ENPM 667 - Project 1 - Implementation of Journal Paper :

Low-level autonomous control and tracking of quadrotor using reinforcement learning

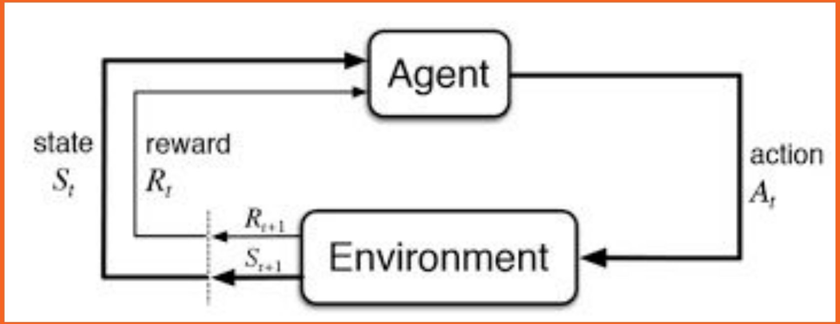
By:-

Anubhav Paras - 116905909
Preyash Rajeshkumar Parikh - 117303698

Contents

Overview of Project - Concept

`



The concept is about implementing a low level flight controller which uses Proximal Policy Optimization where we require the controller not to deviate vastly from the previous policy while aiming to achieve a stable policy. This has to be carried out while maximizing the expected return. Hence, this is a smooth learning process.

Background

- **UAV and Applications !**

  ⟶ Patrolling, Aerial monitoring and others

- **Issues with Traditional and Optimization Methods.**

  ⟶ Traditional methods are required to have parameter adjustments

  ⟶ In Optimization algorithms, quadrotor is subjected to external disturbance.

- **What we proposed ?**

  ⟶ Proximal Policy Optimization

Introduction

- **Flight control systems of quadrotors**

  → Outer Loop - Higher- Level Planner

  → Inner Loop - Lower Level Controller

- **Why reinforcement learning based strategy ?**

  → To have a no trial-and-error tuning of parameters.

- **Why Model Free methods ?**

  → Since domain knowledge is not required.

UNDERSTANDING

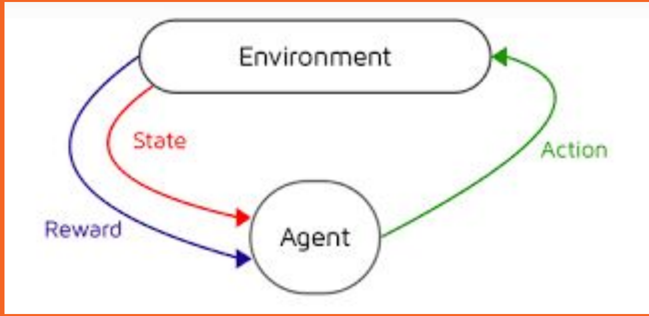
→ **Markov Decision Process**

→ **Policy Gradient**

→ **Proximal Policy Optimization**

Markov Decision Process

The Markov property states that, “ The future is independent of the past given the present.” Once we know the current state, the past information encountered may be thrown away, and that state is a sufficient statistic that gives us the same view of the future as if we have all the past information.

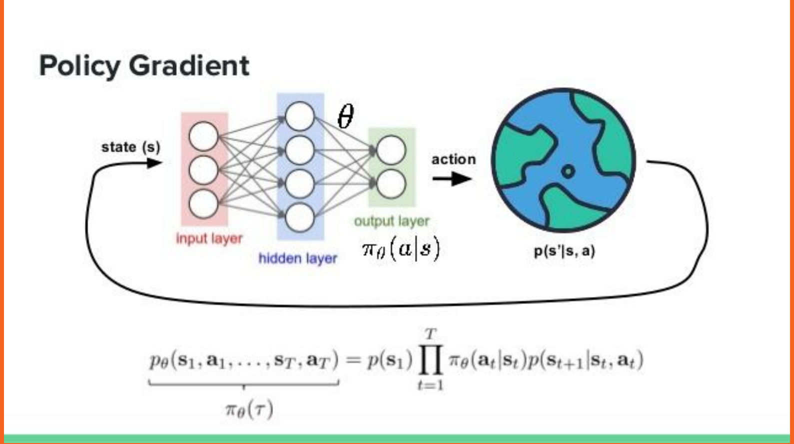


An MDP is a tuple (S, A, P, R, γ)

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

Policy Gradient

Policy gradient methods are a type of reinforcement learning techniques that rely upon optimizing parametrized policies with respect to the expected return.



$$\sum_{s \in S, a \in A} \rho^\pi(s) Q^\pi(s,a) \frac{\partial \pi(a|s)}{\partial \theta}$$

$$= \sum_{s \in S, a \in A} \rho^\pi(s) \left(V^\pi(s) + A^\pi(s,a)\right) \frac{\partial \pi(a|s)}{\partial \theta}$$

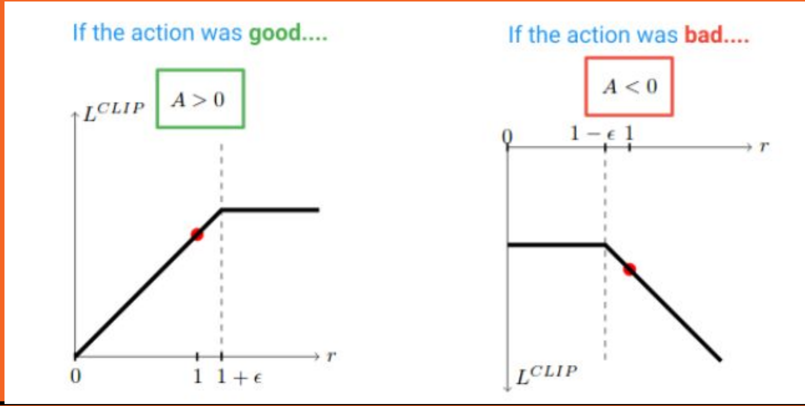
$$= \sum_{s \in S, a \in A} \rho^\pi(s) A^\pi(s,a) \frac{\partial \pi(a|s)}{\partial \theta},$$

Proximal Policy Optimization

→ **Problem with Policy Gradient** is that the training process is too slow and too much variability in training.

→ **PPO improves** the stability of the Actor training by limiting the policy update at each training step.

→ **Clipped Surrogate Function**



QUADROTOR DYNAMICS

→ **Takes gravity and the forces generated by motors.**

→ **Six- Degree rigid body with four motor thrust forces.**

→ **Torque generated is**

$$\tau = J\begin{bmatrix} \frac{1}{\sqrt{2}}L(-F_1 + F_2 + F_3 - F_4) \\ \frac{1}{\sqrt{2}}L(F_1 - F_2 + F_3 - F_4) \\ K_m L(-F_1 + F_2 - F_3 + F_4) \end{bmatrix}$$

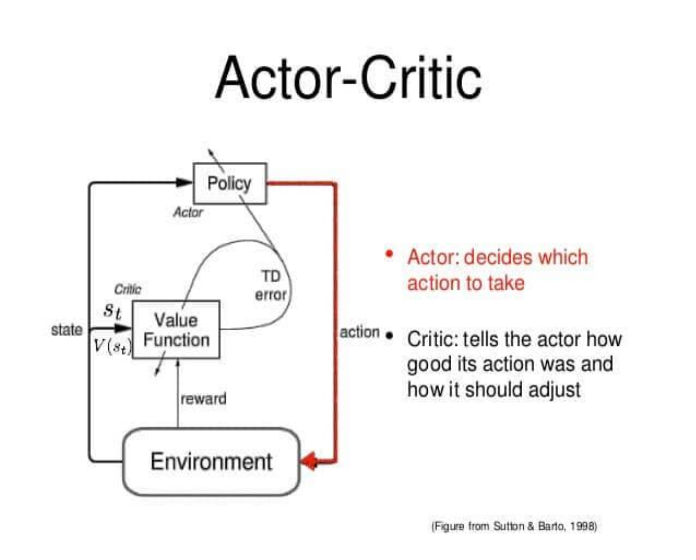
→ **Forced produce by motor spinning,** $F_i = K_f \omega_i^{\,2}, \quad i = 1, 2, 3, 4$

Where, ωi is the speed of the motor and

Km is coefficient of the generated torque.

ALGORITHM - TRAINING

→ Policy Gradient based Actor- Critic architecture.



Value: $V^{\pi}(s_t) = \mathbb{E}\left[r_t + \gamma V^{\pi}(s_{t+1}) \middle| a_t \sim \pi(a|s_t)\right]$

Actor: $\theta_\pi \leftarrow \theta_\pi + \alpha \frac{1}{|B|} \sum_{(s_t, a_t, r_t, s_{t+1}) \in B} \frac{Q^{\pi}(s_t, a_t)}{\pi(a_t|s_t;\theta_\pi)} \frac{\partial \pi(a|s;\theta_\pi)}{\partial \theta_\pi},$

Algorithm- Training

- The learning algorithm comprises of three main components:
    - Actor (Policy Function):
        - This is a simple neural network to approximate the parameterized policy.
    - Critic (Value Function):
        - Neural network to approximate the value function that returns the expected rewards given a state.
    - PPO (Proximal Policy Optimization) Agent: This is the heart of the algorithm that is responsible for maximizing the expected sum of rewards by imposing a constraint on the improved policies.
-

PPO Algorithm -

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Tasks to train

1) Hovering:
    a) The quadrotor is expected to hover over a specific target point: (0,0,10)
    b) For this task the position error (x,y,z) and the orientation error(roll, pitch, yaw) should be minimum.
2) Trajectory following:
    a) The quadrotor is expected to circle around a fixed point in the x-y plane.
    b) To achieve this it has to maintain a constant distance with a point (should be **within certain threshold** of the circle boundary) while also maintaining a constant desired velocity.

Algorithm- Training

For each EPISODE (E):

1) Collect a set of N (PPO steps) transitions (T) where a transition is defined by:
   - T[i] = <state_i, action_i, reward_i, curr_policy_distribution>
   i) Select an <action> from the **current policy** (actor)
   ii) Execute the <action> to get the <next_state> and <reward>
   iii) T[i] = <state, action, reward, curr_policy_distribution>
2) Calculate returns using generalized advantage algorithm and the advantage
   A = Q(s,a) - V(s): Advantage tells how good a particular action is as compared to an average action taken in some state.
3) Use this batch of transitions and the advantages to train the actor-critic network to **update the policy.**

ALGORITHM - TRAINING

Training the actor-critic network:

1) In PPO to train the actor need to optimize the loss function given by:

$$L^{CLIP}(Q) = \hat{E}_t[min(r_t(Q)\hat{A}_t, clip(r_t(Q), 1-\in, 1+\in)\hat{A}_t)]$$

2) Here we clip the ratio between the old and new policy by some factor, here (1-0.2=0.8) to (1+0.2=1.2).
3) For our implementation we modified the loss function as per the paper.

$$maximize\ L_{policy} = \sum_{(s,a)\in T} \min\left[\left(\frac{\pi(a|s)}{\mu(a|s)} - 1\right) A^{trace}, \epsilon|A^{trace}|\right]$$

$$minimize\ L_{value} = \frac{1}{|T|} \sum_{(s,a)\in T} \left(V(s) - V^{trace}\right)^2 .$$

ALGORITHM - TRAINING

Training the actor-critic network:

4) For each EPOCH while training:

- Sample the mini-batch from the batch of transitions (T)
- Feed forward the network to get action and policy distribution (from the actor), values (from the critic)
- Calculate the loss:
  - Surrogate loss for the actor = Adv * min [exp {log(new_distribution) - log(old_distribution)}, 1]
  - MSE loss for the Critic = (returns - V(s))^2
- Backpropagate the total loss through the network using SGD

5) Repeat till convergence (ie old Policy and new policy do not diverge much)

ALGORITHM - TRAINING: Rewards

HOVERING:

1) For the task of reaching a particular target and hovering over there:
   - reward = w1*(position error) + w2*(orientation error) + w3*(action)
2) If the quadrotor reached the defined boundary limits a PENALTY was imposed.
3) If the quadrotor reached within some threshold of the target position, a BONUS was given, where,
   bonus_reward = BONUS* (num_done) * reduction_factor
   where **num_done** = **number of times it reached within the threshold**

ALGORITHM - TRAINING: Rewards

TRAJECTORY FOLLOWING:

1) For the task of reaching a particular target and hovering over there:
   - reward = 1. w_1*(distance error) + w_2*(velocity error) + w_3*(action)
2) $distance_{err}$ = desired_radius - sqrt[(x-x_center)^2 - (y-y_center)^2]
3) $velocity_{err}$ = x*y_vel - y*x_vel - desired_radius*desired_vel
4) If the quadrotor reached the defined boundary limits a PENALTY was imposed.
5) If the quadrotor reached within some threshold of the target position, a BONUS was given, where,
   bonus_reward = BONUS* (num_done) * reduction_factor
   where **num_done = number of times it reached within the threshold**

ALGORITHM - TRAINING: Parameters

Quadrotor Params:

Mass	- 0.665kg
Arm Length	- 0.105m
I_{xx}	- 0.0023 kg-m^2
I_{yy}	- 0.0025 kg-m^2
I_{zz}	- 0.0037 kg-m^2 m

Network Params

Actor network	- 32x32x4
Critic network	-128x128x1
Discount factor (gamma)	- 0.99
Learning rate (alpha)	-1e-4
PPO Steps	-350
Mini-Batch size	-64
Training epochs	-10/20

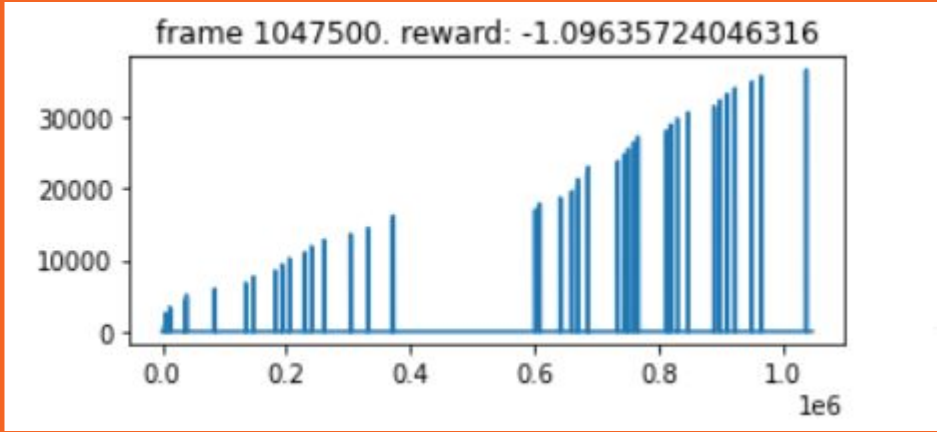
Input = state of the quadrotor = <position, angles, velocity, angular velocity>

Input state vector size = 1x12

RESULTS - TRAINING

Transition vs Rewards:

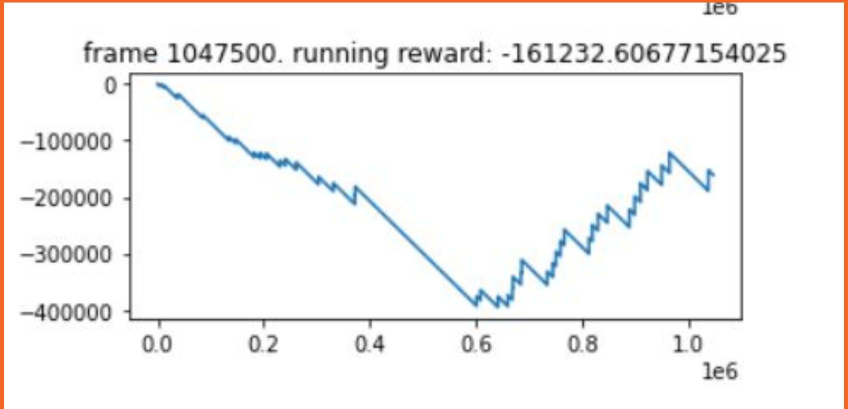
- We can observe that the rewards that the agent received were more In number as the training progressed.



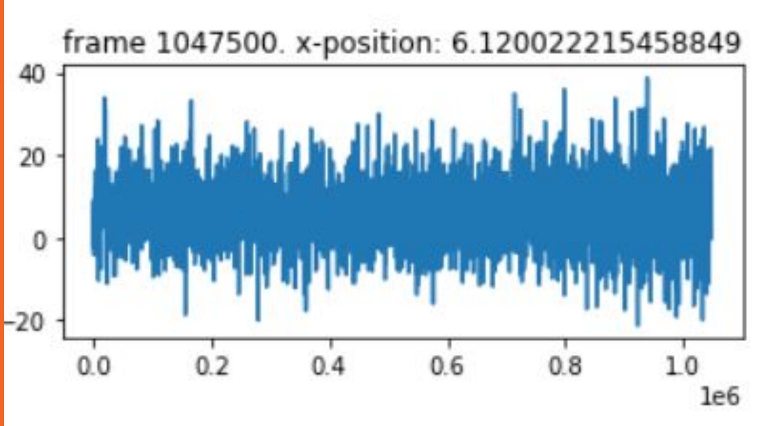
RESULTS - TRAINING

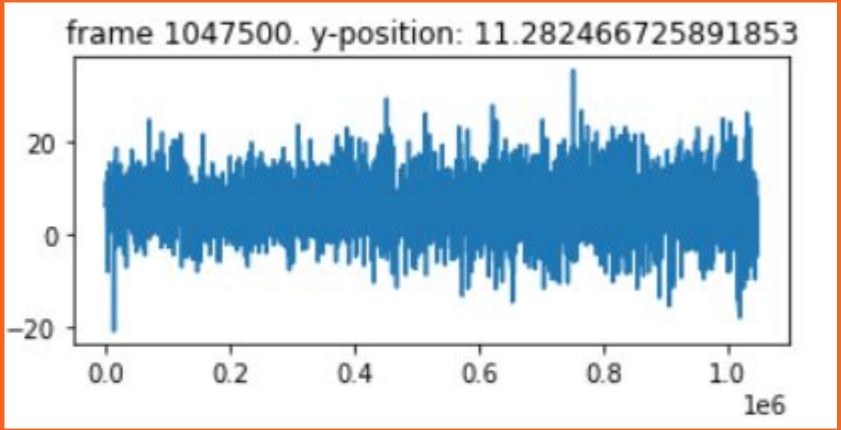
Transition vs Running Rewards:

- We observe that the total rewards started increasing as the training progressed.
- This can be attributed to the fact that more bonus rewards were obtained towards the end of training.



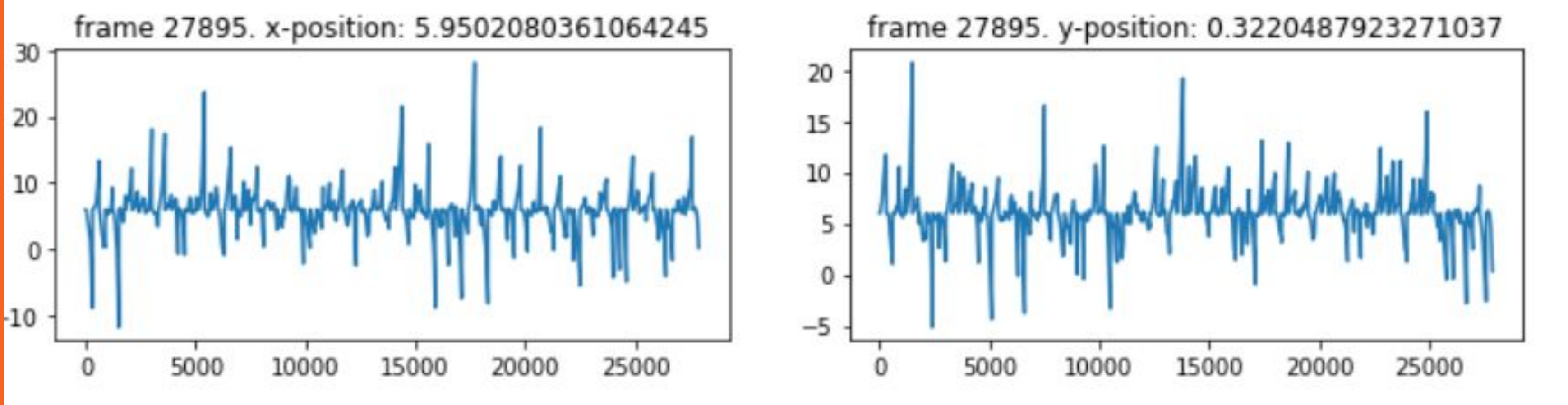
Results - Hovering Task





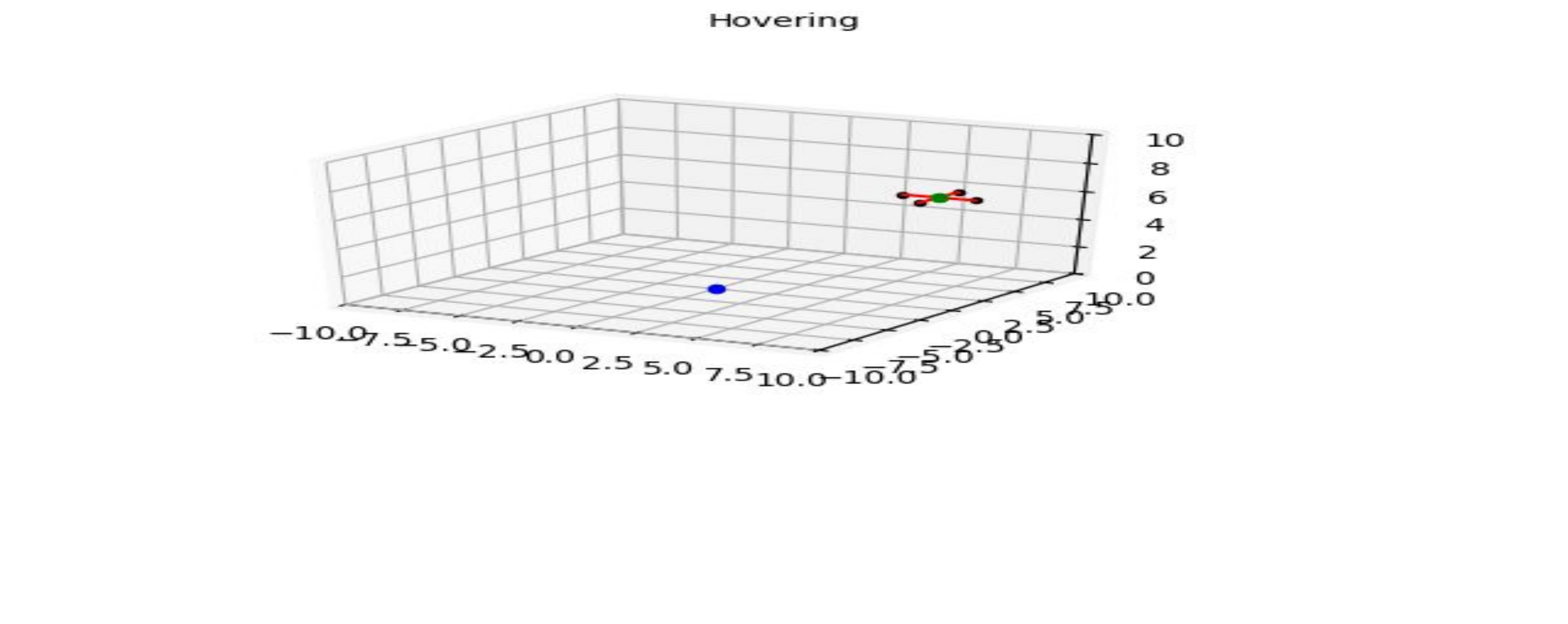
X and Y position were oscillating around 0 during training.

Results - Hovering Task



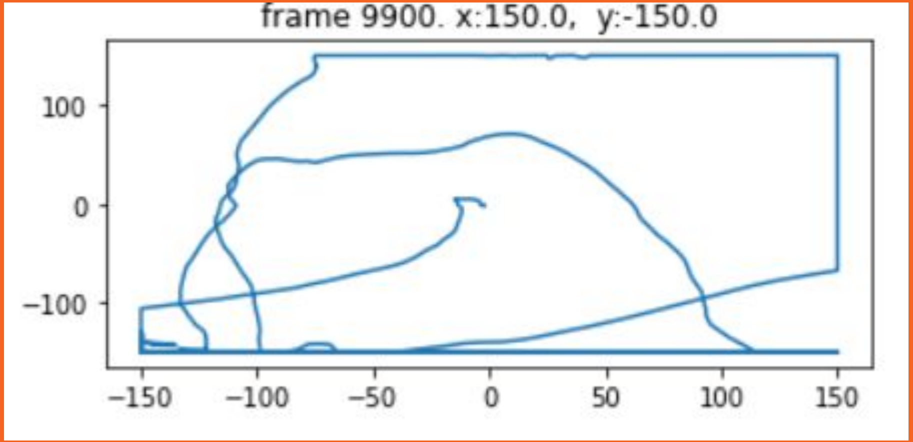
X and Y position oscillating around 0 during testing before converging.

Results - Hovering Task



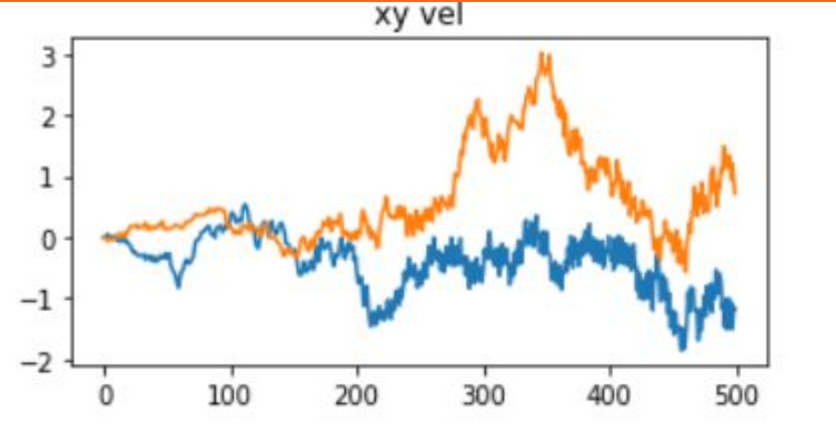
Results - Trajectory Following Task

The quadrotor encircle around the center of the circle.

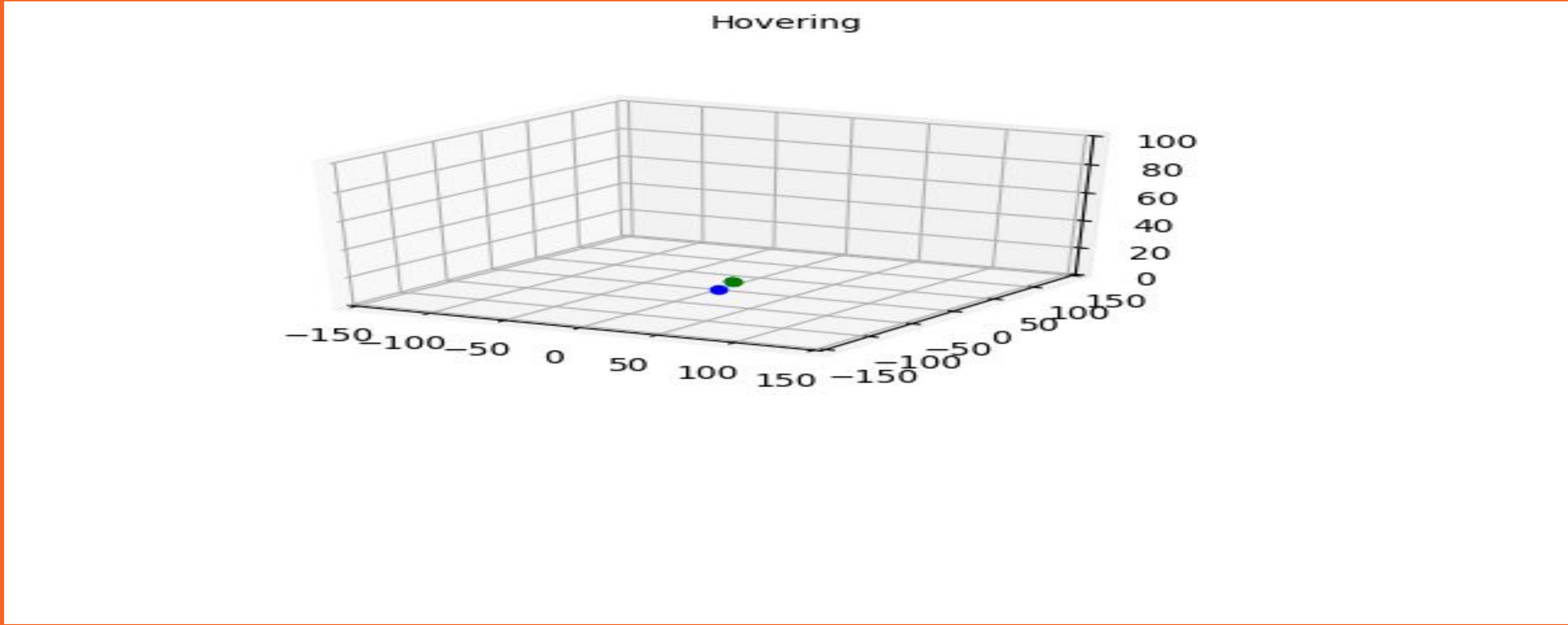


Results - Trajectory Following Task

X and Y component of the of the velocity were moving in a sinusoidal motion so as to maintain a constant desired velocity to move around in a circle.



Results - Trajectory Following Task



Results

1) For the hovering task training was done for ~1M transitions (10K episodes of 350 PPO steps each).
2) For the trajectory following task training was done for ~0.3M transitions (1K episodes of 300 PPO steps each)
3) Quadrotor was made to reach a point (0,0,0) and stay over there.
4) It was following a near-to-circular trajectory.

Conclusion

- We tried to adhere to the functions, parameters and the algorithm as much as we can.
- The task results were satisfactory but can be improvised:
  - with more training episodes,
  - using multiple actors to provide more sample data at a time, i.e. using more diverse experience to improve the policy
  - modification to the loss function (calculating advantages using GAE - generalized advantage estimation),
  - tuning of few hyperparameters
- The training process took a lot of time after the implementation.
- To make it learn properly we had to run around 1 to 1.5M (~10M recommended in the paper) iterations and repeated that with each major or minor change in the code or the parameters to get better results.
-

References

- 1) https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690
  2) https://openai.com/blog/openai-baselines-ppo/
  3) https://jonathan-hui.medium.com/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12
  4) https://stackoverflow.com/questions/46422845/what-is-the-way-to-understand-proximal-policy-optimization-algorithm-in-rl
  5) https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f
  6) https://www.cse.unsw.edu.au/~cs9417ml/RL1/tdlearning.html

Deliverables

→ Hover Code -
https://colab.research.google.com/drive/15vZwsICSzSOU_38TRTV2KuEi7A5ZIt25?authuser=3#scrollTo=fCINsLY6DFvl

→Trajectory Code -
https://colab.research.google.com/drive/1988BaVXe4V81RoQO3oeOvO7hvUTCN6BS?authuser=3#scrollTo=kmx_8l29LWJK

→ Code - https://github.com/anubhavparas/quadrotor_control_ppo

→ Presentation

→ Report

→ Slides

→ Simulator GIF

→ Readme