

Probabilistic Roadmap Method

Han Ul Yoon

hyoon24@uiuc.edu

Abstract

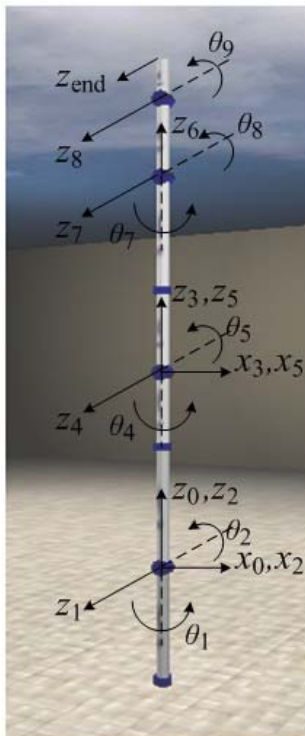
This paper presents *Probabilistic Roadmap Method* (PRM) simulation with straight-line local planner [1]. I set up a simulation environment with 7-links robot, obstacles (bombs), and a goal. First of all, I generated samples in free configuration space. Second, if one sample could be connected to its neighbor samples, store these two samples in a graph with a cost (L^2 -metric distance between two samples). Finally, the path, from start point to the goal, could be found by simple graph search algorithm.

I. 7-Links Robot Implementation

Figure 1 shows that my 7-links robot and its DH-parameters. The robot has 2-spherical joints, and I added up two *Ghost*-axis (z_2 and z_5) arbitrarily; because this is the only way, if we want to start 7-links robot simulation from right stand-up position under DH convention. To clarify it, let's remind the two constraints for DH-representation [2]:

(DH1) The axis x_1 is perpendicular to the axis z_0 .

(DH2) The axis x_1 intersects the axis z_0 .

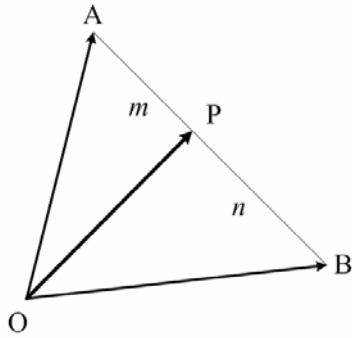


Link	Actual Link	a_i	α_i	d_i	θ_i
1	1	0	-90	0	θ_1^*
2	2	0	90	0	θ_2^*
3	<i>Ghost</i>	0	0	1.5+1.0	<i>fixed</i>
4	3	0	-90	0	θ_4^*
5	4	0	90	0	θ_5^*
6	<i>Ghost</i>	0	0	1.5+1.0	<i>fixed</i>
7	5	0	-90	0	θ_7^*
8	6	1.0	0	0	θ_8^*
9	7	0.5	0	0	θ_9^*

θ_i^* : variables

Figure 1. 7-links robot and DH-parameters.

Also, I was thinking of the extension of this idea to serpentine type robot simulation (as you know, every joint of snake type robot can be represented as a spherical joint), this technique will be helpful unless you would like to move your snake robot from any crooked posture at the beginning. We can determine the position of link3 and link5 by easy (probably middle school level) vector calculation (see Fig 2.).



$$\overrightarrow{OP} = \frac{n \times \overrightarrow{OA} + m \times \overrightarrow{OB}}{m + n}$$

Figure 2. Vector calculation to determine the position of link 3 and link 5.

II. Sample Connection

First, I declared a data type

```
typedef struct element {
    int d1, d2, d4, d5, d7, d8, d9;
    bool collision;
};
struct element sample[MAX_SAMPLE];
struct element free_sample[MAX_SAMPLE];
```

If a randomly generated sample is in collision free space, then store it to free_sample array. These free_sample become vertices v_i in graph G .

Second, I used the adjacency matrix from to represent graph structure [3]; thus if $G=(V, E)$ is a graph with n vertices, $n \geq 1$, the adjacency matrix of G is a two-dimensional $n \times n$ array, say cost. If there exists the edge (v_i, v_j) that can connect two nodes with straight-line planner in $E(G)$, then $\text{cost}[i][j] = L^2\text{-distance}$ between two nodes (samples). If there is no such edge in $E(G)$, $\text{cost}[i][j] = \infty$ (also if $L^2\text{-distance} >$ neighborhood). Figure 3 shows an illustrative example.

Finally, Dijkstra's algorithm (function shortest_path below) was used to find the shortest paths.

```
void shortest_path(int v, double cost[MAX_VERTICES][MAX_VERTICES], int path[], int n,
short int found[])
```

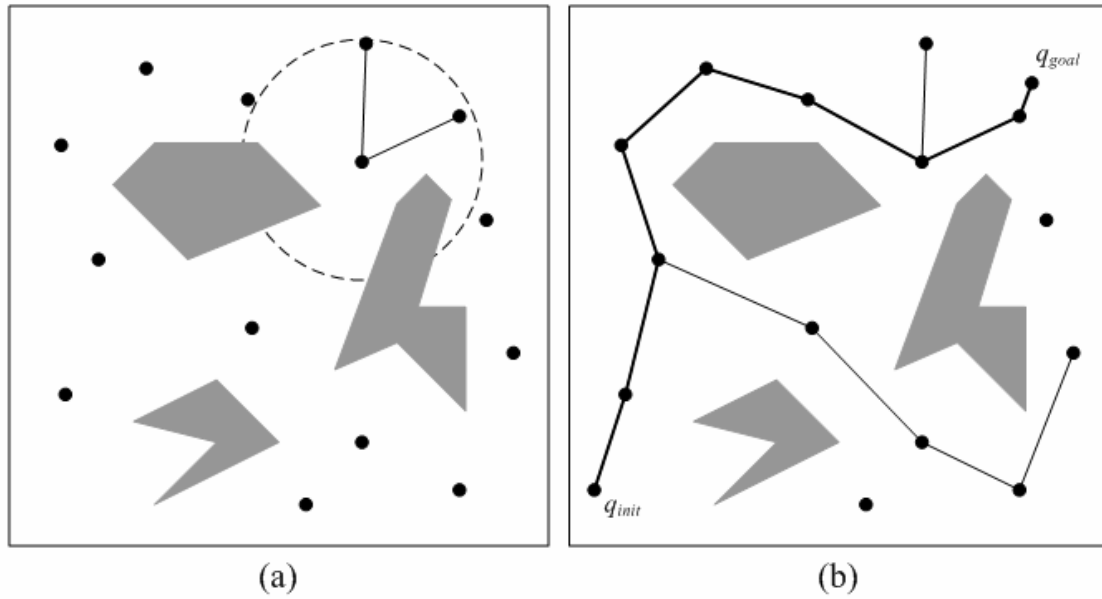


Figure 3. (a) Build a roadmap by connecting a sample to its neighbors, (b) A path by connecting q_{init} and q_{goal} to the roadmap.

```

{
    int i, u;
    bool gcheck = false;
    for(i=0; i<n; i++) found[i] = false;
    found[v] = true;
    u = 0;
    path[pindex++] = u;
    for(i=0; i<n-2; i++) {
        u = choose_path(cost, u, n, found);
        if(cost[i][u]==INF) {
            printf("The last position where could be a goal is: %d\n", u-1);
            printf("**Cannot found goal... Generate more samples...\n");
            exit(1);
        }
        found[u] = true;
        path[pindex++] = u;
        gcheck = goalcheck();
        if(Gcheck) break;
    }
}

```

```

int choose_path(double cost[MAX_VERTICES][MAX_VERTICES], int u, int n, short int
found[])
{
    int i, minpos;
    double min;
    min = 65535;
    minpos = -1;

```

```

for(i=0; i<n; i++) {
    if(cost[u][i]>=INF) n_inf++;
    if(n_inf>=MAX_VERTICES-2) {
        n_inf = 0;
        return path[pindex-2];
    }
    if(cost[u][i]<min && !found[i]) {
        min = cost[u][i];
        minpos = i;
    }
}
n_inf = 0;
return minpos;
}

```

III. Miscellaneous

To work with spherical joint was truly cumbersome...(or I am thinking that my brain has a problem with a geometric concept...)

Acknowledgement

I thank to *Prof. Seth Hutchinson*, my mentor, and *T.A. Sal Candido*, who greatly helped and inspired me through this homework. Specially, Sal's explanation about metric space lifted me up to one level.

References

- [1] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion*, MIT Press, 2005, pp.77-106.
- [2] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. John Wiley & Sons, 2006, pp.163-182.
- [3] E. Horowitz, S. Sahni, and S. A. Freed, *Fundamentals of Data Structure*, Computer Science Press, 1993, pp. 257-303.