QUADROTORS MINIMUM SNAP TRAJECTORY GENERATION AND CONTROL FOR OBSTACLE AVOIDANCE

LEJUN JIANG [LEJUNJ@SEAS.UPENN.EDU], ZHIHAO RUAN [RUANZH@SEAS.UPENN.EDU],

ABSTRACT. We have developed a planning and control system for quadrotors based on A^* algorithm, minimum snap trajectory smoothing, constrained gradient descent, and a geometric nonlinear controller. We validated it on the Crazyflie simulator and achieved desirable performance. Our system has the capability to produce reasonable performance regarding speed, smoothness, and obstacle avoidance, which provides a valuable resource for agile quadrotor navigation in densely cluttered environments. A demo video can be accessed through `https://drive.google.com/drive/folders/1XDKcjWqlBFtxO4lXuxHJHt0cJgrcXa30?usp=sharing`. Our source code can be found at `https://drive.google.com/file/d/1KPaMhowWdNh-wwtMXf0R9PrGa6VA59N7/view?usp=sharing`.

1. INTRODUCTION

Autonomous quadrotors have been widely used in industry. However, the dynamics of quadrotors is in general very complex and inaccurate control of quadrotors usually leads to catastrophic consequences. Plus, due to the highly complex environment the quadrotors need to deal with, an efficient yet reliable path planning algorithm for collision avoidance must be incorporated. Therefore, a system of efficient and accurate algorithms on trajectory generation and control is critical to developing a quadrotor application.

In this report, we will explore a set of quadrotor path planning and control techniques. The rest of the sections are organized as follows: Section 2 introduces the Dijkstra's algorithm and A^* that we are going to use for waypoint generation. Section 3 discusses the optimization based Minimum Snap Trajectory Generation algorithm. Section 4 describes a nonlinear geometric controller that is used to follow the generated trajectory. Section 5 integrates the described algorithms into our full quadrotor trajectory planning and control system. Section 6 presents our experimental results on this system. Section 7 discusses the results and concludes the report.

Our contribution includes: implementation of Dijkstra's & A^* algorithm for 3-D occupancy grid-space collision avoidance and planning; trajectory smoothing achieved by Quadratic Programming based Minimum Snap Trajectory Generation; optimization of time segments for the Minimum Snap Trajectory Generation with Gradient methods; accurate control of drone dynamics through a realization of nonlinear geometric controller.

2. DIJKSTRA'S & A^* ALGORITHM

Dijkstra's algorithm [1] and its variant A^* algorithm [2] are classic yet efficient algorithms for path planning with obstacle avoidance. The algorithms are based on graphs and they search through the graph space to find the minimum cost trajectory to connect two nodes. The pseudocode for the Dijkstra's algorithm is included in Algorithm 2.

The A^* algorithm is different from the Dijkstra's Algorithm in that it adds a term "cost-to-goal" to the distance before each node is pushed into the pq, this speeds up the search significantly since it will allow us to neglect most of the unnecessary nodes. The "cost-to-goal" is a distance estimate for the distance between the current node and the End node, and it needs to be consistent [3]. In our case, we discretized the 3-D space with regular occupancy grids (one grid represents one node) and used the Euclidean distance for the distance calculation between two neighboring nodes, same with the "cost-to-goal" calculation. We specified two nodes to be neighbor if their distance in infinity norm is less or equal to 1 (in occupancy grid coordinate), and of course neither of the nodes should be an obstacle.

2.1. Related Work.

The main references are [1] and [2]. These two algorithms were chosen due to their completeness and optimality. With our heuristics and appropriate discretization, A^* could run quite efficiently even in the 3-D search space. It should be noted, however, there exists other path planning algorithms that are also suitable for this task. Other algorithms, such as RRT [4] and FMT* [5] trades completeness and optimality for efficiency, and therefore are suitable for higher-dimensional searches. When the search space becomes too large for A^* to handle, we can switch to these algorithms to reduce the computation time.

3. MINIMUM SNAP TRAJECTORY GENERATION & TIME SEGMENT OPTIMIZATION

3.1. Formulating the equation of spline.

Given a list of waypoints, minimum-snap trajectory generation tends to interpolate the trajectory between consecutive waypoints using some function $x = x(t)$. Without loss of generality, we can assume $x : \mathbb{R} \mapsto \mathbb{R}$. In a 3-D environment, the location of the quadrotor can be written as $\mathbf{r} : \mathbb{R} \mapsto \mathbb{R}^3$, $\mathbf{r}(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix}^T$, and each axis $x(t), y(t), z(t)$ will be interpolated using a separate function.

As indicated by its name, the algorithm minimizes the square of the "snap" (4th derivative) of the quadrotor position:

$$\min_{x(t)} \quad \int_0^T \left(\ddddot{x}(t)\right)^2 \, \mathrm{d}t = \min_{x(t)} \quad \int_0^T L(t) \, \mathrm{d}t \tag{1}$$

where T is the predefined desired travelling time of this trajectory segment. In the theory of Lagrangian mechanics, the solution to such minimization problem must satisfy the principal of least action, namely

$$\frac{\partial L}{\partial x} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{x}} + \frac{\mathrm{d}^2}{\mathrm{d}t^2}\frac{\partial L}{\partial \ddot{x}} + \cdots + (-1)^k \frac{\mathrm{d}^k}{\mathrm{d}t^k}\frac{\partial L}{\partial x^{(k)}} = 0 \tag{2}$$

which, in the sense of minimizing $L = \left(\ddddot{x}(t)\right)^2$,

$$\frac{\mathrm{d}^4}{\mathrm{d}t^4}\frac{\partial L}{\partial \ddddot{x}} = 2 \times \frac{\mathrm{d}^4}{\mathrm{d}t^4}\left(\ddddot{x}(t)\right) = x^{(8)}(t) = 0$$

indicating that $x(t)$ would be a 7th-order polynomial. Therefore, minimum-snap algorithm essentially uses a 7th-order spline to interpolate between each pair of consecutive waypoints, while minimizing the square of the 4th-derivative of the spline subject to some additional constraints.

3.2. **Set up the optimization problem.** Assume we have N waypoints and thus $m = N - 1$ trajectories, and each of them can be interpolated with a 7th-order spline. As previously mentioned we assume that the general equation for the i-th trajectory: $p_i(t) = c_{i,7}t^7 + c_{i,6}t^6 + c_{i,5}t^5 + c_{i,4}t^4 + c_{i,3}t^3 + c_{i,2}t^2 + c_{i,1}t + c_{i,0}$. We wish to set up an optimization problem that minimizes $\int_0^T (\ddddot{p}_i(t))^2 dt$. Notice that for our trajectories to be smooth, we must define the following constraints:

(1) $p_i(0) = p_i, p_i(T) = p_{i+1}$. The spline must go through the pre-defined waypoints.
(2) $\dot{p}_1(0) = 0, \dot{p}_m(T) = 0$. At the beginning and the end of the entire trajectory, the quadrotor must not be moving.
(3) $\dot{p}_i(0) = \dot{p}_{i-1}(T), \ddot{p}_i(0) = \ddot{p}_{i-1}(T), \dot{p}_i(T) = \dot{p}_{i+1}(0), \ddot{p}_i(T) = \ddot{p}_{i+1}(0)$. The velocity and acceleration at each waypoints in the middle must be consistent in the two consecutive splines (continuity constraint).

Notice that all constraints are linear equality constraints in terms of the coefficients $c_{i,j}$ of the splines and we can write them in matrix form $Ac = b$, where $c = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{m,m} \end{bmatrix}^T$.

Taking the first spline as an example, we can derive the objective of the optimization problem as follows:

$$\begin{aligned} p_1(t) &= c_{1,7}t^7 + c_{1,6}t^6 + c_{1,5}t^5 + c_{1,4}t^4 + c_{1,3}t^3 + c_{1,2}t^2 + c_{1,1}t + c_{1,0} \\ \ddddot{p}_1(t) &= 840c_{1,7}t^3 + 360c_{1,6}t^2 + 120c_{1,5}t + 24c_{1,4} \end{aligned}$$

$$\begin{aligned} \text{objective} &= \int_0^T (\ddddot{p}_1(t))^2 dt \\ &= 100800c_{1,7}^2T^7 + 25920c_{1,6}^2T^5 + 4800c_{1,5}^2T^3 + 576c_{1,4}^2T + 100800c_{1,7}c_{1,6}T^6 + 40320c_{1,5}c_{1,7}T^5 \\ &\quad + 21600c_{1,5}c_{1,6}T^4 + 10080c_{1,4}c_{1,7}T^4 + 5760c_{1,4}c_{1,6}T^3 + 2880c_{1,4}c_{1,5}T^2 \\ &= \begin{bmatrix} c_{1,7} & c_{1,6} & c_{1,5} & c_{1,4} \end{bmatrix} \begin{bmatrix} 100800T^7 & 50400T^6 & 20160T^5 & 5040T^4 \\ 50400T^6 & 25920T^5 & 10800T^4 & 2880T^3 \\ 20160T^5 & 10800T^4 & 4800T^3 & 1440T^2 \\ 5040T^4 & 2880T^3 & 1440T^2 & 576T \end{bmatrix} \begin{bmatrix} c_{1,7} \\ c_{1,6} \\ c_{1,5} \\ c_{1,4} \end{bmatrix} \end{aligned}$$

Notice that the objective function is a quadratic function and we can rewrite it as

$$\begin{aligned} \min_c \quad & f = c^T H c \\ s.t. \quad & Ac = b \end{aligned} \tag{3}$$

Therefore the entire problem becomes a QP (Quadratic Programming) problem and we can solve it using any QP solver. The solution to this QP problem would be the desired spline $p(t)$ that is used to interpolate the states between the two waypoints.

3.3. **Optimize travelling time for each segment.** In the previous settings we have been finding the optimal spline parameters c assuming each spline segment takes a constant time T to travel. However, we may also develop an algorithm to vary T efficiently and find the optimal T_i for each segment $i = 1, \dots, m$. In fact, this can be done using constrained gradient descent.

Assume $\mathbf{T} = \begin{bmatrix} T_1 & T_2 & \cdots & T_m \end{bmatrix}^T$ and $f(\mathbf{T})$ is the optimal value of the optimization problem in Equation (3). In particular, in 3-dimensional space, we should consider $f(\mathbf{T})$ as the average of the optimal values of Equation (3) performed in all three axes. Under this assumption, the optimal $\mathbf{T}$ satisfies the following optimization problem:

$$\begin{aligned} \min_{\mathbf{T}} \quad & f(\mathbf{T}) \\ s.t. \quad & \sum_{i=1}^{m} T_i = 0 \\ & T_i \geq 0 \end{aligned} \tag{4}$$

Unfortunately this is a non-convex problem and we have to perform constrained gradient descent to solve for suboptimal $\mathbf{T}$. Algorithm 1 describes the constrained gradient descent process.

Algorithm 1: Constrained Gradient Descent on $\mathbf{T}$

```
Input  : $\mathbf{T}_0, \varepsilon_0, \varepsilon_1$, some small number $h$
Output : Optimal $\mathbf{T} = \mathbf{T}_{\text{opt}}$
$\mathbf{T} := \mathbf{T}_0$
while solving is not timed out and $\|\mathbf{T} - \mathbf{T}_{\text{old}}\|_2 \geq \varepsilon_0$ do
    $\nabla f(\mathbf{T}) := \sum_{i=1}^{m} \frac{f(\mathbf{T}+h\mathbf{g}_i)-f(\mathbf{T})}{h}\mathbf{g}_i$, where $\mathbf{g}_i = [\frac{-1}{m-1}, \dots, 1, \dots, \frac{-1}{m-1}]$ where the $i^{\text{th}}$ entry is 1;
    Search for $\mathbf{T}_{\text{new}}$ using constrained backtracking line search;
    $\mathbf{T}_{\text{old}} := \mathbf{T}$;
    $\mathbf{T} := \mathbf{T}_{\text{new}}$;
end
return $\mathbf{T}$;
```

4. Nonlinear Geometric Controller

4.1. **Quadrotor dynamics.** Assume each motor on the quadrotor produces some force proportional to the motor angular speed such that $\mathbf{F}_i = k_F \omega_i^2$ and some moment $M_i = k_M \omega_i^2$. Define $u_1 = \sum_{i=1}^{4} F_i$ as the total force produced by the four motors. It is natural to find out that the direction of $\mathbf{F}_i$ should always be the z-axis of the body frame, namely $\mathbf{z}_{\mathcal{B}}$. Furthermore, we can also express the rate of change of the angular momentum of the quadrotor in terms of the mement produced by the rotors:

$$\dot{\mathbf{L}} = \mathbf{M} \tag{5}$$

Expressing $\dot{\mathbf{L}}$ in the body frame with $\mathbf{b}_i \in \{\mathbf{x}_{\mathcal{B}}, \mathbf{y}_{\mathcal{B}}, \mathbf{z}_{\mathcal{B}}\}$, and we have

$$\dot{\mathbf{L}} = \sum_i \dot{L}_{i\mathcal{B}} \mathbf{b}_i + \sum_i L_{i\mathcal{B}} \dot{\mathbf{b}}_i = \sum_i \dot{L}_{i\mathcal{B}} \mathbf{b}_i + \omega \times L_{i\mathcal{B}} \mathbf{b}_i = \sum_i \dot{L}_{i\mathcal{B}} \mathbf{b}_i + \omega \times \mathbf{L} \tag{6}$$

Combining Eq. (5) and Eq. (6) yields

$$I\dot{\omega}_{\mathcal{B}} = \mathbf{M} - \omega_{\mathcal{B}} \times I\omega_{\mathcal{B}} \tag{7}$$

$$m\ddot{\mathbf{r}} = -mg\mathbf{z}_{\mathcal{W}} + u_1 \mathbf{z}_{\mathcal{B}} \tag{8}$$

where $\mathcal{W}$ denotes the world frame and $\mathcal{B}$ denotes the body frame. Since the inertia tensor I in the quadrotor body frame is constant, Eq. (7) actually defines the quadrotor rotation dynamics and indicates that the rotation is only related to the moment produced by the motors. Eq. (8) and (7) collaboratively define the quadrotor dynamics related to the motor rotation speed. Using the fact that motor 1 and 3 rotates clockwise while motor 2 and 4 rotates counterclockwise on a quadrotor, if we express $\omega = p\mathbf{x}_{\mathcal{B}} + q\mathbf{y}_{\mathcal{B}} + r\mathbf{z}_{\mathcal{B}}$ and expand Eq. (7) we can get

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{9}$$

where L is the distance from axis of rotation of rotor to the center of mass of the quadrotor. Therefore we can also define $\mathbf{u}_2 = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix}$, and thus we can construct a direct relationship between $u_1, \mathbf{u}_2$ and $\omega_1, \omega_2, \omega_3, \omega_4$ by

$$\begin{bmatrix} u_1 \\ \mathbf{u_2} \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{10}$$

4.2. **Geometric Non-linear Controller.** The non-linear controller is based on a PD controller written as

$$\mathbf{F}_{des} = -K_p \mathbf{e}_p - K_v \mathbf{e}_v + mg\mathbf{z}_{\mathcal{W}} + m\ddot{\mathbf{r}}_T \tag{11}$$

where $\mathbf{e}_p = \mathbf{r} - \mathbf{r}_T, \mathbf{e}_v = \dot{\mathbf{r}} - \dot{\mathbf{r}}_T$. Given the desired force, u_1 is very easy to compute as it represents the total force from the motor, which is along $\mathbf{z}_{\mathcal{B}}$. Hence, we have

$$u_1 = \mathbf{F}_{des} \cdot \mathbf{z}_{\mathcal{B}} \tag{12}$$

which enables the quadrotor to reach the desired position. Next we calculate the total moment needed for the quadrotor to reach the desired orientation. Notice that the desired orientation can be represented by $\mathbf{z}_{\mathcal{B},des} = \mathbf{F}_{des} / \left\lVert \mathbf{F}_{des} \right\rVert_2$.

If we set $\mathbf{x}_{C,des} = \begin{bmatrix} \cos\psi_T & \sin\psi_T & 0 \end{bmatrix}^T$ where ψ_T is the current yaw angle, we have

$$\mathbf{y}_{\mathcal{B},des} = \frac{\mathbf{z}_{\mathcal{B},des} \times \mathbf{x}_{C,des}}{\left\lVert \mathbf{z}_{\mathcal{B},des} \times \mathbf{x}_{C,des} \right\rVert_2}, \quad \mathbf{x}_{\mathcal{B},des} = \mathbf{y}_{\mathcal{B},des} \times \mathbf{z}_{\mathcal{B},des}, \quad R_{des} = \begin{bmatrix} \mathbf{x}_{\mathcal{B},des} & \mathbf{y}_{\mathcal{B},des} & \mathbf{z}_{\mathcal{B},des} \end{bmatrix} \tag{13}$$

which defines the desired rotation matrix R_{des}. Utilizing the properties of $SO(3)$, we can further define the error on rotation and the error on angular velocity

$$\mathbf{e}_R = \frac{1}{2}(R_{des}^T R - R^T R_{des})^{\vee}, \quad \mathbf{e}_\omega = \omega - \omega_{des} \tag{14}$$

where $\vee$ defines the *vee*-map (the inverse mapping of hat-map) that maps the elements from $SO(3)$ to $\mathbb{R}^3$. $\mathbf{u}_2$ is thus computed by

$$\mathbf{u}_2 = -K_R \mathbf{e}_R - K_\omega \mathbf{e}_\omega. \tag{15}$$

Given $u_1, \mathbf{u}_2$, we can finally compute the desired motor speed by Eq. (10).

4.3. **Related Work.** Another possible controller is to use a simple PID controller. Given the desired position, we can apply a simple PID controller on u_1 that controls the altitude of the quadrotor by adjusting the forces of the four rotors. However the non-linear controller is undoubtedly better as it incorporates the desired position and the orientation nicely using quadrotor dynamics, while keeping it simple to compute.

5. Approach

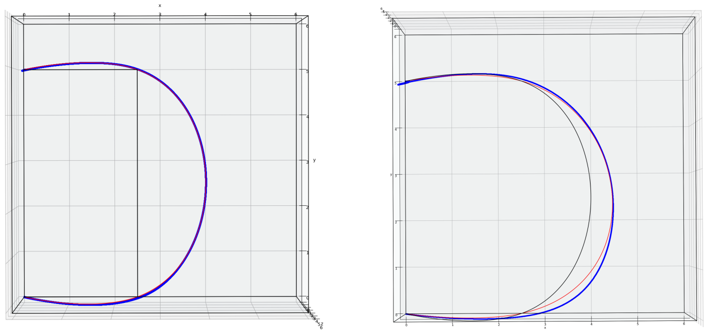
We integrate above algorithms as follows:

(1) Specify the start and goal positions and use the A^* algorithm to find a set of waypoints connecting the start and goal, avoiding any obstacles.
(2) Reduce the number of waypoints to speed up the following minimum snap trajectory generation, we did it here by sampling every one out of three waypoints generated by the previous step, note that there could be better methods of doing this, but we will not focus on it here.
(3) Smoothen the trajectory by the minimum snap trajectory generation algorithm, perform constrained gradient descent to optimize travelling time for each segment.
(4) Track the trajectory generated in the previous step by the geometric nonlinear controller.

6. Experimental Results

We implemented our algorithm on the Crazyflie simulator, which is borrowed from the course MEAM 620. The initialization of spline segment time $\mathbf{T}$ is proportional to the distance of each line segment, with the total time specified as an input. To demonstrate the result of minimum-snap trajectory smoothing, we conducted experiments on the waypoints $\{(0,0,0), (2.5,0,0), (2.5,5,0), (0,5,0)\}$, comparing plain straight line connections with the results after minimum-snap trajectory smoothing. The comparison result is shown in Fig. (1a). We also observe that our controller follows the smoothed trajectory nicely, which could not happen if the trajectory is not smoothened.

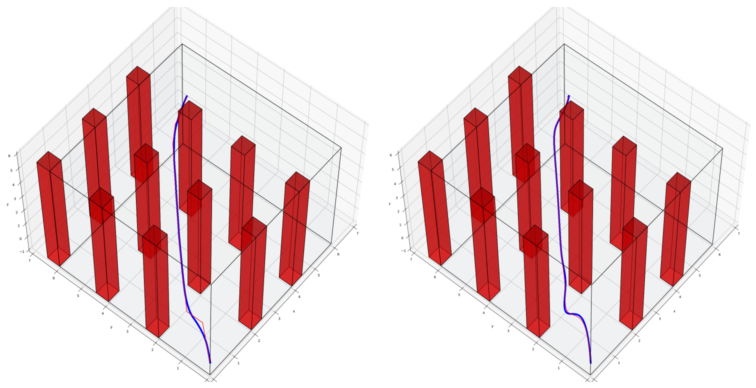
Furthermore, we also evaluated our results on time optimization, and the comparison is shown in Fig. (1b). Through our optimization, $\mathbf{T}$ has changed from `[2.  4.  2.]` to `[1.63984034 4.43712494 1.92303473]` in seconds. We can see that the algorithm allocates more time for the quadrotor to fly in the segment $(2.5, 0, 0), (2.5, 5, 0)$, while decreasing time in the two other segments. This decreases the total snap required for the trajectory completion.



(A) Smoothing vs. no smoothing (B) Time optimized vs. no optimized

FIGURE 1. Left: comparison of using minimum-snap trajectory smoothing (red curve) vs. no smoothing (black curve). Right: comparison of using time optimization (red curve) vs. no optimization (black curve). Blue curve: actual trajectory with controller.

To demonstrate the functionality of our full planning and control system, we constructed an environment with obstacles as shown in Fig. 2. It can be seen that the generated trajectories avoid the obstacles smoothly and efficiently. The minimum snap trajectory smoothens the trajectory further and allows the controller to track it with error close to 0.



(A) Original A^* trajectory (B) Smoothened A^* trajectory

FIGURE 2. Original A^* trajectory (left, interpolated from waypoints) vs. Smoothened A^* trajectory (right) in an environment with dense obstacles. Red curve is the desired trajectory and blue curve is the actual trajectory tracked by the controller in simulation. Start point is at (0, 0, 0) m, end point is at (5, 5, 5) m, total allocated time is 14 s.

7. DISCUSSION

Our experimental results indicates that our planning and control system has the capability to produce reasonable performance regarding speed, smoothness, and obstacle avoidance. This provides a valuable resource for agile quadrotor navigation in densely cluttered environments. Our algorithm takes about 18 seconds to run for the environment specified in Fig. (2b). This is reasonable for static environments where the planning task can be done offline.

As a future work for our project, we would like to improve the runtime of our algorithm for it to adapt dynamic environments [6]. We would also like to switch to a high-fidelity simulation environment such as FlightGoggles [7] to further verify the efficiency of our algorithm.

REFERENCES

[1] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische mathematik 1 (1) (1959) 269–271.

[2] P. Hart, N. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics 4 (2) (1968) 100–107. doi:10.1109/tssc.1968.300136.
URL https://doi.org/10.1109/tssc.1968.300136

[3] Wikipedia, Consistent heuristic, online; accessed 10-May-2021.
URL https://en.wikipedia.org/wiki/Consistent_heuristic

[4] S. M. LaValle, Rapidly-exploring random trees: A new tool for path planning (1998). doi:TR9811.
URL http://msl.cs.uiuc.edu/~lavalle/papers/Lav98c.pdf

[5] L. Janson, E. Schmerling, A. Clark, M. Pavone, Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions, The International Journal of Robotics Research 34 (7) (2015) 883–921, pMID: 27003958. doi:10.1177/0278364915577958.
URL https://doi.org/10.1177/0278364915577958

[6] D. Burke, A. Chapman, I. Shames, Generating minimum-snap quadrotor trajectories really fast (2020). arXiv:2008.00595.

[7] W. Guerra, E. Tal, V. Murali, G. Ryou, S. Karaman, Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality, 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (Nov 2019). doi:10.1109/iros40897.2019.8968116.
URL http://dx.doi.org/10.1109/IROS40897.2019.8968116

APPENDIX A. DIJSKTRA'S ALGORITHM

Algorithm 2: Dijkstra's Algorithm

```
Input   : Graph, Start node, End node
Output : Optimal path from start node to end node with minimum distance
for each vertex $v$ in Graph do
    visited[$v$] := False; // array that stores whether each node is visited;
    dist[$v$] := infinity; // array that stores the distance of each node to the Start node;
    parent[$v$] := none; // array that stores the parent of each node;
end
pq = []; // priority queue storing the current estimated distance of the nodes to the Start node;
dist[Start node] := 0;
visited[Start node] := True;
while pq is not empty and visited[End node] is False do
    $u$ := first unvisited node in pq;
    remove $u$ from pq;
    for each unvisited neighbor $v$ of $u$ do
        distance := dist[$u$] + dist_between($u$, $v$);
        if distance $<$ dist[$v$] then
            dist[$v$] := distance;
            parent[$v$] := u;
            push $v$ with value of distance to pq;
        end
    end
end
Backtrack the optimal path using the parent array from End node to Start node;
```