Deep Flight: Learning Reactive Policies for Quadrotor Navigation with Deep Reinforcement Learning

Ratnesh Madaan, Rogerio Bonatti, Dhruv Mauria Saxena, Sebastian Scherer
The Robotics Institute
Carnegie Mellon University, Pittsburgh, PA 15213
Email: {ratneshm, rbonatti, dsaxena, basti}@andrew.cmu.edu

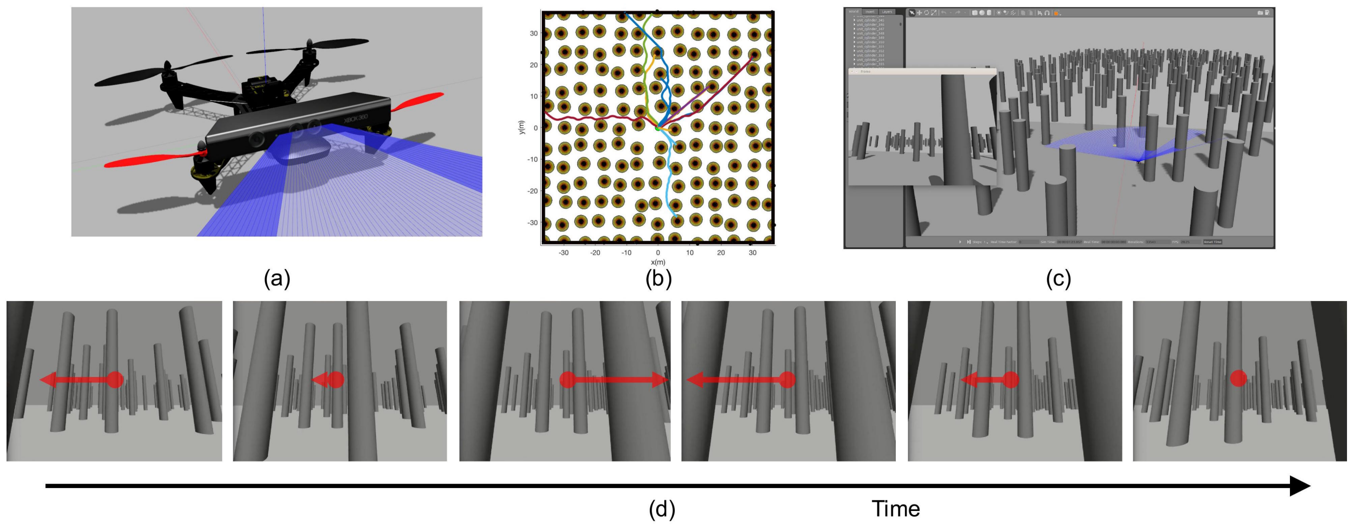

Fig. 1: Our simulation environment in Gazebo. (a) Close-up of our robot model (b) Shows top-down view of test run trajectories on a random forest of cylinders with inset of robot-centric monocular image.(c) Oblique view of forest of cylinders. (d) Sequence of robot images and yaw commands outputted by the network. The direction and the magnitude of the red arrows are in accordance with the yaw velocity being executed by our learnt DQN policy.

***Abstract*—**
Obstacle avoidance for micro aerial vehicles is an essential capability for autonomous flight in cluttered environments, but is often challenging because limited payload capacity only permits the use of lightweight sensors like monocular cameras. We show that Deep Q-learning has the potential to be used for end-to-end training of obstacle avoidance policies for UAVs, mapping directly from a stack of monocular images to controls in a low-textured environment while flying at a constant speed. We validate our algorithm by testing it on a random forest of cylinders and comparing it against baseline algorithms, By training the network over 6,200 self-supervised episodes, we achieve significantly higher performance than the baseline in terms of obstacle avoidance success rate and collision-free flight time. In addition, we released the open-source code of our simulator to facilitate rapid prototyping of deep RL algorithms for flying robots. We are currently working towards deploying our network on a real UAV system. A video of our results is available at https://goo.gl/w5yDTX

I. Introduction

Despite numerous advancements in sensor technologies and motion planning algorithms, navigation of micro unmanned air vehicles (UAVs) is still a challenging problem due to payload weight, size, and power constraints [1, 2]. In general there are three hierarchies of autonomy for the motion planning of UAVs: a global planner provides with a path or a motion plan to be followed, a local planner which follows the global plan and reacts to previously unseen static or dynamic obstacles either due to limitations of the perception system or obstacle distribution in the world, and finally, a low level flight controller that translates the local planner's output (velocity or attitude commands) to appropriate motor thrust [3].

An open question is then which level of autonomy, if any, can leverage machine learning techniques and the recent advances made by the deep learning community. We argue that given a prior map of the environment or a hidden observation obtained using onboard sensors, traditional planning algorithms coupled [4-7] are sufficient - both in terms of solution quality and run time - as far as safe, smooth, time-optimal trajectories with formal guarantees are concerned. However, issues arise when previously unseen obstacles, either static or dynamic, show up in the vicinity of the planned path of the robot, thereby violating the flight safety criteria. Such edge cases break the global plan, and call for a fast reactive planner or policy which can measure the threat of such unseen obstacles implicitly or explicitly,

Preprint submitted to 2018 IEEE International Conference on Robotics and Automation. Received September 15, 2017.

and change the motion of the robot so that it can avoid them immediately. When the reactive planner deems the robot to be threat-free, it can either return to following the previous global plan or if need be, replan again from the current state.

Intuitively, the reactive policy is then an ideal application for learning methods. If we go lower in the stack and directly learn motor thrusts, it is detrimental to the safety of the flight. Linking back to the UAV autonomy hierarchy, learning motor thrusts can result in unstable and unsafe flight due to policy variance. If we go high up in the stack and aim to solve motion planning via learning, we face a similar issue in the robustness of the solutions thus obtained. A reactive avoidance policy that is used only when there is an imminent thread of collisions or maintain the vehicle's safety constraints, is a nice fit for learning based methods. Such a policy should learn a direct mapping from sensory input to control commands, like a direction to fly towards.

Now, a practical constraint for micro UAVs is that their small payload permits for only lightweight sensors like monocular cameras. Previous work for monocular image based obstacle avoidance utilizes domain-specific cues like optical flow [8], relative size changes [1], and depth-prediction [9, 10]. However, these methods are face generalization issues, as explained in the section on related work. On the other hand, methods leveraging deep convolutional neural networks are increasingly becoming an essential component in the pipeline of multiple state of the art approaches for numerous vision tasks. They have also paved the way for the success of deep reinforcement learning algorithms, specifically Deep Q-Networks (DQNs) and their variants, which have shown impressive results on fully observable 2D Markov Decision Processes(MDPs)[11]. However, works applying DQNs to navigation tasks in partially observable 3D scenarios are scarce, barring those attempting the VizDoom challenge [12, 13].

In this work, we use DQNs to learn a mapping from a discrete set of consecutive UAV-centric monocular images to a discrete set of yaw commands, thereby learning a reactive policy for obstacle avoidance. Our training environment, shown in Fig 1, is a forest of low-textured cylinders in a Gazebo [14] simulation environment while flying at a constant speed and altitude to reach a goal point. In order to learn an effective policy, the design of the MDP, especially the reward function is critical. We design our reward function to capture two criteria - avoiding crashing into obstacles and maximizing the minimum distance from obstacles. We take inspiration from traditional vector field motion planning literature [15] and design the reward so that the robot learns to avoid the obstacles as soon as it veers close to them.

Our work has two main contributions: first, application of Deep Q-Network algorithm to learn a reactive obstacle avoidance policy for UAVs using only monocular images as input. To the best of our knowledge, this is the first work applying DQNs to the field of aerial vehicles. We compare our approach with baseline policies and model-based methods for UAV obstacle avoidance. In addition, we provide open source code for our training environment and micro UAV simulator as an OpenAI Gym environment, as well as the DQN implementation. This release can streamline research in the area of reinforcement learning in the domain of UAVs.

II. Related Work

In this work, we use DQNs to learn a mapping from a discrete set of consecutive UAV-centric monocular images to a discrete set of yaw commands, thereby learning a reactive policy for obstacle avoidance. We will briefly describe works related to our method.

A. Model-based obstacle avoidance

Navigation with monocular cameras can be tackled by leveraging domain-specific cues like optical flow [8], relative size changes, and depth [9, 10], each having its own advantages and points of failure. Optical flow uses motion parallax to detect changes in the image, and is of limited use in frontal camera applications because of the small angles relative to the frontal direction and expensive computation [1]. Relative-size changes and depth prediction are dependent on good texture and lighting conditions.

Mori and Scherer [1] explored relative size change by comparing SURF features of frontal images, but their approach requires high-textured environments. Saxena et al. [9] worked on depth recovery from single images using supervised learning, and utilize the depth map to train a policy for ground navigation in simulation and real environments [16]. Saxena does not provide details on failure cases, but we argue that the method is highly dependent on texture-rich environments. A downside of model-based approaches for reactive policies is that they are hand-tuned for specific environments, and relative-size changes and depth prediction are dependent on good texture and lighting conditions.

B. Learning-based navigation

On the context of UAV navigation, there is work published in the field of supervised learning, reinforcement learning and policy search.

Gandhi et al. [17] collected a dataset consisting of positive (obstacle-free flight) and negative (collisions) examples, and trained a binary convolutional network classifier which predicts if the drone should fly in a direction or not. Sadeghi and Levine [18] used a modified neural fitted Q-learning(NFQ) algorithm to train a policy only in simulation and applied it to a real robot, using a single monocular image to predict probability of collision and selecting a collision-free direction to fly towards. NFQ is less efficient as compared to DQNs [19] however.

III. Approach

In this section we describe our simulation environment, algorithm, and implementation details.

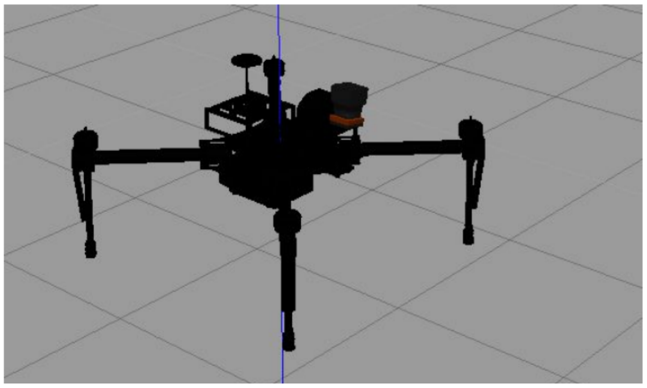
Fig. 3: Close up of our quadrotor model in simulation. It has a static, horizontal, 2D laser scanner, and a monocular camera attached to the frame.

A. Simulation environment

We use a simulated model of a quadcopter in Gazebo [14] which is equipped with a camera and a static 2D laser sensor, as shown in Fig. 3. We attached a laser sensor to our robot in simulation so as to calculate the ground truth distance of the nearest obstacle, and give it a reward accordingly. the forest. It should be noted that the laser scanner is used to only calculate the instantaneous reward, and is *not* used for training. The only sensory input for the DQN are images from the monocular camera.

Our environment is a forest of randomly placed cylinders, whose configuration changes at the end of each episode (Fig. 1). We model the navigation task as a deterministic MDP, which is defined by a state, action, reward, discount tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{R}, \gamma\}$, where:

$\mathcal{S}$: is a sequence of 4 consecutive camera images, each 84×84 in size. Note that we dowsample the 480×640 camera image to 84×84.

$\mathcal{A}$: is a set of 9 uniformly separated yaw velocity commands in the range of [-0.8, 0.8] rad/s (4 anticlockwise, 4 clockwise, and 1 corresponding to zero velocity). Across all actions, the drone has a constant forward velocity of 1 m/s.

γ : is a constant discount factor of 0.99

$\mathcal{R}$: is the reward. We tailor our reward function to avoid crashing into obstacles and maximize the minimum distance from obstacles. A more detailed explanation of the reward scheme follows:

- *Positive reward for flying in free space*:
  Let the minimum laser reading at a particular instant be denoted by d_{obs}. If the d_{obs} is above a threshold distance d_{neg}, then the agent is given a positive reward, r_{safe} for each time step.
- *Maximize the minimum distance from obstacles*:
  Let d_{crash} define a threshold for d_{obs}, below which we consider that the UAV has crashed. When $d_{neg} < d_{obs} < d_{crash}$, implying that the robot is in the vicinity of an obstacle, a linearly increasing (in magnitude) negative reward is given from r_{neg} to $r_{precrash}$, as depicted in Fig. 4. This is inspired by the vector field motion planning literature, where the robot follows a potential function which has high values near obstacles and low values near goals.
- *High negative reward for crashing*: A high negative crash reward, r_{crash} is given if $d_{obs} < d_{crash}$.
  Fig 4 shows the reward scheme for the proximities of a cylinder. Empirically, we found the following threshold distances and reward values to work well:
  $r_{safe} = 0.25$
  $d_{neg} = 2.0$, $r_{neg} = 0.0$, $r_{precrash} = -5.0$
  $d_{crash} = 0.75$, $r_{crash} = -10.0$

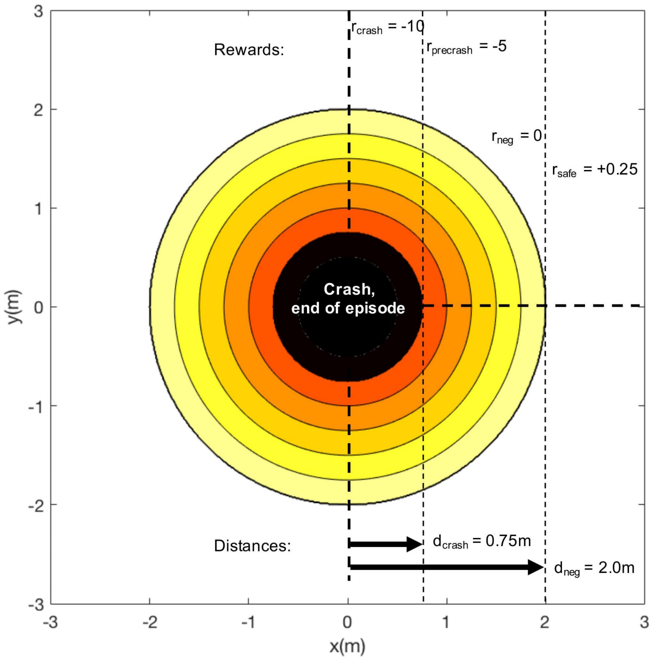

Fig. 4: Reward scheme as a function of distance from cylinder's center

We found that in practice, the reward potential around the obstacles expedites the learning of the optimal policy. We take inspiration from traditional vector field motion planning literature [15] and design the reward so that the robot learns to avoid the obstacles as soon as it veers close to them. Our objective was to make the DQN implicitly learn to associate features related to the increasing scales of cylinders in the direction of flight to negative rewards, accelerating gradients the training process.

B. Quadrotor model

We use a simplified model for velocity control of the quadrotor with a PID loop. The calculated force and torque is then applied to a URDF model of the DJI Matrice 100. This quadrotor velocity controller is a standalone ROS package, which subscribes to the velocity commands published as ROS messages from an OpenAI Gym implementation of our simulation environment.

C. Deep Q-Learning Algorithm

In the reinforcement learning problem, the goal is to find an optimal policy over a given MDP. Under a policy π, the Q-value of a state-action pair (s, a) is defined by:

$$Q_\pi(s,a) = \mathbb{E}[R_1 + \gamma R_2 + ...|s_0 = s, a_0 = a, \pi]$$

where R_i is the instantaneous reward at time step i. Finding an optimal policy, π^* is then equivalent to finding the optimal Q-value, $Q_*(s,a) = max_\pi Q_\pi(s,a)$. Deep Q-Learning [11] uses convnets as a function approximator to extend Q-learning [20] which is a standard off-policy algorithm to solve an MDP. Let θ denote the weights of the network, and the four-tuple (s, a, r, s') be a sample drawn from the replay memory. The loss function that the network minimizes is then defined as :

$$\mathbb{L}(\theta) = \mathbb{E}_{(s,a,r,s')}[\frac{1}{2}(R(s,a,s') + \gamma \ max_{a'} \ Q(s',a';\theta) - Q(s,a;\theta))^2]$$

For convergence, [11] use a target network and experience replay, which we adopt in our training pipeline as well. We refer the reader to [11] for more details involved in the training process of DQNs.

D. Implementation details

We use the architecture and training parameters proposed by Mnih et al. [11], with the input being 4 images of size 84x84. The first layer consists of 32 (8x8) convolutional filters, followed by 64 (4x4) filter, then 64 (3x3) filters, then a fully connected layer with 512 units and a final layer which outputs 9 possible yaw actions.Rectifier nonlinearities are applied after each layer. We implement our pipeline in Keras and Tensorflow and use a NVidia GTX 980M GPU for training the network. The relevant training parameters we use are : a batch size of 32, target update frequency of 10000, train frequency of 4, Huber loss as the objective function.

E. Baselines and Evaluation Metrics

We compare the performance of our method with the following baselines:

- *Random Policy* : chooses random yaw velocities
- *Straight Line Policy* : flies in straight line based on the starting orientation
- *Mori & Scherer* [1] : uses expansion of features to calculate reactive maneuvers for UAVs
- *Dey et al.* [10] : used depth prediction from monocular images for trajectory generation

We evaluate the baselines against our DQN policy using the following metrics:

- Percentage of obstacles successfully avoided on the UAV's path
- Collision-free probability across flight distances. This metric is introduced by [18]

IV. Results

We present simulated experiments with the UAV model learning a reactive obstacle avoidance policy based on monocular images.

A. Training results

During training we had a total of about 6,200 episodes, or an equivalent of 1.4 million control steps in the environment. As seen in Fig. 5-6, we observe significant increase in the reward per episode after roughly 800,000 steps. Fig. 7 depicts an increase of 180% in flight time (average episode length) during testing as compared to the initial random policy.

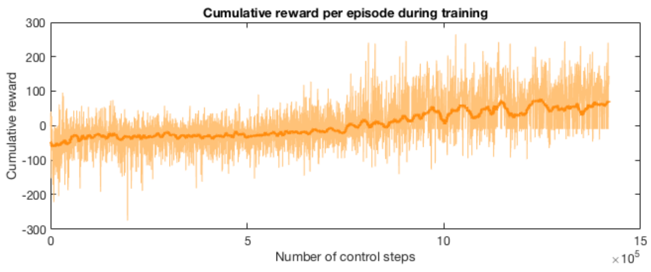

Fig. 5: Train reward with respect to number of control steps.

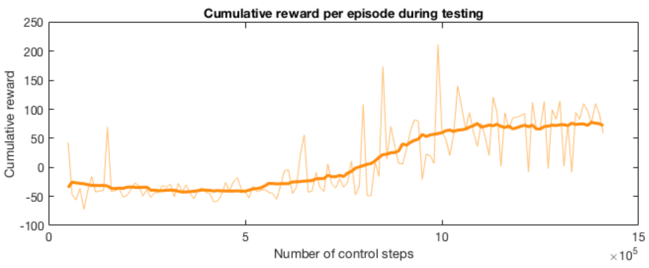

Fig. 6: Test reward with respect to number of control steps.

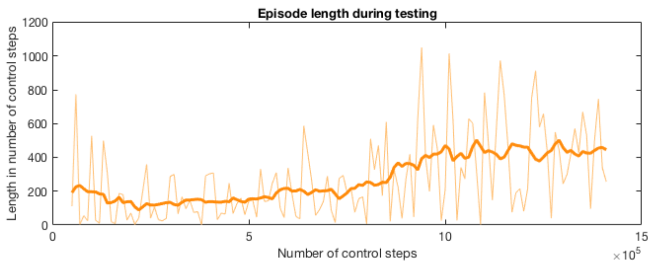

Fig. 7: Test episode length with respect to number of control steps.

B. Quantitative results

Table I compares our DQN reactive policy with the baseline methods. Given an equivalent number of roughly 50 episodes for each method, the DQN policy obtained 96% obstacle avoidance rate significantly outperforming the random policy and straight line policy, with a 62% and 47% obstacle avoidance rates respectively. In addition, the DQN policy swept a much longer flight distance and encountered 486 obstacles cumulatively versus 132 and 93 obstacles for random and straight line policies respectively.

Although performed in different domains, we can also compare our simulation results with real robotics experiments by [1] and [10]. Our DQN approach obtained comparable

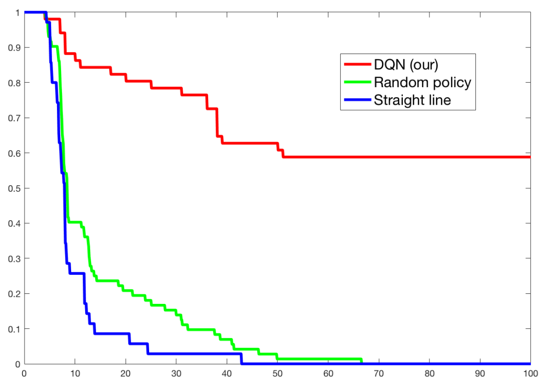

Fig. 8: Percentage of obstacle free flight (y-axis) versus Average distance across test episodes flown (x-axis) of the DQN-policy as compared to various baselines

performance for obstacle avoidance rates, with the three methods in the range of 96-97% rate. It should be noted that this comparison between simulation and real experiments is qualitative, and does not provide guarantees for a real-life application of our method.

Using a different metric, Fig. 8 compares the three simulated policies in terms of collision-free probability across flight distances. It is evident that our DQN policy significantly outperforms both the baselines by a significant margin. While it takes only an average of 12 meters for 80% of the UAVs following the straight line policy to crash and around 25 meters in the case of the random policy, we have roughly 60% of the DQN-policy UAVs remaining after 50 meters. For a length comparison, the square test arena has a sides of length 36 meters.

C. *Qualitative results*

Fig. 9 shows a top-down view of the trajectories obtained by rolling out our learned policy over three different maps, starting at different randomly sampled orientations. Each cylinder holds its reward potential (Fig. 4). Qualitatively, we can see that the UAV avoids the negative reward regions of each cylinder, staying in regions between cylinders. In addition, we see oscillatory motions of the trajectories between corridors of cylinders, caused by the lack of a global planner in the UAV's trajectory.

Fig. 10 shows 12 non-sequential frames from test runs, where the magnitude of yaw velocity command with highest Q value was drawn to scale. It is interesting to notice that the yaw commands outputted by the DQN roughly match the intuition of a human pilot in both direction and magnitude: when seeing objects occupy great part of the right side field of view, it turns left, and the contrary happens for objects on the left side. When objects are far away on the screen, *i.e*, small scales, the forward command is sent.

A video of our learned policy can be found at https://goo.gl/w5yDTX and our code for training the network, as well as an OpenAI Gym [21] environment using ROS and Gazebo is available at https://github.com/madratman/deep$_{flight}$.

V. Discussion

After training our DQN policy on simulation we raised two main points for discussion, and improvement as future work.

A. *Field tests with real UAV*

So far, all of our results were performed in simulation. We are currently working on experiments with a real UAV to be ready by the camera-ready version of this paper. This step naturally involves training a DQN in simulation and testing it in a different domain, a process as transfer learning, and we have three main strategies for a successful simulation-to-reality transition for reactive obstacle avoidance.

One approach is to, analogously as Sadeghi [18], learn a policy using images from environments with several combinations of textures. The other two methods consist in using different types of sensors that can also be carried on a UAV platform such as lidar or a depth camera. We can train the DQN with an input coming from one of these two other sensors. Intuitively we argue that learning on depth or lidar information may be easier for real-world transferability since the raw data would be more similar than images across domains.

B. *Integrate local and global planners*

In this work we discuss training a local reactive planner, but when dealing with a real application of UAVs we must integrate the local and global planners in a mission. The notion of when to switch between both is not trivial. We argue that learning a global-local planner blending function based on current features of the environment is an interesting area to be explored in motion planning.

VI. Conclusion

We show that Deep Q-learning has the potential to be used for end-to-end training of obstacle avoidance policies for UAVs, mapping directly from a stack of monocular images to controls in a low-textured environment while flying at a constant speed. We validate our algorithm by testing it on a random forest of cylinders and comparing it against baseline algorithms. In addition, we released the open-source code of our simulator to facilitate rapid prototyping of deep RL algorithms for flying robots.

We are currently working towards deploying our network on a real UAV system, using a Nvidia Jetson TX-2, on board a DJI Matrice 100. To achieve the same, we are training the DQN on disparity images and lidar data in addition to images, aiming to facilitate transfer learning.

Preprint submitted to 2018 IEEE International Conference on Robotics and Automation. Received September 15, 2017.

TABLE I: Comparison of DQN reactive policy with baseline methods. * refers to methods using real-world experiments

	No. of obstacles encountered	No. of crashes	Obstacle Avoidance Rate
Straight Line Policy	93	49	47%
Random Policy	132	50	62%
Ours (DQN)	486	21	96%
Mori & Scherer*[1]	107	3	97%
Daftry et al.*[10]	-	-	96.6%

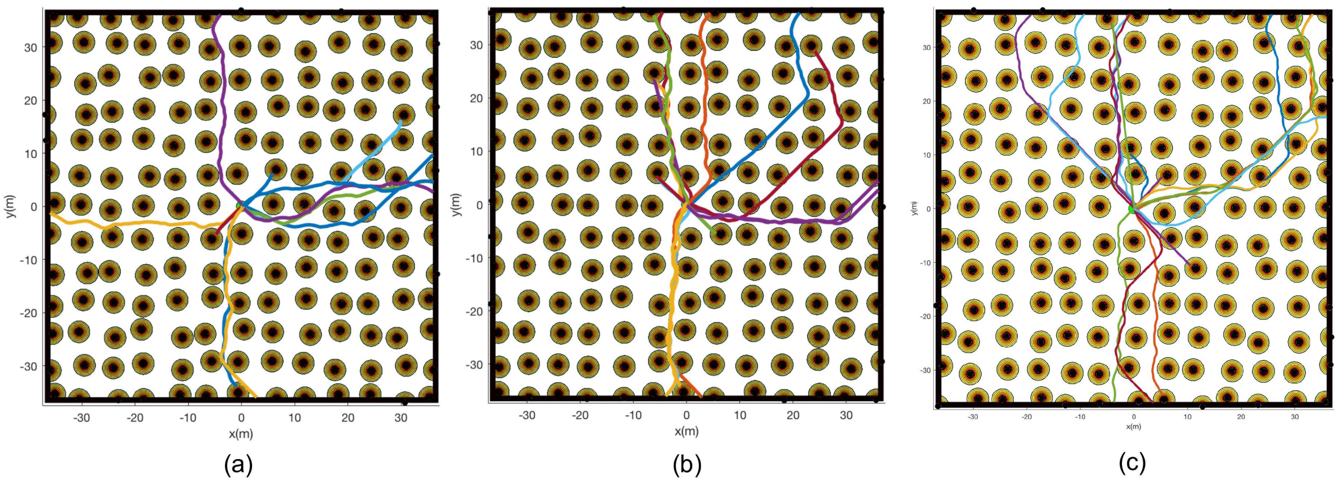

Fig. 9: Results of our learned DQN-policy on three different forests of cylinders. Robot starts from center of the map with a random orientation. Each trajectory has a different color

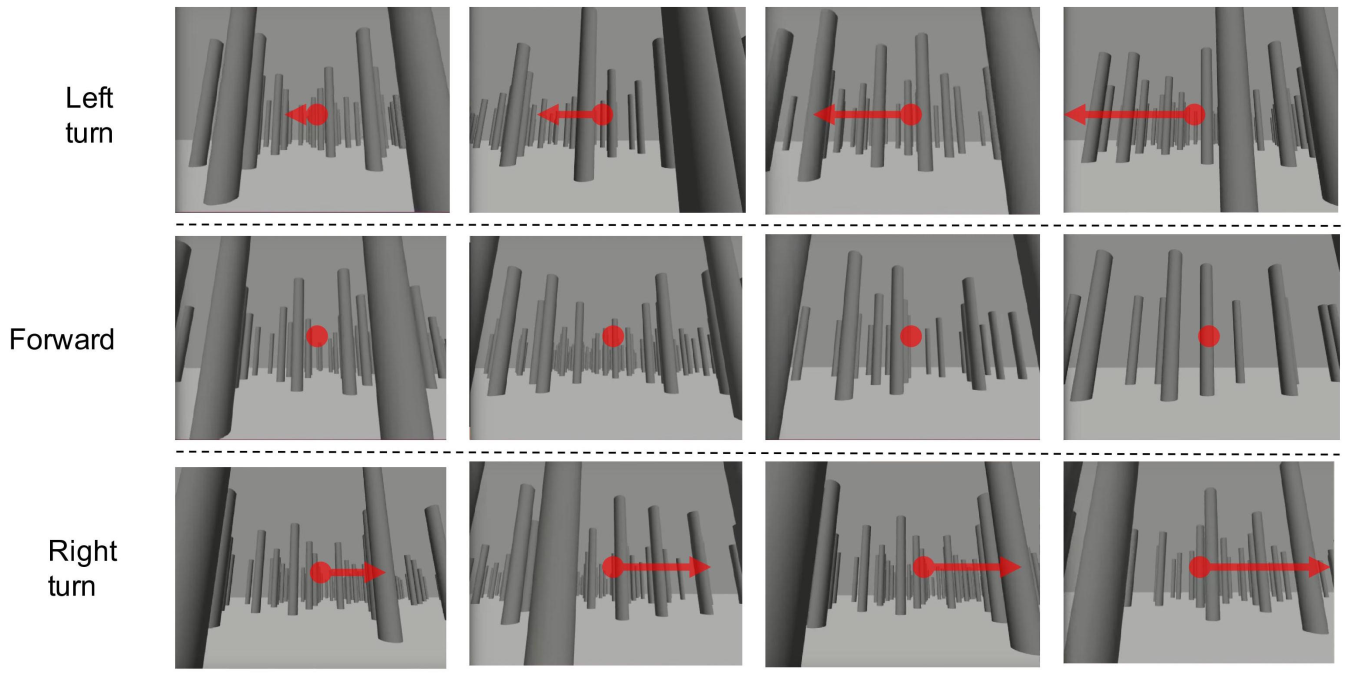

Fig. 10: Independent snapshots from various test runs depicting the optimal action corresponding to our learned policy. The magnitude of the red arrow is proportional to the magnitude of output yaw velocity being executed based on input image

REFERENCES

[1] T. Mori and S. Scherer, "First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1750–1757.

[2] S. P. Soundararaj, A. K. Sujeeth, and A. Saxena, "Autonomous indoor helicopter flight using a single onboard camera," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference*

Preprint submitted to 2018 IEEE International Conference on Robotics and Automation. Received September 15, 2017.

on. IEEE, 2009, pp. 5307–5314.

[3] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[4] 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, 2011.

[5] 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.

[6] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2520–2525.

[7] D. J. Webb and J. van den Berg, "Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 5054–5061.

[8] F. Poiesi and A. Cavallaro, "Detection of fast incoming objects with a moving camera." in *BMVC*, 2016.

[9] A. Saxena, S. H. Chung, and A. Y. Ng, "Learning depth from single monocular images," in *Advances in neural information processing systems*, 2006, pp. 1161–1168.

[10] D. Dey, K. S. Shankar, S. Zeng, R. Mehta, M. T. Agcayazi, C. Eriksen, S. Daftry, M. Hebert, and J. A. Bagnell, "Vision and learning for deliberative monocular cluttered flight," in *Field and Service Robotics*. Springer, 2016, pp. 391–409.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[12] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," *arXiv preprint arXiv:1609.05521*, 2016.

[13] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N. Siddharth, and P. H. Torr, "Playing Doom with SLAM-Augmented Deep Reinforcement Learning," *arXiv preprint arXiv:1612.00380*, 2016.

[14] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, pp. 2149–2154.

[15] R. M. Murray and S. S. Sastry, "Nonholonomic motion planning: Steering using sinusoids," *IEEE Transactions on Automatic Control*, vol. 38, no. 5, pp. 700–716, 1993.

[16] J. Michels, A. Saxena, and A. Y. Ng, "High speed obstacle avoidance using monocular vision and reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 593–600.

[17] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," *arXiv preprint arXiv:1704.05588*, 2017.

[18] F. Sadeghi and S. Levine, "(CAD)^{2}RL: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[20] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

Preprint submitted to 2018 IEEE International Conference on Robotics and Automation. Received September 15, 2017.