Learning UAV Flight Aggression under Map Uncertainty

Anish Bhattacharya

Ravi Konkimalla

Hersh Sanghvi

CIS 700-02,
University of Pennsylvania

Abstract

Fast autonomous flight through cluttered environments is an active research problem involving challenges in state estimation, localization, control, and planning. State-of-the-art algorithms in these respective fields still pose limitations in computation time and complexity, rendering a typical modular pipeline unusable at high speeds unless specifically tuned for the hardware setup. Furthermore, motion blurred sensor information can result in uncertain local map and obstacle information which can be risky in fast flight. We propose a learning based local planner that commands high-level control actions based on an approximation of local map uncertainty calculated directly from an onboard camera. Experiments were done in a photorealistic simulation environment with a standard configuration quadrotor. The reinforcement learning framework avoids an over-engineered reward function by only including positive rewards for forward flight and penalties for collisions. Current results achieve marginally higher task success rate when an estimate of the local map uncertainty is provided to the learner, and also variations in the drone's velocity when noise is added to the depth image. Further experimentation and analysis is necessary to fully understand these difference in behavior and their utility.

1 Introduction

Fast flight in unknown environments is a very challenging task. A typical robotics approach is generally modular, comprising of separate state estimation, localization, control, and planning blocks. However, the limited computation and sensing capabilities possible onboard a lightweight, fast moving quadrotor can produce motion-induced sensing artifacts that result in incomplete information. These can cause inaccurate estimates of obstacles and other surroundings, which can be catastrophic at high speeds. Such failures might be avoided by traveling at slower speeds, thereby improving the fidelity of sensor measurements. A simple solution to this problem would be for an agent to always travel at lower speeds; however, such an approach forgoes the dynamic capabilities of quadrotors, and is not optimal in terms of time to reach a goal position. An agent behaving optimally might instead choose to modulate its velocity and heading according to its own uncertainty in the environment, the same way a person might walk slowly with arms outstretched along a wall in an unlit room.

For an agent navigating an unknown environment, a mapping system is critical. Occlusions and noise in depth images and errors in fusion can cause standard mapping algorithms to produce incomplete maps. The uncertainty in these maps inherently constrains the robot's motion; if the robot does not know what lies to its immediate left, it is obvious that it should not travel quickly in that direction without first turning to scan it. We therefore propose a system that reacts to uncertainty by altering its velocity and heading based on the output of a mapping algorithm.

In this work we specifically focus on uncertainty produced by the perception system. An example visualization of this uncertainty from our method is shown in Figure 1.

2 Related Work

Recent work has involved specialized hardware and software components to deal with these issues, but also delved into learning to adapt to the new regime of fast flight. We aim to use learning to achieve safe, fast flight through an unknown and cluttered environment with just a single onboard vision sensor.

((a)) ((b)) ((c))

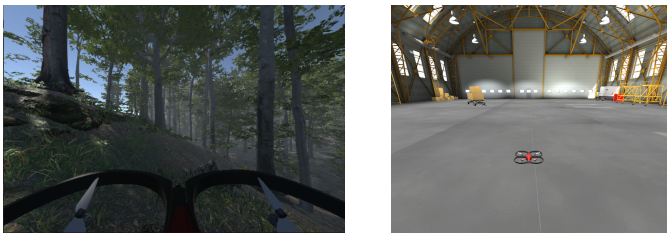
Figure 1: A sequence of images showing map uncertainty as the drone approaches a box in an otherwise clutter-free environment. Note that even though ground truth depth was fed into the mapping framework, it still has uncertainty due to the motion of the drone. (Darker shades indicate less uncertainty.)

Deep reinforcement learning is becoming increasingly popular for learning control policies in an unsupervised setting for performing navigation tasks. In [12] the authors trained a Deep Deterministic Policy Gradients algorithm[10] to learn a continuous control policy for visual navigation of Biped Humanoid robot using only RGB images.The paper also demonstrates successful Sim2Real transfer of the learned policy. In [6] an Proximal Policy Optimization[15] is used to learn a navigation policy in simulation for UAV in discredited action space. Soft Actor Critic Algoritthms [3] uses an off policy maximum entropy RL framework to train a stochastic actor and have been shown to have better convergence properties.In [4] the authors also train a minitaur using SAC to achieve quadrupedal locomotion with only two hours of real world training data or 160K environment steps.

One of the more successful works in using a typical robotics modular system for fast quadrotor flight is [13], in which each particular block in the pipeline is adjusted to operate at low computational strain and time. Here, they were able to reach speeds of around 3m/s in cluttered environments and up to 18m/s in open areas. Custom, high speed hardware can also relieve computation requirements of the fast flight problem [1], but this might not always be available on a given quadrotor platform. In [2] an integrated perception and control method to avoid errors in state estimation. [11] presents a sensing-limited approach that includes a safe stop policy to guarantee collision avoidance. We hope to expand on this type of work by avoiding hand-designed safe planners and instead encoding safe flight into a learned policy.

There has been significant past work on using traditional trajectory optimization style methods to handle uncertainty. Many methods generate plans using motion primitives. [14] compose a set of motion primitives and generate plans and velocities by incorporating information about the object density in the map. [18] formulate the problem as searching for an optimal B-spline over a voxel grid and try to maximize the distance to obstacles, but do not explicitly incorporate perception uncertainty.

There is also significant recent work on using learning to control UAVs in various environments. In [9] the authors trained a quadrotor to generate a heading and velocity based on monocular camera image input to a convolutional neural network (CNN) in simulation and the real world. The maximum speed was 3m/s with a near perfect success rate on a real-world track on which it was trained. [16] uses a deep reinforcement learning (DRL) framework to perform obstacle avoidance by using temporal attention to encode memory into the system. This particular study does not consider fast flight. [8] explicitly uses uncertainty for obstacle avoidance, but this uncertainty is approximated from DRL and incorporated into a collision avoidance cost function. This contrasts with our proposed work in which we aim to use the uncertainty coming from a localization module that is agnostic to the environment in which the system was trained.



((a)) ((b))

Figure 2: (a) Forest environment and (b) warehouse environment in Flightmare

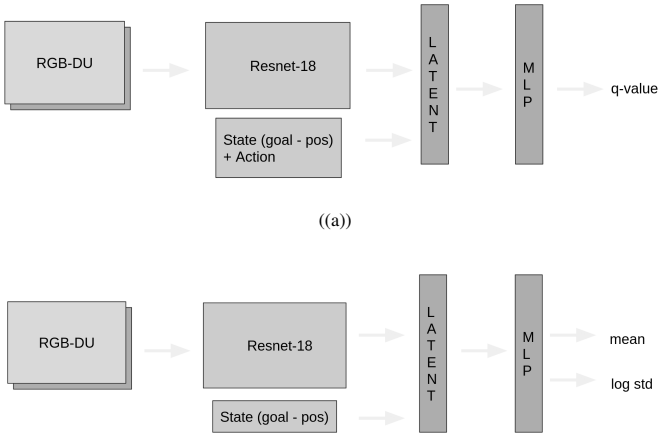


Figure 3: (a) Actor function (b) stochastic critic policy.

3 Approach

Our approach includes a standard quadrotor in a new photorealistic simulator. We use a standard robotics framework (ROS) and corresponding image handling and mapping packages, followed by a reinforcement learning framework that can train quickly on a desktop computer with an AMD Ryzen 9 3900 processor and NVidia RTX 2080 GPU.

3.1 Simulation Environment

We are using the Flightmare quadrotor simulator [17] which uses the RotorS simulator paired with Unity for photorealistic environment rendering. This package includes a reinforcement learning API and a variety of sensors to be used on the simulated UAV. The two simulated environments of interest are shown in Figure 2. The forest environment provides rich training examples due to the varied terrain, shaded regions from trees, and small obstacles such as branches and leaves that might induce a larger uncertainty measurement of the environment. However, we perform initial training and experimentation in the simpler warehouse environment to verify each component of our system and also due to the reduced computation strain for graphics rendering.

Using this new simulator does pose some challenges. For example, in the version we started using, collision detections are not ported into Flightmare from Unity for use in the training pipeline. In this case, we had to rely on a collision detector using the onboard depth camera which was not always reliable. Another issue was a misalignment of the depth image with the RGB image, which we were able to resolve by manually correcting the rotation angle between the frames.

3.2 Mapping and Uncertainty Estimation

A RGBD sensor rigidly mounted on the quadrotor feeds into a mapping module that returns some estimate of the uncertainty in the robot's surrounding scene. We first add noise to the ground truth depth frame obtained from Flightmare, followed by a fixed-lookahead step where we discard certain points that are further away from the quadrotor in order to simulate a sensor with a short range. Next, we convert this noisy depth image into a laser scan point cloud representation, which is fed into Octomap [7] (along with the ground truth pose of the quadrotor) which fuses these point clouds over time and outputs a probabilistic occupancy grid as a map. To obtain our final uncertainty image at each timestep, we query the occupancy probability from Octomap at each point in the current point cloud $p(x,y,z)$, and fill the uncertainty image U as follows:

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = K \begin{bmatrix} x & y & z \end{bmatrix}^T$$
$$i = \frac{x'}{z'}$$
$$j = \frac{y'}{z'}$$
$$U(i,j) = -p(x,y,z)\log_2(p(x,y,z))$$

Where K is the camera intrinsic matrix that maps points from the world frame into the camera frame. Naturally, $U(i,j)$ is maximized when the occupancy probability is 0.5, which is when the mapping framework has no knowledge of if the node is occupied or is free space.

It is worth noting that by this calculation, the uncertainty was not calculated for free space nodes due to the fact none of the points in the point cloud actually occur in free space. Therefore, this version of the uncertainty image only represents the uncertainty about "occupied" points in the Octomap. We discuss our solution to this in Section 4.5.

3.3 Learning Framework

The uncertainties in the occupancy grid will be stacked with the RGBD portion of the camera frame and inputted into a DRL framework that outputs a heading and velocity, as done in [9]. Low-level control is done by the built-in quadrotor control algorithms. We choose a model-free method over model-based, so that we can avoid having to learn the dynamics involved in the aggressive flying task. Since we are training in simulation, sample efficiency is not a primary concern and enables us to run many trials safely. We further choose Soft Actor Critic Algorithm [3] as it allows us to learn policy in a continuous action space.

The reward function is a weighted sum of both the intrinsic and extrinsic rewards. The agent receives extrinsic rewards when the episode terminates successfully upon reaching the goal state or when the agent crashes into an object in the environment. The intrinsic reward is calculated as the weighted difference between the distance to goal position before the the after executing the action in the simulator. This provides incentive to the quadrotor to move in the direction of the goal position as it gives a positive reward whenever the agent moves closer to goal and negative reward when it moves away. The weight hyper-parameter is set such that the overall intrinsic reward the agent gets will be less than the positive extrinsic reward.

$$R = \begin{cases} +r, & \text{successfully reaches goal} \\ -r, & \text{upon collision with obstacles} \\ w \times (\text{change in dist to goal}), & \text{otherwise} \end{cases}$$

We terminate an episode when the agent successfully reaches the goal position or when the agent crashes into an obstacle or when the agent fails to reach the goal with in a certain time limit.

We use ResNet-18[5] pretrainined on Imagenet as our backbone due to limited compuational resources for both the actor and critic. The current displacement required to reach the goal is then appended to the output of the backbone. This latent representation of the state is then fed into a multi layer perceptron module which predicts the q-value in case of critic and mean and standard deviation of the heading and velocity in case of the policy. The network architectures are shown in 3

4 System Validation Experiment

In order to assess the validity of our training pipeline and control scheme, we set up a toy problem of avoiding the boxes in the warehouse environment. Figure 4 shows the four representations of the object from onboard sensors and the entropy calculation.

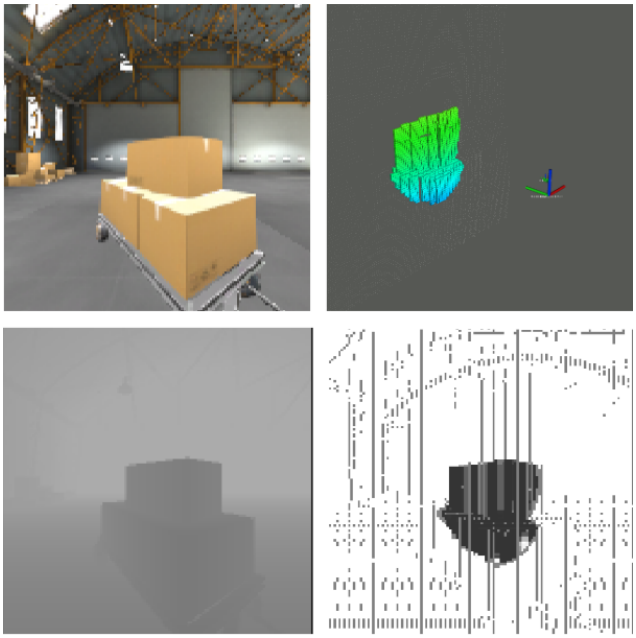


Figure 4: Onboard representations of box obstacle: (top-left) RGB image; (bottom-left) depth image; (top-right) Octomap voxel grid; (bottom-right) entropy frame (without free space correction).

4.1 Training Setup

For our toy problem, we structured a task such that the drone starts 4m away from the boxes, directly facing them. The goal position is any point 20m ahead of the current position. Restrictions are that the agent must stay within 20m laterally of the starting position, 1.5m of the original altitude, and not move more than 5m backwards. Any deviation beyond this defined space is considered a crash.

For the purposes of RL, an episode begins with the drone at the specified starting location and ends when either the drone crashes (an unsuccessful episode) or it reaches the goal distance (a successful episode). After each episode the map is completely reset, including both the voxel inhabitances state and uncertainties.

We trained all of our policies in two stages. In the first stage, each episode is reset with the drone facing directly forward at the same starting position. In the second stage of training, we initialize the networks with the first stage's weights and every new episode resets the drone with a (small) random heading at a random position. The heading distribution was a Gaussian centered around 0 (facing forward) with a standard deviation of $\pi/24$; the x- and y-positions were centered around the first stage's starting position with perturbations following a Gaussian distribution with standard deviations ranging from 0.5 to 1.0. The purpose of this is in order to let the policy learn a path to completion in an easy task before introducing the more difficult problem of starting from a random position.

The hyperparameters for training are shown in Table 3.

4.2 Experiments

For our first experiment, we trained four policies in which we varied the noise added to the depth image and changed whether or not the uncertainty image was given as an input to the policy. We compared these policies using two performance metrics: the percentage of episodes where the policy reached the end goal, and the average commanded velocity per iteration, as a rough gauge of the aggressiveness of the policy.

As a note, the uncertainty referred to in this section is the calculation shown in Section 3.2.

4.3 Evaluation and Results

All policies were evaluated for 100 episodes in the same setting they were trained in (i.e., the noiseless, RGBD-only policy was also tested with noiseless input). The results for the toy problem are shown in Tables 1 and 2. Note that the $^{+f}$ designation indicates a free-space correction in the entropy calculation which is further discussed in Section 4.5.

Policy	RGBD	RGBDU	RGBDU^{+f}
Ground truth depth	0.57	0.59	0.55
Noisy Depth	0.47	0.50	0.52

Table 1: Success rate of policies over 100 evaluation episodes

Policy	RGBD	RGBDU	RGBDU^{+f}
Ground truth depth	1.58	1.50	1.47
Noisy Depth	1.56	1.57	1.48

Table 2: Average velocity per iteration of policies over 100 evaluation episodes

We can see that the ground truth RGBD policy not only has a high success rate, but also exhibits the highest overall velocity, indicating that it is able to fly aggressively but also avoid obstacles. The noisy RGBD policy is the worst performer, showing high velocity but an inability to avoid obstacles. Notably, the ground truth RGBDU policy shows the highest success rate but also the lowest average velocity, indicating that it learns to moderate its aggression around the obstacles and also to avoid them.

Interestingly, including RGBDU in the noisy case slightly improves the performance, but does not moderate the aggression, indicating that in the noisy case, only calculating the uncertainty of objects and not the free space might be insufficient, or that extra supervision might be needed.

4.4 Discussion

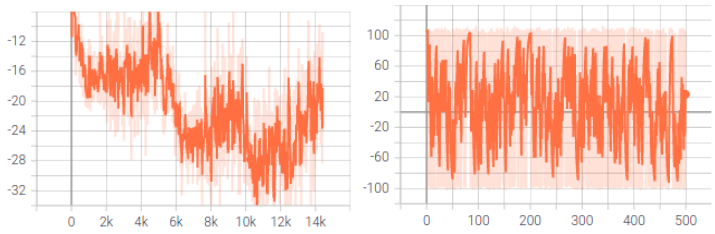
There are a number of confounding factors in this experiment that warrant further discussion and experimentation. Firstly, we observe that our training does not fully converge within the number of allotted episodes, as shown in Figure 5. This is to be expected, since computation limitations constrained the amount of training we could accomplish on all of our policies, and further prevented extensive hyperparameter tuning. In particular, we used an older variant of SAC in which the α parameter, which controls the noise in the policy, was set manually. Future work includes further hyperparameter tuning in order to refine our results.

Second, for the box avoidance task, it is critical that the drone receive a negative reward every time it crashes into the box. However, due to the lack of a prepackaged, ground truth collision detector in Flightmare (at the time of running this experiment), we were forced to use an imperfect collision detector from the ground truth depth image. One of the key failures of our homespun collision detector was that it sometimes allowed the drone to go through the box if it enters exactly orthogonal to the box's surface. This sometimes incentivizes the policy to approach the box even if in most cases it would result in a collision. The noisy, RGBDU policy in particular relied on this exploit quite frequently, which could explain its high aggression. We discuss this further in Section 5.

Finally, as mentioned in Section 3.2, the uncertainty calculation currently only outputs the uncertainty for occupied voxels in the Octomap, due to using the input point cloud in order to calculate the uncertainty image. The results of this can be seen in the bottom right image in Figure 4, where free space is shown as high-entropy (white) even though Octomap is most likely certain that those nodes are unoccupied. Because of this, the uncertainty image mostly acts as a secondary depth image, because the pixels representing free space are not populated. This gives a relatively uninteresting image to train on, especially since most of the space in the warehouse environment is free space.

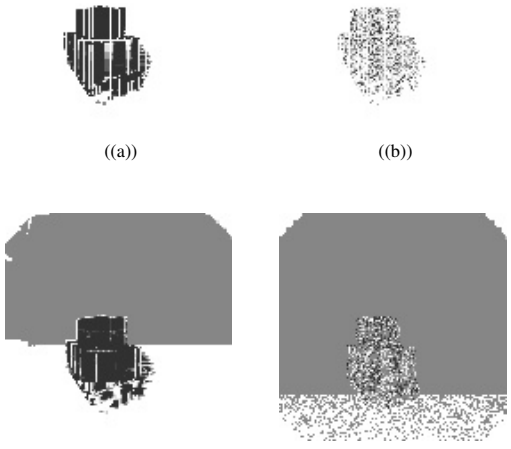
Hyper-parameters	Value
max. training episodes	500
extrinsic reward: r	100
intrinsic reward scale	0.5
gamma	0.95
alpha	0.2 - 1
learning rate	0.0001
image dimensions	(128 $\times$ 128)
max. forward velocity	3 m/s
max. heading	$\pm$45 degrees

Table 3: Hyper parameters used for training



((a)) ((b))

Figure 5: (a) Policy loss and (b) Episode reward for training the ground truth, RGBD policy displayed in Tensorboard with a smoothing factor of 0.6.



((a)) ((b))

((c)) ((d))

Figure 6: (a) Old uncertainty with ground truth depth (b) Old uncertainty with noisy depth (c) New uncertainty with ground truth depth (d) New uncertainty with noisy depth. The images in this Figure are shown after hovering in place for quite some time, so the uncertainty of the box is low. (Darker shades indicate less uncertainty.)

4.5 Adjusted Free-Space Uncertainty Calculation

Following our class presentation, we modified our uncertainty calculation to properly incorporate the uncertainty of Octomap in unoccupied space as well. The new calculation uses points from the depth image as opposed to the point cloud. Depth values that are beyond a certain range–corresponding to free space directly in front of the drone–are capped to some maximum distance, which is then used to query the Octomap. The differences between the various uncertainty images is shown in Figure 6. We can see that the ucnertainty of free space is lower than in the prior images, but not as low as of the box. Intuitievly, this makes sense as free space for a depth sensor will always be more uncertain than a solid object in front of the sensor. Further work needs to be done to find the best representation of obstacles vs. free space in the depth image.

We trained two new policies incorporating these new uncertainty images. In Tables 1 and 2, these policies are referred to as RGBDU^{+f}. It is also important to note that these policies use the slightly modified MLP architecture where the current state of the agent is appended at a deeper layer.

The new uncertainty images show strong results in both metrics. While in the noiseless case, the RGBDU^{+f} shows a comparable (if slightly lower) success rate, the RGBDU^{+f} policy shows strong results in the noisy case, increasing the success rate compared to RGBD by 5%. In addition, both RGBDU^{+f} policies greatly decrease the aggressiveness of flight, confirming our original hypothesis that including uncertainty might teach the policy to modulate its aggressiveness. However, further experiments are needed to fully understand the effects of including this new uncertainty.

4.6 New Collision Detector

We ran training tests with a modified collision detector that specifically triggers a crash if the drone is within an approximate cuboid area centered on the box in the warehouse environment. However, this seems to greatly increase the difficulty of the task, causing low success rates in all policies (even ones that did not rely on this trick). We hypothesize that much more training data, manifested in greater training time, along with tuning the harshness of this new collision detector might improve results.

5 Future Experiments

5.1 Reward Functions

Our next and important experiment involves different terms and weightings in our reward function. In addition to having “standard" terms in our reward function that encode progress towards the goal, we want to explore how incorporating different terms can influence the behavior of our agent. In particular, we will first see if the intrinsic reward function outlined above is sufficient to encode our desired behavior of taking more certain paths and slowing down in the presence of uncertainty. Another possibility we want to explore is an explicit penalty term:

$$p(t) = -v(t) * u(\theta_t) \tag{1}$$

$$\tag{2}$$

Where $v(t)$ is the velocity of the drone, θ_t is the current heading, and $u(\theta)$ is the uncertainty encoded by the occupancy grid in the direction of the current heading. This term would explicitly penalize high velocities in uncertain directions, while allowing high velocity when the uncertainty is low. However, a possibility is that this term might also encourage low velocity all the time.

To increase flight aggressiveness of the agent we want to inversely scale the positive extrinsic reward with the time taken to complete the episode. Following the class discussion we also want to include an auto tuning network for the temperature parameter which will help in reducing the dependency of the learned policy on α.

In addition, we will explore how the relative and absolute weighting scales between these terms influences performance.

5.2 Network Architecture

We will also explore how the structure of our networks influences our performance, and whether or not initializing our networks with pre-trained feature detectors such as ResNet-18 is helpful for the final task.

5.3 Quadrotor Control

While we take inspiration from [9] in learning high-level control commands for the quadrotor, namely a velocity and heading, we do not further fit a smooth trajectory to these parameters as they do. We have seen some promising results with our simplified toy problem of avoiding a single, large obstacle in a largely open environment, but extending this simple and discontinuous control to navigating a highly cluttered environment such as a forest might not work. In adapting to a smoother control scheme, we could still have the learner output a heading and velocity but either fit a smooth, minimum-snap trajectory to it or even develop a correspondence to a library of motion primitives.

6 Conclusion

We have some preliminary results as discussed in Section 4.4 that suggest that adding an uncertainty estimate to the learner inputs can maintain the success rate of the experiment while not reducing the drone’s forward velocity under noisy depth images. This concurs with our original hypothesis that adding uncertainty estimates of the local map to the learned model might help control flight aggression and improve performance.

The immediate next steps of this project include evaluating the efficacy of each module in our RL and simulation pipeline, and tuning the system to be able to handle more complex environments. We hope to

eventually achieve a controlled, fast flight through a highly cluttered environment in simulation, and then port the system to a real-world platform.

Links to our fork of the simulator, the learner code, and roll-out videos of the learned policies are made available.

References

[1] Andrew J Barry, Helen Oleynikova, Dominik Honegger, Marc Pollefeys, and Russ Tedrake. Fast onboard stereo vision for uavs. In *Vision-based Control and Navigation of Small Lightweight UAV Workshop, International Conference On Intelligent Robots and Systems (IROS)*, 2015.

[2] Pete Florence, John Carter, and Russ Tedrake. Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. In *Algorithmic Foundations of Robotics XII*, pages 304–319. Springer, 2020.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[4] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Victoria J Hodge, Richard Hawkins, and Rob Alexander. Deep reinforcement learning for drone navigation using sensor data. *Neural Computing and Applications*, pages 1–19, 2020.

[7] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. doi: 10.1007/s10514-012-9321-0. URL `http://octomap.github.com`. Software available at `http://octomap.github.com`.

[8] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

[9] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. *arXiv preprint arXiv:1806.08548*, 2018.

[10] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[11] Sikang Liu, Michael Watterson, Sarah Tang, and Vijay Kumar. High speed navigation for quadrotors with limited onboard sensing. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1484–1491. IEEE, 2016.

[12] Kenzo Lobos-Tsunekawa, Francisco Leiva, and Javier Ruiz-del Solar. Visual navigation for biped humanoid robots using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(4): 3247–3254, 2018.

[13] Kartik Mohta, Ke Sun, Sikang Liu, Michael Watterson, Bernd Pfrommer, James Svacha, Yash Mulgaonkar, Camillo Jose Taylor, and Vijay Kumar. Experiments in fast, autonomous, gps-denied quadrotor flight. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7832–7839. IEEE, 2018.

[14] M. Ryll, J. Ware, J. Carter, and N. Roy. Efficient trajectory planning for high speed flight in unknown environments. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 732–738, 2019. doi: 10.1109/ICRA.2019.8793930.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[16] Abhik Singla, Sindhu Padakandla, and Shalabh Bhatnagar. Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[17] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. *arXiv preprint arXiv:2009.00563*, 2020.

[18] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight, 2019.