Safe Interval Randomized Path Planning For Manipulators

Nuraddin Kerimov[1, 2], **Aleksandr Onegin**[2], **Konstantin Yakovlev**[1, 3]
[1]FRC CSC RAS
[2]MIPT
[3]AIRI
kerimov.nm@phystech.edu

Abstract

Planning safe paths in 3D workspace for high DoF robotic systems, such as manipulators, is a challenging problem, especially when the environment is populated with the dynamic obstacles that need to be avoided. In this case the time dimension should be taken into account that further increases the complexity of planning. To mitigate this issue we suggest to combine safe-interval path planning (a prominent technique in heuristic search) with the randomized planning, specifically, with the bidirectional rapidly-exploring random trees (RRT-Connect) – a fast and efficient algorithm for high-dimensional planning. Leveraging a dedicated technique of fast computation of the safe intervals, we end up with an efficient planner dubbed SI-RRT. We compare it with the state of the art and show that SI-RRT consistently outperforms the competitors both in runtime and solution cost.

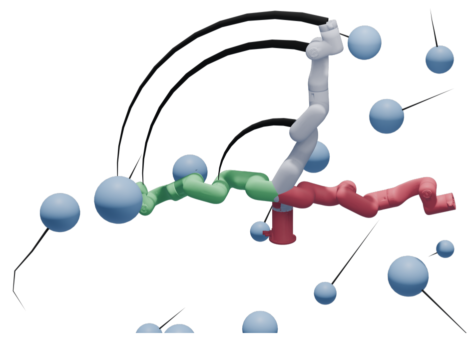

Figure 1: A problem we are interested in – path planning for a manipulator in the presence of moving obstacles whose trajectories are known (accurately predicted).

Code —
https://github.com/PathPlanning/ManipulationPlanning-SI-RRT

Introduction

Robotic manipulators are the high degrees-of-freedom (DoF) robotic systems that are widely used in industrial applications, where their movements can be precomputed. Meanwhile, in numerous other settings their paths should be planned adaptively taking the current state of the environment into account – think, for example, of a household mobile robot that is equipped with a manipulator and uses it for various pick'n'place tasks.

Generally, sampling-based planning algorithms, such as RRT (LaValle 1998), PRM (Kavraki et al. 1996) and their numerous modifications, are widely used for manipulation path planning as these methods (in contrast to the search-based ones rooted in A* (Hart, Nilsson, and Raphael 1968) algorithm) can better handle the high dimensionality of the search space.

In this work we are specifically interested in path planning for high DoF manipulators in dynamic environments, i.e. the ones there the moving obstacles are present – see Fig. 1. If the obstacles' movements are unknown, reactive approaches based on fast re-planning can be used to solve the problem, such as RRT^X (Otte and Frazzoli 2016) or DRGBT (Covic, Lacevic, and Osmankovic 2021). Efficient search based re-planning methods are also known, with D*Lite (Koenig and Likhachev 2002) being one of the most widely used (especially in mobile robotics).

In this work, however, we assume that the trajectories of the moving obstacles are known. E.g. they are accurately predicted via an external motion prediction system or these obstacles are the other robots that execute the known trajectories. In this case it is beneficial to take the information on the obstacles' movements into account while planning. To this end, a planner should reason about the time dimension. An example how this reasoning can be applied to sampling-based planning is presented in (Sintov and Shapiro 2014), where the arrival time is fixed beforehand and time is added to the nodes in the search tree such that each node denotes a specific state in a specific time. A more recent and advanced method is ST-RRT* (Grothe et al. 2022). It is a bidirectional search algorithm that does not need to specify the arrival time but rather updates it on-the-fly while searching. Currently, ST-RRT* can be deemed as state of the art in sampling based planning with time dimension.

In this short paper we wish to advance the state of the art by incorporating the idea of safe interval path planning (SIPP) into the (bidirectional) sampling-based search framework. SIPP was originally introduced in (Phillips and Likhachev 2011) as a search-based planner. However, the main idea of SIPP, i.e. reasoning not over the distinct time moments but rather over the time intervals, is applicable to sampling-based planning as well. Our planner that utilizes this idea, SI-RRT, is shown to notably outperform the competitors in both runtime and solution quality. It can successfully solve the instances with a very high number of moving obstacles where the competitors fail.

We note that we are not the first to explore the combination of safe interval path planning and sampling-based planning. E.g. in (Li and Shah 2019) a combination of SIPP and PRM was proposed for manipulation planning and in (Sim, Kim, and Nam 2024) a combination of SIPP and RRT* was presented for low-dimensional path planning, i.e. to 2D pathfinding. Still, to the best of our knowledge, we are the first to combine SIPP and RRT for high DoF planning, utilize bidirectional search and introduce a dedicated technique to efficiently estimate safe intervals.

Problem Statement

Let $\mathcal{W} \subset \mathbb{R}^3$ be the workspace of a manipulator consisting of n joints. Its configuration space, $\mathcal{C}$, is composed of tuples $\mathbf{q} = \{q_1, ..., q_n\}$, where q_i is the rotation angle of the i-th joint with respect to the previous one. $T = [0, t_{\text{max}}]$ is time, where t_{max} is the maximum time allowed for the operation.

Besides manipulator K obstacles move through the environment and their trajectories are known. Let $\mathcal{O}_i(t)$ denote the subset of $\mathcal{W}$ occupied by the i-th obstacle at time t; and $\mathcal{O}(t) = \bigcup_{i=1}^{K} \mathcal{O}_i(t)$.

A (timed) path or a trajectory of a manipulator is a mapping $\pi(t) : [0, t_{arrival}] \rightarrow \mathcal{C}$. A path is collision free if $\forall t \in [0, t_{arrival}] : \mathcal{R}(\pi(t)) \cap \mathcal{O}(t) = \emptyset$, where $\mathcal{R}(\pi(t))$ denotes a subset of $\mathcal{W}$ occupied by the manipulator time t when following the path π.

The problem is as follows. Given the start and the goal configurations, $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$, find $t_{arrival} \leq t_{max}$ and a collision-free path, $\pi(t)$, for a manipulator, s.t. $\pi(0) = \mathbf{q}_{start}$ and $\pi(t_{arrival}) = \mathbf{q}_{goal}$.

Method

The idea of our method, SI-RRT, is to combine RRT-Connect – a single-query sampling-based planner utilizing bidirectional search, and SIPP – a search-based space-time planner that reasons over the time intervals instead of distinct time moments. We assume that manipulator can either stay put or move with the constant velocity (i.e. inertial effects are neglected) and that an auxiliary collision checking routine is available that given the configuration of the robot and positions of the obstacles checks whether the latter is collision free.

SI-RRT grows the two search trees, one rooted in the start configuration, $\mathcal{T}_{start}$, and the other one rooted in the goal configuration, $\mathcal{T}_{goal}$, until they meet. A node of a tree is identified by a tuple $n = (\mathbf{q}, si = [t_l, t_u])$, where si is a *safe interval* – an interval comprised of the time moments when the robot configured at $\mathbf{q}$ does not collide with the obstacles. We will elaborate on how to compute si later on.

Each node n is characterized by the arrival time $time(n) \in si$ and its predecessor – $parent(n)$. For the

Algorithm 1: SI-RRT Planner

Input: $\mathbf{q}_{start}, \mathbf{q}_{goal}, t_{max}, \Delta_{planner}, \Delta_{parent}$

$n_{start} \leftarrow (\mathbf{q}_{start}, si_{first}); time(n_{start}) \leftarrow 0;$
$\mathbf{n}_{goal} \leftarrow \{(\mathbf{q}_{goal}, si_{first}), ..., (\mathbf{q}_{goal}, si_{last})\};$
$\forall n \in \mathbf{n}_{goal} :\ time(n) \leftarrow$ *upperbound of si*;
$\mathcal{T}_{start} \leftarrow \{n_{start}\}; \mathcal{T}_{goal} \leftarrow \{n_{goal}\};$
$\mathcal{T}_{current} \leftarrow \mathcal{T}_{start}; \mathcal{T}_{other} \leftarrow \mathcal{T}_{goal};$
while *goal not reached or have time to plan* **do**
  $\mathbf{q}_{sampled} \leftarrow$ `sampleCFG()`;
  $\mathbf{q}_{new} \leftarrow$ `extend`$(\mathbf{q}_{sampled}, \mathcal{T}_{current})$;
  if $\mathbf{q}_{new} = \emptyset$ **then**
    `swap`$(\mathcal{T}_{current}, \mathcal{T}_{other})$;
    continue
  $\{si\} \leftarrow$ `getSafeIntervals`$(\mathbf{q}_{new})$;
  $\{n_{new}\} \leftarrow$ `setParent`$(\mathcal{T}_{current}, \mathbf{q}_{new}, \{si\})$;
  if $\{n_{new}\} = \emptyset$ **then**
    `swap`$(\mathcal{T}_{current}, \mathcal{T}_{other})$;
    continue
  $\mathcal{T}_{current}.Add(\{n_{new}\})$;
  $\{n_{join}\} \leftarrow$ `connect`$(\mathcal{T}_{other}, \mathbf{q}_{new}, \{n_{new}\})$;
  if $\{n_{join}\} \neq \emptyset$ **then**
    `uniteTrees`$(\mathcal{T}_{start}, \mathcal{T}_{goal}, \{n_{join}\})$;
    return success;
  `swap`$(\mathcal{T}_{current}, \mathcal{T}_{other})$;
return *failure*;

nodes residing in the start tree, $time(n)$ is the earliest possible time a robot can reach n from $parent(n)$. Ideally, we want to perform $parent(n) \rightarrow n$ movement immediately after arriving $parent(n)$ to make final arrival time as low as possible. However, committing to such move immediately may result in collision. To this end we utilize the so-called 'wait-and-go' actions introduced in the original SIPP paper. That is, we wait at $parent(n)$ for the minimum possible time so that the moving action becomes safe and then perform it. For the goal tree, $\mathcal{T}_{goal}$, the reversed reasoning is applied. I.e. we want to arrive as late as possible to the successor to increase reachability to another nodes and $time(n)$ is computed accordingly. (it is the latest possible arrival time).

We will now explain how SI-RRT grows its trees building upon the pseudocode presented in Alg. 1. We initialise $\mathcal{T}_{start}$ with the node comprised of $\mathbf{q}_{start}$ and the first safe interval of this configuration (we assume that it starts with 0). The arrival time is set to 0. $\mathcal{T}_{goal}$ is initialised by the nodes comprised of $\mathbf{q}_{goal}$ and all safe intervals of this configuration. Arrival times are set as the upper bounds of the respective safe intervals (with the upper bound of si_{last} being t_{max}). Then we initialise $\mathcal{T}_{current}$ as $\mathcal{T}_{start}$ and $\mathcal{T}_{other}$ as $\mathcal{T}_{goal}$.

At each iteration of the main loop we randomly sample a new configuration $\mathbf{q}_{sampled}$ with the `sampleCFG` function. Then, in `extend`, we first compute the nearest configuration to the sampled configuration in the current tree, $\mathbf{q}_{\text{near}} \in \mathcal{T}_{\text{current}}$, using L_2 distance. Subsequently, a new configuration $\mathbf{q}_{\text{new}}$ is computed by advancing from $\mathbf{q}_{\text{near}}$ in the direction of $\mathbf{q}_{\text{sampled}}$ by a distance limited to the maximum step size, Δ_{planner}, which is an input parameter. If the configuration is not valid, i.e. robot is in collision with the static obstacles, we return $\emptyset$, swap the trees and skip to the next iteration. Otherwise, we proceed to computing the safe intervals of $\mathbf{q}_{\text{new}}$.

Computing the safe intervals relies on the auxiliary collision detection routine, that takes the robot's configuration (and shape) and the positions (and shapes) of the obstacles and outputs a boolean indicating whether the collision exists or no. This routine should be sequentially executed at consecutive time steps to detect which ones result in collision and which ones do not. The latter time moments will form the safe intervals. In practice, the frequency of the collision checks should be high enough not to miss the collisions. E.g. in out tests we set this frequency to 30 times per second, while t_{max} was set to 20s. This infers that to compute the safe intervals for a configuration 600 collision checks are needed, which becomes computationally expensive.

To mitigate this issue, we rely on the specifics of the collision checking. Typically, for manipulation planning collision detection is composed of two stages: the broad phase, when possible colliding obstacles are identified, and the narrow phase, when the detailed collision checks are carried out and the answer is provided. To speed up the process we create a scene on which we put all the obstacles at the positions corresponding to all the time steps and we keep the information on which obstacle comes from which time step. Then we perform a single broad phase collision checking that identifies the obstacles (annotated with the time steps) that need to be passed further to the narrow collision detection. Thus, we safe time not invoking broad collision detection sequentially for each obstacle.

After the safe intervals are computed, in `setParent` we try to connect each safe interval, si, with one of the nearest nodes lying no further away than Δ_{parent} from $\mathbf{q}_{new}$. As detailed before, for the $\mathcal{T}_{start}$ we try to arrive at si as early as possible, and for the $\mathcal{T}_{goal}$ – as late as possible. If we succeed and find such a collision-free path from parent to $\mathbf{q}_{new}$ we add the new node $n_{new} = (\mathbf{q}_{new}, si)$ to a set of new nodes $\{n_{new}\}$ that are added further to the tree. In order to find a parent, Δ_{parent} must be greater than Δ_{planner}. Large Δ_{parent} slows down `setParent`, but may lead to better arrival times.

If new nodes are successfully created, function `connect` is called in which we try to connect $\mathcal{T}_{other}$ with $\mathcal{T}_{current}$ by greedily extending the latter towards $\mathbf{q}_{new}$. That is, we iteratively append new nodes to $\mathcal{T}_{other}$ until we either reach $\mathbf{q}_{new}$ or we cannot extend further (due to collisions with the obstacles). In the former case, we have two branches in $\mathcal{T}_{current}$ and $\mathcal{T}_{other}$ that meet at $\mathbf{q}_{new}$ at a certain safe interval. We then invoke `uniteTrees` function that concatenates these branches and reverses the edges of the branch that belongs to $\mathcal{T}_{goal}$ so all actions go forward in time. These actions constitute the sought path.

Please note that the found path may contain a prolonged wait action in $\mathbf{q}_{new}$ as the forward branch has been built with the intention of reaching it as early as possible while the backward branch – as late as possible. We can appropriately trim this wait action (i.e. decrease the wait to the minimal possible duration s.t. the chain of actions outgoing from $\mathbf{q}_{new}$ is still consistent with the respective safe intervals) – the illustration of this process is provided in Fig. 2. In our experiments, we use this trimming.

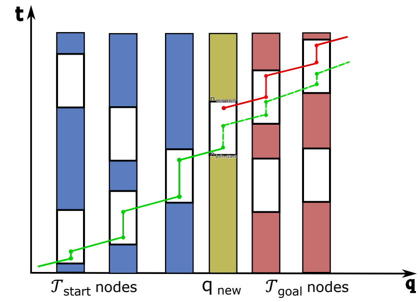

Figure 2: Trimming the wait action in a constructed path.

Experiments

We evaluate SI-RRT[1] and compare it with state-of-the-art competitors in 3D path planning for 6 DoF manipulator, Ufactory xArm 6, modelled as a chain of 3D capsules for collision checking. Its maximal length is approx. 0.7m.

For the evaluation, we create a wide range of path planning instances that involve different number of moving obstacles, from 1 to 300. Each moving obstacle is modelled as a sphere of radius r that is randomly sampled from the interval $[0.05, 0.10]$m. It moves with the constant velocity randomly sampled from the interval $[0, 1]$m/s, changing the movement direction sporadically. When generating each problem instance, we ensure that the obstacles do not hit the initial and the target configuration of the manipulator. An example of a problem instance is shown in Fig. 1. The animated visualization is available in the supplementary material[2].

For each number of obstacles we generate 50 different problem instances. The complexity of the instances increases with the number of obstacles. That is, we keep adding the obstacles to the previously generated ones with the increment of 20 (up to 300 obstacles per instance).

The time range in all tests, t_{max}, is set to 20s. Each test is repeated 10 times. We compare our planner with ST-RRT* (Grothe et al. 2022), a state-of-the-art solver that takes the trajectories of the obstacles into account and plans in space-time (like our planner); and with DRGBT (Covic, Lacevic, and Osmankovic 2021) a modern and fast replanning solver. We take the author's implementation of ST-RRT*[3] and DRGBT[4]. Both of these planners require setting several hyperparameters that influence their performance significantly. To this end, we conduct a grid search to identify the best values. For SI-RRT we use following hyperparameters: $\Delta_{planner} = 1$ radians, $\Delta_{parent} = 3$ radians.

[1] github.com/PathPlanning/ManipulationPlanning-SI-RRT
[2] youtu.be/inTmRr0GXL8
[3] github.com/ompl/ompl
[4] github.com/robotics-ETF/RPMPLv2

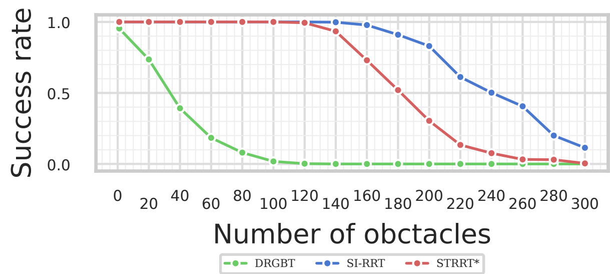

Figure 3: Success rate of the evaluated planners.

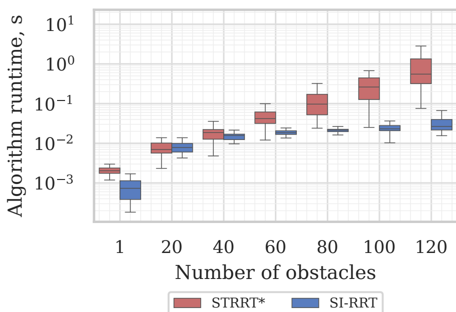

Figure 4: Runtime of SI-RRT and ST-RRT* on the instances that were successfully solved by both of them.

The results of the experiments are shown in Figures 3, 4, 5. Specifically, Figure 3 shows the success rate (SR), i.e. the percentage of tasks that were successfully solved for the increasing number of obstacles. To count SR for DRGBT we consider a test as failed if the planner was not able to reach the goal until $t = t_{max}$ or the robot got in collision before. For ST-RRT* and SI-RRT we consider the test as failed if no solution was constructed under a time cap of 20s. Clearly, our planner outperforms the competitors. E.g. for 200 obstacles the SR of SI-RRT is 77% while the one of ST-RRT* (the closest competitor) is 42%.

Figure 4 shows the runtimes of ST-RRT* and SI-RRT on the instances encountering up to 120 obstacles, i.e. the instances for which the SR of ST-RRT* and SI-RRT is 100% (we did not include DRGBT because of much lower SR). Evidently, as the number of obstacles increases, the runtime of the both solvers increases as well. Still SI-RRT scales much better. For 100-120 obstacles it is one order of magnitude faster than SI-RRT* (note that OY axis is in log-scale).

Figure 5 shows solution cost of SI-RRT and ST-RRT*, that is a time moment when, according to the constructed plan, the manipulator arrives to the goal configuration. As before, the results of DRGBT are omitted due to its low SR. Noteworthy, that ST-RRT* is an anytime planner and has a capability to enhance the cost of the first found solution if time permits. However, in our test we do not use this feature for a fairer comparison to SI-RRT which does not use the cost improvement techniques (although they can be encapsulated to the algorithm; we leave this for future work).

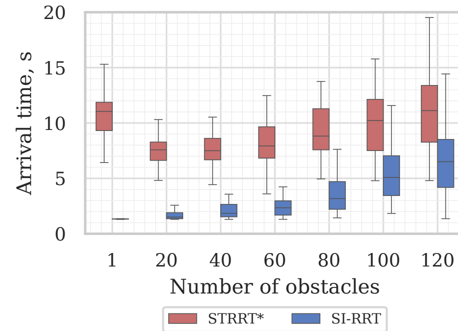

Figure 5: Solution quality (time when the manipulator arrives at the goal configuration) of ST-RRT* and SI-RRT.

This explains the poor cost of ST-RRT* for the tests with 1 obstacle. ST-RRT* grows its forward and backward trees from time moment 0 and time moment t_{max}, respectively, and when they meet, resulting motion has extremely slow speed. As the number of obstacles increases, this effect diminishes. Still the cost of SI-RRT is better compared to ST-RRT*, that can be attributed to the use of safe intervals and not sampling distinct time moments.

Overall, the results of the experiments clearly show that, first, the planners that reason over time are superior over the re-active ones (i.e. ST-RRT and SI-RRT* outperform DRGBT) and, second, the suggested solver, ST-RRT, notably surpass ST-RRT* – current state of the art in high-dimensional path planning in space-time. We believe that the main reason for this is the usage of safe intervals. This confirms that this concept, initially introduced in the context of search-based planning, is beneficial for sampling-based planning as well.

Conclusion

In this work we have suggested a novel planner, SI-RRT, tailored to finding safe paths in the predictably changing environments for multi-degree of freedom robotic systems, such as manipulators. SI-RRT leverages a combination of the well-established techniques such as sampling-based planning, bidirectional search and safe interval path planning. A comprehensive empirical evaluation of SI-RRT comparing it with state of the art was conducted, and the planner was shown to notably outperform the competitors.

The prominent directions for future research include: investigating the theoretical properties of SI-RRT, developing anytime variants of SI-RRT that may gradually improve the solution cost (and converge to optimal solutions in the probabilistic sense), utilizing SI-RRT in multi-agent manipulation planning.

Acknowledgments

This work was supported by the Ministry of Economic Development of the Russian Federation (code 25-139-66879-1-0003).

References

Covic, N.; Lacevic, B.; and Osmankovic, D. 2021. Path Planning for Robotic Manipulators in Dynamic Environments Using Distance Information. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4708–4713. IEEE.

Grothe, F.; Hartmann, V. N.; Orthey, A.; and Toussaint, M. 2022. St-rrt*: Asymptotically-optimal bidirectional motion planning through space-time. In *2022 International Conference on Robotics and Automation (ICRA)*, 3314–3320. IEEE.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4): 566–580.

Koenig, S.; and Likhachev, M. 2002. D* lite. In *Eighteenth national conference on Artificial intelligence*, 476–483.

LaValle, S. 1998. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*.

Li, S.; and Shah, J. A. 2019. Safe and efficient high dimensional motion planning in space-time with time parameterized prediction. In *2019 international conference on robotics and automation (ICRA)*, 5012–5018. IEEE.

Otte, M.; and Frazzoli, E. 2016. RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *The International Journal of Robotics Research*, 35(7): 797–822.

Phillips, M.; and Likhachev, M. 2011. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE international conference on robotics and automation*, 5628–5635. IEEE.

Sim, J.; Kim, J.; and Nam, C. 2024. Safe Interval RRT* for Scalable Multi-Robot Path Planning in Continuous Space. *arXiv preprint arXiv:2404.01752*.

Sintov, A.; and Shapiro, A. 2014. Time-based RRT algorithm for rendezvous planning of two dynamic systems. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 6745–6750. IEEE.