Learning Unmanned Aerial Vehicle Control for Autonomous Target Following

Siyi Li[1]**, Tianbo Liu**[2]**, Chi Zhang**[1]**, Dit-Yan Yeung**[1]**, Shaojie Shen**[2]
[1] Department of Computer Science and Engineering, HKUST
[2] Department of Electronic and Computer Engineering, HKUST
{sliay, czhangbr, dyyeung}@cse.ust.hk, {tliuam, eeshaojie}@ust.hk

Abstract

While deep reinforcement learning (RL) methods have achieved unprecedented successes in a range of challenging problems, their applicability has been mainly limited to simulation or game domains due to the high sample complexity of the trial-and-error learning process. However, real-world robotic applications often need a data-efficient learning process with safety-critical constraints. In this paper, we consider the challenging problem of learning unmanned aerial vehicle (UAV) control for tracking a moving target. To acquire a strategy that combines perception and control, we represent the policy by a convolutional neural network. We develop a hierarchical approach that combines a model-free policy gradient method with a conventional feedback proportional-integral-derivative (PID) controller to enable stable learning without catastrophic failure. The neural network is trained by a combination of supervised learning from raw images and reinforcement learning from games of self-play. We show that the proposed approach can learn a target following policy in a simulator efficiently and the learned behavior can be successfully transferred to the DJI quadrotor platform for real-world UAV control.

1 Introduction

The recent development of perception [Li and Yeung, 2017; Shen *et al.*, 2013] and control [Lee, 2011] technologies for unmanned aerial vehicles (UAVs) enables autonomy in various complex environments, opening up a promising market with applications in aerial photography, monitoring, and inspection. Many of these applications require an aerial robot to autonomously follow a moving target.

In this paper, we consider the problem of controlling a quadrotor with very limited payload to autonomously follow a moving target. Most early research on quadrotor autonomy focused on a two-step pipeline. The first step is on perception where various vision algorithms are used to estimate the underlying state or map [Bachrach *et al.*, 2012; Barry and Tedrake, 2015; Schmid *et al.*, 2013; Bills *et al.*, 2011; Sun *et al.*, 2017]. The second step is to design and tune conventional feedback controllers, such as proportional-integral-derivative (PID) controllers, matching some manually defined rules. The whole system consists of several hard-coded components without any learning capability and relies heavily on tedious tuning by human experts to achieve good performance.

These limitations may be addressed by viewing the problem from the machine learning perspective. In essence, quadrotor control is a sequential prediction problem with the sensory information as input and the motor control commands as output. Standard supervised learning [Giusti *et al.*, 2016] and imitation learning [Ross *et al.*, 2013] have been used to learn various controllers for quadrotors. However, both methods require labeled datasets, which have to be provided by costly human experts. With human experts becoming the bottleneck, such approaches are often restricted to small datasets and thus cannot exploit high-capacity learning algorithms to train the policies.

Recent advances in reinforcement learning (RL) offer new promises for solving the problem. Instead of demanding explicitly labeled samples as in supervised learning, RL only requires a scalar reward function to guide the learning agent through a trial-and-error process interacting with the environment. Most RL approaches belong to either one of two categories: model-based methods and model-free methods. Model-based methods learn an explicit dynamical model of the environment and then optimize the policy under this model. They have been successfully applied to robotics for various applications such as object manipulation [Deisenroth *et al.*, 2015; Levine *et al.*, 2016], ground vehicles [Mueller *et al.*, 2012], helicopters [Abbeel *et al.*, 2010; 2007], and quadrotors [Mohajerin and Waslander, 2014; Punjani and Abbeel, 2015; Zhang *et al.*, 2016]. Model-based methods tend to have a data-efficient learning process but suffer from significant bias since complex unknown dynamics cannot always be modeled accurately. In contrast, model-free methods have the ability of handling arbitrary dynamical systems with minimal bias. Several recent studies in model-free methods, especially deep RL [Mnih *et al.*, 2015; Lillicrap *et al.*, 2016; Mnih *et al.*, 2016; Schulman *et al.*, 2015; Gu *et al.*, 2017], have shown that end-to-end model-free methods are capable of learning high-quality control policies using generic neural networks with minimal feature engineering. However, Model-free methods generally require a data-hungry training paradigm which is costly for real-world physical systems. Although some hierarchical learning methods [Aijun Bai, 2017] are proposed, they are not designed for the quadrotor domain. Specific to the quadrotors, the relative frailty of the underlying system makes a partially trained motor-level policy crash in the initial stage of training. A key question is how to enjoy the richness and flexibility of a self-improving policy by model-free learning while preserving the stability of conventional controllers.

In this paper, we propose to combine the stability of conventional feedback PID controllers with the self-improving performance of model-free RL techniques so that the hybrid method can be practically applied to learning quadrotor control. Model-free methods are used to learn the high-level reference inputs to the PID control loop while the PID controller maps the reference inputs to low-level motor actions. Consequently, both data efficiency and training stability will be greatly improved. Moreover, the proposed hierarchical control system makes it easy to transfer the policy trained in simulators to real-world platforms, since both sides share similar high-level system dynamics. This transfer ability is crucial to many real-world control tasks since learning in simulated environments only incurs low cost. Our experimental validation shows that (1) introducing the PID controller is essential for successful training of quadrotor control policies and (2) the learned high-level policy network can successfully generalize across different environments including real-world scenarios.

2 Preliminaries

2.1 Formulation of Quadrotor Control Problem

The quadrotor target following task can be naturally formulated as a sequential decision making problem under the RL framework. At each time step t, the agent receives an observation o_t from the environment (i.e., the onboard sensor), applies an action a_t according to a policy π, and then obtains a reward feedback r_t. The goal of the agent is to learn an optimal policy that maximizes the expected sum of discounted rewards R_t

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i, \tag{1}$$

where T is the terminated time step and $\gamma \in [0, 1]$ is a discount factor that determines the importance of future rewards. The underlying state s_t of the system includes the physical state configuration (positions, velocities, etc.) of both the quadrotor and the target object (which generally need to be inferred from the observations o_t). Since the actions a_t consist of low-level motor commands, the complex system dynamics (denoted as the distribution $p(s_{t+1}|s_t, a_t)$) make it difficult to learn a stable policy by directly applying existing model-free RL methods.

2.2 Deep Deterministic Policy Gradient

When the system dynamics $p(s_{t+1}|s_t, a_t)$ are not known, model-free RL methods such as policy gradient [Peters and Schaal, 2006] and Q-learning [Sutton *et al.*, 1999] methods 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 π:

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

Consider a Q-function approximator parameterized by θ^Q. It can be 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) [Lillicrap *et al.*, 2016] algorithm, based on Deterministic Policy Gradient [Silver *et al.*, 2014], 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 [Mnih *et al.*, 2015], DDPG can use neural network function approximators for problems that involve continuous action domains.

3 Our Approach

Although model-free methods such as DDPG allow us to optimize complex policies based on raw image observations, they require massive amounts of data to achieve good performance. Besides, function approximators such as neural networks defined on high-dimensional observation spaces are very difficult to train in fragile physical systems such as quadrotors, since the agent can hardly find actions to reach any good state in the exploration process, especially in (infinite) continuous action domains. We now present our proposed approach which is particularly suitable for this task.

3.1 Policy Network Architecture

In order to represent a policy that performs both perception and control, we use deep neural networks. As shown in Figure 1, the policy network maps monocular RGB images and quadrotor configurations to the actions. We discard pooling in the visual processing layers due to the loss of spatial information. Inspired by [Levine *et al.*, 2016], 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. In order to achieve stable flight, it is essential to use the quadrotor configuration $\mathbf{s}_{q,t} = (z_t, \mathbf{v}_t, \mathbf{q}_t, \mathbf{w}_t)$, which includes the altitude,

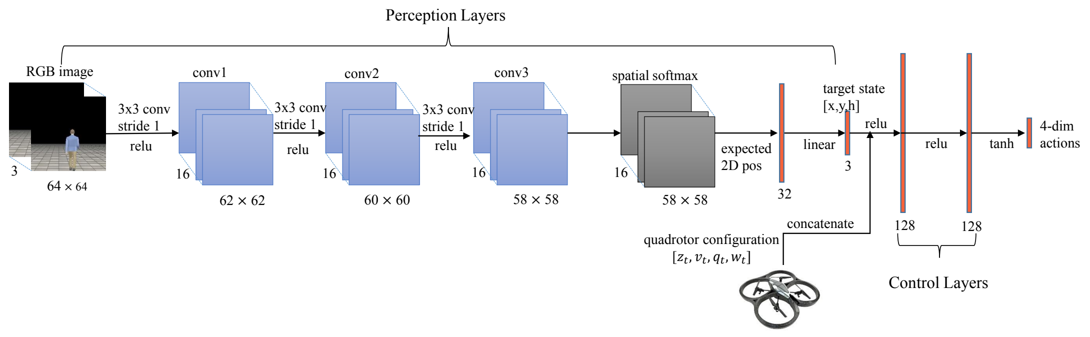


Figure 1: Policy network architecture. The perception layers estimate the target state while the control layers learn the control behavior.

linear velocity, orientation, and angular velocity, as an additional input to the neural network. After the perception layers, the target related state $\mathbf{s}_{o,t}$ is concatenated with the quadrotor state $\mathbf{s}_{q,t}$, followed by fully connected layers to the actions.

While we could choose to directly output low-level motor actions a_t by the policy network, the model will easily get stuck in unwanted local optimum to yield little performance improvement even after tens of thousands of sample experiences. We reason that pure model-free RL methods cannot effectively learn stable policies in such fragile systems. Therefore we introduce another set of high-level actions u_t as the output of the policy network. The high-level actions are then mapped to low-level motor commands by a PID controller.

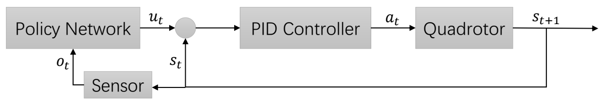


Figure 2: Hierarchical control system which combines the policy network and a PID controller.

3.2 Combination with PID Control Loop

The quadrotor features complex nonlinear aerodynamics which are hard to learn by model-free RL methods. Fortunately, this challenge can be tackled by incorporating a conventional PID controller. Figure 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

$$u_t = (p_x, p_y, p_z, \varphi_{yaw}), \tag{5}$$

which corresponds to the relative position offset in the x-, y-, z-directions and the relative angle offset around the yaw axis. Thanks to the differential flatness property proposed in [Mellinger and Kumar, 2011], the above high-level reference trajectories can be followed by simple cascade PID controllers with dynamical feasibility guarantee. While the quadrotor dynamics of simulated models are often significantly different from those of real-world platforms, their high-level decision strategies are generally very similar. Thus introducing high-level actions also makes it much easier to transfer a policy learned in a simulator domain to a real-world domain.

3.3 Reward Shaping

Due to the continuous 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:

$$r = r_g(\mathbf{s}_g) + r_q(\mathbf{s}_q). \tag{6}$$

Note that for notational simplicity we omit the time step t from the subscript here. The target reward is expressed as the sum of two parts as below:

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

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

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

where $\Delta s_{part} = \|s_{part} - s^*_{part}\|_2$ denotes the ℓ_2-norm between the current state and the desired goal state, τ_1, τ_2 denotes different threshold values. The intuition is that the learner must observe variations in the reward signal in order to improve the policy. This hierarchical form essentially provides an intermediate goal to guide the learning process to find a reasonable solution step by step. 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. Only penalty terms are introduced. By using the same notations as in the target reward, the auxiliary reward takes the following form:

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

where τ_1 denotes the same threshold as in equation (8) and c denotes the penalty weight.

With the reward function defined above, existing policy gradient methods (such as DDPG) can be applied in a game playing environment to train the policy network.

3.4 Training Strategy

Although the policy network can be directly trained end-to-end by DDPG, our empirical finding shows that the agent will suffer severe vibration in the learning process due to the high sample complexity of model-free methods. Another drawback is that the perception layers cannot be guaranteed to accurately locate the target of interest.

We propose to introduce a supervised pre-training stage that allows us to initialize the perception layers of the policy network using a relatively small number of training iterations. The dataset can be easily collected by randomly moving the quadrotor and recording the camera image stream. In a simulator, we can directly get the accurate labels. In real-world domains, we can use existing model-free object trackers [Li and Yeung, 2017] to get the noisy labels. Both are sufficient to train a pose regression CNN. After training on the regression task, the weights of the perception layers can be transferred to the policy network. By factoring the perception and control tasks in learning, we can gain the image generalization power of CNNs across different environments (simulator and real-world). This is more reasonable than some existing approaches [Sadeghi and Levine, 2017] which directly transfer simulator perception to the real world.

After the supervised learning stage, we first fix the perception layers and learn only the weights of the control layers which are not initialized with pre-training. Then the entire policy network is further optimized in an end-to-end manner. Our empirical findings (in section 4.2) show that jointly optimizing the whole network from the very beginning hurts the pre-trained representation ability of the perception layers.

3.5 Transfer to the Real World

As explained in Section 3.2, learning the high-level actions makes it easier to transfer a policy from the simulator domain to a real-world domain. Considering the large gap between the simulated images and the real-world images, we take the factorizing scheme during training as in Section 3.4. Namely, the perception layers are pre-trained by a small dataset collected in real-world scenarios. There are also several challenges to set up a game playing environment for real-world quadrotors. First, we cannot access the true target state efficiently online and thus cannot reliably compute the reward. Second, efficiently resetting the game state upon game termination is difficult. Fortunately we can bypass these issues to directly use the policy behavior trained in simulators. Since in simulators all the underlying true states are available, we can directly learn the control layers with the state input. Finally the perception layers and the control layers are merged to form the policy network which is applicable to real-world domains.

4 Experiments

In this section we present a series of experiments to answer the following research questions:

1. Is introducing the PID controller essential for successful training?
2. How does the training strategy work compared to standard end-to-end training?
3. How does the learned high-level policy network generalize across different environments?

To answer question (1) and (2), we evaluate different variations of the proposed system, in Section 4.2, by training policies for the target following task in a simulated environment. We further evaluate the generalization ability of the learned policy in Section 4.3 by testing it in various simulated environments. Finally, we set up a real-world flight test in Section 4.4.

4.1 Simulator Settings

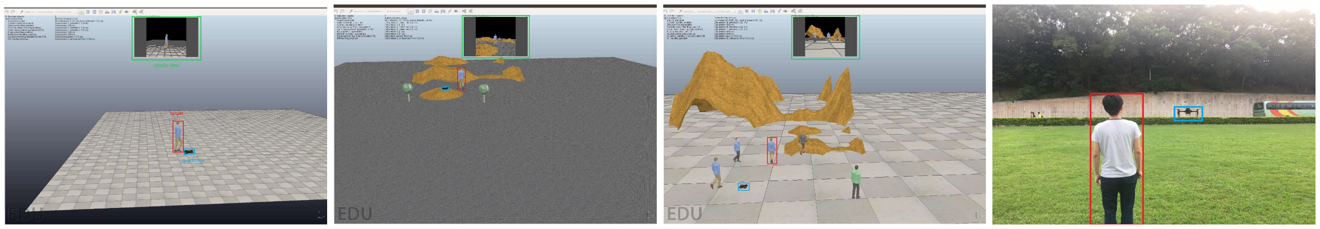
Environment. We set up the simulated target following task on the Virtual Robot Experimentation Platform (V-REP) using the built-in quadrotor model. The observed state of the quadrotor $\mathbf{s}_{q,t} = (z_t, \mathbf{v}_t, \mathbf{q}_t, \mathbf{w}_t) \in \mathbb{R}^{11}$ consists of the altitude, linear 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 RGB image input with resolution 64×64. We require the target to be in the camera view of the quadrotor on initialization. At each time step, the target randomly chooses a direction to walk at a random speed. The game will terminate either when the target is out of the camera view or when the quadrotor crashes, as determined by using simple thresholds on the quadrotor's altitude and orientation. The maximum episode length allowed is set to 1000. Three different simulated scenes are shown in the left part of Figure 3.

Implementation Details. We choose the off-policy actor-critic algorithm DDPG due to its sample efficiency over on-policy methods. Our implementation is based on rllab [Duan *et al.*, 2016]. 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 [Kingma and Ba, 2015] for optimization with the hyperparameters set according to [Lillicrap *et al.*, 2016]. For the reward setting, we use $\tau_1 = 0.05$, $\tau_2 = 0.2$, and $c = 0.5$.

4.2 System Design Evaluation

In this section we validate our design choices of both the hierarchical control system and the proposed training strategy. All of the results are averaged over 5 different runs.

Hierarchical control system vs. end-to-end control system. In this experiment, we show the effectiveness of introducing the PID controller to the control system. We compare two approaches: the first one being an end-to-end control system where the policy network directly outputs a_t (the velocities of the 4 rotors) and the second one being the hierarchical control system with PID where the policy network outputs u_t. Both approaches are trained with standard training strategy (no pre-training and no hierarchical fine-tuning). Figure 4 shows the learning curves of these two different approaches. Intuitively,



(a) Simulated scenario 1. (b) Simulated scenario 2. (c) Simulated scenario 3. (d) Real-world scenario.

Figure 3: Three simulated environments and one real-world environment. The blue, red and green annotation denotes the quadrotor, the target and the sensor view respectively. Object and person distractors are presented in Scenario 2 and 3.

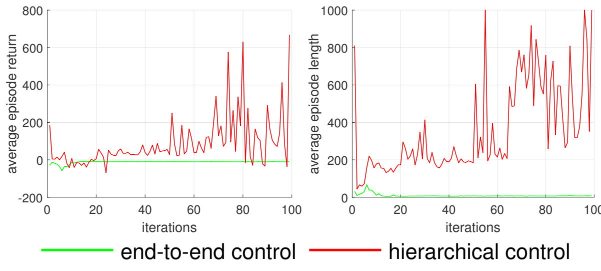


Figure 4: Training results of the end-to-end motor-level control system and the proposed hierarchical control system. The hierarchical control system is substantially easier to train.

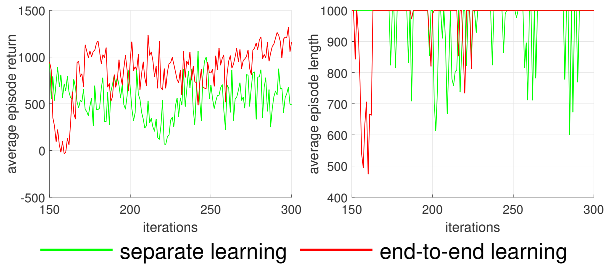


Figure 6: Comparison between separate learning and end-to-end learning. The first 150 iterations are trained by only fine-tuning the control layers in Figure 5.

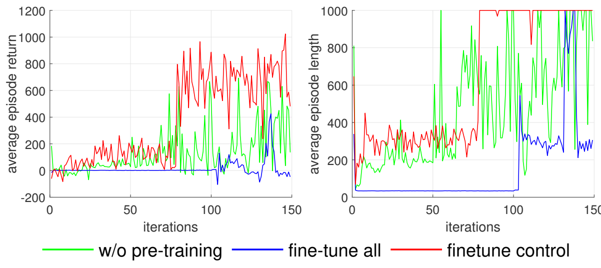


Figure 5: Training results on three different training strategies.

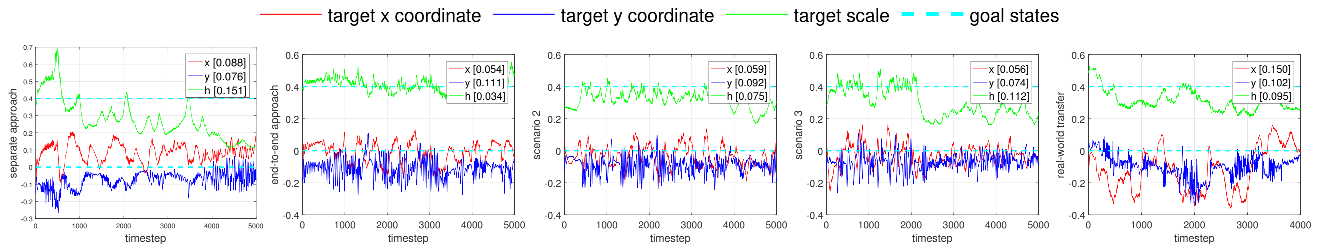
the *average return* measures the precision of control, while the the *average episode length* measures the successfully followed steps (robustness). In both methods, the network is trained for 100 iterations with 2000 time steps for each iteration. We can see that a direct application of DDPG to learn low-level motor commands cannot achieve any performance improvement while the proposed hierarchical control approach achieves substantial improvement as learning proceeds. This large performance gap is mainly due to the complexity and fragility of the quadrotor system.

With pre-training vs. without pre-training. We can observe in Figure 4 that the agent suffers severe vibration in the learning process under standard training strategy. To validate the necessity of supervised pre-training, we compare three different approaches here. The first one is the standard training without any pre-training (**w/o pre-training**). The second one is to fine-tune all layers after pre-training (**fine-tune all**). The last one is our strategy which only fine-tunes the control layers after the pre-training initialization (**fine-tune control**). Figure 5 shows the performance of different methods. We show the learning curves for up to 150 iterations since we observe only minor improvements afterwards. The results demonstrate that supervised pre-training of the perception module greatly increases the stability of the policy network, which has always been a major drawback of many RL algorithms. 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 since it leads the convolutional layers to forget useful features learned through pre-training.

End-to-end learning vs. separate learning. So far, the perception layers and the control layers have been learned separately. We now examine our design choice of end-to-end fine-tuning: 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 (**end-to-end learning**). We also make comparison with a baseline in which only the control layers are fine-tuned (**separate learning**). As shown in Figure 6, the learning curve suggests that jointly training the perception and control layers end-to-end does further boost the performance.

4.3 Policy Evaluation in Simulators

To gain more insights into how the learned policy actually works, we further apply the trained policy network in a number of simulated testing environments in which the policy interacts with the environment until game termination. For testing environment initialization, we randomly set the positions of the quadrotor and the target, making the target appear at different corners of the camera view with different scales.



(a) Separate vs. end-to-end in scenario 1 (left to right). (b) Unseen scenarios 2 and 3 (left to right). (c) Real-world test.

Figure 7: Variations of the normalized (divided by image dimensions) true target state $\mathbf{s}_{o,t} = (x, y, h)$ over time ($x, y \in [-0.5, 0.5], h \in [0, 1]$). The dashed lines show the goal for each states: $(x^*, y^*, h^*) = (0, 0, 0.4)$. The top-right legend shows the average deviation.

The performance of the policy is measured by the deviation of the target state from our desired goal (which is specified in the reward function). We also average the results over 5 different runs.

We first compare two different policies trained by separate learning and end-to-end learning (as described in Figure 6), respectively. By applying the models in the testing environment, we can record the true target state variation, as shown in Figure 7(a), to show the quality of different policies. We also compute the average deviation of each state variable to give quantitative analysis, as shown in the top-right legend. The result is consistent with our findings in the design evaluation. The end-to-end learning approach is clearly better. It is worth noting that, although the policies are only trained with a maximum of 1000 time steps, the agent can generalize well beyond that. Both policies can consistently follow the target for quite a long time, neither crashing the quadrotor nor losing the target from the camera view. However, the separate learning 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 7(a). On the contrary, the end-to-end learning 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 (and even more). For all subsequent experiments in this section we will stick to the model trained by the end-to-end learning approach.

Since training is only performed in scenario 1 (Figure 3), we further test the trained policy network in two unseen environments (scenarios 2 and 3) to evaluate the generalization ability. Scenario 2 has a background significantly different from the training setting and scenario 3 has very similar object distractors. We directly apply the trained policy network to the new testing environments without any adaptation. As shown in Figure 7(b), surprisingly, our policy network exhibits good generalization ability to unseen scenes. We find that in scenario 3 the scale drifts a little bit. This is the result of occlusion by some similar distractors.

4.4 Policy Transfer to the Real World

Our quadrotor testbed is based on the DJI Matrice 100 platform equipped with an Intel NUC and a camera. We use DJI built-in functions to map the high-level actions to actual flight commands, which are significantly more complex than a simple PID controller. This mapping computation is done by the NUC. For speed consideration, the policy network computation is deployed on a ground laptop M1000 GPU which communicates with the onboard NUC by Robot Operating System (ROS).

Figure 7(c) shows the results of the real-world flight test, where the true target states are labeled off-line by an object tracker. With a slight decrease in control precision, the transferred agent can still follow the moving target for up to 4000 time steps (approximately 3 minutes). It is worth noting that this real-world test bears several challenges, including the large gap between the low-level controllers, the data delay, and the state noise in inertial measurement unit (IMU) and GPS. Without any adaptation, the learned policy can still exhibit reasonable following capability. This shows that the policy does have well-generalized performance. With more mature simulators, we believe these are promising directions to pursue in robotics learning.

5 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. The policy is represented by a convolutional neural network which combines perception and control and thus can be trained end-to-end. Instead of directly predicting the low-level motor commands, the policy network is designed to produce high-level actions. This enables both stable learning and good generalization ability to different environments. 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 both simulated and real-world experiments show that our method can successfully perform the target following task with good generalization ability.

Currently the policy is directly transferred to the real-world quadrotor. This can also be used as an initialization scheme and the model can be further trained end-to-end [Zhu *et al.*, 2017]. We will pursue these research directions in our future work.

Acknowledgments

This research has been supported by General Research Fund 16207316 from the Research Grants Council of Hong Kong.

References

[Abbeel *et al.*, 2007] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *NIPS*, 2007.

[Abbeel *et al.*, 2010] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 2010.

[Aijun Bai, 2017] Stuart Russell Aijun Bai. Efficient reinforcement learning with hierarchies of machines by leveraging internal transitions. In *IJCAI*, 2017.

[Bachrach *et al.*, 2012] Abraham Bachrach, Samuel Prentice, Ruijie He, Peter Henry, Albert S Huang, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas 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.

[Barry and Tedrake, 2015] Andrew J Barry and Russ Tedrake. Pushbroom stereo for high-speed navigation in cluttered environments. In *ICRA*, 2015.

[Bills *et al.*, 2011] Cooper Bills, Joyce Chen, and Ashutosh Saxena. Autonomous MAV flight in indoor environments using single image perspective cues. In *ICRA*, 2011.

[Deisenroth *et al.*, 2015] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, 2015.

[Duan *et al.*, 2016] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *ICML*, 2016.

[Giusti *et al.*, 2016] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2016.

[Gu *et al.*, 2017] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *ICRA*, 2017.

[Kingma and Ba, 2015] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[Lee, 2011] Taeyoung Lee. Geometric tracking control of the attitude dynamics of a rigid body on so (3). In *ACC*, 2011.

[Levine *et al.*, 2016] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[Li and Yeung, 2017] Siyi Li and Dit-Yan Yeung. Visual object tracking for unmanned aerial vehicles: A benchmark and new motion models. In *AAAI*, 2017.

[Lillicrap *et al.*, 2016] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[Mellinger and Kumar, 2011] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *ICRA*, 2011.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[Mohajerin and Waslander, 2014] Nima Mohajerin and Steven L Waslander. Modular deep recurrent neural network: Application to quadrotors. In *SMC*, 2014.

[Mueller *et al.*, 2012] Fabian L Mueller, Angela P Schoellig, and Raffaello D'Andrea. Iterative learning of feed-forward corrections for high-performance tracking. In *IROS*, 2012.

[Peters and Schaal, 2006] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *IROS*, 2006.

[Punjani and Abbeel, 2015] Ali Punjani and Pieter Abbeel. Deep learning helicopter dynamics models. In *ICRA*, 2015.

[Ross *et al.*, 2013] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive UAV control in cluttered natural environments. In *ICRA*, 2013.

[Sadeghi and Levine, 2017] Fereshteh Sadeghi and Sergey Levine. (CAD)^{2}RL: Real single-image flight without a single real image. In *RSS*, 2017.

[Schmid *et al.*, 2013] Korbinian Schmid, Teodor Tomic, Felix Ruess, Heiko Hirschmüller, and Michael Suppa. Stereo vision based indoor/outdoor navigation for flying robots. In *IROS*, 2013.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

[Shen *et al.*, 2013] Shaojie Shen, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor. In *RSS*, 2013.

[Silver *et al.*, 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[Sun *et al.*, 2017] Ting Sun, Shengyi Nie, Dit-Yan Yeung, and Shaojie Shen. Gesture-based piloting of an aerial robot using monocular vision. In *ICRA*, 2017.

[Sutton *et al.*, 1999] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

[Zhang *et al.*, 2016] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search. In *ICRA*, 2016.

[Zhu *et al.*, 2017] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017.