Fast and Accurate Trajectory Tracking for Unmanned Aerial Vehicles based on Deep Reinforcement Learning

Yilan Li[1], Hongjia Li[3], Zhe Li[1], Haowen Fang[1],
Amit K. Sanyal[2], Yanzhi Wang[3], Qinru Qiu[1]
[1] *Electrical Engineering & Computer Science*, *Syracuse University*, Syracuse, NY 13244, USA
[2] *Mechanical and Aerospace Engineering*, *Syracuse University*, Syracuse, NY 13244, USA
[3] *Electrical and Computer Engineering*, *Northeastern University*, Boston, MA 02115, USA
{yli41, zli89, hfang02, qiqiu}@syr.edu, aksanyal@syr.edu,
{li.hongjia@husky.neu, yanz.wang@northeastern}.edu

***Abstract*— Continuous trajectory control of fixed-wing unmanned aerial vehicles (UAVs) is complicated when considering hidden dynamics. Due to UAV multi degrees of freedom, tracking methodologies based on conventional control theory, such as Proportional-Integral-Derivative (PID) has limitations in response time and adjustment robustness, while a model based approach that calculates the force and torques based on UAV's current status is complicated and rigid. We present an actor-critic reinforcement learning framework that controls UAV trajectory through a set of desired waypoints. A deep neural network is constructed to learn the optimal tracking policy and reinforcement learning is developed to optimize the resulting tracking scheme. The experimental results show that our proposed approach can achieve 58.14% less position error, 21.77% less system power consumption and 9.23% faster attainment than the baseline. The actor network consists of only linear operations, hence Field Programmable Gate Arrays (FPGA) based hardware acceleration can easily be designed for energy efficient real-time control.**

***Index Terms*—deep reinforcement learning, continuous trajectory tracking, actor-critic algorithm, unmanned aerial vehicles**

I. Introduction

Recently the applications of UAVs have been widely used in numerous real world applications where human operations are limited. With the increasing of data volume and accuracy requirements for practical applications, the reliable operations of UAV, i.e. the stable autonomous guidance and control, have been considered as one of the most critical. Efficient tracking algorithms enable a smooth trajectory and hence a lower system power/energy dissipation during the flight. Traditionally, the PID control mechanism is the state-of-the-art choice for industrial UAV trajectory tracking system. PID controllers are easy to be implemented on FPGA and sufficient for many control problems. They work well when process dynamics are benign and the performance requirements are modest [1] [2]. However, the PID controller cannot treat processes with large time delay efficiently and it shows poor performance for tracking problems requiring aggressive dynamic configurations, including uncertain internal disturbance compensation and imbalances retrieval [3] [4]. For some applications, modified PID models implemented have been explored to improve the performance [5] [6].

Meanwhile, it is a big challenge to control UAVs stability in general using low power cost platforms, especially under uncertain disturbance from environments. The main reason is that it is hard to obtain a high fidelity mathematical model of a UAV which has an under-actuated system with nonlinear dynamics [7]. To improve the stability and real-time control, deep neural networks (DNN) embedded on different hardware platforms are introduced [8] [9]. Through large data training, the DNN-based control system achieves adaptability and robustness that guarantee the stability of the flight [10]. Additionally, the controllers are able to follow the desired trajectory with the tolerance of unexpected disturbance. Similar to PID controllers, the DNN based controller estimates the control actions based on past flight experience to reduce the instantaneous tracking imperfections. None of them considers how the chosen action will affect the subsequent rewards. Hence, they are likely to generate suboptimal solutions.

Reinforcement learning (RL) provides a mathematical framework for learning or deriving policies that map situations (i.e. states) into actions with the goal of maximizing an accumulative reward [11]. Unlike supervised learning, in RL the *agent* (i.e. learner) learns the *policy* for decision making through interactions with the *environment*. The aim of the agent is to maximize the cumulative long-term *reward* by taking the proper action at each time step according to the current *state* of the environment while considering the trade-off between explorations and exploitations. Q-learning is one of model-free RL strategies storing finite state-action pairs and corresponding Q-values in a look-up table and it has been applied for thermal and energy management in autonomous computing systems [12] [13]. The combination of conventional Q-learning and deep neural network, i.e. Deep Q-network [14], provides a breakthrough in deep reinforce-

This work is partially supported by the National Science Foundation under Grant CNS-1739748.

ment learning (DRL). The neural network in DQN needs to accumulate enough samples of values and the data needed for its training can either come from a model-based simulation or from actual measurement [15]. Originally developed by DeepMind, the DRL provides a promising data-drive, adaptive technique in handling large state space of complicated control problems [16]. The actor-critic deep reinforcement learning [17] has overcome difficulties in learning control policies of systems with continuous state and action space, which provides a potential solution for effective real-time mission control of autonomous UAVs.

In this paper, we propose an actor-critic DRL model to track trajectories of UAVs through sets of desired waypoints, and its implementation using FPGA. The detailed framework is discussed in Section IV. Based on the model provided by [18], we aim at actuating one degree of translation motion and three degrees of rotation motion for quadrotor body-fixed UAVs. We construct a fully-connected neural network to learn the optimal tracking policy based on DRL. We choose different sets of desired waypoints as test benchmarks. The experimental results in Section V show that compared to the baseline, our proposed approach can achieve 58.14% less position tracking error and 9.23% faster attainment. The efficient tracking leads to 21.77% power saving during the flight time. In addition, our actor network can easily be mapped to FPGAs for hardware acceleration and the input/output size of network is constrained because of limited dimensions of state/action that an agent/environment can physically process. With a low-cost FPGA, one single decision can be made within 0.0002 second at only $0.03mW$ power consumption in a decision epoch. The speed and power consumption allows the proposed actor-critic framework to be used for real-time on-board control of autonomous systems.

II. Related Works

Due to the wide utilization and increasing stringent constraints on the size and energy dissipation of UAV control system, many researchers have put extensive efforts recently on trajectory tracking system using different processing platforms. However, conventional high-performance computing systems (mainly GPUs) either consume significant amounts of power or are too bulky to be placed on small UAVs. FPGA currently serves as the main cost saving platform in terms of less volume, efficient control response and configurable to allow new features and different approaches [19] [20]. Preliminary FPGA designs were combined with an eight bit microprocessor for a fixed wing aircraft with GPS in [21]. In order to get rid of large and heavy external computers or sensors, Ryo Konomura et al. developed a FPGA-based self-position estimation system which included an ARM 9 dual-core CPU and FPGA in one-chip [22].

With the development of machine learning techniques in the last decade, neural networks have been introduced during the control processing of UAV. Meanwhile, requirements for greater task complexity and higher flight performance tend to increase on-board processing. In 2012, [9] presented a FPGA-based artificial neural network to implement autonomous landing system, which demonstrated to prove the limited data requirements and applicability to low-power implementation. Li et al. developed a combination of a convolution neural network and a conventional feedback PID controller to learn UAV control for tracking a moving target in [23]. They trained the network by reinforcement learning from games of self-play. In order to ensure the stability, they proposed a hierarchical approach which combines a model-free policy gradient method with a conventional feedback PID controller and trained the neural network by a combination of supervised learning from raw images and reinforcement learning from games of self-play. In [24], Wang et al. designed a scalable accelerator architecture with three pipelined processing units for large-scale deep learning networks using FPGA as the hardware prototype. Moreover, an ultra-low power and high-performance DNNs using the circulant weight matrix with FPGA implementation has been proposed in [25]. The work performed the forward pass of the AlexNet in $7.54ms$ with less than $1W$ on-chip power, which accelerated the speed of closing the control loop of autonomous UAV. In [26], Su et al. proposed an FPGA accelerator for DQN algorithms which provided high throughput compared to CPU and GPU platforms. In order to support run-time parameterization to the neural network topology, it increased the flexibility by taking advantages of high on-chip memory bandwidth and customizable hardware resource.

III. Preliminaries

A. Actuation Model of UAV

In this work, a vertical take-off and landing (VTOL) quadrotor/quadcopter UAV model with four identical actuators (propellers), each separated by a scalar distance D from the axis of rotation of the actuators to the center of the UAV [18],

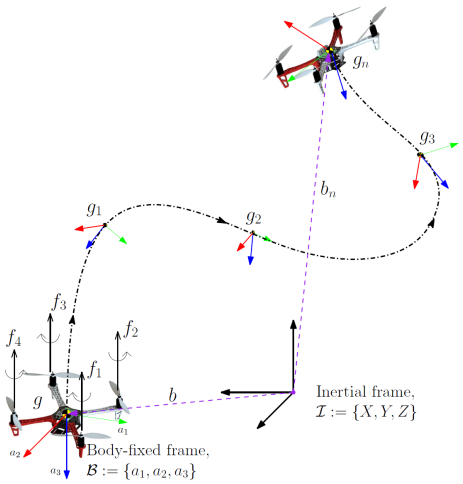


Fig. 1: Guidance through a set of finite waypoints between initial and final configurations on $SE(3)$.

is considered. A conceptual diagram of guidance on special Euclidean group of rigid body motions, $SE(3)$, through a set of waypoints is shown in Fig. 1. Based on this $SE(3)$ model, waypoints are generated considering the actuator locations on the body of the UAV, as well as obstacles and potential collision hazards detected in the field of view of the onboard sensors. Since a quadrotor or multi-rotor UAV with all rotors in one body-fixed plane can only generated a body-fixed thrust perpendicular to that plane, we need control the attitude simultaneously with the position tracking. After waypoints are selected, a C^2 (i.e. continuous and twice differentiable with respect to time) trajectory is generated for the desired inertial position in time, denoted pd_t, connecting these waypoints. According to the pd_t, the desired translational velocity $\dot{pd}_t$ and the desired translational acceleration $\ddot{pd}_t$, i.e. velocity and acceleration of UAV mass center moving from one position to the other, can be obtained. The desired UAV control thrust fd_t depends on the current location p_t, velocity $\dot{p}_t$, acceleration $\ddot{p}_t$, and the desired values of these quantities pd_t, $\dot{pd}_t$, and $\ddot{pd}_t$.

B. Deep Reinforcement Learning

Deep Q-learning [14] (DQL) adopts an offline-built DNN to derive the correlation between each state-action pair (s, a) of the system under control and its *value function* $Q(s, a)$. The Q-value is the expected cumulative (with discounts) reward function when system starts at state s, and follows action a (and certain policy thereafter). To be more specific, at each decision epoch t_k of an execution sequence, the system under control is at state s_k. The DRL agent performs inference using DNN to select action a_k, which is either the one with the highest estimated Q-value, or selected with certain degree of randomness using the ϵ-greedy policy. At the next decision epoch t_{k+1}, the DRL agent stores the newly estimated Q-value calculated based on the total reward $r_k(s_k, a_k)$ observed in time slot k. At the end of the execution sequence, The DRL agent performs mini-batch updating [16] [15] that updates the DNN using the new Q-value estimates. $Q(s, a)$ is given in (1).

$$Q(s, a) = \mathbf{E}(\Sigma_{k=0}^{\infty} \lambda^k r_k(s_k, a_k) | s_0 = s, a_0 = a)) \qquad (1)$$

where r_k is the reward achieved in time slot k, and $\lambda < 1$ is the future reward discount factor. The DQL is restricted when the number of actions you can take any time is infinite. The actor-critic reinforcement learning framework [27] learns policy and state-value function by containing two interacting models, i.e. actor and critic, and is fundamentally premised on solving problems with continuous output space. To accelerate learning and avoid oscillations or divergence in the parameters, an experience replay is deployed [17] and it updates the target network weights θ' based on learned network weights θ by:

$$\theta' = \tau\theta + (1 - \tau)\theta', \qquad \tau \ll 1 \qquad (2)$$

IV. System Architecture and Hardware Design

A. Problem Definition

In this work, we consider the trajectory tracking for under-actuated aerial vehicles through a set of given desired way-points. To state the problem without losing generality, we aim at quadrotor fixed-wing UAVs with four control inputs, one degree of translation motion and three degrees of rotation motion (i.e. pitch, roll and yaw.) The directions of four motions are illustrated in Fig. 2 using different colors (yellow: *translational freedom*; green: *rotation freedom*). It is extremely difficult to integrate detailed mechanistic model of complicated dynamics with classic control theory, a model-free solution for the control problem is preferred. Because actions of the tracking problem are continuous variables (e.g. turning force, thrust etc.), the actor-critic reinforcement learning is adopted, which learns to find the optimal set of actuations that move the UAV towards desired trajectory. The technique presented in [18] is used to generate C^2 trajectory based on a given set of predefined waypoints T_d where each waypoint gives the desired position of the UAV at time t. Meanwhile, desired velocity vd_t, desired acceleration dvd_t and desired attitude Rd_t are extracted from T_d based on kinematics. The positions and attitudes together form the pose of the UAV. The goal of our model is to minimize the differences between desired poses and actual poses during tracking. We define $sd_t = \{pd_t, vd_t, dvd_t, Rd_t\}$ as desired state and $s_t = \{p_t, v_t, dv_t, R_t\}$ as actual state of UAV at time step t. Each of first three variables in the sd_t and s_t are 3-dimensional vectors and the last one in the sd_t and s_t is 3×3-dimension, hence the desired state and the actual state are variables in 18-dimensional space. All variables used at time t are summarized in Table I.

TABLE I: Variables Summary

pd_t : desired position	p_t : achieved position
vd_t : desired velocity	v_t : achieved velocity
dvd_t : desired acceleration	dv_t : achieved acceleration
Rd_t : desired attitude	R_t : achieved attitude
sd_t : desired state	s_t : achieved state
fm_t : applied force	τ_t : applied torques
$\mathbb{A}_t$: applied action	$\mathbb{S}_t = \{sd_t, s_t\}$: agent state
$\mathbb{E}$: environment simulation	

The concatenation of sd_t and s_t forms the agent state $\mathbb{S}_t$. Furthermore, we define action at time t as $\mathbb{A}_t = \{fm_t, \tau_t\}$, where the translational force fm_t and torques τ_t are applied to UAV. The translational force fm_t is a force perpendicular to the top surface of UAV and the three components in τ_t are applied to roll, pitch and yaw directions respectively. The reward $Reward(\Delta_t)$ at time t is defined as the Manhattan distance between the desired pose and the actual pose, i.e. $Reward(\Delta_t) = f(|p_t - pd_t| + |v_t - vd_t| + |R_t - Rd_t|)$.

B. Network Structure

Since the control variables (i.e. fm_t, τ_t) of UAV are in a continuous space which are infinitely large, we build an actor-critic reinforcement learning model instead of discretizing the action space. The actor model is a feed-forward deep neural network of three fully-connected hidden layers with rectified linear units (ReLU) as the activation function. It is used to

predict the optimal action based on current state $\mathbb{S}_t$. The number of neurons in fully-connected hidden layers are 64, 128 and 128 respectively. The size of output layer is 4 and LeakyReLU activation function is used in the output layer. The critic model is another feed-forward neural network that computes an evaluation of the action and that evaluation is used by actor model to update its control policy in particular gradient direction. The critic model has two hidden layers, where the first layer contains two separate fully-connected structures and the number of hidden neurons in each is 32. The addition of outputs from the first hidden layer is fed into the second layer which has 64 hidden neurons. The inputs of critic model are $\mathbb{S}_t$ and $\mathbb{A}_t$ and the output is a single value $Q(\mathbb{S}_t, \mathbb{A}_t)$. The size of the critic and actor is optimized as hyper-parameters through cross-validation. The detailed networks of actor model and critic model are shown in Fig. 3a and Fig. 3b. The overall framework is shown in Fig. 3c. During training, the actor model is pre-trained using labeled pair data $(\mathbb{S}_t, \mathbb{A}_t)$ generated from simulation [18] to predict the optimal action $\mathbb{A}_t$ based on current agent state $\mathbb{S}_t$. Next agent state $\mathbb{S}_{t+1}$ is calculated through environment simulation based on $\mathbb{A}_t$ and is used to predict optimal $\mathbb{A}_{t+1}$ by actor model. The critic model evaluates the resulting $\{\mathbb{S}_{t+1}, \mathbb{A}_{t+1}\}$ pair by predicting a Q-value to fine-tune action prediction. Therefore, the weights in actor model are updated by the gradient between actor and critic model, using chain rule $dQ/dW_{actor} = dQ/dW_{critic} \times dW_{critic}/dW_{actor}$. W_{actor} and W_{critic} indicate the weights of *actor* and *critic* models respectively.

C. Reward Definition

Our goal is to actuate the UAV to be closer to desired pose T_d along the desired trajectory, i.e. minimizing the value of $\Delta P_t = |p_t - pd_t|$. Besides position error, the stability of the UAV should also be taken into consideration. Therefore the values of velocity errors (i.e. $\Delta V_t = |v_t - vd_t|$) and attitude errors ($\Delta R_t = |R_t - Rd_t|$) must also be minimized. However, our experiments show that simply using a linear combination of ΔP_t, ΔV_t, and ΔR_t as the reward function will make convergence difficult in learning process. According to [28], using geometrically discounted reward will prevent the accumulated reward to become infinite and make the model more tractable. Therefore, we define the reward at each time step following a standard normal distribution, guaranteeing the largest reward is accepted when the total differences between desired trajectory and actual trajectory at time t (i.e. $\Delta_t = \Delta P_t + \Delta V_t + \Delta R_t$) reaches zero and closing to zero reward is obtained when Δ_t increases. The total discounted reward is denoted as $\mathbb{R}$.

$$
\begin{aligned}
Reward(\Delta_t) &= \frac{1}{\sqrt{2\pi}} exp(-\frac{\Delta_t^2}{2}) \\
\mathbb{R} &= \sum_{t=0}^{\infty} \gamma^t Reward(\Delta_t)
\end{aligned}
\tag{3}
$$

where $\Delta_t = \Delta P_t + \Delta V_t + \Delta R_t$.

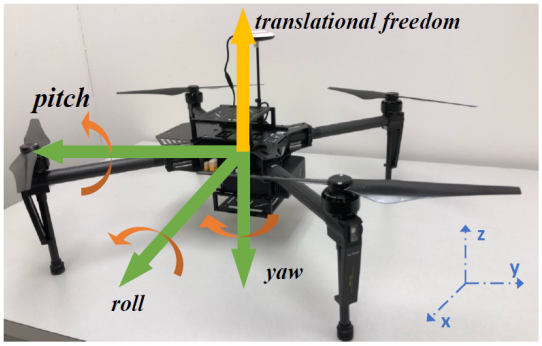
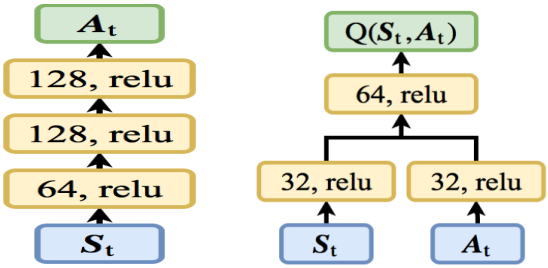
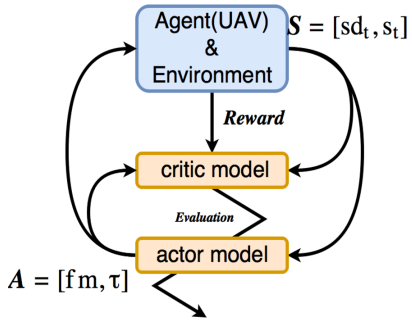


Fig. 2: Illustration of UAV control inputs



(a) actor model structure (b) critic model structure



(c) overall framework

Fig. 3: Architecture details of actor model and critic model, and overall framework of proposed framework.

D. Proposed Hardware Configuration

Fig. 4 shows an overview of hardware design of the proposed UAV controller and its connections. Intel (Altera) Cyclone V 5CGX FPGA is selected as our processing platform and all massive parallel computations are implemented on it. We select this FPGA because of its light weight as the UAV payload, relatively high compute capability, and its low energy cost.

Although there are only linear computations (i.e. multiplications and additions) are need in actor model and the mapping to FPGA is straightforward, two issues need to be addressed. Firstly, all computations cannot be done at once due to resource limitation of FPGA, time-multiplexing is essentially required. Secondly, computation latency introduced by time-multiplexing conflicts with the real-time response requirement for UAV control. Our proposed design aims

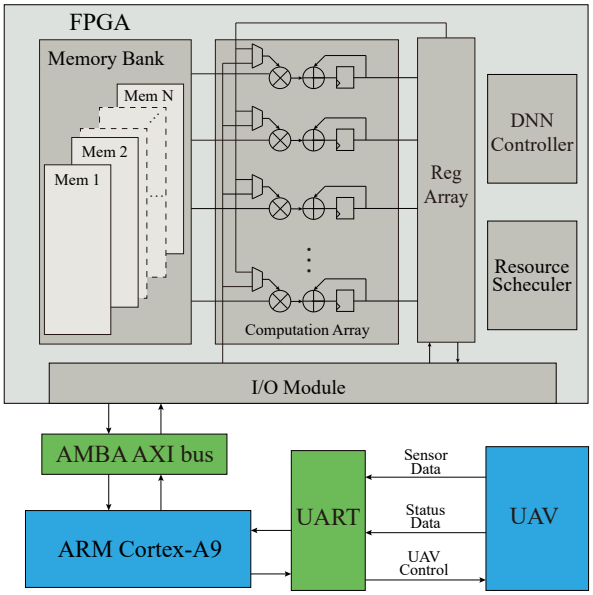


Fig. 4: Hardware configuration of the UAV controller

at exploiting the hardware resource to improve parallelism and to minimize latency, taking two issues mentioned above into consideration. In our design, DSP blocks are used as multiplier. Layer-wise computation is done by computation array, which consists of multiple parallel computation units for multiplication and accumulation performance. Results of intermediate layers are buffered in register array and will feed back to the input of computation array for the computation of next layer. DNN controller builds the communication between FPGA and ARM processor. Resource allocator schedules the time-multiplexed computation. The FPGA is connected to the on-board ARM Cortex-A9 processor. ARM Cortex-A9 takes control commands from FPGA as input, calculates actuations in each freedom, and then sends actuations to UAV controller. In addition, UART handles the communication between ARM Cortex-A9 and UAV. It accepts flight state and sensor data from UAV and sends data to FPGA after preprocessing.

V. Experimental Results

A. Environment Setup

We trained the actor-critic network and implemented the simulation on Nvidia GeForce GTX1070 using Keras [29]. The data that we use to train and test our model is generated using simulator described in [18]. Our dataset consists one thousand different 3-D trajectories of four different shapes, including straight lines, z-shape curves, spiral curves and circles. Each desired trajectory has one thousand desired waypoints, giving enough time for UAV to track it. All desired waypoints are defined by mathematical equations parameterized by time. Eight hundred different trajectories of four different shapes are evaluated in total. The mass of UAV used is $4.34kg$, and its inertial properties $\mathcal{J}$ is a 3×3 diagonal matrix (i.e. $diag[0.820 \quad 0.0845 \quad 0.1377]kgm^2$) which determines the required magnitude of force to accelerate the UAV in each rotation direction respectively. The goal is to minimize the power consumption and time used from trajectory deviated to tracked.

B. PID Implementation

As a baseline approach, we also implement control scheme based on PID theory because it has been widely used in industrial applications and real-time control situations. We define desired position pd_t as observable variables (OV) and actual position p_t as measurable variables (MV). Velocities are regarded as controllable variable (CA). The data of pd_t is gathered from simulator in [18] and is used to derivate desired velocity vd_t of UAV at time t. Three PID controllers are used for each component of velocity respectively, which are indicated by green blocks in Fig. 5. The inner structure of each PID is the same as conventional structure. PID controllers calculate errors of each component between desired velocity and achieved velocity by the UAV, i.e. errors between vd_x and v_x, vd_y and v_y and vd_z and v_z, continuously as references to give velocity corrections. Furthermore, the achieved position p_t is calculated through Environment simulation (i.e. orange block) and used as feedback based on current velocity v_t. As a consequence, the difference between desired position pd_t and actual position p_t is used to update the velocity of UAV through Kinematics block (i.e. purple block). The overall structure of PID-based controller forms a feedback loop and is shown in Fig. 5

C. Results Comparison

All test samples of desired trajectories are generated the same way as mentioned in Section V-A. Fig. 6 shows sample testing achieved trajectories (blue curves) using proposed DRL learning approach and corresponding desired trajectories (red curves). We report the results from four aspects: (1) L1-norm of position tracking error; (2) L1-norm of velocity tracking error; (3) Time used to complete tracking; (4) Power consumption.

Fig. 7 shows the $L1$-norm of position tracking error, where the first four columns are comparison results of trajectories with different shapes respectively and the fifth column is average $L1$-norm of position error of all testing trajectories. According to the figure, our approach has lower average position error especially when the trajectory is more complicated.

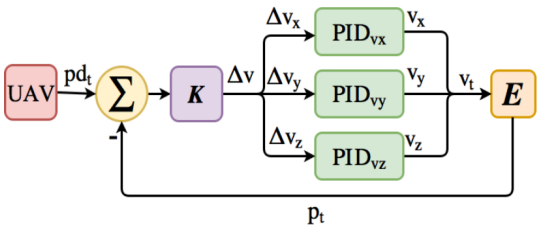
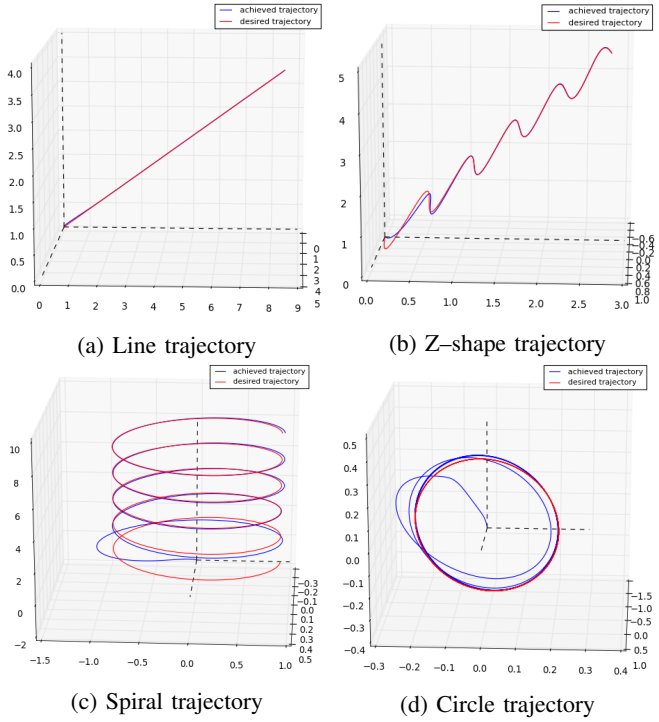


Fig. 5: Structure of PID-based baseline controller



(a) Line trajectory (b) Z–shape trajectory

(c) Spiral trajectory (d) Circle trajectory

Fig. 6: Examples of achieved trajectories and desired trajectories in terms of four different shapes. (*red*: desired trajectories. *blue*: achieved trajectories using proposed DRL-based learning approach.)

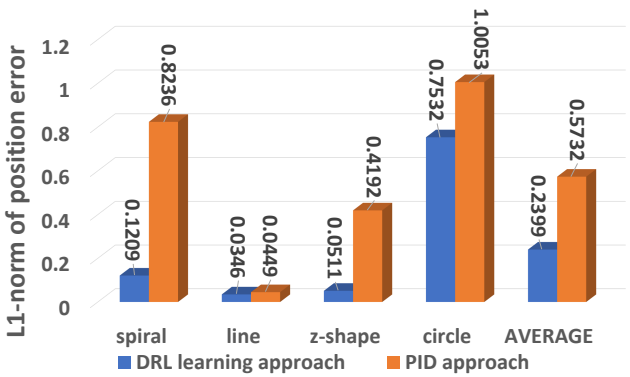


Fig. 7: Tracking result comparison between proposed DRL-based framework and PID-based baseline in terms of $L1$-norm of position error

Compared to PID based baseline control, 22.94% less position error is achieved for straight line trajectory tracking and 25.07% less position error is achieved for circular trajectory tracking. The position tracking error reduction increases to 85.32% for spiral shape tracking and 87.81% for z-shape trajectories tracking respectively. On average, our approach outperforms 58.14% better than PID based control in position tracking of different shapes of trajectories.

Fig. 8 shows the average of $L1$-norm of velocity tracking error. Similar to Fig. 7, the first four columns are comparison results of each type of trajectory with different shapes, and the last column shows overall $L1$-norm of velocity tracking error averaged over all testing trajectories. It shows 29.79% error reduction is achieved for line trajectory and 67.22% error reduction is achieved for circle trajectory. Up to 91.64% and 93.17% error reductions are achieved for z-shape and spiral trajectories. On average, our approach outperforms 58.15% than PID control with respect to velocity tracking error. Again our learning based approach performs better for the more complicated trajectories.

Fig. 9 compares the total time steps used for UAV to follow each shape of desired trajectory and the average time steps used in total to reach stability. We report the number of time steps used for UAV to completely track the desired trajectory. The total time steps for each testing trajectory is one thousand. The time step when trajectory is tracked is regarded as the time t_c after which the $L1$-norm of position error between pd_t and p_t is less than 0.0001. The average tracking time for different types of trajectories, and the average tracking time over all testing trajectories are reported. Our approach is 9.23% faster than PID-based control to achieve stable pose on average. It is especially 13.86% faster for line trajectory and up to 15.58% faster for z-shape trajectory. Moreover, PID has lags in responding current dynamics in all three directions of velocity. Therefore, PID-based controller is not optimally adapted for non-linearity situations, especially not robust in fast dynamic control. It trades off the control performance and time.

Fig. 10 reports average total power consumption after one trajectory is completely tracked using our approach and baseline PID controller. As indicated in the figure, our approach achieves 11.04% less power consumption when tracking z-shape trajectory and consumes 18.64% less power for line trajectory tracking. Furthermore, up to 21.63% and 29.11% power consumption improvements have been achieved for circle and spiral trajectories tracking respectively. An average of 21.77% less power is consumed using our approach for tracking all different trajectories. The noticeable result

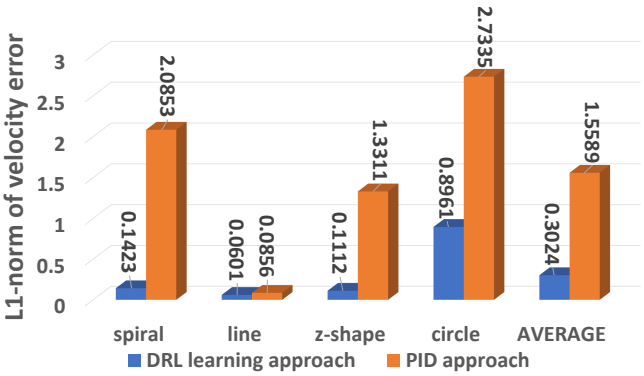


Fig. 8: Tracking result comparison between proposed DRL-based framework and PID-based baseline in terms of $L1$-norm of velocity error

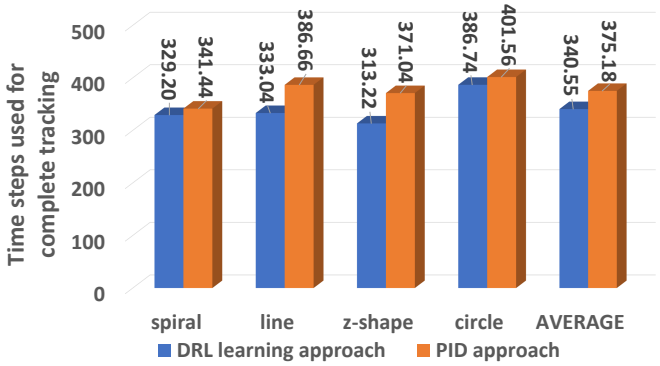


Fig. 9: Tracking result comparison between proposed DRL-based framework and PID-based baseline in terms of used tracking time steps

explains that PID control consumes more power because of oscillations of controllable variables during tracking process.

D. Tracking in a Noisy Environment

To test the robustness of the learning based trajectory tracking, we also add different levels of random Gaussian noise and see if the tracking algorithm can adapt to the changing environment. Our model free DRL approach is compared to the Matlab model based trajectory tracking approach, which calculates the force and torque of the UAV using a 3-dimensional Euclidean space mechanics model in the form of Lie Group Variational Integrator (LGVI) given in [18]. The random noise is a deviation added on the UAV position at same random time step with a duration of 5 time steps. Such noise could be used to model the effect of wind gust, which may deviate the UAV away from its current position. Fig. 11 compares the tracking results in an environment with random noises.

In the first experiment, a relatively small noise is added to the environment. The left figure in Fig. 11a shows the desired and actual trajectories generated by Matlab simulator for the UAV using model based tracking in [18], while the right one shows the similar information for the UAV using our proposed DRL-based tracking model. As we can see, the UAV using the DRL based tracking follows the desired trajectory more closely. After adding small random Gaussian noise, DRL based system is more stable and achieves smaller position tracking error under considerable error precision. The left figure in Fig. 11b shows the $L1$-norm of position tracking error for model based tracking (D_{matlab}) and DRL based tracking (D_{DRL}). The right figure in Fig. 11b shows the difference between these two (i.e. $D_{matlab} - D_{DRL}$). It shows that after deviating from the original trajectory due to random noise, the model based approach will not correct itself right away. Only after the deviation becomes large, it will gradually track back the original trajectory. While the DRL has a relatively more stable position error during all the time. In the second experiment, we add relatively larger noise to the environment. The left and right figures in Fig. 11c give the original and actual trajectories of the model based and DRL based tracking. As indicated, the model based approach is not able to keep the given trajectory, while the learning based approach can.

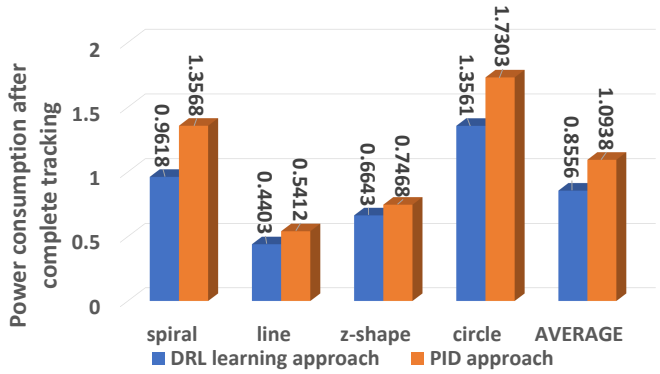


Fig. 10: Tracking result comparison between proposed DRL-based framework and PID-based baseline in terms of power consumption

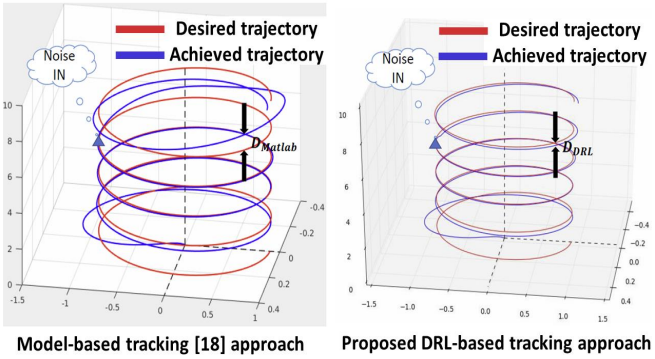
E. Hardware Performance on FPGA

A truly autonomous UAV has three components, sensing, detection and control. One of the benefits of adopting DRL based trajectory tracking is that it solves the control problem using a deep neural network, which is known to be efficient for detection and sensor signal processing. Using a unified computation model (i.e. DNN) for different tasks allows us to design highly optimized application specific hardware for that computation model, instead of relying on flexible general purpose processor, which is either too bulky or cannot provide enough computation power.

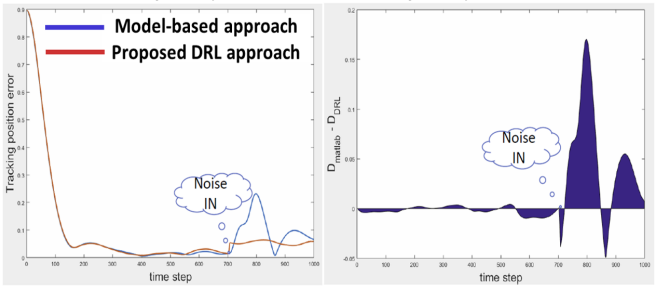
While the training of the DRL framework requires both the actor and critic networks, only the actor network needs to be implemented on the UAV during the runtime and run in real-time. To evaluate the cost, payload and energy impact that the actor network may bring to the UAV, we implement our actor model on the FPGA platform to validate our method on the real-world devices. We choose Intel (Altera) Cyclone V 5CGX FPGA as our evaluation platform. The platform's SoC (System-on-Chip) has the maximum CPU clock frequency of $925MHz$ and embedded DDR3 SDRAM with the memory bandwidth pf around $4,500MB/s$. The actor network has $25,196$ connections, which correspond to $25,196$ multiplication/addition operations. We have to time multiplex the hard-

TABLE II: FPGA performance analysis for actor model in proposed DRL-based framework

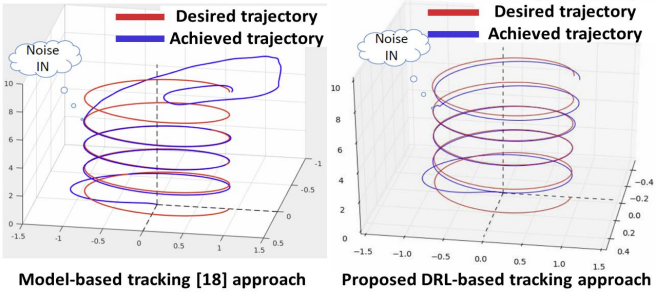
Frequency	373.02 MHz
Throughput	204.96 Action/s
Total Power	33.57 μW
Logic Utilization in ALM	10.03 %
RAM Utilization	20.94 %



(a) Comparison of achieved trajectories with a relatively small noise (*red*: desired trajectory, *blue*: achieved trajectory)



(b) Comparison of tracking position error



(c) Comparison of achieved trajectories with larger noise (*red*: desired trajectory, *blue*: achieved trajectory)

Fig. 11: Experimental results of trajectories tracking in noisy environment comparisons between using model-based tracking [18] and using proposed DRL-based tracking scheme

ware resource in FPGA in order to evaluate the whole network. The amount of computations that can offload to the FPGA is constrained by the size of the on-chip RAM, which will be used to store the weight parameters and intermediate results. With up to 21% utilization of the RAM resources, we use 10% utilization of the programmable logic resource to achieve 200 actions/s throughput. We present the performance and energy consumption of our FPGA implementation in Table II. In this implementation 16-bit wide fixed-point data precision is used. The results show that, the power consumption of the actor network is very low, hence enables the model to run on the UAV devices, which usually have stringent energy resources. Please note we include the static power in the total power. The real computation power should be much lower.

VI. Conclusion

We have introduced a framework for UAV trajectory tracking based on deep reinforcement learning using FPGA. The system structure, processing algorithm and software/hardware performance are presented. In our approach, the UAV tracks a desired trajectory through a set of given waypoints, tolerating random Gaussian noise within considerable range. The hardware consumption of the implementation of this scheme is also provided. The proposed scheme is general and applicable to be applied in real UAVs for fast and accurate trajectory tracking system.

References

[1] F. Sassi, M. Abbes, and A. Mami, "Fpga implementation of pid controller," *Proceedings-Copyright IPCO*, 2014.

[2] L. Marconi and R. Naldi, "Robust full degree-of-freedom tracking control of a helicopter," *Automatica*, 2007.

[3] L. Marconi and R. Naldi, "Aggressive control of helicopters in presence of parametric and dynamical uncertainties," *Mechatronics*, 2008.

[4] P. E. Pounds, D. R. Bersak, and A. M. Dollar, "Stability of small-scale uav helicopters and quadrotors with added payload mass under pid control," *Autonomous Robots*, 2012.

[5] Y. Chen and Q. Wu, ""design and implementation of pid controller based on fpga and genetic algorithm"," in *Electronics and Optoelectronics (ICEOE), 2011 International Conference on*, IEEE, 2011.

[6] F. Goodarzi, D. Lee, and T. Lee, "Geometric nonlinear pid control of a quadrotor uav on se (3)," in *Control Conference (ECC), 2013 European*, IEEE, 2013.

[7] N. Mohajerin and S. L. Waslander, "Modelling a quadrotor vehicle using a modular deep recurrent neural network," in *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*, IEEE, 2015.

[8] M. Collotta, G. Pau, and R. Caponetto, "A real-time system based on a neural network model to control hexacopter trajectories," in *Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM), 2014 International Symposium on*, IEEE, 2014.

[9] A. Din, B. Bona, J. Morrissette, M. Hussain, M. Violante, and M. F. Naseem, "Embedded low power controller for autonomous landing of uav using artificial neural network," in *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, IEEE, 2012.

[10] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[12] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2013.

[13] S. Yue, D. Zhu, Y. Wang, and M. Pedram, "Reinforcement learning based dynamic power management with a hybrid power supply," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, IEEE.

[14] 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.

[15] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, 2016.

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

[17] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[18] S. P. Viswanathan, A. K. Sanyal, and E. Samiei, "Integrated guidance and feedback control of underactuated robotics system in se (3)," *Journal of Intelligent & Robotic Systems*, 2018.

[19] B. L. Sharma, N. Khatri, and A. Sharma, "An analytical review on fpga based autonomous flight control system for small uavs," in *Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on*, IEEE, 2016.

[20] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, *et al.*, "Can fpgas beat gpus in accelerating next-generation deep neural networks?," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ACM, 2017.

[21] R. H. Klenke, "A uav-based computer engineering capstone senior design project," in *Microelectronic Systems Education, 2005.(MSE'05). Proceedings. 2005 IEEE International Conference on*, IEEE, 2005.

[22] R. Konomura and K. Hori, "Phenox: Zynq 7000 based quadcopter robot," in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, IEEE, 2014.

[23] S. Li, T. Liu, C. Zhang, D.-Y. Yeung, and S. Shen, "Learning unmanned aerial vehicle control for autonomous target following," *arXiv preprint arXiv:1709.08233*, 2017.

[24] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "Dlau: A scalable deep learning accelerator unit on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[25] S. Liao, Z. Li, X. Lin, Q. Qiu, Y. Wang, and B. Yuan, "Energy-efficient, high-performance, highly-compressed deep neural network design using block-circulant matrices," in *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*, IEEE.

[26] J. Su, J. Liu, D. B. Thomas, and P. Y. Cheung, "Neural network based reinforcement learning acceleration on fpga platforms," *ACM SIGARCH Computer Architecture News*, 2017.

[27] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[28] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[29] F. Chollet *et al.*, "Keras." https://github.com/fchollet/keras, 2015.