A mathematical motivation for industrial motion planning

Jose Capco
Email: jcapco@yahoo.com

23.05.2014

Abstract

This paper is still a Draft!. This paper is to give a mathematical motivation in the study of robotics specifically in robotic motion planning. We develop the necessary language and terminology used for this research which we refer to as *robotic theory*. We discuss collision, forward kinematics, kinematic singularity and path planning and give both rigorous definition as well as example algorithms. We define cost function and give various example for cost function of robotic paths. Finally we provide examples (applicable in the industry) that would need solution to fundamental path planning problem: We specifically define and introduce bin-picking and coverage planning in a mathematical context.

1 Introduction

For brevity we will always assume that every joint of a robot has only one degree of freedom. Throughout we shall always use the following:

Let $SE(3)$ be the special Euclidean group (group of rigid body transformations) in dimension 3. For any $t \in SE(3)$ let $\text{trans}(t) \in \mathbb{R}^3$ be the translational part of t written canonically as an element in $\mathbb{R}^3$.

Let $n \in \mathbb{N}$ be a fixed integer with $n \geq 6$. We call n the *number of joints (of an articulated robot)*. Fix C to be a closed subspace of $\mathbb{R}^n$ defined by

$$C := \prod_{i=1}^{n} I_i \subset \mathbb{R}^n \qquad I_i = [a_i, b_i] \subset \mathbb{R}, a_i, b_i \in \mathbb{R} \text{ with } a_i \leq 0, b_i \geq 0 \quad \forall i = 1, \ldots, n$$

We call C the *reachable configuration space* and elements of C are called *(reachable) configurations*. Let π_j be the projection from $\mathbb{R}^n$ to $\mathbb{R}^j$ and ρ_j be the projection from $\mathbb{R}^n$ to the j-th coordinate for any $1 \leq j \leq n$. So we have $\pi_j(C) = \prod_{i=1}^{j} I_i$ and $\rho_j(C) = I_j$. Unless otherwise stated, we fix an abstract *robot* $\mathcal{R}$ which consists of

- n number of morphisms in the category of smooth manifolds $f_i : \mathbb{R}^i \to SE(3)$ for $i = 1, \ldots, n$. These functions should have the property that for any $1 \leq i < n$ and $c \in C$ one has

  $$f_{i+1}(\pi_{i+1}(c)) = f_i(\pi_i(c))g_{i+1}(\rho_{i+1}(c))$$

  where $g_{i+1} : \mathbb{R} \to SE(3)$ is a smooth morphism, we call this the *relative (forward kinematic) function of the i-th link*. We call f_i the *forward kinematic of the j-th link*. And we call f_n simply the *forward kinematic (of the robot)*. In this work, we use the symbol F for f_n.

- n number of compact subsets of $\mathbb{R}^3$

  $$L_i \subset \mathbb{R}^3 \qquad i = 1, \ldots, n$$

  We call L_i the *i-th link (of the robot)* and for $c \in C$ we call

  $$f_i(\pi_i(c))L_i = \{f_i(\pi_i(c))x \,:\, x \in L_i\} \subset \mathbb{R}^3$$

the *i-th link at configuration c.* If no confusion arises, we sometimes write $L_i(c)$ to mean the i-th link at configuration c (so we can also treat L_i roughly as a function from C to a subset of the power set of $\mathbb{R}^3$ whose elements are all sets which are geometrically congruent to the i-th link).

In most of our situation we also have another compact subset of $K \subset \mathbb{R}^3$ which we call the *(static) collision geometry.*

Definition 1. We say that there is a *collision at $c \in C$* iff either one of the following condition holds:

1. $\exists i \in \{1, \ldots, n\}$ such that $L_i(c) \cap K \neq \emptyset$ in this case we also say that the *i-th link collides with the collision geometry at configuration c.*
2. $\exists i, j \in \{1, \ldots, n\}$ with $i \neq j$ and such that $L_i(c)^\circ \cap L_j(c)^\circ \neq \emptyset$ in this case we also say that there is a *self-collsion between links i and j at configuration c.*

If there is a collision at a certain configuration then we say that the *robot collides*, otherwise we say that there is *no collision at c* or that the robot does *not collide.*

Construction 1. Define

$$C_f := \{c \in C \ : \ \text{there is no collision at } c\}$$

to be the *collision-free (reachable) configuration space* and let $E_f := F(C_f) \subset SE(3)$ be called the *collision-free (reachable) cartesian space.* We can restrict the forward kinematic to $F|_{C_f} : C_f \to E_f$ and if this does not cause confusion we shall also use the symbol F for this restriction.

We would like to "measure" paths in C_f in such a way that we can evaluate whether a path is a good or not. There are many interpretation of a good path in robotics, but in most of these interpretation, one agrees that concatenating two paths is often less desirable and more "expensive" than just one of the paths, so we will make a definition of a cost function that respect this rule.

Definition 2. Define $\mathcal{P}$ be the set of paths in C_f and define a binary operator $*$ on $\mathcal{P}$ with the property that

$$p_1 * p_2(t) := \begin{cases} p_1(2t) & t \in [0, \frac{1}{2}] \\ p_2(2t-1) & t \in (\frac{1}{2}, 1] \end{cases}$$

for $p_1(1) = p_2(0)$. Now let $\mathfrak{A}$ be a σ-algebra of a set X, and suppose that $\nu : \mathcal{P} \to \mathfrak{A}$ be a function with the property that $\nu(p_1 * p_2) = \nu(p_1) \cup \nu(p_2)$ if $p_1(1) = p_2(0)$. For an outer measure $\mu : 2^X \to [0, \infty]$ we define $\mu \circ \nu : \mathcal{P} \to [0, \infty]$ to be a *path cost function (induced by ν and μ).*

Example 1. Let $\mathfrak{A}$ be the Borel-algebra of $\mathbb{R}^3$ and let $\nu : \mathcal{P} \to \mathfrak{A}$ be defined by

$$\nu(p) := \{x \in \mathbb{R}^3 \ : \ \exists i = 1, \ldots, n \text{ and } t \in [0,1] \ni x \in L_i(p(t))\}$$

then $\nu(p)$ is also called the *swept volume for a path p.* If we take μ to be the Lebesgue measure, then the value of $\nu(p)$ by μ is simply the computed volume of the swept volume in $\mathbb{R}^3$. The cost function from μ and ν is called the swept volume cost function.

Example 2. Let $\mathcal{P}_E$ be the set of images of paths in $\text{trans}(E_f) \subset \mathbb{R}^3$ and let $\mathfrak{A}$ be the σ-algebra generated by $\mathcal{P}_E$. There is an outer measure induced by the function

$$\mu : \mathcal{P}_E \to [0, \infty]$$

such that if $p : [0,1] \to \text{trans}(E_f)$ is a path in $\text{trans}(E_f)$ then

$$\mu(p[0,1]) := \int_0^1 ||J(p(t))|| dt$$

where J is the Jacobian of p (so μ just takes the arc-length of p). Define $\nu : \mathcal{P} \to \mathfrak{A}$ by

$$\nu(p) := \mathrm{trans}(F(p[0,1]))$$

then we see that this satisfies the condition for the definition for the cost function so that $f_c := \mu \circ \nu$ is a path cost function.

Example 3. Let $\mathfrak{A}$ be the σ-algebra generated by the set of lines in $\mathbb{R}^3$. Let μ be the outer measure on the powerset of $\mathbb{R}^3$ induced from a function that, given a line, evaluates the lines length. Define $\nu : \mathcal{P} \to \mathfrak{A}$ by:

$$\nu(p) := \{(1-t)\mathrm{trans}(F(p(0))) + t\mathrm{trans}(F(p(1))) \, : \, t \in [0,1]\} \subset \mathbb{R}^3 \quad p \in \mathcal{P}$$

so, ν takes a path in the configuration space induces a path in $\mathbb{R}^3$ and takes the line in $\mathbb{R}^3$ with the same endpoint as it.We see that this also satisfies the condition for the definition for the cost function so that $f_c := \mu \circ \nu$ is a path cost function.

Remark. $(\mathcal{P}, *)$ is not a semigroup because there is no associativity but one can force associativity in different ways which is beyond the scope of this paper. The swept volume for a path is actually a Lebesgue measurable set.

Definition 3. Let E_f° be the interior of E_f in $SE(3)$ and C_f° be the interior of C_f in $\mathbb{R}^n$. Let $q \in C_f^\circ \cap F^{-1}(E_f^\circ)$ and $a := F(q)$, then *there is no kinematic singularity at q* (or sometimes we say there is no kinematic singularity at a, depending on the space we want to focus) iff there exist an open neighbourhood $U_q \subset C_f^\circ$ of q and a chart $\psi : V_a \to V$ of $SE(3)$, with $V_a \subset F(U_q)$ open, $a \in V_a$ and V an open subset of $\mathbb{R}^6$, such that the Jacobian of the function $\psi \circ F|U_q : U_q \to V$ has a rank equal to 6. This is like saying that we can locally apply the implicit function theorem on F at q. Otherwise we say that *there is a kinematic singularity at q*.

Problem 1. We state a selection of fundamental problems most common in the area of robotic motion planning:

1. Let $a, b \in E_f$, then find a continuous (not necessarily smooth) path $p \in \mathcal{P}$ such that $F(p(0)) = a$ and $F(p(1)) = b$. This problem is called the *fundamental path planning problem (for points $a, b \in E_f$)* in robotics.

2. Let $a, b \in E_f$ and a cost function $f_c : \mathcal{P} \to [0, \infty]$, then solve the fundamental path planning problem for $a, b \in E_f$ such that $f_c(p)$ is minimum. This problem is a *path optimization problem.*

3. Let $a, b \in E_f$ and a path $p_E : [0,1] \to E_f$ with $p(0) = a$ and $p(1) = b$, then find a path $p \in \mathcal{P}$ such that $F \circ p = p_E$. This problem is the *path planning problem in the cartesian space for points $a, b \in E_f$.*

4. Let $a, b \in E_f$ then find a continuous path $p \in \mathcal{P}$ such that $F(p(0)) = a$ and $F(p(1)) = b$ such that for all points $r \in [0,1]$, there is no kinematic singularity at $p(r)$. This is the *path planning problem with (kinematic) singularity avoidance.*

5. Solve the path planning problem with singularity avoidance in such a way that $F \circ p$ is actually a piecewise linear path in E_f.

Our desire is that we get algorithms that gives us solution to these problem with polynomial time complexity.

Most experts try to solve the above problems non-analytically, in most cases they use probabilistic algorithms.

2 Path Collision Check Optimization

Static collision check at a configuration is just a verification whether the robot is colliding with anything at this configuration.

Definition 4. A line between a and b in C is called a *direct (robotic) path.* We divide the line between a and b into equal k-parts (using a given metric in $\mathbb{R}^n$) which we call the number of *divisions* for the collision check. The *resolution* is the distance of these equidistant k parts. The *division points* is the set of the endpoints of the k parts, an element of this set is then a *division point.* We say that the direct path between a and b is *discretely colliding* (with the given resolution) if there is a collision at at least one of the division points. Otherwise we say that the direct path between a and b is not discretely colliding. This whole process of testing whether a direct path between point a and b is discretely colliding is called a *direct path discrete collision check* (between points a and b).

A piecewise linear path in C with finite pieces is called a *robotic path.* And a *(robotic) path discrete collision check* is a direct path discrete collision check on all the linear pieces of the robotic path.

The *collision length* of a direct path is just the longest length of a direct path for which there is a collision. More formally:

Definition 5. Let $L \subset C$ be a direct path of a robot. Then the *collision length* of the path L is

$$\sup\{dist(l) \, : \, l \text{ is a connected component in } L \backslash C_f\}$$

In most industrial applications, a discrete path collision check is used (and a continuous path collision check is often not implemented). A typical direct path collision check is usually performed linearly, see Algorithm 1.

Algorithm 1: Linear direct path collision check

```
Input: Division $k$, configuration points $a$ and $b$ for which a direct path between them should be
       discretely collision checked
Output: collision true or false
$res = ||b - a||/k$;
$v = \text{normalize}(b - a)$;
$t = a - v \cdot res$;
repeat
    $t = t + v \cdot res$;
    if robot collides at $t$ then
        return true;
    end
until $t = b$;
return false
```

Often we do a discrete path collision check that has a significantly better performance (than the linear path collision check), especially if the path being checked has a collision. A pseudocode is shown in Algorithm 3.

Assume first that the number of divisions, k, of a direct path between points a and b is a power of 2, then we need only look at Algorithm 2. Define C_{-1} to be the empty set and

$$C_i := \{a + (b - a)\frac{j}{2^i} \, : \, j = 0, \dots, 2^i\} \qquad i = 0, \dots, \log_2(k)$$

In words C_i is the division points of a direct path between a and b with $2^i \leq k$ as number of divisions. So checking for collision at points in C_i is just doing a direct path collision check with coarser resolution. Algorithm 2 checks for collision with the coarsest division (points a and b) and keeps dividing the number of divisions by two and reiterating the collision check with this

Algorithm 2: binary direct path collision check when the number of division of the direct path is a power of 2

```
Algorithm: BinCheck1
Input: Division $k$ that is a power of 2, direct path with endpoints $a$ and $b$
Output: discrete collision true or false
$j = \log_2(k)$;
$i = 0$;
for $i <= j$ do
    if robot collides at any point in $C_i \backslash C_{i-1}$ then
        return true;
    end
    $i = i + 1$;
end
return false
```

number of division, keeping in mind that we need not recheck the division points checked during the coarser path collision check (thus in the loop we check division points in C_i not in C_{i-1}).

The algorithm we presented was for direct path collision checks with a number of division that is a power of 2. To do direct path collision check for paths with an arbitrary number of divisions we propose Algorithm 3. This algorithm regards the division number as a 2-adic representation and then we do binary collision check between shorter paths which combine to make our path but whose division numbers are all power of 2 (so we refer to Algorithm 2 to do their collision check).

Algorithm 3: binary direct path collision check

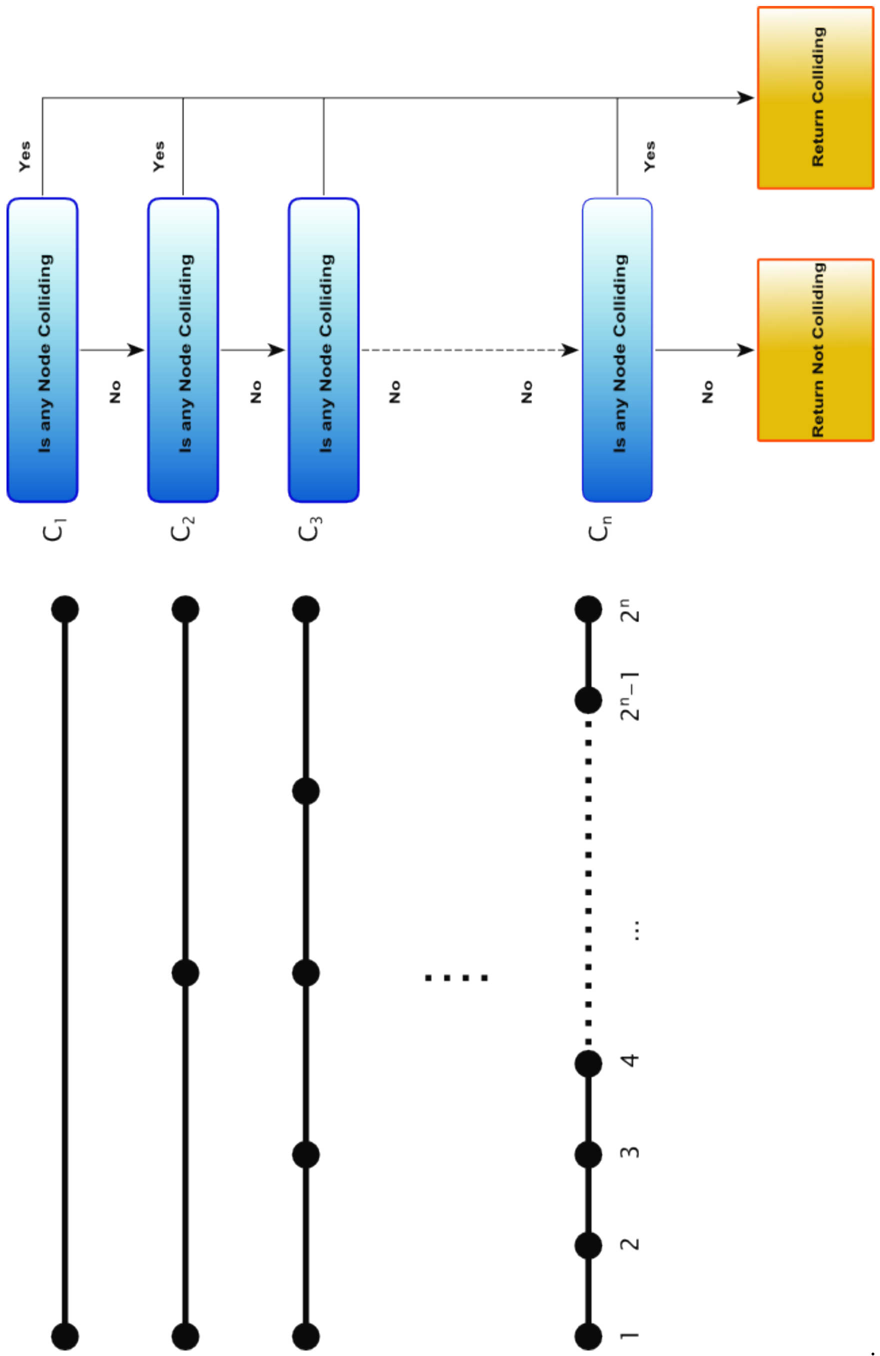
```
Input: Division $k$ , direct path with endpoints $a$ and $b$
Output: discrete collision true or false
Get the (binary) 2-adic representation of $k$ say:
$k = \sum_{i=0}^{l} 2^{j_i}$;
$res$ = dist(b - a) $/k$;
$v$ = normalize$(b - a)$;
$m = 0$;
for $m < l$ do
    if BinCheck1($a + v \cdot res \sum_{i=0}^{m} 2^{j_i}, a + v \cdot res \sum_{i=0}^{m+1} 2^{j_i}$) then
        return true;
    end
    $m = m + 1$;
end
return false;
```

It is easy to argue why this collision check is faster than linear direct path collision check. For simplicity we assume that k (the number of division) is a power of 2, the argument can easily extended for an arbitrary k. If the collision length of a direct path is just as big as its resolution and if there is a point of collision only in $C_k \backslash C_{k-1}$ (for division number k), then the performance of this collision check should be at least as good as that of linear path collision check (in our analysis we also assume that the probability that a collision occurs is equally distributed between division points). There is a little bit of time loss due to $\log_2$ but this is so insignificant that we can conclude that in this special situation the binary collision check will perform just as good as linear collision check. Now we assume that the path has a collision. We can expect a collision with a lower number of division (i.e. coarser resolution). The binary algorithm offers a full direct path collision check with coarser resolution and if this does not suffice it keeps making the resolution finer until we reach our desired resolution (we are making advantage of the fact that if we collide with a coarser resolution we will collide with a finer resolution as well).

Let η_i be the average number of iteration needed to find a collision, by linear collision check, for a path with 2^i division and let $0 \leq p_i \leq 1$ be the probability that there is only a collision in $C_i \backslash C_{i-1}$ (for $i = 1, \ldots, k$). Then for a path of division 2^k the average number of iteration needed using linear collision check is η_k. Using a binary collision check this average iteration is reduced to

$$\sum_{i=1}^{k} \eta_i p_i$$

Since the sequence $\{\eta_i\}_{i=1}^{k}$ is monotonically increasing, the average iteration cannot be greater than η_k (it is strictly less than η_k if $p_k \neq 1$, and by experience this is almost always true for a binpicking application). Thus we have shown that it will take less iteration in average to do a binary direct path collision check than using a linear direct path collision check (the details generalizing this for an arbitrary k is left to the reader).


Is any Node Colliding
Yes
No
Return Colliding
Return Not Colliding
C_1
C_2
C_3
C_n
1
2
3
4
...
2^n-1
2^n

3 RRT algorithm

Here we show an algorithm that solves the fundament path planning problem. This algorithm was first developed and introduced by Lavalle and Kuffner in [6] (see Algorithm 5). They called the algorithm RRT (rapidly growning random tree) algorithm that generates a random point in the collision-free configuration space and then tries to connect the nearest node of the active tree to the random point and for each extended node it tries to check for a connection to the goal node.

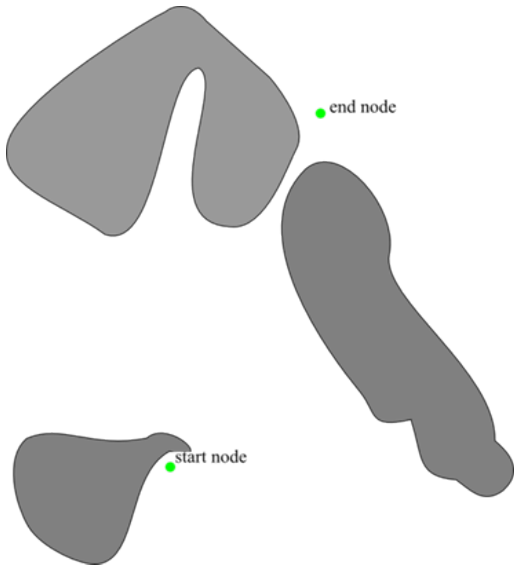
Algorithm 4: Extend, RRT

```
Algorithm: Extend
Input: tree $T$, new node $q_{\text{new}}$, $\epsilon > 0$
Output: new node $q_{\text{new}}$, Advanced or Trapped or Reached
$q$=randomSample();
$q_{\text{near}} = \text{nearest(q, T)}$;
$q_{\text{new}} = q_{\text{near}} + \epsilon \cdot \text{normalize}(\text{q} - \text{q}_{\text{near}})$;
if not collide($q_{\text{near}}, q_{\text{new}}$) then
    $T$.addVertex($q_{\text{new}}$);
    $T$.addEdge($q_{\text{near}}, q_{\text{new}}$);
    if $q_{\text{new}} = q$ then
        return Reached;
    else
        return Advanced;
    end
end
return Trapped;
```

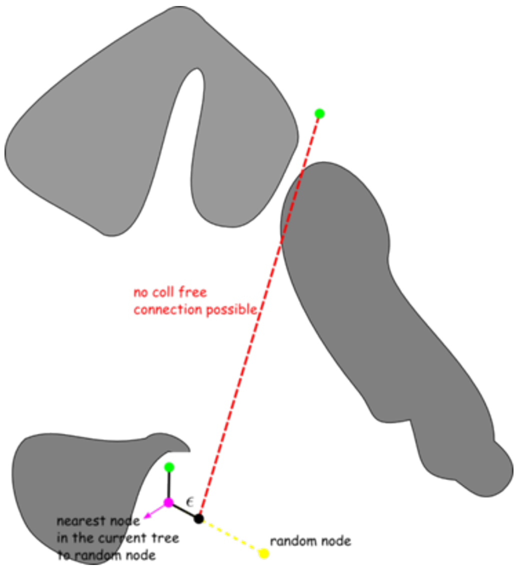
Algorithm 5: Classical RRT

```
Algorithm: RRT
Input: start node $q_{\text{start}}$, goal node $q_{\text{goal}}$, $\epsilon > 0$, timeout
Output: if found within timeout: path from $q_{\text{start}}$ to $q_{\text{goal}}$
TreeA = Tree($q_{\text{start}}$);
while time $\leq$ timeout do
    if not Extend1(TreeA, $q_{\text{new}}, \epsilon$)=Trapped then
        if Connect($q_{\text{goal}}$, $q_{\text{new}}$)=Reached then
            return Path(TreeA, $q_{\text{goal}}$);
        end
    end
end
```

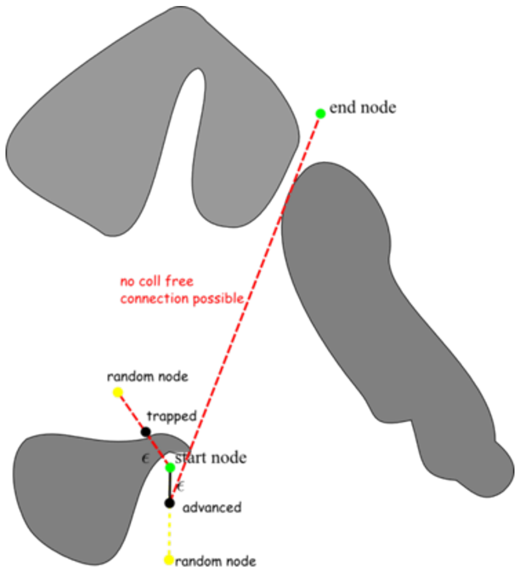
The tree created in the classical RRT hardly have collinear nodes in the collision-free configuration space (see Fig 1d)



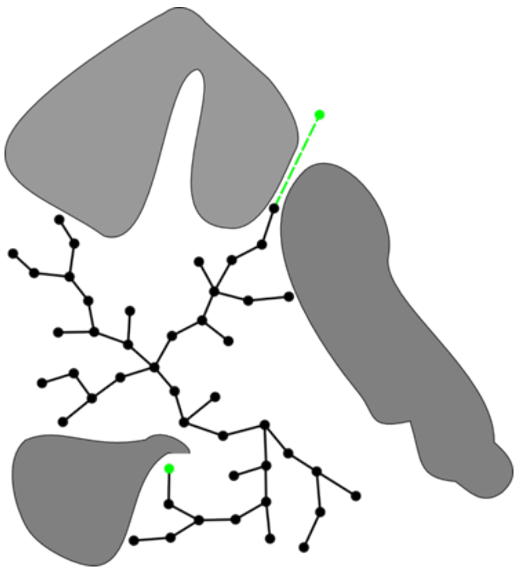
(a) First



(b) Second



(c) Third



(d) Fourth

5 Special Problems

There are situations when we want to change the collision geometry K and sometimes we would like to also change the last link L_n of the robot. This happens in industrial problems such as bin-picking.

The binpicking problem can be informally and naively described in the following way: Given a robot and a container containing randomly placed workpieces (i.e. congruent compact subsets of $\mathbb{R}^3$), the robot should perform the following actions:

1. Travel collision-free from a start-point to a grip-point of a selected workpiece in the container
2. Grip the workpiece in the container
3. Travel collision-free with the workpiece (thus the last link changes geometry having the workpiece attached to it, this attached workpiece is called a *dynamically attachable frame* or DAF) to a certain goal-point and release the workpiece at the goal-point
4. Repeat the last three steps until either the container is empty or a collision-free path by the latter steps are not possible anymore.

This is a very simple way of defining the binpicking problem. In industrial applications this problem is far more complex and there are a lot of intermediate steps between the steps we defined. In fact this a very challenging problem from an industrial and a scientific perspective.

When a workpiece is attached to the last link then the geometry of the last link changes and sometimes even the collision geometry of the environment K changes. Naturally, this also changes the collision-free space C_f. In some cases it is a restriction of the original collision-free space, but this is not always gauranteed. Though the new robot will still have the same kinematic structure (forward kinematics on C_i). We thus define a robot based on $\mathcal{R}$

Definition 6. Given an abstract robot $\mathcal{R}$, a *robot based on* $\mathcal{R}$ is another robot $\mathcal{R}'$ with everything similary defined as $\mathcal{R}$ except that the n-th link L'_n is possibly another compact subset of $\mathbb{R}^3$ that contains L_n and the static collision geometry K' is possibly different from K. Thus the collision-free configuration and cartesian spaces C'_f and E'_f respectively are different.

Now we are ready to formally define the binpicking problem. In fact once a robot based on $\mathcal{R}$ is defined, it is very easy to define bin-picking mathematically:

Definition 7. Let $\mathcal{R}$ be an abstract robot. Let $\mathcal{R}_1, \dots, \mathcal{R}_k$ with $k \in 2\mathbb{N}$ be abstract robots based on $\mathcal{R}$ with collision-free configuration spaces $C_1, \dots, C_k$ and collision-free cartesian spaces $E_1, \dots, E_k$ (so we regard the forward kinematics of these robots using these spaces as domain and codomain). Now let

$$\{(a_i, b_i) \in E_i^2 \,:\, i = 1, \dots, k\}$$

Then solve the fundamental path planning problems for robots $\mathcal{R}_i$ with endpoints a_i and b_i.

Why does a series of solutions to even number of fundament path planning problems solve the binpicking problem? Half of the path planning problems are for the robot to travel without an attached workpiece to the container to a specific *grasping point.* The other half of the path planning problem is for the robot to travel with a picked workpiece from the grasping point to a goal position, thus in this case the last link has changed geometry and the geometry of the environment has possibly changed (for instance due to the physics involved: workpieces being moved and rearranged in the container while the grasping and the picking is taking place.)

We have described bin-picking in the simplest possible form we could. In practice there is more difficulty involved and in the industry it is often required that the path be partially planned in the cartesian space (i.e. a hybrid of path planning in the configuration space and path planning in the cartesian space). Often, for instance, aside from the grip- point it is required to have a pre-grip point, a post grip-point and it is then required to have a linear path in the cartesian space between the these points. Another complex issue in bin-picking is when the workpieces are

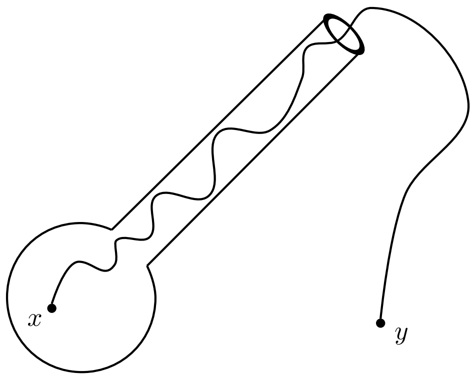


Figure 2: An illustration of the bottleneck situation

placed over each other in an intertwined way and a geometric condition arises which we often call the *bottleneck* situation. To describe the bottleneck is beyond the scope of this paper but in very simple terms: Suppose the measure in the configuration space is quite small at a certain region and quite large at another region connected to the smaller measure region. The bottleneck then arises if it is hoped to plan a path from a point in the larger region in such a way that the path should always pass the smaller region. To illustrate see Figure 2

We modified this classical RRT algorithm to produce a better result (in general we had an average of 80% less iteration on the while loop for the binpicking scenario) for the binpicking problem wherein the bottleneck situation is significant i.e. paths were found faster and in our specific binpicking scenario the average number of points of a path did not exceed 13 (in the classical bidirectional RRT we had an average number of 100 points for a path).

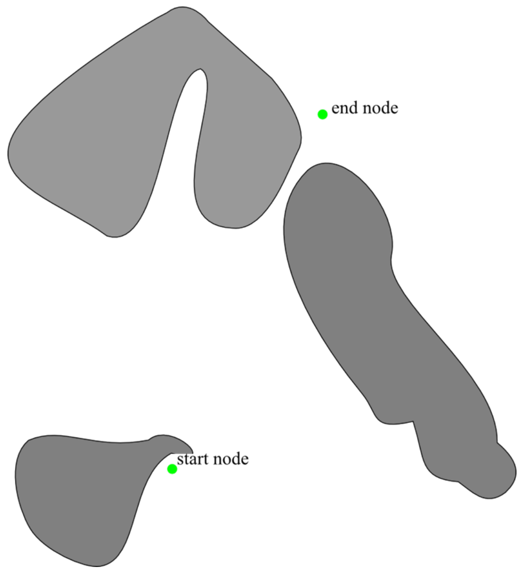
There are modification of RRT that allows two trees growing rapidly, one on the side of the goal and the other on the side of the start configuration. This RRT is called *bi-directional* RRT. It is shown that for some situations the bidirectional RRT results into a faster solution to the fundamental path planning problem. We will not go into this topic, but a uni-directional is often preferred and sufficient in bin-picking because we the arm movement within the bin suffices to reposition the robot arm to a configuration that will allow a direct path from that position to an end position outside the bin. Therefore, the tree is created only within the region of the bin. Here we modified the *Extend* algorithm and used *connectScaled* (see Algorithm 6) instead of simply connecting the new node. We then use the proposed algorithm as seen in Algorithm 7, which we will call *RRT-Binpicking* or *RRTB*.

The main difference between classical unidirectional (i.e. with one tree) RRT and RRTB is that for every pass of the the while loop of RRTB we actually perform several extends in the same direction, this is hidden in the while loop of *connectScaled*. Thus the end-result is a tree with more collinear branches as compared to a classical RRT tree with the same number of vertices.

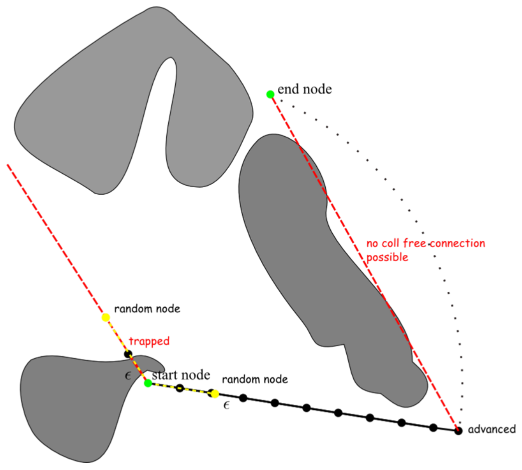
Random-sampling is performed in the main algorithm of RRTB. The sample is then scaled and we attempt to connect this random sample to the nearest node. The scaling and connecting is shown in Algorithm 6.

The reason why we prefer more collinear nodes in the tree is that it will help us optimize the path (and hopefully arrive to the goal) much faster than if we used a totally random tree with less or no collinear nodes.

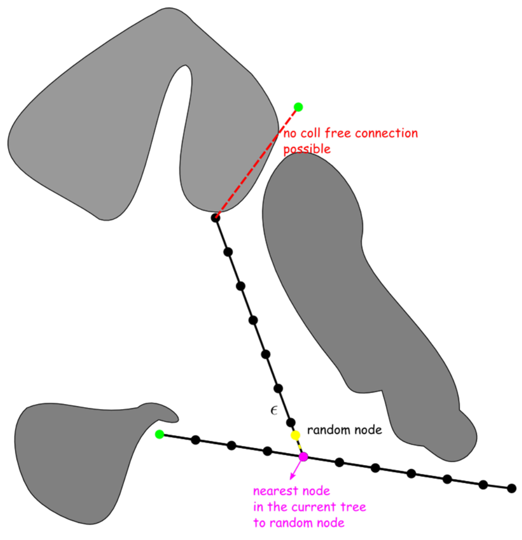
An illustration of RRTB can be seen in Figure 3d



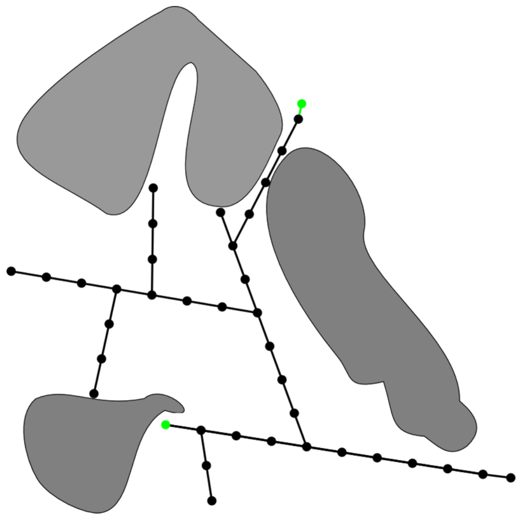
(a) First



(b) Second



(c) Third



(d) Fourth

```
Algorithm 6: connectScaled
  Algorithm: connectScaled
  Input: node $q, q_{\text{goal}}$, tree $T$, new node $q_{\text{new}}$, $\epsilon > 0$
  Output: new node $q_{\text{new}}$, Advanced or Trapped or Reached
  $q_{\text{near}} = \text{nearest(q, T)}$;
  scale $= |q_{\text{near}} - q_{\text{goal}}|/|q - q_{\text{near}}|$ ;
  $q = q_{\text{near}} + (q - q_{\text{near}})\text{scale}$;
  $v = \epsilon \cdot \text{normalize}(\text{q} - \text{q}_{\text{near}})$;
  $s$ =Advanced;
  while $s$=Advanced do
      extend $q$ (using Extend-2) recursively by $v$ until either $q_{\text{new}} = q$ in which case we return
      Reached otherwise return Trapped
  end
```

```
Algorithm 7: RRT-Binpicking
  TreeA = Tree($q_{\text{start}}$);
  while time $\leq$ timeout do
      $q_0$ = randomSample();
      connectScaled(TreeA, $q_0, q_{\text{new}}, \epsilon$);
      if not collide($q_{\text{new}}, q_{goal}$) then
          TreeA.addVertex($q_{\text{new}}$);
          TreeA.addEdge($q_{\text{new}}, q_{\text{goal}}$);
          return Path(TreeA, $q_{goal}$);
      end
  end
```

6 Coverage Planning

In this section we describe a problem in robotics called *coverage planning*. This problem is by no means new, even before robotic similar problem was evolved in the ancient times. A common "coverage planning" in agriculture and routing is to know how one can plow a field with an ox in such a way that the whole field would be raked with minimum time and ox-power. In modern days, cartography or surveying of a helicopter above a landscape benefits from solutions of the coverage planning problem. This problem is however rather new when used with robotics. Surveying with helicopters hasn't been around 100 years ago. A common problem is to detect a part (e.g. automobile part, or the surface of the body or wings of an airplane) for cracks and defects. It is therefore not surprising that this problem is quite related to the travelling salesman problem (TSP) and the minimal hamiltonian path problem. In general we wish to minimize the cost of the path of the robot and yet cover the entire subspace of the affine space.

Definition 8. Given a finite sequence of k points in E_f (with $k \geq 2$) say

$$S_k := \{e_1, e_2, \ldots, e_k\} \subset E_f$$

and a path cost function $f_c : \mathcal{P} \to \mathbb{R}^+$. Find a permutation (re-ordering) $\sigma : S_k \to S_k$ such that

1. There are speicific solutions
$$\{p_1, \ldots, p_{k-1}\} \subset \mathcal{P}$$
to the fundamental path-planning problems for the pairs $(e_{\sigma(i)}, e_{\sigma(i+1)}))$ for $i = 1, \ldots, k-1$.

2. For any other permutation of $\tau : S_k \to S_k$ and solutions
$$\{p'_1, \ldots, p'_{k-1}\} \subset \mathcal{P}$$

of the fundamental path-planning problems for the pairs $(e_{\tau(i)}, e_{\tau(i+1)}))$ for $i = 1, \ldots, k-1$, we have

$$\sum_{i=0}^{k-1} f_c(p_i) \leq \sum_{i=0}^{k-1} f_c(p'_i)$$

The solution to this problem (i.e. solving the permutation σ and the paths $\{p_1, \ldots, p_k\}$ is the solution to the *coverage planning problem* (for k-points in the collision-free reachable cartesian space).

TSP or minimal hamiltonian path problem is not easy to solve if the points are many. If the points are dense (in many application this is the case) then we often separate or ignore fundamental path-planning problems. We regard these finite points and find a permutation such that the cost is minimum (not regarding collision, so we work in C) and in such a way that erasing points from the sequence derived from the permutation would give us a minimum solution for cost had the points not been there in the first place.

In this example we show a sample algorithm that first finds the permutation using help of the lexicographic ordering.

A naive way that we approached this problem was to get a lexicographic ordering for trans(S_k) and considered this to be the the permutation on S_k. A sample algorithm of lexicographic ordering can be seen in Algorithm 8.

Algorithm 8: compare-lexico

```
Algorithm: lexicographic ordering with $x \ll y \ll z$
Input: $a, b \in \mathbb{R}^3$ considered as two triples
Output: $a < b$ true or false
for $i \leq 3$ do
    if $a[i] \neq b[i]$ then
        return $a[i] < b[i]$
    end
end
```

After having this permutation for S_k we do path collision checks for the image of S_k by this permutation. Without loss of generality we could consider the permutation to be the identity map. Then we check direct path collision in the following way:

Algorithm 9: PathCheck

```
Algorithm: Simple Direct Path Search
Input: $a, b \in E_f$ and possibly a point $x$ in the fiber $F^{-1}(a)$
Output: linear path in $C_f$ with endpoints $x, y \in C_f$ if such a path exists
for Every point $y$ in the fiber $F^{-1}(b)$ do
    if $x$ is not given then
        for Every point $x$ in the fiber $F^{-1}(a)$ do
            if The line $\overline{xy}$ is in $C_f$ then
                return $\overline{xy}$
            end
        end
    end
    else
        if The line $\overline{xy}$ is in $C_f$ then
            return $\overline{xy}$
        end
    end
end
```

Algorithm 10: covplanner

```
Algorithm: Heuristic "Coverage Planning"
Input: $S_k$
Output: A finite sequence $\{x_1, \dots, x_m\}$ in $C_f$ such that $F(T) \subset S_k$ and such that $\overline{x_{i-1}x_i} \subset C_f$ for
        $i = 2, \dots, m$.
Without loss of generality suppose PathCheck($a_1, a_2$) returns a value $\overline{x_1x_2}$ (otherwise we iterate
until we get two sequential points in $S_k$ for which there is PathCheck);
Let $T$ be an empty sequence in $C_f$;
$T$.insert($x_1$), $T$.insert($x_1$);
$a = a_2 \quad x = x_2$;
for $i = 3, \dots, k$ do
    if PathCheck($a_i, a, x$) returns a line $\overline{xy}$ in $C_f$ then
        T.insert($y$);
        $x = y \quad a = a_i$;
    end
end
```

References

[1] **C. Ericson**,
Real-Time Collision Detection, Morgan Kaufmann Publishers 2005

[2] **L-P. Ellekilde, J.A. Jorgensen**,
RobWork: A Flexible Toolbox for Robotics Research and Education, International Symposium on Robotics (ISR) 2010, Munich, Germany

[3] **S. Karaman, E. Frazzoli**,
Incremental Sampling-based Algorithms for Optimal Motion Planning, International Journal of Robotics Research, Vol. 30, No. 7, June 2011

[4] **L-E. Kavraki, J-C. Latombe, M.H. Overmars, P. Svestka**,
Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces, IEEE Transactions on Robotics and Automation, Vol. 12, pages 566-580, 1996

[5] **Kineo C.A.M.**,
KineoWorks^{TM},
http://www.kineocam.com/kineoworks-library.php
Last Accessed: 23.01.2013

[6] **J.J. Kuffner, S.M. LaValle**,
RRT-Connect: An Efficient Approach to Single-Query Path Planning, IEEE International Conference on Robotics and Automation 2000

[7] **S.M. LaValle**,
Planning Algorithms, Cambridge University Press, Copyright S.M. Lavalle 2006, http://planning.cs.uiuc.edu/

[8] **S.M. LaValle, J.J. Kuffner**,
Randomized Kinodynamic Planning, International Journal of Robotics Research, Vol. 20, No. 5, pp. 378-400, May 2001

[9] **M. Strandberg**,
Yet Another OBB-Tree Implementation, http://sourceforge.net/projects/yaobi/, 2005
Last Accessed: 01.06.2011