Contents

Lab 2 – TF Tree and Robot Setup

Requirements:

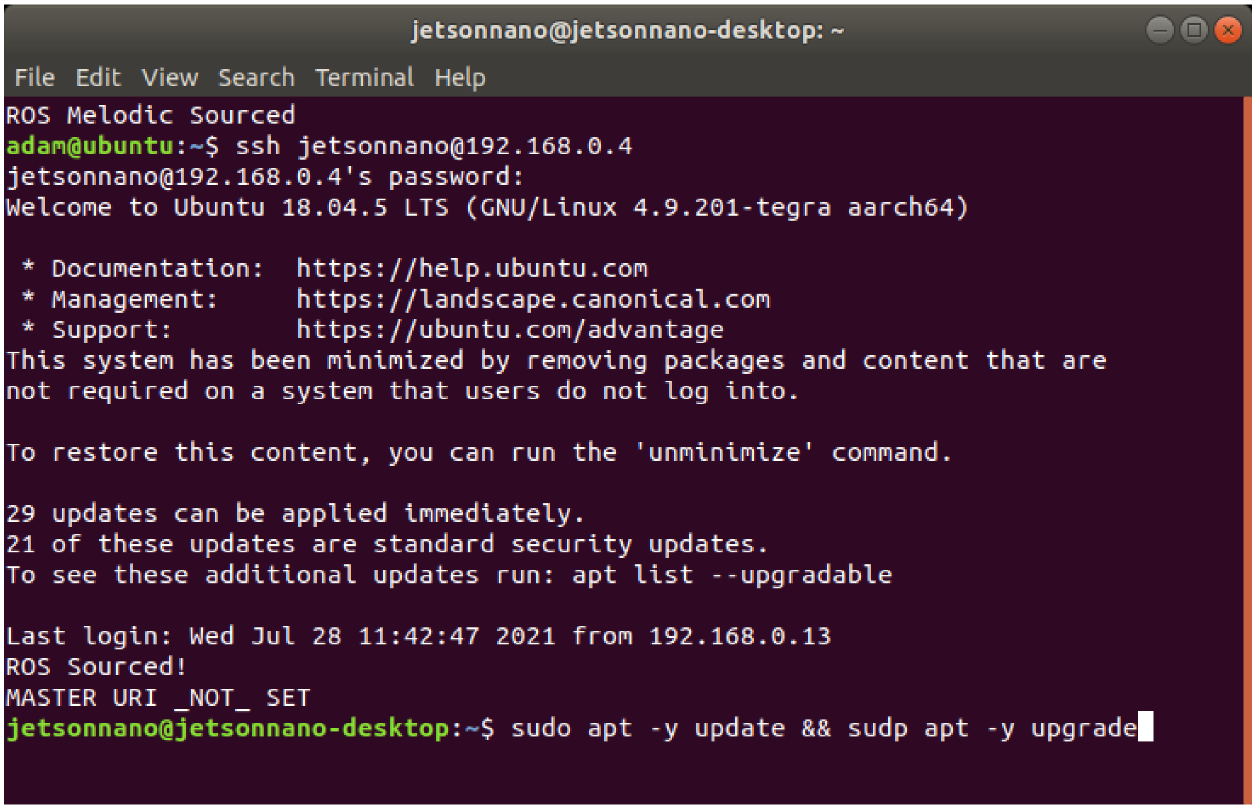
- Completion of Lab 1
- Basic experience accessing machines over SSH
- Basic knowledge of PlatformIO

The purpose of this lab is to set up the LiDAR and use the laser_scan_matcher ROS package to get pose data. Pose is the combination of position and orientation data. It allows us to orient points in 3D space. We will also be interfacing with MacBot's motors to drive it using our keyboard. This will be achieved by working with various nodes that will communicate using the MacBot's CAN bus. In order to do so, we will need to write our own nodes that will help interface between ROS and the MacBot. We will be jumping back and forth between ROS and PlatformIO to cover all aspects of the MacBot relevant to the lab.

Installing the YDLiDAR ROS Package

Start your local virtual machine and log in.

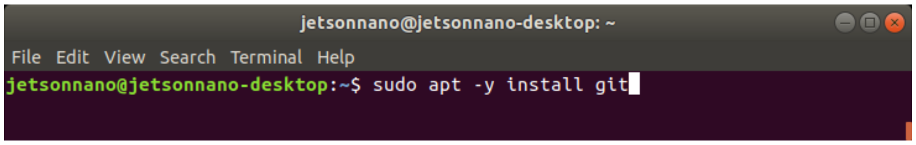
SSH into the MacBot.


jetsonnano@jetsonnano-desktop: ~
File Edit View Search Terminal Help

```
ROS Melodic Sourced
adam@ubuntu:~$ ssh jetsonnano@192.168.0.4
jetsonnano@192.168.0.4's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.9.201-tegra aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

29 updates can be applied immediately.
21 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Wed Jul 28 11:42:47 2021 from 192.168.0.13
ROS Sourced!
MASTER URI _NOT_ SET
jetsonnano@jetsonnano-desktop:~$ sudo apt -y update && sudp apt -y upgrade
```

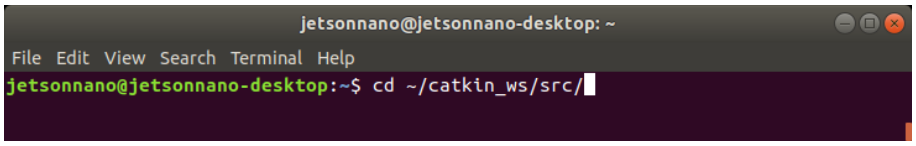
Ensure that the Git version control tool is installed. Do this by using the package manager.

sudo apt install git


jetsonnano@jetsonnano-desktop: ~
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~$ sudo apt -y install git
```

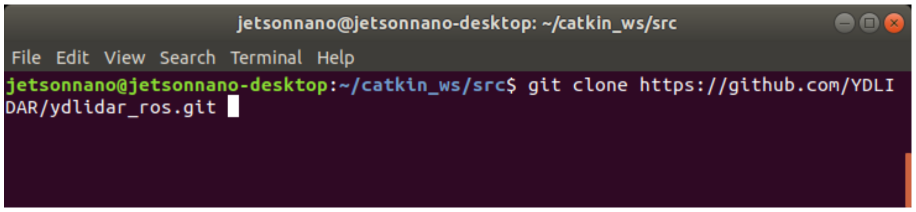
Next, open a new terminal window and navigate to **~/<catkin_workspace_name>/src.**


jetsonnano@jetsonnano-desktop: ~
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~$ cd ~/catkin_ws/src/
```

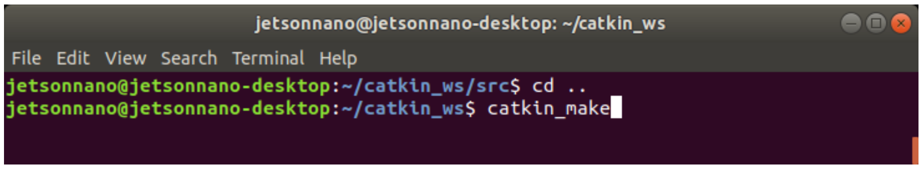
Clone the YDLidar_ROS remote github repository into your catkin workspace source folder using the following command.

git clone https://github.com/YDLIDAR/ydlidar_ros.git


jetsonnano@jetsonnano-desktop: ~/catkin_ws/src
File Edit View Search Terminal Help

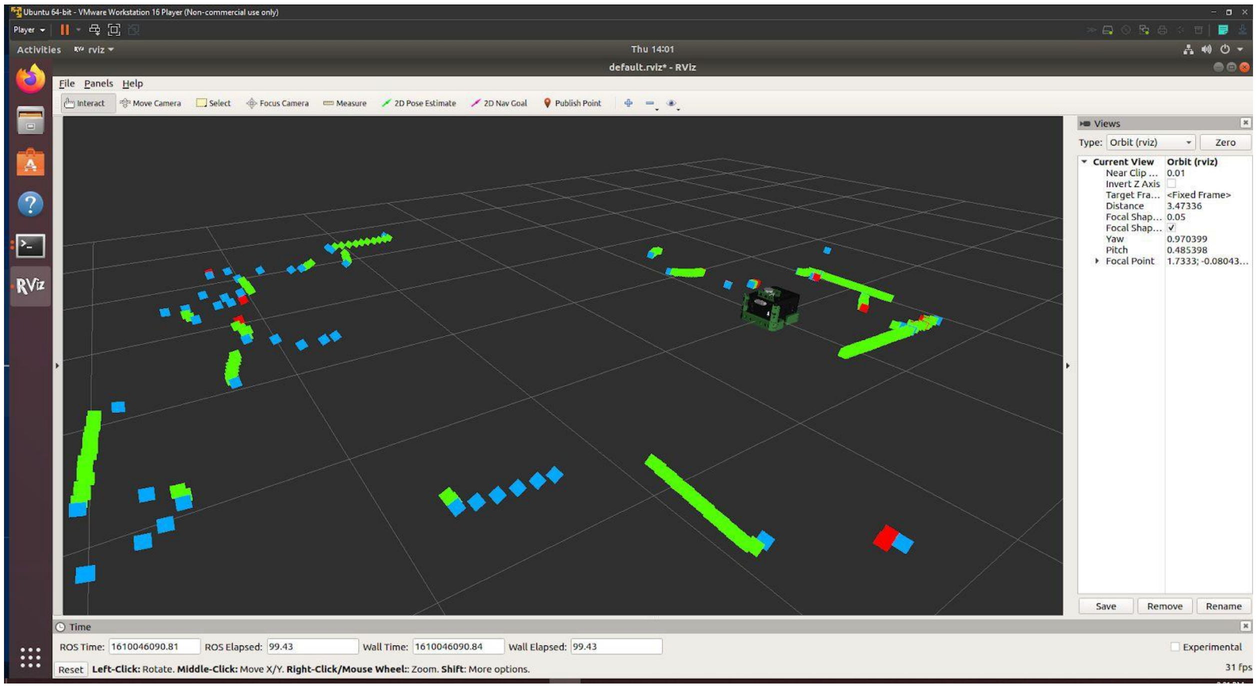
```
jetsonnano@jetsonnano-desktop:~/catkin_ws/src$ git clone https://github.com/YDLI
DAR/ydlidar_ros.git
```

Build the workspace by changing to the workspace root directory **~/<catkin_workspace_name>** and running **catkin_make**.


jetsonnano@jetsonnano-desktop: ~/catkin_ws
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~/catkin_ws/src$ cd ..
jetsonnano@jetsonnano-desktop:~/catkin_ws$ catkin_make
```

Setting Up the LiDAR Publisher Node



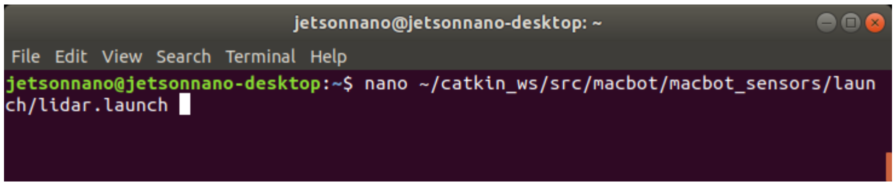
Type: Orbit (rviz) Zero

Current View	Orbit (rviz)
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	<Fixed Frame>
Distance	3.47336
Focal Shap...	0.05
Focal Shap...	✓
Yaw	0.970399
Pitch	0.485398
Focal Point	1.7333; -0.08043...

ROS Time: 1610046090.81 ROS Elapsed: 99.43 Wall Time: 1610046090.84 Wall Elapsed: 99.43 ☐ Experimental

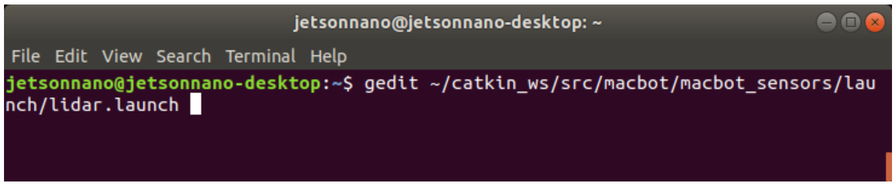
Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

Navigate to and open **~/catkin_ws/src/macbot/macbot_sensors/launch/lidar.launch**

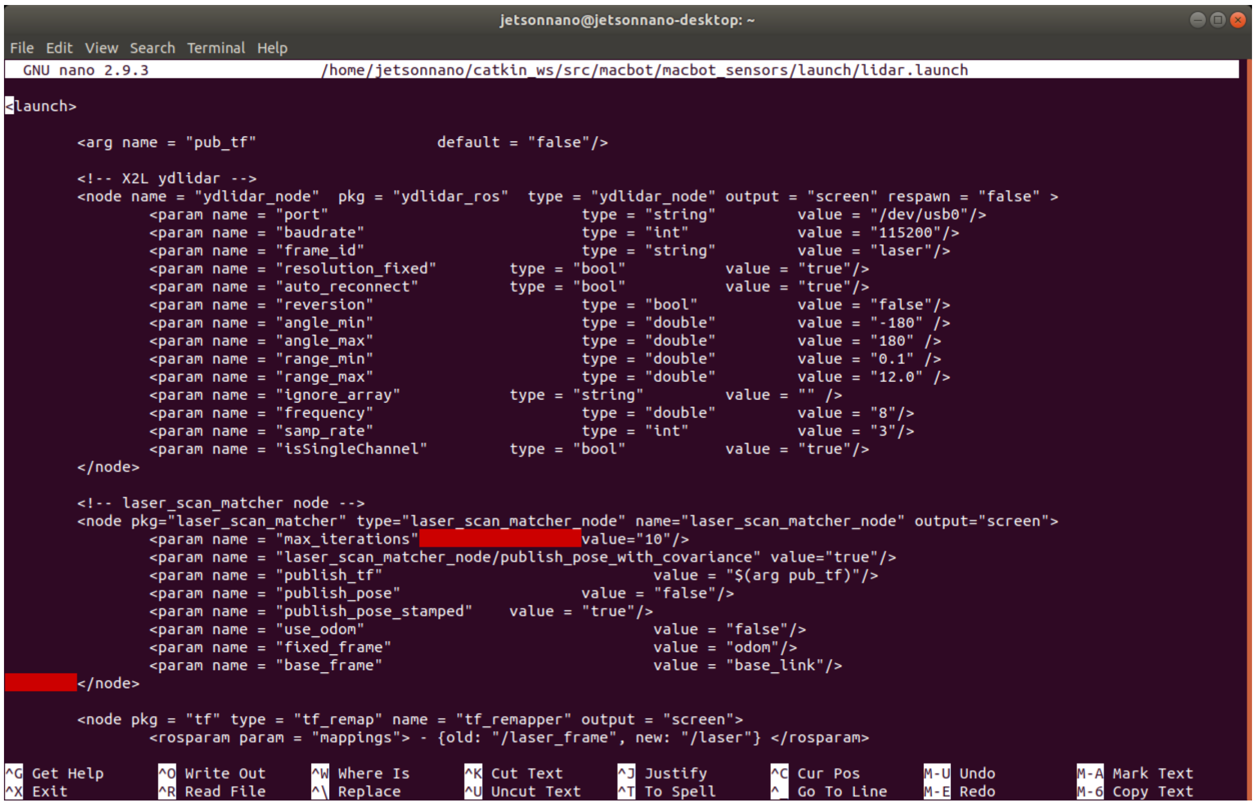

jetsonnano@jetsonnano-desktop: ~
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~$ nano ~/catkin_ws/src/macbot/macbot_sensors/laun
ch/lidar.launch
```

Alternatively, if a GUI is preferred GEDIT or VSCode are great options.


jetsonnano@jetsonnano-desktop: ~
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~$ gedit ~/catkin_ws/src/macbot/macbot_sensors/lau
nch/lidar.launch
```

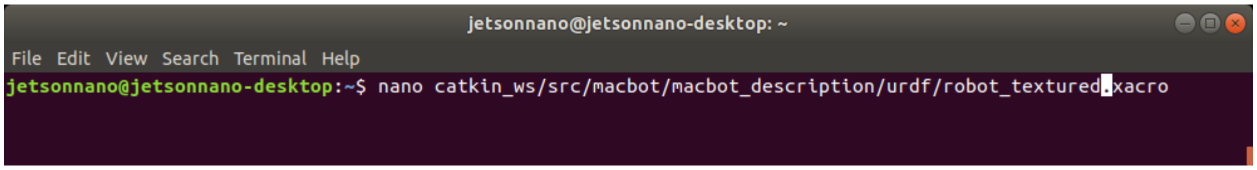

jetsonnano@jetsonnano-desktop: ~
File Edit View Search Terminal Help

```
  GNU nano 2.9.3              /home/jetsonnano/catkin_ws/src/macbot/macbot_sensors/launch/lidar.launch

<launch>

        <arg name = "pub_tf"                    default = "false"/>

        <!-- X2L ydlidar -->
        <node name = "ydlidar_node"  pkg = "ydlidar_ros"  type = "ydlidar_node" output = "screen" respawn = "false" >
                <param name = "port"                            type = "string"         value = "/dev/usb0"/>
                <param name = "baudrate"                        type = "int"            value = "115200"/>
                <param name = "frame_id"                        type = "string"         value = "laser"/>
                <param name = "resolution_fixed"        type = "bool"           value = "true"/>
                <param name = "auto_reconnect"          type = "bool"           value = "true"/>
                <param name = "reversion"                       type = "bool"           value = "false"/>
                <param name = "angle_min"                       type = "double"         value = "-180" />
                <param name = "angle_max"                       type = "double"         value = "180" />
                <param name = "range_min"                       type = "double"         value = "0.1" />
                <param name = "range_max"                       type = "double"         value = "12.0" />
                <param name = "ignore_array"            type = "string"         value = "" />
                <param name = "frequency"                       type = "double"         value = "8"/>
                <param name = "samp_rate"                       type = "int"            value = "3"/>
                <param name = "isSingleChannel"         type = "bool"           value = "true"/>
        </node>

        <!-- laser_scan_matcher node -->
        <node pkg="laser_scan_matcher" type="laser_scan_matcher_node" name="laser_scan_matcher_node" output="screen">
                <param name = "max_iterations"                  value="10"/>
                <param name = "laser_scan_matcher_node/publish_pose_with_covariance" value="true"/>
                <param name = "publish_tf"                              value = "$(arg pub_tf)"/>
                <param name = "publish_pose"                    value = "false"/>
                <param name = "publish_pose_stamped"    value = "true"/>
                <param name = "use_odom"                                value = "false"/>
                <param name = "fixed_frame"                             value = "odom"/>
                <param name = "base_frame"                              value = "base_link"/>
        </node>

        <node pkg = "tf" type = "tf_remap" name = "tf_remapper" output = "screen">
                <rosparam param = "mappings"> - {old: "/laser_frame", new: "/laser"} </rosparam>

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo       M-A Mark Text
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  M-E Redo       M-6 Copy Text
```

First, the YDLIDAR node must be added in our launch file. Since there are different models of LiDAR available from this company, the parameters must be set as outlined in the README.md file located in the YDLidar_ROS GitHub repository(https://github.com/YDLIDAR/ydlidar_ros).. The model provided is the **X2L LiDAR** and the parameters specific to this model must be configured according to the settings listed on the packages GitHub page One thing to keep in mind is that the **frame_id** of the LiDAR node is changed to **/laser** in order to have a functioning TF Tree. A transform tree is another way of describing transforms that influence each other and branch out in a tree hierarchy.

```
 <node name = "ydlidar_node"  pkg = "ydlidar_ros"  type = "ydlidar_node" output = "screen" respawn =
 "false" >
<param name = "port"                       type = "string"        value = "/dev/ydlidar"/>
 <param name = "baudrate"                  type = "int"           value = "115200"/>
 <param name = "frame_id"                  type = "string"        value = "laser"/>
 <param name = "resolution_fixed"          type = "bool"          value = "true"/>
 <param name = "auto_reconnect"            type = "bool"          value = "true"/>
 <param name = "reversion"                 type = "bool"          value = "false"/>
 <param name = "angle_min"                 type = "double"        value = "-180" />
 <param name = "angle_max"                 type = "double"        value = "180" />
 <param name = "range_min"                 type = "double"        value = "0.1" />
 <param name = "range_max"                 type = "double"        value ="12.0" />
 <param name = "ignore_array"              type = "string"        value = "" />
 <param name = "frequency"                 type = "double"        value = "8"/>
 <param name = "samp_rate"                 type = "int"           value = "3"/>
```

<param name = "isSingleChannel"   type = "bool"   value = "true"/>
</node>



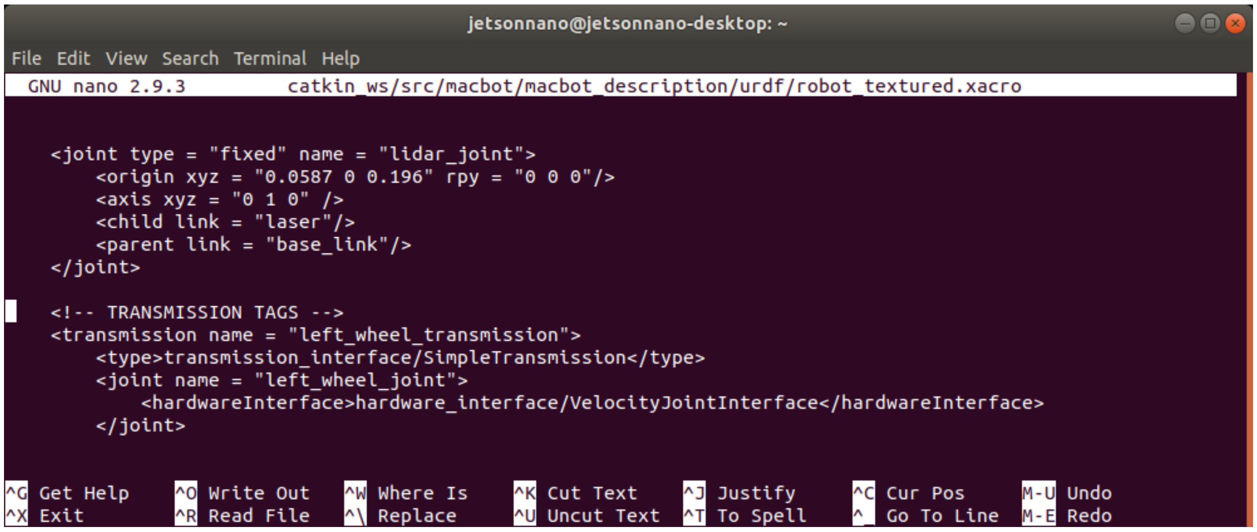
```
jetsonnano@jetsonnano-desktop:~$ nano catkin_ws/src/macbot/macbot_description/urdf/robot_textured.xacro
```

jetsonnano@jetsonnano-desktop: ~
File Edit View Search Terminal Help

And as a reminder, in the **macbot_description/urdf/robot_textured.xacro** file, the LiDAR child link is referred to as:

```
<joint type = "fixed" name = "lidar_joint">
        <origin xyz = "0.0587 0 0.196" rpy = "0 0 0"/>
        <axis xyz = "0 1 0" />
        <child link = "laser"/>
        <parent link = "base_link"/>
</joint>
```

If one is not familiar with URDF linkages, please refer to the ROS WiKi.
More info: http://wiki.ros.org/urdf/XML/joint
In short, they are the physical connections of the robot model that are actuated by joints. The **child** link points down the hierarchy of connections and the **parent** link points up the hierarchy.



```
  GNU nano 2.9.3         catkin_ws/src/macbot/macbot_description/urdf/robot_textured.xacro


    <joint type = "fixed" name = "lidar_joint">
        <origin xyz = "0.0587 0 0.196" rpy = "0 0 0"/>
        <axis xyz = "0 1 0" />
        <child link = "laser"/>
        <parent link = "base_link"/>
    </joint>

    <!-- TRANSMISSION TAGS -->
    <transmission name = "left_wheel_transmission">
        <type>transmission_interface/SimpleTransmission</type>
        <joint name = "left_wheel_joint">
            <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
        </joint>
```

jetsonnano@jetsonnano-desktop: ~
File Edit View Search Terminal Help
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line M-E Redo

The advertised transform MQTT namespace could potentially just be renamed, but it allows us the opportunity to learn about **TF remapping.** TF remapping alters which topic transform data will be communicated over. This can be done by doing the following:

```
<node pkg = "tf" type = "tf_remap" name = "tf_remapper" output = "screen">
        <rosparam param = "mappings"> - {old: "/laser_frame", new: "/laser"} </rosparam>
</node>
```

More info: http://wiki.ros.org/tf_remapper_cpp

Laser_scan_matcher is a ROS node that uses data from the laser scan messages to estimate the robot's position. The result is either a geometry_msgs::**Pose2D** or **tf** transform. This node requires the **base_link** → **laser** transform that was configured in the robot_textured.xacro file. It subscribes to **sensor_msgs/LaserScan** messages, which the LiDAR node is publishing on the **/scan** topic.

More info: http://wiki.ros.org/laser_scan_matcher

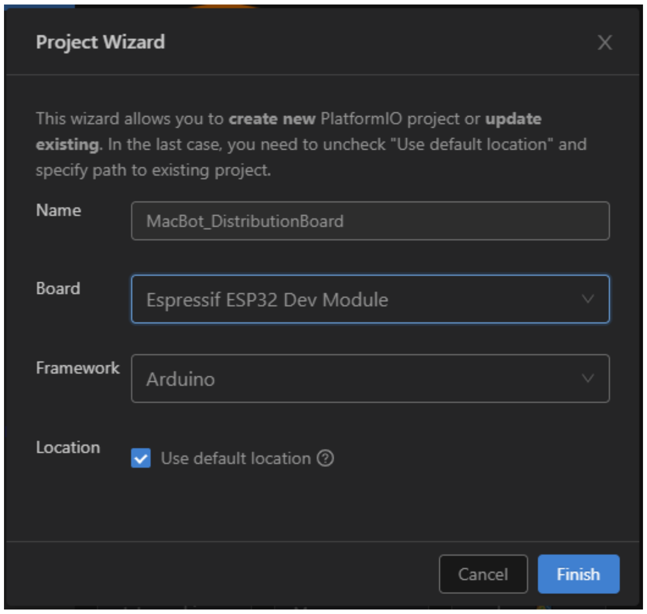
Setting Up the Motor Drivers and Distribution Board

CAN Bus is a message-based protocol that is typically used in vehicle systems. It allows for the communication between embedded devices without a master/slave relationship. Each node on the CAN bus can take on the responsibility of initializing a communication stream and act as a sender or receiver. On the MacBot, CAN is used in the communication between the Jetson Nano and the power distribution board. Two drive motors are then connected to this power distribution board. Because CAN bus is not native to the Jetson that is being used, a CAN gateway is placed as a translator between the Jetson Nano and the distribution board.

Motor Drivers → Distribution Board → CAN Gateway → Jetson Nano

Distribution Board

The role of the distribution board is to distribute power and communicate with devices. It powers the motors and handles position feedback from the encoders. The development environment that will be used to program the distribution board is PlatformIO. PlatformIO is installed on top of existing editors and includes support for a wide variety of microprocessors and well as access to a large repository of compatible libraries. However, in this case some custom libraries will be used that cannot be found using the library manager.


Project Wizard

This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.

Name: MacBot_DistributionBoard

Board: Espressif ESP32 Dev Module

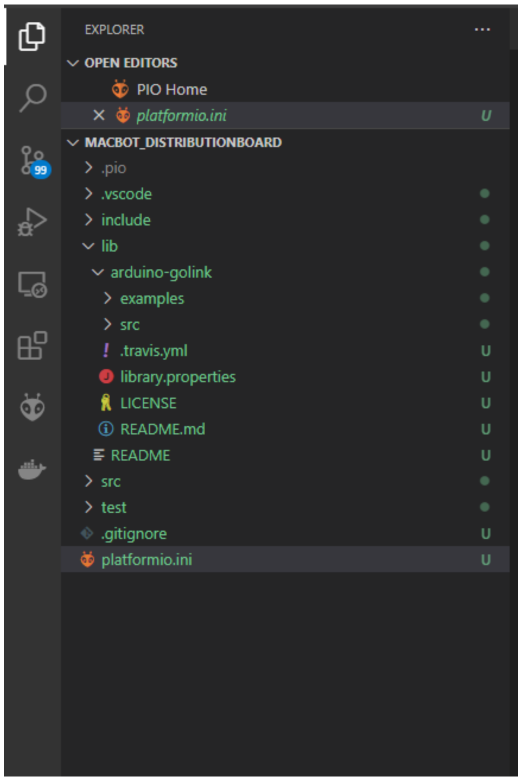
Framework: Arduino

Location: ✓ Use default location

Please add the following libraries to the PlatformIO project:

1. Arduino-GoLink (https://code.roboteurs.com/maciot-libs/arduino-golink)
2. Arduino-ESP32-Encoder (https://code.roboteurs.com/maciot-libs/arduino-esp32-encoder) ‘

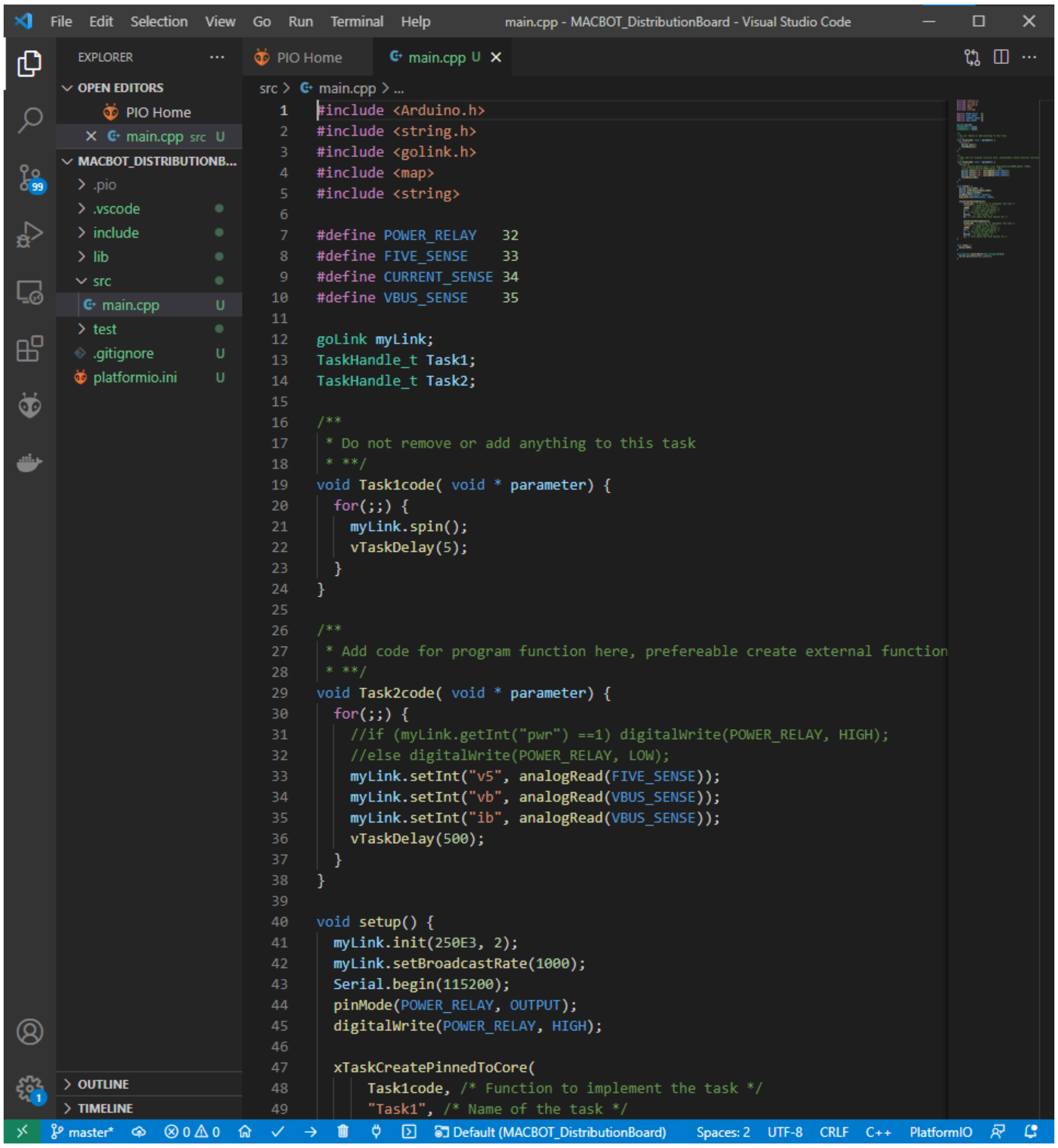
They are added by downloading and extracting each library. Then by dragging each library into the **<project_name>/lib/** folder.



The program that will be flashed to the microcontroller aboard the distribution board can be found here:

Distribution Board Code (https://code.roboteurs.com/snippets/10).

Connect to the board using USB-C.



```cpp
#include <Arduino.h>
#include <string.h>
#include <golink.h>
#include <map>
#include <string>

#define POWER_RELAY   32
#define FIVE_SENSE    33
#define CURRENT_SENSE 34
#define VBUS_SENSE    35

goLink myLink;
TaskHandle_t Task1;
TaskHandle_t Task2;

/**
 * Do not remove or add anything to this task
 * **/
void Task1code( void * parameter) {
  for(;;) {
    myLink.spin();
    vTaskDelay(5);
  }
}

/**
 * Add code for program function here, preferable create external function
 * **/
void Task2code( void * parameter) {
  for(;;) {
    //if (myLink.getInt("pwr") ==1) digitalWrite(POWER_RELAY, HIGH);
    //else digitalWrite(POWER_RELAY, LOW);
    myLink.setInt("v5", analogRead(FIVE_SENSE));
    myLink.setInt("vb", analogRead(VBUS_SENSE));
    myLink.setInt("ib", analogRead(VBUS_SENSE));
    vTaskDelay(500);
  }
}

void setup() {
  myLink.init(250E3, 2);
  myLink.setBroadcastRate(1000);
  Serial.begin(115200);
  pinMode(POWER_RELAY, OUTPUT);
  digitalWrite(POWER_RELAY, HIGH);

  xTaskCreatePinnedToCore(
      Task1code, /* Function to implement the task */
      "Task1", /* Name of the task */
```

Looking at the code, there is a **goLink** object called **myLink** that allows for communication with everything in our system.

When the myLink object is initialized, the bus speed must **ALWAYS** be **250E3**. A unique node ID must also be assigned. In this case, it is **node 2**. Every device on the CAN bus requires their own node ID so it can be addressed.

myLink.init(250E3, 2); myLink.setBroadcastRate(1000);

Next, the broadcast rate needs to be set. Since only basic diagnostic information like battery voltage is required from the distribution board, a high broadcast rate is not necessary. Set your broadcast rate to **1000 ms** (1s).

The code runs two concurrent tasks that are managed by the on-board RTOS. The first task, named **Task1code**, is pinned to one core and runs the **spin()** method. This method runs one messaging cycle. The second task is pinned to the other core on the dual-core ESP32 and is called **Task2code**. It sets 3 integer values to the queue that publishes to the CAN bus. Any node may have as many variables as they want but they must use different keys. However, different nodes may have the same variable as on a different node.

Ex. node 2 can have the can have a key called "v5" and node 8 can also have the same key "v5"

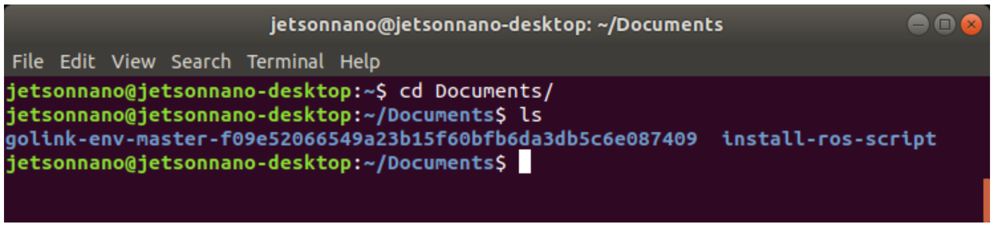
myLink.setInt("v5", analogRead(FIVE_SENSE));

myLink.setInt("vb", analogRead(VBUS_SENSE));

myLink.setInt("ib", analogRead(VBUS_SENSE));

CAN Bus

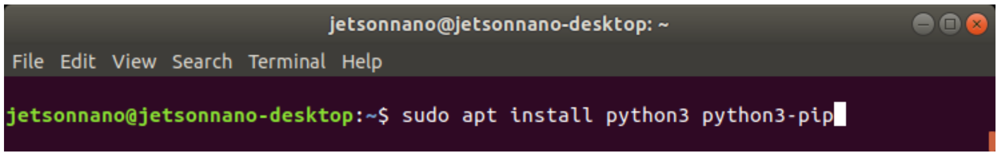
Navigate to an open directory and download the following: https://code.roboteurs.com/maciot-libs/golink-env.


jetsonnano@jetsonnano-desktop: ~/Documents
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~$ cd Documents/
jetsonnano@jetsonnano-desktop:~/Documents$ ls
golink-env-master-f09e52066549a23b15f60bfb6da3db5c6e087409  install-ros-script
jetsonnano@jetsonnano-desktop:~/Documents$ 
```

First, verify that python3 is installed on the macbot.

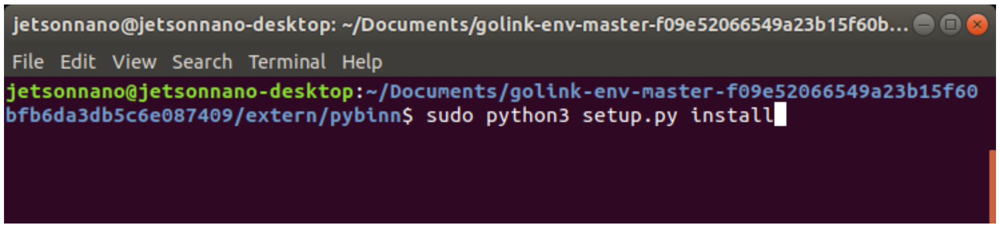
sudo apt-get install python3 python3-pip


jetsonnano@jetsonnano-desktop: ~
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~$ sudo apt install python3 python3-pip
```

Next, please navigate to **golink-env-master/extern/pybinn**. Manually install the python package using the following command:

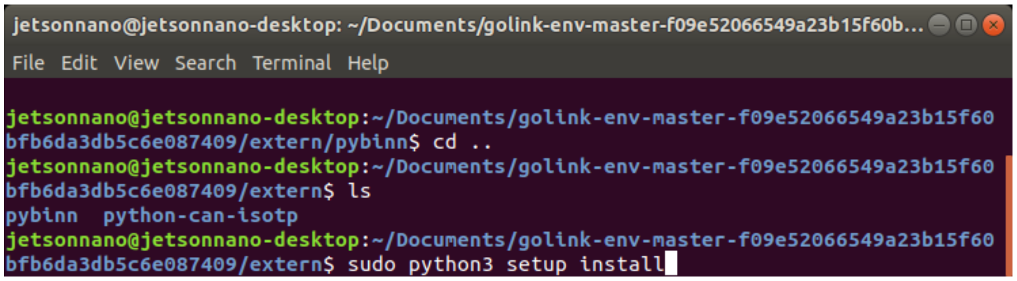
sudo python3 setup.py install


jetsonnano@jetsonnano-desktop: ~/Documents/golink-env-master-f09e52066549a23b15f60b...
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f60
bfb6da3db5c6e087409/extern/pybinn$ sudo python3 setup.py install
```

Next navigate to **golink-env-master/extern/python-can-isotp**. And repeat the install process:

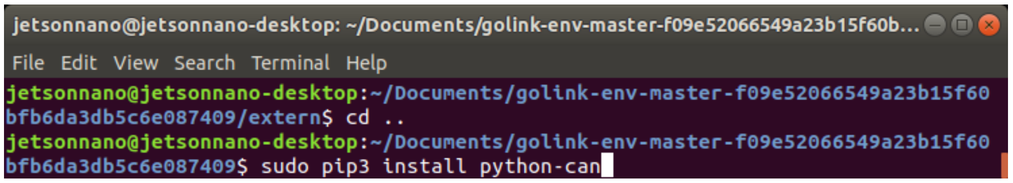
sudo python3 setup.py install


jetsonnano@jetsonnano-desktop: ~/Documents/golink-env-master-f09e52066549a23b15f60b...
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f60
bfb6da3db5c6e087409/extern/pybinn$ cd ..
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f60
bfb6da3db5c6e087409/extern$ ls
pybinn  python-can-isotp
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f60
bfb6da3db5c6e087409/extern$ sudo python3 setup install
```

Finally, install the python-can package using the pip3 package manager:

sudo pip3 install python-can

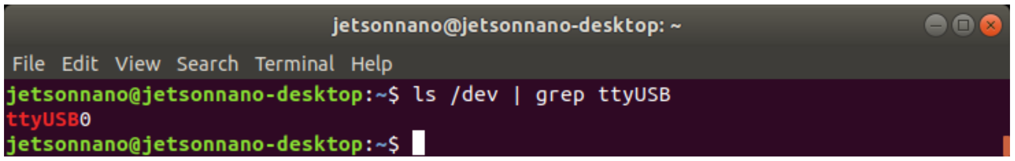

jetsonnano@jetsonnano-desktop: ~/Documents/golink-env-master-f09e52066549a23b15f60b...
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f60
bfb6da3db5c6e087409/extern$ cd ..
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f60
bfb6da3db5c6e087409$ sudo pip3 install python-can
```

Now that the CAN bus packages have been installed, navigate to **golink-env-master/utils**. Run the following command to verify the USB name:

ls /dev | grep USB

It will most likely read ttyUSB0. Linux represents devices as files on the filesystem. This simplifies the way that software exchanges information with devices.


jetsonnano@jetsonnano-desktop: ~
File Edit View Search Terminal Help

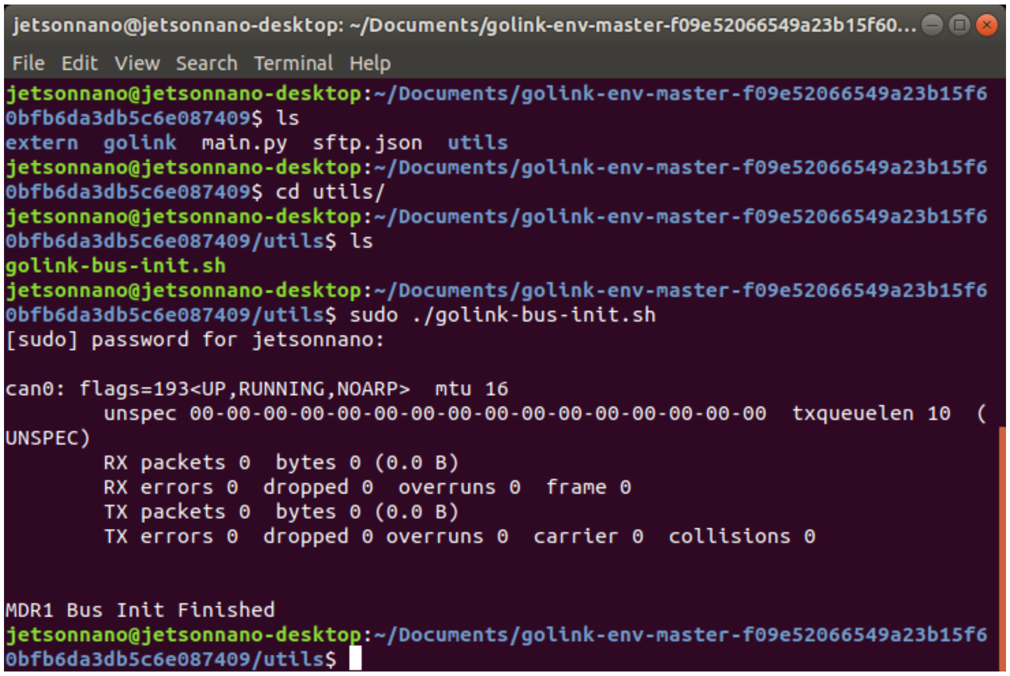
```
jetsonnano@jetsonnano-desktop:~$ ls /dev | grep ttyUSB
ttyUSB0
jetsonnano@jetsonnano-desktop:~$
```

Use nano to open the **golink-bus-init.sh** file and verify that the device reads:

/dev/ttyUSB0 (or whichever device file it is for you).

Afterwards, run the script with sudo privileges:

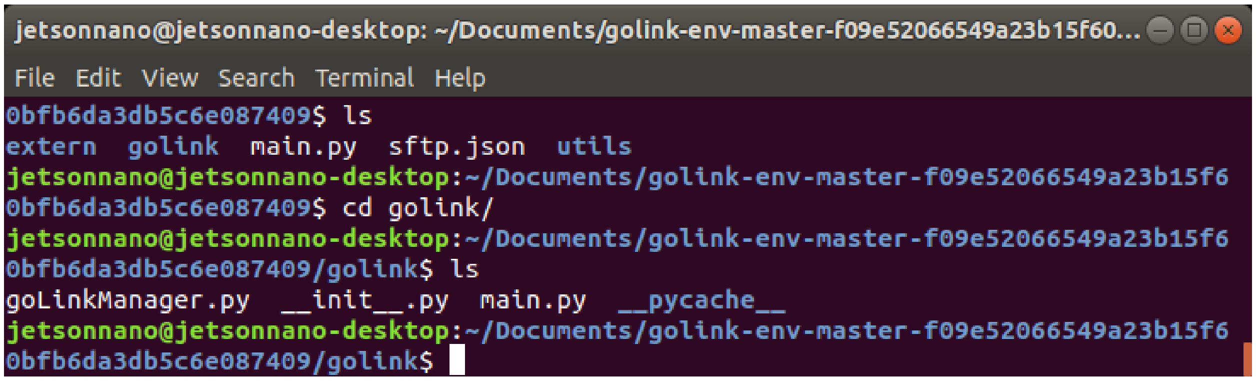
sudo ./golink-bus-init.sh


jetsonnano@jetsonnano-desktop: ~/Documents/golink-env-master-f09e52066549a23b15f60...
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f6
0bfb6da3db5c6e087409$ ls
extern  golink  main.py  sftp.json  utils
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f6
0bfb6da3db5c6e087409$ cd utils/
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f6
0bfb6da3db5c6e087409/utils$ ls
golink-bus-init.sh
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f6
0bfb6da3db5c6e087409/utils$ sudo ./golink-bus-init.sh
[sudo] password for jetsonnano:

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 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0


MDR1 Bus Init Finished
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f6
0bfb6da3db5c6e087409/utils$
```

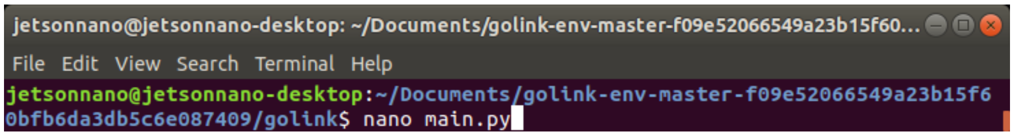
Now that CAN bus is up and running, navigate to the golink library.

The scripts **goLinkManager.py** and a **main.py** should be visible. Please focus your attention on main.py.


jetsonnano@jetsonnano-desktop: ~/Documents/golink-env-master-f09e52066549a23b15f60...
File Edit View Search Terminal Help

```
0bfb6da3db5c6e087409$ ls
extern  golink  main.py  sftp.json  utils
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f6
0bfb6da3db5c6e087409$ cd golink/
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f6
0bfb6da3db5c6e087409/golink$ ls
goLinkManager.py  __init__.py  main.py  __pycache__
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f6
0bfb6da3db5c6e087409/golink$
```

Edit **main.py** to write a short script that will print out the power values that were established in the distribution node.


jetsonnano@jetsonnano-desktop: ~/Documents/golink-env-master-f09e52066549a23b15f60...
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f6
0bfb6da3db5c6e087409/golink$ nano main.py
```

First, begin importing libraries:

import goLinkManager as glm

import time

import sys

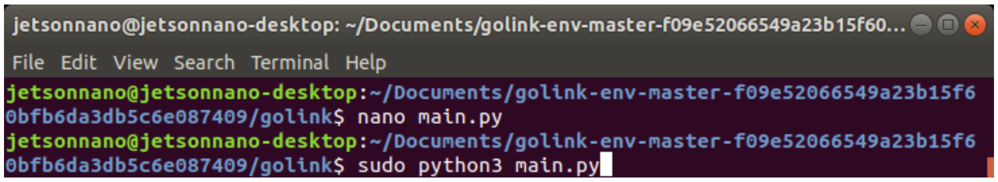
Next, add the main method:

```
 if  name          == " main ":
POWER_DIST = 2

systemNodeIds = [POWER_DIST]

man.glm.GoLinkManager(systemNodeIds) man.startNodes()

while 1:
if man.isNewData(POWER_DIST):
distDict = man.getData(POWER_DIST) print(distDict)
```

Here, the code is polling the POWER_DIST node. If new data is messaged, it will print that data.

Save, exit, and run the python script. Ensure that it is being run using Python 3.

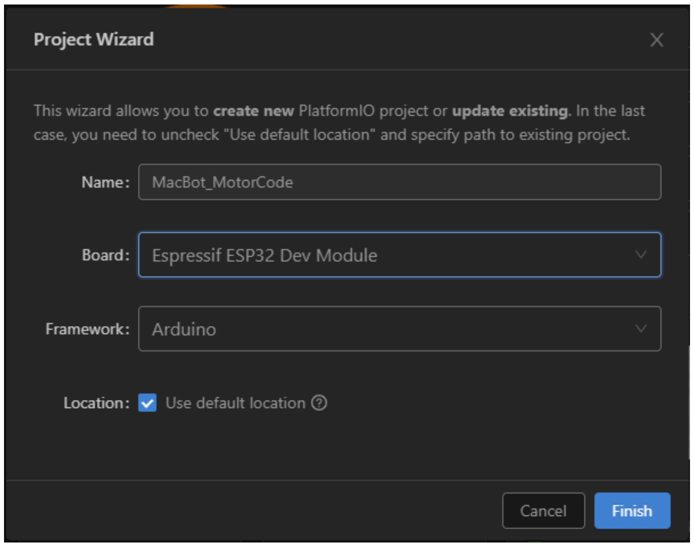
sudo python3 main.py


jetsonnano@jetsonnano-desktop: ~/Documents/golink-env-master-f09e52066549a23b15f60...
File Edit View Search Terminal Help

```
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f6
0bfb6da3db5c6e087409/golink$ nano main.py
jetsonnano@jetsonnano-desktop:~/Documents/golink-env-master-f09e52066549a23b15f6
0bfb6da3db5c6e087409/golink$ sudo python3 main.py
```

Motor Drivers

The motor driver code may be downloaded from https://code.roboteurs.com/snippets/11/raw. It must be uploaded to the motor drivers. Please direct your attention to the **Task2code** function. Notice the “**spr**” speed request variable, “**enc**” encoder variable, and lastly a “**spa**” speed actual variable.



Project Wizard

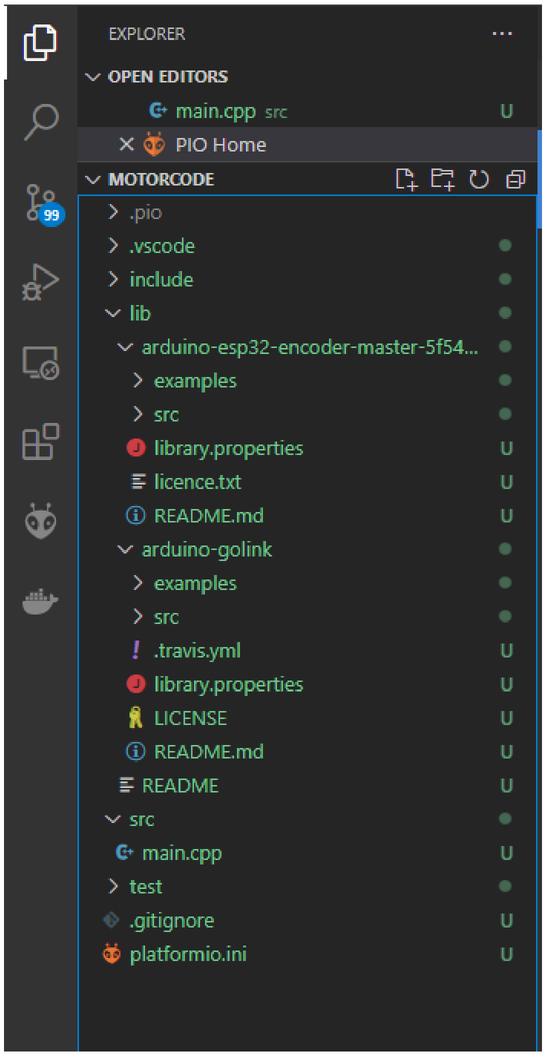
This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.

Name: MacBot_MotorCode

Board: Espressif ESP32 Dev Module

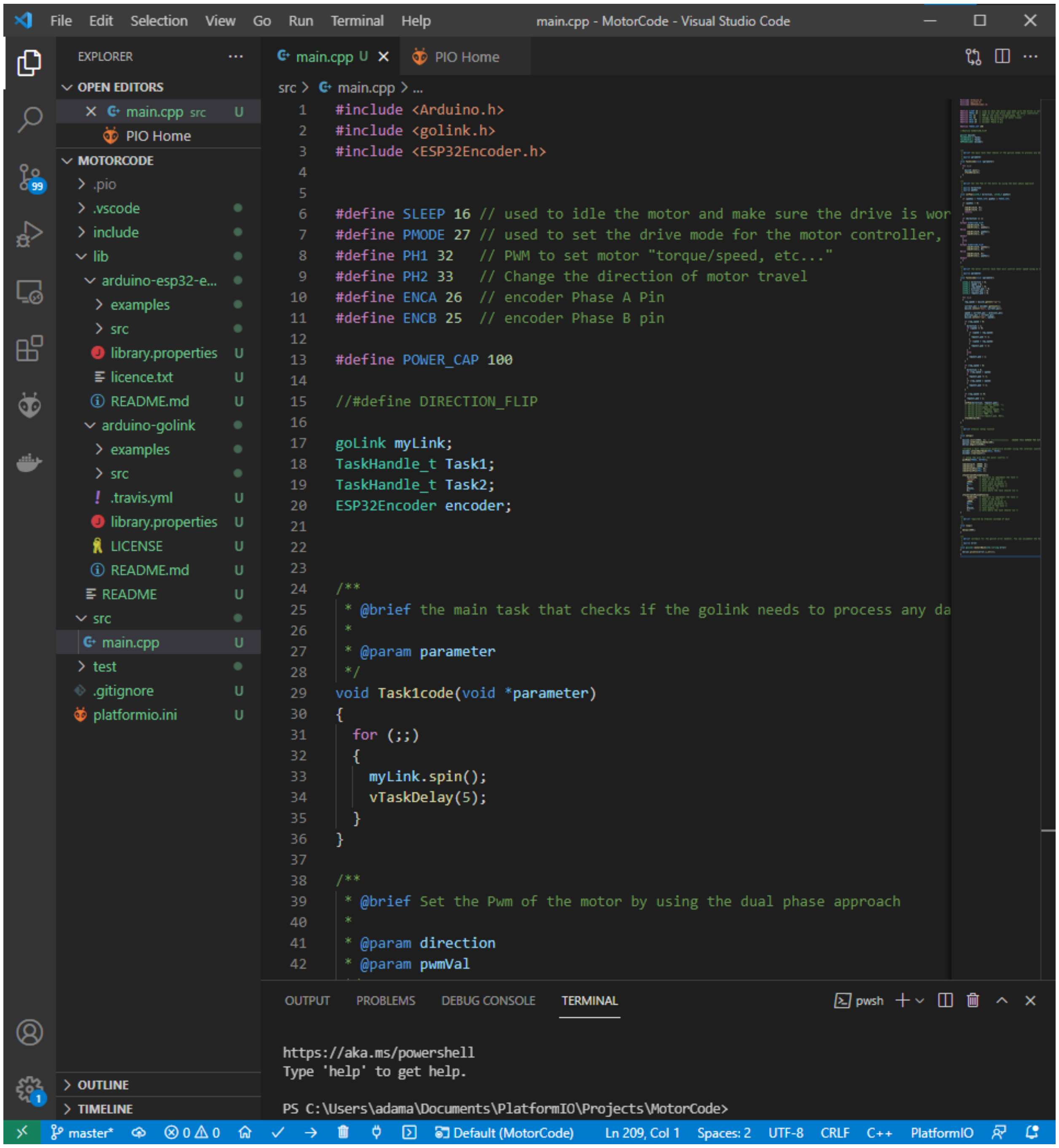
Framework: Arduino

Location: [✓] Use default location



```
EXPLORER
OPEN EDITORS
    main.cpp src                          U
  × PIO Home
MOTORCODE
  > .pio
  > .vscode
  > include
  v lib
    v arduino-esp32-encoder-master-5f54...
      > examples
      > src
        library.properties                U
        licence.txt                       U
        README.md                         U
    v arduino-golink
      > examples
      > src
        .travis.yml                       U
        library.properties                U
        LICENSE                           U
        README.md                         U
      README                              U
  v src
      main.cpp                            U
  > test
    .gitignore                            U
    platformio.ini                        U
```

Navigate to the setup function. Notice that the broadcast rate for myLink is set to **100 ms**. This ensures that the device is communicating with the distribution board and updating the drive motors more frequently. Set the CAN speed and node ID to that of the distribution board.



```cpp
#include <Arduino.h>
#include <golink.h>
#include <ESP32Encoder.h>


#define SLEEP 16 // used to idle the motor and make sure the drive is wor
#define PMODE 27 // used to set the drive mode for the motor controller,
#define PH1 32   // PWM to set motor "torque/speed, etc..."
#define PH2 33   // Change the direction of motor travel
#define ENCA 26  // encoder Phase A Pin
#define ENCB 25  // encoder Phase B pin

#define POWER_CAP 100

//#define DIRECTION_FLIP

goLink myLink;
TaskHandle_t Task1;
TaskHandle_t Task2;
ESP32Encoder encoder;



/**
 * @brief the main task that checks if the golink needs to process any da
 *
 * @param parameter
 */
void Task1code(void *parameter)
{
  for (;;)
  {
    myLink.spin();
    vTaskDelay(5);
  }
}

/**
 * @brief Set the Pwm of the motor by using the dual phase approach
 *
 * @param direction
 * @param pwmVal
```

```
https://aka.ms/powershell
Type 'help' to get help.

PS C:\Users\adama\Documents\PlatformIO\Projects\MotorCode>
```

Connect to the board using USB-C.

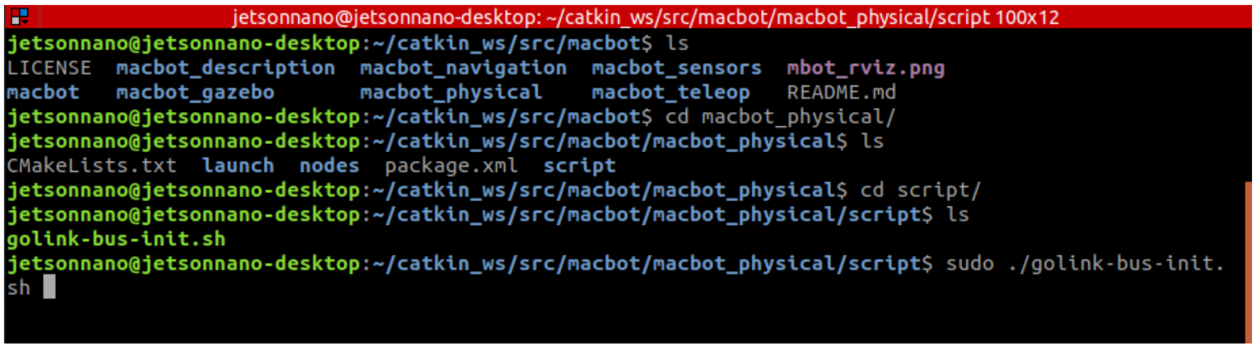
IMPORTANT - CHANGE THE NODE ID SO THAT BOTH MOTOR DRIVERS AREN'T ON THE SAME NODE IN THE LINE OF CODE: myLink.init(250E3, NodeID);

i.e. Left Motor Driver will have myLink.init(250E3, 4);

i.e. Right Motor Driver will have myLink.init(250E3, 5);

Once both the motor drivers and distribution code have been uploaded, jump back to the Jetson Nano. Every time the Jetson Nano is booted, it is necessary to initialize the CAN bus by running the **golink-bus-init.sh** script. Please do so from the file's directory. It will likely require root privileges to initialize correctly.

sudo ./golink-bus-init.sh



```
jetsonnano@jetsonnano-desktop: ~/catkin_ws/src/macbot/macbot_physical/script 100x12
jetsonnano@jetsonnano-desktop:~/catkin_ws/src/macbot$ ls
LICENSE  macbot_description  macbot_navigation  macbot_sensors  mbot_rviz.png
macbot   macbot_gazebo       macbot_physical    macbot_teleop   README.md
jetsonnano@jetsonnano-desktop:~/catkin_ws/src/macbot$ cd macbot_physical/
jetsonnano@jetsonnano-desktop:~/catkin_ws/src/macbot/macbot_physical$ ls
CMakeLists.txt  launch  nodes  package.xml  script
jetsonnano@jetsonnano-desktop:~/catkin_ws/src/macbot/macbot_physical$ cd script/
jetsonnano@jetsonnano-desktop:~/catkin_ws/src/macbot/macbot_physical/script$ ls
golink-bus-init.sh
jetsonnano@jetsonnano-desktop:~/catkin_ws/src/macbot/macbot_physical/script$ sudo ./golink-bus-init.
sh 
```

Back in **main.py**, quickly modify it so that the encoder values can be checked.

Next, add the motor driver code as follows:

if name   == " main ": POWER_DIST = 2

MOTOR_DRIVER_LEFT = 4

systemNodeIds = [PWOER_DIST, MOTOR_DRIVER_LEFT]

man = glm.GoLinkManager(systemNodeIds) man.startNodes()

while 1:

if man.isNewData(MOTOR_DRIVER_LEFT):

distDict = man.getData(MOTOR_DRIVER_LEFT) print(distDict)

Run the script using the following command:

python3 main.py

The following output will be seen:

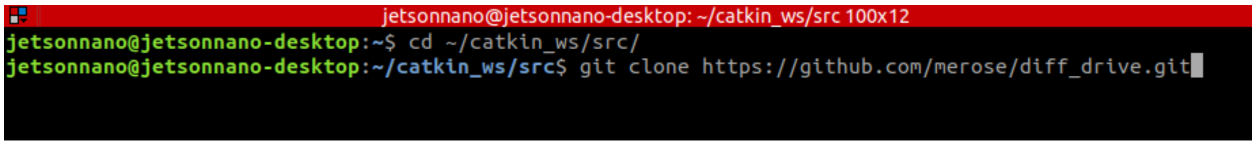
```
{'enc': 0, 'spa': 0}
{'enc': 0, 'spa': 0}
{'enc': 0, 'spa': 0}
{'enc': 3, 'spa': 0}
{'enc': 8, 'spa': 0}
```

Creating the ROS Python Node

The next step is to create a ROS Python node that controls the MacBot's drive motors. This requires accessing the CAN bus and running the differential drive ROS package (https://github.com/merose/diff_drive).

cd ~catkin_ws/src

git clone https://github.com/merose/diff_drive.git



jetsonnano@jetsonnano-desktop: ~/catkin_ws/src 100x12

```
jetsonnano@jetsonnano-desktop:~$ cd ~/catkin_ws/src/
jetsonnano@jetsonnano-desktop:~/catkin_ws/src$ git clone https://github.com/merose/diff_drive.git
```

cd ..

catkin_make

It is important to understand the communication structure this far:

teleop-twist-keyboard (cmd_vel) → diff_drive (diff_drive_controller) → macbot_node.py (custom) ↔ tf_broadcaster.py (custom) ↔ motor_drivers ↔ motors

Using the **teleop-twist-keyboard** node, **std_msgs::Twist** messages will be communicated over the **cmd_vel** topic. These are velocity vector messages that will be used to update the speed of the drive motors. The **diff_drive node** will subscribe to that **cmd_vel topic** and publish **std_msgs::Int32** messages to both the **lwheen_desired_rate** and **rwheel_desired_rate** topics. The custom macbot_node that was previously written will then **monitor those topics and message** the motor speed to the distribution board using a speed request (spr). Concurrently, the macbot_node will also be receiving encoder feedback from the motor driver. This data will be published to a topic. The **tf_broadcaster** will then subscribe to this same topic and **publish the transforms**.

These transforms can then be visualized using the many tools built into ROS.

Please navigate to the macbot_node ROS package and open **macbot_physical/nodes/macbot_node.py**.

The **shebang** is the first line in many python scripts. It determines where the python3 interpreter is located and allows the operating system to recognize it as a python script. This lets the user execute the script as a standalone executable, without needing to type python3. Because the GLM included library requires python3 for interfacing with CAN, the shebang will look like:

```
#!/usr/bin/env python3
```

The dependent libraries must be imported into the project:

```
#!/usr/bin/env python3 import rospy

import time

import goLinkManager as glm import signal

import sys

from std_msgs.msg import (Int32, Float32)
```

The various nodes on the CAN bus must then be defined:

```
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()
```

Next, a class called macbotMotor is defined. In the future, this class may be renamed to reflect including other nodes besides motors. Do not fret if the class is named otherwise.

```
class macbotMotor():

def  init (self, direction_wheel): self.direction = str(direction_wheel) self.direction_check()


self.subTopicRate = str(direction_wheel) + "_desired_rate" self.pubTopicTicks = str(direction_wheel) +
"_ticks" self.pubTopicRate = str(direction_wheel) + "_rate"


self.subRate = rospy.Subscriber(self.subTopicRate, Int32, self.callback) self.pubTicks =
rospy.Publisher(self.pubTopicTicks, Int32, queue_size = 1) self.pubRate =
rospy.Publisher(self.pubTopicRate, Float32, queue_size = 1)


# Drives motor_driver via CAN def callback(self, data):

man.setData(self.motor_driver, {'spr' : -data.data})


def direction_check(self):

if self.direction == "lwheel":

self.motor_driver = MOTOR_DRIVER_LEFT elif self.direction == "rwheel":

self.motor_driver = MOTOR_DRIVER_RIGHT


def publishWheelData(self):

if man.isNewData(self.motor_driver): self.distDict = man.getData(self.motor_driver)
self.pubTicks.publish(self.distDict['enc']) self.pubRate.publish(self.distDict['spa'])
```

Let us review the code. The **init** function acts as the constructor and initializes the class members after object creation. It performs a check to determine if the class is controlling the left or right drive motor on the **differential drive**. Next, topics are created for the node to publish and subscribe to. The name is determined by the **diff_drive_controller** node (**lwheel_desired_rate** and **rwheel_desired_rate**).

Upon subscribing to the wheel_rate topic, a **callback** is invoked that sends a **speed request**(spr) on the CAN bus, which instructs the **motor_driver** to innervate the motors.

Next is a function that publishes the **encoder**(enc) and **actual speed**(spa) of the drive motors.

In the main section of the code, the ROS node is initialized:

Rospy.init_node('macbot_pubsub', anonymous = True)

Both the left and right drive motor objects of class macbotMotor are then instantiated. A super loop then repeatedly publishes drive motor speed data.

Transform Broadcaster

This is the first example in this set of labs that works with **ROS transforms**. An important note is that this node will be executed as a Python 2 script rather than Python 3 (unlike the macbot_node). Because of this, the shebang will reference the Python 2 interpreter instead.

```
#!/usr/bin/env python

import rospy

import tf_conversions import tf2_ros

import geometry_msgs.msg from math import pi


from std_msgs.msg import (Int32)
```

A class called **macbot_tf_broadcaster** that will be publishing the **tf** for both drive motors. Whether it is the left or right drive motor must be passed into the initializer.

```
class macbot_tf2_broadcaster():

def  init (self, direction_wheel):

self.br = tf2_ros.TransformBroadcaster()

self.t = geometry_msgs.msg.TransformStamped() self.direction = str(direction_wheel)
self.direction_check()

self.t.header.frame_id = "base_link" self.t.child_frame_id = self.directionString + "_wheel"


def direction_check(self):
```

if self.direction == "lwheel": self.directionString = "left" self.t.transform.translation.x = 0.0753

self.t.transform.translation.y = 0.137

self.t.transform.translation.z = -0.004 elif self.direction == "rwheel":

self.directionString = "right" self.t.transform.translation.x = 0.0753

self.t.transform.translation.y = -0.137

self.t.transform.translation.z = -0.004

The **header frame** (or parent) will be the **base_link**. In the same way as in the **macbot_node**, a direction check must be done.

Because the link name in the **URDF.XACRO** is "**left_wheel**" or "**right_wheel**", **self.directionString** must be set as "**left**" or "**right**". Then the name of the child link must be called "**left_wheel**" or "**right_wheel**" when strings are added.

Some estimations must be done where the transform is published:

def publish_tf(self, msg):

$$\frac{1122\ [ticks\ per\ revolution]}{360 * number\ of\ revolutions} = \left(\frac{-msg.data}{1122.0}\right) - int\left(\frac{-msg.data}{1122}\right) * self.t.header.stamp$$
$$= rospy.Time.now()$$

self.q = tf_conversions.transformations.quaternion_from_euler(0, (360*revs)*(pi/180), 0)

self.t.transform.rotation.x = self.q[0] self.t.transform.rotation.y = self.q[1] self.t.transform.rotation.z = self.q[2] self.t.transform.rotation.w = self.q[3] self.br.sendTransform(self.t)

Because there are **~1122 ticks per revolution**, the current encoder position must be divided by 1122. This finds the number of revolutions. Then, the remainder is used to determine the fractional percentage (between 0 and 1) of the wheel position.

revs = (-msg.data/1122.0) - int(-sg.data/1122)

This resulting decimal value is then converted into Radians.

if name == " main ":

rospy.init_node('macbot_tf2_broadcaster')

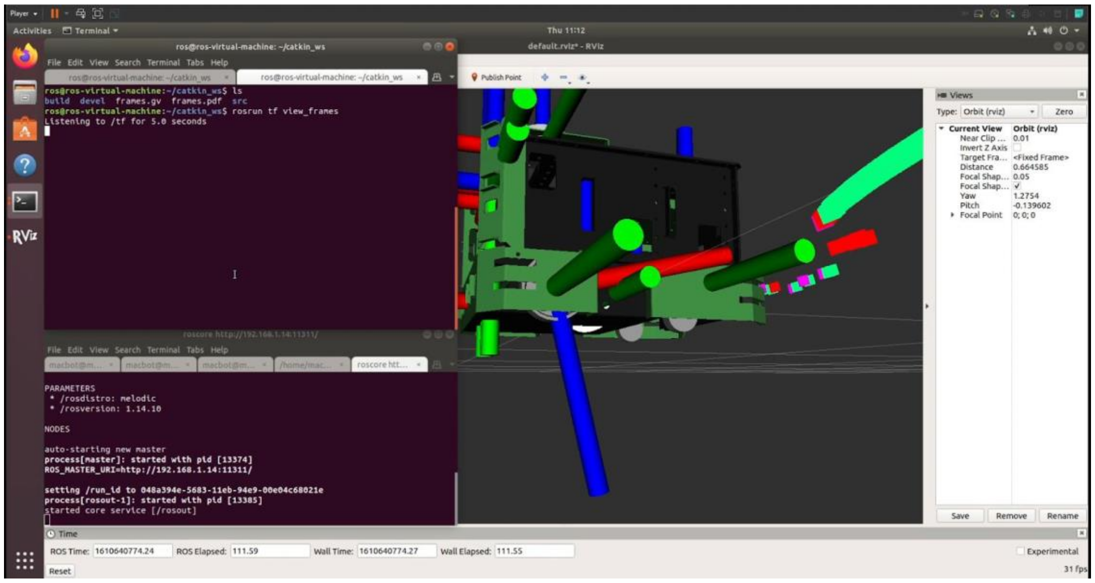
left_wheel_tf = macbot_tf2_broadcaster("lwheel") right_wheel_tf = macbot_tf2_broadcaster("rwheel") rospy.Subscriber("lwheel_ticks", Int32, left_wheel_tf.publish_tf) rospy.Subscriber("rwheel_ticks", Int32, right_wheel_tf.publish_tf)

rospy.spin()

In the main section, the node subscribed to “**lwheel_ticks”** or “**rwheel_ticks**” and the callback function is the **publish_tf** method from the **macbot_tf_broadcaster** class.

Visualizing Odometry Data

The following screenshots only show the left wheel for tf_broadcasting.



```
ros@ros-virtual-machine:~/catkin_ws$ ls
build  devel  frames.gv  frames.pdf  src
ros@ros-virtual-machine:~/catkin_ws$ rosrun tf view_frames
Listening to /tf for 5.0 seconds
```

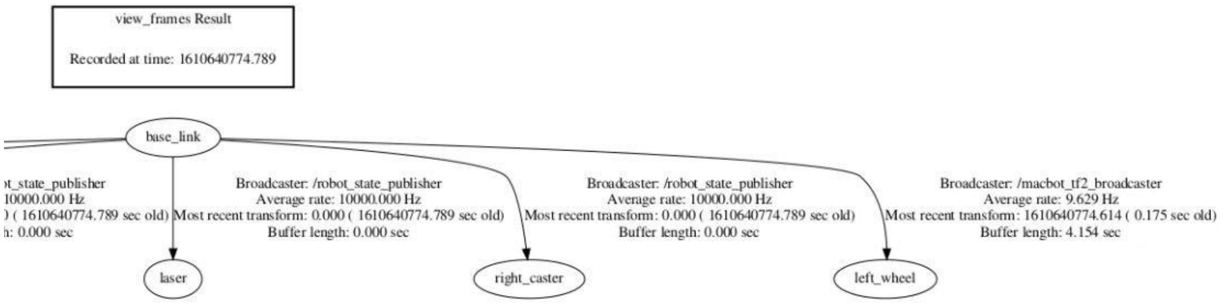
```
PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.10

NODES

auto-starting new master
process[master]: started with pid [13374]
ROS_MASTER_URI=http://192.168.1.14:11311/

setting /run_id to 048a394e-5683-11eb-94e9-00e04c68021e
process[rosout-1]: started with pid [13385]
started core service [/rosout]
```

Views — Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	<Fixed Frame>
Distance	0.664585
Focal Shap...	0.05
Focal Shap...	✓
Yaw	1.2754
Pitch	-0.139602
Focal Point	0; 0; 0

ROS Time: 1610640774.24 ROS Elapsed: 111.59 Wall Time: 1610640774.27 Wall Elapsed: 111.55

Player
Activities
Terminal
Thu 11:12
ros@ros-virtual-machine: ~/catkin_ws
default.rviz* - RViz
Publish Point
roscore http://192.168.1.14:11311/
File Edit View Search Terminal Tabs Help
Save
Remove
Rename
Time
Experimental
Reset
31 fps

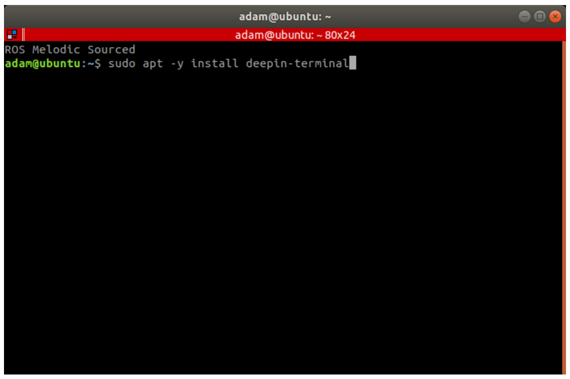


view_frames Result
Recorded at time: 1610640774.789
base_link
t_state_publisher
10000.000 Hz
) (1610640774.789 sec old)
h: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1610640774.789 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1610640774.789 sec old)
Buffer length: 0.000 sec
Broadcaster: /macbot_tf2_broadcaster
Average rate: 9.629 Hz
Most recent transform: 1610640774.614 (0.175 sec old)
Buffer length: 4.154 sec
laser
right_caster
left_wheel

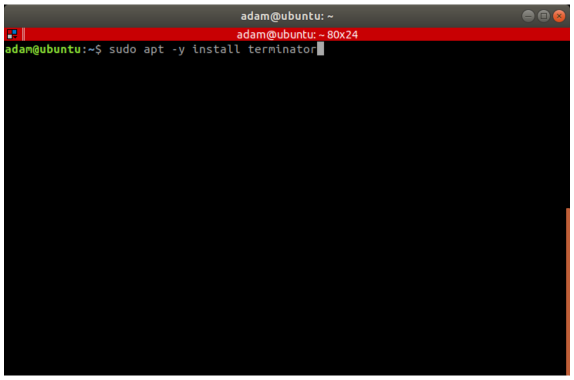
If everything is configured correctly, notice that the macbot_tf_broadcaster node is working correctly.

Multiple terminal windows will be required to run all of the required packages. This can be streamlined using a more advanced terminal emulator program. Two good options are deepin-terminal or terminator.

Both are available in apt. They both offer the ability to split the terminal window horizontally and vertically. Deepin-terminal offers tabs. These must be installed on the local machine that will SSH into the MacBot.

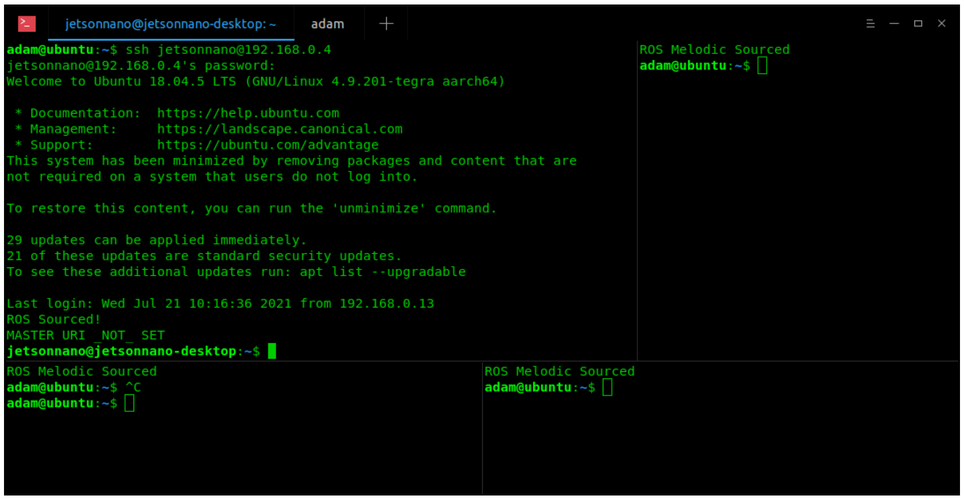


```
ROS Melodic Sourced
adam@ubuntu:~$ sudo apt -y install deepin-terminal
```



```
adam@ubuntu:~$ sudo apt -y install terminator
```

Be sure that the terminal windows are SSH'd into the MacBot.



```
adam@ubuntu:~$ ssh jetsonnano@192.168.0.4
jetsonnano@192.168.0.4's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.9.201-tegra aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

29 updates can be applied immediately.
21 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Wed Jul 21 10:16:36 2021 from 192.168.0.13
ROS Sourced!
MASTER URI  NOT  SET
jetsonnano@jetsonnano-desktop:~$
```

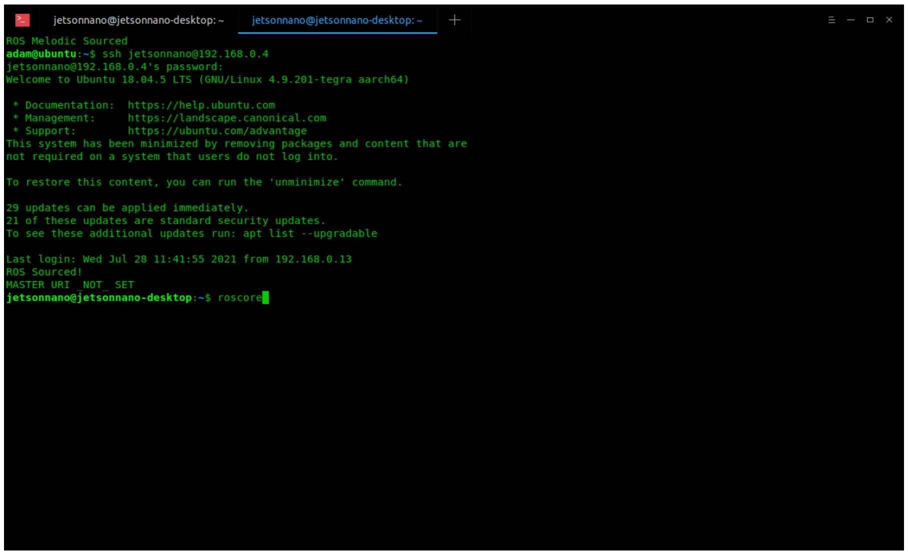
```
ROS Melodic Sourced
adam@ubuntu:~$
```

```
ROS Melodic Sourced
adam@ubuntu:~$ ^C
adam@ubuntu:~$
```

```
ROS Melodic Sourced
adam@ubuntu:~$
```

Now, the ROS packages are run on the MacBot (using SSH) and in the following order:

roscore



jetsonnano@jetsonnano-desktop: ~
jetsonnano@jetsonnano-desktop: ~

```
ROS Melodic Sourced
adam@ubuntu:~$ ssh jetsonnano@192.168.0.4
jetsonnano@192.168.0.4's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.9.201-tegra aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

29 updates can be applied immediately.
21 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Wed Jul 28 11:41:55 2021 from 192.168.0.13
ROS Sourced!
MASTER URI _NOT_ SET
jetsonnano@jetsonnano-desktop:~$ roscore
```

*In tab #2

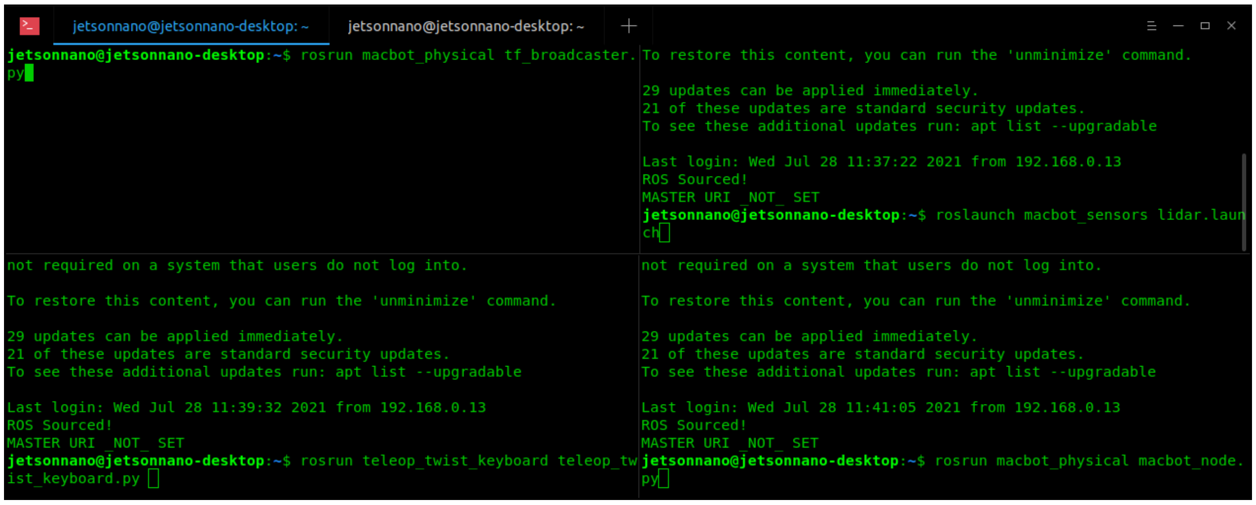
When roscore begins, run the following commands in order in the MacBot terminal to start the individual ROS nodes:

roslaunch macbot_sensors lidar.launch

rosrun teleop_twist_keyboard teleop_twist_keyboard.py

rosrun macbot_physical macbot_node.py

rosrun macbot_physical tf_broadcaster.py



jetsonnano@jetsonnano-desktop: ~
jetsonnano@jetsonnano-desktop: ~

```
jetsonnano@jetsonnano-desktop:~$ rosrun macbot_physical tf_broadcaster.
py
```

```
To restore this content, you can run the 'unminimize' command.

29 updates can be applied immediately.
21 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Wed Jul 28 11:37:22 2021 from 192.168.0.13
ROS Sourced!
MASTER URI _NOT_ SET
jetsonnano@jetsonnano-desktop:~$ roslaunch macbot_sensors lidar.laun
ch
```

```
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

29 updates can be applied immediately.
21 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Wed Jul 28 11:39:32 2021 from 192.168.0.13
ROS Sourced!
MASTER URI _NOT_ SET
jetsonnano@jetsonnano-desktop:~$ rosrun teleop_twist_keyboard teleop_tw
ist_keyboard.py
```

```
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

29 updates can be applied immediately.
21 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Wed Jul 28 11:41:05 2021 from 192.168.0.13
ROS Sourced!
MASTER URI _NOT_ SET
jetsonnano@jetsonnano-desktop:~$ rosrun macbot_physical macbot_node.
py
```

*In tab #1

Run **RViz** on VMware and add the **RobotModel**, **/scan, LaserScan,** and **TF**. On the teleop terminal tab, decrease the speed with “Z” to around **~0.09 m/s**. Next, drive the robot forward.

The wheels should spin based off of the **teleop_twist_keyboard** input.