Internship Report

Karmesh Yadav

July 24, 2017

Summary

This documents presents a report of the work covered during the internship, dated May-July, 2016. The project aimed to propose a strategy to solve the problems of obstacle avoidance and replanning for a mobile robot especially the Novus Stacker. Chapter 1 starts with a brief description of the problem of local planning. It further introduces the available local planners and the inherent problems in each of the planner. Chapter 2 discusses about Timed Elastic Bands and its ROS Package. The chapter ends with a discussion about the changes made in the algorithm for adding it to the robot_navigation code. Chapter 3 explains the costmap converter plugin used along with teb_local_planner for providing obstacles information in the form of polygons. Chapter 4 discusses about the results obtained and the future work possible.

Contents

The Problems of Local Planning **2**
Introduction 2
Desired Characteristics 3
Literature Survey 3

Timed Elastic Bands **5**
Introduction 5
Objective Fuctions and Constraints 5
Improvement 6
Teb Objects 7
Important Parameters 8

Costmap Converter **10**
Introduction 10
Plugins 10

Results **12**
Results and Future Work 12
Future Work 12

The Problems of Local Planning

Introduction

Motion Planning is concerned with the development of feasible paths that respect the kinematic and dynamic motion constraints of a robot. Often this problem in mobile robots is divided into different levels comprising of global planning, local planning and control. While the global planner may provide completely feasible paths to follow, however it is difficult to recompute them everytime an obstacle crosses. Thus it becomes the responsibility of the local planner to continuously compute smooth paths.

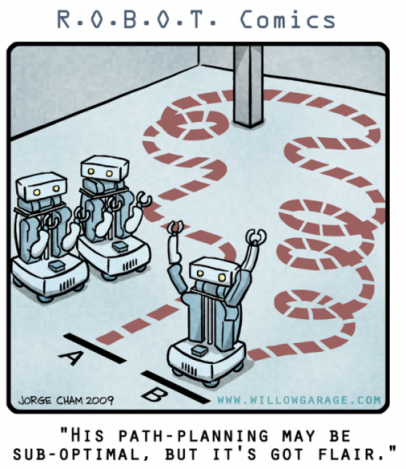


Figure 1: The problem of Motion Planning

The work involved research and implementation of a local planning algorithm for the Novus Stacker. A literature survey of various available planners was carried out and Timed Elastic Bands was chosen for implementation. Further testing and tuning of the algorithm was carried out on MR100.

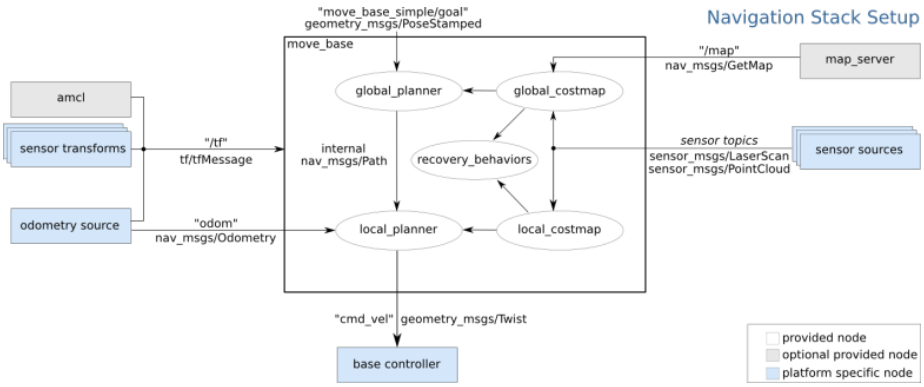


Figure 2: Planning Architecture of Move Base

Desired Characteristics

The following characteristics are desired from a local planner: -

- Avoids Dynamic Obstacles
- Generate smooth paths while obeying kinodynamic constraints
- Have a fast computation time for online replanning

Literature Survey

Local planning is a type of motion planning that requires the robot to move in dynamic environment where its algorithm will respond to the obstacle and the change of environment. Local path planning also can be defined as real time obstacle avoidance by using sensory based information regarding contingency measures that affect the safe navigation of the robot.

Local plans can pe presented in two ways, namely paths or trajectories. Although these terms are often used as synonyms, there is a difference between the two. Path is expressed as a continuous sequence of configurations beginning and ending with the boundary configurations, i.e. the initial configuration and the terminating configuration respectively. In other words, a path is a geometric trace that the vehicle should follow in order to reach its destination without colliding with obstacles. On the other hand, trajectory is represented as a sequence of states visited by the vehicle, parameterised by time and, possibly, velocity. Trajectory planning (also known as trajectory generation) is concerned with the real-time planning of the actual robots transition from one feasible state to the next, satisfying the robotss kinematic and dynamic limits and maintaining smoothness while at the same time, avoiding obstacles including other robots

and humans [1].

Most of the approaches in local motion planning deal with modification of already available global paths. It is a preferable approach due to the inherent uncertainty of the dynamic environments. One of the earliest solution which modifies a path locally is the elastic band algorithm. The main idea of the elastic band approach is to deform an originally given path by considering it as an elastic rubber band subject to internal and external forces which balance each other in the attempt to contract the path while keeping a distance from obstacles [2]. While the approach is able to handle non-holonomic kinematics as well as dyanmic obstacles, however it does not take the dynamics of the robot into account.

The timed elastic band algorithm modifies the elastic band approach by incorporating temporal information thus allowing control of velocity, acceleration and jerk of the trajectory. The algorithm discretizes the initial plan into closely spaced robot configuration states which are distributed at approximately the same time gap. The problem is then solved as a scalarized multi-objective optimization problem [3] [4].

Other approaches that use a discretized representation of the trajectory in configuration space include CHOMP and STOMP. Their proposed objective function contains a finite difference matrix to smooth the resulting trajectory and to additionally satisfy constraints like obstacle avoidance. CHOMP which stands for Covariant Hamiltonian Optimisation for Motion Planning, does not require that the input path to be collision free and therefore can work independantly of a global planner. Using a covarient gradient descent update rule leads to quick convergence of its trajectory to local minima [5]. Hamiltonian Monte Carlo has been suggested as a way to introduce stochasticity into the CHOMP gradient update rule. However, the method is found difficult to work in practice as it introduces additional parameters which are required to be tuned. Further it requires multiple random restarts before a successful solution is obtained [6].

STOMP(Stochastic Trajectory Optimization for Motion Planning) uses a similar cost function to CHOMP, but in contrast it can handle cost functions for which gradients are unavailable [6]. STOMP is an algorithm that preforms local optimization, hence its performance depends upon the initial trajectory used. It cannot be expected that it will solve typical motion planning problems like alpha puzzle in reasonable amount of time. Complete comparison between TEB and STOMP is given in [4].

Timed Elastic Bands

Introduction

Timed Elastic Bands comes from the family of trajectory generators that explicitly augments elastic band with temporal information, thus allowing the consideration of the robots dynamic constraints such as limited robot velocities, accelerations and jerks and therefore allowing direct modification of trajectories rather than paths [3].

The "timed elastic band" problem is formulated in a weighted multi-objective optimization framework. Most objectives are local as they depend on a few neighboring intermediate configurations. This results in a sparse system matrix for which efficient large-scale constrained least squares optimization methods exist [4].

The algorithm generates an initial path (using Probabilistic RoadMaps) composed of a sequence of way points into a trajectory with explicit dependence on time which enables the control of the robot in real time. Due to its modular formulation the approach is easily extended to incorporate additional objectives and constraints [7].

Objective Fuctions and Constraints

Timed Elastic Band approaches the planning problem using a hyper-graph based nonlinear optimization technique and the implementation with an open-source C++ framework called general (hyper-)graph optimization (g2o) [8] which solves graph based nonlinear optimization problems. The objective functions of the TEB belong to two types, (a) constraints and (b) objectives.

Constraints are: -

- Penalties for Velocity constraints along x, y and theta
- Penalties for Acceleration constraints along x, y and theta
- Penalties for non-holonomic constraints

- Penalties for Jerk along x and theta was added as an extra constraints to make the path smoother for the robot to follow.
- Penalties for taking the reverse direction(in differential drive robots)

Objectives are: -

- Distance from the obstacles
- Distance from dynamic obstacles (dynamic obstacles are not supported presently)
- Distance from the waypoints
- Minimum time between two poses

Different behaviours can be obtained from the algorithm by changing the weights of the objective function. Some tuning factors are further explained in the report.

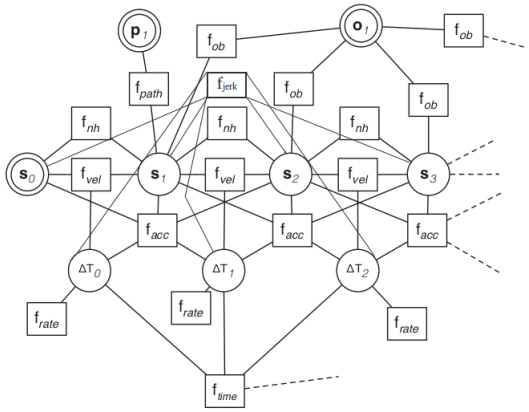
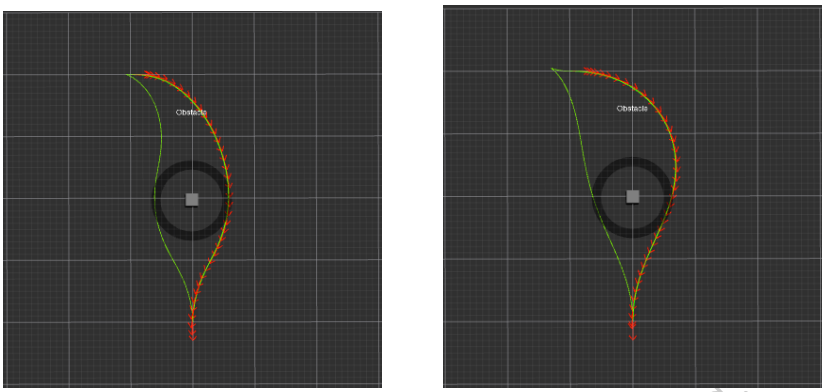


Figure 3: The TEB Hypergraph

Improvement

While the original teb algorithm makes trajectories that avoid obstacles as well as follow kinodynamic constraints, however these trajectories are difficult for the robot to track accurately due to presence of jerk in the motion. Thus to make the trajectories smoother, jerk(rate of change of acceleration) was added as a constraint in the obejctive function.

$$j_i = \frac{3(a_{i+1} - a_i)}{\Delta T_{i+2} + \Delta T_{i+1} + \Delta T_i} \tag{1}$$



(a) Original TEB (b) Minimal Jerk TEB

Figure 4: Difference between the trajectories obtained from the two algorithms

Teb Objects

Timed Elastic Bands has been implemented as a ROS plugin (named teb_local_planner [9]) to the base_local_planner of the navigation stack. To make the code executable in the robot_navigation package, a new class named teb_local_planner_THRSL is developed which has four public functions to create a trajectory. The functions are as follows: -

Function: initialize() - initializes the variables of the class
Inputs: std::string name - name of the instance i.e. "TebLocalPlannerTHRSL"
tf::TransformListener* tf - pointer to a transform listener
costmap_2d::Costmap2DROS* costmap_ros - Cost map representing occupied and free space usually a local costmap
Output: None

Function: setPlan() - Set the plan that the teb local planner is following.
Inputs: std::vector<nav_msgs::Odometry>& waypoints - Should contain the local start(x,y,theta,vx), intermediate waypoints(x,y) and goal(x,y,theta) for the trajectory
Output: True if the plan was updated successfully, false otherwise

Function: getTrajectory() - computes the trajectory upto the local goal.
Inputs: std::vector<TrajectoryPointMsg>& trajectory - It will be filled with the trajectory to be passed to the robot base
Output: True if the plan was updated successfully, false otherwise

Function: isGoalReached() - Check if the goal pose has been achieved
Inputs: None

Output: True if the plan was updated successfully, false otherwise
The flow of the code should be as follows: -

- Call initialize() at the start of the code.
- Find out the current pose and goal of the robot. Push the current state, then the via-points and finally the goal into the Odometry vector and call setPlan() function.
- Use the getTrajectory() function to recieve the trajectory stored as a TrajectoryPointMsg.
- Check if the goal pose has been achieved by using the isGoalReached() function.

Important Parameters

All the parameters of the planner are present in the configuration file(cfg folder) inside the package. Some important parameters for tuning are as follows: -

Parameter: dt_ref - Time based resolution of the planned trajectory i.e. the trajectory is discretized into robot poses at regular time intervals.
Value: 0.2 (usually it is set to the magnitude of the 1/control_rate).

Parameter: global_plan_overwrite_orientation - Some global planners are not considering the orientation at local subgoals between start and global goal, therefore determine it automatically.

Parameter: max_global_plan_lookahead_dist - Specify maximum length(if using teb as a plugin) of the subset of the global plan taken into account for optimization.
Value: Keep it within the visible area of the robot (the length is also bounded by the local costmap size).

Parameter: via_points_ordered - The planner would adheres to the order of via-points sent in waypoints.
Value: True (if the sent via points are part of the global plan).

Parameter: robot's dynamics parameters - Velocity and acceleration limits are the values until which no penalty is applied, Jerk limits are ignored and penalty is applied for all jerk values.

Parameter: min_turning_radius - Minimum turning radius if the robot is not a differential drive.

Parameter: xy_goal_tolerance - Allowed final euclidean distance to the goal position. Keep it low to reach an accurate position.

Parameter: yaw_goal_tolerance - Allowed final orientation error to the goal orientation in radians.

Parameter: min_obstacle_dist - Minimum desired separation from obstacles, which is taken care during optimisation step.

Parameter: include_costmap_obstacles - Whether the obstacles in the costmap should be taken into account directly.
Value: False (if obstacles are being provided externally using a teb_local_planner::ObstacleMsg message)

Parameter: weight_jerk_lim_x - Optimization weight for satisfying the maximum allowed translational jerk.
Value: 1-10

Parameter: weight_jerk_lim_theta - Optimization weight for satisfying the maximum allowed rotational jerk.
Value: 10-25

Parameter: weight_kinematics_nh - Optimization weight for satisfying the non-holonomic kinematics.
Value: 1000 for following the kinematic constraints, 0 for a holonomic robot.

Parameter: weight_obstacle - Optimization weight for satisfying a minimum seperation from obstacles.

Parameter: weight_viapoint - Optimization weight for minimizing the distance to via-points

Costmap Converter

Introduction

teb_local_planner solves the problem of obstacle avoidance by treating them as geometric objects. When the planner is fed with a costmap_2d object, 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). To solve this problem teb_local_planner supports the costmap_converter [10] plugins. Those plugins convert occupied costmap_2d cells to geometric primitives (points, lines or polygons) as obstacles. While, 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.

Plugins

Plugin: CostmapToPolygonsDBSMCCH - Converts occupied cells to a set of convex polygons. Single points are also treated as polygons.

Plugin: CostmapToPolygonsDBSConcaveHull - Converts occupied cells to a set of non-convex (concave) polygons. It first makes convex polygons and then converts them to concave polygons.

Plugin: CostmapToLinesDBSMCCH - Converts occupied cells to a set of lines (and points). It also makes concave obstacles first and then converts polygon edges to lines if there exist at least a predefined number of support points.

Plugin: CostmapToLinesDBSRANSAC - Converts occupied cells to a set of lines (and points). It generates clusters using the DBSCAN algorithm and then uses RANSAC to create lines from those clusters

Of all these plugin, CostmapToPolygonsDBSMCCH was used for finding out the polygons from the costmap since finding out convex polygon was a less time consuming process compared to other algorithms. Also the polygon obstacles

made were safer to follow compared to line obstacles as line obstacles sometimes had gap between them while actually there was an obstacle present. It led to the trajectory being generated through that gap.

Results and Future Work

Brief

Both original and minimal-jerk Timed Elastic Band trajectory planner was tested on the MR100 platform and in simulations on the Novus Stacker. The minimal-jerk teb performed better on both the platforms. The minimal-jerk timed elastic bands improve the smoothness of the paths thus making them more controller friendly. Although costmap_converter polygon is not able to accurately represent the obstacle, however it still can be used along with the package to improve the computation time of the planner.

Future Work

While the planner is giving desirable trajectories, work is still needed in these areas: -

- Improving the trajectory generator to take dynamic obstacles into account while planning using velocity obstacles approach.
- Providing it with obstacle information in the form of an ObstacleMsg. The message contains the geometry, orientation and twist information of the obstacle.

Bibliography

[1] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442, 2015.

[2] Sean Quinlan. *Real-time Modification of Collision-free Paths*. PhD thesis, Stanford, CA, USA, 1995. UMI Order No. GAX95-16902.

[3] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. Trajectory modification considering dynamic constraints of autonomous robots. In *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, pages 1–6. VDE, 2012.

[4] Christoph Rosmann, Wendelin Feiten, Thomas Wosch, Frank Hoffmann, and Torsten Bertram. Efficient trajectory optimization using a sparse model. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 138–143. IEEE, 2013.

[5] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009.

[6] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4569–4574. IEEE, 2011.

[7] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Planning of multiple robot trajectories in distinctive topologies. In *Mobile Robots (ECMR), 2015 European Conference on*, pages 1–6. IEEE, 2015.

[8] Code: General graph optimisation(g2o). `https://github.com/RainerKuemmerle/g2o`. Accessed: Accessed: June 2017.

[9] Package: teb local planner. `http://wiki.ros.org/teb_local_planner`. Accessed: Accessed: June 2017.

[10] Package: costmap converter. http://wiki.ros.org/costmap_converter. Accessed: Accessed: June 2017.