Collision Prevention in Quadrotor using Deep Reinforcement Learning

Rishabh Goel, Navaid Z. Rizvi and Vimlesh K. Ray

***Abstract*—Autonomous indoor quadrotor flight requires the control system to keep safe distance from nearby objects and take evasive action should the need arise. There has been the use of SLAM algorithms for mapping the environment and taking actions. However, these algorithms tend to be highly computationally expensive, which is not possible on an embedded hardware. Far from the capability of the embedded hardware, it is simply not necessary to create a map of the surroundings, as long as the quadrotor may be able to fly without collision and complete its mission objective. This is why a simple, low-cost sensor based system is sufficient, which is capable of running on the embedded hardware. We propose a simple sensor layout with a Neural Network based controller which is not computationally demanding. This work has been inspired by the other such implementations of sensor based systems or neural network based controllers, not necessarily together. Our approach requires the use of rangefinders and a control policy trained using Reinforcement Learning, implemented in simulation and tested experimentally. Our results show the possibility of such a controller with sufficient learning.**

***Index Terms*—Unmanned Aerial Vehicle, Collision Prevention, Deep Reinforcement Learning**

I. Introduction

Quadrotor, or quadcopter, is a 6 DOF under-actuated system. Agility and Compactness make it a serious contender for a diverse field of tasks, be it for indoor or outdoor flight. Indoor flights possess an inherent risk of collision with the nearby walls. The aim is to develop a controller which is able to prevent collision solely on the sensor inputs. A Reinforcement Learning(RL) based controller has been proposed for the quadrotor embedded with rangefinders and trained in a simulation environment. We faced various challenges in developing such a system and these will be highlighted so that future work that may originate from this research has a potential to seeing this goal through.

II. Literature Review

The literature review in this section consists of papers either on collision/obstacle avoidance or works using RL in their control loop.

A fully autonomous quadrotor implementation is presented in [1]. Their approach included the use of SLAM algorithm for complete autonomous flight. The issue with SLAM is the computational requirement, which is not feasible on embedded hardware. [2] used a modified PID controller for obstacle avoidance with the help of low cost sensors.

[3] used RL for path-planning and MPC for control. [4] used imitation learning for trajectory-tracking and obstacle avoidance.

[5] used DRL controller for control even under harsh conditions, for example stabilization from upside down initialization. [6] designed a DRL controller capable of flying through a narrow gap, big enough to fit the quadrotor. [7] concentrated on flying under external disturbance. [8]–[10] applied this DRL for trajectory-tracking. In [11], ANN was used to learn an adaptive control policy with PID.

Attitude control performance for three RL algorithms(DDPG, TRPO and PPO) are presented in [12] with PID as baseline. PPO demonstrated better performance among the three algorithms. While all other research consisted of model-free RL algorithms, [13] used model-based algorithm for low-level control.

In [14], an obstacle avoidance framework was proposed. They predefined a set of actions which the agent was responsible to choose from according to its obstacle avoidance requirement. Their use of a predefined action set rather than training a control policy over the whole action space, means the performance is only limited to environments similar to their own.

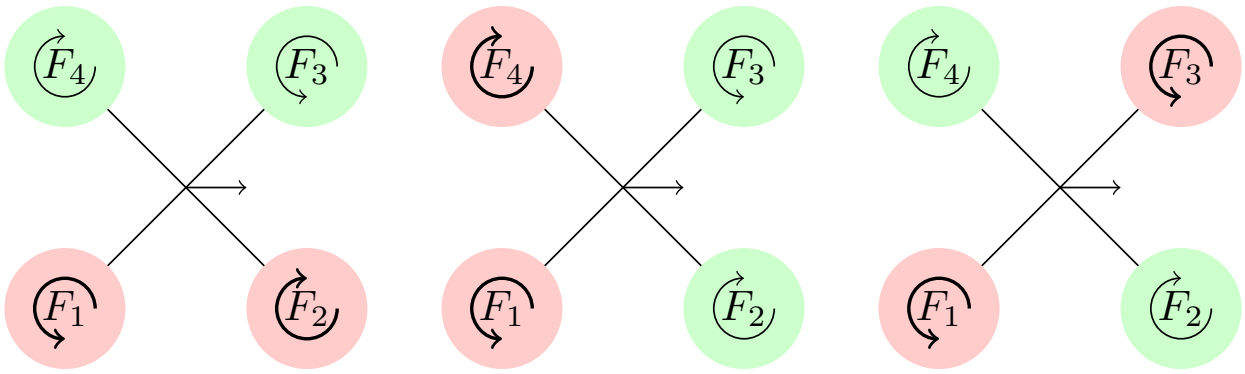
Linear and Non-linear controllers are capable of adequate performance in stabilizing a quadrotor. However, it is possible to augment the performance of the traditional controllers with the help of a supplementary controller, as demonstrated in [15]. Their method involved training a DRL based controller that works in tandem with a 'basic' controller to optimize the performance index. Their eventual controller consisted of cascade PD controller and RL based controller. Simulation results for experiments with wind disturbance showcase the better performance as compared to dynamic surface control method.

III. Theoretical Overview

A. Quadrotor Dynamics

A quadrotor is a non-linear, underactuated, vertical take-off and landing(VTOL) system. It consists of four propellers placed in a '$+$' or '$\times$' configuration and has 6 degrees of freedom. Two rotors spin in clockwise and two in counter-clockwise. The equations of motion [16] are given as follows:

All authors are affiliated to: Department of Computer Science, Gautam Buddha University
Correspondence: goel.r@outlook.com



(a) Roll (b) Pitch (c) Yaw

Fig. 1: Varying roll, pitch and yaw in a quadrotor through different motor combinations

$$\begin{aligned}\dot{\zeta} &= \nu \\ m\dot{\nu} &= RF_b \\ \dot{R} &= R\hat{\omega} \\ J\dot{\omega} &= -\omega \times J\omega + \tau_a\end{aligned}$$

B. Reinforcement Learning

The following theory has been a direct result of [17], [18], [19] and [20].

1) MDP: All environments in RL are defined as a Markov Decision Process. A finite MDP is formally defined by the tuple $(\mathcal{S}, \mathcal{A}, P^a_{ss'}, r, \rho_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P^a_{ss'} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the state transition probability, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the expected reward, ρ_0 is the distribution of initial state and γ is the discount factor. The agent chooses an action according to the policy $\pi(a|s;\theta)$[1].

State value function, $V^\pi(s)$, estimates the expected reward from current state and the policy is followed henceforth, is given as:

$$V^\pi(s) = \mathbb{E}\left[\sum_{l\geq 0} \gamma^l r_{t+1} \,\middle|\, s_t = s, \pi\right] \tag{1}$$

and the action value function, which estimates the expected reward from current state, taking action 'a' and following the policy thereafter, is:

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{l\geq 0} \gamma^l r_{t+1} \,\middle|\, s_t = s, a_t = a, \pi\right] \tag{2}$$

2) DDPG: DDPG in an *off*-policy actor-critic algorithm. It consists of four neural networks: *actor*, *critic*, *target actor* and *target critic*. The *actor* and *critic* networks are updated using gradient descent, where the critic network is updated by minimising the loss:

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathbb{E}}\left[\left(Q_\phi(s,a) - \underbrace{y(r,s',d)}_{target}\right)^2\right] \tag{3}$$

where *target* is calculated with the following formula:

$$r + \gamma Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

and the actor is updated using:

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathbb{E}}[Q^\pi(s, \mu_\theta(s))] \tag{4}$$

Target network weights are updated as follows,

$$\phi_{\text{targ}} \leftarrow \tau\phi_{\text{targ}} + (1-\tau)\phi \tag{5}$$

$$\theta_{\text{targ}} \leftarrow \tau\theta_{\text{targ}} + (1-\tau)\theta \tag{6}$$

τ regulates the update step.

IV. Setup and Training

A. Environment setup

The quadrotor and the training environment were created in MuJoCo [21] and simulations were performed with dm_control [22] as interface between Python and MuJoCo. Fig 2 specifies the training environment. The system of units in mujoco is undefined [23], therefore, MKS unit system will be followed for the rest of the paper. Thus, the environment is a 9m×9m×6m cuboid.

Next, a quadrotor model was developed with rangefinder sensors attached at various locations. Infrared and Ultrasonic sensors are two low-cost implementations as rangefinder. However, they do tend to be noisy when the object is far from the sensor. Consequently, the rangefinders in our model have a cut-off at 3m, i.e. all objects further than 3m would still show a sensor reading of 3m. While the quadrotor model is very basic, the sensor layout needed some deliberation. Two criteria had to be fulfilled by any proposed sensor layout. First, the sensor count should be as low as possible to prevent the system from

[1]The policy here is represented with π, which is generally used to represent stochastic policy. However, in this case it is only used to follow the accepted notation of representation for a policy.. The policy in this study is deterministic, written as μ.

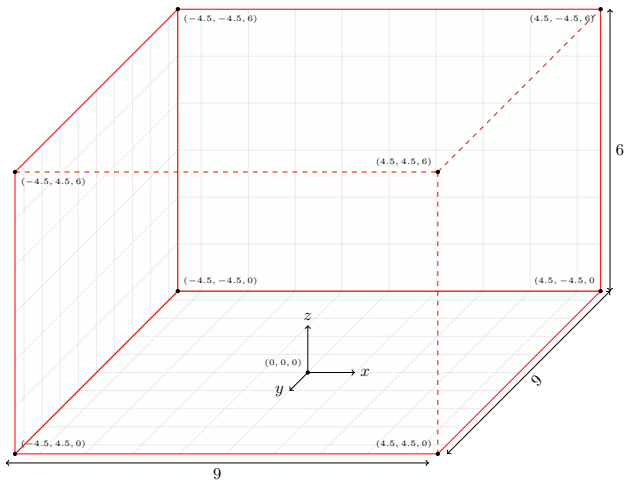


Fig. 2: Mujoco Environment specification

being expensive. Second, the supposed layout should give maximum coverage.

Two proposed layout have been shown in Fig. 3. Even though second layout(Fig. 3b) uses fewer sensors, the first layout(fig. 3a) was adopted for the final model due to better coverage. Fig. 3c and 3d show the coverage of both layouts nears walls.

Next we define state space, action space and reward function as per our MDP. State space(s) is composed of: orientation(represented using rotation matrix R), Inertial Measurement Unit(IMU)(linear acceleration α and angular velocity ω) and rangefinders(d_r). The action space(a) consists of four values which represent the rotor commands.

$$s = \{R, \alpha, \omega, d_r\} \qquad a = \{a_1, a_2, a_3, a_4\}$$

Reward function plays a significant role in accelerating learning. Consequently, our reward function is composed of three elements: penalty for orientation, reward/penalty for proximity to walls and penalty for collision.

Let $\{\angle_r, \angle_p\}$ be the angles along the roll and pitch axes, then the agent will be penalised according to euclidean norm. Therefore, if $\angle_r$ and $\angle_p$ are anything other than 0, then the agent will be penalized. Yaw is not considered in this formulation and its consequences will be evident later on.

$$\angle = \{\angle_r, \angle_p\}$$
$$Rew(\angle) = -\|\angle\|_2 \tag{7}$$

Reward formulation for rangefinders is more straightforward. First, all sensor readings are divided by 3, so as to normalize the range in [0,1]. Then, if any sensor reading is less than $\frac{1}{3}$(≈0.34), then the reading is substituted with -14. This is to present a net negative reward in case single sensor has distance of less than 1m from the nearby wall and rest all sensors give a reading of 3m.

$$d_r = \{d_{r_1}, d_{r_2}, \dots d_{r_{14}}\}$$
$$Rew(d_r) = \sum_{1}^{14} d_r \tag{8}$$

Collision in dm_control is accessed using *ncon* attribute in simulation data field. Therefore, if *ncon* is anything other than 0, the agent is heavily penalised and the episode in terminated.

$$Rew(ncon) = \begin{cases} 0 & ncon = 0 \\ -5000 & ncon > 0 \end{cases} \tag{9}$$

Reward at each timestep is obtained by combining (7), (8) and (9):

$$r_t = Rew(\angle) + Rew(d_r) + Rew(ncon) \tag{10}$$

The actor network architecture is shown in fig. 4 and the critic network as shown in fig. 5. Hidden layer nodes in both networks use tanh activation function. Moreover, in the actor network, output O_i is clipped to be within the range [0,1]. The critic network contains 4 additional nodes in input layer. *Target actor* and *Target critic* follow the same architectures.

B. Training setup

Tensorflow-macos, *tensorflow-metal*(enables GPU training on Apple M1) and *keras* were used for NN building and training. Training hardware was Apple M1 with 7-core GPU. Parallelism was not adopted during training. Table I lists the parameters used during agent training.

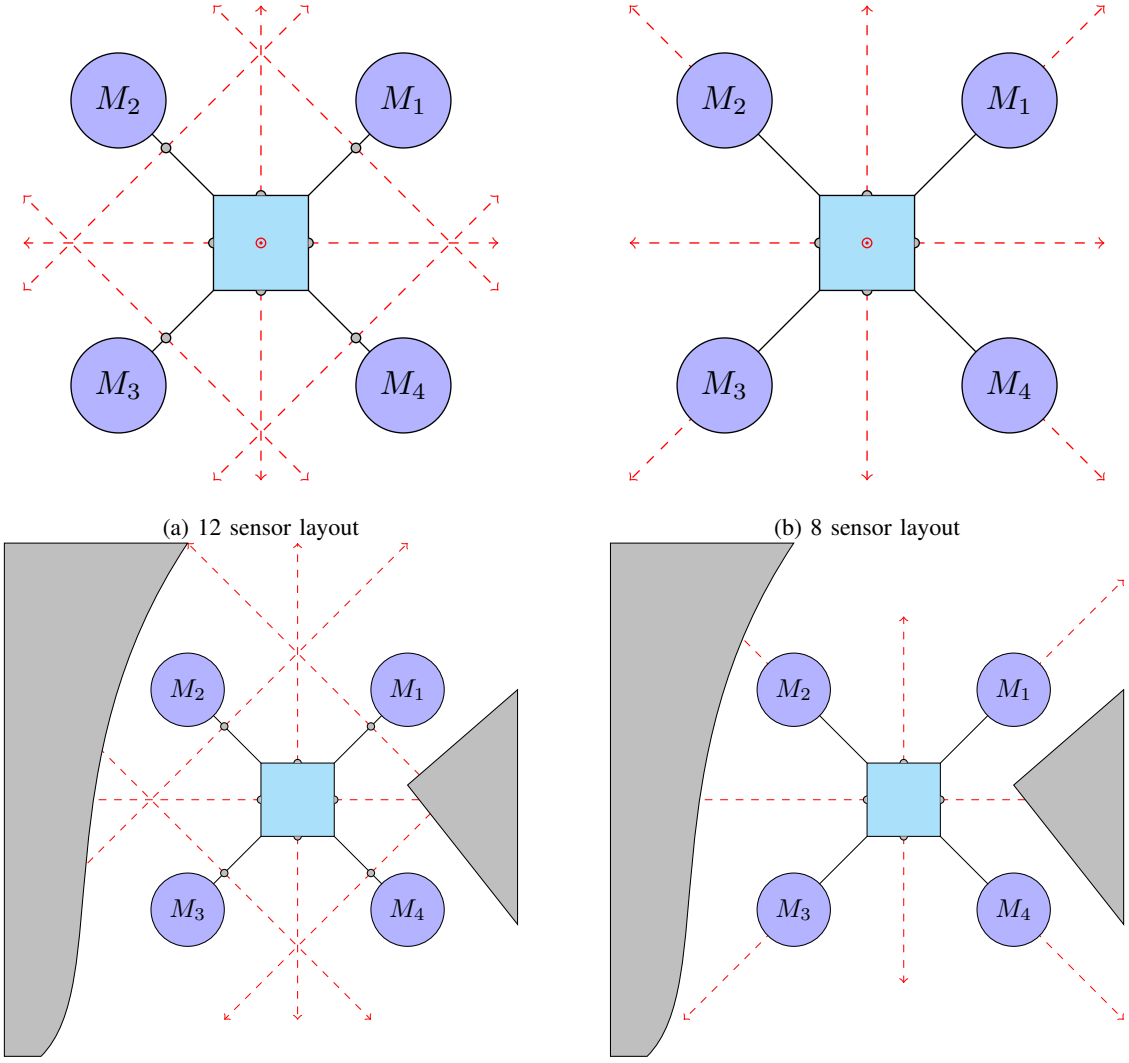
TABLE I: Training Parameters

Parameter	Value
Buffer size	50000
Batch size	128
critic learn rate	0.002
actor learn rate	0.001
Hidden layer nodes	64
Activation Function	*tanh*
γ	0.99
τ	0.995
state space	dim(29)
action space	dim(4)

TABLE II: Random initialization during training

Parameter	Random function
Position x	$\mathcal{N}(0.0, 2.0)$
Position y	$\mathcal{N}(0.0, 2.0)$
Position z	$\mathcal{U}(2, 4)$
Linear velocity	$\mathcal{N}(0.0, 1.0)$
Angular velocity	$\mathcal{N}(0.0, 1.0)$

Quadrotor was initialised randomly in the 3D space. Table II lists the distributions used for initializations, where $\mathcal{N}(\mu, \sigma)$ is Normal distribution with mean μ and standard deviation σ and $\mathcal{U}(a, b)$ denotes sampling from uniform distribution with lower range a and upper range b(inclusive). The length and width of the training environment is 9×9, and the distance from the centre to the nearest wall is 4.5m. $\sigma = 2$ implies the value in the range[-2, 2] is chosen with 68.26% probability and [-4,4] with 95.44%. Very rarely, it is also possible that the quadrotor is initialized outside the training environment. However,



(a) 12 sensor layout

(b) 8 sensor layout

(c) 12 sensor layout near wall corners

(d) 8 sensor layout near wall corners

Fig. 3: Potential sensor layouts for implementation

this is generally followed by a collision because of random linear and angular velocity.

The problem of exploration was addressed by adding some noise in the output of the actor network. Initially it was set to $\mathcal{N}(0.5, 0.3)$ for the first 10,000 episodes. However, from 10,000 episode onwards, the noise was changed to $\mathcal{N}(0.0, 0.3)$. It is hard to say if the noise had any effect on performance this early in training..

C. Testing setup

The initialization parameters for testing is shown in table III.

TABLE III: Random initialization during testing

Parameter	Random function
Position x	$\mathcal{N}(0.0, 0.5)$
Position y	$\mathcal{N}(0.0, 0.5)$
Position z	$\mathcal{U}(2, 4)$
Linear velocity	$\mathcal{N}(0.0, 0.5)$
Angular velocity	$\mathcal{N}(0.0, 0.5)$

V. Results and Discussion

A. Training

The first 10,000 episodes were in multiple sessions. This is mainly attributed to the under-powered machinery for such a gigantic task. Training for even 1000 episodes increased the processor temperature to very high levels. All four NN were saved and reloaded between each session. For the initial 8,000 episodes, the penalty for collision was very high at -10,000. This lead to the quadrotor learning that the best way to optimize rewards is by doing nothing and just free fall.

After 8,000 episodes, some different approach was required, thus he collision penalty was changed from -10,000 to -5,000. Previously, it wasn't even able to get an average reward over 1000(before collision) and now, after training for a mere 2000 episodes, it grossed an average reward of 1200(before collision).

B. Testing

Fig. 7a and fig. 7c showcase the performance of proposed controller over 10 test runs after training for 10,000

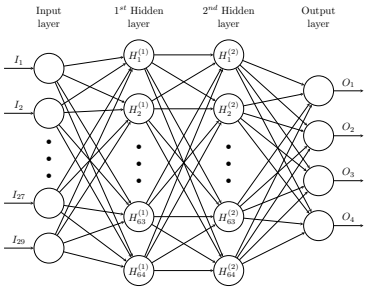


Fig. 4: Actor network architecture

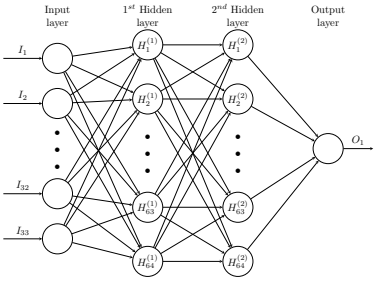


Fig. 5: Critic network architecture

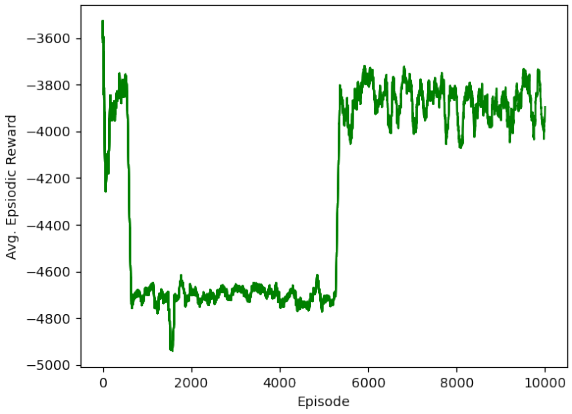


Fig. 6: Average reward during training from 10,000 episodes to 20,000 episodes

episodes and 20,000 episodes respectively. Fig. 7b shows the performance of random controller over 10 test runs. The actions were sampled from a Gaussian distribution denoted as $\mathcal{N}(0.5, 0.3)$. The mean of previously mentioned distribution is not zero, which has a valid reason. The control input range of each rotors is [0-1] and from manual testing in simulation, it was found that the quadrotor requires a value of 0.34 and above to create a positive thrust in the z-direction. Otherwise, the quadrotor starts to fall under gravity.

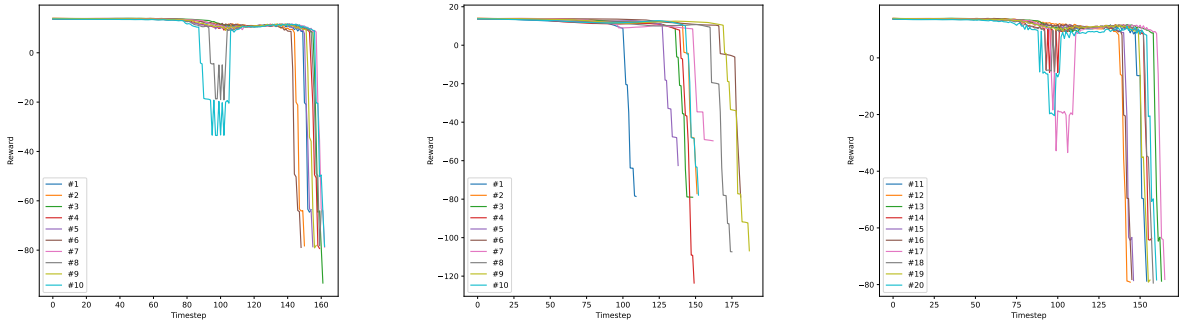
Further analysis of single testcase from each test run has been shown in fig. 8. The first test run from each set was chosen irrespective of its performance.

Testing results put forward the subpar performance of the agent. However, the only significance is that the agent maintained a steady roll and pitch angle until it came into close proximity of the walls. The agent tends to spin about its yaw axis uncontrollably. This hampers its control when it prevents close proximity from the walls. Currently, the agent is not penalised for high velocities which is a reason it takes extreme steps to avoid collision and is unable to stabilize itself thereafter. Even though the proposed system did not perform satisfactorily, it will be unwise to discard this method altogether. Training a control policy using RL requires training that generally ranges in millions of timesteps, while our system was not even trained for half million timesteps. Further training with greater exploration may yield a satisfactory performance, which will be focus of the authors.

C. Discussion

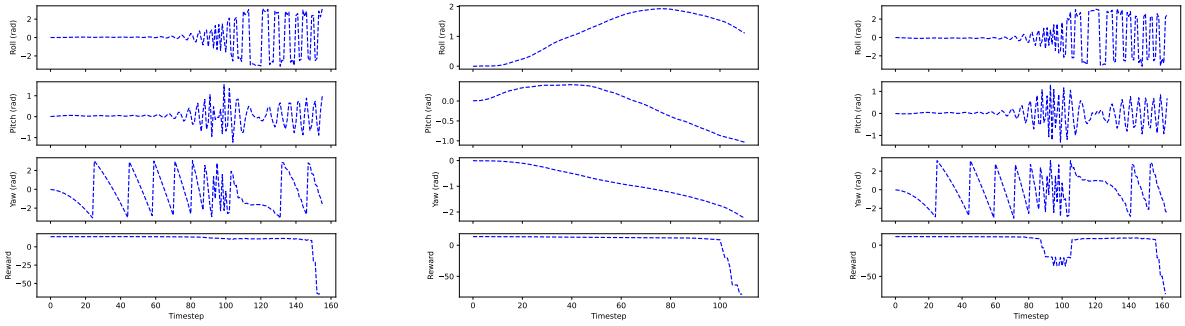
There were various factors for the insufficient performance in above graphs. These are:

1) The agent was trained for insufficient amount. Rather than performing an analysis after training for 10^7 timesteps, the graphs only show the performance for 20,000 episodes, which equates to $\approx 3 \times 10^5$ timesteps.
2) The reward function also has a role to play in this misfortune. In a complete oversight, the yaw angle was not included in the reward calculation. It was presumed that, the direction of facing bears no significance to the overall problem and thus the exclusion from the reward function. During the learning phase, however, the agent learned that it necessary to stabilize rotation about roll and pitch axes. This lead to the agent setting two rotors(on the same diagonal) to 1 and the other two rotors to 0. This action meant the quadrotor was able to stabilize the roll and pitch axes but induced an uncontrollable spin about the yaw axis.
3) It is prudent to say that DDPG is not the preferred choice for training the proposed system on an extremely underpowered machine. During training, the machine reached alarming temperatures and to prevent any damage, the system was trained in batches.
4) Initially it was thought that maybe there was an issue in the implementation of DDPG. However, after noticing the behaviour the agent exhibited with regards to using only two diagonal rotors for thrust, it was clear that the agent was learning just not what was expected.
5) Even though the results don't show this, there is still faith in the proposed system that it might work, albeit after training rigorously. The reward function(refer to eq. (10)) was finalised after a few iterations. The earliest iteration rewarded the agent for each timestep that it survived. This meant there was no weightage to the actual orientation. An upside-down configuration earned the same reward as a perfectly stable orientation.
6) One possible way to improve performance is to introduce a penalty on the basis of linear and angular velocities. If the agent is penalised for high velocities, then in theory it should be able to take precautionary actions to prevent such states. It was



(a) DRL controller after 10k (b) Random controller (c) DRL controller after 20k

Fig. 7: Performance over 10 test runs



(a) DRL controller after 10k (b) Random controller (c) DRL controller after 20k

Fig. 8: Analysis of single testcase for roll, pitch, yaw, and reward

decided that the quadrotor should be allowed to achieve high velocities as long as it is able to prevent collisions. This theory is still not proven.

VI. Conclusion

In this study, a Deep Reinforcement Learning based controller was proposed for a collision free autonomous quadrotor flight. Furthermore, two potential sensor layouts were presented with each having its pros and cons. One sensor on top and bottom proved sufficient as the quadrotor learned to remain stable for optimal performance. Furthermore, the reward function for roll and pitch showed positive result, while the exclusion of yaw proved to be the destabilizing factor in all flights. Potential inclusion of velocity based reward might be needed if the performance of the proposed system remains unsatisfactory even after training for at least 10 million timesteps.

Future work may include using PPO over DDPG, changing reward function to be more inclusive of other parameters like velocity,

References

[1] S. Grzonka, G. Grisetti, and W. Burgard, "A fully autonomous indoor quadrotor," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 90–100, 2012.

[2] N. Gageik, P. Benz, and S. Montenegro, "Obstacle detection and collision avoidance for a uav with complementary low-cost sensors," *IEEE Access*, vol. 3, pp. 599–609, 2015.

[3] C. Greatwood and A. G. Richards, "Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control," *Autonomous Robots*, vol. 43, no. 7, pp. 1681–1693, 2019.

[4] S. Stevšić, T. Nägeli, J. Alonso-Mora, and O. Hilliges, "Sample efficient learning of path following and obstacle avoidance behavior for quadrotors," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3852–3859, 2018.

[5] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[6] C. Xiao, P. Lu, and Q. He, "Flying through a narrow gap using end-to-end deep reinforcement learning augmented with curriculum learning and sim2real," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[7] C.-H. Pi, W.-Y. Ye, and S. Cheng, "Robust quadrotor control through reinforcement learning with disturbance compensation," *Applied Sciences*, vol. 11, no. 7, p. 3257, 2021.

[8] C.-H. Pi, K.-C. Hu, S. Cheng, and I.-C. Wu, "Low-level autonomous control and tracking of quadrotor using reinforcement learning," *Control Engineering Practice*, vol. 95, p. 104222, 2020.

[9] B. Rubí, B. Morcego, and R. Pérez, "Deep reinforcement learning for quadrotor path following with adaptive velocity," *Autonomous Robots*, vol. 45, no. 1, pp. 119–134, 2021.

[10] W. Lou and X. Guo, "Adaptive trajectory tracking control using reinforcement learning for quadrotor," *International Journal of Advanced Robotic Systems*, vol. 13, no. 1, p. 38, 2016.

[11] M. Jafari and H. Xu, "Intelligent control for unmanned aerial systems with system uncertainties and disturbances using artificial neural network," *Drones*, vol. 2, no. 3, 2018.

[12] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, feb 2019.

[13] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep

model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[14] J. Ou, X. Guo, M. Zhu, and W. Lou, "Autonomous quadrotor obstacle avoidance based on dueling double deep recurrent q-learning with monocular vision," *Neurocomputing*, vol. 441, pp. 300–310, 2021.

[15] X. Lin, Y. Yu, and C. Sun, "Supplementary reinforcement learning controller designed for quadrotor uavs," *IEEE Access*, vol. 7, pp. 26422–26431, 2019.

[16] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 5, pp. 4393–4398 Vol.5, 2004.

[17] "Deep deterministic policy gradient." https://spinningup.openai.com/en/latest/algorithms/ddpg.html, March 2018. Accessed: 18-May-2022.

[18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[19] H.-Y. Lin and X.-Z. Peng, "Autonomous quadrotor navigation with vision based obstacle avoidance and path planning," *IEEE Access*, vol. 9, pp. 102450–102459, 2021.

[20] 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.

[21] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033, IEEE, 2012.

[22] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa, "dm_control: Software and tasks for continuous control," *Software Impacts*, vol. 6, p. 100022, 2020.

[23] "Overview." https://mujoco.readthedocs.io/en/latest/overview.html#units-are-undefined, March 2022. Accessed: 23-May-2022.