momobot

Release 1.1.0

SUTD Organisation of Autonomous Robotics

Jul 02, 2022

COMPONENTS

Version 1.1.0

This document contains the relevant specifications and software packages that are used in the MOdular MObile (MOMObot) Robot. MOMObot is a service AGV built for extensibility and to roam autonomously using ROS!

CHAPTER

ONE

GENERATING THE DOCUMENTATION

Note: Ensure that sphinx is installed beforehand!

To generate the documentation locally, git clone the repository & build the document.

```
$ git clone https://github.com/sutd-robotics/momobot
$ cd momobot/docs
$ make html
```

The mainpage would be located at `_build/index.html`.

CHAPTER

TWO

INDEX

2.1 Hardware

2.1.1 1.1 Dimensions

- 52x52x80 cm (just frame)
- 52x52x84 cm (including wheels)

Dimensions decided based on the requirements that MOMO has to fit inside lifts, and pass through doors.

The ratio of 52 cm : 80 cm of the frame is to improve the aesthetics of the MOMO, by observing the golden ratio.

2.1.2 1.2 Specifications

Side panels are tapered on one end to allow for a 270 degree field-of-view to ensure unobstructed LIDAR operation. Components on the bottom are shifted to the rear in addition to tapered side panels to allow for unobstructed view for the LIDAR.

MOMO uses rear wheel drive to ensure maximum stability when driving forward. Front wheel drive systems are inherently unstable due to fluttering of the castor wheel - undesirable oscillations during autonomous navigation (this was shown empirically as the original design was a front wheel drive).

This allows for the robot to deal with any sudden loss of front traction as well as traverse ramps.

Design Payload Weight: 20 kg In actuality, MOMO can take the weight of 1 person (60+ kg) when both motors are moving. (However, it would be unable to perform pivot turns.)

Some quirks in selection and design

2040 Extrusions are utilised to allow for greater loading of frame by increasing the amount of material vertically. This increases the maximum strain which the frame can withstand.

4 corner pillars of MOMO comprise of L-shaped 4040s to prevent horizontal warping of the frame.

Reason for more extrusions in front: - Expose more of the structure to allow lidar to function - Would have to pay attention to how frame is supported

Reason to do rear wheel drive - Enable robot to handle uneven ground:

- Middle wheel drive - on uneven ground, castors lift
- Front wheel drive - spirals out of control, oscillates during navigation

Motors are mounted with 2 thick aluminum plates Motors that slide in from the side allow for ease of maintenance

Bottom layer braced with L shaped extrusions to stop battery from moving too much, as well as to provide a sturdier frame.

Momo as 4 seperate layers:

1. Motor and Battery 1.5. Lidar 2. Electronics - driver, relays, mcu, esc, all else 2.5. Screen 3. Computer 3.5. E-stop 4. Any thing else to be added

Castors chosen so as not to leave marks (stay away from rubber wheels - first waiterbot used rubber castor wheels which left black marks at Fablab when navigating)

After a long day of running Momobot, there is a need to check gussets as they may loosen. Very important to check the bottom as there are over 20 gussets at the bottom securing the batteries in place.

Can be tipped without problem as frame is sturdy

Good design: Low CG - able to stabilize quicky after tilting

CG weight Rear wheel drive - all weight on rear wheel, when accelerate does wheelie, so weight was shifted forward

2.1.3 1.3 Gotchas, Hacky Stuff and Things to Take Note Of

Take note: - Screws to motor -able to come out with vibrations:

- Remember to check gusset bolt tightness if MOMO has been operated with many vibrations
- Caster wheels: Have to use washers to compansate
- Rounded Motor mount bolts:
  - Motor mount mounted to extrusion with 6 bolts - some of the bolts are rounded and cannot be removed
- 2nd layer - to screen - the screen mount was not designed for Momo
- Arm not compatible with back plate, backplate not compatible with screen
- Bought mount, Could not extend to extrusions
- Arm to mount screen forced upwards to fit the screen inside. This however, locks the screen in place.
- Castor wheel mountings - Castor wheel mountings too big for screw.
- Screw to giant washer to washer to attach the caster wheels at the bottom.

2.1.4 1.4 To Dos

1. Redo Lidar Mount - side holder tolerance w
2. Implement a guard for the front LiDAR
3. Implement Rear LiDAR
4. Implement system to improve ease of removing back panel (i.e. magnets, hooks), current back panel is screwed on by 6x rhombus nuts. [In-progress]
5. Implement a charging port for both 55Ah Batteries and 7Ah Batteries

2.2 Electronics

It powers the controlling system, including the lidar, VESC, Teensy, laptop , router as well as two solid state relays Two solid state relays were used.

2.2.1 2.1 Electronic BOM

Power Source

- 2x 12V 55Ah Pb acid batteries (Connected in Series)
- 2x 12v 7Ah Pb acid batteries (Connected in Series)
- Terminal Blocks for Power Distribution
- 3x el cheapo (Taobao) variable voltage Buck (literally worth the buck) Converter (replace pls) (current settings 2x 19V, 1x 12V)

Sensors

- **1x LMS111 LiDAR**
  - 20m range, 270 degree FOV
  - For obstacle detection, and mapping, for navigation
- **Motor Encoders (Came With Motors)**
  - Measure how far motors have turned, important for odometry
- **GY-85 IMU**
  - Important for Odometry, provides another source
- Marvelmind Indoor GPS (Currently not in use, for a 3rd possible global pose data source)

Actuators, Display and Outputs

- **1x Waveshare 13.3" HDMI LCD (H) with case**
  - https://www.waveshare.com/13.3inch-hdmi-lcd-h-with-case.htm
- **2x Flipsky 50A Continuous VESC v4.12 (Using JST-PH)**
  - Normal ESC did not work for our specced motor
- **350W 24V Brushless DC Scooter Hub Motor**
  - https://www.aliexpress.com/item/24V-36V-48V-8Inch-Electric-Wheel-Hub-Motor-350W-Brushless-Non-Gear-Hub-M 32837818637.html
- Teensy 3.2 (Original from PJRC)
- Cheap speakers

Safety and Power Control

- 1x Schnieder DC Circuit Breaker 125V 20A (2-way)
- 1x Schnieder DC Circuit Breaker 125V 16A (1-way)
- 2x CDG1-1DD/40A Solid State Relay
- 1x CDG1-1DD/25A Solid State Relay
- Heat sinks
- **1x A22E-M E-stop**
  - Application: Switch (Option, Others)
  - Conforming series: 22/ 25 switch, indicator light
  - Type: Operating unit (non-illuminated type)
  - Details of shape: Medium type (40)
  - Colour: Red
  - Protection function: IP65 oil resistant type
  - Push button operation: Push to lock/turn to reset
  - Light Source Type: Non-Illumination
  - Voltage Uses Light Source(V): Non-Illumination
- **3x KR2-11 Rocker Switches (Red and Green)**
  - Type: On/Off, DPST
  - Rating: 10A/250VAC
  - Mechanical life: 50,000 cycles
  - Insulation: 500MΩ
- **1x XK-A6/-Y Enclosure Electronics Box (E-box)**
  - Enclosure types: Enclosure with 6 Ø22.5mm holes
  - Shell material:Plastic
  - IP grades:IP65
- **1x AD16-22D/S Green LED Pilot Light Panel Indicator**
  - Voltage: 24V
  - Rated Current: 20mA
  - Material: Plastic, Electric Components
  - Thread Diameter: 22mm / 0.87”
- 1x Terminal Block (within the E-box)

Connectors

- 8x XT90 Connectors (Male and Female)
- 2x 6mm Barrel Jacks
- Laptop power adapter

2.2.2 2.2 Start-Up, Shut-Down Procedure

Start-Up (FULL)

1. Check the battery leads and ensure that the batteries are connected in series
2. Electronics breakers to be Switched to “ON”
3. Set the Green electronics switch to “ON”
4. Motor Breakers to be switched to “ON”
5. Set the E-stop to “OFF”

Shut-down (FULL)

1. Set the E-stop to “ON”
2. Switch the Motor breakers to “OFF”
3. Set the electronics switch to “OFF”
4. Set the electronics breaker to “OFF”
5. Disconnect battery leads

Start-Up (truncated)

This assumes the batteries have been connected beforehand 1. Electronics switch to be set to “ON” 2. E-stop set to “OFF”

Shut-down (truncated)

1. E-stop set to “ON”
2. Electronics switch to be set to “OFF”
3. Disconnect the battery leads

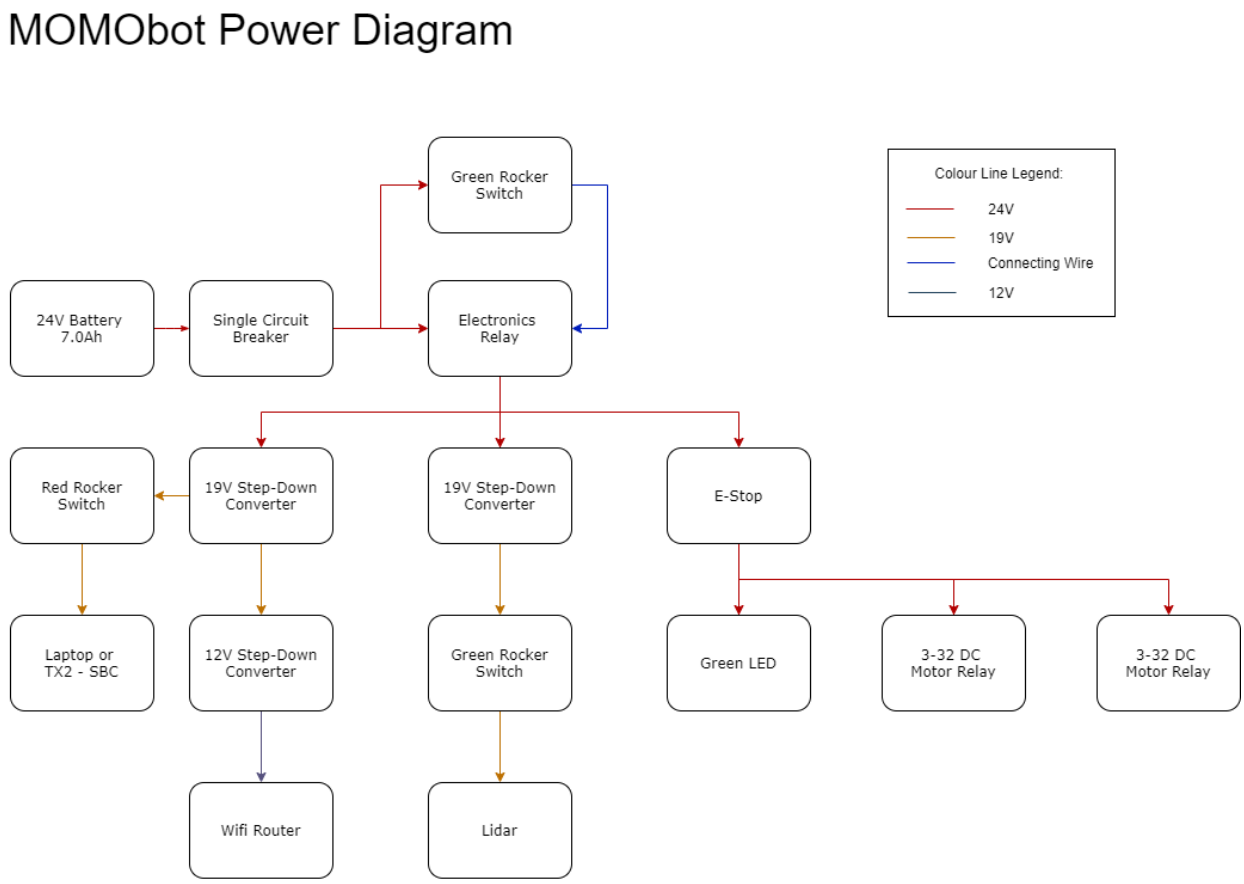
2.2.3 2.3 Gotchas, Hacky Stuff and Things to Take Note Of

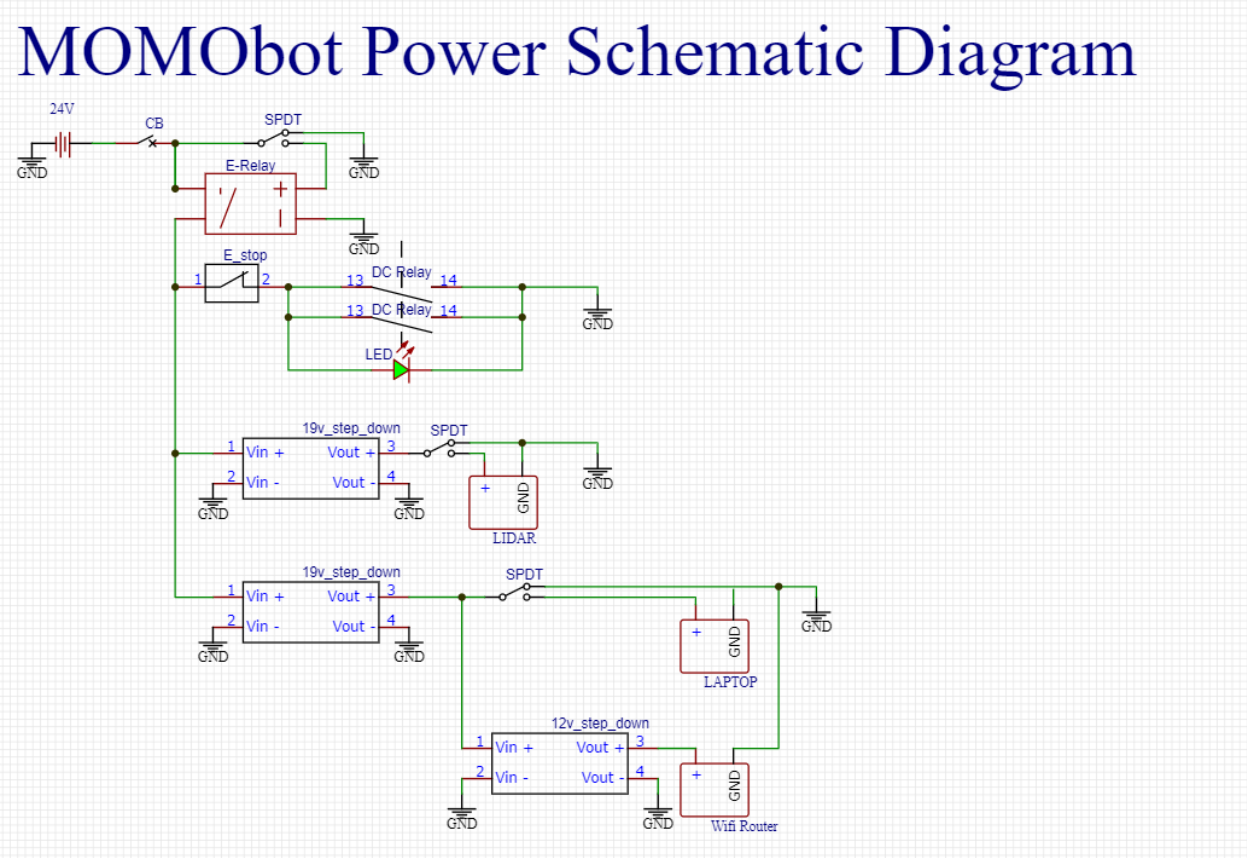
- 2x16AWG wires used to take high current out of battery, as we did not have thick enough wires at the time. The wire usage was not consistent, as some were salvaged PVC wires from the previous bot. Suggested to use all silicone coated wires with low gauge for higher termperature endurance and lower resistance.
- Encoder and PWM input wires from Teensy to the VESC was connected using jumper cables rather than specific JST-PH connectors. Encoder wires were spliced to 2, one to VESC, one to teensy, causing a mess of wires and potential intermittent connections.
- All bulk converter displayers are broken - they show a wrong voltage.
- VESC can be better positioned to be easier USB tunned, and the layout should be revised for easier switch fliping.
- No voltage monitoring circuit included in either of the two electrical system, making monitoring and preparing for recharge difficult. 19v for fully charged, 18.1v for need to charge. Circuit needed to be implemented to prevent either of the batteries from over-discharging.
- The lack of a charging circuit made the life of the maintenance team difficult. Much more troubles of disconnecting the batteries for recharging and connecting back for operation

2.2.4 2.4 Circuit Diagrams

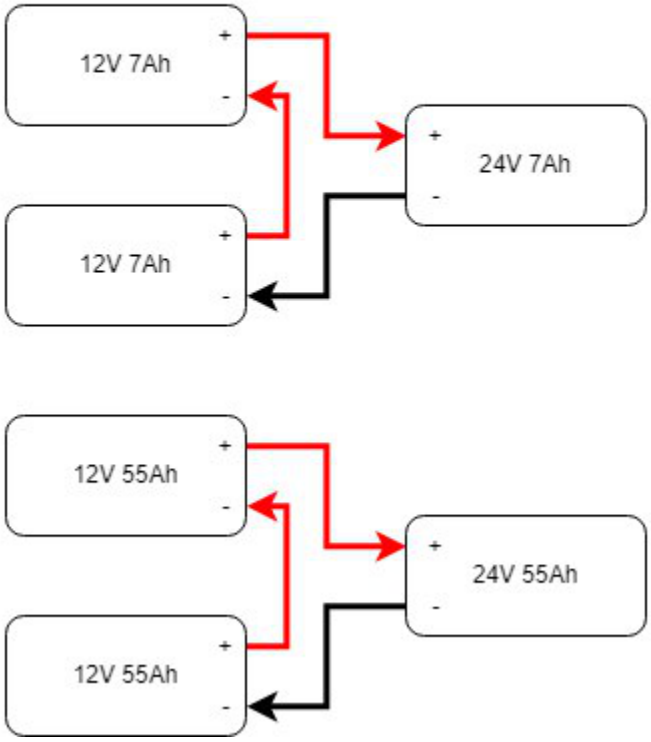
- The robot consists of two power systems
- 24V with smaller battery capacity and 24V with bigger battery capacity
- Both 2 cells in series to boost the voltage for the motors, as well as the lidar

Power

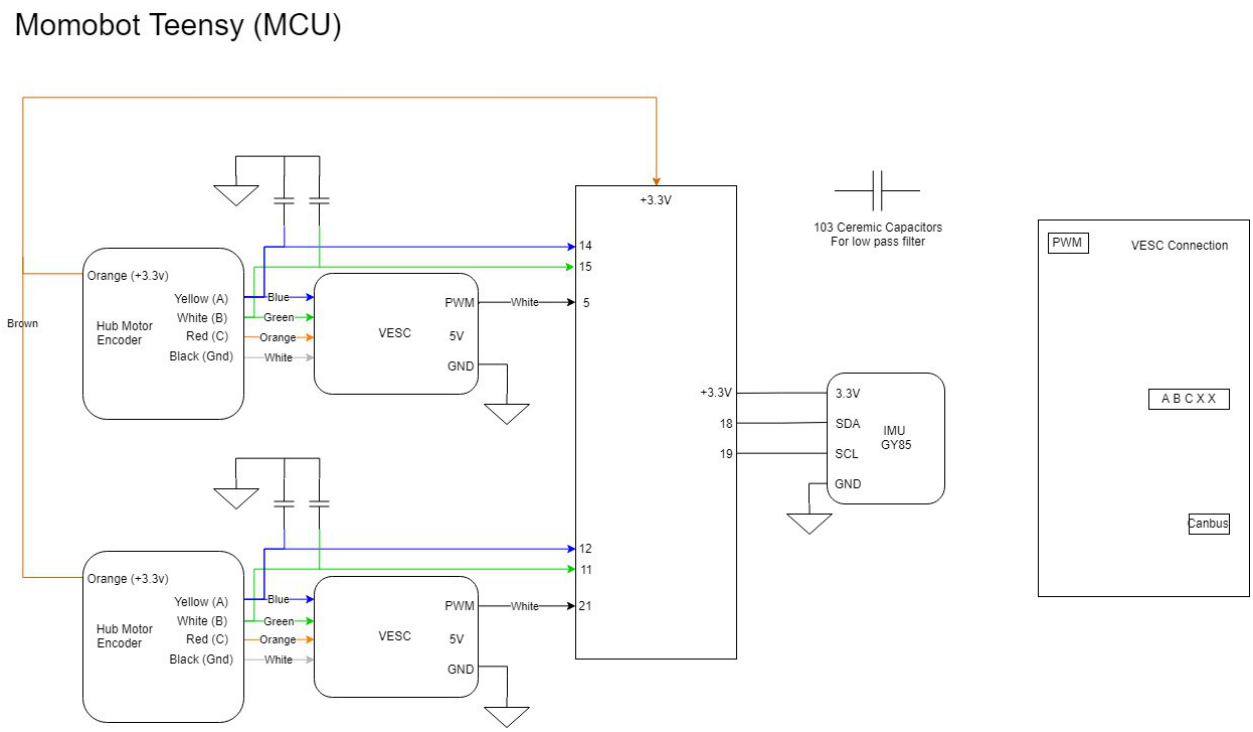

MOMObot Power Diagram
Green Rocker Switch
Colour Line Legend:
24V
19V
Connecting Wire
12V
24V Battery 7.0Ah
Single Circuit Breaker
Electronics Relay
Red Rocker Switch
19V Step-Down Converter
19V Step-Down Converter
E-Stop
Laptop or TX2 - SBC
12V Step-Down Converter
Green Rocker Switch
Green LED
3-32 DC Motor Relay
3-32 DC Motor Relay
Wifi Router
Lidar


MOMObot Power Schematic Diagram
24V
CB
SPDT
GND
E-Relay
E_stop
DC Relay
LED
19v_step_down
Vin +
Vin -
Vout +
Vout -
LIDAR
LAPTOP
12v_step_down
Wifi Router

Momobot Battery Diagram



Electronics Schematic



2.2.5 2.5 Motor Tuning

VESC tuning

Follow link for VESC tuning documentation - *Flipsky VESC 4.12 Documentation*

FOC signal

VESC also has internal PID control which is not modified in the original MOMObot because the PPM signals sent to the motors have been PID-ed in the ROS stack.

- When tuning the PPM signal centre, max and min, ensure that the ROS stack is running.
- The neutral signal will be the “centre”, max forward throttle will be “max” PPM and max reverse throttle “min” PPM.
- The deadband of 10% is selected to allow MOMO to gain enough throttle to overcome initial traction.
- A PPM deadband of < 2% is not recommended as it means any small fluctuation in PPM will command motor response.

  3 days work tuning the settings

Read lots of guides for Duty settings - values that work After tuning the VESC’s PPM and Duty Cycle settings, remember to write the settings else they will not be saved.

Momo charges alot

Possible due to the I component of PID increasing when attempting to Pivot, then when transitioning to a forward movement, the built up I causes a surge in motor response.

Room for improvement

- FOC profile can be improved (FOC settings , will have lots of variables to tune to fit the curve better)
- Positive, negative ramping time

2.2.6 2.6 PID Tuning from the MOMObot side

Edit the config file

1. `roscd momobot/teensy/firmware/lib/config/`
2. `nano momo_base_config.h`
3. Change these parameters:

- Differential drive
- USE_ESC
- Kp, Ki, Kd
- Encoder pins, can be changed in hardware or code

Use the Linorobot PID tuning guide!

2.3 Software

MOMObot is using a modified Linorobot stack!

2.3.1 Overview

- ROS Melodic Morenia
- Jetpack 4.4.1, L4T 32.4.4 [Ubuntu 18.04.5 LTS]

Linorobot to Momobot stack changes:

1. Changed Linorobot to Momobot names in code
2. Changed motor driver code
3. Retargeted several packages

2.3.2 3.1 Pre-Requisites

- ROS Proficiency
- Intermediate Linux/Ubuntu Command Line Proficiency
- Python3 & C++ [Good to have]
- **Linorobot experience**
  - https://linorobot.org

platformio

Would be used to flash the teensy firmware, install via *pip3 install*.

```
$ sudo pip3 install platformio
```

pygame

Ensure that pygame is installed before running *momo_emotions*.

```
$ sudo apt-get install python-pygame
```

ROS Packages

Ensure that the relevant ROS pacakages are installed:

- Robot Localization

```
$ sudo apt install ros-melodic-robot-localization
```

- Ros Serial Python

```
$ sudo apt install ros-melodic-rosserial-python
```

- imu_filter_madgwick

```
$ sudo apt install ros-melodic-imu-filter-madgwick``
```

Teensy

Setup

The teensy driver could be downloaded from their website.

1. Download the udev rules for the teensy and copied it to the `/etc/udev/rules.d/` directory.

```
$ wget https://www.pjrc.com/teensy/49-teensy.rules      #To download the udev rules

$ sudo cp 49-teensy.rules /etc/udev/rules.d/
```

2. Ensure that Arduino has been installed beforehand, if not install the relevant linux package from their webpage. (The Jetson Nano uses the 64bit ARM architecture.)

3. Download the corresponding Teensyduino installer, in our case would be the `AARCH64` package here.
4. Add the execute permission to the installer script and then execute it.

```
$ chmod 755 TeensyduinoInstall.linux64
$ ./TeensyduinoInstall.linux64
```

Configuration

To access the Teensy, change the access permission of the Teensy via *chmod*.

```
$ sudo chmod 777 /dev/ttyACM0
```

Note: Replace `/dev/ttyACM0` to the device port of the Teensy accordingly!

To make it easier to identify teensy, we'll attach a symbolic link to the device name. We'll bind the teensy to a symbolic link (custom name) via udev (So instead of `/dev/ttyACM0`, it would be `/dev/momobase`). Double check the attributes of the device.

```
$ udevadm info -a -p  $(udevadm info -q path -n /dev/ttyACM0)
```

Teensy could be identified via `ATTRS{manufacturer}=="Teensyduino"` in the list, **Note down the values of ``ATTRS{idVendor}`` and ``ATTRS{idProduct}``.** The output would look something like this as shown below.

```
looking at parent device '/devices/70090000.xusb/usb1/1-2/1-2.4':
  KERNELS=="1-2.4"
  SUBSYSTEMS=="usb"
  DRIVERS=="usb"
  ATTRS{authorized}=="1"
  ATTRS{avoid_reset_quirk}=="0"
  ATTRS{bConfigurationValue}=="1"
  ATTRS{bDeviceClass}=="02"
  ATTRS{bDeviceProtocol}=="00"
  ATTRS{bDeviceSubClass}=="00"
  ATTRS{bMaxPacketSize0}=="64"
  ATTRS{bMaxPower}=="100mA"
  ATTRS{bNumConfigurations}=="1"
  ATTRS{bNumInterfaces}==" 2"
  ATTRS{bcdDevice}=="0275"
  ATTRS{bmAttributes}=="c0"
  ATTRS{busnum}=="1"
  ATTRS{configuration}==""
  ATTRS{devnum}=="7"
  ATTRS{devpath}=="2.4"
  ATTRS{idProduct}=="0483"                          <==== Take note of this
  ATTRS{idVendor}=="16c0"                           <==== And this
  ATTRS{ltm_capable}=="no"
  ATTRS{manufacturer}=="Teensyduino"
  ATTRS{maxchild}=="0"
  ATTRS{product}=="USB Serial"
  ATTRS{quirks}=="0x0"
```

(continues on next page)

(continued from previous page)

```
  ATTRS{removable}=="unknown"
  ATTRS{serial}=="2952920"
  ATTRS{speed}=="12"
  ATTRS{urbnum}=="10"
  ATTRS{version}==" 1.10"
```

After that, open up `/etc/udev/rules.d/99-nv-l4t-usb-device-mode.rules` with a text editor.

Important: Double check the files in the `/etc/udev/rules.d/` directory, the device rule file may differ for different devices/computer.

```
$ sudo nano /etc/udev/rules.d/99-nv-l4t-usb-device-mode.rules
```

Add the following line.

```
ACTION=="add", ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="0483", SYMLINK+="momobase"
```

Disconnect and reconnect the teensy for the change to take affect. Listing `/dev/momobase` should be found.

```
$ ls /dev/momobase
```

Flashing the firmware

(Or uploading the code)

To flash the teensy firmware, *platformio* would be used. To do so, simply run `platformio run`.

```
$ cd momobot/momobot_ws/src/momobot/teensy/firmware
$ platformio run --target upload
```

- When platformio is ran for the first time, it would download the relevant manager/tools for the Teensy.

Hint: Facing issues? Check out the troubleshooting section below!

2.3.3 3.2 Setting Up MOMObot

Logging into MOMObot

These settings are for our internal use only. You might have to change it around to whatever WiFi network credentials and addresses your own implementation of the MOMObot stack will use.

1. Login to MOMObot, we use the *SUTD_LAB* WiFi network, with these credentials

   ```
   ssh <USERNAME>@10.21.132.80
   ```

   Or

   ```
   ssh momobot
   ```

Note: Note, this only works if you've configured an SSH Alias for MOMObot!

To do so, put this inside *~/.ssh/config* (Make the file if it doesn't exist using *sudo touch ~/.ssh/config* or *sudo nano ~/.ssh/config*)

```
Host momobot
   # SUTD_LAB
   Port 22
   User <USERNAME>
   HostName 10.21.132.80
```

2. **Access the teensy config file**
    - `roscd momobot/teensy/firmware/lib/config`
    - Opening this file will expose the PID values for tuning

Setting Static IP for New Robots

A static IP is needed so as to be able to ssh into the robot

1. Setup static IP

```
$ route -n # Use this to check default gateway and netmask
```

    - Netmask is the Genmask
    - Gateway is Gateway

    Also check if you need a DNS setup. (SUTD requires this one: 192.168.2.100)

2. Then enable ssh

```
$ sudo apt-get install openssh-server
```

To set Static IP, under WiFi connection settings, click edit connections. Set IPV4, set manual, set static address, Netmask and Gateway, according to *route -n* in the commandline

To work with SUTD wifi for internet, use DNS Server: *192.168.2.100*

Install ROS and Other Packages if needed

1. **Run the install scripts from setup_scripts**
    - Credits: https://github.com/methyldragon/quick-install-scripts
2. **Specific order:**
    - `./convenience_tools_install`
    - `./ros_lino_base_install`
    - `./robot_playground_install`

    This should install your net tools, ROS, as well as the pre-requisite Linorobot stack.
3. Then, copy paste the scripts in the *src* directory inside a ROS workspace, preferably called `momobot_ws`

Setup the ~/.bashrc on MOMOBot as well as your Computer

(Otherwise you won't be able to get any ROS data!!!)

On MOMObot

1. Open up the your prefered shell config in a text editor

Bash (Default)

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

zsh

```
$ sudo nano ~/.zshrc
```

2. Append the following in the shell configuration file:

Important: PLEASE REMEMBER TO CHANGE THE THINGS IN <> & Remember to save also!

Bash (Default)

```
...

source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash

source /home/<USERNAME>/<ROS_WORKSPACE_NAME>/devel/setup.bash

source ~/<ROS_WORKSPACE_NAME>/devel/setup.bash
export LINOLIDAR=lms111
export LINOBASE=2wd

export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:/home/<USERNAME>/<ROS_WORKSPACE_NAME>/src

alias costmap_reset="rosservice call /move_base/clear_costmaps"

export ROS_IP=<IP ADDRESS OF ROBOT>
export ROS_HOSTNAME=<IP ADDRESS OF ROBOT>
```

zsh

```
...

source /opt/ros/melodic/setup.zsh
source ~/catkin_ws/devel/setup.zsh

source /home/<USERNAME>/<ROS_WORKSPACE_NAME>/devel/setup.zsh

source ~/<ROS_WORKSPACE_NAME>/devel/setup.zsh
export LINOLIDAR=lms111
export LINOBASE=2wd
```

(continues on next page)

(continued from previous page)

```
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:/home/<USERNAME>/<ROS_WORKSPACE_NAME>/src

alias costmap_reset="rosservice call /move_base/clear_costmaps"

export ROS_IP=<IP ADDRESS OF ROBOT>
export ROS_HOSTNAME=<IP ADDRESS OF ROBOT>
```

On Computer

1. `sudo nano ~/.bashrc`
2. Append this
    - PLEASE REMEMBER TO CHANGE THE THINGS IN <>

```
ip=$(ip addr show wlo1 | grep -o 'inet [0-9]\+\.[0-9]\+\.[0-9]\+\.[0-9]\+' | grep -o [0-
↪9].*)

export ROS_MASTER_URI=http://<ROBOT_IP_ADDRESS>:11311
export ROS_IP=$ip
export ROS_HOSTNAME=$ip
```

2.3.4 3.3 Running MOMObot Capabilities

This section assumes knowledge of ROS and basic Linorobot packages. This is because MOMObot is based heavily on the Linorobot stack.

Startup and Bringup Process

1. **Ensure that all electronics are connected and turned on**
    - Ensure that the electronics battery has sufficient charge or the LIDAR will fail to function
    - Check to ensure that the LIDAR ethernet cable is connected properly or LIDAR data will not be parsed to the MOMObot stack
    - Disengage the E-stop
2. SSH into Momobot: `ssh <USERNAME>@10.21.132.80`
3. Begin running commands, make sure they're in individual terminals!
    - When a command says **MOMOBOT**, do it in a terminal that is SSHed into MOMObot, or from the MOMObot computer directly
    - When the command says **COMPUTER**, do it on a ground station computer
4. **MOMOBOT:** Start ROSCORE before anything else (in it's own terminal on the MOMObot computer / SSH terminal): `roscore`
    - It is helpful to run *roscore* FIRST, so if the rest of the roslaunch-es die, you can kill and restart them easily without killing the ROS Master
5. **MOMOBOT:** Bringup the MOMObot base controller: *roslaunch momobot bringup.launch*

- Ensure that the robot is not moving during this process as the IMU will be calibrating during bringup.launch. IMU drift will be present if the robot is in motion during this process
- Encoder ticks will be visible in this terminal window. This can be a clear indication that the motor hall sensors are being detected and can also be used for the purposes of PID tuning

6. **COMPUTER: Start the tele operation package in it's own terminal (preferably on the ground station computer, not MOMObot):**
   `rosrun teleop_twist_keyboard teleop_twist_keyboard` - This will enable tele-operation functionality on MOMObot (follow the screen for instructions)

Checking MOMO Functionality

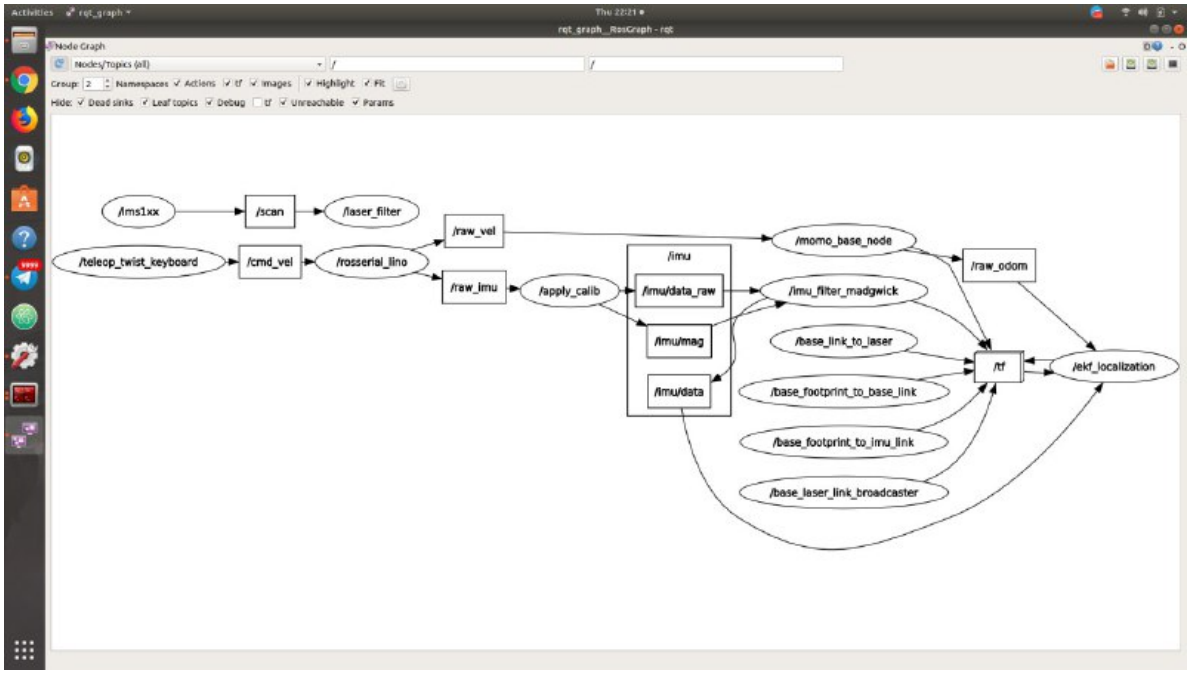
This of course requires that you have MOMObot brought up already.

1. **COMPUTER** - Open rviz:
   - `roscd linorobot/rviz/`
   - `rviz -d odometry.rviz`
   - Under the topic tab on the left menu, it is possible to change the topic between `/odom` and `/raw_odom`.
   - It is also possible to add an additional topic and then set the keep to a larger number (e.g 10000) so as to allow for easier visualization

ROS Graph Debugging Refresher

If visualization of all ROS nodes is required for debugging purposes, run

`rosrun rqt_graph rqt_graph` - This will bringup the rqt graph that visualizes all ROS relations and nodes, allowing for easy debugging.



(ROS nodes as seen in RQT Graph)

- `/scan`: laser data

- `/laser_filter`: custom node for filtering laser data
- `/rosserial_lino`: adapter for connecting to teensy
  - It obtains `raw_vel` which is then sent to the `momo_base_node`
  - This information is then sent to `/raw_odom` for odometry estimations
  - `/raw_odom` information will be sent to `/ekf_localization` to be used for Extended Kalman Filtering estimation of the Robot's location
- `/imu/data`: the raw data from IMU
  - It is also fed into``/ekf_localization``

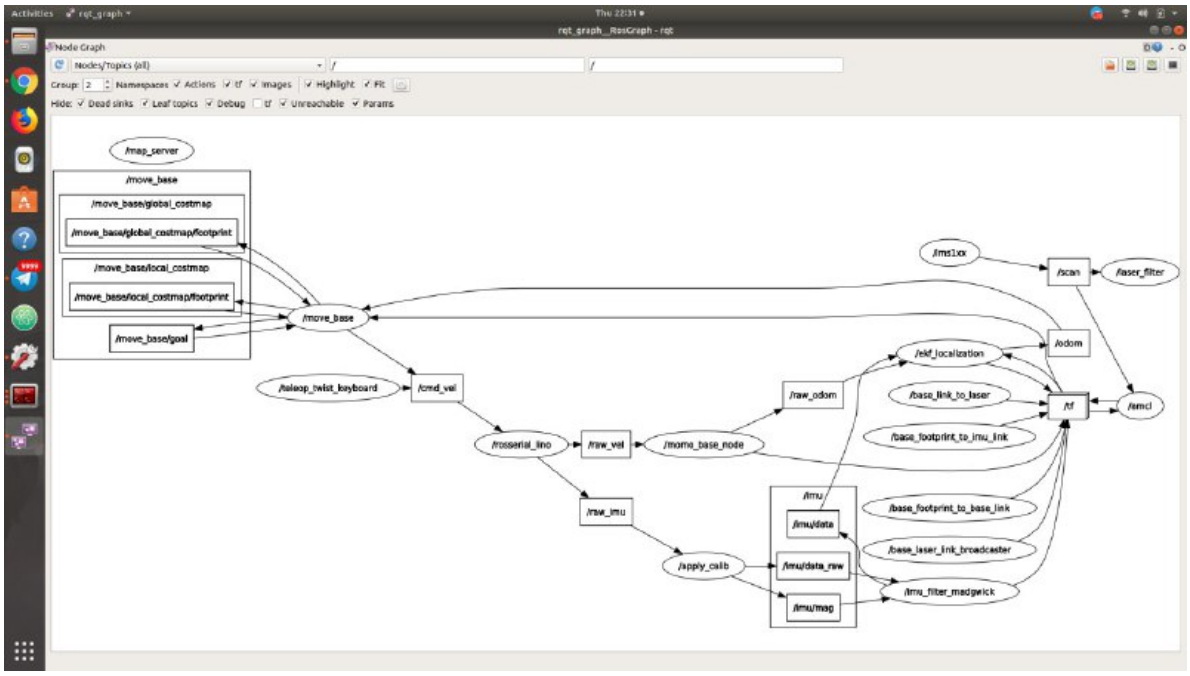
After these commands are ran the following features on Momobot are enabled:

1. Teleop, motors
2. Laser data
3. Encoder and IMU
4. Odometry testing

Running the Navigation stack

Just like the Linorobot!

- Run this command: `roslaunch momobot navigate.launch`



  - `AMCL` is the localisation node that corrects sensor drift using the laser and scanmap
  - `map_server` publishes the map

Autonomous localization and Navigation capabilities enabled and we can tell MOMObot to move by using Rviz or by the using command line:

1. Publishing directly to topics

- Publish goal messages to these topics to get MOMObot to navigate to the coordinate:

  `/move_base/goal`

  `/move_base_simple/goal`

2. Or you can use RViz which is the easiest
   - Just use the RViz interface!

```
$ roscd momobot/rviz
$ rviz -d navigate.rviz
```

     You might have to change some frame IDs and subscribed topics around, but it should work!

   - Get your data by subscribing to those topics above using `rostopic echo <TOPIC_NAME>`

2.3.5 3.4 Tuning MOMObot

Changing The Map

If you happen to have a new map that you want MOMObot to use, simply add the map and its corresponding `.yaml` to the `momobot/maps` directory.

Then, point the `map_server` node to the proper map by editing the `navigate.launch` file!

```
roscd momobot/launch
nano navigate.launch
```

You'll want to replace this line, changing the map_file argument!

```
<arg name="map_file" default="$(find momobot)/maps/<YOUR_MAP_NAME_HERE>.yaml"/>
```

EKF Tuning

Method

1. Mark out a square of 3m x 3m using tape
2. Using tele-operation, drive the robot in a square
3. **on Computer in the same terminal**
   - `roscd/linorobot/rviz/`
   - `rviz -d odometry.rviz`
   - This can be used to view the distance the robot has think it has travelled
   - **In Rviz**
     1. Click on Odometry in the left topic pane (it is possible to select /odom and /raw_odom)
     2. It is recommended to add another topic and indicator for /raw_odom separately to visualize it alongside the filtered odometry
     3. Change the shaft length, radius and etc as necessary (set the color to 0, 255, 0 to ensure that the arrows are distinguishable from /odom arrows)
4. Now you can edit the `robot_localization.yaml` EKF parameters!

```
roscd momobot/param/ekf
nano robot_localization.yaml
```

Tuning Navigation Parameters

- `roscd momobot` to cd into the momobot stack
- `roscd momobot/param/` change these files for navigation parameters

Tuning Localization Parameters

- `roscd momobot/launch/include`, then `nano amcl.launch`
- Parameters of interest: - laser_max_range - min_particles - max_particles - odom_alpha1 (Rotation noise from rotation) - odom_alpha2 (Rotation noise from translation) - odom_alpha3 (Translation noise from translation) - odom_alpha4 (Translation noise from rotation)

2.3.6 3.5 Console Commands

Note: Each command is in each individual terminal, opened INSIDE MOMOBOT (i.e. in a terminal that is SSHed into MOMOBOT)

Set Pose

```
echo resetting pose... && rostopic pub /initialpose geometry_msgs/
↪PoseWithCovarianceStamped '{header: {frame_id: "map"}, pose: {pose: {position: {x: 76.
↪401, y: -15.676, z: 0.0}, orientation: {x: 0.0, y: 0.0, z: -0.10356, w: 0.99462}},␣
↪covariance: [0.25, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.25, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
↪0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0, 0.0, 0.0, 0.06853891945200942]}}' --once && echo clearing costmaps... &&␣
↪rosservice call /move_base/clear_costmaps && echo done!
```

Set Goal Pose for Autonomous Navigation

```
rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped '{header: {frame_id: "map"}
↪, pose: {position: {x: 1.93851232529, y: -0.423947900534, z: 0.0}, orientation: {x: 0.
↪0, y: 0.0, z: -0.125194964757, w: 0.992132158938}}}' --once && rosservice call /move_
↪base/clear_costmaps
```

Computer load up rviz map again on your costmap *roscd/linorobot/rviz/ rviz -d odometry.rviz*

Important: Important to clear costmaps frequently as there will be phantom obstacles created the longer MOMO is in operation.

2.3.7 3.6 Mapping

- `roslaunch momobot slam.launch` on **COMPUTER**
- `rosrun map_server map_saver` to save the map in the current directory. This saves your map files into .yaml files and .pgm files. .pgm can be editted in photoshop, etc like a .png file.
- If you need to change the map name, do it in the `.yaml` file.

Setting your maps

- Under `roscd momobot`, name the map properly as the `.yaml` files will be used to load map files here.

2.3.8 3.7 Exporting Display to MOMObot from Computer

- `export DISPLAY:0` enables you to run commands on other computer instead of your computer. Ensure that you are on the right terminal **MOMOBOT**
- `rosrun momo_emotions cmd_vel_face_tracking.py` to run face cmd_vel tracking script on **MOMObot** from the **Computer Terminal**
- This allows the script to be started during events or situations where it may be difficult to access the on-board computer

2.3.9 3.8 Semantic Pose and Semantic Pose Sounder Packages

Note: The `semantic_pose` package allows the robot to map robot coordinates to named locations. You can use this alongside `semantic_pose_sounder` to **let MOMObot play voice lines upon entering a particular named location!**

- Must be on **MOMOBOT**: `roslaunch semantic_pose_sounder semantic_pose_sound.launch`
- This loads all the ros parameters required.
- Edit them in the associated *config.yaml* in the package (Check it out with `roscd semantic_pose_sounder`)
- The location parameters are set in the following format: `'location: [boundingpt1, boundingpt2, boundingpt3...]'`
  - Set as many bounding points as necessary but **ensure that they are in clockwise or anti-clockwise direction**
  - Bounding points define the boundary for a location (robot will be considered in that location if it is within the boundary)
  - Get bounding points from the map by running navigate.launch. The coordinates for various points on the map can be read easily
  - You can then define their names, which will be the strings published to */location* when the robot is localised within those map zones
  - You can also define the corresponding MP3s to play!
- To check MOMObot's current location: `rostopic echo location`

2.3.10 3.9 Software To-Dos

- Laser scan matcher
- Visual odometry
- Depth camera integration
- **Rear LiDAR integration (Need to combine the /scans somehow)**
  - Once that's done we can swap to the DWA planner to allow MOMObot to reverse!

2.3.11 3.10 Troubleshooting

libusb-0.1.so.4 not found (platformio issue)

If you encounter the following issue:

```
teensy_loader_cli: error while loading shared libraries: libusb-0.1.so.4: cannot open␣
↪shared object file: No such file or directory
*** [upload] Error 127
```

It could fix be downloading the required library through `apt install`.

```
$ sudo apt-get install libusb-0.1-4
```

SDL Issue

```
[CRITICAL] [Window      ] Unable to find any valuable Window provider. Please enable␣
↪debug logging (e.g. add -d if running from the command line, or change the log level␣
↪in the config) and re-run your app to identify potential causes
egl_rpi - ImportError: cannot import name 'bcm'
  File "/usr/lib/python3/dist-packages/kivy/core/__init__.py", line 63, in core_select_
↪lib
    fromlist=[modulename], level=0)
  File "/usr/lib/python3/dist-packages/kivy/core/window/window_egl_rpi.py", line 12, in
↪<module>
    from kivy.lib.vidcore_lite import bcm, egl

sdl2 - ImportError: libSDL2_image-2.0.so.0: cannot open shared object file: No such file␣
↪or directory
  File "/usr/lib/python3/dist-packages/kivy/core/__init__.py", line 63, in core_select_
↪lib
    fromlist=[modulename], level=0)
  File "/usr/lib/python3/dist-packages/kivy/core/window/window_sdl2.py", line 27, in
↪<module>
    from kivy.core.window._window_sdl2 import _WindowSDL2Storage

x11 - ModuleNotFoundError: No module named 'kivy.core.window.window_x11'
  File "/usr/lib/python3/dist-packages/kivy/core/__init__.py", line 63, in core_select_
↪lib
    fromlist=[modulename], level=0)
```

(continues on next page)

(continued from previous page)

```
[CRITICAL] [App         ] Unable to get a Window, abort.
```

The following issue is caused by a missing sdl2 library, could be resolved by installing the python3-sdl2 package.

```
$ sudo apt install python3-sdl2
```

D-Bus Issue (Jetson TX2)

If there are issues running programs that utillises the kivy lib as shown below:

```
[INFO   ] [Window      ] Provider: sdl2(['window_egl_rpi'] ignored)
dbus[24434]: arguments to dbus_message_new_method_call() were incorrect, assertion "path␣
↪!= NULL" failed in file ../../../dbus/dbus-message.c line 1362.
This is normally a bug in some application using the D-Bus library.

  D-Bus not built with -rdynamic so unable to print a backtrace
Aborted (core dumped)
```

Add `DBUS_FATAL_WARNINGS=0` before the program name: (Reference)

```
DBUS_FATAL_WARNINGS=0 {python_script}
```

Replace `python_script` with the program name.

2.4 Flipsky VESC 4.12 Documentation

[Credits]

This is a documentation for how to use and configure the flipSky FSESC 4.12 used on the MOMObot. The exact store page can be found in this link.

2.4.1 Contribution Guidelines

If you update this guide, kindly send a pull request to the original repo listed in the credits!

2.4.2 Introduction

Vedder Electronic Speed Controller (VESC) is a highly configurable (largely) open source ESC that sees a wide variety of applications by hobbyists in the electric scooter or skateboard domain. This makes VESC a suitable ESC for MOMObot as it uses scooter motors for movement. VESC is an excellent choice for hobby robotics of this scale as it is:

- Highly configurable with both conventional BLDC control as well as FOC support
- Supports a wide variety of motors and configurations
- Self-tuning which allows even unmarked motors from Aliexpress or Taobao to be calibrated properly
- Generally has an internal BEC, MOSFET and etc which makes it an all in one package
- Able to control motors using PPM, analog, UART, I2C, USB, CAN-bus

- Effective even with sensorless motors
- Small and light for what it can achieve

2.4.3 Flipsky VESC 4.12

The Flipsky VESC 4.12 is a surprisingly robust product that does not require a laboratory power supply for first start configuration (contrary to some VESC guides on the internet). Despite being made in China, it offers reliable performance that matches some of the more expensive ESCs on the market. It is also one of the few remaining VESC providers that has remained backward compatible with the open source VESC software (many providers have converted the VESC stack to a more proprietary one).

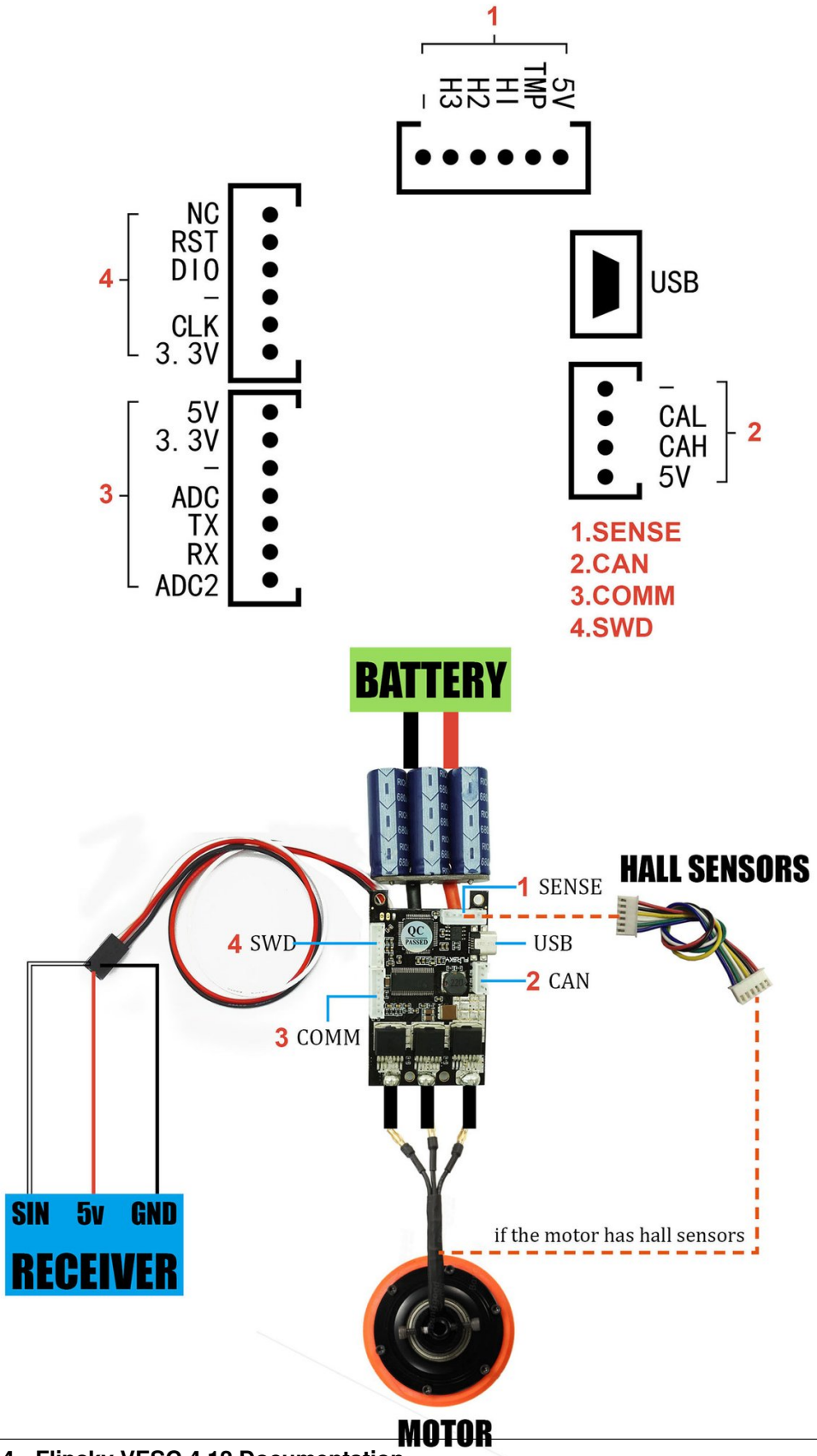
Firmware

Flipsky VESC 4.12 has a hardware version of 4.12 and a software version of 3.4. This means only specific version of VESC tools can be used with the ESC. Because of the finicky nature of download links on the internet (as well as some requiring registration), I have uploaded the specific VESC tool (windows) proven to work with the Flipsky VESC 4.12 here. The exact version is VESC tool version 0.95.

Specifications

- Hardware: v4.12
- Firmware: v3.40
- PCB: 4 layers, size: 40*60mm
- Current: 50A continuous / 240A peak
- Voltage: 8V-60V (cells: 3-13S LiPo)
- Recommended: 10S battery
- BEC: 5V@1.5A
- BEC type: Internal driver support
- Timing: Software Calibration
- Motor control interface: PPM signal (RC servo), analog, UART, I2C, USB or CAN-bus
- Cutoff Voltage: Programmable
- Frequency: PWM Input
- Governor: No
- Weight: 80g
- FSESC4.12 Size: 60x40x20mm
- Programming card: No
- Reverse Direction support: Yes

Connections

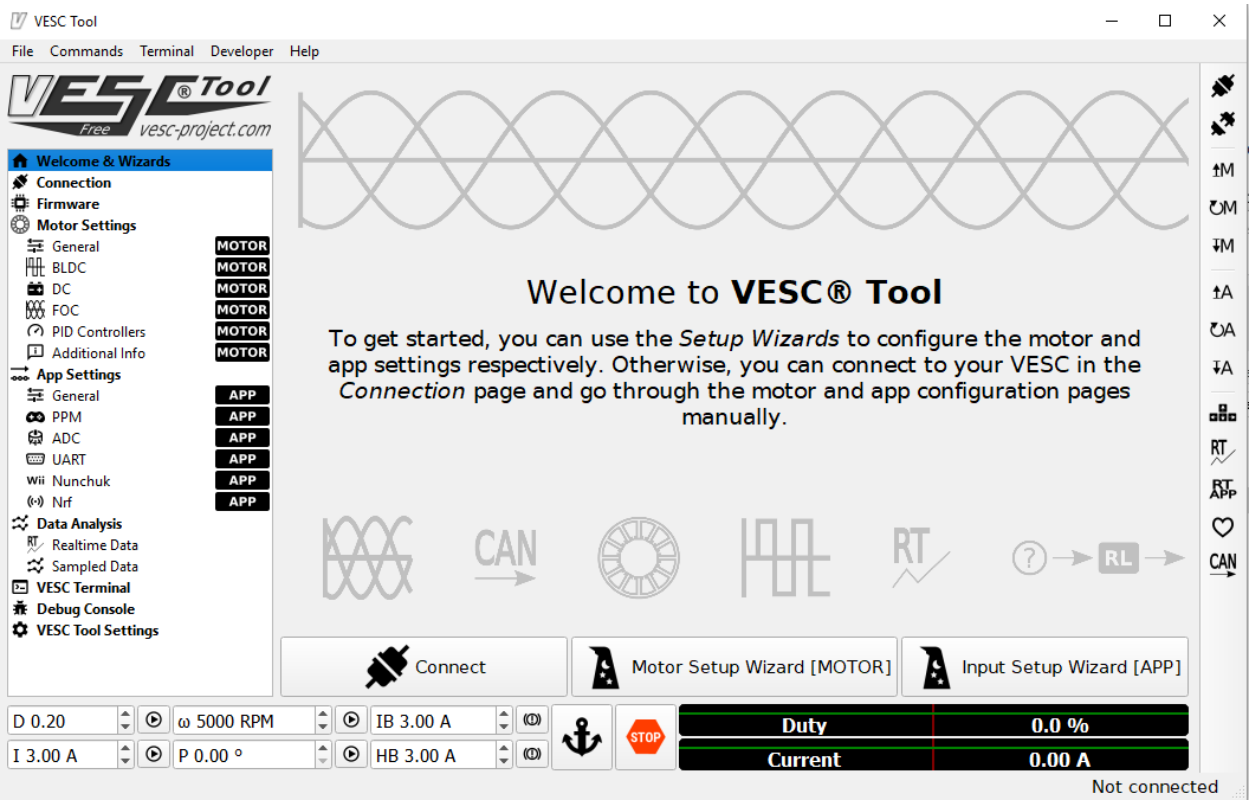


2.4.4 Software Setup

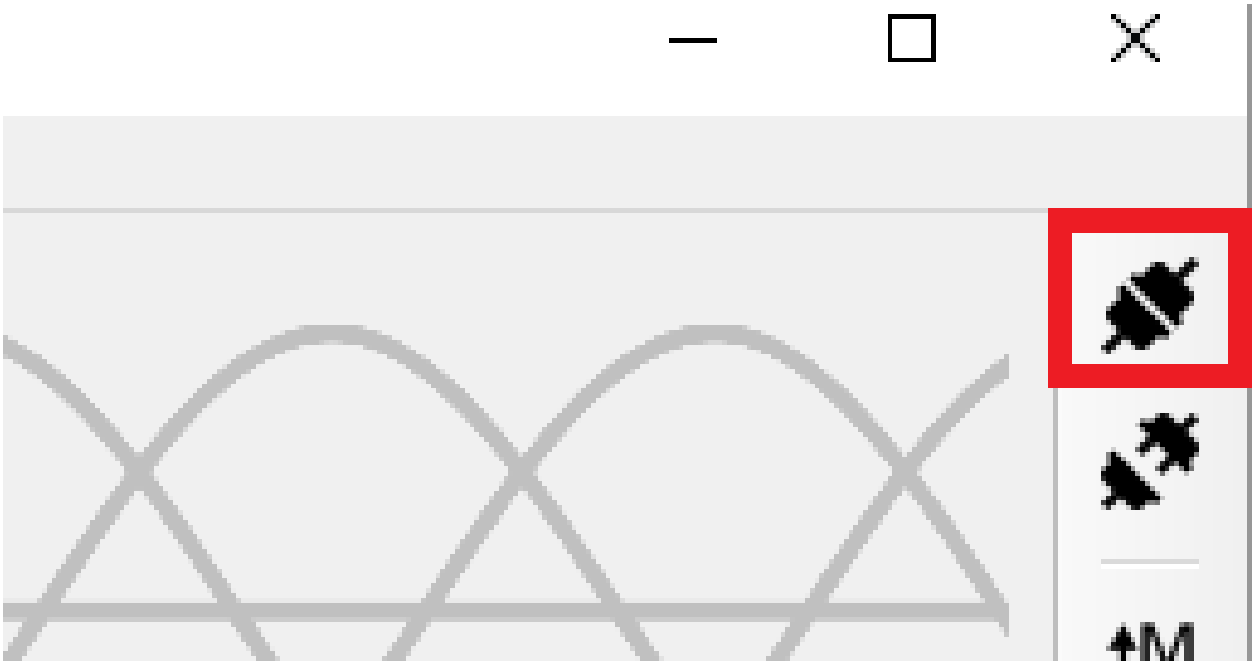
Motor Connection and Setup

Some information was referenced from the Motor Wizard Guide.

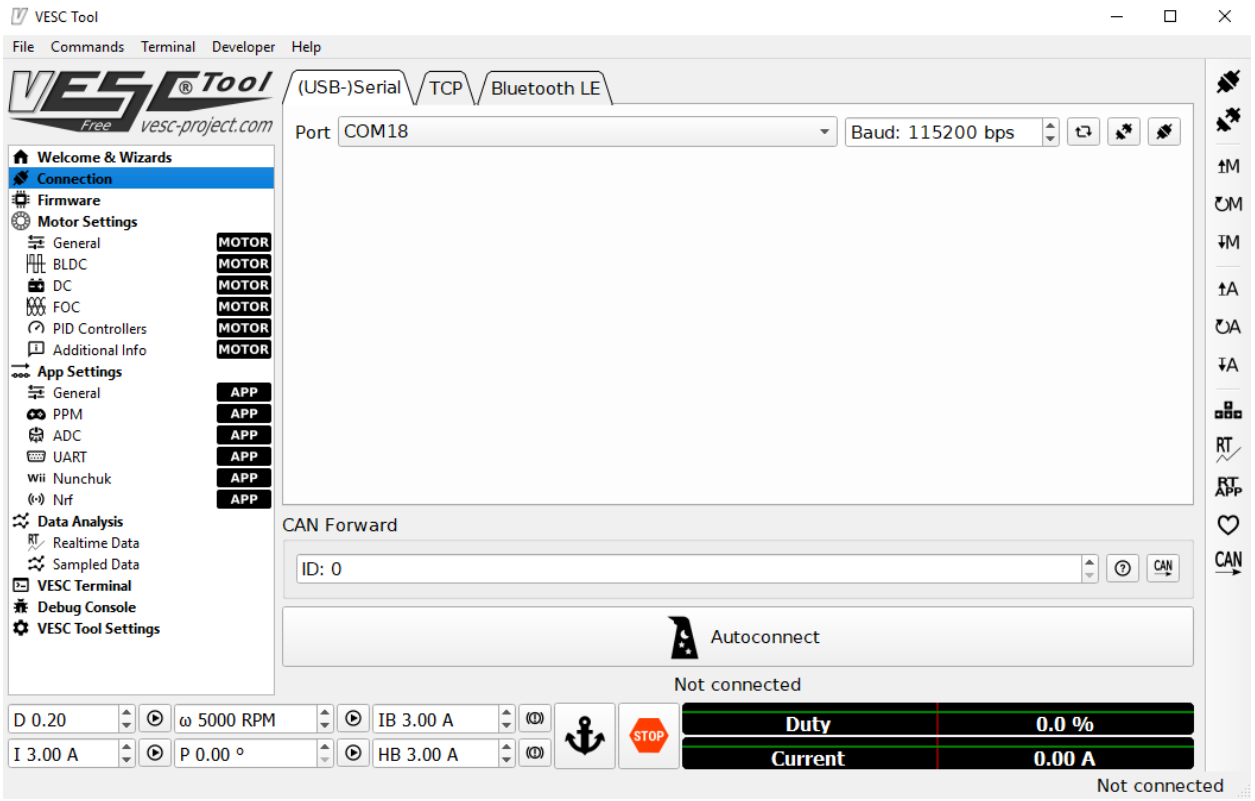
1. Start `vesc_tool_0.95.exe` in administrator mode
    - VESC Tool will prompt to update, don't update
2. The tool should look like this:



3. Plug in the one of the Flipsky VESCs to the computer via mini-USB cable.
4. Click on the connection icon in the top right corner.
    - If prompted to update the software of the VESC, proceed to update!

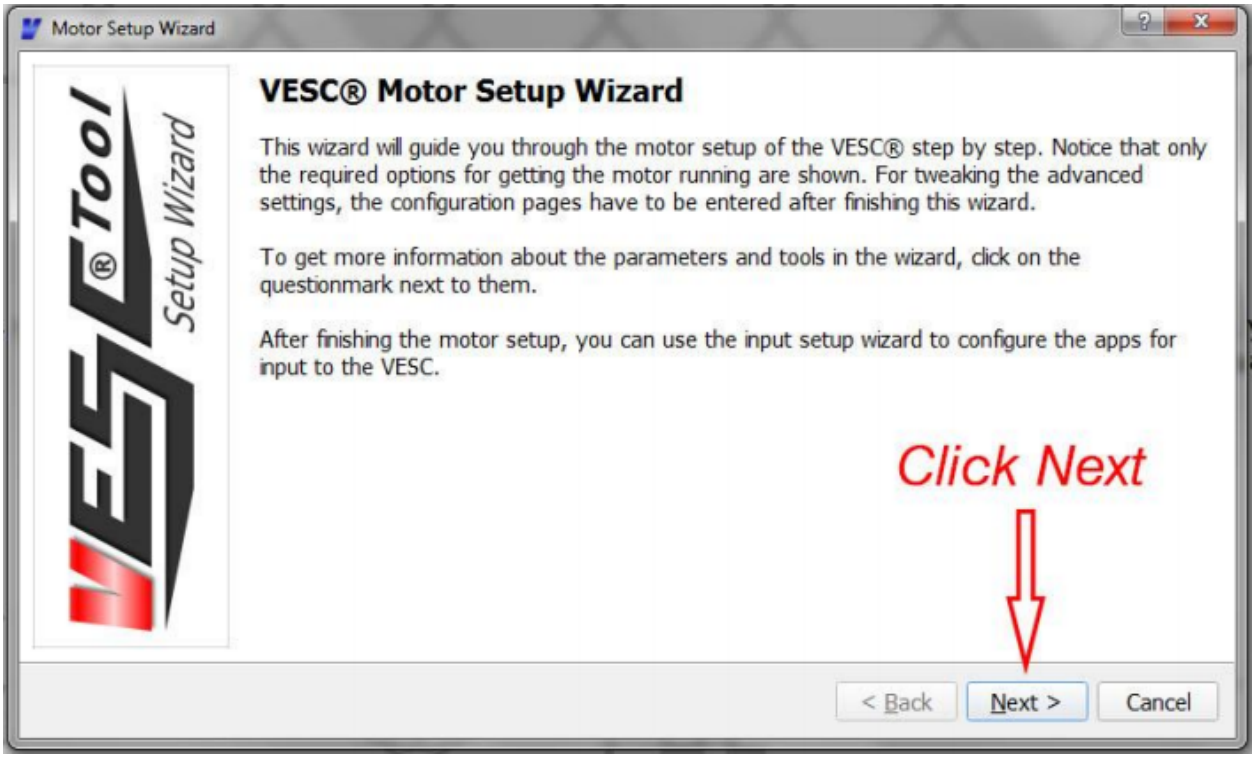


5. Alternatively, click on the `connection` tab in the left menu and select `Autoconnect`:

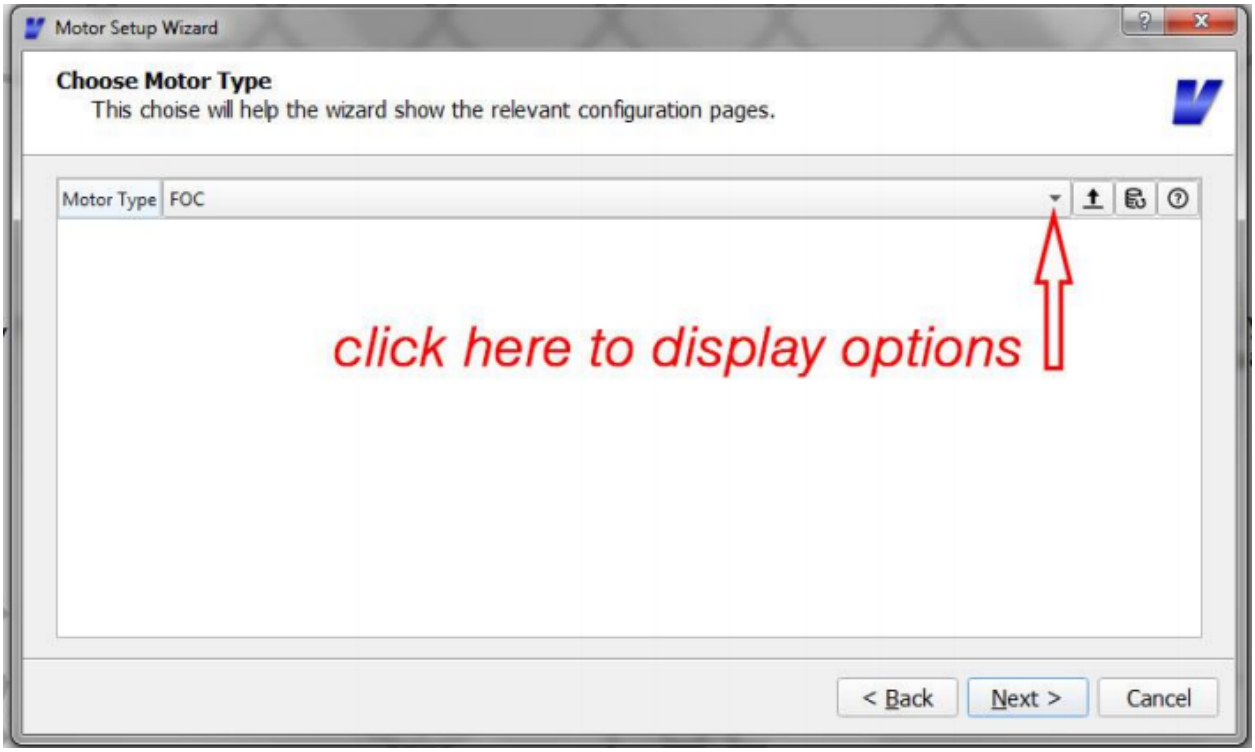


6. The VESC should now have a lit green LED. If there isn't a lit LED, check if the drive has been installed correctly or if the product is defective

7. The motor setup wizard should appear, follow the steps accordingly



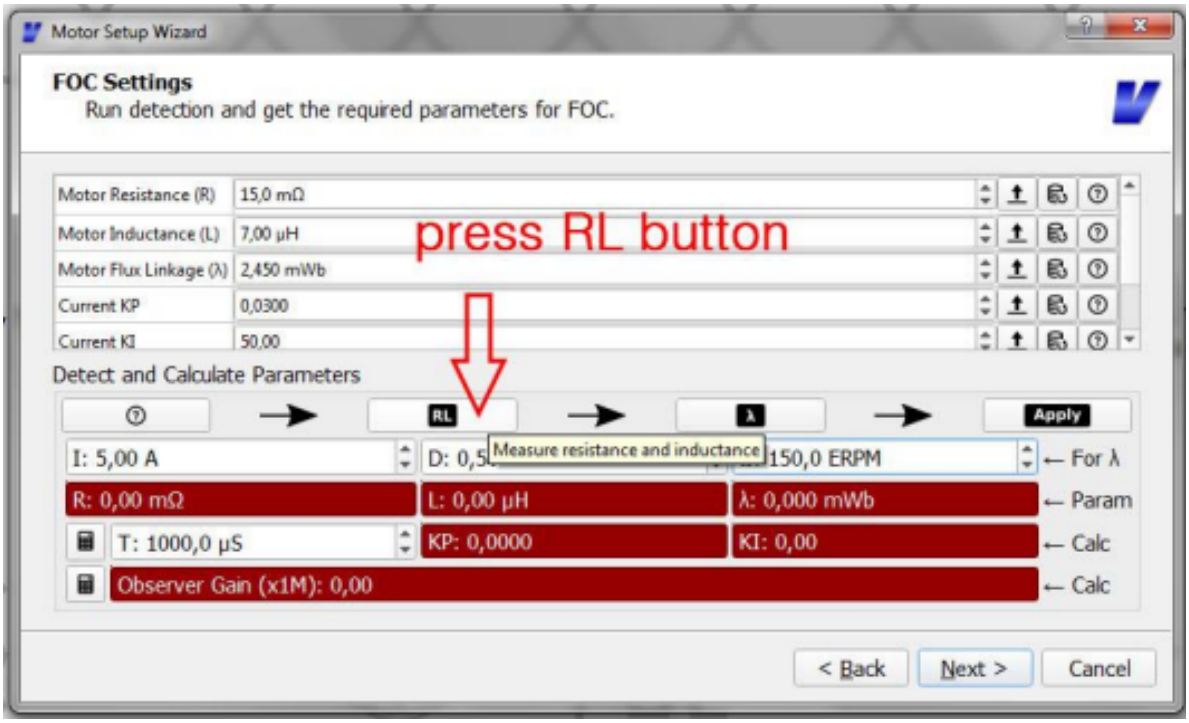
8. In our case, we will be selecting Field Oriented Control (FOC):



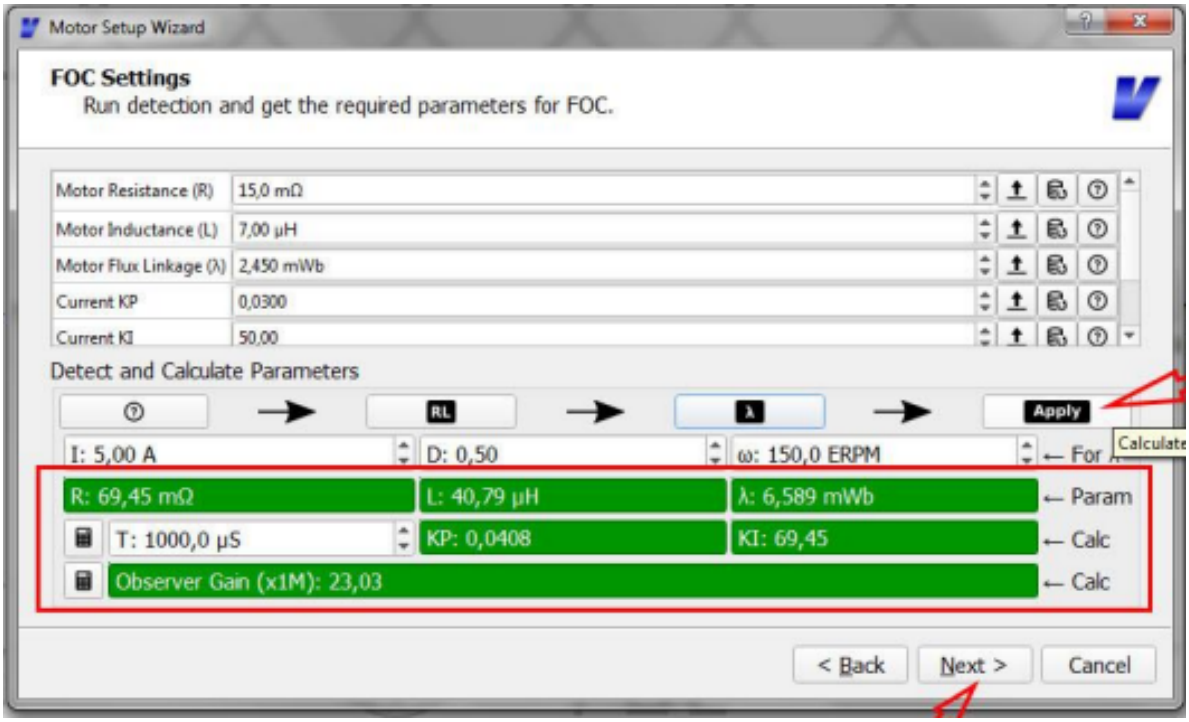
9. Under the motor setup wizard, enter the following parameters for the scooter motors used by the MOMObot:
    - Motor Current Max: 20A
    - Motor Current Max Brake: -20A

- Battery Current Max: 60A
- Battery Current Max Regen: -20A

10. Under the battery cutoff voltage wizard, just enter the correct number of cells (12S) and ensure that Lithium Ion is selected.
    - Click on `Apply` and the battery voltage settings will be set automatically
11. In the sensor mode, select `Hall Sensors`.
12. Prop up the robot such that the wheels are not touching the ground (ensure that the wheels are attached to ensure the correct loading on the motors).
13. Proceed to the FOC settings wizard.



- Press the RL button, the motor will begin to vibrate
- Then press the lamba button, the motor will begin to spin
- After all red boxes turn green (indicating that VESC has detected the motor parameters), click on `apply` to apply the motor values.



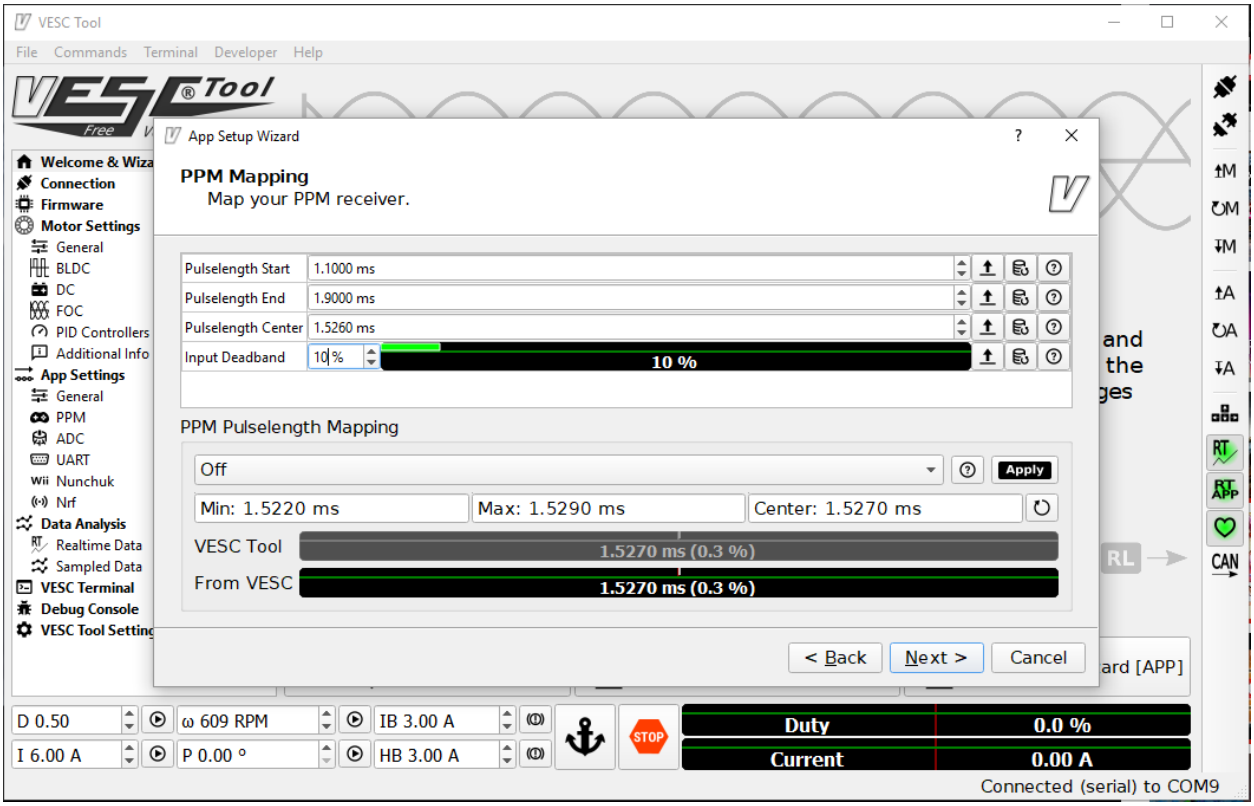
14. Under the Hall Sensor Detection wizard, ensure the motors can rotate freely.
    - Then hit the play button.
    - After the motors rotate and all the Hall sensor values have been detected, click `apply`.
15. Finish the motor setup wizard

App Settings

Next the app settings has to be configured accordingly. For MOMO control, we will be using PPM directly from the Teensy. Information for configuring VESC for RC can be found at the [RC configuration guide](<https://cdn.shopify.com/s/files/1/0011/4039/1996/files/Input_Setup_Wizard_for_single_VESC_using_a_PPM_signal_radio_controller.pdf?11313553160569203029>).

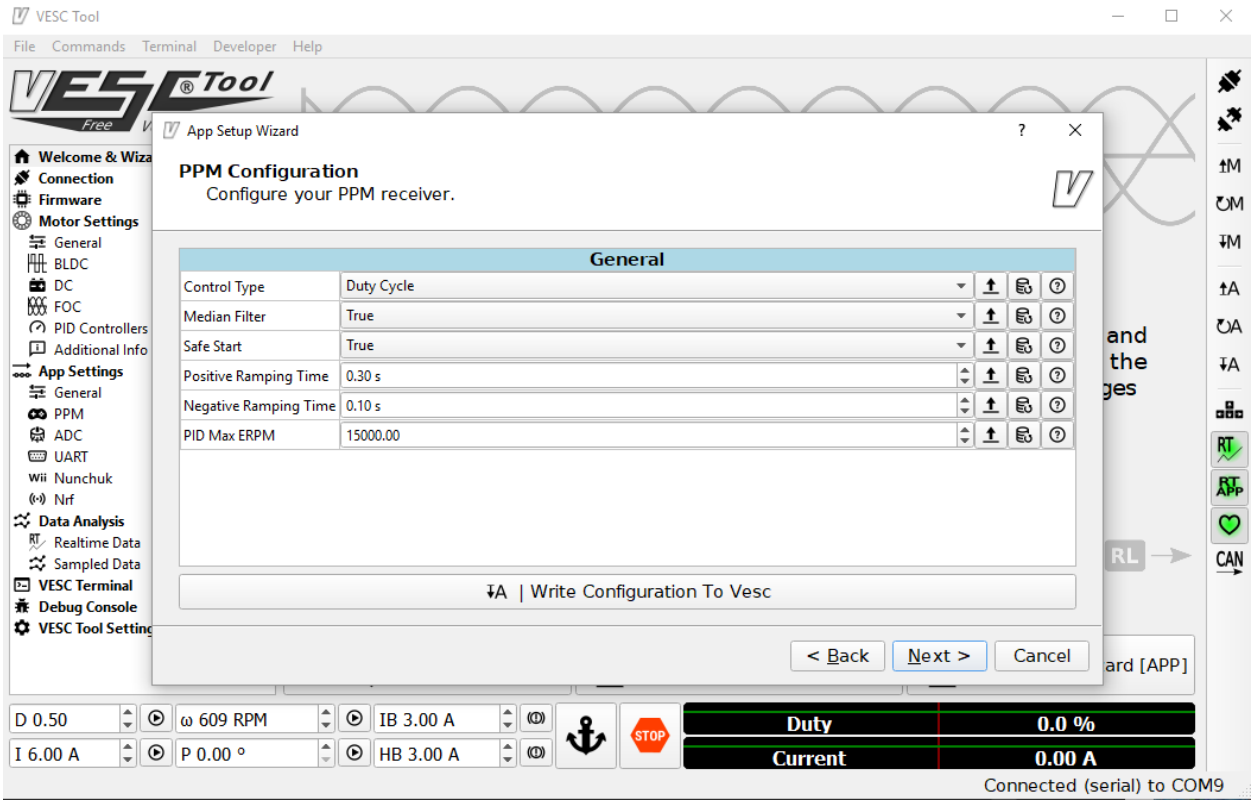
1. Navigate to the *App Settings* in the left menu and select `Input Setup Wizard` at the bottom.
2. Configure the general app settings as follows:

3. Configure the PPM settings as follows:



(Alternatively, using ROS / Teensy, send a neutral signal and monitor what PPM is being detected and use that as the center instead. Then apply the maximum forward and reverse and use those as the start and end values accordingly.)

4. Finally, set the Duty Settings as follows:



5. Remember to write the configuration to the VESC.

That's it! The VESC should be configured for FOC.

2.5 Circuit Diagram Draw.io Links

Here are the links used to generate the various diagrams.

- Momobot Motor Power Diagram
- Momobot Power Diagram
- Momobot Battery Diagram
- Momobot Teensy (MCU)

2.6 Contributions

This documentation has been put together with the combined efforts of members of the **SUTD Organisation of Autonomous Robotics**

2.6.1 v1.1.0: 2019 - 2020

- Photon
- robobdo
- Jia Hwee
- Jerremy
- darthnoward

2.6.2 v1.0.0: 2018 - 2019

- methylDragon
- Shine16
- Fasermaler
- imossim
- Bryan Kong
- Low En
- Senrli

2.7 License

The license could be found here.