Trajectory Planning

Quang-Cuong Pham
School of Mechanical and Aerospace Engineering
Nanyang Technological University, Singapore

March 30, 2014

Abstract

Trajectory planning consists in finding a time series of successive joint angles that allows moving a robot from a starting configuration towards a goal configuration, in order to achieve a task, such as grabbing an object from a conveyor belt and placing it on a shelf. This trajectory must respect given constraints: for instance, the robot should not collide with the environment, the joint angles, velocities, accelerations or torques should be within specified limits, etc. Next, if several trajectories are possible, one should choose the one that optimizes a certain objective, such as the trajectory execution time or energy consumption. This chapter reviews methods to plan trajectories with constraints and optimization objectives relevant to industrial robot manipulators.

1 Introduction

For a robot to accomplish a given *high-level* task – such as grabbing an object on a conveyor belt and placing it on a shelf – this task must be translated into *low-level* commands understandable by the robot operating system. This whole process, known as "Motion Planning", can be broken down into a number of steps, as illustrated in Figure 1. Note that, for clarity, the different levels and steps were represented in a sequential, linear fashion. In practice however, several levels or steps might sometimes be addressed at the same time: for instance, when planning at the "Trajectory" level, one might have to take into account the robot torque bounds, which appear at the "Actuator commands" level.

Until recently, Motion Planning has mostly been performed by human operators: for a given task, a skilled, experienced operator would manually find the commands (using e.g. a command board and visual feedback) that would allow the robot to achieve the task. However, this approach is time-consuming and usually produces sub-optimal trajectories. Further, if the task changes (even slightly), the whole tedious teaching process has to be done over again.

Impressive theoretical advances in the field of Motion Planning in the past few decades has brought about a new picture: it is now possible for a computer to find, in minutes or seconds, optimal commands for a robot to achieve a given task, even in very challenging, cluttered environments. Several companies, some of them spinning off from academia (e.g. Siemens Kineo, France/Germany or Mujin Inc., Japan), are developing software solutions that are in use in actual factories.

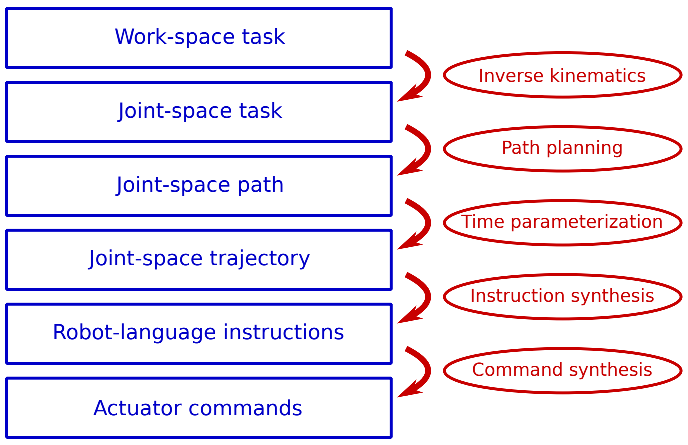

Figure 1: From task to commands. The work-space task may consist in e.g. bringing the end-effector of the robot manipulator from a starting position and orientation in space, where it has just grabbed the object, towards a goal position and orientation, where it can safely release the object onto the shelf. *Inverse kinematics* finds the manipulator joint angles that allow achieving the desired end-effector position and orientation. The joint-space task would then consist in connecting the starting joint angles and the goal joint angles. *Path planning* finds a collision-free path, that is, a continuous set of intermediate joint angles between the starting and goal joint angles, such that the manipulator does not collide with the environment or with itself at any of the intermediate joint angles. *Time parameterization* consists in finding the *time stamps* for the joint angles along the path, while respecting e.g. the torque bounds and/or optimizing the traversal time or the energy consumption. Finally, *instruction synthesis* translates the desired trajectory into a set of instructions in the robot language and *command synthesis* converts these instructions into low-level commands, such as the electrical current fed into the motors. Usually, this last step is performed internally by the robot controller provided by the robot manufacturer and is not accessible to the end-user .

The current chapter discusses *Trajectory Planning*, which is a part of Motion Planning. Specifically, we shall focus on the problem of finding a *trajectory* of the joint angles between given starting and goal joint angles, see Figure 2. It is thus assumed that appropriate computations have been carried out before (e.g. grab synthesis, inverse kinematics, etc.) or will be carried out after (e.g. robot instruction synthesis, command synthesis, etc.) this stage.

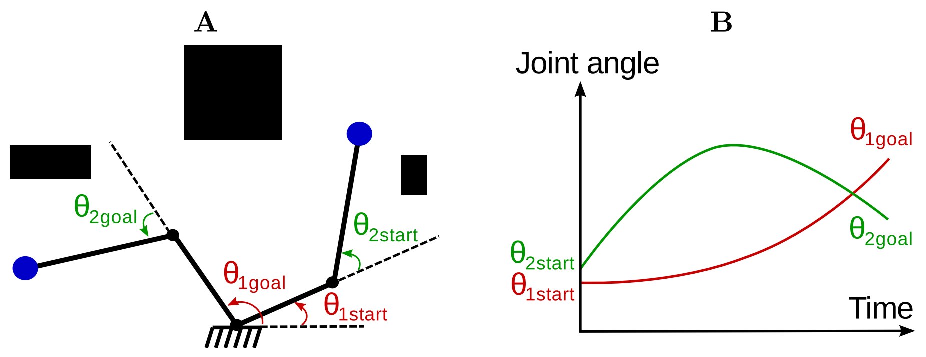

Figure 2: Trajectory planning for a two-link manipulator. **A**: The manipulator and the obstacles (black rectangles) in the work space. **B**: Trajectories connecting $\mathbf{q}_{\mathrm{start}}$ and $\mathbf{q}_{\mathrm{goal}}$ in the joint space.

Two important concepts, namely *constraints* and *optimization*, are essential in trajectory planning. *Constraints* restrict the range of motions that the robot can execute. One can classify them into two categories. First, *geometric constraints* are the constraints that can be expressed solely in terms of the robot joint angles: these include bounds on the joint angles, avoidance of self-collision and of collision with obstacles, etc. These constraints can thus be fully taken into account in the path planning step. Second, *kinodynamic constraints* are the constraints that include higher-order time derivatives of the joint angles, such as bounds on the joint velocities, accelerations, torques, or motor current inputs. These constraints cannot be taken into account by path planning alone and must be considered at the trajectory level.

Optimization comes into play when there are more than one path or trajectory that allow achieving the task while satisfying the constraints. It is then interesting to select the path or trajectory that *optimizes* a given objective. At the path level, one might be interested in finding the shortest path in joint space or in the end-effector space. At the trajectory level, other optimization objectives, such as minimum time, maximum smoothness, minimum energy, etc. can be considered. Note that, because of kinodynamic constraints, a shortest path might not correspond to a minimum-time or minimum-energy trajectory.

Scope of this chapter From the above discussion, it is clear that there exists a very large variety of trajectory planning problems [1, 2], in terms of robot structures, constraints, optimization objectives, etc. The present chapter focuses on problems arising from the industry: we shall specifically consider serial robotic manipulators, constraints arising from collision avoidance and from bounds on the joint angles, velocities, accelerations and torques, and optimization objectives related to the trajectory execution time. Also, we shall concentrate more particularly on the methods that are actually in use in the industry rather than cover the whole panorama of existing planning methods.

Organization The remainder of this chapter is organized as follows. In section 2, some methods for *geometric path planning* and *kinodynamic trajectory planning* are presented. The focus of that section is on finding *one* feasible path or trajectory – optimization is not considered. In section 3, methods for *path and trajectory optimization* under geometric and kinodynamic constraints are reviewed. As stated above, we shall mostly consider the minimization of trajectory execution time. Finally, section 4 offers a brief conclusive discussion.

2 Path and trajectory planning

2.1 Important concepts

We first introduce the important concept of *configuration space* [3] in the context of serial robotic manipulators.

Definition 1 (Configuration space). *A set of joint angles* $\mathbf{q} = (\theta_1, \ldots, \theta_n)$*, where* n *is the number of joints of the robot manipulator, is called a* configuration. *The set of all possible joint angles is called the* configuration space *and denoted* $\mathscr{C}$*. For instance,* $\mathscr{C} = [-\pi, \pi]^n$ *for a manipulator with* n *revolute joints. If a robot is in collision with the environment or with itself at configuration* $\mathbf{q}$ *then* $\mathbf{q}$ *is called a* colliding *configuration. The subset of* $\mathscr{C}$ *consisting of all non-colliding configurations is called the* free space *and denoted* $\mathscr{C}_{\text{free}}$*.*

It is assumed that the starting configuration $\mathbf{q}_{\text{start}}$ and the goal configuration $\mathbf{q}_{\text{goal}}$ are given. The problem thus consists in finding a path or a trajectory (see definition below) connecting $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$ and respecting the constraints.

Definition 2 (Path and trajectory). *Formally, a path can be defined as a continuous function* P

$$\begin{aligned} P: \quad [0,1] \quad &\to \mathscr{C} \\ s \quad &\mapsto \mathbf{q}(s). \end{aligned}$$

For a path connecting $\mathbf{q}_{\text{start}}$ *and* $\mathbf{q}_{\text{goal}}$*, one thus has*

$$\mathbf{q}(0) = \mathbf{q}_{\text{start}}, \quad \mathbf{q}(1) = \mathbf{q}_{\text{goal}}.$$

A path can be regarded as a geometric *object devoid of any timing information. Next, a* time parameterization *of* P *is a strictly increasing function*

$$\begin{aligned} s: \quad [0,T] \quad &\to [0,1] \\ t \quad &\mapsto s(t). \end{aligned}$$

with $s(0) = 0$ *and* $s(T) = 1$*. The function* s *gives the position on the path for each time instant* t*.*

The path P *endowed with a time parameterization* s *becomes a* trajectory Π

$$\begin{aligned} \Pi: \quad [0,T] \quad &\to \mathscr{C} \\ t \quad &\mapsto \mathbf{q}(s(t)), \end{aligned}$$

with T *being the* duration *of the trajectory. Note that the same* path P *can give rise to many different* trajectories Π.

2.2 Path planning under geometric constraints

There exists a large variety of approaches to path planning: combinatorial methods, potential field methods, sampling-based methods, etc.

Combinatorial methods make use of algebraic geometry and mathematical programming tools to provide *complete* algorithms to the path planning problems. "Complete" means that the algorithm can decide in finite time whether the problem has a solution, and provide a solution when it exists. An example of a combinatorial method is the cell decomposition method: when the obstacles are polygonal, the free space $\mathscr{C}_{\mathrm{free}}$ can be decomposed into cells (e.g. triangular cells by a Delaunay triangulation). Solving the problem then boils down to finding a sequence of adjacent cells such that $\mathbf{q}_{\mathrm{start}}$ and $\mathbf{q}_{\mathrm{goal}}$ belongs to respectively the first and the last cell of the sequence (see e.g. [2], section 6.3.2). While combinatorial methods are valuable to obtain theoretical results (for instance, they allow proving that the general motion planning problem is NP-complete), they are too slow to be used in practice, especially in high-dimensional problems. Another issue is that these methods require an *explicit* representation of the obstacles, which is very difficult to obtain in most practical problems.

The *potential-field method* was introduced in [4]. This method constructs a potential field which is high near the obstacles and low at the goal configuration. Steering towards the goal configuration while avoiding the obstacles can then be naturally achieved by letting the robot configuration evolve in that potential field. This method is interesting in that it allows real-time control. However, the issue of local minima – the robot may get trapped in the local minima of the potential field – prevents its use in highly cluttered environments.

Sampling-based methods are perhaps the most efficient and robust, hence probably the most widely used methods for path planning currently. The next two sections review these methods in detail.

2.2.1 Grid Search and Probabilistic Roadmap (PRM)

A first category of sampling-based methods requires building, in a *preprocessing* stage, a *roadmap* that captures the connectivity of $\mathscr{C}_{\mathrm{free}}$. A roadmap is a graph G whose vertices are configurations of $\mathscr{C}_{\mathrm{free}}$. Two vertices are connected in G *only if* it is possible to connect the two configurations by a path entirely contained in $\mathscr{C}_{\mathrm{free}}$. Once such a roadmap is built, it is easy to answer the path planning problem: if (i) $\mathbf{q}_{\mathrm{start}}$ can be connected to a (nearby) vertex u and (ii) $\mathbf{q}_{\mathrm{goal}}$ can be connected to a (nearby) vertex v and (iii) v and v are in the same *connected component* of G (in the graph-theoretic sense), then there exists a collision-free path between $\mathbf{q}_{\mathrm{start}}$ and $\mathbf{q}_{\mathrm{goal}}$ (see Figure 3).

Methods for building the roadmap fall into two families: deterministic or probabilistic. A typical deterministic method is the *Grid Search*, where the configuration space $\mathscr{C}$ is sampled following a regular grid, as in Figure 3A. A sampled configuration is rejected if it is not in $\mathscr{C}_{\mathrm{free}}$. Next, one attempts to connect every pair of adjacent configurations (adjacent in the sense of the grid structure) to each other: if the straight segment connecting the two configurations is contained within $\mathscr{C}_{\mathrm{free}}$, then the corresponding edge is added to the graph G.

In the *Probabilistic Roadmap* method [5], instead of following a regular grid, samples are taken at random in $\mathscr{C}_{\mathrm{free}}$, see Figure 3B. Since there is no a priori grid structure, several methods exist for choosing the pairs of vertices for which connection is attempted:

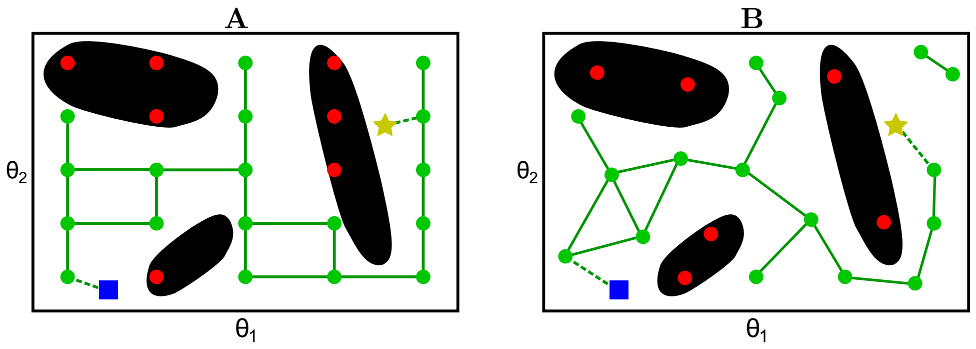

Figure 3: Roadmap-based methods. **A**: Grid Search. The starting and goal configurations are represented by respectively the blue square and the yellow star. Green disks represent sampled configurations lying in $\mathscr{C}_{\text{free}}$ while red disks represent colliding configurations. **B**: Probabilistic Roadmap. Same legends as in **A**.

for instance, one may attempt, for each vertex, to connect it to every vertices that are within a specified radius r from it.

Advantages of the roadmap-based methods The strength of the roadmap-based methods (both deterministic and probabilistic) comes from the global/local decomposition: the difficult problem of path planning is solved at two scales: the *local scale* where neighboring configurations (adjacent configurations in Grid Search, configurations within a small distance r of each other in the Probabilistic Roadmap) are connected by a simple straight segment; and the *global scale*, where the graph search takes care of the global, complex connectivity of the free space.

Note also that it is not necessary for these methods to have an explicit representation of $\mathscr{C}_{\text{free}}$: one only needs an *oracle* which says whether a given configuration is in $\mathscr{C}_{\text{free}}$. To check whether a straight segment is contained within $\mathscr{C}_{\text{free}}$, it suffices to call the oracle on every configurations (or, in practice, on sufficiently densely sampled configurations) along that segment.

These methods also possess nice theoretical guarantees. Specifically, it can be shown that the Grid Search is *resolution complete*, which means that, if a solution exists, the algorithm will find it in finite time and for a sufficiently small grid size. Similarly, it can be shown that the Probabilistic Roadmap method is *probabilistically complete*, which means that, if a solution exists, the probability that the algorithm will find it converges to 1 as the number of sample points goes to infinity [6]. However, the *converge rate* of both methods is difficult to determine on practical problem instances.

Regarding the comparative performances of the deterministic and probabilistic approaches, it has been shown both theoretically and practically that randomness is not advantageous in terms of search time. However, it can be argued that probabilistic methods are easier to implement [6].

2.2.2 Rapidly-Exploring Random Trees (RRT)

The methods just discussed require building the entire roadmap in the preprocessing stage before being able to answer any query [a query being a pair $(\mathbf{q}_{\text{start}}\mathbf{q}_{\text{goal}})$ to be connected]. In applications where only a single or a limited number of queries are needed, it may not be worthy to build the whole roadmap. *Single query* methods, such as the Rapidly-exploring Random Trees (RRT), are much more suited for these applications [7].

Specifically, RRT iteratively builds a *tree* (see Algorithms 1), which is also a roadmap, but which has the property of exploring “optimally” the free space. The key lies in the EXTEND function, which selects the vertex in the tree that is the closest to the randomly sampled configuration (see Algorithm 2 and Figure 4). Thus, the probability for a vertex in the tree to be selected is proportional to the size of its Voronoi region, causing the tree to grow preferably towards previously under-explored regions [7].

Algorithm 1 BUILD_RRT

```
Input: A starting configuration $\mathbf{q}_{\text{start}}$
Output: A tree $\mathcal{T}$ rooted at $\mathbf{q}_{\text{start}}$
  $\mathcal{T}$.INITIALIZE($\mathbf{q}_{\text{start}}$)
  for rep = 1 to $N_{\text{maxrep}}$ do
    $\mathbf{q}_{\text{rand}} \leftarrow$ RANDOM_CONFIG()
    EXTEND($\mathcal{T}, \mathbf{q}_{\text{rand}}$)
  end for
```

Algorithm 2 EXTEND

```
Input: A tree $\mathcal{T}$ and a target configuration $\mathbf{q}$
Effect: Grow $\mathcal{T}$ by a new vertex in the direction of $\mathbf{q}$
  $\mathbf{q}_{\text{near}} \leftarrow$ NEAREST_NEIGHBOR($\mathcal{T}, \mathbf{q}$)
  if $\mathbf{q}_{\text{new}} \leftarrow$ STEER($\mathbf{q}_{\text{near}}, \mathbf{q}$) succeeds then
    Add vertex $\mathbf{q}_{\text{new}}$ to $\mathcal{T}$
    Add edge $[\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{new}}]$ to $\mathcal{T}$
  end if
```

Note: STEER($\mathbf{q}_{\text{near}}, \mathbf{q}$) attempts making a straight motion from $\mathbf{q}_{\text{near}}$ towards $\mathbf{q}$. Three cases can happen: (i) $\mathbf{q}$ is within a given distance r of $\mathbf{q}_{\text{near}}$ and $[\mathbf{q}_{\text{rand}}, \mathbf{q}]$ is collision-free, then $\mathbf{q}$ is returned as the new vertex $\mathbf{q}_{\text{new}}$; (ii) $\mathbf{q}$ is farther than a given distance r of $\mathbf{q}_{\text{near}}$, and the segment of length r from $\mathbf{q}_{\text{near}}$ and in the direction of $\mathbf{q}$ is collision-free, then the end of that segment is returned as $\mathbf{q}_{\text{new}}$ (see Figure 4); (iii) else, STEER returns `Failure`.

Finally, to find a path connecting $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$, one can grow simultaneously two RRTs, one rooted at $\mathbf{q}_{\text{start}}$ and the other rooted at $\mathbf{q}_{\text{goal}}$, and attempt to connect the two trees at each iteration. This algorithm is known as Bidirectional RRT [8]. In practice, Bidirectional RRT has proved be easy to implement, yet extremely efficient and robust: it has been successfully applied to a large variety of robots and challenging environments.

2.3 Trajectory planning under kinodynamic constraints

As mentioned earlier, kinodynamic constraints involve higher-order derivatives of the configuration and cannot therefore be expressed in the configuration space. One approach to kinodynamic planning thus consists of *transposing* the path planning methods (such as PRM or RRT) to the *state space* $\mathscr{X}$, that is, the configuration space $\mathscr{C}$ augmented with velocity coordinates, where the kinodynamic constraints can be appropriately taken into account [9, 10, 11]. One drawback of this approach is that moving

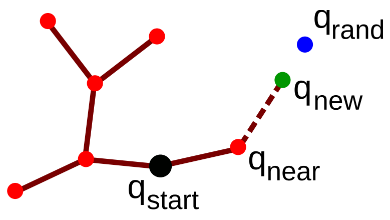

Figure 4: Illustration for the EXTEND function. The tree is rooted at the black disk. Red disks and plain segments represent respectively the vertices and edges that have already been added to the tree. EXTEND attempts at growing the tree toward a random configuration $\mathbf{q}_{\mathrm{rand}}$. For this, $\mathbf{q}_{\mathrm{near}}$ is chosen as the vertex in the tree that is the closest to $\mathbf{q}_{\mathrm{rand}}$. The tree is then grown from $\mathbf{q}_{\mathrm{near}}$ towards $\mathbf{q}_{\mathrm{rand}}$, stopping at $\mathbf{q}_{\mathrm{new}}$, which is at the specified radius r from $\mathbf{q}_{\mathrm{near}}$.

into the state space is associated with a two-fold increase in the dimension of the search space: if $\mathscr{C}$ is of dimension n, then $\mathscr{X}$ is of dimension $2n$. Since planning algorithms usually have complexities that scale *exponentially* with the dimension of the search space [11], a two-fold increase in the dimension may make state-space kinodynamic planning algorithms impractical even for relatively small values of n [12].

A second approach avoids the complexity explosion by *decoupling* the problem: first, search for a *path* in the robot configuration space $\mathscr{C}$ under geometric constraints (using path planning methods discussed earlier) and, in a second step, find a *time-parameterization* of that path that satisfies the kinodynamic constraints [13, 14]. The drawback here is that the path found in the first step may have no time-parameterization at all that respects the kinodynamic constraints. This drawback may be addressed by introducing an Admissible Velocity Propagation scheme, which allows checking, at each PRM/RRT extension, the existence of an eventual admissible time-parameterization [15].

3 Path and trajectory optimization

As stated in the Introduction, we shall focus here on minimizing trajectory execution time, which is the most common optimization objective in the industry. When planning at the *path level*, minimizing execution time can be equated to minimizing the path length, which is covered in Section 3.1. At the *trajectory level* however, shortest paths may not correspond to minimum-time trajectories when kinodynamic constraints come into play. Section 3.2 reviews methods to specifically minimize trajectory duration.

3.1 Minimizing path length

3.1.1 Asymptotically optimal methods

While the roadmap-based methods presented in Section 2.2.1 address the *feasibility* problem, i.e. the problem of finding *one* feasible path, it is straightforward to modify them to include path length *optimization*. Indeed, once $\mathbf{q}_{\mathrm{start}}$ and $\mathbf{q}_{\mathrm{goal}}$ have been connected respectively to vertices u and v in G, classical graph algorithms, such as Dijkstra's search [16] or A^* search [17], can be applied to find the shortest path between u and v in G. In turn, the path lengths between $\mathbf{q}_{\mathrm{start}}$ and $\mathbf{q}_{\mathrm{goal}}$ is minimized.

It can be shown that these algorithms are *asymptotically optimal* in the sense that the path length of the solution returned by these algorithms converges to the minimal path length as the grid size – in the case of Grid Search – goes to 0, or as the number of sampled points goes to infinity – in the case of the Probabilistic Roadmap.

Regarding the single-query problem, it has been shown that the RRT method, if modified in the same way as above, would not yield an asymptotically optimal algorithm. However, RRT* – a modified version of RRT where the EXTEND function tries to extend from not only the nearest vertex but from a specific number of nearest vertices – possesses the asymptotic optimality property [18].

3.1.2 Path shortcutting

While the RRT* algorithm just mentioned has the nice property of being asymptotically optimal, it is two slow to be used in practice. For single-query problems, it turns out that a two-step approach consisting of (i) finding one path using RRT, followed by (ii) post-processing this trajectory by repetitively applying shortcuts – realizes a good trade-off between path quality and computation time [19].

The shortcut method is presented in Algorithm 3. It is simple to implement, yet very effective. A modified version, called *partial shortcut*, consists in shortcutting one joint angle at a time, can yield even higher quality paths but also requires a longer computation time [19].

Algorithm 3 PATH_SHORTCUT

```
Input: A collision-free path $P$
Output: A shorter collision-free path
  for rep = 1 to $N_{\mathrm{maxrep}}$ do
    Pick two random points $\mathbf{q}_1, \mathbf{q}_2$ along the path
    if $[\mathbf{q}_1, \mathbf{q}_2]$ is collision-free then
      Replace the portion of $P$ between $\mathbf{q}_1$ and $\mathbf{q}_2$ by $[\mathbf{q}_1, \mathbf{q}_2]$
    end if
  end for
Note: $[\mathbf{q}_1, \mathbf{q}_2]$ denotes the straight segment between $\mathbf{q}_1$ and $\mathbf{q}_2$ in the joint space.
```

3.2 Minimizing trajectory duration

3.2.1 Fixed-path time minimization

Once a collision-free path has been found, one can give sample configurations along the path as input to the robot. However, for most modern robot manipulators, execution time can be greatly reduced if each sample configuration is accompanied with a *time stamp*, i.e. the time instant when the robot should reach that configuration. This requires *time-parameterizing* the path, that is, transforming it into a *trajectory*.

More specifically, with the notations introduced in Definition 2 of Section 2.1, minimizing the traversal time of a given path P is to find the time parameterization s such that T is minimal and that the parameterized *trajectory* $(\mathbf{q}(s(t)))_{t\in[0,T]}$ satisfies given kinodynamic constraints.

When the constraints are bounds on the joint torques, a very efficient solution to this problem, based on Pontryagin's Maximum Principle, was proposed in the 80's [13, 20], and has been continuously improved until today [21, 22, 23, 24]. This method can also be applied to other types of kinodynamic constraints such as gripper and payload constraints [25] or bounds on the joint velocities and accelerations [26]. More recently, another family of algorithms to solve this fixed-path time minimization, based on convex optimization, was proposed in [27, 28].

3.2.2 Global time minimization

A number of exact [29, 30] and approximate [31] methods exist to *directly* find the time-optimal *trajectory* subject to torque bounds between two configurations. However, these methods are only practical for low-dimensional problems and cannot deal with geometric obstacles.

To take into account both geometric and kinodynamic (e.g. torque bounds) constraints, an effective approach consists of generating a large number of paths, and on each path, apply the fixed-path time minimization described in the previous section. In [32], the author considers a family of paths consisting of Bezier curves. A path of this family can be represented by a set of control points $\mathbf{p} = (p_1, \dots, p_n)$. One can then define the cost $C(\mathbf{p})$ by the duration of the time-minimal parameterization of the Bezier curve represented by $\mathbf{p}$. Finally, one can search for the time-minimal trajectory by a gradient search, where the gradient $\mathrm{d}C/\mathrm{d}\mathbf{p}$ is evaluated numerically.

Another method [33] consists in building roadmaps as in Section 3.1, but where the cost of an edge in the graph search would not be the distance between the adjacent vertices but a heuristic quantity related to the fixed-path time minimization algorithm of section 3.2.1.

3.2.3 Shortcutting with kinodynamic constraints

As in the case of path planning (Section 3.1.2), it turns out that shortcutting is the most effective method to obtain trajectories with short durations [34]. There is however an important difference between trajectory and path shortcutting: in trajectory shortcutting, one usually needs to ensure that the new portion can be inserted into the original trajectory while preserving the *smoothness properties* of the original trajectory. For instance, if one wants to preserve the C^1-continuity of the trajectory (i.e. the property that the trajectory is differentiable and that the derivative is continuous), then it is necessary to generate shortcuts that begin and end, not only at the same configurations $\mathbf{q}_1$, $\mathbf{q}_2$, but also *with the same velocities* $\dot{\mathbf{q}}_1$, $\dot{\mathbf{q}}_2$ as in the original portion.

Algorithm 4 presents shortcutting under velocity and acceleration (or pure kinematic) bounds [34]. This algorithm is very effective thanks to the following property: given the beginning and ending configurations and velocities $(\mathbf{q}_1, \dot{\mathbf{q}}_1)$, $(\mathbf{q}_2, \dot{\mathbf{q}}_2)$, it is possible to compute *analytically* the time-optimal trajectory portion $\Pi^{\mathrm{kin}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$ under given velocity and acceleration bounds [34].

Algorithm 5 presents shortcutting under general kinodynamic bounds [35]. Contrary to the case of velocity and acceleration bounds, there is no analytic expression of the time-optimal trajectory for general kinodynamic constraints. One thus have to resort to the path decoupling approach presented earlier: (i) interpolate a path between $(\mathbf{q}_1, \dot{\mathbf{q}}_1)$ and $(\mathbf{q}_2, \dot{\mathbf{q}}_2)$ respecting C^1-continuity, and (ii) time-parameterize the path optimally under the given kinodynamic constraints. Note that the heuristic to chose the path in step (i) is crucial for the performance of the algorithm.

Algorithm 4 TRAJ_SHORTCUT_KINEMATIC

Input: A collision-free, C^1 trajectory Π satisfying velocity and acceleration constraints
Output: A collision-free, C^1 trajectory satisfying the velocity and acceleration constraints and with shorter time duration

```
for rep = 1 to $N_{\mathrm{maxrep}}$ do
    Pick two random points $\mathbf{q}_1, \mathbf{q}_2$ along $\Pi$
    if $\Pi^{\mathrm{kin}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$ is collision-free then
        Replace the portion of $\Pi$ between $\mathbf{q}_1$ and $\mathbf{q}_2$ by $\Pi^{\mathrm{kin}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$
    end if
end for
```

Note: $\Pi^{\mathrm{kin}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$ denotes the optimal trajectory between $(\mathbf{q}_1, \dot{\mathbf{q}}_1)$ and $(\mathbf{q}_2, \dot{\mathbf{q}}_2)$ under given velocity and acceleration bounds (see text).

Algorithm 5 TRAJ_SHORTCUT_GENERAL

Input: A collision-free, C^1 trajectory Π satisfying general kinodynamic constraints
Output: A collision-free, C^1 trajectory satisfying the kinodynamic constraints and with shorter time duration

```
for rep = 1 to $N_{\mathrm{maxrep}}$ do
    Pick two random points $\mathbf{q}_1, \mathbf{q}_2$ along $\Pi$
    Generate a path $P^{\mathrm{int}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$
    if $P^{\mathrm{int}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$ is collision-free then
        Time-parameterize $P^{\mathrm{int}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$ into $\Pi^{\mathrm{int}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$
        if $\Pi^{\mathrm{int}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$ has shorter time duration than the original portion then
            Replace the portion of $\Pi$ between $\mathbf{q}_1$ and $\mathbf{q}_2$ by $\Pi^{\mathrm{int}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$
        end if
    end if
end for
```

Note: $P^{\mathrm{int}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$ denotes an interpolated path between $(\mathbf{q}_1, \dot{\mathbf{q}}_1)$ and $(\mathbf{q}_2, \dot{\mathbf{q}}_2)$. $\Pi^{\mathrm{int}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$ denotes the time-optimal trajectory obtained by time-parameterizing $P^{\mathrm{int}}(\mathbf{q}_1, \dot{\mathbf{q}}_1, \mathbf{q}_2, \dot{\mathbf{q}}_2)$ under given kinodynamic constraints (see text).

4 Conclusion

We have presented an overview of trajectory planning and optimization methods, with a special emphasis on those relevant to industrial robotic manipulators. It appears from this overview that very efficient methods exist for planning high quality trajectories when the environment (consisting of the robot, the obstacles, etc.) is well-defined and static. A typical work-flow may integrate some of these methods as sketched in Figure 5. The main current challenge of trajectory planning in classical factory automation lies mainly in the development of robust software, as well as practical integration into the work place.

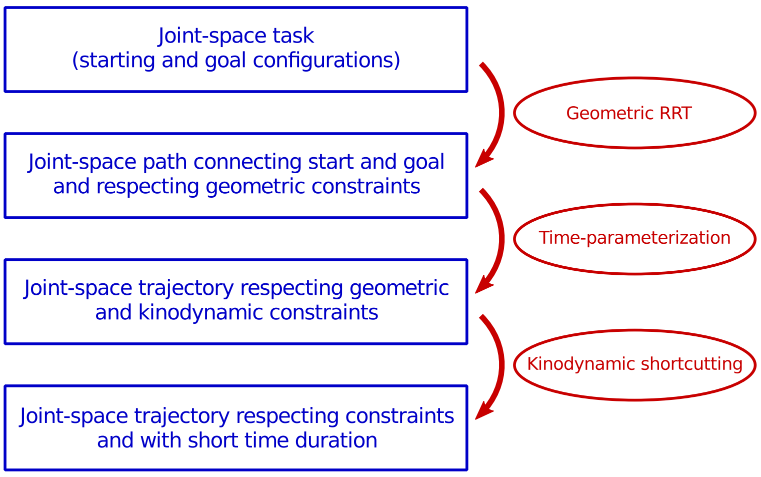
Joint-space task
(starting and goal configurations)
Geometric RRT
Joint-space path connecting start and goal and respecting geometric constraints
Time-parameterization
Joint-space trajectory respecting geometric and kinodynamic constraints
Kinodynamic shortcutting
Joint-space trajectory respecting constraints and with short time duration

Figure 5: Typical work-flow as practiced in a company specialized in motion planning for industrial robots.

The next major step in factory automation is to integrate the robot more tightly with human operators. For this, new methods must be developed, taking into account environments that are by nature time-changing, and sometimes in an unpredictable way, because of the close, possibly physical interaction with human operators. In this context, other types of constraints and optimization objectives must also be considered, such as safety or compliance.

References

[1] Y. K. Hwang and N. Ahuja, “Gross motion planning – a survey,” *ACM Computing Surveys (CSUR)*, vol. 24, no. 3, pp. 219–291, 1992.

[2] S. LaValle, *Planning algorithms.* Cambridge Univ Press, 2006.

[3] T. Lozano-Perez, “Spatial planning: A configuration space approach,” *Computers, IEEE Transactions on*, vol. 100, no. 2, pp. 108–120, 1983.

[4] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.

[5] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[6] S. M. LaValle, M. S. Branicky, and S. R. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 673–692, 2004.

[7] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," *Technical Report 98-11, Iowa State University*, 1998.

[8] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, 2000.

[9] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.

[10] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[11] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[12] Z. Shiller and S. Dubowsky, "On computing the global time-optimal motions of robotic manipulators in the presence of obstacles," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 6, pp. 785–797, 1991.

[13] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985.

[14] J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue, "Dynamically-stable motion planning for humanoid robots," *Autonomous Robots*, vol. 12, no. 1, pp. 105–118, 2002.

[15] Q.-C. Pham, S. Caron, and Y. Nakamura, "Kinodynamic planning in the configuration space via velocity interval propagation," in *Robotics: Science and System*, 2013.

[16] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[17] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[18] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[19] R. Geraerts and M. Overmars, "Creating high-quality paths for motion planning," *The International Journal of Robotics Research*, vol. 26, no. 8, pp. 845–863, 2007.

[20] K. Shin and N. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *IEEE Transactions on Automatic Control*, vol. 30, no. 6, pp. 531–541, 1985.

[21] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *IEEE Journal of Robotics and Automation*, vol. 3, no. 2, pp. 115–123, 1987.

[22] J. Slotine and H. Yang, "Improving the efficiency of time-optimal path-following algorithms," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 118–124, 1989.

[23] Z. Shiller and H. Lu, "Computation of path constrained time optimal motions with dynamic singularities," *Journal of dynamic systems, measurement, and control*, vol. 114, p. 34, 1992.

[24] Q.-C. Pham, "Characterizing and addressing dynamic singularities in the time-optimal path parameterization algorithm," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[25] Z. Shiller and S. Dubowsky, "Robot path planning with obstacles, actuator, gripper, and payload constraints," *The International Journal of Robotics Research*, vol. 8, no. 6, pp. 3–18, 1989.

[26] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," in *Robotics: Science and Systems*, vol. 8, 2012, pp. 09–13.

[27] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, 2009.

[28] K. Hauser, "Fast interpolation and time-optimization on implicit contact submanifolds," in *Robotics: Science and Systems*, 2013.

[29] H. P. Geering, L. Guzzella, S. A. Hepner, and C. H. Onder, "Time-optimal motions of robots in assembly tasks," in *Decision and Control, 1985 24th IEEE Conference on*, vol. 24. IEEE, 1985, pp. 982–989.

[30] E.-B. Meier and A. E. RYSON, "Efficient algorithm for time-optimal control of a two-link manipulator," *Journal of Guidance, Control, and Dynamics*, vol. 13, no. 5, pp. 859–866, 1990.

[31] H. Yang and J. Slotine, "Fast algorithms for near-minimum-time control of robot manipulators," *The International Journal of Robotics Research*, vol. 13, no. 6, pp. 521–532, 1994.

[32] J. Bobrow, "Optimal robot plant planning using the minimum-time criterion," *IEEE Journal of Robotics and Automation*, vol. 4, no. 4, pp. 443–450, 1988.

[33] Z. Shiller and Y. Gwo, "Dynamic motion planning of autonomous vehicles," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, pp. 241–249, 1991.

[34] K. Hauser and V. Ng-Thow-Hing, "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 2493–2498.

[35] Q.-C. Pham, "Planning manipulator trajectories under dynamics constraints using minimum-time shortcuts," in *Second IFToMM ASIAN Conference on Mechanism and Machine Science*, 2012.