2020 20th International Conference on Control, Automation and Systems (ICCAS 2020)
Oct. 13∼16, 2020; BEXCO, Busan, Korea

A Fast and Safe Motion Planning Algorithm in Cluttered Environment using Maximally Occupying Convex Space

Youngsang Suh, Jiseock Kang, and Dongjun Lee*

Department of Mechanical Engineering, Seoul National University,
Seoul, 08826, Korea (sys0427, jskang0894, djlee@snu.ac.kr) * Corresponding author

Abstract: This paper presents a new planning method in cluttered environment using a newly defined concept called a maximally occupying convex space (MOCS). Motion planning methods that use safe flight corridor (SFC) generally construct SFC in every new trial based on each initial path. We define a MOCS in a three-dimensional rectangular world to be prebuilt as SFC, prior to the initial path computation that leads to optimization process without running SFC construction phase. In the rectangular world, creating every possible MOCS covers all the free space overlapping with others if there exists a feasible path among those spaces. Thus, expressing MOCSs as nodes and their overlappings as edges, we can formulate a simplified graph representing the free space. Dijkstra algorithm is applied to a simplified graph in this work. This approach enables to omit constructing SFC every new trial, and to use path planning algorithm with small amount of nodes thereby, achieve efficient computation. We provide a completeness of a planning algorithm in a rectangular space, and show computation efficiency compared with a state-of-the-art algorithm based on SFC.

Keywords: safe flight corridor, convex space, motion planning, dynamic programming

1. INTRODUCTION

Unmanned aerial vehicles (UAVs) have been used in various applications recent years. Yet, fast and safe motion planning of UAV is still a crucial task to achieve. Many novel methods have been demonstrated with the framework, computing initial path by searching-based algorithm or sampling-based algorithm then, proceeding trajectory optimization using polynomial or b-spline trajectory based on initial path [1].

Among those, there have been several studies using safe flight corridor (SFC) to generate safe trajectories [1–5]. SFC consists of overlapping convex spaces that guarantee collision avoidance when waypoints are located inside the overlapping areas. To construct SFC, iterative convex regional inflation by semidefinite programming (IRIS) algorithm that generates maximum volume ellipsoid and its surrounding polytope from a seed point has been used [2, 6]. IRIS algorithm can create large convex spaces in complicated environment. Unfortunately, this method takes several seconds to return a trajectory which is not adequate for real-time applications. In [3], online trajectory generation has been accomplished by initializing SFC with A* algorithm and grid inflation, improving its former method [7]. Also, different ways to generate SFC that overcome computation issue have been introduced [1,4]. However, previous studies need seed points or compute new SFC every trial based on initial path.

A method that creates every SFC before motion planning process even without seed points would enable to reduce computation time spent every trial and to improve seed point issues. Also, in conjunction with the information of prebuilt SFCs, the free space can be described as small number of nodes. Thus, changing the framework can improve the total process.

For this, we particularly propose the new concept of *Maximally Occupying Convex Space* (MOCS) and use that for the fast and safe motion planning in the three-dimensional (3D) rectangular worlds. We define MOCS to be prebuilt in the free space as SFC and our suggested algorithm can compute trajectories in real-time, without seed points. MOCS is defined using an inclusion relation and an algorithm completeness in the rectangular worlds is followed by Lem. 1 and Th. 1. Given 3D cluttered rectangular environment, a dynamic programming method is used to generate every MOCS in the free space. Checking overlapping of MOCSs is done by geometric relation between spaces, and graph is constructed by connecting overlapping MOCSs. Dijkstra algorithm is applied to a graph for initial MOCS path computation. Waypoints are set in the middle of the overlapping spaces for collision avoidance. In this study, we compute 7^{th} order polynomial trajectory with jerk minimization. Feasibility and efficiency of the proposed method is confirmed through RViz, Gazebo simulation and comparison with previous study. Our proposed MOCS-based algorithm turns out to be 1.7 times faster than a well-known SFC-based algorithm [1] in our simulated environment.

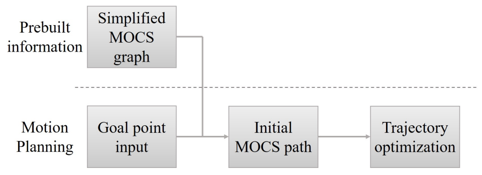

Fig. 1 Overview of the proposed method. Trajectory optimization is proceeded promptly without generating SFC every trial.

The rest of the paper is constructed as follows. Section 2 introduces a concept of MOCS and shows the completeness of the algorithm. Section 3 describes the method of algorithm implementation. Section 4 illustrates the system setup, simulation experiment, and computation time comparison. Finally, Section 5 presents our conclusions.

2. MAXIMALLY OCCUPYING CONVEX SPACE

In this section, we propose the concept of MOCS, which constitutes the core of our motion planning algorithm, and establish the equivalence between the existence of feasible workspace path and the existence of MOCS path in the rectangular worlds. The proof indicates the completeness of the proposed method. We first start with the definitions of the concept of MOCS and the feasible path, and the equivalence is established in the Th. 1 with the necessary Lem. 1 preceding that.

Definition 1. *Given a rectangular world free space $\mathbb{W}$, a set $\mathbb{C}$ of convex subsets in $\mathbb{W}$ and a set $\mathbb{O}$ of cuboid obstacles, a cuboid convex closed set C is a maximally occupying convex space if and only if $\forall c \in \mathbb{C}$ s.t. $c \not\subset C$, $c \cup C$ is not convex set.*

Definition 2. *f is a feasible path if and only if it can be specified by continuous $f : [0, 1] \to \mathbb{W}$ with $\frac{||\Delta f(s)||}{\Delta s}$ being bounded.*

Example of MOCS in 2D is shown in Fig. 2. We can define a set M as

$$M := \cup_{i=1}^{N} C_i, \tag{1}$$

where $C_i \subset \mathbb{W}$ is the i-th MOCS and N is the cardinality of MOCS sets in $\mathbb{W}$. M is finite in rectangular worlds. Also, feasible path should not encounter with obstacles and its velocity cannot increase continuously.

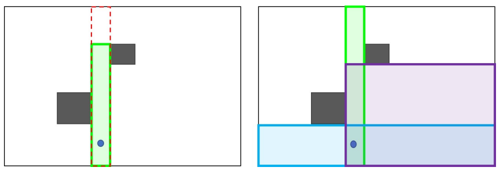
Fig. 2 Example of MOCS in 2D. Left: green convex space is not a MOCS, due to its union with red rectangle is a convex space. Right: green, light blue, purple convex spaces are MOCS that contain a blue point

Now we establish the equivalence between the existence of feasible path and the existence of MOCS path. For this, we need the following Lem. 1, which implies that the maximum value of the shortest distance from a point to its surrounding MOCSs is always greater than 0.

Lemma 1. *For $x \in \text{int}(\mathbb{W})$, convex set $S \subset \mathbb{W}$, let us define $d_S(x) = min\{\|x - y\| \mid y \in \partial S\}$, $D(x) = max\{d_C(x) \mid x \in \text{int}(C),\ C \in M\}$. Then, $D(x) > 0$.*

Proof. For a point $x \in \text{int}(\mathbb{W})$ and a convex set $S \subset \mathbb{W}$, define $d_S(x)$ as

$$d_S(x) = min\{\|x - y\| \mid y \in \partial S\}. \tag{2}$$

There exists neighborhood $N_\epsilon(x)$ in $\mathbb{W}$ for any x that is not an empty set. $N_\epsilon(x)$ is a convex set thus, there is an element of M, MOCS C, including $N_\epsilon(x)$. Then, we can make an inequality

$$d_C(x) \geq d_{N_\epsilon(x)}(x) = \epsilon > 0, \tag{3}$$

and define $D(x)$ which max value exists because M is a finite set. By Eq. (5), $D(x)$ is greater than 0.

$$D(x) = max\{d_C(x) \mid x \in \text{int}(C),\ C \in M\} \tag{4}$$

$$D(x) \geq d_C(x) > 0 \tag{5}$$

We show this Lem. 1 to ensure the min value of $D(x)$ among the path is greater than 0. The length of the maximum path segment penetrating surrounding MOCSs cannot be smaller than the min value of $D(x)$, which helps to conclude that there is a finite sequence of MOCS to cover the feasible path.

Theorem 1. *For $\forall x, y \in \text{int}(\mathbb{W})$, there exists feasible path from x to y if and only if there exists a sequence $\mathbb{C} = \{C_1, C_2, \dots, C_n\}$ s.t. $x \in C_1, y \in C_n, C_i \in M$, $\text{int}(C_i) \cap \text{int}(C_{i+1}) \neq \phi$.*

Proof. To prove the equivalence, the existence of feasible path inside MOCS path is a trivial problem so we only need to show whether there exists MOCS path including such given feasible path.

Let a feasible path denoted by $f : [0, 1] \to \mathbb{W}$, and s as a domain element. Then, we can formulate a sequence $\mathbb{S}$ with the following rule:

$$S_1 = 0, \tag{6}$$

$$S_{i+1} = max\{\tilde{S} \mid C_i \in M, f([S_i, \tilde{S}]) \subset C_i\}, \tag{7}$$

if $S_n = 1$, terminate the process. When $n = 2$, it is trivial so, we deal with the case when $n > 2$.

Feasible path is created inside the free space thus, $D(f(s))$ is bounded, continuous which indicates the existence of min value of $D(f(s))$. By Lem. 1, its value is greater than 0.

$$k = min\{D(f(s)) \mid s \in [0, 1]\} > 0 \tag{8}$$

Now we consider the end point because it is possible to be terminated before reaching maximum boundary at the end. Let $l([a, b])$ as a curve length of f from $f(a)$ to $f(b)$, and $\bar{D}(x) = min\{l([s, x]) \mid C_i \in M, s < x, f([s, x]) \subset C_i, f(s) \in \partial C_i\}$. $\bar{D}(S_n)$ exists that is greater than 0 and we can compute a positive constant $\varepsilon = min(k, \bar{D}(S_n))$.

Since the formulation rule of S_i, $\forall i$, $l([S_i, S_{i+1}]) \geq \varepsilon$. For small Δs it is not possible for $\frac{||\Delta f(s)||}{\Delta s}$ to increase continuously because the path is feasible. $\frac{||\Delta f(s)||}{\Delta s}$ is bounded from above thus, there exists a small positive constant δ satisfying

$$\forall i,\ S_{i+1} - S_i \geq \delta. \tag{9}$$

Due to Archimedean property, the sequence $\mathbb{S}$ is finite, $\mathbb{S} = \{S_1, \dots, S_n\}$ with $S_1 = 0, S_n = 1$. If we construct $\tilde{\mathbb{C}} = \{\tilde{C}_1, \dots, \tilde{C}_{n-1}\}$ for each S_i from the above rule, because of $f(S_{i+1})$, $\tilde{C}_{i+1} \cap \tilde{C}_i \neq \phi$. However, to guarantee non-zero volume overlapping space, the following elements are combined.

$$\bar{C}_i \in M \text{ s.t. } N_\epsilon(f(S_{i+1})) \subseteq \bar{C}_i \tag{10}$$

We can construct a sequence $\mathbb{C}$ as

$$\mathbb{C} = \{\tilde{C}_1, \bar{C}_1, \dots, \tilde{C}_{n-1}, \bar{C}_{n-1}\}, \tag{11}$$

whenever the feasible path is given. Therefore, the equivalence between the existence of feasible path and the existence of MOCS path is shown.

3. ALGORITHMIC IMPLEMENTATION

As the equivalence is confirmed, creating every possible MOCS in the free space is the next step. To construct MOCS, we use dynamic programming method to implement an algorithm. We divide a problem into small solvable problems and merge it together at last. This method enables to deal with additional obstacle inputs thus, possible to use for local planner.

After all MOCS are built, connecting every overlapping MOCS represents the free space as a simplified undirected graph. Reasonable function is adopted for an edge cost function. This graph producing operation is only conducted once because the returned shortest cost path itself is a SFC. Dijkstra algorithm is used to find the shortest cost MOCS path. Receiving the initial MOCS path, we can proceed optimization process.

3.1 MOCS construction

Construction of MOCS using dynamic programming method is described in Fig. 3 and Alg. 1. In Fig. 3, 2D is assumed for convenient explanation. At first, one obstacle input initializes the algorithm, and pertinent MOCSs are built (line 2-4). For obstacles afterward, the algorithm checks overlapping between the obstacle and previously constructed MOCSs. To check overlapping, following inequalities are used referring to a rectangular environment.

$$|a| < \frac{x_1 + x_2}{2},\ |b| < \frac{y_1 + y_2}{2},\ |c| < \frac{z_1 + z_2}{2}, \tag{12}$$

where x_i, y_i, z_i are x, y, z direction lengths of two subjects, and a, b, c are x, y, z difference between two midpoint coordinates, respectively. We only consider MOCSs that overlap with new obstacle. If $MOCS_j$ and obstacle overlap, erase $MOCS_j$ from the previous set. Avoiding the obstacle, segment $MOCS_j$ into MOCSs regarding $MOCS_j$ as the whole space. Then, expand those MOCSs through x, y or z axis to formulate MOCS in the original free space and merge them to the previous set (line 7-12). Proceeding the process with every obstacle inductively, every MOCS in the free space is generated. Duplication checking process is followed subsequently (line 15-17).

The problem is solved by dealing with small problems repeatedly. This method enables the algorithm to be applicable for obstacles that are not just cuboid because segmenting complicated obstacles into small cuboids makes no difference. Thus, if all angles are rectangular, we can implement the algorithm even for obstacles that shape changes along their height or terraced obstacles.

Algorithm 1 MOCS construction

```
1:  for all $O_i \in \mathbb{O}$ do
2:      if $i = 0$ then
3:          Make MOCS in the free space;
4:          $M \leftarrow \{MOCS\}$;
5:          continue
6:      end if
7:      for all $MOCS_j \in M$ do
8:          if Overlapping($O_i$, $MOCS_j$) then
9:              $M \leftarrow M \setminus \{MOCS_j\}$;
10:             Divide $MOCS_j$ into $MOCS_j$ MOCS;
11:             Expand to formulate free space MOCS;
12:             $M \leftarrow M \cup \{MOCS\}$;
13:         end if
14:     end for
15:     for all $MOCS_j$, $MOCS_k \in M$ $(j \neq k)$ do
16:         if Duplication($MOCS_j$,$MOCS_k$) then
17:             $M \leftarrow M \setminus \{MOCS_k\}$;
18:         end if
19:     end for
20: end for
```

3.2 Graph construction

Graph denotes the free space by regarding MOCSs as nodes and overlappings as edges. Graph construction is explained in Alg 2. After every MOCS is produced, the algorithm continues to check whether edges are allowed. The same inequality Eq. (12) is used to verify overlapping between two MOCSs. If $MOCS_i$ and $MOCS_j$ overlap, following cost function defines the edge length

$$E = (\frac{1}{a^2} + \frac{1}{b^2} + \frac{1}{c^2})(||\boldsymbol{x}_i - \boldsymbol{x}_k|| + ||\boldsymbol{x}_k - \boldsymbol{x}_j||), \tag{13}$$

where a, b, c are x, y, z direction lengths of overlapping space and $\boldsymbol{x}_i$, $\boldsymbol{x}_j$, $\boldsymbol{x}_k$ are midpoint coordinates of $MOCS_i$, $MOCS_j$, overlapping space, respectively (line 1-4). The cost function can vary according to the objective of planning.

Suggested cost function assigns large cost where the overlapping space is tight and small cost where the overlapping space is spacious. This way, dijkstra algorithm can return spacious path that guarantees safety indirectly, also considering short distance.

Algorithm 2 Graph construction

```
1:  for all $MOCS_i$, $MOCS_j \in M$ $(i \neq j)$ do
2:      if Overlapping($MOCS_i$, $MOCS_j$) then
3:          $V \leftarrow V \cup \{(MOCS_i, MOCS_j)\}$;
4:          $E \leftarrow E \cup \{edge\}$;
5:      end if
6:  end for
7:  return $G = (V, E)$;
```

4. EXPERIMENT

4.1 System setup

In order to test the algorithm, we implement it in C++ and use Gazebo simulator with RViz. Computation time

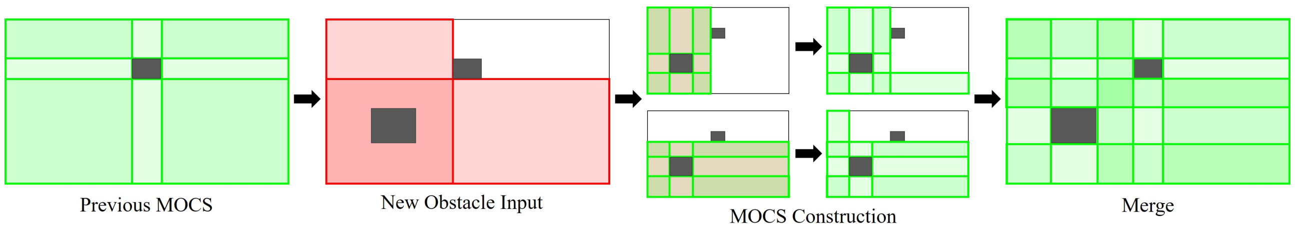

Fig. 3 Dynamic programming method computing every MOCS in cluttered environment. Red MOCSs overlapping with new obstacle input are only considered (New Obstacle Input). Red MOCSs are segmented into MOCSs regarding red MOCS as the whole space. Expansion through x, y, z axis makes MOCS (green) in the original workspace (MOCS Construction). Newly generated MOCSs are merged (Merge).

is evaluated on a laptop with Ubuntu 16.04. Intel Core i5-8250U @ 1.60GHz CPU, 8GB RAM. Quadrotor is modeled as a cube of each edge length 0.55 m referring to 3DR IRIS quadrotor. For convenience, we assume the quadrotor as a point and obstacles 0.55 m larger each edge.

We have built environments that could clarify its application in 3D. Simulation environments are presented in Fig. 4. Map size is 6 m $\times$ 6 m $\times$ 6 m with obstacles constructed in 0.01 m resolution. Grids appearing on the simulation floor are 1 m length each and quadrotor is initialized at (0.5, 0.5, 1). Environments are composed of simple cuboid obstacles with 4 m height and complicated shape obstacle that segmentation is required, respectively.

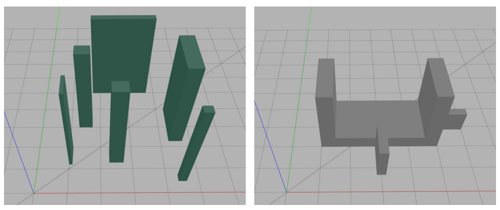

Fig. 4 Simulation environment. Left: simple cuboid environment. Right: environment with obstacle that is not cuboid.

4.2 Trajectory optimization

After initial MOCS path computation, we initiate trajectory optimization. We use unconstrained quadratic program (QP) [8] for 7^{th} order polynomial trajectory jerk optimization and set midpoints of overlapping spaces as waypoints. Furthermore, additional waypoints along the straight line between waypoints are selected for collision avoidance [8]. To follow the desired trajectory, we use geometric controller [9].

The main point of this study is the front end. Trajectory optimization method and controller are selected to check the feasibility of the algorithm. Thus, any optimization method that utilizes SFC is implementable.

4.3 Flight simulation

We apply the algorithm in given simulation environments. For simple cuboid environment, total 20 MOCSs are generated and no collision occurred during the flight. It took less than 10 ms to create every MOCS in the workspace and returned optimized trajectory within 100 ms. For obstacle that is not cuboid, total 14 MOCSs are generated. In this case, we limit the z axis under the maximum height of the obstacle because spacious MOCS located in the sky makes it difficult to ensure the feasibility of the algorithm. No collision occurred similarly and generated trajectory within 100 ms. Computed trajectory is described in Fig. 5 using RViz and Octomap library [10] to represent 3D occupancy map.

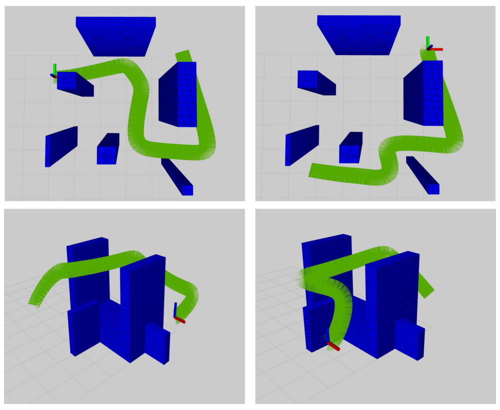

Fig. 5 Flight simulation. Up: trajectory (green) in simple cuboid environment. Down: trajectory (green) in environment with obstacle that is not cuboid.

4.4 Computation time comparison

Among previous studies using SFC, we compare the proposed method with Gao's method [1]. Although planning framework is different, it is possible to analyze which method computes more efficiently. Between fast marching method and A* algorithm, we select A* algorithm for Gao's initial path computation method. The proposed method and Gao's method are tested in the environment as Fig. 6. Map size is 6 m $\times$ 6 m $\times$ 6 m and obstacles are constructed in 0.01 m resolution. However, 2.16×10^8 grids are needed for A* algorithm and the algorithm does not work. For Gao's method, we set the map resolution to 0.1 m.

We evaluate the computation time by setting goal points (0.5, 0.5, 1) $\rightarrow$ (5, 5, 1) $\rightarrow$ (0.5, 0.5, 1) repeatedly. Computation time is shown in Table 1. MOCS construction is conducted once but even we sum all the computation time, our proposed method returns trajectory about 1.7 times faster. Also, Gao's method does not represent the workspace accurately because of the resolution prob-

Table 1 Comparison of average trajectory computation time in 1 trial (30 trials average)

Proposed method	MOCS Construction (once)	Dijkstra	Optimization	Total
Time (ms)	8.24	0.03	44.22	52.49
Gao's method (0.1 m resolution) [1]	A*	SFC Construction	Optimization	Total
Time (ms)	1.30	74.14	15.08	90.52

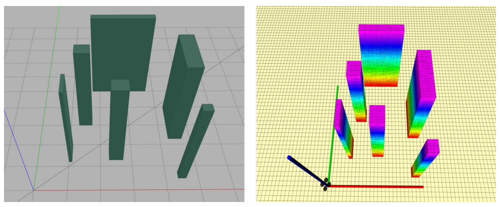
Fig. 6 Comparison simulation environment. Left: proposed method. Right: Gao's method.

lem. We can realize that our optimization method is quite slow but we presume that it can be easily improved by adopting faster QP solver or other optimization methods.

Comparing the front end, the proposed method spends about 60 ms less than Gao's method. We use dynamic programming method to construct MOCS graph but Gao's method generate SFC by expanding grids until they reach obstacles. Moreover, MOCS graph is created during initial trial thus, for multiple trials of motion planning, the proposed method is more advantageous.

5. CONCLUSION

5.1 Contribution

Efficient SFC generation is important for motion planning. Changing the framework can improve the total process. In this work, we prebuild SFC with dynamic programming method by suggesting a new definition called MOCS. The completeness of the algorithm in the rectangular worlds is confirmed by Th. 1. Geometric relation and cost function are used to represent the free space. Jerk minimized trajectories with geometric controller are returned for safe flight. Through the simulation, we checked that no collision occurred and every expected MOCS were created. Compared to a previous algorithm that uses SFC, our framework computes the process more efficiently with no discrepancy of the map.

5.2 Future work

The algorithm tested in the simulation has computation complexity $O(n^3)$. It depends on the number of obstacles thus, it took less than 10 ms to generate MOCS graph. However, searching every MOCS for overlapping with new obstacle can be improved by implementing bounding volume hierarchy. Through this, we presume that algorithm computation complexity can be reduced to $O(n^2 logn)$. Then, for local planning, the complexity scales down to $O(nlogn)$ which makes it more applicable.

More diverse scenarios are to be tested and implementation for local planner will be our future work. Furthermore, other optimization process that shortens our back end procedure are planned to be adopted.

6. ACKNOWLEDGEMENT

This research was supported by a grant to Bio-Mimetic Robot Research Center Funded by Defense Acquisition Program Administration, and by Agency for Defense Development (UD190018ID).

REFERENCES

[1] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 344–351.

[2] R. Deits and R. Tedrake, "Efficient mixed-integer planning for uavs in cluttered environments," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 42–49.

[3] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1476–1483.

[4] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.

[5] J. Park and H. J. Kim, "Fast trajectory planning for multiple quadrotors using relative safe flight corridor," *arXiv preprint arXiv:1909.02896*, 2019.

[6] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic foundations of robotics XI*. Springer, 2015, pp. 109–124.

[7] J. Chen, K. Su, and S. Shen, "Real-time safe trajectory generation for quadrotor flight in cluttered environments," in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2015, pp. 1678–1685.

[8] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.

[9] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 5420–5425.

[10] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “Octomap: An efficient probabilistic 3d mapping framework based on octrees,” *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.