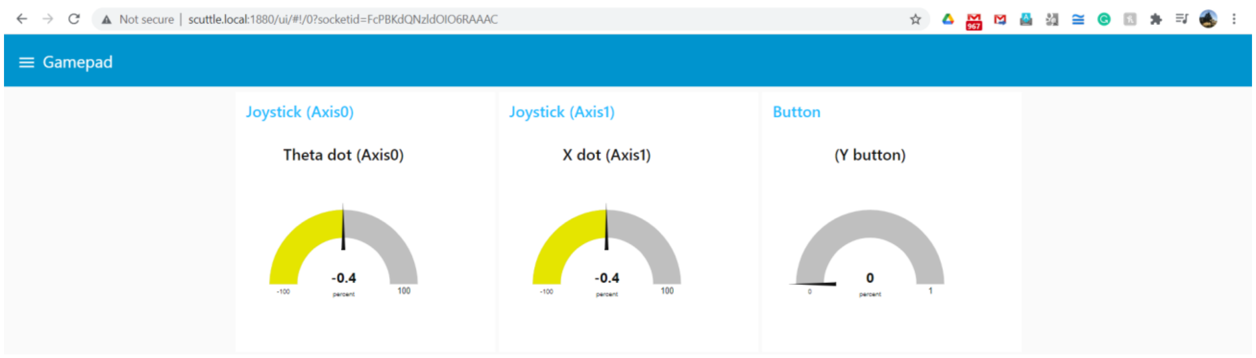


Figure 33

2. With the *L3_gpDemo.py* code running in Cloud9, you should now see changes in the GUI while you move the joystick or press the Y button.
   a. Move the joystick left and right as far as it can go and notice how that is reflected in the range of the GUI.
      i. Axis0 is representative of moving the left joystick left and right.
      ii. Axis1 is representative of moving the left joystick up and down.
   b. Notice how the x-dot and theta-dot values change as the robot is moving faster or slower.
   c. Also notice if the button indicator changes as you press the Y button.

Congratulations, you have completed the gamepad module. Hopefully now you have a better understanding of the way that specific buttons and joysticks can be programmed to control a mobile robot such as the SCUTTLE. You also have gained experience learning about important concepts such as linear and angular velocity. Now you are ready to move to the next module to learn about servo motors.

MODULE 4: SERVO

BACKGROUND

A servo is a small motor that we can add to the SCUTTLE to create more degrees of freedom for the overall system. A servo motor is not the same as the DC motors driving the SCUTTLE wheels. Servos are made to move to precise positions rather than perform repetitive complete rotations. We can control the servo by sending Pulse-Width Modulation (PWM) signals to the signal line which determines the position of the motor. It works as a closed loop control system where the feedback is used to adjust the speed and position to meet the desired result. There are many types of servos, but they all tend to have a common set of connections: positive, ground, and control.

MILESTONE 1: MAKE CONNECTIONS

The assembly guides should have shown you how to connect the servo motor to the Raspberry Pi.

1. You can reference the SCUTTLE Wiring Guide Pi for this specific connection.
    a. Ensure that the wired connections are made the same way as they are shown in the guide.
    b. Verify again that your wiring is done as shown in the diagram before powering on the SCUTTLE.
2. Then power on the SCUTTLE and open up Cloud9 by searching for ‘scuttleXXX.local:8080’ where XXX is your SCUTTLE unit number that you modified in milestone 4 of module 3 in the first section, then the Cloud9 IDE should load.
3. Create a *Module4-Servo* folder in your home directory using the following:

```
mkdir Module4-Servo
```

4. Then go to the SCUTTLE GitHub *basics_pi* to find the file you want to copy to the new *Module4-Servo* folder you just created. Again, copying and pasting the files that are already in the SCUTTLE folder in Cloud9 to the new *Module4-Servo* folder is also acceptable.
    a. View the raw code for the following file and use the *wget* command in the Cloud9 terminal to copy it to your *Module4-Servo* folder.
        i. L1_servo.py
    b. Check the Cloud9 file directory to verify that the files now appear in the *Module4-Servo* folder.

If you made the connections as indicated in the wiring guide and the correct file appears in the folder you created then you are ready to move to the next milestone where you will control the movement of the servo motor.

MILESTONE 2: VERIFY ANGLE AND DIRECTION

The file you copied in the last milestone will turn the servo horn to its minimum, middle, and maximum positions with a small pause in between the movements. Most servos are 180°, meaning that each time the servo horn moves from minimum to middle or middle to maximum position it will rotate about 90°. These movements are controlled by a series of pulses in a process known as pulse-width modulation (PWM). The length of the pulse it what ultimately determines the position of the servo horn.

1. Navigate to the *Module4-Servo* folder and use the following command to execute the main program:

```
python3 L1_servo.py
```

2. Analyze how the servo horn is moving when you run the code.
    a. The servo horn should move clockwise from its minimum point to middle point and maximum point with small pauses in-between the movements.
3. Modify the code so that the order of positions for the servo horn are flipped and run the code again.
    a. Again, observe the movements of the servo horn.
    b. Changing the order of the movements should have caused the servo horn to do the same set of motions but counter-clockwise in the reverse order.

Congratulations, now you have successfully connected the servo to the Raspberry Pi and controlled its motion. Keep in mind that this is a simple example of a servo motor in use, and it can be programmed to be used in many applications where small precise rotations are needed in a system.

MODULE 5: VOLTAGE METER

BACKGROUND

The voltage meter we will use for this example is an Adafruit INA219. It monitors the voltage of the battery as well as the current to within one percent accuracy. Some microcontrollers have a built in ADC which can perform voltage and current readings but since the Raspberry Pi does not, we can use voltage meters to perform the same functions. A voltage meter can be a useful indicator to include in many SCUTTLE projects because it is important to know whether the batteries are fully charged or need to be recharged. We want the batteries close to a full charge of 12V to ensure peak performance from SCUTTLE. When the voltage falls to around 9V the performance begins to drop off and your SCUTTLE should be charged.

MILESTONE 1: VALIDATING THE CONNECTION

The assembly guides should have shown you how to connect the INA219 to the I2C bus.

1. You can reference the SCUTTLE Wiring Guide Pi for this specific connection.
    a. Ensure that the wired connections are made the same way as they are shown in the diagram for the "Voltage Meter – Adafruit INA219".
    b. Verify again that your wiring is done as shown in the diagram before powering on the SCUTTLE.
2. Then power on the SCUTTLE and open up Cloud9 by searching for 'scuttleXXX.local:8080' where XXX is your SCUTTLE unit number that you modified in milestone 4 of module 3 in the first section, then the Cloud9 IDE should load.
    a. The Adafruit GPIO should already be installed either from flashing the correct image or from the encoder module.
3. Then use the following command which will be used to determine if the Raspberry Pi is receiving the proper signals from the correct pins of the I2C board.

```
sudo watch -n0.2 i2cdetect -y -r 1
```

    a. This should print out a column matrix of information in the terminal that displays several columns and rows.
    b. There should now be three addresses that it prints.
        i. 40 and 41 for the left and right encoders.
        ii. A third address for the INA219.
4. To verify that the Raspberry Pi is detecting the correct I2C address from the INA219, unplug it.

   a. You should notice that the third address where it was is no longer in the matrix.

If you made the correct connections to the INA219 as indicated in the wiring guide and performed the *i2cdetect* testing to ensure that the I2C was receiving a signal from the INA219 address then you have verified a successful connection. You are now ready to move to the next milestone to view the readings from the sensor.

MILESTONE 2: MEASURING THE VOLTAGE

We can view the voltage readings of the INA219 as text in the Cloud9 terminal to know if it has a full charge or needs to be charged. To measure noticeable changes in the voltage we can take readings when the SCUTTLE is powered by the battery pack and when it is powered by the computer.

1. We will begin with viewing the voltage readings for a fully powered SCUTTLE so connect the USB from your computer plus the battery pack to the Raspberry Pi and turn it on.
2. In your browser, open up Cloud9 by searching for 'scuttleXXX.local:8080' where XXX is your SCUTTLE unit number that you modified in milestone 4 of module 3 in the first section, then the Cloud9 IDE should load.
3. Now in the cloud9 terminal, use the following command to create a folder called *Module5-VoltMeter*.

```
mkdir Module5-VoltMeter
```

4. Then go to the SCUTTLE GitHub *basics_pi* to find the files you want to copy to the new *Module5-VoltMeter* folder you just created. Again, copying and pasting the files that are already in the SCUTTLE folder in Cloud9 to the new *Module5-VoltMeter* folder is also acceptable.
   a. View the raw code for the following file and use the *wget* command in the Cloud9 terminal to copy them to your *Module5-VoltMeter* folder.
      i. L1_ina.py
   b. Check the Cloud9 file directory to verify that the file now appears in the *Module5-VoltMeter* folder.
5. Then run the code by using the following command in the terminal.

```
python3 L1_ina.py
```

   a. This should print out voltage readings.
      i. The readings should be between 0 and 12V. Hopefully they are closer to 12V, or you really need to charge the battery pack.
6. Then turn off the battery pack, so that the only power the Raspberry Pi receives is from the USB connection to your computer.
7. Run the *L1_ina.py* script again to print out the new voltage readings.

    a. Monitor the readings and notice how much lower the readings are this time.

8. You can also use this method to test the individual batteries of the battery pack in case one has gone bad and does not reach a full charge as the others do. You would know to test for this if your battery pack could not get above 8V or 9V when charging, meaning that two of the 4V batteries are working properly and one is not.

Congratulations, you have finished the last module of the second section. Hopefully now you have a much better understanding of common sensors and how they can interface with mobile robotic systems. These examples should serve as a solid foundation that you can build your experience with robotics upon.