Application of 6-Dof Robot Motion Planning in Fabrication

Hao Wu, Ming Lu, XinJie Zhou, and Philip F. Yuan^(✉)

College of Architecture and Urban Planning, Tongji University, 1239 Siping Rd.,
Shanghai, China

philipyuan007@tongji.edu.cn

Abstract. In practical robotic construction work, such as laying bricks and painting walls, obstructing objects are encountered and motion planning needs to be done to prevent collisions. This paper first introduces the background and results of existing work on motion planning and describes two of the most mainstream methods, the potential field method, and the sampling-based method. How to use the probabilistic route approach for motion planning on a 6-axis robot is presented. An example of a real bricklaying job is presented to show how to obtain point clouds and increase the speed of computation by customizing collision and ignore calculations. Several methods of smoothing paths are presented and the paths are re-detected to ensure the validity of the paths. Finally, the flow of the whole work is presented and some possible directions for future work are suggested. The significance of this paper is to confirm that a relatively fast motion planning can be achieved by an improved algorithmic process in grasshopper.

Keywords: Motion planning · Robot · Fabrication · Brick · Grasshopper

1 Introduction

1.1 Motion Planning

With the increasing variety of architectural forms, more and more buildings need to rely not only on procedurally generated forms but also on the assistance of machines or robots to do bricklaying and wall painting. In the case of bricklaying, for example, many successful projects have been built, such as the curved facade of the West Coast Pool House (Fig. 1). Since most of these projects are built on-site, robot obstacle avoidance, or robot motion planning, needs to be considered. Robot motion planning is a long-standing problem that is still being studied today, and a balance between computational efficiency and reliability needs to be found in practical projects.

In traditional robot construction, robot path planning can be taught manually point by point, but this is very time-consuming and labor-intensive. Nowadays, the mainstream approach is to make the robot's motion in the simulation, adding auxiliary points to avoid the obstacles in the program, and then generate the offline program. However, in this case, the user still needs to manually add auxiliary points and check whether the whole

Fig. 1. XiAn Chi She brick wall

process encounters collisions. So a better approach is to automate the motion planning to do the job.

In building construction, real-time has a great impact on efficiency, but the algorithm of motion planning is a very time-consuming and complex task. There are a large number of papers that study motion planning algorithms, and there are also many software and libraries that implement most of the mainstream algorithms.

1.2 Previous Work

The algorithms for motion planning are divided into sampling point-based and potential field-based. Most motion planning libraries use sampling-based algorithms. The most famous work is the OMPL library, which is based on PRM [1], RRT, EST, SBL, KPIECE, SyCLOP algorithms, and several variants developed from them. Also, the OMPL library can work with many software such as Openrave, MoveIt, etc. Similar to OMPL, there are also planners such as CHOMP and STOMP. All of the above need to be used in Linux systems and require a certain knowledge base.

Vrep has the OMPL library as a built-in plug-in, and RoboDK comes with its own PRM motion planner. However, most architects are familiar with these sorts of software, they use Rhino as a robot simulation environment and program using grasshopper, and use software such as Kuka PRC and FURobot to control the robot to work. The work in this paper is to implement path planning in the FURobot environment.

2 Research

2.1 Potential Field

Obstacles and target poses are known, so a potential field is formed by setting the obstacles to exert repulsive forces on the robot and setting the target poses to exert attractive forces on the robot. This method is the potential field method. The potential field method obtains the gradient according to the external situation and descends step by step to the lowest point, which is the target point, according to the guidance of the gradient. The advantage of the potential field method is that it does not require extensive calculations, so it can be done in real-time.

However, after actual experiments, it is found that the potential field method needs to set at least the coefficients of attraction, repulsion, and descent rate to establish a suitable potential field, so setting these three coefficients is the key. In the case of ordinary moving path planning with a two-link robot arm, the robot's configuration space is two-dimensional, so the potential field is three-dimensional, and it is possible to clearly see the three-dimensional shape of the potential field to adjust these three coefficients. However, in a high-dimensional configuration space, such as the six-dimensional configuration space of a 6-degree-of-freedom robot, the generated potential field is 7-dimensional, so it is difficult to adjust the three parameters to obtain a suitable potential field.

Finally, the potential field method has local minima, which will make the potential field method unable to reach the target point poses. In summary, PRM based on sampling points is chosen in this paper.

2.2 PRM

PRM [1] first creates a series of randomly different poses and eliminates those poses that encounter external objects and keeps the collision-free poses. After establishing a certain number of sample points, PRM searches for neighboring sample points around each sample point and connects the two sample points when found. If the motion path between these two sample points is collision-free, the connection of these two sample points can be added to the whole sample point map. Once all the sample points are connected, the Astar algorithm can be used to find the shortest route.

Since rotating 0 degrees is the same as rotating 360 degrees (the angles are in degrees), the following distance function is needed:

$$Dist(\theta_{goal} - \theta_{start}) = \min(|\theta_{goal} - \theta_{start}|, (360 - |\theta_{goal} - \theta_{start}|))$$

Because the energy required to rotate each axis is different, distance weights can also be added to these six-axis joints (six-axis joints per pose Ang), with specific weight values C that can be set according to the energy consumption and range of influence of the rotation.

$$Dist(Ang_{goal} - Ang_{start}) = \sum_{i=1}^6 C_i * Dist(\theta_{istart} - \theta_{igoal}).$$

3 Implement

3.1 Collision

As mentioned above, when building a map of sampled points, collision detection needs to be performed for each sampled point. In practice, using the mesh-to-mesh collision detection in Rhino is too time-consuming, so the method needs to be simplified. Specifically, the robot parts are simplified to simple geometry, such as cylinders or spheres, and the external collision objects are simplified to multiple points, and then these basic geometric objects are used for collision detection, which will greatly improve the detection efficiency.

The robot's A2, A3, A4, and A6 joint positions are extracted and connected to form three axes, and by setting a safety radius, such as 200 mm for each axis, a cylinder

can be formed correspondingly. In collision detection, choosing the cylinder speeds up the calculation. The collision detection is calculated by detecting whether the shortest distance between the external collision point and the axis of the cylinder exceeds the sum of the radius of the cylinder and the radius of the external collision point [2]. The A1 joint is not included because in general work, the robot is stationary and the range of motion of the A1 joint is not large, so the detection can be ignored.

In addition to the robot's own parts, the tool head also needs to perform collision detection on external objects. But the shape of the tool head is rather irregular. By default, we use a line segment from the flange plate to the tool head coordinates as the axis and set a relatively large safety radius. However, this is not a good simulation, and it would be more appropriate to add multiple cylindrical axes and corresponding safety radius to the tool head (Fig. 2).

Fig. 2. Using different cylinders as collision detection objects for robots

After the sample points are created, it is necessary to find neighboring sample points for each sample point and then test the lines between them for collisions. It is not appropriate to test the entire line segment for a large number of collisions, but rather to test as few as possible to ensure that there are no collisions. Such tests are not even necessary when the distance between sampling points is small.

Once the sample points are created, they can be saved, since they are valid as long as no new obstacle objects are added and only the robot's starting and target poses are changed. It has been tested that the time required to create sampling points is typically three to five times longer than the time required to calculate the shortest path.

3.2 Trajectory

3.2.1 By Grasshopper

The shortest path derived by the Astar algorithm is not a smooth path, we can use a polynomial least-squares fit or use Rhino's spline curve to smooth it.

The specific steps to use rhino's spline curve are:

1. Take the components of the first three axes A1, A2, A3 of the robot arm as the x,y,z components of the first 3d point. Then take the components of the last three axes A4, A5, and A6 as the x,y,z components of the second 3d point.

2. Interpolate these three components separately for the spline curve, which can be done either through the grasshopper's module or through Rhino API's CreateInterpolatedCurve function, which gives two curves because there are two lists of points.
3. The fitted two curves are then averaged and a new list of points is obtained for each of them. The x,y,z components of the decomposition of the points in these two-point lists are taken as A1, A2, A3, and A4, A5, A6.

The path through the fit will change the pose of the shortest path generation (Fig. 3), so it is better to check the collision again.

This trajectory smoothing approach is relatively easy to implement, but requires the use of Rhino's existing commands, and in general situations, other approaches may be chosen.

Fig. 3. Smooth path after fitting

3.2.2 Polynomial

After check, it is feasible to use polynomials to find the smooth path. As an example, we first obtained 5 poses by PRM, and we can list the following polynomials and combine them into a matrix.

$$X = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 & t_0^6 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 & 6t_0^5 \\ 1 & t_1 & t_1^2 & t_1^3 & t_1^4 & t_1^5 & t_1^6 \\ 1 & t_2 & t_2^2 & t_2^3 & t_2^4 & t_2^5 & t_2^6 \\ 1 & t_3 & t_3^2 & t_3^3 & t_3^4 & t_3^5 & t_3^6 \\ 1 & t_4 & t_4^2 & t_4^3 & t_4^4 & t_4^5 & t_4^6 \\ 0 & 1 & 2t_4 & 3t_4^2 & 4t_4^3 & 5t_4^4 & 6t_4^5 \end{bmatrix}$$

We divide the time equally among the 5 poses and assume that the robot takes 1 s to go through the 5 poses, so we have:

$$t_0 = 0, t_1 = 0.25, t_2 = 0.5, t_3 = 0.75, t_4 = 1$$

t_i represents the time of movement to the i th pose. Rows 2 and 7 of X represent the axial joint velocities at the starting and final pose, respectively, which are stationary at both poses, so they are 0. The remaining rows are the values of the joint angles corresponding to the poses in which they are located, so we have:

$$A = [A_0 \ 0 \ A_1 \ A_2 \ A_3 \ A_4 \ 0]^T$$

Set the polynomial coefficients:

$$B = [B_0 \ B_1 \ B_2 \ B_3 \ B_4 \ B_5 \ B_6]^T$$

Since XB equals A , and X is a square matrix, it is straightforward to solve for B by inverse. This is one of the reasons why the X matrix is designed in this way. In this case, the number of known poses is 5, so the dimension of the X matrix is the number of known poses plus 2(two velocity constraints), and so on for different number of known poses from PRM.

In the case of real-time control of the robot, the advantages of this method over the previous ones are:

1. Ensure that the robot's starting and final joint speeds are zero.
2. make sure that the values of the joint axes are consistent with the preset ones when going through the previously derived pose (Fig. 4). Here, taking a particular axis as

Fig. 4. Polynomial, PRM and curve interpolate

an example, it can be found that the curve interpolation does not conform well to the pose from the PRM, while the polynomial interpolation does.

3.3 Check

After doing the path planning, the path can be checked once more depending on the situation. Global collision detection can be performed, or some points that may create problems can be identified, and then detection can be performed on those points. For example, the distance between the two endpoints of the alternative collision cylinder axis of the robot body including the tool head and the point clouds of all surrounding obstacles can be calculated for all sampled points, and then collision detection is performed for some sampled points with the smallest distance. If a problem is detected, a new shortest path is generated and detected again by changing the random number seed that generates the map of sampled points and performing a new calculation.

4 Example

Here is an example to briefly introduce the workflow on bricklaying. The robot is laying bricks and taking bricks at the same time, if there is already an obstacle, then path planning is needed, but even if there is no obstacle, the height of the whole brick wall is constantly getting higher during the process of laying bricks, so it can also be considered as a changing obstacle. Of course, it is possible to write a program manually to follow the height of the brick wall for path presetting, but this increases the workload of the program, so motion planning can be used to automatically generate the brick moving motion (Fig. 5).

Fig. 5. Automatic generation of brick picking and bricklaying movements through motion planning

4.1 Component

The most important part of the whole process is the motion planning component of FURobot, which has the following input interface (Fig. 6):

1. Robot: Enter a robot with a tool head, which can be ABB, KUKA or UR robot.
2. Start angle: In general, the robot's start point pose is defined by the tool head coordinates, and inverse kinematics is necessary to obtain the corresponding pose. The pose can be obtained in FURobot by using the inverse kinematics component or Core.
3. Obstacle object point cloud and radius: All objects are abstracted as numerous spheres, which has the advantage that collision detection is very easy to handle. For simple processing in this input, all spheres have the same input radius.
4. Seed: The generated sampled point map can be changed by adjusting the value of the seed.

Fig. 6. Motion planning component of FURobot

5 Future Work

In the Grasshopper and FURobot environments, using the improved method described above, a real-time (as fast as 2 hundred milliseconds and as slow as 1–2 s including recalculation of the sampled map CPU i7 2.7GHz) path planner can be made and can be used in offline or online construction projects.

Next, the motion planning component can be encapsulated into a single instruction, which becomes a motion instruction along with the existing straight line instruction, point-to-point instruction, etc. The difference is that the path planning instruction reacts to the motion planning component in the offline or online construction project. The difference is that the motion planning instruction reacts to the actual program as a sequence of angular instructions that guide the entire obstacle avoidance process. This significantly reduces the amount of work and the number of components in the Grasshopper programming.

The PRM algorithm may not be the best choice. In future work, the RRT algorithm, or other algorithms that can be processed in parallel, can be tried and it is also possible to parallelize the creation of sampled maps.

References

1. Kavraki, L.E., Švestka, P., Latombe, J.-C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* **12**, 566–580 (1996). <https://doi.org/10.1109/70.508439>
2. Ericson, C.: *Real-Time Collision Detection*, 593 p. Elsevier, Amsterdam, Boston (2005). (Morgan Kaufmann series in interactive 3D technology)
3. Chai, H., Zhang, L., Yuan, P.F.: Advanced timber construction platform multi-robot system for timber structure design and prefabrication. In: Yuan, P., Xie, M., Leach, N., Yao, J., Wang, X. (eds.) *Architectural Intelligence*, pp. 129–144. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-6568-7_9
4. Yuan, P.F., Chen, Z., Zhang, L.: Design optimum robotic toolpath layout for 3-D printed spatial structures. In: Yuan, P.F., Xie, Y.M., Yao, J., Yan, C. (eds.) *CDRF 2019*, pp. 322–330. Springer, Singapore (2020). https://doi.org/10.1007/978-981-13-8153-9_29
5. Ji, Z., Lin, Y.: Path-optimizing agent-based earthwork system: a microscopically precise earthwork system that is adaptable to any form of landscape. In: Yuan, P.F., Xie, Y.M., Yao, J., Yan, C. (eds.) *CDRF 2019*, pp. 355–363. Springer, Singapore (2020). https://doi.org/10.1007/978-981-13-8153-9_32
6. *From Diagrammatic Thinking to Digital Fabrication*. Tongji University Press, Shanghai (2016)
7. Yuan, P.F., Menges, A., Leach, N.: *建筑机器人建造* (2015)
8. Yang, F., Chakraborty, N.: Chance constrained simultaneous path planning and task assignment for multiple robots with stochastic path costs. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6661–6667 (2020)
9. Ichter, B., Schmerling, E., Lee, T.-W.E., Faust, A.: Learned critical probabilistic roadmaps for robotic motion planning. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9535–9541 (2020)
10. Alam, M.S., Rafique, M.U., Khan, M.U.: Mobile robot path planning in static environments using particle swarm optimization. [arXiv:2008.10000](https://arxiv.org/abs/2008.10000) (2020)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.