POTENTIAL FIELD METHODS

Motivation

The potential field approach to motion generation consists of regarding the robot in configuration space as a unit mass particle moving under the influence of the force field F = -(gradient) U. This method is well suited when the obstacle are not known in advance, but are sensed during motion execution. Different techniques have been developed for path generation using the potential field. The depth-first technique may get stuck at local minima of the potential function. The second technique operates in a ***best-first*** mode. It deals with the local minima by ***filling*** them up. However, when the dimension, m of the configuration space becomes large, filling up local minimum wells is no longer tractable. On the other hand, the construction of a navigation function, i.e. a potential field with no local minimum other than the goal configuration, is a difficult problem that has a known solution only when the C-obstacles have simple shapes and/or when the dimension of the configuration space is small (m = 2 or 3). To overcome the abovementioned limitations, Randomized Path Planner (**RPP**) was proposed.

We describe the two techniques : Best-first planning, Randomized motion planning.

Best First Planning

Given a configuration **q** in the m-dimensional grid GC, its **p-neighbors** (1<= p <= m) are defined as all the configurations in GC having atmost p coordinates differing from those of **q**, the amount of the difference being exactly one increment in absolute value.

procedure BFP;

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
begin
   install q_init in T; [initially T is the empty tree]
   INSERT (q_init, OPEN); mark q_init visited;
   [initially, all the configurations in GC are marked unvisited]
   SUCCESS <- false;
   while ¬ EMPTY(OPEN) and ¬ SUCCESS do
      begin
        q <- FIRST(OPEN);
         for every neighbor q' of q in GC do
            if U(q') < M and q' is not visited then
               begin
                  install q' in T with a pointer toward q;
                  INSERT(q', OPEN); mark q' visited;
                  if q' = q_goal then SUCCESS <- true;
               end;
      end;
   if SUCCESS then
      return the constructed path by tracing the pointers in T
      from q_goal back to q_init;
   else return false;
end;
```

The algorithm is guaranteed to return a free path whenever there exists one in the free subset of the grid GC and to failure otherwise.

The size of GC is $O(r^m)$ where r is the number of discretization points along each coordinate axis.
If we represent OPEN as a balanced tree, INSERT and FIRST takes logarithmic time in size of OPEN which can be $O(r^m)$ in the worst case. The time complexity of the algorithm is thus $O(mr^m \log r)$.

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Randomized motion planning

Overview

We assume that the potential function, U>=0 and that any configuration **q** such that U(**q**) = 0 is a goal configuration. The RPP constructs and searches a graph G whose nodes are local minima of the potential function U. Two minima are connected by a link in the graph if the planner has constructed a path joining them.
The planner starts at the input initial configuration $\mathbf{q}_{init}$. From there, it executes a **best-first motion**, i.e. it follows the steepest descent of the potential U. The best-first motion stops when it reaches a minimum $\mathbf{q}_{loc}$. If U($\mathbf{q}_{loc}$) =0, the problem is solved and the algorithm returns the constructed path. Otherwise, it attempts to escape from the local minimum by executing a series of **random motions** issued from $\mathbf{q}_{loc}$. Each random motion is immediately followed by a new best-first motion that attains a minimum. If this minimum is different from $\mathbf{q}_{loc}$, it is installed as a successor of q_{loc} in the graph G. Hence, two adjacent minima in G are connected by a path obtained by composing a random motion and a best-first motion. We proceed like this till the goal configuration is attained or the planner gives up.

Best first motion

The potential U may not tend to infinity when the robot's configuration gets closer to the C-obstacle region. Hence a best-first motion is not collision free. The planner must verify that the next chosen configuration $\mathbf{q}_{i+1}$ lies in the free space. When m is small, best-first motions can be generated by using the m-neighborhood in GC. However, when m becomes larger, it is too large to be explored at every step. One way to take care of this is to check only a small number of randomly selected m-neighbors. If none is selected, we consider $\mathbf{q}_{loc}$ to be local minimum.

Random motion

Random walks:

A series of t motion steps such that the projection of every step along each axis x_i, i = 1, 2, ..., m is randomly

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

$DELTA_i$ or $-DELTA_i$.
Random motions are executed whenever a local minimum $\mathbf{q}_{loc}$ is encountered. Let us assume that each step takes a unit of time, so that t is the duration of the random walk. Without the loss of generalization, let us take the local minimum $\mathbf{q}_{loc}$ as the coordinates' origin. The configuration after the random walk will be $Q(t) = (Q_1(t), Q_2(t), ..., Q_m(t))$ which verifies the following properties :

- The density of $Q_i(t)$ is :

$$p_i(q_i) = \frac{1}{\Delta_i \sqrt{2\pi t}} \exp(-\frac{q_i^2}{2\Delta_i^2 t})$$

- The standard deviation of $Q_i(t)$ is :

$$D_i = \Delta_i \sqrt{t}.$$

The planner should check for collision during the random path. If the new configuration is not free, the planner randomly selects another candidate configuration.

What should be the value of t ?
Should neither be too small nor too large.

Attraction radius $\mathbf{R_i}(\mathbf{q}_{loc})$:
the distance along the x_i-axis between $\mathbf{q}_{loc}$ and the nearest saddle point of U in that direction.

The minimum distance that the robot must travel along any x_i-axis in order to escape from the local minimum is precisely $\mathbf{R_i}(\mathbf{q}_{loc})$.

Therefore, t could be defined as :

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

$$t = \max_{i \in [1,m]} \left(\frac{R_i(\mathbf{q})}{\Delta_i} \right)^2$$

Graph Searching

- Iteratively generating successors of the local minimum (under consideration) having the minimum potential value and limiting the number of random motions to a predefined value K.

Drawbacks :

1. Same local minimum may be attained several times.
2. This may also waste time exploring a local minimum well containing smaller local-minimum well imbricated in one another.

- If $U(\mathbf{q'}_{loc}) > U(\mathbf{q}_{loc})$, we ignore $\mathbf{q'}_{loc}$; otherwise it generates its successors in the same fashion. (Note : If goal configuration is encountered, the problem is solved). If no local minimum with potential value less than $\mathbf{q}_{loc}$ is attained, it is considered to be a dead-end and the search is resumed at the most recently considered local minimum whose K successors have not been generated yet.

Drawbacks:

1. If a low local minimum is attained, the planner may have difficulty attaining a lower one from it.

- In this approach, instead of memorizing the whole graph we proceed by remembering only the constructed path t connecting the initial configuration to the current configuration. The planner iteratively generates K random motions. If a lower local minimum $\mathbf{q'}_{loc}$ is attained then the planner appends the path from $\mathbf{q}_{loc}$ to $\mathbf{q'}_{loc}$ to the path t and proceeds from $\mathbf{q'}_{loc}$ in the similar fashion. However, if no such $\mathbf{q'}_{loc}$ is attained, the planner

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

randomly chooses a configuration $\mathbf{q}_{back}$ in the subset of t formed by random motions, and backtracks to $\mathbf{q}_{back}$.

```
procedure RPP;
begin
   t <- BEST-FIRST-PATH (q_init ); q_loc <- LAST(t);
   while q_loc  != q_goal  do
      begin
        ESCAPE <- false;
        for i = 1 to K until ESCAPE do
              begin
                 t <- RANDOM-TIME;
                 t <- RANDOM-PATH (q_loc , t_0);
                 q_rand  <- LAST (t_i);
                  t_i <- PRODUCT (t_i , BEST-FIRST-PATH (q_rand));
                 q'_loc <- LAST (t_i);
                 if  U(q'_loc ) < U(q_loc ) then
                    begin
                       ESCAPE <- true;
                       t  <- PRODUCT (t , t_i);
                    end;
              end;
        if  ¬ ESCAPE  then
              begin
                   t  <- BACKTRACK (t, t_1, ..., t_K);
                   q_back  <- LAST (t);
                   t <- PRODUCT (t , BEST-FIRST-PATH (q_back));
              end;
```

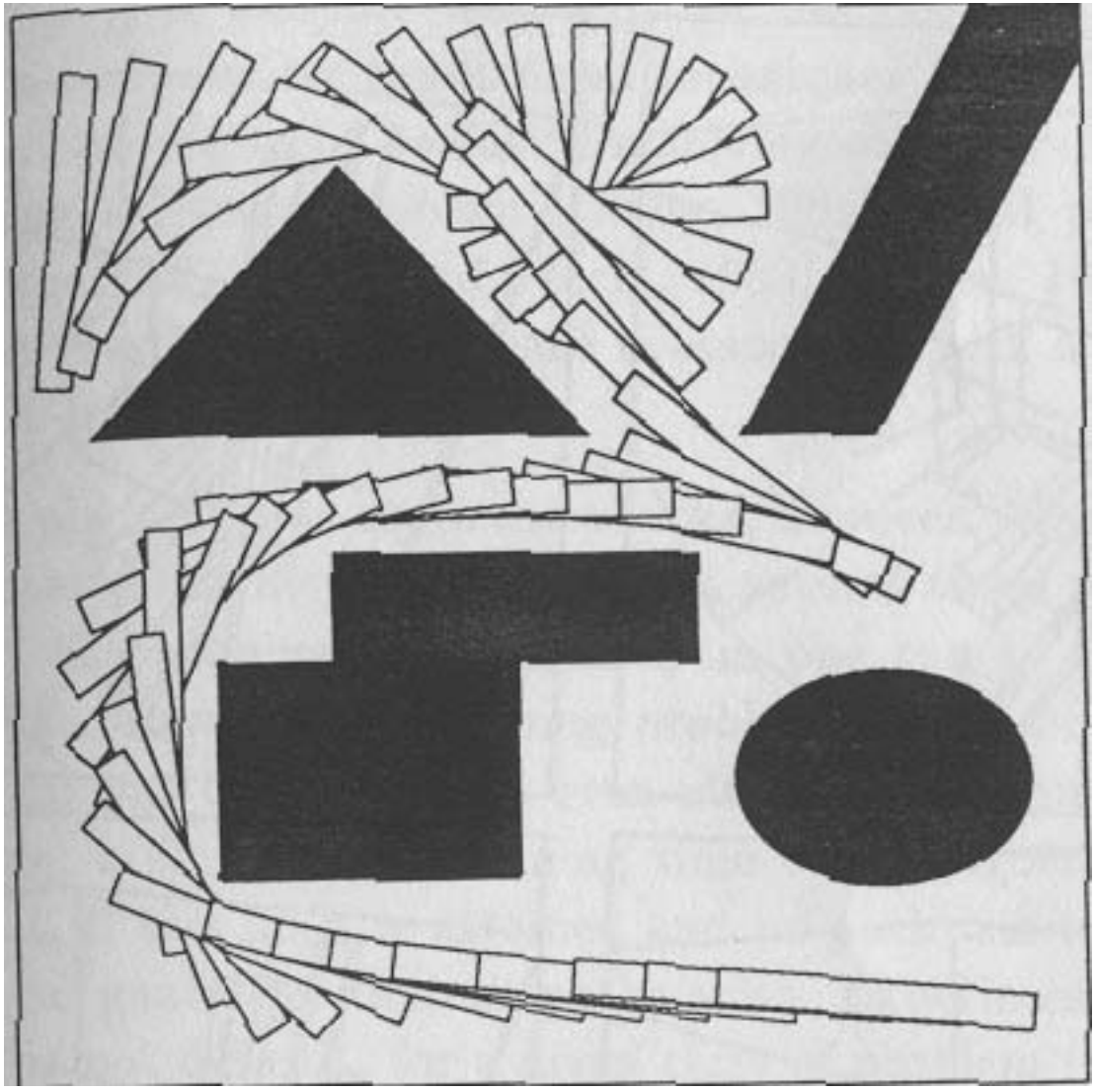
open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
        q_loc  <- LAST (t);
      end;
end;
```

Path Smoothing

Iteratively replace subpaths of decreasing lengths with straight line segments in the space $\mathbf{R}_m$ containing the grid GC. The segments are inserted in the top-down order of the length going down to the grid resolution. Collision checking has to be done at every insertion. More sofisticated variational calculus techniques may be applied to optimize more involved objective criteria.

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com



References

[1]. Robot motion planning , Jean Claude Latombe, Kluwer Academic publishers.

[2]. Barraquand, J. and Latombe, J.C., "A Monte-Carlo Algorithm for Path Planning with Many degrees of freedom", *Proceedings of the IEEE International Conference of Robotics and Automation*, Cincinnati, OH, 1712-1717

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Submitted by :

Vidit Jain (98395)

(@iitk or @yahoo)

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com