Reinforcement Learning Control for Quadrotors using Snapdragon Flight

Conference Paper · January 2019

CITATIONS
0

READS
17

4 authors, including:

Abhishek Shastry
University of Maryland, College Park
5 PUBLICATIONS **16** CITATIONS
SEE PROFILE

Vikram Hrishikeshavan
University of Maryland, College Park
45 PUBLICATIONS **261** CITATIONS
SEE PROFILE

Inderjit Chopra
University of Maryland, College Park
487 PUBLICATIONS **8,321** CITATIONS
SEE PROFILE

Reinforcement Learning Control for Quadrotors using Snapdragon Flight

Eric Solomon
Graduate Research Assistant

Abhishek Shastry
Graduate Research Assistant

Vikram Hrishikeshavan
Assistant Research Scientist

Inderjit Chopra
Alfred Gessow Professor and
Director of the Alfred Gessow Rotorcraft Center
Alfred Gessow Rotorcraft Center
University of Maryland
College Park, Maryland, U.S.A

ABSTRACT

As unmanned aerial systems(UAS) begin to operate in more dynamic and stochastic environments as presented by urban setting, new problems for tracking and control are introduced that must be addressed. Reinforcement-Learning(RL) techniques for control combined with deep-learning are promising methods for aiding UAS in such environments. This paper is an exploration of use of some of the popular Reinforcement Learning algorithms in aerospace, that have already been successfully used in non-aerospace applications such as self-driving cars. The vehicle chosen for this study is a quadrotor. The simulation results demonstrates that a RL controller performs slightly better than PID (Proportional-Integral-Derivative) in well-controlled deterministic environments and hence shows potential for more rigorous testing in highly uncertain environments. Also, some estimations of the processing capability required for these algorithms is presented and is used to demonstrate that these can be executed on some of the commercially available flight computers for drones such as Qualcomm Snapdragon Flight™.

*

Nomenclature

ACKTR	Actor-Critic using Kronecker-Factored Trust Region
API	Application Programming Interface
DQN	Deep Q-network
ELKA	Enhanced Lightweight Kinematic Autopilot
FFNN	Feed Forward Neural Network
LSTM	Long Short-Term Memory network
MDP	Markov Decision Process
NLP	Natural language Processing
PID	Proportional Integral Derivative controller
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
ROS	Robotic Operating System
TRPO	Trust Region Policy Optimization
VREP	Virtual Robot Experimentation Platform

INTRODUCTION

Commercial drone is one of the fastest growing areas and as per the Technavio's market research and analysis report (Ref. 1), it is forecasted to reach $15 billion in value by 2022. The vehicle's applications include geospatial mapping, inspection of civil structures, crowd security and surveillance, aerial photography, search and rescue, and cargo delivery. It is evident that most of these tasks will require safe autonomous operations in an urban setting. The urban environment is very diverse, rapidly changing and largely unpredictable(stochastic), unlike the high skies where most aerospace vehicles have been operating for more than a century. Hence there is a requirement for the development and validation of robust adaptive navigation and control tools, more complex than the conventional PID controller.

Quadrotors have proven to be an important benchmark for testing flight controls using complex control laws. It is mechanically simpler than a conventional helicopter while its dynamics is relatively more straightforward and intuitive than other types of drones. This greatly simplifies the process of controller development for this vehicle. Because many of the popular simulation techniques for adaptive control often require a large amount of memory and processing capability, these techniques must be adapted with the computationally limited real-world hardwares in mind such as Qualcomm Snapdragon Flight™ (Ref. 2). This paper begins with the first portion of the controller design process: simulation.

We explore two types of reinforcement learning(RL) control in this paper, one which requires action (or control) to be a small discrete set and the other which requires it to be continuous. We find that it is best to use continuous action control policies and that neural training is most feasibly carried out by separating the quadrotor's translational and rotational degrees of freedom. The results are merely an exploration of the usage of reinforcement learning for quadrotor control.

Presented at the Vertical Flight Society Autonomous VTOL Technical Meeting and Electric VTOL Symposium, Mesa, Arizona, USA, Jan. 29-31, 2019.

BACKGROUND

Neural networks as a tool have been popular recently for both model prediction and control. Punjani and Abbeel (Ref. 3) showed the ability of feed-forward neural networks to predict quadrotor's trajectory over a short horizon successfully. Ogunmolu and Olalekan (Ref. 4) has shown that deep neural networks are effective model predictors from input-output data. Neural networks with layers originally proven useful for time-series prediction and Natural Language Processing (NLP), namely recurrent and Long Short-term memory networks (LSTM), have shown improved performance in model predction (Ref. 5). Recently, Lou and Guo (Ref. 6) have demonstrated in simulations, that reinforcement learning algorithms can provide good trajectory tracking even in the presence of strong gusts. RL has a few proven real world applications such as the program AlphaGo Zero by Google Deepmind that mastered and defeated world champions in the game of GO (Ref. 7) and a robot that learned how to flip pancakes from human demonstrations (Ref. 8).

The basis of all Reinforcement Learning (RL) algorithms employing neural networks for control is Markov Decision Process (MDP), which is represented by the following figure:

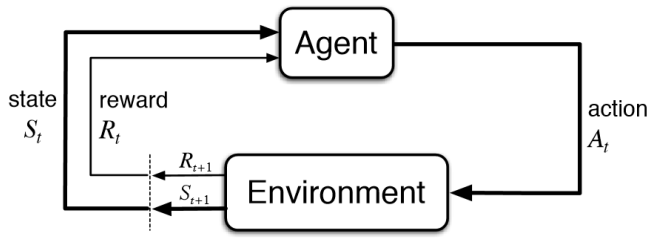

Fig. 1. Markov Decision Process (Ref. 9)

The environment in MDP is the external system to be controlled. The agent is the program executing actions on environment and receiving rewards and the current state of the environment in return. Action by an agent is same as control input to a system in classical control terminology. A policy in MDP is any function which outputs an action for a given state of the environment. It is to be noted that an agent can either follow a policy or act randomly, which is common during training phase of RL algorithms. All RL algorithms are focussed on finding optimum policy functions for any given environment.

RL controllers have primarily two benefits over traditional analytical controllers namely model-free algorithm and emergent behaviour.

Model-free algorithm: RL controller design does not require the knowledge of system's dynamics apriori. It learns the dynamics during training phase. Hence by nature, it can adapt to changes in plant's dynamics if the training is continued online. This adaptive nature of RL controllers, also make it more robust than PID controllers. It can by design tackle problems like gust, turbulence, ground effect (on one or multiple rotors), partial actuator failure, etc.

Emergent behaviour: For RL controllers, system behaviour emerge during training so as to perform the task as decided by the reward function optimally. For PID these are by design limited to the linear regime and are fixed throughout the system operation. The behaviour of an RL system can be nonlinear and change as the environment changes. This comes with the downside that since the behaviour is not known apriori, it could be undesirable in environments the algorithm has not been trained before.

Guided-policy learning has been a stepping stone towards fully model-free real-time neural control. An offboard trained controller has shown performance benefits at test time. This type of controller has performed faster at test time and maintained good performance in stochastic and unstructured environments (Ref. 10). Simulation results have shown the simple DQN (Deep Q-networks) reinforcement learning algorithm to perform on par with and occasionally better than human pilots (Ref. 12).

Recent reinforcement learning (RL) algorithms have been focused on speeding up the training phase. Techniques such as neural network freezing and experience replay have shown to be beneficial (Ref. 13) for controller performance over time. TRPO (Ref. 14) and PPO (Ref. 15) present better bounds on the approximation performance function versus the actual performance function, which has led to significant improvements in controller performance. Hierarchical learning techniques show the benefit of simplifying control actions into known solutions (Ref. 16). Recently, the gradient noise scale of a training dataset has shown to be directly related to the difficulty of training with a particular dataset batch size (Ref. 17).

METHODOLOGY

This section describes the designs for the reinforcement learning quadrotor controllers present in this study. First is a description of the controller structure as a connectionist system. Following that is a description of the adaptive algorithm.

Controller structure

This simulation experiment presents a comparison between two popular deep reinforcement learning algorithms: Deep Q-Network (DQN) and Proximal Policy Optimization (PPO). DQN is a value iteration method of RL, that is to say it learns to approximate the value of each (state, action) pair in the environment as represented by a neural network. The agent in the algorithm then typically follows the greedy policy of taking the action with the highest value (Ref. 18). DQN has been successfully used for solving many atari-style games and even to simulate the optimum move of a car stuck in traffic (Ref. 19). However, DQN requires the action set to be discrete, which has significant limitations for continuous real-time systems such as quadrotor. This is overcome by another type of RL algorithms known as policy gradient

algorithms in which the policy is represented as another neural network. This type of RL algorithms has been used for autonomous driving in cars (Ref. 20), optimization of chemical reactions (Ref. 21), etc. This paper uses one such policy gradient algorithms known as Proximal Policy Optimization (PPO). PPO is presented here in actor-critic style and hence requires two neural nets, one for the policy (the actor) and another for the value (the critic) of each state.

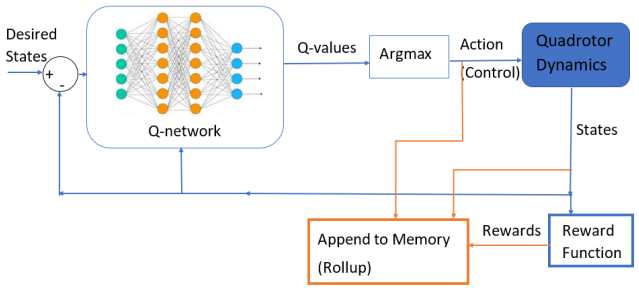

Fig. 2. Deep Q-Network structure. The neural network takes the vehicle state as input and calculates the Q-value of all actions. Under greedy policy, the action with max Q-value is executed.

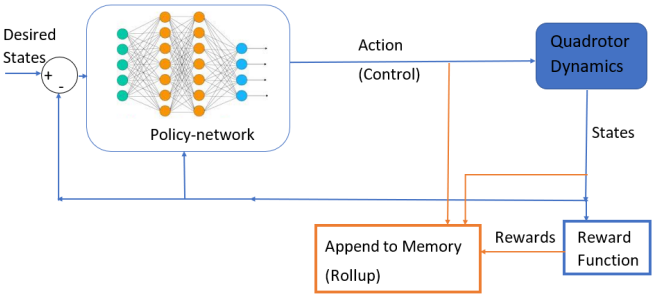

Fig. 3. Proximal Policy Optimization structure. In addition to the value network(as in DQN), policy is also represented by a neural net. The output of policy network is used for control and while the value network is used to train the policy network.

The simulation experiment is designed to test the ability of RL algorithms to control a hovering quadrotor. Neural networks used anywhere in the controller are simple feed forward neural networks. The input data set being used by both the controllers is the 6 Degree-of-Freedom (DoF) pose representing the current state, concatenated with the 6 DoF pose error (Fig. 2). The output of DQN network is the value for all the actions. The output of PPO's critc network is the same as DQN network while the output of its actor or policy network is a probability distribution over the continuous actions space. Since DQN requires discrete action space, the continuous action space is discretized for this purpose into 'bins' for every action variable. The bins in this work for every variable is to increase or decrease or no change of motor rpm of quadrotor by a fixed value. This set of bins is chosen because it requires a coarser discretization of the control action space, yet it allows for a high granularity of control actions to be achieved.

Controllers design

This subsection describes the adaptive control algorithm for DQN and PPO. High variance in results is an important problem in reinforcement learning(or deep learning in general)algorithms, because it leads to non-repeatability of results. Hence different techniques used to minimize the effect of variance on controller training is also presented here.

Network Freezing and Data Rollout The first methods of minimizing the effect of variance are called neural network freezing or controller freezing and data rollout. Freezing is the process of keeping an adaptive controller at a static state. At the same time, data is being collected in the form of the next 'rollout'. A rollout is conceptually the same as a line search in the reinforcement learning sense. It is a collection of data representing the previous test sample. In our case, the rollout is constructed of 1000 data points or a full test to the failure/success point, depending upon which occurs first. By not calculating an adaptive step after each state update, and instead calculating the adaptive step based on the expected value of a batch, the effect of noisy data points is filtered out. An alternative for rollout construction that is not explored in this paper is to keep a sliding filter representing the current rollout. This allows the adaptive step to be calculated at a quicker rate.

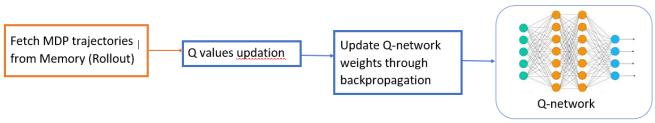

Fig. 4. DQN weight update. Training data for the neural nets is generated by executing the current policy for N(=1000) iterations. This is then used to to train the neural nets in the form of data rollout from memory.

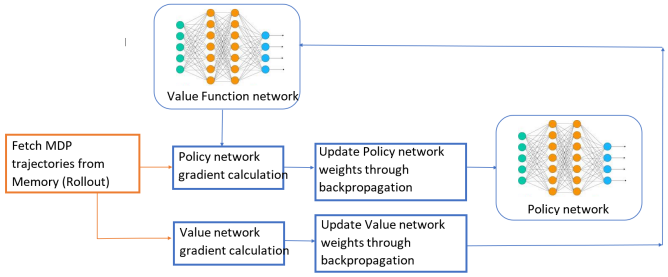

Fig. 5. PPO weight update. Training data for the neural nets is generated by executing the current policy for N(=1000) iterations. This is then used to to train the neural nets in the form of data rollout from memory.

function CONTROLANDROLLUP ▷ Control freezing and data roll-up
  while in flight **do**
    $N \leftarrow 1000$
    $v \leftarrow []$ ▷ Memory of experiences
    for 1 to N **do**
      $s \leftarrow quadstate$ ▷ control is via DQN or PPO
      $r, a, s_{next} \leftarrow control(s)$ ▷ Reward and Action from Control
      $v \leftarrow v :: [r, a, s, s_{next}]$ ▷ Append to memory
    end for
    NORMALIZE(v)
    REINFORCE(v) in parallel ▷ reinforce is also via DQN or PPO
  end while
end function

Parameter freezing The next method used in this paper to minimize the effect of variance is called parameter freezing. Thus, the controller neural networks are trained to integrate more degrees of freedom into control as time goes on. For this study, non-integrated degrees of freedom are controlled by a simple linear gain controller. Ideally, the controller will be fully model-free. However, this was infeasible for a preliminary study. The reinforcement learning controller integrates degrees of freedom in the order shown below.

Parameter freezing/training in steps

1. Planar (horizontal) position
2. Vertical position
3. Roll/Pitch
4. Yaw

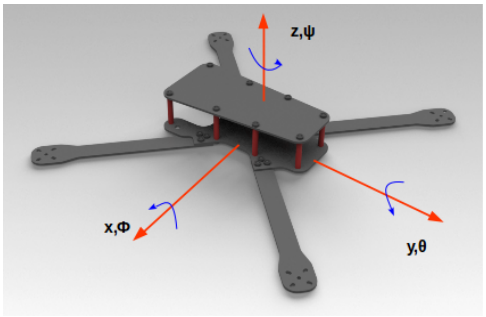

Fig. 6. Motion convention used for the quadrotor in V-REP simulation environment. For DQN's restricted DOF simulations, all linear motions are restricted, and rotational motion in z is restricted for 2-DOF simulation and additionally motion in x is restricted for 1-DOF simulation.

Data normalization Data and parameter normalization has shown to be an extremely effective and very simple method of dealing with data variance. The form of normalization known as 'batch normalization' is shown below (Ref. 22). It is performed using the data from the previous data rollout.

Input: A set of values of x $(x_1, x_2, ..., x_m)$

$$\text{BatchNormalize}(x_i) = \frac{\gamma_b(x_i-\mu)}{\sqrt{\sigma^2+\varepsilon}} + \beta$$

where, $\mu = \frac{1}{m}\sum_i x_i$, $\sigma^2 = \frac{1}{m}\sum_i (x_i-\mu)^2$ and γ_b, β are 'learnable' parameters

DQN parameter update The DQN parameter update method relies on the principles of Q-Learning, which is a popular reinforcement learning method that is guaranteed to find the optimal parametrization of a function(the policy) in a metric space with a given state evaluation function or value function. Value functions typically have the following form.

$$V_\pi(s_i) = \sum_{j=i}^{\infty} \gamma^{j-i} r(s_j, a_k = \pi(s_j))$$

where $r(s_j, a_k)$ is the reward obtained for taking action a_k in state s_j and $\gamma < 1$ is a discount factor (a hyperparameter) to indicate rewards in near future have preference over long-term rewards. Q-learning optimizes the policy for a given value function using another function known as Q-value function of a state-action pair and is defined as,

$$Q(s_i, a_k) = r(s_i, a_k) + \gamma V_\pi(s_{i+1})$$

It is evident that under the optimal policy(π^*), which maximizes the value function, the following relation holds:

$$V_{\pi^*}(s_i) = \max_{a_k} Q^*(s_i, a_k) = \max_{a_k}(r(s_i, a_k) + \gamma V_{\pi^*}(s_{i+1}))$$

Henceforth, we get the following:

$$Q^*(s_i, a_k) = r(s_i, a_k) + \gamma \max_{a_k} Q^*(s_{i+1}, a_k)$$

This is the Bellman equation of optimality (Ref. 23) written in terms of Q-value function. The update law for the Q-value function is derived from the bellman equation using fixed-point iteration method

$$\begin{aligned} Q_{new}(s_i, a_k) = (1-\alpha) Q_{old}(s_i, a_k) \\ + \alpha(r(s_i, a_k) + \gamma \max_{a_k} Q_{old}(s_{i+1}, a_k)) \end{aligned}$$

where α is the relaxation factor also known as 'learning rate' in reinforcement learning literatures.

Using this framework, the process of finding the optimal policy is simply reduced to process of finding a good reward function to optimize. Simple reward functions have recently overtaken expertly crafted reward functions in popularity due to their tendency to produce more robust control solutions (Ref. 24). Here, the reward function is the 2-norm of normalized state versus the normalized desired state.

$$n, n' = BatchNormalize(s, s')$$

function DQNREINFORCE(rollout vector)
  $\gamma \leftarrow 0.95$ ▷ Discount factor
  $\alpha \leftarrow 0.7$ ▷ Learning rate
  $N \leftarrow$ size(rollout vector)
  for 1 to N **do**
    $r, a, s, s_{next} \leftarrow rollout_{vector}.next$
    $q_{target} \leftarrow r + \gamma Q_{Horizon}(s_{next})$
    UPDATE WEIGHTS(q_{target}, Q(s), α) ▷ Weight update of Q-network
  end for
end function

$$R(n, n') = |n - n'|_2$$

The gradient of the Q-value function with respect to the parametrized neural controller is then taken and is used to update with the standard Q-Learning formula.

$$\begin{aligned}\delta w =& \alpha[r + \gamma \max_a Q(s_{i+1}, a, w) - Q(s_i, a, w)]\nabla_w Q(s, a, w) w \\ =& w + \alpha(q_{target} - q)\delta w q\end{aligned}$$

PPO parameter update The PPO parameter update method simultaneously updates the controller (actor) and control action evaluator (critic). The update of the evaluator is essentially the same as the update process for the DQN network. In fact, DQN can be thought of as using only the control action evaluator as a controller itself. PPO parameter update makes use of the advantage function instead of the value function for determining controller performance. The important difference is that the advantage function generates a dataset with much lower variance.

function PPOREINFORCE(rollout vector)
  $\varepsilon \leftarrow 0.2$
  $\delta \leftarrow 0.1$
  $A \leftarrow$ AdvantageRollup(rollout vector, γ)
  UPDATE WEIGHTS(A) ▷
end function

function ADVANTAGEROLLUP(rollout vector,γ)
  $v \leftarrow$ rollout vector
  $T \leftarrow$ length(rollout vector)
  $A \leftarrow []$
  for t in T **do**
    $d \leftarrow reward(v, t) + \gamma V(s_{t+1}) - V(s_t)$ ▷ Advantage
  end for
  for t in T **do**
    $A \leftarrow A :: \sum_t^T \gamma^t d_t$ ▷ Rollup
  end for
end function

The policy performance function is the standard PPO clipping function. It is shown below for reference. PPO and its predecessor TRPO rely on the principles of statistical information divergence in order to bound the parameter update step. The bounded region, after simple tuning, has the properties that it takes an adaptive step in the performance maximization direction that is relatively large compared to other algorithms but not too large.

$$L_t(\theta) = min(R_t(\theta)A_t, clip(R_t(\theta)), 1 - \varepsilon, 1 + \varepsilon)A_t$$

SIMULATION RESULTS

Simulation validation is performed in two stages. The first is on a simple set of reduced degree of freedom non-linear quadrotor rotational dynamics model using only DQN. The second is on a more robust nonlinear quadrotor model described in Virtual Robot Experimentation Platform (V-REP) (Ref. 25). The simplified dynamics is useful as a validation for reinforcement learning on quadrotors in general. The 1-DOF model tests the ability of DQN to control pitch only. All other degrees of freedom have been constrained. The 2-DOF model incorporates control of pitch in addition to roll.

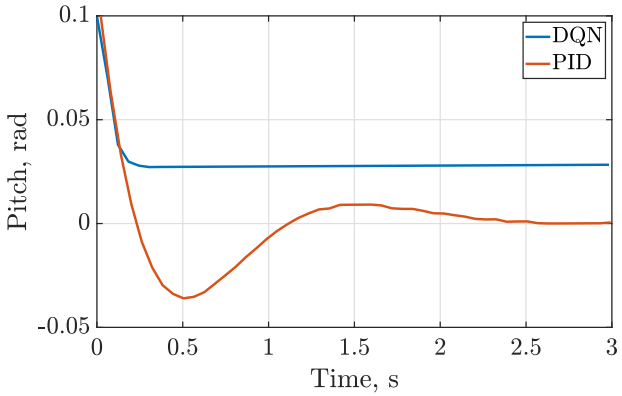

Fig. 7. Validation of Deep Q-Learning on a quadrotor to perform pitch control. The steady state error in DQN is believed to be the result of coarse discretization of continuous controls for quadrotor

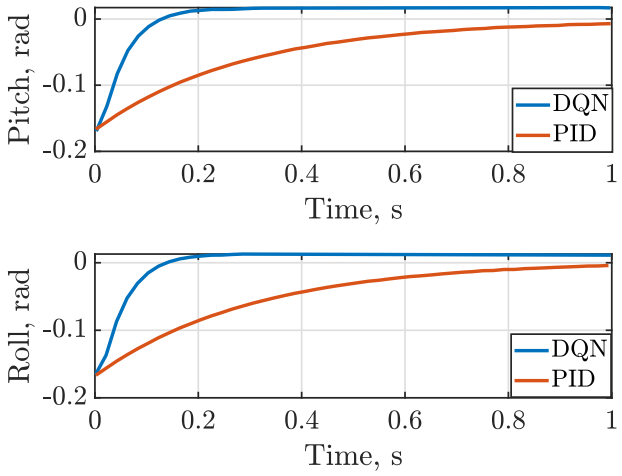

Fig. 8. Validation of Deep Q-learning on a quadrotor for 2d-attitude control. DQN performs only slightly better than linear PID.

As observed from the figures, DQN has atleast similar or slightly better performances to PID controller. The DQN controller though has a steady state error, the source of which is believed to be the discretization of the continuous action set for the controller. In the light of the above results, it is important here to point out that the benefits of RL controllers over traditional analytical controllers as model-free algorithm, being adaptive to changing environments and free from extensive online manual tuning, makes it worthwhile to implement it on flight hardware for validation in future.

Though DQN algorithm demonstrates good performance in the above simulation environment, it cannot be used in real-world environments primarily because it requires the action space to be discrete and finite. To overcome this limitation, continuous action reinforcement learning algorithms are required. Hence, it was decided to incorporate V-REP and the popular OpenAI implementations (Ref. 26) of modern reinforcement learning algorithms. Two continuous action reinforcement learning algorithms, PPO and ACKTR are tested (ACKTR algorithm is not described here, but is accessible through OpenAI baselines (Ref. 29)). For this set of simulation tests, the controller type is modified. A baseline PID controller is created for position and attitude control. This is so that the RL controller model can be easily adapted to the real-world, where avoiding catastrophic failure is of much greater importance.

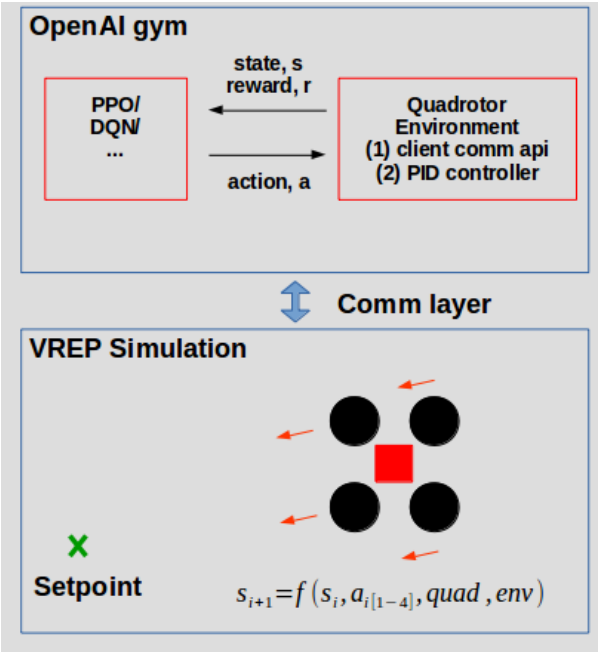

Fig. 9. V-REP + OpenAI quadrotor environment.

Although consistent improvement over PID is not yet shown, initial results suggest that the chosen experimentation method is promising. RL algorithms are run for approximately one million timesteps with time-stepping parameter of 0.001. For both PPO and ACKTR, only about 10 percent of the simulation rollouts ended in a stable quadrotor, however the successful rollouts are spread relatively evenly over epochs with the exception that success does not happen during the first 50 or so rollouts. No pattern for learning beneficial behaviors is immediately visible by taking a selection of the successful flight tests over training epochs. It is suggested that future simulations, experiment with hyperparameter tuning and different reward functions to achieve better results. The present results are not conclusive of successful continuous action RL control, but are nonetheless important. They demonstrate the repeatable and simple process by which a quadrotor with full-state, nonlinear dynamics is able to be tested in simulation. This process is useful for controller and general process design for RL-based quadrotor flight. For example, one may use the V-REP in conjunction with OpenAI simulation environment to examine RL controller learning processes that avoid catastrophic failure by building up control authority incrementally.

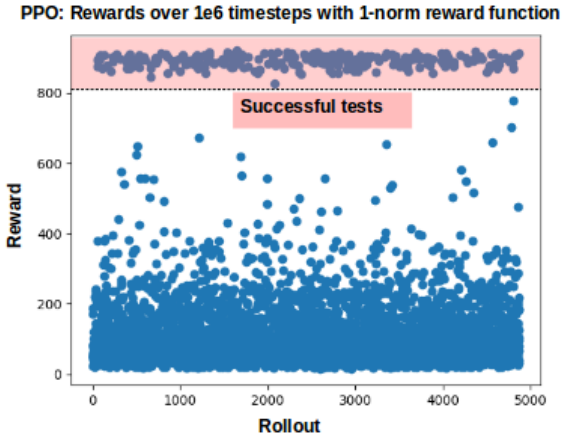

Fig. 10. PPO cumulative rewards over 10^6 timesteps and 5000 rollouts. Rewards over 800 represent tests that end due to reaching a time limit. These results do not demonstrate successful RL control. Hyperparameters tuning(such as learning rate) and different rewards functions are avenues of exploration for better results.

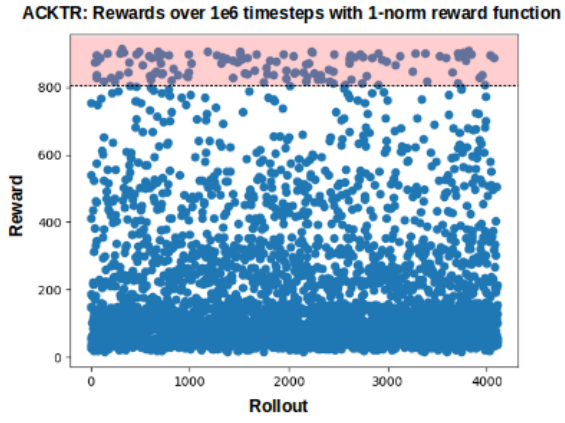

Fig. 11. ACKTR cumulative rewards over 1e6 timesteps and 5000 rollouts. Rewards over 800 represent tests that end due to reaching a time limit.

HARDWARE VALIDATION

A quadrotor platform (Fig.15) utilizing in-house developed ELKA and Qualcomm Snapdragon Flight ™ (Ref. 2) (Ref. 34) has been developed at University of Maryland. Localization using Visual-Inertial Odometry(VIO) has been implemented on Snapdragon Flight, the hardware architecture for which is presented in Figure.12. The position estimation results from VIO are shown in Figure13. A baseline PID controller has been implemented on the hardware and its results are presented in Figure14.

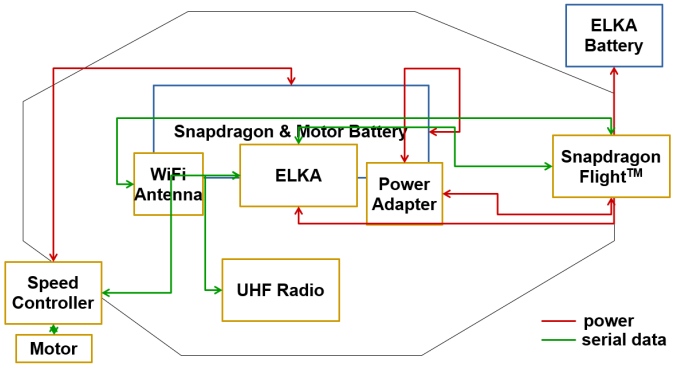

Fig. 12. Hardware architecture for Visual-Inertial Odometry. All the primary hardware components and communication between them is shown here.

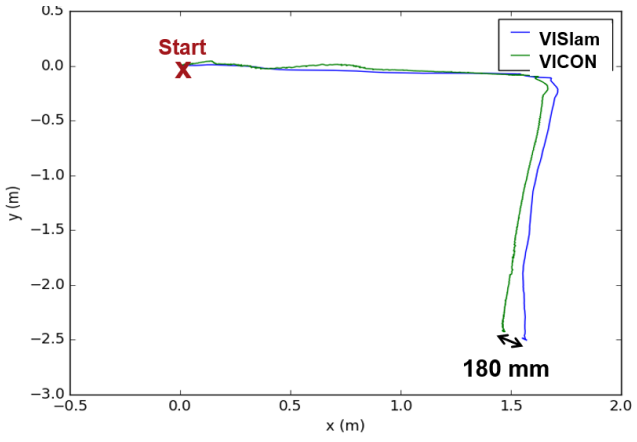

Fig. 13. Results of localization using Visual-Inertial Odometry(VISlam). The Vicon data is assumed as ground truth here. The drift in the VISlam position estimation small enough for it to be used reliably in position control.

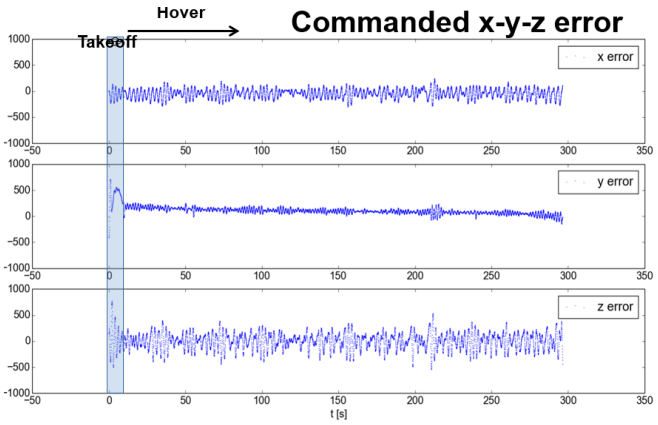

Fig. 14. Results of our naive PID controller for hover position control using VISlam. In order to circumvent tedious gain tuning process, we plan to implement an RL controller within our hardware framework. This will also help with the vehicle's operation in stochastic environments such as in presence of gust, unintended ground effect on one or multiple rotors, partial motor failure, etc.

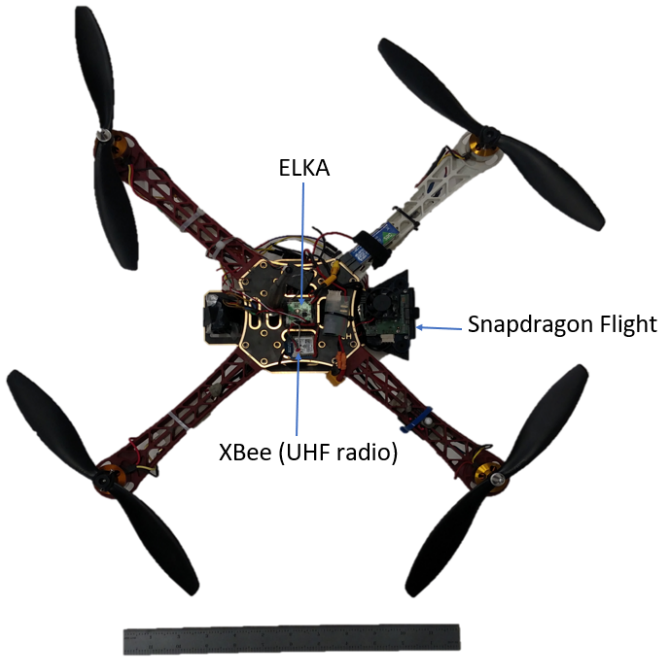

Fig. 15. Snapdragon™-ELKA (Enhanced Lightweight Kinematic Autopilot) controlled quadrotor with a ruler shown for size reference

Future hardware testing will utilize our developed quadrotor hardware using Qualcomm Snapdragon Flight™ coupled with Robotic Operating System (ROS) and Apache MXNet C++ API for tensor processing. Preliminary hardware benchmark tests for the feasibility of a system of symbolic graph computations for neural/tensor processing on an Intel i5-3320M show a PPO algorithm will require maximum operation frequency of 250Hz (Fig. 16). This is sufficient compared

to the necessary position control rate of 20Hz and would allow for 100,000 updates in 7 minutes, which is slightly shorter than the battery life of the tested quadrotor in hover. Also, the processor onboard the snapdragon flight runs at 2.26 Ghz which is similar to base frequency of 2.6Ghz of Intel i5-3320M. This suggests that it is possible to run the neural controller at a similar rate of 250Hz snapdragon. The onboard memory is limited to 2GB which might at worst limit the number of parameters of neural nets and the training batch size to be used for online training.

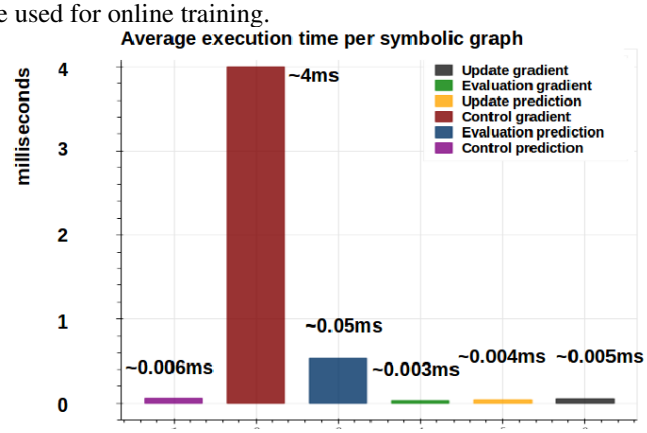

Fig. 16. MXNET PPO implementation time to execute each of the necessary symbolic graph computations. These benchmark tests were performed on a computer with Inteli5-3320M processor, that has a base frequency of 2.6 GHz. This benchmarking result shows that the algorithms can be executed on the Snapdragon at acceptable update rates.

FUTURE WORK

In continuation of this work, the first step is to validate a policy gradient based RL control on quadrotor's attitude dynamics in well-controlled deterministic simulation environment(i.e. without any real world uncertainities such as gust, ground effect, motor failure, etc.). Following this, the method is to be validated on position control of quadrotor in the same environment with tracking performance compared to PID or traditional linear compensators. After that, the controller's performance in stochastic simulation environment such as in the presence of gust, partial motor failure, etc will be tested. All the simulation results will then be verified with experiments on a real quadrotor. It is to be noted here that unlike classical controls, RL control has no proof of guaranteed stability, hence rigorous simulation testing, before hardware testing is even more important to have any kind of confidence on its ability.

CONCLUSION

This paper introduces reinforcement learning algorithms as a viable option for quadrotor control. It discusses their potential benefits over conventional PID controller and presents ways in which RL controllers can tackle problems inherently present in applications of drone in civilian airspace. The paper also describes two popular reinforcement learning algorithms (DQN and PPO) and demonstrate their ability to control a simplified model of a quadrotor in hover. The basis for more robust testing is then established in a repeatable and useful way. Some benchmark results on a conventional computer has been shown, which demonstrates that the amount of processing required for neural control is within the available limit of some of the flight computing boards available in market. This makes it feasible to apply these algorithms in real-time environment on drones. In addition, the paper also discusses methods to create a hybrid controller using both reinforcement learning and conventional PID to mitigate the effects of catastrophic failure on the quadrotor system.

Author contact:Eric Solomon, esolomon1221@gmail.com
Abhishek Shastry, shastry@umd.edu
Vikram Hrishikeshavan, vikram.h1@gmail.com
Inderjit Chopra, chopra@umd.edu

ACKNOWLEDGEMENTS

This work was supported by NAVAIR under the program under the Air Vehicle Test and Evaluation Engineering contract N00421-13-2-0001 with navy lead Mr. James Bumbaugh.

REFERENCES

[1]Technavio, "Global Consumer Camera Drones Market 2018-2022", March 2018, Infiniti Research Limited.

[2]Solomon, E., Hrishikeshavan, V., Chopra, I., (2018). Autonomous Quadrotor Control and Navigation with Snapdragon Flight. 74th Annual American Helicopter Society Forum: Phoenix, AZ.

[3]Punjani A. and Abbeel, P., "Deep learning helicopter dynamics models.", Robotics and Automation (ICRA), 2015 IEEE International Conference on. IEEE, 2015, pp 3223-3230.

[4]Ogunmolu, Olalekan, Xuejun Gu, Steve Jiang, and Nicholas Gans. "Nonlinear systems identification using deep dynamic neural networks." arXiv preprint arXiv:1610.01439 (2016).

[5]Mohajerin, N. and Waslander, S., "State initialization for recurrent neural network modeling of time-series data", International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, 2017, pp. 2330-2337.

[6]Lou, W., and Guo, X. "Adaptive Trajectory Tracking Control using Reinforcement Learning for Quadrotor. International Journal of Advanced Robotic Systems". https://doi.org/10.5772/62128

[7]David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert,

Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel and Demis Hassabis, "Mastering the game of Go without human knowledge", Nature volume 550,pages 354359, 19 October 2017.

[8]Kormushev, P., Sylvain C., and Darwin G. Caldwell. "Robot motor skill coordination with EM-based reinforcement learning." In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pp. 3232-3237. IEEE, 2010.

[9]https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287

[10]Zhang, T., Kahn, G., Levine, S. and Abbeel, P., "Learning Deep Control Policies for Autonomous Vehicles with MPC-Guided Policy Search", 2016 IEEE International Conference on Robotics and Automation (ICRA).

[11]Hwangbo, Jemin, Inkyu Sa, Roland Siegwart, and Marco Hutter. "Control of a quadrotor with reinforcement learning." IEEE Robotics and Automation Letters 2, no. 4 (2017): 2096-2103.

[12]Polvara, R., Patacchiola, M., Sharma, S.K., Wan, J., Manning, A., Sutton, R. and Cangelosi, A. "Autonomous Quadrotor Landing using Deep Reinforcement Lerarning", CoRR, 2017.

[13]Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D., "Rainbow: Combining Improvements in Deep Reinforcement Learning", 2017.

[14]Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization." In International Conference on Machine Learning, pp. 1889-1897. 2015.

[15]Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv:1707.06347v2.

[16]Frans, K., Ho, J., Chen, X., Abbeel, P., Schulman, S., "Meta Learning Shared Hierarchies", 2017.

[17]McCandlish, S., Kaplan, J., Amondei, D., OpenAI Dota Team, "An Empirical Model of Large-Batch Training, 2018.

[18]Mnih, V. et al, "Human-level control through deep reinforcement learning", Nature, doi:10.1038/nature14236, 2015.

[19]Fridman,Lex et al, "DeepTraffic: Crowdsourced Hyperparameter Tuning of Deep Reinforcement Learning Systems for Multi-Agent Dense Traffic Navigation", NIPS 2018, Deep Reinforcement Learning Workshop, doi:10.5281/zenodo.2530457 .

[20]Yu, April, Raphael Palefsky-Smith, and Rishi Bedi. "Deep reinforcement learning for simulated autonomous vehicle control." Course Project Reports: Winter 2016 (CS231n: Convolutional Neural Networks for Visual Recognition) (2016): 1-7,

[21]Zhou, Zhenpeng et al, "Optimizing Chemical Reactions with Deep Reinforcement Learning", ACS Central Science 2017 3 (12), 1337-1344 , doi:10.1021/acscentsci.7b00492 .

[22]Ioffe, S., Szegedy, C., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:448-456, 2015.

[23]Bellman, R.E., "Dynamic Programming", Princeton University Press, Princeton, NJ. Republished 2003: Dover, ISBN 0-486-42809-5

[24]Heess, N., et al, "Emergence of Locomotion Behaviours in Rich Environments", arXiv:2707.02286v2, 2017.

[25]Coppelia Robotics, "Virtual robot experimentation platform", 2018.

[26]OpenAI, "Gym: A toolkit for developing and comparing reinforcement learning algorithms", 2018.

[27]OpenAI, "Baselines: high-quality implementations of reinforcement learning algorithms", 2018.

[28]Klopf, A.H., "Brain Function and Adaptive Systems - A Heterostatic Theory", Air Force Cambridge Research Laboratories, 1972.

[29]Wu, Y., et al., "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation", arXiv:1708.05144v2, 2017.

[30]Annaswamy, A.B., Aziz, A. (1992). An Adaptive Controller for Aerospace Vehicles. Guidance, Navigation and Control Conference: Hilton Head Island, SC.

[31]Berrios, M., Berger, T., Tischler, M., Juhasz, O., Sanders, F., (2017). Hover Flight Control Design for UAS Using Performance-based Disturbance Rejection Requirements. 73rd Annual American Helicopter Society Forum: Fort Worth, TX.

[32]Lewis, F. (2005). Neural Networks in Feedback Control Systems. Mechanical Engineers Handbook, John Wiley: NY

[33]Heess, N. et al. (2017). Emergence of Locomotion Behaviors in Rich Environments. arXiv:1707.02286v2.

[34]Solomon, E., Vorwald, C., Hrishikeshavan, V., Chopra, I.. (2017). Visual Odometry Onboard a Micro Air Vehicle Using Snapdragon Flight. 7th AHS Technical Meeting on VTOL Unmanned Aircraft Systems and Autonomy: Mesa, AZ.

[35]Gremillion, G., Humbert, J. (2010). System Identification of a Quadrotor Micro Air Vehicle. AIAA Atmospheric Flight Mechanics Conference.

[36]Basri, M., Husain, A., Danapalasingam, K. (2015). GSA-based optimal backstepping controller with a fuzzy compensator for robust control of an autonomous quadrotor UAV. Aircraft Engineering and Aerospace Technology: An International Journal. Vol 87 Iss 5 pp. 493-505.

[37]Weiss, S., Achtelik, M., Lynen, S., Achtelik, M., Kneip, L., Chli, M., Siegwart, R. (2013). Monocular Vision for Long-term Micro Aerial Vehicle State Estimation: A Compendium. Journal of Field Robotics. Vol 30 Iss 5.

[38]Wang, J., Olson, E. (2016). AprilTag 2: Efficient and robust fiducial detection. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).