Kinodynamic Motion Planning

June 9, 2022

1 Abstract

Autonomous navigation systems usually include a path planner and a local planner. The first one provides a somewhat kinematically reasonable path, which often cannot be passed by a robot, while the second one tries to adapt the path to the dynamic environment and other constraints that were not checked by the path planner. This work is an attempt to create an optimized RRT-based path planner that takes into account nonholonomic constraints. This is achieved through a combination of configuration space optimization, usage of R-Trees and already well-known techniques, such as goal-biased sampling. As a result, open source reference implementation in Python constructs trees that require far less local (reactive) planning than typically.

1.1 Keywords

Path planning, Collision avoidance, Nonholonomic constraints, Rapidly exploring Random Tree, R-Tree, Reeds-Shepp Car.

2 Introduction

Motion planning and autonomous vehicles have been developing for quite long time already and in recent 20 years those areas have grown from enterprise-only solutions (i.e. robotic arms and other manufacturing devices) to consumer-friendly ones (robot vacuums and ”autopilots” in cars). If initially the problem was to ”get from point A to point B”, by this time it has been expanded with operation in dynamic environments, collision avoidance, computation efficiency and others.

All of the mentioned problems have one thing in common - Configuration Spaces framework that is used to describe the state of the robot in the environment. In order to state the problem one should define C-Space - $\mathcal{C}$, which includes configurations of the robot q, world $\mathcal{W}$ (including obstacles) and other constraints.

This work is aimed to solve the problem of planar path-planning, thus $\mathcal{W} = \mathbb{R}^2$. Some parts of the world are occupied by the obstacles $\mathcal{O} \subset \mathcal{W}$ defined by polygons, which cannot be self-intersecting. And finally the robot - a rigid body $\mathcal{A}$, which can translate in the world, making $\mathcal{C}$ a manifold $M_1 = \mathbb{R}^2$, and rotate around its bounding box center, which adds $M_2 = \mathbb{S}^1$, thus without obstacles:

$$\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1 = SE(2) \tag{2.1}$$

The robot's configuration in the space is defined as a vector $q = (x, y, \theta)$ and the space that it is occupying as $\mathcal{A}(q) \subset \mathcal{W}$. Thus, the configuration space for obstacles is:

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq 0\} \tag{2.2}$$

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs} \tag{2.3}$$

The algorithm is created with holonomic - the robot can only travel in $\mathcal{C}_{free}$ - and nonholonomic constraints in mind - it uses Reeds-Shepp Car model and steering function as the only way to produce a path from $q_1 \in \mathcal{C}_{free}$ to $q_2 \in \mathcal{C}_{free}$, making some configurations in C_{free} unreachable. The Reeds-Shepp Car has only two gears - forward and reverse ($u \in \{-1, 1\}$); and angular velocity that depends on maximum turning radius with unit-velocity r_{turn} - $w = \frac{v=1}{r_{turn}}$. In general, the path would be a non-linear function of time, as a turn is a curve. However, in order for a robot to be able to follow the path, each turn is approximated with a given level of discretization δ to a set of straight lines:

$$\begin{cases} \dot{x} = x_{prev} + u \cdot \delta \cdot \cos\theta \\ \dot{y} = y_{prev} + u \cdot \delta \cdot \sin\theta \\ \dot{\theta} = \theta_{prev} + u \cdot \frac{\delta}{r_{turn}} \end{cases} \tag{2.4}$$

Finally, the starting points are defined as $q_i \in \mathcal{C}_{free}$ and the goal point is $q_g \in \mathcal{C}_{free}$. The local planner should produce a path, which is a set of connected linearized curves that were provided by the steering function.

Generally, industry solves such problem with either a "sampling" algorithm (RRT and variants, PRM) or a graph traversal algorithm like A*. Right now RRT-like algorithms are used more as with a proper set of improvements they can converge quite fast. The data on obstacles is often stored as points KD-Tree and the collision detection is performed by querying it.