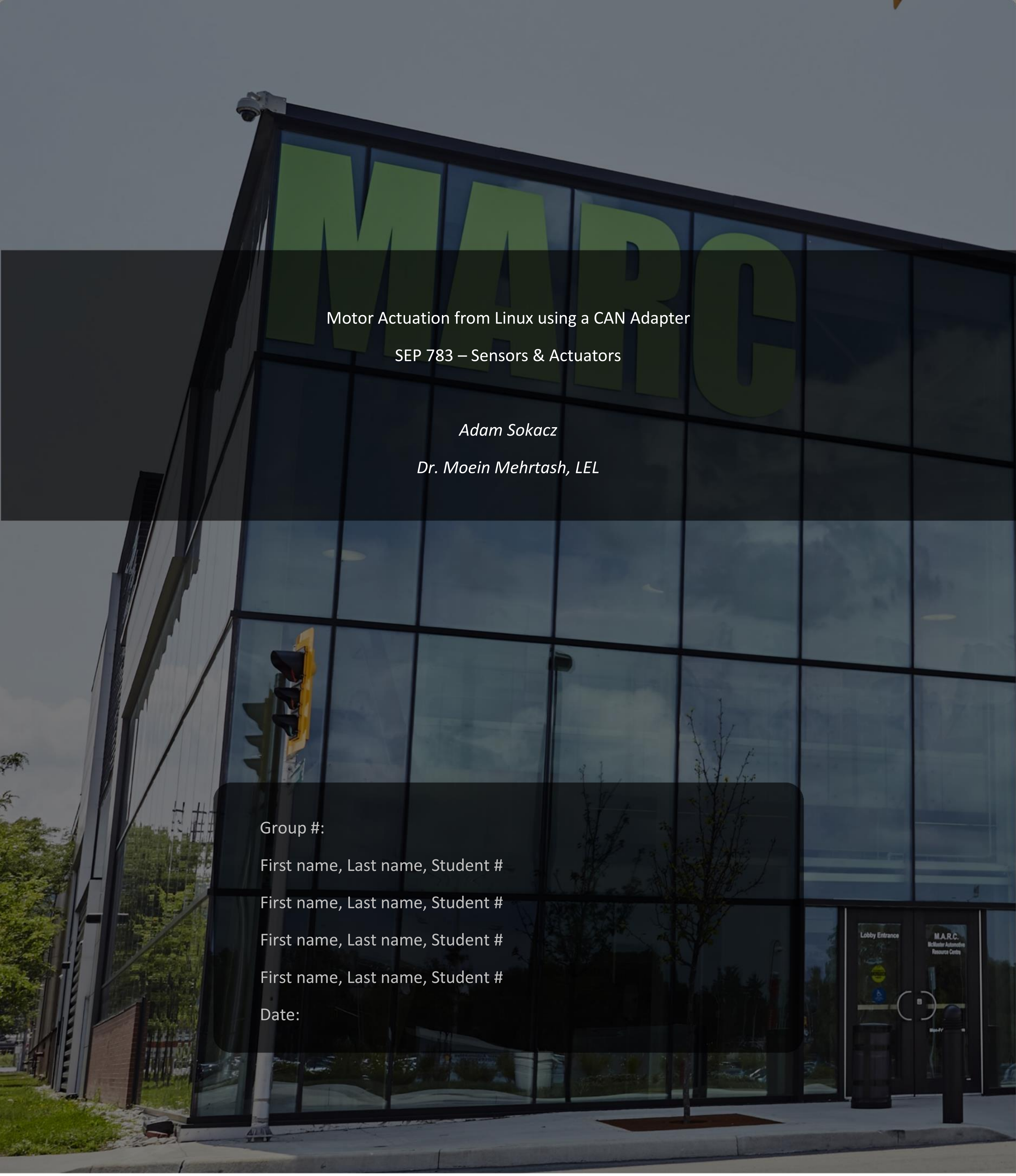


Motor Actuation from Linux using a CAN Adapter

SEP 783 – Sensors & Actuators

Adam Sokacz

Dr. Moein Mehrtash, LEL

Group #:

First name, Last name, Student #

First name, Last name, Student #

First name, Last name, Student #

First name, Last name, Student #

Date:

W BOOTH SCHOOL OF ENGINEERING
PRACTICE AND TECHNOLOGY
McMaster-Mohawk Bachelor of Technology Partnership

McMaster
University
ENGINEERING

mohawk

Objective

Learn about setting up a CAN interface and communicating to devices on a CAN bus using Linux.

Table of Contents

Pre-Lab Questions

Post-Lab Questions

Q1- What is *CAN*? Briefly describe how it works.

(Suggested: Short Paragraph)

Q2 - What is *Git*? How can Git be used to *clone* remote repositories?

(Suggested: Short Paragraph)

Q3 - What is *differential drive* in the context of robotics? Briefly describe how it functions in controlling direction and speed.

(Suggested: Short Paragraph)

Q4 - Which *2 files* must be *sourced* in order for ROS to be able to access the *version* you want to use and the *workspace* you want to use.

(Suggested: 2 sentences)

Q5 - The MacBot uses a *Jetson Nano* to control itself. Compare and contrast a Jetson Nano to a Raspberry Pi 4 B+.

(Suggested: Table)

Q6 - Define the word *odometry* in the context of robotics.

(Suggested: Short Paragraph)

Q7 - In your own words, compare and contrast *global* vs *local* path planning algorithms.

(Suggested: Short Paragraph)

Q8 - Generate an *RQT graph* for the MacBot performing SLAM. Take a screenshot and include it with your report.

(Suggested: Screenshot)

Q9 - List 3 *industry applications* of *SLAM*. What advantages does SLAM have over GPS?

(Suggested: Short Paragraph)

Q11 – Write a brief LinkedIn post about key concepts that were learned in this lab.

(Suggested: Short Paragraph)

Optional Assignment

Follow the following tutorial to install, build, and run TurtleSim. Take a screenshot to demonstrate that you can control it using your keyboard.

WiKi: http://wiki.ros.org/turtlesim

Feedback

Q1 - What would you rate the difficulty of this lab?

(1 = easy, 5 = difficult)

1 2 3 4 5

Comments about the difficulty of the lab:

Q2 - Did you have enough time to complete the lab within the designated lab time?

YES NO

Q3 - How easy were the lab instructions to understand?

(1 = easy, 5 = unclear)

1 2 3 4 5

List any unclear steps:

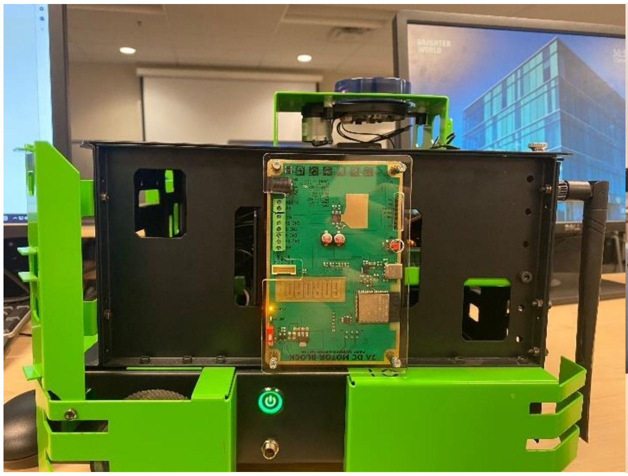
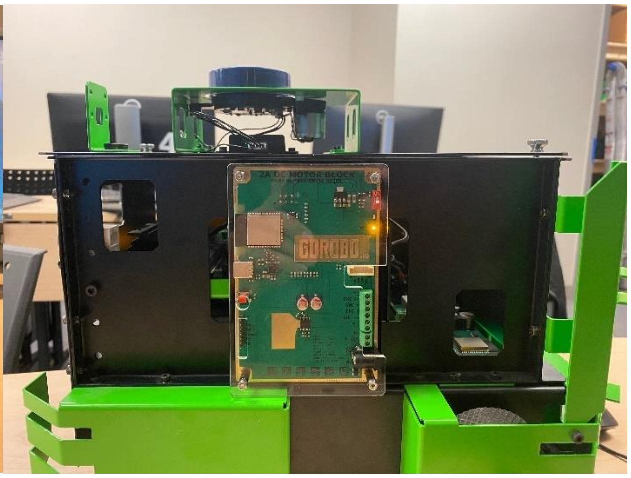
Q4 - Could you see yourself using the skills learned in this lab to tackle future engineering challenges?

(1 = no, 5 = yes)

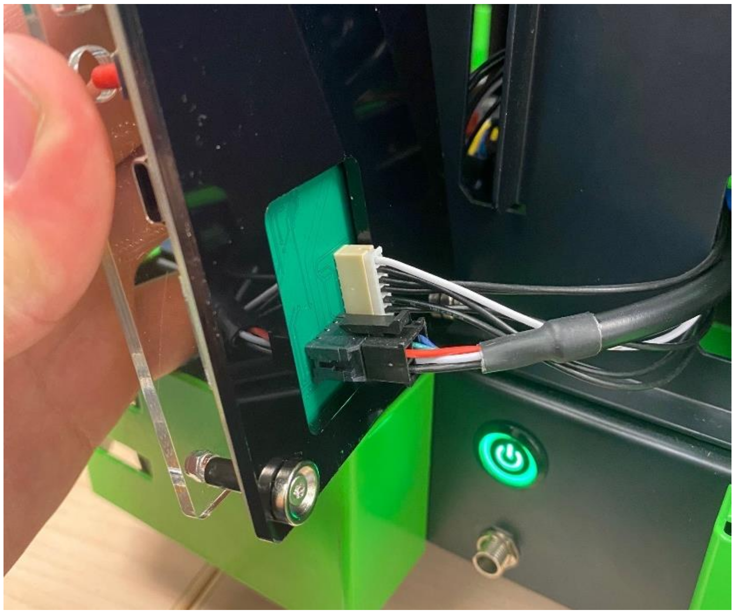
1 2 3 4 5

Motor Driver Setup

For this lab, 2 motor controller modules will be needed. One motor driver unit will be magnetically attached to the left-side of the MacBot and the other will be attached to the right-side of the MacBot.

Each of these motor controller modules requires 2 cables to be connected to the rear end of each module. The 6-pin connector is the encoder cable and carries rotor orientation data to the embedded ESP32 on the motor controller unit. The second connecter is power from the power distribution board inside the MacBot.



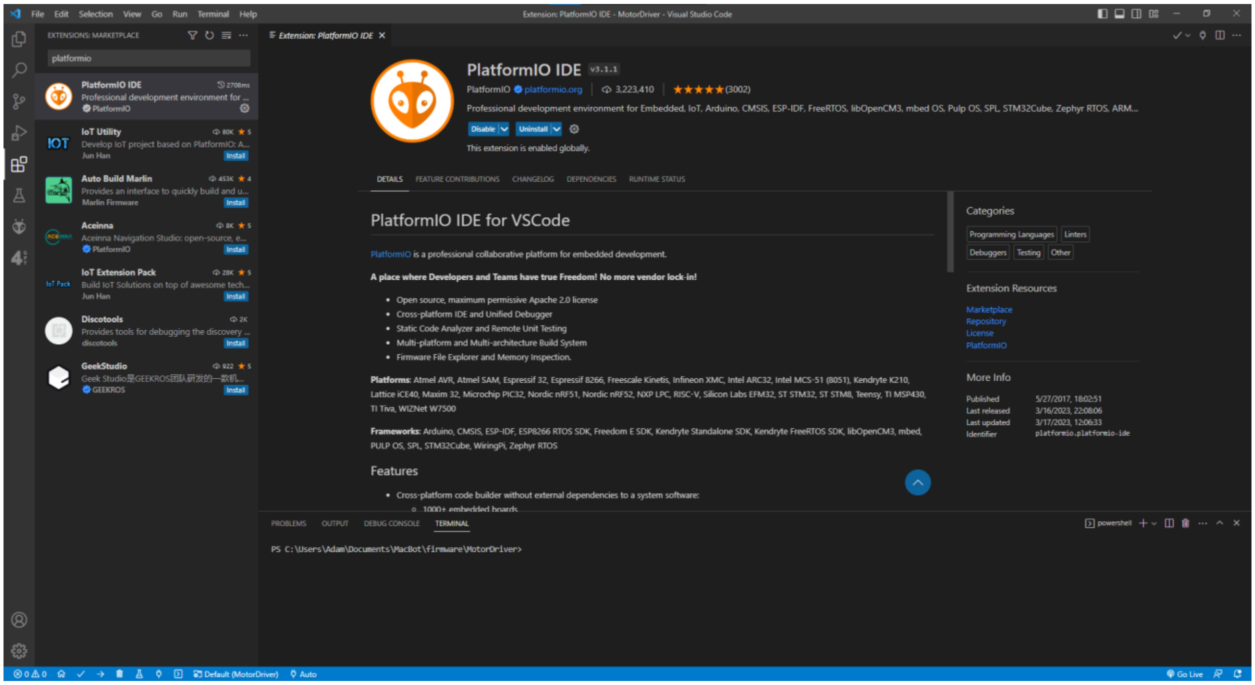
When the MacBot is powered OFF, motor controller firmware can be loaded onto the ESP32 on each motor controller module using a USB-C cable.

The firmware is located in the **firmware/MotorDriver** directory in the project repository.

https://github.com/adamsokacz/macbot/tree/main/firmware/MotorDriver

The development environment used to build and upload the motor driver is called PlatformIO. PlatformIO, abbreviated as PIO, is a cross-platform multi-framework IDE for VSCode that can be installed through the Extension Manager tab.

https://platformio.org/



PlatformIO IDE 2708ms
Professional development environment for ...
PlatformIO

IoT Utility 80K ★ 5
Develop IoT project based on PlatformIO: A...
Jun Han

Auto Build Marlin 453K ★ 4
Provides an interface to quickly build and u...
Marlin Firmware

Aceinna 8K ★ 5
Aceinna Navigation Studio: open-source, e...
PlatformIO

IoT Extension Pack 28K ★ 5
Build IoT Solutions on top of awesome tech...
Jun Han

Discotools 2K
Provides tools for debugging the discovery ...
discotools

GeekStudio 922 ★ 5
Geek Studio是GEEKROS团队研发的一款机...
GEEKROS

PlatformIO IDE v3.1.1
PlatformIO platformio.org | 3,223,410 | ★★★★★(3002)
Professional development environment for Embedded, IoT, Arduino, CMSIS, ESP-IDF, FreeRTOS, libOpenCM3, mbed OS, Pulp OS, SPL, STM32Cube, Zephyr RTOS, ARM...
This extension is enabled globally.

PlatformIO IDE for VSCode

PlatformIO is a professional collaborative platform for embedded development.

A place where Developers and Teams have true Freedom! No more vendor lock-in!

- Open source, maximum permissive Apache 2.0 license
- Cross-platform IDE and Unified Debugger
- Static Code Analyzer and Remote Unit Testing
- Multi-platform and Multi-architecture Build System
- Firmware File Explorer and Memory Inspection.

Platforms: Atmel AVR, Atmel SAM, Espressif 32, Espressif 8266, Freescale Kinetis, Infineon XMC, Intel ARC32, Intel MCS-51 (8051), Kendryte K210, Lattice iCE40, Maxim 32, Microchip PIC32, Nordic nRF51, Nordic nRF52, NXP LPC, RISC-V, Silicon Labs EFM32, ST STM32, ST STM8, Teensy, TI MSP430, TI Tiva, WIZNet W7500

Frameworks: Arduino, CMSIS, ESP-IDF, ESP8266 RTOS SDK, Freedom E SDK, Kendryte Standalone SDK, Kendryte FreeRTOS SDK, libOpenCM3, mbed, PULP OS, SPL, STM32Cube, WiringPi, Zephyr RTOS

Features

- Cross-platform code builder without external dependencies to a system software:
  - 1000+ embedded boards

Categories

Programming Languages | Linters | Debuggers | Testing | Other

Extension Resources

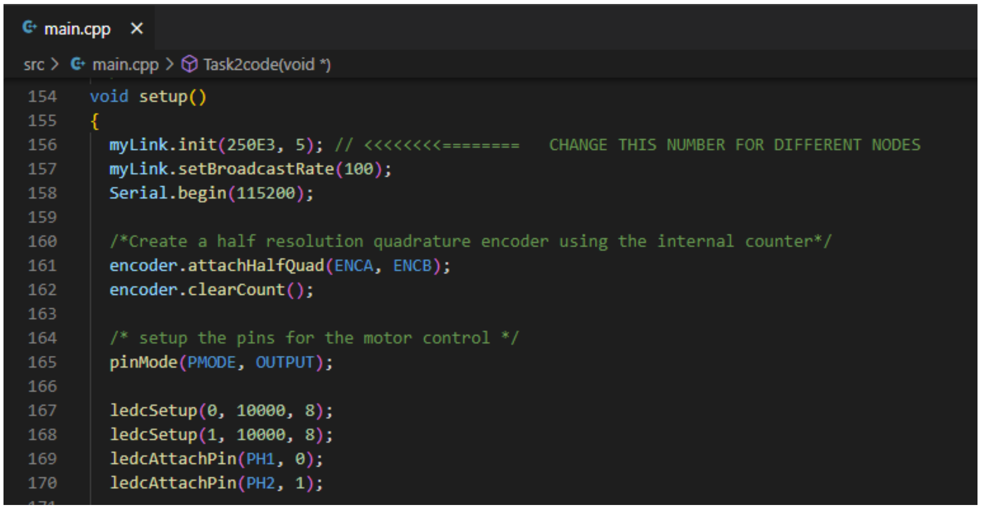
Marketplace
Repository
License
PlatformIO

More Info

Published	5/27/2017, 18:02:51
Last released	3/16/2023, 22:08:06
Last updated	3/17/2023, 12:06:33
Identifier	platformio.platformio-ide

```
PS C:\Users\Adam\Documents\MacBot\firmware\MotorDriver>
```

Each device on a CAN network must have a unique device ID, so that it can be independently addressed. This ID is represented as an integer number and is set when uploading the motor driver code to the motor controller module. It can be found on **line 156** in the **setup method** of the **main.cpp** file.



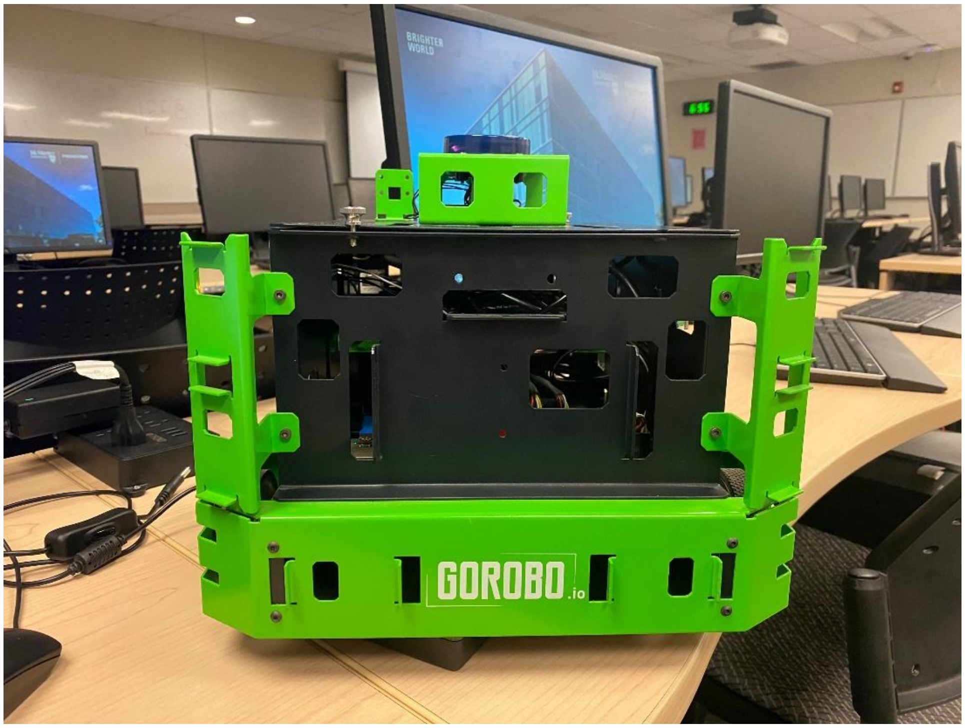
main.cpp ×

src > main.cpp > Task2code(void *)

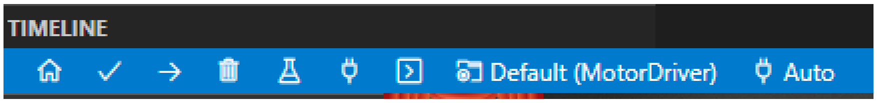
```
void setup()
{
  myLink.init(250E3, 5); // <<<<<<<<========   CHANGE THIS NUMBER FOR DIFFERENT NODES
  myLink.setBroadcastRate(100);
  Serial.begin(115200);

  /*Create a half resolution quadrature encoder using the internal counter*/
  encoder.attachHalfQuad(ENCA, ENCB);
  encoder.clearCount();

  /* setup the pins for the motor control */
  pinMode(PMODE, OUTPUT);

  ledcSetup(0, 10000, 8);
  ledcSetup(1, 10000, 8);
  ledcAttachPin(PH1, 0);
  ledcAttachPin(PH2, 1);
```

To reduce confusion, the **left** motor controller module will be **device #4** and the **right** motor controller module will be **device #5**. The point-of-reference is facing the front of the MacBot.

LEFT (4) < -- > RIGHT (5)



To upload each driver code, click on the verify (✅) and upload (➡️) buttons on the bottom ribbon of the PlatformIO IDE. Ports will automatically be scanned for recognized development boards.



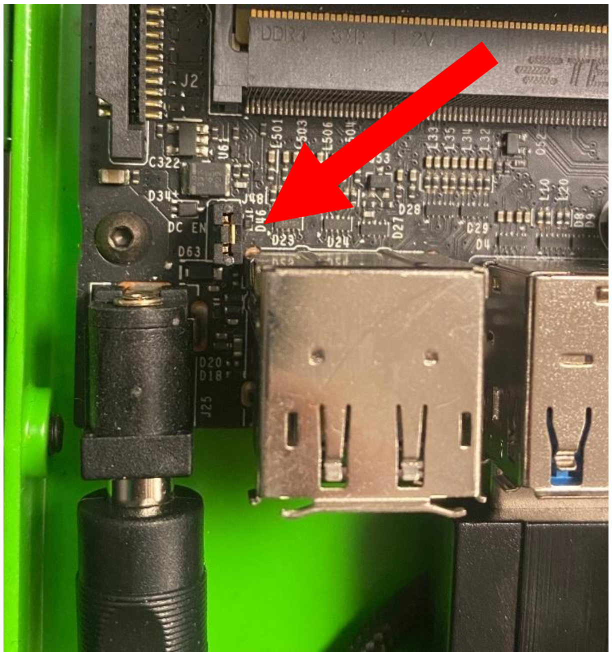
Powering On the MacBot

Before proceeding, I suggest that each group elevate the MacBot off the work space by 2+ cm to ensure that each motor has the freedom to rotate unexpectedly.

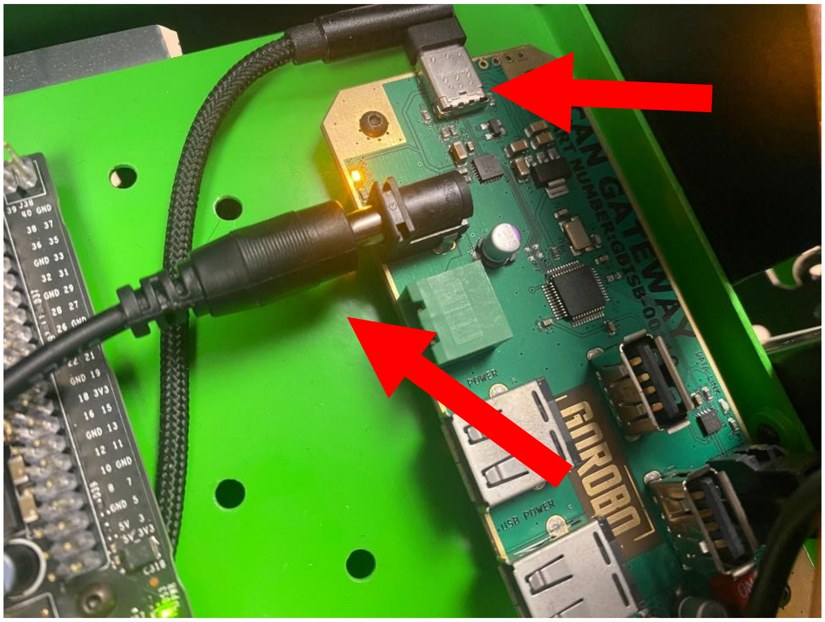
Turn the MacBot ON. It is preferred to operate on battery power as the power distribution board will have the ability to deliver more current to each motor, however a direct-to-wall connection will also work.

To use battery mode, ensure the following prerequisite steps have been taken.

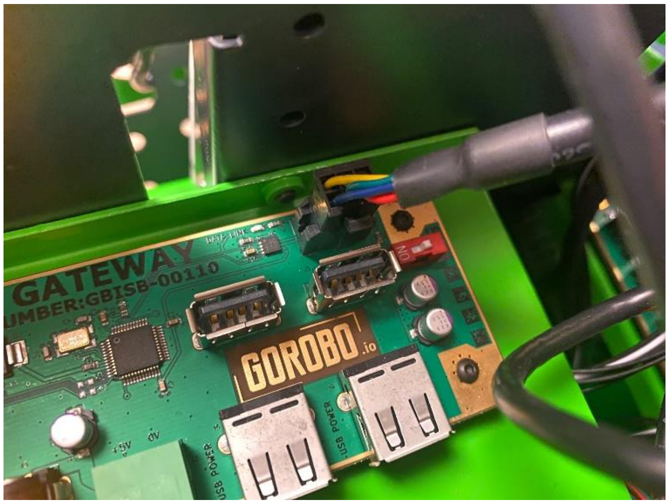
The power input selection jumper should connect the two exposed pins, setting it from MicroUSB 5 W to barrel connector MAXN mode.

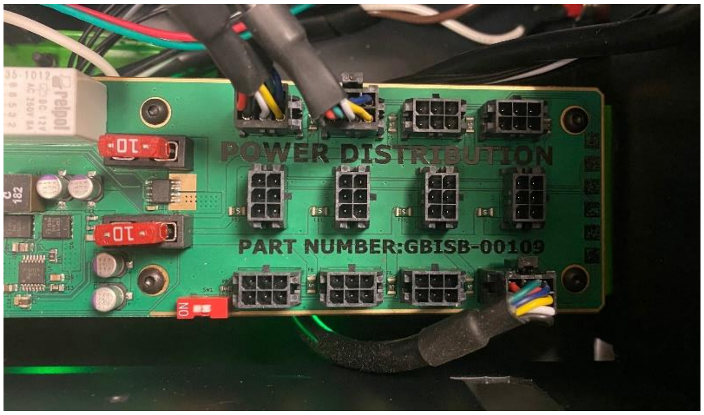


A barrel connector should be used to connect the USB-to-CAN module with the Jetson Nano development board. Ensure the USB-to-CAN module is also connected to one of the Jetson Nano’s USB ports also, using the provided USB to USB-C cable.



Also, ensure that the USB-to-CAN board is receiving power from the power distribution board using a 6-pin connector, and that the appropriate 7.5 to 10 A breakers are used.



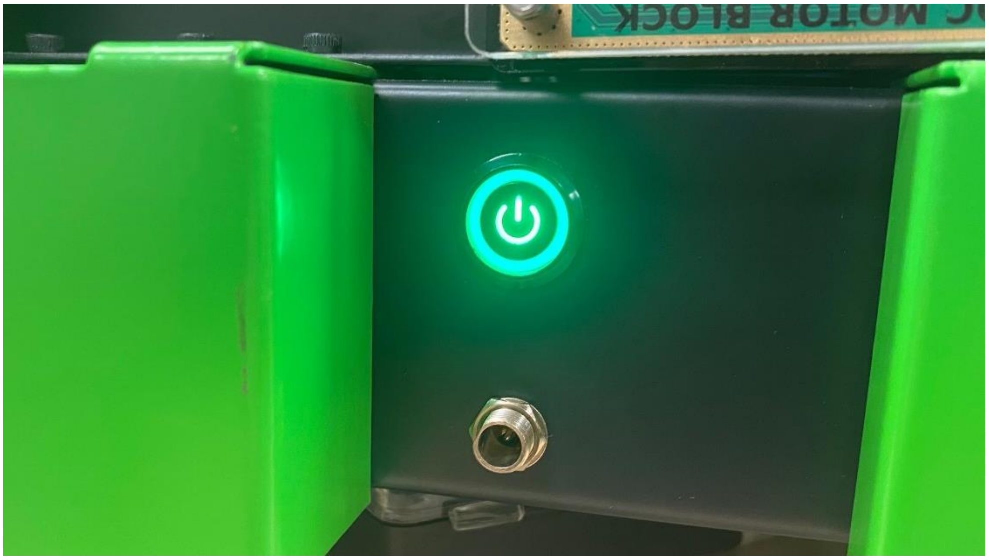


The power distribution board firmware must be downloaded onto the embedded ESP32 device on the board. It can be found in the **firmware/DistributionBoard** directory of the MacBot repository.

https://github.com/adamsokacz/macbot/tree/main/firmware/DistributionBoard

The firmware is required because it must toggle a normally-open power relay on-boot to allow connected devices to receive power. You will hear a click when this happens.

Click the battery power button to turn on the MacBot. You should audibly and visually see indicators power up on the MacBot, as well as the LiDAR begin to spin.



After 1 minute, attempt to connect to the MacBot using the instructions found in the project documentation:

https://adam-36.gitbook.io/macbot/connect

Installing CAN Dependencies

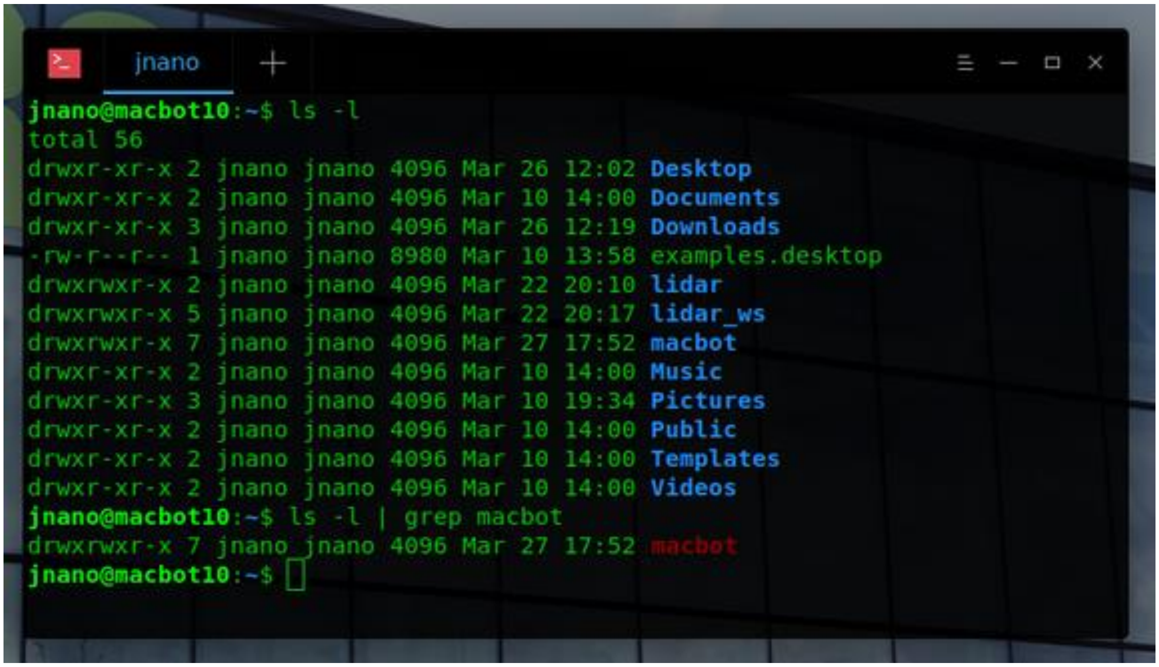
Once the MacBot has been booted and you have successfully connected to the unit, verify that the project files have already been transferred to your user directory.

cd ~

ls -l

If there are many files in this directory, use the GREP command to search file names in this directory.

ls -l | grep macbot



```
jnano@macbot10:~$ ls -l
total 56
drwxr-xr-x 2 jnano jnano 4096 Mar 26 12:02 Desktop
drwxr-xr-x 2 jnano jnano 4096 Mar 10 14:00 Documents
drwxr-xr-x 3 jnano jnano 4096 Mar 26 12:19 Downloads
-rw-r--r-- 1 jnano jnano 8980 Mar 10 13:58 examples.desktop
drwxrwxr-x 2 jnano jnano 4096 Mar 22 20:10 lidar
drwxrwxr-x 5 jnano jnano 4096 Mar 22 20:17 lidar_ws
drwxrwxr-x 7 jnano jnano 4096 Mar 27 17:52 macbot
drwxr-xr-x 2 jnano jnano 4096 Mar 10 14:00 Music
drwxr-xr-x 3 jnano jnano 4096 Mar 10 19:34 Pictures
drwxr-xr-x 2 jnano jnano 4096 Mar 10 14:00 Public
drwxr-xr-x 2 jnano jnano 4096 Mar 10 14:00 Templates
drwxr-xr-x 2 jnano jnano 4096 Mar 10 14:00 Videos
jnano@macbot10:~$ ls -l | grep macbot
drwxrwxr-x 7 jnano jnano 4096 Mar 27 17:52 macbot
jnano@macbot10:~$
```

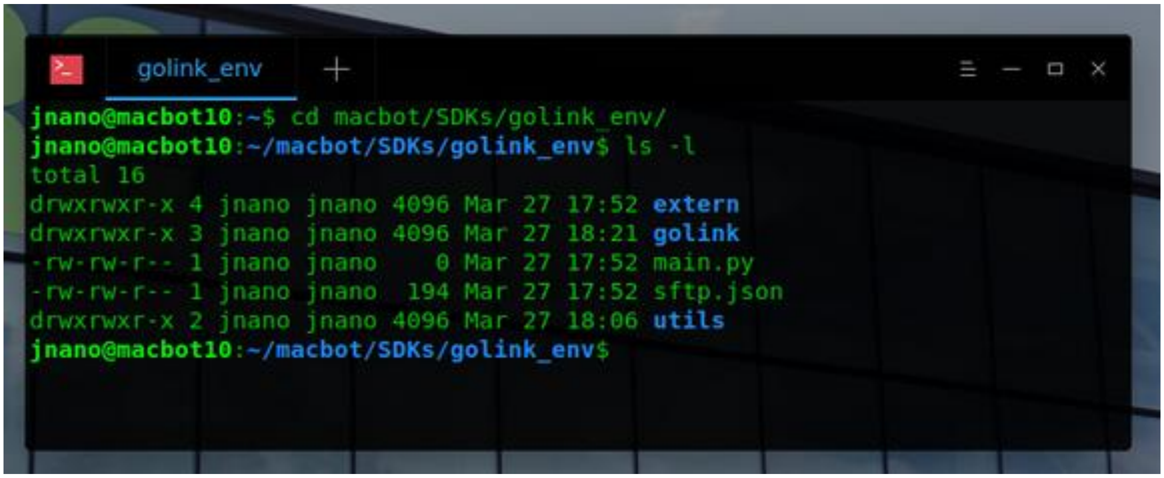
Navigate into the SDK directory inside the macbot project folder and find the **golink_env** folder.

cd macbot

cd SDK

cd golink_env

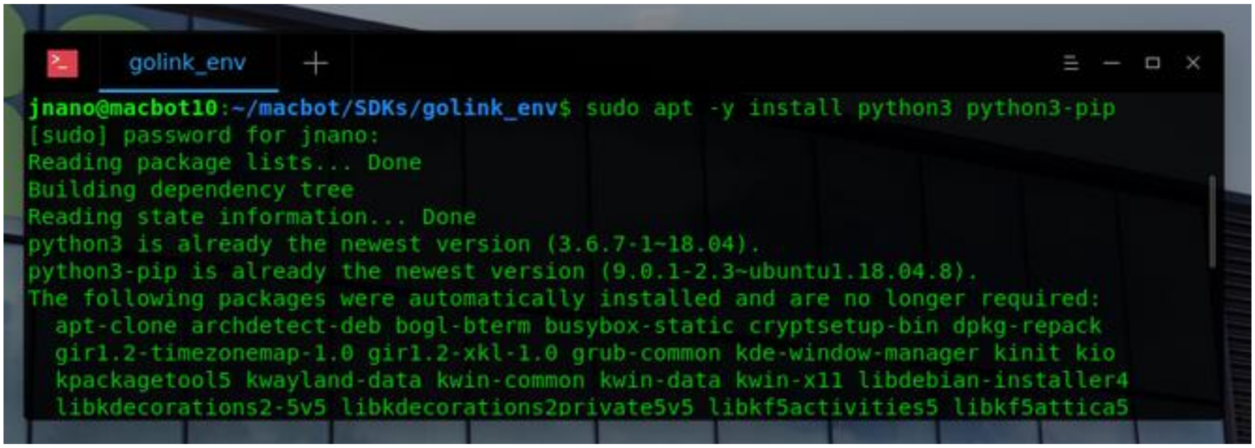
ls -l



```
jnano@macbot10:~$ cd macbot/SDKs/golink_env/
jnano@macbot10:~/macbot/SDKs/golink_env$ ls -l
total 16
drwxrwxr-x 4 jnano jnano 4096 Mar 27 17:52 extern
drwxrwxr-x 3 jnano jnano 4096 Mar 27 18:21 golink
-rw-rw-r-- 1 jnano jnano    0 Mar 27 17:52 main.py
-rw-rw-r-- 1 jnano jnano  194 Mar 27 17:52 sftp.json
drwxrwxr-x 2 jnano jnano 4096 Mar 27 18:06 utils
jnano@macbot10:~/macbot/SDKs/golink_env$
```

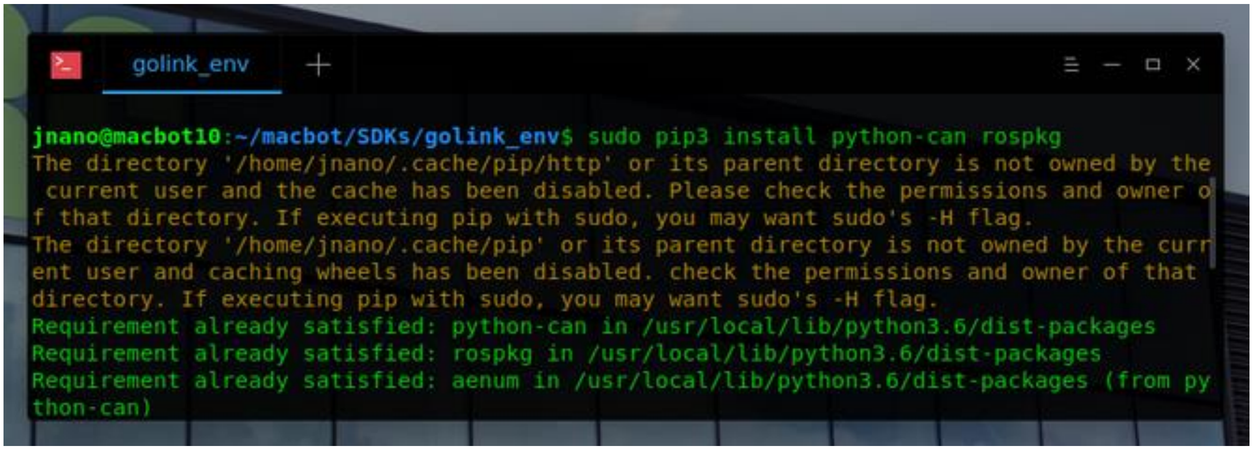
Before using this library, we must ensure that the Python prerequisite packages are installed on our system.

sudo apt -y install python3 python3-pip



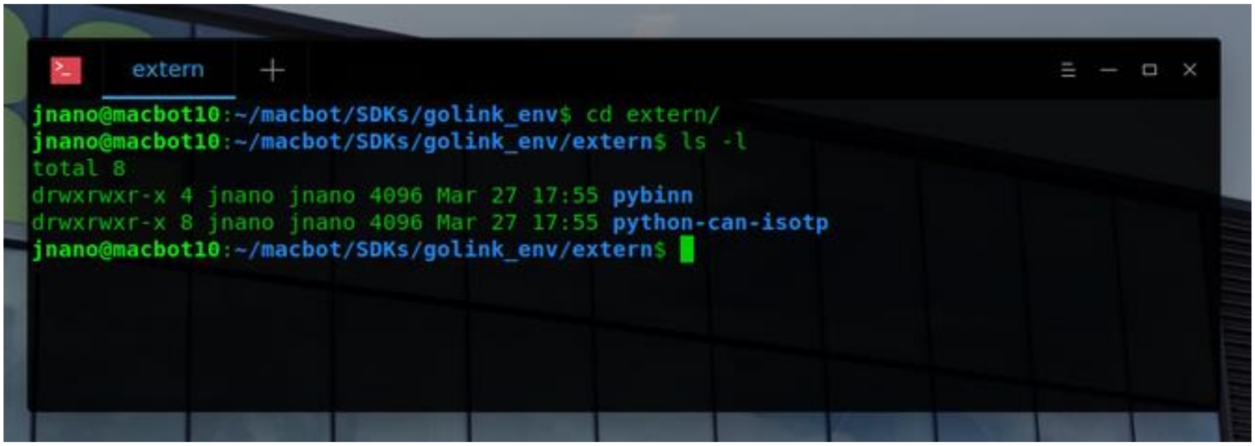
```
jnano@macbot10:~/macbot/SDKs/golink_env$ sudo apt -y install python3 python3-pip
[sudo] password for jnano:
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3 is already the newest version (3.6.7-1~18.04).
python3-pip is already the newest version (9.0.1-2.3~ubuntu1.18.04.8).
The following packages were automatically installed and are no longer required:
  apt-clone archdetect-deb bogl-bterm busybox-static cryptsetup-bin dpkg-repack
  gir1.2-timezonemap-1.0 gir1.2-xkl-1.0 grub-common kde-window-manager kinit kio
  kpackagetool5 kwayland-data kwin-common kwin-data kwin-x11 libdebian-installer4
  libkdecorations2-5v5 libkdecorations2private5v5 libkf5activities5 libkf5attica5
```

sudo pip3 install python-can rospkg



```
jnano@macbot10:~/macbot/SDKs/golink_env$ sudo pip3 install python-can rospkg
The directory '/home/jnano/.cache/pip/http' or its parent directory is not owned by the
 current user and the cache has been disabled. Please check the permissions and owner o
f that directory. If executing pip with sudo, you may want sudo's -H flag.
The directory '/home/jnano/.cache/pip' or its parent directory is not owned by the curr
ent user and caching wheels has been disabled. check the permissions and owner of that
directory. If executing pip with sudo, you may want sudo's -H flag.
Requirement already satisfied: python-can in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: rospkg in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: aenum in /usr/local/lib/python3.6/dist-packages (from py
thon-can)
```

Next, there are a few Python packages that must be installed manually. Navigate to the **extern/** directory. This directory typically contains external dependencies to a project.



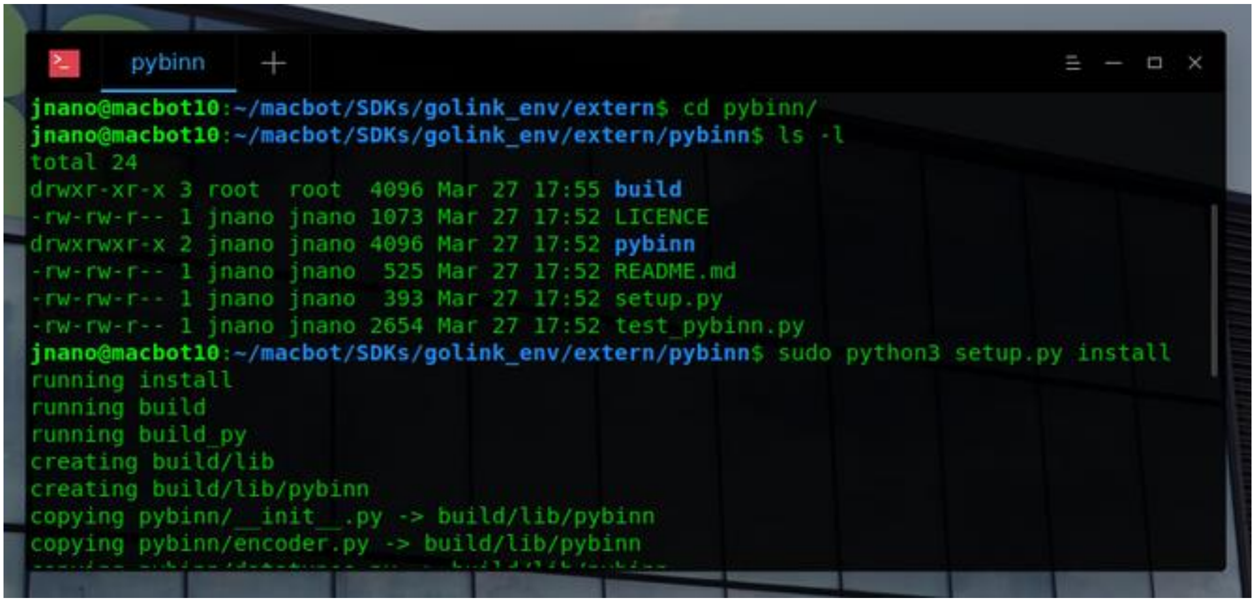
```
jnano@macbot10:~/macbot/SDKs/golink_env$ cd extern/
jnano@macbot10:~/macbot/SDKs/golink_env/extern$ ls -l
total 8
drwxrwxr-x 4 jnano jnano 4096 Mar 27 17:55 pybinn
drwxrwxr-x 8 jnano jnano 4096 Mar 27 17:55 python-can-isotp
jnano@macbot10:~/macbot/SDKs/golink_env/extern$
```

First, lets install **pybinn**. Navigate into the pybinn directory and manually install it.

cd pybinn

ls -l

sudo python3 setup.py install



```
jnano@macbot10:~/macbot/SDKs/golink_env/extern$ cd pybinn/
jnano@macbot10:~/macbot/SDKs/golink_env/extern/pybinn$ ls -l
total 24
drwxr-xr-x 3 root  root  4096 Mar 27 17:55 build
-rw-rw-r-- 1 jnano jnano 1073 Mar 27 17:52 LICENCE
drwxrwxr-x 2 jnano jnano 4096 Mar 27 17:52 pybinn
-rw-rw-r-- 1 jnano jnano  525 Mar 27 17:52 README.md
-rw-rw-r-- 1 jnano jnano  393 Mar 27 17:52 setup.py
-rw-rw-r-- 1 jnano jnano 2654 Mar 27 17:52 test_pybinn.py
jnano@macbot10:~/macbot/SDKs/golink_env/extern/pybinn$ sudo python3 setup.py install
running install
running build
running build_py
creating build/lib
creating build/lib/pybinn
copying pybinn/__init__.py -> build/lib/pybinn
copying pybinn/encoder.py -> build/lib/pybinn
```

Navigate back to the extern/ directory, then into the **python-can-isotp/** directory. Install this package manually.

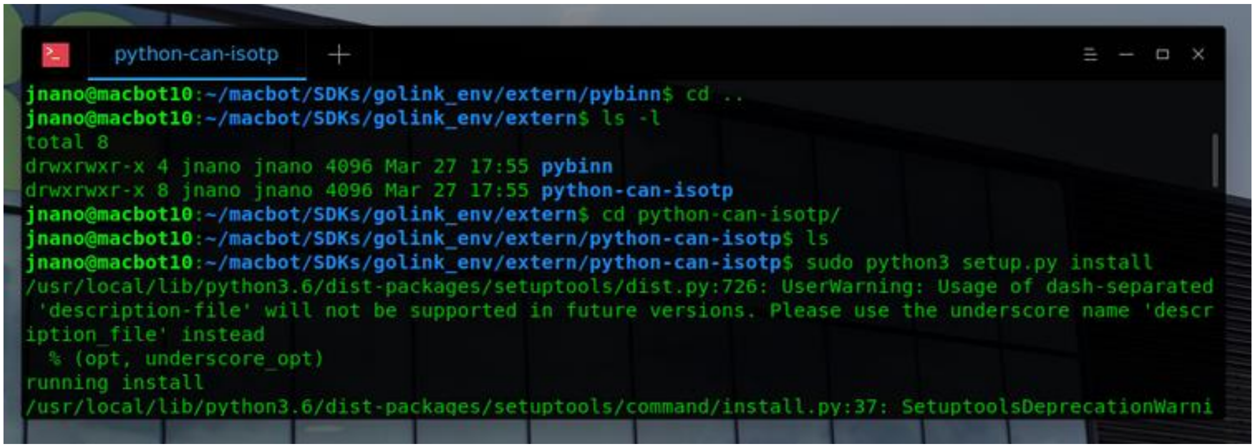
cd ..

ls -l

cd python-can-isotp

ls

sudo python3 setup.py install



```
jnano@macbot10:~/macbot/SDKs/golink_env/extern/pybinn$ cd ..
jnano@macbot10:~/macbot/SDKs/golink_env/extern$ ls -l
total 8
drwxrwxr-x 4 jnano jnano 4096 Mar 27 17:55 pybinn
drwxrwxr-x 8 jnano jnano 4096 Mar 27 17:55 python-can-isotp
jnano@macbot10:~/macbot/SDKs/golink_env/extern$ cd python-can-isotp/
jnano@macbot10:~/macbot/SDKs/golink_env/extern/python-can-isotp$ ls
jnano@macbot10:~/macbot/SDKs/golink_env/extern/python-can-isotp$ sudo python3 setup.py install
/usr/local/lib/python3.6/dist-packages/setuptools/dist.py:726: UserWarning: Usage of dash-separated
 'description-file' will not be supported in future versions. Please use the underscore name 'descr
iption_file' instead
  % (opt, underscore_opt)
running install
/usr/local/lib/python3.6/dist-packages/setuptools/command/install.py:37: SetuptoolsDeprecationWarni
```

Lastly, return to the golink_env directory.

cd ../..

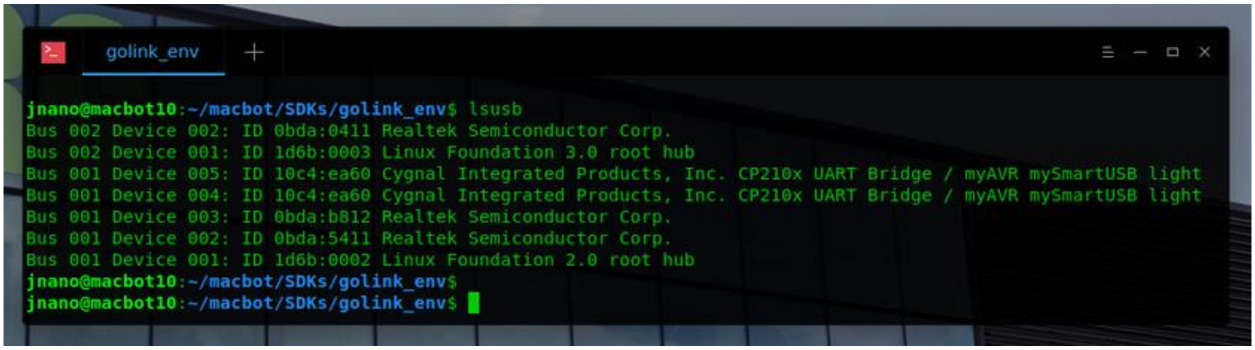
Initializing CAN Communication

In order to begin communicating over CAN, we need to enable the low-level Linux kernel modules that help it communicate with CAN devices.

In order to configure it correctly, we need to know which port the USB-to-CAN board is connected to.

List usb devices. You should notice 2 AVR devices that rely on the CP210X driver. One is the LiDAR and the other is the USB-to-CAN board.

lsusb

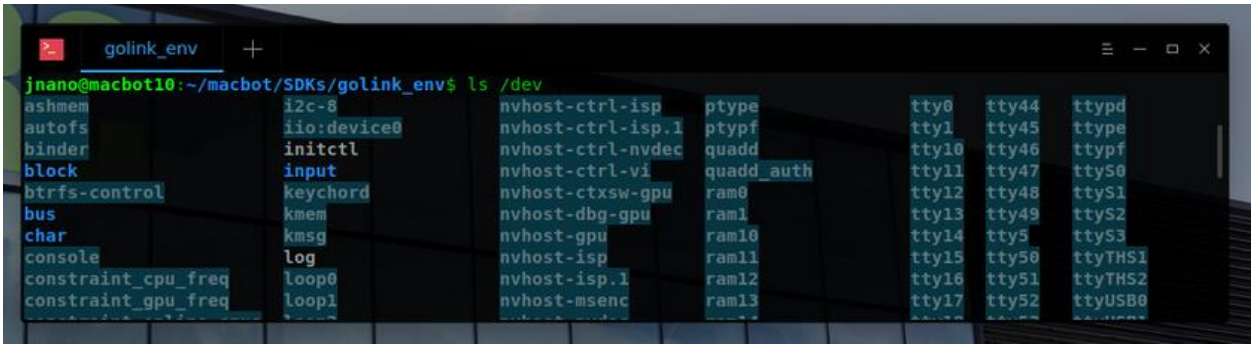


```
jnano@macbot10:~/macbot/SDKs/golink_env$ lsusb
Bus 002 Device 002: ID 0bda:0411 Realtek Semiconductor Corp.
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 005: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP210x UART Bridge / myAVR mySmartUSB light
Bus 001 Device 004: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP210x UART Bridge / myAVR mySmartUSB light
Bus 001 Device 003: ID 0bda:b812 Realtek Semiconductor Corp.
Bus 001 Device 002: ID 0bda:5411 Realtek Semiconductor Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
jnano@macbot10:~/macbot/SDKs/golink_env$
jnano@macbot10:~/macbot/SDKs/golink_env$
```

The USB-to-CAN board is usually the second of the two devices to be initialized, because of a software delay on the power relay in the distribution board firmware.

We must find which USB port the CAN board is connected to.

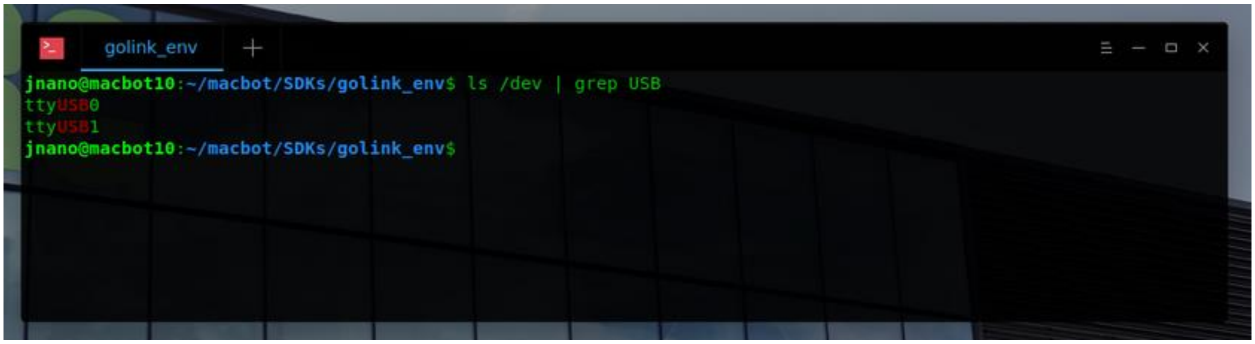
ls /dev



```
jnano@macbot10:~/macbot/SDKs/golink_env$ ls /dev
ashmem                  i2c-8             nvhost-ctrl-isp      ptype            tty0   tty44  ttypd
autofs                  iio:device0       nvhost-ctrl-isp.1    ptypf            tty1   tty45  ttype
binder                  initctl           nvhost-ctrl-nvdec    quadd            tty10  tty46  ttypf
block                   input             nvhost-ctrl-vi       quadd_auth       tty11  tty47  ttyS0
btrfs-control           keychord          nvhost-ctxsw-gpu     ram0             tty12  tty48  ttyS1
bus                     kmem              nvhost-dbg-gpu       ram1             tty13  tty49  ttyS2
char                    kmsg              nvhost-gpu           ram10            tty14  tty5   ttyS3
console                 log               nvhost-isp           ram11            tty15  tty50  ttyTHS1
constraint_cpu_freq     loop0             nvhost-isp.1         ram12            tty16  tty51  ttyTHS2
constraint_gpu_freq     loop1             nvhost-msenc         ram13            tty17  tty52  ttyUSB0
```

You will notice that there are many virtual and hardware connections listed. To narrow it down, search file names using the GREP command.

ls /dev | grep USB



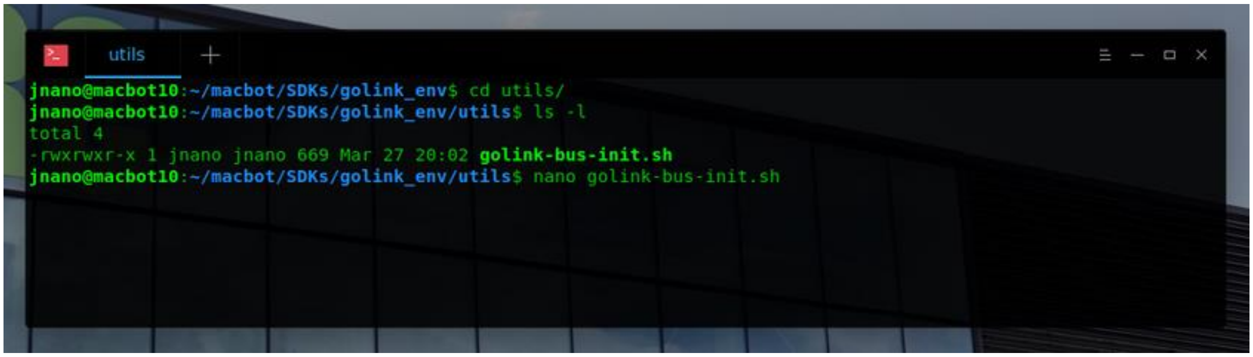
```
jnano@macbot10:~/macbot/SDKs/golink_env$ ls /dev | grep USB
ttyUSB0
ttyUSB1
jnano@macbot10:~/macbot/SDKs/golink_env$
```

Therefore, we can safely assume that the USB-to-CAN board is connected to **/dev/ttyUSB1**.

There is a pre-written bash script inside of golink_env/utils to automate the connection process. Before running it, launch it in a text editor to verify that it attempts to initialize on /dev/ttyUSB1.

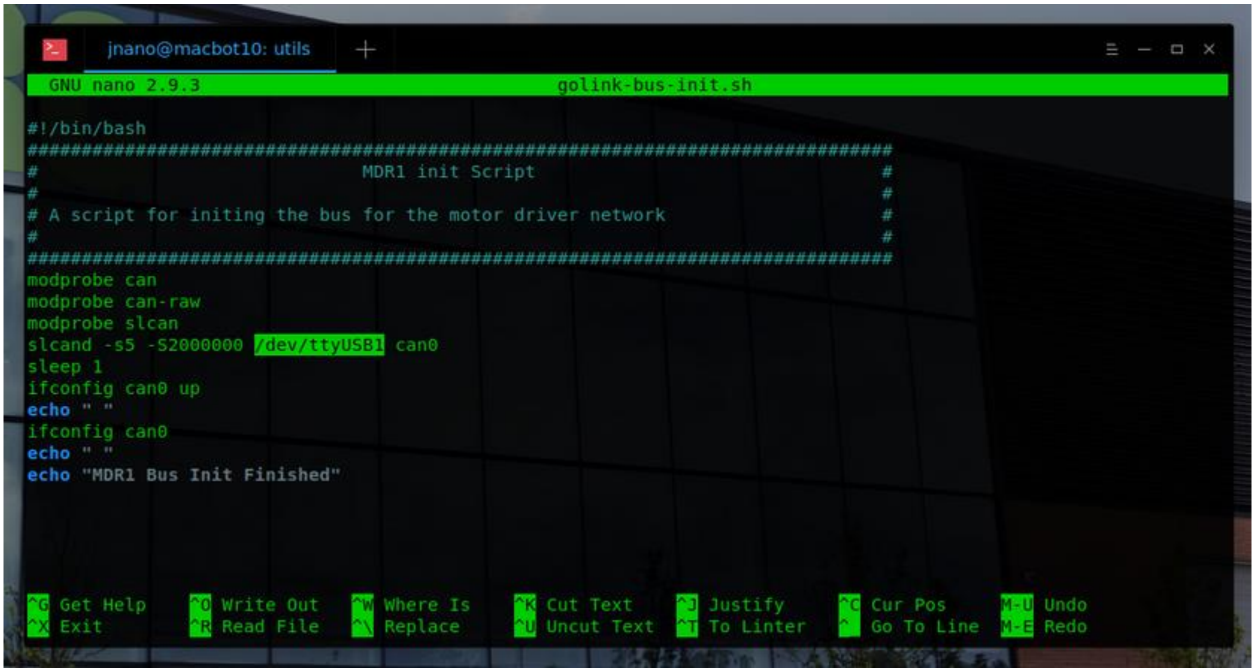
cd utils

nano golink-bus-init.sh



```
jnano@macbot10:~/macbot/SDKs/golink_env$ cd utils/
jnano@macbot10:~/macbot/SDKs/golink_env/utils$ ls -l
total 4
-rwxrwxr-x 1 jnano jnano 669 Mar 27 20:02 golink-bus-init.sh
jnano@macbot10:~/macbot/SDKs/golink_env/utils$ nano golink-bus-init.sh
```

Nano uses commands similar to windows commands to navigate the interface. Use your arrow keys to navigate the text file. Hold down CTRL + arrow keys to skip word-by-word. Use CTRL + S to save the buffer to the file. Use CTRL + X to exit.

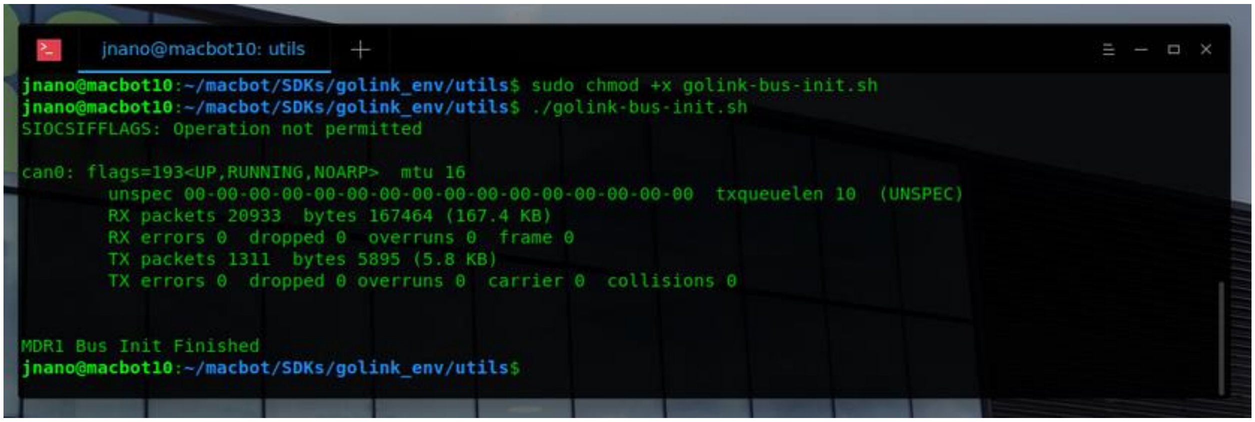


```
#!/bin/bash
################################################################################
#                               MDR1 init Script                               #
#                                                                              #
# A script for initing the bus for the motor driver network                    #
#                                                                              #
################################################################################
modprobe can
modprobe can-raw
modprobe slcan
slcand -s5 -S2000000 /dev/ttyUSB1 can0
sleep 1
ifconfig can0 up
echo " "
ifconfig can0
echo " "
echo "MDR1 Bus Init Finished"
```

Once you have verified that the port matches, set the script to be executable and run this shell script as administrator.

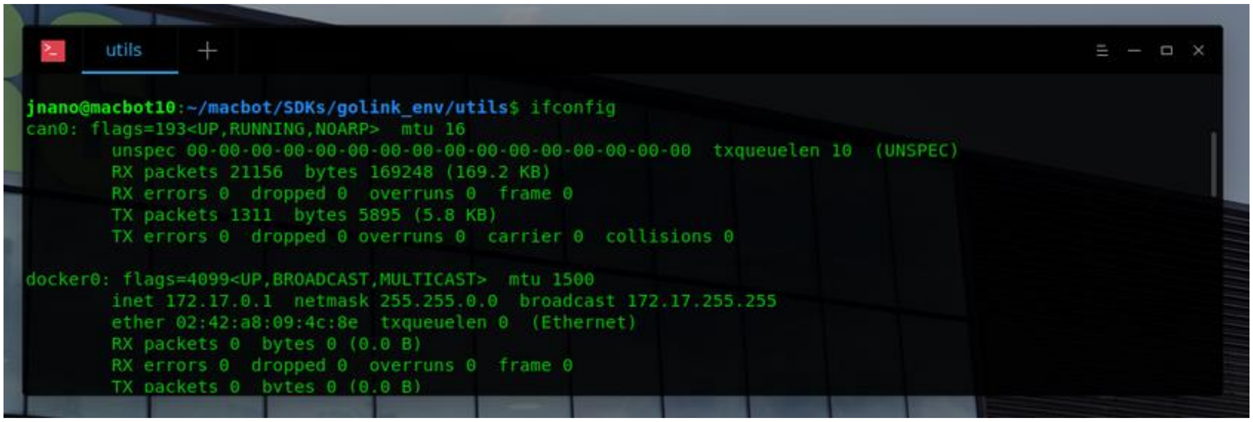
sudo chmod +x golink-bus-init.sh

./golink-bus-init.sh



```
jnano@macbot10:~/macbot/SDKs/golink_env/utils$ sudo chmod +x golink-bus-init.sh
jnano@macbot10:~/macbot/SDKs/golink_env/utils$ ./golink-bus-init.sh
SIOCSIFFLAGS: Operation not permitted

can0: flags=193<UP,RUNNING,NOARP>  mtu 16
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 10  (UNSPEC)
        RX packets 20933  bytes 167464 (167.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1311  bytes 5895 (5.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0


MDR1 Bus Init Finished
jnano@macbot10:~/macbot/SDKs/golink_env/utils$
```

If the initialization was successful, you MUST see the CAN0 interface printed to your terminal window. It can also be viewed with the common **ifconfig** command.



```
jnano@macbot10:~/macbot/SDKs/golink_env/utils$ ifconfig
can0: flags=193<UP,RUNNING,NOARP>  mtu 16
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 10  (UNSPEC)
        RX packets 21156  bytes 169248 (169.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1311  bytes 5895 (5.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:a8:09:4c:8e  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
```

It is important to note that this interface MUST be re-initialized after each boot of the MacBot.

Exercise A:

What is a network interface in Linux?

What command is used to view active network interfaces in Linux?

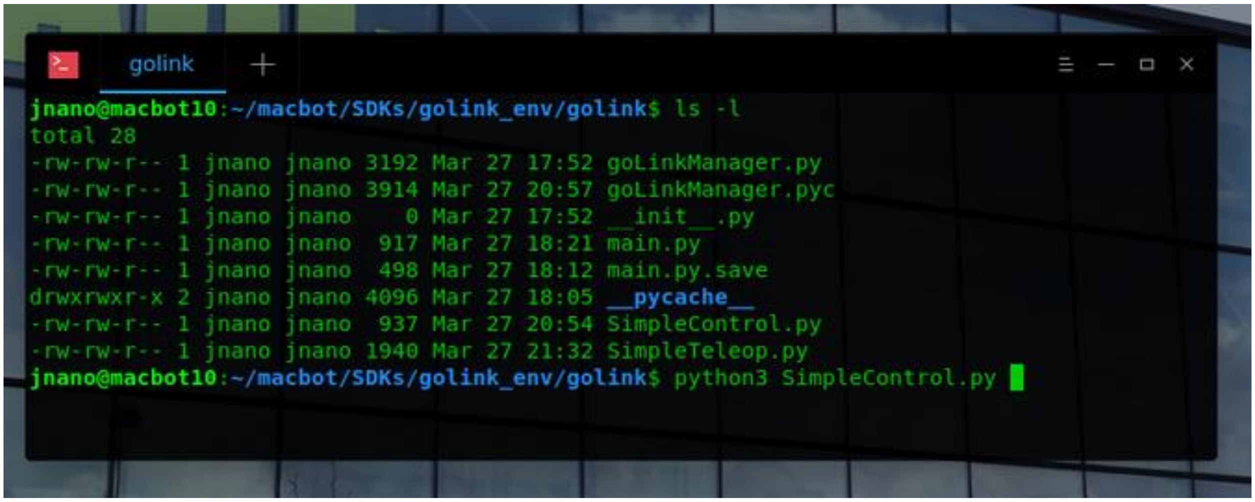
What does the GREP command do in Linux? Why does it used with a | (pipe) character?

Controlling Motors Independently using Python

In the golink_env /golink folder, launch SimpleControl.py.

ls -l

python3 SimpleControl.py

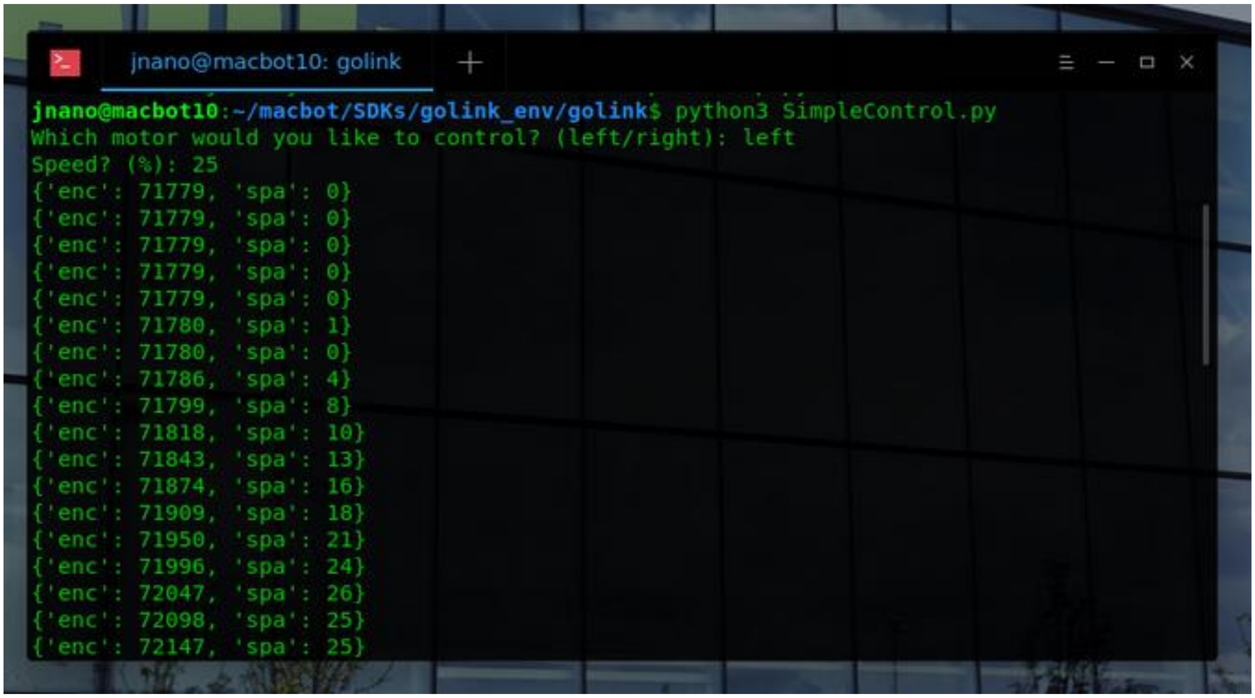


```
jnano@macbot10:~/macbot/SDKs/golink_env/golink$ ls -l
total 28
-rw-rw-r-- 1 jnano jnano 3192 Mar 27 17:52 goLinkManager.py
-rw-rw-r-- 1 jnano jnano 3914 Mar 27 20:57 goLinkManager.pyc
-rw-rw-r-- 1 jnano jnano    0 Mar 27 17:52 __init__.py
-rw-rw-r-- 1 jnano jnano  917 Mar 27 18:21 main.py
-rw-rw-r-- 1 jnano jnano  498 Mar 27 18:12 main.py.save
drwxrwxr-x 2 jnano jnano 4096 Mar 27 18:05 __pycache__
-rw-rw-r-- 1 jnano jnano  937 Mar 27 20:54 SimpleControl.py
-rw-rw-r-- 1 jnano jnano 1940 Mar 27 21:32 SimpleTeleop.py
jnano@macbot10:~/macbot/SDKs/golink_env/golink$ python3 SimpleControl.py
```

Enter **left** and **25**.

left

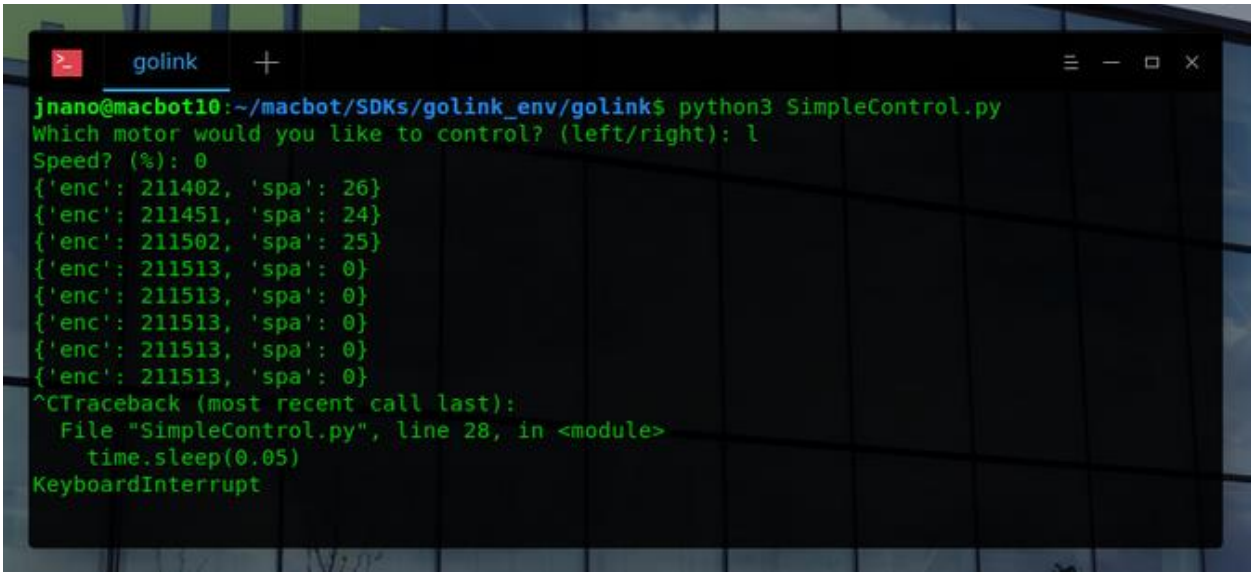
25



```
jnano@macbot10:~/macbot/SDKs/golink_env/golink$ python3 SimpleControl.py
Which motor would you like to control? (left/right): left
Speed? (%): 25
{'enc': 71779, 'spa': 0}
{'enc': 71779, 'spa': 0}
{'enc': 71779, 'spa': 0}
{'enc': 71779, 'spa': 0}
{'enc': 71779, 'spa': 0}
{'enc': 71780, 'spa': 1}
{'enc': 71780, 'spa': 0}
{'enc': 71786, 'spa': 4}
{'enc': 71799, 'spa': 8}
{'enc': 71818, 'spa': 10}
{'enc': 71843, 'spa': 13}
{'enc': 71874, 'spa': 16}
{'enc': 71909, 'spa': 18}
{'enc': 71950, 'spa': 21}
{'enc': 71996, 'spa': 24}
{'enc': 72047, 'spa': 26}
{'enc': 72098, 'spa': 25}
{'enc': 72147, 'spa': 25}
```

Notice that the left motor begins rotating.

Stop the rotation.

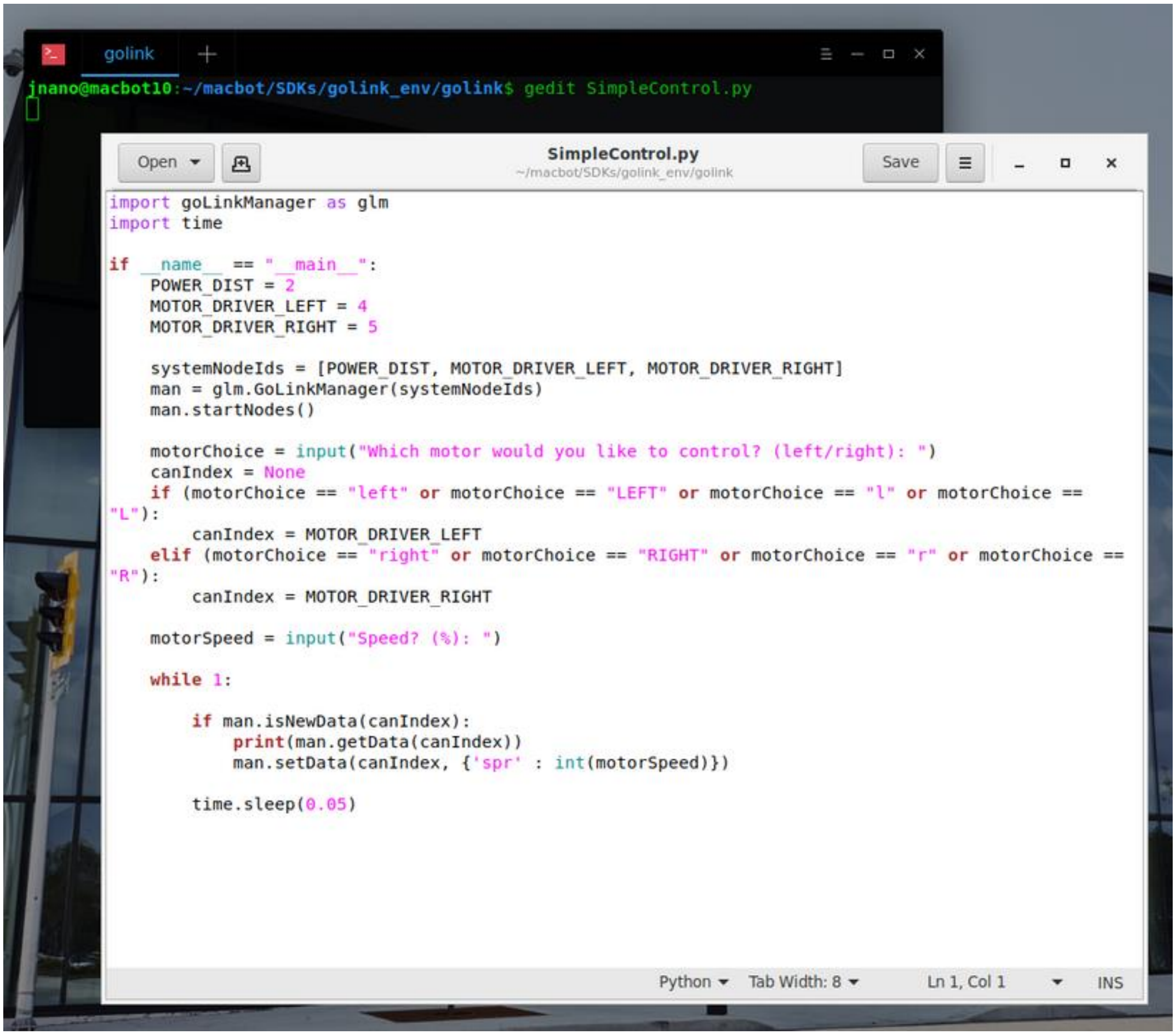


```
jnano@macbot10:~/macbot/SDKs/golink_env/golink$ python3 SimpleControl.py
Which motor would you like to control? (left/right): l
Speed? (%): 0
{'enc': 211402, 'spa': 26}
{'enc': 211451, 'spa': 24}
{'enc': 211502, 'spa': 25}
{'enc': 211513, 'spa': 0}
{'enc': 211513, 'spa': 0}
{'enc': 211513, 'spa': 0}
{'enc': 211513, 'spa': 0}
{'enc': 211513, 'spa': 0}
^CTraceback (most recent call last):
  File "SimpleControl.py", line 28, in <module>
    time.sleep(0.05)
KeyboardInterrupt
```

Ensure that you are able to rotate both the left and right wheels and read corresponding encoder data.

Open the Python script to learn how the library is being used behind the scenes.

gedit SimpleControl.py



```
jnano@macbot10:~/macbot/SDKs/golink_env/golink$ gedit SimpleControl.py
```

```
import goLinkManager as glm
import time

if __name__ == "__main__":
    POWER_DIST = 2
    MOTOR_DRIVER_LEFT = 4
    MOTOR_DRIVER_RIGHT = 5

    systemNodeIds = [POWER_DIST, MOTOR_DRIVER_LEFT, MOTOR_DRIVER_RIGHT]
    man = glm.GoLinkManager(systemNodeIds)
    man.startNodes()

    motorChoice = input("Which motor would you like to control? (left/right): ")
    canIndex = None
    if (motorChoice == "left" or motorChoice == "LEFT" or motorChoice == "l" or motorChoice ==
"L"):
        canIndex = MOTOR_DRIVER_LEFT
    elif (motorChoice == "right" or motorChoice == "RIGHT" or motorChoice == "r" or motorChoice ==
"R"):
        canIndex = MOTOR_DRIVER_RIGHT

    motorSpeed = input("Speed? (%): ")

    while 1:

        if man.isNewData(canIndex):
            print(man.getData(canIndex))
            man.setData(canIndex, {'spr' : int(motorSpeed)})

        time.sleep(0.05)
```

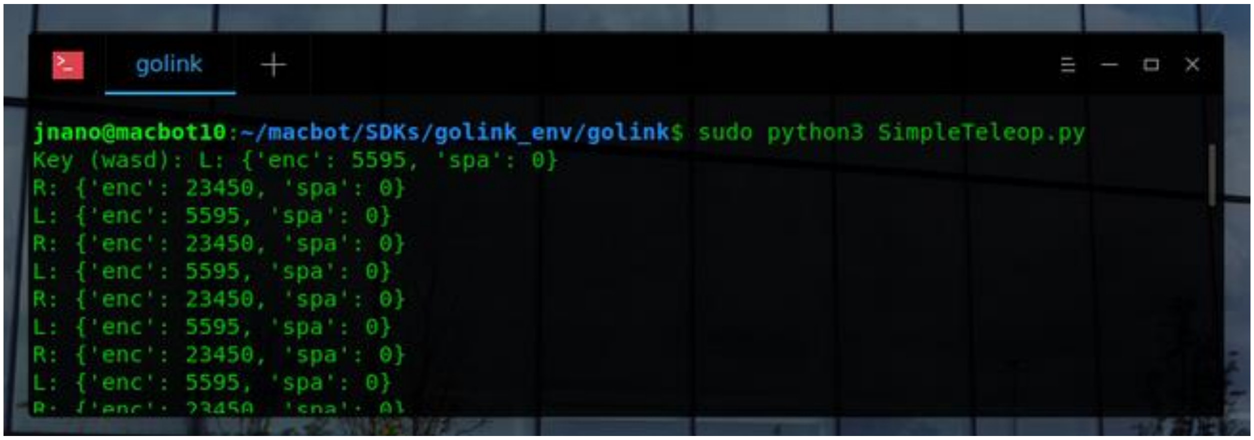
Exercise B:

Record a short video of the wheels responding to your speed change commands. Include this video with your submission.

Coordinating Motors using Python

Launch the SimpleTeleop Python script as superuser.

```
sudo python3 ./SimpleTeleop.py
```



```
jnano@macbot10:~/macbot/SDKs/golink_env/golink$ sudo python3 SimpleTeleop.py
Key (wasd): L: {'enc': 5595, 'spa': 0}
R: {'enc': 23450, 'spa': 0}
L: {'enc': 5595, 'spa': 0}
R: {'enc': 23450, 'spa': 0}
L: {'enc': 5595, 'spa': 0}
R: {'enc': 23450, 'spa': 0}
L: {'enc': 5595, 'spa': 0}
R: {'enc': 23450, 'spa': 0}
L: {'enc': 5595, 'spa': 0}
R: {'enc': 23450, 'spa': 0}
```

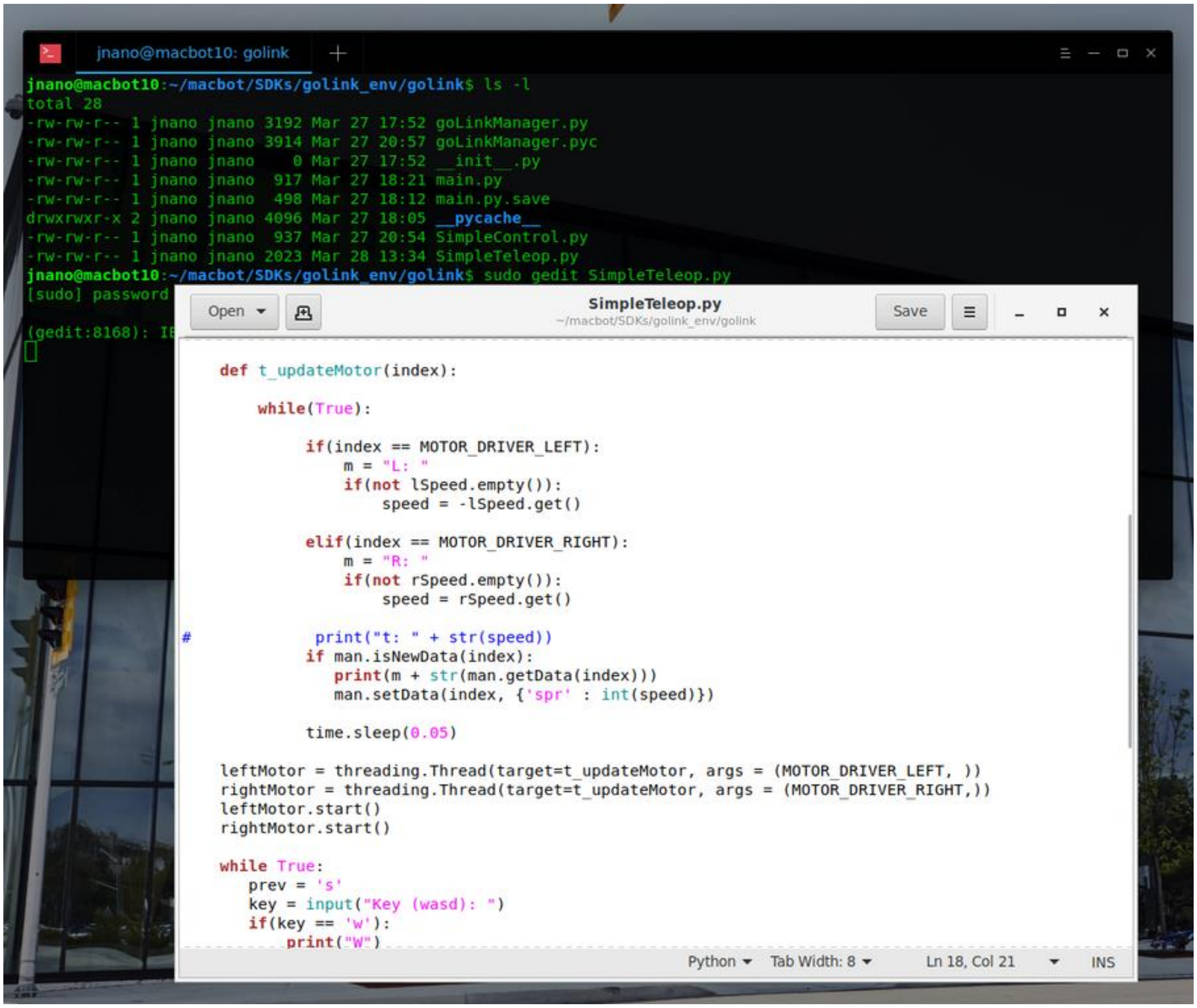
The script reads keyboard input to change direction. Type the following commands:

```
w + <enter>

s + <enter>

a + <enter>

d + <enter>

s + <enter>
```

Once you've established that the teleop controls operate correctly, place the MacBot on the floor and attempt to drive it around, near your lab station.

Next, open the SimpleTeleop Python script using a text editor.

```
ls -l

sudo gedit SimpleTeleop.py
```



```
jnano@macbot10:~/macbot/SDKs/golink_env/golink$ ls -l
total 28
-rw-rw-r-- 1 jnano jnano 3192 Mar 27 17:52 goLinkManager.py
-rw-rw-r-- 1 jnano jnano 3914 Mar 27 20:57 goLinkManager.pyc
-rw-rw-r-- 1 jnano jnano    0 Mar 27 17:52 __init__.py
-rw-rw-r-- 1 jnano jnano  917 Mar 27 18:21 main.py
-rw-rw-r-- 1 jnano jnano  498 Mar 27 18:12 main.py.save
drwxrwxr-x 2 jnano jnano 4096 Mar 27 18:05 __pycache__
-rw-rw-r-- 1 jnano jnano  937 Mar 27 20:54 SimpleControl.py
-rw-rw-r-- 1 jnano jnano 2023 Mar 28 13:34 SimpleTeleop.py
jnano@macbot10:~/macbot/SDKs/golink_env/golink$ sudo gedit SimpleTeleop.py
[sudo] password

(gedit:8168): IE
```

```python
def t_updateMotor(index):

    while(True):

        if(index == MOTOR_DRIVER_LEFT):
            m = "L: "
            if(not lSpeed.empty()):
                speed = -lSpeed.get()

        elif(index == MOTOR_DRIVER_RIGHT):
            m = "R: "
            if(not rSpeed.empty()):
                speed = rSpeed.get()

#         print("t: " + str(speed))
        if man.isNewData(index):
           print(m + str(man.getData(index)))
           man.setData(index, {'spr' : int(speed)})

        time.sleep(0.05)

leftMotor = threading.Thread(target=t_updateMotor, args = (MOTOR_DRIVER_LEFT, ))
rightMotor = threading.Thread(target=t_updateMotor, args = (MOTOR_DRIVER_RIGHT,))
leftMotor.start()
rightMotor.start()

while True:
   prev = 's'
   key = input("Key (wasd): ")
   if(key == 'w'):
       print("W")
```

Notice how each control loop is threaded. Your keyboard interactions are loaded into globally accessible queues. The bottom of each queue (lowest index) is then read during each control loop for the respective motor. The change is then communicated over the CAN bus.

Exercise C:

Record a short video of the MacBot being teleoperated on the floor from your PC over the network. Include this video with your submission.

ROS Integration