A simulated annealing approach to solve path planning

Stefano Martina
stefano.martina@stud.unifi.it
Università degli Studi di Firenze
Scuola di Scienze Matematiche, Fisiche e Naturali
Corso magistrale di Informatica

April 6, 2016

Abstract

This work is about a *simulated annealing* method for performing path planning.

A *B-spline* curve is used to represents the path, and the position of the control vertexes is considered as the status of the system. The number of points of the curve that cross an obstacle is the measure of a *Lagrangian multiplier* that, used together with the status of the system, form a multidimensional surface on which saddle points are the desired configurations of (local) minimal path that don't cross obstacles. A simulated annealing method is finally used to find the optimal between those saddle points.

1 Introduction

The *Path planning* problem is a common problem in robotics, the purpose is to find the best route in a bidimensional (or tridimensional) space with obstacles in it, from one point to another. Many techniques exists for calculating the path, in this work we analyze the application to this problem of a statistical method commonly used in optimization problems.

2 Prerequisites

2.1 Splines

A *spline* $\mathbf{S}(t)$ is a parametric function - defined in a certain space (bidimensional in this work) as a function of a certain parameter t - that is composed from polynomial functions, piecewise in intervals in t.

Formally we define the parametric domain

$$[a, b] \subset \mathbb{R}$$

and a partition of that space defined by the *nodes*

$$\tau = \{\tau_0, \dots, \tau_l\}$$

such that $a = \tau_0 < \tau_1 < \dots < \tau_{l-1} < \tau_l = b$ forming the intervals

$$I_i = \begin{cases} [\tau_i, \tau_{i+1}) & \text{if } i = 0, \dots, l-2 \\ [\tau_i, \tau_{i+1}] & \text{if } i = l-1 \end{cases}$$

is possible to define the following spaces:

Piecewise polynomial functions space $P_{m,\tau}$ is the space of the functions that are polynomials of maximum degree m in each interval I_i of the partition, formally:

$$P_{m,\tau} = \{f : [a,b] \to \mathbb{R} \ \mid \ \exists p_0 \dots p_{l-1} \in \Pi_m \text{ such that} \\ f(t) = p(t),\ \forall t \in I_i,\ i = 0 \dots l-1\}$$

where Π_m is the space of the polynomials of degree from 0 to m. The dimension of this space is $l \cdot (m+1)$ because have l polynomials and the dimension of Π_m is $m+1$.

Classic spline functions space $S_{m,\tau}$ is the space of the piecewise polynomial functions that have continuity C^{m-1} in the junctions of the intervals, formally:

$$S_{m,\tau} = P_{m,\tau} \cap C^{m-1}[a,b].$$

The dimension of this space is $l \cdot (m+1) - (l-1) \cdot m \ = \ l + m$.

Generalized spline functions space $S_{m,\tau,M}$ is the space of piecewise polynomial function that have a discontinuity or a continuity until C^{m-1} in the junctions of the intervals, determined by the values of the multiplicity's vector

$$M = \{m_1, \ldots, m_{l-1}\}, \quad m_i \in \mathbb{N}, \quad 1 \leq m_i \leq m+1.$$

Formally:

$$\begin{aligned} S_{m,\tau,M} = \{f : [a,b] \to \mathbb{R} \ \mid \ & \exists p_0 \ldots p_{l-1} \in \Pi_m \text{ such that} \\ & f(t) = p(t), \ \forall t \in I_i, \ i = 0 \ldots l-1 \text{ and} \\ & p_{i-1}^{(j)}(\tau_i) = p_i^{(j)}(\tau_i), \ j = 0, \ldots, m - m_i, \ i = 1, \ldots, l-1\}. \end{aligned}$$

The dimension of the space is between the previous two, and is true that:

$$\Pi_m \subseteq S_{m,\tau} \subseteq S_{m,\tau,M} \subseteq P_{m,\tau},$$

in fact:

- if $m_i = 1$ for all $i = 1, \ldots, l-1$, then $S_{m,\tau,M} = S_{m,\tau}$;
- if $m_i = m+1$ for all $i = 1, \ldots, l-1$, then $S_{m,\tau,M} = P_{m,\tau}$.

2.1.1 Truncated-powers base for classic splines

A truncated power $(t - \tau_i)_+^m$ is defined by

$$(t - \tau_i)_+^m = \begin{cases} 0, & \text{if} \quad t \leq \tau_i \\ (t - \tau_i)^m, & \text{otherwise.} \end{cases}$$

Is possible to demonstrate that the functions

$$g_i(t) = (t - \tau_i)_+^m) \ \in S_{m\tau}, \quad i = 1, \ldots, l-1$$

are linearly independents, and that

$$1, t, t^2, \ldots, t^m, (t - \tau_1)_+^m, \ldots, (t - \tau_{l-1})_+^m$$

form a base for the classic spline functions space. A generic element from this space can be expressed like

$$\mathbf{S}(t) = \sum_{i=0}^{m} \mathbf{c_i} \cdot t^i \ + \sum_{j=1}^{l-1} \mathbf{d_i} \cdot (t - \tau_j)_+^m$$

but this form is not a practical representation of a spline because there isn't an intuitive correlation between the points $\mathbf{c_i}$, $\mathbf{d_j}$ and the curve. For this purpose we define the *B-splines base* in section 2.1.2.

2.1.2 B-Splines base for classic splines

B-splines are splines defined with a specific base. In this paragraph we consider only the classic splines $S_{m,\tau}$ and not the generalized splines $S_{m,\tau,M}$, furthermore we consider the order $k = m+1$.

For defining the B-splines we need to extend the partition vector $\tau = \{\tau_0, \cdots, \tau_l\}$ with $k-1$ nodes to the left and $k-1$ to the right, so we define a new vector

$$T = \{t_0, \dots, t_{k-1}, t_{k-1}, \dots, t_{n+1}, t_{n+2}, \dots, t_{n+k}\}$$

such that

$$t_0 \leq \cdots \leq t_{k-2} \leq t_{k-1} \equiv \tau_0 \equiv a < \cdots < t_{n+1} \equiv \tau_l \equiv b \leq t_{n+2} \leq \cdots \leq t_{n+k}.$$

τ have $l+1$ elements, so we can calculate the value of

$$n = l + k - 2,$$

and the dimension of $S_{m,\tau}$ is

$$l + m = l + k - 1 = n + 1$$

that is the number of necessary bases for the space.

The $n+1$ basis $N_{i,k}(t)$ of the B-splines of order k for $i = 0, \dots, n$ are defined by the recursive formula:

$$N_{i,1}(t) = \begin{cases} 1, & \text{if} \quad t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$N_{i,k}(t) = \omega_{i,k-1}(t) \cdot N_{i,k-1}(t) \ + \ (1 - \omega_{i+1,k-1}(t)) \cdot N_{i+1,k-1}(t)$$

where

$$\omega_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i}.$$

The elements of the classic splines space can be expressed in the form

$$\mathbf{S}(t) = \sum_{i=0}^{n} \mathbf{v_i} \cdot N_{i,k}(t),$$

and this representation is more convenient respect to the one of section 2.1.1 because the curve $\mathbf{S}(t)$ roughly follow the shape given by the points $\mathbf{v_i}$. Those points are called *control vertexes* and the polygon defined by them is called *control polygon* and they can be used to control the shape of the curve.

2.2 Lagrangian relaxation

A general constrained discrete optimization problem can be expressed in the form:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g(x) = 0. \end{aligned} \tag{1}$$

Where x is the state of the system in a discrete space X, $f(x)$ is the function to minimize, and $g(x) = 0$ is the constrain. the functions can also be in a multidimensional discrete space, in that case the x is a vector $\mathbf{x} = (x_1, \dots, x_n)$ of variables.

For solving this class of problem is necessary a *Lagrange relaxation* method, that augment the variable space X by a *Lagrange multiplier* space Λ of dimension equal to the number of constraints - one in the problem (1).

The *generalized discrete Lagrangian function* corresponding to the problem (1) is:

$$L_d(x, \lambda) = f(x) + \lambda H(h(x)). \tag{2}$$

Where λ is a variable in Λ, and if the dimension of Λ is more than one λ must be transposed in formula (2); $H(x)$ is a non negative function with the property that $H(0) = 0$, the purpose is to transform $g(x)$ in a non negative function - if $g(x)$ isn't already not negative - for instance can be $H(g(x)) = |g(x)|$ or $H(g(x)) = h^2(x)$.

Under the previous assumptions the set of *local minima* in problem (1) - that respect the constraints - coincide with the set of *discrete saddle point* in the augmented space. A point (x^*, λ^*) is a discrete saddle point if:

$$L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*)$$

for all $x \in \mathcal{N}(x^*)$ and for all $\lambda \in \Lambda$, where $\mathcal{N}(x^*)$ is the set of all neighbours of x^*.

For resolving the optimization problem (1) is necessary to calculate all the discrete saddle points (x^*, λ^*) using some optimization method (i.e. simulated annealing) on the surface represented by equation (2), and then choose the one that minimize $f(x^*)$. x^* is then the desired minimum.

3 Path planning

In this project the space is bidimensional and the obstacles are represented as an array of vertexes and are closed polygons - the last vertex is connected to the first one. The path is calculated from a provided point to another provided point in the space, and is assumed that the user don't choose one or both of those points inside an obstacle.

Is also possible - and advised - to add a bounding box around the scene.

The path planning is performed doing a simulated annealing using, as the state of the system, the configuration of the vertexes of the path; a Dijkstra algorithm is performed for finding the initial state of the system, using a pruned Voronoi diagram as basis.

4 Algorithm

4.1 Initialization

The first phase is to initialize the method with an initial state. For doing that, the algorithm first build a graph of possible routes around obstacles, then find a shortest path on that graph, and finally normalize that path.

4.1.1 Building the graph

1. initially the algorithm distribute a series of points along the edges of the obstacles and of the bounding box;
2. then build a Voronoi diagram using that points as input sites;

3. after that, transform the diagram in a graph, using the vertexes and edges of the cells as nodes and edges of the graph;
4. then delete all the edges that cross an obstacle;
5. the final step is to connect the desired start and end points to the graph - is possible to use two different methods for connecting those points, one is to connect to the nearest visible vertex of the graph, the other one is to connect to every visible vertex of the graph.

The result is a sparse graph that embrace all obstacles and that maintain equal distance from obstacles, like in figure 1.

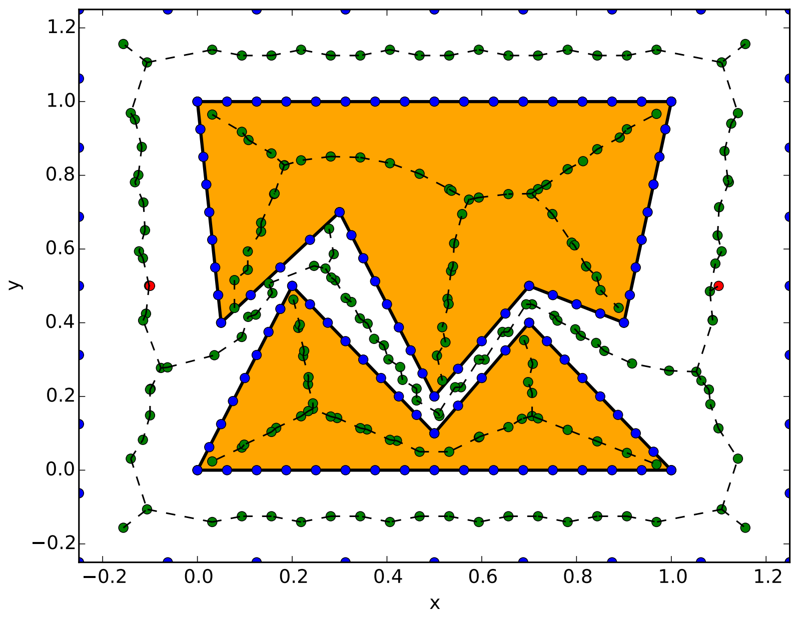

Figure 1: Pruned Voronoi graph.

In the figure the obstacles are the orange polygons; the bounding box coincide with the border of the plot; the blue dots are the ones distributed in point 1; the green points are the vertexes of the pruned Voronoi diagram, constructed on points 2, 3, and 4; the two red dots are the start and end vertexes added on point 5. Note that there are also non influential disconnected graphs inside the obstacles.

4.1.2 Shortest path in graph

The algorithm need to find a shortest path after the graph generation. For doing that use the Dijkstra algorithm from the start point to the end point.

4.1.3 Normalization of path

The purpose of the normalization phase is to have a path with vertexes that have a certain distance between them.

In the specific the algorithm go by all the vertexes of the shortest path, and measure the distance from the previous vertex. If the distance is shorter than a provided threshold merge the current and previous vertexes in a single one located in the middle. If the distance is longer than another provided threshold create a new vertex in the middle between the current and the previous ones.

4.2 Annealing

The annealing phase is the key part of the algorithm, the purpose is to make the initial path shorter, and on the same time keep the path in a state where is possible to apply a specific spline without colliding with obstacles.

4.2.1 Lagrangian relaxation applied to the project

In the project the variable space X is composed of all possible configuration of the path, or in other words is the vector $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_n)$ of all n ordered vertexes $\mathbf{v}_i = (x_i, y_i)$ of the path. The problem (1) is formulated in two different versions in the project.

- The first one is:

$$\begin{aligned} \underset{\mathbf{v}}{\text{minimize}} \quad & \mathit{length}(\mathbf{v}) \\ \text{subject to} \quad & \left| \mathit{bspline}(\mathbf{v}) \cap \bigcup_{i \in I} \mathit{obstacle}_i \right| = 0. \end{aligned}$$

  Where $\mathit{length}(\mathbf{v})$ is the sum of the length of every edge of the path, that is the sum of the distances of every consecutive pair of vertexes $\mathbf{v_{i-1}}$ and $\mathbf{v_i}$; $\mathit{bspline}(\mathbf{v})$ is the set of points of the *B-Spline* obtained as explained in sections 2.1 and 2.1.2, using the path as control polygon; $\mathit{obstacle}_i$ is the surface of the i^{th} of m obstacles, and $I = \{1, \ldots, m\}$.

- The second version is similar, but the function to minimize is $\mathit{angle}(\mathbf{v})$ instead of $\mathit{length}(\mathbf{v})$, where

$$\mathit{angle}(\mathbf{v}) = \frac{\sum_{i=2}^{n-1}(1 + \cos(\measuredangle \mathbf{v}_{i-1}\mathbf{v}_i\mathbf{v}_{i+1}))}{n-2},$$

  and

$$\mathit{cos}(\measuredangle \mathbf{v}_{i-1}\mathbf{v}_i\mathbf{v}_{i+1}) = \frac{(\mathbf{v}_{i-1} - \mathbf{v}_i) \cdot (\mathbf{v}_{i+1} - \mathbf{v}_i)}{\|\mathbf{v}_{i-1} - \mathbf{v}_i\| \; \|\mathbf{v}_{i+1} - \mathbf{v}_i)\|}.$$

The constraint function is not negative, and is calculated as the ratio

$$\mathit{constraint}(\mathbf{v}) = \frac{|\{\mathbf{p} \in \mathit{spline}(\mathbf{v}) \mid \exists i \in \{1, \ldots, m\} : \mathbf{p} \in \mathit{obstacle}_i\}|}{|\{\mathbf{p} \in \mathit{spline}(\mathbf{v})\}|}$$

where the points $\mathbf{p}$ of the spline are calculated in a discrete form. That function is not negative, so the Lagrangian function correspondent to equation (2) is

$$L_d(\mathbf{v}, \lambda) = \mathit{length}(\mathbf{v}) + \lambda \cdot \mathit{constraint}(\mathbf{v}), \tag{3}$$

or, alternatively, the correspondent one that use $\mathit{angle}(\mathbf{v})$ instead of $\mathit{length}(\mathbf{v})$.

4.2.2 Annealing phase

The target of the simulated annealing phase is to find the minimum saddle points in the curve represented by the equation (3).

Algorithm 1 Annealing

```
1: procedure ANNEALING($\mathbf{x}$)
2:     $\lambda \leftarrow initialLambda$
3:     $T \leftarrow initialTemperature$
4:     while not $terminationCondition()$ do
5:         for all number of trials do
6:             $changeLambda \leftarrow$ true with $changeLambdaProb$
7:             if $changeLambda$ then
8:                 $\lambda' \leftarrow neighbour(\lambda)$
9:                 $\lambda \leftarrow \lambda'$ with probability $\mathrm{e}^{-([energy(\mathbf{x},\lambda)-energy(\mathbf{x},\lambda')]^+/T)}$
10:            else
11:                $\mathbf{x}' \leftarrow neighbour(\mathbf{x})$
12:                $\mathbf{x} \leftarrow \mathbf{x}'$ with probability $\mathrm{e}^{-([energy(\mathbf{x}',\lambda)-energy(\mathbf{x},\lambda)]^+/T)}$
13:            end if
14:        end for
15:        $T \leftarrow T \cdot warmingRatio$
16:    end while
17: end procedure
```

The algorithm 1 is the annealing process, on line 2 λ and the temperature are initialized; the *while* on line 4 is the main loop and the terminating condition[1] is given by a minimum temperature or a minimum variation of energy between two iterations; the *for* at line 5 repeat the annealing move for a certain number of trials, on each iteration the algorithm probabilistically try to make a move of the state of the system, first on line 6 make a choice if moving in the Lagrangian space or in the space of the path, after that based on that choice try to move the system in a neighbouring state - in the Lagrangian space at line 8 or in the path space at line 11 - the choice is made probabilistically in the meaning that if the energy increase in the Lagrangian space or decrease in the path space the probability of choosing the new state is 1, if the energy decrease in the Lagrangian space or increase in the path space then the new state is accepted with a probability that is[2]:

$$\exp(-\frac{\Delta energy}{T}).$$

Finally at the end of every trial set, at line 15, the temperature T is cooled by a certain factor.

The *neighbour* function choose a neighbour of the state and is defined depending on the input

- for λ move uniformly in a range $[-maxLambdaPert, maxLambdaPert]$;

[1]For the animation there isn't a terminating condition.
[2]Note that $[x]^+ = \max(0, x)$.

- for path pick randomly one of the nodes, except the extremes, then pick an angle in $[0, 2\pi]$ and a distance uniformly[3] in a specific range.

The *energy* function is equivalent of L_d in the equation (3):

$$energy(\mathbf{x}, \lambda) = length(\mathbf{x}) + \lambda \cdot constraints(\mathbf{x})$$

or

$$energy(\mathbf{x}, \lambda) = angle(\mathbf{x}) + \lambda \cdot constraints(\mathbf{x})$$

depending of the chosen method. The function $length(\mathbf{x})$ returns the total length of the path represented by the points $\mathbf{x}$, $angle(\mathbf{x})$ is the mean complementary angle between every pair of consecutive segments in the path $\mathbf{x}$, and $constraints(\mathbf{x})$ return the ratio between the points inside the obstacles and all the points of the B-spline obtained using the path $\mathbf{x}$ as control polygon.

The annealing process find a saddle point probabilistically increasing the energy, moving λ and decreasing the energy moving the points.

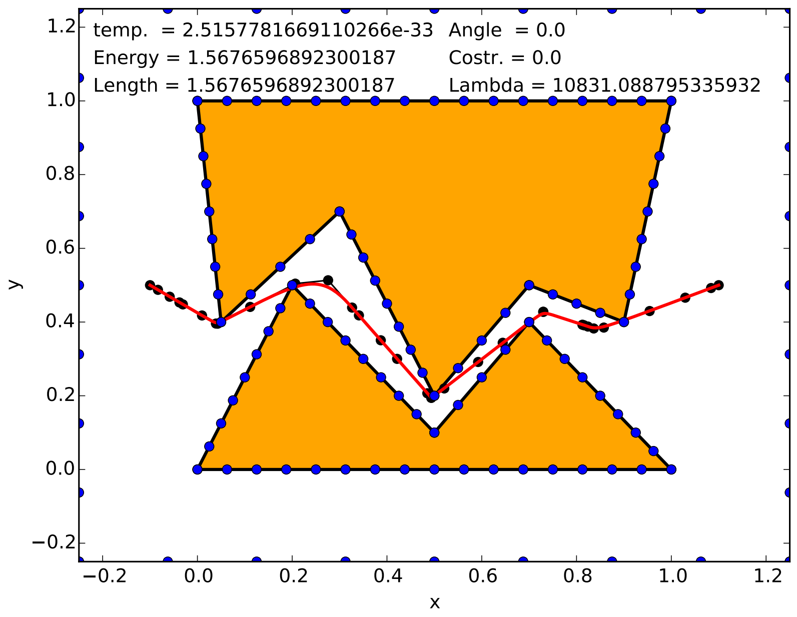

Figure 2: Final configuration after annealing.

In figure 2 is visible the state of the system after a certain amount of iterations.

[3]Or at choice with a Gaussian with mean in the perpendicular direction of the two neighbouring points.

References

[Ho and Liu, 2010] Ho, Y.-J. and Liu, J.-S. (2010). Simulated annealing based algorithm for smooth robot path planning with different kinematic constraints. In Shin, S. Y., Ossowski, S., Schumacher, M., Palakal, M. J., and Hung, C.-C., editors, *SAC*, pages 1277–1281. ACM.

[Wah and Wang, 1999] Wah, B. W. and Wang, T. (1999). Constrained simulated annealing with applications in nonlinear continuous constrained global optimization. In *ICTAI*, pages 381–. IEEE Computer Society.