Autonomous Car

Project Overview

I decided to take a break for 2 months and build a 1/X scale (X = 10, 12, 16, 20 etc) autonomous car for indoor environments. I want this car to serve as a platform for robotics research and experimentation. In the future I would like to implement and test the latest algorithms for perception and path planning on this car.

The underlying idea is intriguing, use a **C**ommercial **O**ff **T**he **S**helf (**COTS**) R/C chassis and convert it into a robotic platform. Another interesting idea is that the R/C vehicle uses Ackerman steering, which mimics a full sized automobile. As you know, most robots are designed around differential steering. In differential steering, vehicles such as bulldozers and tanks move their tracks at different speeds with the vehicle changing direction as a result. This mechanism is not limited to tracks of course, wheels may also be used, such as in the JetsonBot. If we want to build a self driving car, then we need to take into account the fundamental difference between the two approaches for path planning algorithms. Source: Jetson Racecar - Blog Post 1

Objectives:

1. Practice what I have taught and learnt in the past 1 year
2. Participate in autonomous racing competitions
3. Let the project serve as my CV
4. Build a general robot
5. Participate in challenges such as this one https://developer.nvidia.com/embedded/community/jetson-challenge

Timeframe: 8 weeks

Plan:

1. **Week 1** (18-22 Jun): Starts 18th June. Survey of parts, talk to people, post on forums

2. **Week 2** (25-29 Jun): Order parts, install ROS
3. **Week 3** (2-6 July): Start testing lidar and the SBC
4. **Week 4** (9-13 July): Implement localization using lidar and IMU using a table on wheels
5. **Week 5** (16-20 July): Install SBC, lidar, IMU and battery on the car.
6. **Week 6** (23-27 July): Configure the car to run with the VESC and control it remotely using keyboard
7. **Week 7** (30 Jul - 3 Aug): Explore path planning algorithms. Start implementing the MIT one
8. **Week 8** (6 - 10 Aug): Finishing touches

MIT Car base

This consists of the chassis, wheels, servo motor for steering, brushless DC motor for 4/2 wheels battery and charger. The model recommended at https://github.com/mit-racecar/hardware is for $404. The model has:

1. 4WD
2. Brushless DC motor
3. Servo motor for steering
4. An electronic speed controller (ESC)
5. 1/10 scale

This model (and similar other models) are meant for high speed, since they are meant for racing. Minimum speed is ~6 mph. That is too fast for indoor environments. Lower speed can be achieved by reducing the width of PWM (Pulse Width Modulation) signal. However the ESC on board the RC car model does not support this. Fortunately there is an open source ESC called VESC. 'V' stands for Vedder, the last name of the person who developed and open sourced this technique. VESC allows for a finer control of speed. So one needs to replace the stock ESC with VESC.

About RC Cars

There are many new terms thrown at you once you start reading about RC cars - 2/4 WD, brushed/brushless motors, rally/short course/buggies/trucks etc. Checkout this post to demystify these terms - How to Pick Out the Right R/C Vehicle. Key learnings:

1. Brushed motors are cheaper
2. Its difficult to control 2WD cars
3. Basher vehicles are usually built for durability and aren't overly concerned about how heavy the vehicle is and how quickly it can get up so speed. They are more worried about surviving that

fifteen-foot jump and not breaking every time you hit a rock. To get an idea of how durable these cars are, see this video https://youtu.be/ym9E1YG3_QQ

4. Vehicles intended for racing are very lightweight and built of more rigid, less durable material.

So it seems like I should go for a ***4WD*** car with a ***brushed*** motor. 4WD for easier control and brushed motor to control costs.

Steering Motor

An important criteria while selecting a car is a PWM controlled steering servo motor.

```
I would suggest finding a vehicle with a conventional pwm servo motor for
steering. Some of the cheapest RC cars have steering motors that do not
allow fine control.
```

Source: mit-racecar/hardware/issues/6

The above link also confirms that 4WD is not required and a smaller car will do, if it has enough space for mounting components.

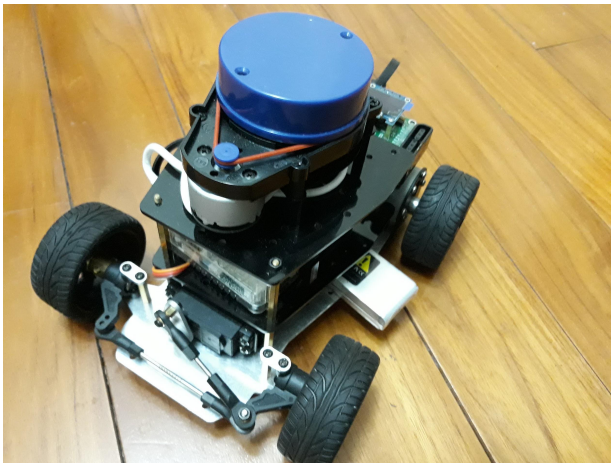
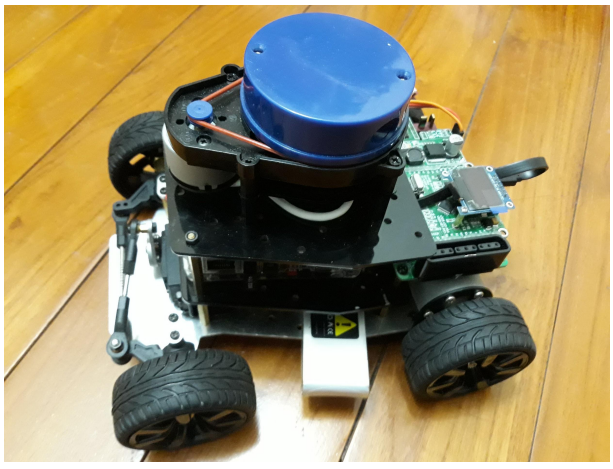
Cheaper Alternatives

There are other cheaper models, however they differ in the following aspects:

1. 2WD
2. Brushed DC motor
3. 1/12, 1/16 or 1/20 scale

Will a smaller size work? The model needs to be big enough to accommodate all the electronics without having a high center of gravity. For the components which I have ordered, a 1/16 scale or smaller model won't work since there is no space to mount all the electronics. See this video for a size comparison - Traxxas Slash 4x4 - 1/16 vs 1/10 (Comparison). I need to evaluate between the 1/12 and 1/10 scales now.

See the pictures on this 1/12 scale car. Looks like you should be able to mount electronics on this. Although the creators of HyphaROS have built a 1/20 model with the LIDAR and IMU on it:

It was not immediately clear if VESC allows for control of brushed DC motors. This thread seems to suggest that it does. If that is the case then I can probably save on car.

Other models:

1. https://www.aliexpress.com/item/1-10-Brushless-2-4G-4WD-RC-Electric-Radio-control-top-speed-racing-truck-Off-Road/32733465355.html
2. https://www.tamiyaclub.com/forum/index.php?/topic/84373-tamiya-alternative-to-traxxas-slash-4wd/
3. http://rcbazaar.com/product.aspx?productid=4066
4. http://www.hobbyandyou.com/110-scale-professional-rc-electric-car-with-high-powered-bldc-motor
5. https://www.amazon.com/Tamiya-58346-Grasshopper-RC-Car/dp/B000BMVIGM/
6. https://www.banggood.com/HSP-110-Flying-Fish-1-Drifting-2_4G-RC-On-Road-Car-94123-p-1237741.html

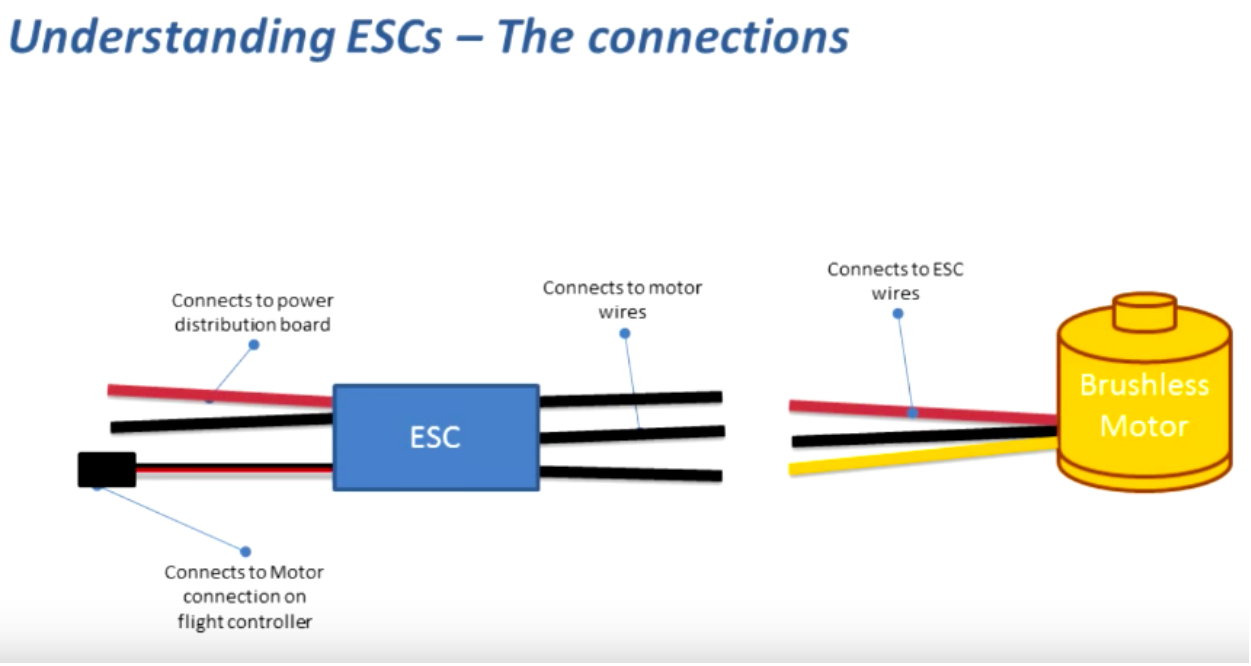
This issue thread suggests that mounting the same electronics on a 1/20 scale is possible. This can further lower the cost.

Motor Control & Battery

So now it has become clear that there are 2 well known approaches to building this car—MIT Approach and the UPenn approach. These approaches are mainly for motor control and battery. This blog post summarises the differences.

Hardware

Understanding ESC



Source: [RC Basics - Understanding Electronic Speed Controllers (ESC)]

Platform

Creating a prototype platform is discussed very well in [Jetson RACECAR Part 5 – Lower Platform Mounting and Components] and [RACECAR/J Platform Preparation].

Battery

You need 2 battery sources, one for the the drive/servo motor and the other one for all the electronic components. The TX2 board needs 19V power. Seems like it is possible to plug it in to a 3S LiPO battery directly. [This blog post comment] seems to suggest so.

There are other power consuming components - LIDAR and IMU. They need different voltages. How to supply these different voltages?

Choosing a battery

The F1/10 BOM recommends a 11.1V 3S battery. HyphaRos uses 7.4V 2S battery. Typical C rating is 30C or 25 C. So which one to use for Traxxas car?

Read this article - [Traxxas Battery Basics]. Important points:

1. Both 7.4V (2S) and 11.1V (3S) will do. Higher the voltage, faster will be the car.

2. Have a larger capacity (> 500 mAh)
3. Have a C rating > 25C
4. LiPO battery should be stored at 50% charge. Read the above article for more storage instructions.

All about lipos for RC models has more information on storage of LiPo batteries.

Travel Tips Tuesday: Safely Packing Batteries for Your Trip

Low Voltage Buzzer

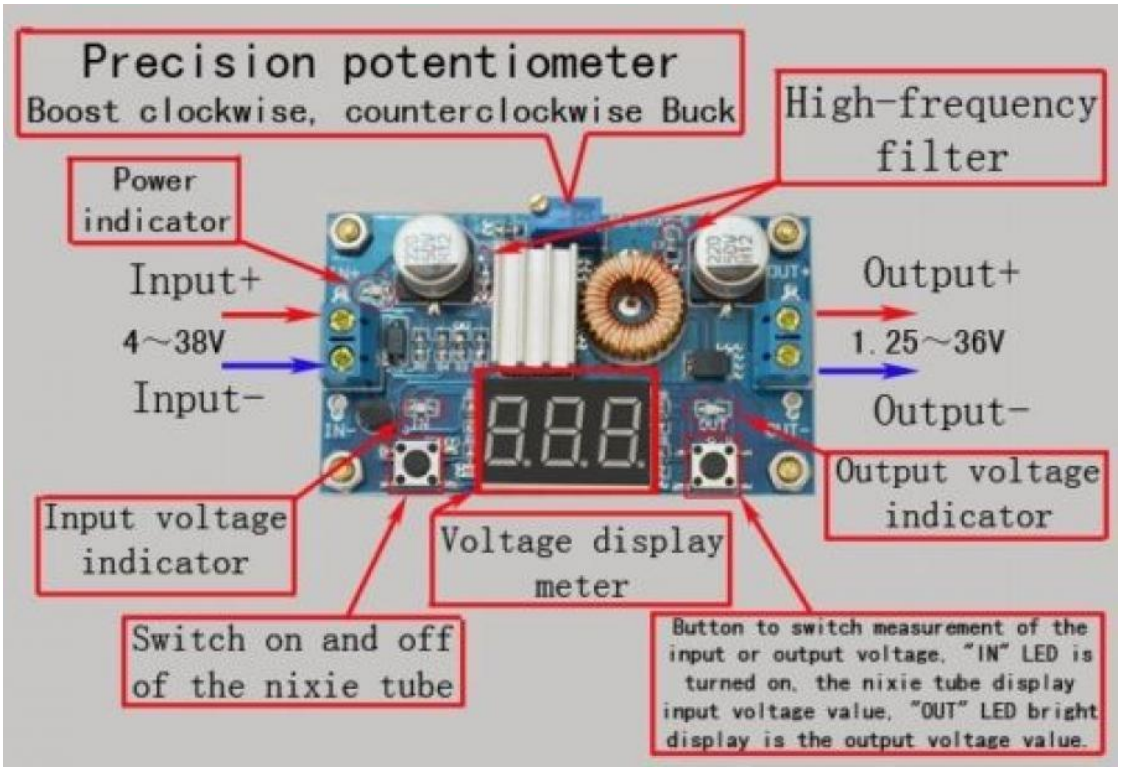
Watch this 1-8s LiPo Voltage Tester Alarm DEMO. Cheap (~ Rs 200). Available in India.

Battery eliminator circuit (BEC)

BEC is a term used mostly by hobbyists and roboticists whereas as in electronics terminology it is called a step down (or buck) DC-DC converter. It reduces an input DC voltage to a lower level. In our case we will use a 7.4V input from a LiPo battery to 5V required by the Odroid. There are 2 kinds of BEC - switching BEC (SBEC) or linear BEC. You frequently hear of another term UBEC, which is a trade name. SBEC uses a fast switching circuit to perform PWM (pulse width modulation). Whereas a linear BEC simply uses a resistance to drop down a voltage and dissipate the extra energy as heat. Hence linear BEC is not preferred for high current applications.

The Jetson Hacks model uses 2 separate power sources (or batteries) - one for the drive/servo motor and the other for electronics. The Hypha ROS model uses a single power source for everything. It uses BECs for stepping down voltage required for LIDAR, SBC and a DC2DC step up converter for the drive motor. This video is a great explanation of BECs.

In our case the Odroid's input power supply is rated 5V 5A. Lower current rating BECs are cheaper than higher current rated ones.



The one that I ordered did not work. Output voltage remained same as the input voltage no matter how much I turned the potentiometer. I had to order a replacement and wait for it. Fortunately for me discovered this on Friday and immediately ordered a new one from robu.in. It was delivered on Saturday so I could resume working on Monday. 1 day was wasted due to this. Had this not happened on a Friday, 2 days would have been wasted.

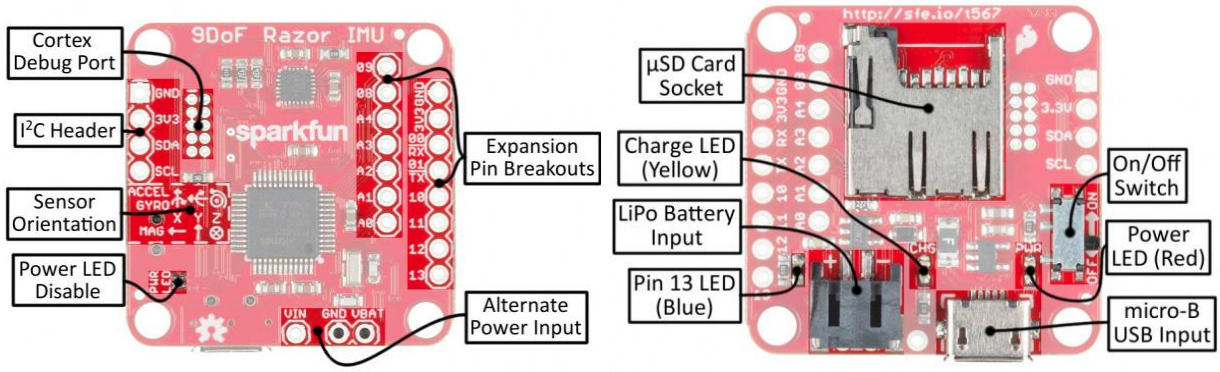
IMU

`Sidebar`: Apparently VESC can give odometry data directly. It has a wheel encoder on board (read the team 5 documentation about localization). One can fuse together the odometry data with the AMCL for localization. So is the IMU really required? According to this blog post both the UPenn F1/10 car and MIT racecar use the Sparkfun Razor 9DOF IMU. Read this blog post for more details on this IMU's functionality.

The SparkFun 9DoF Razor IMU M0 combines a SAMD21 microprocessor with an MPU-9250 9DoF (nine degrees of freedom) sensor to create a tiny, re-programmable, multi-purpose inertial measurement unit (IMU). The 9DoF Razor IMU features three, three-axis sensors – an accelerometer, gyroscope, and magnetometer – which gives it the ability to sense linear acceleration, angular rotation velocity, and magnetic field vectors. Read these Sparkfun tutorials:

1. Gyroscope Basics
2. Accelerometer Basics

`Gyros are often used on objects that are not spinning very fast at all. Aircrafts (hopefully) do not spin. Instead they rotate a few degrees on each axis. By detecting these small changes gyros help stabilize the flight of the aircraft. Also, note that the acceleration or linear velocity of the aircraft` **`does not affect`** `the measurement of the gyro. Gyros only measure angular velocity.`

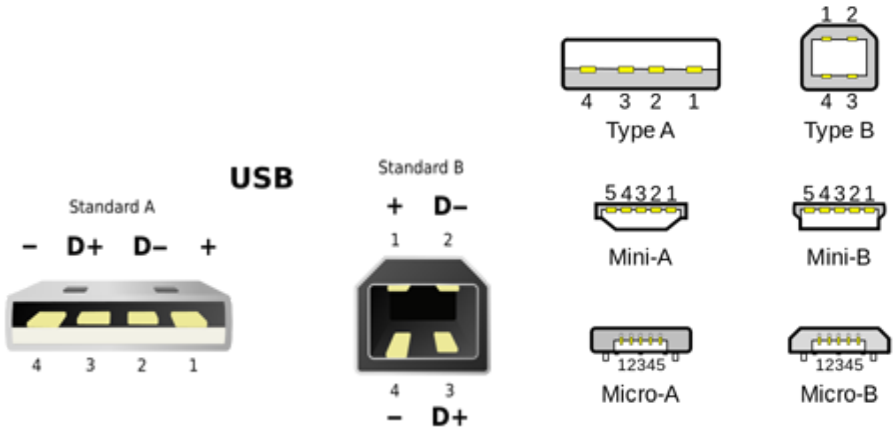


The Razor IMU is designed to work with either a **USB power source** or a single-cell Lithium-polymer (LiPo) battery. Power from either the USB or LiPo battery sources are **regulated down to 3.3V**, which is used to power both the SAMD21 and MPU-9250. The VIN, VBAT, and GND pins can be used to supply the 9DoF Razor IMU's 3.3V regulator, instead of the USB or LiPo JST inputs. Voltage on the VIN pin should not exceed 6V, and the VBAT pin should only be connected to a single-cell LiPo battery. Finally, the ON/OFF switch, on the bottom side of the board, controls power between both input sources and the rest of the components on the board.

Accessories

USB Cable Guide

This [USB buying guide] by Sparkfun is an excellent resource on different USB cables and standards. Here are some images:



99% of host controllers will have a USB-A receptacle, so when you're looking for a USB cable, you'll usually be looking for a **"USB A-to-something"** cable. A lot of the time the USB-A end is assumed, since that's the connector on most PCs, and cables will be named after the connector on the opposite end. Some smaller hosts use a Mini or Micro receptacle but they usually supply a pigtail adapter to USB-A.

A USB device can't initiate communication, only the host can, so even if you can find the right cable: connecting two devices with no host just doesn't work. Also, because USB cables carry both power and data, connecting two hosts without a device in-between can be disastrous (causing **high-currents, short circuits, even fires)**.

Serial Terminal

See this awesome Sparkfun tutorial on [Serial Terminal Basics]. It explains terms like COM ports. Baud rate. Flow control. Tx. Rx.These are all words that get thrown around a lot when working with electronics, especially microcontrollers.

USB Hub

USB hubs increase the number of USB devices that can connect to a computer without having to add additional hardware. Additionally, USB hubs can be useful with devices like laptops that can't physically add more USB ports. The difference between powered and non-powered USB hubs is that the former draws its power from an electrical outlet while the latter draws its power from the computer connection.

Source: What Is the Difference Between Powered & Non-Powered USB Hubs? Powered vs Non-powered hubs - This post tells you why to use a powered USB hub.

Again, according to this blog post, both the MIT and UPenn cars use a powered USB hub 3.0 hub, which seems like a reasonable approach if you are using as many sensors and devices as they are onboard. Can these devices be powered via the USB hub or do they need a separate power source? Answer - separate power source for LIDAR and onboard computer (SBC or single board computer).

Wire Guide

You need to be careful about the wires that you use on your system. The wire's cross sectional diameter determines its current carrying capability. Larger the diameter, larger the capacity. ***Using a wire of insufficient diameter may cause it to overheat, burn the insulation or even cause a fire!***

The odroid has a max current rating of 4A. The Odroid power supply has a wire rated 18 AWG. AWG stands for American Wire Gauge, a standard for the diameters of round, solid, nonferrous, electrically conducting wire. It starts with 0 (biggest wire) and goes upto 39 (smallest).

The AWG tables are for a single, solid, round conductor. The AWG of a stranded wire is determined by the cross-sectional area of the equivalent solid conductor. Because there are also small gaps between the strands, a stranded wire will always have a slightly larger overall diameter than a solid wire with the same AWG. Source Wikipedia

The current flows through the area closer to the surface of the conductor. Hence stranded wires have larger overall diameter. Some thumb rules:

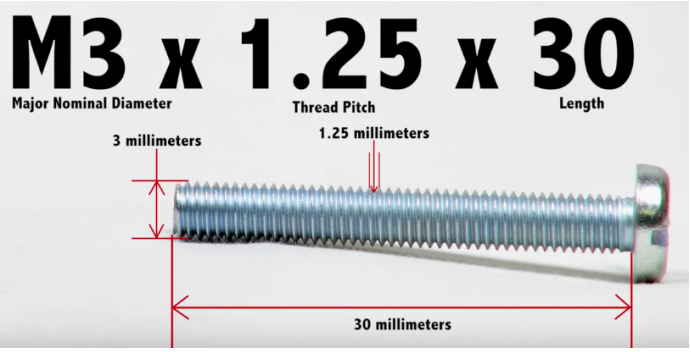
1. For the same cross section, aluminum wire has a conductivity of approximately 61% of copper, so an aluminum wire has nearly the same resistance as a copper wire smaller by 2 AWG sizes, which has 62.9% of the area.
2. Stranded wires are specified with three numbers, the overall AWG size, the number of strands, and the AWG size of a strand. The number of strands and the AWG of a strand are separated by a slash. For example, a **22 AWG 7/30** stranded wire is a 22 AWG wire made from seven strands of 30 AWG wire.

Usually wires are made of stranded copper wires with a PVC insulation. If the copper wires are coated with tin then the wires appear silver in color (TINNED COPPER VS BARE COPPER). Silicone insulation is more flexible than PVC and higher tolerance for heat. So if your wires are dissipating a lot of heat then use silicone insulated wires.

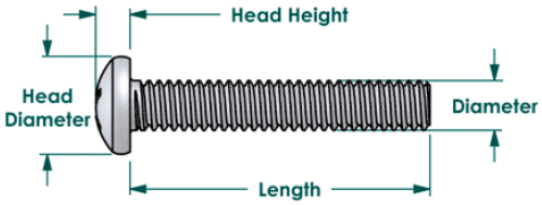
You can use this calculator to find the ampacity of the wire. I have a 18 AWG wire which is about 10 cms long. The calculator says it can handle about 8A, which is more than enough for my system.

Screw Guide

You will frequently come across terms like M3 screws, M4 screws or 4-40 machine screws. The following is a metric screw:



These are machine screws. A 4-40 machine screw would mean a screw with 4 mm diameter and a thread count of 40 threads per inch.



LIDAR

The MIT Racecar uses a Hokuyo lidar, which costs a whooping $1700!! Way beyond my budget for the entire car. Hokuyo has the following specs:

1. Range: upto 10 meters
2. Angular resolution: 0.25 degrees
3. Angular range: 270 degrees
4. Scanning frequency: 40 hz (2400 rpm)

On searching online cheaper alternatives were available.

Specs	**Hokuyo**	**RP LIDAR A1**	**RP LIDAR A2**	**Scanse Sweep**
Range (meters)	10	12	12	12
Angular resolution (deg)	0.25	1	0.9	3.6
Angular range (deg)	270	360	360	

Scan Frequency (hz)	40	2 - 10	10	10
Points frequency (hz)	43k!!	2000	>= 4000	1000
Accuracy +/- (mm)	40			
Weight (g)	130	200	340	
Power (V)	12 or 24	5	5	
Price ($)	1680	99	310	

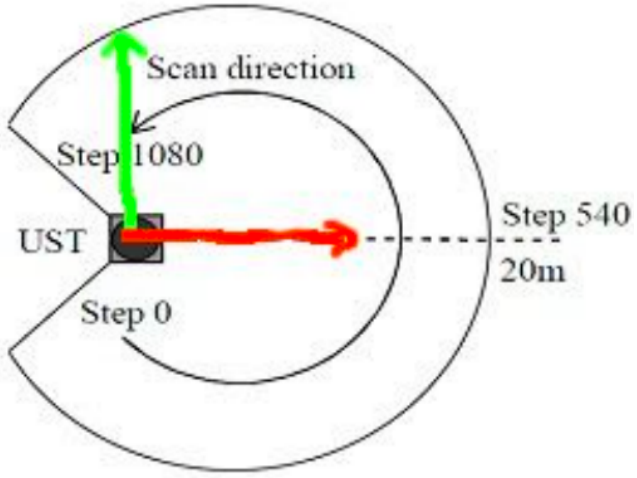
More comparisons here: [Comparing low cost 2d scanning lidars]

Clearly the RP Lidar wins out over Hokuyo on cost and over sweep on angular resolution and frequency. I ordered RPLidar A1M8 from https://www.dfrobot.com/. The shipment arrived from Beijing to Delhi in 2 days, however the Delhi customs department caused a delay of 20 days.

Laser Data

- The ranges list is 1081 points long (indexes 0 – 1080)
  - The LIDAR has 270° vision
  - 270 x 4 = 1080; measurements every 0.25°

Measurement steps 1081
Detection angle 270°
Angular resolution 0.25°



Source: La Canada High School Lecture 8

In the case of RPLidar A1M8, the angular resolution is 1 degree, so the length of `ranges` list is 360. The `LaserScan` message looks like this:

```
header:
  seq: 987
  stamp:
    secs: 1536241653
    nsecs: 174742419
  frame_id: "laser"
angle_min: -3.12413907051 # -179 degrees
angle_max: 3.14159274101 # 180 degrees
angle_increment: 0.0174532923847  # 1 degree
time_increment: 0.000221567141125
scan_time: 0.079542607069
range_min: 0.15000000596
range_max: 12.0
ranges: [ list with 360 readings in it ]
```

Onboard Computer

At $299 the Jetson TX2 is a very expensive choice for the onboard computer. Pros: Has GPU, powerful processor, onboard camera and wifi module. Cons: Cost.

Cheaper alternatives: Odroid XU4 and Raspberry Pi 3
https://www.electromaker.io/blog/article/odroid-xu4-vs-raspberry-pi-3-b

If you don't need GPU capabilities but need a powerful onboard computer so that you can perform localization using ray casting then Odroid XU4 seems to be sufficient. **Keep in mind that the RAM for XU4 is only 2GB whereas the one on TX2 is 8GB.**

Criteria	Odroid XU4	Jetson TX2 Development Kit
RAM (GB)	2	8
CPU	ARM Cortex-A15 / ARM Cortex-A7 (32-bit) 2GHz octa core	4x ARM Cortex-A57 + NVIDIA Denver2 (dual-core) (64-bit) 2GHz hexa-core
GPU	Mali-T628 MP6	Nvidia Pascal (256 cores, 1300MHz)
Input Voltage V	4.8V - 5.1	5.5V - 19.6

Dimensions (cm)	8.2 x 5.8 x 2.2	17 x 17.1 x 6
Weight	38 grams (without fan)	1600 grams
Price ($)	79	599 (299 with discount)

As you can see the development kit is 6 times larger, 40 times heavier and about 7.5 times costlier than XU4!! Source of this information - [Solved] Dimensions of the Jetson TX1 developer kit and Comparison between ODROID-XU4, Jetson TX2 | Board-DB.org

Keep in mind that in the TX2 development kit the TX2 module comes with a carrier board. The weight of just the TX2 module is 85 grams and dimensions are 50 x 87 x 6 cm3.

Running Code on Workstation

Can we offload computation via wifi to my laptop and hence go for a cheaper onboard computer? **I have decided to use an onboard computer to make development easier.**

However this YouTube video: SLAM with LIDAR and wheel encoders in ROS, proves that you can run the heavy computations on a workstation and transfer data from the robot over wifi:

> Also, you likely won't be able to do the SLAM+planning with any kind of fluidity directly on the rpi3; instead, you'll need to connect it with a nearby laptop or desktop over wifi (this is fairly straightforward if both are running linux and you can SSH from one to the other using private keys to avoid entering passwords; just set **ROS_IP_REMOTE**, **ROS_MASTER_URI**, and **ROS_IP** correctly on both. This also works over openvpn, btw, though it obviously adds more latency.)

The comment also says that you won't be able to do any heavy computations on the Raspberry Pi 3.

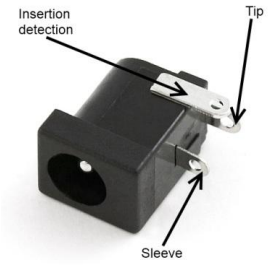
Voltage levels on Odroid XU4

All the native GPIO pins on the XU4 operate at 1.8V. However, many devices support only 3.3V or 5V. Eg. Sparkfun 9DOF IMU has IO levels of 3.3 V, too high for Odroid. Source: XU4 Shifter Shield. IMO you can skip the GPIO and connect the IMU to the USB hub if it has a USB slot. I have posted this issue on the Hypha ROS repo, awaiting confirmation (**Update**: Confirmed. The newer IMU doesn't need FTDI or Arduino Nano since it as a USB output). If it does not have a USB slot then you can use a FTDI board. More on FTDI - How does an ftdi chip work?. Also see this thread - RAZOR IMU Integration with the XU4 OF Poppy.

More on FTDI - Sparkfun USB to serial UART boards hookup guide

DC Cable Connector

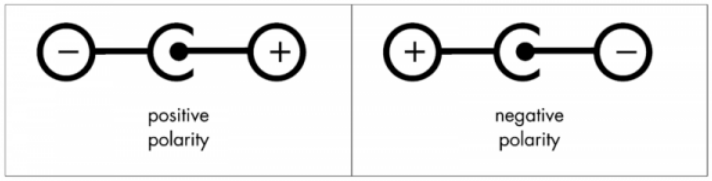
The Odroid is requires a 5V DC supply rated 4A. It has a female barrel jack connector on board. It looks like this:



The power supply which comes with the Odroid has an AC wall adaptor which directly converts AC power supply to 5V DC. The male barrel connector at the end of the power supply looks something like this:

The outer cylinder is called the `sleeve` and inner `tip`. There are 2 kinds of polarity for such connectors:

1. Sleeve 0 and pin positive (positive polarity). Most common type of connector.
2. Sleeve positive and pin 0 (negative polarity). Less common



The Odroid needs a positive polarity supply i.e. ***inside (pin) is positive and outside (sleeve) is negative***. So when you are trying to connect a battery power to the Odroid, you need to ensure 2 things:

1. Input is 5V and can supply at least 4A.
2. The barrel connector plug has positive polarity

You can measure the polarity with a multimeter before plugging the cable into the Odroid. If your battery supplies a different voltage then use a voltage regulator. In my case I had a 7.4V LiPo battery. So I had to use a buck converter (or BEC in robotics parlance) to convert the 7.4 to 5.

Wifi Remote

I had to buy a wifi remote for the car. Any off the shelf RC remote with 2 channels will work. 1 channel for throttle and 1 for steering. I bought this one:

[FS-GT2B 2.4G 3CH Radio Model Remote Control Transmitter & Receiver for RC Car]

It is a 3 channel remote. The third channel can be used to switch between manual and autonomous modes. You can have more channels for advanced functionality.

Receiver Setup

Steering connection goes into channel 1. ESC goes into channel 2. This video was helpful [How To: Replace or Swap out Traxxas Receiver].

Resources

Other Projects

1. https://diyrobocars.com/diy-robocar-projects/
2. BERKELEY AUTONOMOUS RACE CAR
3. Auto Rally
4. F1/10
5. HyphaROS

HyphaROS

A project which built this car in < $600! https://github.com/Hypha-ROS/hypharos_racecar

This project does not use a Jetson, uses a Odroid board for processing. Does not use VESC.

Blog Post Series

There are many different blog post series out there which have tutorials to build the MIT Racecar.

1. Jetson Racecar: Starts Jan 2016 and goes until June 2017. You can treat this as the v1 of Jetson racecar. This deals with controlling the servo motor for steering and drive motor directly using **Jetson TK1** using a PWM driver **PCA9685.** The driver generates PWM signals for the stock ESC on the Traxxas car. This series also discusses creating prototype mount platforms using acrylic boards, stand-offs and screws.

2. Racecar/J: Starts June 2017. You can treat this as V2 of the Jetson racecar. This version makes many modifications, cosmetic and important ones. Cosmetic improvements—use better laser cut platforms to mount the electronics, replace front bumper for better protection, replace springs for better suspension (the electronics & battery are heavy). Use a custom ESC (VESC) to control the motors.

3. Team 5 Documentation: This series of blog posts is by the TAs of the MIT course 6.141 Robotics: Science & Systems class. These blog posts deal with the theory about perception, localization and path planning algorithms.

Blog post series #1 (Jetson Racecar) and #2 (Racecar/J) mostly deal with hardware and software dependencies.

Master's Thesis

I hope to get answers to some of the unknowns by referring to this master's thesis. This thesis is based on the F1/10 model of UPenn. The authors wanted to participate in the F1/10 competition. The date for this thesis is in May 2017, so not a long time back. An interesting note from this thesis

```
Unfortunately, the April 2017 round of the competition was cancelled,
because the organizers "did not receive enough team confirmations in time
for organizing, planning, and ensuring an exciting racing event"
```

This tells you that this car is not easy to assemble and program. Also it is time consuming and if you get stuck on technical issues then help is not very easy to find.

`Currently we use the Gazebo simulator for car simulation. The developed simulator is the topic of Jiří Kerner's thesis (see [10]).`

Interesting, so I can also learn how to build the simulator.

The thesis also talks about sensor fusion. The authors fused together the pose from the AMCL and IMU with a **Unscented Kalman Filter (UKF)**. It also gives details about the UKF implementation.

The thesis also talks about identification of the mechanism for steering & drive. Read through sections **3.10 Car Identification**. Section 3.11 talks about IMU experiments. The IMU has an accelerometer, gyroscope & magnetometer. The thesis talks about fusing readings from these 3 different sensors with the AMCL. The IMU readings are affected by the motor vibration. Hence tuning of the provided measurement covariance matrix is required by making experiments.

In section 3.13.4 the thesis concludes that the ZED camera is useless.

F1/10 Lectures

Lecture 2.1 - Sensors

Video. Comments:

1. Effects of vibration on IMU data is filtered out by Kalman filtering of the raw data? (Time 2:57)

Lecture 2.2 - Localization

Video. It explains how odometry is generated using laser scans by performing scan matching.

Lecture 2.3 - PID Control

Video. Explains PID control.

Tutorial 6 - Distance Finder and PID Control

Video. The aim of this tutorial is to give a brief explanation of simple algorithm to maintain the car parallel in a corridor. It involves how to use the sensor data from lidar and how to formulate a controller for the same.

Assignment 2 Driving Straight

Video. Implementation of tutorial 6.

Ackermann Messages

We are interested in mobile robots using Ackermann steering geometry, as shown in this diagram (from Wikipedia). Since these robots cannot turn in place, they are difficult for the standard ROS navigation stack to support.

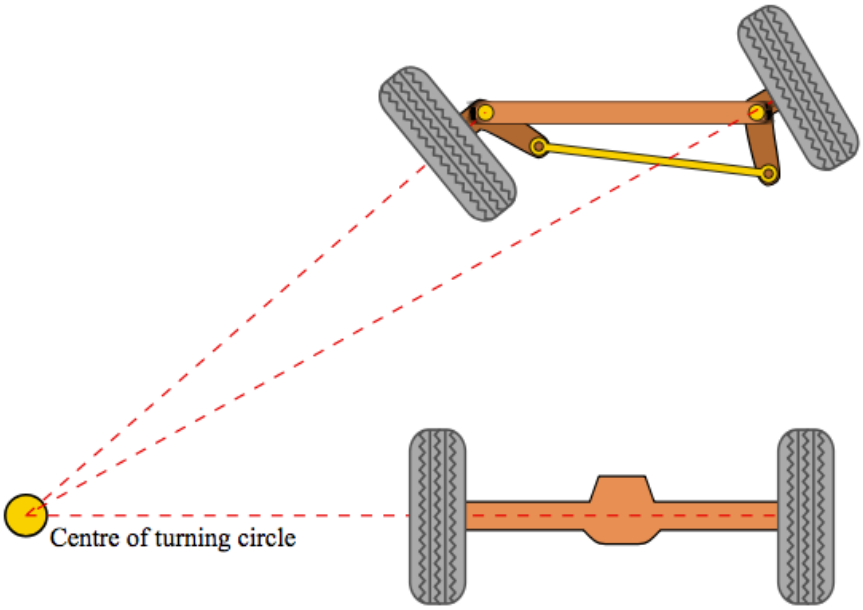


Diagram by User:Bromskloss [GFDL (www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0 (www.creativecommons.org/licenses/by-sa/3.0/)], via Wikimedia Commons. Original image cropped by Andy Dingley.

Read this interesting group discussion, which made the AckermannDriveStamped.

Virtual steering angle

For a car-like robot, the left and right front wheel are in general not steered by the same amount (in order to make sure that the IRC is uniquely defined). However, in order to limit the number of command, people normally give as input the steering angle corresponding to a virtual wheel located at the center of the front axle. If the steering wheels are steered by independent motors, the individual steerings must be computed from the steering of the virtual wheel. If there is a mechanical linkage between the wheels (as in a real car), then this takes care of moving the steering units appropriately.

Holonomic v/s non-holonomic

Source Robot Locomotion. Based on the way robots move, they can be further classified as "Holonomic" or "Non-Holonomic" drive Robots.

Holonomic refers to the relationship between controllable and total degrees of freedom of a robot. A robot built on castor wheels or Omni-wheels is a good example of Holonomic drive as it can freely move in any direction and the controllable degrees of freedom is equal to total degrees of freedom.

If the controllable degree of freedom is less than the total degrees of freedom, then it is known as non-Holonomic drive. A car has three degrees of freedom; i.e. its position in two axes and its orientation. However, there are only two controllable degrees of freedom which are acceleration (or braking) and turning angle of steering wheel. This makes it difficult for the driver to turn the car in any direction (unless the car skids or slides).

Control messages for holonomic robots

A standard interface for differential drive and holonomic robots typically takes in geometry_msgs/Twist, which is:

```
# This expresses velocity in free space broken into linear and angular
parts.
Vector3  linear
Vector3  angular
```

Is this the best solution for Ackermann robots as well? No. Steering angle is important. Although it is theoretically possible to compute that from velocity and yaw rate using some trigonometry, there are significant practical problems. When the vehicle accelerates, the *changing velocity mandates a changing yaw rate to model a constant steering angle*.

Path Planning

Visualization

This repository was very helpful - https://github.com/AtsushiSakai/PythonRobotics. It contains Python sample codes for robotics algorithms with some sample code.

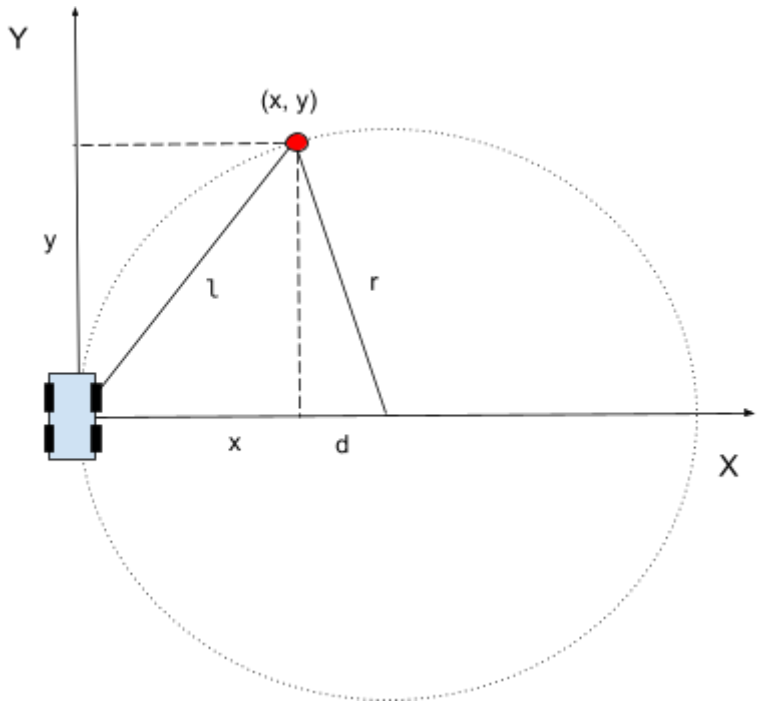
Pure Pursuit

What is pure pursuit?

This document was helpful in understanding it Implementation of the Pure Pursuit Path Tracking Algorithm.

Pure pursuit is a tracking algorithm that works by calculating the curvature that will move a vehicle from its current position to some goal position. The whole point of the algorithm is to choose a goal position that is some distance ahead of the vehicle on the path. The name pure pursuit comes from the analogy that we use to describe the method. We tend to think of the vehicle as chasing a point on the path some distance ahead of it - it is pursuing that moving point. That analogy is often used to compare this method to the way humans drive. We tend to look some distance in front of the car and head toward that spot. This lookahead distance changes as we drive to reflect the twist of the road and vision occlusions.

Geometry of the algorithm



The following equations hold:

$$x^2 + y^2 = l^2 \qquad (1)$$

$$x + d = r \qquad (2)$$

$$d^2 + y^2 = r^2 \qquad (3)$$

So by doing some elementary algebra, you can derive `r` as a function of `x` and `l`.

$$r = \frac{l^2}{2x}$$

The curvature has been related to the `x` offset of the goal point fcm the origin by the inverse square of the lookahead distance `l`.

Effect of lookahead distance

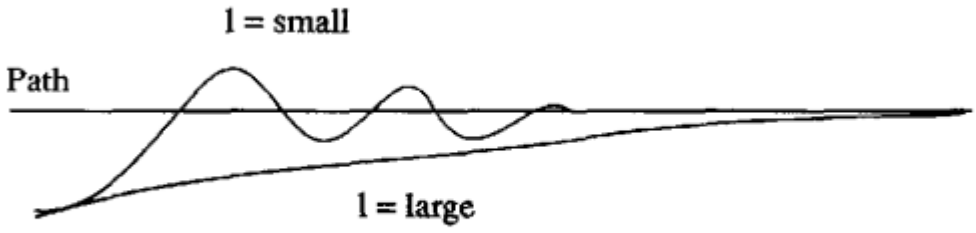


Figure 2. **Regaining the path.**

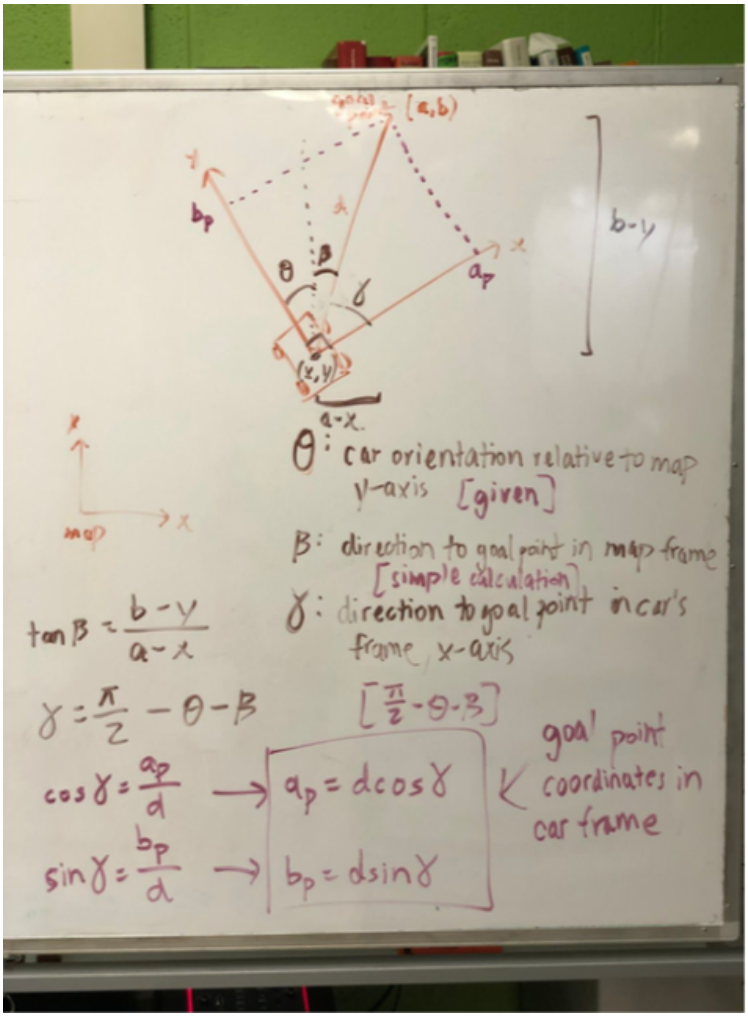
Disadvantages of pure pursuit

Note that there are also limitations for our pure pursuit algorithm. Here is a list of them:

1. Car is at constant velocity, on straightaways and turns. Ideally we want faster straightaways and slower turns.
2. Look ahead distance is constant for straightaways and turns. We probably want a larger lookahead distance for straightaways so car doesn't oscillate left and right when it goes faster, and a smaller look ahead distance for turns so car doesn't look too far ahead and end up cutting off a corner too tight and hitting the wall.
3. We want the code to be able to do loop closure, that is have the car know when it is completing a loop and continue back to the first index.

Transforming points from map to car

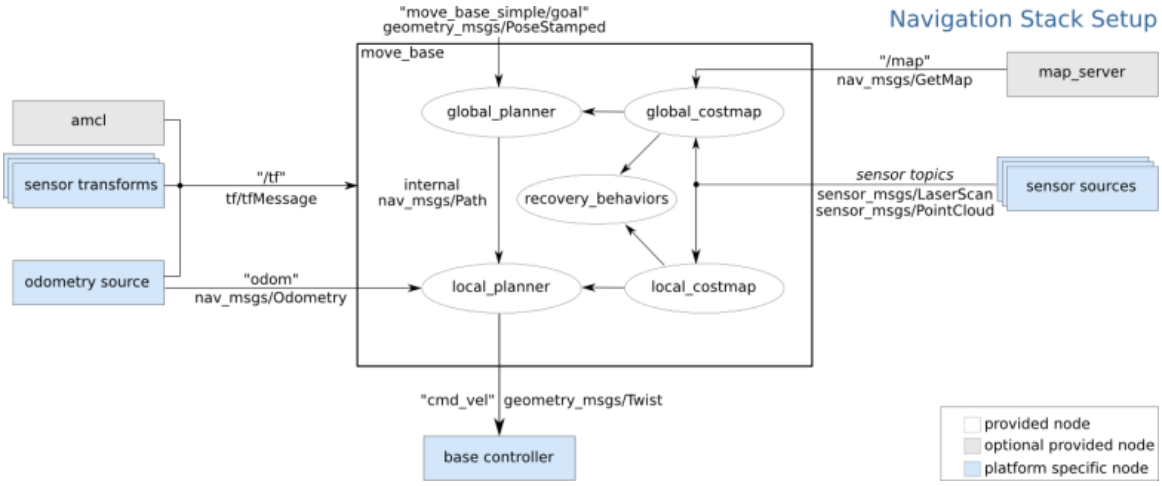
The following whiteboard diagram explains the derivation of UPenn instructors (*Question: Why isn't tf transforms package used for this?*).



The build guide also explains the algorithm with pictures.

ROS `move_base`

The `move_base` package lets you move a robot to desired positions using the navigation stack.



One needs to first get familiar with the setup of the navigation stack. See this tutorial Note: [Setup and Configuration of the Navigation Stack on a Robot](#).

- The navigation stack uses information from sensors to avoid obstacles in the world, it assumes that these sensors are publishing either `sensor_msgs/LaserScan` or `sensor_msgs/PointCloud` messages over ROS.
- The navigation stack requires that odometry information be published using tf and the `nav_msgs/Odometry` message.
- The navigation stack assumes that it can send velocity commands using a `geometry_msgs/Twist` message assumed to be in the base coordinate frame of the robot on the "`cmd_vel`" topic. This means there must be a node subscribing to the "`cmd_vel`" topic that is capable of taking (vx, vy, vtheta) <==> (cmd_vel.linear.x, cmd_vel.linear.y, cmd_vel.angular.z) velocities and converting them into motor commands to send to a mobile base i.e. Ackermann Messages.
- The navigation stack does not require a map to operate, but we'll use one.

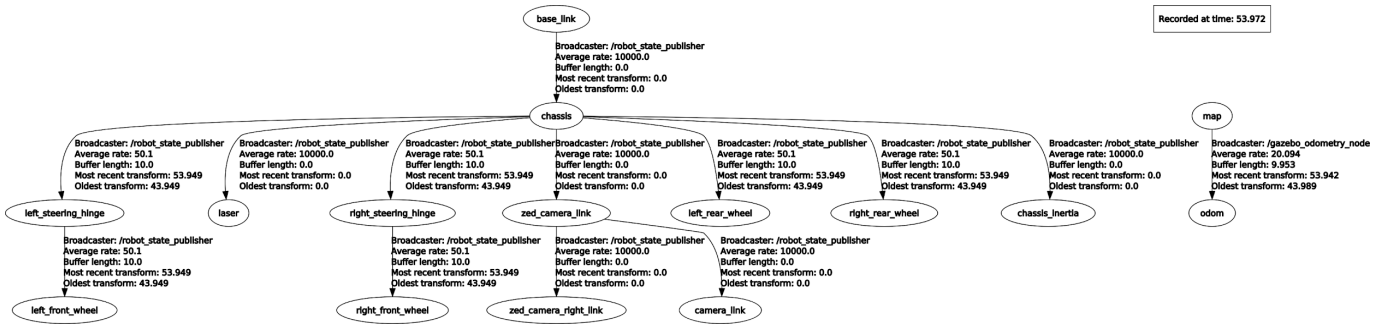
So the navigation stack requires localization component as well as an odometry component. And it requires laser scans.

Transform Tree

The transforms from `map` to `odom` and `odom` to `base_link` need to be published. So the setup that we have with the racecar should work well:

1. VESC publishes the `odom -> base_link` transform
2. Localization component publishes the `map -> odom` transform
3. RPLidar provides `LaserScan` messages

I decided to test this on the simulator first. However when I visualized the transform tree it looked like this:



There was a `map` to `odom` transform but no `odom` to `base_link` transform. On some digging I realized that I have to set the following in `racecar/config/racecar-v2/vesc.yaml`:

```
# publish odom to base link tf
vesc_to_odom/publish_tf: true
```

Still nothing. Then I realized that the `vesc_to_odom` node has not been launched yet. So I added the following code to my launch file:

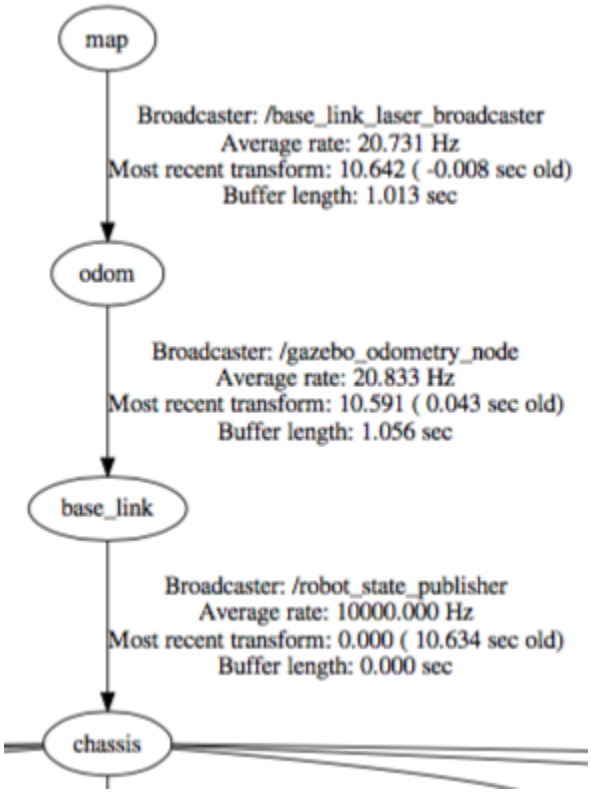
```
<arg name="vesc_config" default="$(find racecar)/config/$(arg
racecar_version)/vesc.yaml" />
<rosparam file="$(arg vesc_config)" command="load" />

<!--publish odom to base_link transform-->
<node pkg="vesc_ackermann" type="vesc_to_odom_node" name="vesc_to_odom">
    <param name="publish_tf" value="true"/>
</node>
```

Still no odometry or transform being published. Then I noticed that the `vesc_to_odom` node subscribes to a topic `/sensors/core` which has messages of type `vesc_msgs/VescStateStamped`. The `vesc_driver` node publishes messages on this topic after reading data from the VESC. So that means I cannot use the `vesc_to_odom` node for odometry while I am using a simulation. I can only use it on the car.

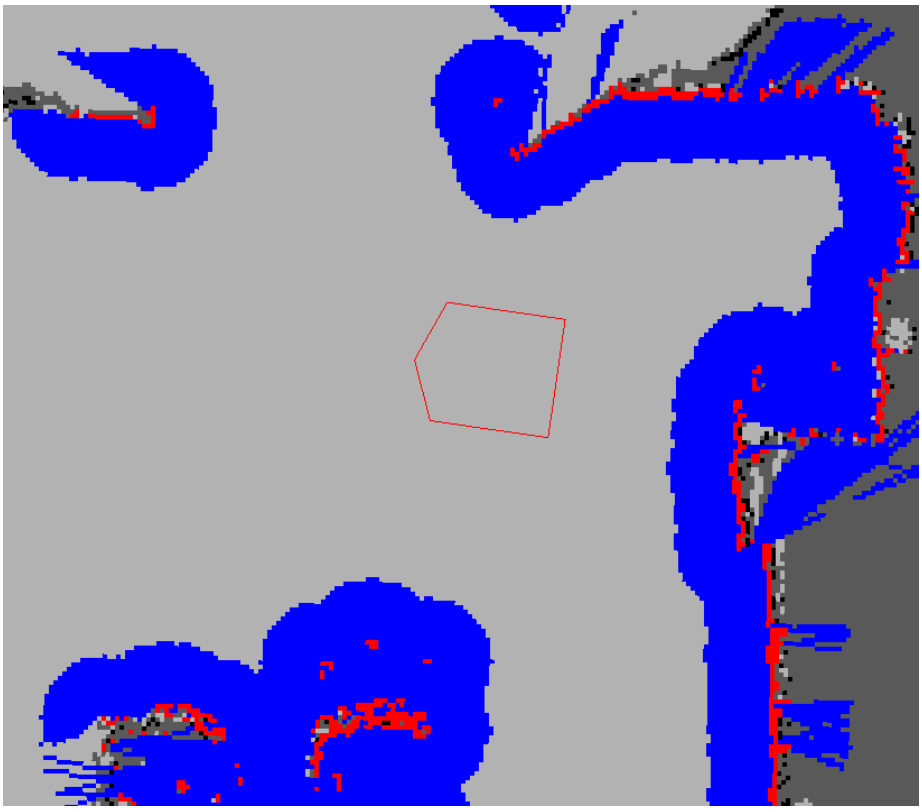
Workaround: The node `gazebo_odometry_node` (in `racecar_gazebo/scripts/gazebo_odometry.py`) is publishing these things:

1. Transform from `map` to `odom`.
2. Pose of the robot on the topic `/vesc/odom` or `/pf/pose/odom` with *frame_id* `map` and *child_frame_id* `odom`.

To make it work with move_base I decided to keep the `odom` frame at the `map` frame by publishing a static transform. And I changed the frame_ids in `gazebo_odometry.py` so that it publishes a transform from `odom` to `base_link`. Now the transform tree is good to be used with move_base:



Costmap_2d



In the picture above, the red cells represent obstacles in the costmap, the blue cells represent obstacles inflated by the inscribed radius of the robot, and the red polygon represents the footprint of the robot. For

the robot to avoid collision, the footprint of the robot should never intersect a red cell and the center point of the robot should never cross a blue cell. Documentation is here.

Each bit of functionality exists in a *layer*. For instance, the static map is one layer, and the obstacles are another layer. By default, the obstacle layer maintains information three dimensionally (see voxel_grid). Maintaining 3D obstacle data allows the layer to deal with **marking and clearing more intelligently**.

Marking and clearing

The costmap automatically subscribes to sensors topics over ROS and updates itself accordingly. Each sensor is used to either mark (insert obstacle information into the costmap), clear (remove obstacle information from the costmap), or both.

Static Map Layer

The parameters for this layer are explained here.

Obstacle Layer

The parameters for this layer are explained here.

Inflation Layer

Inflation is the process of propagating cost values out from occupied cells that decrease with distance.

Following goals

As mentioned earlier the move_base node publishes a `geometry_msgs/Twist` message assumed to be in the base coordinate frame of the robot on the "`cmd_vel`" topic.

Thanks to an idea from the F1/10 team (from their build guide v2), you can create a simple control node which subscribes to the /cmd_vel topic and generates ackermann messages with the following code:

```
velocity = sqrt(x^2 + y^2)
steering angle = atan2(WHEELBASE_LENGTH * theta_dot / velocity)
// where theta_dot is the z angular velocity (aka yaw)
```

Note that because the default local planner in move_base is designed for differential drive robots (robots that can spin in place, like the Roomba vacuum cleaner robots), the paths that are generated are not ideal for our car which is an Ackermann steering robot. TEB (Timed Elastic Band) local planner which can be used for Ackermann robots.

Sending goals

Move_base node subscribes to the topic `/move_base_simple/goal`, on which one can publish geometry_msgs/PoseStamped messages for the next goal of the robot.

```
rostopic pub -1 /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 1
  stamp: now
  frame_id: 'map'
pose:
  position:
```

```
    x: -5.97
    y: -0.44
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: -0.76
    w: 0.65"

rostopic pub -1 /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 1
  stamp: now
  frame_id: 'map'
pose:
  position:
    x: 2.0
    y: 2.0
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 1.0"

rostopic pub -1 /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 0
  stamp:
    secs: 1541327779
    nsecs: 975647974
  frame_id: "map"
pose:
  position:
    x: -4.88958501816
    y: 1.53346705437
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.999200105167
    w: -0.0399893715077"
```

Terrace:

```
rostopic pub -1 /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 0
  stamp:
    secs: 1541404781
    nsecs: 834938637
  frame_id: "map"
pose:
  position:
```

```
    x: 6.98282814026
    y: -0.0308477580547
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.539703699524
    w: 0.841855044958"

rostopic pub -1 /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 1
  stamp:
    secs: 1541404872
    nsecs: 669876874
  frame_id: "map"
pose:
  position:
    x: -4.42678451538
    y: 0.609466791153
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: -0.694155995779
    w: 0.719824599138"
```

[ERROR] [1542098207.355370086]: Client [/waypoint_server] wants topic /move_base/status to have datatype/md5sum [actionlib_msgs/GoalStatus/d388f9b87b3c471f784434d671988d4a], but our version has [actionlib_msgs/GoalStatusArray/8b2b82f13216d0a8ea88bd3af735e619]. Dropping connection.

Or one can use rviz to publish the next goal using the GUI. It works well, it tested it when running the simulation in Gazebo! Rviz also publishes on the topic `/move_base_simple/goal`.

Generating waypoints

Now let us drive the car around using a joystick and generate waypoints for the car to follow.

TEB Local Planner

The time elastic band local planner is supposed to be an improvement over move_base's local planner and it also has support for car like robot. It works as a plugin for `move_base`. Instead of using the `base_local_planner` you use the `teb_local_planner`.

It has great documentation and tutorials http://wiki.ros.org/teb_local_planner/Tutorials/. Get started here - Setup and Test Optimization. The tutorials repository teb_local_planner_tutorials was super useful with a sample launch file and parameters.

Costmap conversion

The teb_local_planner package supports costmap_converter plugins. Those plugins convert occupied costmap_2d cells to geometric primitives (points, lines, polygons) as obstacles.

Without activating any plugin, each occupied costmap cell is treated as single point-obstacle. This leads to a large amount of obstacles if the resolution of the map is high and may introduce longer computation times or instabilities in the calculation of distinctive topologies (which depend on the number of obstacles). On the other hand, the conversion of obstacles also takes time. However, the conversion time strongly depends on the selected algorithm and it can be performed in a separate thread.

However right now this feature is experimental. So we will give this a miss.

Problems

I faced a few issues with the teb_local_planner:

1. **Very low translational velocity**: The planner output control signals with translational velocity as low as 0.08. That is not sufficient for my car to move. So I had to use this workaround - Allow setting of minimum velocity.
2. **Oscillations**: See this issue Oscillations around global path while using teb local planner? My problem is exactly the one faced by the author of OP. Too much back and forth performed by the car.
3. **Improper `robot_base_frame`**: As mentioned in this post - Steering axis of carlike robot with teb_local_planner - the robot_base_frame needs to be attached to the rear axle. Mine was attached to the `base_link` which is the center of the robot. However as mentioned by another user in the oscillations link (bullet point #2) this does not fix the issue of oscillations. Will attaching a large weight for backward motion help? No it did not.

Click the above image to view the video

You can see that the oscillations are too much. I increased the maximum angular velocity and acceleration and there wasn't any zig zagging any more.

However now there was another problem - *overshoot*. The relationship between max steering angle, minimum turning and wheelbase is:

```
max_steering_angle=atan(wheelbase/min_turning_radius)
```

The wheelbase of my car is 0.34 and the measured minimum turning radius is 0.82m (I drove the car using the joystick with a maximum steering angle). So the maximum steering angle comes to around 21.8 degrees, which is **0.38 radians**.

I echoed messages from `/vesc/high_level/ackermann_cmd_mux/input/nav_0` (topic on which the control commands are sent to VESC). I noticed that the steering angle sent by the controller are above the maximum limit. When this happens the speed is 0.5 (which is the min speed set in the controller). This means the TEB controller is sending speeds which are below the minimum allowable speed. T

Another problem. I kept getting this error:

```
[WARN]: Could not transform the global plan to the frame of the controller

[ERROR] [1540891292.516547788]: Extrapolation Error: Lookup would require
extrapolation into the future.  Requested time 1540891292.505149570 but the
latest data is at time 1540891292.470663776, when looking up transform from
frame [odom] to frame [map]

[ERROR]: Global Frame: odom Plan Frame size 12: map
```

These errors seemed to be related to amcl's `transform_tolerance` parameter. This parameter future dates the map to odom transform published by `amcl`. I tried setting different values for this -0.1, 0.1, 0.2, 0.3 etc. Nothing worked. This was quite frustrating. I wasted 2 days in this.

~~**[Solution]** I suspected that the errors might be due to the low update rate of amcl. So I decided to give MIT's particle filter a shot. Update: The MIT's particle filter performs ~28 iterations per second. I tried it and it fixed these problems.~~

~~I am guessing amcl's update rate was very low on my machine. The gap between 2 readings must have been greater than `transform_tolerance` parameter. This parameter specifies the acceptable delay in transforms required by `move_base` - map to odom and odom to base_link.~~

~~Imagine this case: if at any instance `move_base` is requesting a map to odom transform and if more than `transform_tolerance` seconds have passed since the last transform, the next valid transform would be into the future (since amcl post dates transforms using its own `transform_tolerance` parameter). I tried reducing amcl's transform tolerance, even made it zero, but the problem remained. Another approach would be to increase move_base's transform tolerance (it was already at 0.5 for me), however that would cause errors in path planning since the robot would have moved by a non-trivial distance during that time.~~

[**Solution**] Along with trying out the particle filter I also changed the `global_frame` parameter in local_costmap_params.yaml to `map` from `odom`. Apparently this fixes the above errors. Even if you use the particle filter and set the `global_frame` to `odom`, you will still get the above errors.

One problem fixed, onto another :(I put the goal in the last bed room. This would mean the car has to enter a narrow corridor and enter through an even narrow door. However the car seemed to get stuck at the entrance of the corridor itself. It seemed like the costmap allows a very narrow path through the entrance. So may I should decrease *minimum obstacle distance, inflation radius* etc.

My problem seemed to be the same, as described over here Problem moving differential drive robot through narrow corridors. The thread doesn't seem to have a solution yet.

The global planner trajectory seems to work fine. However the local planner trajectory is all over the place. I need some more insights into how the planner works. I got them over here Differential drive favor backwards movement teb_local_planner.

The local planner follows the global reference plan by tracking the farest available intermediate pose from the global plan within the local costmap (and also not exceeding *max_global_plan_lookahead_dist*).

Ideally, the global plan already contains reasonable and valid orientations since the global planner is aware of the complete map. Since some global planners do not even provide the orientation part for intermediate poses (NavFcn) or just support forward orientations (default in GlobalPlanner) the teb_local_planner may overwrite orientations by default (*global_plan_overwrite_orientation*) except the final orientation.

The following strategy is implemented currently for *global_plan_overwrite_orientation==true*:

Reference goal orientation:

- if global goal (last pose of the global plan): take desired final orientation
- if intermediate goal: the reference orientation of local goals is chosen according to the tangential vectors between pose k and pose k+1 in the global plan (always forward).

Initial Trajectory (TEB) for optimziation

The local trajectory is initialized to the current reference goal (see above, within local costmap). Note, the optimizer just finds local minima, hence the initialization is quite important.

- if *allow_init_with_backwards_motion==true* and goal is located behind the robot (w.r.t. heading): initialize all orientations backwards for the subsequent optimization.
- Else: initialize with forward orientations.

This strategy has the following implications: If the global goal is close to the robot (within the local costmap window) and is located behind it, the robot drives backwards. Otherwise, the robot always tries to drive forward.

In my opinion, if you would also initialize backwards for goals outside of the local costmap window, you would have to drive backwards along the complete global plan until you reach the global goal. Otherwise, we would need to add more assumptions and heuristics which is not appropriate for a local planner that has just a limited view of the environment. I think, this might be better decided by a global planner in general as mentioned above. Furthermore, most robots have more sensors in their front hence

keeping the default strategy to drive forwards for long distances and backwards just for short ways is reasonable in my eyes. But I would be happy to discuss whether better or more intuitive strategies exist.

The params affecting this behaviour are:

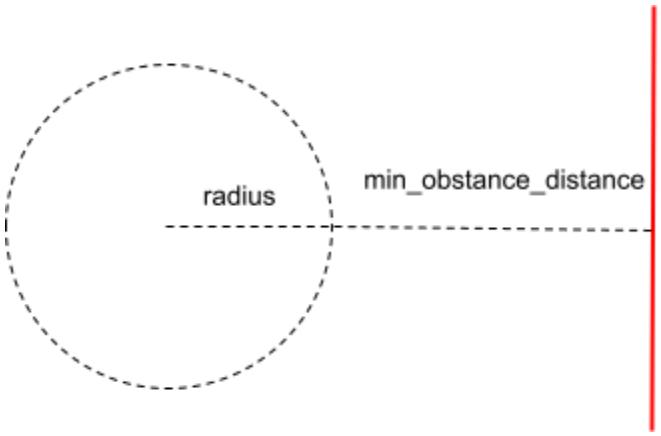
1. max_global_plan_lookahead_dist
2. global_plan_overwrite_orientation
3. allow_init_with_backwards_motion

Also, maybe, I should increase weight_kinematics_turning_radius. It defines the penalty weight for satisfying min_turning_radius for carlike robots. I will try these things tomorrow.

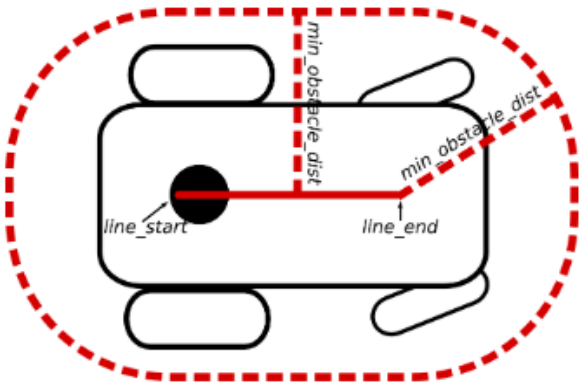
Footprint Model

To see how parameters weight_obstacle, min_obstacle_dist, penalty_epsilon, dt_ref, obstacle_poses_affected affect trajectory planning, see this Tutorials/Obstacle Avoidance and Robot Footprint Model.

My earlier robot footprint model was a circle with a radius of 0.28m.



The entrance to the corridor was only 0.96 m wide. My minimum obstacle distance was set to 0.2m. This meant that the corridor width was lesser than 0.2 + 0.28x2 + 0.2 = 0.98. So I decided to switch to a line model.



A line length of 0.3m and minimum obstacle distance of 0.2m should do the job. My robot's length is 0.56m and width is 0.3m. The line robot is useful for robots that exhibit different expansions/lengths in the longitudinal and lateral directions, like mine.

The concluding lines from the above link drove home the point for me:

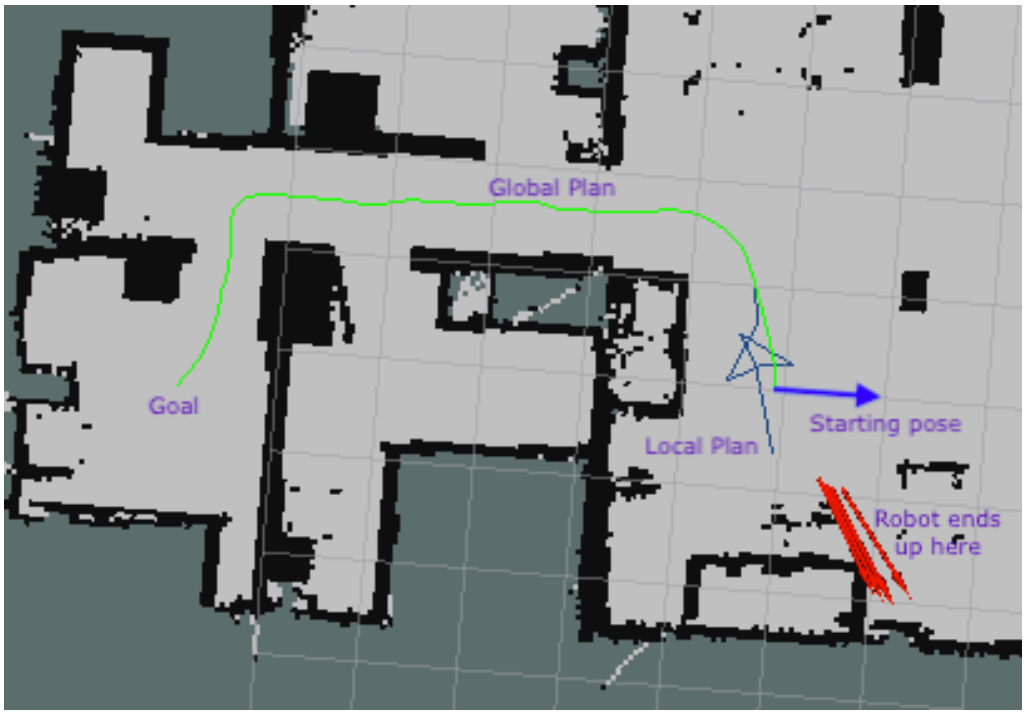
- `If you are driving in narrow environments, make sure to configure the obstacle avoidance behavior`
- `(local planner and global planner) properly. Otherwise, the local planner might reject an`
- `infeasible trajectory (from its point of view), but the global planner in contrast could further`
- `think that the selected (global) plan is feasible: the robot could get stuck.`

This was exactly what was happening with me. Hopefully by correcting the footprint model this would get fixed.

Back to the problem. I made the following changes:

1. Changed the footprint from circle to line.
2. Increased from `weight_kinematics_turning_radius` 1 to 500.

None of it worked. The global and local plan now look like this:



The local plan seems infeasible, it does not follow the motion model constraints. I decided to try everything in the simulator first. I would need to create a model of my house in gazebo.

More problems still persist. I opened a couple of issues on github:

1. Robot gets stuck entering a narrow pathway

2. Robot stops very often, even on straight paths with no obstacles around

My problems are similar to the ones faced by others:

1. Navigation problems when using TEB on UGV
2. teb_local_planner: avoid constant path replanning
3. Trajectory is not feasible
4. Unexpected robot oscillations even with tutorial example

The robot has problems following a straight path. Its steering over shoots. There is too much back and forth. It has problems navigating in tight spaces and it stops very often.

MIT's Particle Filter

I tried MIT's particle filter and all the errors related to amcl went away went away. It also worked much faster than amcl.

It publishes a `map to laser` transform directly. One needs to modify the code so that it publishes a `map to odom` transform instead. The repository contains code which allows one to publish a `map to base_link` transform. This is a good starting point.

Rotations - Matrices and Quaternions

This document was helpful in understanding rotations.

If you look at the `geometry_msgs/Pose` message, you will notice that it gives you the orientation of the robot as a `geometry_msgs/Quaternion`. There are four main representations of rotations:

1. Fixed-axis angles
2. Euler angles
3. Rotation matrix
4. Quaternion

In this class, we will mostly use the rotation matrix and quaternion representations.

Fixed-axis / Euler angles

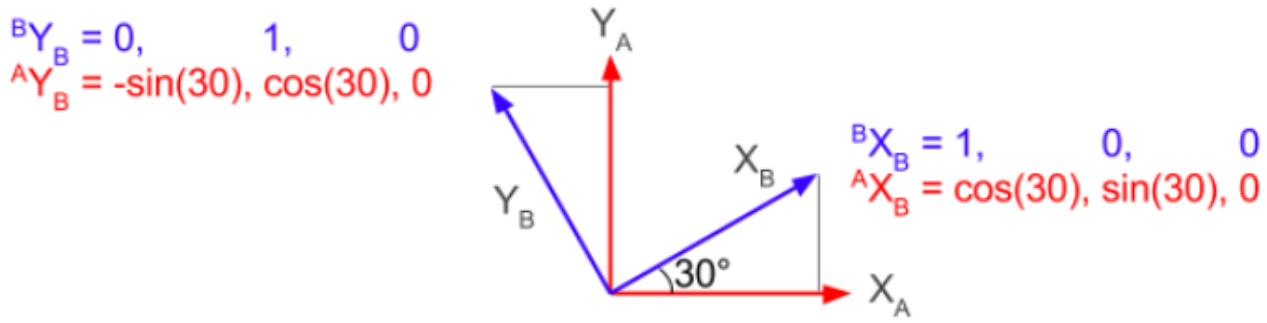
In these representations, you use 3 parameters to express rotations about the X, Y, or Z axes. In the fixed-axis representation, you rotate about the X/Y/Z axes of a fixed frame. In the Euler angle representation, the first rotation (e.g., about the X-axis) generates an intermediate frame. The second rotation is then rotated about an axis (e.g., the Y-axis) of this intermediate frame, which generates a second intermediate frame. *Finally, the third rotation is applied to the second intermediate frame*. In both representations, the order in which you apply these rotations matters. Additionally, which axes you rotate about can differ (e.g., x-y-z, z-x-z), etc. **The advantage of this representation is that it only uses 3 parameters.** Disadvantage - Gimbal Lock.

Rotation matrix

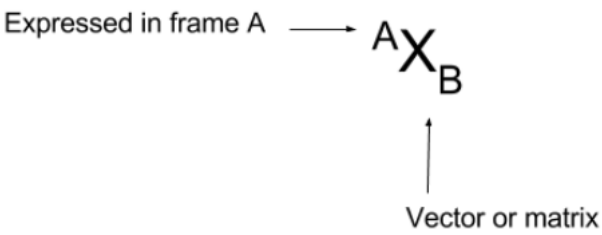
The 3x3 rotation matrix is probably the most useful representation of an orientation, **but the downside is that it uses 9 parameters to represent**.

Imagine a two coordinate frames, A and B. B is the same as A, but rotated about the Z-axis of A by 30 degrees. Then note the following:

1. The Z-axes of the two frames are the same.
2. The XB, the unit X-axis (1, 0, 0) of B, is (cos(30), sin(30), 0) in the coordinates of A.
3. YB is (-sin(30), cos(30), 0) in the coordinates of A.



We use a leading superscript to specify which coordinate frame a vector or matrix is expressed in.



The rotation matrix ${}^{A}R_{B}$ specifies how frame B is rotated relative to frame A. The cool thing that makes rotation matrices so useful is that they are extremely easy to interpret. If there's only one thing you learn from this entire section, it's the sentence below:

The columns of a rotation matrix ${}^{A}R_{B}$ are the unit X, Y, and Z vectors of B, expressed in the coordinate frame of A.

In our example with the frame rotated 30 degrees about the Z-axis, the rotation matrix ${}^{A}R_{B}$ is:

```
| cos(30)  -sin(30)  0 |
| sin(30)   cos(30)  0 |
| 0          0       1 |
```

Notice that the first column, cos(30), sin(30), 0, is the same as ${}^{A}X_{B}$, and notice that the second column is ${}^{A}Y_{B}$. The third column is 0, 0, 1, which tells you that the Z-axis of B is the same as the Z-axis of A.

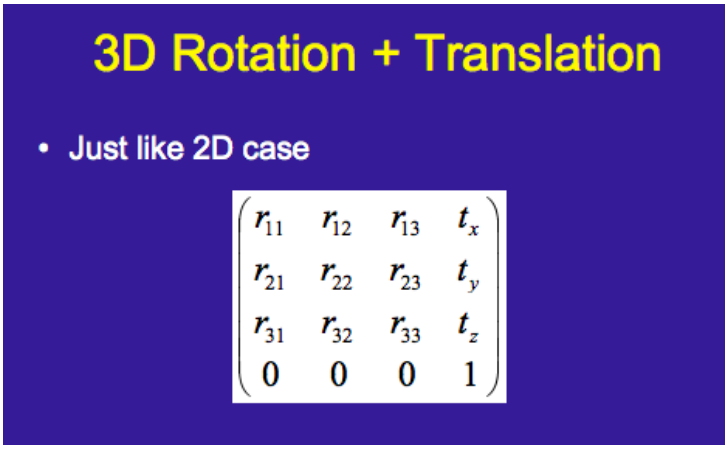
Test your knowledge: What rotation does this rotation matrix describe?

```
|  0.698, -0.707, 0.111 |
|  0.698,  0.707, 0.111 |
| -0.156,  0,     0.988 |
```

This matrix looks confusing at first, but once you think about it as columns of B, it makes more sense.

- The unit X vector of B, 0.698, 0.698, -0.156, points at a 45 degree angle in quadrant 1, but also points slightly downward.
- The unit Y vector of B, -0.707, 0.707, 0, points at a 135 degree angle (quadrant 2) and is in the X-Y plane of A.
- The unit Z vector of B, 0.111, 0.111, 0.988, mostly points up (it is approximately 0, 0, 1).

Note that the columns of a rotation matrix must be of unit length.



The rotation matrix, `R`, is in the top left 3x3 corner and the translation vector `T` is the 4th column. This is applied (dot product) to a vector `X`, which is given as `(x, y, z, 1)`. You can split this equation as `R.dot(X) + T`.

For 2D robotics i.e. where there is no movement along Z axis the only rotation would be in the form of Euler angle `yaw`. In that case the rotation matrix, which gives the rotation around Z axis, is given by:

```
| cos(yaw)  -sin(yaw)  0 |
| sin(yaw)   cos(yaw)  0 |
| 0          0         1 |
```

Quaternions

Quaternions are 4-dimension complex numbers of the form w + ix + jy + kz, where i, j, and k are imaginary numbers such that i2 = j2 = k2 = ijk = -1. Quaternions have a number of advantages, including being easy to interpolate, avoiding gimbal lock (Gimbal Lock Explained), and being relatively compact. However, the main disadvantage is that they are hard to interpret.

`(Lecture)`: Introducing The Quaternions and Quaternions and Rotations

Converting between rotation matrices and quaternions

In Python, use the `tf.transformations` module. In this module, quaternions are represented as vectors of the form `[x, y, z, w]`. Matrices are represented as a 4x4 numpy arrays.

`tf.transformations` uses 4x4 homogeneous transform matrices, which is a generalization of 3x3 rotation matrices. In a homogeneous transform matrix, the rotation matrix is embedded in the upper left corner. The 4th row and column are both 0, 0, 0, 1.

```
>>> import tf.transformations as tft
>>> import numpy as np
>>> help(tft) # Use this to look up documentation.
>>> mat = tft.quaternion_matrix([0, 0, 0, 1])
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
>>> mat[:3, :3]
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> tft.quaternion_from_matrix(mat)
array([ 0.,  0.,  0.,  1.])
>>> mat2 = np.array([[0.866, -0.5,   0, 0], # sin(30), -cos(30)
                     [0.5,    0.866, 0, 0],
                     [0,      0,     1, 0],
                     [0,      0,     0, 1]])
>>> tft.quaternion_from_matrix(mat2)
array([ 0.        ,  0.        ,  0.25882081,  0.96591925])
>>> from geometry_msgs.msg import Quaternion
>>> yaw_by_30 = Quaternion(x=0, y=0, z=0.25882081, w=0.96591925)
```

MIT's Approach

MIT explains its approach in Lab 6 of their course. The Challenge Design Document mentions this:

In order to understand the ad-hoc obstacles, a local occupancy grid relative to the well-localized robot is created. Laser scan ranges are projected into the global frame and the resulting occupancy grid is published, and the planner efficiently integrates the local and global maps to use for planning.

In order to efficiently implement local mapping, the transformation from the laser frame to the global map indices is precomputed with information from tf and the map's resolution information. This transformation is then applied to the laser scan points with ***numpy vectorization***. The occupancy grid is limited to the points which lie in a small square around the robot's location in order to reduce the cost of publishing the map, and the local map grid squares are aligned to those of the global map for rapid integration to the global map on the planner's side.

Setup

Odroid XU4

Here is a very good video giving an overview of the hardware on XU4 - ODROID-XU4 Hardware Introduction. Read this post to get started with Odroid XU4 - Getting started with the Odroid XU4.

The Odroid XU4 can serve as a standalone computer. You can connect a monitor to the XU4 using an HDMI cable and a mouse and keyboard as well, **giving you full desktop capabilities.** However it comes without any OS installed. There's a slew of operating system (OS) choices for the Odroid XU4. Many Linux distros run on the Odroid XU4 including Ubuntu, Kali Linux, Armbian, and Arch Linux.

You can download your preferred OS image from here Odroid XU4 Wiki: OS images. Under Linux it has Ubuntu 16.04 and 18.04. The latest ROS release Melodic Morenia supports Ubuntu Artful (17.1) and Ubuntu Mate (18.04 Bionic Beaver). For Ubuntu Mate it has 2 options - minimal and mate. The minimal option does not give you a desktop OS. So we will go with the 'Mate' option. The image itself is 4GB in size. So you need a storage with at least 8GB of space. Odroid supports microSD or eMMC memory. eMMC is faster, just like how SSD speeds up your machine. However it is quite expensive than a SD card. I am using 16GB microSD card.

First you need to flash the storage with the image of the OS. You can use this easy to use app https://etcher.io/ to flash the microSD card. This app is available for Mac and Linux OSes. The process is:

1. Open the etcher app
2. Select the OS image
3. Select the SD card
4. Click 'flash'

Next: Bootup the odroid for the first time. For all the next set of instructions, please follow this document: Headless Setup with Ubuntu Mate. To summarize the document has instructions for remote desktop access to your XU4 if you don't have a spare mouse, keyboard and monitor. It has a guide to perform the following steps:

1. Insert the SD card on XU4
2. Switch the storage selector to microSD card (or eMMC if you are using it)
3. Connect the ethernet cable from your router to the XU4
4. Switch on the power supply
5. Figure out the XU4's IP address
6. SSH into the XU4
7. Install VNC server (the post recommends x11vnc)
8. Start the VNC server

To figure out the XU4's IP address I logged into my router (http://192.168.0.1) and went to "Setup -> LAN Setup". Over there I could find "odroid" in the DHCP client list, along with other machines on the network. When I tried to start the VNC server I got an error:

-auth guess: failed for display=':0'

While trying to debug this issue I stumbled across this What is display 0?. Based on my hunch I decided to prepend `sudo` to my command and it worked! My joy was short lived. Every few minutes I kept getting this error and the server shut down:

***** stack smashing detected ***: <unknown> terminated**

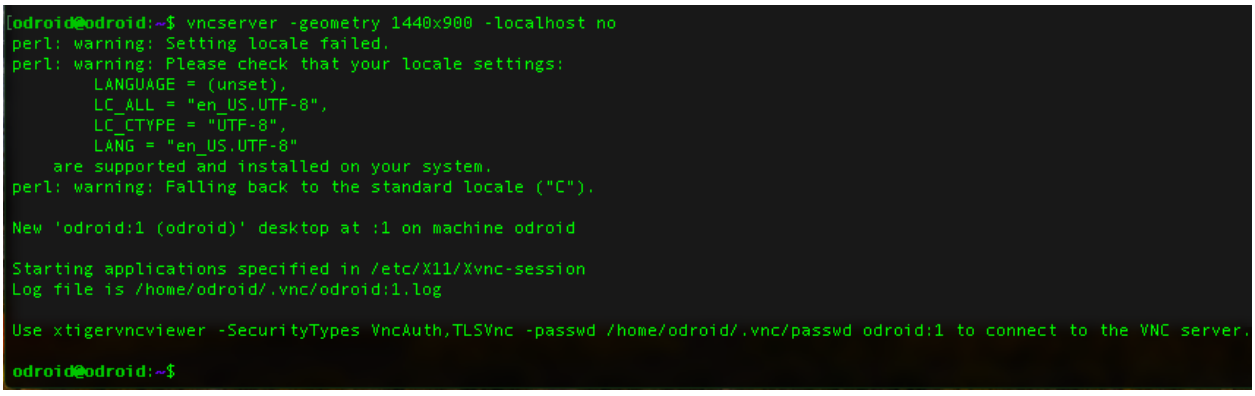
I stumbled upon this issue on the x11vnc github repository - Stack smashing error every few minutes on Arch Linux. It turns out that the original author of this tool handed over the development and maintenance to the open source community. The author's last release was in 2011, this release is installed by the package manager (apt). The community has released 2 versions after that, the latest being in Feb 2018. Seems like the latest version should help fix this bug. But the installation is not available via the package manager and one has to build it from the source. *Apparently TigerVNC and TightVNC are much better than x11vnc. I will give it a try tomorrow.*

Sidenote: The wifi setup for odroid is simple, just plug in the wifi router module and access the XU4 via a VNC viewer. You will be able to see wifi connections and connect to them.

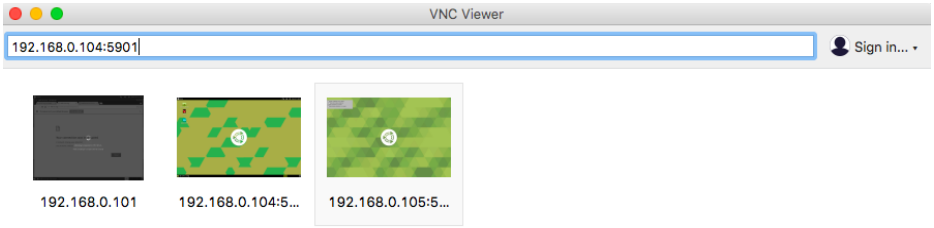
Finally managed to install TigerVNC Server on the XU4. Running the server is simple, just run the command `vncserver` in the terminal. The RealVNC viewer, however, could not connect to the server. I got the error "`Connection Refused by host`". After a lot of pulling my hair out, I stumbled on this issue github:tigervnc/issues/117: unable to connect to socket: Connection refused(10061). Running the server using `-localhost no` option made it work for me. Also this helped a lot TigerVNC - ArchWiki. The final command is:

```
vncserver -geometry 1440x900 -localhost no
```

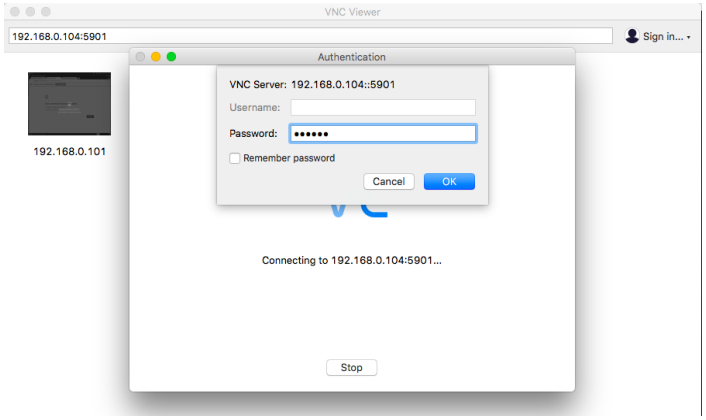
On the terminal you should see something like this:



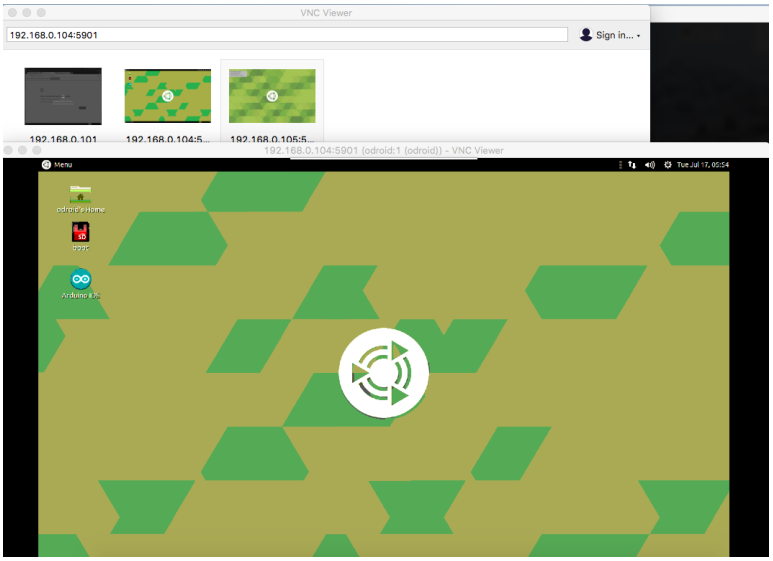
Next you need to install "VNC viewer" on your host machine so that you can have remote desktop access to your XU4. I installed VNC viewer for mac from realvnc.com. To start the VNC viewer, open the app and add the IP as `X.X.X.X:5901`, where `X.X.X.X` is the Odroid's IP address.



You will prompted for a password, it is `odroid`.



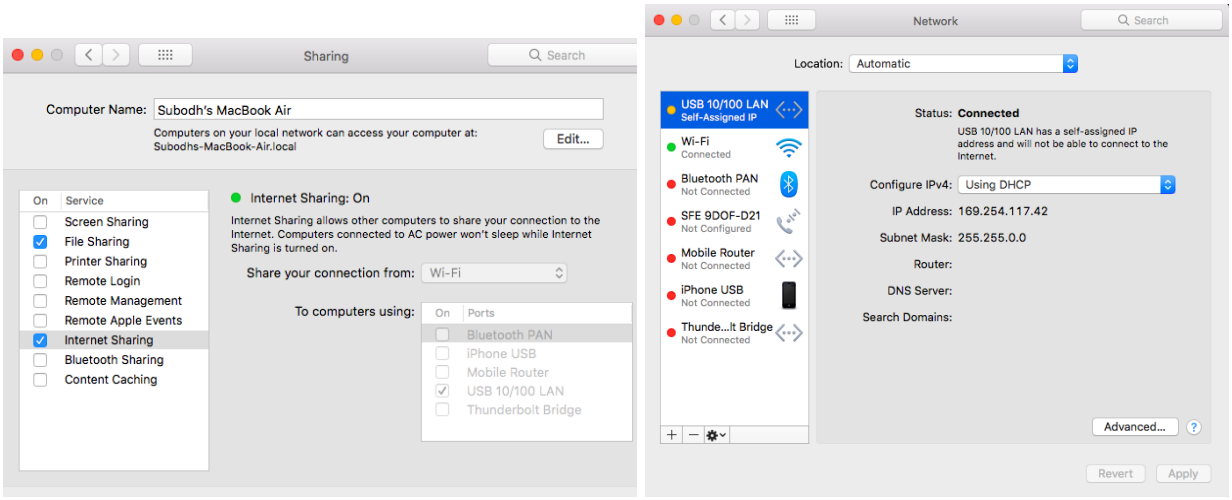
A new window will open up where you have desktop access to your odroid. In full screen mode you won't realize that it is a virtual connection!

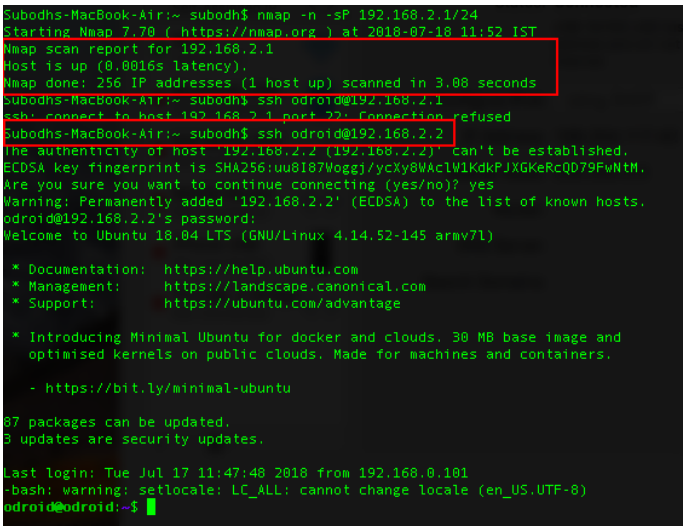


I got an ethernet to USB adapter so that I could directly connect the Odroid to my Mac, instead of connecting it to the router. This post [How to SSH into your Raspberry Pi with a Mac and Ethernet Cable] and [my comment on this post] helped me do this. Steps:

1. Power on the XU4
2. Connect the ethernet cable to your Mac's USB port via the adapter
3. Open 'Sharing' preferences, turn on internet sharing for port `USB 10/100 LAN`

4. Open ‘Network Preferences and check if `USB 10/100 LAN` is connected and has been assigned an IP
5. Run the command ifconfig and search for `bridge100`
6. Get the IP under `inet` of `bridge100`
7. Run `nmap -n -sP X.X.X.X/24` using the IP address from step 6
8. You should see 2 hosts - one for your computer and one for Odroid. However I could only see 1 host at `192.168.2.1`
9. I tried my luck by doing `ssh odroid@192.168.2.2` and it worked!





RPLIDAR A1M8

Setting it up was quite easy. Just clone the [rplidar_ros] repository in the `src` folder and run `catkin_make`. And launch it using `roslaunch rplidar_ros rplidar.launch`. Next: visualize output using rviz.

I could not use rviz on odroid, so I will try to run it on VM. To do this I read through these sections of the doc:

1. [Running ROS across multiple machines]

2. ROS NetworkSetup

Still some confusion existed about the environment variable ROS_IP. Without it the Rviz could not communicate with the master process on the Odroid. Finally figured this out after reading Specification of ROS_MASTER_URI and ROS_HOSTNAME .

Let us assume the Odroid IP address is `192.168.2.2` and the VM's IP address is `10.0.2.15` (you can find this by running the command `ifconfig`). *We will run all our nodes on the Odroid and only Rviz on the VM*. On the Odroid perform these steps:

```
export ROS_MASTER_URI=http://192.168.2.2:11311
export ROS_IP=192.168.2.2
catkin_make
source devel/setup.bash
roslaunch rplidar_ros rplidar.launch
```

On the VM perform these steps:

```
export ROS_MASTER_URI=http://192.168.2.2:11311
export ROS_IP=10.0.2.15
rosrun rviz rviz -d ~/.rviz/rplidar.rviz
```

Note that the master URI is same for both, the ROS_IP values are different. I was finally able to visualize the output in rviz. FYI - the laser data is published on the `/scan` topic. You can run the command `rostopic list`,in a new tab, to find this. **Note**: If you open a new tab then you have to define the `ROS_MASTER_URI` and `ROS_IP` again the new tab.

I later found out that the above setup resulted in a one directional communication. I could ping Odroid from the VM but I could not ping the VM from the Odroid or my host machine. The problem was with VirtualBox network setup. I was using **NAT**. I setup port forwarding for port 23, but the problem persisted. Then I changed the network to **Bridged Adapter**. The problem persisted. Then in advanced settings I set the promiscuous mode to *"Allow All"*. Now I could ping VM from the Odroid. However I could not ping the Odroid from the VM now and the ROS nodes on the VM were unable to connect to the master, running on the Odroid. Also I lost shared internet access. So I reverted back to the NAT mode. I spent half a day on this and could not figure out the problem. **I decided to use a workaround. Any publisher had to be on the Odroid. Only listeners could be on the VM.**

For the power supply, I found this in the datasheet:

```
Power Supply and Consumption

Ranging scanner system and motor system are powered separately in RPLIDAR
A1. External system should provide power supply for them separately in
order to ensure data accuracy. Below chart showed a recommended power mode.
More specification is provided in the following table.
```

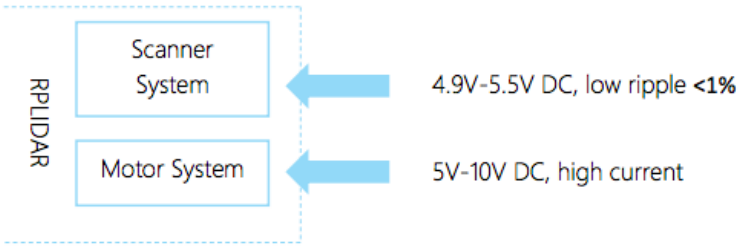


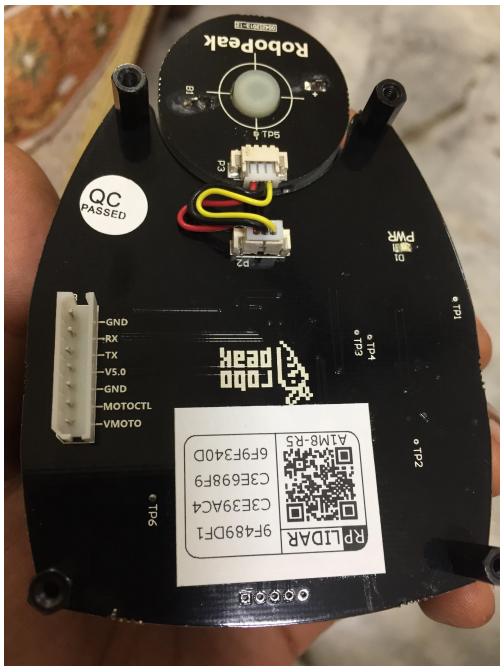
Figure 2-7 RPLIDAR A1 Power Recommended Power Mode

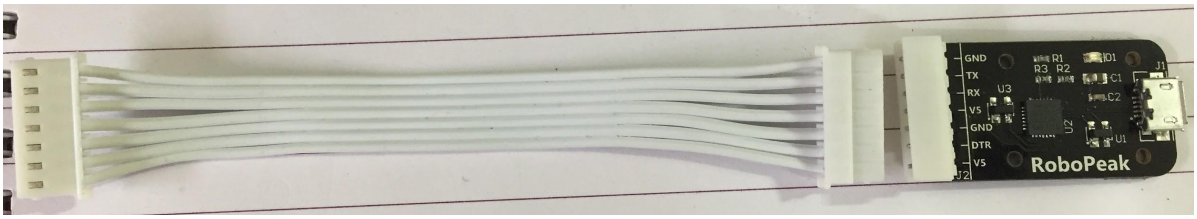
So the motor needs to be powered separately. The A1M8 development kit comes with a chip which converts the 7 pins coming out of the LIDAR to micro USB. The 7 pins are labelled as follows:

Interface	Signal Name	Type	Description	Min	Typical	Max
Motor Interface	VMOTO	Power	Power for RPLIDAR A1 Motor	-	5V	9V
	MOTOCTL	Input	Enable signal for RPLIDAR A1 Motor/PWM Control Signal	0V	-	VMOTO
	GND	Power	GND for RPLIDAR A1 Motor	-	0V	-
Core Interface	VCC_5	Power	Power for RPLIDAR A1 Range Scanner Core	4.9V	5V	6V
	TX	Output	Serial output for Range Scanner Core	0V	-	5V
	RX	Input	Serial input for Range Scanner Core	0V	-	5V
	GND	Power	GND for RPLIDAR A1 Range Scanner Core	-	0V	V5.0

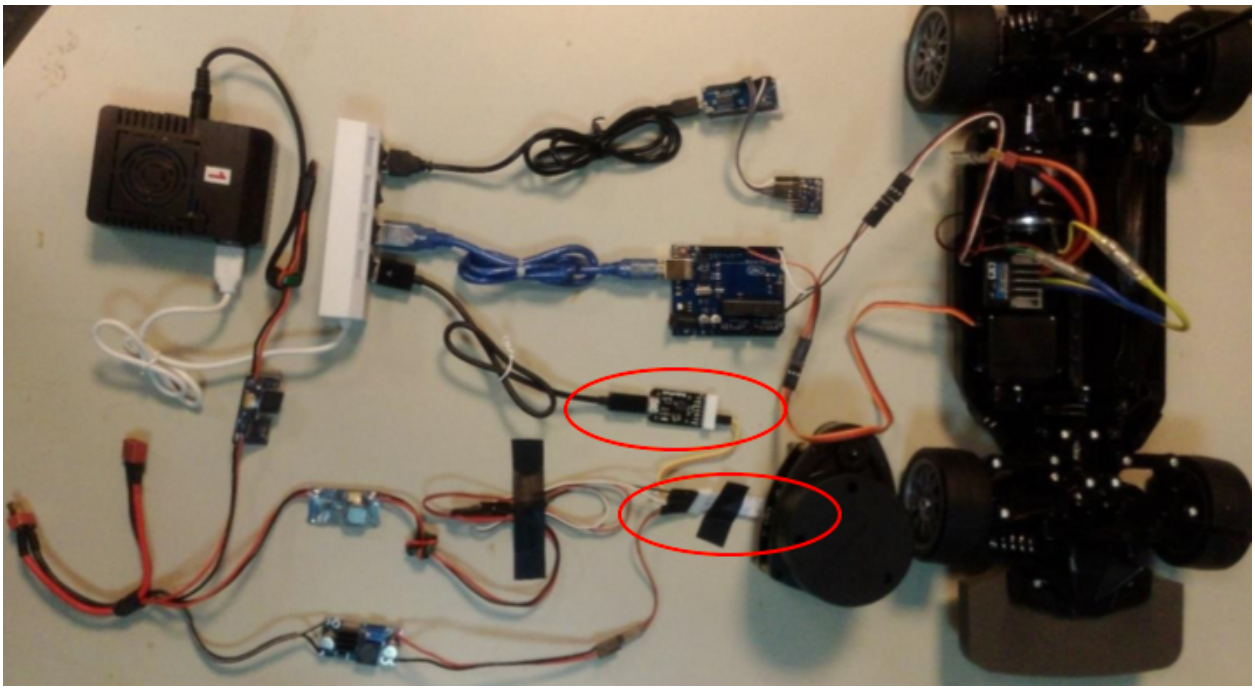
Figure 2-6 RPLIDAR A1 External Interface Specifications

You can directly connect the micro USB cable to the computer and both the motor and the scanner will be powered via the computer's USB port. However for better operation, the datasheet recommends using a different, high current power supply for the motor. How this is done is unclear to me.

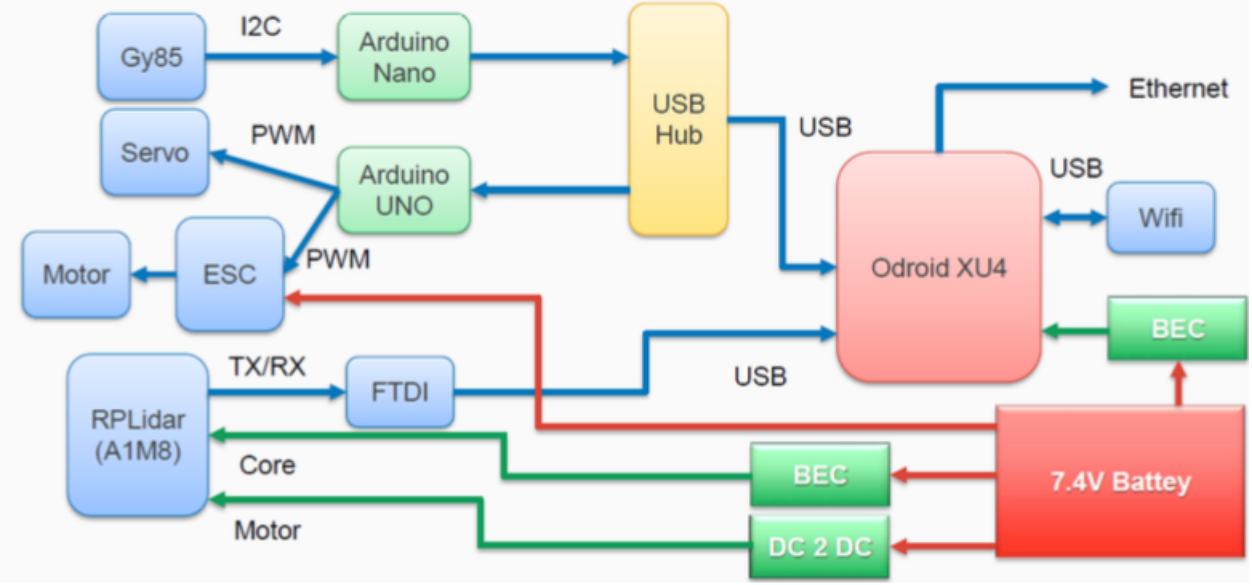




The Hypha ROS tutorial came to my rescue. There is 1 image in the tutorial which made things clearer.



You can see that they use the USB cable for communication with the Odroid, however they do not use all the 7 pins from the small board. Only 2 pins (TX/RX) are used from the board. This image makes it clearer:



However this now leaves us with the question - what about `MOTOCTL`? That is the one which provides the control PWM signal to the motor to control its speed. I tried connecting a 5V supply (basically 2 pins - VMOTO and GND). The motor did not start. So it means that the PWM signal is required. Why doesn't HyphaROS use it? I created an issue on their github repository - RPLidar Connection. Lets see if they reply. Logically speaking `MOTOCTL` should be between `GND` and `VMOTO`. So it cannot come from a circuit which is isolated from `VMOTO` and `GND`. I really hope the HyphaROS team replies back.

And they did reply back. Their solution was to short the `VMOTO` and `MOTOCTL` pins so that it is a 100% duty cycle PWM. I will try this and see how it works. Also, weirdly and according to HyphaROS team, the `VMOTO` voltage controls the max frequency. At 5V the max frequency is 5 Hz. To increase the max frequency one needs to increase `VMOTO`. **TODO**: I need to verify this.

[COMING BACK]: After spending over a month to figure out Gazebo and simulating mapping & localization in it I am back to work on the car. I have installed the platform on the car. I need to implement mapping & localization on the car. For this I need to limit the RPLidar's scanning range so that it can ignore scans behind the robot. So I posted this question Limit range of scanning angle. Lets see if someone replies back.

Razor 9DoF IMU

For any hardware to work with ROS, you need the necessary device driver that enables the communication between ROS on the controller and the device. One of the reasons for choosing the Sparkfun 9DoF IMU is that it already has a driver package http://wiki.ros.org/razor_imu_9dof. The Razor has an ***onboard Arduino*** which runs Attitude Heading Reporting System (AHRS) firmware that works with this ROS driver. You have to load the ROS AHRS firmware onto it ***using the Arduino IDE***. The board as shipped from Sparkfun only contains sensor-value-printing firmware, called the example

firmware in the Sparkfun 9DoF Razor IMU M0 Hookup Guide. The ROS version of the AHRS firmware is in this package, and is derived from the original AHRS Razor firmware by Peter Bartz.

Note that the instructions on ROS Wiki are for an older version (SEN-10736) of the hardware, which has been retired by Sparkfun. This is confirmed by this document Install SparkFun 9DoF Razor IMU M0. The latest instructions are in the github repository https://github.com/KristofRobot/razor_imu_9dof, the repo README says:

> For **SEN-14001 (9DoF Razor IMU M0),** you will need to follow the same instructions as for the default firmware on https://learn.sparkfun.com/tutorials/9dof-razor-imu-m0-hookup-guide and use an updated version of SparkFun_MPU-9250-DMP_Arduino_Library from https://github.com/lebarsfa/SparkFun_MPU-9250-DMP_Arduino_Library (an updated version of the default firmware is also available on https://github.com/lebarsfa/9DOF_Razor_IMU).

The main difference between the ROS Wiki documentation and github documentation is with respect to the board to be used for flashing the firmware to the IMU. The ROS Wiki says:

> **Caution:** choose the correct Razor hardware revision when compiling and uploading the firmware to the board. The setting is located in Razor_AHRS.ino under "HARDWARE OPTIONS"!

So it is important to follow the latest instructions. I have summarized the things to be taken care of in this comment.

Note: If you go through the setup on the hookup guide, you are required to install board definitions - general Arduino SAMD boards and then Sparkfun SAMD boards. When I tried to install the later, the version being shown was 1.4.0. On clicking 'install' I was getting an error "Tool bossac is not available for your operating system.". On selecting an earlier version (1.3.2) the installation happened successfully.

Tested the IMU by echoing messages from the topic `/imu`, after launching the IMU code by executing `roslaunch razor_imu_9dof razor-pub.launch`. Next up is IMU calibration.

Calibration: Follow steps from ROS wiki. You don't have to perform calibration on the Odroid, you can do it on your workstation. The command to find the port number of USB device on Mac is:

```
ioreg -p IOUSB -l -b | grep -E "@|PortNum|USB Serial Number"
```

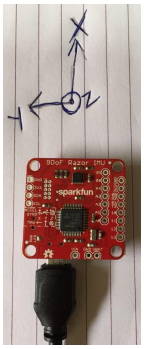
For 3 days I struggled with the coordinate system for values of orientation, acceleration and angular velocities. The ROS wiki: razor_imu_9dof says:

> The Razor_AHRS firmware uses:
>
> - X axis pointing forward (towards the short edge with the connector holes)
> - Y axis pointing to the right
> - Z axis pointing down
>
> Ignore the labelling on the board and use this coordinate frame when mounting the Razor.
>
> This produces a right-handed coordinate system. However, this is different from ROS' right-handed coordinate frame defined in REP-103, which uses:
>
> - X axis pointing forward (towards the short edge with the connector holes)

- Y axis pointing to the left
- Z axis pointing up

The ROS coordinate frame is rotated 180 degrees around the X axis relative to the Razor_AHRS coordinate frame. The razor_imu_9dof node transforms the Razor_AHRS measurements into the ROS coordinate frame. **Caution: it is easy to get confused when viewing data from the Razor_AHRS on a serial monitor and from the ROS /imu topic.** Think hard!

~~I found this to be incorrect!! I found that the Razor AHRS firmware actually uses the ROS coordinate system.~~ See this issue github:KristofRobot/razor_imu_9dof/issues/43.

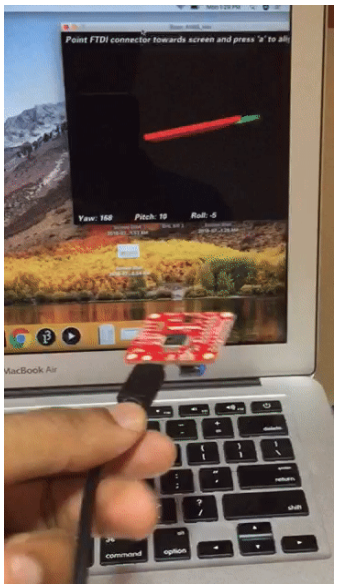


~~I had to make some changes to imu_node.py (commit1, commit2 and commit3) to rectify the directions of coordinate axes.~~ These changes were not required. It turns out that IMUs measure the acceleration opposite of gravitational acceleration in their static state. See these posts:

1. IMU convention for robot_localization
2. Why do 3-axis accelerometers seemingly have a left-handed coordinate system?
3. IMU data to be used with robot_localization

So the razor_imu_9dof package provides data in the right format. ***No changes are required to the code.*** 1 more day wasted in figuring this out :(

Also I discovered that my magnetometer is faulty :(**It is not tracking rotations around the Z axis very well.**



Or it could also be due to this [Troubleshooting: Unstable yaw (aka heading aka azimuth) readings / Yaw drift](). It says:

The most common problem when using the tracker (like with all these kinds of trackers that are "self-contained" and don't have a fixed "base-station") is drift in the horizontal plane. Originally drift means that the yaw angle is constantly drifting away in one direction. This is caused by the gyroscope which is never actually outputting zeros, even when the board is lying still. This is called sensor noise and mis-calibration and even though you (hopefully followed the tutorial and) calibrated the gyro, it will always be a little off. It's a cheap gyro after all. Now what you most likely experience as drift or false yaw readings is actually caused by a flawed drift-correction in the firmware itself. The firmware constantly uses the magnetometer and accelerometer data to calculate a magnetic north. This magnetic north is then used to correct gyroscope drift. So if you do a fast rotation and then see the yaw slowly drifting back (or forth) and "locking in" on a certain angle, then this is caused by a flawed drift-correction due to false magnetometer data. If your drift is constant without "locking in" then this is gyroscope-drift, which is very unlikely to happen if you calibrated the sensors. Here's a little summary of things that you can do and check to improve magnetometer-related drift:

- Check that you really selected the right hardware in `Razor_AHRS.ino` (see *Setting up the software*)!
- Try setting `DEBUG__NO_DRIFT_CORRECTION` in `Razor_AHRS.ino` to true temporarily. The magnetometer will not be used. Yaw should be more stable, but you'll have the constant gyroscope drift mentioned earlier, which makes it unusable in the long run. But at least you know it's a magnetometer issue now.

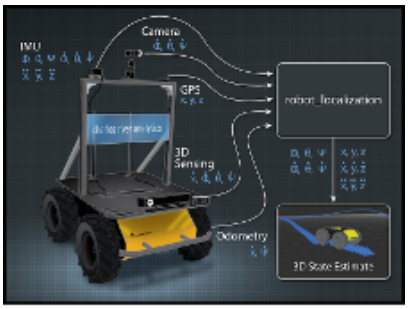
- Calibrate the sensors, and especially the magnetometer (see *Sensor calibration*). Use the *Extended magnetometer calibration* in the exact setup and environment that you plan on using the Razor in. Any objects that actively or passively influence the magnetic field should be present during calibration and keep their position relative to the Razor after calibrating. They can move with the Razor - as long as they keep their relative position everything is fine and even strong distortion can be "calibrated away". Magnetic fields that change over time on the other hand - like the ones generated by nearby motors that are turned on and off - can not be handled with the current static calibration.
- If all this didn't help, it's also possible that one or some of your sensors are broken. You should see this when calibrating them: they are either stuck on one value or the values they output are very far off the ones given in the default calibration in `Razor_AHRS.ino`.

It turned out that the error in yaw was due to non-calibrated sensors. After calibration it was working fine.

For localization, in ROS, there are some packages like:

1. robot_pose_ekf
2. robot_localization
3. amcl

AMCL stands for `Adaptive Monte Carlo Localization`. The package `amcl` uses laser based data and map and performs particle filter localization. The other 2 packages are general state estimation packages, using Kalman Filters, which can take input from a variety of sensors like IMU, wheel encoder, camera (visual odometry), GPS.



Source: docs: robot_localization

Seems like `robot_localization` is the package which should work for us. The package also seems to be more actively maintained.

`Installation`: [SOLVED] How can i install robot_localization?

The repo robot_localization has various branches. Most activity seems to be happening on `kinetic-devel`, latest release is for the branch `melodic-devel`. Since I am on melodic, I used the melodic-devel branch.

```
git clone -b melodic-devel https://github.com/cra-ros-pkg/robot_localization.git
```

Then follow this advice:

Source: [Could not find a package configuration file provided by "geographic_msgs"](). So I executed:

```
rosdep install --from-paths src --ignore-src
```

The package robot_localization has 1 dependency geometric-msgs. On executing the command I got the following log:

```
executing command [sudo -H apt-get install ros-melodic-geographic-msgs]
```

After that run `catkin_make` and hopefully your workspace should be compiled. *Next*: run `robot_localization` by taking data from the IMU.

To get started with this package this video [2015 ROSCon talk]() and this PDF guide [A Generalized Extended Kalman Filter Implementation for the Robot Operating System]() were helpful. A very interesting para from this PDF guide:

`It is worth noting that despite the fact that the odometry only truly measures x and yaw velocity,` **`we can infer more information.`** `The platform is not going to obtain any instantaneous z or y velocity due to platform constraints, i.e., it cannot fly and is nonholonomic. We can therefore` **`fuse the zero values`** `in those data fields with our estimate, providing that the measurement's covariances are set appropriately (Table I).` *`In general, if the measurement of a quantity is implied through kinematic constraints, it is best to treat that quantity as a measured value`*`.`

`Sidebar (Blog Post)`: [How I built ROS odometry for Ackermann vehicle without encoder]()

Things to watch out for when working with IMUs and `robot_localization`:

1. Signs for axes used in the IMU, driver and other packages that you are using to process the IMU data
2. Frames of reference `base_link, odom, map` etc.
3. Transforms and [transform publishers]()

Working with an IMU is definitely a good topic to blog about. Too many unanswered questions for a beginner.

`Sidebar (Videos)`: [ROS Tutorials] Introduction to ROS Navigation. Seems like a good start on navigation in ROS.

First, start with the official tutorials on Navigation: http://wiki.ros.org/navigation/Tutorials. To get started with the `robot_localization` package, this video was **God sent** - [Merging Odometry and IMU data]()

for robot localization! It tells you how to specify the configuration of the `robot_localization` package.

You can read through these posts on ROS Answers:

1. How does robot_localization package work and what should the output be?
2. Can I use robot_localization package with only imu data? If I could,how to config the package?
3. Using ekf_localization_node from robot_localization package for IMU only navigation

To use robot_localization, follow these steps:

1. Create a new package
2. Add a launch file
3. Set parameters

Watch the above video to get understand the launch file parameters. From the docs keep this in mind:

`The state estimation nodes of robot_localization produce a state estimate whose pose is given in the map or odom frame and whose velocity is given in the base_link frame.`

Read through the instructions in Preparing Your Data for Use with robot_localization. This page has this to say on transforms:

`Broadcast of the` **`odom->*base_link*`** `transform. When the world_frame parameter is set to the value of the odom_frame parameter in the configuration file,` **`robot_localization's state estimation nodes output both a position estimate in a nav_msgs/Odometry message and a transform from the frame specified by its odom_frame parameter to its base_link_frame parameter.`** `However, some robot drivers also broadcast this transform along with their odometry message. If users want robot_localization to be responsible for this transform, then they need to disable the broadcast of that transform by their robot's driver. This is often exposed as a parameter.`

For only IMU based localization we are going to assume that we have the following coordinate frames:

1. `base_imu_link` (You can get this from the `frame_id` value of the IMU message)
2. `base_link`
3. `odom`
4. ~~`map`~~ (not required since we are in an indoor environment)
5. `earth`

To start the localization we need to create a launch file with the following information:

1. Transform from `base_link` to `base_imu_link`
2. Coordinate frames
3. Sensor configuration for IMU (see this Configuring robot_localization)
4. ~~Measurement covariance matrix~~ Not required since this is published by the sensor itself as a part of the sensor_msgs/imu message
5. Initial state covariance matrix (setting covariance and params for robot_localization)
6. Process noise covariance matrix

Although the IMU provides `roll, pitch and yaw`, we are only going to fuse yaw information by taking advantage of robot_localization's `2d_mode`. This will fuse zero values for pitch & roll. Similarly for

linear accelerations we will only fuse acceleration in `X`, since our robot cannot move laterally. And for velocities we will only fuse the value of `vel_yaw`

```
Last, we come to the IMU. You may notice that we have set the Y¨ to false.
This is due to the fact that many systems, including the hypothetical one
we are discussing here, will not undergo instantaneous Y acceleration.
However, the IMU will likely report non-zero, noisy values for Y
acceleration, which can cause your estimate to drift rapidly.
```

Next launch the launch file and test the output.

What is the output of robot_localization package? Answer it is a message of type nav_msgs/Odometry on the topic `odometry/filtered` See the Published Topics section of the docs.

I launched the launched file and I could echo messages from the topic `odometry/filtered`. Everything worked fine except rviz. For some reason `rviz` would not work with VNC. Trying to debug this issue. Apparently running the `rviz` on the local machine is the only way to go, see this README on getting RVIZ to work over multiple computers. So I can try running ROS on my local Ubuntu VM and on Odroid. This is achieved in the LIDAR setup section.

Platform

After the individual sensor setups are done, I needed to mount all the sensors and electronics on a platform. For the platform I used an exam pad

The advantages of using this pad are:

1. It is cheaper and easily sourceable than acrylic sheet
2. Flexible, so you can easily drill holes with a manual hand drill machine
3. Strong enough to hold all the sensors and electronics while I am experimenting

The disadvantage of this pad is that it can bend and deform if you put a lot of weight on it. This pad will serve as a good first version. Once I am done experimenting and have arrived at a final layout, I can use a stronger acrylic sheet.

Next I needed screws and standoffs which could be used to mount electronics. I could not find the dimensions of the screws, to be used to mount the lidar, in the datasheet or online. I decided to measure things manually myself.

For the RPLidar I measured the inner diameter of the female thread, it was about 2 mm, slightly more. I could not be sure as I did not have a very accurate measuring device. It is tricky to measure at such a small scale. Length of the shaft is 10mm. So 4 nos. M2x8 or M2.5x8 screws for the lidar would be

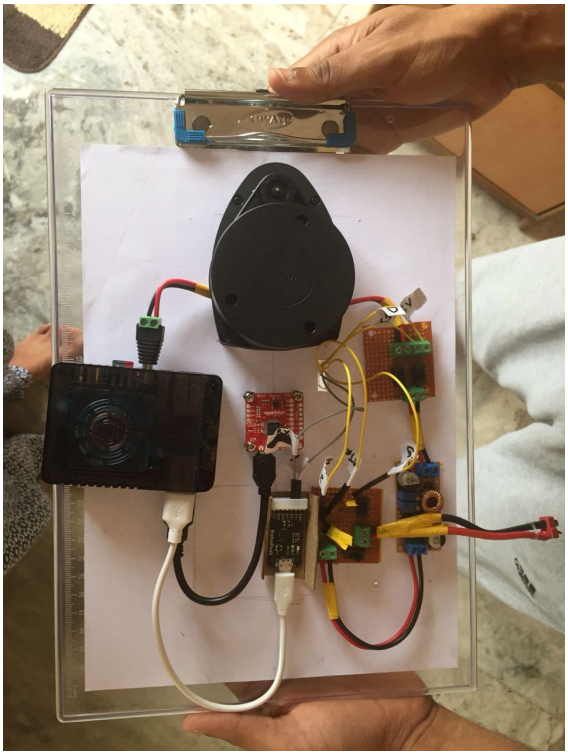
sufficient. I could only know after trying both on. For the electronics I need screws of diameter 2.5 mm or 3mm, length about 5mm. So M2.5x5 or M3x5 will do.

Ordering M2, M2.5 and M3 screws online is incredibly hard. There is no single website which stocks all these screws. I found these screws at:

1. https://www.mgsuperlabs.co.in/
2. https://www.diy-india.com/
3. http://robu.in/

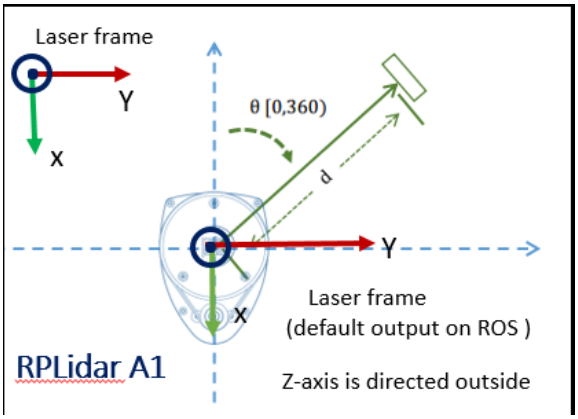
However their delivery charges meant I would be spending about 200 bucks just for delivery and I would have to wait 4-5 days for the delivery. I did not have so much time to spare. So I decided to head out to Lamington Road (officially Dr. Dadasaheb Bhadkamkar Marg) in Mumbai. It is a small street near Grant Road railway station, where one can find all kinds of shops supplying electrical and electronics components and tools. They sell things at a whole sale rate, and almost every IC (integrated circuit) and electronics components (like PCBs, switches, soldering tools,motors, cameras, sensors etc) required for robotics are available. However travelling there is a cumbersome process (1.5 hours by local train one way). So one would need to spend at least a half working day to visit the market.

I found a shop named American Nuts and Bolts. It had M2, M2.5 and M3 screws and bolts. It also had M3 hex standoffs. However their length was limited to 11 mm. Longer standoffs were not available. I bought 25 M2 and M3 screw-bolt pairs and 25 M3 hex standoffs. Spent the next 2-3 days drilling holes in the exam pad, aligning them with mount holes on the lidar, making a rudimentary power distribution board with PCB mount screw terminals and slide switches. This was the result:



The lidar connection was tricky. The RX & TX of the lidar are connected to the circuit board supplied by the manufacturer which converts serial data to USB. The voltage connections for the motor and scanning module are connected to 7.4V and 5V respectively. There is a voltage regulator (or BEC) which converts the 7.4V of the LiPo battery to 5V. **TODO:** Next I need to test if everything is running fine. Yes it is.

Today (5th September) I realized that I have put the LIDAR in the reverse position (i.e. its front is pointing towards the back of the car!). I will have to take care of the transforms so that the system is taking in the right data. The image on the RPLidar repo is confusing:



Source: rplidar_ros github repository

However the above image suggests that my laser is oriented in the right direction.

Today (16th October) I realized that the LIDAR is in the right orientation. I spent *2 full days* debugging this. The map generated by Hector Mapping seemed to have inverted axes (i.e. directions reversed). I don't know what the issue was earlier. The above image is correct. I have posted an issue on the RPLidar Github repository - What is the correct orientation of RPLidar?.

Gazebo

Before trying out localization using the actual IMU and Lidar, I decided to simulate the setup in Gazebo. Simulation allows you to perform experiments quickly. I would be using ros libraries like `robot_localization`, `razor_9dof_imu`, `gmapping/hector_slam`, `amcl`. Each library comes with its own set of configuration parameters which may require to be tweaked.

The objective of this simulation would be to:

1. Create a robot which has LIDAR and IMU mounted on it.
2. Create an indoor building layout
3. Control the robot using the keyboard inside the layout
4. Create a map from LIDAR data using a package like `gmapping`
5. Run `amcl` in the map using LIDAR data
6. Localize the robot using IMU data, `amcl` and a package like `robot_localization`
7. Test path planning algorithms on the bot before trying them out on an actual robot

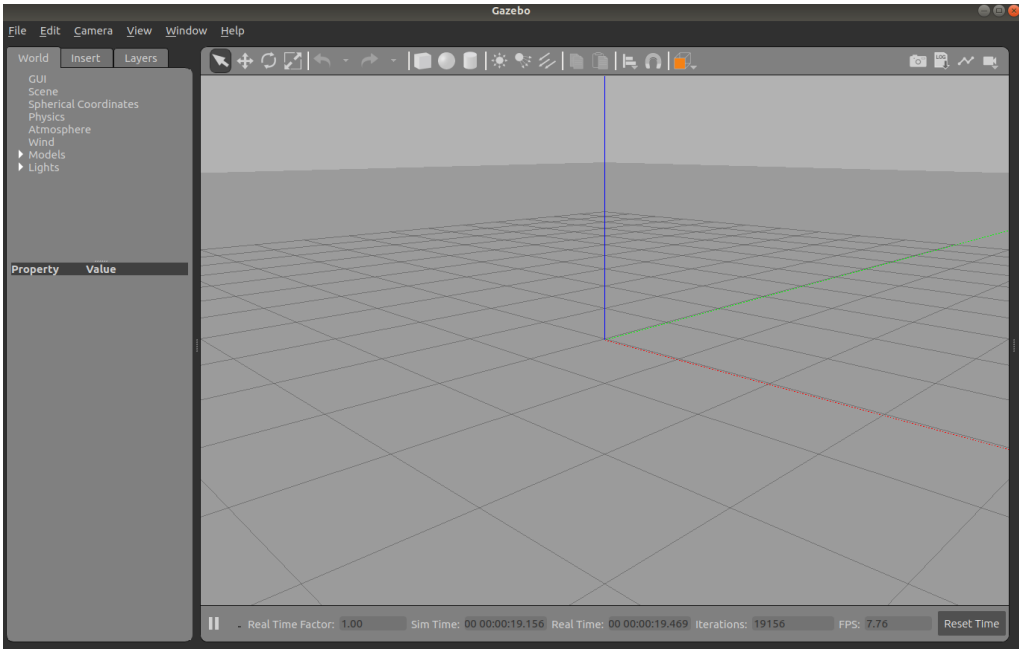
Getting started with Gazebo is a bit difficult for new comers. I banged my head around for 4-5 days to get my simulation world setup. I hope reading this document will help you save this precious time.

First things first - go through the ‘guided’ set of tutorials on the tutorials page. You don’t have to understand everything. Just give it one read.

I decided to use the MIT Racecar Simulator. The repository is not maintained and it lacks documentation. I stumbled on to a website of Introduction to Intelligent Systems offered at the Seoul National University. This course uses the MIT Racecar Simulator. The end project schedule on the website gave me a rough guideline to follow. I went through the project/assignment instructions and files, available here. The first two assignments deal with getting started with Gazebo, loading the model of the MIT racecar and drive it around one of the simulator tracks. After going through these files I got some idea about how Gazebo simulation works.

For a simulation to run in Gazebo, one needs two basic things:

#1: World - It contains all the objects that you want to simulate. The most basic world, `empty_world`, has a ground plane and a light source (sun). It looks like this:



To simulate a real environment like an office space, building, gas station etc, you can add `models` to this empty world. Click on the insert tab. Gazebo sim Model database has readymade models which you can add to your world. One can also create custom models using the SDF (Simulator Description Format) files. Gazebo comes with a building editor where one can create custom indoor layouts. See this tutorial Building Editor.

#2: Your Robot - You need to create a model of your robot and add it to your world. There are 2 different ways to create a robot for Gazebo. The first one uses Universal Robot Description Format (URDF), which is an older and more cumbersome way of doing things. The second one used SDF (Simulator Description Format), which is a much cleaner way create models. However the catch here is that SDF format is unfriendly to a beginner and lacks easy to understand documentation, although there are examples in Gazebo tutorials. The ros website has documentation and examples for using the URDF

format. You will also find few (and rare) open source tutorials, videos and blog posts which use the URDF format.

So choose wisely.

Creating the robot model

A robot consists of the following:

1. **Links**: Each removable part of a robot can be considered a link. Eg. each of the wheels, chassis and sensors can be considered as separate links.
2. **Joints**: Joints connect links to each other. Eg. the wheel and chassis is connected by a joint. A joint can be of different types. Eg. fixed joint implies that the 2 links are rigidly fixed to each other and move together. A camera and the chassis are connected by a fixed. On the other hand a continuous (or revolute) joint connects a wheel to the chassis.

The official documentation has examples however they it lacks documentation about parameters used to create examples. After banging my head around for 2-3 days I finally realized that the only way out of this was to learn from tutorials and code examples.

To my disappointment, the MIT Racecar Simulator turned out to be too demanding on my VM (I am running a Ubuntu 18.04 VM on Mac host). The simulation took ages to start. It ran very very slowly. MIT Racecar uses a Ackermann steering 4WD model of the car in Gazebo. And I think the components to steer the car cause a lot of overload on the resources. I tried changing the VM settings, increasing CPUs and memory. It did not make any difference. After spending about 1 day on this I decided to build my own simulation model, simple & light on resources.

To keep things simple I decided to start with a 2 wheeled (plus castor) robot with a differential drive system. I had assumed that I would readily get open source simulation models with such a car, laser and IMU. I was wrong. As far ROS tutorials are concerned its a wild west out there. A lot of the tutorials and code examples are old (many 4-5 years old) and use older versions of ROS and Gazebo API. Fortunately for me I came across a tutorial which created a 2 wheeled differential drive robot with a camera on it - Robotic simulation scenarios with Gazebo and ROS. This was a very good starting point.

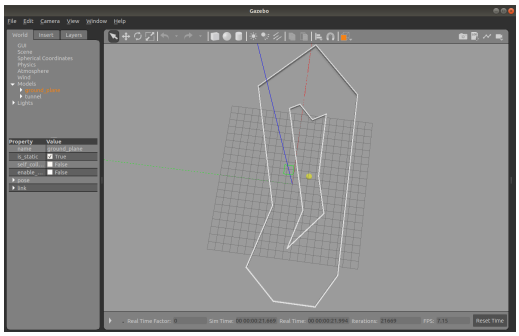
Once the robot was created I needed to add a LIDAR on top of it. By now I had understood these points:

1. The visual model of the laser had nothing to do with the sensor behaviour. In ROS the appearance is only for visualization in Gazebo. Nothing more.
2. Actual sensor behaviour is controlled by Gazebo Plugins. Gazebo plugins give your URDF models greater functionality and can tie in ROS messages and service calls for sensor output and motor input. Source: Tutorial: Using Gazebo plugins with ROS

So I created a link with the shape of a box. In the future I could use an actual 3D mesh of the RPLidar to make the simulation look visually more accurate. This post helped me add a plugin for the laser Tutorial: Using Gazebo plugins with ROS.

Indoor Building Layout

Next I needed a model of an indoor building layout. My first approach was to look for readymade open source models. After spending some time online I figured out that it is quite easy to build the model using Gazebo's Building Editor. See this [Tutorial] Building Editor. This is how my layout looks from the top



Not too complex. I just want to test out my localization algorithms, so I went with a simple layout.

Controlling the robot using the keyboard

The tutorial, which I had referred to for creating the robot model, also added plugins to the robot's wheel for control via ROS messages. The controllers listened on the topic `/car/cmd_vel`. I could use `turtle_teleop_key` node from the inbuilt `turtlesim` package to control the robot. This would work since both my robot and the turtles in the `turtlesim` package are 2 wheeled differential drive robots. I just had to remap the topic on which the messages are published. This went into the launch file:

```
<node name="key_op" pkg="turtlesim" type="turtle_teleop_key">
       <remap from="/turtle1/cmd_vel" to="/car/cmd_vel"/>
</node>
```

Initially the control appeared to be haphazard. The robot would quickly go out of control. On examining the controller I realized that they employed PID controllers with a proportional gain of 100 and differential gain of 15. Exactly how these PID controller worked was not clear to me. After toying around with the PID parameters for some time I found a configuration which worked well.

Mapping: Part 1

This tutorial Turtlepi #1: RViz + Gazebo-Turtlebot localization in simulation, was helpful in giving me a very high level overview of the task at hand (creating a robot with sensors, controlling it with a keyboard, creating a map ad localization). Reading through posts online I figured out that the most common libraries used for mapping were `gmapping` and `hector_mapping`. The tutorial used `gmapping` and so I too decided to use `gmapping`.

Download and install openslam_gmapping, this answer that I wrote on ROS Answers helped in setting up `gmapping`. Launching gmapping was straightforward. It listens on a topic `/scan` with message type sensor_msgs/LaserScan. Your laser should publish data on this topic.

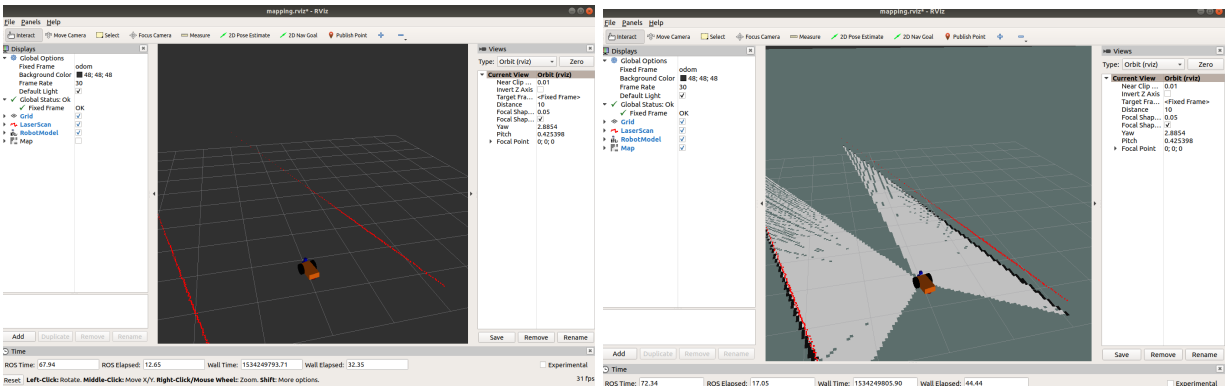
Rviz

Rviz is a powerful visualization tool which comes bundled with ROS. It helps you visualize the sensor data like camera images, laser scans, point clouds, the map, robot and its pose. The following resources were helpful in setting up the Rviz visualization:

1. Lab 11: Visualizations in ROS with RViz

2. [Tutorial] Autonomous Navigation with the ROS Navigation Stack (part 3)

After setting things up my visualization looked like this (**left**: robot & laser scan, **right**: also shows map data):



Mapping: Part 2

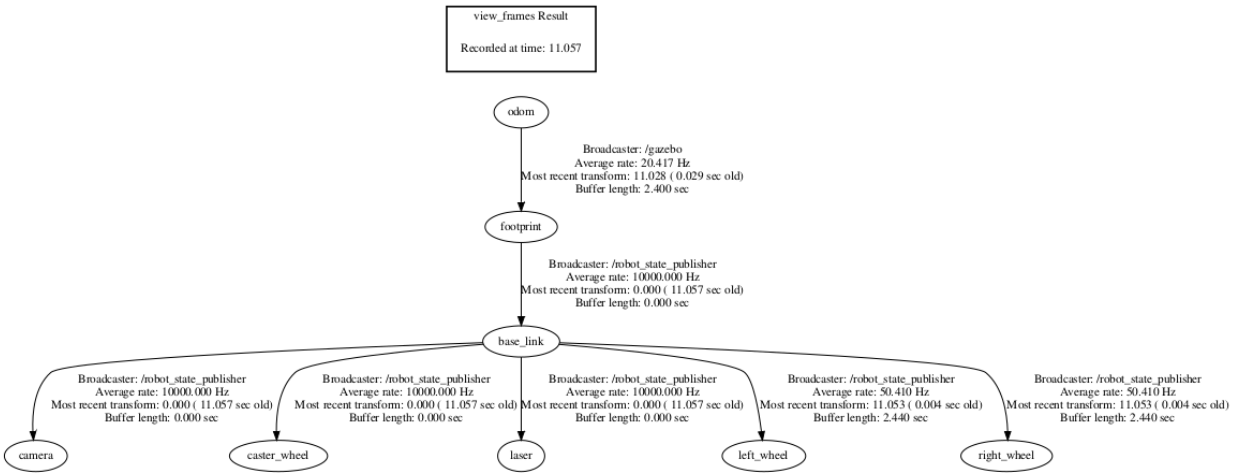
While trying to setup the rviz visualization I realized that my transforms were not properly setup. You can view transforms between different frames by running this command:

```
rosrun tf view_frames
```

View subscribers & publishers of topics:

```
rosrun rqt_graph rqt_graph
```

After properly setting up the `link` names and `frame_ids` transform tree looked like this:



I had to change the name of the link `chassis` to `base_link`. I have no idea how the transform from `odom` to `footprint` is published. **I have now realized that the `frame_id` in sensor messages is actually the name of the link which is responsible for sending this data.** ~~The transform from base_link to laser has to published manually. This code does it:~~

```
<node pkg="tf2_ros" type="static_transform_publisher"
    name="bl_laser" args="0 0 0 0 0 0 1 base_link laser"/>
```

This transform is not required since the `robot_state_publisher` takes care of it.

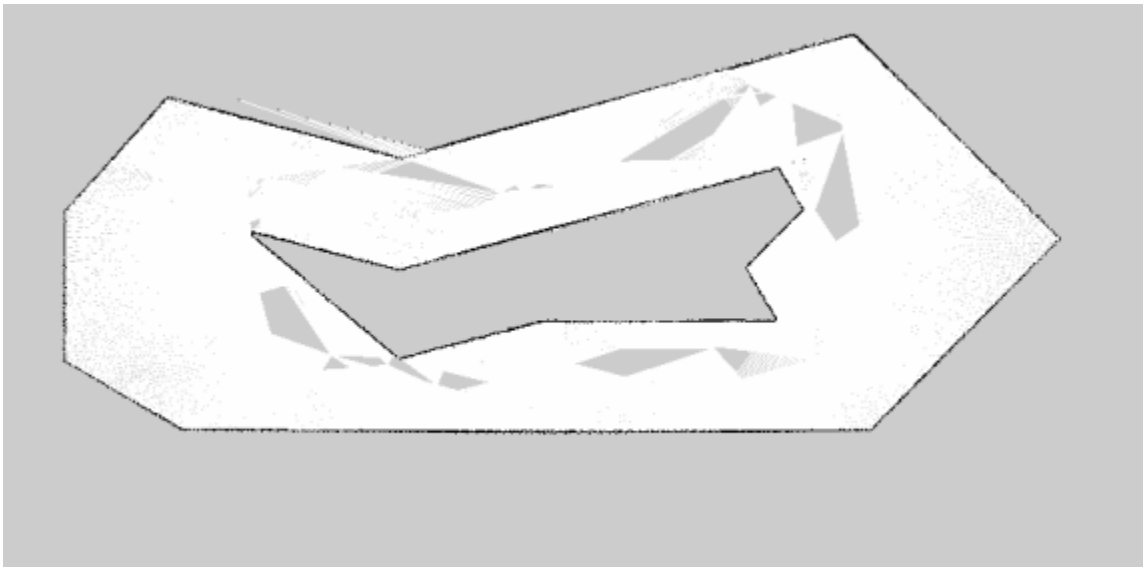
Now I could launch the `gmapping` node, which publishes data on the `/map` topic:

```
<node pkg="gmapping" name="slam_gmapping" type="slam_gmapping"
      output="screen">
      <param name="scan" value="/scan"/>
</node>
```

You should now drive the robot around the map by controlling it via the keyboard. After you have traversed the entire map you can save the map by running this:

```
rosrun map_server map_saver -f mymap
```

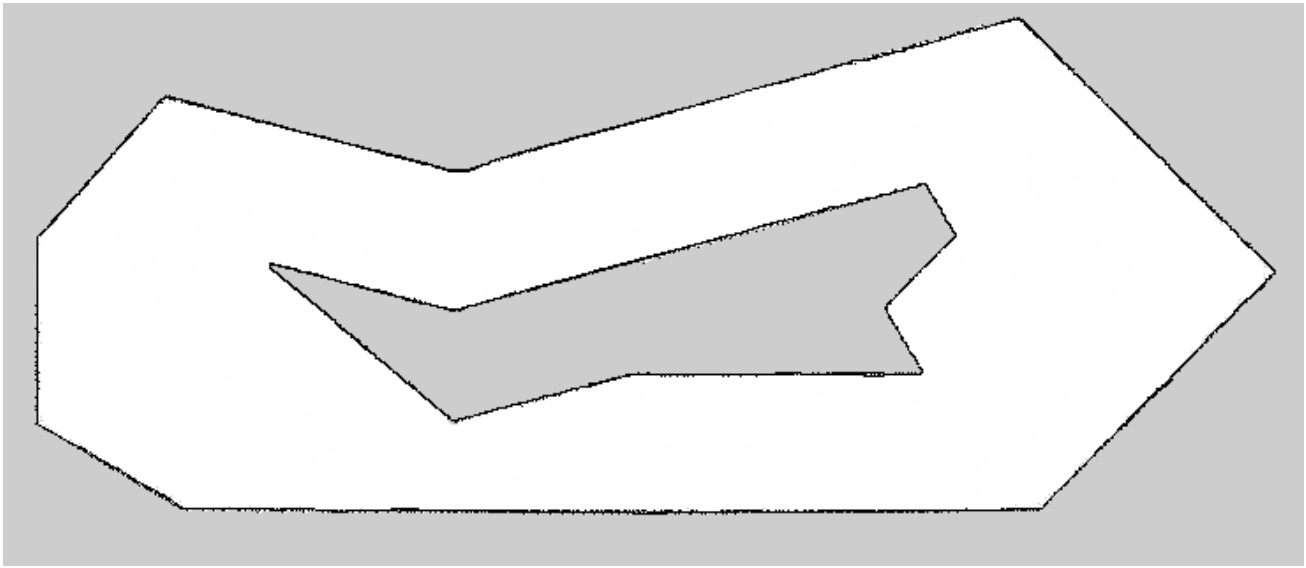
This will save 2 files - `mymap.pgm` and `mymap.yaml`.This is how the map looked like:



Not perfect but good enough. When you compare it to the indoor layout, described before, you can see that the map has some blind spots. I hope I can edit the map and remove these blind spots. **EDIT**: You can edit the map with software like GIMP. The map is an `OccupancyGrid`, with value 0 indicating no obstacle and a value 1 indicating obstacle.

Edited Map

This is how the edited map looks like



ROS Navigation Stack

The `map_server` package, which is required to save the map generated by `gmapping`, is a part of the ROS navigation stack. As of this writing the melodic release of ROS Navigation Stack was not available. So I had to install from source i.e. `git clone` the repo in the workspace's `src` directory.

Localization using AMCL: Part 1

AMCL stands for Adaptive Monte Carlo Localization or popularly known as the particle filter localization. The ROS Navigation Stack comes with `amcl` bundled with it.

The first step to use `amcl` is to load the created map and serve it via a ROS topic or service. The navigation stack has a package `map_server` which helps with this. Add this to your launch file:

```
<!-- Map server -->
<arg name="map_file" default="$(find mapping)/maps/mymap.yaml"/>
<node name="map_server" pkg="map_server" type="map_server" args="$(arg
map_file)"/>
```

Next launch the `amcl` node. We need to inform AMCL about the initial pose of the robot so that it can initialize its particles for the particle filter. We need to publish a message on the topic `/initialpose`.

```
subodh@racecar$ rostopic pub /initialpose
    geometry_msgs/PoseWithCovarianceStamped "header:
      stamp:
      secs: 0
      nsecs: 0
      frame_id: 'map'
    pose:
      position: {x: -3.5, y: 1.0, z: 0.0} // we know the initial position
      orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0} // and orientation
      covariance: [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0,0.0, 0.0, 0.0, 0.0, 0.0, 1.0]"
```

An alternative to this would be to initialize the particle filter at the default location (0,0,0) and use large values for the variances.

```
<node pkg="amcl" type="amcl" name="amcl" output="screen">
    <param name="laser_min_range" value="0.1"/>
    <param name="laser_max_range" value="10.0"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="odom_model_type" value="diff-corrected"/>
    <param name="base_frame_id" value="footprint"/>
    <param name="update_min_d" value="0.5"/>
    <param name="update_min_a" value="0.3"/>
</node>
```

Read through the documentation of `amcl` to understand these parameters.

Why `amcl` needs odometry?

`amcl` apparently uses odometry data. I think it fuses the output of particle filters and odometry to get a more robust localization output. This post - ROS AMCL does not need odometry data? - suggests so. Also this page ROS Mapping and Localization, says `amcl` requires odometry message. This post - Robot localization with AMCL and EKF - nails it. So basically a package like `robot_localization` is used to fuse together inputs from wheel odometer, IMU etc to provide a better odometry estimate for `amcl`.

How can I forget this? Particle filters have 2 steps - predict and update. In the prediction step the particles are propagated using the latest control inputs (you need to have a motion model of the car to propagate the particles). This prediction output is something which is supplied by the odometry message. So of course odometry is required!!

I will try generating odometry using IMU data with `robot_localization`.

Changing initial Gazebo view angle

This is addressed by this post - gazebo starting view pose control. You need to add a `gui` element to the `world`

```
<world name="office">
  <gui>
    <camera name="user_camera">
      <pose>0 0 0 0 0 0</pose>
    </camera>
  </gui>
  ...
</world>
```

This will change the starting view angle of Gazebo. I used to spend 2-3 mins after each launch to tediously set the view angle. The current view angle can be found in the `World` tab under `camera -> pose`.

amcl transforms

I got confused with the map to odom transform generated by amcl. Why is this published? Both are world fixed frames according to REP-105. In my opinion this transform should be static. I posted a question here: Confused with amcl map to odom transform. Awaiting answers.

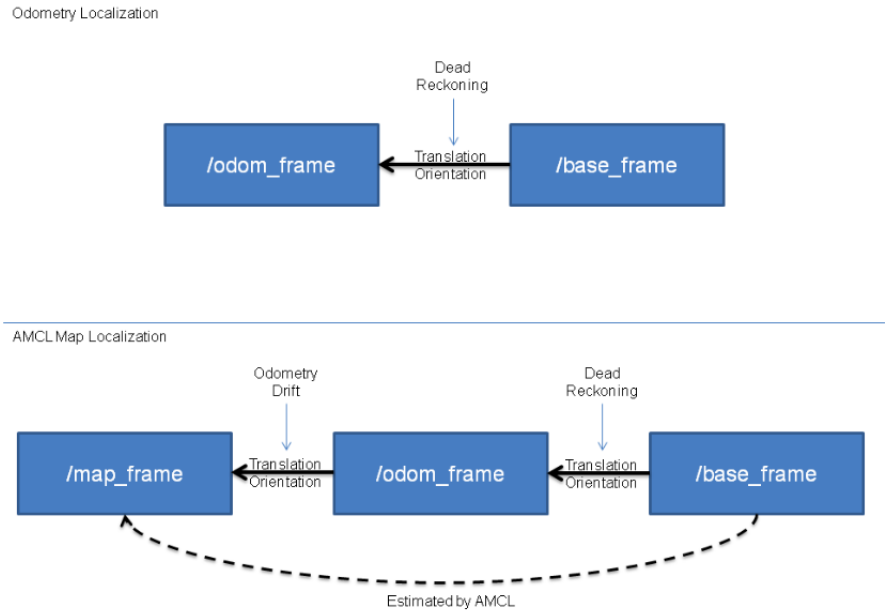
I answered it myself. Here it goes:

I think now I understand why amcl publishes the map to odom transform. This is explained in the Transforms section of the amcl documentation. Reproducing it here with the relevant emphasis:

`amcl transforms incoming laser scans to the odometry frame (~odom_frame_id). So there must exist a path through the tf tree from the frame in which the laser scans are published to the odometry frame.`

An implementation detail: *on receipt of the **first laser scan**, amcl looks up the transform between the laser's frame and the base frame (~base_frame_id), and latches it **forever**.* So amcl cannot handle a laser that moves with respect to the base.

The drawing below shows the difference between localization using odometry and amcl. During operation amcl estimates the transformation of the base frame (~base_frame_id) in respect to the global frame (~global_frame_id) but it only publishes the transform between the global frame and the odometry frame (~odom_frame_id). **Essentially, this transform accounts for the drift that occurs using Dead Reckoning.**

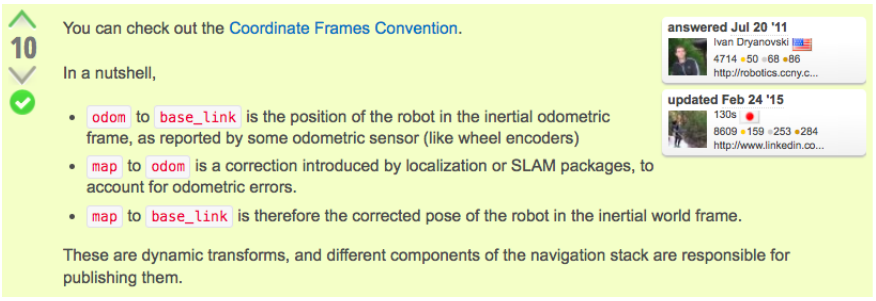


Additionally these posts also helped me cement this understanding:

1. How to broadcast a transform between /map and /odom
2. transform base_link to base_laser,map,odom

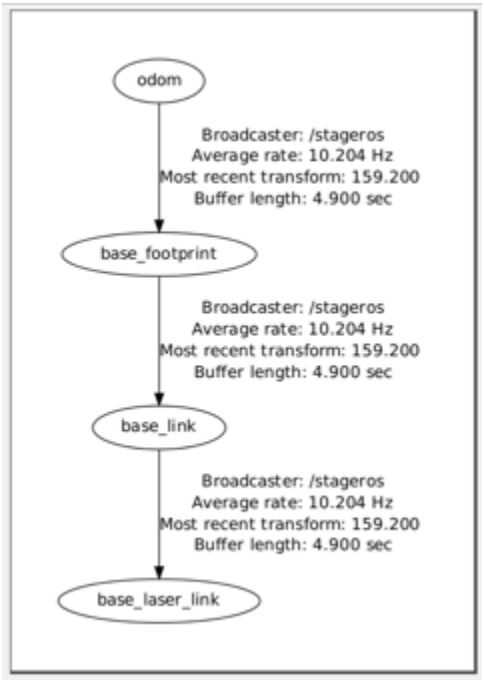
So to summarize:

1. Both odom and map are world fixed frames
2. However the pose in odom frame will drift due to dead reckoning
3. To fix this you publish a map to odom transform, which essentially fixes the pose of the robot in the map frame



.

Here is an example tf tree:



1. **odom** – a world-fixed frame, its origin is positioned at the robot's starting position
2. **base_footprint** – base of the robot at zero height
3. **base_link** – rotational center of the robot
4. **base_laser_link** – base of the laser

To add to this confusion:

- The map coordinate frame is a global frame fixed to the map
- The pose of a robot relative to the map frame should not significantly drift over time
- **The map->odom transform is typically published by a localization component such as amcl or gmapping**
- The localization component constantly re-computes the robot pose in the map frame based on sensor observations

Using the IMU to generate the odometry for amcl

The robot_localization package generates a transform from odom frame to base_link frame. Any other driver or package should not publish this transform. Gazebo publishes this transform. To prevent this I did the following change:

```
<!-- push robot_description to factory and spawn robot in gazebo -->
<node name="car_spawn" pkg="gazebo_ros" type="spawn_model"
      output="screen"
      args="-urdf -param robot_description -model car -x -3.5 -y 1.0">
   <remap from="tf" to="gazebo_tf"/>
</node>
```

This is just a workaround to the problem. **Actually this did not work. The following worked:**

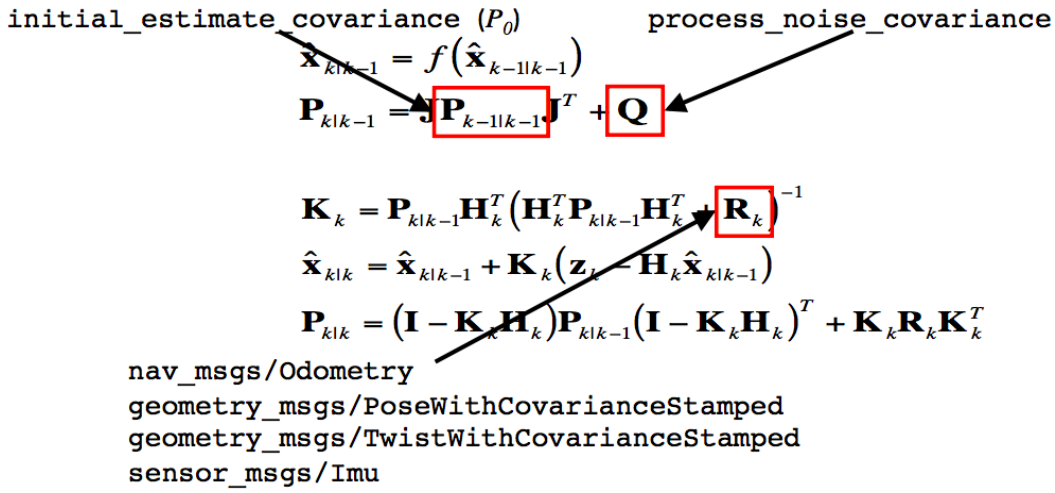
```
<gazebo>
   <plugin name="differential_drive_controller"
           filename="libgazebo_ros_diff_drive.so">
        ...
        <publishOdomTF>false</publishOdomTF>
        ...
    </plugin>
</gazebo>
```

`robot_localization` outputs a `sensor_msgs/Odometry` message on the topic `/odometry/filtered`. You can visualize the odometry output in rviz. When I did this I observed that the odometry marker did not move even though the car moved by a non-trivial distance. Although I did not expect the odometry derived from IMU to be very accurate, I expected it to at least move as the robot moved. Until this works correctly, `amcl` won't work well. I will try and visualize the Odometry published by diff drive controller. I tried this and it is working fine. So there must be a problem with how `robot_localization` is setup.

These words from the robot_localization documentation, related to the process covariance matrix, came to my rescue:

```
# However, if users find that a given variable is slow to converge, one
approach is to increase the
# process_noise_covariance diagonal value for the variable in question,
which will cause the filter's predicted error
# to be larger, which will cause the filter to trust the incoming
measurement more during correction.
```

I increased the covariance values and now the odometry marker was moving. **Why? I realized that the IMU sensor noise was set to zero.** After I added a large sensor noise (stddev of 0.5), then amcl pose marker would move even without the robot moving. The large process covariances were causing the EKF to trust sensor more than the output of process model prediction. Hence the large sensor noise was causing the marker to move. I reduced the sensor noise to stddev 0.05 and also reduced the process covariances to 0.1 (had set them to 2, 3 etc before). The following image was helpful:



So this tells you that the process noise covariance matrix constantly adds to the error after each predict iteration.

The robot_localization estimate kept on drifting even when the robot was stationary. I tried adjusting the robot_localization configuration parameters, to no avail. Finally I posted this question: robot_localization estimate (using only IMU) drifts for a stationary robot (Gazebo Model). Hope to get some answers on this.

Next day: No answers yet. At first I believed that the drifting was due to noise. I decided to set the noise to zero and try again. Again the same drift. Puzzling. Then I changed some parameters in the robot_localization config - `remove_gravitational_acceleration` to `false`. Now there was no drift! **Puzzling**.

Tuning robot_localization's covariance matrix

Next day: I set the IMU noise to zero for the time being. Then I set `publish_tf` to `false` for `robot_localization`. And I set `publishOdomTF` to `true` for the differential drive controller. I wanted AMCL to use Gazebo's odometry for localization. Then I visualized 3 markers:

1. pose estimated by amcl
2. Odometry published by Gazebo
3. Odometry estimated by robot_localization using IMU data

I realized that the odometry published by Gazebo accurately follows the robot's position. The Odometry published by `robot_localization`, however, only follows the robot as long as it is moving straight. When the robot slows down or turns the keeps moving ahead and reacts very slowly to robot's turn/deceleration. **This suggests that the kalman filter is trusting the prediction updates more than it is trusting the sensor readings.** So I need to increase the process noise parameters for yaw, yaw velocity and x acceleration.

First lets see if the estimate slows down as the robot slows down. I increased the covariance for x acceleration from 0.5 to 2. Situation improved, however the estimate still lagged the robot's movements. The response to deceleration was very slow. I observed that the response to yaw is fast enough. So I don't need to touch the covariances for yaw & yaw velocity. I only need to increase the covariance for x acceleration. Increased this from 2 to 3. Still a lag. Let us try increasing this to 5. Much better. There was

a slight lag. Lets increase this further to 7. Still a lag. Increase it to 10 (seems too unrealistic). The lag reduced however it is still non-trivial. I don't want to increase the covariance any further.

I will now try using the `robot_localization's` odometry for `amcl`. This time it was reasonable, not very far off. However over a long time the estimate drifts considerably. Also it jumps when the laser applies correction to the incorrect Odometry data.

According to answers to this question Using AMCL without wheel odometry (only laser + IMU), it is difficult to get any useful information with just an IMU. You need another sensor like an Odometer. I highly doubt this since the MIT, UPenn & HyphaROS race cars just use an IMU and laser. Or do they also use wheel encoders? I will have to check and get back here.

Comparison with HyphaROS Localization

I observed that HyphaROS uses `rf2o_laser_odometry` to generate Odometry via laser. Essentially this package gives an estimate of `vx` and `vy`. The github page says this:

```
Estimation of 2D odometry based on planar laser scans. Useful for mobile
robots with inaccurate base odometry.

Its very low computational cost (0.9 milliseconds on a single CPU core)
together with its high precision, makes RF2O a suitable method for those
robotic applications that require planar odometry.
```

According to the HyphaROS presentation:

1. IMU is good for rotation estimation.
2. Odometers are good for translation (`x`, `vx`).

HyphaROS uses `rf2o_laser_odometry` and IMU data to generate an Odometry to be used with `amcl`. Essentially it uses the laser data twice. Once for estimating Odometry and once for use in particle filters. This is interesting. I also observed that HyphaROS tunes some parameters for the `amcl` as well.

Tuning amcl parameters

HyphaROS tuned the following amcl parameters (meaning others were left at their default values):

```
<param name="transform_tolerance" value="0.2" /> (default: 0.1)
<param name="gui_publish_rate" value="10.0"/> (default: disabled)
<param name="min_particles" value="500"/> (default: 100)
<param name="kld_err" value="0.05"/> (default: 0.01)

<!-- laser measurement parameters -->
<param name="laser_max_beams" value="50"/> (default: 30)
<param name="laser_z_hit" value="0.5"/> (default: 0.95)
<param name="laser_z_short" value="0.05"/> (default: 0.1)
<param name="laser_z_rand" value="0.5"/> (default: 0.05)

<!-- movement thresholds -->
<param name="update_min_a" value="0.01"/>
<param name="update_min_d" value="0.01"/>

<!-- particle filter parameters -->
<param name="resample_interval" value="1"/> (default: 2)
```

```
<!-- translation std dev, m (prediction noise) -->
<param name="odom_alpha3" value="0.8"/>
```

On reading the `amcl` documentation I observed that it uses a **'Likelihood Field'** model for laser measurements. I need to understand more on this. So I decided to read the relevant chapter of Probabilistic Robotics, whose one of the authors is Sebastian Thrun.

The parameters `laser_z_hit, laser_z_short, laser_z_rand` and `laser_z_max` can be understood from the following diagram:

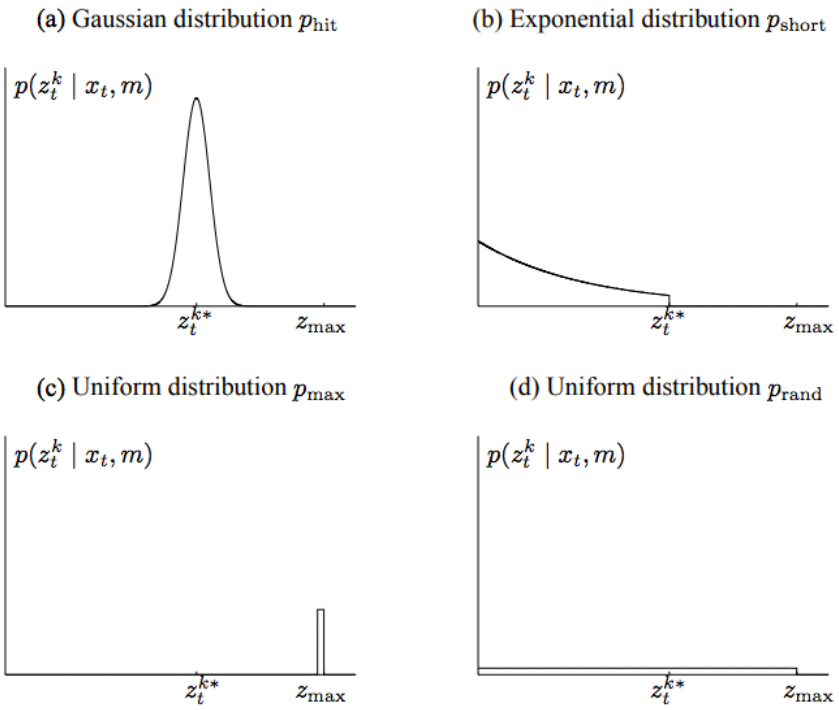


Figure 6.2 Components of the range finder sensor model. In each diagram the horizontal axis corresponds to the measurement z_t^k, the vertical to the likelihood.

Source: Probabilistic Robotics

The `beam` model uses all 4: `z_hit, z_short, z_max`, and `z_rand`. The `likelihood_field` model uses only 2: `z_hit` and `z_rand`. Note that whichever mixture weights are in use **should sum to 1**. The following 2 parameters control the hit & short distributions:

```
~laser_sigma_hit (double, default: 0.2 meters)
      Standard deviation for Gaussian model used in z_hit part of the model.

~laser_lambda_short (double, default: 0.1)
      Exponential decay parameter for z_short part of model.
```

The book Probabilistic Robotics (Chapter 6) goes into choosing these 6 intrinsic parameters - `z_hit, z_short, z_max, z_rand, laser_sigma_hit` and `laser_lambda_short` through experimental data.

~~Also from reading this chapter it appears to me that the `beam` model is suitable for real life situations where the robot may encounter non-static random obstacles whereas the `likelihood_field` model is suitable for Gazebo simulation~~. The book describes an alternative model, called likelihood field model, which overcomes

these limitations. This model lacks a plausible physical explanation. In fact, it is an “ad hoc” algorithm that does not necessarily compute a conditional probability relative to any meaningful generative model of the physics of sensors. However, the approach works well in practice. In particular, the resulting posteriors are much smoother even in cluttered space, and the computation is typically more efficient.

Localization using AMCL: Part 2

After 2 days, localization is working reasonably. However now the problem is that the estimated pose lags behind the ground truth. Earlier the estimated pose would move ahead of the actual pose. I guess reducing the process noise will make it better.

I used the following plugin to view the ground truth:

```
<!-- plugin for publishing the robot's 3D pose -->
<!-- read more:
http://ebrenna8thesis.blogspot.com/2017/11/plugin-for-publishing-robot-pose
-to-ros.html -->
<!-- QUESTION: what should the frame name be? -->
<gazebo>
    <plugin name="ground_truth" filename="libgazebo_ros_p3d.so">
        <frameName>map</frameName>
        <bodyName>base_link</bodyName>
        <topicName>car/ground_truth</topicName>
        <updateRate>30.0</updateRate>
    </plugin>
</gazebo>
```

The following parameter changes made things better:

```
<param name="update_min_a" value="0.01"/> (default: pi/6)
<param name="update_min_d" value="0.01"/> (default: 0.2)
<param name="odom_alpha3" value="0.8"/>
```

Also I should mention that I edited and fixed the map. You can see the map in the **Edited Map** section. Further tuning is required to make the localization perfect.

Should IMU & LIDAR be enough?

I posted this question - Indoor localization with 2D lidar and IMU, to understand if IMU & LIDAR should be enough for indoor localization. I have read in other places that integrating acceleration (from accelerometers) is a tough job. You need at least some estimate of speed or distance travelled (can come from wheel encoders). The answer posted on the above question also concurs with this.

rf2o_laser_odometry

So here is my plan:

1. In simulation use `rf2o_laser_odometry`, just like HyphaROS
2. On the actual car use odometry information from VESC

The demo video of `rf2o` on http://mapir.isa.uma.es/work/rf2o looks promising. However when I tried it, the results were not very impressive. I guess the problem was the feature-less environment (long tunnel). So it would be difficult to estimate the velocity using laser scans. Let me try zig-zagging my way through the environment. If the results are better, then my assumption would be right.

Observation: Zig-zagging improves things. Great, so this should work well in a real life environment. However the performance is nowhere near the one in the demo video.

Observation 2: It was even worse when using rf2o on actual laser data from the car. I think this is a useless library.

Fusing rf2o and IMU data

Next I tried fusing `X_dot` and `yaw` from the `rf2o` with the IMU data using `robot_localization`. The result was OK. Not very accurate. Now I will try fusing just `x_dot` from `rf2o` since the `yaw` and `yaw_dot` from the IMU are good enough. The results were much much better now!!

Now it is time to use `amcl`. It worked very well this time. Finally success!! However the execution was slow. I think this was due to the addition of `rf2o`. Fortunately this should not be a problem on an actual machine as I would be using Odometry given by VESC instead of `rf2o`.

I should also mention that this time I changed the order of starting nodes. Instead of starting nodes in a single launch file I used separate launch file. The order was:

1. Gazebo world
2. `rf2o` (this needs an initial pose to be sent on the `/initial_pose` topic)
3. `robot_localization` (EKF)
4. `amcl`

I suspect the `sensor_timeout` parameter may trigger some weird functionality if the `rf2o` does not start on time.

Mapping

Now that I have implemented mapping in Gazebo using `gmapping`, its time to use it on the car. I realized now that `gmapping` requires `Odometry` information, in other words it looks for a `odom -> base_link` transform. Apparently one can fake this information by using `laser_scan_matcher`. See the comments on this answer SLAM without odometry: gmapping or hector_slam?. `hector_mapping` does not require any Odometry data. Also see this gmapping without /odom. I will give `hector_mapping` a shot.

Recording bag file

I decided to create a bag file of laser scan data and `odom -> laser` transforms and then reply the bag file to create a map using `hector_mapping`. Recording bag file is easy:

1. Start the `rplidar_ros` node
2. In another terminal run `robag record -a`
3. `-a` option is to record all topics. Otherwise you have to specify individual topic names.

Odom to laser transform

Before replaying the bag file I decided to publish the `odom -> laser` transform. I have mentioned before that I have installed the laser in the reverse position. So the transform needs to have 180 degrees rotation about the z axis. One can use these functions:

```
quaternion = tf.transformations.quaternion_from_euler(roll, pitch, yaw)

quaternion = (pose.orientation.x, pose.orientation.y, pose.orientation.z,
          pose.orientation.w)
euler = tf.transformations.euler_from_quaternion(quaternion)
roll, pitch, yaw = euler[0], euler[1], euler[2]
```

I input the values `roll = 0, pitch = 0` and `yaw = pi` in the above function to get `(0, 0, 1, 0)`.

Replaying bag file

Replaying from a bag file is also easy:

1. Start `roscore`
2. `rviz` and `odom -> laser` transform
3. `rosbag play <bag_file_name>`

This led to an error which said *"Message removed because it is too old".* This post came to my rescue. Error in rosbag play despite setting use_sim_time param. This is the correct order of running commands:

1. Start `roscore`
2. `rosparam set /use_sim_time true`
3. `rviz` and `odom -> laser` transform
4. `rosbag play` **`--clock`** `<bag_file_name>`

Notice the `--clock` argument while playing the rosbag.

I ran into another problem. This time the time stamps in the bag file and the ones on RPLidar messages differed by ~1000 seconds. Due to this I got the message saying *"Message removed because it is too old"*. Was this because the clocks on the odroid and my laptop (VM, where I recorded the bag file were different?). I found these posts helpful:

1. rosbag --clock time for files recorded on a different machine
2. ROS time across machines
3. How to associate Rosbag time with data

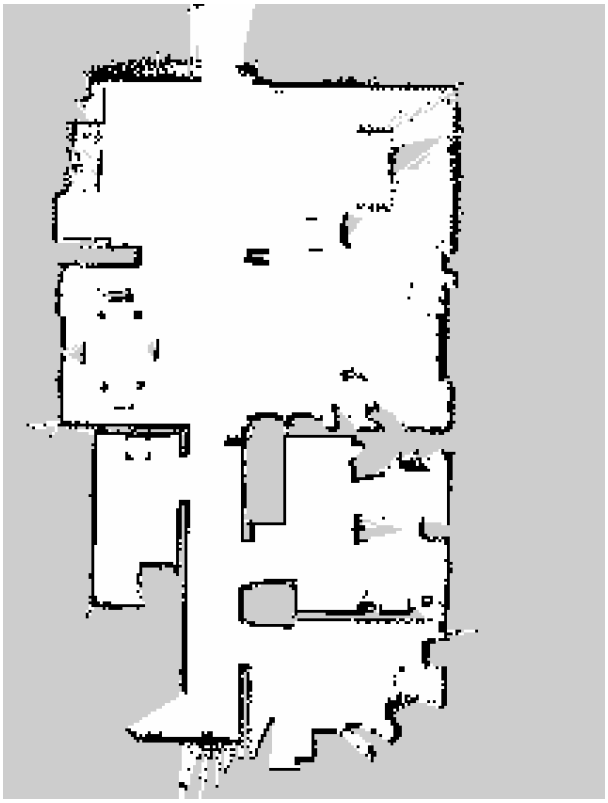
As suggested in the first post, I just repaired the bag by running the code in Rewrite bag with header timestamps, with the modification posted in the 2nd answer on the 3rd post:

```
msg.header.stamp.secs = t.secs
msg.header.stamp.nsecs = t.nsecs
```

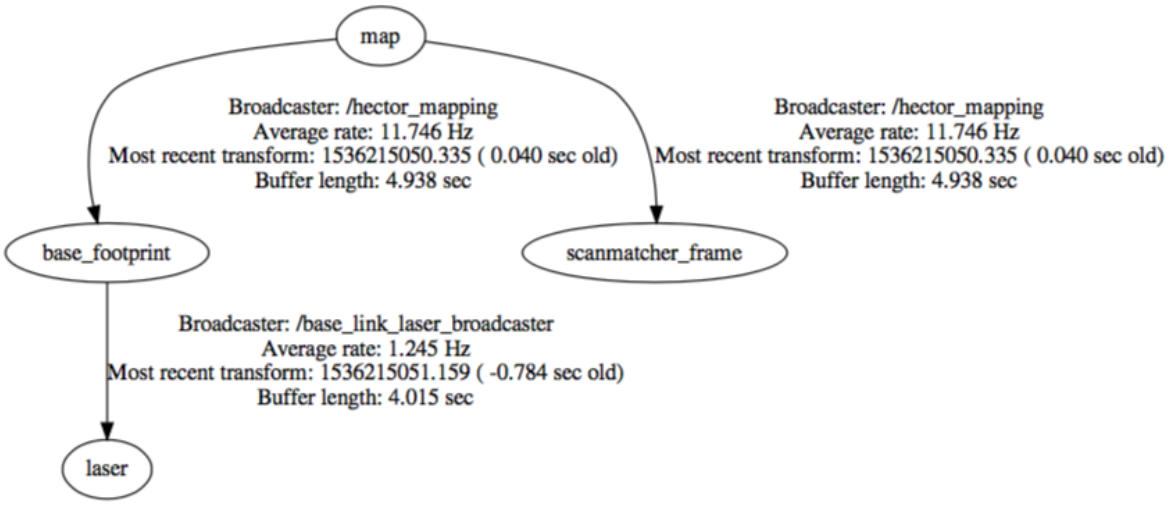
This fixed the error!

Hector Mapping

Time to create a map using the recorded bag file. These tutorials were useful [How to set up hector_slam for your robot]. More specifically the section on using `hector_mapping` without the `odom` frame, since the robot does not yet have an estimate of odometry. This is the map generated:



Good enough! I will edit this map in gimp so that it is accurate. This is the transform tree when hector_mapping is creating the map



Mapping with known pose

When you are using Gazebo, it can publish the actual pose of the robot on a certain topic. So you may not need to rely on hector_mapping to compute the transforms. In that case you can use these 2 undocumented parameters of hector_mapping to create the map:

```
<arg name="map_with_known_poses" value="true"/>
<arg name="use_tf_pose_start_estimate" value="true"/>
```

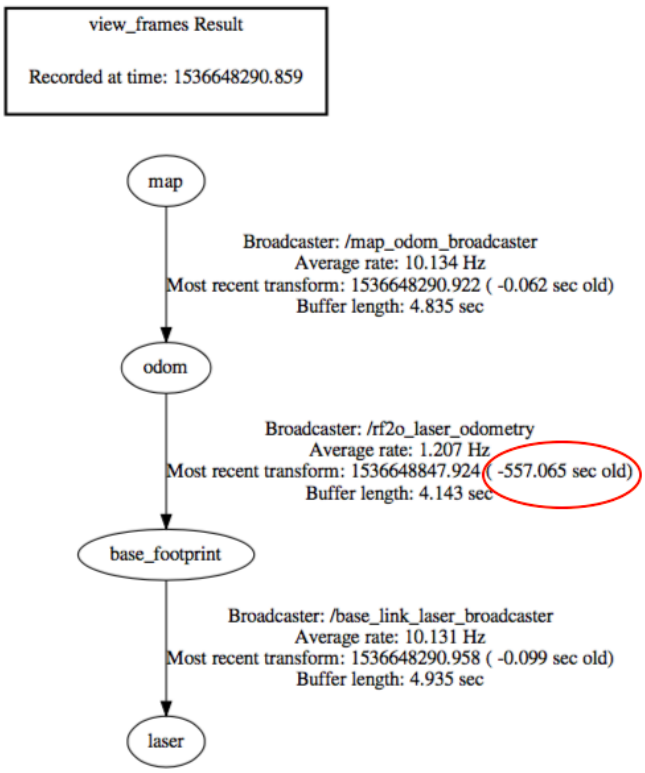
And then:

```
<arg name="pub_map_odom_transform" value="false"/>
```

Please see this post https://answers.ros.org/question/185496/modify-position-in-hector_mapping-code/

Time Synchronization

I realized that the Odroid was using timezone `UTC` and the VM was using `IST`. The clocks on the 2 machines need to be synced because messages use timestamps in their headers. If a particular message has an old timestamp then it is ignored. This is important especially for `/tf` messages. Have a look at this tf tree:

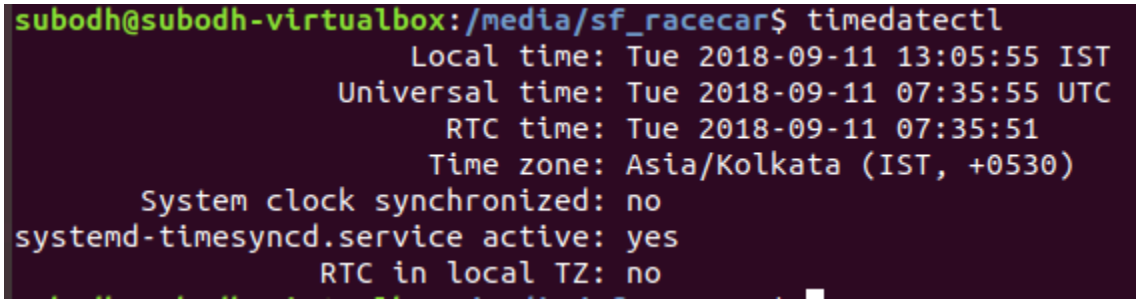


Although the tree looks fine, one of the transforms is too old, about 10 minutes old. The tf method lookupTransform, will throw an error if a particular transform is too old. In my case the `rf2o_laser_odometry_node` , which was being run on the Odroid, was using this method to lookup the transform between `base_footprint -> laser`. Whereas the static transforms between the `map -> odom` and `base_footprint -> laser` were being published on the VM. So I got these errors:

```
[ERROR] [1536648967.296507406]: "base_footprint" passed to lookupTransform argument target_frame does not exist.
[ERROR] [1536648968.304547065]: "laser" passed to lookupTransform argument target_frame does not exist.
```

This blog post helped me fix the time zone on the Odroid - How To Set Up Time Synchronization on Ubuntu 18.04. To check the timezone use the command `date` in the terminal. To list timezones use `timedatectl list-timezones`. To set a timezone use `sudo timedatectl set-timezone Asia/Kolkata`.

After setting both the timezones to `Asia/Kolkata`, I realized that the VM time was behind the actual time by about 10 minutes (same as the oldness of the transform in the above tf tree). Running the `timedatectl` command showed that the system clock is not synchronized on the VM.



This post helped in fixing the system clock How to sync the time to network with timedatectl on Ubuntu 18.04?. Although the `systemd-timesyncd.service` was shown as active, it was actually dead. I had to restart it. The command to do this is `sudo systemctl restart systemd-timesyncd.service`. Now both the clocks are synced. However I will have to run this everytime I start the VM.

A more permanent solution would be to sync the time of the VM and the odroid using the VM's time. This document was helpful in getting this done - Time synchronization. I decided to skip this and come back to it later.

Localization

I was wondering, if hector_mapping can create such a good map using just laser scans then it should also be able to perform localization very well. I must try it. After reading through some posts online, I realized that there are many different approaches to localization using laser scans and IMU.

1. `amcl` with `rf2o_laser_odometry`
2. `amcl` with `rf2o_laser_odometry` & IMU (fused using robot_localization)
3. `hector_mapping` with mapping turned off (fake off)
4. `amcl` with `odom->base_link` transform from `hector_mapping`
5. `amcl` global localization

I went through these posts:

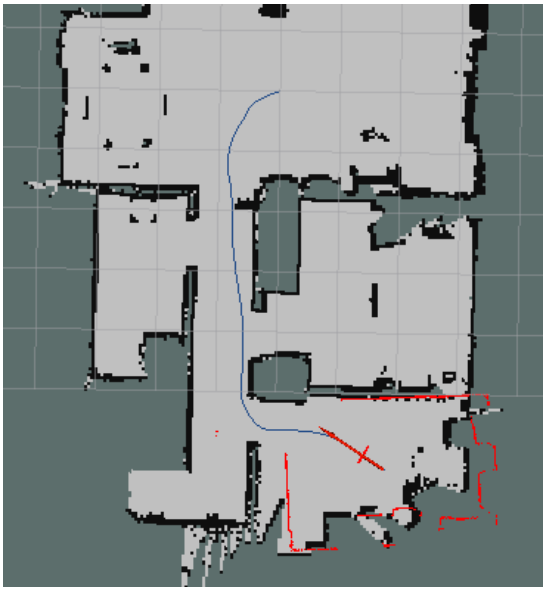
1. Autonomous Localization using AMCL
2. Using Hector mapping for odometry with amcl for localization in a pre given map
3. Using hector slam with amcl for localization in a pre-made map

In order that I don't have to switch on the robot again and again I recorded IMU and LIDAR data into a bag file. Then I tried various techniques using this data.

hector_mapping only

I wanted to exploit the fact that `hector_mapping` gives pretty accurate position estimate. I decided to load a premade map using the `map_server` and then plot the estimated pose on the map. In order to prevent `hector_mapping` from publishing the map, I remapped the topic `/map` to something else for the `hector_mapping` node. No other changes as compared to mapping.

I used `hector_trajectory_server` to map the trajectory of the robot. The result was this:



Quite close but not very accurate. You can conclude this from the difference between the laser scans and the walls in the last frame.

hector_mapping with amcl

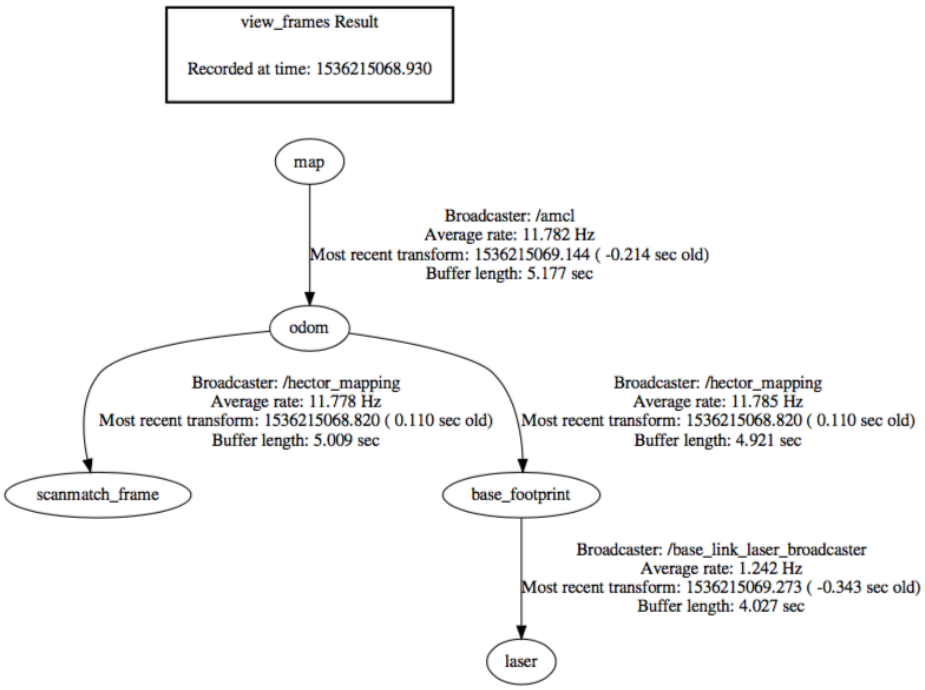
This won't work. `amcl` requires odometry estimate i.e. transform between odom frame and base_frame. Whereas `hector_mapping` directly estimates the pose of the laser in the map frame. You can't use `hector_mapping` as an odometry source. See my comment here Using Hector mapping for odometry with amcl for localization in a pre given map. Also see this answer Difference between scan matcher and odom frame?

```
The scanmatcher_frame always provides the LIDAR pose directly in the used
map frame (mainly intended for debugging). The map->odom transform will be
(notably) different if the robot system provides odometry as essentially
only the odometry error is corrected in that case. See the setup tutorial
for info for info on how IMU data can be incorporated.
```

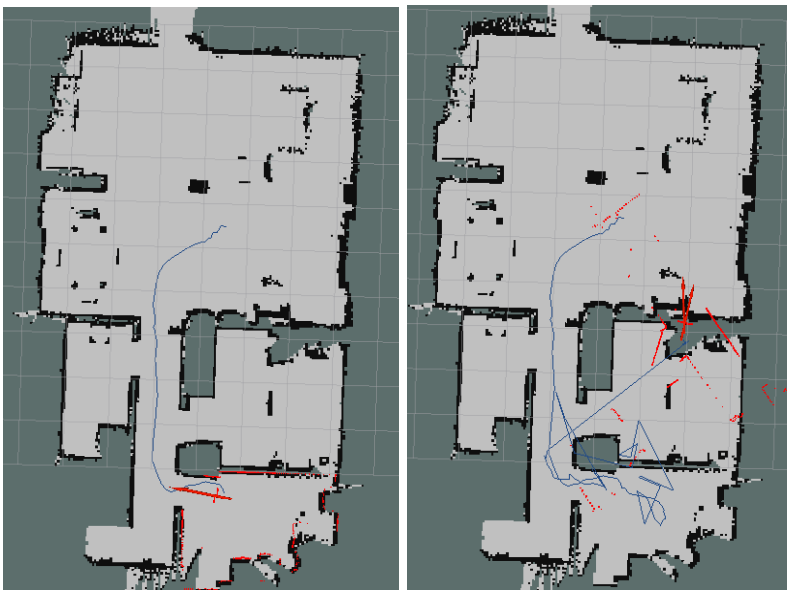
So using this hint I made the following modifications for `hector_mapping`:

```
<arg name="map_frame" value="odom"/>
<arg name="base_frame" value="base_footprint"/>
<arg name="odom_frame" value="base_footprint"/>
```

The tf tree looks like this:



amcl used the “map_frame” to “odom_frame” transform from `hector_mapping` as its odometry estimate. This works well initially but then quickly loses its way, as is evident from these pictures:



So this doesn’t look like a viable option to me.

amcl with rf2o

I started with first visualizing the rf2o Odometry only. I was getting the following error when trying to play back data from a bag file:

`"base_footprint" passed to lookupTransform argument target_frame does not exist`

The mistake was launching all the static transforms and the `rf2o_laser_odometry_node` node in the same launch file.

`rf2o_laser_odometry_node` needs all the transforms to be published before it is started. Starting things in the following order fixed it for me:

1. Start roscore & publish static transforms
2. Play back data from the bag file
3. Start rf2o_laser_odometry_node

However the result was a lot worse. I am not even sure if this node is remotely useful or not.

amcl with IMU

First step is to visualize only IMU odometry.

Laser_scan_matcher

http://wiki.ros.org/laser_scan_matcher.I decided to skip this since the repository did not have any updates since 2016.

VESC

VESC stands for Vedder Electronic Speed Controller, an open source electronic speed controller(ESC). The Traxxas car also comes with an ESC, however the minimum speed configured on them is 6 mph and the ESC is not programmable. VESC is a programmable controller which allows lower speeds. It has the feature of strong, reliable and progressive electric brakes. It is widely used in electronic skateboards. Read more here The most advanced speed controller for electric skateboards - VESC.

VESC is now a registered trademark. There are several manufacturers who build VESC compatible hardware, but expect different names since the registration. Eg. FOCBOX Speed Controller, Maytech

VESC. I decided to buy from Maytech (sold on Aliexpress) because it was cheaper at $110 vs $250 for FOCBOX.

Reading this was helpful New vesc user? Read This - COMPLETE WALKTHROUGH OF THE VESC.

BLDC Tool

I decided to install the bldc-tool in the VM. This document was helpful installBLDCToolHost.sh. After the setup the video in this blog post RACECAR/J – Programming the Electronic Speed Controller is all you need to load the firmware and configuration onto the VESC.

Firmware Part 1

The Maytech description on Aliexpress says *"Firmware is updated to 3.30 now. If you need 2.18 firmware, please leave note when place order"*. However the MIT firmware assumes 2.18. There is this PR make vesc_driver work with latest VESC firmware 3.38. Need to check this out.

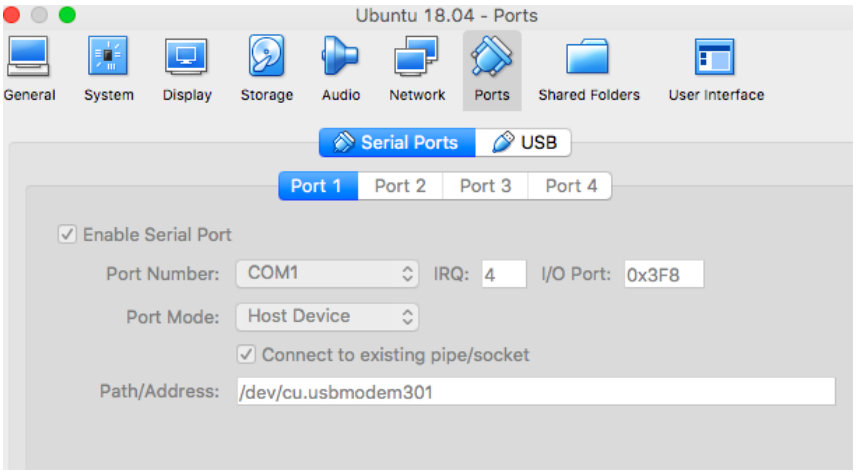
Setting up USB & serial ports in Virtualbox

I connected the VESC to the LiPO battery and plugged the USB cable from the VESC to the computer. Mac could detect the USB device (command `ioreg -p IOUSB`), it starts with `ChibiOS/RT`. This is the OS used on the VESC. However the VM could not. Then I followed instructions over here How to set up USB for Virtualbox?. I had to install VirtualBox Extensions Pack and then had to add the device (*STMicroelectronics ChibiOS/RT Virtual COM Port*) so that I could access it from the guest OS. Now on the VM we can do lsusb and see the device *STMicroelectronics STM32F407* listed.

Now in the BLDC Tool when I hit Connect, it said *"Serial port error: 2"*. Googling suggested it has something to do with the permissions for he serial port. I needed to setup serial port on the VM. I gave up doing this on the VM. I decided to use the Odroid to configure the VESC in the interest of time. On the Odroid I ran into the familiar problem.

```
The X11 connection broke: Unsupported extension used (code 2)
XIO:  fatal IO error 22 (Invalid argument) on X server ":1"
      after 6 requests (6 known processed) with 0 events remaining.
```

So I went back to the VM. After setting up the serial port



This blog post was helpful: Serial Port Access in VirtualBox Machines running on Mac OS X.

When I tried to boot the VM it said `'/dev/cu.usbmodem.301'` is behaving improperly. I also noticed that the red LED on the VESC was blinking continuously (repetition of 2 times blinking in quick succession). Usually a red LED means something bad is happening. I stumbled on to this discussion:

Source: Vesc Leds explained - SOLVED

> The blue is for the power, the green is for the command, the red is for the drv failure (and it is normal the see the red led blinking in the vesc boot cycle).
>
> > Not true, the red blinking means there's a fault code. Which could be a drv related fault code, it could mean your powering with a voltage outside of the range you set in the vesc (ie, you configured it for 38-42 volts (10s), but you are powering them with an 8s), it could be another issue.
> > Red blinking after the initial 3 means there's a fault code. Hook it up to the bldc tool, connect via usb, go to live data tab, click activate sampling, and the line labeled fault code will tell you your problem (along with the other data you see there).

Could this be because I am powering the VESC outside the voltage limits (the device says 8-60V, my input was about 7.9 V). I can try charging the battery upto 8.2V and try again or else I will have to use 2 batteries in series. I charged the battery to 8.23V and there was no red LED blinking anymore!!

I restarted the VM and there was no error message anymore! However on hitting the *'Connect'* button in the BLDC tool I still got an error *"Serial port error: 2"*. I fixed this after reading this thread serial port terminal error. Basically I had to add the user `subodh` to the group `dialout`. One can find if the user is already part of the group by running `groups username`.

Then I restarted the VM. And now on hitting the 'Connect' button in the BLDC tool I got a message dialog saying *"The firmware on the connected VESC is too old. Please update it using a programmer"*.

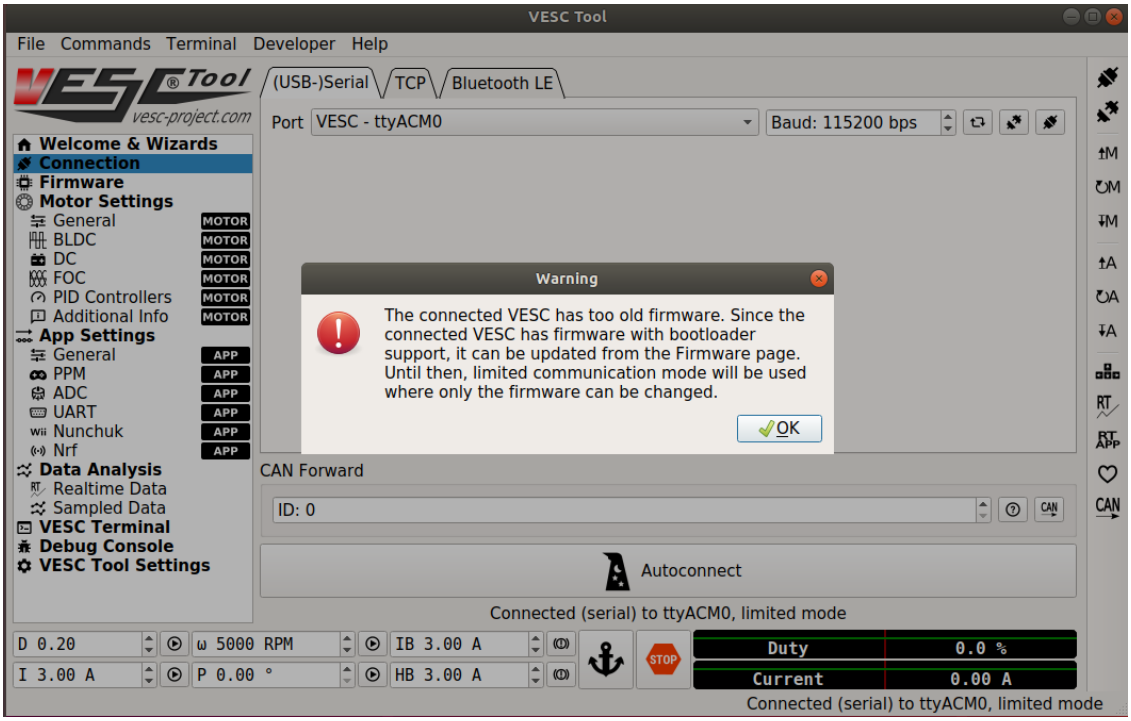
I suspected the message may be incorrect and that the current firmware version (3.30) is too new for the BLDC tool (which supports 2.18). My suspicions were confirmed by this comment:

> The firmware is actually too NEW for the BLDC_Tool, it doesn't know how to handle 3.32, so gives message that firmware in unit is too old!
>
> After Firmware 2.18 use VESC Tool instead of BLDC Tool. Works fine and even allows you to go back to V2.18 to use BLDC Tool again
> Source: 4.12 "The firmware on the connected VESC is too old"

The above thread suggested using the VESC Tool to downgrade the firmware to 2.18. This, apparently, allows you to use the BLDC tool again. I will give it a try.

VESC Tool

I installed the VESC Tool, using instructions from here, and it works fine. It shows me this message on start:



It shows me hardware version 4.10 and firmware version 3.30. So the information on the Maytech website and product labelling is incorrect (label says HW v4.12). The same thing happened for this user. Or maybe the tool is wrong.

Firmware Part 2

I have 2 options now:

1. Update the firmware to 3.40 (Instructions: Updating the Firmware)
2. Or downgrade it to 2.18

For the upgrade to work I will have to use code from this PR make vesc_driver work with latest VESC firmware 3.38, but before that I need to ensure that there are no configuration changes required to the files used by MIT Racecar team. I have left a comment on the PR. I will wait for a reply before upgrading the firmware on the VESC.

```
You should update! 2.xx is outdated. The new FW is a lot better. It's not
recommended to use the old FW and BLDC-Tool any more.

Frank
Source: VESC-Project.com is online! Public VESC-Tool download
```

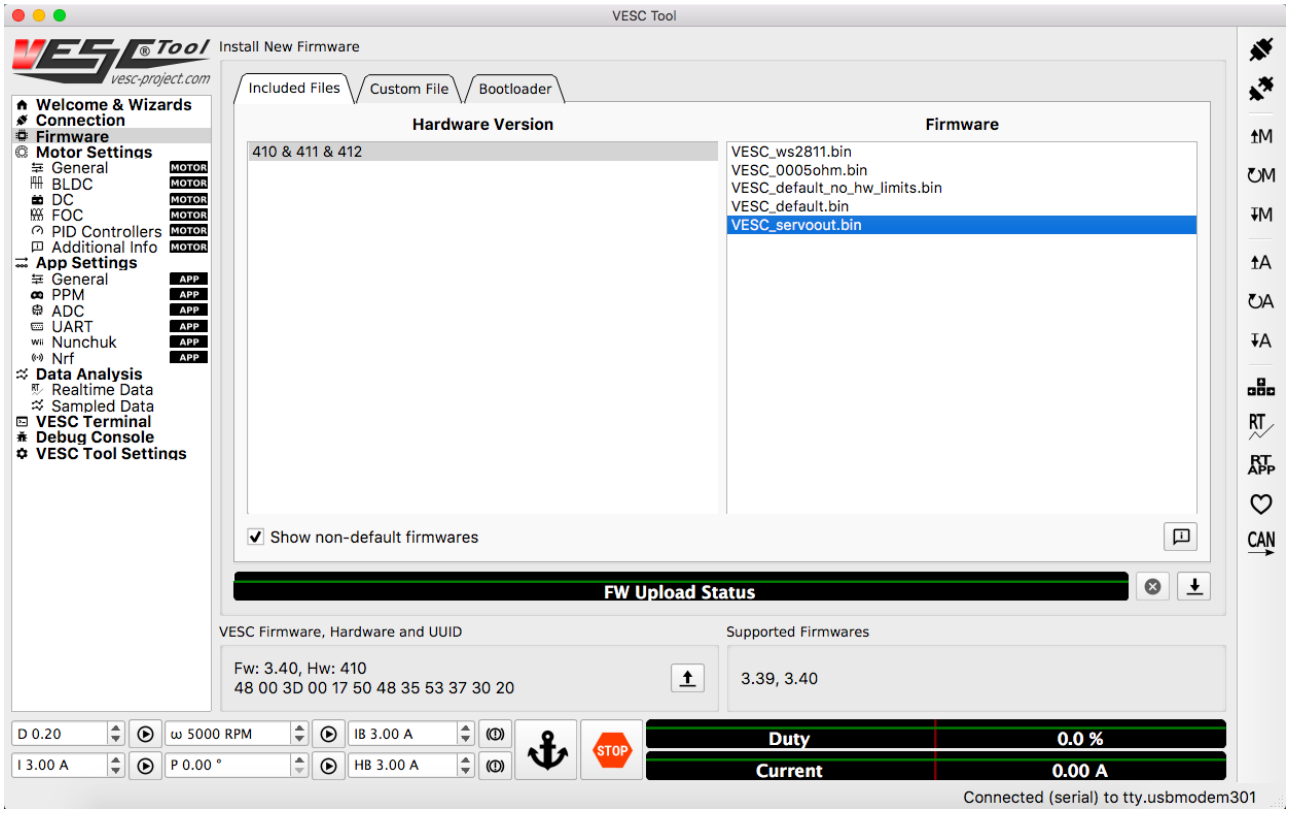
Upgrading may make sense since the BLDC tool has not been updated since 2016.

I got no response to the comment I left on the PR about making the VESC driver work with firmware 3.38. On my gut instinct I decided to update the VESC firmware to 3.40 (latest available).

VESC Configuration

Once the firmware was updated to v3.40 (VESC_default.bin), it is time to update it to `VESC_servoout.bin`. First I need to understand what the *'servo out'* means. Apparently this enables the VESC to control the servo motor independently of the DC motor. Read this:

1. VESC Servo Control
2. Servo issue after fmw update
3. What is the VESC_servoout.bin build?



The firmware was updated to `VESC_servoout.bin` using the VESC Tool. Now the configuration.

I noticed that the MIT's VESC configuration has the following settings:

```
<l_min_vin>7.5</l_min_vin>
<l_max_vin>57</l_max_vin>
<l_battery_cut_start>10</l_battery_cut_start>
<l_battery_cut_end>8</l_battery_cut_end>
```

So, if I understand this right:

1. The VESC will only function if the input voltages between 8V and 57 V. This seems straightforward.
2. As the voltage falls below 10V, the power supplied to the motor will fall.
3. For voltage below 8V, the motor won't spin at all.

A fully charged 2S LiPO is 8.4V and nominal voltage is 7.4V. So for a 2S battery these cut-off settings are high and to make it work I have to:

1. Set 'Minimum input voltage' to somewhere around 7.0V

2. Set 'Battery cutoff start' to somewhere around 7.5V
3. Set 'Battery cutoff end' to 'Minimum input voltage' i.e. 7.0V

My understanding comes from going through the following resources:

1. DIY Electric Skateboard: How to Configure the VESC Tutorial
2. Battery Cutoff Start/Stop Failure

Apparently there is this code somewhere:

```
// Battery cutoff
   if (v_in > conf->l_battery_cut_start) {
      conf->lo_in_current_max = conf->l_in_current_max;
   } else if (v_in < conf->l_battery_cut_end) {
      conf->lo_in_current_max = 0.0;
   } else {
      conf->lo_in_current_max = utils_map(v_in, conf->l_battery_cut_start,
            conf->l_battery_cut_end, conf->l_in_current_max, 0.0);
   }
```

So the intention is to not allow current below the low cut-off. My concern is that setting minimum voltage to 7V should not brick the VESC. Its expensive!!

To confirm my understanding I have posted in the following forums: F1/10, vesc-project and vedder.se. I have also posted these questions to the supplier on Aliexpress and to Shreeyak. I will wait for a reply.

[UPDATE] I heard back on the vesc-project forum. It turns out that both the F1/10 and MIT Racecar projects use bigger batteries. The F1/10 BOM says:

1. NiMH battery 3000mAh, 7 cell
2. Li-Po battery 5000mAh, 3 cell 11.1V

The MIT Racecar github repo says "Traxxas 2940X Series 3 3300mAh NiMH 7-Cell, 8.4V Battery". This battery is 10.5V fully charged, 8.75 50% charged, 7V fully discharged. Further information - The VESC has a chip DRV8302, which is a gate driver IC for three phase motor drive applications. Its operating voltage is 8-60 V. It will behave funnily or may damage the VESC at input voltages lower than 8V. So I guess the solution to use my two 2S batteries in series.

[UPDATE 2] According to Shreeyak using two 2S batteries in series for the Traxxas could be a bad idea since they that would be too high a voltage for the motors and I may end up damaging them. He says the VESC works fine with a 2S battery, with settings similar to the ones I have mentioned above.

Motor Setup Wizard - Part 1

First things first - go through this tutorial Motor Setup Wizard.

I was trying to setup my motor using this wizard and on the BLDC Parameters (Cycle Integrator Limit and BEMF Coupling) tab, I ran into a "bad detection result". I tried various parameters for current, erpm and duty cycle. One set of example values is current = 3.0A, erpm = 250, duty cycle = 0.05.

I was able to finally complete the motor setup wizard. Things worked out quite luckily for me. For a long time I kept getting the error "bad detection result". Then by sheer chance I happened to load the XML configuration from vesc-firmware/VESC-Configuration/VESC_30k_erpm.xml, and I ran the detection again and it worked. I ran the detection 7 times and this is the result that I got:

Current (A)	ERPM	Duty Cycle	Cycle Integrator Limit	BEMF Coupling	Unknown Hall Error
3	600	0.05	39.51	465.13	255
3	600	0.08	39.55	376.29	255
3	600	0.10	39.16	259.68	255
3	500	0.05	39.27	492.83	255
3	400	0.05	39.44	472.62	255
3	300	0.05	39.59	448.56	255
3	300	0.10	40.04	262.94	255

You can see that the *Cycle Integrator Limit* values are quite consistent. The *BEMF Coupling* values are also consistent, but for 2 outliers. So I took the average of these values, excluding the outliers, and set that 450 as the value for *BEMF Coupling*. However when I tested the motor with a duty cycle of 0.2 it cogged (earlier it spun with 6000 RPM). So I lowered the value to 250 and this time the motor rotated by 6000 RPM. I could not explain this. I needed to understand what *BEMF Coupling* and *Cycle Integrator Limit* were.

It looks like it has something to do with the sensors. From the motor configuration tutorial, linked above,:

> Sensors help the VESC to know the rotor position in reference to the copper coils. This way the motor can perform a relatively smooth startup from 0 RPM. If your motor features sensors you should make use of them. At higher RPM, the VESC will use back EMFs to calculate the rotor position. The RPM for switching to back EMF operation can be defined later.
>
> No sensors: without motor sensors the motor needs to turn a bit, so the VESC can calculate the rotor position from the back EMF current flow that the magnets will induce into the copper coils when they rotate around the the motor stator. ***Startup from 0 RPM is possible but a bit shaky.***

It seems like you have to be very careful when using the VESC with a high voltage rating motor as the Traxxas Velineon motor (3500 kV!!) See this thread Driving a high KV Motor.

Another person faced the same problem as me, checkout this thread - Some question about sensorless BLDC(Velineon 3500) spin up not smoothly. This line conveyed an important message:

> If you try to drive power MOSFETs with a gate voltage lower than about 10V, they start performing worse, and heat up. ***This heat can damage the mosfets, So to protect the mosfets the HARDWARE has an "under voltage lock out" implemented.*** It should kick in close to about 8V. You cannot change that limit by configuring the software to attempt to work below 8V.

So by setting lower voltages in the VESC tool I may be damaging the VESC. Maybe I can use 2 batteries in series and lower the max duty cycle? But first I need to understand the working principles of BLDCs.

How does BLDC work?

1. This video explains it well Brushless DC Motor, How it works ? It explains the rotor, stator and need for a Hall sensor.
2. Kv = RPM per volt KV Rating? Higher kv implies higher top RPM, but lower torque.
3. The rotor can have 2 poles, 4 poles etc. The Traxxas Velineon 3500 motor I think has 2 poles?

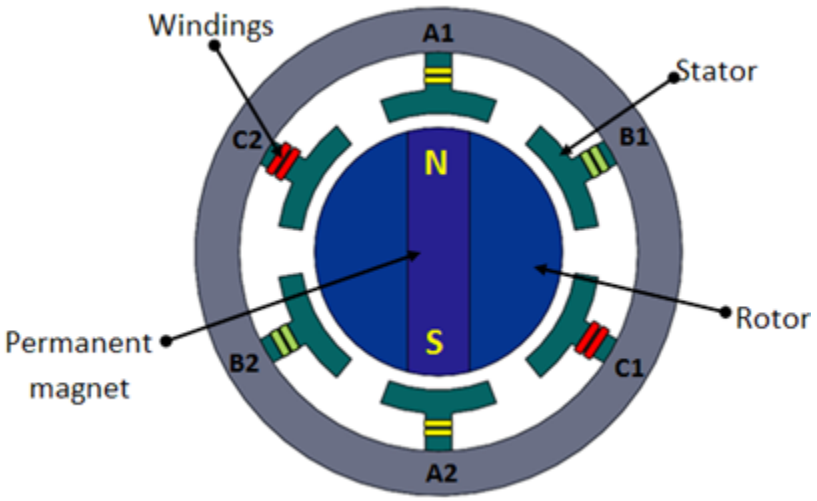


Image Source: NPTEL

Hall Sensors

Unlike a brushed DC motor, the commutation of a BLDC motor is controlled electronically. To rotate the BLDC motor, the stator windings should be energized in a sequence. It is important to know the rotor position in order to understand which winding will be energized following the energizing sequence. Rotor position is sensed using Hall effect sensors embedded into the stator. Most BLDC motors have three Hall sensors embedded into the stator on the non-driving end of the motor. Whenever the rotor magnetic poles pass near the Hall sensors, they give a high or low signal, indicating the N or S pole is passing near the sensors. Based on the combination of these three Hall sensor signals, the exact sequence of commutation can be determined. Source: Electrical Machine Drives.

Sensorless Operation

The Traxxas Velineon 3500 motor is sensorless. So how does the controller determine the position of the poles in a sensorless operation? This thread helped me understand it Board The Min Speed of BLDC with VESC ??

```
To be able to commutate at the right moments, the controller has to measure
the phase voltages. These voltages are small and drown in the electrical
noise when the speed of the motor is low. So, below a certain speed, the
controller is "blind" if your motor doesn't have sensors.
```

Now read this: All About BLDC Motor Control: Sensorless Brushless DC Motor Controllers

A voltage applied across a motor's winding forces the motor's rotor to turn. The movement of the rotor through the motor's magnetic field, however, is analogous to the behavior of a generator, and consequently the motor not only receives an applied voltage but also generates its own voltage. This voltage is referred to as back electromotive force, or back EMF, **and it is proportional to the motor's rotational speed.** Back EMF can be used to determine a motor's rotor speed and position—no sensors are required.

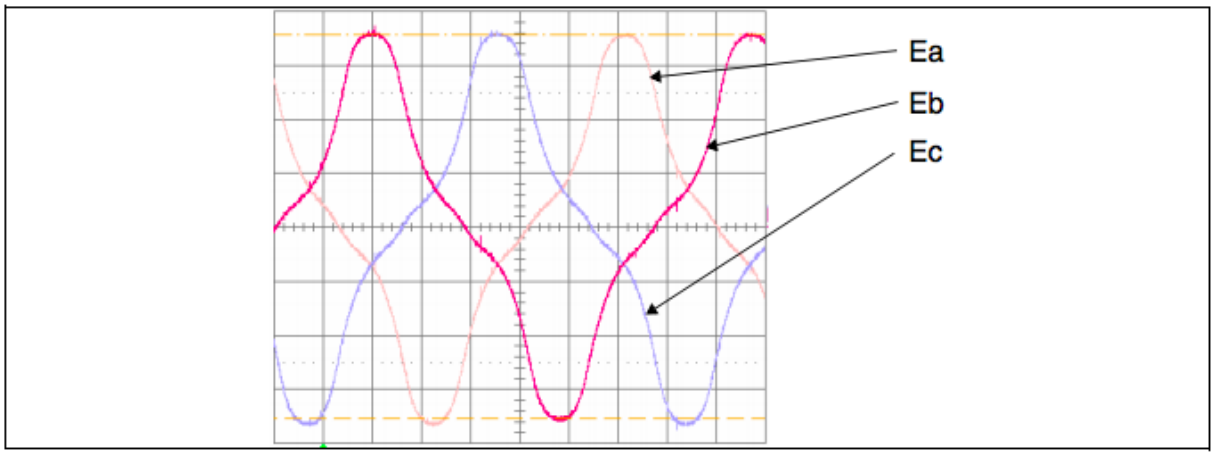
When the rotor of a sensorless BLDC motor is rotating, its sensorless scheme can work perfectly. However, this is not the case when the motor's rotor is ***stationary***, and this leads us to one major disadvantage of using sensorless BLDC motors. **When the motor's rotor is not turning, no back EMF generated.** Without back EMF, the drive circuitry lacks the information it needs to properly control the motor.

And now this SENSORLESS BLDC MOTOR CONTROL AND BEMF SAMPLING METHODS WITH ST7MC.

Because BEMF is proportional to the rotor speed, this implies that the rotor should turn at a minimum speed to generate sufficient BEMF.

Figure below shows the three BEMF voltages referenced to the neutral point for a motor running a constant speed without excitation (the motor is not supplied, and the rotor is manually rotated).

Figure 2. Phase voltage versus Neutral For each stator winding

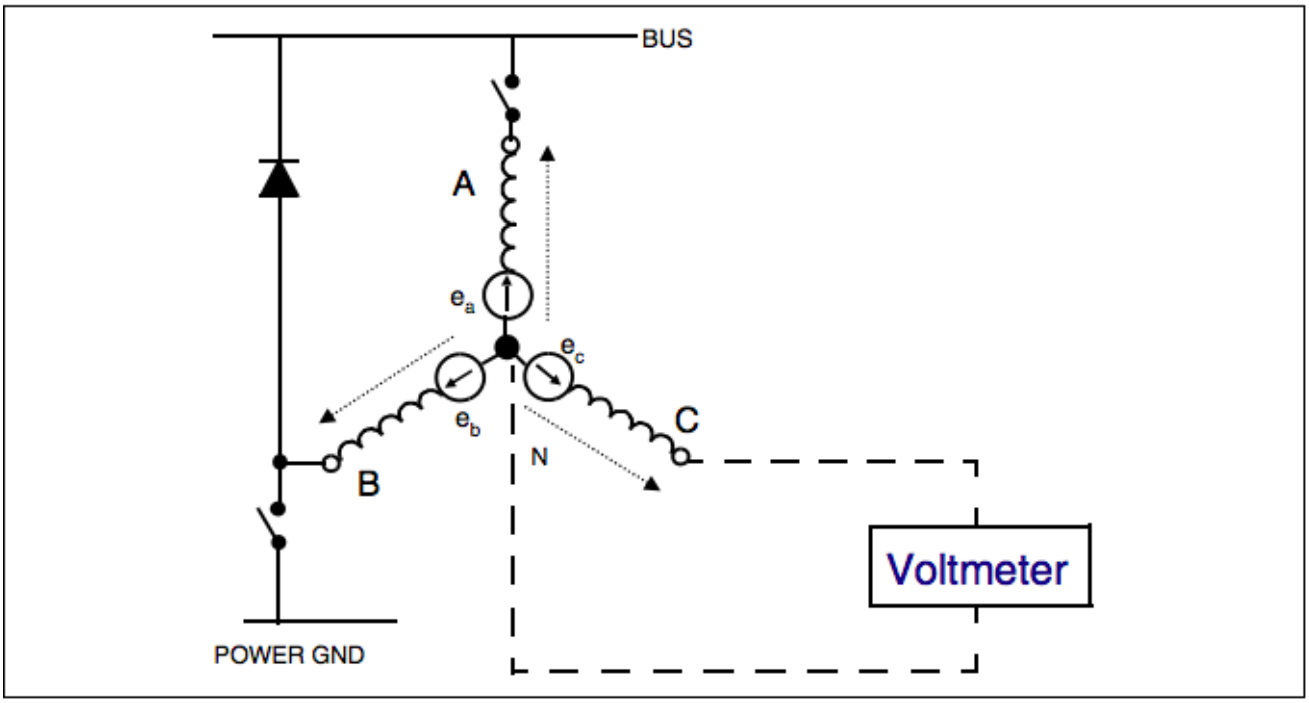


The sensorless method uses the **zero crossing of BEMF** to synchronize phase commutations. For each phase two zero crossings must be detected during a period:

- One "rising crossing" when BEMF passes from negative to positive,
- One "falling crossing" when BEMF passes from positive to negative.

In the non-energized winding (phase C here), the current is zero and the voltage measured is the BEMF of the motor (Figure below).

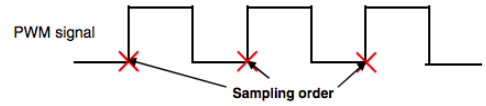
Figure 5. BEMF measurement synoptic.



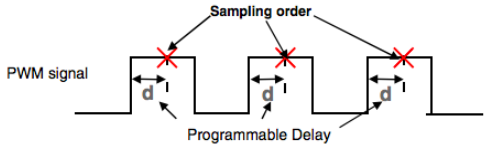
Note 1: Because BEMF is proportional to the rotor speed, this implies that the rotor should turn at a minimum speed to generate sufficient BEMF. This minimum speed varies from one motor to another. For very low speeds it may be required to amplify BEMF in order to control the motor. This is presented in the appendix of this application note.

The ST7MC microcontroller allows the implementation of four methods to sample and detect BEMF zero crossing to run a BLDC motor in sensorless mode (Figure below).

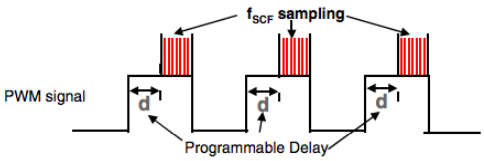
Sampling during PWM OFF State at PWM frequency (SPLG=0 & DS[3:0]=0)



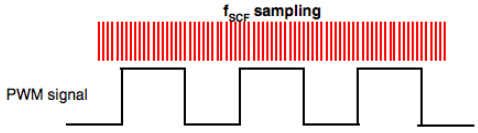
Sampling during PWM ON state only at PWM frequency (SPLG=0 & DS[3:0]=xxx)



Sampling during PWM ON state only at High frequency (SPLG=1 & DS[3:0]=xxx)



Sampling either during OFF or ON state at High frequency (SPLG=1 & DS[3:0]=0)



Now read this blog post by Vedder himself Startup torque on sensorless BLDC motors.

```
One of the major challenges when working on my custom open source ESC was
to get good startup torque and low-speed performance with sensorless
motors.

Techniques I used:
```

1. In order to get a smooth and quick startup without sensors I used several tricks, namely:
2. There are no hardware low-pass filters that introduce phase delay. Such filters can be avoided because the ADC samples are synchronized to the PWM timer and adjusted dynamically every time the duty cycle or switching frequency is changed.
3. After a zero crossing, the ***area under the voltage*** is integrated until a threshold for commutation based on the motor parameters. This integration is robust agaist acceleration and provides good SNR since an integrator is a low-pass filter.
4. There is a parameter that defines the ***voltage coupling between the windings*** when measuring the back-emf. This make a huge difference when running at low speeds with low duty cycle. This compensation has a RPM dependence though, which is something I tried to avoid where possible because the RPM estimation has a delay and thus causes problems during acceleration.
5. To get better voltage samples, the switching frequency is adaptive and proportional to the duty cycle. This is because the ***back-EMF only can be sampled during the ON-time*** of the PWM cycle and low duty cycles have short ON time. Since the motor is running slowly on low duty cycles, sampling and switching does not have to be as fast to keep up with the motor which makes this less of a problem. Lower switching frequency also decreases switching losses, which is a positive side-effect.
6. When the motor is un-driven, the back-emf on all phases is analysed to track the position, direction and speed of the motor. So when the motor is already spinning, the algorithm will begin in the exact state of the motor (position, duty-cycle, direction).
7. I have also used some hack-ish conditions based on trial and error to improve the startup.
8. Closed loop operation is used from the first commutation, since the measured values are clean enough when using these techniques.

Limit of the motor RPM? Simply apply a ERPM LIMIT

Source: Additional settings: Throttle curves, ERPM limits, safety features etc.

RPM limits can be a very useful feature. Possible application: You may want your bike, board, boat, car, etc. to reach a certain top speed only. The VESC-Tool allows you to specify a ERPM-Limit. Many people don't know what the difference between ERPM and RPM is and why VESC-Tol doesn't simply use an RPM value. The relation between RPM and ERPM has something to do with the motor you use and the VESCs doesn't know the exact construction of your motor. But you may know it or you can find out easily.

All you need to know to calculate the ERPM is the number of pole pairs your motor features. Poles are e.g. the magnets of a BLDC-Outrunner/Inrunner motor. If your motor has 14 magnets installed, your motor has 7 pole pairs. In this case the RPM limit of your motor is simply the ERPM limit divided by 7.

Example: 7 Pole Pairs in motor, ERPM limit set to 70000 >>> RPM = 70000 / 7 = 10000 RPM.

There are separate settings for forward and reverse motor operation. In some application you may want a speedy forward movement but a slow running motor in reverse operation.

Excusion: ERPM limits, motor KV, system voltage and hardware limits.

To reach the 10000 RPM at 44.4 V, our motor would need to have a minimum of 226KV. (226 RPM/V x 44,4V = 10034 RPM). In consequence the setting above would only make sense for motors with a higher KV than 226KV if you run them at 44.4V. Depending on the Hardware you use, it may be limited to 60000 ERPM (e.g. HW rev. 4.xx) and you should pick a motor not reaching this limit. Going beyond this limit may damage your Hardware! The VESC SIX design can reach up to 150000 ERPM.

If your motor would reach higher than 60K ERPM and you are using a HW 4.xx controller, you want to set the ERPM limit to 60000 to stay on the safe side and avoid hardware damages.

Motor kv, gear ratio, current, voltage and efficiency

Choosing the right BLDC motor and battery setup for an electric skateboard.

Motor Setup Wizard - Part 2

VESC Sensorless Settings

These definitions are taken from the VESC tool info bubbles.

1. **Cycle integrator limit**: This is how much area will be integrated under the back EMF after a zero crossing before doing a commutation. A too low value will cause a too early commutation, and a too high value will cause a too late commutation. A too late commutation will cause more problems than too early commutations.
2. **Minimum ERPM (BLDC Only)**: Run the motor in open loop when the estimated ERPM is below this value.
3. **Min ERPM Integrator:** The minimum ERPM for which the integrator limit is calculated. Setting this too low will make the coupling compensation too large at low speed resulting in bad startup
4. **BEMF coupling:** Roughly describes how much of the input voltage is seen on the BEMF at low modulation. Compensating for that at low speed helps the startup al lot.

Trying the BLDC Tool

I thought using the BLDC tool with a firmware of 2.18 will fix my problems. So I used the VESC tool to first downgrade the firmware to 2.18. Then I got hold of the Mac version of the BLDC tool from here. EnertianBoards hosts the Windows & Mac versions of the BLDC tool. In the BLDC tool I first uploaded the motor configuration from Jetsonhacks. Then I performed BLDC parameters detection. It worked fine. However testing the motor with a duty cycle of 0.2 cogged the motor. It did not spin well. So I guess this is not a solution to my problem.

Using a 3S LiPo

I ordered a 3S LiPo. Maybe the increased voltage would fix things. I am not very hopeful though.

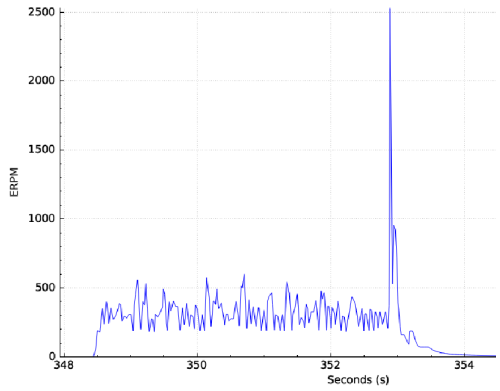
I faced the same issue with a 3S battery. The motor startup in duty cycle control was not smooth. The motor would cog at lower duty cycles. I was still not able to reliably detect the “BLDC” params.

Depending on the value of min ERPM and min ERPM Integrator, the BEMF coupling value was too low (~150) or too high (~450).

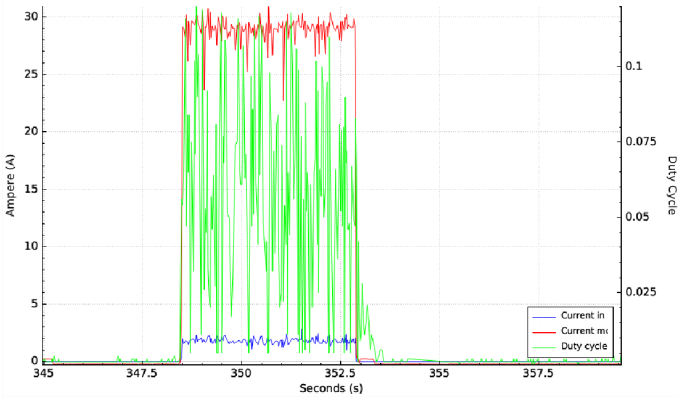
Problems with motor startup

I am following the build guide to tune my PID gains. When I set the duty cycle to 0.2, my motor hardly turns at 300 ERPM, whereas as per the build guide it is supposed to turn at 16000 ERPM. And the motor current is almost at the max limit (30A).

This is my RPM graph when I set the duty cycle to 0.2



This is the current graph for the same setting:



You can see that the motor current hits max limit (30A), without meaningful RPM.

I realized that my BLDC sensorless settings are different than the configuration used my MIT racecar team. When I use the MIT config values and duty cycle of 0.2, my motor cogs for a bit (bad startup) and then runs at 9000 EPRM.

My BLDC sensorless settings are as follows (determined by "Detect BLDC Params" feature of the VESC Tool):

```
Cycle Integrator Limit: 39.5
Min ERPM: 200
Min ERPM Integrator: 600
BEMF Coupling: 450
```

The corresponding MIT racecar config has these values:

```
Cycle Integrator Limit: 62
```

```
Min ERPM: 200
Min ERPM Integrator: 600
BEMF Coupling: 100
```

Setting the BEMF coupling to 100 fixes this issue and the motor runs at about 9000 ERPM with a duty cycle of 0.2. But then why does the VESC tool detect the same parameter as 450? What is the consequence of setting a low BEMF?

Now I try PID control with BEMF Coupling set to 100, Ki = 0.0001 and Kp = Kd = 0. When I set the speed to 3000 ERPM the motor jerks back and forth and does not rotate smoothly. The current also peaks to the max limit that I have set (30 A). What could be wrong with the VESC configuration?

Using FOC Mode

After struggling with this for 3 days I gave up. I used the **FOC mode** instead of the Brushless mode and it worked in the first go. The PID gains could be tuned easily and now the motor startup is very smooth.

I came across some posts on this forum and the ESK8 forum, which recommended not using FOC mode with Maytech VESCs. I decided to give it a go since the brushless mode simply wasn't working for me. Hopefully it won't damage my VESC.

Looking back

Later, after having a spent a lot of time on motor configuration I came across this comment:

> The VESC requires that you tune parameters for each motor. See the "Motor Configuration" section on VESC website http://vedder.se/2015/01/vesc-open-source-esc/. He also has a video. *Tuning motors for small vehicles tends to be tricky due to the low inductance*.
>
> Vedder's VESC GUI sends commands to the VESC the same way that the ROS node does. If you can get the motor to run smoothly with the GUI, you should see similar performance with the ROS node.

So I should have heeded this warning before and had a look at the “Motor Configuration” section on the VESC website.

> For *small low-inductance high-speed motors*, the delay commutation mode can be used in case the integrate mode does not work. It does not require many parameters, just the minimum RPM which usually can be around 1500. I haven’t tested this mode much, but it is more or less how most hobby ESCs work (which is why it doesn’t require so many motor-specific parameters).

So Benjamin Vedder does admit that for low-inductance & high speed motors (like Traxxas) the integrate commutation mode may not work.

Tuning PID gains

TODO: The Upenn build instructions have a section on tuning the PID gains of VESC. Check it out.

As mentioned above, in the FOC mode the PID gains could be tuned easily and now the motor startup is very smooth.

Logitech Wireless Gamepad

I bought a wireless gamepad so that I could control the car with the vesc using a wireless remote. The gamepad has a switch to go from X to D and vice versa. The great thing about this gamepad is that it works as both an *XBOX 360 wireless controller* (using X-input) and a *Logitech Cordless Rumblepad 2* for the older games (using Direct-Input).

Depending on whether you set the switch to X or D it will be read as *Wireless Gamepad F710* or *Cordless Rumblepad 2*.

Make sure you set the switch to D.When I run `lsusb` in virtualbox, it says:

```
subodh@subodh-virtualbox:/media/sf_workspace$ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 046d:c21f Logitech, Inc. F710 Wireless Gamepad
[XInput Mode]
Bus 001 Device 002: ID 80ee:0021 VirtualBox USB Tablet
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

I don't know why it says *'[XInput Mode]"*? The switch was set to D. After some trial and error I get the virtualbox to detect it as Wireless Rumblepad 2:

```
subodh@subodh-virtualbox:/media/sf_workspace$ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 046d:c219 Logitech, Inc. Cordless RumblePad 2
Bus 001 Device 002: ID 80ee:0021 VirtualBox USB Tablet
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Test the joystick

And I am able to test the buttons by running `sudo jstest /dev/input/jsX` (you may have to try values `0, 1` or `2` for `X`. Try `ls /dev/input`). `jstest` is located in the `joystick` debian package. If it is missing you can install it by running `sudo apt-get install joystick`.

Note: I noticed that the `X` in `jsX` keeps on changing. On the first day it was 1, next day it was 2.

Configuring joystick for use with ROS

See this tutorial [Configuring and Using a Linux-Supported Joystick with ROS](). Following these instructions I had to `sudo chmod a+rw /dev/input/js1`.

Test joystick in ROS

Test the joy ROS package, which is used in the MIT Racecar code.Run these commands:

```
$ roscore
$ rosparam set joy_node/dev "/dev/input/jsX" # in another terminal
$ rosrun joy joy_node
$ rostopic echo joy
```

You should now see messages as you move the joystick controls.

Testing joystick with MIT's code

The commands from the joystick have a higher priority over any other commands (algorithms or safety controller). See the Understanding MIT's Code Architecture. So let's test the joystick interruption in wall follower implementation. Use this in the launch file:

```
<node pkg="joy" type="joy_node" name="joy_node" >
    <param name="dev" value="/dev/input/js1"/>
</node>
```

It was working well. Now you have to hold down the left bumper `LB` button to enable manual control. `LB` is the dead man's switch. The car will stop without it.

However the autonomous navigation is not working. No one is listening to its commands.

Creating udev rules

The f1/10 build instructions helped with this.

When you connect the VESC and Razor to the Odroid, the operating system will assign them device names of the form `/dev/ttyACMx` , where `x` is a number that depends on the order in which they were plugged in. For example, if you plug in the Razor before you plug in the VESC, the Razor will be assigned the name `/dev/ttyACM0` , and the VESC will be assigned `/dev/ttyACM1`.

Also the joystick will be assigned a name `/dev/input/jsx`, where x is one of 0, 1 or 2. The RPLidar will be assigned `/dev/ttyUSBx`.

This is a problem, as the car's ROS configuration scripts need to know which device names the Razor RPLidar, VESC and joystick are assigned, and these can vary every time we reboot the Odroid, depending on the order in which the devices are initialized.

Fortunately, Linux has a utility named `udev` that allows us to assign each device a *"virtual"* name based on its vendor and product IDs. For example, if we plug a USB device in and its vendor ID matches the ID for RPlidar laser scanners (10c4), `udev` could assign the device the name `/dev/sensors/rplidar` instead of the more generic `/dev/ttyUSBx` . This allows our configuration scripts to refer to things like `/dev/sensors/rplidar` and `/dev/sensors/vesc` , which do not depend on the order in which the devices were initialized. We will use udev to assign persistent device names to the LIDAR, VESC, and joypad by creating three configuration files ("rules") in the directory `/etc/udev/rules.d`.

Create a file `/etc/udev/rules.d/99-vesc.rules` and add the following rule：

```
KERNEL=="ttyACM[0-9]*", ACTION=="add", ATTRS{idVendor}=="0483",
ATTRS{idProduct}=="5740", MODE="0666", GROUP="dialout", SYMLINK+="sensors/vesc"
```

Do the same for Razor and RPLidar with the following 2 changes:

1. For RPLidar, Instead of `ACM`, use `USB`
2. Use the appropriate vendor and product IDs (you can find these by doing `lsusb`)

For the joystick create a new `.rules` file and add the following rule:

```
KERNEL=="js[0-9]*", ACTION=="add", ATTRS{idVendor}=="046d",
ATTRS{idProduct}=="c219", SYMLINK+="input/joypad-f710"
```

Creating a wireless hotspot on Odriod

I wasted an entire day setting up a password protected wireless hotspot on the Odroid. It turns out that this is very tricky on Ubuntu. Creating one without a password is very easy. I followed instructions from these 2 sources (there were many others during troubleshooting, but these were the main ones):

1. [HOWTO] Wifi hotspot w/ Network Manager
2. [Odroid Wiki] Wireless Access Point

Finally I created one without the password. Starting/stopping it is easy:

```
sudo nmcli connection up/down Hotspot
```

The default IP address of the hotspot is `10.42.0.1`.

The access point kept disconnecting every 10 seconds. This suggestion helped [SOLVED] WiFi Module 3 randomly disconnect:

```
Unfortunately the WiFi#3 have power management, it will turn the adapter
off when no network activity and turn on again when C-1 make an out going
connection. since my usecase is a headless server it will turn off forever.
```

Open the following file:

```
nano /etc/pm/config.d/blacklist
```

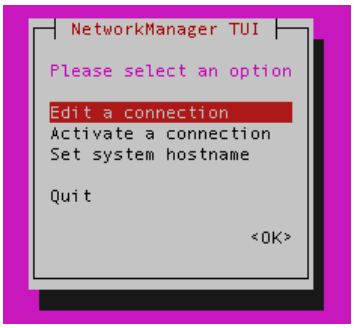
And add this line:

```
HOOK_BLACKLIST="wireless"
```

After all this, I was unable to connect to a wifi from the VNC UI. So I used network manager instead:

```
sudo nmtui
```

This window appears:



Select “Activate a connection” and enter the password.

I was unable to launch roscore, I got this error message:

```
Unable to contact my own server at [ http://odroid:35449/]
```

It also said the network was not properly setup. Then I realized that I had set the IP for odroid in the /etc/hosts file:

```
#192.168.2.2     odroid
```

I commented out the above line and tried again.

Workaround!!

A simple idea - create a mobile hotspot and connect both the laptop and Odroid to the hotspot. SSH works via the hotspot, but not via the router because *access point (AP) isolation* is enabled by default in the DLink router that I have. This works well.

Sidebar - UPenn Build Instructions

This repo mlab-upenn/f1_10_sim has the setup of the MIT racecar simulator. I will try this out. Tried it out and it works. (I remember now, it worked earlier as well, its just that the keyboard teleop needed many other dependencies (turtlebot) to work).

Lastly, System contains code from MIT Racecar that the car would not be able to work without. For instance, System contains ackermann_msgs (for Ackermann steering), racecar (which contains parameters for max speed, sensor IP addresses, and teleoperation), serial (for USB serial communication with VESC), and vesc (written by MIT for VESC to work with the racecar).

Software

Understanding MIT's code architecture

The MIT repositories can be found here https://github.com/mit-racecar/. Here are the various repositories necessary for the car to work:

1. vesc
2. racecar
3. racecar-simulator

vesc

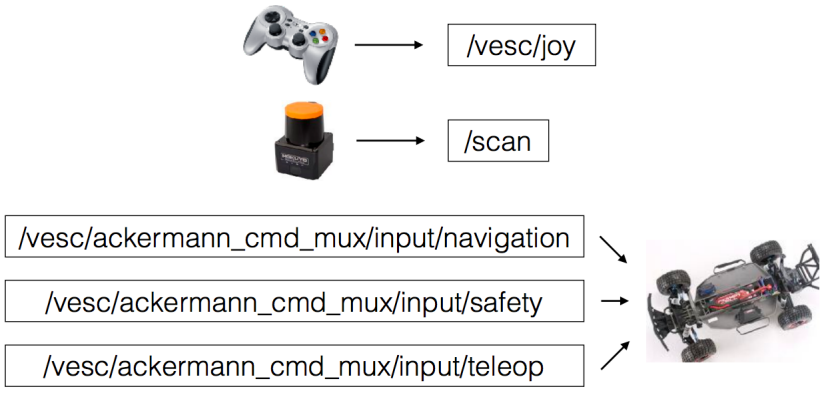
This repository contains code to:

1. ROS driver to communicate with the VESC
2. Publish odometry based on the state of the VESC
3. Subscribe to `AcermannDriveStamped` messages published by the safety/joystick/navigation controller and convert these messages to the format required by VESC

ackermann_cmd_mux

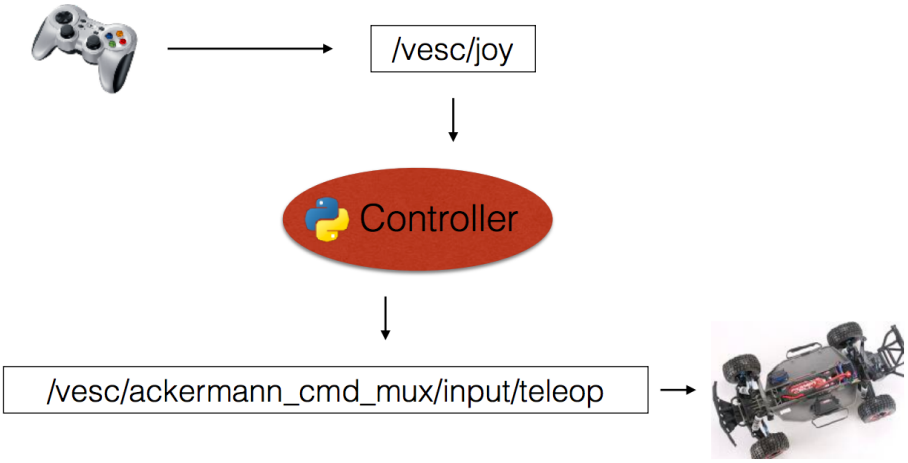
Part of the racecar repository.

A multiplexer for ackermann command velocity inputs. Arbitrates incoming ackermann_cmd messages from several topics, allowing one topic at a time to command the robot, based on priorities. It also deallocates current allowed topic if no messages are received after a configured timeout. All topics, together with their priority and timeout are configured through a YAML file, that can be reload at runtime. Blatantly derived/copied from `yujin_ocs/yocs_cmd_vel_mux`.



The messages on topic `/vesc/joy` and `/scan` are processed to generate input messages for the car's motor controller (VESC).

Structure of Teleop



Commands published to all three /ackermann_cmd_mux topics are read by the chassis, but are followed in the following order of precedence:

`/navigation < /safety < /teleop`

- **Teleop** is short for teleoperation (operation from a distance). The teleop controller gets its commands straight from a human operator via the gamepad, so has the highest precedence
- The **`/safety`** channel will always have a safety node running. This node will stop the car from hitting something ahead of it.
- The **`/navigation`** channel will be published to by your controllers. All autonomous programs should publish to `/nav`

For example, you may write a node to only look at the front scan data and decide if there is an obstacle in the front of the car and stop the car if so.

joy_teleop

Generic joystick teleoperation node. Reads data coming from the joystick and publishes to the `ackermann_cmd_mux/input/teleop` topic with a message of type `AcermannDriveStamped`.

Wall Follower

Now we will build a controller which will follow a wall on its right side. The nodes would be:

1. **Distance finder**: This node will subscribe to the `/scan` topic and publish the distance from the wall on the topic `/pid_error`
2. **Control**: This node will subscribe to the `/pid_error` topic and publish the drive parameters, speed and steering angle, on the `/drive_params` topic
3. **Sim Control**: This node will subscribe to the `/drive_params` topic and publish a message on the `ackermann_cmd_mux/input/teleop`

The code was simpler than expected. To follow the right wall, follow these steps:

1. Subscribe to the laser scan messages
2. Take the right sided scan
3. Use algorithm described by F1/10 tutorials to find the distance from the wall and the orientation off the car
4. Publish `AckermannDriveStamped` message on the navigation channels

Using joystick with autonomous algorithm

I was facing issues using the joystick simultaneously with the autonomous algorithm. I wanted the ability to interrupt for course correction, if required. The topics were set up fine. I was using the Joystick incorrectly.

If you are using the Logitech Gamepad F710 then the way it is setup by default is:

1. teleop requires LB button to be active
2. Autonomous mode requires RB button to be active if the joystick is plugged in. Otherwise it does not require any button to be active

The mistake I was - I was holding the RB and LB buttons down simultaneously. I was under the impression the dead man's switch (LB button) worked in both cases - teleoperation as well as autonomous navigation. I was wrong. It only works for joystick operation.

And the topics are:

1. *joy_teleop* node should publish to `/vesc/low_level/ackermann_cmd_mux/input/teleop`. In the source code it publishes to low_level/ackermann_cmd_mux/input/teleop you can add the prefix `/vesc` by remapping in the launch file.
2. The autonomous algorithm should publish to `/vesc/high_level/ackermann_cmd_mux/input/nav_0` (there are also other topics ending with `nav_x` where `x` is one of *[1,2,3,default]*. The value of `x` determines the priority. See this file high_level_mux.yaml for the order of priorities.

Static transforms issue with bag files: **TODO** [How to save static transforms in bag files?]

Tuning VESC parameters

speed_to_erpm_offset_

I noticed that the odometry published on the topic `/vesc/odom` has non zero values for speed in X (`odom->twist.twist.linear.x`) even if the car is stationary. On examining the code in `vesc_to_odom.cpp`, this line is the source of the problem:

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

`speed_to_erpm_offset_` was set to zero. The state here is the VESC state reported on the topic `/vesc/sensors/core`. On echoing messages from this topic I realized that the `state.speed` is reported to be in the range of 39 to 120. This is the ERPM. This means I have to set some value for `speed_to_erpm_offset_`. On reinspection it appears that the `state.speed` value oscillates between 37 and 140. So I should probably set the value for `speed_to_erpm_offset_` as (37+140)/2 = ~90.

On fine tuning I arrived at a value of 70. How? The value of `odom->pose.pose.position.x` should be very close to zero for a stationary car. 90 led to the position slowly going backwards. 70 worked well.

However, a couple of days later, a value of 91.5 worked well for this. Did this have anything to do with the replaced servo motor?

speed_to_erpm_gain_

This was relatively simpler. I moved the car backward and forward by approximately 2m and I observed the value of `odom->pose.pose.position.x`. Gain of 1900 (was set to ~3000 by default) worked well for me.

A couple of days later I had to change this value to 2100. I don't know why the value had to be changed. The battery was fully charged in both cases.

wheelbase_

The wheelbase was set to 0.25m in vesc.yaml. The value for my car is 0.34 m. This is probably the reason why the turns are overestimated by /vesc/odom. The wheelbase is related to the yaw by the following equations:

```
angular_velocity = current_speed * tan(current_steering_angle)/wheelbase_;
yaw_ += angular_velocity * dt.toSec();
```

So an increased wheelbase will result in a lower yaw.

To verify this I don't want to start the car again. I would like to use the existing bag file. So I did the following:

```
<group ns="vesc">
```

```
  <arg name="vesc_config" default="racecar/config/racecar-v2/vesc.yaml"/>
  <rosparam file="$(arg vesc_config)" command="load" />
  <node pkg="vesc_ackermann" type="vesc_to_odom_node"
name="vesc_to_odom2"/>
</group>
```

I launched the vesc_to_odom node again and then replayed the bag file with a remapped topic:

```
<!-- the odometry is incorrect due to the incorrect wheel_base -->
<!-- this is fixed by relaunching the vesc_to_odom node with the correct
parameters above -->
<node pkg="rosbag" type="play" name="player" output="screen"
      args="--clock --pause rosbags/data6.bag">
    <remap from="/vesc/odom" to="/vesc/old_odom"/>
</node>
```

steering_angle_to_servo_gain & steering_to_servo_offset

The relationship between the steering angle and angular velocity is:

```
steering_angle = (servoCmd.data -
steeringToServoOffset)/steeringToservoGain
angular_velocity = speed * tan(steering_angle) / wheelbase
```

The default values worked well with the Traxxas 2075 servo motor. On changing the servo motor I had to tune the `steering_angle_to_servo_gain` from `-1.2135` to `-1.0`.

Servo Problem

I noticed that while performing teleoperation, the servo motor would stop following commands sometimes. After some time gap of not following commands it would start working again. Sometimes it would just turn in one direction and not the other. Also the 3 cables connecting the servo to the VESC would heat up.

Now there could be 3 sources of this problem:

1. Racecar software
2. VESC
3. Servo motor

The servo motor has 3 wires:

1. Black (GND)
2. Red (VCC)
3. White (signal)

It connects to the brown (GND), red (VCC) and orange (signal) cables of the VESC respectively.

On visualizing the node graph I noticed that the servo commands are sent on the topic /vesc/sensors/servo_position_command to the vesc_driver node, which then sends control commands to the motor. On echoing messages from this topic I noticed that it sends the following data:

1. 0.85 for right
2. 0.15 for left

3. 0.5304 for center

This happens in both cases - when the servo motor is and is not working. So the racecar software is ruled out as a cause of this problem.

I tried to measure the voltage on the signal and VCC wires when sending control commands to turn the motor via the VESC and the RF remote control. These are the results:

	VESC			RF Remote	
Command	**Signal**	**VCC**		**Signal**	**VCC**
Center	0.25	5.17		0.31	5.98
Left	0.19	5.17		0.21	5.98
Right	0.30	5.17		0.41	5.98

The values sent by the VESC are comparable to the RC and look alright. Surprisingly the servo motor stopped working after some time when using the RF remote. And the servo cables heated up. So one can conclude that the VESC is not at fault over here.

Hence the servo motor seems to have malfunctioned. According to this video how to fix your traxxas 2075 servo, there could be a short on the servo's circuit board. I opened the servo lid and I saw that the 3 wires were connected to a circuit board. Some of the leads on the circuit board showed signs of overheating.

There could be a problem with the fuse, as shown in this video How to repair a Traxxas 2075 servo (fuse). The fuse looked fine to me.

I tried checking if there is a problem with the gears, by opening up the servo as shown in this video Rebuilding Traxxas 2075 Servo. The gears looked fine to me.

Traxxas 2075 Servo Problems

These posts point to a problem with the Traxxas 2075 servos:

1. ESC or servo problem - intermittently switching off
2. What's the concensus on the steering servo problem?

Replacing the servo motor seems to be the solution.

Replacing Servo

But which one should I buy? Traxxas 2075 servo has the following specs:

1. Torque: @6.0V 9.00 kg-cm (125 oz-in)
2. Speed: @6.0V 0.17sec/60 degrees
3. Servo Size: 55mm x 20 x 38mm
4. Weight: 45.0 g
5. Modulation: Digital

The stock motor is $40 and with import duties and taxes it would cost Rs 4000 in India. Its available for Rs 4567 on amazon.in. Ajmal, from HobbyCentral, said he will sell a used one for 30% discount to the

US$ price (converted to INR). So basically about Rs 2100. That is too expensive. There are cheaper alternatives for about Rs 400-500.

I found comparable servo motors here:

1. [robu.in] TowerPro MG995 Servo High-Speed Digital Metal Gear Servo Motor with CNC Aluminum Steering Servo Horn Arm- Good Quality
2. [amazon.in] TowerPro MG996R High Torque Metal Gear Servo motor for Robotics, Arduino(15KG cm)

They have these specs:

1. Torque: @4.8V - 13kg/cm, @6.0V - 15kg/cm
2. Speed: 4.8V - 0.17sec/60 degrees, @6.0V - 0.14sec/60 degrees
3. Servo Size: 41mm x 20 x 43mm
4. Weight: 55g
5. Modulation: Digital

The torque is a bit higher but this is as close as I could get. The one on robu was out of stock. So I ordered the one from Amazon. **Remember that these are copies and not original motors**. So I don't know if I can expect a good performance from these. This review TowerPro MG995 metal gear servo (clone) mini-review says that these motors are very slow. Fingers crossed.

2 days later the new servo arrived! And it works as expected. I tried it with the stock Traxxas ESC. From the specs it seemed as if the length of the new servo would be shorter. It is an exact fit for the Traxxas servo.

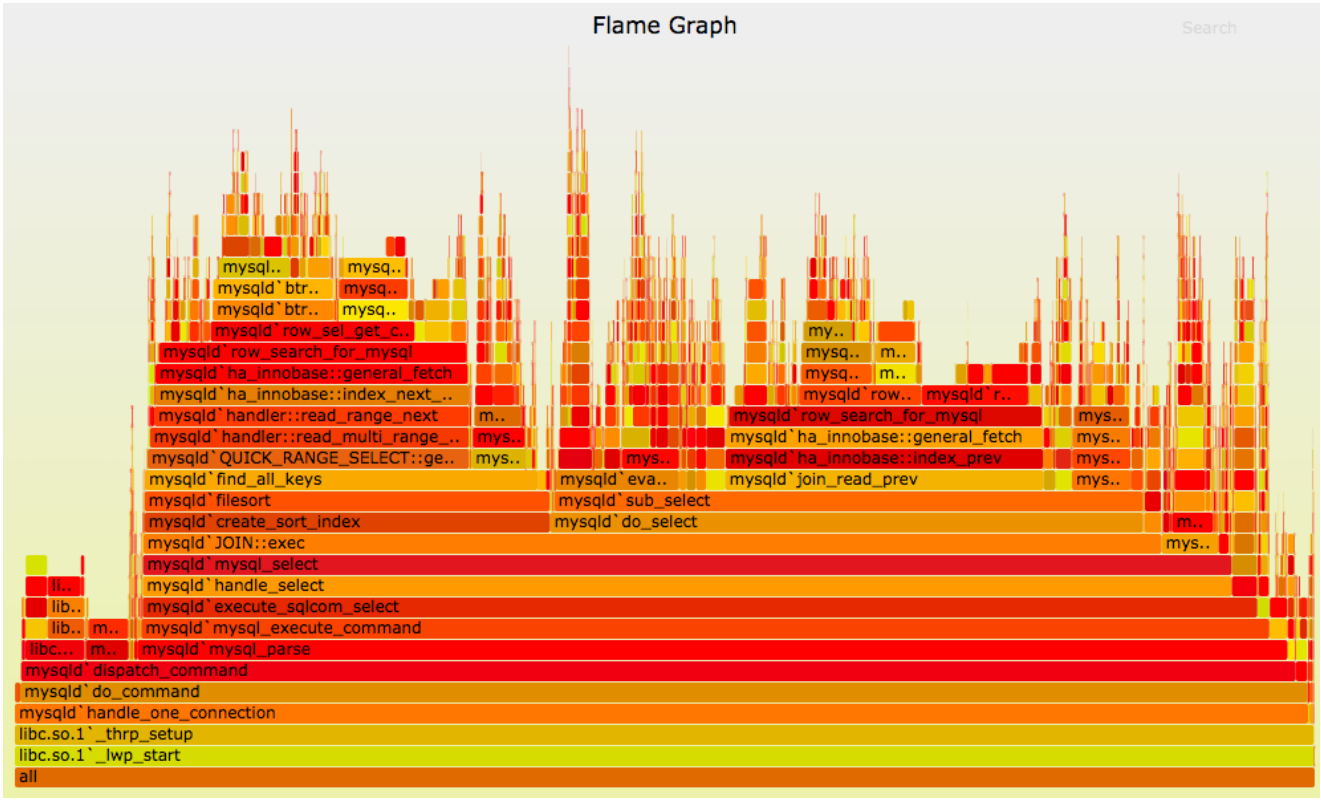
If a cheap servo works then I wonder if a cheaper BLDC motor and chassis will work. After all this platform is designed for research, not for bashing or racing. This is worth exploring.

Profiling tools in ROS

FlameGraphs

Here is a link from the MIT Racecar page [Profiling tools in ROS](#).

`FlameGraph` were built by Brendan Gregg, an industry expert in computing performance and cloud computing. He is a senior performance architect at Netflix, where he does performance design, evaluation, analysis, and tuning.



Interpreting FlameGraph

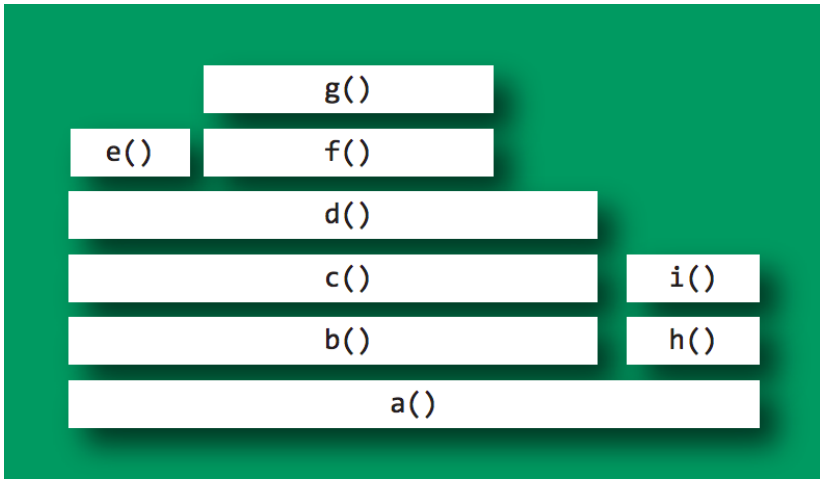
This visualization is fully explained in the article [The Flame Graph](#).

- The top edge of the flame graph shows the function that was running on the CPU when the stack trace was collected. For CPU profiles, this is the function that is directly consuming CPU cycles. For other profile types, this is the function that led directly to the instrumented event.
- Look for large plateaus along the top edge, as these show a single stack trace was frequently present in the profile. For CPU profiles, this means a single function was frequently running on-CPU.
- Reading top down shows ancestry. A function was called by its parent, which is shown directly below it.
- Reading bottom up shows code flow and the bigger picture. A function calls any child functions shown above it, which, in turn, call functions shown above them.

- The widths of function boxes can be directly compared: wider boxes mean a greater presence in the profile and are the most important to understand first.
- For CPU profiles that employ timed sampling of stack traces, if a function box is wider than another, this may be because it consumes more CPU per function call or because the function was simply called more often. The function-call count is not shown or known via sampling

Example

Have a look at this example FlameGraph:



Imagine this is visualizing a CPU profile, collected using timed samples of stack traces (as is typical).

- The top edge shows that function g() is on-CPU the most; d() is wider, but its exposed top edge is on-CPU the least.
- Functions including b() and c() do not appear to have been sampled on-CPU directly; rather, their child functions were running.
- Functions beneath g() show its ancestry: g() was called by f(), which was called by d(), and so on.
- Visually comparing the widths of functions b() and h() shows that the b() code path was on-CPU about four times. more than h(). The actual functions on-CPU in each case were their children.
- A major fork in the code paths is visible where a() calls b() and h(). Understanding why the code does this may be a major clue to its logical organization. This may be the result of a conditional (if conditional, call b(), else call h()) or a logical grouping of stages (where a() is processed in two parts: b() and h())

Generating FlameGraph

This is a 3 step process:

1. Use a profiler to gather stack traces (e.g., Linux perf_events, DTrace, Xperf).
2. Convert the profiler output into the “folded” intermediate format. Various programs are included with the FlameGraph software to handle different profilers; the program names begin with “`stackcollapse`”.

3. Generate the flame graph using `flamegraph.pl`. This reads the previous folded format and converts it to an SVG flame graph with embedded JavaScript.

The folded stack-trace format puts stack traces on a single line, with functions separated by semicolons, followed by a space and then a count. Example, have a look at the profile with these 3 stack traces:

```
func_c
func_b
func_a
start_thread

func_d
func_a
start_thread

func_d
func_a
start_thread
```

In the folded format this becomes:

```
start_thread;func_a;func_b;func_c 1
start_thread;func_a;func_d 2
```

Modifications for launching ROS nodes

This is described here [Creating Flame Graphs in ROS](#).

The primary consideration is how to load params which would be typically be provided natively in ROS when you use roslaunch. This is how we do it, you might be able to discover a better method. This should work both with direct invocation and with the standard roslaunch method.

Modify your code to accept a "`-config`" argument which allows you to specify a yaml file.

Python code: `particle_filter.py`

```
import argparse
parser = argparse.ArgumentParser(description='Particle filter.')
parser.add_argument('--config', help='Path to yaml file containing config parameters. \
Helpful for calling node directly with Python for profiling.')

def load_params_from_yaml(fp):
    from yaml import load
    with open(fp, 'r') as infile:
        yaml_data = load(infile)
        for param in yaml_data:
            print "param:", param, ":", yaml_data[param]
            rospy.set_param("~"+param, yaml_data[param])

if __name__=="__main__":
    rospy.init_node("particle_filter")
    args,_ = parser.parse_known_args()
    if args.config:
        load_params_from_yaml(args.config)
    pf = ParticleFilter()
    rospy.spin()
```

YAML config parameters: params.yaml

```
scan_topic: "/scan"
odometry_topic: "/vesc/odom"
angle_step: 18
max_particles: 5000
range_method: "pcddt"
theta_discretization: 108
max_range: 10
# etc...
```

Install Python FlameGraph

```
pip install git+https://github.com/evanhempel/python-flamegraph.git
```

Example command line invocation

```
# 0.001 is the sampling rate (1000Hz here)
# out.log is the output file
python -m flamegraph -i 0.001 -o out.log ./src/particle_filter.py --config
./params.yaml
```

(optional) Filter the profiling information to contain only the interesting bits

If you do this with ROS, there will be a bunch of extra stack information in there which is not relevant to you. You can filter the log file to contain only the information you care about with grep. It might help to look at the unfiltered version to identify the functions you care about.

Example command line invocation:

```
# filter only stack frames above function called "/scan`update"
grep scan\`update out.log > out_filtered.log
```

Generate flame graphs with flamegraph.pl

First download the necessary Perl script from here: [brendangregg:FlameGraph/flamegraph.pl]

Example command line invocation:

```
./flamegraph.pl --title "Your title here" out.log > flames.svg
./flamegraph.pl --title "Your title here" out_filtered.log > flames_filtered.svg
```

Now open the svg file in your browser.

rosprofiler and rqt_graphprofiler

These repositories have not been updated since 2014 - [osrf/rosprofiler] and [osrf/rqt_graphprofiler].

Htop, Top and Sysprof

Have a look at the answers to this question - [How to see nodes CPU usage in ROS ?]

```
Since normal nodes are started as distinct processes a simple top or htop
should do the trick. Sadly i don't know of a way to see the load if you are
working with nodelets that reside within the same process. Maybe someone
```

else can shed some light, if this is even possible? (Other than starting the nodelets in different managers and making them distinct processes?)

The best place to start is top or htop. If you want more information then I recommend using sysprof so that you can see which function calls are taking the most time and figure out where optimizations can be made. That would even help with nodelets.

Builtin ROS tools

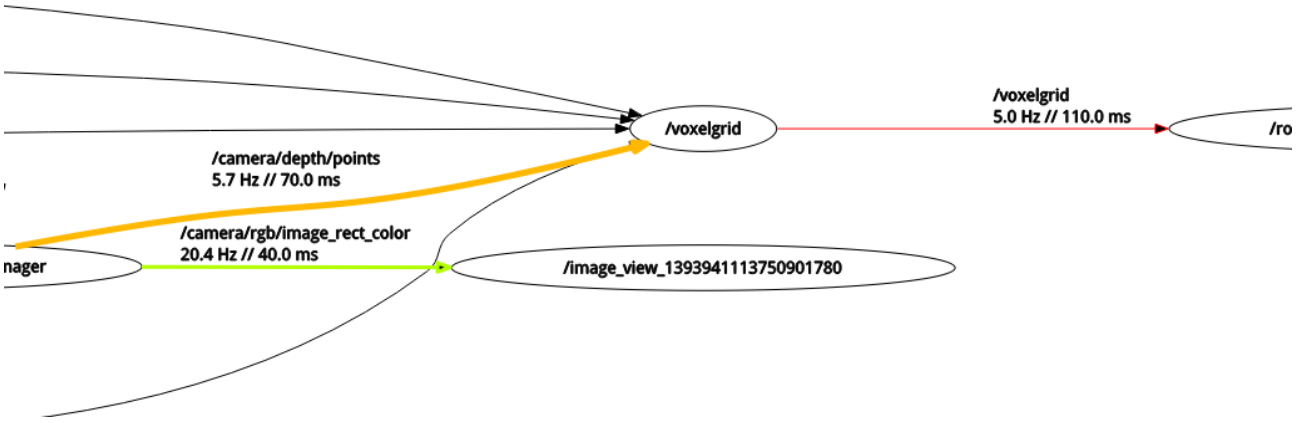
See the answer to this question - How to profile message activity?

Since ROS Indigo there is builtin support for topic statistics in ROS. Based on the documentation, these are the metrics that are collected:

1. Period of messages by all publishers (average, maximum, standard deviation)
2. Age of messages, based on header timestamp (average, maximum, standard deviation)
3. Number of dropped messages
4. Traffic volume (Bytes)

To enable this feature, the **/enable_statistics** parameter needs to be set to **true**.

Checkout the documentation - Topic_statistics. This is an example of rqt_graph with statistics measurement enabled:



Legend:

1. **"5.0 Hz":** indicates the observed publish frequency.
2. **// "110.0ms":** indicates the average age of the message (might be very large when replaying bag files without simtime).
3. **Line color:** red marks a high average age, green a young age. The limits are normed to the ages observed in the ROS graph.
4. **Line thickness:** A thick line means the connection has a large throughput in Bytes/s (e.g. an unfiltered PointCloud).

Website Resources

Good landing pages

1. http://moveit.ros.org/
2. https://www.expensetron.com/
3. https://vecnarobotics.com/ (background video)
4. https://fetchrobotics.com/

Compressed Directional Distance Transform (CDDT)

CDDT: Fast Approximate 2D Ray Casting for Accelerated Localization

The main contribution of this paper is a new acceleration data structure, called the Compressed Directional Distance Transform (CDDT) which allows near constant time two dimensional ray casting queries for an occupancy grid map of fixed size. The authors provide an open-source implementation of CDDT and the other methods evaluated in a library called RangeLibc.

To combat the computational challenges of ray casting while localizing in a two-dimensional map, Fox et al. [3] suggest the use of a large three-dimensional lookup table (LUT) to store expected ranges for each discrete (x, y, θ) state. While this is simple to implement and does result in large speed improvements as compared to ray casting, it can be prohibitively memory intensive for large maps and/or resource constrained systems. In a 2000 by 2000 occupancy map, storing ranges for 200 discrete directions would require over 1.5GB. While this memory requirement may be acceptable in many cases, it scales with the area of the map - a 4000 by 4000 map would require over 6GB for the same angular discretization, which is larger than the random-access memory on-board many mobile robots.

Camera to ground frame transformation

Going through this research paper - [AutoRally: An open platform for aggressive autonomous driving](), I noticed these words:

`For training, a survey of the track boundaries,` **`calibrated inertial measurement unit sensor to camera transform,`** `and state estimate are used to automatically label image pixels.`

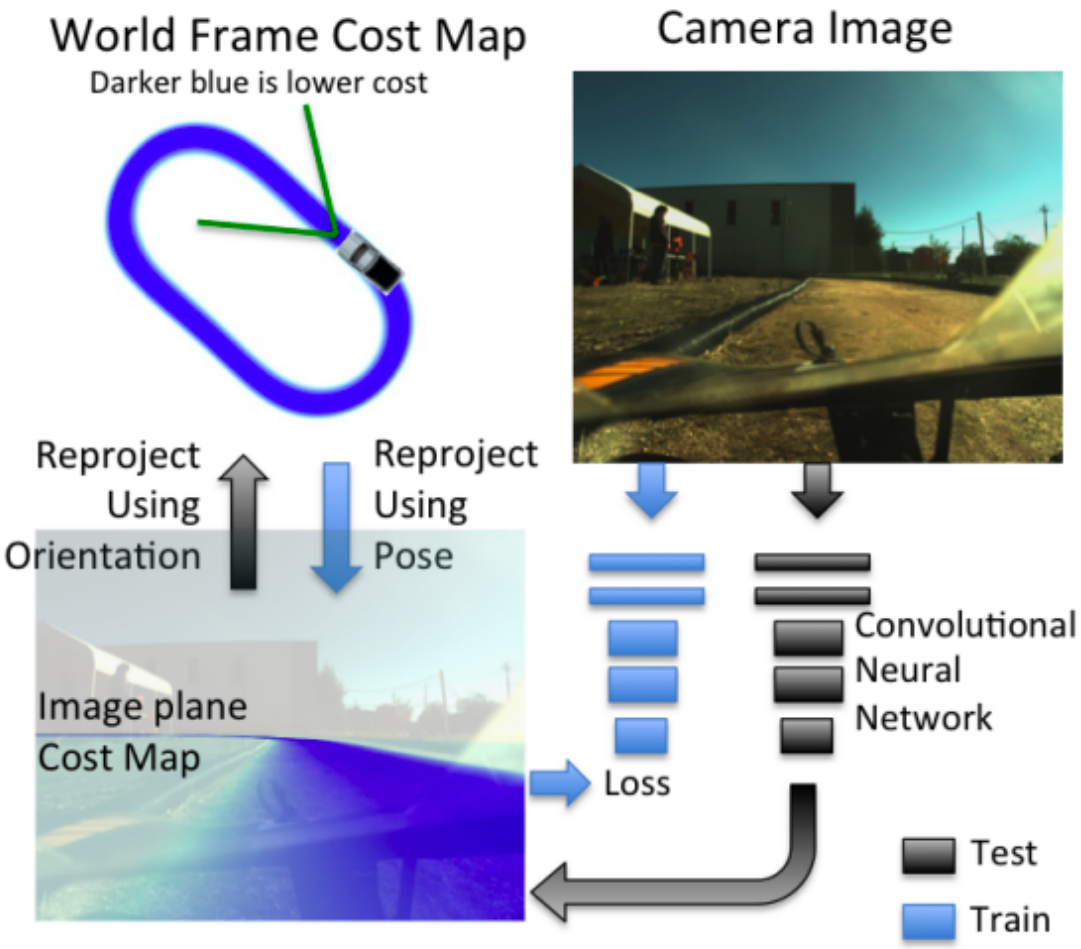


Figure 16: Convolutional neural network pipeline for training and testing pixel-wise image labeling of track and not track using images from onboard AutoRally. For training, a survey of the track boundaries, calibrated inertial measurement unit sensor to camera transform, and state estimate are used to automatically label image pixels. For testing, the network labels each pixel of an input image as track or not track.

Using the method described in the paper, all points on the image plane

for a given image can be projected to the ground plane and given a ground truth label.

So to break it down, the images were labeled using:

1. Precomputed GPS survey of the test track
2. Output from the state estimator,
3. IMU to perform camera calibration.

The state estimator frame is at the IMU, so a homography matrix must be computed that transforms the surveyed track map from world coordinates to image plane coordinates using the calibrated transformation between the IMU and the camera:

$$H = k * T^{car}_{im} * T^{world}_{car}$$

where T^{world}_{car} is the position of the car in world coordinates, T^{car}_{im} is the transformation between the IMU and camera reference frames, and k is the matrix of camera intrinsics. Given this

mapping, points in the ground coordinate frame can be projected into the image:

$$p_{im} = H * p_{world}$$

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix}$$

Where p_{im} and p_{world} are homogeneous points. **Using this method, all points on the image plane for a given image can be projected to the ground plane and given a ground truth label.** More information on this is there in these lectures:

1. Lecture 12: Camera Projection
2. Lecture 13: Camera Projection II

These lectures have a very good explanation of the derivation of rotation and translation equations for projection matrix, and getting to 2D pixel coordinates from 3D world coordinates. Also on affine and projective transformations.

Sidebar: Homogeneous Coordinates

It is important to understand homogeneous coordinates and affine transformations. Here are 2 good resources:

1. What Homogeneous Coordinates Mean
2. Homogeneous Coordinates (this is a good repository of tutorials)
3. University of Washington: Cameras, Projection, and Image Formation
4. Camera Projection

Estimation

This research paper deals with the following estimation problems:

1. Vehicle parameters for motion model
2. Vehicle state (orientation and position)

State Estimation

From the same paper, this is what they say about state estimation:

GPS is inherently low rate and lacks orientation information. IMU measurements are relatively high rate but do not directly provide heading or linear velocity information. By combining the time-synchronized signals from these two sensors, a very accurate and high rate estimate of position, velocity, and orientation can be obtained. This state information is sufficient for many advanced control systems.

Online, vehicle state is estimated using a factor graph-based optimization framework with GPS and IMU data and a cost map of the terrain, similar to a traversability grid, is generated from monocular camera images for use in a stochastic MPC framework.

Parameter Estimation

Parameter estimation deals with estimating the motion model (vehicle model) of the car.

Parameter estimation is an essential part of controller design, especially for **model-based controllers such as MPC**, which rely on accurate dynamics models for motion prediction. This section details the offline and online estimation performed with the AutoRally platform. Offline, parameters were estimated to determine the platform moments of inertia (MOI) using the bifilar pendulum method. Three different vehicle models of **increasing**

fidelity are presented. While the higher fidelity models can be used to more accurately predict vehicle motion, the model parameters can be significantly more difficult to estimate, and computationally expensive to compute. A joint-state unscented Kalman filter (JS-UKF) was used to find the parameters of single-track and double-track vehicle models with a realistic tire forces model. An 11 DOF full vehicle model was estimated using an adaptive limited memory unscented Kalman filter (ALMUKF).