


F1TENTH Boot Camp

Lecture1 : Introduction to F1tenth

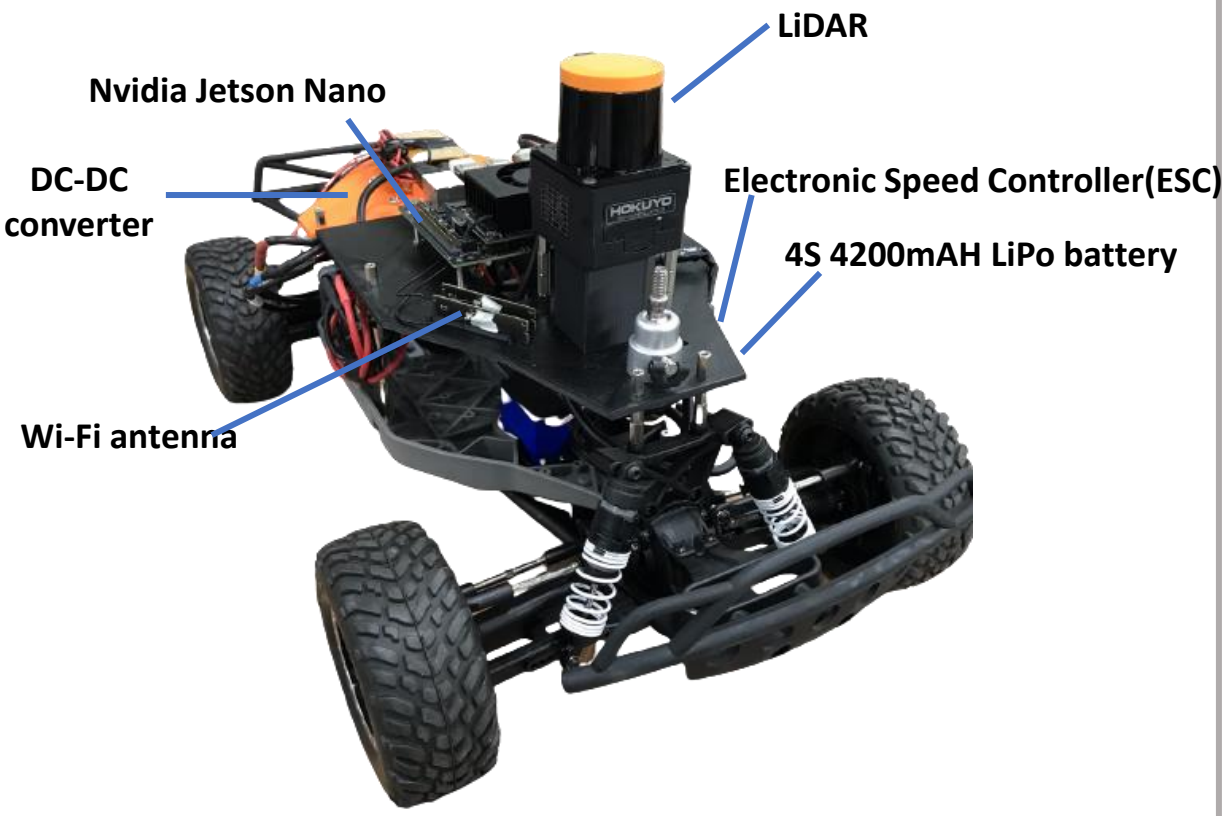
Mechanical Engineering

Cheolhyeon Kwon

2024.09.26

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Creating Autonomous Vehicle



F1/10

Autonomous Racecar

System components

System architecture

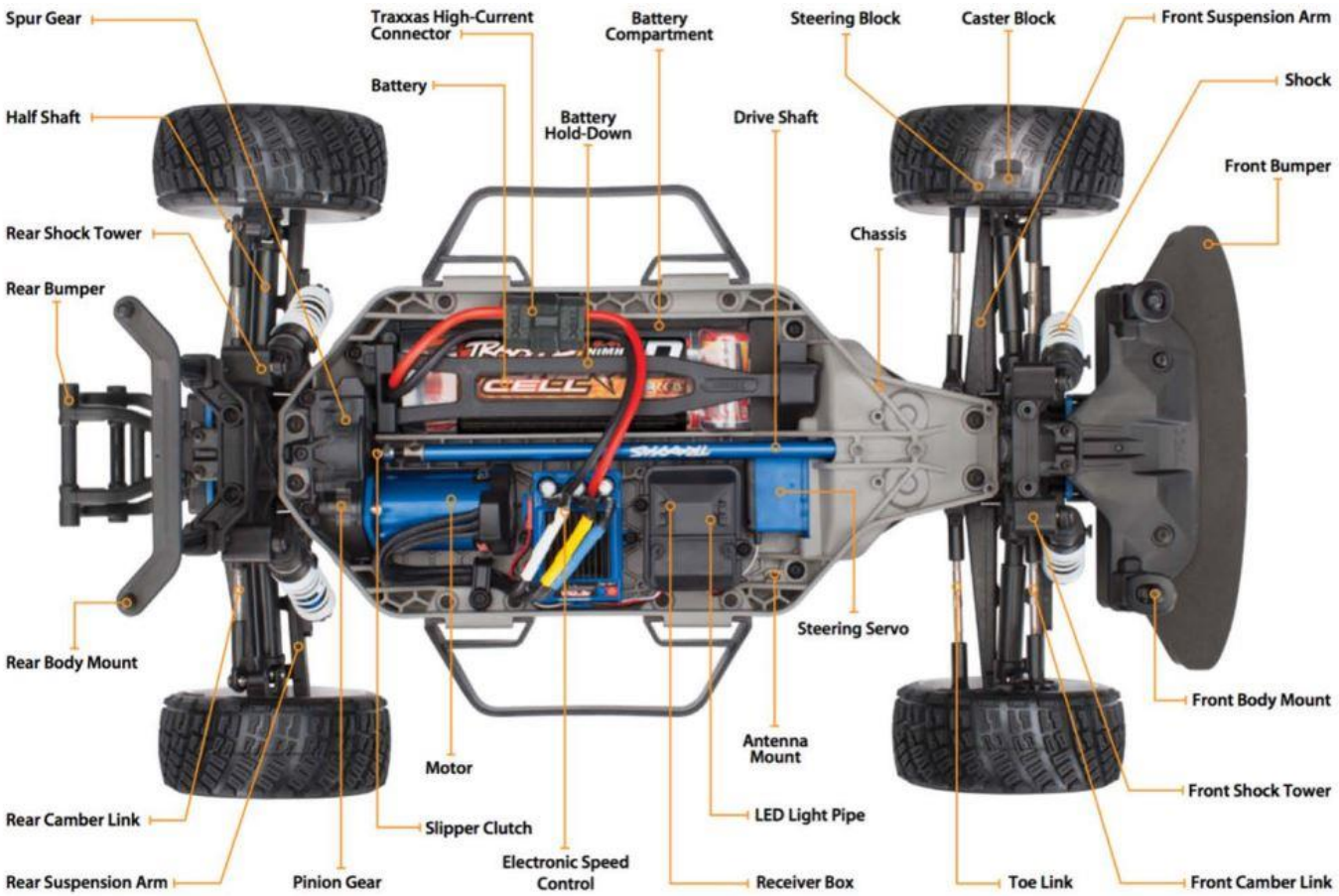
Configuring the VESC

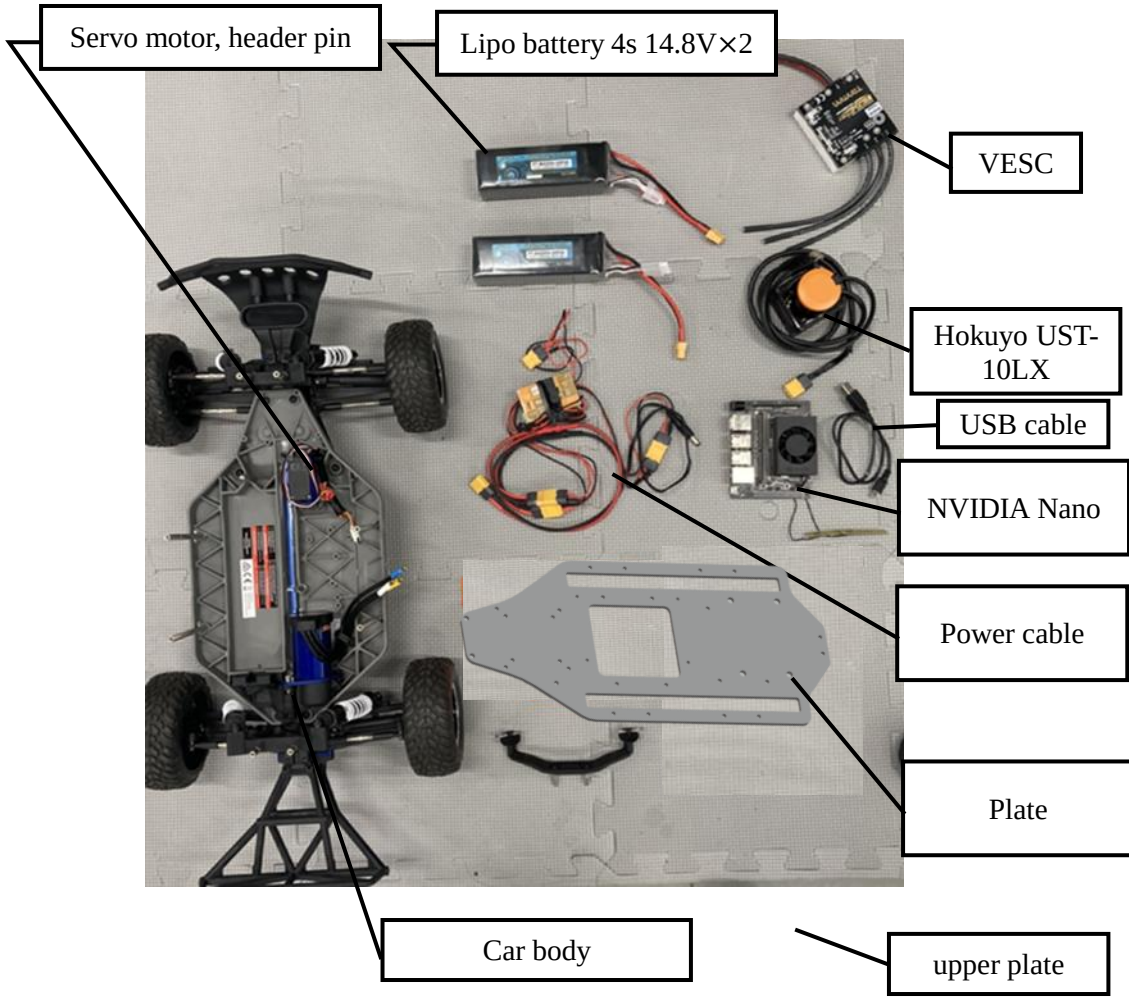
System components & System architecture

The kit each team receive contains :

1. F1/10 Autonomous Racecar (BLDC motor + Servo motor)
2. 2 Batteries - 4Cell LiPo battery
3. Jetson Nano Development Kit
4. Hokuyo UST-10LX
5. UBEC DUO DC-DC converter
6. VESC6 MK VI
7. Xbox wireless Joystick controller

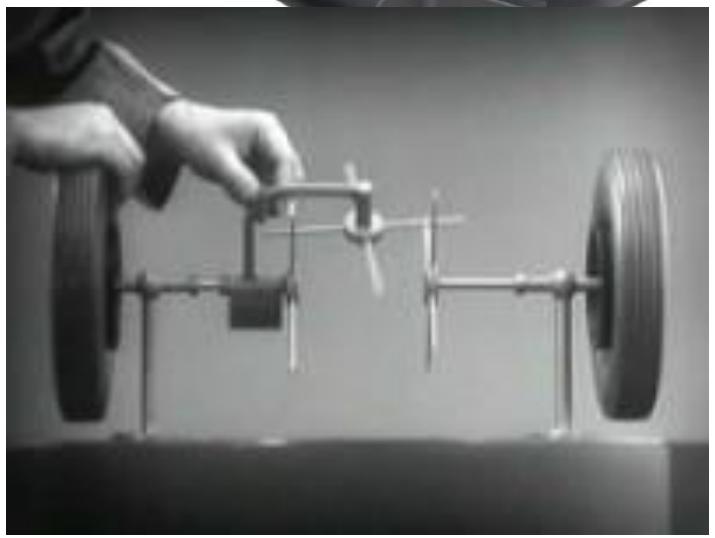
F1/10 scale RC race car





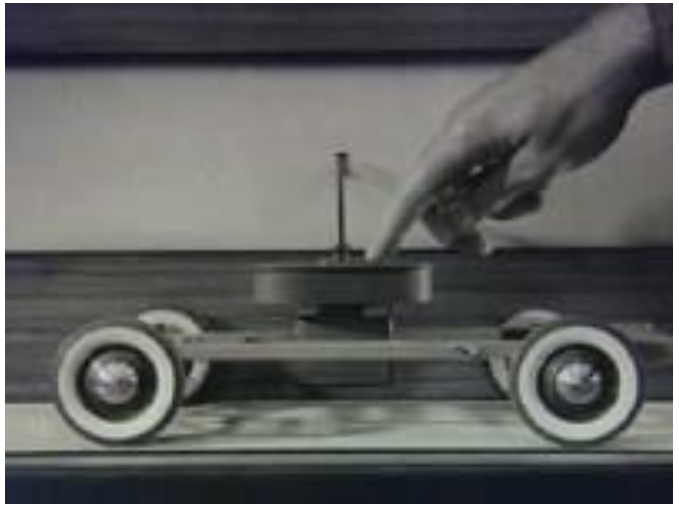
F1/10 scale RC race car chassis

- Drive shaft
  - telescoping, universal-joint driveshaft
- Differential Gear



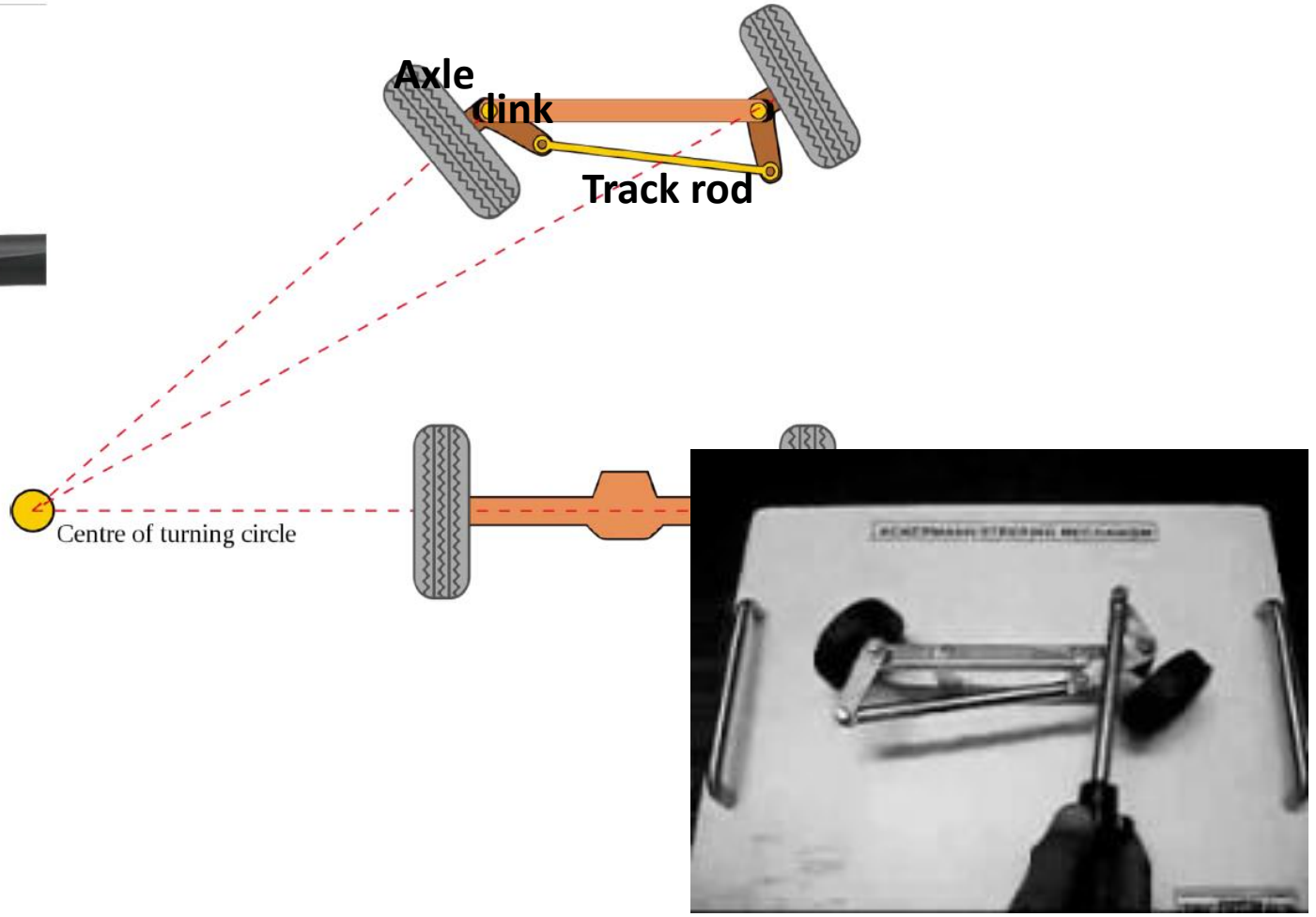
F1/10 scale RC race car chassis

- Oil filled shocks
  - Fully tunable with a wide range of oils, springs, and pistons
- Long travel suspension



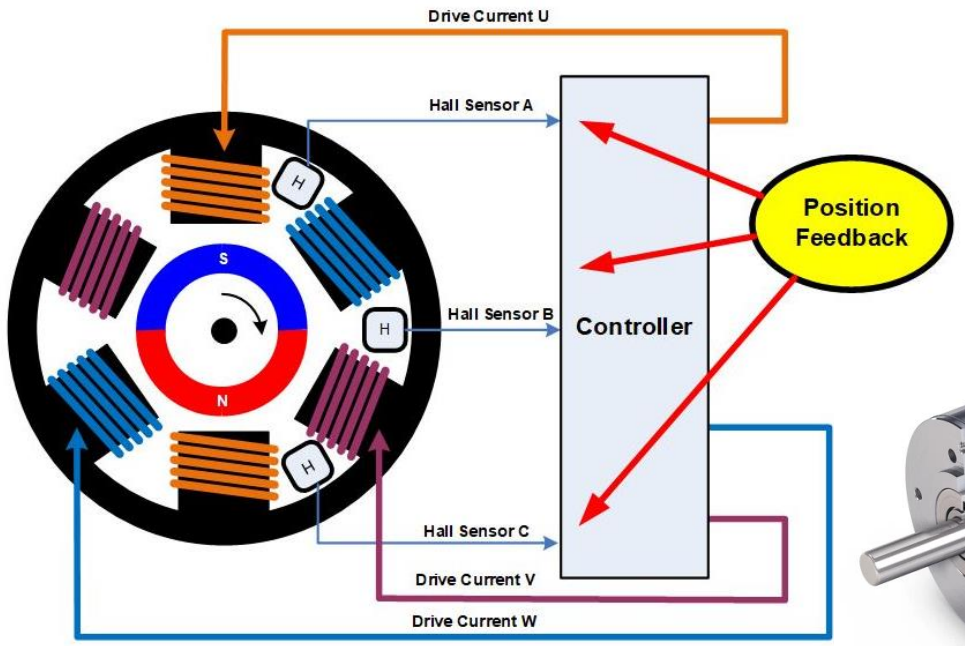
F1/10 scale RC race car chassis

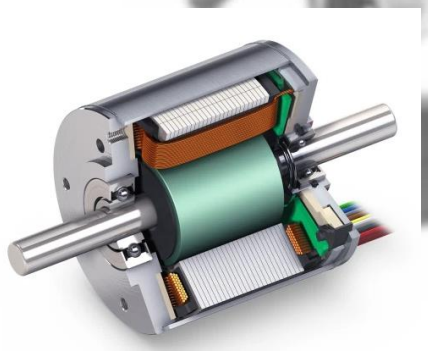
- Ackerman steering



Sensorless Brushless DC motor (BLDC motor)

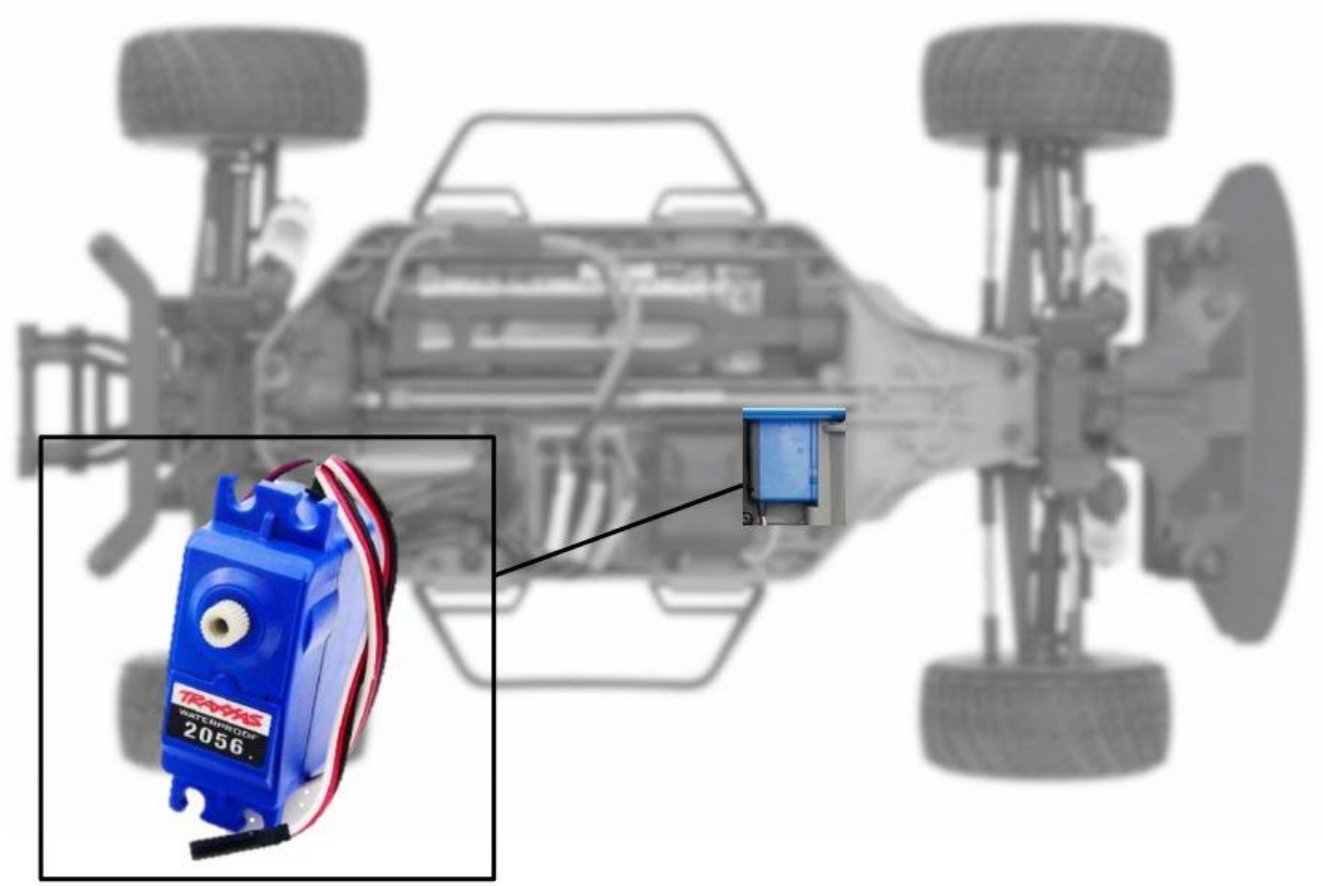
- RPM/volt: 3500
- Max RPM: 50,000.
- Wire Size: 12-gauge



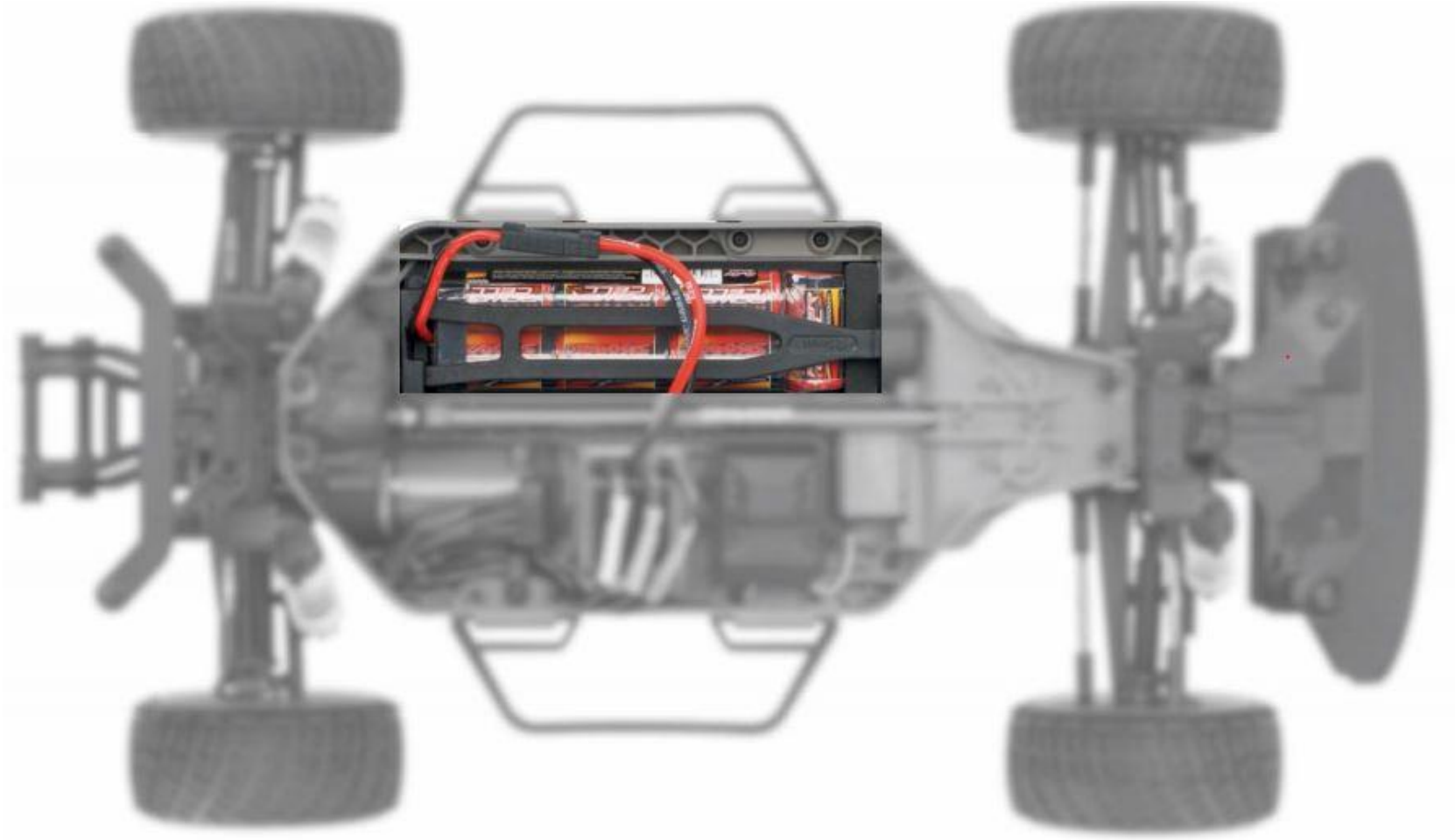


Servo motor for steering

- High torque waterproof servo
- Torque : 6.0V - 5.76 kg-cm
- Speed : 6.0V - 0.23sec/60deg
- Dimensions: 55.1 x 20.1 x 42.9mm, 45g
- Motor Type : Brushed
- Rotation Range : 60 deg
- Pulse Cycle : 2ms
- Pulse Width : 858-1670 μs



4cells LiPo Battery pack



Complete battery pack

4cell 14.8V 4200mAh LiPo Battery pack & Charger

Charging battery pack

1. Connect 4 cells ports & power port

2. Push start button for 2 sec and one more push start button (Confirm)

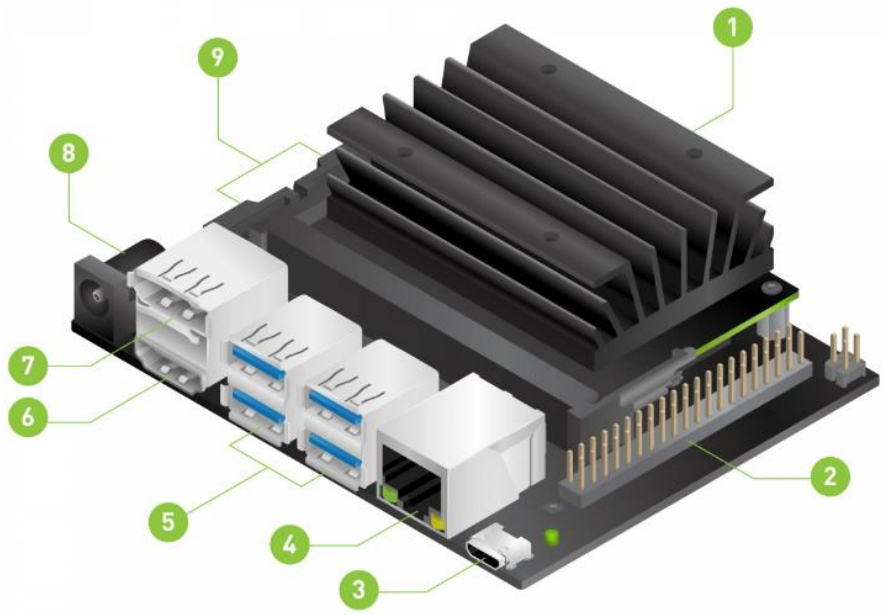
3. The alarm goes off when the charging is complete

Keep your seat when you charge the battery, Failure to disconnect will result in a fire

Charge only using the provided charger

Nvidia Jetson Nano Developer Kit

- GPU : 128-core Maxwell
- CPU : Quad-core ARM A57 @ 1.43 GHz
- Connectivity :
  - Gigabit Ethernet, M.2 Key E
  - HDMI and display port
  - 4x USB 3.0, USB 2.0 Micro-B
  - GPIO, I2C, I2S, SPI, UART



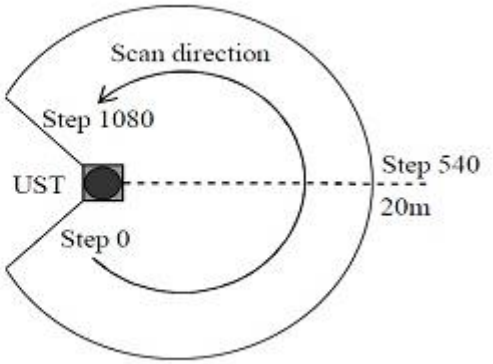
1. microSD card slot for main storage
2. 40-pin expansion header
3. Micro-USB port for 5V power input, or for Device Mode
4. Gigabit Ethernet port
5. USB 3.0 ports (x4)
6. HDMI output port
7. DisplayPort connector
8. DC Barrel jack for 5V power input
9. MIPI CSI-2 camera connectors

[Getting Started With Jetson Nano Developer Kit | NVIDIA Developer]

Hokuyo UST-10LX : Scanning Laser Range Finder, 2D LiDAR

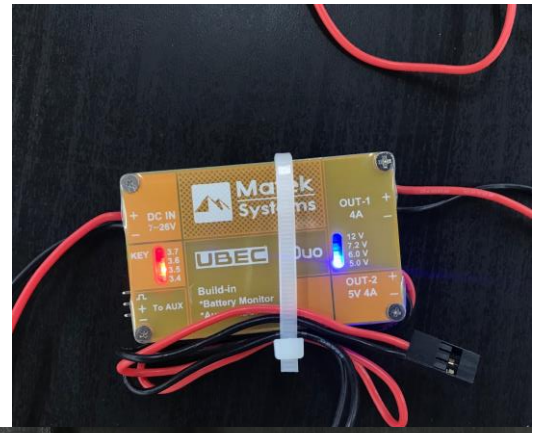
- Supply voltage : **DC 12V**/DC 24V
- Supply current : 150mA or less
- Detection range : 0.06m to 10m (white Kent sheet)
  0.06m to 4m (diffuse reflectance 10%)
  Max. detection distance : 30m
- Scan angle : 270 deg
- Scan speed : 25ms
- Angle resolution : 0.25 deg
- Start up time : Within 10 sec
- Possible detect size, position and the moving direction objects

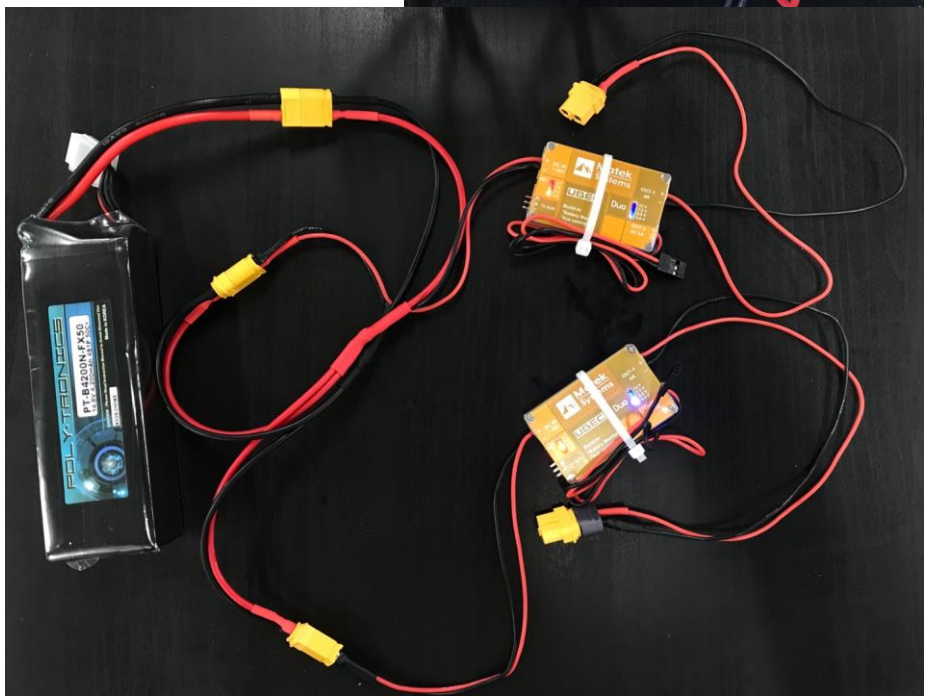
Measurement steps 1081
Detection angle 270°
Angular resolution 0.25°



UBEC DUO DC-DC converter with USB interface

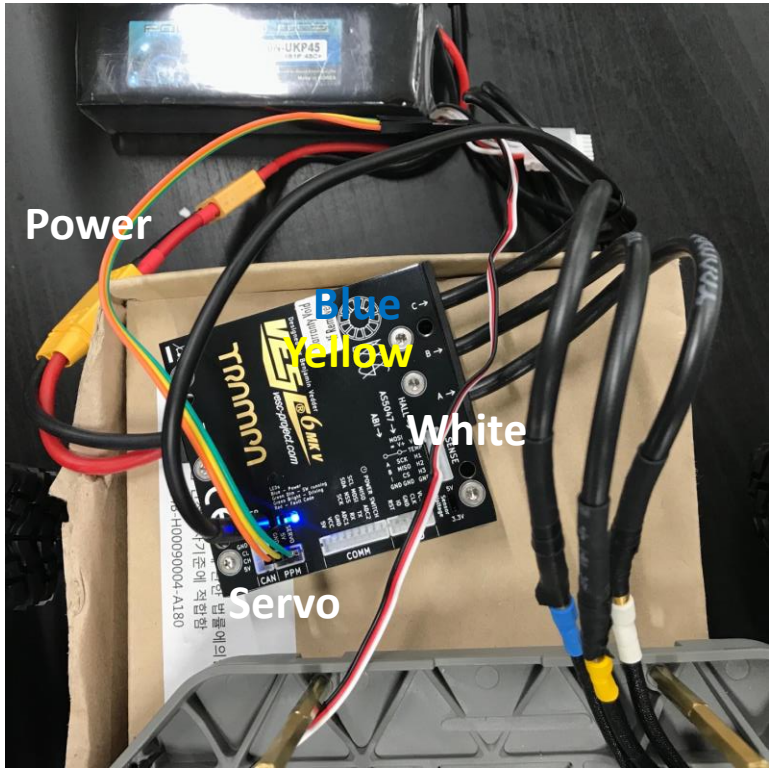
- UBEC DUO, 4A/5~12V & 4A/5V
- Input voltage range : 7~26V (2~6S Lipo battery)
- Out-1 : 5V, 6V, 7.2V, 12V (Adjustable), 0~4A, MAX. 6A
- Out-2 : 5V, 0~4A, Max.6A
- LiPo voltage alarm setting : 3.4V, 3.5V, 3.6V,3.7V/Cell & OFF





VESC : Vedder – Electronic Speed Controller

- Current and voltage measurement on all phases (three phase shunts)
  - Adjustable current and voltage filters
  - Built in IMU chip (accelerometer, gyro, compass)
  - Regenerative braking
  - Traction control (single and twin setup)
  - Configurable RPM-, power limits
  - Communication ports : USB, CAN, UAVCAN, UART
  - Separate throttle curves for acceleration and brakes
  - Real time data analysis
  - Adjustable protection against :
    - Low input voltage
    - High input voltage
    - High motor current
    - High input current
    - High regenerative braking current (separate limits for the motor and the input)
    - High RPM (separate limits for each direction).
    - Over temperature (MOSFET and motor)

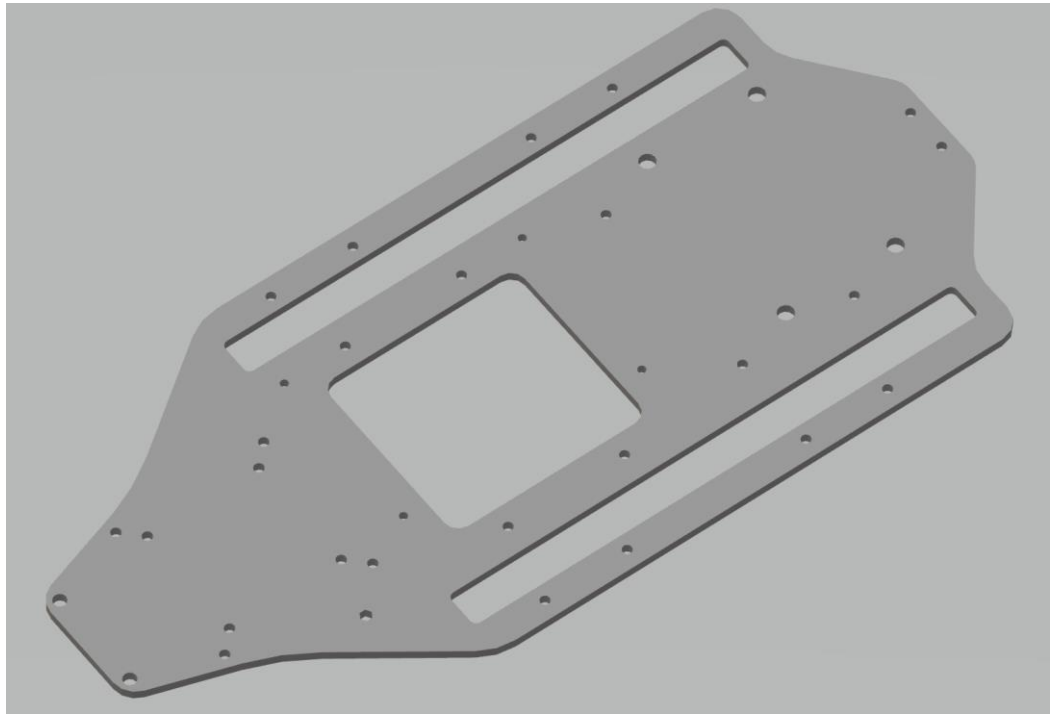


Making Autonomous driving car

- Key components
  - **NVIDIA Jetson Nano**: It is the world's smallest Embedded AI supercomputer for autonomous robots and edge computing devices, which can run code after installing ROS. Wi-Fi is available for wireless communication.
  - **Lidar (Hokuyo UST-10LX):** It is a compact, lightweight 2D LiDAR sensor used for **obstacle detection and localization on autonomous mobile robots and automated guided vehicles and carts**. Equipped with an Ethernet interface, it can obtain high-speed, accurate measurement data in a **270° field-of-view up to 10 meters**. The received data matches the message type and topic, enters NVIDIA Jetson Nano, and is used in the executed code
  - **VESC**: The electric circuit that controls and regulates the speed of the electric motor can limit the maximum speed and the minimum speed. **It receives signals from the NVIDIA Jetson Nano to control the car**
  - **UBEC DUO:** The electrical circuit that converts battery voltage into a suitable voltage for Lidar and Jetson Nano. (12V and 5V, respectively)

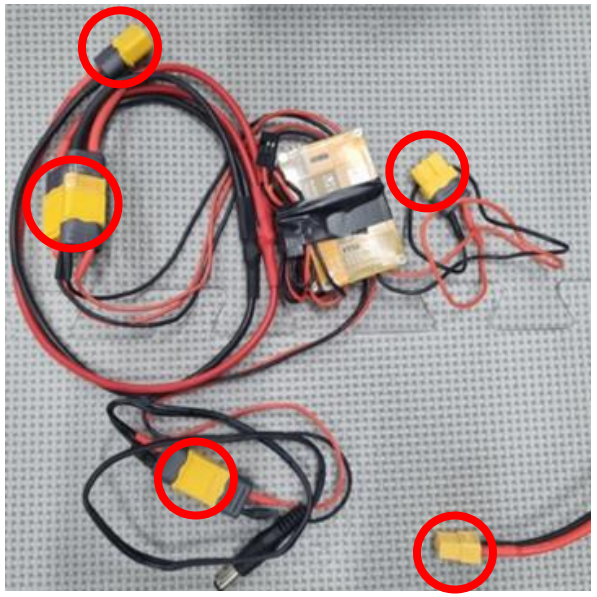
Making Autonomous driving car

- Procedure
  1. Measure the size of the car body and make the plate using a 3D tool (solid works). After printing it out using a 3D printer, check that there is no problem with the plate



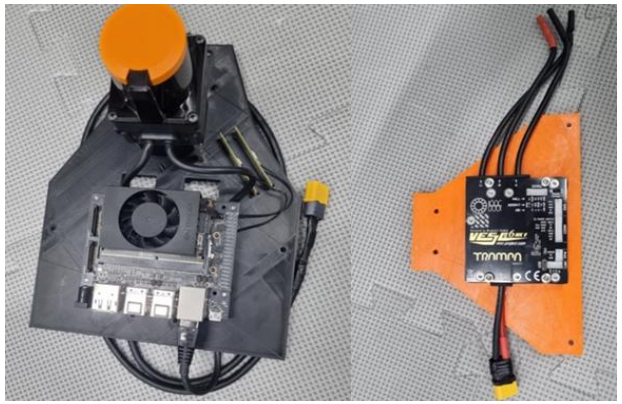
Making Autonomous driving car

- Procedure
  2. Solder the connecting parts of each part and attach the gold connector so that they can be connected

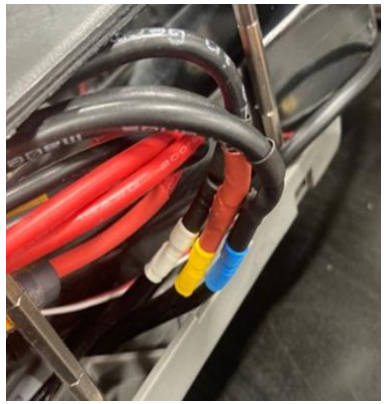


Making Autonomous driving car

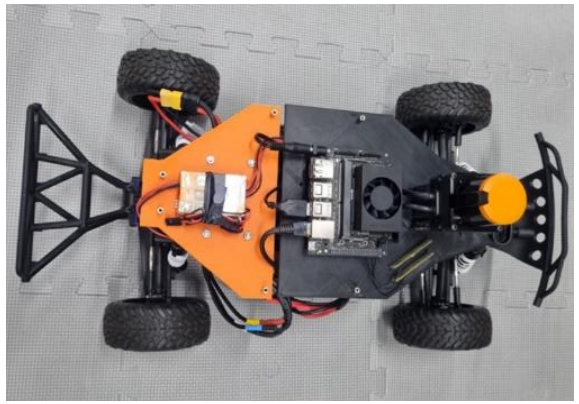
- Procedure
  3. Connect each parts and arrange lines to complete the autonomous vehicle's hardware



Each device is connected to the upper and lower plates using screws



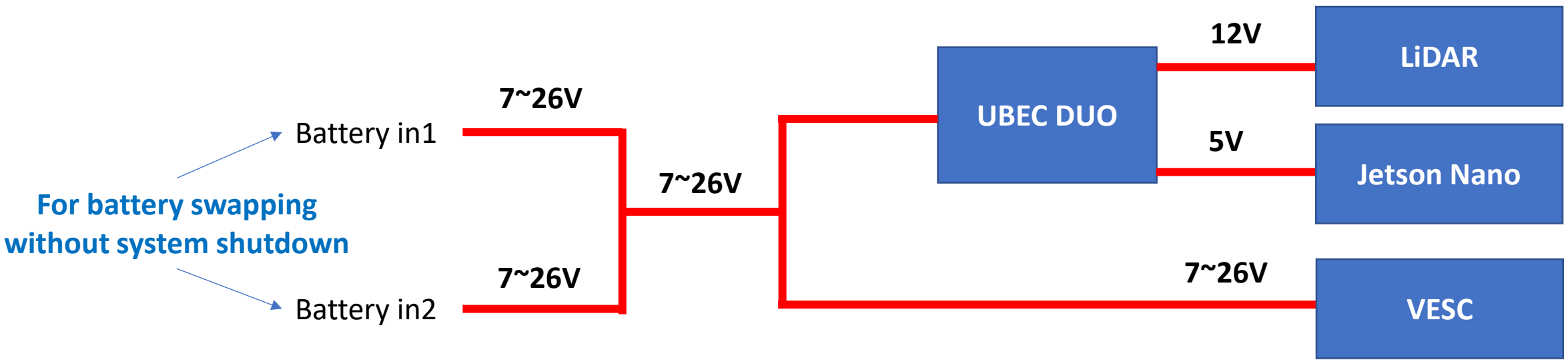
Connect VESC and servomotor using header pins and connect the VESC and motor



Complete!

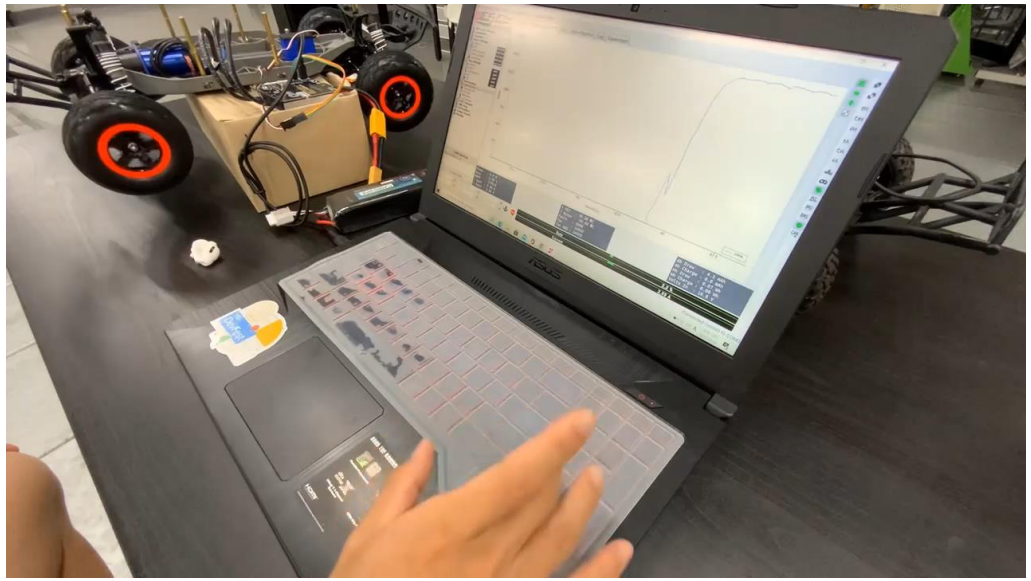
Making Autonomous driving car

- Procedure
  4. Pay attention to the power line connections as shown below.



Part2 : Install RC Car Software – Configuring the VESC

- Required Equipment :
  - Fully built RC car vehicle
  - Box or Car stand to put vehicle on
  - Laptop/computer (both window and Linux are possible)



1. Installing the VESC Tool

We need to configure the VESC so that it works with the ROS driver package. Before you start, you'll need to install the VESC Tool v6.0.0

https://github.com/HMCL-UNIST/Bootcamp

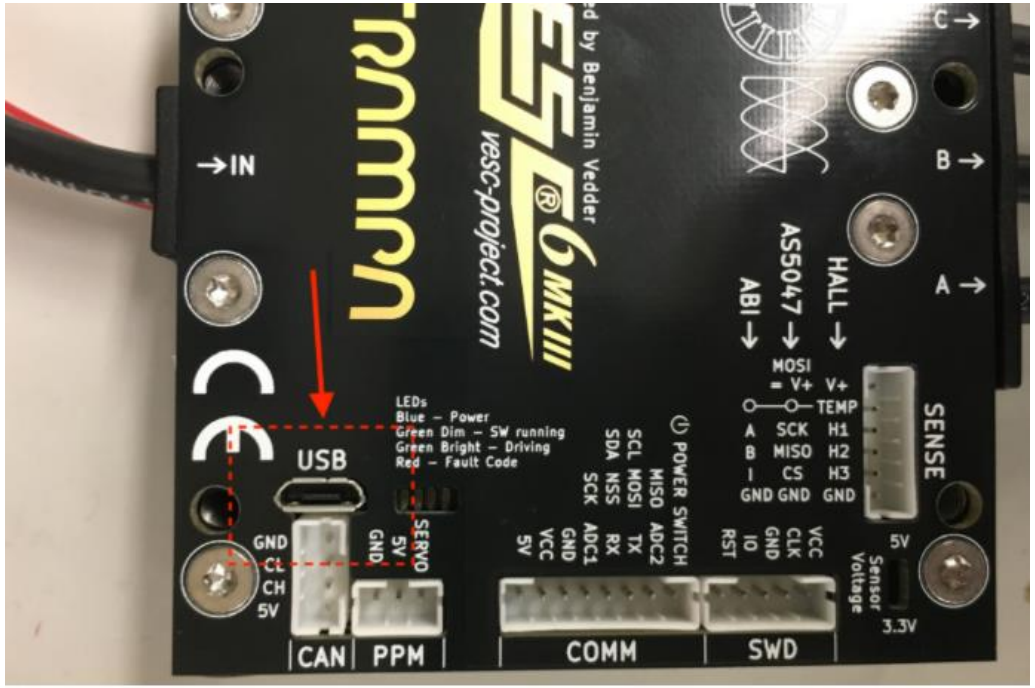
Install RC Car Software – Configuring the VESC

2. Powering the VESC

First, we need to power the VESC. Plug the battery at battery in port.

Note that you don’t need to turn on the Jetson Nano for configuring the VESC.

Next, unplug the USB cable of the VESC from the Jetson Nano and plug the USB into your laptop.

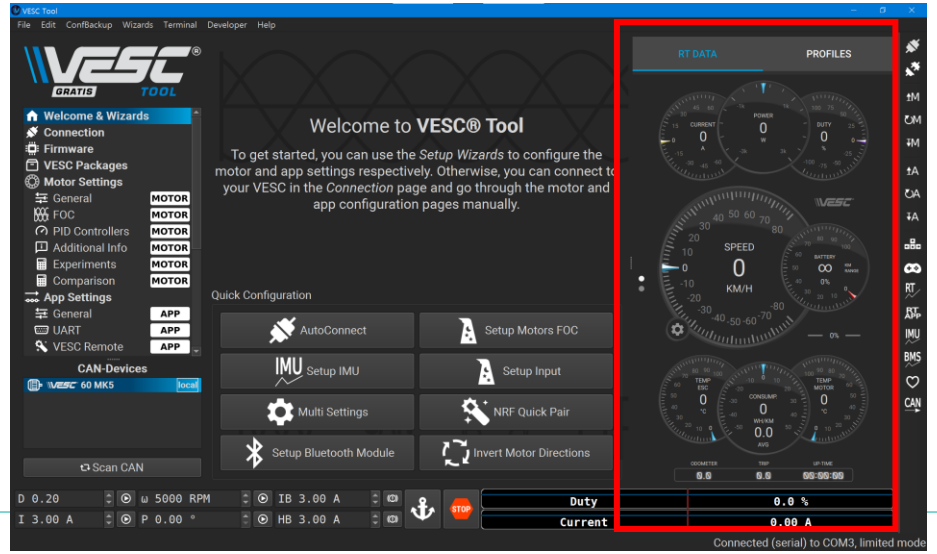


Plug a longer micro USB cable from the VESC to your computer.

Install RC Car Software – Configuring the VESC

3. Connecting the VESC to Your Laptop

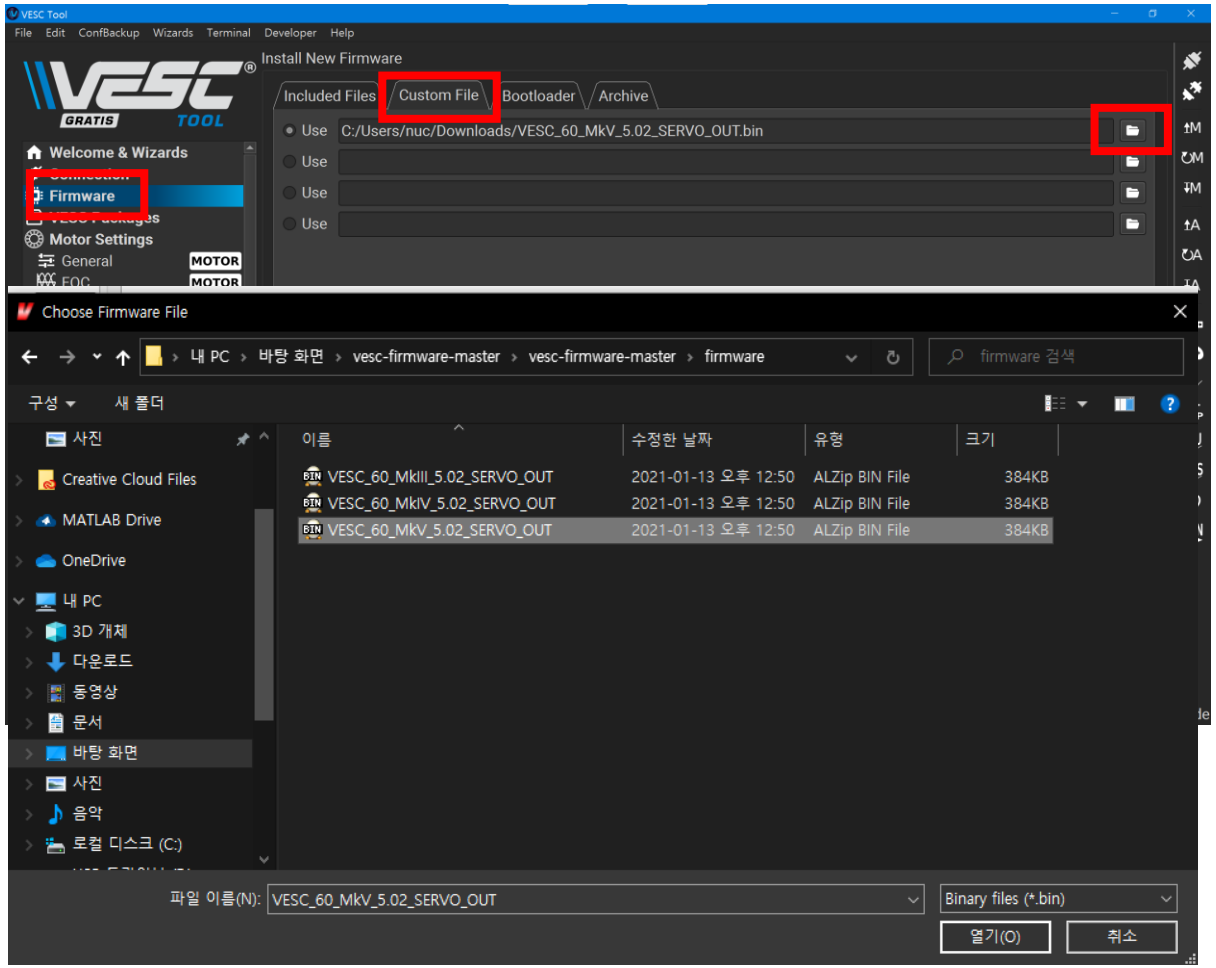
Launch the VESC Tool. On the Welcome page, press the **Connect** button on right. After the VESC is connected, you should see a dashboard on right.



Install RC Car Software – Configuring the VESC

4. Updating the Firmware on the VESC

The first thing you'll need to do is to update the firmware onboard the VESC. On the left side of the screen, click on the **Firmware** tab and click **Custom File**. And then update custom file. Afterwards, on the bottom right of the page, press the button with the down arrow to update the firmware on the connected VESC. A status bar at the bottom of the page will show firmware update status. After it's finished, follow the prompt on screen.

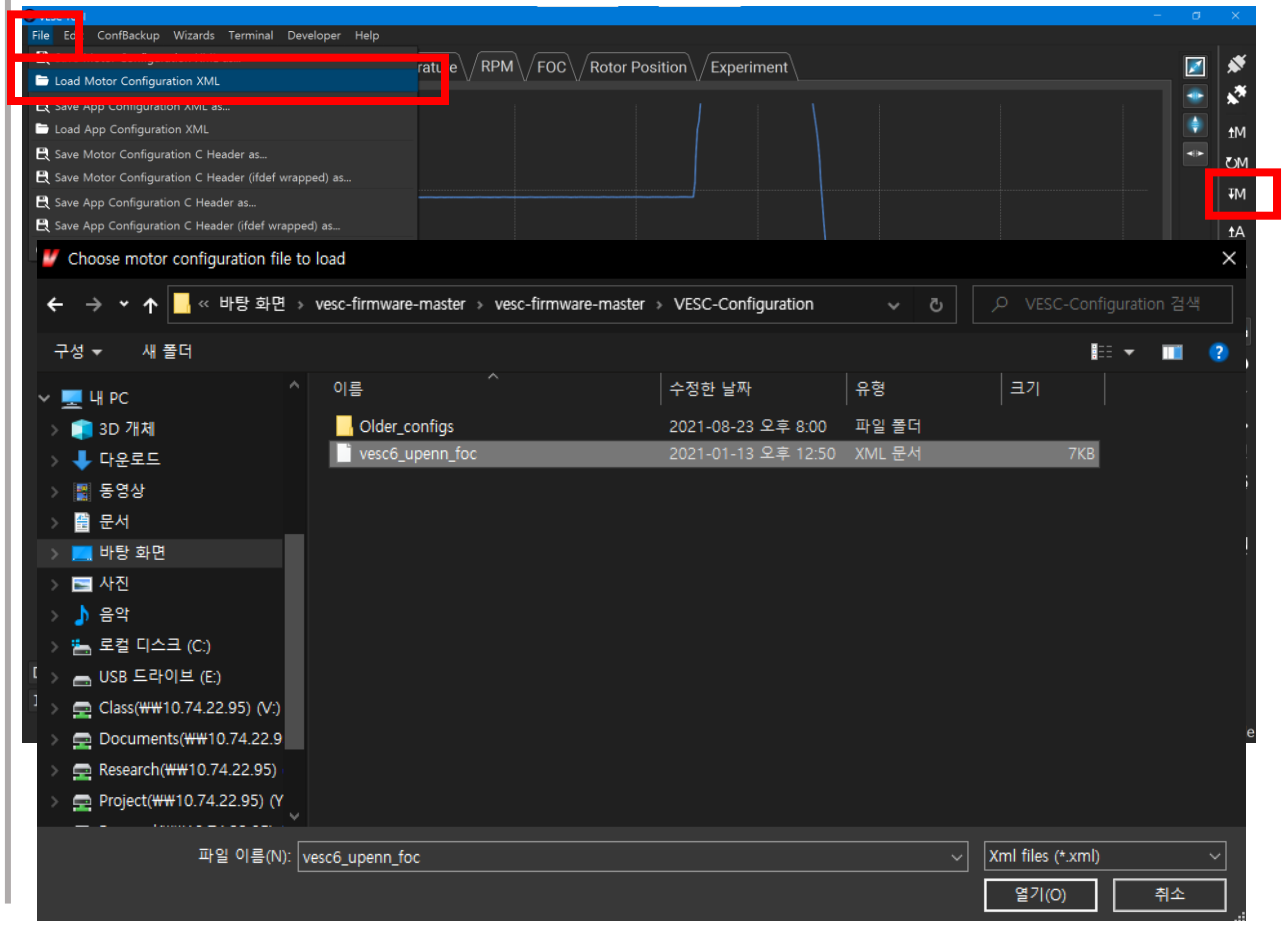


Install RC Car Software – Configuring the VESC

5. Uploading the Motor Configuration XML

After firmware update, Select **Load Motor Configuration XML** from the drop down menu and select the provided XML file. After the XML is uploaded, click on the **Write Motor Configuration** button (the button with a down arrow and the letter M) on the right side of the screen to apply the motor configuration. Note that in the future, you'll have to press this button whenever you make a change in motor configuration.

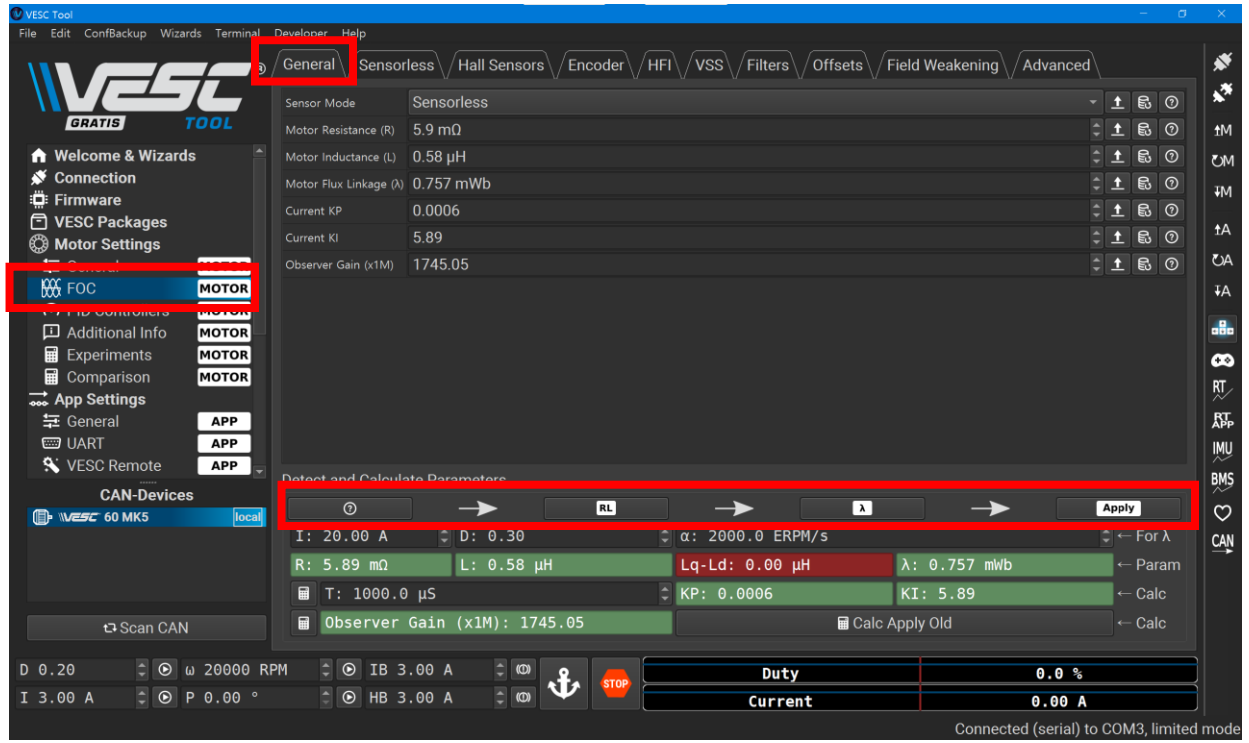
XML file: General > Class Materials > Supplements > vesc6_upenn_foc.xml



Install RC Car Software – Configuring the VESC

6. Detecting and Calculating Motor Parameters

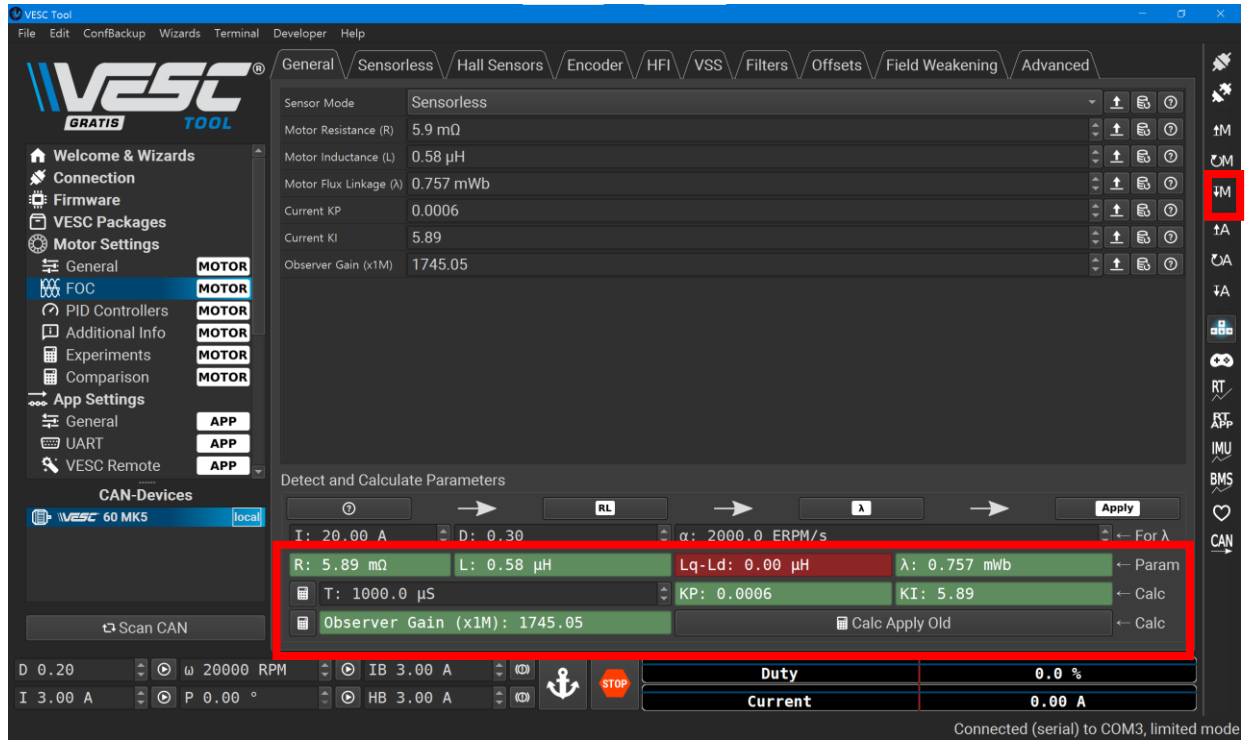
To detect and calculate the FOC motor parameters, navigate to the **FOC** tab under **Motor Settings** on the left. At the bottom of the screen, follow the direction of the arrows and click on the four buttons one by one, and follow the on screen prompt. Note that during the measuring process, the motor will make noise and spin, make sure the wheels of your vehicle are clear.



Install RC Car Software – Configuring the VESC

After the motor parameters are measured, the fields at the bottom of the screen turn green. (It's okay if "Lq-Ld" is red.)

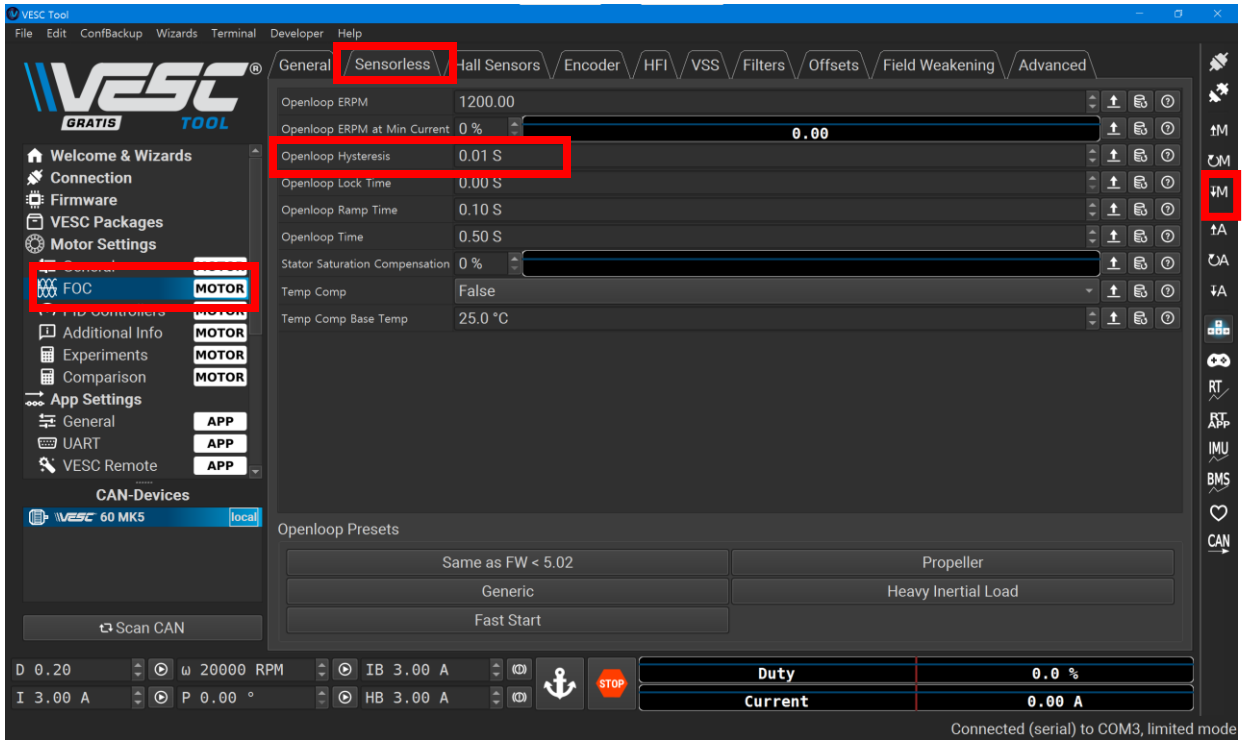
Click on the **Apply** button, and click the **Write Motor Configuration** button.



Install RC Car Software – Configuring the VESC

7. Changing the Openloop Hysteresis and Openloop Time

Navigate to the **Sensorless** tab on top of the screen. Change the **Openloop Hysteresis** and **Openloop Time** to 0.01, and click the **Write Motor Configuration** button.



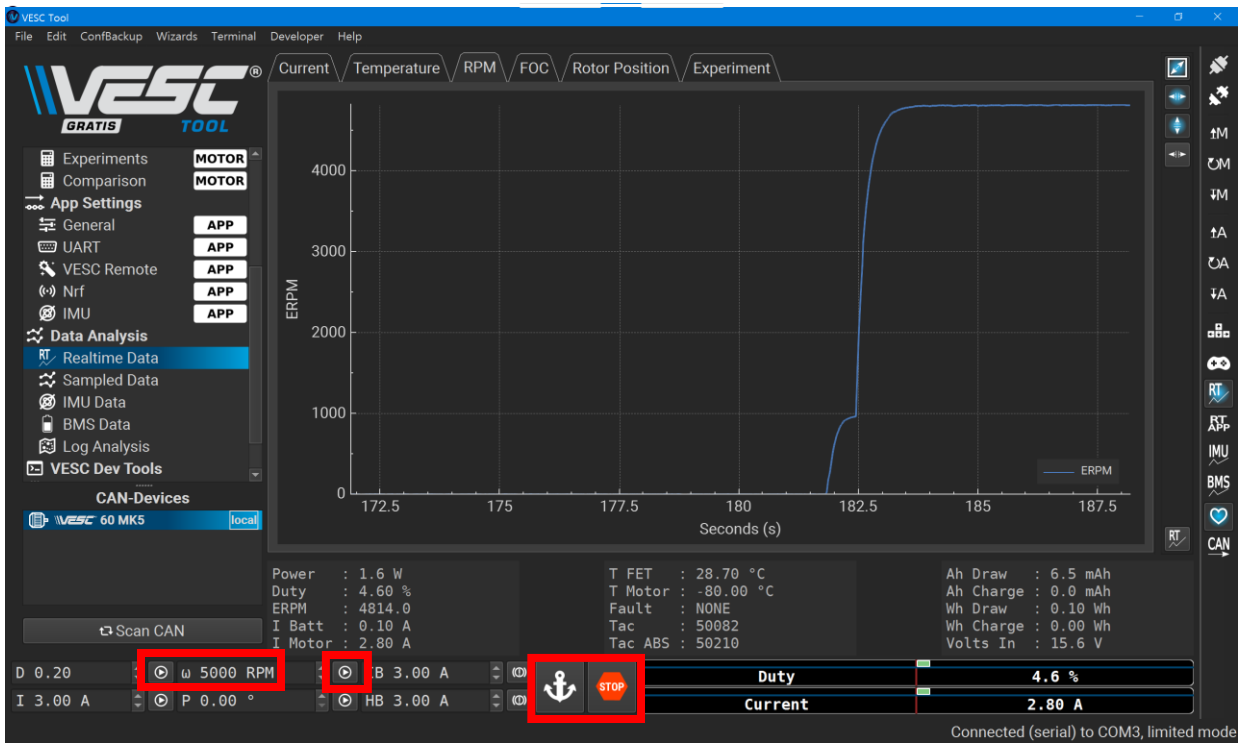
Install RC Car Software – Configuring the VESC

8. Tuning the PID controller

Now you can start tuning the speed PID controller. To see the RPM response from the motor, navigate to the **Realtime Data** tab under **Data Analysis** on the left. Click **Stream Realtime Data** button on the right (the button with letters RT), and navigate to the **RPM** tab on the top of the screen. You should see RPM data streaming now.

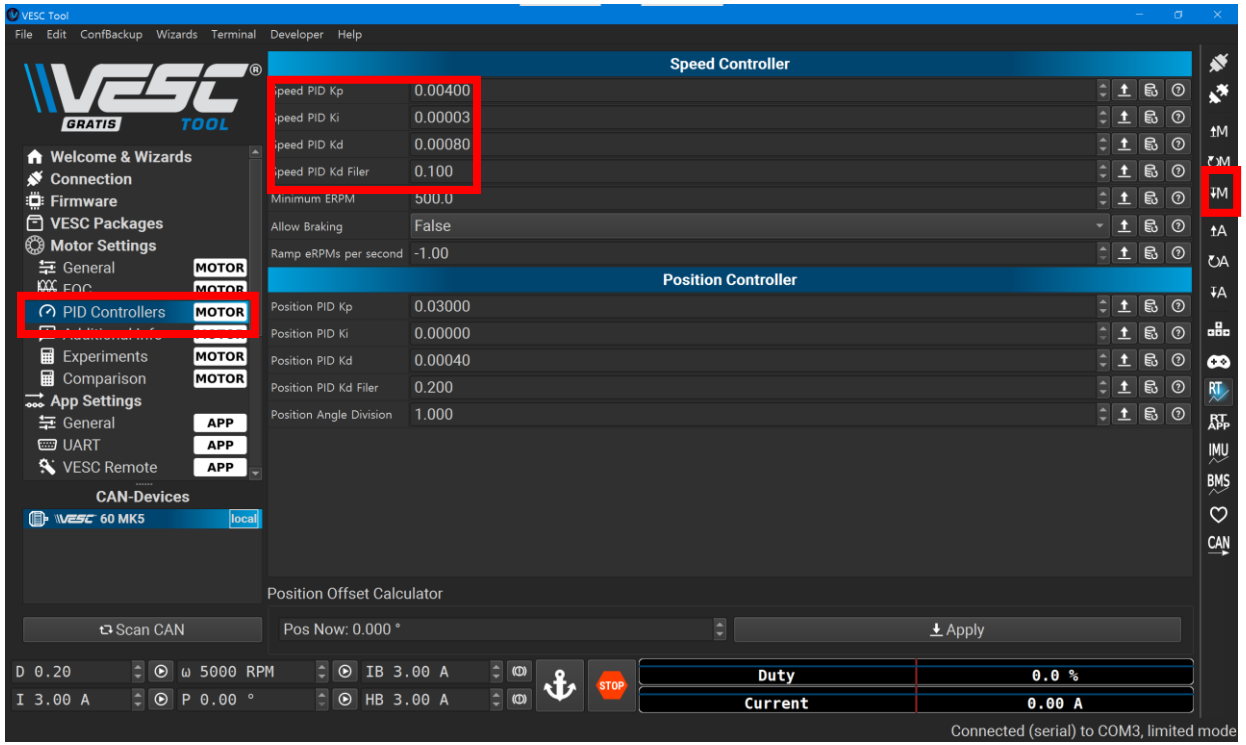
Install RC Car Software – Configuring the VESC

To create a step response for the motor, you can set a target RPM at the bottom of the screen (values between 2000 - 10000 RPM). Click the play button next to the text box to start the motor. Note that the motor will spin, so make sure the wheels of your vehicle are clear from objects. Click the Anchor or STOP button to stop the motor.



Install RC Car Software – Configuring the VESC

You want to look for a clean step response that has a quick rise time and zero to very little steady state error. Adjust the gains accordingly by navigating to the **PID Controllers** tab under **Motor Settings** on the left, and change the Speed Controller gains. General rules of tuning PID gains apply. If you're seeing a lot of oscillations, try changing the Speed PID Kd Filter.



https://vesc-project.com/

https://www.youtube.com/channel/UCcslH9JifCwInHV6mP5KGFQ

Calibrating the Odometry

- The VESC receives input velocities in m/s and steering angles in radians. However, the motor and servo require commands in revolution per minute (RPM) and servo positions.
- The conversion parameters will need to be tuned to your specific car.

1. The parameters in vesc.yaml need to be calibrated. This yaml file is located at

$YOURWORKSPACE/f1tenth_system/f1tenth_stack/config/vesc.yaml

Calibrating the Odometry

- The VESC receives input velocities in m/s and steering angles in radians. However, the motor and servo require commands in revolution per minute (RPM) and servo positions.
- The conversion parameters will need to be tuned to your specific car.

1. The parameters in vesc.yaml need to be calibrated. This yaml file is located at

$YOURWORKSPACE/config/vesc.yaml

Calibrating the Odometry

2. We need to check if our motor is rotating in the right direction.

   → If when given a positive velocity, or commanded moving forward with the joystick, the motor is spinning in the reverse direction, swap 2 of the 3 connections from the vesc to the BLDC motor.

Calibrating the Odometry

3. We'll also need to check if the vesc driver is interpreting the motor rotation direction correctly.

→ Echo the "/odom" topic

→ Give a positive velocity command (forward throttle with the joystick), and pay close attention to the pose/pose/position/x field of the echoed messages.

→ If it is increasing in the reverse direction, modified the source code located at:

$YOURWORKSPACE/vesc/vesc_ackermann/src/vesc_to_odom.cpp

→ Modify line 100

```
double current_speed = -(-state->state.speed - speed_to_erpm_offset_) / speed_to_erpm_gain_;
```

→ If you change the code, build again.

Calibrating the Odometry

4. Tune the steering offset

→This parameter will determine the offset we put on the servo, and whether the car can drive straight when given no steering input.

→ Drive the car in a straight line a couple times with no steering input for a couple meters, and see if it's driving straight.

→ Adjust the **steering_angle_to_servo_offset** parameter in **vesc.yaml**

→ Make small adjustment everytime (in the magnitude of 0.1). Repeat until you find the correct offset so the car drives straight.

Calibrating the Odometry

5. Tune the steering

→ This parameter determines the smallest turning radius of the car.

→ We'll determine the desired turning radius given the maximum steering angle of the car we're setting, which is 0.36 radians in both directions.

→ The corresponding turning radius could be then calculated with $R = L(2\sin(\beta))$, where L is the wheelbase of the car, which is around 0.33 meters; β is calculated as $\arctan(0.5\tan(\delta))$, with δ being the maximum steering angle of car.

→ After calculations, when turning a half circle, the desired diameter of the half circle should be 1.784 meters

Calibrating the Odometry

5. Tune the steering

→ Place the car at the 0 of the tape measure such that the rear axle of the car is parallel to the tape measure, and the x-axis (roll axis) of the car coincides with the tape measure at 0.

→ Start teleop. Set the steering angle to the maximum to one side depending on your setup (e.g. if the rest of the tape measure is on the left side of the car, turn left).

→ Hold the steering, and drive forward slowly and steadily until the car runs over the tape measure again and the rear axle realigns with the tape measure.

→ Take a measurement on the tape measure. The goal is 1.784 meters.

→ If the measurement overshoots, increase the gain slightly (0.1 at a time). If it undershoots, decrease the gain. Repeat the process until you've hit 1.784 meters.

→ If you notice that the wheels turn to different angles on the two directions when give maximum steering angles, adjust until they're symmetric.

Calibrating the Odometry

6. Change the software speed limit

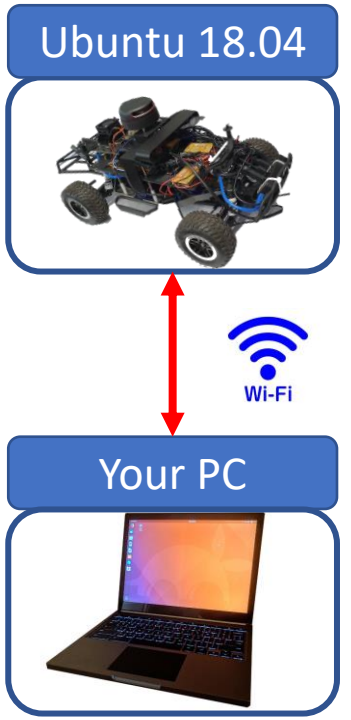
→ All you'll need to do is also change the speed_min and speed_max values in vesc.yaml

Part 3: Emergency Braking

- The goal of this lab is to develop a safety node for the race cars **that will stop the car from collision** when travelling at higher velocities.
- We will implement **Time to Collision (TTC)** using the **LaserScan** message

Software Environment

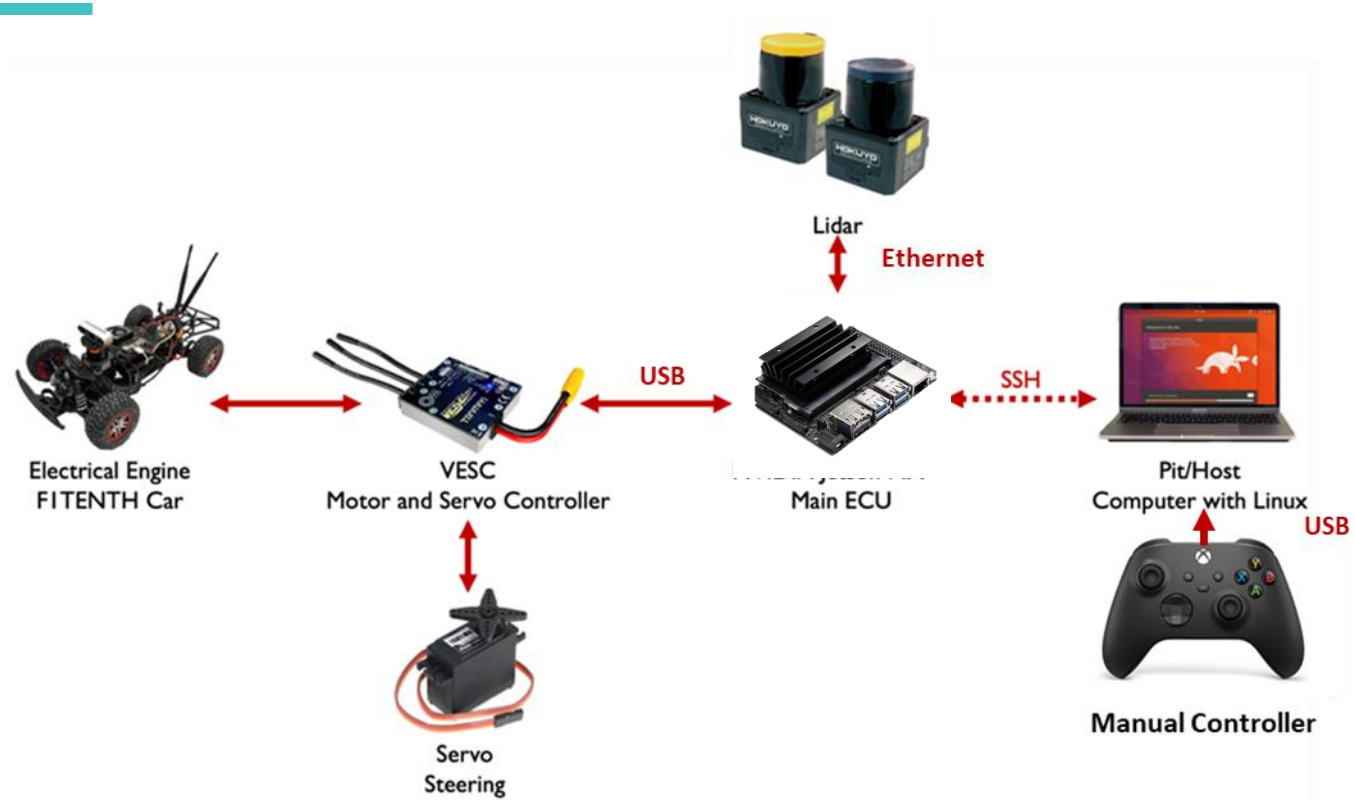
- Linux OS – Ubuntu 18.04 (Bionic)
  - Recommend to have access to Ubuntu 18.04(Bionic)
  - Options:
    1. Install an ubuntu virtual machine on any OS. (e.g. VirtualBox)
    2. Dual boot your laptop



Ubuntu 18.04
Window
MAC etc..

- Robot Operating System – Melodic
  - Programming Language → Python (mainly covered) / C++ or MATLAB / etc..

System Interface Configuration



- The **NVIDIA Jetson Nano** is the main brain of the entire system. It gives commands to the **VESC** which controls the **Servo** and the **Brushless Motor** on the Vehicle. The **NVIDIA Jetson Nano** also receives information from the **Lidar** either via USB or Ethernet. The **Pit/Host** laptop is where we can connect remotely via **SSH** to the **NVIDIA Jetson Nano.**

TTC Calculation

- **Time to Collision (TTC)** is the time it would take for the car to collide with an obstacle of it maintained its current heading and velocity. Between the car and its obstacle, we can calculate it as :

$$\text{TTC} = \frac{r}{\dot{r}}$$

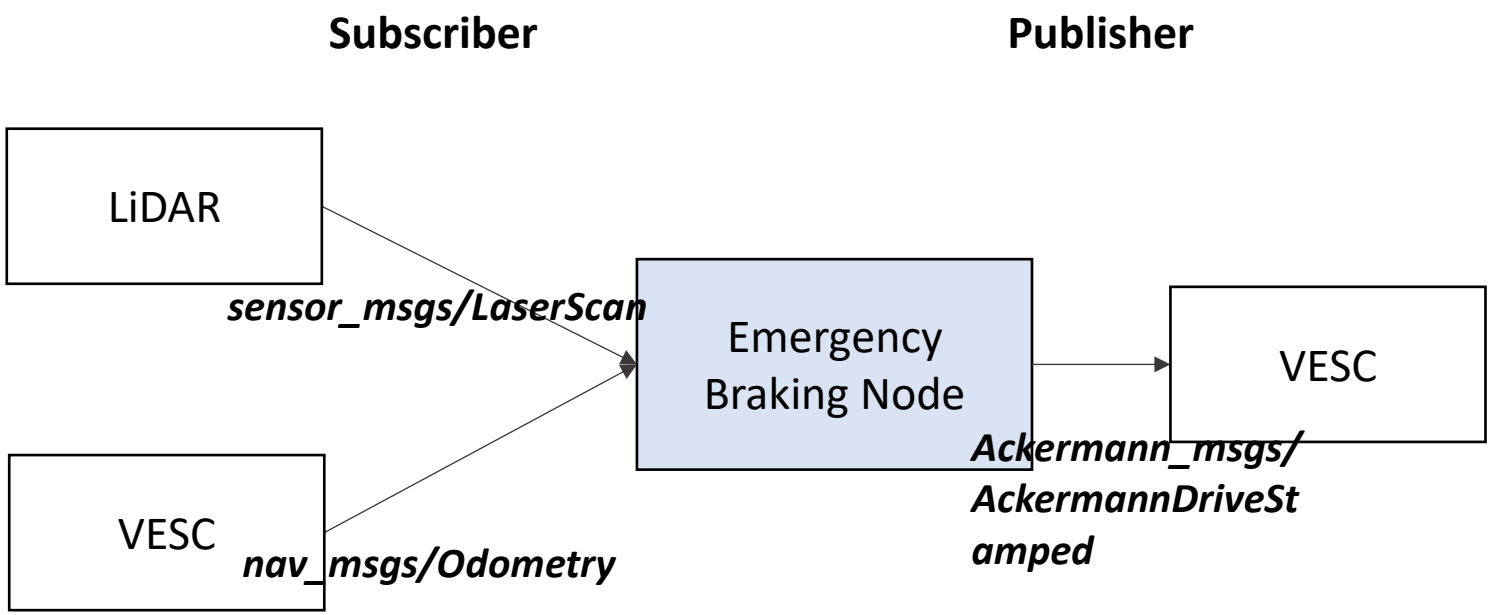
Where r is the distance between the two objects and $\dot{r}$ is the time derivative of that distance. $\dot{r}$ is computed by projecting the relative velocity of the car onto the distance vector between the two objects.

Overview

- For this lab, you will make a '**Emergency Braking Node**' that should halt the car before it collides with obstacles.

```python
#!/usr/bin/env python
import sys
import math
import numpy as np

#ROS Imports
import rospy
from sensor_msgs.msg import LaserScan
from ackermann_msgs.msg import AckermannDriveStamped
from nav_msgs.msg import Odometry
```

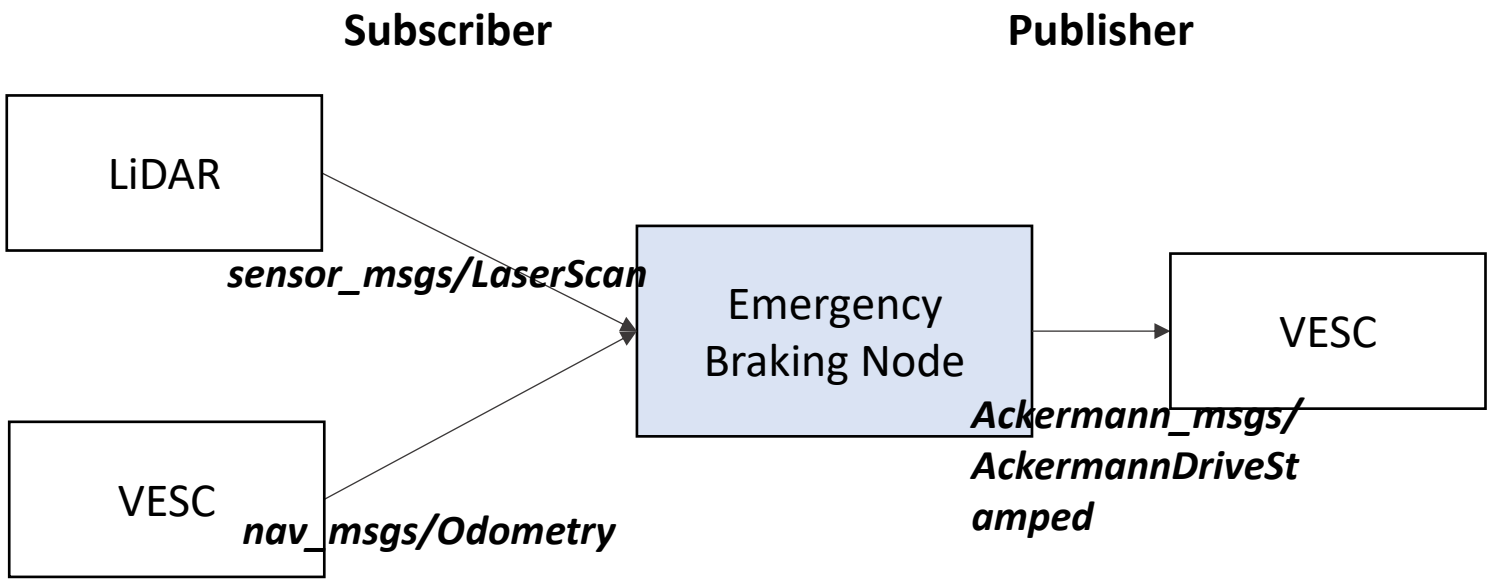


Overview

- You will need to first subscribe to the scan topic and calculate TTC with the ***LaserScan*** messages and ***Odometry*** messages
- We are sending ***AckermannDriveStamped*** messages with the velocity set to 0.0.

```python
#!/usr/bin/env python
import sys
import math
import numpy as np

#ROS Imports
import rospy
from sensor_msgs.msg import LaserScan
from ackermann_msgs.msg import AckermannDriveStamped
from nav_msgs.msg import Odometry
```



How to Start

- Step 1. Check the ***ip address*** of your car

$ ifconfig

```
hmcl@hmcl-desktop:~$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:b5:8f:66:39  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.16  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::c0d7:2545:55f4:8feb  prefixlen 64  scopeid 0x20<link>
        ether 48:b0:2d:5c:24:36  txqueuelen 1000  (Ethernet)
        RX packets 1  bytes 60 (60.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 76  bytes 7248 (7.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 150  base 0xe000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 232  bytes 18323 (18.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 232  bytes 18323 (18.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

rndis0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether e2:f1:56:1b:95:45  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usb0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether e2:f1:56:1b:95:47  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.50.196  netmask 255.255.255.0  broadcast 192.168.50.255
        inet6 fe80::4bac:6201:f4b9:f155  prefixlen 64  scopeid 0x20<link>
        ether 14:f6:d8:75:85:a5  txqueuelen 1000  (Ethernet)
        RX packets 7502  bytes 11196210 (11.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2873  bytes 285707 (285.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

$ gedit ~/.bashrc

```
export ROS_MASTER_URI=http://192.168.50.196:11311
export ROS_IP=192.168.50.196
```

How to Start

- Step 2. Connect to VESC

  $ vesc

```
level/ackermann_cmd_mux-11]: started with pid [9720]
process[vesc/ackermann_to_vesc-12]: started with pid [9721]
process[vesc/vesc_driver-13]: started with pid [9723]
process[vesc/vesc_to_odom-14]: started with pid [9748]
process[base_link_to_imu-15]: started with pid [9759]
[FATAL] [1679910038.504004114]: Failed to connect to the VESC, SerialException F
ailed to open the serial port to the VESC. IO Exception (13): Permission denied,
 file /home/hmcl/f1tenth/src/serial/src/impl/unix.cc, line 151. failed..
[vesc/vesc_driver-13] process has finished cleanly
log file: /home/hmcl/.ros/log/6bcdcf30-cc83-11ed-bcce-48b02d5c2436/vesc-vesc_dri
ver-13*.log
```

**** If you got serial port error,**

1. $ cd ~/dev && ls
   -> go into the 'dev' folder
   -> check the 'ttyACM0' exists.
   (if 'ttyACM0' doesn't exist, please plug the cable into computer again.)

```
hmcl@hmcl-desktop:~$ cd /dev
hmcl@hmcl-desktop:/dev$ ls
autofs          max_gpu_power   ptyp9   tty18   ttyACM0
block           max_online_cpus ptypa   tty19   ttyGS0
btrfs-control   media0          ptypb   tty2    ttyp0
bus             mem             ptypc   tty20   ttyp1
```

2. $ sudo chmod 777 ttyACM0
   -> Set 777 permission
   that it will be readable, writable and executable by all users

```
hmcl@hmcl-desktop:/dev$ sudo chmod 777 ttyACM0
```

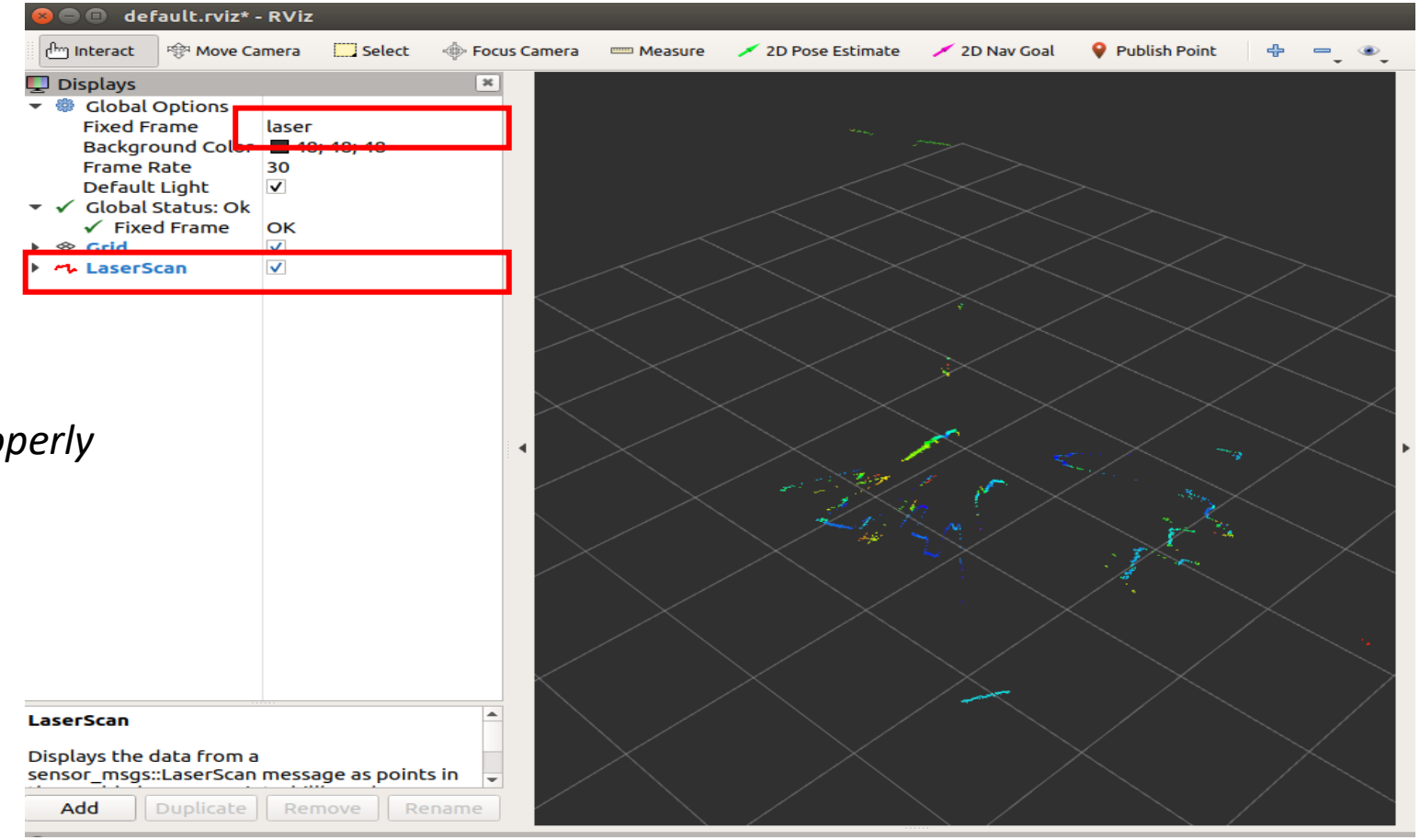
How to Start

- Step 3. Connect to Lidar

$ lidar

$ rviz

-> Using Rviz,
check the scan topic is published properly



Topics/Messages for Emergency Braking

$ rostopic list

```
hmcl@hmcl-desktop:~$ rostopic list
/clicked_point
/dev/null
/diagnostics
/initialpose
/joy
/laser_status
/move_base_simple/goal
/rosout
/rosout_agg
/scan
/tf
/tf_static
/urg_node/parameter_descriptions
/urg_node/parameter_updates
/vesc/ackermann_cmd
/vesc/ackermann_cmd_mux/input/navigation
/vesc/ackermann_cmd_mux/input/safety
/vesc/ackermann_cmd_mux/input/teleop
/vesc/commands/motor/brake
/vesc/commands/motor/speed
/vesc/commands/servo/position
/vesc/high_level/ackermann_cmd_mux/active
/vesc/high_level/ackermann_cmd_mux/input/default
/vesc/high_level/ackermann_cmd_mux/input/nav_0
/vesc/high_level/ackermann_cmd_mux/input/nav_1
/vesc/high_level/ackermann_cmd_mux/input/nav_2
/vesc/high_level/ackermann_cmd_mux/input/nav_3
/vesc/high_level/ackermann_cmd_mux/output
/vesc/high_level/ackermann_cmd_mux/parameter_descriptions
/vesc/high_level/ackermann_cmd_mux/parameter_updates
/vesc/high_level/ackermann_cmd_mux_nodelet_manager/bond
/vesc/low_level/ackermann_cmd_mux/active
/vesc/low_level/ackermann_cmd_mux/input/navigation
/vesc/low_level/ackermann_cmd_mux/input/safety
/vesc/low_level/ackermann_cmd_mux/input/teleop
/vesc/low_level/ackermann_cmd_mux/output
/vesc/low_level/ackermann_cmd_mux/parameter_descriptions
/vesc/low_level/ackermann_cmd_mux/parameter_updates
/vesc/low_level/ackermann_cmd_mux_nodelet_manager/bond
/vesc/odom
/vesc/sensors/core
/vesc/sensors/servo_position_command
```

$ rostopic info <Topic Name>

```
hmcl@hmcl-desktop:~$ rostopic info /vesc/odom
Type: nav_msgs/Odometry

Publishers:
 * /vesc/vesc_to_odom (http://192.168.50.196:40965/)

Subscribers: None
```

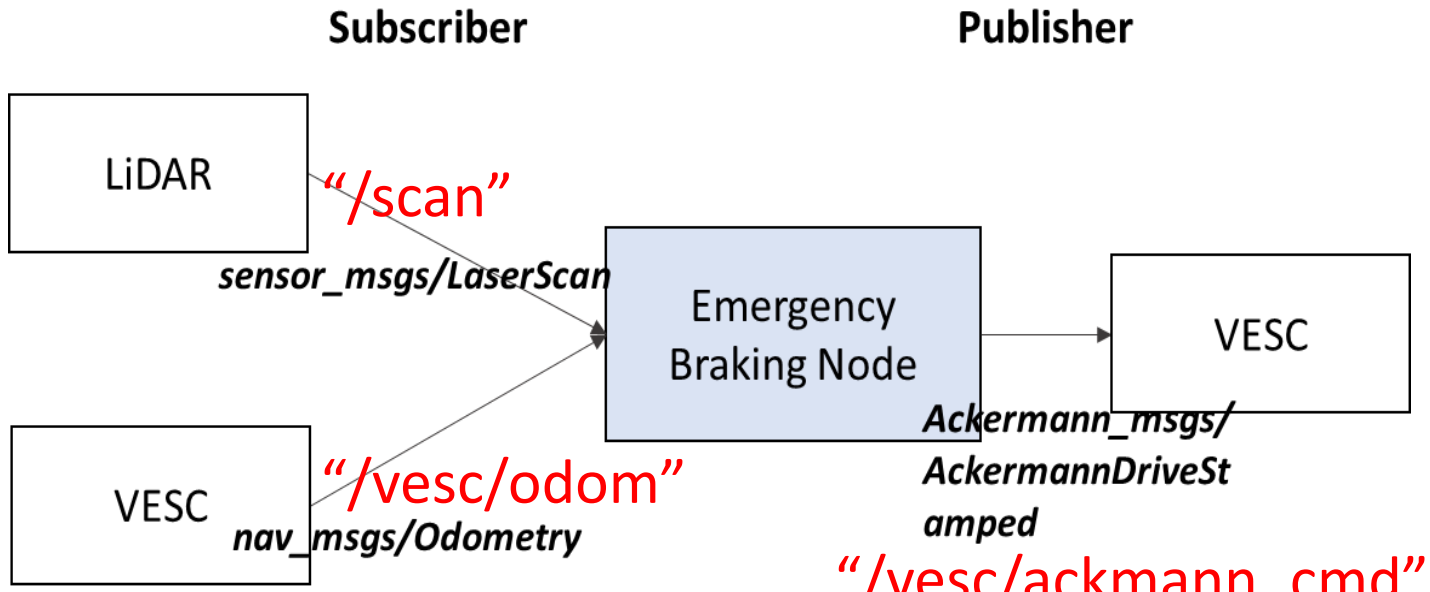
$ rosmsg info <Message Name>

```
hmcl@hmcl-desktop:~$ rosmsg info nav_msgs/Odometry
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

Topics/Messages for Emergency Braking

$ rostopic list

```
hmcl@hmcl-desktop:~$ rostopic list
/clicked_point
/dev/null
/diagnostics
/initialpose
/joy
/laser_status
/move_base_simple/goal
/rosout
/rosout_agg
/scan
/tf
/tf_static
/urg_node/parameter_descriptions
/urg_node/parameter_updates
/vesc/ackermann_cmd
/vesc/ackermann_cmd_mux/input/navigation
/vesc/ackermann_cmd_mux/input/safety
/vesc/ackermann_cmd_mux/input/teleop
/vesc/commands/motor/brake
/vesc/commands/motor/speed
/vesc/commands/servo/position
/vesc/high_level/ackermann_cmd_mux/active
/vesc/high_level/ackermann_cmd_mux/input/default
/vesc/high_level/ackermann_cmd_mux/input/nav_0
/vesc/high_level/ackermann_cmd_mux/input/nav_1
/vesc/high_level/ackermann_cmd_mux/input/nav_2
/vesc/high_level/ackermann_cmd_mux/input/nav_3
/vesc/high_level/ackermann_cmd_mux/output
/vesc/high_level/ackermann_cmd_mux/parameter_descriptions
/vesc/high_level/ackermann_cmd_mux/parameter_updates
/vesc/high_level/ackermann_cmd_mux_nodelet_manager/bond
/vesc/low_level/ackermann_cmd_mux/active
/vesc/low_level/ackermann_cmd_mux/input/navigation
/vesc/low_level/ackermann_cmd_mux/input/safety
/vesc/low_level/ackermann_cmd_mux/input/teleop
/vesc/low_level/ackermann_cmd_mux/output
/vesc/low_level/ackermann_cmd_mux/parameter_descriptions
/vesc/low_level/ackermann_cmd_mux/parameter_updates
/vesc/low_level/ackermann_cmd_mux_nodelet_manager/bond
/vesc/odom
/vesc/sensors/core
/vesc/sensors/servo_position_command
```



Topics/Messages for Emergency Braking

$ rostopic echo <Topic name>

```
      y: 0.0
      z: 0.017649522253
      w: 0.999844235051
  covariance: [0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 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, 0.0, 0.4]
twist:
  twist:
    linear:
      x: -0.0596012136974
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0
  covariance: [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, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
header:
  seq: 3060
  stamp:
    secs: 1680093214
    nsecs: 750045231
  frame_id: "odom"
child_frame_id: "base_link"
pose:
  pose:
    position:
      x: -3.61225543616
      y: -0.121803118017
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.017649522253
      w: 0.999844235051
  covariance: [0.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2, 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, 0.0, 0.4]
twist:
  twist:
    linear:
      x: -0.0591677503251
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0
  covariance: [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, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
---
```

**** To initialize “vesc/odom” topic**

$ joy

-> After connecting with xbox controller, you need to initialize the joy node.

LiDAR: Message Structure

```
hmcl@hmcl-desktop:~$ rosmsg info sensor_msgs/LaserScan
std_msgs/Header header  # timestamp in the header is the acquisition time of the first ray in
  uint32 seq            the scan
  time stamp
  string frame_id
float32 angle_min       # start angle of the scan [rad]
float32 angle_max       # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]
float32 time_increment  # time between measurements [seconds]
float32 scan_time       # time between scans [seconds]
float32 range_min       # minimum range value [m]
float32 range_max       # maximum range value [m]
float32[] ranges        # range data [m]
float32[] intensities   # intensity data
```

Array A[2160]: A[i] is distance measurement of i^{th} step.

* sensor_msgs/LaserScan:
http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html

VESC: Message Structure

```
hmcl@hmcl-desktop:~$ rosmsg info nav_msgs/Odometry
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

```
hmcl@hmcl-desktop:~$ rosmsg info ackermann_msgs/AckermannDriveStamped
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
ackermann_msgs/AckermannDrive drive
  float32 steering_angle
  float32 steering_angle_velocity
  float32 speed
  float32 acceleration
  float32 jerk
```

* nav_msgs/Odometry:
http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/Odometry.html

* Ackermann_msgs/AckermannDriveStamped:
http://docs.ros.org/en/melodic/api/ackermann_msgs/html/msg/AckermannDriveStamped.html

Automatic Emergency Braking with TTC

1.

```
#TO DO: Tune parameters
#PARAMS
VELOCITY = 2.0 # meters per second
MIN_ACC = 1.0
```

```
class EmergencyStop:
    def __init__(self):
        #Topics & Subs, Pubs
        self.stop_signal = 0
        self.velocity_sub = rospy.Subscriber("/vesc/odom", Odometry, self.callback_vel)#: Subscribe to VESC
        self.lidar_sub = rospy.Subscriber("/scan", LaserScan, self.callback)#: Subscribe to LIDAR
        self.drive_pub = rospy.Publisher('/vesc/low_level/ackermann_cmd_mux/output', AckermannDriveStamped, queue_size=10)#: Publish to drive
```

2.

```
    def callback_vel(self,data):
        #TO DO: Subscribe Current Velocity
```

3.

```
    def callback(self, data):
        #TO DO: 1. Subscribe LiDAR data
        #       2. Calculate minimum distance
        #       2. Calculate Time to Collision(TTC) based on current velocity
        #       3. Publish drive.speed. (If TTC is smaller than minimum time, stop the car )
        #
        #       Example:
        #               drive_msg = AckermannDriveStamped()
        #               drive_msg.header.stamp = rospy.Time.now()
        #               drive_msg.header.frame_id = "laser"
        #               drive_msg.drive.speed = 0
        #               self.drive_pub.publish(drive_msg)
```

Warning!!

- Do not test your algorithm with real driving first.
- To do so,
  1. Take off the wheel from the ground
  2. Send slow velocity command first using xbox controller, or publishing the message
  3. You can move any obstacle close to the vehicle to check whether your algorithm is working or not. (vehicle is static, your obstacle is dynamic in this case)
  4. If brake is applied when you place your obstacle close enough to the vehicle.
  5. Now it is ready to test with real driving.
  6. Place it on the ground, and give a soft velocity ramp to the wall.