A.M.R. Project report

Caterian Lacerra, Jary Pomponi

July 5, 2017

1 Introduction

In this project we have studied the problem of motion planning. More precisely we have used and compared different algorithm.

Informally speaking, the motion planning problem aims to find a path, for a specified robot, from a point A to a point B avoiding obstacles. In practices this problem is very hard for a computational complexity point of view, since need to map the environment in the configuration space of the robot.

The strength of the algorithm that we have used is that, instead of build and use an explicit representation fo the robot in its configuration space, they rely on a collision checking module. With this approach we build a graph of feasible trajectories in the obstacle free space. The the road map can be used to build the solution for planning problem for a specific robot.

2 Motion planning problem

2.1 Problem formulation

In this section we will formalize the problem of path planning.

Let $\chi \in (0,1)^d$ be the configuration space, where $d \in \mathbb{N}$ with $d \geq 2$, and χ_{obs} is the obstacle region of the space. The χ_{free} is the portion of the space that does not contains obstacles. The we define an initial position $x_{init} \in \chi_{free}$ and a goal region $\chi_{goal} \subset \chi_{free}$. A path planing problem is defined by a triple $(\chi_{free}, x_{init}, \chi_{goal})$.

The goal of a path planner is to find a feasible path from x_{init} to χ_{goal} in χ_{free}, if exists, and report failure otherwise.

To do this we build a DAG (direct acyclic graph) of possible road maps from the initial point to the other point in χ_{free}. The we get the optimal path from all the possible ones.

2.2 Primitive procedures

Before discussing the implemented algorithm we introduce the principal procedures that we will use.

SampleFree: is a function that return a point $x_r \in \chi_{free}$. The samples are assumed to be draw from a uniform distribution, and each one is independent from others.

NearestNode: Given a a graph $G = (V, E)$ and a point $x \in \chi_{free}$, where $v \subset \chi$, the function return a point $x_{near} \in V$ that is the closest to x given the euclidean distance. Formally:

$$getNearest(G, x) := argmin_{v \in V} ||x - v||$$

Another version of this function, taken another value r, will return a set of $x \in V$ that are contained in a ball of radius r centred in x.

Steer: Given two point $x, y \in \chi$, the function steer returns a point $z \in \chi$ that minimize the distance from z to y while $||z - x|| \leq \eta$, with $\eta > 0$.

collisionFree: Given two point $x, y \in \chi_{free}$ the function return true if the line that connect x to y lies in χ_{free}.

Parent: Given a Graph $G = (V, E)$ and an $x \in V$, the method return a point $y \in V$ that satisfy $(y, x) \in E$. By convention, if x is the root node of V this function will return x.

Line: Given two points $x, y \in \chi$, the methods return a line that go from the point x to the point y.

Cost: Given a graph $G = (V, E)$ and $v \in V$, the methods will return the cost $c \in \mathbb{R}$ of the unique path from the root of the graph to the node v. By convention the cost of the root node is zero.

2.3 Existing Algorithm

In this section we will see the existing algorithm that we have implemented and compared.

2.3.1 Rapidly-exploring Random Trees (RRT)

Rapidly-exploring Random Trees is an algorithm aimed at a single query application. In the basic version, the algorithm build a graph of feasible trajectory rooted at the initial node; the basic version of this algorithm is showed in **Algorithm 1**.

At each iteration the algorithm sample a new point in the free space. Then calculate the nearest vertex to the sampled point, and then the algorithm tries to connect the nearest vertex to this random point, using the **Steer** function. If the connection does not collide with any obstacles the new vertex and the edge, that connects the new point and the nearest one, are added to the graph.

In this version of the algorithm we perform those operations n times, so the algorithm does not stop if reach the goal.

Algorithm 1: RRT

```
G = ($x_{init}, \emptyset$)
for i = 1... n do
        $x_{rand} \leftarrow$ SampleFree
        $x_{nearest} \leftarrow$ NearestNode(G,$x_{rand}$)
        $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
        if collisionFree($x_{nearest}, x_{new}$) then
                $V \leftarrow V \cup \{x_{new}\}$
        $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$
```

2.3.2 Anytime RRT

2.3.3 RRT*

This algorithm still use a tree graph, but with the improvement that the graph are modified at each iteration, if it discover better path from the root to the actual node.

This algorithm add a point to the graph in the same way as RTT does, but it also consider connection from a subset of the vertex called xNear. This set contains the nodes that are inside a ball of radius

$$min\left\{\gamma_{RRT}\left(\frac{log(card(V))}{card(V)}\right)^{1/d}, \eta\right\}$$

where d are the dimension of the configuration space χ, η is the constant of steering function and γ_{RRT} are :

$$2\left(1+\frac{1}{d}\right)^{1/d}\left(\frac{\mu(\chi_{free})}{\varsigma_d}\right)^{1/d}$$

where $\mu(\chi_{free})$ denotes the volume of the free space and ς_d is the volume of the unit ball in the d-dimensional Euclidean space.

In our case d is equal to 2.

Algorithm 2: RRT*

```
G = $(x_{init}, \emptyset)$
for i = 1... n do
$x_{rand} \leftarrow$ SampleFree
$x_{nearest} \leftarrow$ NearestNode(G, $x_{rand}$)
$x_new \leftarrow Steer(x_{nearest}, x_{rand})$
    if collisionFree($x_{nearest}, x_{new}$) then
        $radius \leftarrow min\{\gamma_{RRT}(\frac{log(card(V))}{card(V)})^{1/d}, \eta\}$
        $xNear \leftarrow nearestNode(G, x_{new}, radius)$
        $V \leftarrow V \cup \{x_{new}\}$
        $x_{min} \leftarrow x_{nearest}$
        $c_{min} \leftarrow Cost(x_{nearest}) + ||x_{nearest} - x_{new}||$
        foreach $x_{near} \in xNear$ do
            if $collisionFree(x_{near}, x_{new}) \wedge Cost(x_{near}) + ||x_{near} - x_{new}|| < c_{min}$
            then
                $xmin \leftarrow x_{near}$
                $c_{min} \leftarrow Cost(x_{near}) + ||x_{near} - x_{new}||$
        $E \leftarrow E \cup \{(x_{min}, x_{new})\}$
        foreach $x_{near} \in xNear$ do
            if $collisionFree(x_{near}, x_{new}) \wedge Cost(x_{near}) + ||x_{near} - x_{new}|| < Cost(x_{near})$
            then
              $x_{parent} \leftarrow Parent(x_{near})$
              $E \leftarrow (E \setminus \{(x_{parent}, x_{near})\} \cup \{(x_{new}, x_{near})\})$
```

3 Set up and Experiments

In this section we will explain the implementation and we'll show the results of the experiments.

3.1 Implementation of the algorithm

3.2 Experiments

4 Conclusion