Quad-rotor Routing Using RRT Sampling Based Path Planners in R^3

Bram Bornhijm 4692721 Rogier Ketwaru 4698975 Alexander Keijzer 4372697 Wouter Meijer 4691659
Group 20, code under: https://github.com/h0uter/PDM-project

Abstract— **In this report we present a comparison between RRT, RRT* and Informed-RRT* path planners in R^3 for a general quad-rotor. We compare their runtime performance and optimal path cost in two unique environments. We introduce kinodynamic constraints to prevent collisions and evaluate their impact on runtime performance.**

I. INTRODUCTION

Path planning involves finding an optimal trajectory from a starting state to a goal state while avoiding invalid configurations (like obstacle collisions). A quad-rotor is simulated in environments in R^3 with static spherical and polygonal objects. For collision detection purposes we assume the quad-rotor to be a spherical rigid body. While this assumption will cause some configurations to be rejected without causing a collision with the actual quad-rotor geometry, this does allow the collision detection to be independent on the orientation of the quad-rotor reducing its complexity.

Because collision detection can be expressed in R^3 and because of the differential flatness of the system for roll and pitch, we have chosen to execute path planning in R^3 as well (with yaw controlled to 0 deg). The result of planning will be a path through the environment which will then be followed by a PID controlled quad-rotor. This controller is tuned to have low overshoot, however, some margins in planning are still needed. The quad-rotor has limitations on its accelerations and is not able to accelerate in every direction instantaneously. The controller orients the quad-rotor as needed to follow the path closely.

The state space of this planning problem is still quite high with 6-dimensions (three positions and velocities) which means an incremental sampling based method such as probabilistic roadmaps (PRM) [1] or rapidly-exploring random trees (RRT) [2] will likely have a lower computational cost for similar optimality compared to cell decomposition methods. While PRM and RRT are probabilistic complete (their chance of not finding a solution approaches zero with an increasing step count) they are not optimal – they will not find the shortest path in the configuration space. A choice for RRT was made since our problem does not need multi-query as there is a single start and goal location for each environment.

Expansions to RRT have been proposed with RRT* [3] being asymptotically optimal (the solution will approach the optimal solution with increasing step count), informed RRT* [4], where the sample space for the RRT implementation is dynamically bounded to solutions that produce shorter paths than the current shortest and RRTx [5] which allows quick replanning in dynamic environments. What is missing from all these planners is the ability to set kinodynamic constraints on the solutions. Dustin J. Webb and Jur van den Berg (2013) developed Kinodynamic RRT* [6] which allows setting linear differential constraints while still being asymptotically optimal.

In this report we will compare the performance and runtime of RRT, RRT* and informed RRT*, while also including the ability to reject paths based on kinodynamic constraints. This approach is different from Kinodynamic RRT* as it will not find the optimal path between nodes but does guarantee that the found solution will not collide with obstacles. This allows for a removal of the safety margin.

In this project, all solutions were implemented in Python using the packages numpy (for numerical computations), pyquaternion (for orientation transformations), matplotlib (for plotting and animations) and mpltoolkits (for drawing polygons). We have implemented ourselves: the simulation, visualization, drone control, obstacle definition, collision detection and planning algorithms.

II. ROBOT MODEL

We define the quad-rotor as a single rigid body to operate in R^3, as this more closely mirrors real life applications of aerial robots than a planar implementation. In R^3 the quad-copter is unconstrained in the x, y and z direction in both translation and rotation, the system thus has six unique degrees of freedom, the configuration space is therefore defined as $R^3 \times S^3$. Where R^3 represents the position of the quad copter expressed in three dimensional Cartesian space and S^3 represents the orientation of the quad-rotor expressed in Euler angles about each of the inertial axes of the inertial frame.

A. *Definitions and transformations*

We define two frames, the body fixed frame, B, centered at the center of mass of the quad-rotor, aligned with the arms of the quad-rotor to which the motors are attached, and the global inertial frame, A, positioned at the origin of the environment and used as a reference frame. An example configuration is shown in figure 1.

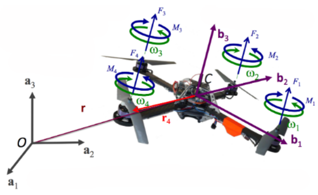


Fig. 1: Example configuration of quad-rotor in configuration space [7]

We present the 3x3 rotation matrix $R_B^A(\phi, \theta, \gamma)$ mapping frame B to frame A, resulting from the successive rotation about each global orthogonal axes of frame A, a_1, a_2 and a_3, expressed using the Euler angles, ϕ, θ, γ. Here $s\alpha$ is short for $\sin\alpha$ and $c\alpha$ is short for $\cos\alpha$.

$$R_B^A(\phi, \theta, \gamma) = \begin{bmatrix} c\phi c\theta & c\phi s\theta s\gamma - s\phi c\gamma & c\phi s\theta c\gamma + s\phi s\gamma \\ s\phi c\theta & s\phi s\theta s\gamma + c\phi c\gamma & s\phi s\theta c\gamma - c\phi s\gamma \\ -s\theta & c\theta s\gamma & c\theta c\gamma \end{bmatrix} \tag{1}$$

We also define $r_{b,org}$ as the origin of frame B as expressed in the global frame A in three dimensional Cartesian space.

B. Quad-rotor model

The quad-rotor is modelled as an infinitely stiff x shaped frame where on each end a motor is mounted, we neglect effects like air resistance, propeller flapping and the effect of angle of attack during flight.

By rotating at a given speed, denoted in figure 1 as ω_x, where x denotes the number of the motor, each motor produces a vertical lift vector expressed in the local frame and a counter torque about the local b_3-axis on the quad-rotor, shown in figure 1 as F_x and M_x. These can be expressed as follows, where k_f and k_m are torque and lift coefficients describing the behavior of the system, and d_x is the positive direction of rotation for that motor, either 1 or -1. Here, $\dot{\omega}_x I_{prop}$ is the torque produced to accelerate the propeller, where I_{prop} is the inertia of the propeller about its axis of rotation. All motors are torque and speed limited.

$$F_x = [0, 0, k_f \omega_x^2]^T \tag{2}$$

$$M_x = [0, 0, -d_x k_m \omega_x^2 - \dot{\omega}_x I_{prop}]^T \tag{3}$$

Each motor is located at a position of r_x, expressed in frame B, relative to the center of mass of the quad-rotor. We can therefore express the total force and moment vector expressed in the local B frame as follows.

$$F_{tot} = \sum_{x=1}^{4} F_x(\omega_x) \tag{4}$$

$$M_{tot} = \sum_{x=1}^{4} r_x \times F_x(\omega_x) + M_x(\dot{\omega}_x, \omega_x) \tag{5}$$

From this we extract the Newton-Euler equations of motions in the inertial frame A, where m denotes the total mass of the quad copter, I denotes the inertia tensor of the system and g the gravitational constant.

$$m\ddot{r}_{b,org} = R_B^A(\phi, \theta, \gamma) F_{tot} + [0, 0, -mg]^T \tag{6}$$

$$I[\ddot{\phi}, \ddot{\theta}, \ddot{\gamma}]^T = R_B^A(\phi, \theta, \gamma) M_{tot} - [\dot{\phi}, \dot{\theta}, \dot{\gamma}]^T \times I[\dot{\phi}, \dot{\theta}, \dot{\gamma}]^T \tag{7}$$

III. MOTION PLANNING

This section describes the motion planning algorithms and the process of their implementation, including how they have been adapted for use with a quad-rotor model. We explore both RRT, RRT* and informed RRT*.

The quad-rotor operates in a subdomain $\mathcal{C} \in \mathbb{R}^3$. The quad-rotor itself occupies a space $\mathcal{C}_{drone} \in \mathcal{C}$. We define the obstacle space $\mathcal{C}_{obs} \in \mathcal{C}$ and the initial and goal configurations q_0 and goal q_{goal}, each in $\mathcal{C} \cap \mathcal{C}_{obs}$. The objective is to find a collection of configurations which together form a path P connecting q_0 to q_{goal} where C_{drone} remains outside of $\mathcal{C}_{obs}$. This is achieved using a progressively generated graph $\mathcal{G}(V, E)$ consisting of randomly sampled vertices or nodes E and vectors connecting these V.

In algorithm 1 the `steer` function simulates the quad-rotor movements between two configurations using the controller, depending on whether dynamic simulation is enabled, this takes into account the state of the quad-rotor and the quad-rotor's kinodynamic constraints, and returns a list of discretized configurations along this simulated path or along the direct path between these nodes. Enabling this would allow the elimination of the preset safety margin, but increase computational complexity. To reduce this, the final state for the drone is stored in each node.

A line of sight check [9] is also implemented using `lineOfSightCheck`, this connects any new configurations to q_{goal} directly if no collisions occur.

Algorithm 1 functions

```
1: function COLLISIONCHECK($path$, $C_{obs}$)
2:     for $q$ in $path$ do
3:         if ($\mathcal{C}_{obs}$ at $q$) $\cup$ $\mathcal{C}_{obs} \neq \emptyset$ then
4:             return False
5:     return True

6: function LINEOFSIGHTCHECK($q$, $q_{goal}$, $\mathcal{G}$)
7:     if collisionCheck(steer($q$, $q_{goal}$), $C_{obs}$) then:
8:         $\mathcal{G}$.extend($q, q_{goal}$)
```

The RRT [2] (algorithm 2) procedure is as follows: first a graph $\mathcal{G}$ is initialised with the starting point q_0, then configurations q_{rand} are randomly sampled in $\mathcal{C}$. It is checked whether the point can be connected to nearest node q_{near} in $\mathcal{G}$ without passing through $\mathcal{C}_{obs}$. The algorithm stops when the max number of iterations K is reached, then if q_{goal} is in $\mathcal{G}$ A* is ran. For determining how to sample the random configurations the only criteria used are that the point lies outside of $\mathcal{C}_{drone}$ and inside $\mathcal{C}$. `scale` enforces the max distance Δq a new node can be connected from $\mathcal{G}$. The A* function returns the shortest path in $\mathcal{G}$ between the two nodes given as inputs using an A* implementation 3.

Algorithm 2 RRT

```
1: $\mathcal{G}$.init($q_0$)
2: $q_{new} = q_0$
3: for $i$ = 1 to $K$ do
4:     lineOfSightCheck($q_{new}$, $q_{goal}$, $\mathcal{G}$)
5:     $q_{rand}$ = sampleSpace($\mathcal{C}$)
6:     $q_{near}$ = nearestNode($q_{rand}, \mathcal{G}$)
7:     $q_{new}$ = scale($\Delta q, q_{near}, q_{rand}$)

8:     if collisionCheck(steer($q_{new}$, $q_{near}$), $C_{obs}$) then
9:         $\mathcal{G}$.extend($q_{new}, q_{connect}$)

10: if $q_{goal} \in \mathcal{G}$ then
11:     return $A^*(q_0, q_{goal})$
12: else
13:     return failure
```

RRT* [3] (algorithm 3) expands upon RRT by including two additional measures, the core principle remains the same. For brevity only the difference between RRT and RRT* is displayed in algorithm 3, line 3-12 in RRT* replaces line 8-9 in RRT. First the shortest overall path length of nodes within proximity γ of q_{near} is now taken into account. Dictionary D contains the shortest paths from q_0 to all our candidate nodes Q. In line 13 D is filled with the shortest paths resulting from each candidate point. Secondly a check is ran in line 18-19 determining whether the existing paths can reduce their overall length by taking a shortcut over the new node.

Subsequently the informed variant of RRT* [4] (algorithm 3) has been implemented. Here an ellipsoid is calculated whose primary axis is the vector spanned from the start to goal configuration and which encloses the shortest path between these configurations. This function then returns the union of this ellipse space and $\mathcal{C}$, this space is then set as the new sample space and is updated every time a shorter path is found.

Algorithm 3 RRT star

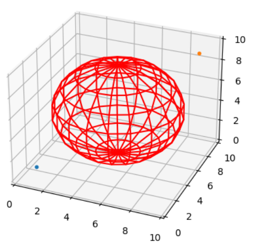
```
1:  for $i$ = 1 to $K$ do
2:      ...
3:      $Q = \text{findVerticesInRange}(q_{new}, \gamma, \mathcal{G})$
4:      for $q$ in $Q$ do
5:          if collisionCheck(steer($q$, $q_{new}$), $\mathcal{C}_{obs}$) then
6:              $D \leftarrow (q, A^*(q_0, q))$

7:      $q_{connect} = D[\min |D|]$
8:      $\mathcal{G}$.extend($q_{connect}, q_{new}$)
9:      $D$.remove($q_{connect}$)

10:     for $q, p$ in $D$ do
11:         if $\min |D| + |q - q_{new}| < |p|$ then
12:             $\mathcal{G}$.rewire($p[-2], q, q_{new}$)
13:     ...
```

IV. RESULTS

In this section we discuss the experimental setup and the results of the experiments. We compare the planners with each other in the aspects of runtime and cost of the optimal found path as a function of maximum amount of iterations. We defined runtime as time taken for a motion planning algorithm to complete its maximum amount of iterations and cost as euclidean length of the shortest found path. All of the experiments are executed on a 4.2 GHz AMD processor with 16 GB of RAM. For these experiments we choose the seeds 1, 2, 3, 42 and 69 to calculate average performance. The chosen maximum iterations for the scenarios are 600, 800, 1000, 1200 and 1400.

A. Simulation environment scenarios

Both scenarios are inside a cubic domain of 10x10x10 meters in the x, y and z axes and increase in complexity. In figure 2 the initial drone position is displayed as a blue point and the goal position is displayed as an orange point.


(a) Scenario 1

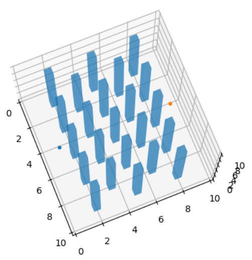
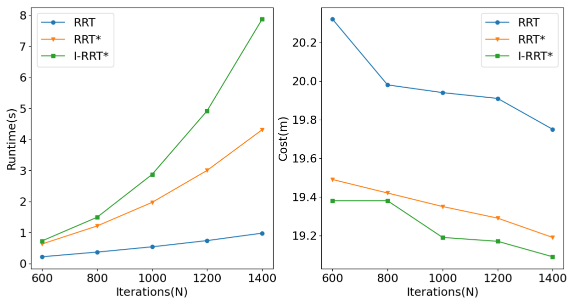

(b) Scenario 2

Fig. 2: Simulation scenarios 1 and 2

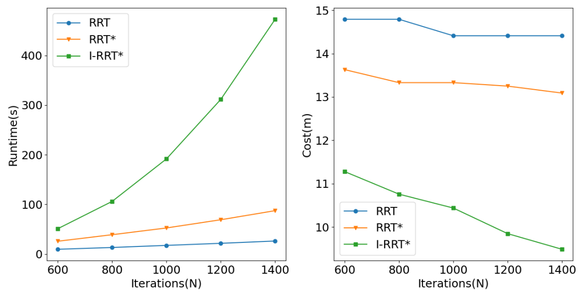
The first scenario contains a sphere with a diameter of 80% the height of the domain and is placed in the center of the domain. The initial position of the drone and its goal are placed diagonally from each other to force the motion planners to plan a path around the sphere as seen in figure 2a.

The second scenario contains a grid of pillars with dimensions of 0.5 by 0.5 meters and the height of the domain. The rows of pillars alternate in phase of approximately π to allow for less straightforward path planning. The initial position of the drone is near the center of a wall in the domain and the goal is near the center of the opposite wall as shown in figure 2b.

B. Quantitative analysis



(a) Scenario 1



(b) Scenario 2

Fig. 3: Graphs containing results of scenarios 1 and 2

We can observe in table I that the results across all planners are fairly similar regarding costs of optimal found path with a decrease of approximately 6% between the highest cost and the lowest. That is, the difference between RRT with 600 iterations and I-RRT* with 1400 iterations. The common trend is a decreasing cost when the amount of iterations increases, this is further shown in the cost graph of figure 3a. There is also a small distinction in cost between RRT, and RRT*, I-RRT* where the latter planner pair has a consistently lower cost than RRT. When looking at the total runtime for scenario 1, we see a clear distinction between the planners as well, where RRT* and I-RRT* have a significantly higher runtime than RRT. This points to a correlation between a higher runtime and a smaller cost for the most optimal found path. Table II reinforces this hypothesis where the differences in cost and runtime become more apparent as shown in 3b. Here the difference between the highest cost and lowest is a decrease of about 36% where the solution with the smallest cost was found by informed RRT* with 1400 iterations. However this performance does seem to come at a cost, the increase in runtime between RRT with 600 iterations and the planner with the smallest cost is close to 5000%. It is noticeable as well that the graph for I-RRT* shows a non-linear relation between runtime and amount of iterations. There were also some experiments done using kinodynamic RRT and RRT*, from these we concluded that the performance increase was negligible with a dramatic increase in runtime of about 14,000% in the worst case for scenario 1, that is RRT* versus kinodynamic RRT* with 1400 iterations.

TABLE I: Table containing results of planners for scenario 1

Scenario 1	RRT		RRT*		I-RRT*	
Iterations	Runtime (s)	cost (m)	Runtime (s)	cost (m)	Runtime (s)	cost (m)
600	0.22 ± 0.01	20.32 ± 1.65	0.63 ± 0.03	19.49 ± 0.54	0.73 ± 0.15	19.38 ± 0.57
800	0.37 ± 0.01	19.98 ± 0.92	1.21 ± 0.06	19.42 ± 0.43	1.49 ± 0.37	19.38 ± 0.57
1000	0.54 ± 0.02	19.94 ± 0.84	1.97 ± 0.11	19.35 ± 0.42	2.87 ± 0.63	19.19 ± 0.40
1200	0.74 ± 0.02	19.91 ± 0.83	3.00 ± 0.19	19.29 ± 0.40	4.92 ± 0.86	19.17 ± 0.39
1400	0.98 ± 0.04	19.75 ± 0.76	4.31 ± 0.25	19.19 ± 0.56	7.87 ± 1.52	19.09 ± 0.31

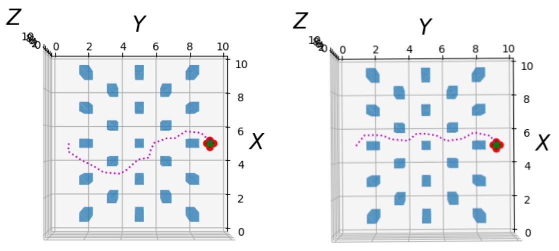
TABLE II: Table containing results of planners for scenario 2

Scenario 2	RRT		RRT*		I-RRT*	
Iterations	Runtime (s)	cost (m)	Runtime (s)	cost (m)	Runtime (s)	cost (m)
600	9.56 ± 0.33	14.79 ± 0.45	25.74 ± 1.35	13.63 ± 1.30	51.03 ± 0.23	11.28 ± 0.52
800	13.19 ± 0.62	14.79 ± 0.45	38.90 ± 2.13	13.33 ± 1.33	106.06 ± 0.34	10.76 ± 0.68
1000	17.37 ± 0.78	14.41 ± 1.05	52.51 ± 2.94	13.33 ± 1.33	191.76 ± 0.23	10.44 ± 0.86
1200	21.60 ± 1.00	14.41 ± 1.05	69.07 ± 4.62	13.25 ± 1.39	311.67 ± 0.32	9.85 ± 0.42
1400	26.17 ± 1.36	14.41 ± 1.05	87.35 ± 5.00	13.09 ± 1.34	472.98 ± 0.51	9.49 ± 0.20

C. *Qualitative analysis*

For our qualitative analysis we compare the paths found by our path planners for the different scenarios.

Since, as is evident from the quantitative analysis, the costs of the optimal solutions are similar for scenario 1, we will not qualitatively analyse this scenario and instead only analyse scenario 2. The paths of RRT* and I-RRT* will be analysed



(a) RRT* (b) I-RRT*

Fig. 4: Optimal solutions found

only, because the path of RRT as shown in tables I and II has a much worse performance compared to the other planners. As can be seen from figure 4 RRT* and I-RRT* have paths that look like slaloms, in contrast to RRT which had a more single arc shaped path. This explains the consistently lower cost of the optimal found path for both RRT* and its informed variant. When comparing RRT* and I-RRT* we see that the path of RRT* makes two large arcs, while the path of I-RRT* makes several smaller arcs. The path of RRT* would be easier to follow by a drone, because the changes in direction are less extreme, however the path found by I-RRT* is significantly closer to the optimal solution.

V. DISCUSSION

The RRT solutions presented are complete, while the RRT*-based methods are also asymptotically optimal. However, this is only true in practice if the system has no kinodynamic constraints as the planner might create paths that are impossible to follow for the system. Our system has limitations on its accelerations and is not able to accelerate in every direction instantaneously. This means that without the kinodynamic constraints the planners will always give a solution and a RRT*-based planner will eventually give an optimal solution, however, in some cases these trajectories are not valid once simulated.

To solve this, during path planning the actual trajectories between nodes will be simulated and the state of the system is saved at every node. This results in rejection of paths that collide with an obstacle due to errors in path following. However, this also means that in some situations the path planner might not be able to find a valid path as we only simulate a single trajectory between nodes. The actual path planning is therefore not complete and not optimal.

The time complexity of RRT and RRT* is $O(nlogn)$ and the space complexity is $O(n)$ where n is the number of samples [3]. While informed RRT* and the kinodynamic constraints do change the search and solution space of the algorithm they do not change the time complexity. The space complexity for adding the kinodynamic constraints increases to $O(ns)$ where s is the length of the state vector since a state vector is saved at every node.

With 600 iterations every trial and method found a solution to the path planning problem for both environments. Increasing the amount of samples shows a decrease in path length in the RRT* based methods as expected. Informed RRT* finds a more optimal solution with the same amount of samples compared to RRT*. However, without kinodynamic constraints these paths can in some cases cause collisions with obstacles.

With kinodynamic constraints enabled there is a guarantee there will be no collisions with obstacles. However, this comes at a large runtime cost. It might be possible that solving a linearized model optimization problem like in kinodynamic RRT* would result in more optimal solution while still having a lower runtime.

While we have not encountered this problem, it is possible for the path planner to be unable to find a suitable path when the kinodynamic constraints are enabled. Depending on the use-case this could be a clear downside of using this method.

A. *Future Work*

As discussed, the main shortcomings of the current planner are that it does not have optimal node to node control and its inability to avoid obstacles dynamically, which limits its usage to static and controlled environments. The current implementation is also only single trajectory, meaning it could not find the optimal path between multiple destinations.

If the objective is to generalize to dynamic environments then the current method of motion planning could be changed to a variant of RRTx or a behavioral tree controlled MPC controller, assuming fast enough on board processing capability.

References

[1] L. Kavraki, P. Svestka, J.-C. Latombe, M. Overmars (1996). *Probabilistic roadmaps for path planning in high-dimensional configuration spaces.* IEEE Trans. on Robotics and Automation 12(4):566580.

[2] Steven M. Lavalle (1998). *Rapidly-Exploring Random Trees: A New Tool for Path Planning.*

[3] S. Karaman, E. Frazzoli (2011). *Sampling-based algorithms for optimal motion planning.* Int. J. of Robotics Research 30(7):846-894

[4] Gammell, J. D., Srinivasa, S. S., Barfoot, T. D. (2014, September). *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 (pp. 2997-3004).

[5] Otte, M., 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.

[6] Dustin J. Webb, Jur van den Berg (2013). *Kinodynamic RRT*: Optimal Motion Planning for Systems with Linear Differential Constraints.* IEEE International Conference on Robotics and Automation (pp. 5054-5061).

[7] J.A.Mora (2020) *Planning and Decision Making. Lecture 6: Incremental Motion Planning.* Cognitive Robotics, TU Delft

[8] J. H. Reif (1979). *Complexity of the mover's problem and generalizations.* In Proceedings of the IEEE Symposium on Foundations of Computer Science.

[9] Zhang, Changwu Zhou, Li Liu, Hengzhu. (2019). *LaLo-Check: A Path Optimization Framework for Sampling-Based Motion Planning With Tree Structure.* IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2930634.