

Robot Motion Planning: A general Review

Siwei Xie

st165900@stud.uni-stuttgart.de

Abstract. Motion planning is an essential research topic in robotics, and its research results have been applied into diverse applications. This seminar work begins with an introduction and a review of the state of the art in robot motion planning. Then the basic concepts and different methods will be presented, and the focus will be the two philosophies in motion planning methods – sampling-based planning and combinatorial planning. In closing, one in practice widely used framework – MoveIt! will be introduced.

Keywords: Motion planning · Sampling-based planning · Combinatorial planning.

1 Introduction

Robot Motion Planning, just as its name implies, exists to compute a route for the robot, so that it can move from the start location to the target location without collision, self-collision also not included. To accomplish a successful robot motion planning task, it usually goes through two phases - *path planning* and *path optimisation*.

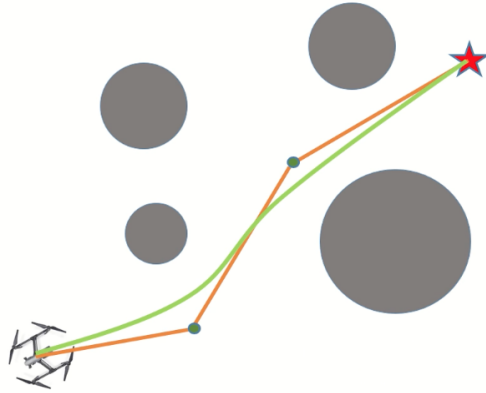


Fig. 1. The orange line is the output of *path planning* and the green curve is the optimized path (trajectory) that will be eventually used for the drone.

Path planning illustrates how to figure out a path that builds a connectivity between departure and destination inside a predefined working space. Normally, the output of conducting a *path planning* process is a composition of line segments that are in head-to-tail arrangement. Considering that the robot itself has dynamic and kinematic constraints, such a path may cause unnecessary and jerky motions or does not follow user defined criteria, which means the robot must adjust its configuration at junctures in order to adapt the line conditions.

Path optimisation, based on these line segments, generates a fitted time-varying curve by applying different optimisation algorithms (e.g. *Minimum Snap*) to output a path in different geometrical representations (e.g. *Bézier curves*). Therefore, the ultimate result can be reckoned as a more hypothetically ideal trajectory for the robot.

This seminar work, mainly focuses on *path planning*, including basic concepts and various methods, and all below mentioned *robot motion planning* de facto refers to *path planning* and does not cover *Path optimisation*. Moreover, it is also valuable to give an insight into those functional, in industry widely used tools.

2 The State of the Art

The most active fields in robot motion planning are always around obstacles avoidance, shortest path planning, target following, unknown area exploring, etc. Penetrating in research and refinements of methods in those aspects promotes the development of robotics. Here is a classification of corresponding methods, techniques or algorithms:

- **Obstacle Avoidance[1]:** Classical Methods: Cell Decomposition, Potential Field Methods; Heuristic Methods: Fuzzy Logic Controller Technique, Neural Network Technique, Neuro-fuzzy Technique, Genetic Algorithm, Ant Colony Optimisation Technique, Machine Learning or Predictive Approach
- **Shortest Path Planning[2]:** Dijkstra’s Algorithm, Floyd-Warshall Algorithm, Bellman-Ford Algorithm, Genetic Algorithm
- **Target Following:** Combined use of depth sensors and inertial sensors for 3D ranging and tracking[3]; Simulation-based methods for multi target tracking, e.g. sequential Monte Carlo (SMC), Particle filtering (PF)[4]
- **Unknown Area Exploring (relate to SLAM and Multi-Robot System)[5] :** A collaborative localisation scheme within multi-robot systems under the help of various sensors for purpose of accelerating the exploration and improving the accuracy of positioning:
 1. Location : each robot locates itself individually.
 2. Communication: the robots share their location and relative distance.
 3. Improve accuracy: incorporate with dynamic adaptive noise covariance.

3 Definition of Motion Planning

Before we dig deeper into different kinds of methods used in automated industrial environment, we may need firstly to walk through several terminologies that construct the definition of a basic robot motion planning problem step by step. They are quite beneficial for us to understand how those methods work and thereby draw inferences.

3.1 Configuration Space

Intuition In a lot of algorithmic path planning studies, we are habituated to treating the target object as a free mass point in order to simplify the complex planning scenarios, but in fact, robots are complex pieces of hardware whose whole body and assembly units are restricted to certain kinematic or dynamic coupling constraints, which means it doesn't move arbitrarily as we thought it would. Meanwhile, the obstacles that may obstruct the connectivity of a path and the potential interaction between robots and obstacles should also not be ignored. Upon these issues, we require a more powerful specification of space rather than a plain "workspace".

Definition The *configuration Space* (also called \mathcal{C} -space) introduced by the seminal paper of Lozano-Pérez[6] refers to the state space for robot motion planning, which is a set of possible transformations that could be applicable to the robot movements. Or said differently, if a robot configuration stands for a vector that records the positions and orientations of all robot points relative to a fixed coordinate system, then the \mathcal{C} -space is the aggregation of all available robot configurations, which is described as a topological manifold and usually not that of a Cartesian space.



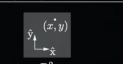
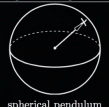
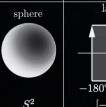
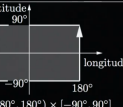
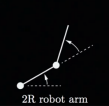
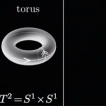
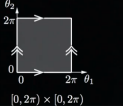



system	topology	sample representation
 point on a plane	 \mathbb{E}^2	 \mathbb{R}^2
 spherical pendulum	 S^2	 $[-180^\circ, 180^\circ] \times [-90^\circ, 90^\circ]$
 2R robot arm	 $T^2 = S^1 \times S^1$	 $[0, 2\pi) \times [0, 2\pi)$
 rotating sliding knob	 $\mathbb{E}^1 \times S^1$	 $\mathbb{R}^1 \times [0, 2\pi)$

Fig. 2. Examples: Simple systems geometrically represented in topological space[7]

Modelling the obstacle region \mathcal{C}_{obs} and free space \mathcal{C}_{free} In the case of without collision constraints, we can directly conduct motion planning algorithms in the attainable configuration space \mathcal{C} . Nevertheless, obstacles are inevitable in the real world, so we are also concerned to avoid robots colliding with obstacles. Generally, the first step is to compute the obstacle region \mathcal{C}_{obs} and remove this infeasible section from \mathcal{C} , then the leftover space is the so-called free space \mathcal{C}_{free} , where in true sense a solution path should be allowed to traverse. It is essential to understand how to model \mathcal{C}_{obs} . When using combinatorial methods to solve the motion planning problem, constructing the representation of \mathcal{C}_{obs} is the first step needed to succeed. While using sampling-based methods, the deep comprehension in constructing \mathcal{C}_{obs} helps us understand why such a method should be used or avoided in view of efficiency and complexity. Here displays two examples:

– **A polygonal \mathcal{C} -space obstacle, translation only**

The *star algorithm* is used for determining \mathcal{C}_{obs} in the context of a convex polygonal robot \mathcal{A} moving around a convex polygonal obstacle \mathcal{O} in a 2D space. \mathcal{C}_{obs} can be built incrementally by adding boundaries that comes from sliding the robot \mathcal{A} along the edge of the obstacle \mathcal{O} .

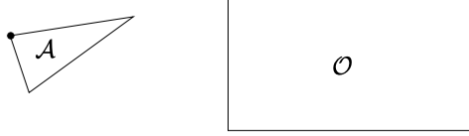


Fig. 3. A triangular robot and a rectangle obstacle[8]

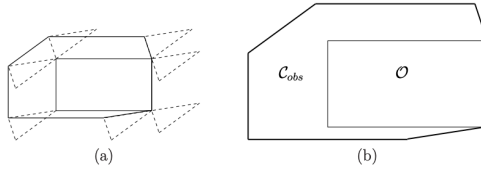


Fig. 4. The computed obstacle region \mathcal{C}_{obs} [8]

– **A nonlinear \mathcal{C} -space obstacle, translation and rotation**

If the robot consists of multiple bodies, the \mathcal{C}_{obs} will be more complicated to specify. Here shows the example of a robot with two revolute joints moving in a 2D space that contains several polygonal obstacles.

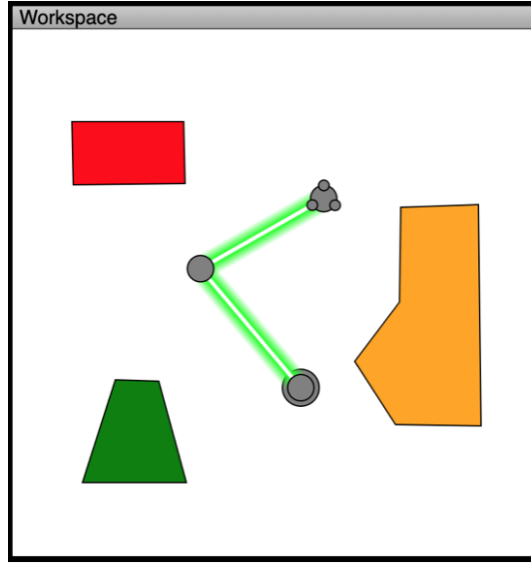


Fig. 5. A robot with two revolute joints moving in a 2D workspace [10]

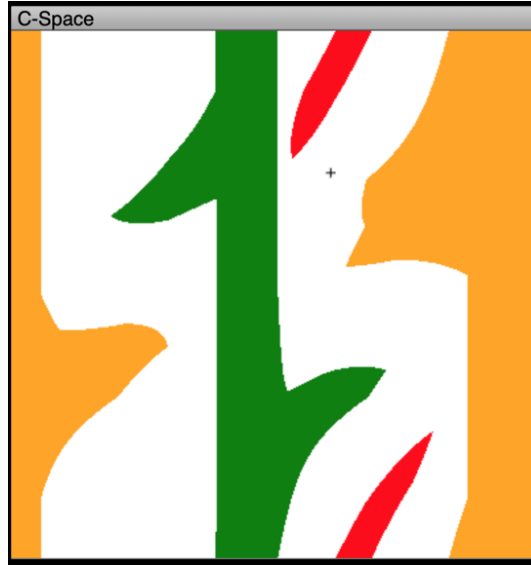


Fig. 6. Automatically-generated \mathcal{C}_{obs} . Blocks of different colour correspond to different obstacles.[10]

3.2 Definition of a basic motion planning problem

By virtue of using the concept of *C-space*, it is simple to formalise the basic motion planning problem and this problem is proved to be *PSPACE-hard* by Reif[13].

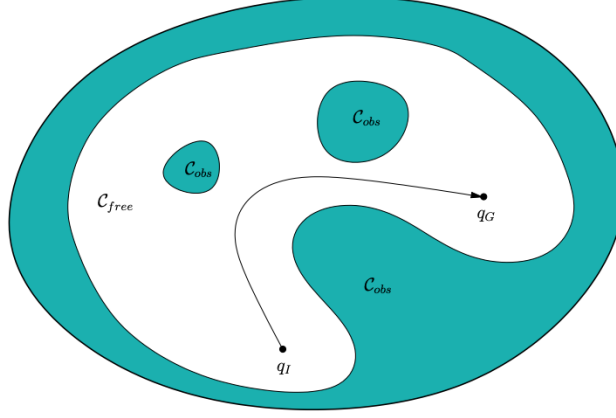


Fig. 7. Definition of a basic motion planning problem[8]

1. A workspace \mathcal{W}
2. The union of all obstacles $\mathcal{O} \subset \mathcal{W}$
3. A robot \mathcal{A} , which may be a simple rigid body, or more complicated, e.g. a multi-arm robot
4. The corresponding configuration space \mathcal{C} , obstacle region \mathcal{C}_{obs} , and free space \mathcal{C}_{free} , determined from \mathcal{W} , \mathcal{O} and \mathcal{A} .
5. The *initial configuration* and the *goal configuration* of the robot, $q_I, q_G \in \mathcal{C}_{free}$, and the pair (q_I, q_G) is called a *query pair*.
6. A complete algorithm should return a continuous path that only traverses in \mathcal{C}_{free} and whose start and end are overlapping q_I and q_G respectively, i.e. $\tau : [0, 1] \rightarrow \mathcal{C}_{free} \wedge \tau(0) = q_I \wedge \tau(1) = q_G$ or correctly tells that no solution exists.

4 Methods of Motion Planning

To solve robot motion problems, continuous space need to be discretised, and very importantly, the primitive connectivity information should be rigorously maintained, so that a *planner* — an algorithm that automatically computes the route, can conduct a correct discrete graph-based search. In fact, the aim of multiple motion planning methods is to create such a graph called *roadmap* \mathcal{G} .

4.1 Roadmap[8]

Definition Let \mathcal{G} be a topological graph that concisely represents \mathcal{C}_{free} , whose vertex symbolises a configuration in \mathcal{C}_{free} and whose edge is a path without collision wandering in \mathcal{C}_{free} . Supposing there is a query pair (q_I, q_G) , the graph \mathcal{G} is called a roadmap if it meets following two requirements:

1. **Accessibility:** there exists a path from q_I to some $q'_I \in \mathcal{G}$ and a path from some $q'_G \in \mathcal{G}$ to q_G . (Graphical expression: Vertex q_I connects to Vertex q'_I ; Vertex q'_G connects to q_G .)
2. **Connectivity-preserving:** there exists a path from aforesaid q'_I to q'_G (Graphical expression: Vertex q'_I connects to q'_G .)

Different Goals Different methods of constructing a *roadmap* are aimed at different goals. Generally, they can be fall into two categories:

- **Maximum-Clearance Roadmaps:** the path should be as far as possible from \mathcal{C}_{obs} .
- **Shortest-Path Roadmaps:** the path length should be as short as possible, of course perforating in the \mathcal{C}_{obs} is not allowed.

Note that these two goals are in conflict. Maximum-clearance roadmaps decrease the possibility of collision by enlarging the distance; while the shortest-path roadmaps can give a more efficient solution but at high risk of touching the obstacles.

4.2 Classification of Motion Planning Algorithms

For discretisation, there are mainly two philosophies — *combinatorial motion planning* and *sampling-based motion planning*. *Combinatorial motion planning* lays emphasis on explicitly characterising the geometrical representation and the potential transformation that can be applied to the robot, which means the connectivity of \mathcal{C}_{free} can be exactly seized by a *roadmap*; while *sampling-based motion planning* evades these issues, but relies on a collision detection module as a black box and a sampling scheme to incrementally form the \mathcal{C}_{free} , which implies the graph it constructs is kind of random and simply an approximative roadmap of \mathcal{C}_{free} , whose accuracy depends on the density of the sampled configurations. The philosophy of “sampling and discrete searching” boosts the development of motion planning algorithms because it detaches specific geometric and kinematic models from motion planning, even in a more tortuous workspace with a more complex robot it could be comparatively functional.

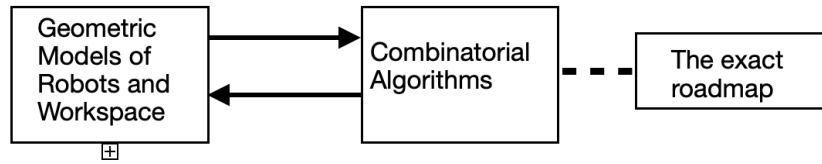


Fig. 8. The combinatorial planning philosophy

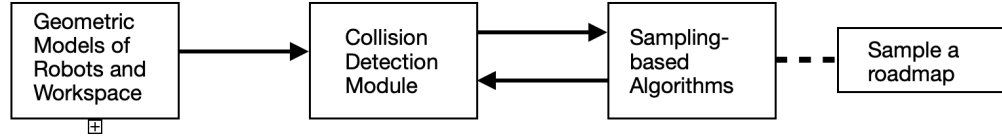


Fig. 9. The sampling-based planning philosophy

4.3 Combinatorial Motion Planning vs. Sampling-based Motion Planning

Completeness[8] *Completeness* is a criterium used to assess motion planning algorithms, or rather, how possible it is to solve the problem when the given algorithm is applied. More specifically, three different notions of completeness are defined, from strong to weak:

- *complete*: Given any input, the algorithm always correctly outputs whether there is a solution, which means if there exists a solution, it must be found in finite time.
- *resolution complete*: If there exists a solution, the algorithm has to return one in finite time; however if there exists none, the algorithm may be trapped in nontermination.
- *probabilistically complete*: The probability that the algorithm will find a solution converges to 1 as the number of iteration approaches infinity.

Theoretically, all the *combinatorial motion planning* methods are able to be *complete*, despite the fact that the space and time complexity might be too disadvantageous to be omitted when confronting millions of geometric primitives, or even impossible to explicitly represent \mathcal{C}_{obs} . From another perspective, it indicates that the algorithms which are much more efficient but not invariably output a proper solution should be tolerated. Majority of them are *sampling-based motion planning* methods, which are able to solve previously unsolved problems, and thereby more widely used in industry. The two weaker notions are especially referring to them.

Applicability Compared to *sampling-based methods*, the functionality of *combinatorial methods* is more dependent on the input of the algorithms. What is the dimension of the workspace? What kind of geometrical representations are used for the robot, obstacles and the whole workspace? What transformations can be applied to the robot? Is the robot convex? Are the obstacles convex. The answers to these questions must be carefully figured out in order to make the most use of their advantages. Generally, *combinatorial methods* are predominantly suitable for those cases that have certain features, such as low dimensionality

and convex models. It can as well be used to test a special case of a motion planning problem in many applications. The special case simplifies the model of the generalised case, so the fundamental geometric properties are outstanding and thus can be utilised. Furthermore, for research interest, these algorithms present the theoretical upper bound for time complexity.

When the dimension of \mathcal{C} -Space increases and the model becomes more complex, by using *combinatorial methods*, it is difficult to handle the situation from the beginning – explicitly construct \mathcal{C}_{free} , let alone the exploitation in both time and space complexity. Under the circumstances, the preference of *sampling-based methods* are predictable in despite of a weak guarantee of finding a solution. After all, testing whether a specified configuration is located in \mathcal{C}_{free} by a collision detection module is much more effortless and quicker than straight forwardly producing the geometric model of \mathcal{C}_{free} . However, we should be aware of a fact, complicated problem are still intractable for *sampling-based methods* to solve.

In conclusion, understanding both methods are worthwhile and helps us analysing concrete situations to achieve a trade-off. Next, popular methods that implement these two philosophies will be introduced in detail.

4.4 Combinatorial motion planning methods

Note that *combinatorial methods* are complexity-limited to simple problems. In higher dimensions or with complicated obstacles and robots, a deterministic problem will be solved in exponential time, which is quite impractical. Therefore, it does not lose generality that we currently work over a point robot moving in a two-dimensional world only furnished with polygonal obstacles. Here presents three methods:

The Visibility Graph

Create a graph as follows:

- **Nodes:** All vertices of obstacles \mathcal{C}_{obs} and the query pair (q_I, q_G) construct the set of nodes.
- **Edges:** If one node is “visible” to another node, then these two nodes should be directly connected with an edge, and all obstacle edges do duty for edges in the visibility graph.

In practice, the visibility graph method is not safe enough, because the robot is extremely close to the obstacles and no error space is left for uncertainty, which violates the rule of “maximal clearance”. Besides, it cannot be scaled well to three dimensions. Shortest path in 2D can be guaranteed via vertices, but in 3D this is not the case.

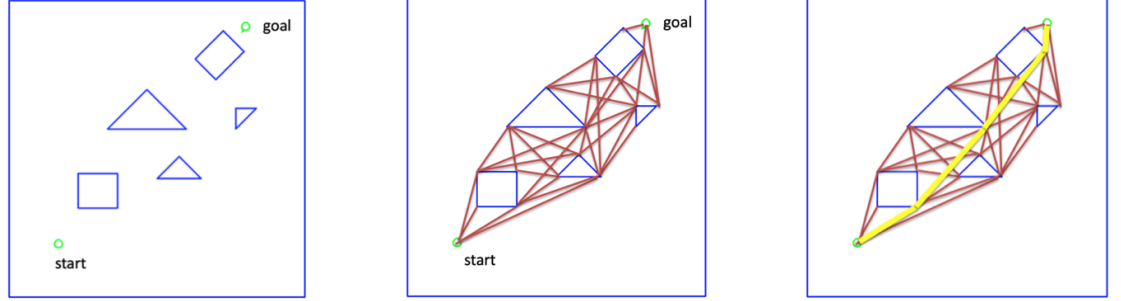


Fig. 10. Example: Construct a visibility graph[9]

Cell Decomposition

Create a graph as follows:

- **Compute Non-overlapping Decomposition:** The decomposition of \mathcal{C}_{free} can be executed through triangulation or trapezoidation, which makes no change to the original representation; or in an approximative way, for instance, firstly build up a regular grid and then decide whether the grid should be free or filled.
- **Select Nodes:** Select a node for each cell (e.g. the centroid) and other supported nodes.
- **Append Edges:** Connect those nodes who represent adjacent cells.

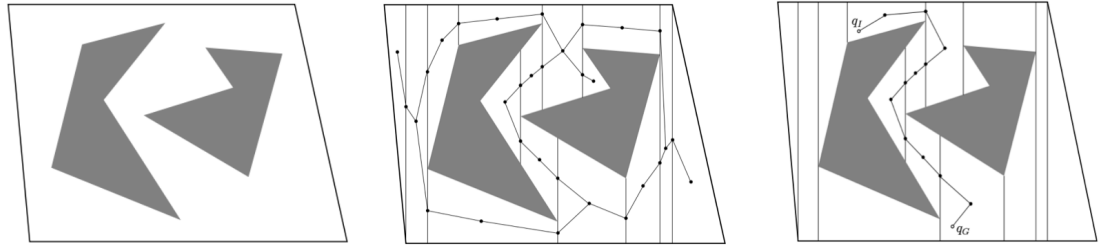


Fig. 11. Example: Conduct a cell decomposition through trapezoidation and find a path[9]

The decomposition plays an important role at this point. For exact decomposition, triangulation or trapezoidation are well-known computational geometrical problems that are hard to implement and time-consuming. Similarly, it is not

suitable for three dimensions. For approximative decomposition, it is efficient and expandable but the quality of the path it finds largely depends on the resolution of the graph.

Generalised Voronoi diagrams

The ordinary Voronoi diagrams are designed for points. Here we expand the options to polygons by discretising obstacles to a finite set of points or using a regular grid for a approximative representation. The generalised Voronoi diagrams should also keep certain features of “equidistance”:

- Each point on the edge of the roadmap is equidistant from two points on the obstacle boundaries.
- The intersection point of two or more edges is equidistant from three or more points on the obstacle boundaries.

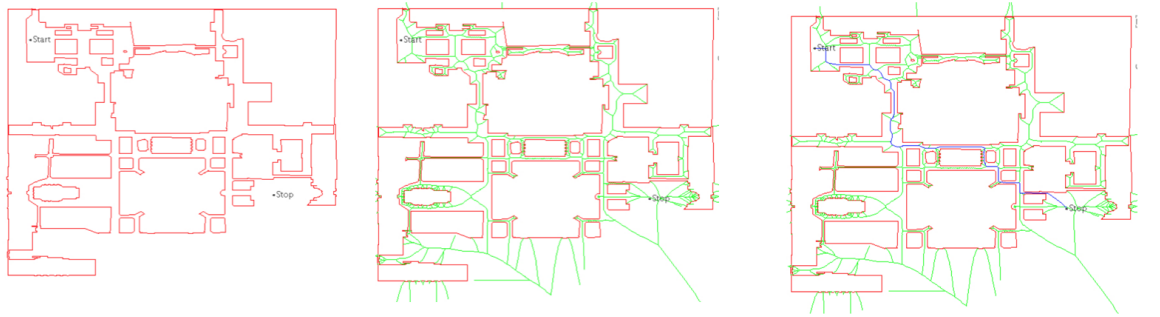


Fig. 12. Example: Compute a Voronoi diagram and find a path[9]

What calls for a special attention is that generalised Voronoi diagrams compute a *maximum-clearance* path roadmap. Therefore, it can be a functional planner for mobile robots that need to carry out an uncertain task with the assistance of fast computing methods, but can not be a good choice for open space exploration.

4.5 Sampling-based motion planning methods

Most of *single-query, sampling-based motion planning algorithms* can be summarised into the following general framework[8]:

- **Initialize the Graph:** Create an undirected search graph $\mathcal{G}(V, E)$; Add q_I , q_G , or both and other vertexes definitely in \mathcal{C}_{free} into V .
- **Select a Node:** Select a node $q_{cur} \in \mathcal{C}_{free}$ as the start point of subsequent sampled path.

- **Local Planning:** Choose some $q_{new} \in \mathcal{C}_{free}$ and try to construct a path between q_{new} and q_{cur} . Then the path needs to be discretized to a set of configurations, and the collision check module must check whether all configurations are located in \mathcal{C}_{free} . If collision exists, then return to step 2.
- **Insert an Edge:** Insert the edge from last step in to E . If q_{new} is still not in V , add it to V too.
- **Solution Check:** Check whether \mathcal{G} contains a solution path from q_I to q_G . (The solution can be more than one, unless it is a *single search tree*.)
- **Return to Step 2:** When the termination condition is not satisfied, continue the iteration.

The essence of the process mentioned above can be briefly summarized as "sampling and collision check". For sampling a new point, normally a priority queue is utilized for a optimal solution. For collision checking, the collision detection module is a critical component, although we treat it as a black box. In majority of sampling-based motion planning methods, collision checking is the most time-consuming part. Here presents two fundamental methods:

Probabilistic roadmaps (PRM)[9]

Basic Algorithm:

Algorithm 1: PRM

Input: Configuration Space \mathcal{C}

Output: Roadmap $\mathcal{G}(V, E)$

```

 $\mathcal{G} \leftarrow \emptyset$ ;
while not terminal condition do
     $q_{rand} \rightarrow \text{RANDOM\_CONFIG}(\mathcal{C})$ ;
    if CLEAR( $q_{rand}$ ) then
         $\mathcal{G}.\text{addVertex}(q_{rand})$ ;
         $\mathcal{N}_{near} \leftarrow \text{NEAREST}(\mathcal{G}, q_{rand})$ ;
        for  $q_{near} \in \mathcal{N}_{near}$  do
            if LINK( $q_{rand}, q_{near}$ ) then
                 $\mathcal{G}.\text{addEdge}(q_{near}, q_{rand})$ 
            end
        end
    end
end
end

```

Intepretation Firstly, initialize a empty graph \mathcal{G} . Then the function RANDOM_CONFIG() samples a new configuraion q_{rand} , and the function CLEAR(q_{rand}) checks whether q_{rand} is overlapping with \mathcal{C}_{obs} . If yes, sample a new point, if not, add it to V and find all points from V that are close to it (There are many strategies to find its neighbours, e.g. all the points inside a circle with a given radius). And then the function LINK() should be invoked for pairwise neighbour and the new sample point to test whether they can be connected. If the formed

line does not collide with \mathcal{C}_{obs} , then insert an edge between them. Iterate all above steps, until the terminal condition (e.g. find a solution path) is satisfied.

Rapidly exploring random trees (RRT)[9]

Basic Algorithm:

Algorithm 2: RRT

Input: Configuration Space \mathcal{C} , Initial Configuration q_I , Goal Configuration q_G
Output: Tree $\mathcal{G}(V, E)$

```

 $\mathcal{G}.$ addVertex( $q_I$ ) ;
while not reach  $q_G$  do
     $q_{rand} \rightarrow \text{RANDOM.CONFIG}(\mathcal{C})$ ;
    if CLEAR( $q_{rand}$ ) then
         $q_{near} \leftarrow \text{NEAREST}(\mathcal{G}, q_{rand})$ ;
        if LINK( $q_{rand}, q_{near}$ ) then
            |  $\mathcal{G}.$ addVertex( $q_{rand}$ )  $\mathcal{G}.$ addEdge( $q_{near}, q_{rand}$ )
        end
    end
end

```

Intepretation The initialization in RRT should add the initial configuration q_I to the vertex set V . Then, similarly to PRM, using the functions RANDOM.CONFIG() and CLEAR() to sample a proper new point. Afterwards, find the nearest point q_{near} rather than a neighbour set like in PRM and check the connectivity between q_{near} and q_{rand} . If no collision exists, add q_{rand} into V and insert an edge between q_{rand} and q_{near} . Iterate all above steps, until the goal configuration q_G can be reached.

The advantages and disadvantages of *sampling-based methods* are both obvious. They are more efficient but have weaker *completeness* or the quality of the solution path varies a lot. Hence, in practice, path pre-processing and post-processing technique should also be provided. Under some circumstances, e.g. q_G inside a narrow passage, the basic *sampling-based methods* are problematic to sample a path that is moving toward the goal. Therefore, based on fundamental PRM and RRT, many advanced algorithms with specialized sampling strategies are developed, so that it can probe reasonably confronting different circumstances.

In a word, *sampling-based methods* are preferred in practice, although they lack completeness and optimality. For instance, OMPL, which will be introduced in the next section, is a popular motion planning library consisting of diverse *sampling-based algorithms*.

5 Motion Planning in Practice

Traditional industrial robots are preprogrammed and are not sensitive to changes in their surroundings. As the workspace changes, it is not trivially easy to adjust their manipulation. However, the current tendency is that an increasing number of robots will work in tandem with workers, who are close to them, sharing the same workspace and collaborating to perform a task. A robot working in such a complex workspace must be aware of its surroundings dynamically and handling situation in an intelligent way. In that background, MoveIt — an integrated development platform on ROS, is developed and becomes state of the art framework for robot motion planning.

MoveIt can generate a set of algorithms that allow robots to plan their movements in various ways and ultimately get from one location to another safely. Meanwhile, it can monitor the robot's surroundings in real time and give feedback duly to the system. The robot then makes dynamic adjustments according to these changes. Compared with its predecessor — `arm_navigation`, MoveIt lowered its dependency on ROS and makes code reuse more efficient. It also provides configuration files and configuration interfaces so that beginners can quickly get started with it.

5.1 Basic System Architecture of MoveIt

move_group MoveIt takes `move_group` as the primitive node and it is easy to extend its capabilities. `move_group` can be configured through ROS Param Server and meanwhile obtain robot description information. We don't need to worry about how to configure `move_group`, since the MoveIt Setup Assistant will automatically generate its launch file for us. In general, it is an integration of various components for ROS *actions* and *services*.

User Interface Users can manage *actions* and *services* in three ways: GUI(Rviz Plugin), `move_group_interface`(C++), `moveit_commander`(Python).

Robot Configuration From ROS Param Server, `move_group` can get the robot information including URDF files (Unified Robot Description Format), SRDF files (Semantic Robot Description Format) and other configurations, such as joint limits.

Robot Interface By means of ROS *topics* and *actions*, `move_group` can determine the current state information by listening to publishers on *joint_states* *topic*, monitor the transform information via ROS TF library, and then inform a server on the robot serving this action through FollowJointTrajectoryAction interface. `move_group` also uses the world geometry monitor inside the Planning Scene Monitor to get the surrounding environment data collected by sensors or

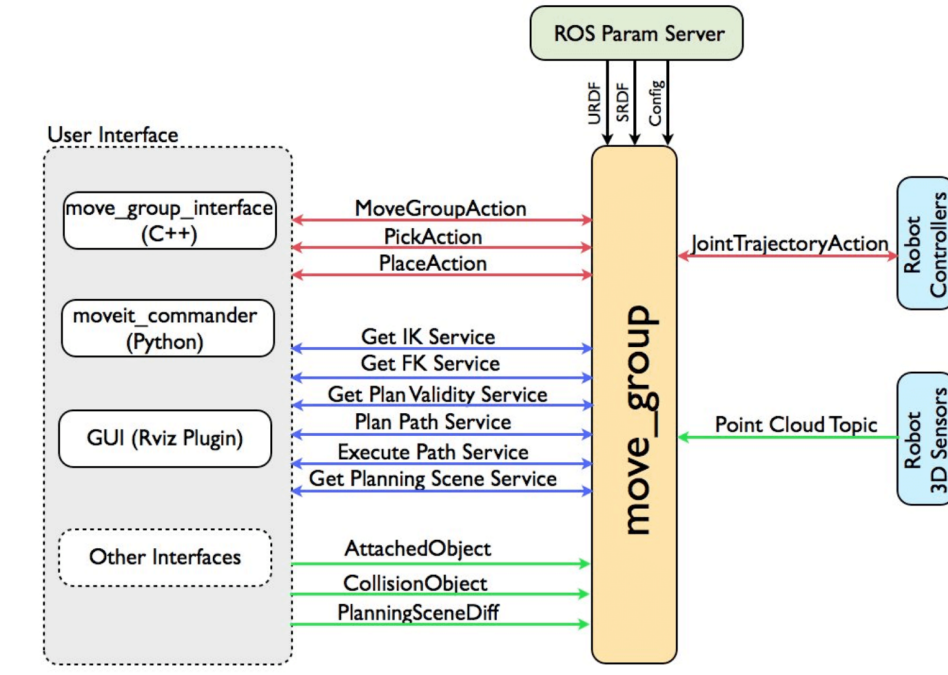


Fig. 13. System Architecture of MoveIt[11]

user input, afterwards the occupancy map monitor can build 3D representation of the robot and the workspace by listening to the *planning_scene topic*. In this dynamic way, a Planning Scene can be maintained.

5.2 Plugin Interfaces in MoveIt

MoveIt provides a lot of interfaces in different fields and make them connectable. It is convenient for user to merge their new ideas without changing the core structure. By using a set of ROS yaml parameters and the ROS pluginlib library, these plugin interfaces can be configured and are ready to use. Here displays different kinds of important plugins and their usages.

Motion planning Through the motion planning plugin, MoveIt is allowed to choose different planners from multiple libraries including OMPL, Pilz industrial motion planner and CHOMP. The default one is OMPL, which implements the basic primitives of sampling-based methods, such as PRM and RRT. The mechanism is that the selected motion planner receives a request including the goal configuration and kinematic constraints, and the request must be pre-processed by a component called planning request adapters. Then the motion planner will generate a trajectory. Similarly, the trajectory it generates should be first post-processed by the planning request adapters, then will be sent back as a response.

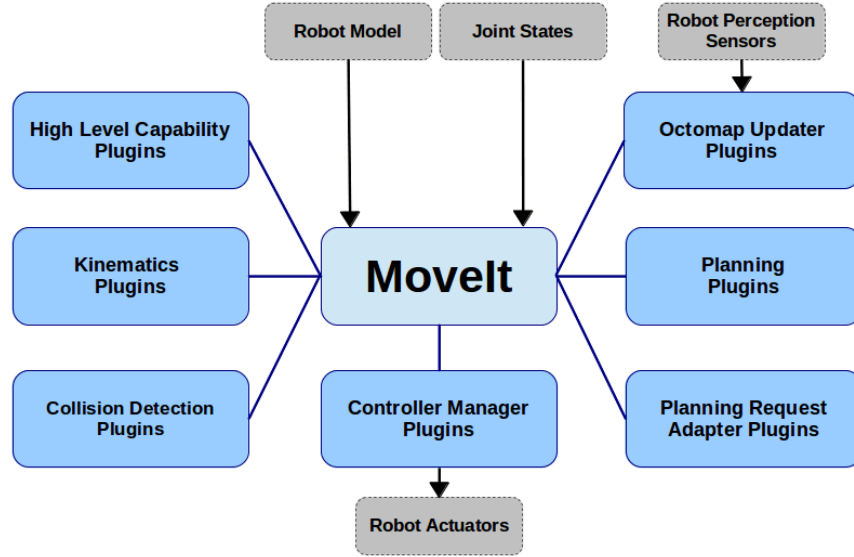


Fig. 14. Plugin interfaces in MoveIt[12]

Kinematics Kinematic algorithms are the core of various algorithms for robotic arms, especially the inverse kinematics algorithms (IK). MoveIt allows users using their own inverse kinematics algorithms and its default inverse kinematics plugin is the KDL numerical Jacobian-based solver.

Collision Checking By using the *CollisionWorld* object inside Planning Scene, collision checking can be conducted within the scope of meshes, primitive shapes and *Octomap*. Collision detection is the most time-consuming operation in motion planning, often taking up 90% of the time. To reduce calculation, it can be optimized by setting an Allowed Collision Matrix(ACM). If the ACM between two bodys is set to 1, it means that these two bodys will never collide and thereby no collision detection is needed.

Trajectory processing The motion planner can conventionally only produces a path with no time information. By using *trajectory processing routine*, MoveIt makes certain optimisation, for instance, a time-parameterised trajectory, so that the robot can satisfy the needs of velocity and acceleration constraints.

6 Conclusion and Perspectives

In conclusion, *Robot motion planning* is a highly disciplinary topic. This seminar work just gives an brief overview of the most fundamental concepts in this

area. It attempted to answer two basic questions: How to formally define a motion planning problem? How to solve a well-defined motion planning problem? An emphasis is placed on the two solving philosophies – *combinatorial motion planning* and *sampling-based motion planning*. *Combinatorial motion planning* is ideal for problems that can be explicitly represented in some way. It can compute an elegant solution by virtue of primitive geometrical characteristics. But when confronting complex environment, the cost of using *combinatorial motion planning* is quite expensive, so we come up with the idea of sampling a comparatively inexact roadmap by using a collision detection module as a black box, namely, *sampling-based motion planning*. The results are predictable: it established its success in solving intractable problems and numerous methods were developed to improve planning quality, which are popular in industry nowadays.

References

1. Abhishek, T.S. et al: Obstacle Avoidance Algorithms: A Review. IOP Conf. Ser.: Mater. Sci. Eng. 1012 012052 (2021). <https://doi.org/10.1088/1757-899x/1012/1/012052>
2. Kairanbay, M., Jani, H.M.: A Review and Evaluations of Shortest Path Algorithms. International Journal of Scientific & Technology Research 2(6):99-104 (2013)
3. Abraham, L.: Target Tracking (2017). <https://doi.org/10.13140/RG.2.2.19898.08643>
4. Ng, William. et al: A review of recent results in multiple target tracking(2005). <https://doi.org/10.1109/ISPA.2005.195381>
5. Ghosh, D. et al: COLLABORATIVE LOCALIZATION USING DYNAMIC NOISE COVARIANCE AND ROBOT MOTION MODEL FOR UNKNOWN AREA EXPLORATION. Conference: CLAWAR (Moscow, 2020). <https://doi.org/10.13180/clawar.2020.24-26.08.id#>
6. Lozano-Pérez, T.: Automatic planning of manipulator transfer movements. IEEE Transactions on Systems, Man & Cybernetics 11(10):681-698 (1981)
7. Northwestern CENTER FOR ROBOTICS AND BIOSYSTEMS Modern Robotics webpage. http://hades.mech.northwestern.edu/index.php/Modern_Robotics. Last access 21 May 2021
8. LaValle, S.M.: Planning Algorithms. Cambridge University Press (2006). <http://lavalle.pl/planning/par2.pdf>
9. Human-Oriented Robotics: Robot Motion Planning. <http://srl.informatik.uni-freiburg.de/teachingdir/ws13/slides/12-RobotMotionPlanning.pdf>. Last access 21 May 2021
10. Configuration Space Visualization of 2-D Robotic Manipulator. <https://www.cs.unc.edu/jeffi/c-space/robot>. Last access 21 May 2021
11. MoveIt Concepts. <https://moveit.ros.org/documentation/concepts/>. Last access 21 May 2021
12. MoveIt Plugin Interfaces. <https://moveit.ros.org/documentation/plugins/>. Last access 21 May 2021
13. Reif, J.H.: Complexity of the mover's problem and generalizations. In proceedings IEEE Symposium on Foundations of Computer Science, pages 421-427(1979)