Discrete Sampling-Based Planning

Dapeng Zhao (Eagle) dapengz
Rockey Hester (Quint) rockeyh
Viraj Parimi vparimi

December 20, 2020

1 Abstract

We introduce dRRT, a discrete sampling-based planner that uses Bresenham's line algorithm and techniques from search-based planning to achieve anytime qualities and reuse computation. Results on planning for a 5-DOF robotic arm demonstrate faster performance than traditional RRT.

2 Introduction

For the project, we devised a discrete sampling-based planner called dRRT. Conventional sampling-based planners like RRT execute in the continuous configuration space (C-space). The problem with such approaches is that since their underlying sample space is so large, one needs to run them for multiple iterations to generate a plan, even in trivial cases. Variants of RRT like RRT-Connect and RRT* have been proposed to enhance convergence time and solution quality, respectively.

As an alternative, we propose a discrete version of RRT by leveraging ideas from search-based planning algorithms. Intuitively, using this approach, one would be able to generate the tree more efficiently. Further, one can even improve the collision checker's performance by utilizing the hierarchical decomposition of the problem.

3 Approach

3.1 Planning Representation

In order to represent the problem, we utilize the following planning representation,

$$\begin{aligned} M^R &= < \theta > \\ M^W &= < \texttt{obstacle/free space} > \\ S^R_{current} &= < \theta_{current} > \\ S^W_{current} &= < \texttt{constant} > \\ C &= < \texttt{Euclidean Distance} > \\ G &= < \theta_{goal} > \end{aligned}$$

where M^R represents the model of the robot, M^W represents the model of the world, $S^R_{current}$ represents the current model of the robot, $S^W_{current}$ represents the current model of the world, C represents the cost function, and G represents the goal. For experiments, we utilize the setup provided in HW2 and consequently implement the planner on a 5-DOF arm robot. This setup implies that θ is a 5-dimensional variable of joint angles.

3.2 Algorithm

Algorithm 1 presents the pseudo-code of our proposed algorithm. The algorithm's input is the q_{init} and q_{goal} configurations of the robot. Since these configurations lie in the continuous space, we first utilize the `LocateOnGrid` function to map them to particular points on the grid representation of the C-space. To generate this grid, we start with a resolution λ of 2 and keep increasing this in multiples of 2 until we can map q_{init} and q_{goal} to grid points. Once mapped, we add the grid point v_{init} to the Tree $\mathcal{T}$.

We then sample a random configuration q_{rand} from the grid C-space by utilizing the `Sample` function. To mitigate situations where it's hard to find the goal, we utilize goal-bias sampling. This function's key idea is that we maintain a set $\mathbb{Q}$ of *not-to-draw points*. This set consists of grid points that have been either sampled or added to the Tree $\mathcal{T}$ and is analogous to the *closed list* in search-based planning. Once we get q_{rand} we use the `Extend` function where like the regular `RRT` we detect the nearest grid point v_{near} that is part of $\mathcal{T}$ from q_{rand}. To identify which grid points would allow to reach q_{rand} we exploit the `Bresenham` function. For each of the points v_{new} returned by it, we check whether the line connecting them is valid and is collision-free. If it was then, we add v_{new} to both $\mathcal{T}$ and $\mathbb{Q}$.

When all the possible grid points at a given resolution λ get exhausted either because they were added to $\mathcal{T}$ or because they were in collision, we increase the resolution of the underlying grid by using the `Inflate` function. We carefully update the nodes in $\mathcal{T}$ and $\mathbb{Q}$ except v_{init} and update their corresponding parents within this function. This technique allows us to skip the collision checking in finer resolutions for already existing nodes, thereby saving a lot of time even though the grid size is huge. This idea was also the reason as to why we chose to connect grid points greedily via `Bresenham` function because connecting two nodes without the intermediate points would violate this re-usability. Finally, when the algorithm can extend to q_{goal}, we terminate and publish the path.

4 Experimental Analysis

4.1 Setup

There are ten pairs of start-goal, and each pair is evaluated **5 times** for each algorithm, instead of only running once. If at one test, the planner couldn't find any viable path, the test is deemed as failed, and the number of this test is not considered for evaluation. Since the primary point of comparison is with `RRT`, we chose random start and goal configurations that favored `RRT` over the other algorithms. This way, we ensured that we could have a fair comparison between `RRT` and `dRRT`. A brief video showcasing our experiments can be found at `https://youtu.be/Pwc9E1cjk7A`. Our complete implementation can be found at `https://bitbucket.org/eaglez1111/16782_final_proj/`

4.2 Metrics Definition

The metrics used for evaluation:

- Time(s): How long the planner takes to return a feasible plan
- Succ.(%): Success rate
- #Spl.: The number of nodes in the tree/graph
- Cost: The euclidean length of the path in C-space
- Leng.: The number of configurations in the returned plan, the length of the plan

Algorithm 1: Discrete RRT

Function dRRT(q_{init}, q_{goal}):
  LocateOnGrid($[q_{init}, q_{goal}]$)
  $\mathcal{T}$.add($\mathbf{v}_{init} = \{q_{init}, \oslash\}$)
  while $q_{goal} \notin \mathcal{T}$ **do**
    $q_{rand} \leftarrow$ Sample(λ)
    if q_{rand} **then**
      Extend($\mathcal{T}, q_{rand}$)
    else
      Inflate(λ)
Function LocateOnGrid($\mathbb{Q}$):
  $\lambda \leftarrow 2$
  while $\neg$Discretize($\mathbb{Q}, \lambda$) **do**
    $\lambda \leftarrow \lambda * 2$
  return λ
Function Extend($\mathcal{T}, q_{rand}$):
  $\mathbf{v}_{near} \leftarrow$ NearestNeighbor($q, \mathcal{T}$)
  $\mathbb{Q}_{line} \leftarrow$ Bresenham($\mathbf{v}_{near}.q, q_{rand}$)
  for $q_{new} \in \mathbb{Q}_{line}$ **do**
    if IsValidLine($\mathbf{v}_{near}.q, q_{new}$) **then**
      $\mathbf{v}_{new}.q \leftarrow q_{new}$
      $\mathbf{v}_{new}.parent \leftarrow \mathbf{v}_{near}$
      $\mathcal{T}$.add($\mathbf{v}_{new}$)
      $\overline{\mathbb{Q}}.add(\mathbf{v}_{new}.q)$
      $\mathbf{v}_{near} \leftarrow \mathbf{v}_{new}$
    else
      return *NotReached*
  return *Reached*
Function Sample(λ):
  if *goal bias sampling* **then**
    return q_{goal}
  else
    if $\|\overline{\mathbb{Q}}\| == \lambda^D$ **then**
      return $\oslash$;
    $\aleph_\lambda \leftarrow$ all points at current resolution λ
    $q_{rand} \leftarrow (\mathbb{Q}_{pool} = \aleph_\lambda - \overline{\mathbb{Q}})$[RandomInt()]
    $\overline{\mathbb{Q}}.add(q_{rand})$
    return
Function Inflate(λ):
  $\lambda \leftarrow \lambda * 2$
  for $\mathbf{v}_i \in \mathcal{T} \setminus \{v_{init}\}$ **do**
    $\mathbf{v}_{new}.q \leftarrow (\mathbf{v}_i.q + \mathbf{v}_i.parent.q)/2$
    $\mathbf{v}_{new}.parent \leftarrow \mathbf{v}_i.parent$
    $\mathbf{v}_i.parent \leftarrow \mathbf{v}_{new}$
    $\mathcal{T}'$.add($\mathbf{v}_{new}$)
    $\overline{\mathbb{Q}}.add(\mathbf{v}_{new}.q)$
  $\mathcal{T}$.append($\mathcal{T}'$)

4.3 Performance (mean)

Average values of the metrics are shown here:

Table 1: Average values of the metrics

Algo	Time(s)	Succ.(%)	#Spl.	Cost	Leng.
RRT	0.076	100.0	255.8	7.95	12.50
dRRT	0.027	100.0	3.6	10.58	4.80
RRT-Connect	0.008	100.0	343.7	14.36	23.83
RRT-Star	0.018	100.0	242.9	8.03	12.90
PRM	0.770	83.3	3002.0	4.00	2.83

4.4 Performance (variance)

For each state-goal pair and each algorithm, the variance of the 5 trials is calculated; variances are then averaged for each algorithm to be reported here:

Table 2: Variance values of the metrics

Algo	Time	Succ.	#Spl.(10^5)	Cost	Leng.
RRT	0.141	0.0	3.7143	16.33	52.92
dRRT	0.004	0.0	0.0001	0.09	3.47
RRT-Connect	0.000	0.0	0.2697	9.58	24.14
RRT-Star	0.000	0.0	0.3127	7.94	33.36
PRM	0.003	0.1	0.0000	0.73	0.13

4.5 Discussion

- `dRRT` performed faster than `RRT` on average, which shows the performance boost discrete sampling provided mainly due to reusing of computation for collision checking. However, `RRT-Connect` and `RRT*` were better performing algorithms in terms of time. In terms of success rate, all the algorithms except `PRM` performed flawlessly. This situation happens due to the biased configurations which we took, as mentioned previously.

- The number of nodes within the tree is minimal for `dRRT` compared to others, which happens since we are working in the discrete C-space. This allows us to skip large distances over the C-space, thereby allowing us to connect far-off configurations. This inference is further corroborated by the number of nodes in the final path. `dRRT` and `PRM` returned smaller paths compared to other algorithms.

- The only metric where `dRRT` is beaten by `RRT` is the path cost. This finding was expected due to the huge skips we make over the C-space. One potential improvement of `dRRT` would be to optimize the path to return lower-cost solutions.

- `dRRT` algorithm provides resolution-based completeness guarantee where if the solution was to exist in a particular resolution, the algorithm is bound to find it. This is true because before we move to a finer resolution, we exhaust all the possible grid points in the current resolution. By doing that, we ensure that if there were a potential solution, `dRRT` would never miss it.

- The hierarchical decomposition routine allows us to propose a quick solution very fast at higher resolutions. This anytime solution can be further improved depending on how much time is available by moving down the hierarchy and working with finer resolutions.

- Low variance on performance metrics is essential. `dRRT` is more stable and predictable, but we believe that this could be because of very few nodes in the tree. For metrics like path cost and runtime, being consistent helps algorithm designers and programmers better manage the system.

5 Conclusion

We developed and evaluated `dRRT`, a new algorithm for discrete sampling-based planning that sacrifices initial solution quality in order to outperform `RRT` in time and space complexity. Future work entails development of discrete versions of other sampling-based planners such as `RRT*`, `RRT-Connect`, and `PRM`, as well as developing an efficient post-processing tool to smooth `dRRT` solutions.