Final Report: Motion planning of a pick & place robot in complex environments

Team2

Chengxi Dong ; Liangyu Chen; Linglin Zhang; Sikai Geng

***Abstract*—This report is aimed to provide a prototype of a pick & place robot working in complex environments. Under these conditions, the robot should be able to avoid obstacles. Further, we need a efficient, optimal motion planner to make it work better. The motion planning part is partly based on PRM* algorithm, it's the most efficient multi-queries algorithm. By using this method, we can make it pick objects sequentially without reaching obstacles. We will also discuss some approaches to reduce the complexity of collision check. A classifier will be used to predict if a point is in free space or not.**

***Index Terms*—Motion planning, probabilistic roadmaps, support vector machine, sampling-based algorithm, optimal motion planning, collision check.**

I. INTRODUCTION & LITERATURE REVIEW

Pick & place robot has received a considerable amount of attention. Most of this kind of robots are used for manufacture, assemble and logistic. Modern robots may possess significant differences in model, sensor, actuator, workspace, etc. However, the motion planning problems are almost the same in the configuration space.

Being faced with the curse of dimensionality, PRM [1] and RRT [2] are the most commonly used algorithms. PRM is always used in multi-query problems. There are many modified algorithms such as PRM* and RRT* [3], that can find an asymptotically optimal path.

However, when the obstacles have too many faces, to check obstacles would be of high computational complexity. If the robot can predict whether a point is in free-space or not, then we don't have to run the collision check function for that. An effective way is to maintain a data structure and estimate the minimum distance from a point to obstacles set [4], but the coefficient α need to be chose. [5] gives a simple concept that we can use a trained machine learning model to find if a point is in free-space.

II. GOALS & HYPOTHESES

Our goal is to make the robot sequentially pick the targets in a complex environment. The model we choose is a prototype of Stanford Manipulator. It's of 6 DOFs, in order to determine the direction and position.

In this case, we will use PRM* algorithm to solve the multi-query problem. In the meanwhile, we train a SVM [6] classifier to approximately check collision. It can reduce the computational complexity when the obstacles are of odd shapes.

To simplify the tasks, we have some hypotheses.

First, we assume the structures of robot and the obstacles are several convex polytopes, And we can get all information of those convex polytopes. Generally, concave polytopes can be decomposed into several convex polytopes. The method of decomposition was discussed in [7]. Then, we can use Hyperplane separation theorem [8] to check collision between each pair of these polytopes.

Second, we assume the sensors are ideal. i.e. We can get the true status at any time without noise. It's important for the simulation. There are many ways to solve the errors, like robust control theory, but it's not the main topic of this report.

Finally, we ignore the distances between link 1,2,3 and 4,5. Hence that, The DH table would be:

TABLE I. DH-TABLE OF THE 6-DOFS MANIPULATOR

Link	**a**	**α**	**d**	**θ***
1	0	-90	0	θ1
2	0	+90	0	θ2
3	0	0	d3*	0
4	0	-90	0	θ4
5	0	+90	0	θ5
6	0	0	d6	θ6

There will be only two shafts to represent the robot as Fig.1.

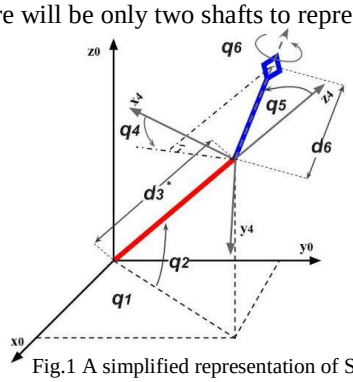


Fig.1 A simplified representation of Stanford Manipulator

III. FOUNDATIONS

A. *Configuration Space*

By using forward kinematic method and inverse kinematic method. We can get a one-to-many mapping from the status of end effector **x=**[x, y, z, α, β, γ]' to the generalized coordinates **q=**[q1,q2,d3,q4,q5,q6]'. Since the mapping is one-to-many, we can choose one solution got from the inverse kinematic method as the principal solution. Then the mapping is one-to-one. The details of inverse and forward kinematic solutions can be found in Hw4 Programming part, so it would not be explained in this report.

B. *Collision Check*

By *hyperplane separation theorem*, in 3D space, two polytopes are not intersected if and only if they are separated by a plane. By projecting it to 2D space, we can find there must be a face of these two polytopes satisfies that. It's shown as Fig.2

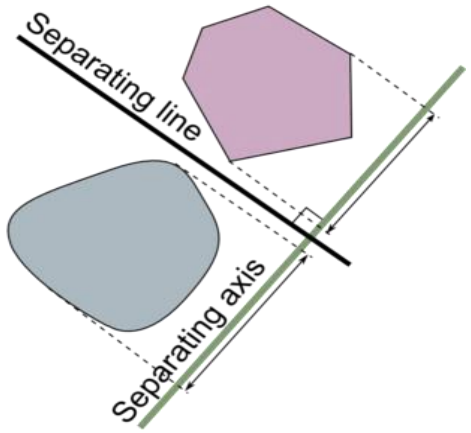


Fig.2 Illustration of the hyperplane separation theorem. From Wiki.

C. *Support Vector Machine*

After choosing landmark points, we run collision check for these points. Then we use kernel function (In this case, it's Gaussian kernel.) to map these points into infinite dimensional space. They will become linear separable. Then, SVM will find a separating plane to divide two sets in the infinite dimensional space.

In a nutshell, A trained SVM model can predict if a point is in free space. The running time is O(a) where a is the number of landmarks. In complex environments, obstacles may have many faces, then the collision check will be terribly complex. In this case, learning method like SVM can reduce the running cost.

Further, an online learning SVM model can change the landmarks and parameters every time. It's important for avoiding dynamic obstacles.

IV. MATH & ALGORITHM

There are two main parts of this project: motion planning and control. The first part is based on PRM* algorithm, we will train a SVM classifier for collision predict and finally construct a graph G = (V, E) in configuration space. Then, we will find a path in this graph by Dijkstra algorithm. To keep the computational complexity in accordance with part 1, it should be Fibonacci heap implementation [9]. Finally, we generate a desired state function with respect to time by interpolation.

Due to space limitations, the control part will not be discussed in detail. Generally, we can implement a PD controller to calculate the input by desired states, and tune the system until it's nearly optimal.

Algorithm 1 SVM_TRAIN(n,m)

Initialize σ, λ, V=Ø
For i=1,2….,n
    q = random sample
    V_landmark = V_landmark ∪{q}
end
For i=1,2….,m
    r = random sample
    V_validation = V_validation ∪{r}
    If check_collision (r_i)
        $y_i = 0$
    Else
        $y_i = 1$
end
$w = \mathrm{argmin}_w \, F_{cost}(w, V_{landmark}, V_{validation}, \lambda, \mathrm{kernel}(\sigma))$

Note that the argmin process will be implemented by ***stochastic gradient descent*** [10]. After trained, the SVM classifier would be able to predict if a point is in free space.

In Probabilistic Roadmap algorithm, we have to check collision at segment points as a replacement of free path check. So there will be a lot of points need to be checked. If we can predict the collision, the collision check for path would be unnecessary.

Algorithm 2 SVM_PREDICT (w, V_landmark, q)

For i=1, 2 … n
q_i= V_landmark(i)
end
$J=\sum_{i=1}^{n} k(q_i - q) - b$
If J≥0
    return True
else
    return False

Algorithm 3 PRM*

E=Ø, v= Ø
For i=1, 2 … n
    v_i=Random sample
    V=V∪{v_i}
end
For each $v_i \in V$
If j≠i & $\|v_j - v_i\| < \gamma(\log n/n)^{\frac{1}{d}}$
    If Freepath (v_i, v_j)
    then E=E∪{(v_i,v_j)}
end

Algorithm 4 Freepath (v_1, v_2, M, w, V_landmark)

```
N=M*||v1-v2||
For  i=1, 2 … N
     V=v1+(i/N)(v2-v1)
     If  ! SVM_PREDICT(w, V_landmark,V)
     then      return False
end
return True
```

PRM* is different on choosing neighbors with PRM, sPRM, and k-PRM. The differences were specified in [3]. We can get an asymptotically optimal path between every two vertices. i.e. When the iteration number is sufficiently large, the path will be optimal. We use **Dijkstra algorithm** to find the path in G.

Algorithm 5 Dijkstra(G=(V, E), start, end)

```
For  each v∈V
     d[v]=inf
     prev[v]=null
end
d[start]=0
S=Ø
Q=Fibonacci_heap(V)
while Q≠Ø
     u=extract_min(Q)
     S=S∪{u}
     For each edge (u, v)
          relax (d[v], u, v)
     end
end
k=0
u=End
while(u≠null)
          path[k++]=u
          u=prev(u)
end
path=reverse(path)
```

Note that, in order to make the running cost O(NlogN) in accordance with previous part. We need a special data structure for Q. The find_min and delete functions should be O(logN) or O(1). Fibonacci heap is almost the optimal choice. A binary heap or binomial heap is also OK.

Finally, we need to generate a trajectory. We have a discrete sequence $\{q_n\}$, there are infinite number of trajectories satisfy the sequence. We can do optimization to minimize anything. Such as minimizing energy, minimizing input, or minimizing any function. Basically, it's a calculus of variations [11] problem. And there are also some new methods [12][13] for that.

In this case, we will only use interpolation method. We assume the function for every independent variable is a polynomial of 3th order. Then, assume the time from q_i to q_{i+1} is always a period T0. Such that:

$$f(t) = a_j + b_j T + c_j T^2 + d_j T^3 \quad (T \in [0, T0]) \qquad (1)$$

Where $$t = T0 * j + T \qquad (2)$$

To solve the equations with 4n unknowns. We need 4n equations.

$$f(jT0) = q_j \qquad \text{j= 0,1,2….n} \qquad (3)$$

Then, we still need 4n-(2n-2)-2=2n equations.

The derivatives should be continuous at every intermediate point.

$$f'_j(jT0) = f'_{j+1}(jT0) \qquad (4)$$

$$f''_j(jT0) = f''_{j+1}(jT0) \qquad (5)$$

And specify the boundary conditions, generally, the manipulator should keep still at first and last status. Such that

$$f'(0) = f'(nT0) = 0 \qquad (6)$$

$$f''(0) = f''(nT0) = 0 \qquad (7)$$

Solve the 4n*4n system of linear equations, then we get all the parameters for interpolation functions.

Since now, we have finished the motion planning part. The q_desired(t) has been obtained. The remaining part is a controller to control the robot.

Due to space limitation, we won't discuss the control law and optimization. Let's focus on motion planning.

V. Analysis

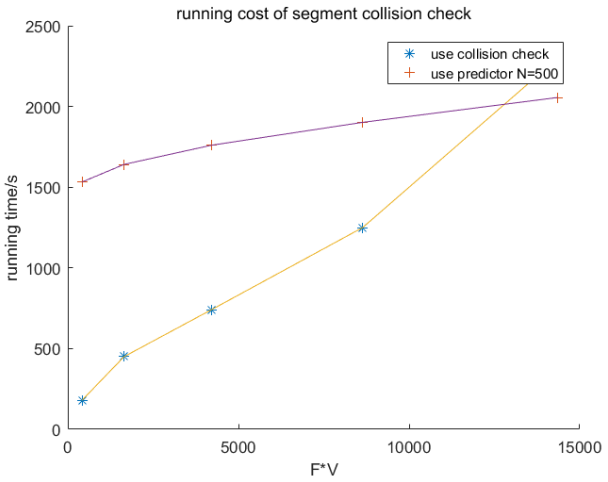
In this section, we will analyze the model in two aspects: correctness and running cost.

The analysis of PRM* was done in [1][3], so we don't want to repeat it nor compare it with RRT or other algorithms.

We'll focus on the question: ***does SVM classifier help?***

Consider the segment collision check. Denote the number of landmarks is N, and the number of faces of all obstacles and links is F, number of vertices is V.

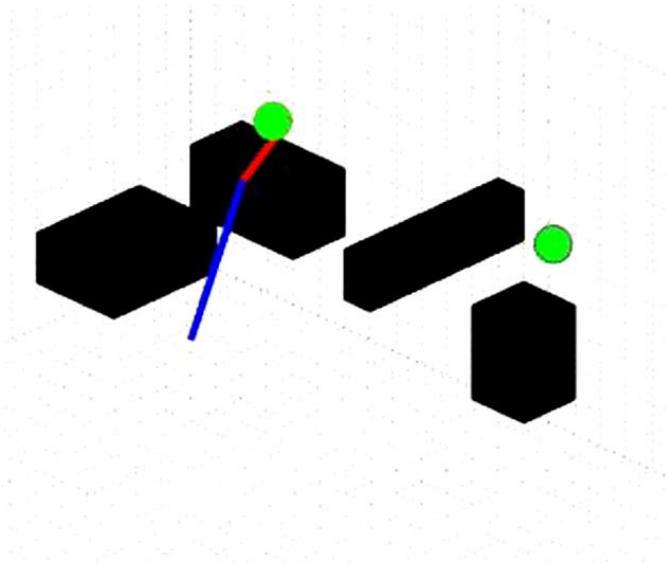
The running cost of collision check would be O(FV). The running cost of prediction is O(N), if FV>>N, the SVM classifier would be useful and vice versa. That's the reason why we addressed 'complex environments'.



This program was tested on PC, both generating 1000 samples and query the path. Due to SVM training need collision check, the running cost for training predictor will also slowly increase with the growth of FV.

Then, we will analyze the accuracy of the classifier. as we know, a machine learning model may need a lot of data to train the parameters. In this case, we use N samples to train and N samples to validate, in a random generated environment. It's not as complex as we want due to the limitation of PC efficiency. However, it's hard to find the best fit σ. We guess it's a function of sampling number n and dimension d. After finding a nearly optimal radius for Gaussian kernel, the accuracy could be up to 98.4% for N=5000. Without any doubt, the accuracy will converge to 100% as N goes to infinity.

VI. Simulation



This manipulator avoids all the obstacles and pick these targets sequentially.

You can see the video on: https://youtu.be/ktayu8AEWbM

VII. Conclusion

This method does work for motion planning in complex environments. However, when the geometric structures are not very complex, we would better not add a predictor to it.

The precision and performance of the industrial robot in a production line has always been vital to the overall efficiency of a factory. A self-planning manipulator will be able to seize, place, combine anything in determined positions and directions. Which may boost the efficiency for manufacture, logistic and assemble.

VIII. Future Extension

Heuristic Algorithms

Nowadays, more and more heuristic motion planning algorithms are proposed. Whereas none of them can reduce the order of magnitude of running cost. A good news is that no one has proved the lower bound of running cost. Finding a better algorithm is still possible. Perhaps we can get real-time query with high precision in the future by improving the algorithm.

Human-Like Learning

In this case, we have combined the SVM method with PRM*. However, consider '***How human beings avoid obstacles?***' I guess the answer is: We see the obstacles, and we act. Go deep into this phenomenon, the light stimulates optic nerves, then the electrons flow from dendrites to axons. It's a neural network.

Compare a human to a robot. The input is light, output is action. A robot can also get the information by recognition, perception or something else. So, it can also learn how to avoid obstacles by artificial neural networks.

The most important thing is, just like a human being, a well-trained neural network can adapt to any environment. i.e. When environment changes, the parameters, weights remain. No need to train it again. Hence that, It could easily handle the dynamic motion planning problem even in a terribly complex environment. It's a precious property.

A brain has 900,000,000 neurons, it's impossible for an artificial neuron network, so there is still a long way for computers. As the scale of integrated circuit decreases, the efficiency of CPUs increases. Perhaps one day motion planning will never be a problem.

References

[1] L. E. Kavraki, P. Svestka, J. C. Latombe, M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces", *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566-580, Aug. 1996.

[2] LaValle S M. Rapidly-exploring random trees: A new tool for path planning[J]. 1998.

[3] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning[J]. The International Journal of Robotics Research, 2011, 30(7): 846-894.

[4] Bialkowski J, Karaman S, Otte M, et al. Efficient Collision Checking in Sampling-Based Motion Planning[M]//Algorithmic Foundations of Robotics X. Springer Berlin Heidelberg, 2013: 365-380

[5] Burns B, Brock O. Sampling-based motion planning using predictive models[C]//Proceedings of the 2005 IEEE International Conference on Robotics and Automation. IEEE, 2005: 3120-3125.

[6] Suykens J A K, Vandewalle J. Least squares support vector machine classifiers[J]. Neural processing letters, 1999, 9(3): 293-300.

[7] Tor S B, Middleditch A E. Convex decomposition of simple polygons[J]. ACM Transactions on Graphics (TOG), 1984, 3(4): 244-265.

[8] Boyd S, Vandenberghe L. Convex optimization[M]. Cambridge university press, 2004.

[9] Fredman M L, Tarjan R E. Fibonacci heaps and their uses in improved network optimization algorithms[J]. Journal of the ACM (JACM), 1987, 34(3): 596-615.

[10] Bottou L. Large-scale machine learning with stochastic gradient descent[M]//Proceedings of COMPSTAT'2010. Physica-Verlag HD, 2010: 177-186.

[11] Gelfand I M, Silverman R A. Calculus of variations[M]. Courier Corporation, 2000.

[12] Kalakrishnan M, Chitta S, Theodorou E, et al. STOMP: Stochastic trajectory optimization for motion planning[C]//Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011: 4569-4574.

[13] Hargraves C R, Paris S W. Direct trajectory optimization using nonlinear programming and collocation[J]. Journal of Guidance, Control, and Dynamics, 1987, 10(4): 338-342.