2024 Design, Automation & Test in Europe Conference (DATE 2024)
Special Initiative "Autonomous Systems Design"

Adaptive Perception Control for Aerial Robots with Twin Delayed DDPG

Veera Venkata Ram Murali Krishna Rao Muvva
School of Computing
University of Nebraska - Lincoln
Lincoln, Nebraska
krishna@huskers.unl.edu

Kunjan Theodore Joseph
Biological Systems Engineering
University of Nebraska - Lincoln
Lincoln, Nebraska
kjoseph2@unl.edu

Kruttidipta Samal
VMWare
Atlanta, Georgia
deep.samal@gmail.com

Marilyn Wolf
School of Computing
University of Nebraska - Lincoln
Lincoln, Nebraska
mwolf@unl.edu

Santosh Pitla
Biological Systems Engineering
University of Nebraska - Lincoln
Lincoln, Nebraska
spitla2@unl.edu

***Abstract*—Robotic perception is commonly assisted by convolutional neural networks. However, these networks are static in nature and do not adjust to changes in the environment. Additionally, these are computationally complex and impose latency in inference. We propose an adaptive perception system that changes in response to the robot's requirements. The perception controller has been designed using a recently proposed reinforcement learning technique called Twin Delayed DDPG (TD3). Our proposed method outperformed the baseline approaches.**

***Index Terms*—Closed loop systems, Control Systems, Deep Learning, Neural Networks, UAV, UAS, Drone,**

I. Introduction

Advances in deep learning, particularly convolutional neural networks (CNNs), enable robust perception systems for robots [1] [2] [3] [4] [5]. However, these neural networks are static and cannot adapt in accordance with the dynamics of the environment [6]. In addition, CNNs require a significant amount of computations, which presents a challenge to systems with limited computational resources. These include aerial robots (such as multirotors and fixed-wing aircraft) [4] [6]. Furthermore, CNNs have a tradeoff between accuracy and latency [7] [8]. A computation-intensive neural network is likely to be highly accurate, however, it tends to have a high inference delay [9]. Conversely, lightweight neural networks have a low inference delay, but they tend to be less accurate [10]. One or the other alone cannot be sufficient to achieve the goals of a robot when designing agile autonomous systems, such as the flight planning for unmanned aerial systems (UAS). The UAS requires to respond quickly in some instances (such as reacting to the changes in the dynamic environment), whereas in others it requires to respond with more confident state estimation (such as keeping track of goal pose) [6]. Thus, it is not possible to resolve this issue with only a high accuracy high latency (HLHA) or a low accuracy low latency (LLLA) CNN. The problem can be resolved by adopting an adaptive perception approach. The testbed's HLHA network has accuracy of 98% and latency of 400 milliseconds, the LLLA network has accuracy of 75% and latency of 280 milliseconds (These are described in detail in section II-A). Either of these networks are not sufficient to accommodate the plant due to the trade-off of latency vs accuracy [11] [6]. Furthermore, unlike some applications that evaluate individual images from different times and locations, in the robotic perception system the frames are continuous, so the success or failure of the detection is dependent on the previous detection [12]. (It is more likely that the next frame will have fewer critical feature points if the current frame has fewer, since the frames are consecutive). We have therefore formulated the error of perception in such systems as a Markovian process, and implemented a reinforcement learning-based perception controller to achieve adaptive perception in response to changes in the environment. Perception reinforcement learning control is implemented using the Twin Delayed Deep Deterministic Policy Gradient (TD3) [13] approach.

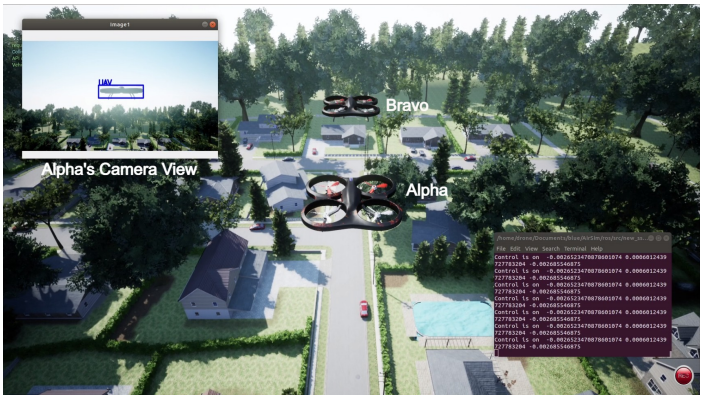

Fig. 1: A scene from $Alpha$ tracking $Bravo$.

To test our approach, we have used a UAS-UAS tracking scenario (Refer figure 1) in which $Bravo$ moves in a trajectory that is unknown to $Alpha$. $Alpha$'s objective is to track and follow $Bravo$ through the perception guided navigation. Since $Bravo$ can change directions rapidly at certain times, $Alpha$

979-8-3503-4859-0/DATE24/© 2024 EDAA

requires a robust perception system in order to be successful. According to the results, our proposed system outperforms the baseline HLHA and LLLA perception systems. Since the high latency in HLHA perception system made $Alpha$ unable to identify the quick changes in $Bravo$'s pose and adjusting its course. On the other hand, despite the fact that LLLA perception system was capable of providing inferences quickly, its detection failures prevented $Alpha$ to obtain precise state estimation of $Bravo$. The proposed perception system is adapted to changes in $Bravo$'s behavior by acting at times as a HLHA network, at other times as a LLLA network, and at remaining times as a combination of HLHA and LLLA. As a result, it is capable of tracking $Bravo$ more effectively than other methods.

The major contributions of this paper are

- **Adaptive Darknet :** We have modified the Darknet C code and implemented Adaptive Darknet. Through this one can create their own Adaptive YOLO CNNs. These adaptive CNNs are capable of turning on and off the filters in accordance with their significance for the scenario.
- **Adaptive Perception Controller :** Implementation of a novel adaptive perception controller with state-of-the-art of reinforcement learning technique i.e. Twin Delayed DDPG (TD3). In accordance with our knowledge this is the first attempt of implementing a perception reinforcement learning controller.

II. Methodology

A. Experimental Setup

The experimental machine contains AMD Ryzen 9 3900X 12-Core processor, NVIDIA's two GeForce RTX 2080 Ti Graphics Processors, and 32 GB primary memory. On the software side we have established a co-simulation setup by integrating Airsim [14], ROS [15], and Darknet. AirSim was used to simulate UASs. Tiny YOLOv3 [16] [17] Convolutional Neural Network (CNN) was trained by Darknet for the perception. ROS is used for integrating and executing the perception and control modules commands. In order to emulate UAV onboard computing resources (typically, UAVs do not contain GPU resources), systems computations (i.e. neural networks computations and motion control computations) ***are conducted only on the CPU*** (i.e. neural networks calculations and motion control computations). To measure the latency of the network we have utilized the *getrusage* module in C, the hardware timer has precision of 10 milliseconds [18]. The *getrusage* module measures user time (the time the processor spent working on the program) and system time (the time the processor spent working with the OS functions for the program) separately [19]. For the latency calculations the authors have modified the Darknet C code and added *ru_utime* (resource usage user time) to measured the latency, which is illustrated in figure 5 . The baseline approaches for experiments are as follows. The original Tiny Yolov3 acts as a high latency and high accuracy (HLHA) network, which has an accuracy of about 98% and it uses all the 3220 filters, and has latency around 400 milliseconds on tested CPU. The other is a truncated Tiny Yolov3 network that contains 75% of the standard Tiny Yolov3 convolutional filters, with an accuracy of approximately 75% and has latency around 280 milliseconds on tested CPU. This network acts as a low latency low accuracy (LLLA) network. The proposed method has varying accuracy and latency.

B. Design and Training of the Adaptive CNN

1) Design of Adaptive CNN: CNNs are made up of convolution, pooling, and fully connected layers. However, the majority of the computations happen in the convolutional layers. Let m be the number of convolutional layers, and n_i be the number of filters in the i_{th} layer. Let $\mathscr{C}$ represents the number of computations requires for a CNN. It can be shown mathematically as equation 1. Pooling and fully connected operations are not shown for simplicity.

$$\mathscr{C} = \sum_{a_m=1}^{n_m} F_{ma_m} * \left(.... * \left(\sum_{a_1=1}^{n_1} F_{1a_1} * I + b_1 \right) ... \right) + b_m \tag{1}$$

In order to achieve adaptive perception, the filters are turned on and off in accordance with their significance. The significance of each filter is quantified by the combination of mean absolute sum of the filter and gradient of the filter. The mean absolute value of the filter indicates its significance in extracting the feature from the image. The gradient of the filter quantifies the importance of the filter in the inference. We created a metric to assess the significance of the filters by taking these two values into the account, which is shown in equation 2.

$$f^{iv} = \varphi \left(\frac{1}{uv} \sum_{a=1}^{u} \sum_{b=1}^{v} |f_{ab}| \right) + (1-\varphi) \left(\frac{1}{uv} \sum_{a=1}^{u} \sum_{b=1}^{v} \frac{\partial \mathscr{L}}{\partial f_{ab}} \right) \tag{2}$$

Let τ be the impact threshold, for a given situation all filters with impact values less than τ are discarded and only filters with impact values greater than τ are used for inference. In the next subsection, we will discuss how the perception controller calculates the impact threshold τ. The required computations for such a convolutional neural network is shown in the equations 3. So from equation 1 and equation 3 $\mathscr{C}'$ has less CNN computations than $\mathscr{C}$. However, in the existing deep learning libraries such as TensorFlow and PyTorch, deactivating certain filters was performed by replacing filters with zero weights. In such a scenario computations are still happening. So we have utilized and modified the original Darknet C code to avoid those unnecessary computations.

$$\mathscr{C}' = \sum_{a_m=1}^{n'_m} F_{ma_m} * \left(.... * \left(\sum_{a_1=1}^{n'_1} F_{1a_1} * I + b_1 \right) ... \right) + b_m \tag{3}$$

$$where\ n'_1 \leq n_1,\ n'_2 \leq n_2,\ n'_3 \leq n_3,\ ....,\ n'_{m-1} \leq n_{m-1},\ n'_m \leq n_m$$

2) Training of Adaptive CNN: A dataset containing two thousand images of UAS were created from AirSim to train the CNN. These images include several variations, such as different UAS's poses, orientations, different lighting conditions, and occlusion effects. The dataset was preprocessed to incorporate rotation and shearing effects.

When training the CNN the challenge is to achieve a linear drop in accuracy as the filters are deactivated. Often times, when one or a few filters are deactivated, the accuracy of the network falls to single digits. In order to overcome this problem we have used random regularized parameters during the training process. The high regularized parameter penalizes the large weights and forces them to be smaller, whereas the low regularized parameter allows the weights to be within a wide range of intervals [20]. For our training we selected a random regularized parameter for each epoch between selected small and large regularized values. Since a high regularized parameter forces the weights to fall in a small range of interval, it is difficult to achieve a gradual drop in accuracy. As the impact threshold increases most of the filters get discarded and the accuracy is dropped significantly. On the other hand, a small regularized parameter allows the weights to be in a wide range of intervals, allowing for a wide range of impact thresholds. However, a small regularized parameter also makes certain filters responsible for extracting specific features. When those weights were discarded the accuracy of the network drops significantly. A randomized regular parameter which oscillates between high and low regularization values makes the CNN behave in the way we wanted. When acting as a low regularized parameter it makes the network weights fall in a wide range of intervals. At other times when it acts as a high regularized parameter, it makes the network not reliant on only a few filters to extract the crucial features for detection. Consequently, the filters are discarded in linear fashion as impact threshold increases, which achieves a linear drop in accuracy (Refer figure 5). A for loop was used to calculate the accuracy pattern by increasing the impact threshold. The impact threshold was increased by 0.0002 in each iteration, and the inference was performed based on that threshold across the entire test dataset in order to calculate the accuracy. And latency [6] is determined by taking the average of the inference time of all the images on the test dataset, i.e. $l = \frac{1}{n(\mathcal{T})} \sum_{i=1}^{n(\mathcal{T})} \mathscr{T}\Big(C_{NN}(I_i, w, \tau)\Big)$. Where l represents the latency, $\mathcal{T}$ represents the test dataset, C_{NN} represents the adaptive CNN, I represents the image, w represents the weights of the network, τ represents the impact threshold of the iteration, $\mathscr{T}$ represents the inference time of the network with given image and impact threshold.For the filters the impact value is calculated with φ value of 0.82.

C. Perception Controller

1) Motivation for MDP Design: CNN-based object detectors trained on static image datasets assume that each image is sampled from an IID (Independent and Identically Distributed) distribution. Therefore, the predictions of such networks are also assumed to be independent. However in applications like this (Robotics or Autonomous Systems), the IID assumption does not hold as consecutive frames have similar patterns and hence the network prediction at any given time may dependent on the past predictions. So the the failure or success pattern of detection depends on previous frames.

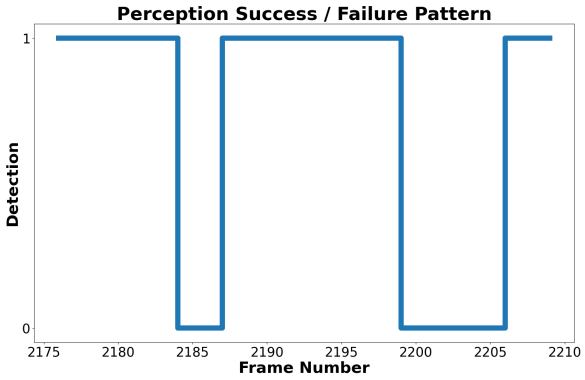

Fig. 2: Non-IID nature of CNN inference

Figure 2 shows this phenomenon occurring in a scenario. It can be observed that the frames from 2176 to 2184 and frames from 2188 to 2199 have consecutive successful detections (True Positives). And frames from 2200 to 2206 and frames from 2185 to 2187 have failed detections (False Negatives) continuously. These failures are resulting from occlusion, low illumination, and among others. Some failures are resultant of rotation of UAS as well. This phenomenon can be observed more in aerial robotics since CNNs are not rotational invariant. Aerial vehicles (such as multicopters and fixed wing aircrafts) rotate as they take maneuvers which is not common with ground vehicles. So certain failures are resultant of these maneuvers. This emphasis that the CNN assisted perception error distribution in autonomous systems is Markovian [12]. Hence we modeled the error of the perception module as a Markov Decision Process (MDP) with perception error and controller error, which are used to train the reinforcement learning based controller.

2) Design of Perception Controller: The perception controller was designed using a state-of-the-art deep reinforcement learning techniques known as Twin Delayed Deep Deterministic Policy Gradient (TD3). TD3 requires one actor, two critics, and target networks for each. The controller takes number of consecutive false negatives (C_{fn}) and $Bravo$'s image pose error (e_i) as input and produces τ as output. Based on the situation, the reward function is designed to balance perception accuracy and latency. Since the accuracy factor forces CNN to use all filters, the negative reward is proportional to the number of non active filters out of the total number of filters, shown in equation 5. F_t^A represents the set of actives filters at time step t, F represents set of all filters. Therefore, if the number of active filters decreases, the model gets penalized. The latency factor causes the CNN to discard certain filters in order to provide sufficiently decent inferences. The negative reward is inversely proportional to the Manhattan distance between $Alpha$ and $Bravo$, shown in 6. In case of a low Manhattan distance, $Alpha$ is tracking $Bravo$ well, so the model forces

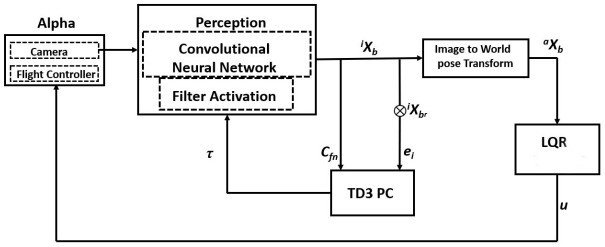

Fig. 3: Overall System Architecture.

it to produce a high impact threshold in order to reduce the number of active filters. In this way, the model attempts to find an impact threshold that is sufficient enough to give the best estimate of $Bravo$'s state while utilizing a minimum number of convolutional filters. Therefore, it attempts to achieve a dynamic perception by selecting the impact threshold based on the scenario. This reduces number computations conducted by convolutions. The overall reward is shown in 4. Upon the successful tracking (T_s) or failure tracking (T_f) of the episode a huge positive (100) or negative (-100) reward is provided. The training procedure of the TD3 perception controller is shown in algorithms 1.

$$r_t = r_t^p + r_t^l \tag{4}$$

$$r_t^p = -\xi_1\Big(1 - \frac{n(F_t^A)}{n(F)}\Big) \tag{5}$$

$$r_t^l = \frac{-\xi_2}{(|x_a - x_b| + |y_a - y_b| + |z_a - z_b|)} \tag{6}$$

D. Motion Controller

1) Image to World Pose Conversion: Image to global coordinate conversion is performed to provide the pose information to the controller. The CNN provides the image pose of the *Bravo*, which is illustrated in figure 4. The image pose of *Bravo* is converted to the world coordinate system with respect to the *Alpha*'s frame of reference. We perform a perspective transformation in order to obtain the coordinates of the front-back (x) axis, left-right axis (y) and the up-down axis (z). The pose conversions is shown in equation 7. *Bravo's* pose with respect to *Alpha* is sufficient for the controller input. However if needed transformation matrices can be used to determine the *Bravo* pose with respect to the world coordinate system.

$$[{}^a x_b,\ {}^a y_b,\ {}^a z_b, 1]^T = [R|T]^{-1} K^{-1} [u, v, 1]^T \tag{7}$$

2) LQR Controller: A linear quadratic regulator (LQR) controller is equipped to carry out the motion control strategy of the *Alpha*. The state space equation is shown in equation 8, where $A = 0_{4x4}$, $B = I_{4x4}$, $C = I_{4x4}$, $D = 0_{4x4}$, $\hat{\mathbf{x}} = [x, y, z, \psi]^T$ and $\mathbf{u} = [\dot{x}, \dot{y}, \dot{z}, \dot{\psi}]$

$$\dot{\hat{\mathbf{x}}} = A\hat{\mathbf{x}} + B\mathbf{u}\ ,\ \mathbf{y} = C\hat{\mathbf{x}} + D\mathbf{u} \tag{8}$$

The block diagram of LQR controller can be seen in figure 3. ${}^w X_b'$ is actual state of $Bravo$ in real world, whereas ${}^w X_b$ is the estimated state of $Bravo$ with the detection. The LQR gain K is calculated with Riccati equation with the help of state cost matrix Q and input cost matrix R. The control vector $\mathbf{u}$ is calculated as shown in equation 9. Where $\hat{x}_r = {}^w X_b + [x_d, y_d, z_d]^T$. And $[x_d, y_d, z_d]^T$ safe distance to keep away from $Bravo$, which is $[1, 0, 0]^T$.

$$\mathbf{u} = -K * (\hat{x}_r - \hat{x}_t) \tag{9}$$

Algorithm 1 TD3 Perception Controller Algorithm

Initialize actor network π_ϕ, critic networks Q_{θ_1} and Q_{θ_2} with initial weights ϕ, θ_1 and θ_2
Initialize target networks $\phi' \leftarrow \phi$, $\theta_1' \leftarrow \theta_1$, and $\theta_2' \leftarrow \theta_2$
Initialize replay buffer $\mathcal{B}$
Load adaptive CNN with weights w
Initialize AirSim and load $Alpha$ and $Bravo$
for $episode = 1, 2, \ldots, episodes$ **do**
  Initialize $Alpha$, Initialize $Bravo$
  $C_{fn} \leftarrow 0$, $e_i \leftarrow 0$, $\xi_1 = -0.5$, $\xi_2 = -0.25$
  while episode not done **do**
    $s \leftarrow [C_{fn}, e_i]$, $\tau \leftarrow \pi_\phi(s)$, $\bar{a} \leftarrow \pi_\phi'(s')$
    $o = C_{NN}(I, w, \tau)$
    if no detection **then** $C_{fn} += 1$
    else $e_i \leftarrow RMSE({}^i X_b)$, $C_{fn} \leftarrow 0$
    end if
    $s' \leftarrow [C_{fn}, e_i]$, Store (s, τ, r, s') in $\mathcal{B}$
    Fetch batch of N transitions of (s,a,r,s') from $\mathcal{B}$
    $\bar{a} \leftarrow \pi_\phi'(s') + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma')$
    $Q_1' \leftarrow Q_{\theta_1'}(s', \bar{a})$, $Q_2' \leftarrow Q_{\theta_2'}(s', \bar{a})$
    $Q_1 \leftarrow Q_{\theta_1}(s, a)$, $Q_2 \leftarrow Q_{\theta_2}(s, a)$
    $Q' \leftarrow r + min(Q_1', Q_2')$
    $\theta_1 \leftarrow argmin_{\theta_1} N^{-1} \sum (Q' - Q_1)^2$
    $\theta_2 \leftarrow argmin_{\theta_2} N^{-1} \sum (Q' - Q_2)^2$
    if $episode$ mod $delay$ **then**
      $\nabla_\phi J(\phi) \leftarrow N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
      $\phi' \leftarrow \eta\phi + (1 - \eta)\phi'$
      $\theta_1' \leftarrow \eta\theta_1 + (1 - \eta)\theta_1'$, $\theta_2' \leftarrow \eta\theta_2 + (1 - \eta)\theta_2'$
    end if
  end while
end for

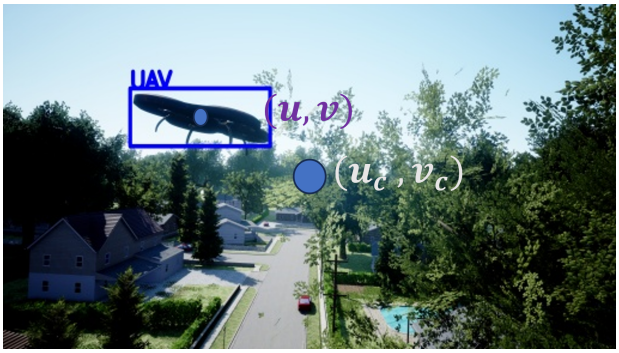

Fig. 4: Image Pose of Bravo.

III. Results

This section describes the results of the training of perception network, controller network, and overall results of the perfor-

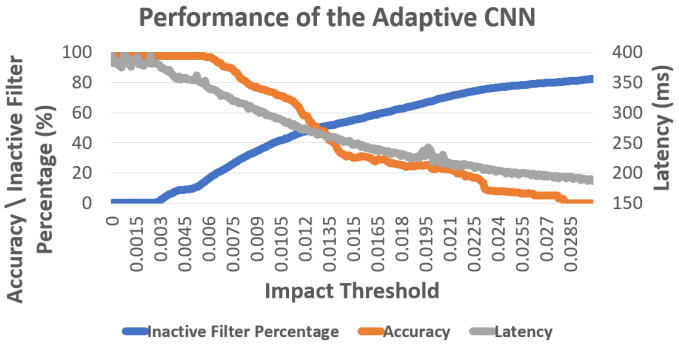

Fig. 5: Performance of CNN in terms of accuracy and latency as the inactive filter percentage increases

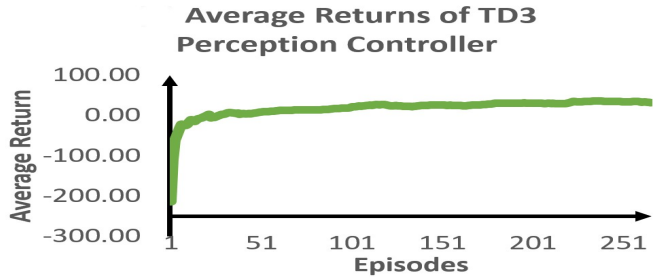

Fig. 6: Average Return over TD3 Perception Controller Training.

mance of the system.

A. Perception Performance

The CNN is trained with varying regularized parameter, as stated above. The training procedure is carried with two thousand images. The performance of the network can be seen in the figure 5. We achieved the plot using Yolov3 with the modified Darknet (Darknet is compiled with the 'CPU' option). It can be observed that accuracy drops proportionally as the inactive filter percentage increases, which also leverages the latency advantage. By varying the impact thresholds, this network is capable of providing different configurations of accuracy and latency. So an adaptive perception could be achieved with this network in accordance with the requirement, which can yield different sampling rates.

B. Perception Controller Performance

Perception controller TD3 model training was carried out as shown in the algorithms 1. The average returns of training is shown in figure 6. It took around 200 episodes for the training to reach equilibrium. The living penalty caused few fluctuations, since action for accuracy is action against the latency. So an impact threshold τ which produces small negative reward of r^p leads to huge negative reward of r^l. However these fluctuations fell within a specific positive interval since the 200th episode. Thus, training was discontinued following this event.

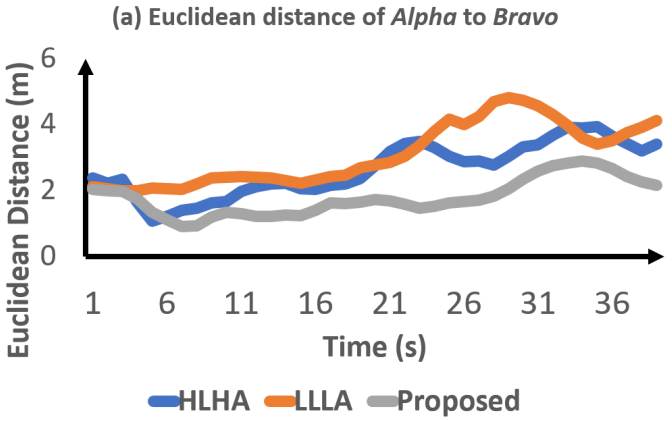

Fig. 7: Tracking performance of baseline vs proposed method (Euclidean Distance)

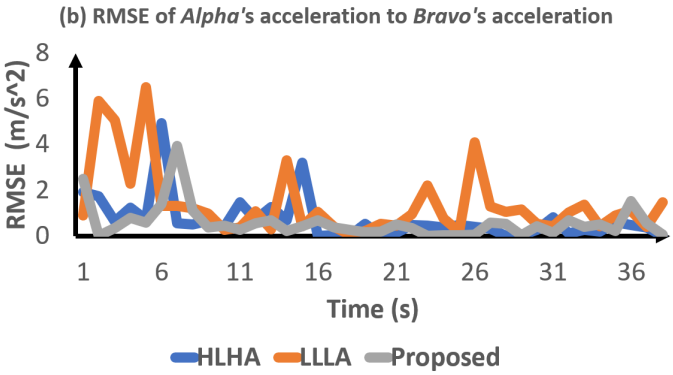

Fig. 8: Tracking performance of baseline vs proposed method (Acceleration RMSE)

C. Experimental Results

Our method has been compared against the baseline approaches mentioned in II-A. The tracking capability of $Alpha$ is tested using the baseline HLHA, LLLA, and proposed approach. Figure 7 shows the Euclidean distance of $Alpha$ to $Bravo$ during the flight.($Alpha$ was guided to maintain 1 meter distance from $Bravo$ along X axis for safe flight). The proposed method is observed to outperform both the

HLHA and LLLA methodologies. In spite of the fact that LLLA can provide faster state estimations, it has a higher rate of false negatives than other networks. The false negatives made it difficult for $Alpha$ to estimate the position of $Bravo$ and follow it. Hence, the performance of the LLLA perception system was poorer than those of the other two methods. On the other hand, the HLHA network is capable of providing accurate state estimation. Nevertheless, it is not sufficiently fast. The result of this was that when $Bravo$ performed an agile maneuver, $Alpha$ was unable to identify it quickly in order to adjust its navigation immediately. The proposed method, on the other hand, has the ability to adapt to different situations so that it is effective in tracking $Bravo$ and following it. The figure 8 illustrates the RMSE of $Alpha$'s accelerations to $Bravo$'s acceleration. The plot shows how fast $Alpha$ can adjust its course in response to changes in $Bravo$'s navigation. It can be

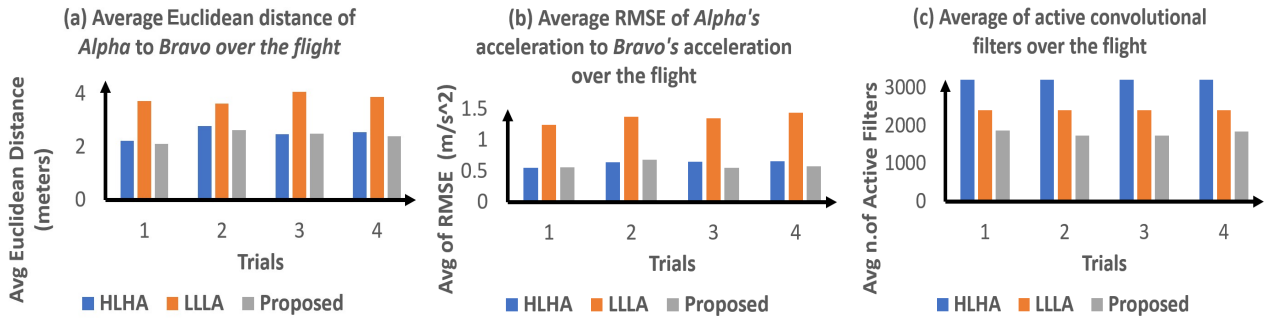

Fig. 9: Tracking performance of baseline and proposed methods for various trials

observed that the proposed system has the lowest RMSE compared to the other two approaches. Since the proposed system adapts to the situation and acts quickly or more confidently depending on the situation, so it has the lowest RMSE. On the other hand, HLHA had difficulties reacting quickly, and LLLA had difficulties estimating state accurately.So from the results it is evident that the proposed method is able to outperform both HLHA and LLLA.

In order to test the repeatability of our approach, we conducted four trials, figure 9 illustrates these results. Figure 9a represents the average Euclidean distance, 9b represents the average RMSE of acceleration, and 9c represents the average number of active convolutional filters used over the flight. According to the plot, the LLLA method performed worse than the other two methods. Its Euclidean distance and acceleration RMSE larger than the both of other models. The proposed method and HLHA have almost similar performance, with the proposed method having a slightly better performance than HLHA method. In addition, the proposed method performs better than HLHA with only using around fifty six percent of active filters compared to HLHA's active filters.

IV. Conclusion

Our paper proposes a novel perception controller using a state-of-the-art reinforcement learning technique known as Twin Delayed DDPG (TD3). This controller dynamically select the required latency-accuracy configuration for the given scenario by turning on / off the filters of the CNN. The proposed system was tested on a UAS-UAS tracking scenario. It is observed that the proposed perception system outperformed the baseline HLHA, and LLLA approaches with a less deviation in trajectory tracking with less convolutional computations. Future work concerns with the energy advantage of the proposed system.

References

[1] D. Canedo, P. Fonseca, P. Georgieva, and A. J. Neves, "A deep learning-based dirt detection computer vision system for floor-cleaning robots with improved data collection," *Technologies*, vol. 9, no. 4, p. 94, 2021.

[2] C. Nuzzi, S. Pasinetti, M. Lancini, F. Docchio, and G. Sansoni, "Deep learning based machine vision: first steps towards a hand gesture recognition set up for collaborative robots," in *2018 Workshop on Metrology for Industry 4.0 and IoT*. IEEE, 2018, pp. 28–33.

[3] D. Kim, H. Ryu, J. Yonchorhor, and D. H. Shim, "A deep-learning-aided automatic vision-based control approach for autonomous drone racing in game of drones competition," in *NeurIPS 2019 Competition and Demonstration Track*. PMLR, 2020, pp. 37–46.

[4] K. Muvva, J. M. Bradley, M. Wolf, and T. Johnson, "Assuring learning-enabled components in small unmanned aircraft systems," in *AIAA Scitech 2021 Forum*, 2021, p. 0994.

[5] V. V. R. M. K. Muvva, G. Li, and M. Wolf, "Autonomous uav landing on a moving uav using machine learning," in *AIAA SCITECH 2022 Forum*, 2022, p. 0789.

[6] V. V. R. M. K. Muvva, K. Samal, J. M. Bradley, and M. Wolf, "A closed loop perception subsystem for small unmanned aerial systems," in *AIAA SCITECH 2023 Forum*, 2023, p. 2673.

[7] M. Jubran, A. Abbas, A. Chadha, and Y. Andreopoulos, "Rate-accuracy trade-off in video classification with deep convolutional neural networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 1, pp. 145–154, 2018.

[8] X. Zhao, M. Hosseinzadeh, N. Hudson, H. Khamfroush, and D. E. Lucani, "Improving the accuracy-latency trade-off of edge-cloud computation offloading for deep learning services," in *2020 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2020, pp. 1–6.

[9] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[10] J. Terven and D. Cordova-Esparza, "A comprehensive review of yolo: From yolov1 to yolov8 and beyond," *arXiv preprint arXiv:2304.00501*, 2023.

[11] Y. V. Pant, H. Abbas, K. Mohta, T. X. Nghiem, J. Devietti, and R. Mangharam, "Co-design of anytime computation and robust control," in *2015 IEEE Real-Time Systems Symposium*. IEEE, 2015, pp. 43–52.

[12] K. Samal, T. Walton, H. Tran, and M. Wolf, "A markovian error model for dnns in perception-driven control systems," in *2023 International Joint Conference on Neural Networks (IJCNN)*. Under Review, 2023, pp. 1–8.

[13] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[14] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*. Springer, 2018, pp. 621–635.

[15] M. Zhang, H. Qin, M. Lan, J. Lin, S. Wang, K. Liu, F. Lin, and B. M. Chen, "A high fidelity simulator for a quadrotor uav using ros and gazebo," in *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2015, pp. 002 846–002 851.

[16] P. Adarsh, P. Rathi, and M. Kumar, "Yolo v3-tiny: Object detection and recognition using one stage improved model," in *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*. IEEE, 2020, pp. 687–694.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[18] E-Linux, "Kernel timer systems," *https://elinux.org/Kernel_Timer_Systems*, Accessed Jan 18th, 2024.

[19] C, "Resoruce usage," *https://www.gnu.org/software/libc/manual/html_node/Resource-Usage.html*, Accessed Dec 22nd, 2023.

[20] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv preprint arXiv:1712.01312*, 2017.