Robot Navigation

Anson, Tsan Kwong Wong

8 December 2016

1 Introduction

This project aims to explore robot navigation in the context of theories, implementations, limitations and improvement. Robot navigation is a basis of Human-Robot Interaction, which can be extended in many different applications: Entertainment, Education, Manufacturing Industry, Home and companion robotics, Rehabilitation and Elder Care, as well as Robot Assisted Therapy (RAT), etc. In order to ensure safety, a robot must be able to navigate with the presence of obstacles.

In this project, the basic navigation stack in ROS(Robot Operating System) will be introduced as a background, followed by experiments of two existing robot navigation algorithms. Section IV will talk about some insights from this project, including limitations of the algorithms and improvement for future work.

2 ROS - Navigation Stack

ROS is a collection of software frameworks which helps software engineer to develop robotic applications. In particular, navigation stack describes the entire flow of robot navigation framework in ROS. It receives odometry and sensor

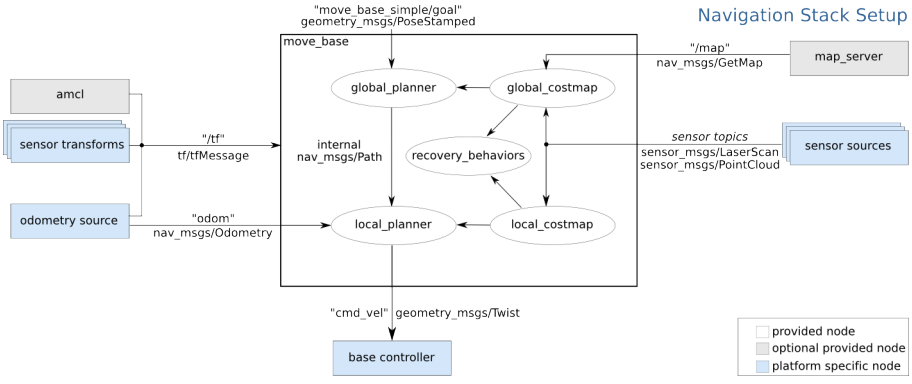

Figure 1: An overview of navigation stack

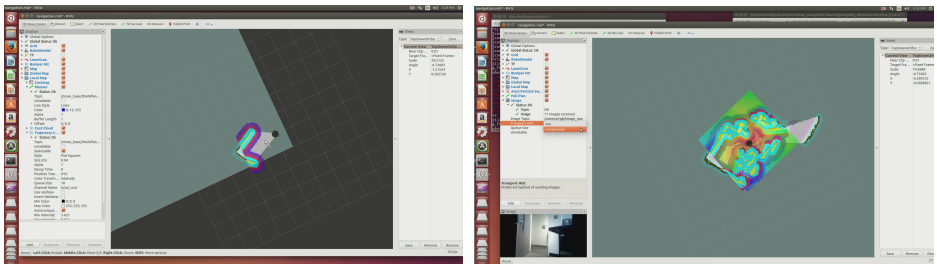

Figure 2: SLAM

information and outputs velocity commands to a mobile base. The robot needs to have a tf transform tree and publishes sensor data. Fig. 1 shows a overview of the navigation stack. More importantly, before the robot can be able to navigate, it first is required to obtain a map of the environment, which can be built with SLAM (Simultaneous Localization and Mapping).

2.1 SLAM

SLAM refers to constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. To perform SLAM, the robot first retrieve environment information from the surrounding through its sensors. SLAM will always use several different types of sensors like visual sensors providing visual information like raw images, and tactile sensors providing sensitive information like weight and texture.

Fig. 2(a) shows a SLAM process, visualized by RVIZ(a ROS package that visualizes robots, point clouds, etc.). The black circle in the center is a turtlebot, an integrated robot for ROS application development. The colored area shows the costmap of the state. It basically represents places that are safe for the robot. The costmap can be updated from the map with a localizatiton system, like AMCL, that allows the robot to register obstacles in the map; or updated from map directly within an area, dropping information if it moves too far from a given area. Fig. 2(b) shows the robot finished building the map in the office. With the map built, the robot has knowledge about the environment in the office and would be able to navigate there.

There are two types of costmap here, for two types of navigation: global costmap for global navigation and local costmap for local navigation. It gives rise to global and local planner.

2.2 Global and local planner

Global planner computes a minimum-cost global plan to move to the destination. It considers static obstacles in the map and gives a brief direction to the robot to follow. Local planner computes adjustment on velocities to traverse the current segment of the global path. It takes dynamic obstacles into account, avoiding collision by updating its local plan. By combining both planners, the robot

Planner	Time1*	Time2*	Time3*	Computational Power Level
DWA	0.13s	6.21s	16.34s	Low
TEB	0.04s	3.35s	7.78s	High

Table 1: Comparison between two planners. Time1: Time needed to reach a goal without rotation motion and without obstacles. Time2: Time needed to reach a goal without rotation motion and with obstacles. Time3: Time needed to reach a goal with rotation motion and with obstacles.

would be able to come up with a full plan for navigation.

3 Experiment

Two different planner are implemented, tested and compared with respect to computation time and power requirement: DWA(Dynamic Window Algorithm) planner and TEB(Timed Elastic Band) planner, and both are available in ROS.

3.1 Dynamic Window Algorithm

DWA samples from the set of achievable velocities for each simulation step under the given acceleration limits of the robot. It then performs forward simulation from the robot's current state and evaluate each resulting trajectory using metrics like speed, proximity to obstacles and the goal. It picks the trajectory with the highest-score and repeats.

3.2 Timed Elastic Band

TEB formulaes the planning problem into a weighted multi-objective optimization framework. It optimizes the initial trajectory generated by the global planner during run time with respect to minimizing the trajectory execution time, distance from obstacles and compliance with kinodynamic constraints such as satisfying maximum velocities and accelerations.

3.3 Comparison

Table. 1 illustrates the comparison between two planners. Result shows that TEB planner is fast generally in terms of computational time but requires higher computational power. Regarding TEB, the path planning algorithm cannot be done on the processor attached to the turtlebot, which is a NVIDIA Jetson TK1. Another machine with better configuration is needed to run the planner part, and sends the trajectory information to the robot during run time. Apart from it, TEB outperforms DWA, with the fact that in some cases DWA cannot find a valid path even when there is one, or it performs recovery behavior which is rotating for 360 degree before navigating and this makes the path planning process much longer.

4 Insight

After comparing the two planners, I discovered several limitations and improvements that can be done in terms of navigation.

4.1 Limitation

DWA chooses the highest-scoring trajectories among all achievable velocities, and sometimes it can not achieve the desired pose at the end. For instance, the robot's goal is to achieve at a destination and make a right turn. If the goal point is near to an obstacle, the robot tries to go directly to the goal and performs rotation at the goal. However, since it is too close to the obstacle, the motion is prevent due to potential collision. It can be problematic if this planner is used to simulate a car-like agent. Also this motion looks unrealistic and unnatural, and may incur barrier between the robot and human when the robot is exposed to human.

TEB works well on this, but the motion is not smooth neither. Both of them are also not "social aware", which means not taking how human interact in reality into account. If a person is running very fast to the left and the robot meets him at the crossroad, the robot will still try to continue its motion. However, in reality most people will stop as we are aware of this dangerous situation. In order to perform well in human-aware environment, robots would need to be acquired with those human-like behaviors.

4.2 Improvement

One problem of DWA is that the robot does not slow down early enough before arriving the goal if it is to make turns afterwards. One idea can be dividing the trajectory into several intermediate goal, while reconfigure its setting between consecutive goals with an objective like rotation needed for the next goal, to obtain an optimal combination between rotation and forward motion.

Meanwhile, pure collision avoidance algorithms seem not suitable for navigation through crowds of human or cars, as they do not consider human-human interaction. They can work well in designated environments, in which the presence of human is unlikely. However, for navigation into crowds of pedestrians, or autonomous driving vehicles, the motion and interaction must be realistic and human-like. Introducing human factors would definitely be helpful, which gives rise to social robotics.

4.3 Future work

I want to work on robot navigation with social robot element for my next project, and prevent the limitation discovered above. Most of the problems can be addressed by introducing human-human interaction. For example, gaze and head orientation provides information for human-human interaction, could be introduced to human-robot interaction.