# Learning Unmanned Aerial Vehicle Control for Autonomous Target Following

Anonymous CVPR submission

Paper ID 2639

## Abstract

*Deep reinforcement learning has been successfully applied to a range of challenging problems in which the learned policy maps raw sensory inputs such as images directly to actions. However, function approximators such as neural networks defined on high-dimensional observation spaces are difficult to train in safety-critic unstable physical systems because such systems can fail catastrophically before an effective policy has been learned. In this paper, we consider one such scenario in which the goal is to control an unmanned aerial vehicle (UAV) to consistently follow an unknown moving target. To acquire a strategy that combines perception and control, we represent the policy by a convolutional neural network. To reliably control the UAV, we develop a hierarchical approach that combines a model-free policy gradient method with a conventional feedback proportional-integral-derivative (PID) controller. Given raw sensor readings, the policy network provides high-level reference actions to the PID control loop while the PID controller maps the reference inputs to low-level actions. The neural network is trained by a combination of supervised learning from raw images and reinforcement learning from games of self-play. We evaluate the proposed method by learning a target-following policy for a simulated quadrotor using onboard vision sensors only.*

## 1. Introduction

In recent years we have witnessed the advent of a new era of robots, unmanned aerial vehicles (UAVs) or drones [8]. The recent commercial drone revolution has attracted an increasing number of research laboratories to work on small, affordable, human-friendly drones. The rapid development of drones significantly extends the potential for many civilian applications. Images from drones that are capable of flying meters above the ground provide a promising alternative between low-resolution satellite images and car-based images that are restricted to the human perspective. Among many other applications, flying drones can help to track animals and find people where ground clutter poses serious obstacles.

In this paper, we consider the problem of controlling a drone with very limited payload to autonomously follow a moving target. To be more specific, the drone is required to consistently track the moving target, using a passive monocular RGB camera as its only exteroceptive sensor, where both the drone position and the target position are assumed unknown. The traditional pipeline typically works as follows. First, a standalone perception module is used to estimate the target location. Then, a conventional controller, such as a proportional-integral-derivative (PID) controller, is designed and tuned to follow some manually defined control rules. The whole system consists of several hard-coded components without any learning capability and relies heavily on tuning by human experts to achieve good performance.

In contrast, deep reinforcement learning offers new promises for achieving human-level control in robotics. Several recent studies have shown that end-to-end reinforcement learning approaches are capable of learning high-quality control policies using generic neural networks [14, 16, 18]. However, using deep neural networks for quadrotor controllers that map image pixels and states directly to low-level rotor commands poses a number of unique challenges. From the control perspective, observations from the quadrotor's onboard sensors do not reveal the full state of the system. Important state information such as the position of the target must be inferred from raw camera images. Besides, quadrotors are underactuated and fragile systems with highly nonlinear dynamics, making a partially trained rotor level policy crash very frequently during the training process. A key question is how to obtain the best of both worlds: to enjoy the richness and flexibility of a self-improving policy by data-driven learning while retaining the stability of conventional controllers.

We propose to combine a deep neural network with a conventional PID controller in a hierarchical way such that the neural network is trained to estimate the reference inputs to the PID control loop while the PID controller maps the reference inputs to low-level rotor actions. Consequently, the whole system is less susceptible to instability that often

rises during the training process. We represent both the perception module and the control module using a single convolutional neural network. The visual processing layers of the network are first trained in a *supervised learning* manner. This provides fast, efficient learning updates based on high-quality gradients. Next, the whole network is trained in a *reinforcement learning* manner. This both adapts the perception module to optimize the task performance and adjusts the control module towards the goal of following the moving target. We demonstrate that the proposed approach can successfully learn to control the quadrotor to follow a moving target under various simulated circumstances.

## 2. Related Work

Deep convolutional neural networks have achieved unprecedented success in many visual domains such as image classification [12] and object detection [9]. Recently, this success has been extended to reinforcement learning problems that involve visual perception. The Deep Q-Network (DQN) [18] architecture has been demonstrated to successfully learn to play Atari games directly from raw pixels as observations. [16] extended DQN to continuous action domains by an actor-critic algorithm using deep function approximators. An asynchronous version of DQN was also proposed recently [17]. [21] applied a memory-based DQN to the Minecraft game. These model-free reinforcement learning methods make it possible to learn policies with minimal feature and policy engineering using raw observations directly as input to the neural networks. However, considering the usually long, data-hungry training paradigm and the relative frailty of robots especially autonomous drones, this approach poses significant challenges when applied to robotic domains.

On the other hand, model-based reinforcement learning methods, which learn an explicit dynamics model of the environment and then optimize the policy under this model, are more data efficient and have been successfully applied to robotics for various tasks such as object manipulation [6], ground vehicles [20] and helicopter flight [1, 2, 3]. [10] also proposed to accelerate Q-learning by incorporating learned models. However, such methods are typically applied to only one component of the robotic control pipeline and accept processed state input from an existing vision pipeline. [13, 15, 14] proposed a guided policy search method, which trains complex policies that combine perception and control by supervised learning in which the supervision is provided by a model-based trajectory optimization method. As a consequence, the learned policy is limited to be no better than the learned model and the model-based trajectory optimization method.

Extensive research work on the control and navigation of UAVs has been published recently. RGB-D sensors [4] were used in indoor environments for detailed scanning and followed by simultaneous localization and mapping (SLAM) methods. Vandapel et al. [27] proposed outdoor planning approaches for UAV navigation using lidar data. While most of these approaches design hand-crafted perception and control modules, several learning based methods have become powerful alternatives in recent years. Much effort has been devoted to autonomous helicopter flight [1, 2, 3] where apprenticeship learning is used for modeling the system dynamics and then combined with optimal control methods to design controllers. Neural networks have also been used to learn the dynamics of both helicopters and quadrotors in later work [23, 19]. [24] proposed an imitation learning based technique to learn a linear controller of the quadrotor's left-right velocity for obstacle avoidance based on visual input, where the features used are still hand-crafted. [28] incorporated model predictive control to the guided policy search framework and directly learned the low-level rotor commands for quadrotor navigation. However, the raw sensory input of that task is rather simple and has low dimensionality. Different from the existing work, in this paper we focus on the more challenging target following task in which both the raw camera observation and the system dynamics are more complicated without assuming that the target movement be known.

## 3. Preliminaries

In this section, we formulate the quadrotor control problem from the learning perspective and briefly review the Q-learning and policy gradient methods.

### 3.1. Problem Formulation

In reinforcement learning, the goal is to learn a policy $\pi_\theta(a_t|o_t)$ which allows an agent to choose actions $a_t$ in response to observations $o_t$ to control a dynamical system, so as to maximize the expected cumulative reward according to a reward function $r(s_t, a_t)$. The policy comes from some parametric class parameterized by $\theta$, such as a neural network. The system is defined by states $s_t$, actions $a_t$, and observations $o_t$. The system dynamics are defined by an initial state distribution $p(s_1)$ and a dynamics distribution $p(s_{t+1}|s_t, a_t)$. The observations $o_t$ are distributed according to some observation distribution $p(o_t|s_t)$. In general, both the dynamics and the observation distribution are not assumed to be known. To keep the notation simple with a slight abuse of notation, we also use $\pi_\theta(a_t|s_t)$ to denote the distribution over actions conditioned on a state. Applying a policy $\pi$ to the system defines a Markov chain and we use $\mathbb{E}_\pi$ to denote the expectation over this chain. We define $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$, where $\gamma \in [0, 1]$ is a discount factor which helps to maintain a tradeoff between immediate and future rewards. The optimization problem is to

maximize the expected cumulative discounted reward:

$$R = \mathbb{E}_\pi[R_1]. \tag{1}$$

In the case of the quadrotor control problem, the states $s_t$ include the physical state configuration (positions, velocities, etc.) of both the drone and the target object, while in general the observations $o_t$ from the onboard sensor cannot give a full description of the states. Besides, the actions $a_t$ consist of low-level rotor commands, implying that a partially trained policy is likely to perform unsafe actions that will crash the system. These challenges make it difficult to learn a stable policy by directly applying existing model-free reinforcement learning methods.

### 3.2. Deep Deterministic Policy Gradient

When the system dynamics $p(s_{t+1}|s_t, a_t)$ are not known, policy gradient methods [22] and Q-learning [26] are often preferred. Assuming that the environment is fully observed so that $o_t = s_t$, the Q-function $Q^\pi(s_t, a_t)$ represents the expected return after taking an action $a_t$ in state $s_t$ and thereafter following policy $\pi$:

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi[R_t|s_t, a_t]. \tag{2}$$

Consider a Q-function approximator parameterized by $\theta^Q$. It is optimized by minimizing the loss:

$$L(\theta^Q) = \mathbb{E}_\pi\left[\left(Q(s_t, a_t|\theta^Q) - y_t\right)^2\right], \tag{3}$$

where $y_t = r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a|\theta^{Q'})$ is the target Q-value estimated by a target Q-network.

However, for continuous action problems Q-learning becomes difficult since it requires maximizing a complex, nonlinear function at each update to improve the current policy. Therefore continuous domains are often tackled by actor-critic methods, where a separate parameterized "actor" policy is learned in addition to the Q-function. The Deep Deterministic Policy Gradient (DDPG) [16] algorithm, based on Deterministic Policy Gradient [25], maintains a parameterized actor function $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping each state to a unique action. The actor is updated by performing gradient ascent based on the following policy gradient:

$$\nabla_{\theta^\mu}\mu \approx \mathbb{E}_{\mu'}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}]. \tag{4}$$

By incorporating the ideas of sample replay buffer and target network backup originated from DQN [18], DDPG can use neural network function approximators for problems that involve continuous action domains.

## 4. Our Approach

DDPG allows us to optimize complex policies based on raw observations, like in applications when the policy takes images captured by a quadrotor's onboard vision sensor as input. However, function approximators such as neural networks defined on high-dimensional observation spaces are difficult to train in safety-critic unstable physical systems such as quadrotors. In this section, we develop a hierarchical learning approach that is particularly suitable for this task. The policy is represented by a convolutional neural network which is trained to estimate high-level reference actions to the PID control loop given raw observations, where the PID controller maps the reference inputs to low-level actions. Our training approach combines a supervised pre-training procedure with a reinforcement learning process to exhibit data efficiency and flexibility.

### 4.1. Policy Network Architecture

The policy network maps monocular RGB images and quadrotor configurations to high-level reference actions. The quadrotor configuration $\mathbf{s}_{q,t} = (z_t, \mathbf{v}_t, \mathbf{q}_t, \mathbf{w}_t)$ includes the altitude, linear velocity, orientation, and angular velocity. Note that the absolute positions of the quadrotor and the target are assumed unknown. Intuitively, some relative distance between the quadrotor and the target is sufficient for the task, and such information can be inferred from the camera image. The high-level actions are then mapped to low-level rotor commands by a PID controller and will be explained in detail later.

Our network architecture is shown in Figure 1 which combines perception and control. The visual processing layers have minor differences from the conventional network architectures used for image classification. First, we do not use pooling since the spatial information discarded by pooling is useful for determining the positions which are important for our task. Second, as inspired by [14], we incorporate a spatial-argmax layer after the last convolutional layer to convert each pixel-wise feature map into spatial coordinates in the image space. The spatial-argmax layer consists of a spatial softmax function applied to the last convolutional feature map and a fixed sparse fully connected layer which calculates the expected image position of each feature map. The spatial feature points are then regressed to a three-dimensional vector, $\mathbf{s}_{o,t} = (x_t, y_t, h_t)$, which represents the 2D position and scale (here we only keep the height information) of the target on the image plane, by another fully connected layer. After the visual processing layers, the target related state $\mathbf{s}_{o,t}$ is concatenated with the quadrotor's state configuration $\mathbf{s}_{q,t}$, followed by fully connected layers to the high-level actions.

### 4.2. Combination with PID Control Loop

The quadrotor features highly nonlinear dynamics and complex aerodynamics that are hard to learn using model-free reinforcement learning methods. Fortunately, this challenge can be tackled by a conventional PID controller. Fig-

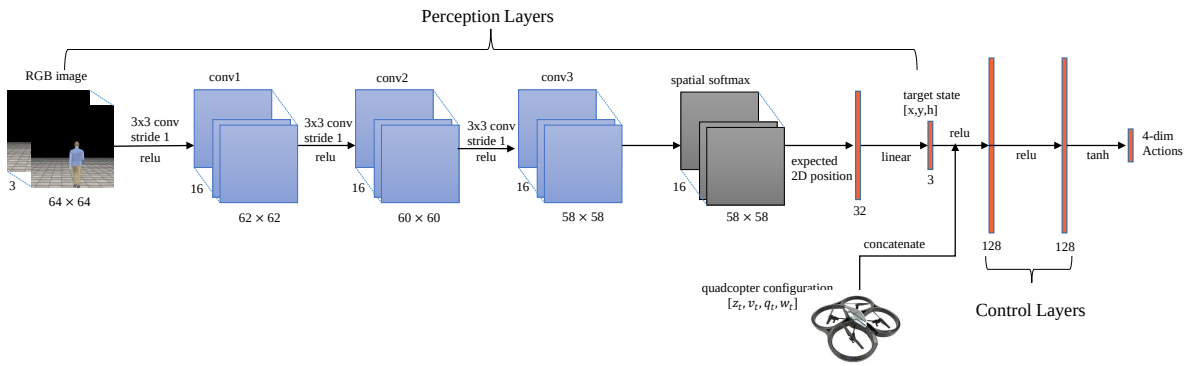


Figure 1. Policy network architecture. The perception module contains three convolutional layers, followed by a spatial softmax and a fixed weight layer that converts each pixel-wise feature map in the last convolutional layer to spatial coordinates. The spatial coordinates are then regressed to target related states. The control module consists of two fully connected layers which map the quadrotor state and target state to high-level actions.

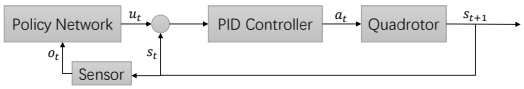


Figure 2. Hierarchical control system with the policy network.

ure 2 shows the proposed hierarchical control system. At each time step $t$, given the observed image, the policy network generates a four-dimensional high-level reference action $u_t$,

$$u_t = (v_x, v_y, v_z, w_{yaw}), \tag{5}$$

which corresponds to the linear velocities in the $x$-, $y$-, and $z$-directions and the rotational velocity around the yaw axis. With the given reference input and the observed quadrotor's current state, the PID controller calculates the four low-level rotor thrusts $a_t$ accordingly. Such a state feedback control loop enables the quadrotor system to work safely and reliably while a partially trained policy explores the environment.

### 4.3. Supervised Learning

While model-free reinforcement learning methods provide a simple, direct approach to train policies for complex tasks with minimal feature and policy engineering, the sample complexity of model-free algorithms tends to be high, particularly when using function approximators defined on high-dimensional observation spaces. To reduce the amount of experience needed to train our policy network, we introduce a supervised pre-training stage that allows us to initialize the policy network using a relatively small number of training iterations. The intuition behind our pre-training scheme is that, although we ultimately seek policies that combine both perception and control, the low-level vision representation can be initialized independently to achieve higher efficiency.

To speed up training, we pre-train the visual processing layers of the policy network to predict the target related state $\mathbf{s}_o$, which is not directly provided in the observed image. Specifically, we randomly move the quadrotor, making the target appear at different locations of the image captured by the onboard sensor. Meanwhile, the camera images and the corresponding labels (i.e., the target location and scale) are recorded. This dataset is used to train a pose regression CNN, which exactly matches the visual processing layers of the policy network. After training on the regression task, the weights of the visual processing layers are transferred to the policy network. This enables the policy network to learn the target appearance before learning the behavior in the game playing environment.

### 4.4. Reinforcement Learning

The goal of reinforcement learning algorithms is to learn policies that maximize the cumulative long-term reward. In this subsection we discuss our choice of the reward function in greater detail.

In many domains, it seems natural to provide rewards only upon task completion. However, if so designed, such sparse rewards may not provide effective enough feedback to the learner. Due to the continuous control nature of the target following task, an immediate reward feedback at every time step is essential. Besides, the instability and fragility of the quadrotor system also poses some additional challenges to the design of the reward function. A naive reward function based solely on the target related state $\mathbf{s}_{o,t}$ will lead to suboptimal policies that cannot guarantee flying stability. Therefore the reward function should consider both the quadrotor state and the target related state.

To that end, we design the reward function as a combination of the goal-oriented target reward and the auxiliary quadrotor reward:

$$r(s, a) = r_o(\mathbf{s}_o) + r_q(\mathbf{s}_q). \tag{6}$$

Note that for notational simplicity we omit the time step $t$

from the subscript here. By overloading the symbol $r_o$, the target reward is expressed as the sum of two parts as below:

$$r_o(\mathbf{s}_o) = r_o(x, y) + r_o(h), \tag{7}$$

which correspond to the position reward and scale reward, respectively. Let $s_{part}$ denote part of the state $\mathbf{s}_o$, which corresponds to either $(x, y)$ or $h$. Then the corresponding reward takes the following form:

$$r_o(s_{part}) = \begin{cases} \exp(-\Delta s_{part}) & \Delta s_{part} \leq \tau_1 \\ 0 & \tau_1 < \Delta s_{part} \leq \tau_2 \\ -(\Delta s_{part} - \tau_2) & \Delta s_{part} > \tau_2 \end{cases}, \tag{8}$$

where $\Delta s_{part} = \|s_{part} - s^*_{part}\|_2$ denotes the $\ell_2$-norm between the current state and the desired goal state. The intuition is that the learner must observe the variance in the reward signal in order to improve the policy. The above hierarchical form provides a kind of intermediate goal to guide the learning process to find a reasonable solution. Also with a slight abuse of notation, the auxiliary reward is expressed as follows:

$$r_q(\mathbf{s}_q) = r_q(z) + r_q(q_1, q_2, q_3, q_4), \tag{9}$$

which correspond to the quadrotor altitude and orientations (expressed as quaternions), respectively. Different from the target reward, the auxiliary reward is used to impose additional constraints on the flying gesture. So we introduce a hierarchical penalty. By using the same notations as in the target reward, the auxiliary reward takes the following form:

$$r_q(s_{part}) = \begin{cases} -c(1 - \exp(-\Delta s_{part})) & \Delta s_{part} \leq \tau_1 \\ -c & \Delta s_{part} > \tau_1 \end{cases}. \tag{10}$$

With the reward functions defined above, existing policy gradient methods (such as DDPG) can be applied in a game playing environment to further train the policy network. During the reinforcement learning process, the fully connected control layers are first optimized with the perception layer fixed, since they are not initialized with pre-training. Then the entire policy network is further optimized in an end-to-end manner. Empirical findings show that jointly optimizing the whole system from the very beginning hurts the pre-trained representation ability of the perception layers and hence leads to suboptimal policies.

# 5. Experiments

In this section we present a series of experiments to evaluate the proposed approach, aiming at answering the following research questions:

1. Is the policy network capable of processing raw images for our task?
2. Is the pre-training step essential to learn good policies?
3. Does training the perception and control modules jointly provide better performance?

To that end, we first compare different architectures for the perception module on the task of localizing a target object. We then evaluate different variations of the proposed learning approach by training policies for the full target following task. This allows us to validate the importance of each individual component in the proposed approach. We also evaluate the generalization ability of our approach by testing the learned policy in new environments not observed during training.

## 5.1. Experiment Platform

All experiments are conducted on the Virtual Robot Experimentation Platform (V-REP)[1] using the built-in quadrotor model. The observed state of the quadrotor $\mathbf{s}_{q,t} = (z_t, \mathbf{v}_t, \mathbf{q}_t, \mathbf{w}_t) \in \mathbb{R}^{11}$ consists of the altitude, linear velocity, orientation, and angular velocity, where the velocities are expressed in the body frame. The state of the target object is unknown and must be inferred from the sensory input. We use an onboard vision sensor to capture the RGB images from the quadrotor perspective, where the images are downsampled to $64 \times 64$ pixels. This type of observation is quite challenging to be directly integrated into simple control methods, but can be easily processed by a convolutional neural network policy.

## 5.2. Pre-training Evaluation

We now evaluate our network architecture used in the perception module in comparison to several other possible variants. To isolate it from the other factors in our approach, we measure the performance of different architectures on a pose regression task, which is to predict the target state, $\mathbf{s}_o = (x, y, h)$, with a given input image. We construct the dataset by collecting about 2000 images from the simulator, making the target appear at different locations of the image. We use 1500 images for training and 500 others for testing. The accuracy is measured by the Euclidean error. This task serves to validate the network's ability to learn useful spatial representations in a rather small dataset. We compare several networks that differ in the following three aspects: whether to use pooling in between convolutional layers, whether to include a spatial softmax layer, and whether to replace the fixed position expectation layer with a fully connected layer. This results in a total of eight different network architectures. For all the convolutional layers, we use $3 \times 3$ kernels with stride 1. The network is trained by performing stochastic gradient descent.

As shown in Table 1, our network reduces the test error substantially and outperforms the more common variants

[1] http://www.coppeliarobotics.com/

that use pooling and fully connected layers. The spatial softmax and the fixed position expectation layer allow the network to efficiently learn feature points and provide the upper layers with a concise representation for spatial constraints without overfitting. Also, removing pooling in the network helps to retain more spatial information. All the hyperparameters are fixed across networks and the result in terms of accuracy is not sensitive to them.

| network architecture | test error |
|---|---|
| expectation | 0.075 |
| fully connected | 0.035 |
| pooling + expectation | 0.075 |
| pooling + fully connected | 0.019 |
| pooling + softmax + fully connected | 0.058 |
| pooling + softmax + expectation | 0.004 |
| softmax + fully connected | 0.074 |
| softmax + expectation **(ours)** | **0.003** |

Table 1. Average regression error measured as Euclidean error.

### 5.3. Target Following Task

In this section, we present an evaluation of our full policy training algorithm on the target following task.

#### 5.3.1 Task Details

Figure 3 depicts the simulated environment with which we train the convolutional neural network to learn good policies. In the beginning of every game episode, the target is set to a random position relative to the quadrotor, making the target appear at a specific location on the onboard camera frame. The moving pattern of the target at every time step is also designed in a similar random manner: the target randomly chooses a direction to walk at a randomly generated speed until some distance is reached. The random nature of the environment is to prevent the agent from achieving good performance simply by remembering the action sequences. The game will terminate either when the target is out of the camera frame or when the quadrotor crashes, as determined by using simple thresholds on the quadrotor's altitude and orientation. The maximum episode length is set to 1000. We also add a negative reward (i.e., penalty) in the case of early game termination to prevent the agent from getting stuck in suboptimal policies.

#### 5.3.2 Implementation Details

The environment is implemented under the framework of OpenAI Gym [5] to provide unified compatibility with existing reinforcement learning toolkits. We choose the off-policy actor-critic algorithm DDPG to train the policy network due to its sample efficiency over on-policy policy gradient methods. We use the rllab [7] reinforcement learning library for policy training. The Q network shares the same architecture with the policy network, except that the last two fully connected layers have a smaller number of units (32) and the actions are included in the second to last layer. We use Adam [11] for learning the neural network parameters with base learning rates of $10^{-4}$ and $10^{-3}$ for the actor and critic, respectively. The other hyperparameters are set according to [16]. The implementation will be made publicly available upon paper acceptance to facilitate reproducible research.

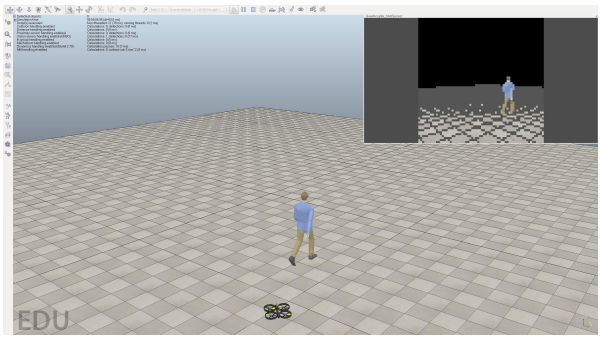


Figure 3. The quadrotor must learn to follow the target (person) using only onboard vision sensing. The top-right corner shows the streaming image from the onboard vision sensor.

#### 5.3.3 Reinforcement Learning Evaluation

Our full policy training algorithm includes supervised learning from raw images and reinforcement learning from playing games. One may wonder whether supervised learning is essential for the target following task, since DDPG has already been shown to learn good policies directly from raw image observations. However, the success of end-to-end training of DDPG has only been demonstrated in rather simple tasks such as the classic cartpole. DDPG tends to learn very unstable policies on harder tasks such as car racing. Besides, the required sample size is still quite large when trained with image observations. To validate the necessity of supervised pre-training, here we also train a policy network by DDPG without any initialization. As the policy network is fine-tuned by reinforcement learning, we are also faced with the choice of jointly optimizing all layers from the very beginning, or only updating the control layers until a certain performance is reached.

To address the above problems, all the three different training approaches are evaluated here. Figure 4 shows the performance of different methods. In all the methods, the policy network is trained for 150 iterations with 2000 time steps for each iteration. Figure 4 demonstrates that supervised pre-training of the perception module greatly in-

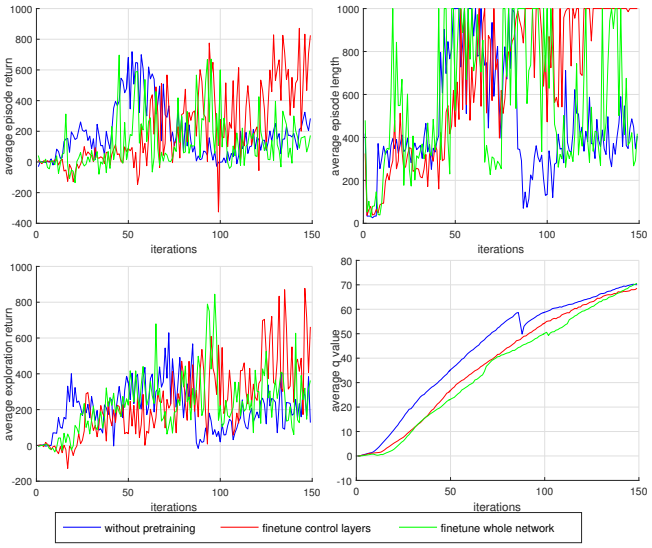


Figure 4. Training results on three different approaches. From top-left to bottom-right shows the average episode return, average episode length, average exploration return, and estimated Q value with respect to the training iterations. Pre-training the visual processing layers greatly increases the stability of the reinforcement learning process. Optimizing different layers in a hierarchical way also benefits the learning process.

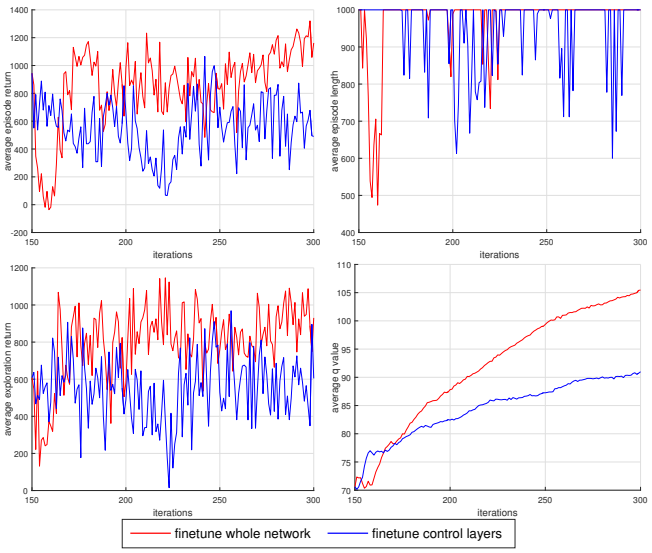


Figure 5. Training results on the end-to-end approach and baseline method. In both methods, the first 150 iterations are trained by only fine-tuning the control layers in Figure 4. From top-left to bottom-right shows the average episode return, average episode length, average exploration return, and estimated Q value with respect to the training iterations. End-to-end training improves the performance.

creases the stability of the policy network, which has always been a major drawback of many reinforcement learning algorithms. Without pre-training, the policy suffers severe vibration. Actually, further experiments show that even with more training iterations, the pure DDPG algorithm still gets stuck in suboptimal policies. Another important observation is that jointly optimizing the whole network from the very beginning might hurt the overall performance. The intuition is that when the error signal due to the untrained control layers is large, optimizing the whole network leads the convolutional layers to forget useful features learned by pre-training.

We finally present an evaluation of the full policy training algorithm. The aim of this evaluation is to answer the following question: does training the perception and control layers jointly provide better performance? After initializing the control layers as above, we fine-tune the whole network in an end-to-end manner. We also make comparison with a baseline in which only the control layers are fine-tuned. Figure 5 compares the end-to-end training approach and the separate training approach. The learning curve suggests that jointly training the perception and control layers in an end-to-end manner does improve the performance a lot. Although pre-training the visual processing layer helps to reduce the training time, by itself it is not sufficient for learning good features for the target following policies.

To further understand the performance gap between the two methods, we apply the trained model in a testing environment in which the agent interacts with the environment until game termination. We record the true target state in Figure 6 to show the quality of different policies. Although the policies are trained with a maximum episode length of 1000 in both methods, the agent generalizes well beyond that. Both policies can consistently follow the target for quite a long time, neither crashing the quadrotor nor losing target from the camera perspective. However, the separate training approach only manages to learn a suboptimal policy with which the resulting target state is relatively far from the desired goal state, as indicated by the dashed line in Figure 6. On the contrary, the end-to-end training approach achieves a very stable policy to successfully maintain the target state within a small range of the goal state for up to 5000 time steps. The testing video demos can be found in the supplementary material.

#### 5.3.4 Generalization to Unseen Environments

To evaluate the generalization ability of the policies learned by our approach, we further test the trained policy network in two unseen environments: a scenario with a background significantly different from the training setting (scenario 1), and a scenario with very similar object distractors (scenario 2). Figure 7 shows the two environments.

We directly apply the trained policy network to the new testing environments without any adaptation. In Figure 8, we again show the target state variations over time during the testing phase. Surprisingly, our policy network exhibits

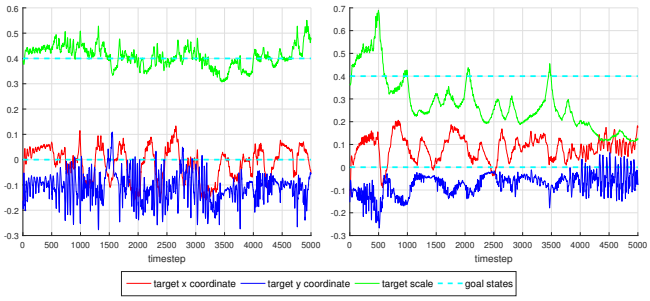


Figure 6. The true target state $\mathbf{s}_{o,t} = (x, y, h)$ varies as the trained agent interacts with the testing environment. From left to right shows the results of the end-to-end approach and the separate training approach, respectively. The upper dashed line shows the goal state for the target scale ($h^* = 0.4$) while the lower dashed line shows the goal state for the target coordinates ($(x^*, y^*) = (0, 0)$).


Figure 7. New testing environments not previously observed when learning the policies. The upper one shows scenario 1 where a different background and some trees are added. The lower one shows scenario 2 where some other people similar to the target being tracked are also added.

good generalization ability to unseen scenes. The perception module demonstrates moderate tolerance to different backgrounds and distractors and the control module learns general goal-driven strategies. We find that the results in scenario 2 are significantly worse, which indicates that the perception module tends to produce high errors in case of occlusion by similar objects. This could be alleviated by modeling temporal information in the network.

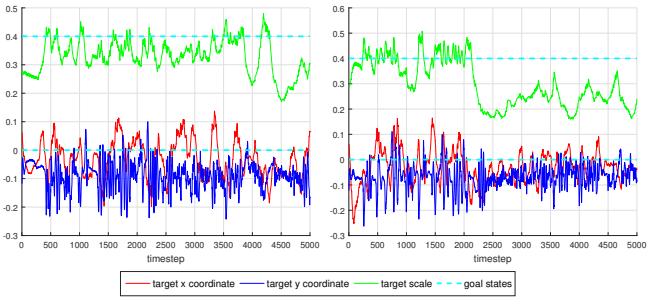


Figure 8. The true target state $\mathbf{s}_{o,t} = (x, y, h)$ varies as the trained agent interacts with the testing environment. From left to right shows the results of scenaro 1 and scenario 2, respectively. In each plot the upper dashed line shows the goal state for the target scale ($h^* = 0.4$) while the lower dashed line shows the goal state for the target coordinates ($(x^*, y^*) = (0, 0)$).

## 6. Conclusion and Future Work

In this paper, we have explored the potential of applying a machine learning approach to the challenging autonomous UAV control problem. We present a method for learning target following policies on the quadrotor platform that uses raw sensory input from a monocular camera. The policy is represented by a convolutional neural network which combines perception and control and thus can be trained end-to-end. To avoid taking unsafe exploration actions by a partially trained policy which might cause the destruction of a fragile system, we develop a hierarchical approach that combines a model-free policy gradient method with a conventional feedback PID controller. Given raw sensor readings, the policy network provides high-level reference actions to the PID control loop while the PID controller maps the reference inputs to low-level actions. Our training approach consists of supervised learning from raw images and reinforcement learning from games of self-play. This training decomposition greatly alleviates the instability of existing model-free policy gradient methods. Results from the experiments show that our method can successfully perform the target following task with good generalization ability and end-to-end training yields significant improvement when compared with using fixed perception layers.

While our approach can successfully learn to control the quadrotor with raw images as input, it still bears some limitations. The perception module is only moderately tolerant to some visual distractors but does not generalize well to dramatically different settings, such as long-term occlusion. We may rely on incorporating temporal information into the model to alleviate this problem. Currently, the low-level action is produced by a standalone PID controller. Ultimately this low-level mapping can also be learned by the policy. Also, transferring our approach to the real-world quadrotor setting is also an interesting direction. We will pursue these research in our future work.

## References

[1] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 2010. 2

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *NIPS*, 2007. 2

[3] P. Abbeel, V. Ganapathi, and A. Y. Ng. Learning vehicular dynamics, with application to modeling helicopters. In *NIPS*, 2005. 2

[4] A. Bachrach, S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy. Estimation, planning, and mapping for autonomous flight using an RGB-D camera in GPS-denied environments. *The International Journal of Robotics Research*, 31(11):1320–1343, 2012. 2

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016. 6

[6] M. P. Deisenroth, D. Fox, and C. E. Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, 2015. 2

[7] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *ICML*, 2016. 6

[8] D. Floreano and R. J. Wood. Science, technology and the future of small autonomous drones. *Nature*, 521(7553):460–466, 2015. 1

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 2

[10] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep Q-learning with model-based acceleration. In *ICML*, 2016. 2

[11] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6

[12] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012. 2

[13] S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *NIPS*, 2014. 2

[14] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016. 1, 2, 3

[15] S. Levine and V. Koltun. Guided policy search. In *ICML*, 2013. 2

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016. 1, 2, 3, 6

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016. 2

[18] 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*, 518(7540):529–533, 2015. 1, 2, 3

[19] N. Mohajerin and S. L. Waslander. Modular deep recurrent neural network: Application to quadrotors. In *SMC*, 2014. 2

[20] F. L. Mueller, A. P. Schoellig, and R. D'Andrea. Iterative learning of feed-forward corrections for high-performance tracking. In *IROS*, 2012. 2

[21] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of memory, active perception, and action in Minecraft. In *ICML*, 2016. 2

[22] J. Peters and S. Schaal. Policy gradient methods for robotics. In *IROS*, 2006. 3

[23] A. Punjani and P. Abbeel. Deep learning helicopter dynamics models. In *ICRA*, 2015. 2

[24] 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*, 2013. 2

[25] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014. 3

[26] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999. 3

[27] N. Vandapel, J. Kuffner, and O. Amidi. Planning 3-D path networks in unstructured environments. In *ICRA*, 2005. 2

[28] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search. In *ICRA*, 2016. 2