Drone delivery in cities to balconies through funneled RRT*

Nadine Duursma
nduursma
4665236

Amber Elferink
aelferink
5418526

Simon van Gemert
simonvangemert
4932269

Lonit Peeters
lmpeeters
5126533

I. Introduction

Package delivery by drones will become increasingly important in this modern age. Companies such as Amazon, Google, DHL and Walmart have shown interest in the possibility of using drone based delivery, creating the carrier pigeons of the new age [1], [2]. Furthermore, drone delivery could also be used for e.g. delivering food packages after natural disasters such as earthquakes [2].

However, a problem with drones is their short battery life. A strategy to limit the drone battery usage is to drive the drones to a common area with a truck. Then one or more drones are released that deliver at short distances [1]–[4]. This way, the drones perform last-mile delivery over a shorter distance, and thus need less energy to deliver.

There are multiple papers about last-mile delivery algorithms, e.g. [1]–[10]. In some of these papers, a combined optimization of the truck and the drone is presented [3], [4], [7], [9]. Furthermore, in most of the papers packages are delivered with a fleet of drones in an urban zone. For example, Chatterjee and Reza [10] describes a method to create drone fleet delivery systems within cities, using a rapidly exploring random tree (RRT) algorithm to find the optimal paths for the drones. Brunner, Szebedy, Tanner, *et al.* [5] even let their drone deliver at balconies. However, they required both GPS and visual data to do so.

In this paper, the path planning for a drone from a stationary truck to a final delivery destination in a city will be investigated. The delivery destination will be a balcony, as in [5], in a part of New York. However, we will not use visual navigation. Instead, we will use the RRT* algorithm for the complete path planning, since it is suitable for high dimensional planning, and often used for drones in city navigation [10], [11]. For RRT*, we will use parts of the Python code originally created by Sakai, Ingram, Dinius, *et al.* [12]. Furthermore, we will use CoppeliaSim [13] as a visualisation software for our simulations. The code of the project can be found on Github [14].

Currently, the RRT library for the city scale takes long to plan. We propose an algorithm which improves the performance of the RRT(*) algorithm. The samples taken within the default RRT* are spread over the entire space, without taking the goal into account. Shkolnik, Walter, and Tedrake [15] propose a method to take the local reachability of the random samples into account by applying differential constraints. Tang, Chen, Lan, *et al.* [16] and Xinyu, Xiaojuan, Yong, *et al.* [17] consider a potential field to guide the RRT* algorithm. We propose an algorithm to converge faster without potential fields. The random node is sampled from a funnel in the direction of the goal, which is broadened if it cannot find a valid path in a given threshold of iterations.

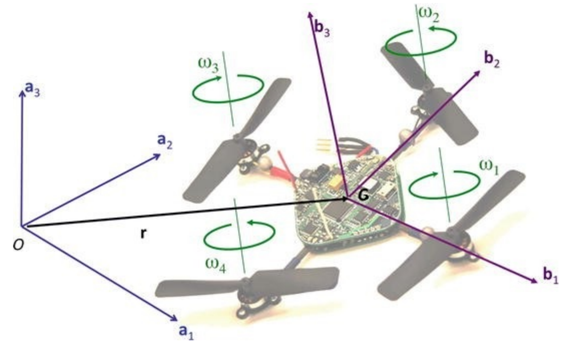

Fig. 1. Placement of the body frame in reference to the center of mass and the world frame. *Reprinted from [18].*

In subsection II-A Equations of Motion the equations of motion for the drone will be discussed. In subsection II-B Workspace and Configuration Space the differences between work and configuration space will be highlighted. In section III Motion Planning the path planning will be elaborated on in greater detail. The simulation step will be discussed in section IV Simulation Setup.

II. Robot Model

A. Equations of Motion

In order to define the equations of motion, first a world frame and a body-fixed frame are defined in three dimensions. The body-fixed frame is attached to the drone in the center of mass G as can be seen in Fig. 1. The first two axes ($\mathbf{b}_1$ and $\mathbf{b}_2$) of the body-fixed frame are perpendicular and aligned with the arms of the drone, ($\mathbf{b}_3$) is normal to the plane spanned by $\mathbf{b}_1$ and $\mathbf{b}_2$ and points towards the top of the drone. The world frame is defined with the $\mathbf{a}_3$-direction pointing opposite to the gravity and initially $\mathbf{a}_1$ and $\mathbf{a}_2$ are aligned with $\mathbf{b}_1$ and $\mathbf{b}_2$ respectively.

Each of the four rotors rotates with a certain angular velocity ω_i. Two rotors that are opposite each other have an angular velocity with the same sign. However, the other pair of rotors has an opposite sign. This way the moments produced by the pairs of rotors cancel out if both spin with the same angular velocity. By rotating with an angular velocity ω_i, each rotor generates a force F_i and a moment M_i, of which the magnitudes are defined as

$$F_i = k_F \omega_i^2, \tag{1}$$
$$M_i = k_M \omega_i^2, \tag{2}$$

where k_F and k_M are motor coefficients found in Table I. The forces are directed in the positive $\mathbf{b}_3$-direction. The moments are around the $\mathbf{b}_3$-direction, directed opposite to the angular velocity ω_i. Furthermore, the forces also cause moments $\mathbf{r}_i \times \mathbf{F}_i$, where $\mathbf{r}_i$ is a vector from the center of the drone towards the center of the corresponding rotor and $\mathbf{F}_i$ is the force vector. Lastly, there is the gravitational force mg, with m the mass of the drone and g the gravitational constant. This force points in the negative $\mathbf{a}_3$-direction.

The forces expressed in the body-fixed frame can be expressed in the world frame by multiplying them with the 3D rotation matrix $\mathcal{R}$. This leads to the following Newton equations of motion in (3):

$$\ddot{\mathbf{r}}m = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \mathcal{R} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix}, \tag{3}$$

where $\ddot{\mathbf{r}}$ is the acceleration of the center of mass of the drone in the world frame.

Assuming that rotor 1 and 3 rotate in the negative $\mathbf{b}_3$-direction and rotor 2 and 4 in the positive $\mathbf{b}_3$-direction, with the moments directed in opposite direction, the Euler equations of motion can be written as:

$$I\dot{\mathbf{\Omega}} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \mathbf{\Omega} \times I\mathbf{\Omega}, \tag{4}$$

where $\mathbf{\Omega}$ contains the angular velocity components of the drone expressed in the body-fixed frame (derived from the derivatives of the roll, pitch, and yaw angles, not from the rotor angular velocities ω_i), and I is the diagonal inertia matrix of the drone. Furthermore, we assume here that the length of $\mathbf{r}_i$ is equal to L for all four arms of the drone. To simplify these Euler-equations, the coefficients k_F and k_M can be written as a fraction b:

$$b = k_M / k_f. \tag{5}$$

Using this, the Euler equations of motion can be written as a function of the forces generated by the motors (F_i) and the angular velocity vector in the body-fixed frame ($\mathbf{\Omega}$) as follows:

$$I\dot{\mathbf{\Omega}} = \begin{bmatrix} 0 & L & 0 & -L \\ -L & 0 & -L & 0 \\ -b & b & -b & b \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} - \mathbf{\Omega} \times I\mathbf{\Omega}. \tag{6}$$

The drone has four control inputs, namely F_1 though F_4. However, the drone is a rigid body in 3D, which has six degrees of freedom. This means that the drone is underactuated. It can be shown that the drone is differential flat, and that the following four flat variables can be used:

$$\sigma = [x, y, z, \psi] \tag{7}$$

where x, y, z are the coordinates of the center of mass of the drone, and ψ is the yaw angle [18].

In this project, we will make use of the Hummingbird drone, which has been described by Powers, Mellinger, and Kumar [18]. The inertial and motor characteristics of the Hummingbird are given in Table I.

k_M	1.5×10^{-9} Nm/rpm^2
k_F	6.11×10^{-8} N/rpm^2
I_{xx}	2.32×10^{-3} kg/m^2
I_{yy}	2.32×10^{-3} kg/m^2
I_{zz}	4.00×10^{-3} kg/m^2
m	0.5 kg
L	0.175 m

TABLE I
INERTIAL AND MOTOR CHARACTERISTICS OF THE HUMMINGBIRD. *Taken from [18].*

B. Workspace and Configuration Space

The drone needs to navigate in an approximately 2 km^2 area between Times Square and the Central Park in New York, where it needs to fly from the start location on the truck to the delivery location on a balcony. The 3D models of the city and truck were obtained from [19], [20]. Since the drone can move in x, y and z direction, the workspace is $W \in \mathbb{R}^3$.

Since we consider the differential flatness of the drone, we can describe the configuration of the drone by three position components (x, y, and z) and one yaw angle (ψ). Therefore, the configuration space is $C \in \mathbb{R}^3 \times \mathbb{S}^1$.

III. Motion Planning

The Rapidly-exploring Random Tree * (RRT*) path planning algorithm is asymptotically optimal and probabilistic complete. Rapidly-exploring Random Tree (RRT) [21] works by connecting randomly generated nodes to existing nodes until a node can be connected to the goal. RRT* differs from RRT in how it connects new nodes to the nearest existing node. In RRT* the new node first checks its vicinity and connects only to the node via which it has the lowest cost to the start.

a) RRT application:* In the city the drone flies though spaces much larger than the drone itself. The drone is not expected to preform any complex maneuvers where the orientation of the drone allows passage where it is otherwise restricted. With this in mind, the path planning space is restricted to $\mathbb{R}^3$ and the yaw angle is calculated afterwards following the computed path. This project implements an RRT* algorithm created by Sakai, Ingram, Dinius, *et al.* [12], which was altered to meet the needs of the project.

RRT* needs to find a path within a decent amount of time. To achieve this, concessions are made in path length and we apply a priory knowledge about the environment to edit the search behaviour. A large workspace can make the

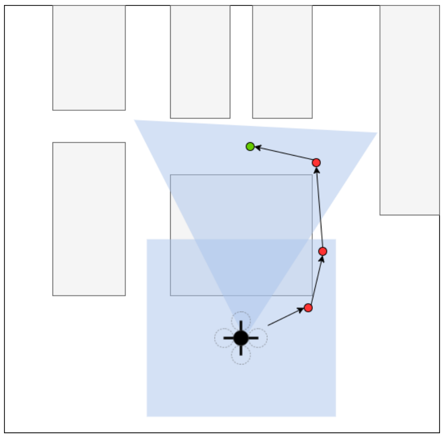

Fig. 2. Altered RRT* operation visualized in 2D. Blue is the search zone, the red dots the path, green the goal, and the obstacles in grey.

algoritm slow; for the algorithm to search the entire city is less than ideal because the size of the space would require many iterations to find a solution. Therefore, a funnel is defined, centered in the drone and pointing towards the goal in which the algorithm generates new nodes (Tang, Chen, Lan, *et al.* [16]). $\boldsymbol{p}$ in (8) is any random point in a funnel originating in start($\boldsymbol{p_s}$). Here ϕ and ψ are the Euler angles of the direction to the goal with θ around the z-axis and ψ around the y'-axis. θ is the apex-angle of the funnel, 0.25π by default. α is the angle around the center line and r is the radius. l is the distance from the start between a minimum and maximum.

$$\begin{cases} l, & l_{min} \leq l \leq L * i/i_{lim} \leq l_{max} \\ r, & 0 \leq r \leq l * \tan(\theta/2) \\ \alpha, & -\pi \leq \alpha \leq \pi \\ \boldsymbol{p} & = \begin{bmatrix} \cos\phi\cos\psi & \sin\phi\cos\psi & \sin\psi \\ \sin\psi & \cos\phi & 0 \\ -\sin\psi\cos\phi & -\sin\psi\sin\phi & \cos\psi \end{bmatrix} \begin{bmatrix} l \\ r\cos\alpha \\ r\sin\alpha \end{bmatrix} + \boldsymbol{p_s} \end{cases} \tag{8}$$

Use of the funnel is possible because there are no maze like structures in the city at the scale at which RRT* plans the route. The funnel search restriction can lead to an impossible scenario where the drone is facing a building, while the goal is behind it, and the funnel does not extend around the sides of the building. As a solution, a secondary cubical search zone is introduced around the drone with $\boldsymbol{p}$ defined by (9). Independent of how $\boldsymbol{p}$ is found, there exists a hard limit such that $z \leq 0$.

$$\begin{cases} x, & -l_{min} \leq x \leq l_{min} \\ y, & -l_{min} \leq y \leq l_{min} \\ z, & -l_{min} \leq z \leq l_{min} \\ \boldsymbol{p} & = [x \quad y \quad z]^T + \boldsymbol{p_s} \end{cases} \tag{9}$$

Both the funnel and cubical search zones are visualized in Fig. 2 in the xy-plane. Note that the cube is aligned with world axis while the funnel faces the goal. If RRT* fails to find a path with the allotted iteration, it enlarges θ and increases the maximum number of iterations.

Direction	x	y	z
linear P (P_L)	300	300	7000
linear I (I_L)	0.02	0.02	6
linear D (D_L)	0.450	450	4200
angular P (P_A)	22000	22000	1500
angular I (I_A)	0	0	2
angular D (D_A)	12000	12000	0

TABLE II
PID TUNING WEIGHTS

The second improvement comes from RRT* trying to connect new nodes to existing nodes. In the first few iterations there are only a few nodes to connect. The search area is therefore limited to a distance away from the start that increases with each iteration up to a maximum. The limiting factor is implemented in (8). L is the distance from start to goal, i is the current iteration, and i_{lim} is the iteration from where the full length is used.

b) PID controller: The drone follows the path using a PID controller from [22] which was modeled and tuned to the specifications of the humming bird (Table I). The PID weights can be found in Table II. The controller implements the non-linear model (3)-(6) of the drone. The state can be described in twelve variables $\boldsymbol{x} = [\boldsymbol{r}^T \boldsymbol{\psi}^T \dot{\boldsymbol{r}}^T \boldsymbol{\omega}^T]$.

Thrust is calculated based on the propeller radius and pitch. The controller updates the motor speeds every time step $dt = 0.002s$ with the flat variables (7) using the set of equations in (10). Here $\boldsymbol{l_i}$ and $\boldsymbol{a_i}$ are linear and angular integration errors, $\boldsymbol{P_L}$, $\boldsymbol{P_A}$, $\boldsymbol{D_L}$, $\boldsymbol{D_A}$, $\boldsymbol{I_L}$, and $\boldsymbol{I_A}$ are the linear and angular weights (table II) in diagonal matrix form. $\boldsymbol{\psi}$ is the vector of the drone angles θ, ϕ, and γ. Since the system in deferentially flat θ and ϕ can be expressed as functions of γ. $\boldsymbol{\Delta r_i}$ is the linear error.

$$\begin{cases} \boldsymbol{l_i} = \boldsymbol{l_{i-1}} + \boldsymbol{I_L}\boldsymbol{\Delta r_i} \\ \dot{\boldsymbol{r_d}} = \boldsymbol{P_L}\boldsymbol{\Delta r_i} + \boldsymbol{D_L}(-\dot{\boldsymbol{r_i}}) + \boldsymbol{l_i} \\ \theta_d = s_\theta(\dot{x_d}\sin\gamma - \dot{y_d}\cos\gamma) \\ \phi_d = s_\phi(\dot{x_d}\cos\gamma + \dot{y_d}\sin\gamma) \\ \gamma_d = s_\gamma\gamma - \dot{\gamma_i} \\ \boldsymbol{\Delta\psi_i} = \boldsymbol{\psi_d} - \boldsymbol{\psi_i} \\ \boldsymbol{a_i} = \boldsymbol{a_{i-1}} + \boldsymbol{I_A}\boldsymbol{\Delta\psi_i} \\ \boldsymbol{\Omega} = \dot{z_d} + \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \\ 0 & -1 & -1 \end{bmatrix} (\boldsymbol{P_A}\boldsymbol{\Delta\psi_i} + \boldsymbol{D_A}(-\boldsymbol{\omega_i}) + \boldsymbol{I_A}\boldsymbol{a_i}) \end{cases} \tag{10}$$

IV. Simulation Setup

The bounding boxes covering the obstacles in the city and the targets are shown in Fig. 3. RRT*, the dynamic model of the drone, and the PID controller were implemented in Python and connected to CoppeliaSim [13] to visualize the path.

To test the effect of the funneled search, the drone was instructed to fly from Start through all ordered targets (Gi) until arriving at the goal G5, see Fig. 3. At each target the RRT* tree was reset, so the target it arrived at was set as the

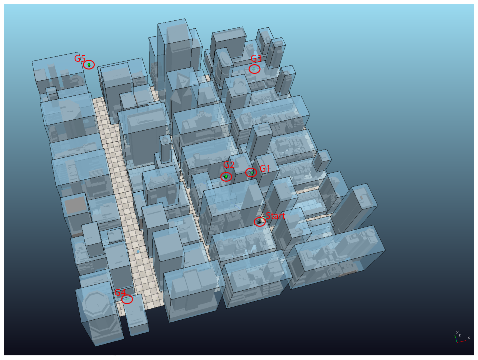

Fig. 3. The 3D model with corresponding bounding boxes. The robot starts at Start, then flies through G1, G2, G3, G4 and G5 in order. At each target, the RRT tree is reset and it plans again from its current position.

new starting point. The path G1 to G2 is expected to be hard for the funnel since the funnel will look directly towards the building which blocks the spawned points. G2 to G3 tests an intermediate distance (direct 1.4km) with a ending point in a narrow area. For G3 to G4 a long distance (direct 2.4km) was simulated. Then G4 to G5 is a route with much open area. This was repeated 10 times for the funnel algorithm, and 10 times for the default. For all of the RRT* generations, the planning time, flight time, number of nodes placed and iterations to calculate a route were saved.

V. Results

Fig. 4 shows the time, number of nodes and iterations needed to find a path with and without the funnel algorithm. The funneled algorithm was faster, used less nodes and took less iterations to find to the solution in all cases. This is to be expected since the random points are generated in a smaller volume, and therefore it finds a node to fit there more often. Additionally, the tree gets less broad and more depth, therefore reaching the goal in less iterations, nodes and time.

Something of note is the suspected difficulty of the path. G2 was planned as the most difficult target to reach but it falls short compared to G4 and is generally comparable to G1 without the funnel. With the funnel the computational complexity of reaching G2 was larger than for reaching G1. The distance to the target was of far greater influence than the location, in generations. Surprising is that the actual drone flight time was also shorter for all cases with the funnel algorithm. This was not expected, because the path was planned beforehand. More research is needed to explain this result.

VI. Discussion

RRT* will find a solution if the solution exists when enough samples are taken. With infinite computation time it finds the optimal solution. However, infinite computation time is not available, therefore the algorithm cannot generate an infinite number of samples which makes the motion path found optimal only up to a certain resolution. In practise, the found path was not remarkably non-optimal. The task of

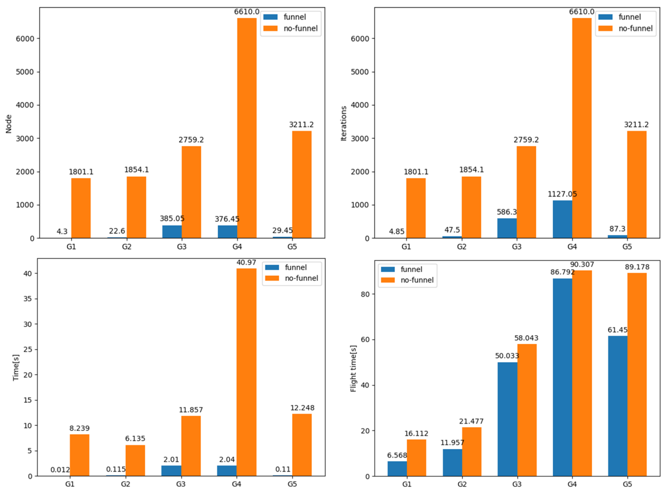

Fig. 4. The time, number of nodes and iterations required for performing RRT, either with or without the proposed funnel algorithm.

delivering packages does not require an accuracy of seconds, thus a fully optimal path is not desired for this purpose. The computational complexity for RRT* is $O(n \log n)$ [23], optimality and computational time can be traded off as shown in the results.

RRT* is a single query method, thus the road map cannot be re-used after the robot moves, which makes it best suitable for (but not limited to) static environments. If the drone is expected to work in the same environment, a multi-query method will probably provide more reliable and optimal results. This can be paired with another path planner or MPC to check and avoid obstacles locally.

The way that RRT* handles obstacles in this implementation is still very limited. Because the environment is based on New York the obstacles are all axis aligned boxes. To apply this algorithm to any other city would require the use of different shapes. A general polygon obstacle detection will offer more options for application. Obstacles can also be grouped together in large groups and excluded from search if they are not relevant to the search field.

Despite multiple attempts at tuning the controller, one of the bigger issues with using PID is that the controller attempts to round-off corners and sometimes collides. A partial fix to this issue was already implemented. That is, to add additional nodes to the path in order to limit the controllers freedom. This only works partially and for future expansion a different type of controller is recommended.

For further research it is also recommended to evaluate the motion planner in dynamic environments. This is representative for real cities and is therefore needed before the drone makes a test flight in package delivery. If it then turns out that RRT* is not sufficiently reactive, other methods like motion primitives or MPC could be considered for motion planning which are expected to be faster. However, these do not have feasibility guarantees.

References

[1] S. Poikonen, X. Wang, and B. Golden, *The vehicle routing problem with drones: Extended models and connections*, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/net.21746.

[2] K. Dorling, J. Heinrichs, G. G. Messier, and S. Magierowski, "Vehicle routing problems for drone delivery," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 1, pp. 70–85, 2017. DOI: 10.1109/TSMC.2016.2582745.

[3] S. Sawadsitang, D. Niyato, P. Tan, and P. Wang, "Joint ground and aerial package delivery services: A stochastic optimization approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 6, pp. 2241–2254, 2019. DOI: 10.1109/TITS.2018.2865893.

[4] M. Moshref-Javadi, A. Hemmati, and M. Winkenbach, "A truck and drones model for last-mile delivery: A mathematical model and heuristic approach," *Applied Mathematical Modelling*, vol. 80, pp. 290–318, Apr. 2020. DOI: 10.1016/j.apm.2019.11.020. [Online]. Available: https://doi.org/10.1016/j.apm.2019.11.020.

[5] G. Brunner, B. Szebedy, S. Tanner, and R. Wattenhofer, "The urban last mile problem: Autonomous drone delivery to your balcony," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019, pp. 1005–1012. DOI: 10.1109/ICUAS.2019.8798337.

[6] B. Rabta, C. Wankmüller, and G. Reiner, "A drone fleet model for last-mile distribution in disaster relief operations," *International Journal of Disaster Risk Reduction*, vol. 28, pp. 107–112, Jun. 2018. DOI: 10.1016/j.ijdrr.2018.02.020. [Online]. Available: https://doi.org/10.1016/j.ijdrr.2018.02.020.

[7] H. D. Yoo and S. M. Chankov, "Drone-delivery using autonomous mobility: An innovative approach to future last-mile delivery problems," in *2018 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 2018, pp. 1216–1220. DOI: 10.1109/IEEM.2018.8607829.

[8] L. D. P. Pugliese and F. Guerriero, "Last-mile deliveries by using drones and classical vehicles," in *Springer Proceedings in Mathematics & Statistics*, Springer International Publishing, 2017, pp. 557–565. DOI: 10.1007/978-3-319-67308-0_56. [Online]. Available: https://doi.org/10.1007/978-3-319-67308-0_56.

[9] P. Kitjacharoenchai, B.-C. Min, and S. Lee, "Two echelon vehicle routing problem with drones in last mile delivery," *International Journal of Production Economics*, vol. 225, p. 107 598, Jul. 2020. DOI: 10.1016/j.ijpe.2019.107598. [Online]. Available: https://doi.org/10.1016/j.ijpe.2019.107598.

[10] A. Chatterjee and H. Reza, "Path planning algorithm to enable low altitude delivery drones at the city scale," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, Dec. 2019. DOI: 10.1109/csci49370.2019.00142. [Online]. Available: https://doi.org/10.1109/csci49370.2019.00142.

[11] Nasa, *Nasa/daidalus*. [Online]. Available: https://github.com/nasa/daidalus.

[12] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques, *Pythonrobotics: A python code collection of robotics algorithms*, 2018. arXiv: 1808.10703 [cs.RO].

[13] Coppelia Robotics, *Coppeliasim*, version 4.1.0, Jul. 1, 2020. [Online]. Available: https://www.coppeliarobotics.com/.

[14] N.Duursma, A. Elferink, S. Gemert, and L. Peeters, *Pdm-python-rrt*. [Online]. Available: https://github.com/sion559/PDM-python-RRT.

[15] A. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *2009 IEEE International Conference on Robotics and Automation*, IEEE, May 2009. DOI: 10.1109/robot.2009.5152874. [Online]. Available: https://doi.org/10.1109/robot.2009.5152874.

[16] Z. Tang, B. Chen, R. Lan, and S. Li, "Vector field guided rrt* based on motion primitives for quadrotor kinodynamic planning," *Journal of Intelligent & Robotic Systems*, vol. 100, no. 3, pp. 1325–1339, 2020, ISSN: 1573-0409. DOI: 10.1007/s10846-020-01231-y. [Online]. Available: https://doi.org/10.1007/s10846-020-01231-y.

[17] W. Xinyu, L. Xiaojuan, G. Yong, S. Jiadong, and W. Rui, "Bidirectional potential guided rrt* for motion planning," *IEEE Access*, vol. 7, pp. 95 046–95 057, 2019. DOI: 10.1109/ACCESS.2019.2928846.

[18] C. Powers, D. Mellinger, and V. Kumar, "Quadrotor kinematics and dynamics," *Springer*, p. 20, Aug. 2015. DOI: https://doi.org/10.1007/978-90-481-9707-1_71.

[19] Anonymous, *Manhattan city buildings 3d model*. [Online]. Available: https://open3dmodel.com/3d-models/manhattan-city-buildings_481564.html.

[20] printable models, *Vocational truck 3d model*. [Online]. Available: https://free3d.com/3d-model/vocational-truck-v1--260059.html.

[21] S. M. LaValle and J. James J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001. DOI: 10.1177/02783640122067453. eprint: https://doi.org/10.1177/02783640122067453. [Online]. Available: https://doi.org/10.1177/02783640122067453.

[22] A. Majumdar, "Quadcopter$_{s}$*imulator*," 2018. [Online]. Available: https://github.com/abhijitmajumdar/Quadcopter_simulator.

[23] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, Jun. 2011. DOI: 10.1177/0278364911406761. [Online]. Available: https://doi.org/10.1177/0278364911406761.