Learning High-Speed Flight in the Wild with Perception-Aware Trajectories

Jonathan Lee, Soham Bhave, Venkata Nagarjun Pudureddiyur Manivannan, Tony Tao
Course Project, 16-831 Introduction to Robot Learning, Spring 2024, CMU
{jlee6, snbhave, vpudured, longtao} @andrew.cmu.edu

1 Motivation

A significant advancement in autonomous navigation, particularly in flight, necessitates the ability of the agent to maneuver through dynamic and obstacle-laden environments safely, yet swiftly and agilely. This entails minimizing latency in planning and executing maneuvers to successfully navigate complex surroundings. Quadrotors are notably adept at achieving high velocity and acceleration while remaining nimble, making them valuable in various applications such as disaster relief, exploration of unknown terrain, and cinematography, while performing tasks like object detection, payload delivery, and mapping.

The paper by Loquercio et al. [1] proposes an end-to-end pipeline for enabling high-speed autonomous flight in challenging environments. Unlike traditional methods that rely on separate modules for sensing, planning, mapping, and control, this approach employs a single neural network with onboard sensing and computation. The methodology involves training a sensorimotor policy in simulation using privileged learning, where a sampling-based planner serves as the privileged expert, providing high-quality training data.

2 Problem Statement and Related Work

Loquercio et al. [1] implements a student policy as part of the pipeline to plan a trajectory and control the quadrotor for agile and fast flight. The policy predicts collision-free trajectories in a receding horizon based on sensor measurements and a goal direction, with a model predictive controller tracking the trajectory with the lowest collision probability. The input representation accounts for real-world sensor noise, with depth perception playing a crucial role in high-speed obstacle avoidance. Utilizing only depth data instead of the entire RGB-D space significantly reduces computational cost.

Furthermore, we propose to modify Loquercio et al. [1] to track a desired yaw angle reference and utilize a perception cost to align the quadrotor with obstacles and the unmapped space intersecting its trajectory. This approach aims to facilitate early detection of unseen obstacles, allowing the planner sufficient time to adapt to new obstacles while executing high-speed flight. We make our code available online[1].

A key insight for agile quadrotor navigation is that yaw angle can be expressed independently from the quadrotor's position reference trajectory. This was demonstrated by Mellinger and Kumar [2] who proved via the differential flatness property that a time parameterized reference trajectory of just position, $\mathbf{x} = [x, y, z]^T$, and yaw, ψ, is sufficient for trajectory tracking control. Decoupling the yaw angle has benefits to perception-aware quadrotor platforms with limited field of view (FOV) sensors which must actively point the sensor towards regions of interest in order to navigate through cluttered environments [3, 4, 5, 6]. Perception-aware trajectory planning can broadly be classified

[1]Our implementation is available in the feature branches of https://github.com/jonlee48/agile_autonomy

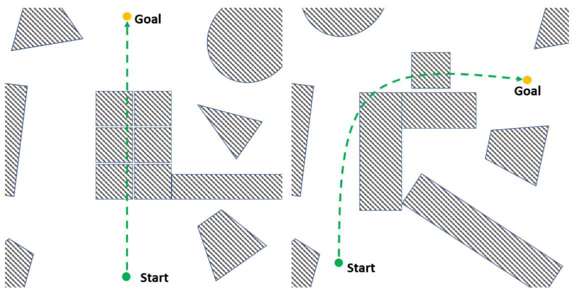
into approaches that seek to *exploit* visible features in the environment for state estimation, and approaches that actively *explore* unknown regions in the environment for future navigation.

In systems that use visual-inertial odometry (VIO) [5, 6], maintaining visual tracks of image features is critical for an accurate state estimate. This objective can be satisfied by orienting the quadrotor towards observed regions of high texture. While our learning baseline [1] does rely on VIO state estimates, the authors test in feature-rich outdoor environments such as forests and abandoned buildings, rather than featureless structures such as empty corridors. Additionally, the trajectory optimization in [6] is too slow to run online and the authors assume a known environment where visual features are pre-determined.

In contrast, our task is more similar to perception-aware techniques that actively explore unknown regions near the planned trajectory. In RAPTOR [3], yaw angle planning is decomposed into a graph search and then trajectory optimization step. The idea is to find a sequence of yaw angles along the position reference trajectory that maximize visibility of voxels from the occupancy map that are nearby the trajectory. This is equivalent to finding a minimum cost path in the graph of yaw angles that minimizes the total cost via Dijkstra's algorithm. Finally, a smooth polynomial (B-spline) is fit to the discrete set of yaw angles to form a continuous-time reference for the yaw angle. We choose to experiment with the yaw refinement module of [3] to compute a desired yaw angle and perception cost.

3 Our Idea

The student policy model currently only outputs positions, and during runtime, the yaw is set either as a constant or follows the instantaneous velocity direction. However, we hypothesize that adding a notion of perception awareness and using a yaw that encourages exploration of unknown spaces along the trajectory could help the quadrotor detect obstacles sooner and perform better overall. Figure [1] illustrates two specific situations where using a constant or velocity-based yaw is likely to lead to collisions, which can be avoided if a perception-aware yaw is used instead.



(a) (b)

Figure 1: (a) A large obstacle placed along the straight-line reference trajectory. (b) An obstacle placed right behind the corner.(Image Source: [3])

In order to incorporate perception awareness, we draw inspiration from the yaw-angle planning module in [3], where they select a yaw angle that maximizes the total information gain. The information gain per voxel is computed using an exponential decay function with respect to the distance of the voxel from the trajectory, as given by Equation 1.

$$I_g(\mathbf{v}) = e^{-wd_v} \tag{1}$$

In our approach, we expand the action space of the student policy to include heading information. This heading output is then supervised by using the yaw that maximizes information gain, as described above.

4 Methodology

4.1 Method Overview

A brief overview of [1] is provided before describing our modifications. To perform agile flight through previously unseen environments, a sensorimotor policy is trained using a teacher-student imitation learning framework shown in Figure 2.

(**A**) The privileged expert has access to the full environment point cloud, quadrotor state, and global reference trajectory. The expert generates a set of collision-free trajectories τ representing the desired state of the quadrotor $x_{des} \in \mathbb{R}^{13}$ over the next second, starting from the global reference trajectory τ_{gbl} computed using the approach of Liu et al. [7]. The Metropolis-Hastings algorithm is used to generate trajectories from a probability distribution around the global trajectory. A total of 50 thousand trajectories are sampled and scored using a discrete version of the trajectory cost function:

$$c(\tau, \tau_{gbl}, \mathcal{C}) = \int_0^1 \lambda C_{collision}(\tau(t)) + \int_0^1 [\tau(t) - \tau_{gbl}(t)]^\top \mathbf{Q} [\tau(t) - \tau_{gbl}(t)] dt \tag{2}$$

where $\lambda_c = 1000$, $\mathbf{Q}$ is a positive semi-definite state cost matrix, and $C_{collision}$ is a measure of the distance of the quadrotor to the points in the environment point cloud $\mathcal{C} \in \mathbb{R}^{n \times 3}$.

(**B**) The sensorimotor agent has access to only the current depth image, body rates, and desired direction and is trained to predict the next one second of the top 3 expert trajectories. Specifically, the model takes as input a depth image $d \in \mathbb{R}^{640 \times 480}$, the quadrotor's velocity $v \in \mathbb{R}^3$ and attitude $q \in \mathbb{R}^9$, and goal direction $\omega \in \mathbb{R}^3$. The network consists of two branches including a pre-trained MobileNet-V3 backbone and a four-layer perceptron to process the platform's current velocity, attitude, and goal direction. The model outputs the predicted positions $\tau_n^k \in \mathbb{R}^{10 \times 3}$ for the top $M = 3$ trajectories for the next one second.

(**C**) To attain the full quadrotor state during test time, the predicted positions $\tau_n^k \in \mathbb{R}^{10 \times 3}$ are projected on to 5th order B-spline polynomials for each axis independently. This representation allows for a smooth reference trajectory with continuous position, velocity, and acceleration to be tracked with a Model Predictive Controller (MPC) [8].

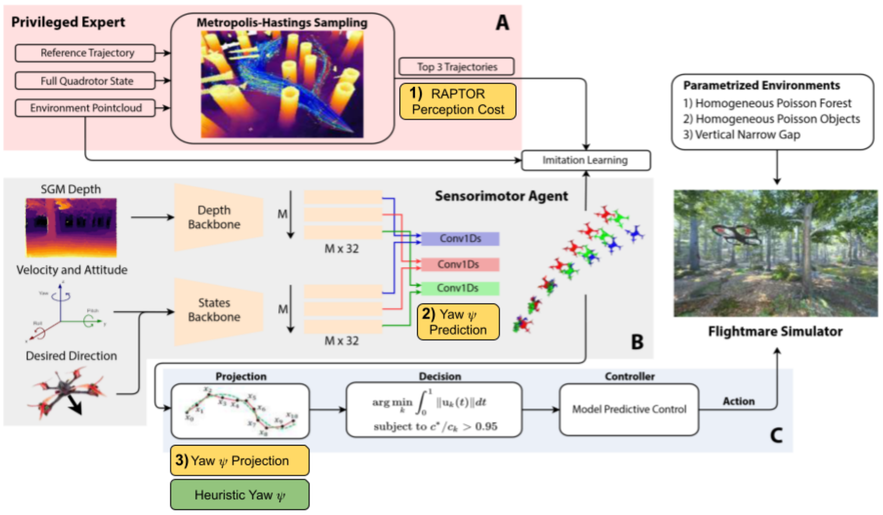


Figure 2: Method overview. Our first approach adds the green box to set a desired yaw based on a simple heuristic function. Our second approach is shown in yellow boxes with the following modifications: (**1**) a RAPTOR perception cost introduced in the expert, (**2**) a yaw prediction in the sensorimotor agent, and (**3**) projection of yaw onto the space of B-splines.

All training is performed in simulation using the Flightmare simulator [9] with the RotorS Gazebo plugin [10] and Unity rendering engine [11]. In order to provide realistic depth images and minimize the sim2real gap, the depth image is computed from a stereo camera using semi-global matching [12] instead of using the ground truth depth generated by Unity.

We experiment with two approaches to incorporating yaw angle planning into the teacher-student framework. Additionally, we ran experiments with and without velocity tracking yaw control enabled to better understand the role of yaw for the task.

1. **Heuristic yaw approximation**: Develop a heuristic to approximate a desirable yaw from the sensorimotor agent predicted positions. Described in Section 4.2.
2. **RAPTOR perception cost**: Modify the expert to compute the optimal yaw angle and agent network to predict the optimal yaw in addition to the position reference. Described in Section 4.3.

4.2 Heuristic Yaw Approximation

The baseline model implemented in [1] utilizes a fixed-value yaw angle throughout the flight of the quadrotor or performs a velocity tracking yaw. The velocity tracking yaw would orient the quadrotor along the velocity vector (indicated by the black arrow) in Figure 3.

To perform fast and agile motion, we hypothesize that it is necessary for the quadrotor to collect information about future way-points to give it enough time to respond and react to environment along its trajectory. Therefore, we employ a heuristic to set orient the yaw of the quadrotor to ***n*-look-ahead** along the trajectory. This would enable it to perceive the environment further along its motion and hence, avoid obstacles along its trajectory by performing timely and appropriate reactive decisions.

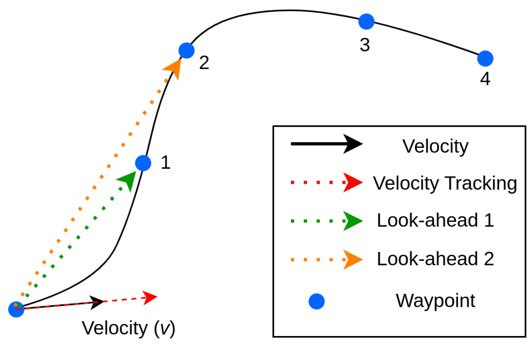


Figure 3: Possible Yaw Directions Based on N-Look-Ahead Heuristic

4.3 RAPTOR Perception Cost

The optimal yaw angle is computed from within the privileged expert (Figure 2A) by computing the information gain from sampled sensor observations and a volumetric map of the environment. In order to compute observed volumes from a particular sensor observation, we first convert the ground truth point cloud into a voxel grid with a resolution of 0.2 meters. Then for a given position and heading, we perform ray casting using the Amanatides and Woo Ray Casting algorithm [13] to obtain the free voxels in the field of view (FOV) of the depth image. For each voxel, information gain is then computed using Equation 1, and the total gain is computed as the sum of gains for each of these voxels.

To find the yaw that maximizes information gain Zhou et al. [3] employs a Dijkstra-based search algorithm over sampled waypoints in the trajectory. However, to speed up computation, we chose to sample 20 yaw angles at each point along the trajectory and select the yaw angle that maximizes information gain. In order to reduce computation time, we implemented a multi-threaded algorithm that computes the information gain for each view in parallel. Despite the parallelized implementation, runtime was on the order of minutes, limiting the expert to being run offline. Figure [4] visualizes the information gains computed for various voxels at various yaw configurations.

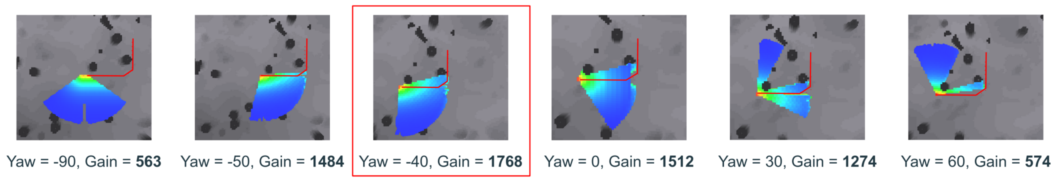


Figure 4: Different yaw configurations and corresponding information gains (Warmer colors indicate higher gains)

The computational expense of calculating the optimal yaw motivates our rational for the sensorimotor to learn an approximation of this function. We modify the sensorimotor agent to predict a yaw angle by changing the dimension of the last layer of the model, and reusing the pre-trained weights from the rest of the model (Figure 2B). The agent now predicts the next one second of the top $M = 3$ trajectories, $\tau_n^k \in \mathbb{R}^{10\times4}$, with each column vector corresponding to the positions x, y, z and yaw ψ at time $t = \frac{i}{10}$. Finally, a set of B-splines are fit to each axis of the predictions independently and tracked via MPC (Figure 2C).

5 Experiments

5.1 Baseline Experiments

The primary baseline we have chosen for this project is the results achieved by Loquercio et al. [1] in their paper presenting the end-to-end pipeline for high-speed autonomous flight. Their experiments are primarily formed by flight in two categories of environments: natural and human-made. Figure 5 shows images of a collection of different simulated environments tested by the researchers.

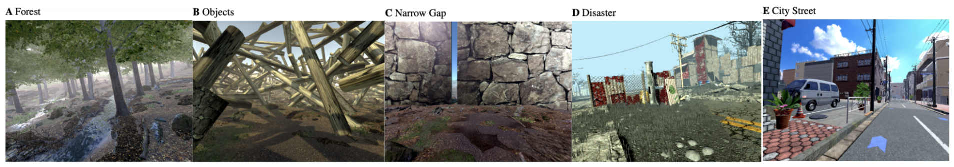


Figure 5: Images of simulated environments used by the researchers and that we plan on (Image Source: [1])

For each environment, the researchers chose to test flight speeds from the set: 3m/s, 5m/s, 7m/s, and 10m/s. The metric used to measure the performance of the algorithm was chosen to be the success rate over a number of trials. For our experiments, we use the forest environment with 7m/s as the main velocity.

In addition to replicating the results of the researchers as our baseline, we conducted the following experiments to explore the sensitivity of the success rate to different parameter changes:

1. **Velocity Tracking Yaw Control:** We ran experiments with and without velocity tracking yaw control enabled to better understand the role of yaw for the task. Velocity tracking yaw naively sets the yaw angle based on the direction of the velocity vector of the robot. When velocity tracking yaw is disabled, the robot is positioned with a fixed yaw angle which is initialized in the beginning.

2. **Fine-tuning:** Through fine-tuning, we aimed to improve on the results achieved by the checkpoint provided by the researchers so that our results are more comparable to those achieved in the original paper. We fine-tuned the model using both pre-collected object and forest datasets.
3. **Varying Obstacle Densities:** To analyze the effectiveness of perception awareness, we measure the Distance to Collision (D2C), which is the distance between the centroid of the quadrotor and the closest detected collision point. We expect D2C to indicate the level of "surprise", where a smaller D2C would indicate that the quadrotor was more surprised by an obstacle. With perception awareness, we can expect a reduced level of "surprise".

6 Results

6.1 Baselines

We were successfully able to replicate the results of the original paper as shown by Figure 6A. These results were achieved through testing using pre-trained weights. We also test the performance of the baseline model with a fixed yaw angle and environments with varying tree density.

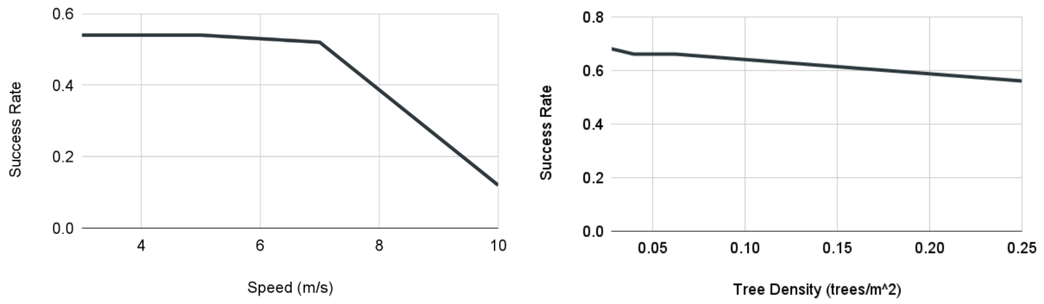


Figure 6: Success Rate of Baseline Model vs Speed (A) and Tree Density (B)

Table 1: Comparison of different Fine-tuning Approaches

Model	Average Number of Crashes	Average Closest Distance	Success Rate	Avg. Closest Dist. when Success
From Scratch	1.24	0.2449	0.32	0.5474
Pre-trained Checkpoint	0.62	0.4124	0.52	0.6755
Trained with Trees	0.92	0.3451	0.42	0.6628
Trained with Both Datasets	0.90	0.3245	0.38	0.6831

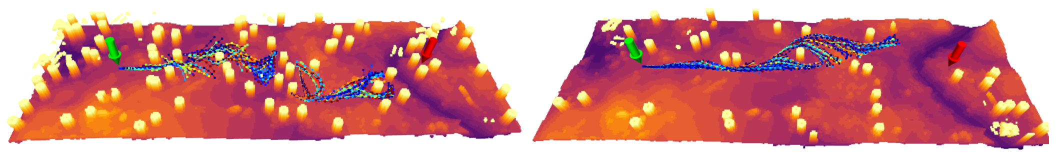


Figure 7: Example trajectory rollouts in random forest environments. The trajectories predicted by the model are indicated by the dotted lines. Starting and goal position are indicated by the green and red arrows.

6.2 Our Models vs Baselines

In our experiments we find that enabling velocity yaw control decreases success rate when compared to using a fixed yaw angle. As shown by Table 2, not only does having a fixed yaw outperform velocity tracking yaw in success rate, it also results in a larger average closest distance to obstacles and has a lower average number of crashes. We think that, in part, this could be due to the goal being in front of the robot, but believe that shows that naive velocity tracking often under-performs even on such simple tasks.

Table 2: Model Performance Comparison on 50 Random Forest Environments

Model	Average Number of Crashes	Average Closest Distance	Success Rate	Avg. Closest Dist. when Success
Baseline Velocity Tracking	0.62	0.4124	0.52	0.6755
Baseline Fixed Yaw	0.40	0.4639	0.66	0.6465
1-Look Ahead	0.52	0.4456	0.63	0.6319
2-Look Ahead	0.66	0.4508	0.60	0.6731
3-Look Ahead	0.54	0.4176	0.62	0.6128
4-Look Ahead	0.42	0.4628	0.62	0.6806
5-Look Ahead	0.62	0.3549	0.52	0.5689
RAPTOR Yaw	0.84	0.3818	0.46	0.6945

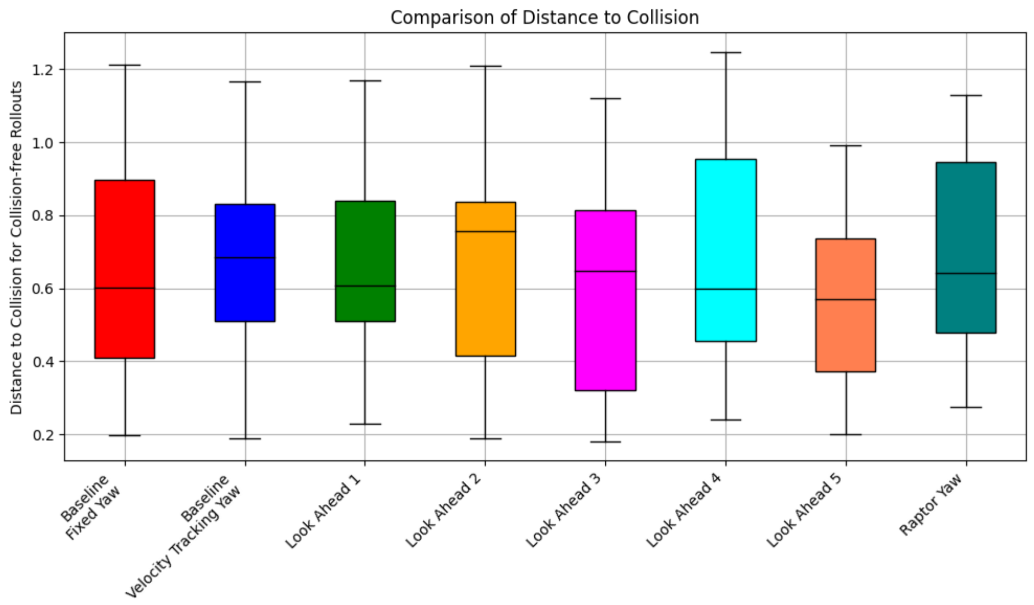


Figure 8: Comparing distance to collision metric for various yaw angle models in random forest environments for collision-free trajectories.

7 Discussion

As part of our efforts, we aimed to improve on two metrics: success rate and average closest distance to obstacles. We observed that none of the experiments with a different yaw were able to achieve a higher success rate than the baseline with zero yaw. We hypothesize that the reason for this is the data itself. We reused the dataset that the authors collected, which has depth images corresponding to a constant yaw. Thus, when we run the model with a different yaw there is a data distribution shift, which causes the model to make errors and encounter states which it has not seen before, potentially leading to a compounding effect and causing more crashes.

Additionally, we observed that we are able to achieve a higher median distance to collision with the 2 and 3-look ahead and RAPTOR yaw compared to the baseline fixed yaw. Even though the difference is relatively small, it validates our initial hypothesis that including yaw information that helps the quadcopter explore unknown spaces close to the trajectory aids in some situations. We expect that re-collecting data incorporating these yaws would lead to much better performance than what we currently observe.

8 Summary and Future Work

We focus on the deficiencies in current state-of-the-art approaches for achieving fast and agile motion of quadrotors in diverse environments, specifically addressing the lack of yaw control and learning objectives. We implement two solutions for determining optimal yaw during quadrotor flight: (a) heuristic n-look-ahead method and (b) a perception module to compute yaw for maximum information gain. We find that the heuristic method selects trajectories farther away from obstacles and the RAPTOR Perception yaw prediction method has yet to reach the baseline success rate.

This prompts us to further investigate and improve upon on our methodologies. Future directions include designing a learned policy that can also adapt the top flight speed to adjust to varying environment clutter, and incorporating a global planner to tackle higher-level objectives such as planning waypoints for full environment sensor coverage. We also plan to collect new data suitable for our implementation to train and validate our approach and any further implementations.

References

[1] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021. doi:10.1126/scirobotics.abg5810. URL `https://www.science.org/doi/abs/10.1126/scirobotics.abg5810`.

[2] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525. IEEE. ISBN 978-1-61284-386-5. doi:10.1109/ICRA.2011.5980409.

[3] B. Zhou, J. Pan, F. Gao, and S. Shen. RAPTOR: Robust and Perception-Aware Trajectory Replanning for Quadrotor Fast Flight. 37(6):1992–2009. ISSN 1941-0468. doi:10.1109/TRO.2021.3071527.

[4] J. Tordesillas and J. P. How. Deep-PANTHER: Learning-Based Perception-Aware Trajectory Planner in Dynamic Environments. URL `http://arxiv.org/abs/2209.01268`.

[5] X. Wu, S. Chen, K. Sreenath, and M. W. Mueller. Perception-Aware Receding Horizon Trajectory Planning for Multicopters With Visual-Inertial Odometry. 10:87911–87922. ISSN 2169-3536. doi:10.1109/ACCESS.2022.3200342.

[6] M. Watterson, S. Liu, K. Sun, T. Smith, and V. Kumar. Trajectory optimization on manifolds with applications to quadrotor systems. 39(2-3):303–320. ISSN 0278-3649, 1741-3176. doi:10.1177/0278364919891775.

[7] S. Liu, K. Mohta, N. Atanasov, and V. Kumar. Search-Based Motion Planning for Aggressive Flight in SE(3). 3(3):2439–2446. ISSN 2377-3766, 2377-3774. doi:10.1109/LRA.2018.2795654.

[8] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza. PAMPC: Perception-Aware Model Predictive Control for Quadrotors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. doi:10.1109/IROS.2018.8593739.

[9] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza. Flightmare: A Flexible Quadrotor Simulator.

[10] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. RotorS—A Modular Gazebo MAV Simulator Framework. In A. Koubaa, editor, *Robot Operating System (ROS)*, volume 625, pages 595–625. Springer International Publishing. ISBN 978-3-319-26052-5 978-3-319-26054-9. doi:10.1007/978-3-319-26054-9_23.

[11] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange. Unity: A General Platform for Intelligent Agents. URL `http://arxiv.org/abs/1809.02627`.

[12] H. Hirschmuller. Stereo Processing by Semiglobal Matching and Mutual Information. 30(2):328–341. ISSN 1939-3539. doi:10.1109/TPAMI.2007.1166.

[13] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. *Proceedings of EuroGraphics*, 87, 08 1987.