(CAD)^{2}RL: Real Single-Image Flight without a Single Real Image

Fereshteh Sadeghi[1] and Sergey Levine[2]

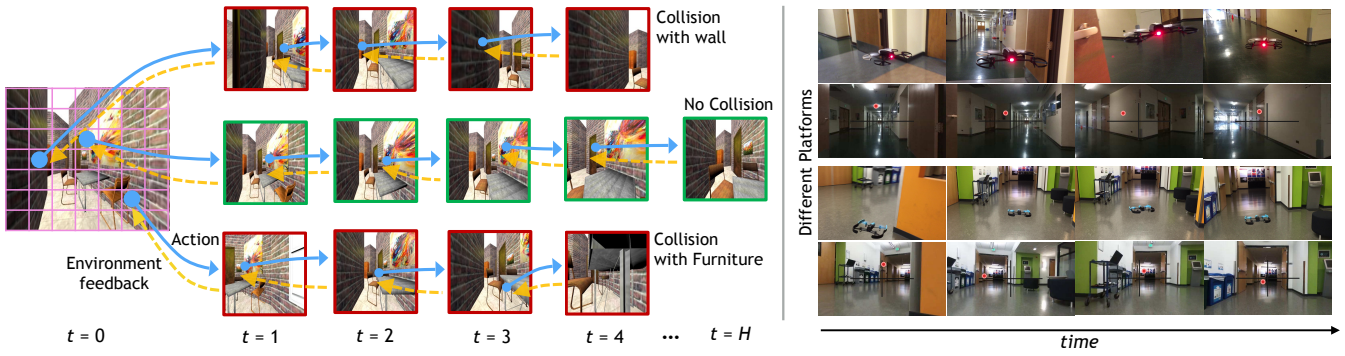

Fig. 1. We learn an autonomous indoor flight controller, **(CAD)^{2}RL**, using a novel **C**ollision **A**voidance via **D**eep **R**einforcement **L**earning algorithm which is entirely trained in a simulated environment constructed by 3D **CAD** models. Left: (CAD)^{2}RL uses *single image* inputs from a monocular camera, is exclusively trained via CAD simulations and does not see any real images at the training time. Right: (CAD)^{2}RL generalizes well to real indoor flight.

***Abstract*— We propose (CAD)^{2}RL, a flight controller for Collision Avoidance via Deep Reinforcement Learning that can be used to perform collision-free flight in the real world although it is trained entirely in a 3D CAD model simulator. Our method uses only single RGB images from a monocular camera mounted on the robot as the input and is specialized for indoor hallway following and obstacle avoidance. In contrast to most indoor navigation techniques that aim to directly reconstruct the 3D geometry of the environment, our approach directly predicts the probability of collision given the current monocular image and a candidate action. To obtain accurate predictions, we develop a deep reinforcement learning algorithm for learning indoor navigation, which uses the actual performance of the current policy to construct accurate supervision. The collision prediction model is represented by a deep convolutional neural network that directly processes raw image inputs. Our collision avoidance system is entirely trained in simulation and thus addresses the high sample complexity of deep reinforcement learning and avoids the dangers of trial-and-error learning in the real world. By highly randomizing the rendering settings for our simulated training set, we show that we can train a collision predictor that generalizes to new environments with substantially different appearance from the training scenarios. Finally, we evaluate our method in the real world by controlling a real quadrotor flying through real hallways. We demonstrate that our method can perform real-world collision avoidance and hallway following after being trained exclusively on synthetic images, without ever having seen a single real image at the training time. For supplementary video see: https://fsadeghi.github.io/CAD2RL**

[1]Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195 fsadeghi@cs.washington.edu

[2]Department of Electrical Engineering and Computer Science, University of California, Berkeley, Berkeley, CA 94709 svlevine@eecs.berkeley.edu

I. Introduction

Indoor navigation and collision avoidance is one of the basic requirements for robotic systems that must operate in unstructured open-world environments, including quadrotors, mobile manipulators, and other mobile robots. Many of the most successful approaches to indoor navigation have used mapping and localization techniques based on 3D perception, including SLAM [3], depth sensors [33], stereo cameras [27], and monocular cameras using structure from motion [6]. The use of sophisticated sensors imposes additional costs on a robotic platform, which is a particularly prominent issue for weight and power constrained systems such as lightweight aerial vehicles. Monocular cameras, on the other hand, require 3D estimation from motion, which remains a challenging open problem despite considerable recent progress [10], [16]. In this paper, we explore a learning-based approach for indoor navigation, which directly predicts the safety of candidate motor commands from monocular images, without attempting to explicitly model or represent the 3D structure of the environment. In contrast to previous learning-based navigation work [5], our method uses reinforcement learning to obtain supervision that accurately reflects the actual probabilities of collision, instead of separating out obstacle detection and control. The probability of future collision is predicted from raw monocular images using deep convolutional neural networks.

Using reinforcement learning to learn collision avoidance, especially with high-dimensional representations such as deep neural networks, presents a number of major challenges. First, reinforcement learning tends to be data-intensive, making it difficult to use with platforms such as aerial vehicles, which have limited flight time and require time-consuming battery changes. Second, reinforcement learning relies on

trial-and-error, which means that, in order to learn to avoid collisions, the vehicle must experience at least a limited number of collision during training. This can be extremely problematic for fragile robots such as quadrotors.

A promising avenue for addressing these challenges is to train policies in simulation, but it remains open question whether simulated training of vision-based policies can generalize effectively to the real world. In this work, we show that we can transfer indoor obstacle avoidance policies based on monocular RGB images from simulation to the real world by using a randomized renderer. Our renderer forces the network to handle a variety of obstacle appearances and lighting conditions, which makes the learned representations invariant to surface appearance. As the result, the network learns geometric features that enables it to robustly detect open spaces.

In contrast to prior work on domain adaptation [26], [31], our method does not require even a single real world image during training. We demonstrate that this approach can effectively navigate through real-world hallways by a real quadrotor using only a monocular camera, without depth or stereo. By training entirely in simulation, we can also use simple and stable reinforcement learning methods that exploit the ability to reset the environment to any state. Figure 1 shows a diagram of our $(CAD)^2RL$ algorithm. The algorithm evaluates multiple actions at each state using the current policy, producing dense supervision for the Q-values at that state. Training the Q-function to regress onto these Q-values then corresponds to simple supervised learning. This algorithm sidesteps many of the hyperparameter tuning challenges associated with conventional online RL methods, and is easy to parallelize for efficient simulated training.

For training, we designed a collection of synthetic 3D hallways that can be used to generate large datasets of randomized scenes, with variable furniture placement, lighting, and textures. These hallways can be used both for training and for benchmarking obstacle avoidance RL methods. Our simulated comparative evaluation shows that our approach outperforms several baselines, as well as a prior learning-based method that predicts turning directions [11]. Our real-world experiments demonstrate the potential for purely simulation-based training of deep neural network navigation policies. Although the policies trained entirely in simulation do experience some number of collisions in the real world, they outperform baseline methods and are able to navigate effectively around many kinds of obstacles, using only monocular images as input. We therefore conclude that simulated training is a promising direction for learning real-world navigation for aerial vehicles.

II. Related Work

Any robotic system that must traverse indoor environments is required to perform basic collision avoidance. Standard methods for collision-free indoor navigation take a two step approach to the problem: first map out the local environment and determine its geometry, and then compute a collision-free path for reaching the destination [29]. This approach benefits from independent developments in mapping and localization as well as motion planning [28], [20], [4]. The 3D geometry of the local environment can be deduced using SLAM with range sensors [3], consumer depth sensors [33], [12], stereo camera pairs [27], as well as monocular cameras [6]. Reconstruction from monocular images is particularly challenging, and despite considerable progress in recent years [10], [16], remains a difficult open problem. In another recent approach, called IM2CAD, CAD model of a room is generated from a single RGB image [14]. IM2CAD produces neat and exact 3D models but the computational overhead makes it less suitable for autonomous indoor flight where quick inference for finding open spaces is critical rather than categorical and exact 3D models. It is worthwhile to mention that, the synthetic data generated by [14] can effectively be used for various robotics simulations.

In our work, we sidestep the challenges of 3D reconstruction by developing a learning algorithm that can directly predict the probability of collision, without an explicit mapping phase. Learning has previously been used to detect obstacles for indoor flight [5], [15], as well as to directly learn a turn classifier for outdoor forest trail following [11]. In contrast to the work of [5], our method directly learns to predict the probability of collision, given an image and a candidate action, without attempting to explicitly detect obstacles. However, our approach still affords considerable flexibility in choosing the action: a higher-level decision making system can choose any collision-free action based, for example, on a higher-level navigational goal. This is in contrast to the prior work on trail following, which simply predicts the action that will cause the vehicle to follow a trail [11]. Furthermore, unlike [11], our method does not require any human demonstration or teleoperation at training time.

Besides presenting a deep reinforcement learning approach for collision avoidance, we describe how this method can be used to learn a generalizable collision predictor in simulation, such that it can then generalize to realistic and systematically different scenarios as well as the real world. Simulated training has been addressed independently in the computer vision and robotics communities in recent years. In computer vision, a number of domain adaptation methods have been proposed that aim to generalize perception systems trained in a source domain into a target domain [32], [13]. In robotics, simulation to real-world generalization has been addressed using hierarchies of multi-fidelity simulators [9], priors imposed on Bayesian dynamics models [8]. At the intersection of robotics and computer vision, several works have recently applied domain adaptation techniques to perform transfer for robotic perception systems [31], [26], [25]. In contrast to these works, our method does not use any *explicit* domain adaptation. Instead, we show how the source domain itself can be suitably randomized in order to train a more generalizable model, which we experimentally show can make effective predictions on a range of systematically different target domains.

Our method combines deep neural networks for processing raw camera images [18] with reinforcement learning. Such

deep reinforcement learning algorithms have previously been explored in context of Q-iteration [24], and more recently for online Q-learning using temporal-difference algorithms [19]. We propose a simple and stable deep reinforcement learning for indoor collision avoidance that is well suited for our simulated training environment.

III. Collision Avoidance via Deep Reinforcement Learning

Our aim is to choose actions for indoor navigation that avoid collisions with obstacles, such as walls and furniture. While we do not explicitly consider the overall navigation objective (e.g. the direction that the vehicle should fly to reach a goal), we present a general and flexible collision avoidance method that predicts which actions are more or less likely to result in collisions, which is straightforward to combine with higher-level navigational objectives. The input to our model consists only of monocular RGB images, without depth, IMU inputs, or other sensors, making it suitable for low-cost, low-power platforms. Formally, let $\mathbf{I}_t$ denote the camera observation at time t, and let a_t denote the action, which we will define in Section III-A. The goal of the model is to predict $P(C|\mathbf{I}_t, a_t)$, where C is the discounted expectation of a collision event:

$$P(C|\mathbf{I}_t, a_t) = \sum_{s=t}^{t+H} \gamma^{s-t} P(c_s|\mathbf{I}_s, a_s) \quad (1)$$

where $\gamma \in (0, 1)$ is the discount, c_s is an indicator for a collision at time s, and future actions are assumed to be chosen by the current policy. The horizon H should ideally be ∞, but in practice is chosen such that γ^H is small.

Collisions are assumed to end the episode, and therefore can occur only once, ensuring that the sum is always in the range $(0, 1]$. This can be interpreted as the probability of a Markov chain entering an absorbing collision state, with a probability γ of entering a non-collision absorbing state at each time step.

Our model for $P(C|\mathbf{I}_t, a_t)$ is learned using reinforcement learning, from the agent's own experience of navigating and avoiding collisions. Once learned, the model can be used to choose collision-free actions a_t simply by minimizing $P(C|\mathbf{I}_t, a_t)$. Training is performed entirely in simulation, where we can easily obtain distances to obstacles and simulate multiple different actions to determine the best one. By randomizing the simulated environment, we can train a model that generalizes effectively to domains with systematic discrepancies from our training environment. We will first describe the formulation of our model and our reinforcement learning algorithm, and then present details of our simulated training environment.

A. Perception-Based Control

Our perception-based policy uses an action representation that corresponds to positions in image space. The image $\mathbf{I}_t$ is discretized into an $M \times M$ grid of bins, and each bin has a corresponding action, such that a_t is simply the choice of bin. Once chosen, the bin is transformed into a velocity command $\mathbf{v}_t$, which corresponds to a vector from the camera location through the image plane at the center of the bin a_t, normalized to a constant target speed. Intuitively, choosing a bin a_t causes the vehicle to fly in the direction of this bin in image space. A greedy policy can use the model $P(C|\mathbf{I}_t, a_t)$ to choose the action with the lowest probability of collision. We will use $\pi(\mathbf{I}) = a$ to denote this policy, such that $P_\pi(C|\mathbf{I}) = P(C|\mathbf{I}, \pi(\mathbf{I}))$.

This representation provides the vehicle with enough freedom to choose any desired navigation direction, ascend and descent to avoid obstacles, and navigate tight turns. One advantage of this image-space grid action representation is the flexibility that it provides for general navigational objectives, since we could easily choose the bin using a higher-level navigational controller, subject to the constraint that the probability of collision not exceed some user-chosen threshold. However, in order to evaluate the method in our experiments, we simply follow the greedy strategy.

B. Initialization via Free Space Prediction

In order to initialize our model with a reasonable starting policy, we use a heuristic pre-training phase based on collision detection. In this pretraining phase, the model is trained to predict $P(l|\mathbf{I}_t, a_t)$, where $l \in \{0, 1\}$ is a label that indicates whether a collision detection raycast in the direction $\mathbf{v}_t$ corresponding to a_t intersect any obstacle. The raycast has a fixed length, which we chose to be 1 meter. This is essentially equivalent to thresholding the depth map by one meter.

This initialization phase roughly corresponds to the assumption that the vehicle will maintain a predfined constant velocity $\mathbf{v}_t$. The model, which is represented by a fully convolutional neural network as described in Section III-D, is trained to label each bin with the collision label l, analogously to recent work in image segmentation [7]. The labels are obtained from our simulation engine, as described in Section IV.

C. Reinforcing Collision Avoidance

The initial model can estimate free space in front of the vehicle, but this does not necessarily correspond directly to the likelihood of a collision: the vehicle might be able to maneuver out of the way before striking an obstacle within 1 meter, or it may collide later in the future even if there is sufficient free space at the current time step, for example because of a narrow dead-end. We therefore use deep reinforcement learning to finetune our pretrained model to accurately represent $P(C|\mathbf{I}_t, a_t)$, rather than $P(l|\mathbf{I}_t, a_t)$.

To this end, we simulate multiple rollouts by flying through a set of training environments using our latest policy. Our score map of $M \times M$ bins, explained in III-A, determines the space of actions. Based on our score map, we consider a total of M^2 actions $\mathcal{A} = \{a_1, ..., a_{M^2}\}$ that can be taken after perceiving each observation $\mathbf{I}$. We start each episode by placing the agent at a random location and with random orientation and generate a rollout of size K, given by $(\mathbf{I}_0, a_0, \mathbf{I}_1, ..., a_{K-1}, \mathbf{I}_K)$. The actions a_t are

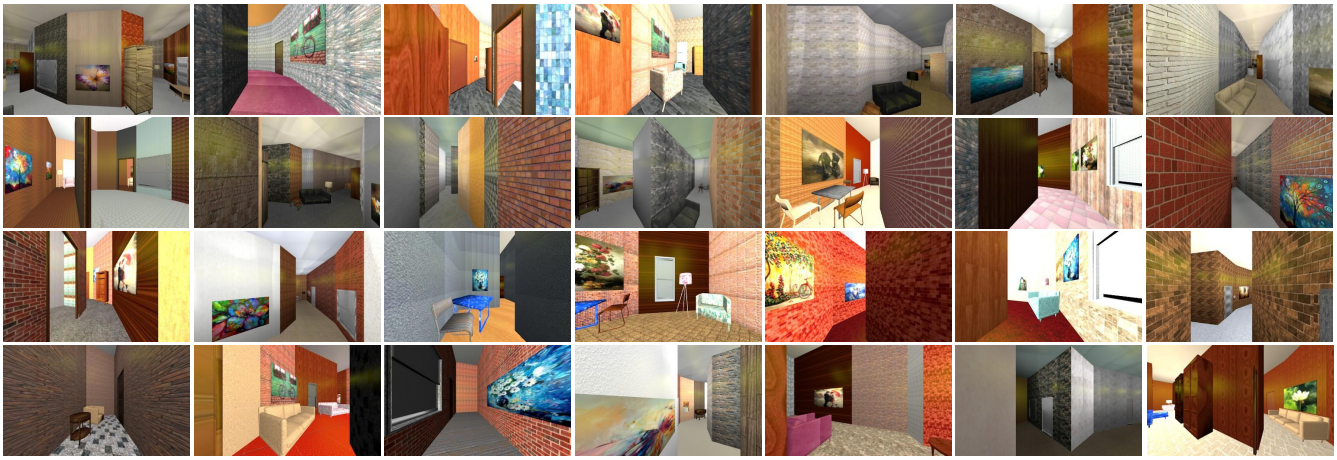

Fig. 2. Visualization of our training environments. The rendering uses randomized textures, lighting conditions, and furniture placement to create a diverse range of visual scenes.

selected according to the policy π after each observation $\mathbf{I}_t$. For each observation $\mathbf{I}_t$ encountered along the episode, we perform $M \times M$ additional rollouts using the policy π for every possible action a_t that can be taken at that time step, and evaluate the return of a_t according to Equation (1). Note that this corresponds to policy evaluation with the cost function $c(\mathbf{I}_t, a_t) = P(c_t|\mathbf{I}_t, a_t)$. Since evaluating Equation (1) requires rolling out the policy for H steps for every action, we choose $H = 5$ to reduce computation costs, and instead use a simple approximation to provide smooth target values for $P(C|\mathbf{I}_t, a_t)$. If the there is no collision within H steps, we use $P(c_{t+H}|\mathbf{I}_t, a_t) = c/d_{t+H}$, where d_{t+H} is the distance to the nearest obstacle at time $t + H$, and c is the radius of the vehicle, such that $c/d_{t+H} = 1$ when the vehicle is in collision. Our approximation can be interpreted as setting a terminal value at the last time step $t + H$, which smoothly drops from 1 based on the distance to the nearest obstacle.

This policy evaluation phase provides us with a dataset of observation, action, and return tuples $(\mathbf{I}_t, a_t, P(C|\mathbf{I}_t, a_t))$, which we can use to update the policy. Since we evaluate every action for each image $\mathbf{I}_t$, the dataset consists of densely labeled images with labels in the range $[0, 1]$, similarly to the datasets during pretraining, but the labels now correspond to the probability of collision for the current policy π, rather than free space indicators.

Our method can be interpreted as a modification of fitted Q-iteration [24], in the sense that we iteratively refit a Q-function estimator to samples, as well as a variant of modified policy iteration (MPI) [22], in the sense that we estimate Q-values using multi-step rollouts of the current policy. In this interpretation, $P(C|\mathbf{I}_t, a_t)$ is the Q-function and $P(c_t|\mathbf{I}_t, a_t)$ is the cost function. However, the particular details of the approach, including the evaluation of each action at each state, are specifically designed for our simulated training setup to exploit the capabilities of the simulation and provide for a simple and stable learning algorithm. We perform rollouts in simulated training hallways. This allows us to perform multiple rollouts from the state at each time step, perform ground truth collision detection raycasts for pretraining, and removes concerns about training-time collisions. Unlike conventional RL methods that perform rollouts directly in the test environment [19], we perform rollouts in simulated training hallways. However, this also means that our model must have generalization from the simulated training hallways to real-world environments at test time. To that, we developed a randomized simulated training environment, which we describe in the next section.

D. Network Architecture

In order to represent the Q-function and the initial open space predictor, use a deep fully convolutional neural network, built on the VGG16 [30] architecture following [7]. The output score map corresponds to a grid of 41×41 bins, which constitutes the action space for deep reinforcement learning. The network is trained with stochastic gradient descent (SGD), with a cross-entropy loss function.

IV. Learning from Simulation

Conventionally, learning-based approaches to autonomous flight have relied on learning from demonstration [1], [2], [21], [25]. Although the learning by demonstration approach has been successfully applied to a number of flight scenarios, the requirement for human-provided demonstrations limits the quantity and diversity of data that can be used for training. Since dataset size has been demonstrated to be critical for the success of learning methods, this likely severely limits the generalization capacity of purely demonstration-based methods. If we can train flight controllers using larger and more diverse datasets collected autonomously, we can in principle achieve substantially better generalization. However, in order to autonomously learn effective collision prediction models, the vehicle needs to see enough examples of collisions during training to build an accurate estimator. This is problematic in real physical environments, where even a single collision can lead to damage or loss of the vehicle. To get the benefits of an autonomous learning from the agent's own experience and overcome the limitations of data collection in learning from demonstration method, we use a simulated training environment that is specifically designed to enable effective transfer to real-world settings.

We manually designed a collection of 3D indoor environments to form the basis of our simulated training setup. The environments were built using the Blender[1] open-source 3D modeling suite. Our synthetic dataset contains 24 different hallways, shown in Figure 3, which represent a variety of structures that can be seen in real hallways, such as long straight or circular segments with multiple junction connectivity, as well as side rooms with open or closed doors. We use 21 different items of furniture that commonly exist in the hallways (benches, chairs, etc). We have slightly randomized the size of these furniture to provide diversity. The walls are textured with randomly chosen textures, chosen from a pool of 200 possible textures (e.g. wood, metal, textile, carpet, stone, glass, etc.), and illuminated with lights that are placed and oriented at random. Pretraining images are generated by flying a simulated camera through the hallways with randomized height and random perturbations of the yaw angle, in order to provide a diversity of viewpoints. In all, we randomize textures, lighting, furniture placement (including placement and identity of furniture items), and camera position and angle.

The randomization of the hallway parameters produces a very large diversity of training scenes, a sample of which can be seen in Figure 2. Although the training hallways are far from being photo-realistic, the large variety of appearances allows us to train highly generalizable models, as we will discuss in the experimental evaluation. The intuition behind this idea is that, by forcing the model to handle a greater degree of variation than is typical in real hallways (e.g., wide ranges of lighting conditions and textures, some of which are realistic, and some not), we can produce a model that generalizes also to real-world scenes, which might be systematically different from our renderings. That is, the wider we vary the parameters in simulation, the more likely we are to capture properties of the real world somewhere in the set of all possible scenes we consider. Our findings in this regard are aligned with the results obtained in other recent works [23], which also used only synthetic renderings to train visual models, but did not explicitly consider wide-ranging randomization of the training scenes.

V. Experimental Results

To evaluate the performance of our proposed method we ran several experiments and compare against a set of baselines. We first provide the details of the experimental setup, evaluations criterion and the different baselines we compare against. Then we present experimental results on two different types of test environments. Finally, we discuss experimental results for real-world flight using the policies obtained with our method.

A. Experimental Setup

Our aim is to evaluate the performance of our trained policy in terms of the duration of collision free flight. To do this, we run continuous episodes that terminate upon experiencing a collision, and count how many steps are taken before a collision takes place. We set the maximum number of steps to a fixed number throughout each experiment.

[1] https://www.blender.org

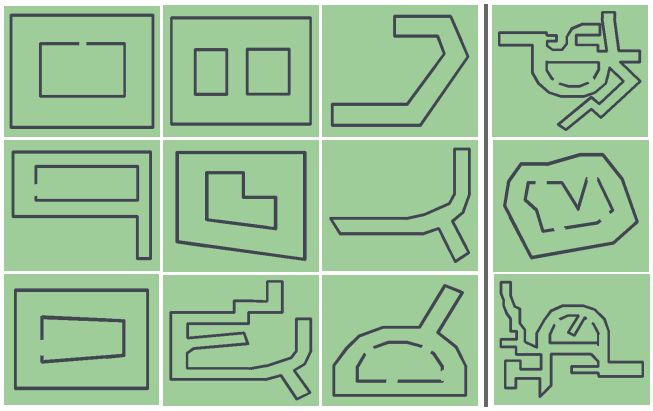

Fig. 3. Floor plans of the synthetic hallways. The first three column show the hallways used for training, while the last column shows the test hallways used for evaluation. Note that the test hallways are more complex.

An episode can begin in any location in the choice environment, but the choice of the initial position can have a high impact on the performance of the controller, since some parts of the hallway, such as dead ends, sharp turns, or doorways, can be substantially more difficult. Therefore, to make an unbiased evaluation and to test the robustness of the learned policies, we start each flight from a location chosen uniformly at random within the free space of each environment. We use random initialization points and keep them fixed throughout all experiments to provide a fair comparison between different methods, including prior work. In the experiments, the quadrotor has constant velocity during each flight, and we convert the number of steps to meters in order compute the distance traveled in each flight.

We evaluate performance in terms of the percentage of trials that reached a particular flight length. To that end, we report the results using a curve that plots the distance traveled along the horizontal axis, and the percentage of trials that reached that distance before a collision on the vertical axis. More formally, vertical axis represent $\sum_{i=1}^{|T|} (1 - \mathbb{1}(C_d(T_i)))$ where T denotes a trial and $C_d(T_i)$ is the event of collision happening for the ith trail at distance d in the x axis. This provides an accurate and rigorous evaluation of each policy, and allows us to interpret for each method whether it is prone to collide early in the flight, or can maintain collision-free flight at length. Note that achieving completely collision-free flight in all cases from completely randomized initial configurations is exceptionally difficult.

B. Methods

We compare the performance of our method with previous methods for learning-based visual obstacle avoidance. We describe the prior methods and baselines below.

1) Left, Right, and Straight (LRS) Controller: This method, based on [11], directly predicts the flight direction from images. The commands are discretized into three bins: "left," "right," or "straight," and the predictions are made by a deep convolutional neural network from raw camera images.

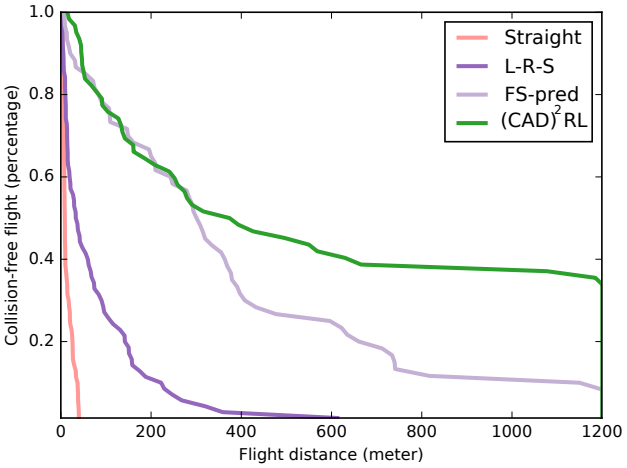

Fig. 4. Quantitative results on a realistically textured hallway. Our proposed approach (CAD)^{2}RL, outperforms the baselines

Prior work trained such a model using real-world images collected from three cameras carried manually through forest trails. One camera was pointed left, one right, and one straight, and left camera images were supervised as right turns, right images as left turns, and straight camera images as straight motion. We simulated the same training setup in our training environments, with the cameras placed 40 degrees from the forward axis, and used the same paths as in our model's pretraining to produce training data. The network was a finetuned VGG16 [30] model, pretrained with ImageNet classification. This method can be considered a human-supervised alternative to our autonomous collision avoidance policy, and we refer to it as "LRS."

2) Straight Controller: This lower bound baseline flies in a straight line without turning. In a long straight hallway, this baseline establishes how far the vehicle can fly without any perception, allowing us to ascertain the difficulty of the initialization conditions.

3) Ground Truth Free Space Controller: This baseline simulates a quadrotor equipped with perfect LIDAR range sensors or depth cameras. The performance of this baseline shows the upper-bound of the performance of a free-space prediction based controller. The policy always selects the most central free-space labeled bin in the spatial grid of the current image. Note that this does not always result in the best performance, since the behavior of this baseline is myopic, but it does serve to illustrate the difficulty of the test environments.

C. Realistic Environment Evaluation

As we explained in Section IV, we trained our policy on a set of synthetic 3D models of hallways and in a fully simulated environment with no real images. In order to evaluate how well such a model might transfer to a realistic environment, we used a realistic 3D mesh provided by [17]. In [17], the 3D mesh model of a building interior is captured by a backpack system that uses scan-matching-based localization algorithms combined with an extra stage of image-based alignment for texturing the meshes with natural images. As the result, evaluating on this data can provide us a close proxy of our performance in a real indoor environment. More specifically, we used the 3D model of Cory Hall on the UC Berkeley campus. This experiment also evaluates the generalization capability of our method in a systematically different environment than our training environments. Figure 6 shows the floorplan of this hallway, as well as several samples of its interior view. As can be seen from the floorplan, this environment contains several long corridors, various junctions, and small rooms connected via open doors or narrow hallways. Although the rendering does not perfectly reflect the appearance of the real world, the realistic textures are a substantial departure from our simple simulated training environments.

We generated 60 random initialization point from various locations in the hallways. These points are fixed and all baselines are evaluated on the same set of points so that their performance is directly comparable. Figure 5.a depicts the initialization points as red dots. The velocity of the quadrotor is fixed to 0.2 meters per time step in this experiment, and the maximum number of steps is set to 6000 which is equal to 1.2 kilometers.

In this experiment, we compare against LRS and the straight controller. We also report the performance of our base free space prediction (FS-pred) controller to analyze the improvement obtained by incorporating deep reinforcement learning. Note that in this experiment, we did not evaluate the ground truth free-space baseline, as we observed that some of the 3D meshes had non-exact alignment. This resulted in an unfair comparison, as the raycast produces noisy ground truth masks that confused the ground truth controller.

1) Quantitative Evaluation: Figure 4 summarizes the performance of our proposed (CAD)^{2}RL method compared with other baselines. Our method outperforms the prior methods and baselines in this experiment by a substantial margin. Qualitatively, we found that the LRS method tends to make poor decisions at intersections, and the coarse granularity of its action representation also makes it difficult for it to maneuver near obstacles. (CAD)^{2}RL is able to maintain a collision-free flight of 1.2 kilometers in about 40% of the cases, and substantially outperforms the model that is simply trained with supervised learning to predict 1 meter of free space in front of the vehicle. This experiment shows that although we did not use real images during training, our learned model can generalize to substantially different and more realistic environments, and can maintain collision-free flight for relatively long periods.

2) Qualitative Evaluation: To be able to qualitatively evaluate and compare the performance and behavior of (CAD)^{2}RL with our perception based controller and the LRS method, we visualized the trajectory of the flights overlaid on the floor-plan of the hallway as shown in Figure 5. For this purpose, we sorted the trajectories of each method based on the traveled distance and selected the top 25 longest flights from each method. The trajectories are color coded based on the color wheel shown in the bottom left of the figure and demonstrate the flight direction at each point. The trajectories traveled by the LRS method are depicted in

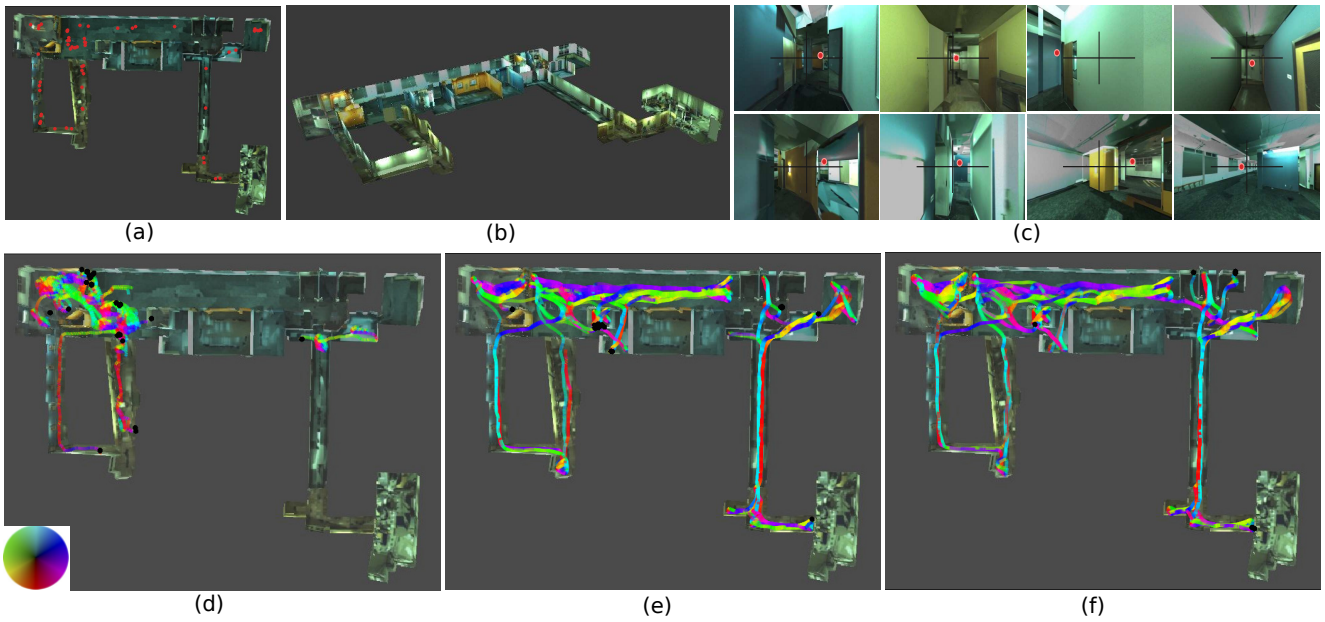

Fig. 5. Qualitative results on a realistically textured hallway. Colors correspond to the direction of trajectory movement at each point in the hallway as per the color wheel. (a) Red dots show flight initialization points (b) Overlook view of the hallway (c) Red dots show the control points produced by (CAD)^{2}RL. (d) LRS trajectories (e) Perception controller (FS-pred) trajectories (f) (CAD)^{2}RL trajectories.

Figure 5.d while the trajectories of our FS-pred controller and (CAD)^{2}RL controller are depicted in Figure 5.e and Figure 5.f, respectively. The black dots indicate the locations of the hallway where collisions occurred. This visualization shows that (CAD)^{2}RL could maintain a collision-free flight in various locations in the hallway and has fewer collisions at the dead-ends, corners, and junctions compared with the other two methods. By comparison, the LRS method often experienced collisions in corners, and was more vulnerable to bad initial locations. The policy trained with free space prediction rather that reinforcement learning outperformed the LRS method, but often became trapped in rooms or fail near junctions and corners. This illustrates that the controller trained with reinforcement learning was able to acquire a better strategy for medium-horizon planning, compared to the directly supervised greedy methods.

D. Synthetic Environment Test

This experiment is aimed at comparing (CAD)^{2}RL in the presence of different hallway geometries, distractors, and obstacles, in synthetic hallways that are distinct from the ones used but similar in visual style (see Figure 3). Note that the test hallways were intentionally designed to be larger and more challenging. We rendered all images at the test time using a randomization of 100 different test textures that were not seen at the training time. We tested our method and the prior and baseline methods in two conditions: in the first condition, the hallways contained randomly placed furniture, and in the second, no furniture was present. Both scenarios have fixtures such as open or closed doors, windows, and paintings, but the hallways with furniture provide an additional challenge due to the more complex geometry, which is substantially more elaborate than the scanned hallway used in the previous section. We randomly sampled 100 random locations as initialization point. These points were selected uniformly from challenging locations such as junctions as well as less challenging locations in the middle of hallways. The velocity was 0.3 meters per step. Figure 6 compares the performance of our method compared with other baselines in each of the "with furniture" and "without furniture" test scenarios. (CAD)^{2}RL consistently outperforms the other baselines, as well as the model trained with supervised learning for free space prediction, FS-pred. In the hallways that do not contain any furniture, the ground truth free space baseline (FS-GT) obtains the best performance, while the presence of furniture in the second test scenario effectively reduce its performance due to the greedy strategy. In both scenarios, (CAD)^{2}RL has the highest performance among the learning-based methods.

E. Free Space Prediction Evaluation

In this experiment, we are interested to see how well our free space prediction model can detect free spaces and obstacles compared with its performance on the synthetic images. To this end, we used the simulator to compute the mask of Free Space (FS)/Obstacle(O) of 4k rendered frames which were sampled uniformly along the hallways. For the performance metrics, we use precision and Jaccard similarity. The precision shows the ratio of correctly labeled pixels as corresponding to "FS" or "O". The Jaccard similarity is the intersection over union of the result and ground truth labels for both free-space and obstacle labels. Table V-E summerizes the obtained results. The first row of the table shows the results obtained for the images rendered in from our test hallways with test textures. The second row shows the results obtained on the photo realistic images of [17]. Although, there is a 10% precision gap between the performance on synthetic images and photo-realistic images,

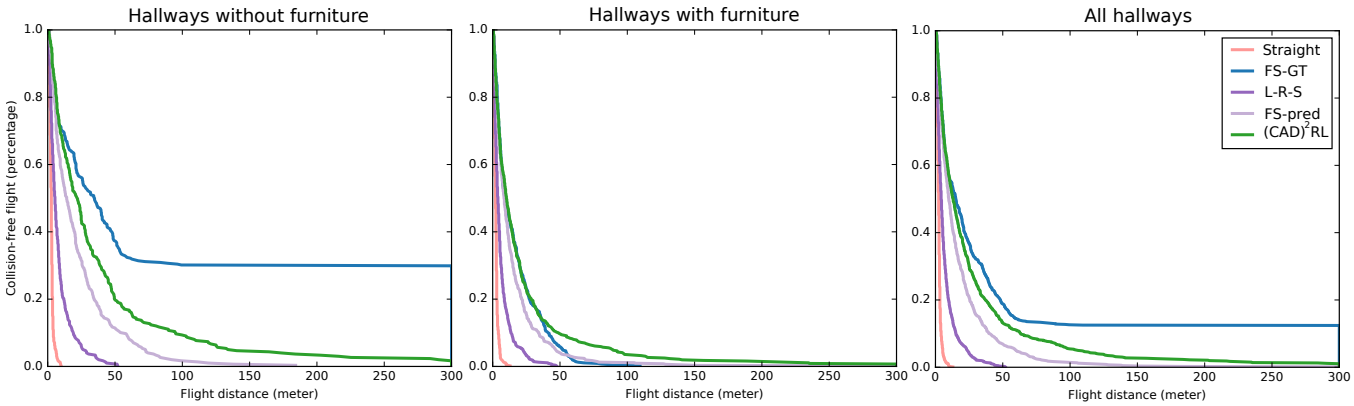

Fig. 6. Quantitative results on the simulated test hallways. Aside from the upper bound baseline FS-GT, which uses ground truth collision raycasts, our method (CAD)^{2}RL, achieves the longest collision-free flights.

TABLE I
PIXEL-WISE FREE SPACE PREDICTION RESULTS.

	Precision	Jaccard(FS)	Jaccard(O)
Synthetic hallways	90.38	56.99	87.26
Realistic hallways	80.33	63.39	63.30

which is due to the domain shift, the obtained precision results on the unseen photo-realistic images is high, i.e. 80%. Note that our synthetic hallways are much narrower than the real hallways of [17]. This results in smaller free-space areas and larger obstacle areas in the synthetic images compared with [17] where images have more balanced distribution of free-space vs obstacles. This results in lower Jaccard(FS) in the synthetic images.

F. Real World Flight Experiments

We evaluated our learned collision avoidance model by flying a drone in real world indoor environments. These flights required flying through open spaces, navigating hallways, and taking sharp turns, while avoiding collisions with furniture, walls, and fixtures. Since the aim of our experiments was to evaluate the potential for policies trained entirely in simulation to transfer to the real world, these experiments used the same models as the ones discussed in the previous sections, which were trained entirely in simulation without being exposed to a single real-world image.

We used the Parrot Bebop drone controlled via the ROS Bebop autonomy package[2]. We ran real flight experiments using two different drone platforms: the Parrot Bebop 1.0 and the Bebop 2.0. Although these two drones have similar SDK, they have different physical specifications in terms of dimensions, weight, and maximum speed. Note that the images produced by the onboard drone camera are center cropped to remove the fish-eye effect, and thus the objects appear closer than they really are.

Figure 7 shows the sequence of images captured by the flying drone in various scenarios. The red dots show the action computed by our policy. Our controller can successfully navigated the drone throughout free spaces while avoiding collision. Due to imperfect stabilization, turbulence and air currents, the drone may sometimes drift to the left or right, and our controller can recover from these situations by stabilizing the drone at each time step based on the current image observation. This suggests that, though our model is fully trained in simulation without seeing any real images or using any human provided demonstration, it has the capability to generalize to real-world images and conditions. In the following sections we evaluate the real flight performance both quantitatively and qualitatively.

1) Quantitative Evaluation: To quantitatively evaluate the performance of (CAD)^{2}RL in the real world, we ran controlled experiments on the task of hallway following. We fixed all the testing conditions while navigating the drone with either of the (CAD)^{2}RL and a baseline controller. The testing conditions include the initial velocity, angular speed, drone platform and the test environment. As was concluded from the experiments in section V-C and V-D, FS-pred was the strongest baseline, and we therefore included it as a comparison in this experiment. We ran experiments in two different buildings, Cory Hall and SDH (Sutardja Dai Hall), both located on the UC Berkeley campus. These buildings have considerably different floor plans, wall textures, and lighting conditions, as can be seen in Figure 7.c and Figure 7.d. Our testing environment in Cory Hall contained three turns and two junctions, while the SDH test environment had one turn and one junction. The width of the Cory hall hallway is ~ 3 meters while the SDH hallway is ~ 2 meters wide.

Table V-F.1 summarizes the results. The safe flight time is given by the average length of a collision free flight in terms of distance or time between collisions. (CAD)^{2}RL experienced fewer collisions and has longer expected safe flight. This suggests that the (CAD)^{2}RL policy makes fewer mistakes and is more robust to perturbations and drift. Both methods performed better in Cory Hall, since SDH has narrower hallways with glossy textureless walls as well as stronger air currents. While we fixed the test environment and the flying speed, the traveled distance and time is slightly different from one algorithm to another due to the fact that the algorithms generated different commands and navigated the drone to slightly different locations in the hallways.

[2] http://bebop-autonomy.readthedocs.io

TABLE II
REAL WORLD FLIGHT RESULTS.

Environment	Traveled Distance (meters)	Travel Time (minutes)	Collision (per meter)	Collision (per minute)	Safe Flight (meters)	Safe Flight (minutes)	Total Collisions
Cory FS-pred	162.458	12.01	0.080	1.081	12.496	0.924	13
Cory $(CAD)^2RL$	163.779	11.950	0.0366	0.502	27.296	1.991	6
SDH FS-pred	53.492	4.016	0.130	1.742	7.641	0.573	7
SDH $(CAD)^2RL$	54.813	4.183	0.072	0.956	13.703	1.045	4

2) Qualitative Evaluation: For quantitative evaluation, we performed the real world flight in a number of different indoor scenarios. Here, we briefly explain each of the tested scenarios. The sequence of snapshots in these scenarios are demonstrated in Figure 7.

(a) Flying near furniture, around corners, and through a window: In this scenario, the drone starts from the end of a hallway that is connected to a small lounge area with chairs and a table. Initially, the controller is confused by the large free spaces near the lounge, turns away from the wall on the right, and travels towards the lounge. At this point, the drone loses track of the corridor and becomes trapped in the corner of the room. However, it detects an opening in the wall which is visually similar to an open doorway (or an open window), and successfully adjusts its height and flies through it. The drone then encounters a reflective glass door, which reflects the hallway behind it. Since no such structures were present during training, the reflective door fools the controller, causing the drone to crash into the door. Note that the controller navigates multiple structures that are substantially different, both visually and geometrically, from the ones encountered during simulated training. Adding reflective and translucent surfaces during training would likely further improve the robustness of the controller. This scenario is shown in Figure 7.a.

(b) Flying up a staircase: Here, the goal is to evaluate the generalization capability of the controller in understanding horizontally placed solid obstacles. A staircase provides a good example of this. To avoid colliding with such obstacles, the drone must move forward while increasing the altitude. As can be seen from the snapshots in the Figure 7.b, the controller produces actions that increase the altitude of the drone at each step along the staircase. Since we used an altitude limit for safety reasons, the drone only flew halfway up the staircase, but this experiment shows that the controller could effectively generalize to structures such as staircases that were not present during training.

(c) Navigating through narrow corridors: In this example, the drone follows the vanishing point to fly through a corridor and follow a corner. As can be seen in the Figure 7.c frames 1-6, there is a dead end at the junction, and the controller successfully takes a turn to avoid the dead end. The corridors in this test scenario are narrow (~ 2 meters), and there was a strong air current because of air conditioning.

(d) Navigating through junctions and fly through rooms: Similar to the example (c), the drone navigates through the hallway and around corners. At the end it enters a doorway which is connected to a study room. The controller successfully navigates the drone through the narrow door and the drone flies around chairs and desks without collision.

(e) Flying through a maze of random obstacles in a confined space: In this experiment, we built a small U-shaped maze out of low obstacles in the lab. This maze is built using several chairs with various appearances and pieces of lightweight cardboard and wooden boards with various colors. For a flying drone, a simple policy to avoid colliding with such ground level obstacles is to fly over them. To avoid this trivial solution, we limited the maximum altitude to 3 feet, and thus forced the drone to find a collision free path between the obstacles. Note that flying the drone at low altitude is challenging, as the air turbulence becomes significant and affects the drone's stability. The cardboard shifts due to air turbulence, and the open area is very narrow (~ 1 meter), making this a challenging test environment. The sequence in Figure 7.e shows that the controller successfully navigates the drone throughout the maze, making a turn near the red chair, and turning back into the maze, without colliding with any of the obstacles.

(f) Avoiding dynamic obstacles: In this scenario, the drone begins in the lab with no obstacles at the beginning, and an altitude of around 3 feet. We then place a chair in the path of the drone, as seen in frames 7-8 of Figure 7.f. The controller recovers and avoids an imminent collision with the chair, passing it on the left.

The above qualitative evaluation study shows the generalization capability of our trained model and demonstrates the extent of the maneuvering skills learned by $(CAD)^2RL$. Although our model is specifically trained for the task of hallway navigation, the limited number of furniture items present in simulation also force the policy to be robust to oddly shaped obstacles, and train it to change altitude to avoid collisions. Navigating through the obstacles in the scenarios (a), (b), (e), and (f) required collision avoidance with general obstacles and other than just walls. We observed that our model could perform reasonably well in these cases, and could often recover from its mistakes, though particularly novel situations proved confusing. For example, in scenario (a), the controller gets trapped in a corner because of a bad decision in the first step, and is eventually confused by the reflection of the corridor in a window, which was never seen during training. Also, flying the drone in constrained indoor environments is generally challenging, since the propellers create air turbulence that affects the drone stabilization and potentially introduces additional sources of error. This leaves

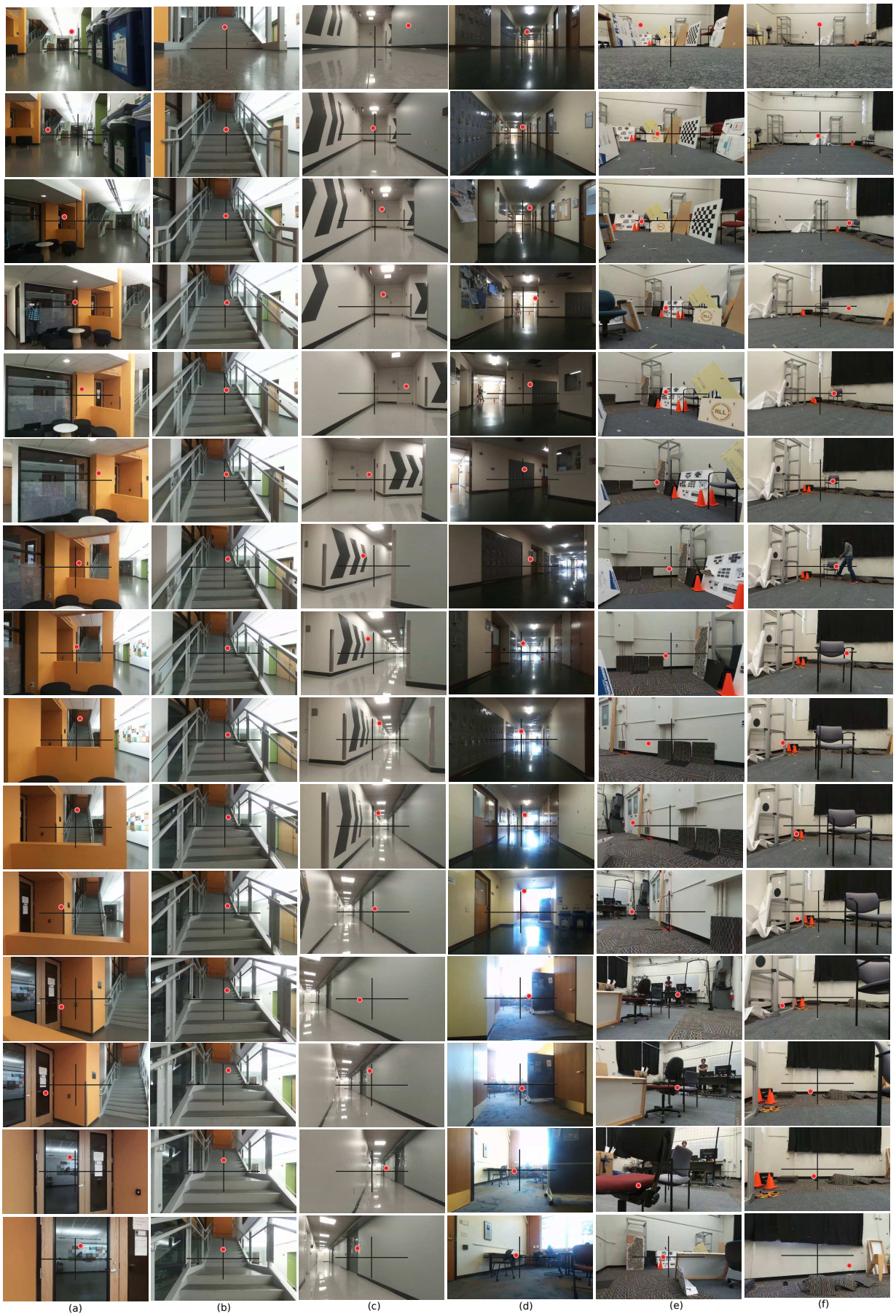

Fig. 7. Snapshots of autonomous flight in various real indoor scenarios. Frames are ordered from top to bottom. Red dots show the commanded flight direction produced by $(CAD)^2RL$. (a) Flying near furniture, around corners, and through a window; (b) Flying up a staircase; (c) Navigating through narrow corridors; (d) Navigating through junctions and fly through rooms; (e) Flying through a maze of random obstacles in a confined space; (f) Avoiding dynamic obstacles.

little margin for recovering from controller mistakes and motivates further research on developing robust controllers, as well as higher fidelity simulations with more degrees of variation that can be randomized at training time.

VI. Discussion and Future Work

In this paper, we presented a method for training deep neural network policies for obstacle avoidance and hallway following, using only simulated monocular RGB images. We described a deep reinforcement learning algorithm that is particularly well suited to this domain, which uses multiple branching rollouts to avoid the challenges associated with bootstrapping of the Q-function approximator. This produces a simple and stable algorithm suitable for learning in simulation. We also demonstrate that training on randomized simulated scenes produces a model that can successfully fly and avoid obstacles in the real world. Our simulated evaluation also shows that our method outperforms several baselines, as well as a prior end-to-end learning-based method.

Our aim in this work is specifically to evaluate the potential of policies trained *entirely* in simulation to transfer to the real world, so as to understand the benefits and limitations of simulated training. Of course, a reasonable approach to attain the best results in real environments is to combine simulated training with some amount of real data. Extending our approach via finetuning or domain adaptation is therefore a promising direction for future work that is likely to improve performance substantially.

Although we demonstrate considerable generalization to various target domains using only simulated training, this approach still requires considerable manual engineering to design suitable training environments. Promising future directions for addressing this limitation include procedural generation of randomized environment geometries and floor plans, as well as automated adaptation of the simulation to real-world data. Effective simulated training can make it practical to deploy powerful end-to-end trained navigation policies on real-world robotic systems, substantially improving reliability and runtime performance. Our work represents an early step in this direction, and future work might also explore similar simulated training methods for other tasks, including goal-driven navigation, interaction with dynamic obstacles, and more complex, unpredictable environments, such as cluttered outdoor scenes for flight and autonomous driving.

ACKNOWLEDGMENT

This work was made possible by an ONR Young Investigator Program Award and support from Google.

References

[1] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *IJRR*, 2010.
[2] P. Abbeel, A. Coates, M. Quigley, and A. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *NIPS*, 2006.
[3] A. Bachrach, R. He, and N. Roy. Autonomous flight in unstructured and unknown indoor environments. In *EMAV*, 2009.
[4] A. J. Barry and R. Tedrake. Pushbroom stereo for high-speed navigation in cluttered environments. In *ICRA*. IEEE, 2015.
[5] C. Bills, J. Chen, and A. Saxena. Autonomous mav flight in indoor environments using single image perspective cues. In *ICRA*, 2011.
[6] K. Celik, S. Chung, M. Clausman, and A. Somani. Monocular vision SLAM for indoor aerial vehicles. In *IROS*, 2009.
[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015.
[8] M. Cutler and J. P. How. Efficient reinforcement learning for robots using informative simulated priors. In *ICRA*, 2015.
[9] M. Cutler, T. J. Walsh, and J. P. How. Reinforcement learning with multi-fidelity simulators. In *ICRA*. IEEE, 2014.
[10] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *ECCV*, 2014.
[11] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 2016.
[12] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *International Journal of Robotics Research*, 31(5):647–663, Apr. 2012.
[13] J. Hoffman, S. Guadarrama, E. S. Tzeng, R. Hu, J. Donahue, R. Girshick, T. Darrell, and K. Saenko. Lsda: Large scale detection through adaptation. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *NIPS*. 2014.
[14] H. Izadinia, Q. Shan, and S. M. Seitz. IM2CAD. *arXiv preprint arXiv:1608.05137*, 2016.
[15] D. K. Kim and T. Chen. Deep neural network for real-time autonomous indoor navigation. *arXiv preprint arXiv:1511.04668*, 2015.
[16] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality ACM International Symposium on*. IEEE, 2007.
[17] J. Kua, N. Corso, and A. Zakhor. Automatic loop closure detection using multiple cameras for 3d indoor localization. In *IS&T/SPIE Electronic Imaging*, 2012.
[18] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.
[19] 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*, 2015.
[20] K. Mohta, V. Kumar, and K. Daniilidis. Vision based control of a quadrotor for perching on planes and lines. In *ICRA*, 2014.
[21] A. Punjani and P. Abbeel. Deep learning helicopter dynamics models. In *ICRA*, 2015.
[22] M. L. Puterman and M. C. Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 1978.
[23] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. Playing for data: Ground truth from computer games. *arXiv preprint arXiv:1608.02192*, 2016.
[24] M. Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning (ECML)*, 2005.
[25] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert. Learning monocular reactive uav control in cluttered natural environments. In *ICRA*. IEEE, 2013.
[26] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.
[27] K. Schmid, T. Tomic, F. Ruess, H. Hirschmller, and M. Suppa. Stereo vision based indoor/outdoor navigation for flying robots. In *IROS*, 2013.
[28] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar. Vision-based state estimation for autonomous rotorcraft mavs in complex environments. In *ICRA*, 2013.
[29] B. Siciliano and O. Khatib. *Springer handbook of robotics*. Springer Science & Business Media, 2008.
[30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
[31] E. Tzeng, C. Devin, J. Hoffman, C. Finn, X. Peng, S. Levine, K. Saenko, and T. Darrell. Towards adapting deep visuomotor representations from simulated to real environments. *arXiv preprint arXiv:1511.07111*, 2015.
[32] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko. Simultaneous deep transfer across domains and tasks. In *ICCV*, pages 4068–4076, 2015.
[33] Z. Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 2012.