Informed RRT* algorithm on a mobile manipulator in simulation

Jakob Bichler, Christian Reinprecht, Korneel Somers, Nicolas Wim Landuyt
GitHub repository, Simulation video

***Abstract*—Mobile manipulators find extensive applications, such as serving as picking agents in logistics centers for delivery services. This paper introduces the implementation of two random sampling algorithms—RRT* and informed RRT*—enabling the robot to manipulate its arm to retrieve items from a box and navigate a 2D environment to reach a predefined goal point, all while avoiding static obstacles. We conduct a comparative analysis, highlighting the superior performance of informed RRT* over RRT* in various environments and assessing the impact of specific hyperparameters.**

I. INTRODUCTION

In Dutch households, package deliveries have become commonplace. In 2022, a substantial 298,35 million items were delivered in the Netherlands alone, encompassing not just non-food supplies like books and technology, but also groceries and food. According to Thuiswinkel, 61.990 food items were sold in 2022 over the internet and delivered to the customers [1]. Services such as Flink have emerged to cater to this demand, offering the convenience of doorstep deliveries.

Despite these advancements, a pivotal challenge remains in the logistics of storage units where efficient item retrieval is essential, so the riders can ensure a timely delivery. This is where our robotic solution steps in to address this specific bottleneck.

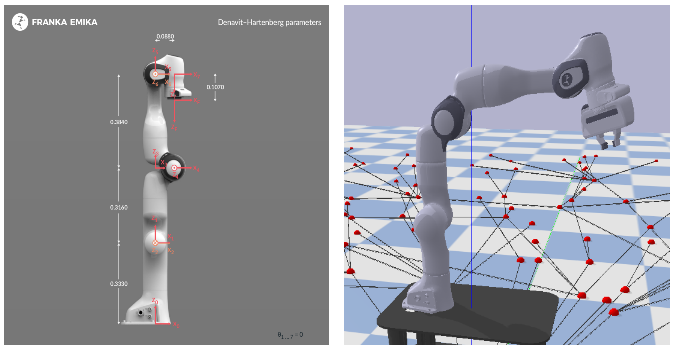
The robot's task is divided into two main steps: First, it picks up an item from a nearby box using the end effector and sets it on the mobile platform. Afterward, the mobile base moves towards a designated goal position p_{target}. Not only does this make the riders' jobs easier by allowing them to concentrate on the delivery, but it also makes the handling of the items in the warehouse safer and more efficient.

An algorithm has to be implemented for the robot to navigate the environment with static obstacles. Random sampling algorithms are suitable for high-dimensional state spaces due to their probabilistic completeness, which guarantees a solution if one exists. The Rapidly-exploring Random Tree (RRT) is notable for its speed in finding solutions in labyrinths to date, but it lacks asymptotic optimality. To address this problem, S. Karaman and E. Frazzoli developed the RRT* algorithm, which has been proven to be both asymptotically optimal and computationally efficient [2]. It improves RRT by rewiring nodes in a more efficient way as more nodes are found, thus improving the efficiency of all paths within the graph. An extension of this algorithm, which looks for an even better path in a narrowed-down configuration space, is informed RRT* [3].

In this project, we will implement both the RRT* as well as the informed RRT* algorithm. The RRT* algorithm will also be used for the mobile manipulator to find a collision-free path to grab an item. We will then explore the effect of different hyperparameters on the quality of the found path, and examine the differences between RRT* and informed RRT* in different environments. Upon that, we will further improve path quality by performing spline interpolation to smoothen out the found path.

II. ROBOT MODEL

The mobile manipulator that we are planning to use in this project is a robotic model known as the Albert Robot, available through a URDF package specifically designed for the OpenAI gymnasium environment (see Github). We implement this robot with a base containing four wheels, simulated as a non-holonomic robot, which will maneuver close to the goal position. It is equipped with the serial-link manipulator Panda developed by Franka Emika. This manipulator has seven joints, providing it with seven degrees of freedom (DoF). This enables it to avoid obstacles that are in between the robot's arm and the desired goal position.



(a) Measurements (b) Manipulator in urdf env

Fig. 1: Panda by Franka Emika

A. Equations of Motion

The mobile base and robotic arm are two kinematic parts to consider separately. For a detailed explanation of the symbols and terminology used in describing these components, please refer to table I.

For the **mobile base**, we have used the following states:

$$v^{\mathsf{T}} = \begin{pmatrix} v_x & 0 & \dot{\theta} \end{pmatrix} \tag{1}$$

The **robotic arm** can be used with the Denavit-Hartenberg (DH) Convention. This allows us to assign coordinate frames to each joint and determine the transformations between them. To know the endpoint pose in the Cartesian coordinate system, we can multiply all the transformation matrices between the joints and thus establish the forward kinematics (see equations 2, 3).

$$Z \times X = T_{(\text{between } i-1 \text{ and } i)} =$$

$$\begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_i \\ \cos\alpha_i \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i & -\sin\alpha_i d_i \\ \sin\alpha_i \sin\theta_i & \cos\theta_i \sin\alpha_i & \cos\alpha_i & \cos\alpha_i d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$T = \prod_{i=0}^{7} T_i \tag{3}$$

The Jacobian matrix is then computed using the direct kinematics derived from Craig's DH parameters. This matrix is crucial for calculating the inverse kinematics, which can be expressed as:

$$\dot{q} = J(q)^{-1}\dot{x} \tag{4}$$

In table I, the elements used in the equations above are listed.

Symbol	Description
v_x	Linear velocity in the x-direction
$\dot{\theta}$	Angular velocity
θ_i	Angle of the i-th joint in the robotic arm
a_i	Length of the i-th link in the robotic arm (Craig's DH parameter)
d_i	Offset of the i-th link along its previous joint axis in the robotic arm (Craig's DH parameter)
α_i	Twist angle between the i-th link and the previous link in the robotic arm (Craig's DH parameter)
T	Total transformation matrix of the robotic arm
J	Jacobian matrix of the robotic arm
$\dot{q}$	Joint velocity vector
$\dot{x}$	Cartesian velocity vector

TABLE I: Notation used in the equations of motion and kinematics

B. Configuration space and Workspace

The Albert robot lives in $\mathbb{R}^3$ however, the configuration and workspace need to be considered for the base and the arm separately.

1) Mobile base: Since the mobile base only moves on the ground plane in 2D, its workspace is $W_{\text{base}} = \mathbb{R}^2$. Since it can rotate around its z-axis and move forward and backward, its configuration space is $C_{\text{base}} = \mathbb{R}^2 \times \mathbb{S}^1$.

2) Mobile manipulator: The arm of the robot can move in 3D space, thus its workspace is $W_{\text{arm}} = \mathbb{R}^3$. Due to its seven joints that each have one rotational axis, the configuration space of the arm is $C_{\text{arm}} = \mathbb{S}^1 \times \mathbb{S}^1 \times \mathbb{S}^1 \times \mathbb{S}^1 \times \mathbb{S}^1 \times \mathbb{S}^1 \times \mathbb{S}^1$.

C. Virtual Environment

The robot operates in a storage facility, where it is responsible for picking an item from a box and moving it to a goal position. Therefore, it needs to autonomously navigate around the environment and avoid static obstacles, which can be shelves, boxes or other items. The robot and its surroundings are modeled based on the gym_envs_urdf repository [4], which uses Gymnasium and PyBullet to create a realistic environment and physics. We can gradually increase the environment's complexity, using obstacle configurations from the motion_planning_scenes repository [5] of varying size and shape, representing boxes, co-workers, or items.

III. MOTION PLANNING

A. Mobile Base

In the following, the workings of the informed RRT* will be explained. It is an extension of the RRT* algorithm, which is in turn based on the RRT algorithm.

1) Informed RRT:*

- *Sample_new_node:* Samples a new random configuration in the configuration space of the robot
- *Check_collision(q_{rand})* Check if the new configuration collides with any of the obstacles of the environment
- *Find_nearest_node:* Within a certain radius, the nearest node to q_{rand} is found and added as its parent
- *Check_reachable:* Check if the q_{rand} is within a certain radius of q_{near} and check if the path in between them is obstacle-free. If so, set no_collision = True.
- *Rewire_tree (T):* According to the RRT* algorithm, rewire the tree to minimize path cost
- *Draw_ellipsoid (Q_{config}):* Define the new configuration space in which to sample new points as an ellipsoid with its focal points in q_{init} and q_{goal}, and its width so that the first found path is enclosed completely.

Algorithm 1 Informed RRT* Algorithm

```
1: **Input:** Initial configuration $q_{\text{init}}$, goal point $q_{\text{goal}}$, maximum iterations $K$, configuration space $Q_{\text{config}}$
2: Initialize tree $T$ with a single node at $q_{\text{init}}$
3: for $k = 1$ to $K$ do
4:     while no_collision == False do
5:         Sample_new_node $q_{\text{rand}}$ in $Q_{\text{config}}$
6:         Check_collision ($q_{rand}$)
7:         Find_nearest_node $q_{\text{near}}$ in $T$ to $q_{\text{rand}}$
8:         Check_reachable ($q_{rand}$, $q_{near}$)
9:         $q_{\text{new}}$ = $q_{\text{rand}}$
10:    end while
11:    Add $q_{\text{new}}$ to $T$ with $q_{\text{near}}$ as its parent
12:    Rewire_tree ($T$)
13:    if $q_{\text{new}}$ == $q_{\text{goal}}$ then
14:        $Q_{\text{config}}$ = Draw_ellipsoid ($Q_{config}$)
15:    end if
16: end for
17: Extract the best path from $T$
```

2) Path Smoothing: Upon completion, the path is smoothed through spline interpolation. This ensures a fast execution while still maintaining a safe distance in between the robot and the obstacles.

3) Execution: The execution of the planned path is managed through a Proportional-Integral-Derivative (PID) controller. This controller continuously adjusts the robot's movements, aligning it with the path and effectively leading it towards the goal.

B. Robotic Arm

The arm's motion planning utilizes the RRT* algorithm, similar to the mobile base, with modifications for dual target goals and the arm kinematics. Unlike the mobile base, it does not incorporate the additional informed sampling step.

1) RRT:* The core steps from algorithm 1 are used for the motion planning with the following modifications to tailor it for the arm's movements:

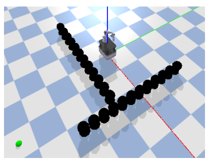
- The algorithm runs twice, once for each target goal, to pick an object out of a box and then store it on the base for transport.
- Kinematics of the arm are factored into the planning, ensuring the path accommodates the arm's range and movement constraints.
- Sampling, collision checking, and path smoothing are similar but adapted for operation in three dimensions, accounting for the higher-dimensional workspace of the arm.

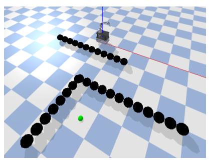
2) Execution: In the execution phase for the robotic arm, the path determined by the RRT* algorithm is translated into joint movements. This involves calculating the necessary joint movements to reach each point along the planned path based on the inverse kinematics (see equation 4) and a Proportional-Derivative (PD) controller to continually adjust the arm's movements. This control mechanism responds to real-time positional feedback, ensuring that the arm's trajectory remains aligned with the planned path.

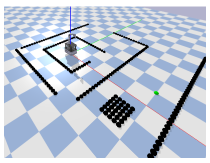
IV. Results

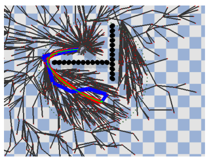
A. Mobile Base

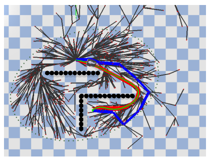
In this section, we present the outcomes for the mobile base that uses the Informed RRT* Algorithm. Initially, environments of different complexities are solved using RRT*, yielding the blue path. This is then refined by further sampling in an ellipsoid surrounding the discovered path, leading to the enhanced red path and making the RRT* informed. Finally, the path is smoothed to create the ultimate green path that the robot can follow.

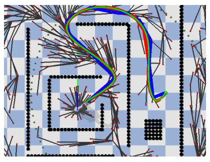

(a) Env. Easy


(b) Env. Medium


(c) Env. Maze


(d) Easy - Solved


(e) Medium - Solved


(f) Maze - Solved

All following graphs were created based on the average over 50 runs for all hyperparameter combinations. The simulation was run with a AMD Ryzen 7 4800H CPU and a NVIDIA GeForce GTX 1650 GPU.

1) Hyperparameter Tuning RRT:* To tune the step length and rewire radius of the RRT* algorithm we used two evaluation metrics, the time to compute a path and the cost (found distance).

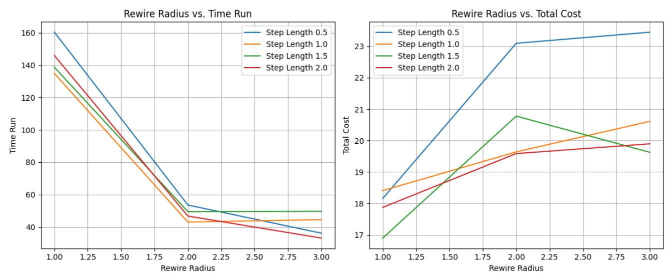


Fig. 3: Runtime and cost (distance) over rewire radii

Fig. 3 suggests that the rewire radius outweighs the step length in its effect, making it the key hyper-parameter to fine-tune. In our environment (fairly sparse obstacles and wide passages), the rewire radius in RRT* is crucial for enhancing path optimization over a wide area, which is essential for efficiency in obstacle-free spaces, unlike step length, which is less impactful in such contexts. Both graphs show an inverse correlation between time and cost, indicating a trade-off. To approximate the optimal trade-off, we maintained a step length of 1 and conducted a more detailed search of the rewire radius parameter space, as shown in Fig. 4.

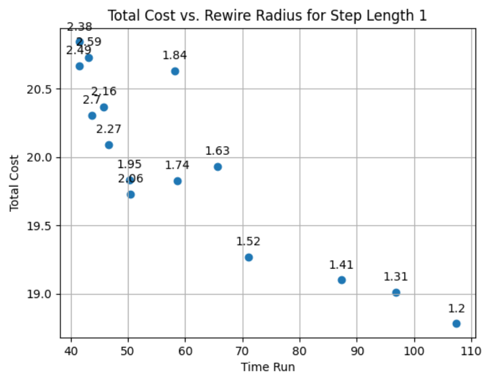


Fig. 4: Corresponding runtime and cost for rewire radii

In this graph, we observe an optimal point at the elbow, which marks the onset of diminishing returns. Beyond this point, longer computation times yield only minimal reductions in cost, making it difficult to justify the additional time. Therefore, for the medium environment, the chosen rewire radius is 1.5.

2) Hyperparameter Tuning Informed RRT:* The following graphs were used to determine the number of additionally sampled points within the ellipsoid produced by Informed RRT*.

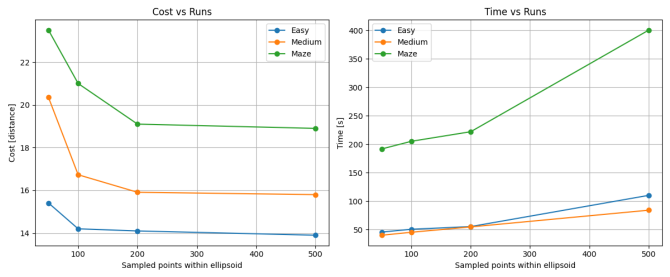
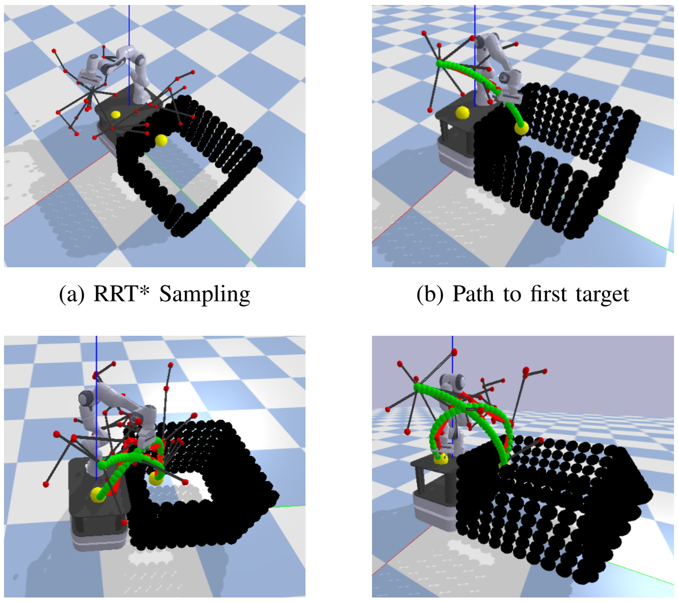


Fig. 5: Distance and time of found path with respect to sampled points

Based on the data from these graphs, we have decided to sample 200 additional configurations. This number offers a good balance between finding shorter paths and the time needed. The left graph shows that performance, in terms of path distance, improves significantly up to around 200 samples, and then levels off. The right graph indicates that the time needed to sample 200 points is reasonable. Beyond this point, the slight improvement in path distance does not justify the substantial increase in time.

B. Robotic Arm

In the following, we present the outcomes for the robotic arm utilizing the RRT* algorithm. Figure 6 illustrates the box within the virtual environment. The other images show the sequential actions of the arm as it retrieves an item from the box and places it onto the base while avoiding colliding with the box.



(a) RRT* Sampling (b) Path to first target

(c) Path to second target (d) Final position

Fig. 6: Robotic Arm Simulation

V. Discussion

In summary, the informed RRT* algorithm significantly outperforms the standard RRT* in finding a path through static obstacles, particularly when considering the balance between path efficiency and computational time. Hyperparameter tuning revealed an optimal rewire radius that enhances path optimization in sparse environments, and additional sampling within the Informed RRT*'s ellipsoid effectively reduced path lengths with reasonable additional computational costs.

A. Possible Improvements

- The algorithm's present runtime is excessively long for practical real-world applications. Thus, there is a need to either optimize the existing Python code further or consider redeveloping the entire project in a more efficient language such as C++ or Rust.
- The environment does not use non-spherical obstacles, but spherical obstacles as approximations. Consequently, this adjustment results in the motion planner's collision avoidance mechanism processing a larger number of obstacles, leading to increased computational complexity.
- For the complex environment setup and dynamics, we recommend using automated differentiation for the Jacobian calculation due to the robot's complexity, instead of the manual computation that we started out with ourselves.

B. Possible Extensions

- Utilizing RRT Connect, tailored for path planning scenarios without differential constraints, making it an ideal choice for our context as outlined in [6].
- Enhancing the arm's RRT* with a three-dimensional ellipsoid to refine the arm's path optimization process with Informed RRT*.
- Introducing dynamic obstacle handling, i.e. incorporating a local path planner to navigate around moving obstacles. This involves using RRT* and informed RRT* for global path formation, coupled with an alternate local path planner for evading dynamic obstacles like staff or other mobile units in the warehouse, thereby enhancing the robot's adaptability in active, real-world settings.
- Due to time limitations, the path planning for the robotic arm was conducted in $\mathbb{R}^3$ on the endpoint. This approach, while practical, could potentially lead to scenarios where individual joints collide with obstacles, even though the arm's endpoint successfully avoids them. Implementing a motion planner within the arm's original configuration space would mitigate this.
- Continuously adjusting the ellipsoid shape based on the trajectories discovered over time to make Informed RRT* more computationally efficient.

References

[1] *Thuiswinkel Markt Monitor FY 2022*. https://www.thuiswinkel.org/media/ei0fokzh/thuiswinkel-markt-monitor-fy-2022-light-versie.pdf. Accessed: 09.01.2024.

[2] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894.

[3] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. "Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic". In: *2014 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2014, pp. 2997–3004.

[4] Max Spahn. *Urdf-Environment*. Version 1.0.0. DOI: 10.4121/19362017. URL: https://github.com/maxspahn/gym_envs_urdf.

[5] Max Spahn. *Motion Planning Scenes*. URL: https://github.com/maxspahn/motion_planning_scenes.

[6] James J Kuffner and Steven M LaValle. "RRT-connect: An efficient approach to single-query path planning". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE. 2000, pp. 995–1001.