Authors:
John McLinden
Contact: john_mclinden@my.uri.edu
Andrew Sullivan
Contact: amsullivan2@wpi.edu

Setting Up Your Raspberry Pi:

This initial project is running on Ubuntu 20.04 on a Raspberry Pi (Model 3B/3B+) and ROS Noetic. Previous attempts at following the official ROS beginner tutorials (http://wiki.ros.org/ROS/Tutorials) using Raspbian Buster ran into issues when trying to get the beginner tutorial software (though it is almost certainly feasible, see the tutorials at http://wiki.ros.org/ROSberryPi). We followed both the tutorials for ROS Kinetic and ROS Melodic, but both versions seemed to be more compatible with Ubuntu than with Raspbian. Some benefits using Ubuntu include:

- ROS officially targets Ubuntu 20.04
- It allows us to use ROS Noetic, the most recent version of ROS (07/22/2020)

Installing and configuring Ubuntu on the raspberry pi was done by following the tutorial found at: https://ubuntu.com/tutorials/how-to-install-ubuntu-on-your-raspberry-pi#1-overview. Information about setting up ROS Noetic for Ubuntu can be found at: http://wiki.ros.org/noetic/Installation/Ubuntu.

Below are the steps we took to configure ROS Noetic on a Raspberry Pi 3B/B+ running Ubuntu 20.04.

1. Using the Raspberry Pi imager (https://www.raspberrypi.org/downloads/), flash an Ubuntu 20.04 image to a microSD card (at least 8 GB).
2. Before attempting to boot the image on the pi, open the network-config file now found on the microSD card. There should be a section that looks like this:

```
#wifis:
#  wlan0:
#    dhcp4: true
#    optional: false
#    access-points:
#      <wifi network name>:
#        password: <"wifipassword">
```

   Uncomment the section and replace `<wifi network name>` with the name of your wifi network (may need to be in quotes if there is a space in the name). Replace `<”wifipassword”>` with your wifi password. Save the file, safely remove the microSD card, and insert it into the Pi to begin your first boot.

3. After booting, you will need to change the default password (default password is “ubuntu”).

4. (Optional, but required to run the beginner tutorials on your pi): Install a desktop environment. The environment we are currently using is lubuntu, as it is relatively lightweight, though still fairly slow.

   ```
   $ sudo apt install lubuntu-desktop
   ```

   Afterwards, you should be able to set up your Pi as normal ()
5. Setup your sources list to accept packages from packages.ros.org:

   ```
   $ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
   ```

6. Setup your keys:

   ```
   $ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
   ```

7. Ensure that everything is updated:

   ```
   $ sudo apt update
   ```

8. Install ROS Noetic. The desktop version is sufficient to run all of the beginner tutorials (some include rqt/rviz).

   ```
   $ sudo apt install ros-noetic-desktop
   ```

9. Edit ~./setup.bash file to allow for automatic sourcing of /opt/ros/noetic/setup.bash on every new shell:

   ```
   $ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc

   $ source ~/.bashrc
   ```

   From here, you should be able to follow the ROS beginner tutorials: http://wiki.ros.org/ROS/Tutorials. Be sure to configure the Pi for interfacing by following the steps in the next section.

Configuring Pi for Interfacing:

I2C- In order to properly access the I2C line on the Raspberry Pi with Ubuntu, here are a few steps that you need to take:

1. Install raspi-config. This is normally only intended for use in Raspbian, but appears to work in Ubuntu. This is also an older version of raspi-config. Details about this workaround to access the i2c bus are found here: https://askubuntu.com/questions/1130052/enable-i2c-on-raspberry-pi-ubuntu
   a. `$ wget https://archive.raspberrypi.org/debian/pool/main/r/raspi-config/raspi-config_20160527_all.deb -P /tmp`
   b. `$ apt-get install libnewt0.52 whiptail parted triggerhappy lua5.1 alsa-utils -y`
   c. `$ apt-get install -fy`
   d. `$ dpkg -i /tmp/raspi-config_20160527_all.deb`
   e. `$ sudo mount /dev/mmcblk0p1 /boot`

2. Run raspi-config:
   `$ sudo raspi-config`
   In the GUI, select “Advanced Interfacing Options”, and enable the I2C bus.
3. Create an I2C user group
   a. `$ sudo groupadd i2c`
4. Change the group’s ownership
   a. `$ sudo chown :i2c /dev/i2c-1`
5. Change file permissions
   a. `$ sudo chmod g+rw /dev/i2c-1`
6. Add your user to the I2C group
   a. `$ sudo usermod -aG i2c username`
   b. Replace `username` with your username
7. Reboot so the changes take effect
   a. Logging off and logging on again also works
   b. The goal is to get `i2cdetect -y 1` to work
8. Make the changes permanent
   a. `$ sudo su`
   b. `$ # echo 'KERNEL=="i2c-[0-9]*", GROUP="i2c"' >> /etc/udev/rules.d/10-local_i2c_group.rules`

This information was adapted from a tutorial that can be found at:
https://lexruee.ch/setting-i2c-permissions-for-non-root-users.html

GPIO- In order to use the GPIO pins on the Raspberry Pi with Ubuntu, here are a few steps that you need to take:

Python 3

1. `$ sudo apt-get update`
2. `$ sudo apt-get upgrade`
3. `$ sudo apt-get install python3-pip python3-dev`
4. `$ sudo pip3 install RPi.GPIO`

Python 2

1. `$ sudo apt-get update`
2. `$ sudo apt-get upgrade`
3. `$ sudo apt-get install python-pip python-dev`
4. `$ sudo pip install RPi.GPIO`

This information was adapted to from https://raspberrypi.stackexchange.com/questions/81570/installing-rpi-gpio-on-ubuntu-core, which just covers Python 2.

NOTE: It is important to note that any code that uses the GPIO pins will need to be run as the root user. For example:

Python 3

```
$ sudo python3 file.py
```

Python 2

```
$ sudo python file.py
```

However, with ROS, things are a bit more complicated. Using `rosrun` to run a script cannot be executed as a root user. As a workaround, we have used short scripts called with the `subprocess` library in our ROS node scripts when we need to access the GPIO library. An example of this can be found in the `led_receiver.py` script. Upon receiving a Boolean message on the topic `ledstuff`, the script calls another script, `led_blinker.py`, with the necessary root privileges to use the library and the received message as an argument (a cumbersome process that should be changed, if possible). Our current understanding of this issue is that the GPIO library is not intended for Ubuntu, so the permissions that are normally set automatically in Raspbian are not set during Ubuntu installation.

In Python, this work around can be accomplished using the subprocess library, including the `subprocess.call()` command. This command allows you to create and communicate with subprocesses. More information on the use of this command can be found at: https://pymotw.com/2/subprocess/ and https://queirozf.com/entries/python-3-subprocess-examples. Being able to spawn and communicate with other processes means that within one Python script, `subprocess.call()` can spawn another script, and, most notably, it can run it with root privileges using `sudo`. Running the script with root level privileges allows for the use of GPIO pins, which solves the problem we were encountering. More information about subprocesses is available under the **Subprocess Note** section.

Setting Up a Ubuntu Virtual Machine:

Using Ubuntu on a virtual machine provides a Linux environment to develop code and familiarity with Linux without the computing power limitations that a Raspberry Pi brings and without the potential damage that comes from dual-booting a PC for the first time. (Using a Linux machine or dual-booting Linux are best saved for more experienced users.) The Linux virtual machine also makes connecting to the Raspberry Pi easier.

The following instructions are adapted from the “Course Preparation Instructions” PDF available at https://rsl.ethz.ch/education-students/lectures/ros.html#course_material and the tutorial at https://medium.com/riow/how-to-open-a-vmdk-file-in-virtualbox-e1f711deacc4. The course materials that ETH Zurich provides include a YouTube series with accompanying presentations, exercises, and relevant material for learning ROS. These lessons are based in C++, but, the very first lecture video and accompanying exercise are very educational introductions, regardless of programming language.

1. Download the “Ubuntu_18_04_ROS_COURSE.zip” file from https://polybox.ethz.ch/index.php/s/iSwkLBhRJNKuv8a
    a. This file is about 5.6GB, so make sure you have room

2. Download VirtualBox from https://www.virtualbox.org/wiki/Downloads and install it
3. Open VirtualBox and click the button to create a new machine
4. Name your machine
    a. Take note of the destination listed after “Machine Folder:”
    b. If it does not do so automatically, change the “Type:” field to “Linux” and change the “Version:” field to “Ubuntu (64-bit)”
    c. Hit “Next” to continue
5. Stepping away from VirtualBox for a moment, unzip the “Ubuntu_18_04_ROS_COURSE.zip” folder to the destination that you noted in the previous step, and then pull VirtualBox back up to continue configuring your machine
6. Choose however much memory (RAM) you would like to be allocated to the virtual machine, and click “Next”
    a. The virtual machine runs fine with 1024 MB of memory (RAM), but the machine does tend to lag when you have multiple terminals and an internet browser open, so going higher is helpful depending on how you will use the machine
7. For a “Hard disk,” choose the “Use an existing virtual hard disk file” option, and use the file shaped button on the right.
8. From here, choose the “Add” button from the top left. This will open up your file explorer, where you can navigate to the destination the “Ubuntu_18_04_ROS_COURSE.zip” was extracted to in Step 5.
9. Choose the file named “Ubuntu_18_04_ROS_Course-cl1.vmdk” and press open
10. Click through any prompts that show up, and finally click the “Create” button to create the virtual machine
11. To run the virtual machine, select it from the menu on the left, and click the green “Start” arrow on the top menu bar.
    a. The default password for this image is “student”

ROS Tutorials:

The ROS tutorials, available at http://wiki.ros.org/ROS/Tutorials, are a great resource for getting started, and cover the use of both C++ and Python. (We opted for the Python route to allow for faster prototyping and ease of programming.) The tutorials walk you through configuring the environment, and introduce you to ROS’s Nodes, Topics, Messages, Publishers, Subscribers, and other fundamental core concepts of ROS. The scripts (specifically `talker.py` and `listener.py`) for these sections of the tutorials are actually the basis for what we used to create the LED Blinker.

Later on in the tutorial, .bag files are introduced. These files can be used to log and store data, whether it be commands, sensor data, or robot behavior. We found that this particular type of file could potentially be used to record behaviors and play them back to your robot later on, or even just extract the data. This information is covered in here and here.

LED Blinker:

Here is an example of a ROS talker and listener that takes a raw input from the user in the form of either `0` or `1`, and then turns the LED off or on respectively. To accomplish this, a string called `command_input` is defined as whatever the user puts in. (In the case of the current state of the code, only input values of 0 and 1 are valid.) Then, due to the nature of the `input()` command, the string is then converted to a `float`, which is then converted to a `bool`. This allows `led_reciever.py` to publish the user's input to the `led_command` topic, which is then read by the subscriber, `led_reciever.py`. The subscriber reads the message, and then, using `subprocess.call()` as outlined in the **Configuring Pi for Interfacing** section's note, calls a Python script, `led_blinker.py` using the Boolean message as an argument. This argument takes the form of `data.data` in this line of code:

```
subprocess.call(['sudo','python3','led_blinker.py','%s' % data.data])
```

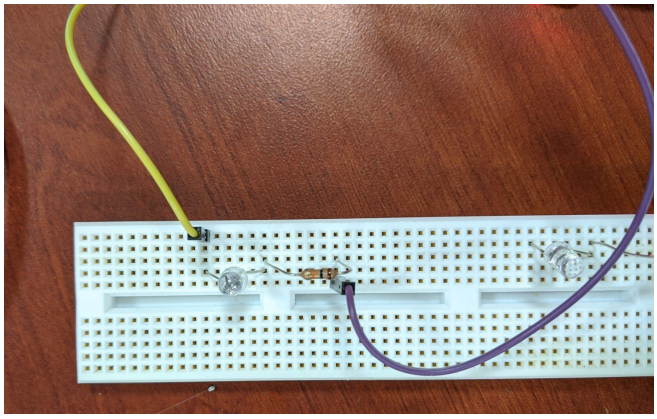
The (led_test) talker.py/listener.py Scripts

The talker.py/listener.py scripts are meant to be run in conjunction, similar to those found on the ROS tutorial site. To run these, in three separate shells, run

```
$ roscore
$ rosrun led_test talker.py
$ rosrun led_test listener.py
```

Assuming the bme280 is wired in accordance with the tutorial that the `talker.py` script is largely based on:
https://www.raspberrypi-spy.co.uk/2016/07/using-bme280-i2c-temperature-pressure-sensor-in-python/

The `talker.py` script should begin outputting the temperature as a string to the ledstuff topic, while the `listener.py` script should blink the LED (connected to pin 8 of the GPIO header) and print the topics read on the ledstuff topic. This script demonstrates an interface with an external sensor that is capable of sending/receiving sensor data in the ROS framework.



To set up the circuit (pictured above) for this example, use the output from GPIO pin 8 (purple wire) on the Raspberry Pi to power the LED, and use any of the Pi's ground pins to

ground it (yellow wire). Be sure to add a resistor between the power from the Pi and the LED. Refer to https://www.raspberrypi.org/documentation/usage/gpio/ for the pinout of the Raspberry Pi

The code for this example can be found on our GitHub: github.com/amichael1227/ros_led_test

Making the Package:

To get this ROS package to work properly, you must first run the command `catkin_create_pkg led_test std_msgs rospy roscpp` from the `~/catkin_ws` directory created in the beginner tutorials. From there, you should clone the GitHub repository into a *different folder*, and then copy the contents of that folder in the `~/catkin_ws/src/led_test` folder. Finally, change the directory back to `~/catkin_ws` and run the `catkin_make` command to build the package.

Subprocess Note:

The example below is a line of code within a listener that uses `subprocess.call()` to run a different Python script with root privileges, thus allowing for GPIO usage. (The `data.data` at the end sends an argument to the `led_blinker.py` program, and is talked about more under the **LED Blinker** heading). `subprocess.call(['sudo','python3','led_blinker.py','%s' % data.data])`

Another command in the subprocess library is `subprocess.Popen()`. Popen differs slightly from call, as it is a more general use function that does not wait for the process to finish to continue the original script. Call is useful in cases where the output of the called subprocess is required later on in the original script, while Popen allows the called script to run in the background without the original waiting for it to complete, which is useful in situations where a delay in listening could lead to missed messages. An example of Popen usage is available in the `led_receiver.py` script, where the `blink_test.py` script is called before calling the listener function. The led blinks 3 times while the listener initializes. If `subprocess.call()` were used, the listener would only initialize after the LEDs finished blinking. Aside from increased complexity, another drawback is that the ROS scripts that call these other scripts **MUST BE RUN FROM THE /scripts DIRECTORY** in the package to be able to locate the scripts that interact with the GPIO interface (i.e. cannot “rosrun” from just anywhere).

Pushing Changes to the Git from the Raspberry Pi:

This section is based on the tutorial found here:
https://uoftcoders.github.io/studyGroup/lessons/git/branches/lesson/

1. Fork the repository on github: https://github.com/amichael1227/ros_led_test
2. Clone it to a directory you’re comfortable working in:
   `$ git clone https://github.com/amichael1227/ros_led_test.git`
3. Create a new branch (make sure you’re in the directory you just cloned):
   `$ git checkout -b branchName`
4. Make your edits, and add your changes:
   `git add yourfilename` or `git add -A` to add all files.

5. Commit your changes
   `$ git commit -m "Your Message"`
6. Push the new branch to your forked version:
   `$ git push origin branchName`
7. Submit a pull request on GitHub. Assuming that it is accepted, merge with the main and delete the branch on GitHub and locally on the Pi:
   `$ git checkout master && git pull upstream master && git branch -d branchName`

NOTE: If you have multi-factor authentication enabled on your account, simply inputting your username and password will not work. GitHub has a tutorial at https://docs.github.com/en/github/authenticating-to-github/creating-a-personal-access-token about creating a personal access token. Follow the tutorial to create one, and then use the token as your password when prompted in the command line.

Running ROS on Multiple Machines:

The documentation for ROS provides a tutorial on running it across multiple machines at http://wiki.ros.org/ROS/Tutorials/MultipleMachines. When executed on two Raspberry Pis, both running Ubuntu 20.04, this tutorial worked without any major issues. One important step that needs to be taken in order for the two machines to work is letting ROS know what your IP address is. To check if ROS knows which IP to subscribe and receive on, run the `echo $ROS_IP` command. If this command prints out a blank block of text, you need to run `export ROS_IP=machine_ip_addr`, replacing `machine_ip_addr` with the IP address of your machine. Other than that, the tutorial provided in the ROS Documentation worked, as proven through our test of running the `led_controller.py` and `led_reciever.py` scripts on different Raspberry Pis, successfully blinking the LED.

To make following the tutorial easier, change the hostname of the Pi to allow you to use a name, just like `hal` and `marvin` in the tutorials, rather than typing in the typing in the IP addresses each time. To do this, run `sudo nano /etc/hostname` and replace the old name, likely `ubuntu`, with your desired hostname. From there, run `sudo nano /etc/hosts` and update any reference to the old hostname, if any, to reflect the new one. Then, run `sudo reboot`. The final step is to let ROS know the hostname, just like giving it the IP address in the paragraph above. Run `export ROS_HOSTNAME=your_hostname`, replacing `your_hostname` with the hostname you just assigned your machine.

References:

Here are websites and documents that were used as general references about ROS, Ubuntu, GitHub, and using the three together on a Raspberry Pi.

https://learning.oreilly.com/library/view/robot-operating-system/9781484234051/

This book provides a brief introduction to Linux operating systems, with most of the focus on Ubuntu, as well as some basic commands. It also covers the absolute basics of ROS with both C++ and Python.

https://gist.github.com/drmaj/20b365ddd3c4d69e37c79b01ca17587a

Covers building ROS Melodic with Python 3 rather than Python 2.7, was used during testing different ROS distros.

https://rsl.ethz.ch/education-students/lectures/ros.html#course_material

Course materials for a C++ centric ROS course from ETH Zurich, first lecture and exercise are very useful for general ROS intro, other lectures focus more on ROS with C++.

https://surfertas.github.io/ros/2017/03/06/ros-husky-robocup.html

Goes over building a .launch file for Exercise Session 1 of the above course website and provides an example solution.

https://answers.ros.org/question/253445/arduino-cmd_vel-and-odom-with-pololu-motors-w-encoder/

Example of using ROS with the motor drivers that are used on the UTAP daughterboard, only this example is run on and Arduino with C++ rather than a Raspberry Pi with Python, but still helpful for example integration

https://pythonprogramminglanguage.com/user-input-python/

Reviews commands to get raw inputs from users in Python 3 and gives example uses.

https://stackoverflow.com/questions/22990069/text-game-convert-input-text-to-lowercase-python-3-0

Covers how to change a user's raw input into all lower-case to eliminate code errors stemming from differences in capitalization. Simply change `.lower()` to `.upper()` for upper-case.

https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-vnc-on-ubuntu-18-04

Installing VNC on a Ubuntu system so it can be used when you do not have access to a monitor, mouse, and keyboard for the Raspberry Pi.

https://linuxize.com/post/how-to-enable-ssh-on-ubuntu-18-04/

Enabling SSH on a Ubuntu system so it can be used when you do not have access to a monitor, mouse, and keyboard for the Raspberry Pi.

https://www.liberiangeek.net/2013/09/copy-paste-virtualbox-host-guest-machines/

Setting up VirtualBox to allow for copy and paste between host and guest machines.

https://www.howtogeek.com/341944/how-to-clone-your-raspberry-pi-sd-card-for-foolproof-backup/

Making a copy of the image of Ubuntu that runs on the Raspberry Pi to allow for backups and cloning the SD card.

https://www.raspberrypi.org/forums/viewtopic.php?t=36856
https://unix.stackexchange.com/questions/118716/unable-to-write-to-a-gpio-pin-despite-file-permissions-on-sys-class-gpio-gpio18
https://stackoverflow.com/questions/30938991/access-gpio-sys-class-gpio-as-non-root

These three links cover the GPIO issue that we were having, and show other people having the same issues, and how they worked around the issue.

https://uoftcoders.github.io/studyGroup/lessons/git/branches/lesson/

Using GitHub on Linux within a terminal.

http://wiki.ros.org/ROS/NetworkSetup

Goes over different parts of configuring ROS to connect to a network, thus allowing you to communicate with ROS across multiple machines.

https://www.cyberciti.biz/faq/ubuntu-change-hostname-command/

Changing the hostname of a Ubuntu system, useful for the above configuration.

https://ieeexplore.ieee.org/document/6107001

IEEE paper on implementing ROS on the Yellowfin AUV.

https://ieeexplore.ieee.org/document/8729755

IEEE presentation report on using ROS on a REMUS 100 AUV, references the Python based libraries and packages that they built to interface with the AUV, however these packages and libraries do not seem to be publicly available.