RRT* With a Shrinking Sample Space - Group 3

Danish Ansari (5094755) Jasper van Brakel (5323215) Tyler Olson (5946182) Gijs Zijderveld (5306728)

***Abstract*— Testing an adaptation of the RRT* algorithm with a sampler that continuously shrinks the sample space as a global planner using a holonomic high-five robot in a static environment.**

I. INTRODUCTION

In this project, we implement a novel motion planner for a simulated holonomic mobile manipulator in the `gym_envs_urdf` environment [1]. Our robot has been tasked with navigating through a scene and giving a human a high-five. The workspace $\mathcal{W} \in \mathbb{R}^3$ consists of randomly generated connected rooms with openings of different sizes, which the robot has to manoeuvre through to reach a differing number of successive goal poses.

The chosen robot has a complex configuration space since it has 7 degrees of freedom. Due to the level of complexity this introduces, the problem is simplified by using a holonomic point base for the robot (provided by `gym_envs_urdf`). However, this choice is not completely unrealistic since real-world robots also use this type of base for mobile manipulators, for example, the Stretch robot of Boston Dynamics [2].

As a baseline, planning will be done in the configuration space using the RRT* algorithm. Collision information is retrieved from a simulated robot using `pybullet`.

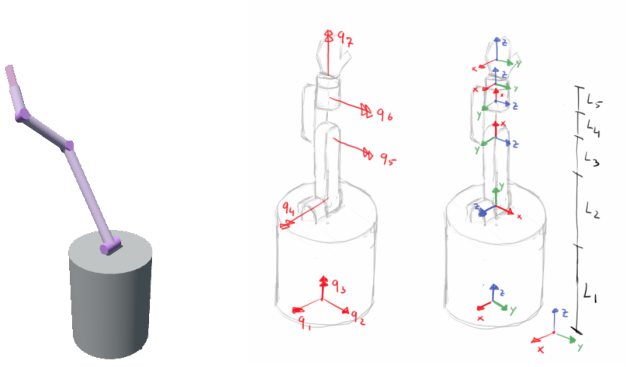
This project aims to modify the RRT* method to cut down on the time it takes to find a feasible path from a starting position to the goal point. To achieve this, we modify how new samples are taken by removing colliding obstacle volumes from the sample space whenever they are encountered.

In static environments, this modified sample space can be kept between multiple missions, so it should be faster than using a uniform sampling strategy since samples will not be taken in colliding regions which have previously been explored. This motivated us to run multiple missions in succession in a fixed environment when assessing the performance of the proposed sampling method.

Three metrics will be used to assess the performance of the proposed planning pipeline. 1) The number of iterations until a path is found. The altered sampling method tracks obstacles in the configuration space which should result in fewer iterations being needed until a path is found since fewer sampled poses will lie inside obstacles. 2) The number of rejected nodes during optimisation. The number of sampled nodes causing collisions should decrease with time since fewer samples will be taken in existing obstacles. 3) The length of the optimised path in the configuration space. The length of an optimised path is expected to be shorter when using the altered sampling method since fewer iterations will result in collision samples, which leaves more iterations for optimisation.

II. ROBOT MODEL

Figure 1 below shows the URDF model of the robot. Additionally, the degrees of freedom and joint frames are visualised.



(a) URDF Model (b) Robot DoF and Joint Frames

Fig. 1: Robot Model

The base of the robot is modelled as a planar joint which is always fixed to the floor. The position on the floor is denoted by q_1 and q_2 and the base can freely rotate around its vertical axis without limit, denoted by q_3; $SE(2)$. The arm has four joints, three of which can rotate within limits, denoted by q_4, q_5 and q_6; $\mathbb{R} \times \mathbb{R} \times \mathbb{R}$. The last joint can rotate without limit, denoted by q_7; $\mathbb{S}^1$. Thus, the configuration space $\mathcal{C}$ of the robot is isomorphic to $SE(2) \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{S}^1$. This configuration of joints allows the end effector to freely move in all six degrees of freedom of 3D space. The obstacles and environment are defined in $\mathbb{R}^3$ which makes the workspace $\mathcal{W}$ isomorphic to $\mathbb{R}^3$.

To determine the 3D positions of the joints in the workspace, forward kinematics are necessary. The kinematic equations are derived using the Denavit-Hartenberg (DH) parameters [3]. Table I provides the DH parameters for each link, which can be used to derive the homogeneous transformation matrices from the base of the robot to each other frame on the robot. The transformation between the frames connected to Link 4 could not be done in a single step using the DH parameters, which results in two sets of DH parameters for Link 4 as denoted in Table I.

With seven joints and an arm with four linkages, providing the symbolic derivations of the homogeneous transformation

TABLE I: Denavit-Hartenberg Mobile Manipulator

Link #	d	θ	r	α
1	l_1	$\pi/2$ + q_3	0	$\pi/2$
2	0	$\pi/2$ + q_4	l_2	$\pi/2$
3	0	q_5	l_3	0
4-I	0	q_6	l_4	0
4-II	0	$\pi/2$	0	$\pi/2$
5	l_5	q_7	0	0

matrices becomes quite lengthy. Instead, the method used to retrieve the transformation matrices using the DH parameters in Table I is provided. Matrix (1) is used to determine the homogeneous transform matrix between two consecutive frames.

$$ {}^{i-1}_{i}\mathrm{T} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & r_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & r_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

The homogeneous transformation matrix between the world base frame and robot base frame, as shown in (2), is derived by hand.

$$ {}^{0}_{1}\mathrm{T} = \begin{bmatrix} 1 & 0 & 0 & q_1 \\ 0 & 1 & 0 & q_2 \\ 0 & 0 & 1 & l_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2} $$

The homogeneous transformation matrix between any two frames can then be determined using the property of homogeneous transformation matrices shown in (3).

$$ {}^{0}_{3}\mathrm{T} = {}^{0}_{1}\mathrm{T}\,{}^{1}_{2}\mathrm{T}\,{}^{2}_{3}\mathrm{T} \tag{3} $$

The forward kinematics of the robot are now fully defined and can be used to retrieve the position of the robot base and the joint positions in Cartesian space using the robot configuration.

Since our goal is to validate the performance of RRT* with the altered sampling method, we make the simplifying assumption that the robot will be able to perfectly follow the provided path. Therefore, no additional controller will be implemented at this time. In future work, a controller could be implemented by deriving the Jacobian matrix of the robot using the homogeneous transformation matrices that are obtained from the DH-Table. The Jacobian matrix relates joint velocities to point velocities on the robot. This relationship can be used in combination with a PID controller to determine what velocity commands to send to the robot joints at each time step. This way the end-effector can be commanded to move to a certain position.

III. MOTION PLANNING

Because we need to alter the node sampling strategy, we have elected to use our own implementation of RRT*, however, no additional changes have been made to the RRT* algorithm.

A Euclidean distance metric is used for all of the joints with joint limits. The given mobile manipulator has two degrees of freedom with manifolds isomorphic to $\mathbb{S}^1$, so the distance metric for these two joints is defined as the shortest distance between two angles. Additionally, The goal position is sampled with a probability of 5% to ensure that the goal is part of the final optimised tree.

The main difference for the proposed sampler lies in how new collision samples are handled. Whenever a collision is detected, add a constraint $f : \mathcal{S} \mapsto \{True, False\}$ where $f(\mathcal{C}_{obs}) = True$ and $False$ otherwise. Define $\mathcal{S}_{obs} = \{s \in \mathcal{S} : f(s) is True\}$. Note that $\mathcal{S}_{obs} \cup \mathcal{S} = \mathcal{C}$. *All configurations of the robot which collide with the selected sample $s \in \mathcal{S}$ can be described by calculating the null space for the robot with a prismatic joint replacing the colliding link.* This fictitious prismatic joint is then limited to the length of the link it replaced.

The null space of this fictitious link is a manifold $M_{ns} \in \mathcal{S}$ of dimension one lower than the number of links in the chain embedded in the higher-dimensional sample space. However, to effectively reduce the size of the sample space a region of the same dimension as the space is necessary, as the probability of sampling on the manifold again is precisely 0. Such a region is a higher dimensional rejection of $s_{obs} \subset \mathcal{S}_{obs}$ and can be constructed by first creating a structure $\mathcal{P}_{ns}$ bounded by the null space for two collisions $M_{ns}(\mathcal{O}_0)$ and $M_{ns}(\mathcal{O}_1)$ through which every point on a curve connecting them $\mathcal{O}_t = g(t) \subset \mathcal{W} | t \in [0, 1]$ is also in collision. Then $\mathcal{P}_{ns}$ can be sealed by solving just the patch boundary $\delta M_{ns}(\mathcal{O}_t)$ for a collision point interpolated along the curve. With parameterised null space equations, this becomes equivalent to marching over a hypercube of dimension one higher than M_{ns}. Finally project $\mathcal{P}_{ns}$ in each remaining dimension of $\mathcal{S}$ to create s_{obs}. An example of an equivalent procedure common in 3D modelling software is the extruded cut. Suppose we construct a square on the XY plane by marching over each of its sides, then project this square through Z to create a rectangular prism, subtracting this volume from the rest of the 3D space. The implementation of the collision space removal algorithm is given in algorithm 1.

We can additionally perform the same evaluation for links in the chain which would also collide with the discovered obstacles until $\mathcal{O}_1$ and $\mathcal{O}_2$ are out of reach of the reduced chain. For now this is left out intentionally for future work.

By removing newly detected collision regions $\mathcal{S}_{obs}$ from $\mathcal{S}$, our implementation does not waste time evaluating sample points that are already known to cause collisions. Because the sampling space is shrunk with every collision, the probability of a future sample causing a collision monotonically decreases. As more samples are taken, $\mathcal{S}_{obs}$ will converge to $\mathcal{C}_{obs}$ such that new nodes which RRT* inserts into the tree will always be in free space. Furthermore, in dynamic environments if we have perfect knowledge of obstacle trajectories, $\mathcal{S}$ can be carried between time steps, making

Algorithm 1 Our proposed algorithm for updating the sample space.

function UPDATESAMPLESPACE(collisions $\mathcal{O}_1, \mathcal{O}_1 \in \mathcal{W}$, sample space $\mathcal{S}$, joint number q)
  $\mathcal{P}_{ns,q} \leftarrow M_{ns,q}(\mathcal{O}_0) \cup M_{ns,q}(\mathcal{O}_1)$
  for all $t \in [0, 1]$ **do**
    $O_t = g(t)$
    $\mathcal{P}_{ns,q} \leftarrow \mathcal{P}_{ns,q} \cup \delta M_{ns,q}(\mathcal{O}_t)$
  $s_{obs} \leftarrow \emptyset$
  for all dimensions $d > q$ of $\mathcal{S}$ **do**
    $s_{obs} \leftarrow s_{obs} \cup \mathcal{P}_{ns,q}$ projected into d
  $\mathcal{S} \leftarrow \mathcal{S} \setminus s_{obs}$
  return

it faster to generate new paths in the future.

In order to make storing the high-dimensional sample space tractable, we elected to segment the sample space into high-dimensional voxels. Even with coarse resolutions, storing occupancy information for such a space requires the use of a sparse data structure. We implemented a multi-dimensional sparse occupancy tree inspired by the sparse voxel octree described by NVIDIA researchers in [4]. As each voxel in the sampling space contains only binary information, topological nodes in the tree need only contain the total number of voxels set in each of their children. As a consequence, if either none or all the voxels in a child have been set, it and any of its descendants can be pruned from the tree to free up memory.

When collisions are detected, the sample space must be updated accordingly. For each link, first, the null space is calculated. A dense set of solutions is found by the method previously described. Note that because joints q_1 and q_2 lie on the ground plane, a continuous curve (a circle) connects a set of collisions for the base, therefore only one collision is necessary when evaluating the obstacle boundary for this link. Using an adaptive step size to ensure the boundary is closed, we can efficiently identify all the voxels that the collision boundary passes through and remove them all at once. Conservatively, we also set the neighbors of each identified voxel on the boundary which helps avoid holes caused by errors in floating point arithmetic. The interior of this region is then removed with the flood fill algorithm. By using a z-order numbering scheme for each voxel (see [5]), finding the neighbours is trivial and larger megavoxels (regions in free space of lower resolution that have not yet been split by insertion of obstacles) can be removed all at once.

Using a sparse occupancy tree also makes generating samples a linear time operation. Starting at the root node, simply traverse the tree by randomly selecting from the existing children until a leaf node has been reached, then take a uniform random sample within the space that node covers. Weighing the samples by the sum of their content (the total number of set voxels contained by this node), the probability a region in free space is selected is equal to the volume of the region. This satisfies the uniform sampling restriction of RRT* as outlined in chapter 5.2 of [6]. Thus regions which are already known to cause collisions are never sampled.

There exists a major tradeoff between computation effort and fully resolving the empty regions of sample space, so we have slightly modified this approach by only representing the first three dimensions in sample space, and taking a random sample for the latter joints. This trade-off results in faster collision region removal and maintains the property that known collisions of the base will never be resampled.

However, collisions with later links in the chain cannot be remembered. This is a good trade-off because earlier links in a robot have larger null spaces which are easier to compute, so more samples can be taken by the path planner. We attempted to find path solutions with a four-dimensional sample space (that is joints q_1 through q_4), however, the current implementation was not able to create the boundary manifolds within a desirable amount of time. Due to time constraints, this is left as future work.

IV. RESULTS

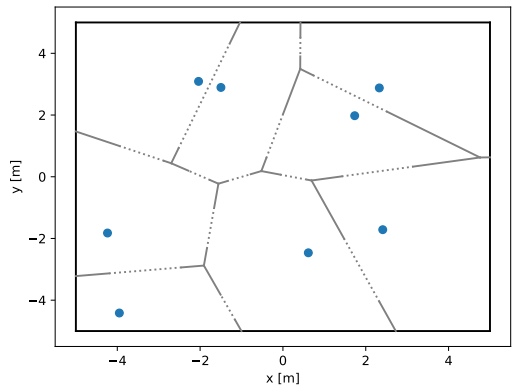
To test the implementation of the shrinking sample space, several tasks were run, first using the uniform sampler and another time using the modified sampler. The environments and start and goal positions are randomised for each task, however, stay consistent between the two samplers. The worlds are randomly generated based on a Voronoi tessellation of a set of random points, where random gates are inserted in the walls if possible. Start and goal configurations are randomly chosen based on the initial set of random points, as shown in Figure 2.

Initially, only one mission is run, i.e. one single start and goal position. Table II shows the percentage difference between the shrinking and uniform sampler. A negative percentage indicates that the shrinking sampler performs better, i.e. has a lower score.

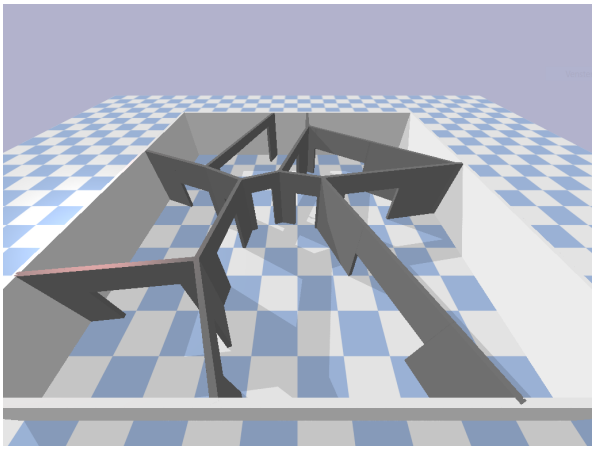
TABLE II: Single Run Statistics

Task	Initial Path at iteration % difference	Rejected Nodes % difference	Path Length % difference
1	137.65	-0.42	-0.75
2	137.65	-9.86	-0.75
3	235.29	11.05	-2.04
4	235.29	-6.70	1.89
5	79.41	7.11	-2.39
6	79.41	-6.32	-17.73
7	Uniform DNF	Uniform DNF	Uniform DNF
8	Uniform DNF	Uniform DNF	Uniform DNF
9	-90.63	-46.02	-6.48
10	-90.63	-45.60	-7.20
11	-90.63	-43.93	-7.53
12	-72.55	-17.90	-7.00
13	-24.63	-9.72	3.59
14	-68.92	-14.36	9.13
AVG	-	-15.22	-3.11

The amount of rejected nodes decreases by an average of over 15% throughout the different tasks. This means that



(a) The 2D representation of the World as generated, where the dotted lines are gates and the blue dots are the points used to generated the tessellation.



(b) The world inserted into the PyBullet simulator

Fig. 2: Generated Random world based on Voronoi tessellation

the proposed method of shrinking the sample space to stop sampling nodes in earlier encountered obstacles seems to be working. Additionally, the path length in the configuration space seems to decrease by a little over 3%. This result is not significantly different, however, it does support the initial hypothesis of decreasing the total path length. This is especially true for tasks that are run on a higher count of iterations, namely tasks 9 through 14.

On the other hand, the amount of iterations until an initial path is found does not seem to have a definite difference. Tasks 1 through 6 show that it takes more iterations, while tasks 9 through 14 see improvements in this metric. This is mainly because of environmental complexity. In simpler environments with fewer rooms (tasks 1 through 6), the uniform sampler performs better. As the environment gets more complex in tasks 9 through 14, the benefit of using the shrinking sampler becomes more apparent. For this reason, the average difference was omitted for this first column. The lower performance of the shrinking sampler for tasks 1 through 6 is most likely explained by random chance.

Tasks 7 and 8 resulted in no path being found with the uniform sampler, therefore no results were given.

As mentioned earlier, the main benefit of using the shrinking sampler is that running multiple missions in the same environment should be more efficient, since obstacle regions have already been identified in the configuration space. Table III shows the same metrics, however, averaged over multiple missions in the same environment. The difference in path length in the configuration space does not significantly change, however, the amount of rejected nodes has decreased even further. This shows that in consecutive runs, even fewer samples are taken within obstacle regions, which was the goal of the shrinking sampling method.

TABLE III: Multi-Run Statistics

Task	Initial Path at iteration % difference	Rejected Nodes % difference	Path Length % difference
1	24.18	-9.74	-2.38
2	12.67	-20.44	5.90
3	200.00	-8.64	6.35
4	-30.52	-22.61	-4.79
5	56.60	-27.28	-3.99
6	-23.42	-17.03	0.33
7	-31.54	-20.00	-2.47
8	-81.29	-37.94	-12.98
9	44.58	-15.50	6.47
10	-69.87	-21.24	-12.21
11	-40.00	-25.12	-8.43
12	Modified DNF	Modified DNF	Modified DNF
13	-40.85	-23.13	8.82
14	Modified DNF	Modified DNF	Modified DNF
15	-17.73	-19.81	0.79
AVG	-	-20.65	-1.43

Table IV shows the percentage change of the amount of rejected nodes for each run in the multi-goal tasks since this is the metric that sees the most significant difference between the uniform sampler and shrinking sampler. Especially in the more complex environments corresponding to tasks 9 through 15, the percentage change of rejected sample nodes decreases even further. This behaviour is generally consistent, but the path complexity also plays a role in these values, which explains the outliers.

TABLE IV: Rejected point per point percentile difference

Task	Point 1	Point 2	Point 3	Point 4	Point 5
1	-11.83	-7.01			
2	-19.15	-21.86			
3	20.38	39.63			
4	-23.25	-16.13	-19.67	-30.77	
5	-27.12	-25.27	-29.90	-26.73	
6	-20.87	-16.14	-13.37		
7	-24.12	-13.04	-22.00		
8	-46.98	-26.14			
9	4.03	-29.64	-23.48		
10	-9.50	-21.53	-27.14	-27.55	
11	-15.58	-30.04	-32.35	-23.49	
12	-14.04	-12.58	-8.84	-6.94	
13	-15.05	-32.82	-35.34	-5.31	-24.57
14	-5.08	-37.61	-31.68	-30.94	-14.34
15	-6.57	-24.77	-24.87	-23.81	-19.89
AVG	-14.32	-18.33	-24.42	-21.94	-19.60

V. DISCUSSION

In a theoretical sense, the employed method with the altered node sampling does not change the inherent properties of RRT*. This means that the implemented method remains anytime optimal and probabilistically complete. Given enough samples, RRT* will find a solution if there is one. Additionally, RRT* continues to optimise for new samples to find the shortest path.

In a practical sense, the implemented method performs better on the given metrics, as shown in Section IV. The results could be improved further by using the constraints for the links of the mobile manipulator to generate configuration space obstacles for degrees of freedom q_3 through q_7. Adding obstacles in these higher dimensions will allow a higher coverage of the configuration space with obstacles, which will result in even fewer nodes being sampled within obstacle regions. It is important to note that the further the link is in the chain, the smaller the obstacle region will be in the configuration space since there are constraints for each added degree of freedom involved. Simply said, there are diminishing returns to this method. Adding these DoF to the configuration space obstacles would require additional computational time.

Another important note is that the benefits of using the shrinking sampler are more apparent when the algorithm is run multiple times in the same environment, as shown in Table IV. Moreover, the shrinking sampler will no longer be valid if existing obstacles in the workspace are removed since the previous obstacle regions will not be used for future samples.

The shrinking sampling method does take significantly longer to run than a uniform sampler. For example, running RRT* with uniform sampling for 1000 iterations takes no longer than 10 seconds. Running it with the shrinking sampler, 1000 iterations take around 4 minutes. Adding constraints and configuration space obstacles for higher dimensions will require more computations, and therefore more time.

This discrepancy between the improved performance on the metrics and the decreased performance on time most likely comes down to the current implementation. The sample space-shrinking code is written in Python which leaves a lot of overhead. If the code was written in a programming language like C++ or Rust, the performance could be improved. Additionally, given the amount of time, the code leaves room to be optimised. More efficient methods could be used to define and fill in the obstacles in the configuration space.

For future work, parallelisation techniques can be explored to improve the time it takes to insert obstacle regions into the configuration space since this is currently the step that takes up most of the time.

Furthermore, the resolution of the higher dimensional spaces can be decreased to decrease memory usage and improve computational time.

A final idea would be to perform a reachability analysis on the free regions to ensure that they are not surrounded by obstacles and fill these regions otherwise.

REFERENCES

[1] M. Spahn, *Urdf-Environment*, version 1.0.0. DOI: 10.4121/19362017. [Online]. Available: https://github.com/maxspahn/gym_envs_urdf.

[2] "Stretch — boston dynamics." (Dec. 11, 2023), [Online]. Available: https://bostondynamics.com/products/stretch/.

[3] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *Journal of Applied Mechanics*, vol. 22, no. 2, pp. 215–221, Jun. 1, 1955. DOI: 10.1115/1.4011045.

[4] S. Laine and K. Samuli, "Efficient sparse voxel octrees – analysis, extensions, and implementation," NVIDIA Research, Tech. Rep., 2010.

[5] R. Raman and D. Wise, "Converting to and from dilated integers," *IEEE TRANS. COMPUT.*, vol. 99, no. 13, Jul. 5, 2007.

[6] S. M. Lavalle, *Planning Algorithms*. Cambridge university Press, 2006, ch. 5.2, pp. 195–209.