Duckiebot Setup Guide

Gregory Grebe

Shareable link:
https://docs.google.com/document/d/1J0EYZkcoZcbjndBVh_g7ouEWFcPu9WZ6be_LVghDoII/edit?usp=sharing

Disclaimer: This guide is a work in progress. It's contents may change slightly as we work through setup.
Please let me know if something does not work for you, or if you had trouble but found a way to fix it, so that I can update these instructions.

Contents

Part List

The following will be included in you Duckiebot Kit.

- Magician Chassis (includes chassis body, wheels, and motors)
- Battery
- Raspberry Pi 2, Model B
- Pi Camera (Fish Eye Lens)
- Camera Mount and Case (Fixed angle)
- WiFi USB Adapter
- DC+Stepper Motor Pi Hat
- Servo/PWM Pi Hat
- 32 GB MicroSD card + Adapter
- 32GB USB Flash Drive (For data logging / additional storage)
- USB Splitter
- USB to Barrel Adapter
- Rubber Duckie Driver / Mascot
- (16x) 12mm Nylon Standoffs (M2.5 size)
- (4x) 5mm Nylon Standoffs (M3 size)
- (4x) Nylon Nuts (M2.5 size)
- M/M Jumper Cable Pack
- F/F Jumper Cable Pack

Please check your part kit upon receiving it, and make sure you have all of the parts.
Please **DO NOT** throw out any of the boxes or packaging that came with your parts. Keep them in the box for now.

Computer Setup

This section is based on the following documents: Setup Step 1.8 Setting Up Ubuntu Laptops and the Duckietops, Setup 1.9 - Github basics

Tips: if you select text and press middle click or ctrl-C, and then press middle click on the console, you will do copy and paste, that could save you a lot of time.
Also when writing on console, you can use the tab key to autocomplete code.

You **MUST** have Ubuntu 14.04 LTS (Trusty). The version of ROS we are using will not run on the newly released 16.04 (Xenial).
Note: that this section currently assumes you already have an Ubuntu 14.04 machine (either physical or virtual) set up.

Setup Script

Start by downloading the following setup script which will install most of the requirements for you:

```
$ wget
https://raw.githubusercontent.com/rpiRobotics/duckietown/master/setup/duckietop_setup.sh
```

And then run it using:

```
$ source duckietop_setup.sh
```

This script may take ~30-40 minutes to complete and you may need to answer prompts during the setup.

Github and the Duckietown Software Repository

The code for duckietown is all located in a github software repository. To download the repo you will therefore need a Github account.

If you do not already have an account please set one up (we don't care what email you use). Be sure to verify your account by email before continuing.

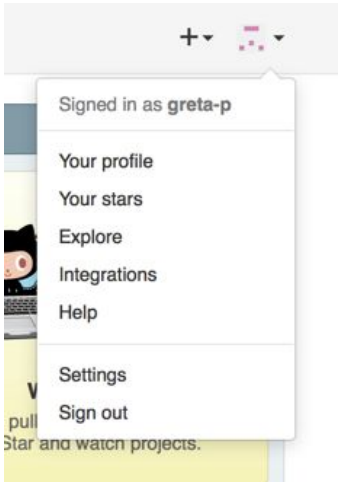
We will only briefly discuss github basics and practices in this document, but you are encouraged to read the rest of the duckietown Github basics guide or follow githubs tutorials after creating your account to become more familiar with git.

SSH Keys

Note:
This section is **not required** (you could simply enter a username and password each time), however **it will save you some time and make things easier for you.** We will assume that you have set up this key so keep in mind that if you choose not to, some of the later instructions may be slightly different for you.

We are now going to create an SSH key so that github knows it can talk to your computer without requiring you to enter your github username and password each time.

Go to settings



SSH and GPG keys
New SSH key

Choose a title for the key (i.e. Greg's Laptop SSH Key) and leave this window open, we will get back to the "key" box..

Now open a terminal on your laptop and generate the ssh key using
$ ssh-keygen -h
You will be asked to name the ssh key-file and enter a password.
The file name should be something like: ~/.ssh/<username>@<computer name>
I.e. copy what you see in the terminal before the $
The password should be left blank (just hit enter)

Example (you type the parts in orange):
$ ssh-keygen -h
Generating public/private rsa key pair.
Enter file in which to save the key (/home/**ubuntu**/.ssh/id_rsa):
/home/.ssh/greg@greg-Ubuntu

```
Enter passphrase (empty for no passphrase): <press enter>
Enter same passphrase again: <press enter>
Your identification has been saved in /home/greg/.ssh/greg@greg-Ubuntu
Your public key has been saved in /home/greg/.ssh/greg@greg-Ubuntu
The key fingerprint is:
4c:f8:c4:72:57:ba:ce:f1:e6:97:13:0d:27:39:43:ef greg@greg-Ubuntu
The key's randomart image is:
+--[ RSA 2048]----+
|          .    |
|      o   o  .  |
|     o = o  . o |
|      B . .  * o|
|       S o    O |
|        o o  . E|
|         o o  o |
|          o  +  |
|           .. . |
+-----------------+
```

Once the key has been generated we copy the public key file
Display it on your terminal using

```
$ cat ~/.ssh/<name>@<computer>.pub
```

Then select the text and copy it, and paste it into the “key” area on the github page and press **add key**.

The key is now created, but it is greyed out, so we need to authenticate it.
We first need to edit the ~/.ssh/config file. Any text editor can be used but I will assume we are using nano.

```
$ nano ~/.ssh/config
```

Now add the following line

```
IdentityFile ~/.ssh/<name>@<computer>
```

And press

ctrl+x → y → enter

To save and exit nano.

Now to authenticate we do

```
$ ssh -T git@github.com
```

And you should see something similar to the following

Warning: Permanently added the RSA host key for IP address '192.30.252.128' to the list of known hosts.

Hi greg! You've successfully authenticated, but GitHub does not provide shell access.

If you refresh the github page, your key should have now turned green.

Global Settings

On your computer we are going to configure git with the following global settings

$ git config --global user.email “<email>”

$ git config --global user.name “<name>”

$ git config --global push.default simple

Where <email> and <name> should be the email address and name you would like to appear in the repository’s history (note that this will be publicly available).

Clone the Duckietown Repo

We will now clone (copy) the duckietown repository to your local computer.

Note: Please notice that we are using the rpiRobotics fork of the repo NOT the MIT repo… we don’t have access to edit the MIT repo, and it is still actively under development and is somewhat unstable

$ git clone git@github.com:rpiRobotics/duckietown.git duckietown

This is a large repo so it may take several minutes to fully checkout.

Once this finishes, we will need to run the following commands to install several additional dependencies.

$ cd ~/duckietown

$ source duckietown_install_car.sh

$ source duckietown_install_laptop.sh

Set up ROS environment

We will now make the workspace. First we need to source the ROS indigo environment.

$ source /opt/ros/indigo/setup.bash

Then, we can build the catkin workspace (for those of you familiar with Cmake or similar make files, catkin workspaces are essentially a file structure that is set up to be built using the catkin

make command, and is used extensively for ROS). We must be under the catkin workspace folder to invoke the make command.

```
$ cd ~/duckietown/catkin_ws
$ catkin_make
```

You should see a lot of output as the source code files are all built. If your build finishes with an error, try running catkin_make one more time. Some of the dependencies may have just been messed up.

Now we can set up your ROS environment in your ~/.bashrc so that you don't need to source so many files every time you start a terminal session.
Open the ~/.bashrc file in your editor of choice, i.e….

```
$ nano ~/.bashrc
```

Make sure the following lines are in your .bashrc file and if not add them at the end of the file

```
source ~/duckietown/environment.sh
source ~/duckietown/set_ros_master.sh
export ROSLAUNCH_SSH_UNKNOWN=1
```

This last line means that by default your laptop will set itself as the ROS_MASTER.
Once you save and close the file, remember to source your .bashrc for the changes to take effect in your terminal session.

```
$ source ~/.bashrc
```

You will not need to do this from now on when you open a new terminal window, as the .bashrc file is automatically sourced.

Check that everything is ok by running the following

```
$ echo $ROS_MASTER_URI
```

You should see the following output

```
http://<your-computer>.local:11311/
```

Reboot your computer

```
$ sudo reboot
```

At this point, your laptop is set up!

Duckiebot Setup

Soldering the Pi Hat Circuit Boards

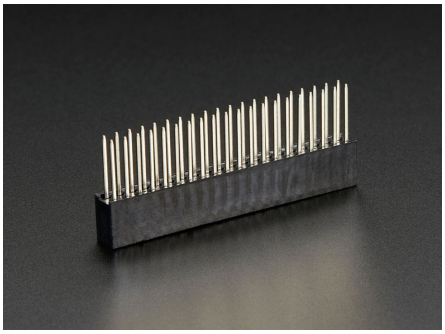
This section is based on Soldering Boards

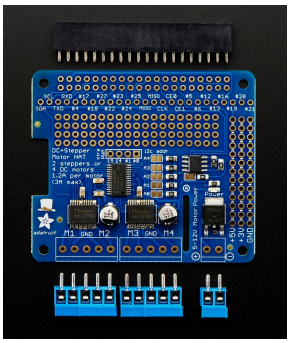
If you do not feel comfortable with soldering the circuit boards, please let a TA know and we will try to help you out. We don't have spare boards if something gets messed up.

DC/Stepper Motor Hat

The following parts should in one bag in your kit

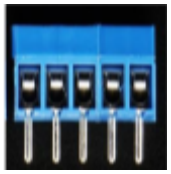
- GPIO Stacking Header
- Adafruit DC & Stepper Motor HAT for Raspberry Pi





Instructions

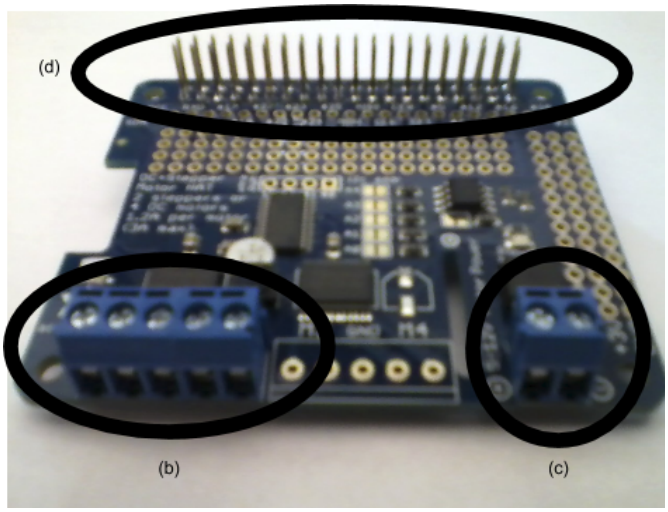
a) Make a 5 pin terminal block by sliding the included 2 pin and 3 pin terminal blocks onto each other Make a 5 pin terminal block by sliding the included 2 pin and 3 pin terminal blocks into each other.

(a) 

b) Slide this 5 pin block through the holes just under "M1 GND M2" on the board. Solder it on (we only use two motors and do not need connect anything at the "M3 GND M4" location) -- See figure below

c) Slide a 2 pin terminal block into the corner for power. Solder it on. -- See figure below

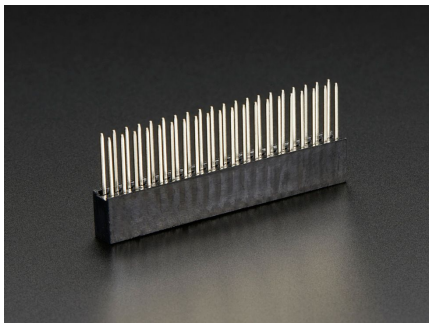
d) Slide in the GPIO Stacking Header onto the 2x20 grid of holes on the edge opposite opposite the terminal blocks. Solder it on. -- See figure below

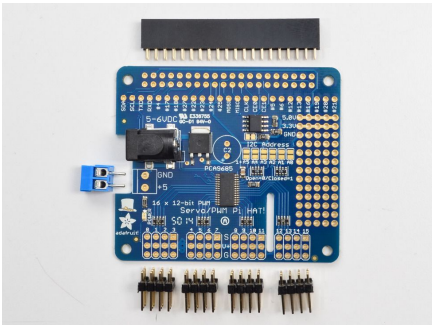


16-Channel PWM/Servo HAT

Based on [adafruits directions]
The following parts should in one bag in your kit

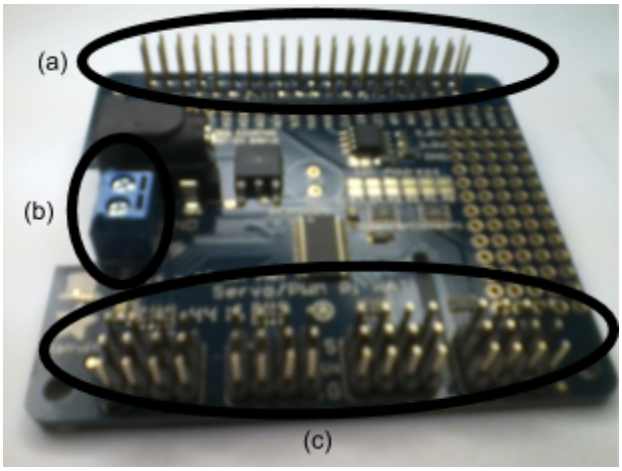
- GPIO Stacking Header
- Adafruit 16-Channel PWM / Servo HAT for Raspberry Pi





Instructions

a) Solder the GPIO Stacking Header at the top of the board, where the 2x20 grid of holes is located. -- See Figure Below
b) Solder the 2 pin terminal block next to the power cable jack -- See Figure Below
c) Solder the four 3x4 headers onto the edge of the HAT, below the words "Servo/PWM Pi HAT!" -- See Figure Below



Robot Assembly

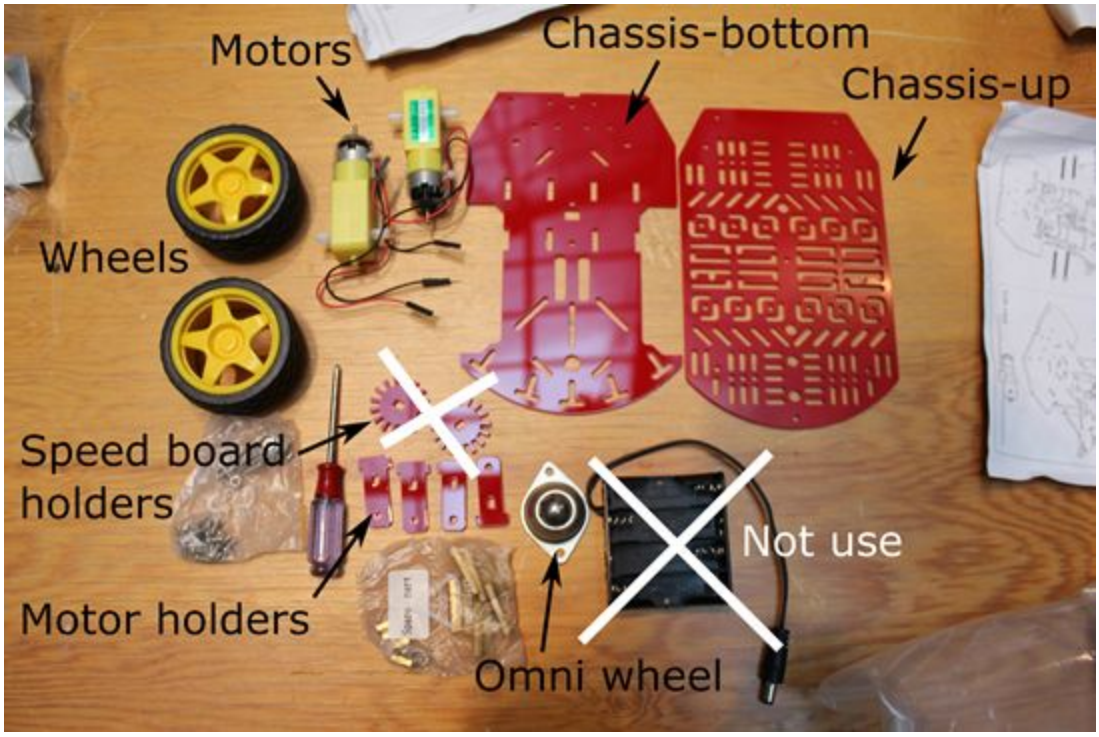
Based on [Assembling the Duckiebot]

Assembling the Car - bottom part

You will need the following parts for this step:

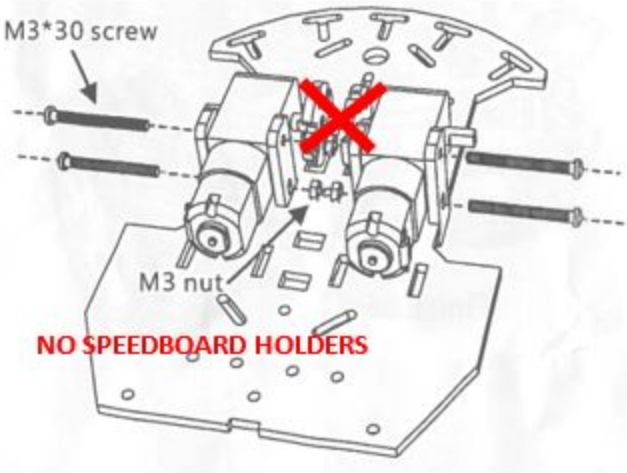
- Chassis Kit

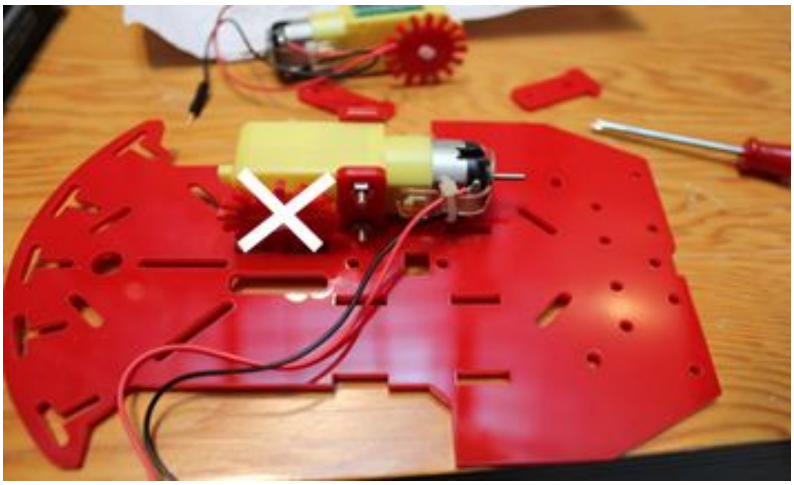
Set aside the batteries package **and the speed board holders** (see picture below), we will not be using them.



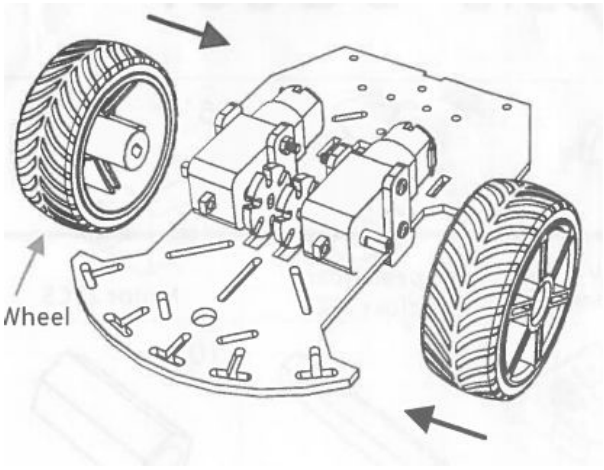
1. Insert motor holders on the chassis-bottom and put the motors as below (with the longest screws M3*30 and M3 nuts)

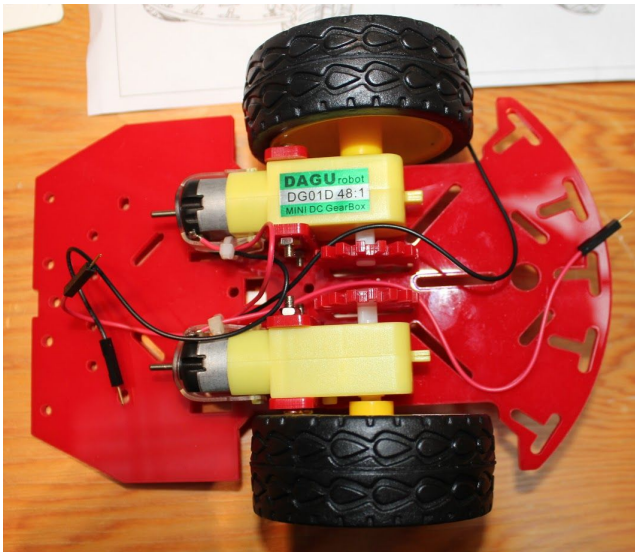
NOTICE: Put a motor so that its wires go inward (toward the center of the chassis-bottom) and 'black' wire is closer to the chassis-bottom. This makes wiring easier later on.
NOTICE: Note that although the speed board holders are in the pictures below we will not be using them



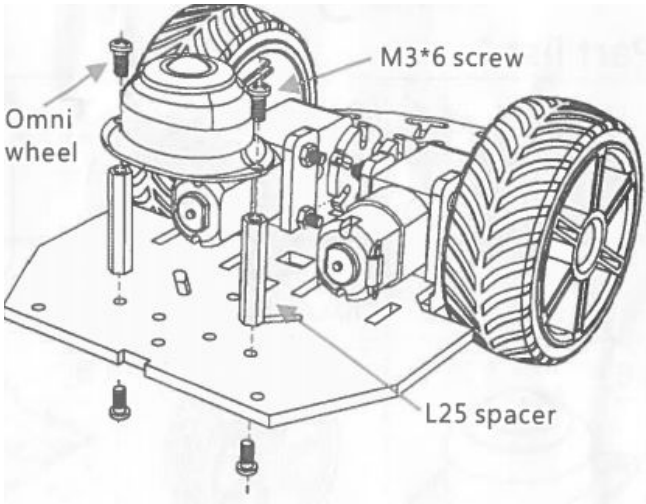


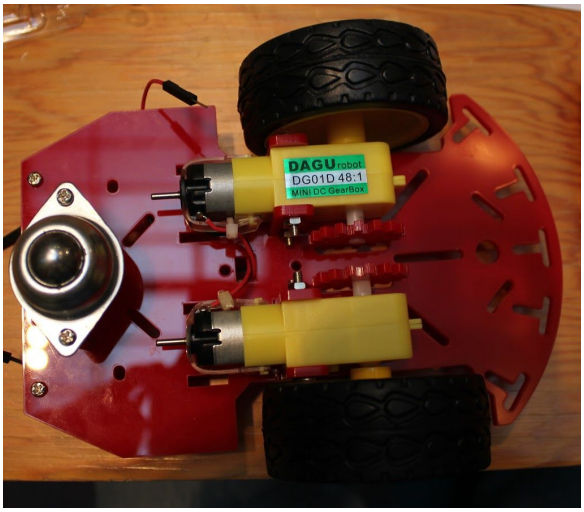
2. Assemble wheels



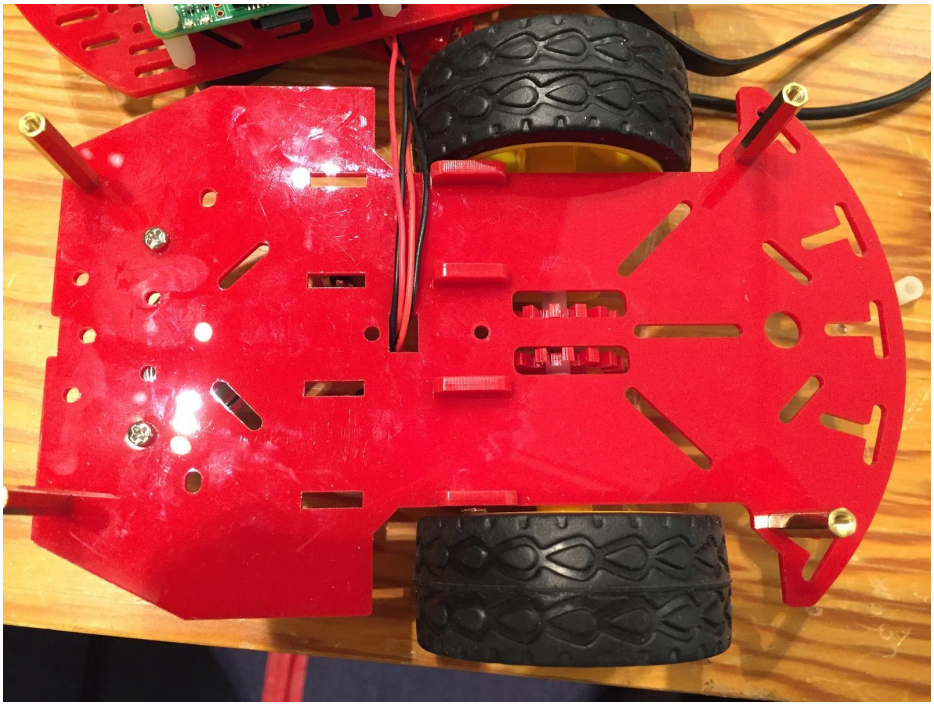


4. Assemble Omni wheel





5. Put the car upright (omni wheel pointing towards the table) and arrange wires so that they go through the center rectangle. Put 4 spacers with 4 of M3*6 screws on each corner as below.



We now have a complete bottom part! Don’t put the chassis-up yet. We will put a Raspberry Pi on it first.

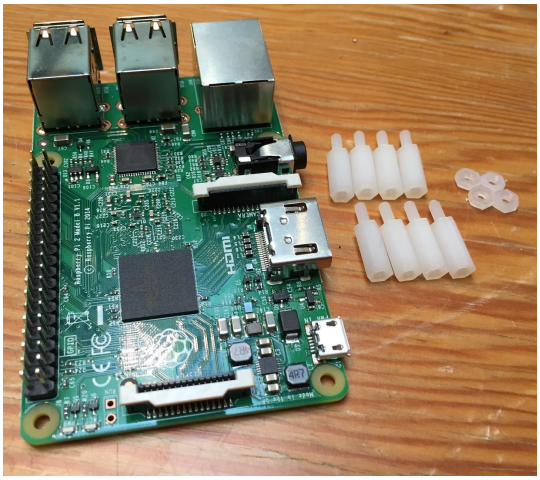
Putting on a PI, camera, and hats - top part

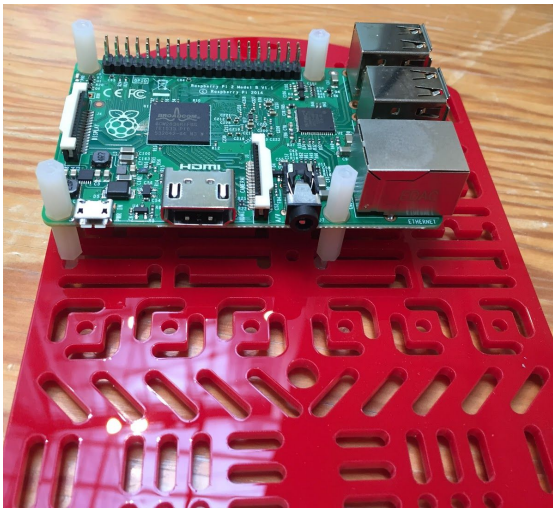
You will need the following parts for this step:

- Chassis Kit
- Raspberry Pi
- Pi Camera
- Pi Camera Mount / Case
- 12mm Nylon Standoffs
- 5mm Nylon Standoffs
- Nylon Nuts
- Motor Hat
- PWM Hat
- Male-Male wire
- Zip ties (Not included in your duckiebot kit -- Ask a TA)

1. Put a Raspberry PI on the top of the chassis-up using 8 standoffs. Fasten 4 nylon nuts on the opposite side.

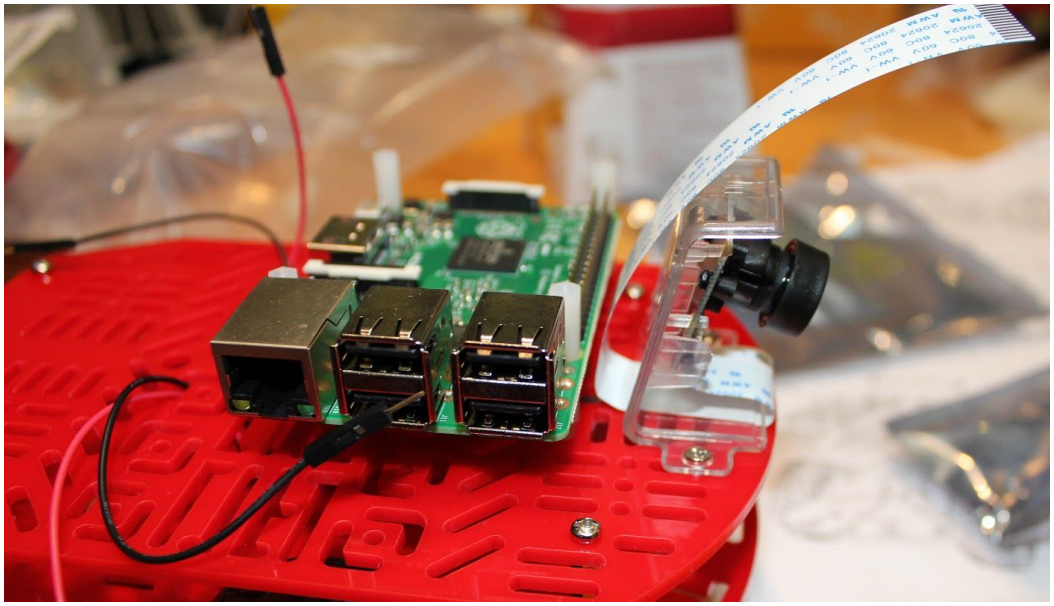
NOTICE: In the Robotics I kits there are 4 standoffs which have slightly longer threaded ends. Use these to go through the chassis and screw into the nut, the others are not long enough.



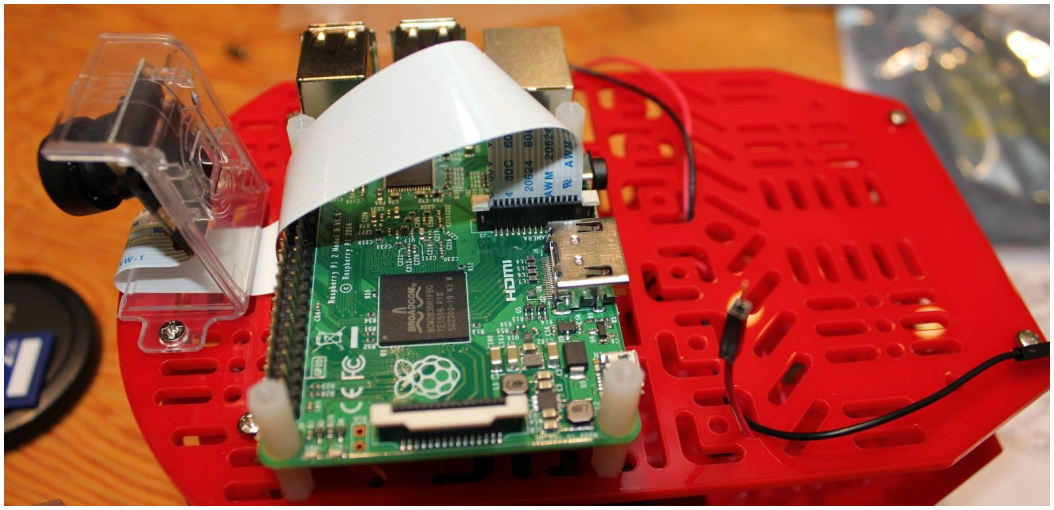


2. Take out the camera mount, and the Pi Camera. Find two each of the M3*10 flathead screws and M3 nuts (which were for a battery holder **in the chassis kit**).

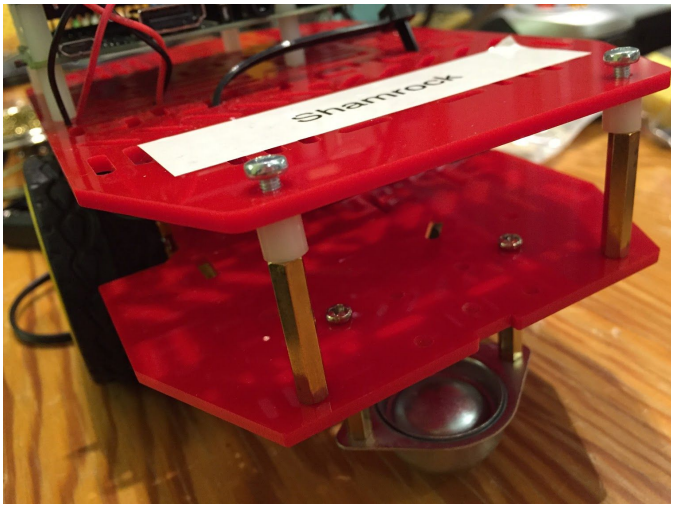
Put the camera inside the mount, and wrap the cable under the bottom of the mount. Attach the mount to the chassis using the M3 screws and nuts. Make sure the cable stays between the camera mount and the chassis.



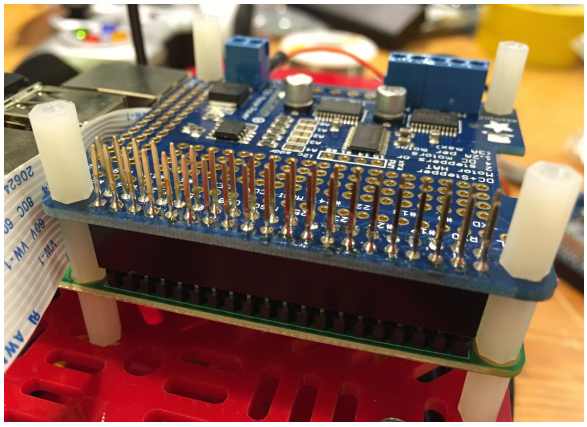
3. To connect the cable to the PI, pull up on the black connector (it will slide up) and then you can stick the cable in. Then slide the connector back down to lock it in, the cable should not be loose. Notice that the plain white side of the cable should be facing the USB ports while the side with text faces the HDMI port..



4. Now, let’s put the chassis-bottom and chassis-top together. Find 4 of the 5mm nylon spacers and 4 3x6mm screws. Screw the 4 nylon spacers into the top of 4 gold spacers of the chassis-bottom. This is to make more room for a battery. Place the chassis-up on top of the spacers and screw through the chassis and into spacer to fasten the pieces together.



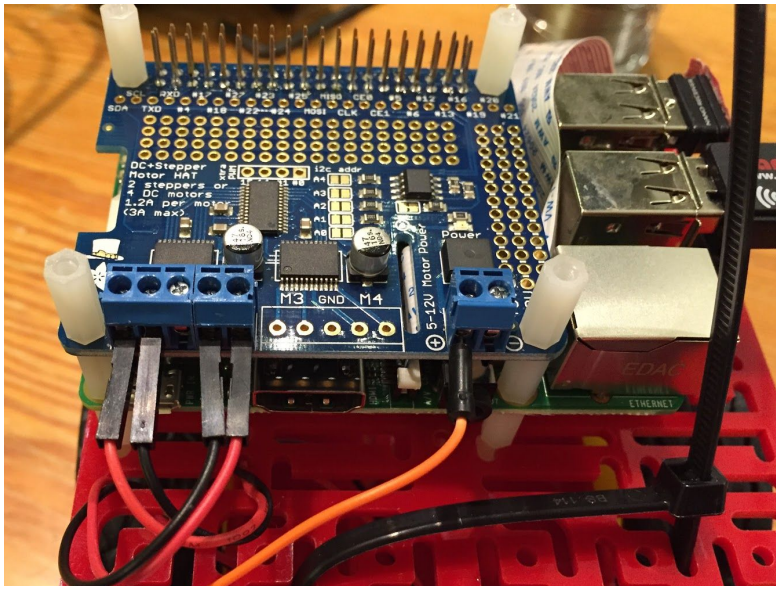
5. Put the DC Motor Hat on top of the Raspberry Pi. The GPIO pins on the Hat should fit over the Pi’s GPIO pins. Fasten the hat down with 4 more standoffs



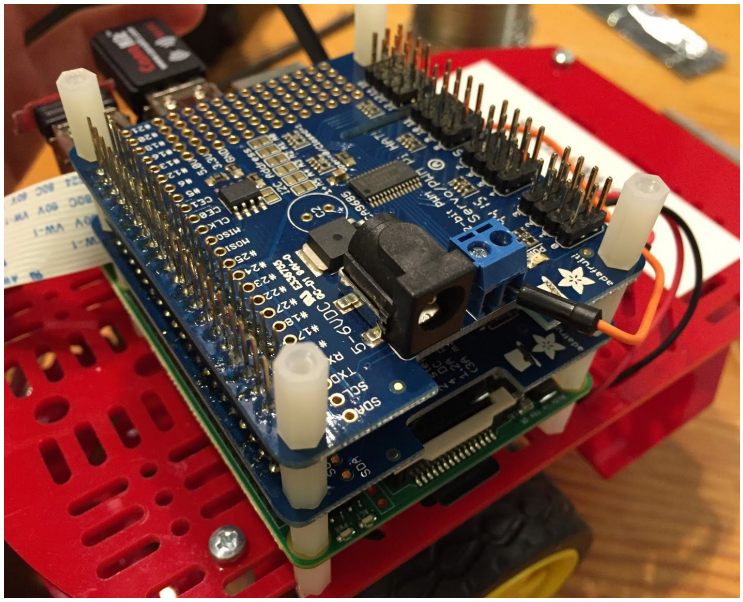
6. Connect the motors' wires. Find a male-male wire. Connect it to +5V input as below (this will be used to connect power from HAT to HAT).

Notice: you might need a smaller screwdriver for this step and the next step. Ask a TA for the (smaller) screwdriver. :)

Notice: We are using M1 and M2. The left (in robot frame) motor is connected to M1 and the right motor is connected to M2. If you have followed the earlier steps correctly, leftMotor:Red - leftMotor:Black - rightMotor:Black - rightMotor:Red will be correct wiring.

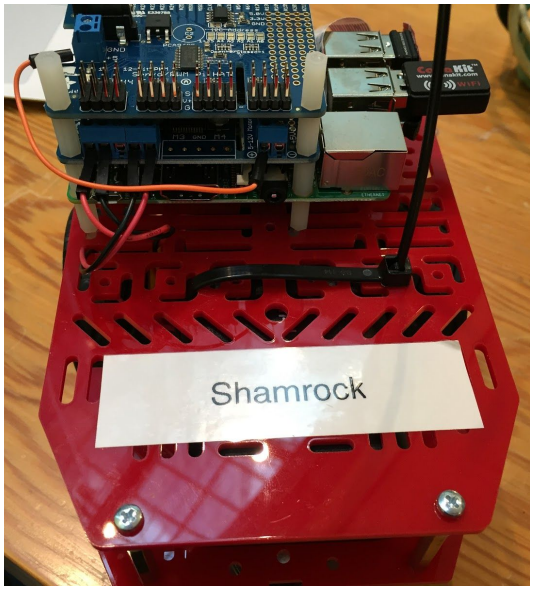


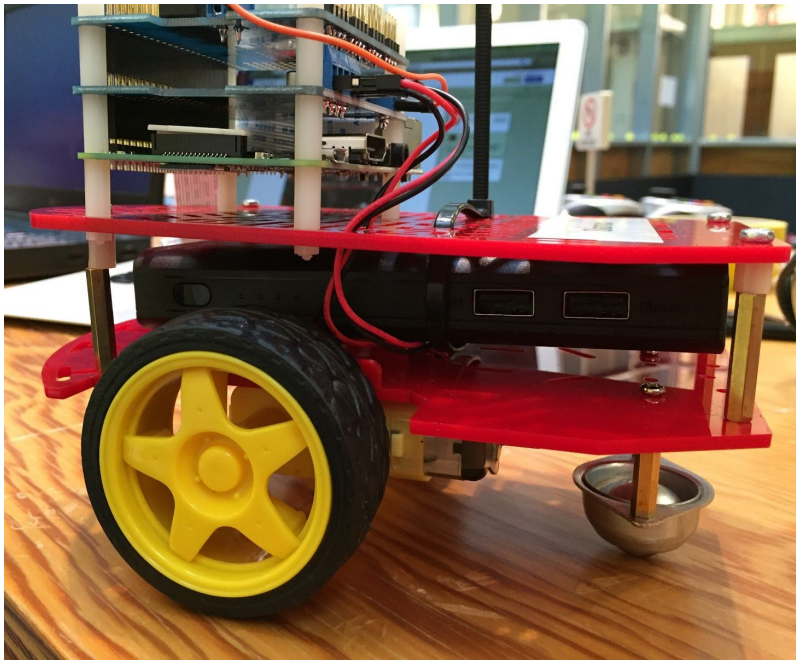
7. Attach the PWM/ Servo Pi HAT board with 4 standoffs like you did with the previous HAT. Connect the male-male wire to +5V output.



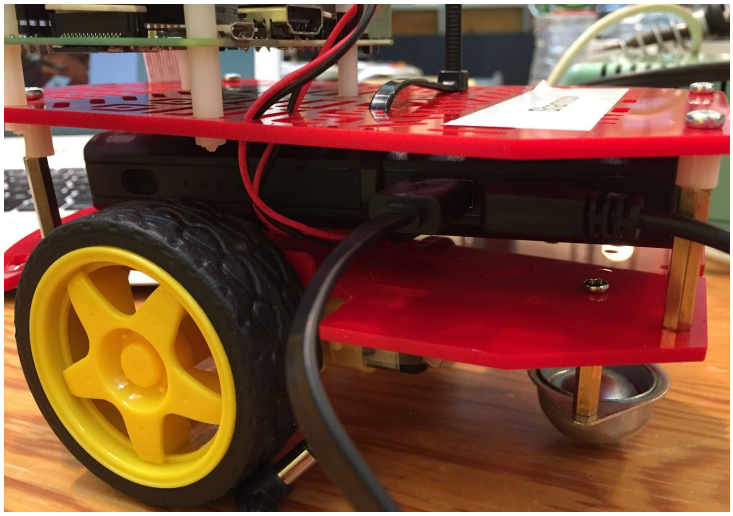
11

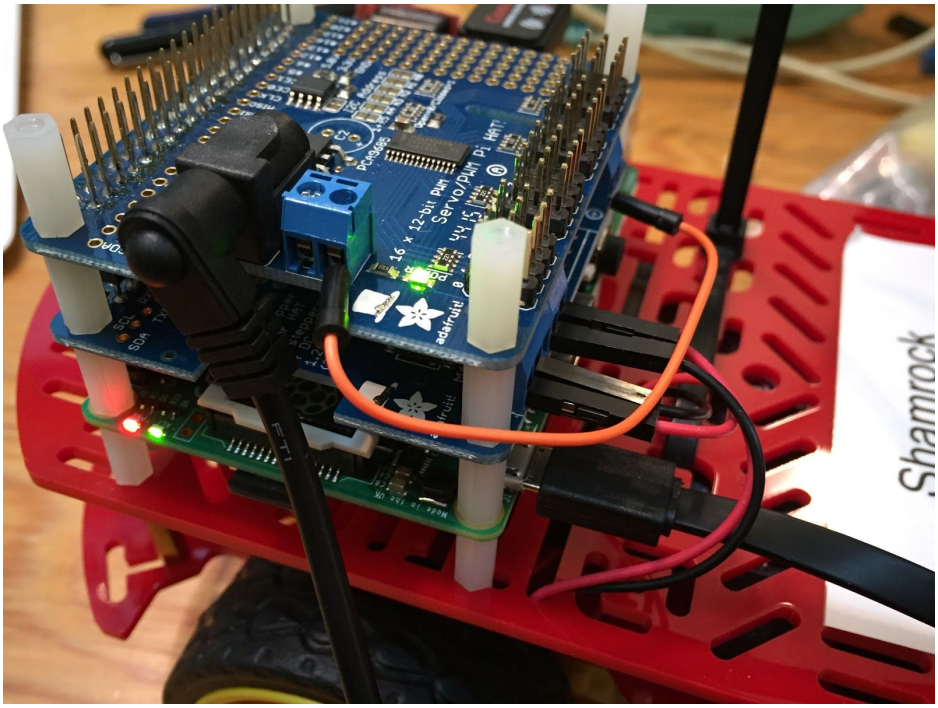
8. Slide your battery into place between the top and bottom pieces of your chassis. You want the USB ports of the battery to be at the back of your robot. Secure it using zip ties.





9. Connect power cables (USB A to 5mm cable **in your Duckiebox** and USB cable **in the battery box).**





Your Robot is now fully assembled!

Loading The Duckiebot OS

This section is based on the following document: Step1.5 - Burn the Image

For this step you will need the following parts:

- microSD card and adapter

We will start by flashing our SD card with our own duckiebot OS (Linux Ubuntu with some packages already installed).

Note: I did not realize that VirtualBox does not by default support SD card readers if they are built into the computer. If you have a USB SD card reader, that is one option. Using your regular OS is also an option. I followed these instructions and didn't have any problems, so I will reproduce the steps below.

Download Ubuntu Image

Download the Ubuntu Image to your computer and unzip it. (Note the file is ~2GB large compressed so it will take some time to download (if you are in the lab we have ethernet cables you can use). Its full unzipped size should be ~7GB so make sure you have sufficient disk space and be patient!)

On (NATIVE - NOT A VM) Linux you can use:

```
$ wget https://github.com/rpiRobotics/duckietown/releases/download/resources_v0.0/2016-09-14_duckiebot.img.gz
$ gunzip 2016-09-14_duckiebot.img.gz
```

OSX or Windows you can just click the link above and extract it using a file extracter of your choice (7zip, WINrar)

Burn Image to SD card

Put the SD card into the adapter. Make sure the adapter is not in read only mode (the small switch on the left side of the adapter should be up -- farther from where you insert the SD card) If the adapter is left in read only mode you will likely get an error later on reading something like "filesystem is read only". In this case remove the adapter, check the write-protection switch, and try again.

Insert the adapter into the computer.

On Windows

I suggest you use Win32DiskImager (note you may need to run the program as administrator) Simply select the SD card drive letter (BE VERY CAREFUL YOU SELECT THE CORRECT ONE), and the path to the image.

Press “Write”.

Wait for the imager to complete, then eject the SD card.

On OSX

I don’t have a Mac to test this on, so if you are really having trouble feel free to stop by office hours and I can flash your SD card for you.
If you want to try this out yourself, I suggest you follow these instructions from the Raspberry Pi website.

On Native (Not a VM) Linux

Check it’s location:

```
$ df -h
```

You should see a list of devices and one (or two) that looks something like

```
/dev/mmcblk0...
```

Unmount all of the part partitions

```
$ sudo umount /dev/mmcblk0p1
$ sudo umount /dev/mmcblk0p2
```

Use pv to monitor the progress of the operation

```
$ sudo apt-get install pv
```

Burn the image to the disk using dd and monitor its progress using pv

```
$ sudo dd bs=4M if=2016-09-14_duckiebot.img | pv | sudo dd of=/dev/mmcblk0
```

Ideally you should see speeds of around ~7MB/s - 15MB/s. If you are not seeing near that range there may be a problem with your computer or the adapter.

The total size of the file being written to the card is 7.9GB, you can use pv to monitor how close the process is to finishing. This step will likely take about an hour to complete.

Once the process completes remove the SD card and insert it again into your computer, you should see the filesystem we just wrote now placed on the SD card.

Check that the filesystem is valid using mount

```
$ mount
```

You should again see a list of devices, including

```
/dev/mmcblk0p1 on /media/<username>/AB3E-B34D type vfat (rw, nosuid, nodev, uid=1000, gid=1000, shortname=mixed, dmask=0077, utf8=1, showexec, flush, uhelper=udisks2)
/dev/mmcblk0p1 on /media/<username>/….. (etc)
```

Unmount the device (both partitions) and remove the SD card adapter from your computer. Place the SD card back into the Pi.

Troubleshooting:
If you receive an error while writing the disk saying there is no space left on the device something may have gotten screwed up with the partitions. You may need to completely reformat the SD card and possibly try using a different machine (or even a different OS) to write the image to the disk.

The remainder of this document is based on parts of the following documents: Setting Up Ubuntu Laptops and the Duckietops, GitHub basics, From SD image to RC control

Connect to the Pi for the first time

There are 2 options for this part, connecting the Pi to an HDMI monitor or connecting to the Pi over SSH.
Note, that if you have an HDMI monitor available to you it may be more convenient to use it up until the step Setup an Ubuntu Laptop to SSH to the Duckiebot, however it is not necessary.

Regardless of which of the next steps you choose, you may need or want the following:

- An ethernet cable (not included in your duckiebot kit - ask a TA if you dont have one)

With an HDMI monitor

For this step you will need:

- An HDMI cable and compatible monitor (not included in duckiebot kit)
- A USB keyboard (not included in duckiebot kit)

Make sure the SD card we set up before is placed back in your PI.

Attach the HDMI cable to the PI. (Note: it is important to do this **before** powering up the PI).

Attach a USB keyboard.

Plug your ethernet cable into the Pi and the other end into an ethernet switch.

Power up. You should see a login screen!

The login credentials are:
    Username: ubuntu
    Password: ubuntu

Connect to the Pi over SSH

This step may be made a little more complicated by the fact that we are connecting over the network, and to start off all of the duckiebots have the same computer hostname. If you know that no one else is setting up the same time you are, you don't need to worry. Unfortunately, if there are other people setting up their duckiebots at the same time as you, you will need to make sure that only one is connected to the network at a time until you have changed the hostname from the default value.

Connect the ethernet cable to your Pi and connect the other end to a port for the network your computer is currently on.
NOTE: Campus network connections will NOT work (i.e. rpi_wpa2) so you may need to do this in the lab where we have a private network.

Turn on the Pi and wait a few seconds.

Make sure you can ping the Pi
    $ ping ubuntu.local

This seems to never work on the first try… It may require several tries of restarting your PI, plugging and unplugging the ethernet cable, trying it over wifi, etc… Its a real pain, but be patient. Once you connect the first time it seems to get better at connecting every time after that.

If you are using a VM you may need to set some additional settings to make sure the VM can access the host network (in Virtual Box make sure your network adapter is set to Bridged Adapter). There are some known issues with connecting by hostname with a VM, so let someone know if you have problems and we will try to work them out.

If you get a response from the ping you should be able to ssh in
    $ ssh ubuntu@ubuntu.local
And when prompted enter the password "ubuntu"

Set the hostname

The first thing we are going to configure on the robot is it's name so that we can connect to it on the network. Choose a name for your robot. This is a simple string that will always appear lowercase. You may use, letters (a-z) or numbers (0-9) **only**. (Normally hostnames may also contain hyphens but ROS does not allow them).

Please write this name on your duckiebot kit sticker under "Robot Name"

Suppose that the name is "duckiebot".

Edit /etc/hostname and replace "ubuntu" with "duckiebot"

```
$ sudo nano /etc/hostname
```

Note: If you choose to use nano, and are not familiar with it. Save the file and exit with "Ctrl+x" it should display the location it will write the file and give a prompt. Type "y" and press enter and you should be returned to the regular terminal.

Edit /etc/hosts and again replace "ubuntu" with "duckiebot"

```
$ sudo nano /etc/hosts
```

Note: the command "sudo hostname duckiebot" is not enough. The change will not persist.

NEVER ADD ADDITIONAL HOSTNAMES IN /etc/hosts (e.g. duckiebot.local) only change the one that is currently there!

Then reboot

```
$ sudo reboot
```

If you are using a monitor, once you reboot you should see the new hostname:

```
Ubuntu 14.04.5 LTS duckiebot tty1
duckiebot login:
```

If you are SSH'ing in, you should now be able to connect using the new hostname.

```
$ ssh ubuntu@<robot name>.local
```

Byobu

It is suggested that you use byobu to allow you to more easily work with the command line, and splitting the terminal session into multiple windows. There is no GUI on the duckiebot, so if you

want to run separate processes you will need some way of “opening new terminals” byobu provides exactly this.

On the duckiebot run

```
$ byobu
```

Byobu is a “GNU screen” with some fancy configurations. You can learn more about byobu from here: http://byobu.co/

Some useful commands, for reference:

F2: opens a new terminal
F3/F4: switches between terminals
Ctrl+F6: closes the current terminal

You can also split your screen into multiple windows as follows:

Shift+F2 - split the terminal window vertically
Ctrl+F2 - split the terminal window horizontally
Shift+ArrowKeys - switch between open terminal windows on the screen.

To quit a terminal you can also type

```
$ exit
```

Connect to the WiFi Network

For this step you will need:

- WiFi Adapter Dongle

While the Pi is off, plug in your WiFi adapter (if the Pi is already on you are welcome to try connecting it, but the Pi likely won’t connect until you reboot)

Boot up your Pi

If you are in the lab, the duckiebot should already be configured to connect to the “duckietown” router so *theoretically* it should just work…. We all know that rarely happens in practice.

If you are not in the lab, please note that there is no way to configure the network on the Pi without somehow already being connected to the Pi (be it monitor and keyboard or ssh’d in…) If you change the network on the PI and reboot, but can’t connect to your duckiebot, you may need a hardwire connection to fix it.

To test it out, connect your computer to the “duckietown” network and try to ping your bot

```
$ ping <robot-name>.local
```

If you are successful, great! You can SSH into your bot and move on to the next step: Camera Test.

If not then continue on to the next section to troubleshoot and learn how to configure the WiFi.

Configure the WiFi

Keep in mind that to change any of these settings, we will need to be connected to our Pi already. This means either connect via ethernet and ssh in to change settings until you can retry, or more preferably, connect to a monitor.

First, we are going to make sure that the wifi dongle is visible to the PI.
Run

```
$ lsusb
```

And look for an entry similar to one of the following

```
Ralink Technology. Corp RT5370 Wireless Adapter
Edimax Technology Co., Ltd EW-7811Un 802.11n Wireless Adapter
```

If you don't see this, your adapter might be malfunctioning and you will likely need a new one.

Now we will check the status of wlan0

```
$ ifconfig wlan0
```

If you are not connected to any network you will likely see something similar to the following:

```
wlan0     Link encap:Ethernet  HWaddr 00:0f:60:05:ff:e8
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

This means the adapter is currently not configured to use the internet.

If you don't see the wifi interface at all, it is possible that your SD card remembers the MAC address of another body (wifi dongle, ethernet chip, etc). In that case you can make it forget the MAC addresses by removing the 70-persistent-net.rules file and reboot to trigger the regeneration of the file with the current wifi and ethernet hardware.

```
$ sudo rm /etc/udev/rules.d/70-persistent-net.rules
$ sudo reboot
```

We can scan for all available WiFi networks using the following

```
$ sudo iwlist wlan0 scan
```

If this produces too much information we can either pipe it to less and use the enter key to advance one line at a time (or type “q” to quit)

```
$ sudo iwlist wlan0 scan | less
```

Or we can pipe it to grep to find the fields we want, for example, for just the ESSID’s we would use

```
$ sudo iwlist wlan0 scan | grep ESSID
```

To configure the adapter to connect to the internet we need to edit the following file

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

And change the following sections based on our network…

For a WPA:

```
network={
        ssid=”YourSSID”
        psk=”password”
        key_mgmt=WPA-PSK
        priority=10
}
```

For WEP:

```
network={
        ssid=”YourSSID”
        wep_key0=”password12345”
        key_mgmt=NONE
        priority=10
}
```

For other network configurations, check the wpa_supplicant.conf Man Pages

Note that we can add multiple networks in wpa_supplicant.conf and set the priority so that it knows which one to connect to if multiple are available. This may be useful if you have a home network you would like the duckiebot to also connect to.

Save and close the file and then run the following lines to reset your network adapter

```
$ sudo ifdown wlan0
$ sudo ifup wlan0
```

You can verify your connection by typing

```
$ ifconfig wlan0
```

And you should now see that you have an IP address and are connected to the network.

You can verify the wireless connection by typing

```
$ iwconfig
```

And you should see the SSID of the network you are connected to.

Note: when switching Wifi adapters, or putting an SD card in a different robot body, remove the file:

```
$ sudo rm /etc/udev/rules.d/70-persistent-net.rules
```

Probably do it now, just in case, and then reboot.

Camera Test

We can test the camera right away to make sure it is attached correctly without setting up ROS.

With a Monitor

If you are directly connected to a monitor, try the following:

```
$ raspistill -t 100000 -o out.jpg
```

You should see an image on the screen. If it is black, remove the lens cap and try again.

Rotate the lens until the image is in focus.

Press ‘Ctrl+c’ to exit

Without a Monitor

If you don’t have a monitor you can still test out the camera. While SSH’d into the duckiebot

```
duckiebot $ raspistill -t 1 -o out.jpg
```

Then, transfer the file to your computer with scp, open a new terminal on your computer and enter:

```
laptop $ scp ubuntu@<robot name>.local:~/out.jpg out.jpg
```

Passwordless SSH to the duckiebot

Now we are going to setup passwordless ssh to the robot.

> ***Protip**: In general, if you find yourself:*
>
> - *typing an IP*
> - *typing a password*
> - *typing “ssh” more than once*
> - *using a screen / USB keyboard*
>
> *it means you should learn more about Linux and networks, and you are setting yourself up for failure. Yes, you “can do without”, but with an additional 30 seconds of your time. The 30 seconds you are not saving every time are the difference between being*

productive roboticists and going crazy. Really, it is impossible to do robotics when you have to think about IPs and passwords...

Connect your laptop to the same network as the robot.

From your laptop, you should now be able to ping the duckiebot

```
$ ping duckiebot.local
```

Do not continue if you cannot do this successfully.

Verify that you can ssh to the PI (from your laptop)

```
$ ssh ubuntu@duckiebot.local
```

Say “yes” if you get asked whether you want to add it to a list of known hosts and enter the password when prompted.

Your terminal should now change to

```
ubuntu@duckiebot:~$
```

You are successfully ssh’d in, you no longer need a monitor to connect to the duckiebot’s terminal.

Now we are going to set up **passwordless ssh** to the duckiebot. This is again done with ssh keys much like when we were setting up github.

The duckiebot already has an ssh key saved to it, so we only need to set up our laptops.

Download the duckiebot key

```
$ wget -O ~/.ssh/duckietown_key1
```

https://raw.githubusercontent.com/rpiRobotics/duckietown/resources/duckietown_key1

We have to edit the permission of the file to make ssh happy. The key can’t be readable or writable from other users or groups

```
$ chmod 600 ~/.ssh/duckietown_key1
```

We then regenerate the public key with

```
$ ssh-keygen -f ~/.ssh/duckietown_key1 -y > ~/.ssh/duckietown_key1.pub
```

Now on your laptop edit the ~/.ssh/config file to add the following lines, replace “duckiebot” with your bot name.

```
Host duckiebot
    Hostname duckiebot.local
    User ubuntu
    IdentityFile ~/.ssh/duckietown_key1
```

To test if it all worked, close any open ssh session to your bot and rerun

Laptop $ ssh duckiebot

You should be able to connect without typing in a password

Fix Date Issues

This step is important for the PI to accept SSL certificates for git and wget using ssh and https protocols.

Duckiebot $ sudo ntpdate -u us.pool.ntp.org

This currently needs to be repeated every time the system is restarted (but sourcing ~/duckietown/environment.sh will automatically do this as well)

Set Up Github Repo (again)

We now need to set up our repositories on the duckiebot.

We start by ssh'ing into the duckiebot.

$ ssh duckiebot

(it should not ask you for a password; if it does, fix that before continuing)

SSH Keys

Now we will generate an ssh key like we did before (you type the parts in orange):

$ ssh-keygen -h
Generating public/private rsa key pair.
Enter file in which to save the key (/home/**ubuntu**/.ssh/id_rsa):
/home/ubuntu/.ssh/ubuntu@duckiebot
Enter passphrase (empty for no passphrase): <press enter>
Enter same passphrase again: <press enter>
Your identification has been saved in /home/ubuntu/.ssh/ubuntu@duckiebot
Your public key has been saved in /home/greg/.ssh/ubuntu@duckiebot
...

Once generated we return to the github page, and navigate again to settings > SSH and GPG keys.

Create a new key for the duckiebot, name it something appropriate.

Get the key to paste in the box

```
$ cat ~/.ssh/ubuntu@duckiebot.pub
```

Copy and paste it in, and press add key.

Edit the ~/.ssh/config file.

```
$ nano ~/.ssh/config
```

Add the following line

```
IdentityFile ~/.ssh/ubuntu@duckiebot
```

Save and exit.

Now to authenticate we do

```
$ ssh -T git@github.com
```

We should see a message saying that we have authenticated.
Refresh the github page and our key should now be green.

Clone the Repo

Clone the repo using

```
$ git clone git@github.com:rpiRobotics/duckietown.git duckietown
```

This will take a few minutes to check out the entire repo.

Set up the ROS environment on the Duckiebot

We will again run the setup scripts to install some libraries that are not in the current version of the OS image

```
duckiebot $ cd ~/duckietown
duckiebot $ ./duckietown_install_car.sh
```

Now we are ready to make the workspace. First we need to source the baseline ROS environment.

```
$ source /opt/ros/indigo/setup.bash
```

Then build the workspace

```
$ cd ~/duckietown
$ make build
```

We seem to pretty consistently require more than one build to get everything built without error. There must still be some dependency issues they haven't worked out.

Change the Machine Files

We need to add some entries to a couple of files to make it easy for ROS to find our new bot.

On your laptop open the ~/duckietown/scuderia.yaml file and add the following at the bottom:

```
<robot>:
    name:<your name>
```

We will now commit the change to the repo. Note that you should ONLY commit to the master branch when you have explicitly been told to.

Before we commit and push, pull down any recent changes to make sure we don't have any conflicts (It is a good idea to ALWAYS do this or we will end up with lots of unnecessary merges)

```
$ git pull
```

To check that git see's you have made changes to the file, use

```
$ git status
```

Then add the file to tell git you would like to track these changes and make them a part of your next commit.

```
$ git add scuderia.yaml
```

Finally commit using

```
$ git commit -m "Adding <robot name> to scuderia"
```

Push the changes to the repo with

```
$ git push origin master
```

SSH into your robot and pull down the changes.

```
$ git pull
```

You should now have the changes both on your laptop and on your robot.

Proper Shutdown and Restart Procedures

To shutdown, **DO NOT DISCONNECT THE POWER**, the system might get corrupted.
Instead, issue the following command:

```
$ sudo shutdown -h now
```

And then wait ~30 seconds before disconnecting the power.
Disconnect the power from the **battery end** of the cable, not the end plugged into the PI, as the more you use this the better chance you have of damaging it.

To reboot your Pi (soft reboot), Issue the following command

```
$ sudo reboot
```

Joystick Demo

This section is based on the following documents: From SD image to RC control, RC control launched remotely, Joystick + camera output in remote laptop

Joystick launched on PI

We now have everything set up so that we should be able to run the joystick demo

Plug the joystick USB into one of the usb ports on the PI

SSH into your bot. Go to the duckietown folder and invoke the following scripts

```
duckiebot:~/duckietown$ source environement.sh
duckiebot:~/duckietown$ source set_ros_master.sh
```

The environment.sh sets up the ROS environment at the terminal. The set_ros_master.sh by default sets the PI as its own rosmaster.

Make sure the motor shield is connected and powered on.

NOTE: for the rest of this document the environment variable ${VEHICLE_NAME} will be used to represent the name of your Duckiebot. When running any command that has this variable you either need to replace it with the name of your robot OR set the environment variable. The environment variable can be set with

```
$ source ~duckietown/set_vehicle_name.sh <robot-name>
```

Where <robot-name> needs to be the exact hostname of your robot.

Run the command:

```
duckiebot $ roslaunch duckietown joystick.launch veh:=${VEHICLE_NAME}
```

If there is no “red” output in the command line then pushing the left joystick knob controls throttle, right controls steering.

The expected result of the commands is as follows:

Left joystick up = forward
Left joystick down = backward
Right joystick left = turn left (positive theta)
Right joystick right = turn right (negative theta)

If is possible that you will have to unplug and replug the joystick or just push lots of buttons on your joystick until it wakes up. Also make sure that the mode switch on the top of the joystick is set to “X” not “D”.

Close the program with Ctrl-C.

Troubleshooting - robot moves weirdly (forward instead of backward): Either the cables or the motors are inverted. Please refer to the assembly guide for pictures of the correct connections.

Troubleshooting - left joystick does not work: If the green light on the right to the “mode” button is on, click the “mode” button to turn the light off. The “mode” button toggles between left joystick or the cross on the left.

Troubleshooting - robot does not move: The joy_mapper_test.launch assumes that the joystick is at /dev/input/js0. To make sure that the joystick is there, you can do

```
$ ls /dev/input/
```

and check if there is a js0 on the list.

To test whether or not the joystick itself is working properly (without ROS), you can do

```
$ jstest /dev/input/js0
```

Move the joysticks and push the buttons and check the printouts.

Troubleshooting - robot moves very weirdly: Check that the joystick has the switch set to the position “x”. And the mode light should be off.

shutting down the robot:
sudo shutdown -h now
then physically disconnect power cables

Remote launch Joystick Demo

As long as you have placed your bot information in the scuderia.yaml file on your laptop (if you do this now you may need to source environment.sh again), you should be able to remotely launch nodes on your bot without first SSH'ing in.
This is simply done as follows:

```
laptop $ roslaunch duckietown joystick.launch veh:=${VEHICLE_NAME}
```

Note that if you have rebooted your PI it may be necessary to first ssh into your bot and run the following

```
$ ss -s
$ source environment.sh
$ source set_ros_master.sh
```

You should be able to drive the vehicle with the joystick just like the last example. Note that remotely launching nodes from your laptop doesn't mean that the nodes are running on your laptop, they are still running on the PI in this case.

You may also notice that significantly fewer printouts are displayed at the terminal. This is one of the limitations of remote launch, however we can still view all of these printouts using rqt_console.
With the joystick demo running, open a new terminal and enter

```
$ rqt_console
```

A GUI interface should open listing all of the printouts in real time.

You can Ctrl-C at the terminal window where the roslaunch command was executed to stop all nodes launched by the launch file.

Remote Launch Joystick + Camera

Launch the joystick remotely using the same procedure as given in the previous section…

```
$ roslaunch duckietown joystick.launch veh:=${VEHICLE_NAME}
```

You should be able to drive the duckiebot with the joystick.

Now we will launch the camera. Open a new terminal window on your laptop and enter

```
$ roslaunch duckietown camera.launch veh:=${VEHICLE_NAME}
```

You should see the red light on the camera come on.

Open a third terminal window.

You can see a list of all of the ros topics currently available by entering

```
$ rostopic list
```

You should see something similar to

```
/diagnostics
/${VEHICLE_NAME}/camera_node/camera_info
/${VEHICLE_NAME}/camera_node/image/compressed
…
/rosout
/rosout_agg
```

You can use “rostopic hz” to get some additional statistics about the data:

```
$ rostopic hz /${VEHICLE_NAME}/camera_node/image/compressed
```

You should see something similar to

```
average rate: 1.621
        min: 0.535s max: 0.995s std dev: 0.14352s window: 16
```

You can view the messages in real time with the “rostopics echo” command

```
$ rostopic echo /${VEHICLE_NAME}/camera_node/image/compressed
```

In this case a lot of numbers should be printed to your terminal… That is the “image” data.

We need to use another program to visualize the data as an image.
The simplest way to do this is with the built in image_view package
Try the following:

```
$ rosrun image_view image_view image:=/${VEHICLE_NAME}/camera_node/image
compressed
```

Troubleshooting: Note there is a space between “image” and “compressed” in the command up above but formatting is making it hard to see that

We could also start an rviz session and visualize the image from there.

```
$ rviz
```

In the rviz interface, click “Add” on the lower left, then the “By topic” tag, the select the “Image” under /${VEHICLE_NAME}/camera_node/image/compressed. Then click ok.
A window should be added to the GUI containing a live stream of the image from the PI.

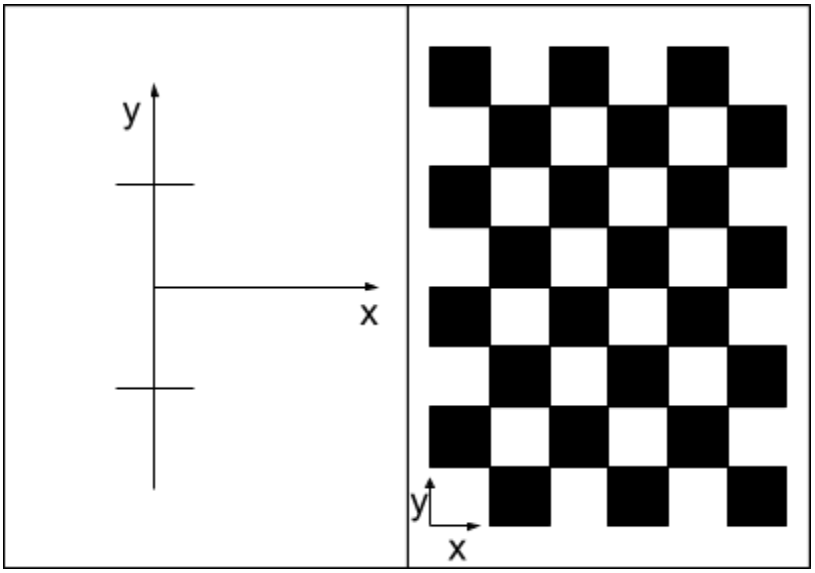
Calibration

This section is based on the following documents: Wheels Calibration, Camera Calibration, Duckietown Operations Manual

Camera Calibration

We are going to start off by calibrating the camera. Camera calibration is performed in two steps: intrinsic calibration - where we determine the internal properties of the camera, and extrinsic calibration - where we determine how the camera view relates to the physical world. We will use a checkerboard pattern to perform both calibrations.

Put the two pages side by side on a planar surface as show below.



A table will be a good place. Fix them on the table top so that they will not move during calibration. Make sure that coordinate frames are conforming.

Make sure the camera_calibration package is installed on your laptop.

```
$ sudo apt-get install ros-indigo-camera-calibration
```

Intrinsic Calibration

As you may have noticed, the fisheye lens on the duckiebot can cause a significant amount of distortion. Most of this distortion can be modeled by several camera parameters and then corrected for. The intrinsic camera calibration will use multiple views of the checkerboard to determine these parameters (as well as other parameters important to computer vision algorithms).

On your laptop run the following:

```
$ cd ~/duckietown
$ make
$ roslaunch duckietown intrinsic_calibration.launch veh:=${VEHICLE_NAME} raw:=true
```

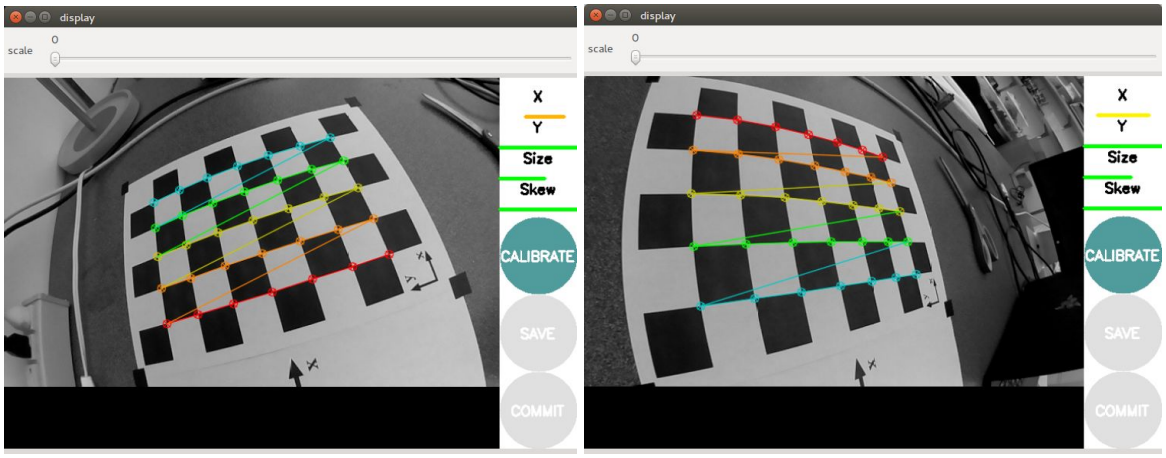
A window will pop up showing the camera view and some buttons. If a window does not pop up you may need to simply try launching it again. (I was occassionally seeing "[duckiebot/intrinsic_calibrator-2] process has finished cleanly" messages indicating the window process for some reason exited)

Grab your duckiebot and point the camera towards the checkerboard (or move the checkerboard in front of the camera).

Move it around the checkerboard. You will notice that if the camera sees all corners of the checkerboard, it will automatically collect data and the display will show detected corners and lines on the camera image.

For good calibration we need to collect multiple, diverse views. As you move the car, you will notice that the 4 bars on the upper right side begin to increase. Each bar shows the observed range of the checkerboard in the camera's field of view.

- X bar: the observed horizontal range (left - right)
- Y bar: the observed vertical range (top - bottom)
- Size bar: the observed range in the checkerboard size (forward - backward from the camera direction)
- Skew bar: the relative tilt between the checkerboard and the camera direction



Once you have collected enough images, all bars on the upper right side will be green and the 'CALIBRATE' button will be enabled. Press the 'CALIBRATE' button and it will perform the calibration. Depending on the number of images collected, this may take a minute or so. The

window may appear greyed out; wait. After the calibration finishes, the rectified (corrected for distortion) video will be displayed.

If you are satisfied with the calibration, press the 'COMMIT' button. It will automatically save to a calibration file **on your bot**. The location of the file is: ~/duckietown/catkin_ws/src/duckietown/config/baseline/calibration/camera_intrinsic/${VEHICLE_NAME}.yaml

You can now ctrl+C in the terminal to stop all of the launched nodes.

Troubleshooting - File not created / error in terminal:
If you are as unfortunate as I was, you may see an error pop up on the terminal where you launched the node from. You can ctrl+C to stop the running nodes, but **DO NOT CLOSE THE TERMINAL YET**, we may need some info from it.

Start by checking your duckiebot for the file, just in case. A quick way to do this is with git status. In a **NEW TERMINAL**

```
$ ssh ${VEHICLE_NAME}
$ cd ~/duckietown
$ git status
```

If you don't see your new file listed under "untracked files", bummer… I guess we will have to make it ourselves.

Scroll back up in the terminal you launched the nodes from and find a line similar to

setting /run_id to d7c54eca-7113-11e6-b44f-2477039ce3b0

The long string of numbers will be different of course, but it should be easy to spot as it will be bolded.

This is the ros run ID and we can find all of the log files and terminal output generated by this launch command will be saved to a folder located (on your laptop) at

~/.ros/log/<run_id>

Lucky for us as long as we successfully calibrated, all of the calibration info we need should have been output to a file under that folder called

${VEHICLE_NAME}-intrinsic-calibrator-2-stdout.log

Somewhere in that file we should see something similar to the following

```
('D = ', [-0.28744871990024995, 0.05682779814184501, 0.00702382763600209,
0.0007274841353560647, 0.0])
('K = ', [320.2678746217217, 0.0, 304.6993435418223, 0.0, 321.65799519188397,
232.49666084860903, 0.0, 0.0, 1.0])
('R = ', [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0])
```

('P = ', [218.3247528076172, 0.0, 303.98131130671936, 0.0, 0.0, 253.5058135986328, 236.72647423584021, 0.0, 0.0, 0.0, 1.0, 0.0])
oST version 5.0 parameters

This is all of the calibration info we need.

We can create a copy of the default.yaml file and then modify it for our bot

$ cp ~/duckietown/catkin_ws/src/duckietown/config/baseline/calibration/camera_intrinsic/default.yaml ~/duckietown/catkin_ws/src/duckietown/config/baseline/calibration/camera_intrinsic/${VEHICLE_NAME}.yaml

Open the file for our bot and change the “camera_name…” line to the following

camera_name: /${VEHICLE_NAME}/camera_node

Now we see several matrices given in the form

matrix_name:
rows: #
cols: #
data: [#,#, … , #]

We need to replace the “data” lines with the data we noted from our log file.
The data for “camera_matrix” corresponds to ‘K=’ in the log file.
The data for “distortion_coefficients” corresponds to ‘D=’ in the log file.
The data for “rectification_matrix” corresponds to ‘R=’ in the log file.
The data for “projection_matrix” corresponds to ‘P=’ in the log file.

In this case the later instructions where we push to the repo you may need two commits, one from your laptop to commit this file, and one from the duckiebot to commit any additional files.

Extrinsic Calibration

Now we are going to perform the extrinsic calibration.
On your laptop, launch the camera using

$ roslaunch duckietown camera.launch raw:=1 veh:=${VEHICLE_NAME}

In another terminal, run the ground_projection node with

$ roslaunch ground_projection ground_projection.launch veh:=${VEHICLE_NAME} local:=1

Note the ‘local’ param specifies that this program will actually run on your computer, therefore the file generated at the end will be locally on your computer, and not on your bot.

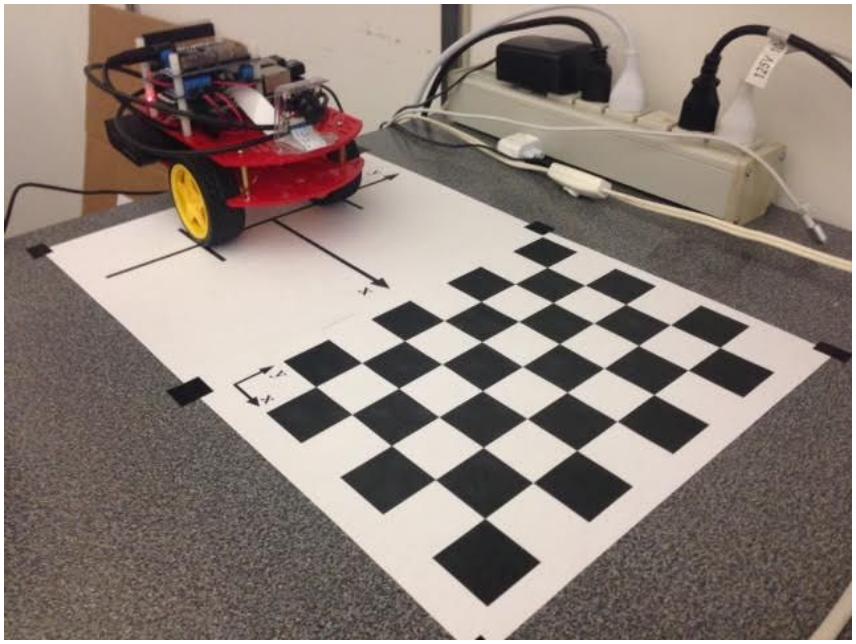
Once this is running, if you open another terminal you can use the rostopic list command to see the new topics

```
$ rostopic list
/<VEHICLE_NAME>/camera_node/camera_info
/<VEHICLE_NAME>/camera_node/image/compressed
/<VEHICLE_NAME>/camera_node/image/raw
/<VEHICLE_NAME>/ground_projection/lineseglist_out
/<VEHICLE_NAME>/line_detector_node/segment_list
/rosout
/rosout_agg
```

The ground_projection node also has two services. They are not used for operation, only for calibration and debugging.

```
$ rosservice list
…
/<VEHICLE_NAME>/ground_projection/estimate_homography
/<VEHICLE_NAME>/ground_projection/get_ground_coordinate
...
```

To do the extrinsic calibration, place your car on the alignment map (next to the calibration grid) and align the wheels with the two lines parallel to the x-axis.



You may want to put a uniform white wall behind the checkerboard pattern to block off any clutter and get the best result possible.

We now estimate the homography by calling the service

$ rosservice call /${VEHICLE_NAME}/ground_projection/estimate_homography

It may take several minutes to finish and won't really do anything in the terminal until it does. Once it finishes, it automatically saves the estimated homography to the following file on your **laptop**

~/duckietown/catkin_ws/src/duckietown/config/baseline/calibration/camera_extrinsic/${VEHICLE_NAME}.yaml

Checking Your Calibration

We can do a quick check to see how well our calibration performed.

Place your duckiebot on the mat like you did for the extrinsic calibration.
Run the following

$ roslaunch duckietown test_camcalib.launch veh:=${VEHICLE_NAME}

This program will find the corners of the checkerboard and compare them to the ground truth locations. It will print out a total mean error as well as a result ("passed"/"failed"). If the result passes our calibration is ok to use. If the result fails it might still be ok. As long as the error reported is fairly low and the calibration patterns in your rectified image look straight and undistorted there shouldn't really be any problem with your calibration.

Wheels Calibration

You might have noticed that your vehicle doesn't really seem to go in a straight line when you command it to do so using the joystick. You may also notice that the velocity does not seem to be what you expect.

This version of the duckiebot does not have any encoders on the motors to provide feedback for how far the wheels have actually turned, all of the control is done in an open loop manner by simply sending a desired velocity to both motors without ensuring that that desired velocity is met.

Slight differences between the motors and the wheels can cause the left and right wheel to travel slightly different distances even though they have made the same rotation.
We can attempt to counter this behaviour by calibrating the "gain" and "trim" on the commands that are sent to the wheels.

The inverse_kinematics_node under the dagu_car package is in charge of translating a desired velocity and angular velocity command, also called a Twist2D, to motor voltages. It also provides several ros service calls to adjust the wheel voltage commands by some gain and trim value. The relationship between the velocities and the output voltages are defined as:

Right_wheel_voltage = (gain+trim)*(linearVelocity + angularVelocity*0.5*baseline)

Left_wheel_voltage = (gain-trim)*(linearVelocity - angularVelocity*0.5*baseline)

The baseline is the distance between the two wheels.

With gain > 1.0, the vehicle would go faster given the same velocity command, and for a gain < 1.0 it would go slower.

With trim > 0.0, the right wheel will turn slightly more than the left wheel given the same velocity command, and for trim < 0.0 the left wheel will turn slightly more than the right wheel. In other words, if your vehicle is drifting to the right, you want to increase the trim, if it is drifting to the left you should decrease the trim.

Manual Method

To calibrate the wheels, we will control the duckiebot with the joystick and then set the gain and trim parameters.

We will need multiple terminals (remember byobu can make this very convenient!)

In the first terminal launch the joystick control

```
$ roslaunch duckietown joystick.launch veh:=${VEHICLE_NAME}
```

Verify that you can move your bot with the joystick.

Now in another terminal we are going to make rosservice calls to change the gain and trim values. Lets start by finding the services we want to use

```
$ rosservice list | grep inverse_kinematics_node
/duckiebot/inverse_kinematics_node/get_loggers
/duckiebot/inverse_kinematics_node/save_calibration
/duckiebot/inverse_kinematics_node/set_baseline
/duckiebot/inverse_kinematics_node/set_gain
/duckiebot/inverse_kinematics_node/set_k
/duckiebot/inverse_kinematics_node/set_limit
/duckiebot/inverse_kinematics_node/set_logger_level
/duckiebot/inverse_kinematics_node/set_radius
/duckiebot/inverse_kinematics_node/set_trim
```

We see all of the services provided by the inverse_kinematics node. I have highlighted the ones we care about.

Now lets have our duckiebot move forward without steering it. If it can make it to the end of the lane without crossing over the lane lines (crossing just a little or touching is probably ok) our calibration should be OK to use. If not we will start by adjusting the trim.

The trim starts at 0.0 so determine if you need to increase or decrease the trim based on which direction the duckiebot drifted (+ for drifting right, - for drifting left)

Choose a value you would like to try and then set it using the following

```
$ rosservice call /${VEHICLE_NAME}/inverse_kinematics_node/set_trim -- <trim_val>
```

Note that the use of -- allows us to enter negative numbers for <trim_val>
It is best to start with very small values (i.e. 0.01)

Once you have set the trim, try again! Repeat until your bot can make it to the end of the lane. Note that because this is all open loop, this is not an exact science… The same trim value may work very well sometimes and seem to have the opposite effect at others. Just try to find a value that seems to do a decent job.

Once the trim is set, we can set the gain in a similar manner.
If you would like the duckiebot to go even faster when the throttle is all the way forward, try increasing the gain, if you would like it to go slower at full throttle, try decreasing the gain. Remember that your duckiebot will be expected to obey traffic laws while driving in duckietown =)

Choose the value you would like to set and then change it by calling

```
$ rosservice call /${VEHICLE_NAME}/inverse_kinematics_node/set_gain -- <gain>
```

Once you are satisfied with your settings we will save them to a calibration file by using the save_calibration service

```
$ rosservice call /${VEHICLE_NAME}/inverse_kinematics_node/save_calibration
```

Your settings will be saved **on your duckiebot** at the following path

```
~/duckietown/catkin_ws/src/duckietown/config/baseline/calibration/kinematics/${VEHICLE_NAME}.yaml
```

This file will be loaded automatically now every time your bot runs the inverse_kinematics_node.

Automated Wheels Calibration Check

A program has also been created to allow you to perform a slightly more automated wheel calibration check, however it requires some additional setup, and you should have already completed the Line Detection Demo before trying this.

In one terminal launch the following

```
$ roslaunch duckietown indefinite_nav_calibration.launch veh:=${VEHICLE_NAME}
```

Then in the second terminal

```
$ rosrun indefinite_navigation test_straight_line.py veh:=${VEHICLE_NAME}
```

You will be prompted

Drive forward?

Hit enter and the vehicle will start moving down the lane.

The program will use the lane filter and the stop line detector to detect whether it has run over the lane or reached the stop line.

Once the test ends it will print a report stating whether the trim and gain tests passed or failed. If a test has failed, set the trim and gain the same as we did with the manual method and retry the test until it does pass. Again, note that since this is all open loop, this is still not an exact science. One test may fail some runs and pass others. As long as it can pass sometimes the parameters should be good enough to do lane following.

This test also seems to report some velocity and trim errors that honestly don't seem to make any sense… So don't put too much faith in it. If your vehicle drives fairly straight with a joystick most of the time there isn't much more you can do without feedback.

Turns Calibration

Note that we haven't created a full intersection or duckietown map so you will need to do your own turns calibration to determine how to turn left/right.

Use the standardized intersection that has start spots marked in tape, like the mats for the gain/trim test.

1) `duckiebot $ rostest indefinite_navigation calibrate_turn.test veh:=${VEHICLE_NAME} type:={right, left, forward}`

Where "right", "left", "forward" tell which type of turn to test.

2) You can also use the make commands "`make test-turn-left`", "`make test-turn-right`" and "`make test-turn-forward`"

Syncing all of Your Calibration Files

You will need all of your calibration files on both your laptop and your duckiebot. We will therefore commit the files to the repo. Some of the calibration scripts were saved to your computer and some were saved to the duckiebot so we will have to make 2 commits.

On your computer:

$ cd ~/duckietown

$ git pull

Use git status to see which files have been created

```
$ git status
```

If the only files that have been added are these calibration files (which it SHOULD be at this point…) you can use

```
$ git add --all
```

Otherwise you will need to add each of the files by name

```
$ git add <path-to-file>
```

Before you commit use git status again to check that you have added all of the files you want to add (we are trying to avoid extraneous commits)

```
$ git status
```

If everything looks good you can commit and push.

```
$ git commit -m “add intrinsic calibration file of duckiebot ${VEHICLE_NAME}”
$ git push
```

Repeat this process while ssh’d into your bot.
Note that when you git pull you’re calibration files should be downloaded.

We can now simply pull our repo on our laptop one final time to sync the changes.

```
laptop $ git pull
```

Other Demos

Line Detection Demo

Troubleshooting:
Please try running this demo as is but if you are unable to get the files to launch my current explanation is that it seems like something is built wrong with scipy in the distribution I gave you all.
You will need to reinstall some dependencies ON THE DUCKIEBOT (the last command will likely take a while)

```
$ sudo apt-get update
$ sudo apt-get install build-essential python-dev python-setuptools python-numpy python-scipy python-sklearn libatlas-dev libatlas3gf-base libblas-dev liblapack-dev libatlas-base-dev gfortran -y
$ sudo pip install --force-reinstall --upgrade scikit-learn
```

If that still doesn’t work. You may need to try reinstalling scipy, but be warned the installation takes a long time (**~2 hours)** to complete.

```
$ sudo pip install --force-reinstall --upgrade scipy
```

If you have tried all of this and it still doesn’t work send me an email

Start the joystick process in one window

```
$ roslaunch duckietown joystick.launch veh:=${VEHICLE_NAME}
```

Make sure you can move your bot around.

Place your duckiebot on the line detection calibration tile.

In a new window start the lane filter node

```
$ roslaunch duckietown lane_filter.launch veh:=${VEHICLE_NAME}
```

In a third window, launch rqt_image_view

```
$ rqt_image_view
```

Select the topic:

```
/${VEHICLE_NAME}/line_detector_node/image_with_lines
```

Likely no image will show up right away.

Press the RB button on the joystick to activate verbose mode for the line detector, which will begin publishing the line detection image.

Observe the performance of the line detector.

Press the Y button on the joystick to start the “anti-instagram” node which will attempt to correct the image for illumination differences. You may see a slight improvement in the filter output.

Autonomous Lane Following Demo

By this point we have everything set up to do lane following around a track.

Place your duckiebot on the circular track in the middle of the lane.

Note that the joystick will not be used in this demo and anything launched with the joystick.launch file should not be running at this point.

Start the line following node with the following command

```
$ roslaunch duckietown lane_controller.launch veh:=${VEHICLE_NAME}
```

Once all of the nodes launch the duckiebot should start making its way around the track until you cancel the program.

AR (April) Tags Demo

The AR tag implementation on the duckiebot is known as April Tags. The april tags we will be using for signs can be printed from this document.

We can start the tag detection nodes by running the following command

```
$ roslaunch duckietown apriltags.launch veh:=${VEHICLE_NAME}
```

The nodes that are launched publish several images as well as topics that can be used to get information about the detected tag, including its pose relative to the camera frame.

Lets start an rviz session to visualize what we can get with the detections

```
$ rviz
```

Once rviz starts we probably need to set the fixed frame. Under ‘Global Options’ and ‘Fixed Frame’ select

```
${VEHICLE_NAME}/camera_optical_frame
```

This means that the data we visualize should be given relative to the camera’s frame

Now press the “Add” button and we are going to add the image view of the detector.
By topic > ${VEHICLE_NAME}/tag_detections_image/
Choose “Image” and switch the dropdown to “compressed”
Press OK.

You should now see a live stream of your camera. You can move this window to the center of the screen and resize it but be careful where you drop it. Rviz has a bug that if you try dropping it somewhere it doesn’t expect it will close the window and the only way to get it back is to recreate it again.

Now if you hold an AR tag in front of the camera you should see the tag is highlighted and a red dot appears at the center of the tag along with the tags ID number.

This is great for verifying that the tag detector is working.

Lets see what other info we can get from the AR tag topics. Lets again press “Add” in Rviz and this time we are going to add the tag poses.
Add By Topic > /${VEHICLE_NAME}/tag_detections_pose
Choose the “PoseArray” and press “Ok”

Now when a tag is detected we should notice an arrow appear on the grid in the center of the screen in Rviz.

When we move the tag around the arrow gets updated to point in the direction of the tag. If multiple tags are detected there will actually be an arrow for each tag!

Of course this information isn't just available in Rviz, Rviz is just great for visualizing it. We can get all of this information (and use it in our programs!) by subscribing to the topics the april tag node publishes.

Lets close Rviz and back in the terminal do

```
$ rostopic list
```

We see a lot of topics but most of them are published by the camera.
We only care about the ones published by the tag detector.
To see essentially all the info that is available at once we can subscribe to the following

```
$ rostopic echo /${VEHICLE_NAME}/apriltags_postprocessing_node/apriltags_out
```

You may need to expand your terminal to actually see everything that is available to you!

Some other topics of interest include

```
/${VEHICLE_NAME}/apriltags_postprocessing_node/tag_pose
/${VEHICLE_NAME}/tag_detections
/${VEHICLE_NAME}/tag_detections_pose
```

These are (for the most part) just smaller subsets of the data that is available.

Running Other Demos

There are many demos that have already been put together, and I will not explain all of them (I'm not even sure what all of them do, but many of them are explained in the Duckietown Operations Manual), but I will explain the shortcut way to running them.

In ~/duckietown you will find a Makefile. Makefiles are basically instruction sets for "making" (or building) software using the "make" command. This works in a very similar way to the catkin workspace that we store our packages in and build using the catkin_make command. The Makefile has the advantage that we can specify certain tags after a "make" command that tells make to execute that specific set of build instructions.

For example, we have already used

```
$ make build
```

to build our ros workspace. In fact, if you look inside the Makefile and find the "build" tag you will see that the instructions under it are simply to run catkin_make.

We use the makefile to simplify the launching of files from the duckiebot, to condense all of the setup that must happen every time into one simple make command.

The makefile is meant to be used when ssh'd into the duckiebot.

To be safe we should start with

```
$ git pull
$ make catkin-clean
$ make build-parallel
```

This will rebuild our ros packages. If we know that no code has changed on our duckiebot (or updates have been made to the repo) since we last ran this command you may skip it.

Now we may run any of the demos.

For example:

```
$ make demo-joystick
```

Will launch the joystick nodes

```
$ make demo-joystick-camera
```

Will launch both the joystick and the camera nodes.
Remember that to view the camera from your computer you must change your ros master since you launched this from the duckiebot.
This can be done with

```
laptop $ cd ~/duckietown
laptop $ source set_ros_master.sh ${VEHICLE_NAME}
```

You may then view the image topics using rqt_image_view, rviz, image_view, etc…

To check your line detection you could run

```
$ make demo-lane_following-default
```

Note that for this demo (and any other demo that needs the joystick) you will have to use byobu to also start the joystick processes, or launch the joystick processes locally from your computer. Once this node is running the process is the same as Line Detection Demo, except you again need to remember that this was launched from the duckiebot so your computer must set your duckiebot to be the ros master.

Once duckietown is completely built (or as it is being built) you can experiment with the **openhouse** demos. These final demos (and there are several of them) put everything together to drive the duckiebots autonomously around duckietown. You will have to look into them on your own to see what each demo does in terms of intersection control, obstacle avoidance, and many other things that the team at MIT have already incorporated into the duckiebot.

Robot Raconteur Interface

There is a Robot Raconteur service on the duckiebot that will allow you to control the duckiebot's motion, capture images, detect AR tags, and view lane information all from Matlab or Python without the need for ROS or a linux workstation (provided you can SSH into your bot from your computer to start the service).

To start the service, ssh into your duckiebot, make sure environment.sh has been sourced, and then run the following script, replacing <port> with the an arbitrary port number (usually in the thousands since low port numbers are reserved for special purposes) that you will use to connect to the service

```
$ roslaunch duckietown duckieRR.launch veh:=${VEHICLE_NAME} port:=<port>
```

If you have not gone through the line detection demos and fixed scikit-learn (which sadly it seems like everyone needs to do) this will likely return an error. You should run the troubleshooting instructions in that section to fix the error permanently. But you can also disable to lane filter and the service should run normally, you just won't get lane information.

If you would like to run the service without lane information or without AR tags you can optionally turn either or both of them off by applying additional arguments while launching the service.

```
$ roslaunch duckietown duckieRR.launch veh:=${VEHICLE_NAME} port:=<port> lanefilter:=false artags:=false
```

You will need the appropriate version of Robot Raconteur installed on your computer. Be sure that the Robot Raconteur version you download is for your system architecture and for the language -- matlab or python -- that you want to use.
For Matlab the folder that you download simply needs to be placed on your Matlab path.
For python there are .exe installers for Windows computers and tar files for Mac/Linux that just need to be extracted to the appropriate location. Be sure to follow the install instructions posted on LMS (also found in the Robot Raconteur Documentation).

Example clients have been posted on LMS for both Matlab and Python. You will notice that both work very similarly so it is really up to which you are more comfortable working in or which is more convenient.
In either case you will need to change the connection URL string for your specific duckiebot:

```
"tcp://<HOST>:<PORT>/DuckiebotServer.<DUCKIEBOTNAME>/Duckiebot"
```

<HOST> should be replaced with either <DUCKIEBOTNAME>.local or your duckiebot's IP address (which is returned when you ping your duckiebot by hostname).

<PORT> should be replaced with whichever port number you specified earlier

<DUCKIEBOTNAME> should be replaced with the name of your duckiebot.

Examples: “tcp://duckiebot.local:1234/DuckiebotServer.duckiebot/Duckiebot”
“tcp://192.168.1.1:1234/DuckiebotServer.duckiebot/Duckiebot”

RR Service Def

The Service Definition is a special file that tells Robot Raconteur all of the functions and parameters that should be available on the service. I copied the duckiebot service definition below so you know everything that is available to you. You may need to look at some of the Robot Raconteur documentation to fully understand how to access / use everything.
# Indicates a Commented line

```
#Service to provide simple interface to the Duckiebot
service Duckiebot_Interface
option version 0.5

struct DuckieImage
   field int32 width
   field int32 height
   field int32 step
   field uint8[] data
end struct

struct ImageHeader
   #field string format
   field int32 width
   field int32 height
   field int32 step
end struct

struct LanePose
   field double d
   field double phi
   field double sigma_d
   field double sigma_phi
   field uint8 status
   field uint8 in_lane
end struct

struct AprilTag
   field double size
   field double[3] pos
```

```
    field double[4] quat
end struct

object Duckiebot
    property double v
    property double omega
    property double x
    property double y
    property double theta
    property uint8 camera_open
    property LanePose lane_pose
    property AprilTag{list} april_tags
    event TagDetected(AprilTag at)
    function void sendCmd(double v, double omega)
    function void sendStop()
    function void openCamera()
    function void closeCamera()
    function DuckieImage getImage()
    function ImageHeader getImageHeader()
    pipe DuckieImage ImageStream
end object
```

Creating New ROS packages and Git

From this point on, you are ready to make your own contribution for the code base.

Storing your code directly in the duckietown github repo will make your life infinitely easier. You won't need to set up any additional repositories on your duckiebot, and it is easier to use the packages already included in the repo. Furthermore, we can easily get code onto our duckiebot by simply pushing it to the repo and then pulling it on the duckiebot (as we have already seen).

Because we all need to use this github repository there are a few guidelines that we need to follow moving forward. For those of you familiar with git and best practices, bear with me…. Hopefully you share my desire to keep the code base clean.

The first stems from the idea of a clean working master.
The master branch is at all times meant to be a working version of your code base. It may not be the latest and greatest version with all of your newest features, but it should be reliable, and anyone who wants to start with a brand new duckiebot shouldn't have any problems using it.

Obviously you need to add to the code base to implement your new features and in some cases you may even want to check these in to the repository, so that you can share them amongst team members, or pull them onto your robot for example.

Git gives us several options to deal with this and still keep a clean working master… “forking” and “branching”. I discuss forking below for those of you that are unfamiliar with git and interested in understanding it a little more. However, we will not be using forks, so if you just want to know what you need for the project skip ahead to “Creating a Branch”.

Forking a Repo

The first way to create a new “sandbox” for code development is exactly what was done with the rpiRobotics/duckietown repo. We created a “fork”. A fork in github is literally an exact replica of the “parent” (or “upstream repository”) copied over to a whole new repository. It will be checked out to your account, you will have admin rights to the repo, and in most cases that gives you free reign to change what you will in the repo.

Because a fork is an entirely new repository, with its own master branch, nothing you change will effect the repo you forked from. However, git does keep a reference to the original repository so that you can compare the history of your two projects. This makes it easy to pull changes from the parent, and if desired request that your work be incorporated back into the parent repository (this is known as a “pull request”). The key word there is ‘request’ because the admins on the parent repo can certainly reject your request if your code is not up to their standards.

Some teams prefer to use forks to do all of their feature work because it forces the use of pull requests to merge back with the original master. After the pull request has been created the team can discuss the code changes, try them out, and ultimately decide if it is ready to be merged back in.

In many cases, however, a fork is overkill. They are more often used for enormous overhauls, where so much might change that you can pretty much think of the new fork as a whole new repo with only loose ties to the original.

Most teams instead use “branches”

Creating a Branch

Branches are an integral part of the git workflow when creating new features in your repo. They can be thought of as a “sandbox” environment to try things out without leaving your repo.

If you branch off of the master branch, your new branch will appear at first to be the same as the master. You can make commits on the branch, as it keeps its own history. You can even branch

off of your branch if needed. Ideally, at some point these commits will be merged back into the master when the feature work is complete.

Once we have made all of our commits, tested the branch thoroughly, and are ready to merge it back into the master. We can create a pull request to let everyone else know we would like to merge the branch back and let them review the changes.

The rest of this section we will walk through the git workflow and the ros package creation workflow. This process should be followed for any new work to the repo.

A branch has already been created for the RPI robotics projects.
We will start by checking that branch out

```
$ cd ~/duckietown
$ git pull
$ git checkout rpifall2016
```

Check that we are on the correct branch with

```
$ git branch
```

rpifall2016 should be highlighted.

Now we will create a new branch off of this branch for your project work.

```
$ git branch rpifall2016-<branch-name>
$ git checkout rpifall2016-<branch-name>
```

Your branch name should be something descriptive, and in the context of the Robotics I projects should probably be something like your group name or project name.

Again, lets just make sure that we have actually changed our branch

```
$ git branch
```

The branch you just created should be returned.

We should do a merge with the master branch to make sure any commits (like our calibration files that we committed to the master) make it into our branch

```
$ git merge origin/master
```

You should not have any merge errors, (BUT PLEASE LET ME KNOW IF YOU DO!)

Now we need somewhere to put our source code.
Inside of the catkin_ws we can have multiple levels of folders and the catkin build process will traverse through the folders and find all the packages. To keep things organized, a fall2016_rpi folder has been created for Robotics I projects, and we will create one folder per group inside that to hold all of your packages.

Inside the fall2016_rpi folder make a directory with a descriptive name unique to your project files.

```
$ mkdir catkin_ws/src/fall2016_rpi/<folder_name>
```

Now lets move to our new folder and create a README file

```
$ cd catkin_ws/src/fall2016_rpi/<folder-name>
$ nano README
```

In the README put your group information:

Group Number
Team Members
Project Name

Save the file and exit the editor

Add the file in git so that the file gets tracked, and then commit the file.

```
$ git add README
$ git commit -m “Adding project folder for <team>”
```

Before we push, pull down any recent changes to make sure we don’t have any conflicts (It is a good idea to ALWAYS do this or we will end up with lots of unnecessary merges)

```
$ git pull
```

Finally we need to push the branch so that it is publically available on the repo.

```
$ git push --set-upstream origin rpifall2016-<branch-name>
```

Our branch is all ready for us to create new packages!

Creating a ROS Package

We aren’t going to go through the ROS package tutorials in this document but you are encouraged to go through them on your own or refer to them as needed to learn how to create a package. If you go through the tutorial in the repo, please don’t check it in, we don’t need a bunch of redefinitions of the same package all in our repo. You can set up a gitignore to ignore the package so that git stops warning you it is there and hasn’t been checked in.

The ROS tutorials can be found here:
http://wiki.ros.org/ROS/Tutorials

Once you are ready to create your own package for your work, follow the same steps as given in the tutorials, but create your new package inside of your teams folder on the branch that you created in the previous step.

Again, your branch won't be monitored and won't break anything for everyone else, so you can feel free to add commit and push as you see fit.

Creating the Pull Request

At the end of the semester we want to merge your progress back to the rpifall2016 branch so that everything is kept together. We will accomplish this merge by creating what is called a "pull request" (you are requesting that the base branch pull any changes from your branch). This will also be extremely useful for determining your final grade for the project since everything that you did for the entire semester will be displayed in one pull request!

If you implemented something particularly useful we may even want to merge it back to the master branch for future students to use.

To create a pull request, make sure that all of the changes that you want to include have been committed and pushed to your branch.

We can then create the pull request from the repo's github page.

Go to https://www.github.com/rpiRobotics/duckietown and sign in.

You should see a dropdown that says "Branch" and most likely is currently set to "master". We are going to change this to the branch you created "rpifall2016-<branch>".

Now right next to this there is a button that says "New Pull Request". Press that button.

The page will change to a comparison view of a "base fork" and a "branch fork", and the branch of each to compare. Hopefully the "branch fork" should be set to your current branch. If not change this value first to be your branch.
Then change the "base fork" to "rpiRobotics/duckietown" and for the branch choose "rpifall2016".

You should see all of the changes you have made.
Name your pull request.
Write a message if you want.
Create the pull request.

And you're done! You should see your pull request is open. We will review it, look at your work. Maybe make some comments, and then ultimately accept it and your code will be merged back to the rpifall2016 branch!

At Home Networks

I will try to detail a couple of options for connecting to your duckiebot outside of the lab network.

Add a WiFi Network to your Duckiebot's Config Files

As mentioned in the Configure the WiFi section, it is possible to add networks to your wpa_supplicant.conf file which is where your Pi stores the connection info (SSID, passkey, etc). If you had a network at home that you would like your duckiebot to be able to connect to we can add it here (remember that rpi_wpa2 won't work because of the authentication).

The catch-22 here is that you need to already be connected to the duckiebot in some way to be able to change these config files. That means either come to the lab and do it over the duckietown network, or use a keyboard and monitor and connect physically to the Pi.

Once you are connected the process is fairly simple

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Your file will likely look as follows:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
 ssid="duckietown"
 scan_ssid=1
 psk="quackquack"
 priority=9
}
```

Lets say that you have a home network that also uses WPA authentication (most common but check Configure the WiFi if you use a different authentication type). We can simply add another entry in the file for your home wifi

```
network={
 ssid="MyHomeWiFi"
 scan_ssid=1
 psk="password1234"
 priority=8
}
```

The priority does not matter much, since you will likely never have the duckiebot seeing both networks and having to choose between the two. However, for your reference, the Pi will connect to the network with the highest priority number.

Now once you are at home, your Pi *should* connect to your home WiFi and as long as your computer is connected to the same network you should be able to connect to the Pi.

Direct Wired Connection

It is also possible to set up a direct ethernet connection (as in Pi ethernet directly to computer ethernet) with no router involved.

The catch here is that you are stuck with a physical wire connection but for programming at home that is hopefully ok.

I have only tried this on windows and linux, but I am sure OSX has similar steps.

We can first configure your VM networking to allow this.

In Virtual Box, Select your computer and then press “Settings” to open the settings for your VM. Switch to the Network Tab.

Under Adapter 1 we should have the adapter enabled and attached to a “bridged adapter”. Under “Name” we should see the physical card that we are connecting with. It is likely your wireless card, take a note of this, but if you look through the dropdown you should see a wireless and an ethernet connection.

Now switch to Adapter 2.
Enable the Network Adapter.
Again choose a bridged adapter but for this time the adapter we choose should be the other one (i.e. the ethernet adapter this time).

Now your VM has two network adapters, one connected to the WiFi network your computer is connected to and one connected to Ethernet LAN.

We can go ahead and connect the ethernet between our duckiebot and our computer and then turn the duckiebot on.

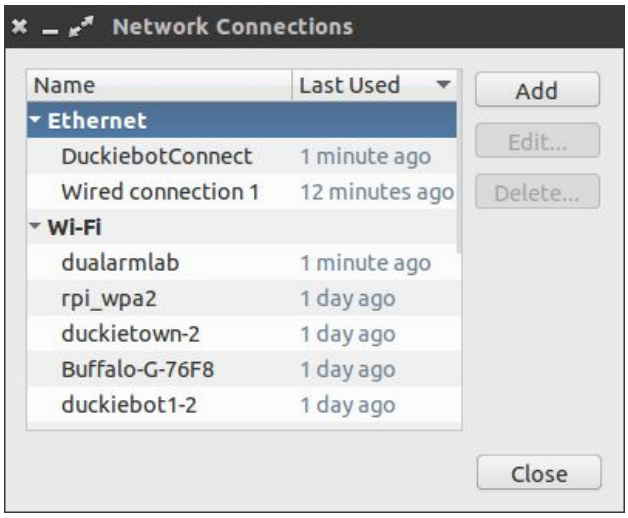
There is one more step we need to ensure the duckiebot has full functionality. We are essentially going to ‘share’ the wifi connection from our computer over the LAN to our duckiebot.

The first step is to connect to any WiFi network as normal (even rpi_wpa2 is ok in this case).

Linux Computers

In the network configuration menu

Click Edit Connections

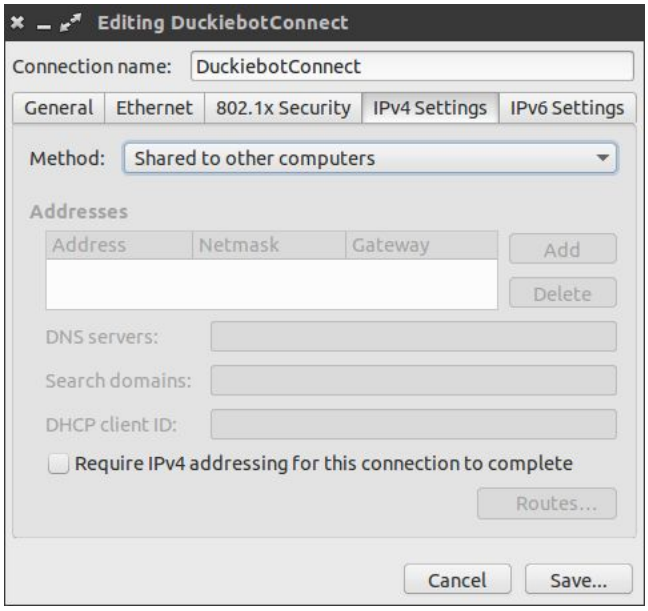


Then Click Add
Select the type to be Ethernet and press Create

Up at the top where it says Connection Name, name the connection something like "<Robot>Connect"... as long as you remember what it is.

Now go to the IPv4 Settings page

Change the Method to “Shared to other computers”



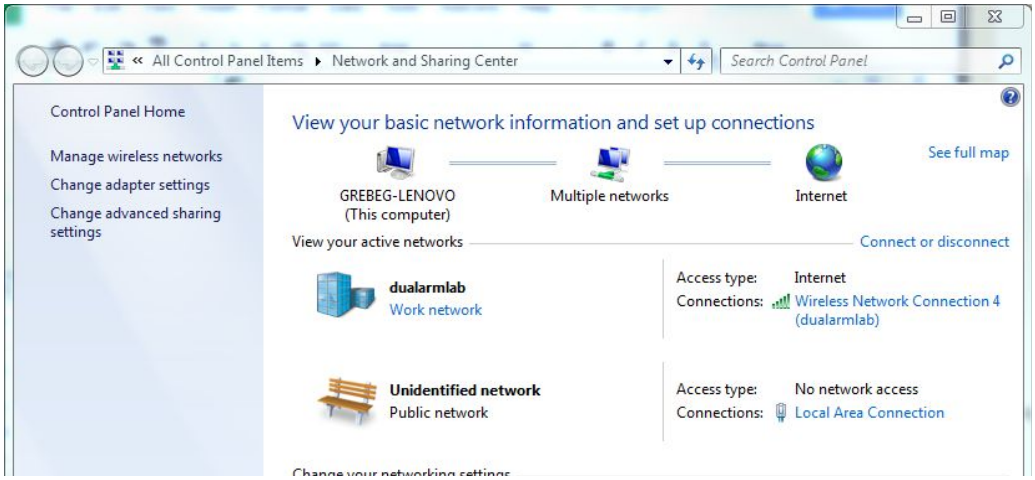
Press Save.

Now you should be able to go into the network manager again, and make sure that the “Ethernet Network” you are connected to is the one you just created.

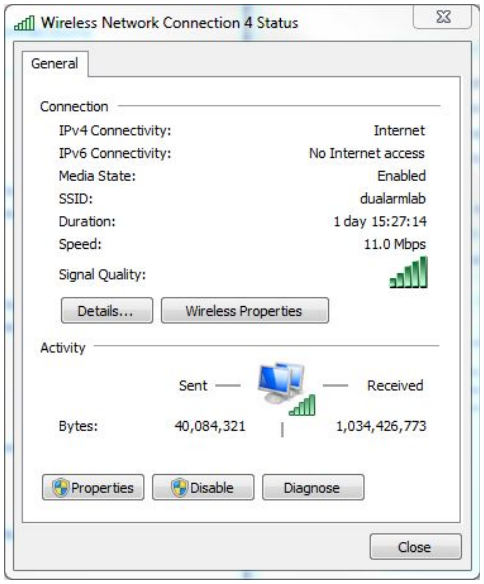
Now your duckiebot gets its internet from your WiFi but is essentially connected to your LAN.

Windows Instructions

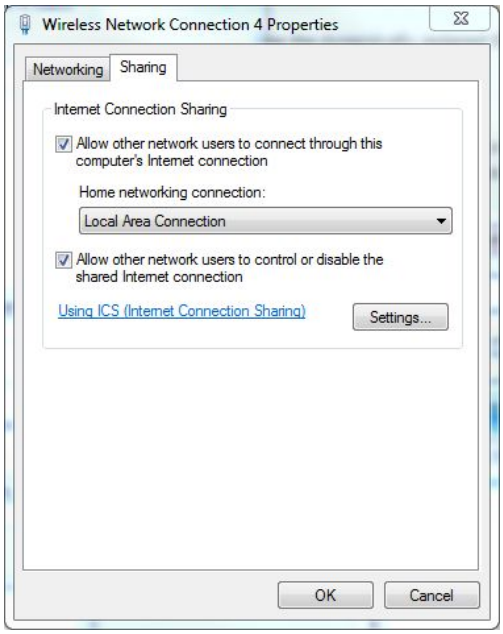
Open “network and sharing manager”
You should see that you are connected to multiple networks, likely the WiFi and some other Local Area Network (thats your duckiebot).



We first need to modify our Wireless connection settings:
Click Next to the Wireless Connection where it says “Wireless Network Connection ...” and a new window should pop up.



Click “Properties” and then switch to the “Sharing” tab
Check both of the checkboxes, and if you have multiple connections you may also see a dropdown listing other connections. Make sure you select the connection that your Pi is attached to (If you notice the figure above from network and sharing center, my pi is attached to the “Local Area Connection” so that is what I chose)



Click OK, close the other tabs.

Check your Connection

We can check that we can see our PI.
Open up a terminal and try to ping it

```
$ping ubuntu.local
```

If we get a response we are connected, otherwise, you may need to restart your computer before the changes will take effect.

Things I have already tried but DID NOT work

- creating an Ad-Hoc network on your computer (on windows)
    - It seems like the Pi does not know how to connect to one of these ad-hoc networks, as it does not just treat the ad-hoc network like a wireless access point.
- Creating an Ad-Hoc network on the Pi
    - The Pi is capable of creating an ad-hoc network as well, and my (windows) computer was able to connect to this network
    - There is too much setup involved, and even though I was connected to the Pi's network I wasn't able to ping the Pi.
    - I dont think the VM knows how to connect to this ad-hoc network even if I did figure out how to get the Pi and computer to ping one another.
- Using some networking software like "Virtual Router" to "mimic" the duckietown network

- The Pi connected to the network, but it wasn't able to resolve a hostname or get an IP address… so I couldn't ping the Pi.
- This software is only supported on windows 7 and 8 so this likely wouldn't help a lot of people anyways.