Real-Time Constrained Trajectory Optimization for Quadrotors via Hierarchical RRT and MPC

RO47005 Planning & Decision Making

Gonçalo Fernández-Nespral Vaz
Carlota Alvear Llorente
María Sanz Piña
Tomás Leggeat Bargueño

Abstract— **This work presents a hierarchical motion planning framework for quadrotor navigation in dynamic indoor environments. The architecture combines a global RRT planner with a local MPC to generate feasible, collision-free trajectories in real-time. Simulation results demonstrate the system's robustness, achieving a 10-fold improvement in tracking precision over a PID baseline and an average solver time of 7.48 ms, ensuring safe operation around static and moving obstacles.**

I. INTRODUCTION

Autonomous navigation of aerial robots in indoor environments is an active research topic in robotics, with applications ranging from inspection and monitoring to service robotics. Indoor navigation poses significant challenges due to limited space, complex obstacle layouts, and the presence of dynamic elements such as people. Ensuring safe and reliable operation in such environments requires motion planning algorithms that can generate collision-free trajectories while respecting the system dynamics and operating in real time.

To address this, modern motion planning often adopts a hierarchical approach. While sampling-based algorithms like Rapidly-exploring Random Trees (RRT) are standard for efficiently finding global paths through complex static mazes, they lack the reactivity needed for moving obstacles. Therefore, recent literature pairs global planners with Model Predictive Control (MPC) [1], which optimizes trajectories locally to handle robot dynamics and avoid collisions. In this work, we implement this combined framework to enable safe navigation in a mixed static-dynamic environment.

II. PROBLEM DEFINITION

A. Problem statement

This work focuses on the motion planning of a quadrotor drone operating in an indoor environment. The system consists of a global planner and a local planner. A global planner based on Rapidly-exploring Random Trees (RRT) is used for computing the collision-free reference path from the start to the given goal. Model Predictive Control (MPC) is used as the local planner to generate the feasible trajectories updated continuously during execution.

The environment consists of both static and dynamic obstacles. The positions of the static obstacles are known, while dynamic obstacles are modeled with constant speed along predefined paths. The objective of the motion planning system is to generate collision-free trajectories that will allow the quadrotor to safely navigate the environment while taking into account both static and moving obstacles.

B. Robot Model

The quadrotor is modeled as a rigid body with six degrees of freedom. To reduce complexity, the model is linearized around a hovering equilibrium condition ($u_1 \sim mg, \theta \sim 0, \phi \sim 0, \psi \sim \psi_0$) using a first-order Taylor expansion. This results in the following equations for the linear dynamics:

$$\begin{aligned} \ddot{r_1} &= \ddot{x} = g(\theta\cos\psi + \phi\sin\psi) \\ \ddot{r_2} &= \ddot{y} = g(\theta\sin\psi - \phi\cos\psi) \\ \ddot{r_3} &= \ddot{z} = -g + \frac{u_1}{m} \end{aligned} \tag{1}$$

And the angular dynamics:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \mathbf{I}^{-1} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{2}$$

Where u_1 represents the total thrust magnitude, and u_2, u_3, u_4 denote the control moments (roll, pitch, and yaw torques).

C. Configuration space

The quadrotor is a 6-DOF underactuated system whose full state resides in the Special Euclidean group $SE(3)$. However, the system is *differentially flat*, allowing the full state to be recovered from a reduced amount of outputs. These are the position of the center of mass $\mathbf{p} = [x, y, z]^T$ and the heading angle (yaw) ψ.

Consequently, the effective configuration space for the drone is defined as:

$$\mathcal{C} = \mathbb{R}^3 \times S^1 \tag{3}$$

where each configuration is denoted as $\mathbf{q} = [x, y, z, \psi]^T$.

D. Environment description

The drone operates in an indoor corridor-like environment simulated in PyBullet. The workspace consists of two parallel walls and a ground plane, where all the obstacles are placed. The drone operates entirely within this corridor.

Multiple static obstacles representing tables are placed along the corridor. These tables are modeled using a URDF description and are fixed to the ground plane.

Dynamic obstacles represent possible students crossing the corridor. They are modeled as cylindrical bodies moving back and forth between two predefined waypoints. Their motion is constrained to the horizontal plane and follows deterministic trajectories along the corridor width.

III. MOTION PLANNING METHOD

A. Global Planner

The global planner is responsible for computing collision-free paths for the drone to navigate from its initial position to the designated goal within its environment. It is executed once, at the beginning of the trajectory.

Two different path planning methods have been implemented: Probabilistic Roadmaps (PRM) with A* search, and Rapidly-exploring Random Trees (RRT). These two were compared throughout development, with the final implementation making use of only RRT.

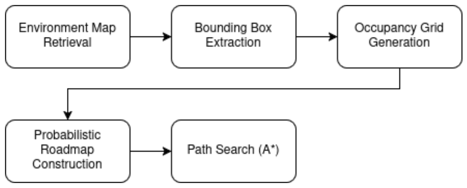

Environment Map Retrieval
Bounding Box Extraction
Occupancy Grid Generation
Probabilistic Roadmap Construction
Path Search (A*)

Fig. 1. Flow graph overview of A* search with PRM.

The PRM with A search* approach, outlined in Figure 1, begins by retrieving the environment state from the Pybullet simulation engine and computes axis-aligned bounding boxes (AABBs) for all obstacles. This data is used to generate a 3D occupancy grid of a set resolution. The PRM is then constructed by sampling random points in free space to serve as nodes, with edges connecting nodes if they are within a distance δ_{PRM}, and if the straight-line path between them is collision-free. Once the graph is established, the A* algorithm executes a search to identify the optimal path, using Euclidian distance to the goal as the search heuristic.

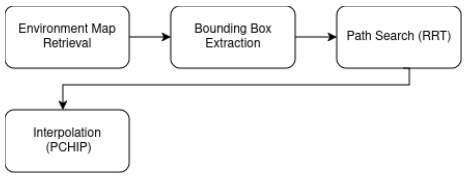


Fig. 2. Flow graph overview of RRT.

The RRT method constructs a path incrementally by growing a tree, as shown in Figure 2. After the tree has been initialized at the initial drone position, the algorithm iteratively samples random points, and extends the nearest existing node toward that sample by a fixed distance δ_{RRT}. Each new sampled node is verified against the bounding boxes extracted from the map, to check for collisions. The cycle repeats until a node reaches the goal (within a specific tolerance), at which point the graph is rerouted. A final post-processing smoothing step is applied to the generated path, utilizing Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) [3]. This interpolation strategy maintains the average shape of the path while completely preventing overshoot in significantly jagged regions (as opposed to traditional spline-based methods).

The two algorithms show distinct performance characteristics (compared visually in Figure 3), making each suitable for different situations. During experimentation, RRT demonstrated significantly lower computational requirements. However, the A* approach consistently yielded high quality paths due to its optimality, while the raw RRT paths tend to be more jagged due to their randomized nature. However, the post-processing interpolation step smooths the paths sufficiently. Thus, RRT's superior computational time makes it the preferred choice for this project.

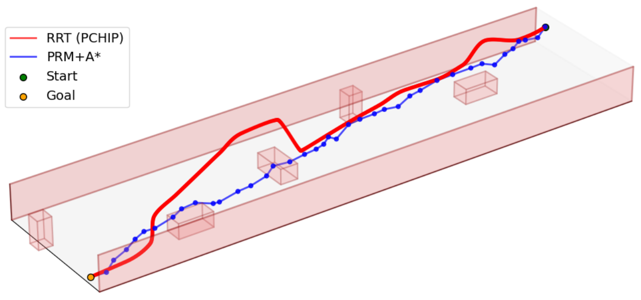


Fig. 3. Example comparison between paths generated with the two strategies.

This comparison contrasts with more detailed performance benchmarks found in literature. Notably, a study by Zammit and van Kampen [4] suggests that A* may be the superior candidate for real time 3D path planning applications. In their analysis, A* consistently outperformed RRT in both optimality and computational speed, due to the convergence pressure applied by a well designed search heuristic. Furthermore, RRT's success rate dropped significantly in more complex environments, while A* maintained its performance.

Overall, the discrepancy between these results and the observed performance of the current implementation suggest that further tuning must be performed on the RRT implementation used for this project. Specifically, parameters such as the occupancy grid resolution, the PRM node threshold δ_{PRM} and the RRT distance δ_{RRT} play a large role in the performance of these planners.

B. Local Planner

The local planner is responsible for following the trajectory specified by the global planner while dynamically avoiding obstacles. It is designed as a cascaded hierarchy, separating the slow translational dynamics from the fast rotational dynamics. As illustrated in Figure 4, the system consists of two primary loops.

First, a high-level Position Controller (MPC) operates at a lower frequency. It receives the reference trajectory from

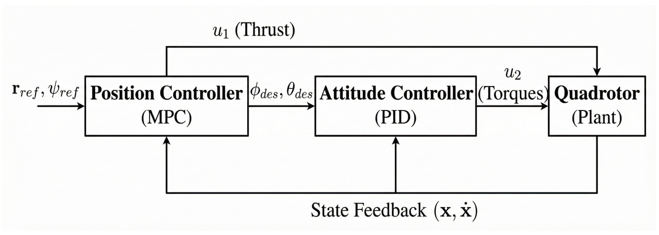

u_1 (Thrust)
$\mathbf{r}_{ref}, \psi_{ref}$
Position Controller (MPC)
ϕ_{des}, θ_{des}
Attitude Controller (PID)
u_2 (Torques)
Quadrotor (Plant)
State Feedback $(\mathbf{x}, \dot{\mathbf{x}})$

Fig. 4. Hierarchical control architecture.

the global planner and optimizes the state of the drone to compute the required total thrust (u_1) and the desired attitude angles (ϕ_{des},θ_{des}). Second, a low-level Attitude Controller (PID) operates at a higher frequency. This controller takes the desired angles from the MPC and generates the necessary body torques ($\mathbf{u_2}$) to stabilize the drone's orientation. This decoupling allows the MPC to focus on trajectory feasibility and obstacle avoidance without the computational burden of resolving high-speed angular dynamics.

To implement the position controller, the linearized dynamics equations (1) from subsection II-B are discretized with the Forward Euler Method. The resulting state-space model is as follows:

$$\mathbf{x}_{k+1} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ z_k \\ \dot{x}_k \\ \dot{y}_k \\ \dot{z}_k \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & g\,\Delta t \\ 0 & -g\,\Delta t & 0 \\ \Delta t/m & 0 & 0 \end{bmatrix} \begin{bmatrix} u_{1,k} \\ \phi_k \\ \theta_k \end{bmatrix} \tag{4}$$

Here, the roll (ϕ) and pitch (θ) angles are treated as direct control inputs for the translational dynamics. It is important to note that this model assumes the drone's heading is aligned with the global frame ($\psi \approx 0$). The resulting optimal control inputs are then rotated by the measured yaw angle ψ to transform them into the body frame before being executed by the attitude controller.

The MPC framework utilizes this discrete model to predict the quadrotor's future states over a finite horizon $N = 48$. At each time step k, the controller solves a Constrained Quadratic Programming (QP) problem to determine the optimal control sequence over the receding horizon that minimizes trajectory error and control effort. The optimization objective is defined by the following cost function J:

$$J = \sum_{i=0}^{N-1} \left(\|\mathbf{x}_{k+i} - \mathbf{x}_{ref,k+i}\|_Q^2 + \|\mathbf{u}_{k+i}\|_R^2 \right) + \|\mathbf{x}_{k+N} - \mathbf{x}_{ref,k+N}\|_P^2 \tag{5}$$

where $\mathbf{x}_{ref}$ represents the reference trajectory. The matrices Q and R represent the running costs, penalizing state deviation and control effort respectively. The matrix P represents the terminal cost, which helps with stability.

Finally, obstacle avoidance is incorporated into the optimization problem via linear inequality constraints. To prevent solver infeasibility in cluttered environments, these are implemented as soft constraints using slack variables. For each obstacle i, a separating hyperplane is defined by a normal vector $\mathbf{n}_i$ and a boundary scalar b_i. The constraint at step k is formulated as:

$$\mathbf{n}_i^T \mathbf{p}_k \geq b_{i,k} - s_{i,k}, \quad s_{i,k} \geq 0 \tag{6}$$

where $\mathbf{p}_k$ is the drone's position and $s_{i,k}$ is the slack variable.

IV. RESULTS

A. Experimental Setup

The experimental setup is based on the indoor corridor described in subsection II-D. The environment includes three static obstacles representing tables and two dynamic obstacles representing students crossing the corridor, as shown in Figure 5.

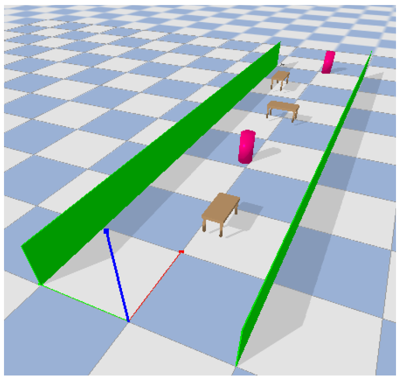


Fig. 5. Indoor corridor environment used in the simulation.

The quadrotor is initialized at the beginning of the corridor, while the goal position is located at the opposite end. The RRT-based global planner computes a collision-free path from the start to the goal once at the beginning of each experiment. During execution, the MPC-based local planner generates the trajectories followed by the drone.

Each experiment consists of a full simulation run in which the drone navigates from the initial position to the goal while avoiding both static and dynamic obstacles. If the drone reaches the goal, a collision occurs or the predefined time limit is exceeded, the experiment is terminated. Through each run, performance metrics are recorder for post-evaluation of the method.

B. Performance Metrics

To objectively evaluate the performance of the proposed MPC framework, the system's behavior has been analyzed using four quantitative indicators. These metrics assess the controller's precision, efficiency, computational feasibility, and safety.

The analysis of the RRT global planner was not carried out. Given the strict report length limitations, the evaluation is concentrated exclusively on the local MPC controller, as it constitutes the main focus of this work.

1) Tracking Error: It quantifies the accuracy with which the drone follows the reference trajectory. To summarize the performance over the entire trajectory of N steps, the Mean Squared Error (MSE) and the Root Mean Squared Error (RMSE) of the Euclidean position error are utilized:

$$MSE = \frac{1}{N}\sum_{k=1}^{N} ||\mathbf{p}_{ref,k} - \mathbf{p}_{real,k}||^2 \tag{7}$$

$$RMSE = \sqrt{MSE} \tag{8}$$

In the implementation, the MPC demonstrates exceptional tracking capability and low position error, most of which can be explained by the deviations needed to avoid the dynamical obstacles from the implementation.

- **RMSE:** 0.0566 m
- **MSE:** 0.0032 m^2

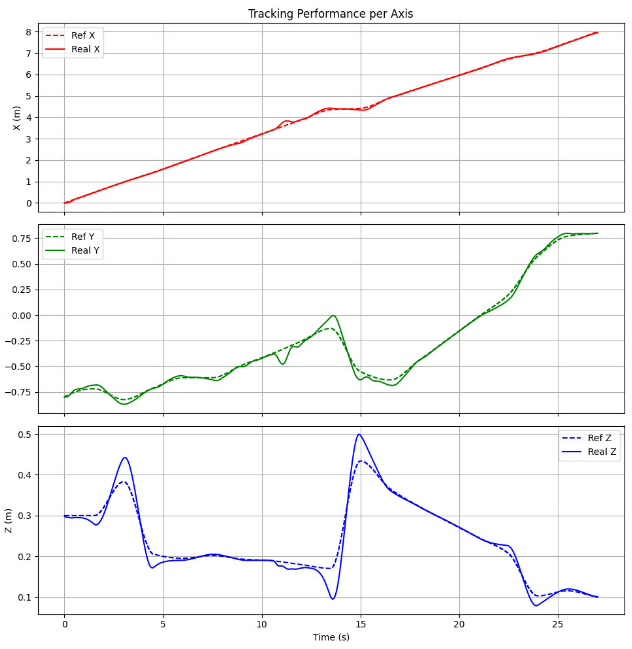


Fig. 6. Tracking performance of each axis.

2) Cost of Control Input and Smoothness: The control effort is calculated as the sum of the squared motor angular velocities (RPM) over the simulation. Additionally, the control smoothness is also analyzed, defined as the mean absolute difference between consecutive control commands (Roll, Pitch, and Thrust).

- **Total Control Effort:** 9.09×10^{11} RPM^2. This value is dominated by the thrust required to counteract gravity (mg).
- **Smoothness Score:** 0.01643. This low value indicates fewer oscillations in the control signal, which reduces motor heating and extends actuator life.

3) Computation Time: For Model Predictive Control, real-time feasibility is the critical bottleneck. The solver must find the optimal solution within the control period Δt.

With a control frequency set to 48 Hz, the available time budget is approximately 20.8 ms.

- **Average Solve Time:** 7.48 ms.

The controller utilizes less than 50% of the available CPU time per cycle, confirming that the solution is real-time feasible with a safe margin for handling unexpected disturbances.

4) Number of Collisions: A collision is registered whenever the physical boundaries of the drone intersect with an obstacle or the ground plane (excluding takeoff/landing zones).

- **Collisions:** 0.

The drone successfully adheres to the trajectory without strictly violating safety constraints.

C. Method Comparison

To assess the tracking capabilities and the feedforward anticipation of the proposed MPC versus the baseline PID implementation of the repository, a periodic Lissajous curve is employed. This is a standard trajectory in robotics, used as a benchmark for performance [2].

TABLE I
PERFORMANCE COMPARISON: PID VS. MPC ON FIGURE-8 TRAJECTORY

Metric	**PID (Baseline)**	**MPC (Proposed)**	**Gain**
Tracking RMSE [m]	0.4296	**0.0417**	$\approx 10\times$
Tracking MSE [m^2]	0.1845	**0.0017**	$\approx 100\times$
Smoothness (Score)	0.0037	**0.0028**	24%
Computation Time [ms]	**0.27**	10.17	–
Control Effort [RPM^2]	1.21×10^{12}	1.21×10^{12}	Similar

The quantitative superiority of the MPC is visually corroborated by the trajectory plots presented in Figure 7. As observed in Figure 7b, the baseline PID controller exhibits a significant phase lag, systematically ”cutting corners”. This behavior is characteristic of reactive control schemes, which require a non-zero error to generate corrective thrust.

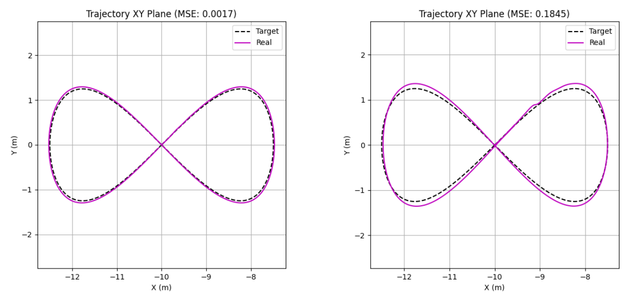
In contrast, the proposed MPC implementation (Figure 7a) virtually overlaps with the reference path. By incorporating the reference velocity vector into the optimization problem, the MPC anticipates the required acceleration changes before entering the curves, effectively eliminating the tracking lag.

Although the computational cost is higher than the PID, the average solve time of 10.17 ms proves that the predictive solver is fully real-time feasible within the 48 Hz control loop, justifying the computational overhead for the $10\times$ gain in precision.

V. DISCUSSION AND CONCLUSION

A. Optimality and Completeness

The proposed motion planning framework does not guarantee global optimality. The global planner is based on RRT, which is a sampling-based method that is probabilistically complete but not optimal. Furthermore, the local planner relies on MPC, which optimizes the control inputs over a



(a) MPC (b) PID

Fig. 7. Trajectory comparison (XY View).

finite prediction horizon. Global optimality would only be guaranteed in the limit of an infinite horizon, which is not feasible in real-time applications. Despite the lack of formal optimality guarantees, the obtained results show that the generated trajectories are of high quality in practice. The low tracking error, smooth control inputs, and absence of collisions indicate that the suboptimal solutions produced by the proposed approach are sufficient for safe and effective navigation in the considered environment.

Regarding completeness, the global planner is probabilistically complete, meaning that a feasible path will be found with high probability given sufficient computation time. However, the overall system is not deterministically complete. The use of a finite-horizon MPC and the presence of dynamic obstacles may result in situations where no feasible local solution exists, even if a global path is available.

In practice, the controller remained feasible throughout all experiments and no collisions were observed, indicating that the combined RRT-based global planner and MPC-based local planner provides a robust solution for the considered indoor navigation task.

B. Complexity

The implemented MPC is formulated as a convex Quadratic Program (QP). By utilizing the OSQP solver, the computational complexity scales linearly with the prediction horizon, $\mathcal{O}(N)$, rather than cubically [6]. To further improve computational cost, *parametrized programming* [7] is used by creating the MPC problem once at the initial phase of the simulation instead of initializing the problem in each iteration.

On the other hand, according to the asymptotic analysis provided by Karaman and Frazzoli [8], the theoretical time complexity of the RRT algorithm is $\mathcal{O}(N \log N)$ regarding the number of samples N.

C. Limitations and Recommendations

The results presented in this report demonstrate the effectiveness of the joint global-local planning architecture in navigating indoor environments with obstacles. However, the performance of the quadrotor remains constrained by specific modeling simplifications. RRT was selected as the global planner due to its lower computational requirements during development, allowing for fast initial path generation. This contrasts with the A^* approach, which consistently yielded smoother, optimal paths at a higher computational cost. The performance of these planners is highly sensitive to manually tuned parameters, such as the occupancy grid resolution and the RRT expansion distance δ_{RRT}. While RRT was prioritized for speed, literature suggests that A^* can outperform it in both speed and optimality for 3D planning [4], indicating that the system would benefit from a detailed sensitivity analysis.

Furthermore, the global planner's use of axis-aligned bounding boxes causes paths to be over-conservative, as the entire object is treated as a single block. Local trajectory generation is managed by the MPC, which uses linearization to maintain real-time feasibility within the $20.8\,\mathrm{ms}$ ($48\,\mathrm{Hz}$) control budget. This strategy is inherently limited for non-convex geometries, as imposing only a single linear constraint provides the controller with an inaccurate representation of the collision-free space. Additionally, the MPC does not fully optimize for the yaw degree of freedom, deriving heading primarily from the direction of movement. Dynamic obstacles are modeled as moving linearly at constant velocity, which fails to account for more complex, stochastic behaviors.

To address these limitations, several improvements are proposed. Implementing multiple linear constraints would allow the local planner to accurately represent non-convex geometries, while adopting A^* would improve path optimality and reduce computational workload. Applying the global planner iteratively would allow the drone to adapt to unexpected environmental changes, rather than relying on a static path execution. Furthermore, incorporating active yaw control would enable the system to anticipate upcoming curves. Finally, for improved safety in human-robot interaction, implementing a reactive layer such as Optimal Reciprocal Collision Avoidance (ORCA) [5] would account for the reciprocity in human agents, achieving more fluid and natural movement.

References

[1] R. V. Cowlagi and P. Tsiotras, ”Hierarchical motion planning with kinodynamic feasibility guarantees: Local trajectory planning via model predictive control”, *2012 IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, USA, 2012, pp. 1835–1840.

[2] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart, “Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System,” in *Robot Operating System (ROS): The Complete Reference (Volume 2)*, A. Koubaa, Ed. Cham: Springer International Publishing, 2017, pp. 3–39.

[3] F. N. Fritsch and J. Butland, “A Method for Constructing Local Monotone Piecewise Cubic Interpolants,” *SIAM Journal on Scientific and Statistical Computing*, vol. 5, no. 2, pp. 300–304, 1984.

[4] C. Zammit and E. van Kampen, “Comparison of A* and RRT in real-time 3D path planning of UAVs,” in *AIAA Scitech 2020 Forum*, Orlando, FL, 2020. doi: 10.2514/6.2020-0861.

[5] J. van den Berg, S. J. Guy, M. C. Lin, and D. Manocha, “Reciprocal n-body Collision Avoidance,” in *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2009.

[6] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, “OSQP: an operator splitting solver for quadratic programs,” *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, Feb. 2020.

[7] J. A. Goguen, ”Parameterized Programming”, *IEEE Transactions on Software Engineering*, vol. SE-10, no. 5, pp. 528-543, Sept. 1984, doi: 10.1109/TSE.1984.5010277.

[8] 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.