Navigation using a topological map with an autonomous vehicle

Jin PAN

UTBM, Computer Science engineering student
email: jin.pan@utbm.fr

June 20, 2018

Abstract

Autonomous vehicle navigation gains increasing importance in various growing application areas. In this article we describe a project it navigates the vehicle autonomously to its destination. This project use Openstreetmap(OSM) to help define a topological map, so we can decide the path by listing the waypoint on the map, then make a link between the OSM and the SLAM based global map to feed ROS navigation Stack.

Introduction

An autonomous vehicle, known also as a self-driving vehicle, has became a concrete reality in the past dozen years, and may pave the way for future systems where computers take over the human drivers.

As an autonomous vehicle, it is capable of sensing its location, navigating its way toward its destination and avoiding obstacles without human input. Autonomous vehicles sense their surroundings with such techniques as radar, lidar, GPS, computer vision etc,. Advanced control systems interpret information from sensors to identify appropriate navigation paths, as well as obstacles.

For any autonomous vehicle, navigation is a key aspect of design and also a highly complex task. In general, the key technologies of automatic driving can be divided into four parts: environmental awareness, behavioral decision, path planning and motion control. The navigation task can be defined as the combination of three basic

competencies: localization, path planning and vehicle control. Localization denotes the robot's ability to determine its own position and orientation (pose) within a global reference frame. Path planning defines the computation of an adequate sequence of motion commands to reach the desired destination from the current robot position. The potential application areas of the autonomous navigation of mobile robots include automatic driving, guidance for the blind and disabled, exploration of Dangerous regions, transporting objects in factory or office environments, collecting geographical information in unknown terrains like unmanned exploration of a new planetary surface, etc.

This article will concentrate on one of many navigation methods, which use a topological map. Firstly, it will present the topological map and navigation stack. Furthermore, this article will specifically introduce the principle of topological navigation. This article will also discuss the application in the autonomous vehicle, the limitations and constraints that might be encountered while using topological navigation.

Open street map for ROS navigation

Localization and path planning are two of the most important components in autonomous robots. ROS provides a great foundation for working with maps and path planning. This project uses the geodata from the OpenStreetMap for the navigation and we could visualize the osm data using rviz.

OpenStreetMap (OSM) is a collaborative project which aims to create a free to use and editable map of the world. Different from commercial map distributors like Google, OSM is public domain and created by volunteers performing systematic ground surveys with a handheld GPS receiver.

Since its creation in 2004, Openstreetmap has grown to over 2 million registered users, who can collect data using manual survey, GPS devices, aerial photography, and other free sources. Rather than the map itself, the data generated by the OpenStreetMap project is considered its primary output. OSM provides detailed information paths, buildings and other landmarks, which plays a crucial role in self-driving car navigation and localization.

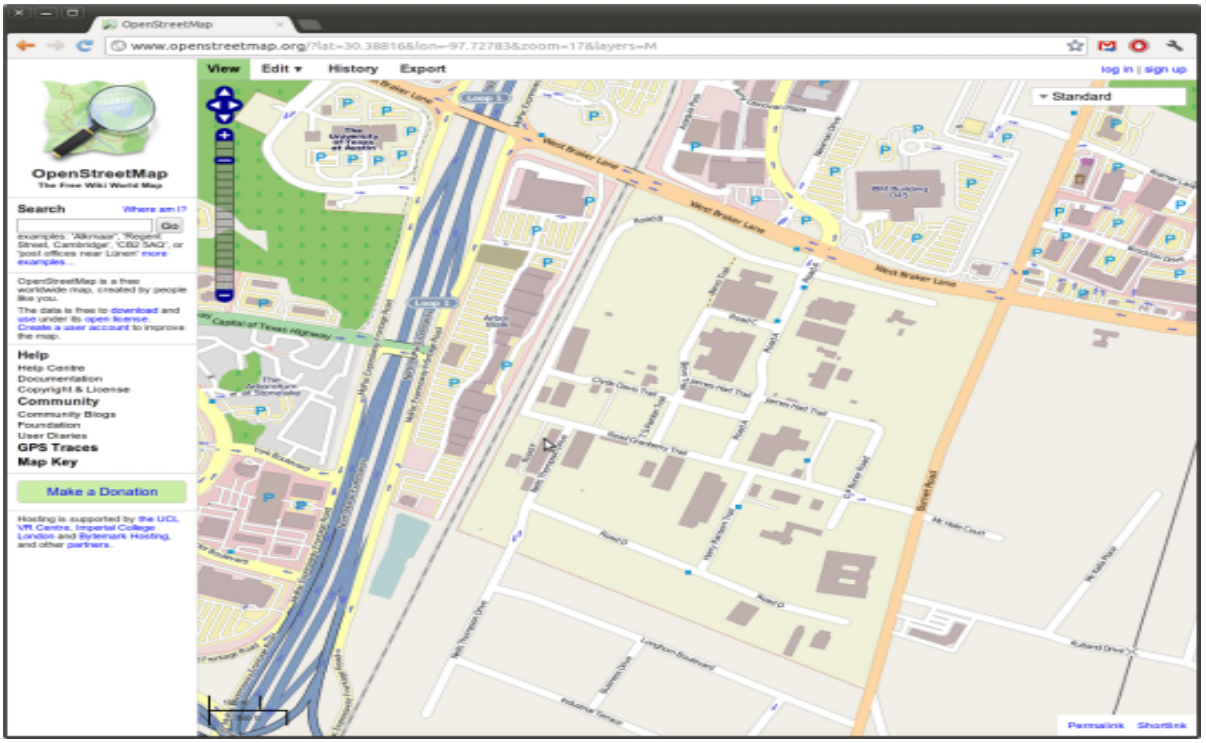

Figure : Map selected from OSM

OSM allows the user to export a map of an selected area in .osm format, the map is composed by a serie of node, which contain id, location, type etc,.

In order to visualize the data from OSM, we need a ROS node that can read the .xml file and publish a bunch of markers in rviz. OSM_cartography is an open-source package that accomplishes just that. for this task, we need realize a transform tree to make a link between \map and \local_map. Once the .osm file is visualized using rviz, it looks something like the figure at the next page.

The red, green and purple lines represent different type of pathways, while the blue lines represent buildings or other structures. If you look very closely, you will find yellow marks, which are waypoints on a path. This kind of visualization is quite useful when it comes to localization and path planning.

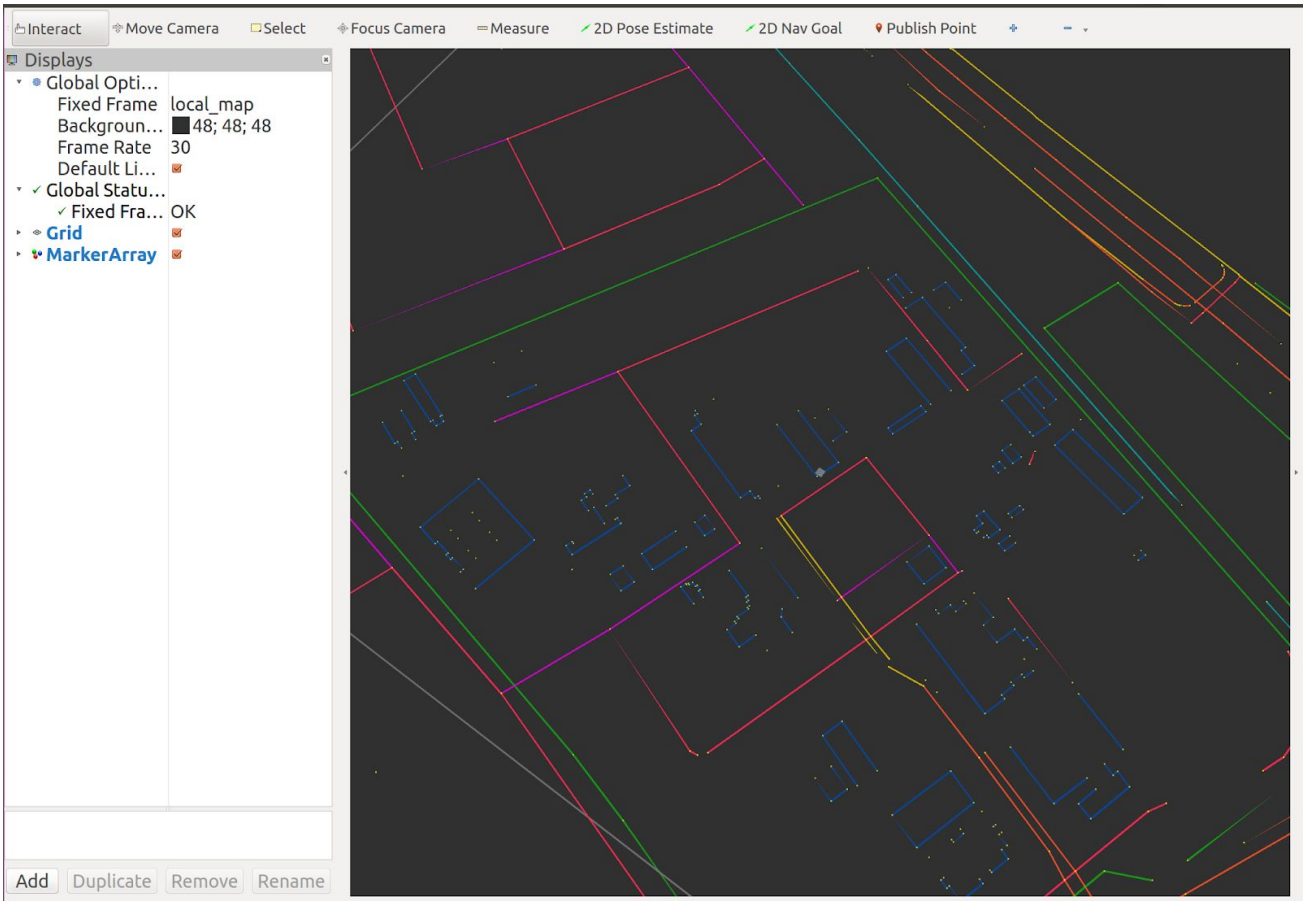

Figure : OSM map view in RVIZ

Topological map

For behavior-based robots, using topological map to navigate is simpler and more natural. When we show someone else the way, we use often the instructions such as “go along this street, turn left at the first intersection, and then enter the third building on your left” naturally, we also hope that the vehicles or the robots can understand the instructions like this. Even without a complete map, as long as the robot knows “intersection” “building”, it is enough to complete the navigation task.

In order to develop an autonomous vehicle able to navigate through environment composed by streets and highways, it must be assumed that the robot know its approximate position, environment map and the path to be followed (origin/way points/destination), in this way, navigation task consists on a predefined path.

In a sense, topological map abstracts the urban environment, it serves as the basis for the autonomous navigation of intelligent vehicles and provides the paths for global planning. As we don’t need a very detailed environment map (metric map), topological maps provide a graph to represent the main elements, a simpler path representation consist on roads as edges, junctions, parking lots, etc. as nodes, and these informations are provided by the OpenStreetMap (OSM) city area topological map.

As the path is predefined, the main objective of navigation is to detect current node in a topological map and to know which next point should pass by, being useful to autonomously decide when and how to proceed in order to go straight, turn left or right.

We consider that topological map is a list of nodes (Way Points) that can be defined in a YAML format file, the file looks like this:

```yaml
- node:                                  # A node
    name: WayPoint1                      # Node Name
    pointset: MAP_NAME                   # Map name
    edges:                               # List of connections from this node
    - action: planner                    # Action connecting this node to the next
      edge_id: WayPoint1_WayPoint2       # Name of the connection
      node: WayPoint2                    # Name of the node that is connected by this edge
    pose:                                # Position of the node
      orientation:
        w: 0.776898920536
        x: 4.04172055823e-08
        y: 4.60677940239e-09
        z: -0.629625380039
      position:
        x: -1.77065169811
        y: -3.21753334999
        z: 0.00247192382812
    verts:                               # Vertices of the influence area
    - x: 0.689999997616                  # (area around the node considered to be the same location)
      y: 0.287000000477
    - x: 0.287000000477
      y: 0.689999997616
    - x: -0.287000000477
      y: 0.689999997616
    - x: -0.689999997616
      y: 0.287000000477
    - x: -0.689999997616
      y: -0.287000000477
    - x: -0.287000000477
      y: -0.689999997616
    - x: 0.287000000477
      y: -0.689999997616
    - x: 0.689999997616
      y: -0.287000000477
- node:
    edges:
    - action: planner
      edge_id: WayPoint2_WayPoint1
      node: WayPoint1
    - action: planner
      edge_id: WayPoint2_WayPoint3
      node: WayPoint3
    name: WayPoint2
    pointset: INB3123
    pose:
      orientation:
        w: 0.384249478579
```

Navigation

Navigation is a huge challenge we must overcome when designing an autonomous vehicle. Navigation describes how an autonomous vehicle intelligently moves and interacts with its environment. There are two main focuses when designing a navigation system for an autonomous vehicle: path planning and obstacle avoidance. Navigation function can be described by the following four questions: Where will I go? This problem is usually done by people or task planners. What is the best path? This is a question about path planning and is the most concerned point in navigation problems. Where have I been to? The map is important for the autonomous vehicle or robot to analysis the environment. Where am I now? In order to track the path or build a map, the vehicle must know where it is, the so-called localization. Path planning attempts to generate an intelligent movement scheme for the vehicle. Obstacle avoidance attempts to account for objects the vehicle may encounter while traversing its path. This makes it more capable to handle the real world situations and environments where there are many unknowns. When driving a car, we not only have to avoid known obstacles, but it also have to avoid unknown obstacles like wild animals.

a. **Navigation stack**

The main objective of the Navigation Stack is to move a vehicle from a position A to a position B, ensuring it won't crash against obstacles, or get lost in the process. The Navigation Stack is a set of ROS nodes and algorithms which are used to autonomously move a vehicle from one point to another, avoiding all obstacles the vehicle could find its way. In order to do this, the Navigation Stack will take the current location of the vehicle as input, the destination the vehicle wants to go (goal pose), the Waypoints the vehicle should pass by, the Odometry data(”Odom” topic) of the Robot (wheel encoders, IMU, GPS...) and data from the sensors. In exchange, it will output the velocity commands (“cmd_vel” topic) in this form below:

x, y (linear velocity)
z (angular velocity)

with them the vehicle is controlled to move to the specified position.

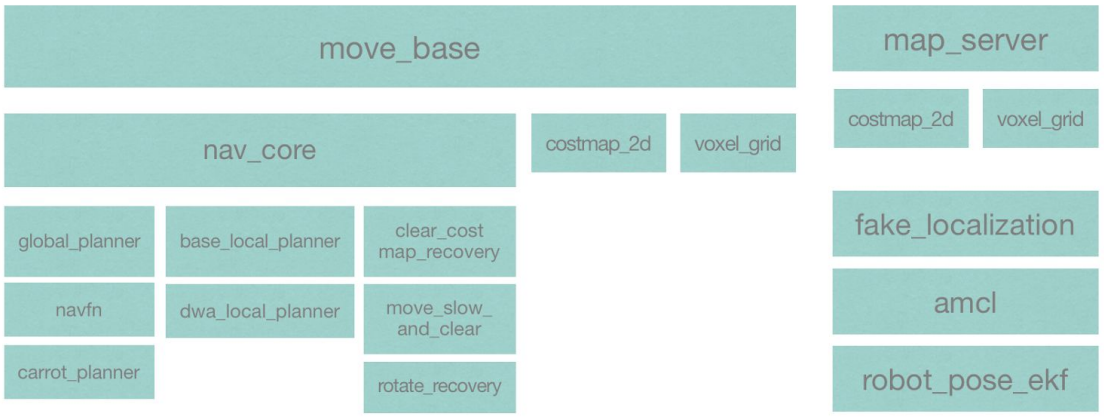

Figure : Navigation Stack source package structure

According to the shown diagram, the navigation stack can be divided into three parts.

move_base provides a basic framework, which contains the configuration, operation, and interaction interfaces of ROS navigation. Its main tasks are as follows: Maintain a global map, maintain a local map, maintain a global path planner to complete the global path planning task and maintain a local path planner to complete the local path planning task.

nav_core provides the interfaces of global path planner and local path planner, so that we could integrate different algorithms with.

carrot_planner is the simplest global path planner, it generate a path of straight line between current point and the target point.

global_planner and **navfn** are basicly doing the same thing. They are all to realize the global path planning between the target point and the current point. The are both integrated with Dijkstra algorithm and A* algorithm.

base_local_planner searches through the map data for multiple paths to the target, uses some evaluation criteria (whether it will hit obstacles, required time, etc.) to select the optimal path and calculate the required real-time velocity(x,y,z). There are two local path planning algorithms: Trajectory Rollout and Dynamic Window Approach (DWA) algorithm. DWA is implemented in the **dwa_local_planner**.

rotate_recovery and **clear_costmap_recovery,** both of these packages are inherited from the recovery_behavior class. They are used when the vehicle finds no

way to go, then moves around and updates the surrounding obstacle information to see if there is dynamic obstacles move away then find a way to continue.

map server give a management of map, which is used to read and write maps and publish map messages for the subscribers of other function packages.

costmap_2d can be seen as an input processor for navigation. Input data could be very different according to the different sensors. Through the costmap_2d, different data are processed into a unified format: a grid map, the weights are processed by probabilistic methods, representing obstacles, unknown and safe areas. The generated costmap is the planner's input.

fake_localization provides an implementation of localization.

robot_pose_ekf is mainly modify the vehicle’s mileage value according to the sensor informations.

amcl is used to estimate the location of the vehicle on the map using the odometer value and the map information.

b. **Using OSM for navigation**

As mentioned previously, the actual control of vehicle is done by move_base, we can see in the figure below the inputs and outputs of move_base. To make it work, we have to build well these inputs and outputs.

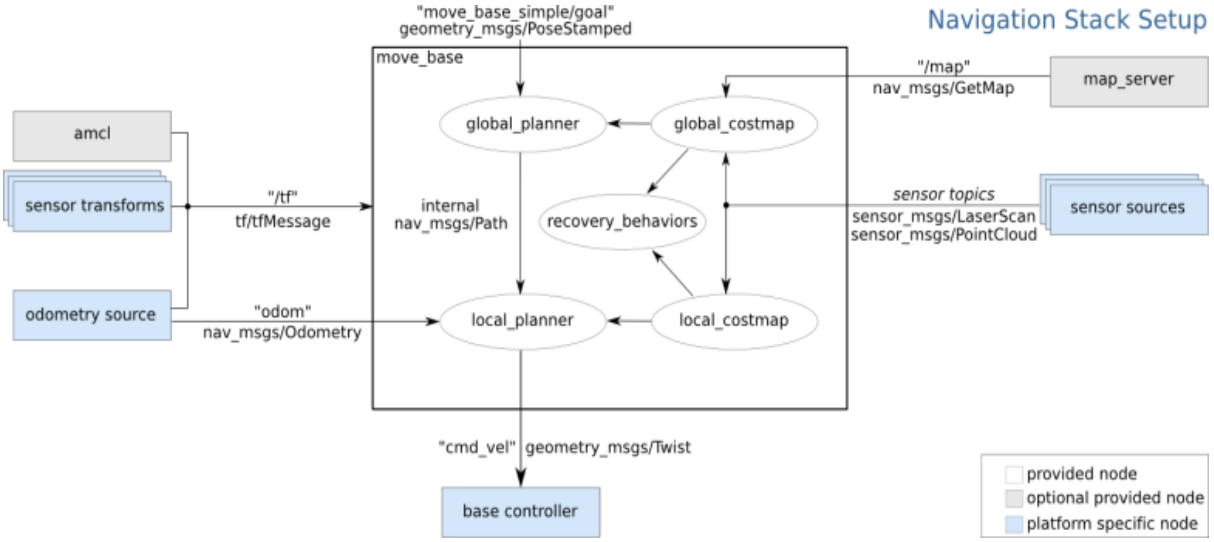

Figure : Navigation stack

Necessary inputs:

Goal : Expected target position on the map.

Tf : Transform between coordinate systems. (/map → /local map, /map frame → /odom frame, /odom frame → /base_link frame)

Odom: Calculate from the left and right wheel speeds of the vehicle a estimation of a position and velocity.

LaserScan: Laser sensor information for positioning.

Output:

Cmd_vel: The Twist message is published on the cmd_vel topic. This message contains the vehicle's expected velocity.

move_base provide a service to listen to the nav_msgs::goal, use the global planner to plan the global path, then sends the global path informations to the local planner. The local planner combines the surrounding obstacle information (from the costmap it maintains), global path information, target point information to obtain a best trajectory, then return the velocity value, send the Twist (geometry_msgs/Twist) messages to cmd_vel to control the vehicle movement.

We use amcl to get the position of the vehicle in the map. Publishing the map -> Odom transform and by requesting the transform map -> base_link, we can get the position.

Application in autonomous vehicules

Autonomous vehicle navigation gains increasing importance in various growing application areas. Nowadays, autonomous vehicles have basically achieved safe driving in terms of technology, however, according to the SAE Automation Levels, most current autonomous cars are still at level 2 or 3, level 3 vehicles can make informed decisions for themselves such as overtaking slower moving vehicles. However, unlike the higher rated autonomous vehicles, human override is required when the machine is unable to execute the task at hand or the system fails.
Today there are already some autonomous vehicles on the road for some specific uses, some mini shuttle buses has been used in a campus in China. Level 3 vehicles might be sufficient for these situations: route is fixed, surroundings is simple, it could be relatively easy to implement. In the future, for the sharing cars or private cars, it required at least the level 4 vehicles, which are able to intervene themselves if things go wrong or there is a system failure, these cars are left completely to their own devices without any human intervention in the vast majority of situations.

There are many good reasons for getting excited about the applications on autonomous vehicles, they can significantly reduce traffic congestion by searching the best trajectory according the gathering informations, and by maintaining an exact distance from the neighbour cars. They can also increase the accessibility for the disabled and elderly persons. However, autonomous technologies need still to be improved, there is still a long way to go to build the ultimate autonomous vehicles.

References:

1.移动机器人的导航与路径规划的研究 吕永刚 谢存禧
2.Autonomous car, wikipedia
3.ROS Navigation Stack之概要介绍 https://blog.csdn.net/TurboIan/article/details/79475672
4.ROS Navigation wiki http://wiki.ros.org/navigation
5.Introduction to AI Robotics, Robin R. Murphy
6.ROS by example, Patrick Goebel
7.ROS Navigation Tuning Guide, Kaiyu Zheng
8.Introduction to Navigation using ROS, Giorgio Grisetti