Sphere Sampling in CSpace for Motion Planning

Yinan Zhang

March 23, 2014

1 Introduction

Traditionally, sampling based motion methods use discrete data to construct paths for motion tasks. The disadvantage of this kind of implementation is obvious:

1. CSpace is a continuous space, discrete samples cannot represent it well;
2. Many samples are actually not necessary and consume a lot of memory, e.t. samples far away from obstacles.

Our method is to sample spheres which contain collision-free area in configuration space. Using these hyper-spheres in c-space, we can:

1. describe configuration space in a continuous way;
2. find an optimal path using a modified A* search;
3. save memory by reducing unnecessary samples.

2 Sphere Sampling Algorithm

Data: Configuration Space
Result: Spheres that cover most of the free space
s0 = sampleSphere(randomly position);
S = {s0};
`// Every two boundary points distance equals to` δ
boundaryQueue = a queue that contains all boundary points of s0;
while *boundaryQueue is not empty* **do**
  `// Candidate point to sample next sphere.`
  point = boundaryQueue.pop();
  if *point not in any spheres* **then**
    si = sampleSphere(point);
    if $\delta \leq$ *si.radius* **then**
      S = S $\cap$ {si};
      boundaryQueue.put(all boundary points of si);
    end
  end
end

Algorithm 1: Sphere sampling algorithm

2.1 Something we know

2.1.1 Will the algorithm stop?

As is shown in the algorithm, we get boundary points of an existing sphere by limiting the distance between every two points to be equal to δ. Then the algorithm samples new spheres centered at these points. So the distance between two spheres centers is always larger than or equal to δ.

The problem is equivalant to proof there could be finite number of points put in the space such that the distance between every two points is larger than or equal to δ.

Assume we partition the whole configuration space(including free space and obstacle space) into hyper-cubes. The length of every edge of these hyper-cubes is exactly δ. As long as the configuration space has finite volume, the number of hyper-cubes is finite. Assume the number of hyper-cubes is N, the number of hyper-cubes that is totall or partly in free space is less than N. Thus the number of spheres is finite, and less than N.

So the algorithm will always converge.

2.1.2 δ-clearance

[1] proposed an algorithm, named PRM*, to find a path. It requires the actual optimal path to have δ-clearance so the algorithm can find a approximately optimal path. Yet, the algorithm cannot explicitly control the quality of the path by defining δ. In our alogrithm, we can define the actually value of δ and control the coverage of free-space by sampling spheres.

2.1.3 Find Containing Spheres

In the algorithm, we need to know if a candidate point is inside any spheres. Naively, we need to iterate each existing spheres to determine if a point is inside, which requires O(n) time. This can fortunately be done with less time by hashing spheres to uniform grids of the c-space. Each grid contains references to spheres that intersect/contain the grid. By hasing a candidate point in the same way, we can easily know if a point is inside any spheres already.

(Can we use Locally Sensitive Hashing?)

2.1.4 Inaccurate matric

Suppose we have an inccurate matric which gives us the distance to obstacle: dist[i] = accurate_dist[i]/c[i], where $1 \leq$ c[i] $\leq$ upper_bound. When sampling at a point near the obstacle, dist[i] = accurate_dist[i]/c[i] = δ the algorithm will stop sampling at that point. So the farthest distance from spheres boundary to obstacles is δ * upper_bound.

If the optimal path has δ * upper_bound clearance. The algorithm can still cover the path and eventually find it.

2.1.5 How to deal with unlinked configuration space?

A C-space can be partitioned into several parts by obstacles, each part is not linked to others.

2.2 Some Questions:

2.2.1 How to use less hyper-spheres to cover the space?

2.2.2 How to get the distance to obstacles more accurately?

3 Modified A* search

References

[1] Karaman, Sertac, and Emilio Frazzoli, *"Sampling-based algorithms for optimal motion planning"*. The International Journal of Robotics Research 30.7 (2011): 846-894.