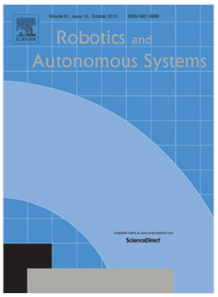
Journal Pre-proof



An innovative bio-inspired flight controller for quad-rotor drones: Quad-rotor drone learning to fly using reinforcement learning

Amir Ramezani Dooraki, Deok-Jin Lee

PII: S0921-8890(20)30511-X
DOI: https://doi.org/10.1016/j.robot.2020.103671
Reference: ROBOT 103671

To appear in: *Robotics and Autonomous Systems*

Received date : 8 October 2019
Revised date : 30 June 2020
Accepted date : 16 October 2020

Please cite this article as: A.R. Dooraki and D.-J. Lee, An innovative bio-inspired flight controller for quad-rotor drones: Quad-rotor drone learning to fly using reinforcement learning, *Robotics and Autonomous Systems* (2020), doi: https://doi.org/10.1016/j.robot.2020.103671.

An Innovative Bio-inspired Flight Controller for Quad-rotor Drones: Quad-rotor Drone Learning to Fly Using Reinforcement Learning

Amir Ramezani Dooraki[a,*], Deok-Jin Lee[a,*]

[a]*Smart Autonomous System Lab, Department of Mechanical Engineering, Kunsan National University, Korea*

Abstract

Animals learn to master their capabilities by trial and error, and with out having any knowledge about their dynamics model and mathematical or physical rules. They use their maximum capabilities in an optimized way. This is the result of millions of years of evolution where the best of different possibilities are kept, and makes us rethink *How does the nature perform things?*, particularly when natural systems outperform our rigid systems.

In this study, inspired by the nature, we developed an innovative algorithm by enhancing an existing reinforcement learning algorithm (proximal policy optimization (PPO)). Our algorithm is capable of learning to control a quad-rotor drone in order to fly. This new algorithm called Bio-inspired Flight Controller (BFC) does not use any conventional controller such as PID or MPC to control the quad-rotor drone. The goal of BFC is to completely replace the conventional controller with a controller that acts in a similar way to the animals where they learn to control their movements. It is capable of stabilizing a quad-copter in a desired point, and following way points. We implemented our algorithm in an AscTec Hummingbird quad-copter simulated in Gazebo, and tested it using different scenarios to fully measure its capabilities.

Keywords: Reinforcement Learning, Autonomous System, Bio-inspired Artificial Intelligence, Policy Optimization, Artificial Neural Network,

*Corresponding author

Email addresses: a.ramezani.dooraki@gmail.com (Amir Ramezani Dooraki), deokjlee@kunsan.ac.kr (Deok-Jin Lee)

Bio-inspired Controller, Machine Learning

1. Introduction

The design and creation of robots that behave in an intelligent and autonomous way are **well known lines** of research in artificial intelligence (AI) and to be specific in machine learning (ML). Among several methods used in order to reach these goals are methods inspired by living organisms or bio-inspired artificial intelligence [Floreano and Mattiussi (2008)] (e.g. artificial neural network (ANN)). At the same time, there are countless examples of robots that can move or act in an intelligent way, but they are neither autonomous nor capable of learning to perform an intelligent act or behavior—they are merely pre-defined rigid systems that are designed and implemented to perform a very specific task.

Learning phenomena in the nature, generally speaking, can be seen to happen specifically in two ways among other possible ways, which are complementary and have some familiar counterparts in machine learning community: learning by imitation (e.g. imitation learning) and learning by experience (e.g. Reinforcement Learning [Sutton and Barto (1998)])—which are discussed more in detail in [Dooraki (2018)].

By implementing *learning by experience* in machine we can shape a system that learns from its own environment. Such a system learns by performing a sequence of interactions with its environment or in other words by forming a perception from its environment and acting according to that perception in every moment of time (time step). This system can also be called a behavioral system and can be used to create an *autonomous robot*. One good definition for autonomous robots among other definitions is "behavioral machines capable of operating in partially unknown and changing environments without, or with limited, human intervention" [Floreano and Mattiussi (2008)]. Reinforcement Learning family of algorithms (other algorithms may also be used for the implementation) are capable of creating such a system (e.g., an autonomous robot or agent). The ability of learning from experience is specifically useful in cases where training data is not prepared, unlike a supervised learning scenario.

One recent advance in machine learning that uses reinforcement learning is an agent that is able to play Atari 2600 with human level control [Mnih et al. (2015)]. Another example is the AlphaGo agent [Silver et al.

(2016)], which uses Monte Carlo tree-search and deep learning to play *Go* better than many human players. AlphaZero [Silver et al. (2017)] is a more advanced version of AlphaGo that learns to play chess by only playing with itself. AlphaZero even played with the famous chess engine of Stockfish [stockfishchess.org] and had great results [AlphaZero-Stockfish].

Reinforcement learning is also used in cognitive and autonomous systems, such as in an agent capable of autonomous exploration using active learning [Forestier et al. (2016)]—this agent is also capable of learning from human guidance— or in a system to extract temporal regularities from environment and use them as skills [Kompella et al. (2017)]. Reinforcement learning algorithms have also been used for conventional objectives for robots, such as navigation and obstacle avoidance [Ramezani Dooraki and Lee (2018)].

Reinforcement learning family of algorithms recently had more progress in terms of high-dimensional and low-dimensional control (optimization) problems. One of these algorithms is *deep deterministic policy gradient (DDPG)* [Lillicrap et al. (2015)], which is an actor-critic based algorithm that searches for an optimal deterministic policy. The next algorithm is *Trust Region Policy Optimization (TRPO)* [Schulman et al. (2015)], a stochastic policy gradient based algorithm. And finally *Proximal Policy Optimization (PPO)* [Schulman et al. (2017)], which is an enhancement over TRPO as mentioned by the authors, "proximal policy optimization (PPO) have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically)" [Schulman et al. (2017)]. These mentioned algorithms are capable of learning to converge to an optimal control policy for walking, jumping and some conventional control problems such as Acrobot, CartPole and Pendulum. They are also capable of learning to play Atari 2600 games. Nevertheless, to our knowledge these are not tested in a real or simulated quad-rotor drone to fully control the velocities of the engines with the goal of controlling the position and attitude of the drone without the help of a conventional controller.

In the following we emphasize on the achievements and contributions of our work:

- We introduce enhancements to the existing reinforcement learning methods that make them capable of learning an optimal flight policy.
- Based on our enhancements, we introduce a bio-inspired flight controller (BFC) that controls a quad-rotor drone without the help of any conventional controller such as PID, MPC and etc.

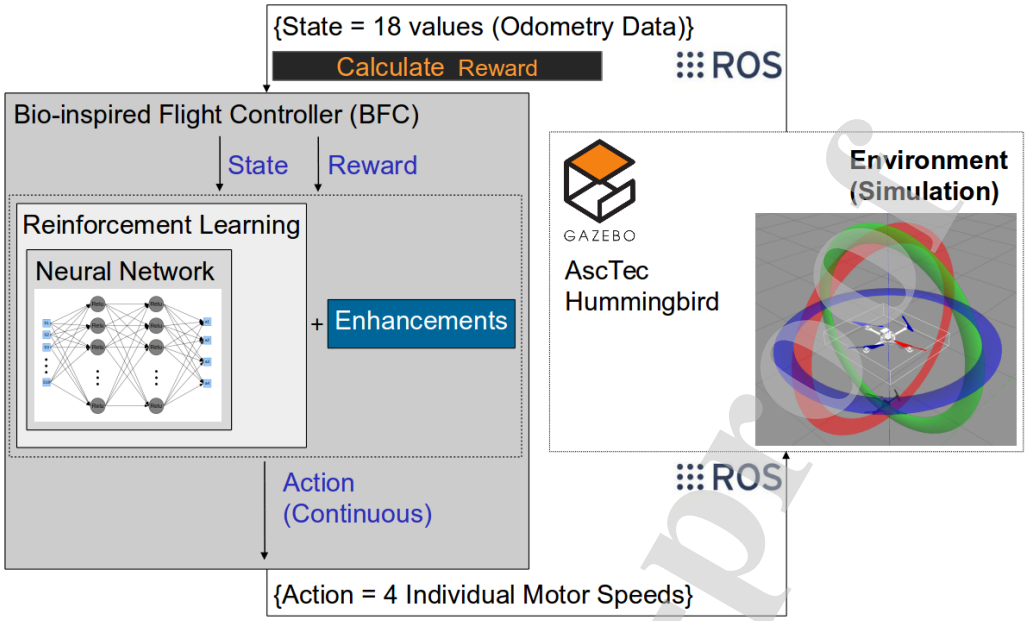


Figure 1: Bio-inspired Flight Controller and how it interacts with the simulation environment. In each time-step BFC takes the odometry data and generates the proper motor speeds.

- We tested BFC using different scenarios and showed how it learns to fly in a simulated environment, and compared its performance with and without noise. We also compared BFC with a linear-MPC.

In a previous work of ours [Dooraki and Lee (2019)], we briefly introduced the possibility of training a quad-copter to fly autonomously, and showed it is possible to enhance TRPO algorithm in order to learn a flight policy. This current text is hugely different with the previous work as explained in the following. In this work, we upgraded our previous work and tested other important reinforcement learning algorithms famous for being able to learn continuous action control policies such as TRPO, PPO and DDPG, unlike the previous work where we only discussed about enhancing TRPO. As a result, in this work we enhanced the PPO algorithm to create our bio-inspired flight controller (BFC) instead of TRPO. Moreover, in this paper, we explored the accuracy of our bio-inspired flight controller and the effect of several degrees and types of the noises on it. We also compared our algorithm with a Linear-MPC in detail, and explored the performance of BFC following way points

in our result section, unlike our previous work.

In the rest of this paper, we first write about the related and similar works, next we define our problem of "learning an optimal flight policy" and then analyse a reinforcement learning based solution which by itself is not capable of solving the problem. As a result, we continue by introducing our method in order to enhance the mentioned reinforcement learning based methods. Since the result of our enhancements is an enhanced reinforcement learning based method capable of learning to fly we call this new algorithm a bio-inspired flight controller (BFC) —the reason of this naming is because of the trial and error that is used in this controller in order to learn to fly which is similar to the learning by experience method in nature used by animals, also one of our enhancements is inspired by the way birds learn to fly. In the next step, in order to evaluate our method and controller we run extensive tests. Finally we discuss about our work and conclude it.

2. Related Works

The idea of using reinforcement learning to create intelligent autonomous robot is an established research topic and several works used this idea in un-manned aerial vehicles (UAVs).

There are a couple of studies that used reinforcement learning to create a high level controller that sends commands to a low level controller that controls the velocities of UAV motors. For example, Sadeghi and Levine [Sadeghi and Levine (2016)] used Deep Reinforcement Learning (DRL) to make a drone able to explore and avoid obstacles using a monocular camera, but they did not directly control the drone motors velocities. There are also works such as [Zhang et al. (2015)] by Zhang et al. where a model predictive control (MPC) with the assistance of a guiding system such as Vicon is used to provide samples for training a model in a supervised learning fashion, and used it in guided policy search (GPS).

Reinforcement learning is also used for creating low level controller for UAVs where it **directly controls** the velocities of UAV motors. In a work published by Ng et al. [Ng et al. (2006)] supervised learning and data collected from 391 seconds of flight performed by a human pilot is used to create a model of the heli-copter dynamics. This model can estimate a new state when having current state and action of the helicopter and is used in a reinforcement learning based algorithm for learning to control the helicopter in an inverted position. A human pilot was necessary to take the helicopter

to an inverted position and then they let the autonomous controller control the helicopter. In another paper by Abbeel et al. [Abbeel et al. (2006)], authors developed a reinforcement learning based algorithm using an extension of linear quadratic regulator (LQR) and differential dynamic programming (DDP). This was used to learn an autonomous model that is able to perform "forward flip and sideways roll at low speed, tail-in funnel, and nose-in funnel" helicopter maneuvers. A human pilot helped to generate the helicopter dynamics model and the reward function of the mentioned maneuvers. Comparing to our work, [Ng et al. (2006)] and [Abbeel et al. (2006)] used a human pilot for collecting data to create a dynamics model for training the quad-rotor and generating the reward function. They also focused on specific maneuvers demonstrated by the pilot. But we used the physics engine simulator of Gazebo simulator [Gazebosim.org] and a quad rotors model created by Fadri et al. [Furrer et al. (2016)] to simulate the state transition highly accurately. We also did not use any demonstration in the training process of our quad-copter, instead inspired by the nature, our algorithm tries to learn and control the quad-copter by trial and error.

Hwangbo et al. [Hwangbo et al. (2017)] worked on an algorithm that used reinforcement learning and proportional-derivative (PD) controller to stabilize a thrown quad-copter and follow way-points. The algorithm takes a state (18-dimensional vector) and generates an action (4-dimensional vector) for the quad-copter engines. Our work is similar to Hwangbo et al. work where a reinforcement learning based algorithm is used to control a quad-copter engine velocities, but is also different from Hwangbo et al. work where we used a pure reinforcement learning based controller in both training and playing phases, but they have used a proportional-derivative (PD) controller in combination with their reinforcement learning based controller during the training process , "The sum of the outputs of the two controllers are used as a command"[Hwangbo et al. (2017)]. While in many reinforcement learning algorithms the exploration strategy is initially random and gradually the learned policy changes the randomness nature of the algorithm, in the Hwangbo et al. paper a PD controller directs the exploration strategy which is as a result of using a command that is the sum of the output of the two controller (PD controller and RL policy). In this way, the reinforcement learning algorithm is guided by the PD controller to learn from the PD controller demonstrations while having exploration noise which helps it to learn better than the PD controller. As a result, the work is more similar to a Learning by Demonstration work. **Moreover, we introduced enhancements such as**

Time-dependent Terminal State (Section 4.3) to avoid catastrophic forgetting that could happen during the training phase. Our work title is similar to Lambert et al. [Lambert et al. (2019)]. However, they used a neural network to learn the forward dynamics model of the system and then used it with a Model Predictive Controller (MPC) to control the drone, which is very different comparing to our study in this paper. In work by Koch et al. [Koch et al. (2019)], they used reinforcement learning algorithms to perform the attitude control part of a flight controller. Comparing to our work, we designed and implemented a complete flight controller that flies to any desired position from any point in three-dimensional space. As a result, it controls the attitude and position of a quad-rotor drone and completely replaces the low-level flight controller.

3. Problem Definition

Consider an infinite-horizon Markov Decision Process (MDP), defined by the tuple (S, A, P, r, D, γ), where S is a finite set of states, A is a finite set of actions, $P : S \times A \times S \mapsto R$ is the transition probability distribution, $r : S \mapsto R$ is the reward function, $D : S \mapsto R$ is the distribution of the initial state s_0, and $\gamma \in (0, 1)$ is the discount factor. Our goal is train a reinforcement learning based agent to learn a flight policy.

Our state space is the odometry data generated based on IMU:

$$S_t = \{pos_x, pos_y, pos_z, vel_x, vel_y, vel_z, acc_x, acc_y, acc_z, att_x, att_y, att_z, ang\_vel_x, ang\_vel_y, ang\_vel_z, ang\_acc_x, ang\_acc_y, ang\_acc_z\}$$

where *pos* is used to define the position, *vel* is used for velocity, *acc* for acceleration, *att* is the angle with the axis, *ang_vel* is the angular velocity and *ang_acc* is the angular acceleration. Our action space is a n dimensional continuous space vector, $a_1..a_n$ where n is the number of rotors, and a_1 to a_n are individual motor speed.

To evaluate an action in a reinforcement learning based solution for a Marko-

Table 1: List of Symbols.

Symbol	Explanation
t	discrete time step
A_t	action at time t
S_t	state at time t
R_t	reward at time t
π	policy, decision-making rule
$\pi(s)$	action taken in state s under policy π
$\pi(a \mid s)$	probability of taking action a in state s under policy π
$\pi(a \mid s, \theta)$	probability of taking action a in state s given parameter θ
$Q_\pi(s_t, a_t)$	value of taking action a in state s under policy π
$V_\pi(s_t)$	value of state s under policy π (expected return)
$A_\pi(s, a)$	advantage of selecting a certain action from a certain state
ρ_π	the discounted visitation frequency of policy π
$\eta(\pi)$	expected discounted reward of policy π
KL	an improvement bound using Kullback-Leibler (KL) divergence, D_{KL} [Schulman et al. (2015)]
$P(s_{t+1} \mid s_t, a_t)$	probability of transitioning to state s_{t+1} from state s_t with action a_t
D	starting state distribution
γ	discount-rate parameter
pos	position
vel	velocity
acc	acceleration
att	attitude
ang_{vel}	angular velocity
ang_{acc}	angular acceleration
$dpos$	desired position
$mind$	minimum distance
$maxd$	maximum distance
TDT	Time-dependent Terminal
$TDTS$	Time-dependent Terminal State
CSS	Controlled Starting State
JD	Jumping Distribution
SV	Square Value, this is the value of each square when we divide the XY ground plane to equal squares. The quad-copter will be initialized in these squares.
ζ	Distance Threshold used for TDTS.

vian environment, we use the following conventions:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+1})\right],$$
$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+1})\right],$$
$$A_\pi(s, a) = Q_\pi(s_t, a_t) - V_\pi(s_t),$$

where

$$a_t \sim \pi(a_t \mid s_t), s_{t+1} \sim P(s_{t+1} \mid s_t, a_t) \text{ for } t \geq 0.$$

Considering the method and calculation explained about Trusted Region Policy Optimization by Schulam et al. [Schulman et al. (2015)], let π denote a stochastic policy $\pi : S \times A \mapsto [0, 1]$, and let $\eta(\pi)$ denote its expected discounted reward. We use the following identity expression as the expected return of another policy $\widetilde{\pi}$ in terms of the advantage over π:

$$\eta(\widetilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \ldots, \sim \widetilde{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)\right] \tag{1}$$

Let ρ_π be the discounted visitation frequencies:

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s)\gamma^2 P(s_2 = s) + \ldots,$$

where $s_0 \sim \rho_0$, and the actions are chosen according to π. We can rewrite equation 1 with a sum over states instead of time-steps:

$$\eta(\widetilde{\pi}) = \eta(\pi) + \sum_s \rho_{\widetilde{\pi}}(s) \sum_a \widetilde{\pi}(a \mid s) A_\pi(s, a) \tag{2}$$

Considering the complex dependency of $\rho_{\widetilde{\pi}}(s)$ on $\widetilde{\pi}$, equation 2 is difficult to calculate, and the following local approximation is introduced instead:

$$L_\pi(\widetilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \widetilde{\pi}(a \mid s) A_\pi(s, a) \tag{3}$$

If we have a parameterized policy π_θ, where $\pi_\theta(a \mid s)$ is a differentiable function of parameter vector θ, then L_π matches η to the first order. That is, for any parameter value θ_0,

$$L_{\pi_{\theta_0}} = \eta(\pi_{\theta_0}),$$
$$\nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta) \mid_{\theta=\theta_0} = \nabla_\theta \eta_{\pi_{\theta_0}} \mid_{\theta=\theta_0} \tag{4}$$

TRPO uses a conservative policy iteration to provide explicit lower bounds on the improvement of η. For the improvement of our policy, we also define an improvement bound using Kullback-Leibler (KL) divergence, D_{KL} [Schulman et al. (2015)].
Since we consider the parameterized policy $\pi_\theta(a \mid s)$ with parameterized vector θ, we will overload our previous notation to use functions of θ rather than π, e.g. $\eta(\theta) := \eta(\pi_\theta), L_\theta(\theta) := L_{\pi_\theta}$, and $D_{KL}(\theta \mid\mid \theta) := D_{KL}(\pi_\theta \mid\mid \pi_\theta)$. θ_{old} is used to denote the previous policy parameters that we want to improve.

We collect a sequence of states by sampling $s_0 \sim \rho_0$ using policy $\pi_{\theta_{old}}$ to generate a trajectory $s_0, a_0, s_1, a_1, ..., s_{T-1}, a_{T-1}, s_T, a_T$ where T is the length of an episode. We also calculate $q(a \mid s) = \pi_{\theta_{old}}$. $Q_{\theta_{old}}(s, a)$ should be calculated at each state-action pair (s_t, a_t) by taking the discounted sum of future rewards along the trajectory.

$$\max_\theta L_{\theta_{old}}(\theta)$$
$$\text{subject to } \overline{\mathbb{D}}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta \tag{5}$$

Using equation 3, we obtain the following equation:

$$\max_\theta \sum_s \rho_{\theta_{old}}(s) \sum_a \pi_\theta(a \mid s) A_{\theta_{old}}(s, a)$$
$$\text{subject to } \overline{\mathbb{D}}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta \tag{6}$$

For each s_n, we can calculate the first term by:

$$\sum_a \pi_\theta(a \mid s_n) A_{\theta_{old}}(s_n, a) = \mathbb{E}_{a\sim q}\left[\frac{\pi_\theta(a \mid s_n)}{q(a \mid s_n)} A_{\theta_{old}}(s_n, a)\right]$$

Finally, we approximately solve the following constrained optimization problem and update policy parameters:

$$\max_\theta \mathbb{E}_{s\sim\rho_{\theta_{old}}, a\sim q}\left[\frac{\pi_\theta(a|s)}{q(s|s)} Q_{\theta_{old}}(s, a)\right]$$
$$\text{subject to } \mathbb{E}_s \sim \rho_{\theta_{old}}[D_{KL}(\pi_{\theta_{old}}(. \mid s) \mid\mid \pi_\theta(. \mid s))] \leq \delta \tag{7}$$

In the TRPO paper, two methods are mentioned regarding the training: single path and vine. Here we selected the single path method where we select a sequence of states by sampling $s_0 \sim \rho_0$ and then simulating the policy $\pi_{\theta_o ld}$ for some number of time-steps to generate trajectory $s_0, a_0, s_1, a_1, ..., s_{T-1}, a_{T-1}$,-$s_t$. Hence, $q(a \mid s) = \pi_{\theta_{old}}$. $Q_{\theta_{old}}(s, a)$ is computed at each state-action pair (s_t, a_t) by taking the discounted sum of future rewards along the trajectory. In PPO, the following equation (the result of TRPO) is enhanced to make the policy more general:

$$\max_{\theta} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(s|s)} \hat{A}_t \right]$$
$$\text{subject to } \hat{\mathbb{E}}_t[KL[\pi_{\theta_{old}}(. \mid s), \pi_\theta(. \mid s)]] \leq \delta \tag{8}$$

The theory justifying TRPO suggests a penalty value instead of a constraint; i.e., solving the unconstrained optimization problem:

$$\max_{\theta} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(s|s)} \hat{A}_t - \beta KL[\pi_{\theta_{old}}(. \mid s), \pi_\theta(. \mid s)] \right] \tag{9}$$

However, it is explained in the TRPO and PPO papers that a fixed penalty coefficient β and optimizing with SGD are not sufficient, and additional modifications are required.

Considering $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, where $r(\theta_{old}) = 1$, TRPO maximizes a surrogate objective:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right] \tag{10}$$

where CPI is an acronym for conservative policy iteration [Kakade and Langford (2002)]. To maximize equation 10 without a constraint while avoiding excessively large policy updates, in PPO paper, the equation is modified as follows to penalize changes to the policy that move $r_t(\theta)$ away from 1:

$$L^{CLIP}(\theta) =$$
$$\hat{\mathbb{E}}_t \left[min(r_t(\theta)) \hat{A}_t, clip(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right] \tag{11}$$

where ϵ is a hyper-parameter, which can be equal to 0.2, for example. Instead of Equation 11 or in addition to it, it is possible to use the following Adaptive

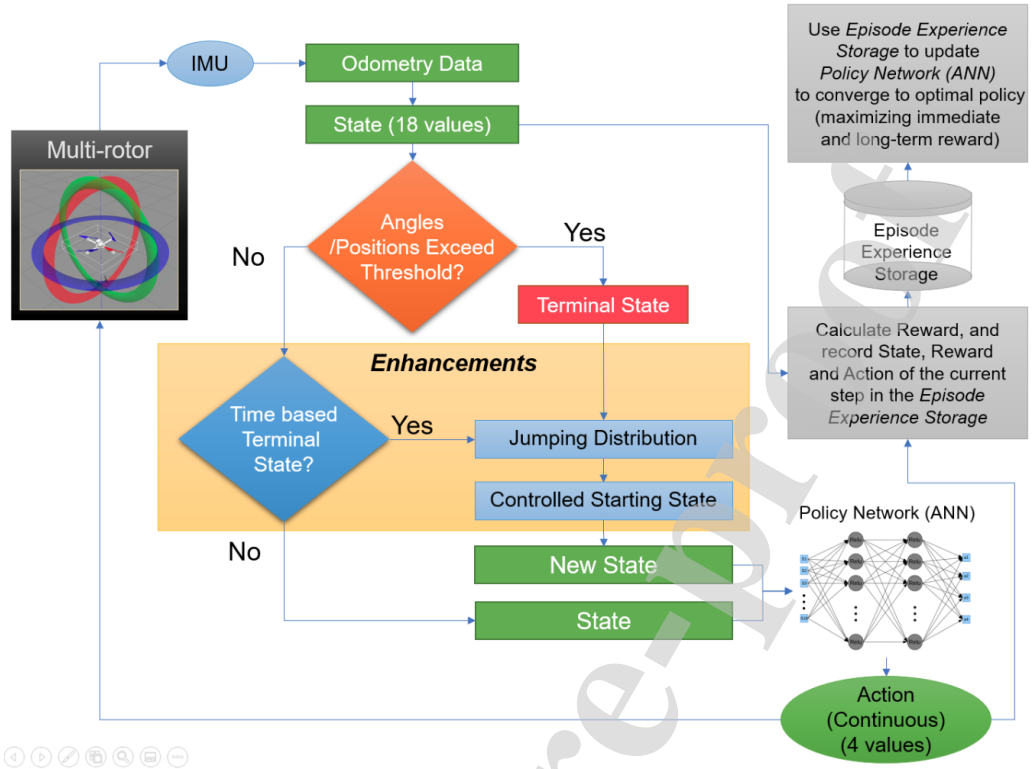


Figure 2: Flowchart of bio-inspired flight controller algorithm.

KL Penalty Coefficient objective, as mentioned in the PPO paper:

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(s|s)}\hat{A}_t - \beta KL[\pi_{\theta_{old}}(.\mid s), \pi_\theta(.\mid s)]\right] \tag{12}$$

where the following is computed in each iteration to obtain the new β:

$$d = \hat{\mathbb{E}}_t\Big[KL[\pi_{\theta_{old}}(.\mid s), \pi_\theta(.\mid s)]\Big]$$
$$\text{If } d < d_{targ}/1.5, \beta = \beta/2$$
$$\text{If } d > d_{targ} \times 1.5, \beta = \beta \times 2$$

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) - c_2 S[\pi_\theta](s_t)\right] \tag{13}$$

where c_1 and c_2 are coefficients, S is an entropy bonus, and L_t^{VF} is a squared-error loss $(V_\theta(s_t) - V_t^{targ})$

One style of policy gradient implementation popularized by Mnih et al. [Mnih et al. (2016)] runs the policy for T time-steps, where T is much less than the episode length and uses the collected samples for an update. This style requires an advantage estimator that does not look beyond time-step T. The estimator used by Mnih et al. [Mnih et al. (2016)] is:

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + ... + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \tag{14}$$

Generalizing this choice, we can use a truncated version of generalized advantage estimation that reduces to the following equation:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \quad \text{where } \delta_t = r_t + \gamma V_{(s_{t+1})} - V(s_t) \tag{15}$$

This optimization is capable of learning a ground robot control algorithm, but it is not able to learn an optimal flight policy. To solve this issue, we introduce some enhancements.

4. Enhancements

We consider our enhancements to be part of the exploration mechanism of a reinforcement learning algorithm. As a result, it can be used in combination with DDPG, TRPO, and PPO. Furthermore, we argue our enhancements to be part of the reinforcement learning algorithm and not environment-specific enhancements since it can be used for any MDP problem and is not limited to our learning problem and our *ab initio* designed environment using Gazebo sim and ROS.

4.1. Jumping Distribution ***(JD)***

We define the starting state space (D) of the MDP to be:

$$\begin{aligned} D = \{&pos_x, pos_y, pos_z, vel_x, vel_y, vel_z, acc_x, acc_y, acc_z, att_x, att_y, \\ &att_z, ang\_vel_x, ang\_vel_y, ang\_vel_z, ang\_acc_x, ang\_acc_y, ang\_acc_z\} \\ &pos_x = [-x_{mind}, +x_{maxd}] \\ &pos_y = [-y_{mind}, +y_{maxd}] \\ &pos_z = 0.0 \end{aligned} \tag{16}$$

where all the variables are equal to 0.0 except for pos_x and pos_y. This results from the fact that the quad-rotor should start on the ground and take off autonomously, and being on the ground means all values except the ground position components (pos_x, pos_y) should be equal to 0.0. We define a new complementary starting state distribution called the *Jumping State* distribution ($D_{Jumping-State}$) to be used beside the actual MDP starting state distribution (D):

$$\begin{aligned} D = \{&pos_x, pos_y, pos_z, vel_x, vel_y, vel_z, acc_x, acc_y, acc_z, att_x, att_y, \\ &att_z, ang\_vel_x, ang\_vel_y, ang\_vel_z, ang\_acc_x, ang\_acc_y, ang\_acc_z\} \\ &pos_z = [-z_{mind}, +z_{maxd}] \end{aligned} \tag{17}$$

It is similar to the starting state space D, and the only difference is that pos_z is equal to a positive value instead of 0. Thus, the quad-rotor starts up in the air after reaching a terminal state. A Jumping State distribution allows the agent to obtain information about states that it would not necessarily visit otherwise. **This idea might look similar to Curriculum Learning [Bengio et al. (2009)], wherein the goal is to show the more complicated states to the agent after learning simpler states. However, by using JD, we try to show states that might not be visited without JD (in a normal scenario), and those states are not necessarily more complicated; they are just other states.**
In the policy gradient in each time-step, we move to the next state (S_{t+1}) based on the current action (a), which is selected by policy π. As a result, the agent always follows the trajectories starting from the state starting space (D). Using $D_{Jumping\_State}$ as a complementary starting state distribution will give the agent the possibility of experiencing new trajectories.

For selecting the starting state from D (the starting state distribution) or JD (the starting state distribution we defined, as the Jumping Distribution),

we use a probability function that gives the highest probability to select from JD at the beginning of the training process. We reduce this probability while the time-steps increase, and finally, the probability of selecting from the JD distribution will be equal to 0.0 after k time-steps. This is similar to ϵ in the ϵ-greedy policy.

This idea is borrowed from the nature where fledglings (baby birds) start to jump out of the nest or off a cliff to learn to fly. In a similar way, the quad-rotor starts up in the air to mimic a jumping scenario.

4.2. Controlled Starting State ***(CSS)***

As explained in the previous section, we select the next state from starting state distribution D or the complementary starting state distribution JD after reaching a terminal state. Either choosing the staring state from D or JD, pos_x and pos_y are returned randomly considering the defined permitted ranges:

$$pos_x = [-x_{mind}, +x_{maxd}]$$
$$pos_y = [-y_{mind}, +y_{maxd}]$$

Selecting randomly from these ranges (two-dimensional points) plus the value of pos_z will allow the agent to start its trajectory from different points in three-dimensional space and experience all of the different possibilities. Nonetheless, selecting these starting states purely by random will cause the agent to be able to fly from some of the starting states better than the others. In order to mitigate this error, we define a mechanism in that allows the agent to calculate the rewards gathered from each region to be able to select the starting state from the regions that have less average reward.
We divide the two-dimensional space of $\{pos_x, pos_y\}$ to n squares where each square has a $Square\_Value_n$ that is equal to the average of rewards acquired by starting from the states within the borders of that square. Thus, by starting from a square n at time t, the corresponding $Square\_Value$ **(SV)** will change using the following equation:

$$SV_n = \frac{1}{2}(SV_n + \frac{1}{n}\sum_{S_t}^{S_{t+n}} r) \tag{18}$$

4.3. Time-dependent Terminal State ***(TDTS)***

We define a time-dependent terminal state with the goal of helping the agent not to forget what it learned so far by gathering experiences from all areas within the state space. The agent reaches a time dependent terminal state if the following condition is true for a period of time equal to *threshold_time*:

$$
\begin{gathered}
S_t \in D_{TDT} \\
\text{where} \\
D_{TDT} = \{pos_x, pos_y, pos_z, vel_x, vel_y, vel_z, acc_x, acc_y, acc_z, att_x, att_y, \\
att_z, ang\_vel_x, ang\_vel_y, ang\_vel_z, ang\_acc_x, ang\_acc_y, ang\_acc_z\} \\
\text{and} \\
pos_x \in [d_x - \zeta, d_x + \zeta] \\
pos_y \in [d_y - \zeta, d_y + \zeta] \\
pos_z \in [d_z - \zeta, d_z + \zeta]
\end{gathered} \tag{19}
$$

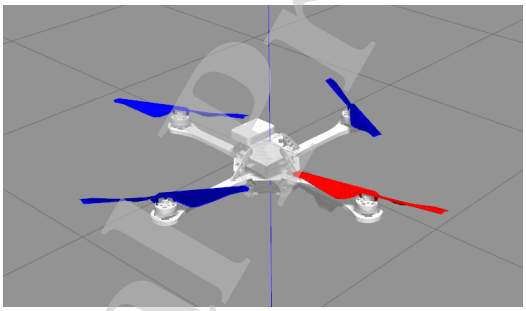


Figure 3: Simulated Ascending Technology Hummingbird quad-copter in Gazebo.

5. Bio-inspired Flight Controller (BFC)

We enhanced the PPO algorithm to create the BFC. The BFC is a policy gradient based, model-less, on-policy reinforcement learning algorithm that tries to converge to an optimal policy. A policy that provides for the agent the possibility of taking an optimal action a in any specific state s and returns the maximum immediate and discounted future reward r.

Algorithm 1 Bio-inspired Flight Controller (BFC) Algorithm

Initialize Policy's parameter vector (θ)
reset environment and s_t = environment()
while True **do** ▷ Loop for infinite
    $a_t = \pi(s_t)$
    $s_{t+1}, r_t, terminal$ = environment(a_t)
    Record all episode data
    if (terminal) **then**
        Reset Using Jumping Distribution
        Set Position Using Controlled Starting State
    else
        if Time_based Terminal State **then**
            Reset_Position()
        end if
    end if
    $s_t = s_{t+1}$
    if Horizon **then**
        Do optimization according to the solution provided in *Problem Definition* section
    end if
end while

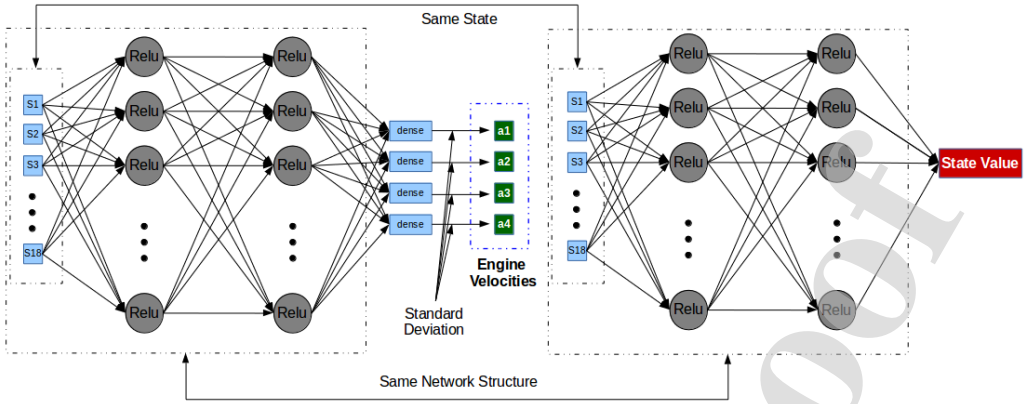


Figure 4: BFC neural network architecture. Standard Deviation applies for stochastic policy.

BFC completely replace the quad-rotor conventional controller and directly controls the engines velocities by observing the quad-rotor state. It is able to follow way-points and stabilize the quad-rotor attitude and position accordingly.

We call this reinforcement learning based flight controller *bio-inspired flight controller (BFC)* because it learns to control the quad-rotor in a similar way to biological creatures such as birds, and also the idea of *Jumping Distribution* enhancement is borrowed from the nature as explained in that section.

6. Experiments

During the learning phase, BFC should learn to control the drone and flies it to a desired position and attitude while the drone can be released from any position and attitude according to the Section 4.1. After training phase, BFC learns to fly to a desired position and attitude from any position and attitude. Further, it is possible to change the desired points in the flight time and let BFC to follow way-points.
We setup different experiments to illustrate the effect of our enhancements and capability of BFC (enhanced-PPO):

- We tested DDPG, TRPO and PPO for learning an optimal flight policy. We then made the enhancements and trained the quad-rotor drone again using the enhanced version of the mentioned algorithms

- In each test, we gathered information such as the reward acquired by the agent, the position and attitude error, and the rate of successful flights of the agent. We show these results only for the enhanced versions of TRPO and PPO (because these two algorithms were able to learn an optimal policy after enhancement)
- **In order to show the importance of each enhancement on the baseline algorithm (PPO), we performed an ablation study, where we measured the average, and accumulated reward, position, and attitude error, and averaged unsuccessful flights for each algorithm (that is, JD-CSS-TDTS, JD-CSS, JD-TDTS, JD, CSS-TDTS, TDTS, and CSS). Further, we illustrate the trajectory of flights performed by each algorithm.**
- We added different amounts of normal noise and OU (OrnsteinUhlenbeck) noise to our odometry data in order to measure the stability of BFC with noise
- We compare the performance of BFC with a linear-MPC controller in terms of starting to fly from rough attitudes.
- Finally, we use BFC to follow way-points.

6.1. Environment

We implemented an environment using Gazebo simulator, which is a robust physics engine simulator. It can simulate our agent, provide the odometry data which we use for creating our state and execute our action. Executing an action will change the agent state, where this change is precisely calculated by Gazebo and the new state will be returned.

To communicate with the environment, we prepared the necessary connections between the algorithm and Gazebo to perceive the current state of the quad-rotor drone, which contains the accelerations calculated in our algorithm in addition to the odometry data. It also runs an action (the action is a 4-dimensional vector where each continuous dimension is the velocity of an individual motor), receives the subsequent state, and generates the appropriate reward or resets the agent when it is necessary. There is no Gazebo-based environment by default in OpenAI gym that provides the functionalities we need as far as we know, so it is necessary to implement the mentioned parts.

We used the Robot Operating System (ROS) [ROS.org] for the communications between our algorithm and Gazebo simulator. On of the benefits of using ROS is when we want to use our algorithm in a different environment or in the real environment, we only need to change the old ROS topics provided by the old environment with the new ROS topics provided by the new environment.

6.2. Agent

Our agent is a simulated Ascending Technology Hummingbird quadcopter [Furrer et al. (2016)] in Gazebo. The agent executes the action provided by our algorithm, the action is a four-dimensional vector that contains four values where each value is the speed of one of the four motors.

6.3. Learning

We used OpenAI Baselines [Dhariwal et al. (2017)] implementation for DDPG, TRPO and PPO, connected them to our environment, and executed the first groups of tests. After that, we modified the codes, implemented our enhancements, and executed the next round of the tests. We performed several tests using each algorithm to have reliable results. In our experiments, we used a 2-layer MLP neural network architecture. We used 2 layers of 400 neurons for DDPG, 2 layers of 64 neurons for TRPO and 2 layers of 256 neurons for PPO (the PPO and TRPO neural network architecture is visualized in Figure [4]). The DDPG network architecture is identical to that in the original DDPG paper [Lillicrap et al. (2015)]. The hyper parameters of the algorithms and their neural networks are the default ones unless mentioned the otherwise.

6.4. Reward function

One of the most important parts of a reinforcement learning based algorithm is the reward function. We used a reward-shaping technique that is based on positions and angles. If the rotors exceed a threshold in pitch and roll, we reset the position because we consider it as a terminal state. We also limited the z, x and y components of quad-rotor drone's position, and if the rotors exceed those limitations, there will be a negative reward equal to a terminal state. Finally, our reward is a function of angles and positions, where the lower angles and distances with the desired point and attitude return higher rewards for the agent.

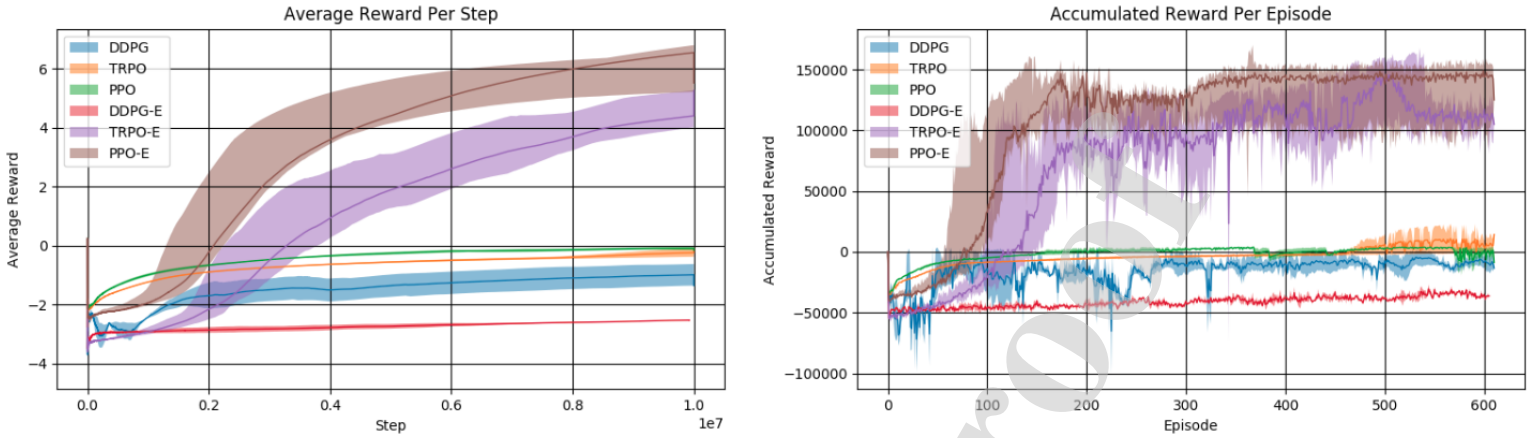
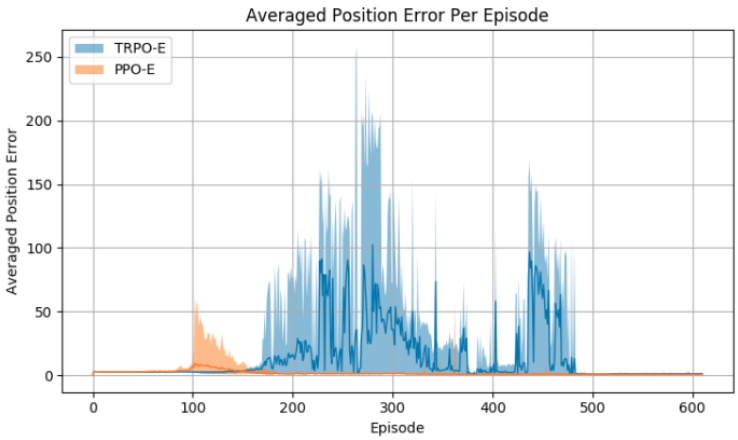


Figure 5: DDPG, TRPO, PPO, Enhanced-DDPG, Enhanced-TRPO and Enhanced-PPO (BFC) average rewards.

7. Results

Figures 5 shows that among different algorithms only enhanced TRPO and enhanced PPO are capable of learning an optimal flight policy. Therefore, for the sake of simplicity, we only show the detailed results of enhanced TRPO and enhanced PPO because only these two algorithms can learn an optimal flight policy. One important difference between enhanced TRPO and enhanced PPO is that enhanced PPO converges in a considerable shorter amount of time compared to enhanced TRPO.

Figure 6 shows the learning error of the enhanced TRPO and enhanced PPO algorithms in terms of their average position and attitude error from the desired point and attitude. As the results show, the capability of enhanced PPO is better than that of enhanced TRPO. Finally, our last diagram in this section shows the rate of unsuccessful flights of each enhanced algorithm. In a successful flight, the UAV starts from a random position in the air or the ground and flies toward the desired point, stabilizes the rotors position and attitude at that point, and stays there until we reset the quad-rotors or give it a new desired point. Thus, a higher average rate of unsuccessful flight means the quad-rotor drone is not able to fly optimally from a random point, and it ends up in a terminal state that is related to exceeding a position or attitude threshold. As we can see in Figure 7, enhanced PPO can stably reduce its unsuccessful flight rate and is better than enhanced TRPO. In the following steps, we tried to measure the accuracy of our algorithm and the effect of



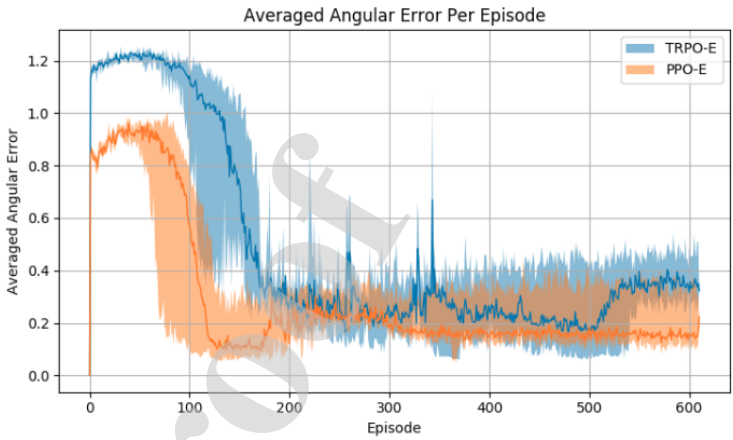


Figure 6: Comparing enhanced-TRPO and enhanced-PPO (BFC) position error.

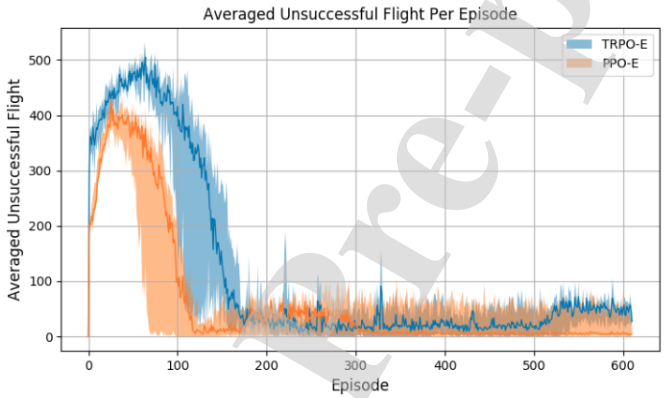


Figure 7: Enhanced-TRPO and enhanced-PPO (BFC) successful flight rates.

noise on a learned model and compared its capability to that of a linear-MPC controller.

7.1. Ablation Study

In this section, we illustrate the result of using each enhancements or combination of them on the baseline algorithm (PPO). We checked the following algorithms for the ablation study:

- **JD-CSS-TDTS: PPO baseline algorithm enhanced by JD (jumping distribution), CSS (Controlled Starting State), and TDTS (Time-Dependent Terminal State).**

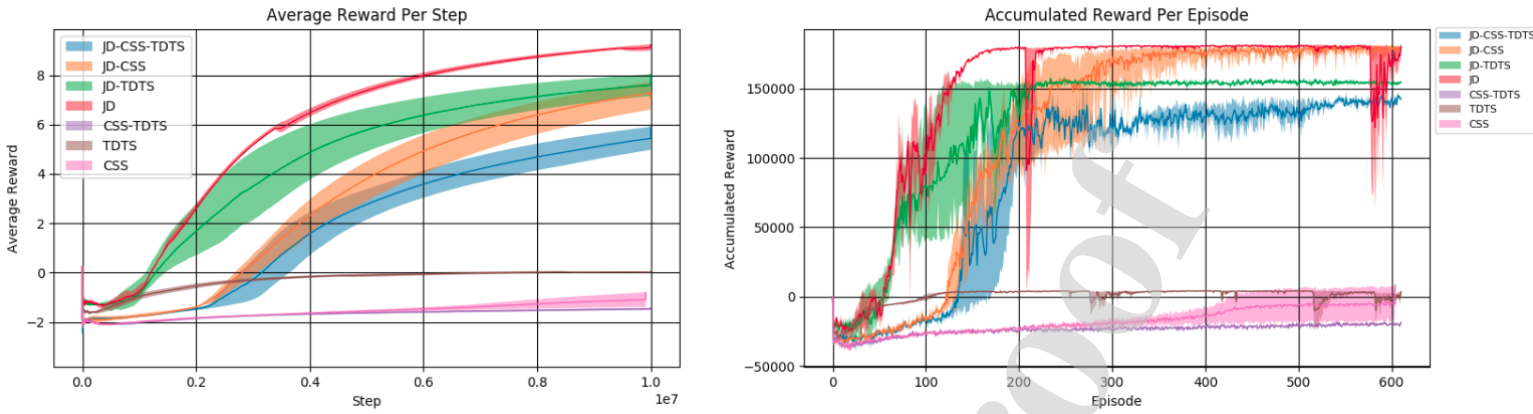


Figure 8: Average and cumulative rewards for JD-CSS-TDTS, JD-CSS, JD-TDTS, JD, CSS-TDTS, TDTS, and CSS.

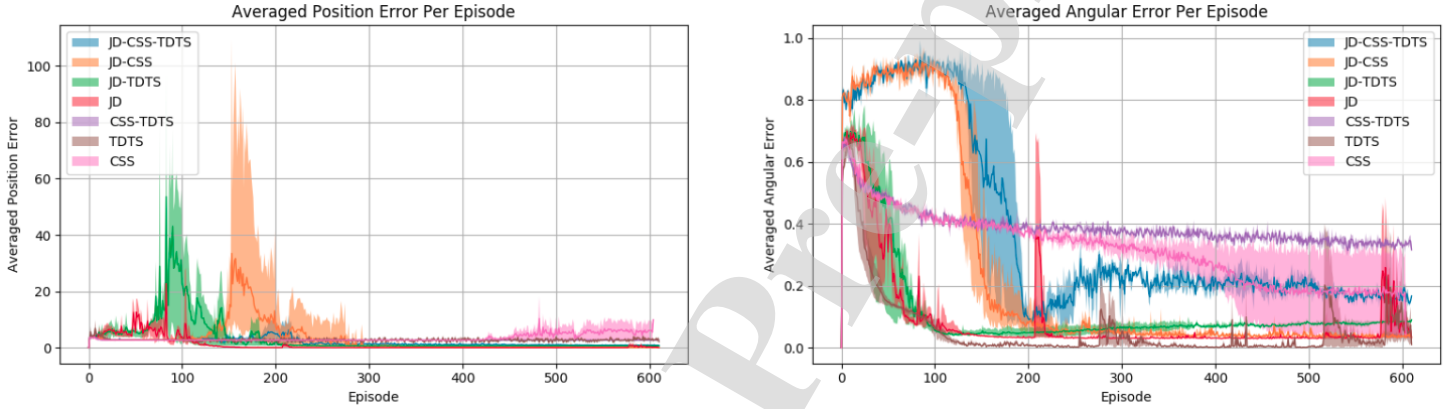


Figure 9: Position and attitude errors for JD-CSS-TDTS, JD-CSS, JD-TDTS, JD, CSS-TDTS, TDTS, and CSS.

- **JD-CSS: PPO baseline algorithm enhanced by JD (jumping distribution) and CSS (Controlled Starting State).**
- **JD-TDTS: PPO baseline algorithm enhanced by JD (jumping distribution) and TDTS (Time-Dependent Terminal State).**
- **CSS-TDTS: PPO baseline algorithm enhanced by CSS (Controlled Starting State) and TDTS (Time-Dependent Terminal State).**
- **JD: PPO baseline algorithm enhanced by JD (jumping distri-**

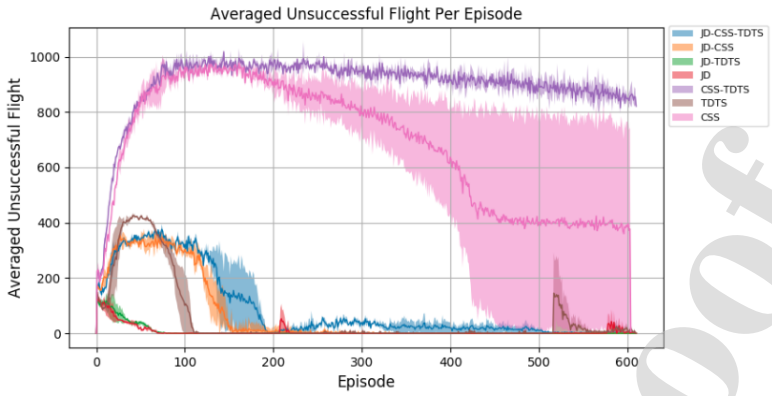


Figure 10: Averaged unsuccessful flights for JD-CSS-TDTS, JD-CSS, JD-TDTS, JD, CSS-TDTS, TDTS, and CSS.

bution).

- **CSS: PPO baseline algorithm enhanced by CSS (Controlled Starting State).**
- **TDTS: PPO baseline algorithm enhanced by TDTS (Time-Dependent Terminal State).**

Table 2: Algorithms Results.

Algorithm	Averaged Success	Averaged Fail
PPO + JD-CSS-TDTS (BFC)	90.5	9.5
PPO + JD-CSS	67	23
PPO + JD-TDTS	25	75
PPO + JD	16	82

The result of testing all algorithms is shown in Figure 8 and Figure 9, and Figure 10, the following algorithms are able to learn a flight policy (a policy that is able to maximize its average reward along the time): JD-CSS-TDTS, JD-CSS, JD-TDTS, and JD. We display the trajectories of hundred random flights performed by each one of the mentioned algorithms in Appendix 1).

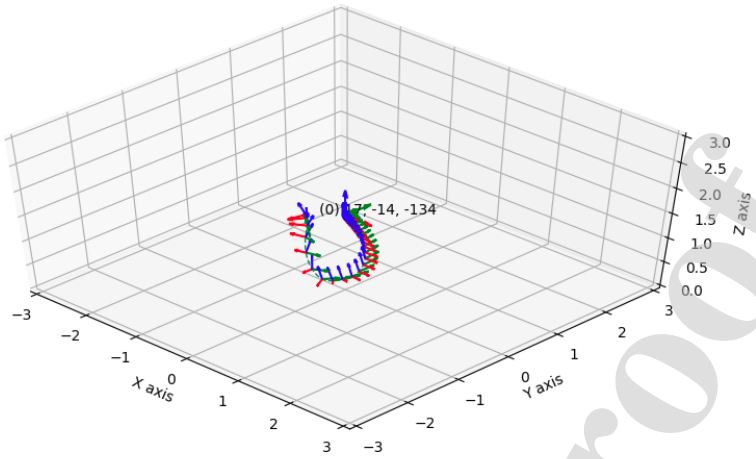


Figure 11: One sample flight trajectory where the quad-copter is started at a random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in each 20 steps in the trajectory.

One important observation based on Figure 8 and Figure 9 is that JD is a vital enhancement for the baseline algorithm to learn a flight policy, algorithms that are not enhanced using JD are not able to learn a flight policy.

Considering merely the results shown in Figure 8, Figure 9, and Figure 10, JD shows the best performance and after that JD-CSS, JD-TDTS and finally JD-CSS-TDTS. Nonetheless, our in detail testing results show that the best performance belongs to JD-CSS-TDTS (as shown in Table 2). We checked each algorithm in detail using the following method (Figure 15 can clarify the testing scenario):

- **First, we release the quad-rotor drone at a random point and attitude in space while the controller (BFC) is active and set for the** *desired_point* **of [0,0,150] (centimeter).**
- **Then, we start gathering the position and attitude data and counting for 10 seconds while the flight controller frequency is 250 Hz.**
- **We consider a** *desired_point_threshold* **equal to 10 cm and define a** $desired\_area = desired\_point \pm desired\_point\_threshold$**. If the**

quad-rotors reach to the desired area and stay there for 1 second then we consider this as 'Success' for the controller, otherwise it will be considered as 'Fail'.

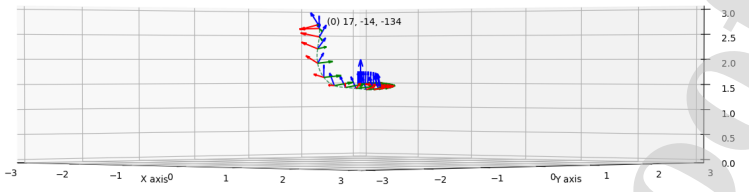


Figure 12: One sample flight trajectory where the quad-copter is started at a random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in each 20 steps in the trajectory.

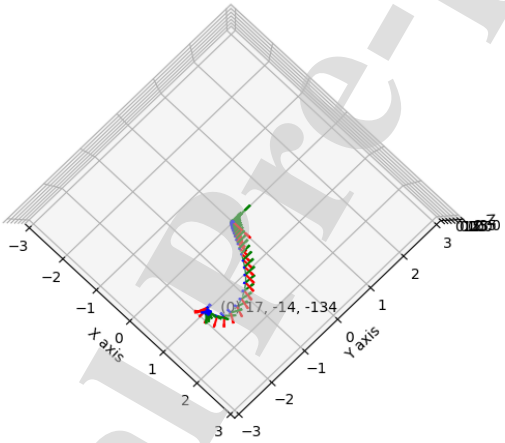


Figure 13: One sample flight trajectory where the quad-copter is started at a random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in each 20 steps in the trajectory.

Table 2 shows the results gathered by releasing the quad-rotor in a hundred random position and attitude and measuring how many times the BFC (each version) can successfully control the drone and fly it to the desired point within the desired time. The hundred positions and attitudes are initialized randomly; however, the same hundred positions and attitudes used for testing all the

different versions of the BFC. The results shown in Table are the averaged result performed by several trained algorithms using one specific setup (For example, we trained three algorithms by enhancing using JD and tested each one of them using the same hundred random position and attitude and averaged the number of success and failure).

To clarify our results, we illustrate the trajectory of flights performed using our best BFC version (JD-CSS-TDTS) in the next section. The flight trajectories of the other versions mention in Table 2 are shown in Appendix 1.

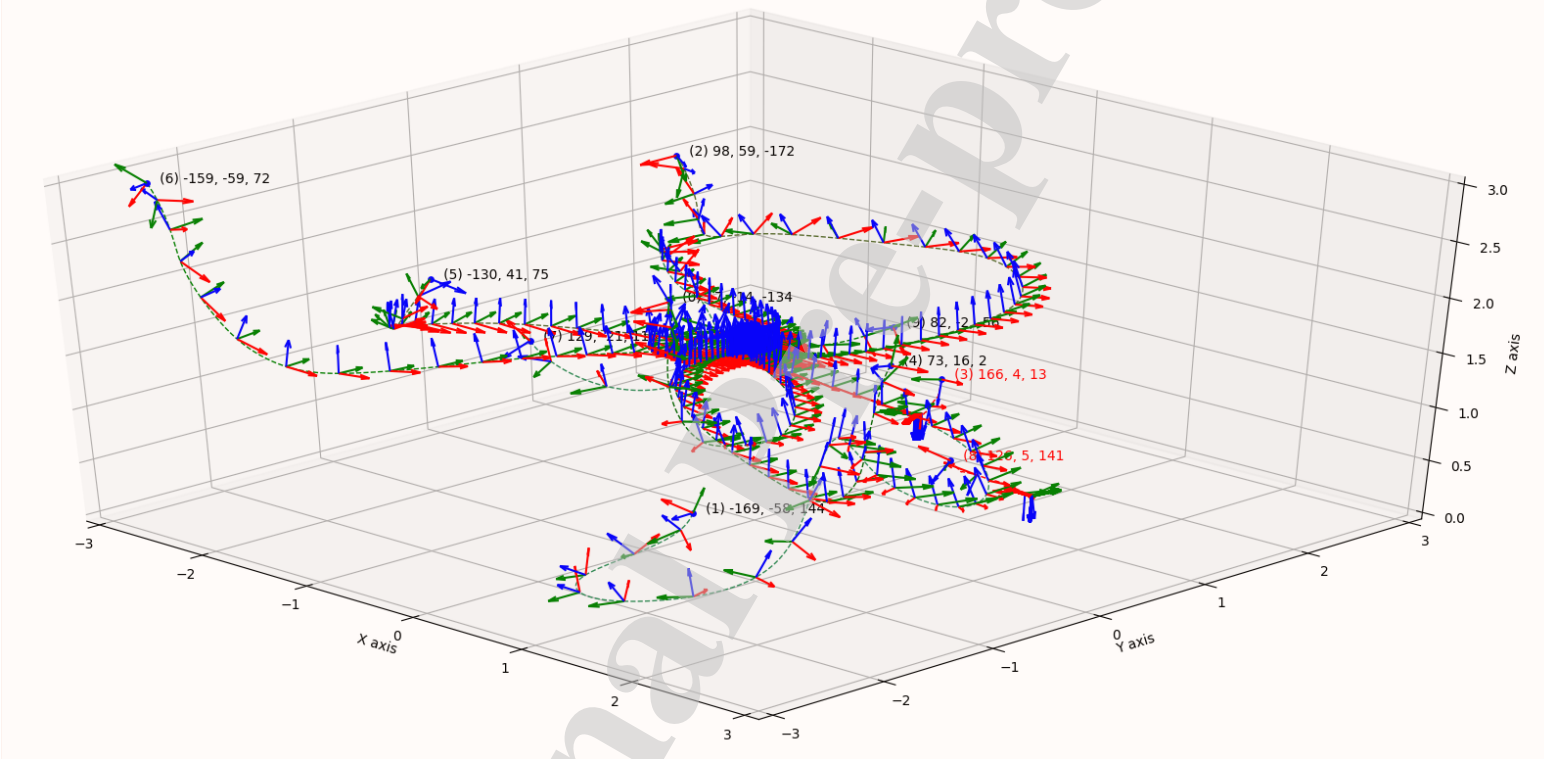


Figure 14: Ten sample flight trajectory where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in starting step of each trajectory..

7.1.1. Flight Trajectory

In this section, we show the trajectory of each flight from the time of releasing the quad-copter in a random position and atti-

tude to the time of reaching the desired point or hitting the ground (maximum in ten seconds). In each trajectory, we show the position in the 3D space and show the attitude using a right-hand coordinate system where the X-axis is shown with the red arrow, Y-axis is shown with the green arrow, and blue axis shows the Z-axis.

In Figure 11, a random starting position and attitude are selected for the quad-copter controlled by BFC (JD-CSS-TDTS). Further, to increase the flight trajectory clarity, we put a coordinate system in every 20 steps of the flight trajectory. To clarify the flight of the copter more in detail, we shown two more figures related to the same flight trajectory, but from different points of view, Figure 12 and Figure 13 (side-view and top-view). In the next step, we show ten random flight trajectories of BFC (JD-CSS-TDTS), with a right-hand coordinate system at the starting point of each flight trajectory, and redrawing the coordinate system in every 20 steps (The side-view and top-view of this ten random flights are shown in Appendix 1).

So far, we showed one sample flight trajectory, and ten sample flight trajectories to explain the idea of our flight trajectory visualization. Next, we show a hundred flight trajectories. Nevertheless, in the case of a hundred flight trajectories, it is not possible to show the coordinate system in every 20 steps; as a result, we only show the coordinate system of the first attitude in the trajectory.

7.2. Accuracy

In order to measure the accuracy of BFC (enhanced PPO), it is necessary to gather separate samples after the training is finished to focus on its stabilization capability rather than accumulated error over time. Figure 17 shows the BFC error in terms of position and attitude stabilization. The accuracy is measured in a period of ten seconds where each second is divided to 200 time-steps. We gathered the position and attitude error in each time-step.

7.3. The System with Noise

Next, we added random noise to the odometry data of the agent receiving from Gazebo simulator. The noise was added to the system after training and achieving an optimal policy. Figure 18 shows the position and attitude

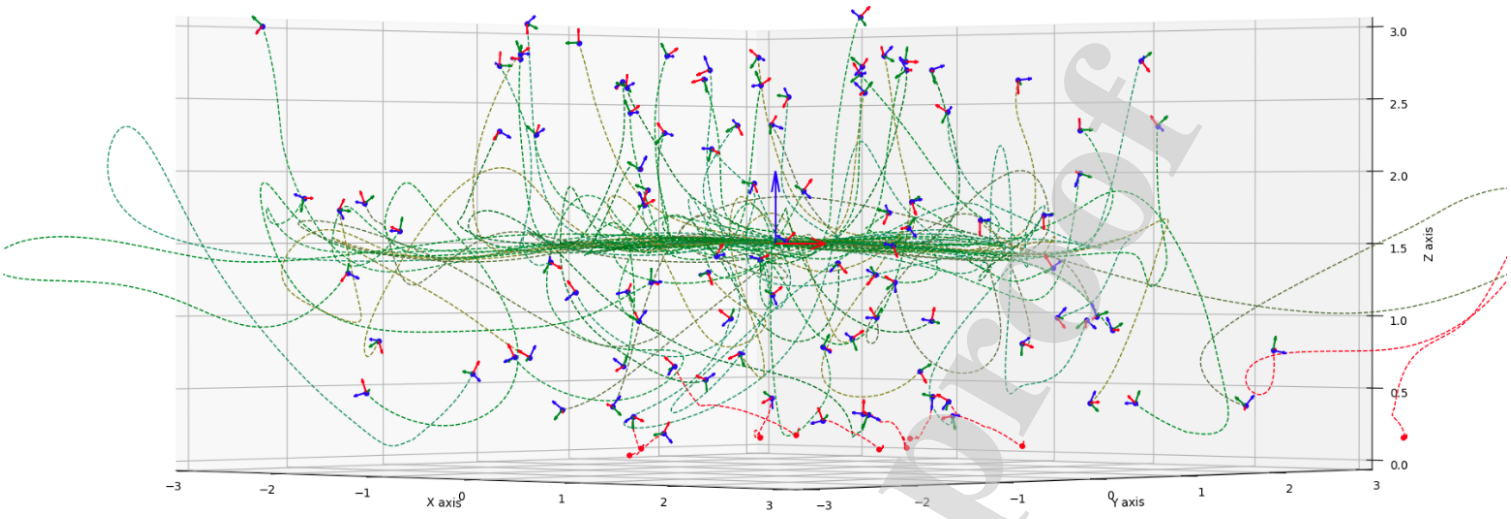


Figure 15: Hundred sample flight trajectory using E-PPO (JD-CSS-TDTS) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in starting step of each trajectory.

error after adding normal noise with mean of 0.0 and standard deviations of 0.05, 0.1, and 0.2. It also shows BFC with OU (OrnsteinUhlenbeck) noise with mean of 0.0 and standard deviation of 0.05. In the case of OU noise, first, we added the noise to all the elements of odometry and then we added noise to the position related elements of odometry. The results show that BFC is capable to keep the quad-rotor drone stable despite having odometry noise.

7.4. Comparison of BFC with Linear-MPC

Linear-MPC controllers are among the capable flight controllers for quad-rotors. We tried to compare BFC with an existing linear-MPC controller [Kamel et al.] provided for Gazebo simulator.
We performed our test in the following way:

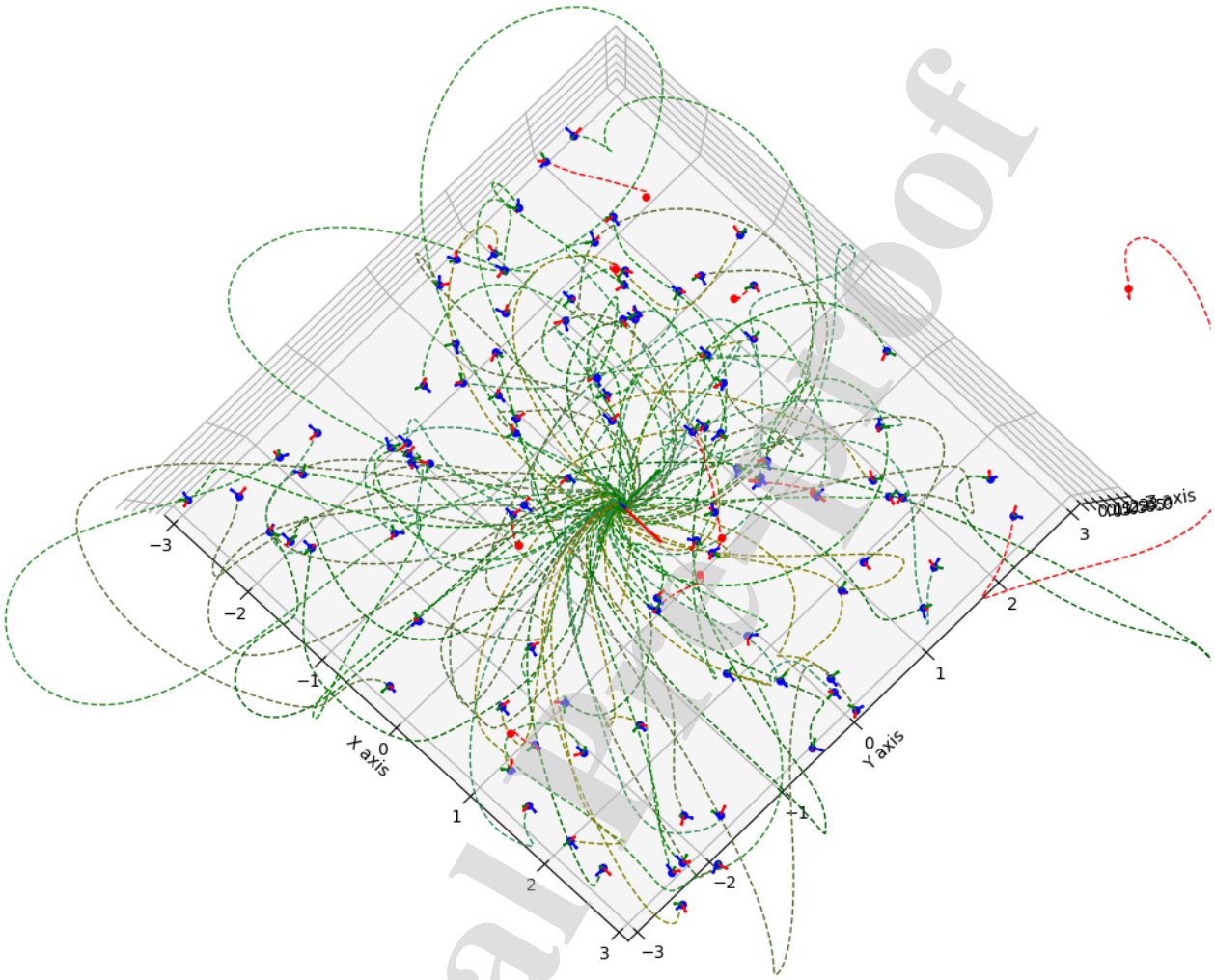


Figure 16: Hundred sample flight trajectory using E-PPO (JD-CSS-TDTS) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in starting step of each trajectory. (Top-view).

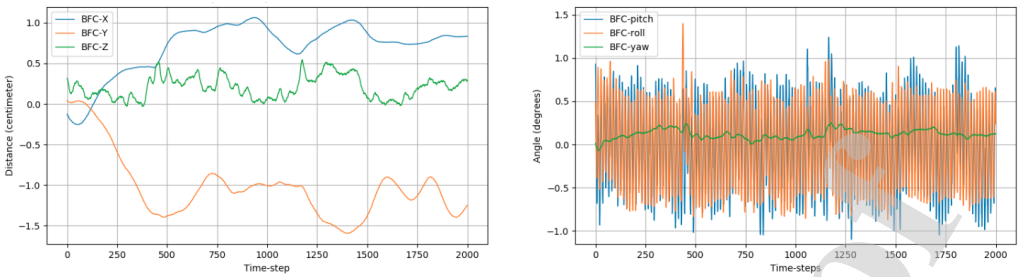


Figure 17: The accuracy of BFC in terms of position and attitude stabilization.

- First, we release the quad-rotor drone at a random point and attitude in space while the controller (BFC or linear-MPC) is active and set for the *desired_point* of [0,0,150] (centimeter).
- Then, we start gathering the position and attitude data and counting for 10 seconds while the flight controller frequency is 250 Hz.
- We consider a *desired_point_threshold* equal to 10 cm and define a $desired\_area = desired\_point \pm desired\_point\_threshold$. If the quad-rotors reach to the desired area and stay there for 1 second then we consider this as 'Success' for the controller, otherwise it will be considered as 'Fail'.

We performed this test for 100 random points (first, we generated 100 random points and then used this set for BFC and linear-MPC tests) and the result is shown in Figure 21. BFC acts much better in terms of stabilizing the desired attitude and position from any random starting attitude and position. Figure 19 and 20 show the detailed result of stabilizing attitude and position when starting from 10 random points using BFC and linear-MPC.

7.5. Following way points

Figure 22 shows BFC following way points, where the left image in the figure shows BFC following sparse way points (only four way points) and the right image in the Figure shows BFC following dense way points. Previously, we explained about the result of BFC stabilizing in a desired position (In the Accuracy section and Figure 17), but here we can observe the trajectory that BFC is choosing by itself to reach a way point. As Figure 22, this trajectory depends on the distance between two subsequent way points.

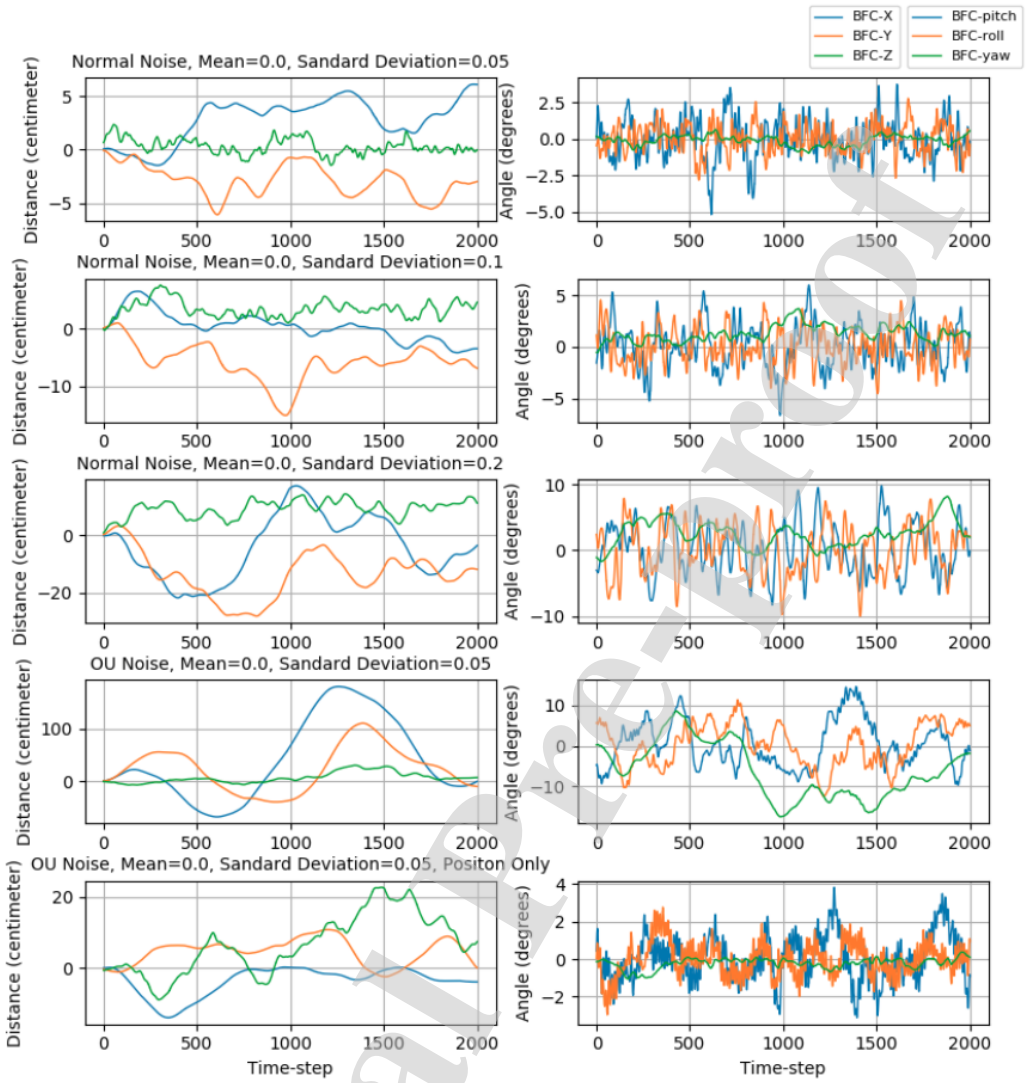


Figure 18: The effect of several degrees of random and OU noise on position and attitude stabilization accuracy of BFC.

8. Discussion

Animals learn to master their capabilities by trial and error which let them to reach the frontier of their capabilities. Unlike animals and the way they learn to control themselves, conventional controllers are rigid pre-designed systems which are designed based on the limited knowledge of their creators

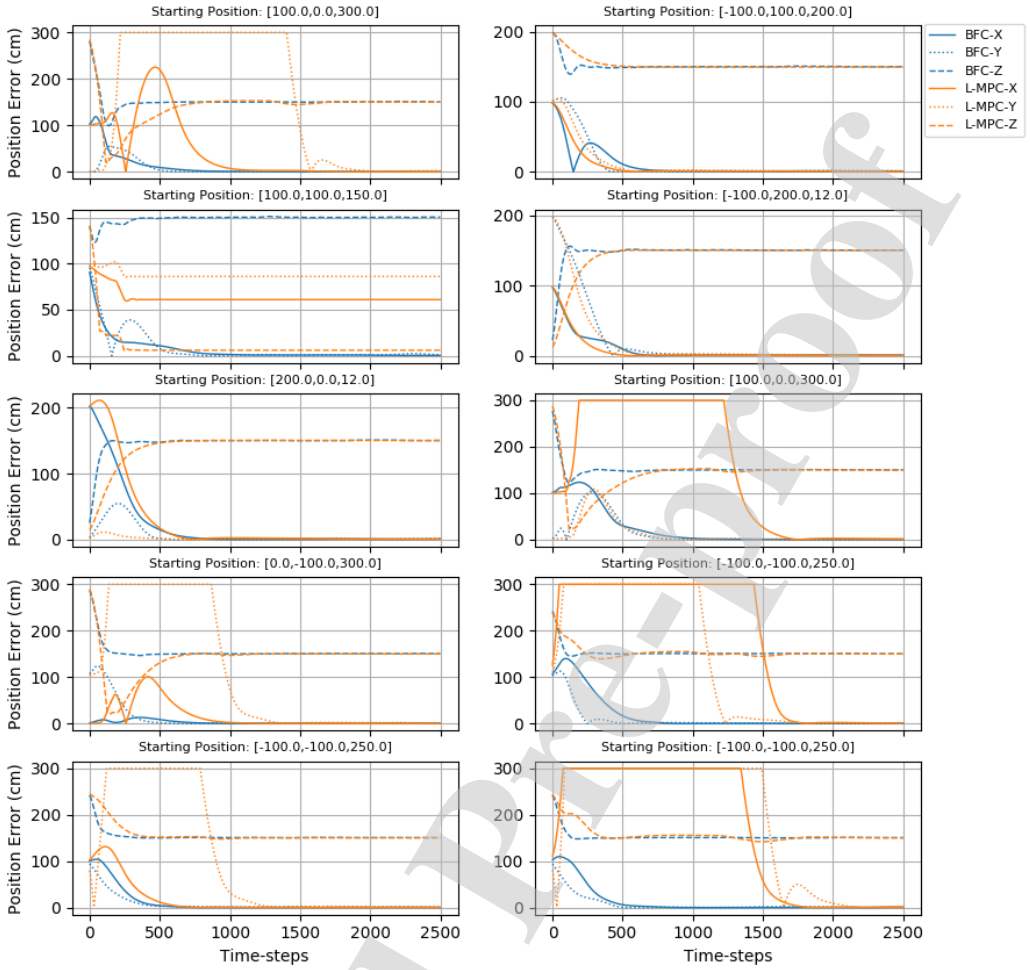


Figure 19: Comparing reaching to the desired position in BFC and linear-MPC in terms of the number of necessary time-steps, in top of each sub-plot there is the starting position of the quad-copter and the desired point for all the tests is [0.0, 0.0, 150.0] cm. (The position unit is centimeter). The x axis of the diagrams is time-step and each diagram shows a period of time equal to 10 seconds while the frequency of simulation is 250 Hz.

in terms of the system dynamics, mathematical and physical rules. Using reinforcement learning in a system instead of a conventional controller lets the system to discover possibilities that may not be recognized otherwise, and as a result can creates a more capable system, this argument is supported by our result in Figure 21 where BFC can stabilize the quad-rotor from initial

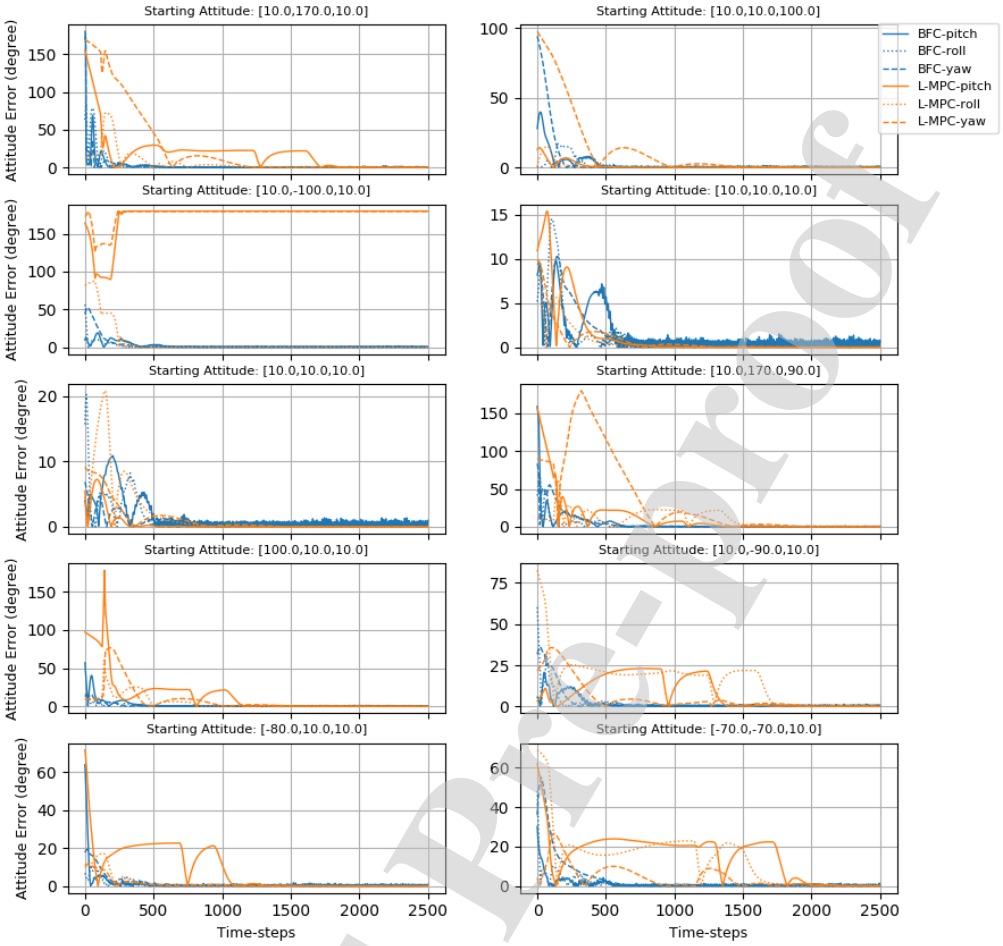


Figure 20: Comparing the attitude error of BFC and linear-MPC, in top of each sub-plot there is the starting attitude of the quad-copter and the desired attitude for all the tests is [0.0, 0.0, 0.0] cm. (The attitude unit is degree). The x axis of the diagrams is time-step and each diagram shows a period of time equal to 10 seconds while the frequency of simulation is 250 Hz.

attitude and positions that linear-MPC was not able to do.

BFC is a reinforcement learning based system which learns from itself and it does not need to know about the model of the system dynamics, but in case of conventional controllers such as MPC, LQR and SMC, we need to have the accurate model of the system dynamics which is very hard to obtain in the real world. There is also conventional controllers such as PID which

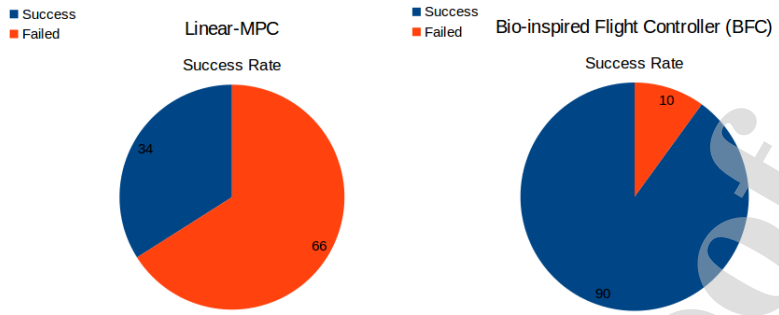


Figure 21: Comparison between BFC and Linear-MPC by initializing the quad-rotor attitude and position in 100 random position and attitude. The quad-rotor then need to stabilize itself in a desired position of [0, 0, 150] cm and attitude of [0, 0, 0] degree. It is also important to consider the real time uncertainties such as IMU noise. Even though we have tested our algorithm against noise in Section 7.3, but this test is performed without noise in order to focus on the flight controllers capabilities in terms of recovering from random positions and attitudes.

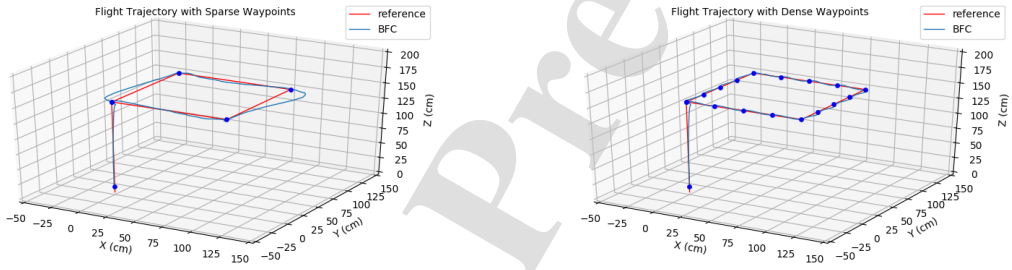


Figure 22: BFC following way points (sparse and dense way points).

are model-free but are not robust, and also they need to be tuned for PID gains.

Regarding the ablation study we performed in this research, and the paradox between our results shown in Figure 8, and Figure 9, and Figure 10 and Table 2 where JD-CSS-TDTS accumulate the lesser reward but has a higher performance, we explain in the following.

The main reason that JD has a higher average and accumulated reward comparing to JD-TDTS, JD-CSS, and JD-CSS-TDTS is the fact that it stays for a longer time in the global maxima, and as

a result gathers the maximum average and accumulated rewards. That is, the algorithm learns to fly to the desired point and stabilize the drone there. However, the lack of TDTS lets the drone to stay drone for the rest of the episode, and this will provide fewer samples for the algorithm regarding the recovering and take-off and flying to the desire position and more samples about stabilizing in the desired point. As a result, simple JD forgets what it learned about recovering and flight trajectory since it is focusing on stabilization because more samples are provided from its experiences over stabilization. In the case of JD-TDTS and JD-CSS-TDTS, the drone will be repositioned after hitting the TDT state, and as a result, it gathers lower reward in the long term; however, it gathers better experiences comparing to simple JD.
Despite that, using CSS in JD-CSS and JD-CSS-TDTS, slows the learning and reduce the accumulated reward, since it forces the algorithm to restart from different positions and especially those with lesser accumulated reward, however, as it can be seen in Table 2, it highly improves the performance of JD. All in all, adding TDTS and CSS to JD decreases the averaged and accumulated gathered reward as shown in Figure 8, Figure 9, and Figure 10, mainly because the algorithm is facing more experiences, however, it dramatically improves the algorithm accuracy and performance as shown in Figure 2.

9. Conclusion

In this work, we introduced a new flight controller called bio-inspired flight controller (BFC) by implementing some enhancements on an existing policy gradient based algorithm called PPO (a reinforcement learning based algorithm). We call this flight controller a bio-inspired flight controller because it learns to fly in a similar way to birds—by trial and error— and also the first enhancement is inspired by the nature. Our flight controller completely replaces *a conventional flight controller* and directly controls the engine velocities of the quad-rotor.

We used the physics engine of Gazebo and a simulated model of a real quad-copter (Ascending Technology Hummingbird) to implement BFC (enhanced PPO) and testing its learning capabilities. Our results show that our enhancements is crucial to make this reinforcement learning based flight

controller. Our results show the capabilities of BFC in terms of avoiding noise and following way points and having a better performance comparing to a linear-MPC controller when starting in a random initial attitude and position.

In the process of training or playing of the quad-rotor no human pilot or conventional controller is used. The artificial neural network we used for our paper is not pre-trained, and the learning process happens from scratch and autonomously.

In this paper, we used a well prepared simulated model of AscTec Hummingbird in Gazebo Sim. for training our drone, and even though we added noise to the system to create a realistic scenario, it is crucial to notice that transferring this trained model to a real drone is challenging and at the same time fascinating. Using BFC opens a new door to drone flight controllers where according to our results it can get a better maneuver capability. In terms of the necessary hardware for the implementation, it is important to consider implementation of the artificial neural network and BFC using C and C++ rather than Python and running it on micro-computers such as Nvidia Nano or Raspberry PI 4.

In future works, it is possible to implement the real system, where the difficult issues will be providing odometry data and the motor speed control.It is also possible to use information-theoretic approaches to increase the learning speed and capability.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NO.2019R1F1A1049711).

References

Abbeel, P., Coates, A., Quigley, M., Ng, A.Y., 2006. An application of reinforcement learning to aerobatic helicopter flight, in: Proceedings of the 19th International Conference on Neural Information Processing Systems, MIT Press, Cambridge, MA, USA. pp. 1–8. URL: http://dl.acm.org/citation.cfm?id=2976456.2976457.

AlphaZero-Stockfish, . Alphazero vs stockfish. URL: https://www.chess.com/news/view/updated-alphazero-crushes-stockfish-in-new-1-000-game-match.

Bengio, Y., Louradour, J., Collobert, R., Weston, J., 2009. Curriculum learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, Association for Computing Machinery, New York, NY, USA. p. 4148. URL: https://doi.org/10.1145/1553374.1553380, doi:10.1145/1553374.1553380.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., Zhokhov, P., 2017. Openai baselines. https://github.com/openai/baselines.

Dooraki, A.R., 2018. Experience, imitation and reflection; confucius' conjecture and machine learning. arXiv:1808.00222.

Dooraki, A.R., Lee, D., 2019. Multi-rotor robot learning to fly in a bio-inspired way using reinforcement learning, in: 2019 16th International Conference on Ubiquitous Robots (UR), pp. 118–123. doi:10.1109/URAI.2019.8768681.

Floreano, D., Mattiussi, C., 2008. Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies. The MIT Press.

Forestier, S., Mollard, Y., Caselli, D., Oudeyer, P.Y., 2016. Autonomous exploration, active learning and human guidance with open-source Poppy humanoid robot platform and Explauto library. The Thirtieth Annual Conference on Neural Information Processing Systems (NIPS 2016). URL: https://hal.inria.fr/hal-01404399. poster.

Furrer, F., Burri, M., Achtelik, M., Siegwart, R., 2016. Robot Operating System (ROS): The Complete Reference (Volume 1). Springer International

Publishing, Cham. chapter RotorS—A Modular Gazebo MAV Simulator Framework. pp. 595–625.

Gazebosim.org, . Gazebo simulator. URL: http://gazebosim.org/.

Hwangbo, J., Sa, I., Siegwart, R., Hutter, M., 2017. Control of a quadrotor with reinforcement learning. IEEE Robotics and Automation Letters 2, 2096–2103. doi:10.1109/LRA.2017.2720851.

Kakade, S., Langford, J., 2002. Approximately optimal approximate reinforcement learning, in: Proceedings of the Nineteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. pp. 267–274. URL: http://dl.acm.org/citation.cfm?id=645531.656005.

Kamel, M., Stastny, T., Alexis, K., Siegwart, R., . Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system, in: Koubaa, A. (Ed.), Robot Operating System (ROS) The Complete Reference, Volume 2. Springer.

Koch, W., Mancuso, R., West, R., Bestavros, A., 2019. Reinforcement learning for uav attitude control. ACM Trans. Cyber-Phys. Syst. 3, 22:1–22:21. URL: http://doi.acm.org/10.1145/3301273, doi:10.1145/3301273.

Kompella, V.R., Stollenga, M., Luciw, M., Schmidhuber, J., 2017. Continual curiosity-driven skill acquisition from high-dimensional video inputs for humanoid robots. Artificial Intelligence 247, 313 – 335. URL: http://www.sciencedirect.com/science/article/pii/S000437021500017X, doi:https://doi.org/10.1016/j.artint.2015.02.001. special Issue on AI and Robotics.

Lambert, N.O., Drew, D.S., Yaconelli, J., Calandra, R., Levine, S., Pister, K.S.J., 2019. Low level control of a quadrotor with deep model-based reinforcement learning. CoRR abs/1901.03737. URL: http://arxiv.org/abs/1901.03737, arXiv:1901.03737.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. CoRR abs/1509.02971. URL: http://arxiv.org/abs/1509.02971.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning, in: Balcan, M.F., Weinberger, K.Q. (Eds.), Proceedings of The 33rd International Conference on Machine Learning, PMLR, New York, New York, USA. pp. 1928–1937. URL: http://proceedings.mlr.press/v48/mniha16.html.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., et al., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.

Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E., 2006. Autonomous inverted helicopter flight via reinforcement learning, in: Ang, M.H., Khatib, O. (Eds.), Experimental Robotics IX, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 363–372.

Ramezani Dooraki, A., Lee, D.J., 2018. An end-to-end deep reinforcement learning-based intelligent agent capable of autonomous exploration in unknown environments. Sensors 18. URL: http://www.mdpi.com/1424-8220/18/10/3575, doi:10.3390/s18103575.

ROS.org, . ROS.org — powering the worlds robots. URL: http://www.ros.org/.

Sadeghi, F., Levine, S., 2016. (cad)$ˆ2$rl: Real single-image flight without a single real image. CoRR abs/1611.04201. URL: http://arxiv.org/abs/1611.04201, arXiv:1611.04201.

Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P., 2015. Trust region policy optimization. CoRR abs/1502.05477. URL: http://arxiv.org/abs/1502.05477, arXiv:1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. CoRR abs/1707.06347. URL: http://arxiv.org/abs/1707.06347, arXiv:1707.06347.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D., 2016.

Mastering the game of go with deep neural networks and tree search. Nature 529, 484–489. URL: http://dx.doi.org/10.1038/nature16961. article.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T.P., Simonyan, K., Hassabis, D., 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. CoRR abs/1712.01815. URL: http://arxiv.org/abs/1712.01815, arXiv:1712.01815.

stockfishchess.org, . Stockfish chess engine. URL: http://stockfishchess.org/.

Sutton, R.S., Barto, A.G., 1998. Introduction to Reinforcement Learning. 1st ed., MIT Press, Cambridge, MA, USA.

Zhang, T., Kahn, G., Levine, S., Abbeel, P., 2015. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. CoRR abs/1509.06791. URL: http://arxiv.org/abs/1509.06791, arXiv:1509.06791.

Appendix A. Flight Trajectories

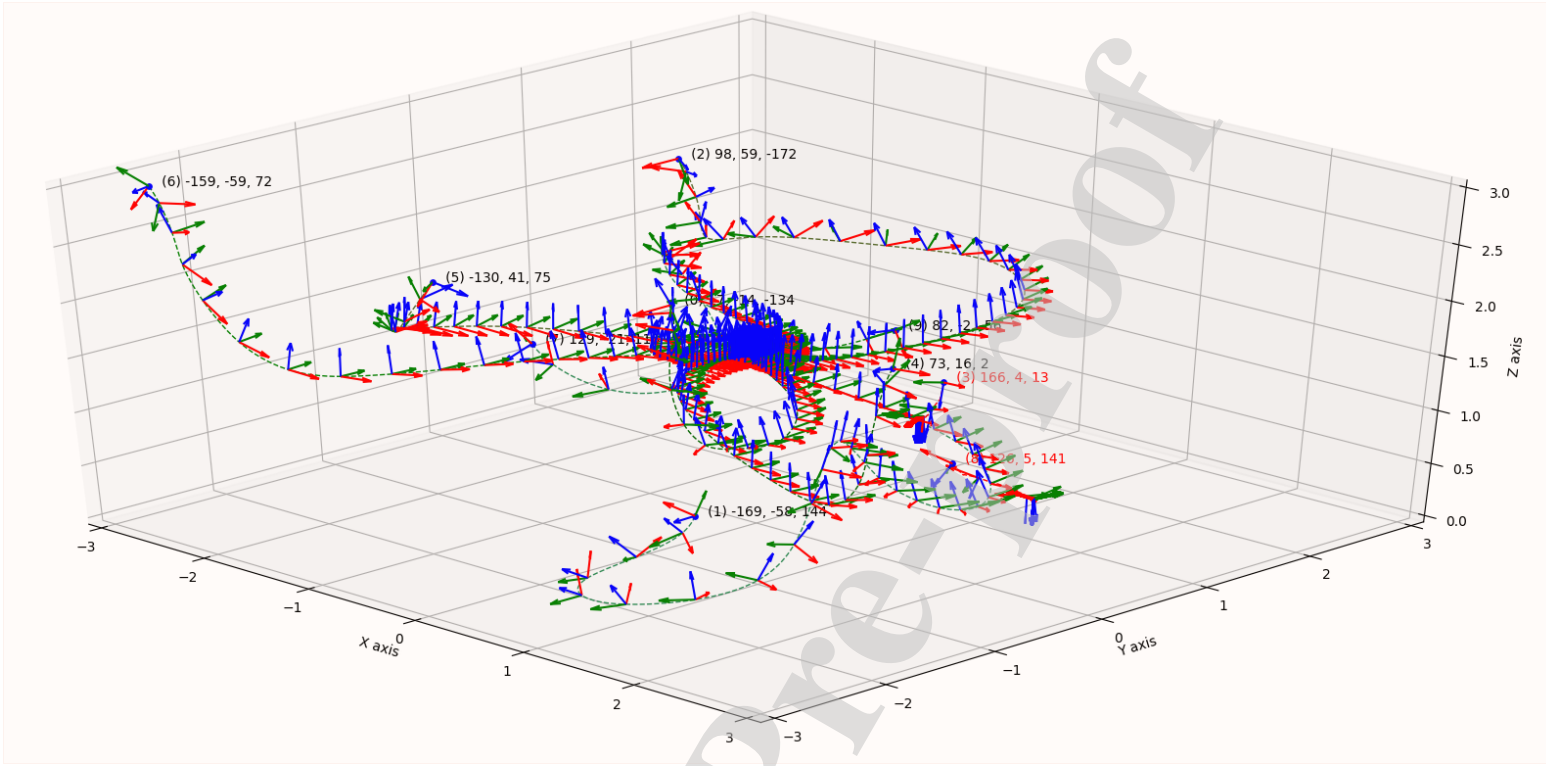


Figure A.23: Ten sample flight trajectory using E-PPO (JD-CSS-TDTS) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used in order to show the attitude of the drone in each step in each trajectory (3D).

References

Abbeel, P., Coates, A., Quigley, M., Ng, A.Y., 2006. An application of reinforcement learning to aerobatic helicopter flight, in: Proceedings of the 19th International Conference on Neural Information Processing Systems, MIT Press, Cambridge, MA, USA. pp. 1–8. URL: http://dl.acm.org/citation.cfm?id=2976456.2976457.

AlphaZero-Stockfish, . Alphazero vs stockfish. URL: https://www.chess.com/news/view/updated-alphazero-crushes-stockfish-in-new-1-000-game-match.

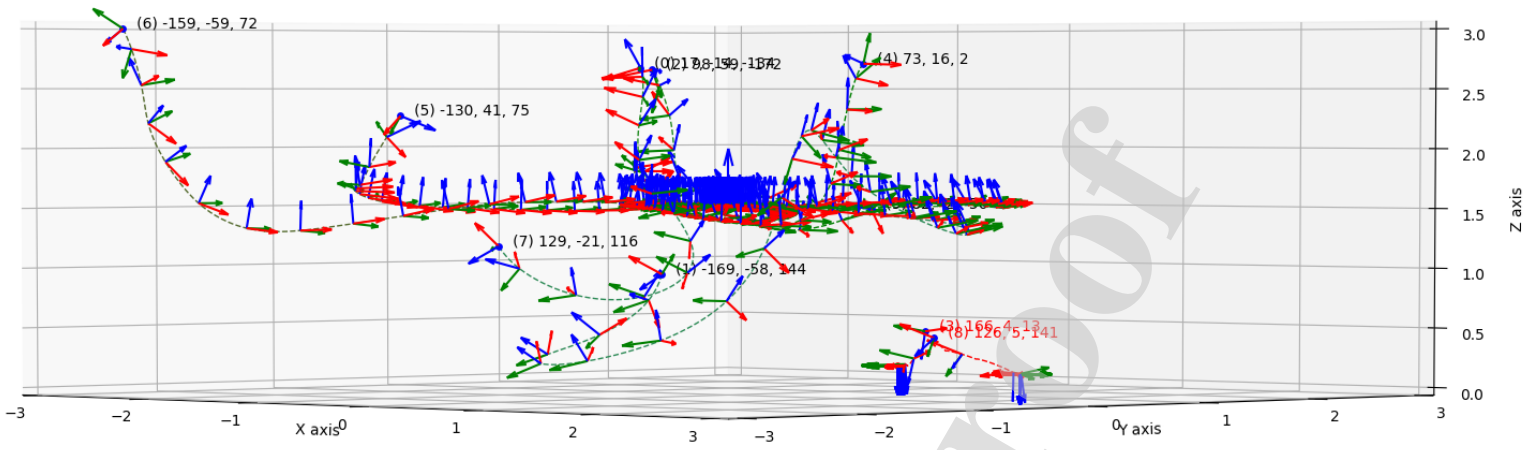


Figure A.24: Ten sample flight trajectory using E-PPO (JD-CSS-TDTS) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used in order to show the attitude of the drone in each step in each trajectory (side-view).

Bengio, Y., Louradour, J., Collobert, R., Weston, J., 2009. Curriculum learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, Association for Computing Machinery, New York, NY, USA. p. 4148. URL: https://doi.org/10.1145/1553374.1553380, doi:10.1145/1553374.1553380.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., Zhokhov, P., 2017. Openai baselines. https://github.com/openai/baselines.

Dooraki, A.R., 2018. Experience, imitation and reflection; confucius' conjecture and machine learning. arXiv:1808.00222.

Dooraki, A.R., Lee, D., 2019. Multi-rotor robot learning to fly in a bio-inspired way using reinforcement learning, in: 2019 16th International Conference on Ubiquitous Robots (UR), pp. 118–123. doi:10.1109/URAI.2019.8768681.

Floreano, D., Mattiussi, C., 2008. Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies. The MIT Press.

Forestier, S., Mollard, Y., Caselli, D., Oudeyer, P.Y., 2016. Autonomous exploration, active learning and human guidance with open-source Poppy humanoid robot platform and Explauto library. The Thirtieth Annual Conference on Neural Information Processing Systems (NIPS 2016). URL: https://hal.inria.fr/hal-01404399. poster.

Furrer, F., Burri, M., Achtelik, M., Siegwart, R., 2016. Robot Operating System (ROS): The Complete Reference (Volume 1). Springer International Publishing, Cham. chapter RotorS—A Modular Gazebo MAV Simulator Framework. pp. 595–625.

Gazebosim.org, . Gazebo simulator. URL: http://gazebosim.org/.

Hwangbo, J., Sa, I., Siegwart, R., Hutter, M., 2017. Control of a quadrotor with reinforcement learning. IEEE Robotics and Automation Letters 2, 2096–2103. doi:10.1109/LRA.2017.2720851.

Kakade, S., Langford, J., 2002. Approximately optimal approximate reinforcement learning, in: Proceedings of the Nineteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. pp. 267–274. URL: http://dl.acm.org/citation.cfm?id=645531.656005.

Kamel, M., Stastny, T., Alexis, K., Siegwart, R., . Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system, in: Koubaa, A. (Ed.), Robot Operating System (ROS) The Complete Reference, Volume 2. Springer.

Koch, W., Mancuso, R., West, R., Bestavros, A., 2019. Reinforcement learning for uav attitude control. ACM Trans. Cyber-Phys. Syst. 3, 22:1–22:21. URL: http://doi.acm.org/10.1145/3301273, doi:10.1145/3301273.

Kompella, V.R., Stollenga, M., Luciw, M., Schmidhuber, J., 2017. Continual curiosity-driven skill acquisition from high-dimensional video inputs for humanoid robots. Artificial Intelligence 247, 313 – 335. URL: http://www.sciencedirect.com/science/article/pii/S000437021500017X, doi:https://doi.org/10.1016/j.artint.2015.02.001. special Issue on AI and Robotics.

Lambert, N.O., Drew, D.S., Yaconelli, J., Calandra, R., Levine, S., Pister, K.S.J., 2019. Low level control of a quadrotor with deep

model-based reinforcement learning. CoRR abs/1901.03737. URL: http://arxiv.org/abs/1901.03737, arXiv:1901.03737.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. CoRR abs/1509.02971. URL: http://arxiv.org/abs/1509.02971.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning, in: Balcan, M.F., Weinberger, K.Q. (Eds.), Proceedings of The 33rd International Conference on Machine Learning, PMLR, New York, New York, USA. pp. 1928–1937. URL: http://proceedings.mlr.press/v48/mniha16.html.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., et al., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.

Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E., 2006. Autonomous inverted helicopter flight via reinforcement learning, in: Ang, M.H., Khatib, O. (Eds.), Experimental Robotics IX, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 363–372.

Ramezani Dooraki, A., Lee, D.J., 2018. An end-to-end deep reinforcement learning-based intelligent agent capable of autonomous exploration in unknown environments. Sensors 18. URL: http://www.mdpi.com/1424-8220/18/10/3575, doi:10.3390/s18103575.

ROS.org, . ROS.org — powering the worlds robots. URL: http://www.ros.org/.

Sadeghi, F., Levine, S., 2016. (cad)$ˆ2$rl: Real single-image flight without a single real image. CoRR abs/1611.04201. URL: http://arxiv.org/abs/1611.04201, arXiv:1611.04201.

Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P., 2015. Trust region policy optimization. CoRR abs/1502.05477. URL: http://arxiv.org/abs/1502.05477, arXiv:1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. CoRR abs/1707.06347. URL: http://arxiv.org/abs/1707.06347, arXiv:1707.06347.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D., 2016. Mastering the game of go with deep neural networks and tree search. Nature 529, 484–489. URL: http://dx.doi.org/10.1038/nature16961. article.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T.P., Simonyan, K., Hassabis, D., 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. CoRR abs/1712.01815. URL: http://arxiv.org/abs/1712.01815, arXiv:1712.01815.

stockfishchess.org, . Stockfish chess engine. URL: http://stockfishchess.org/.

Sutton, R.S., Barto, A.G., 1998. Introduction to Reinforcement Learning. 1st ed., MIT Press, Cambridge, MA, USA.

Zhang, T., Kahn, G., Levine, S., Abbeel, P., 2015. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. CoRR abs/1509.06791. URL: http://arxiv.org/abs/1509.06791, arXiv:1509.06791.

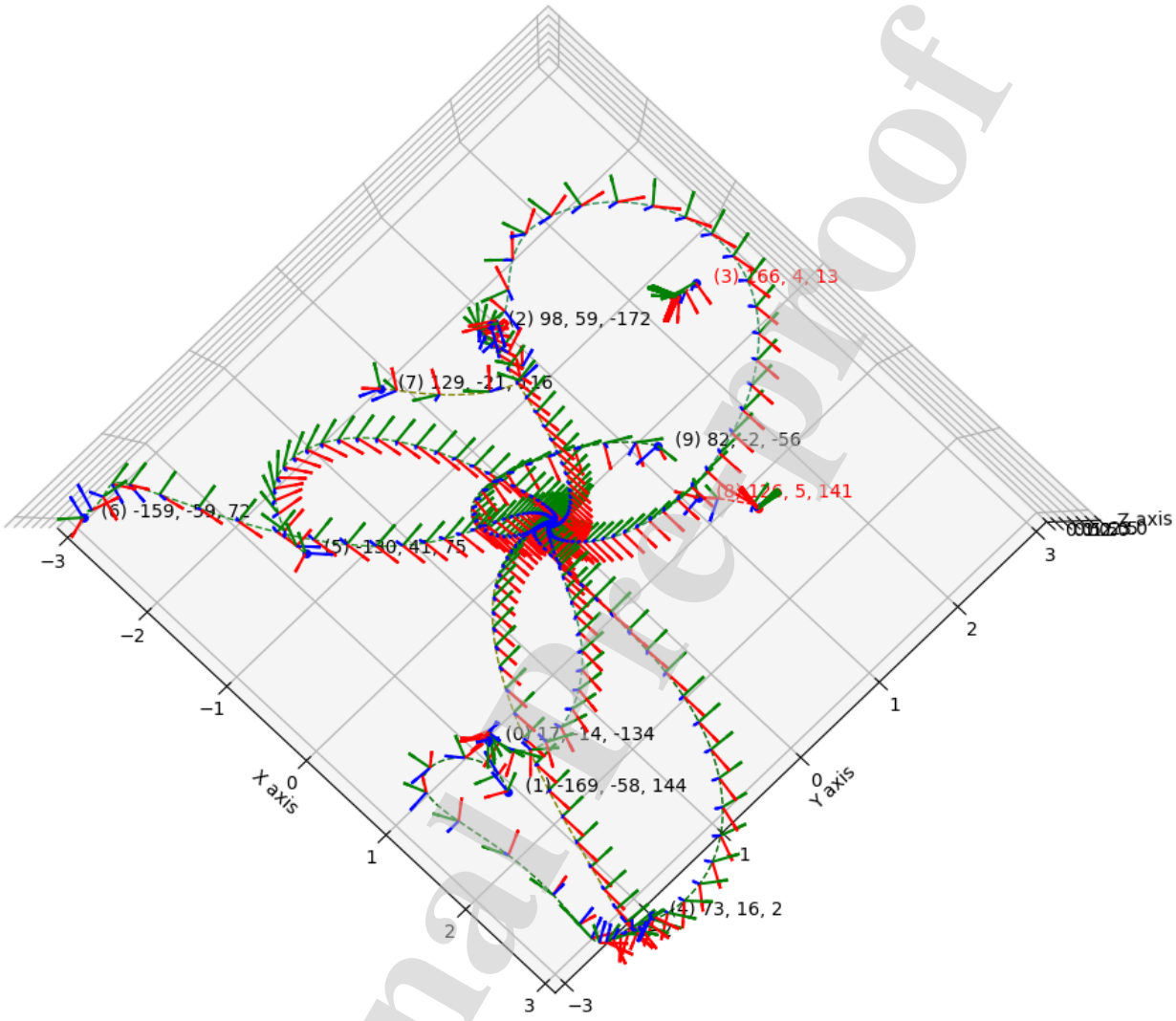


Figure A.25: Ten sample flight trajectory using E-PPO (JD-CSS-TDTS) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used in order to show the attitude of the drone in each step in each trajectory (top-view).

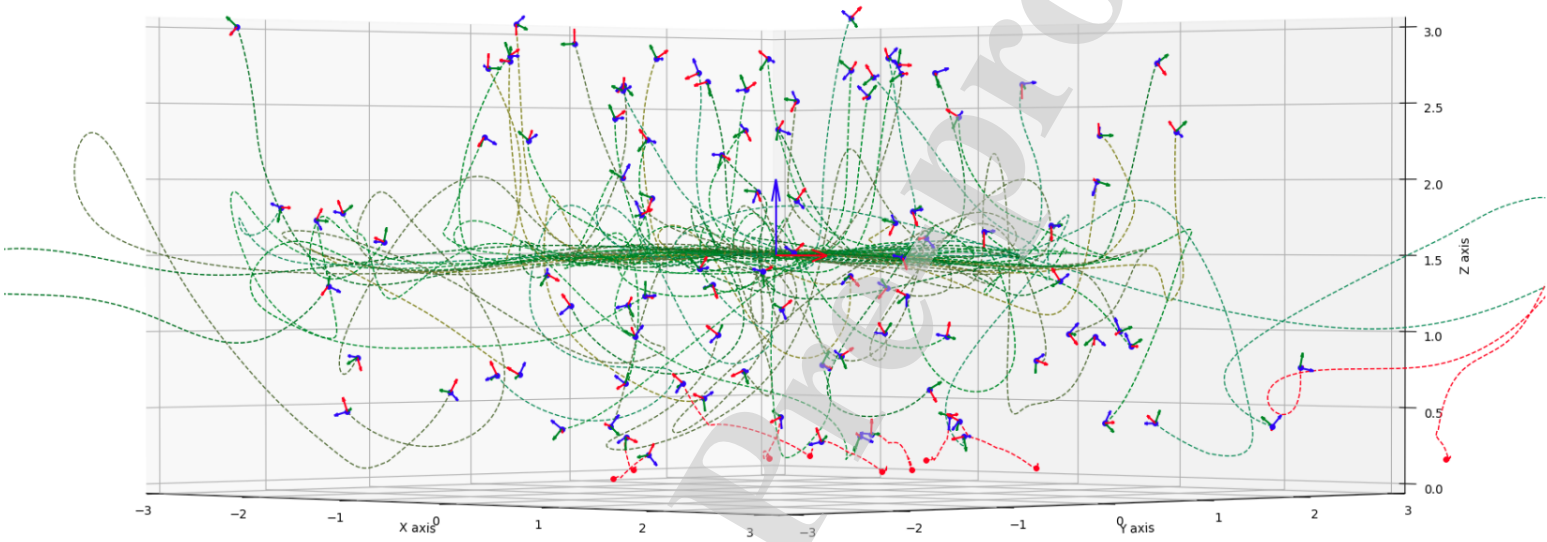


Figure A.26: Hundred sample flight trajectory using E-PPO (JD-CSS-TDTS) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in starting step of each trajectory (side-view).

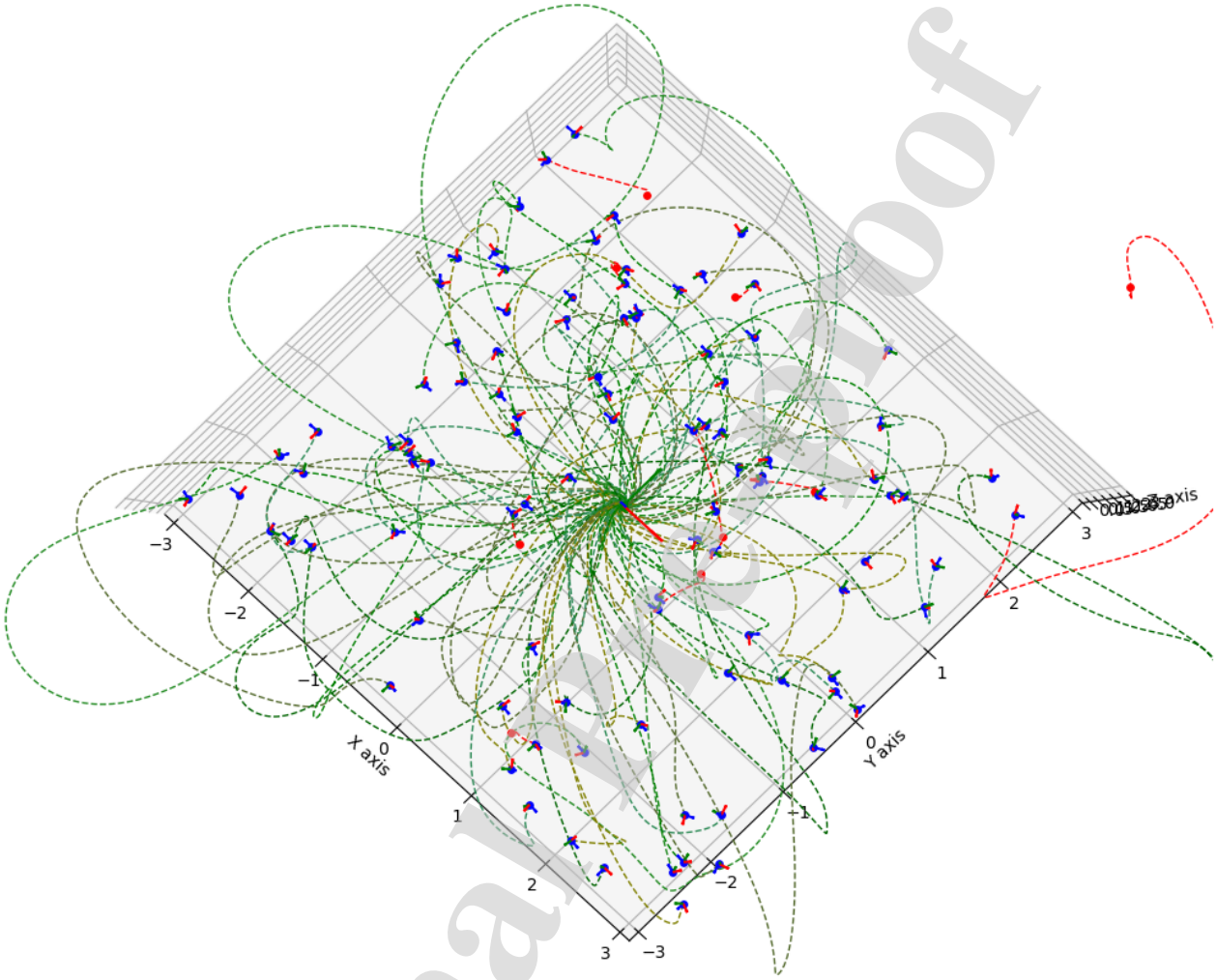


Figure A.27: Hundred sample flight trajectory using E-PPO (JD-CSS-TDTS) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in starting step of each trajectory. (top-view).

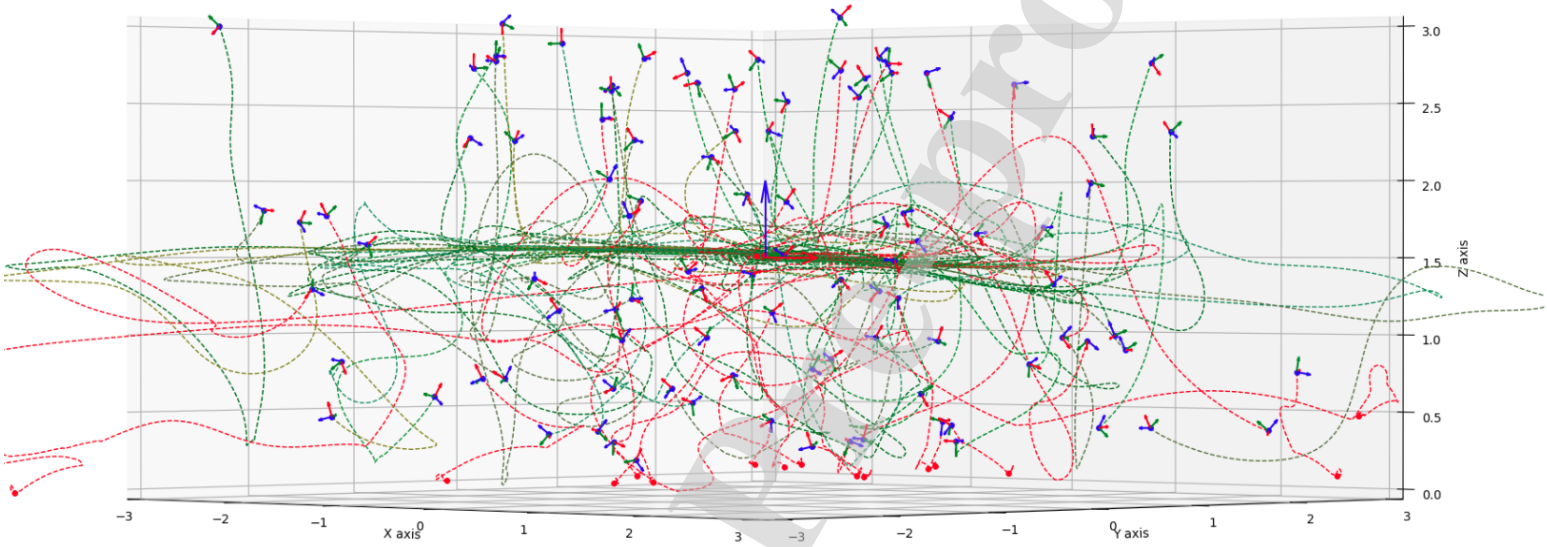


Figure A.28: Hundred sample flight trajectory using E-PPO (JD-CSS) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in starting step of each trajectory (side-view).

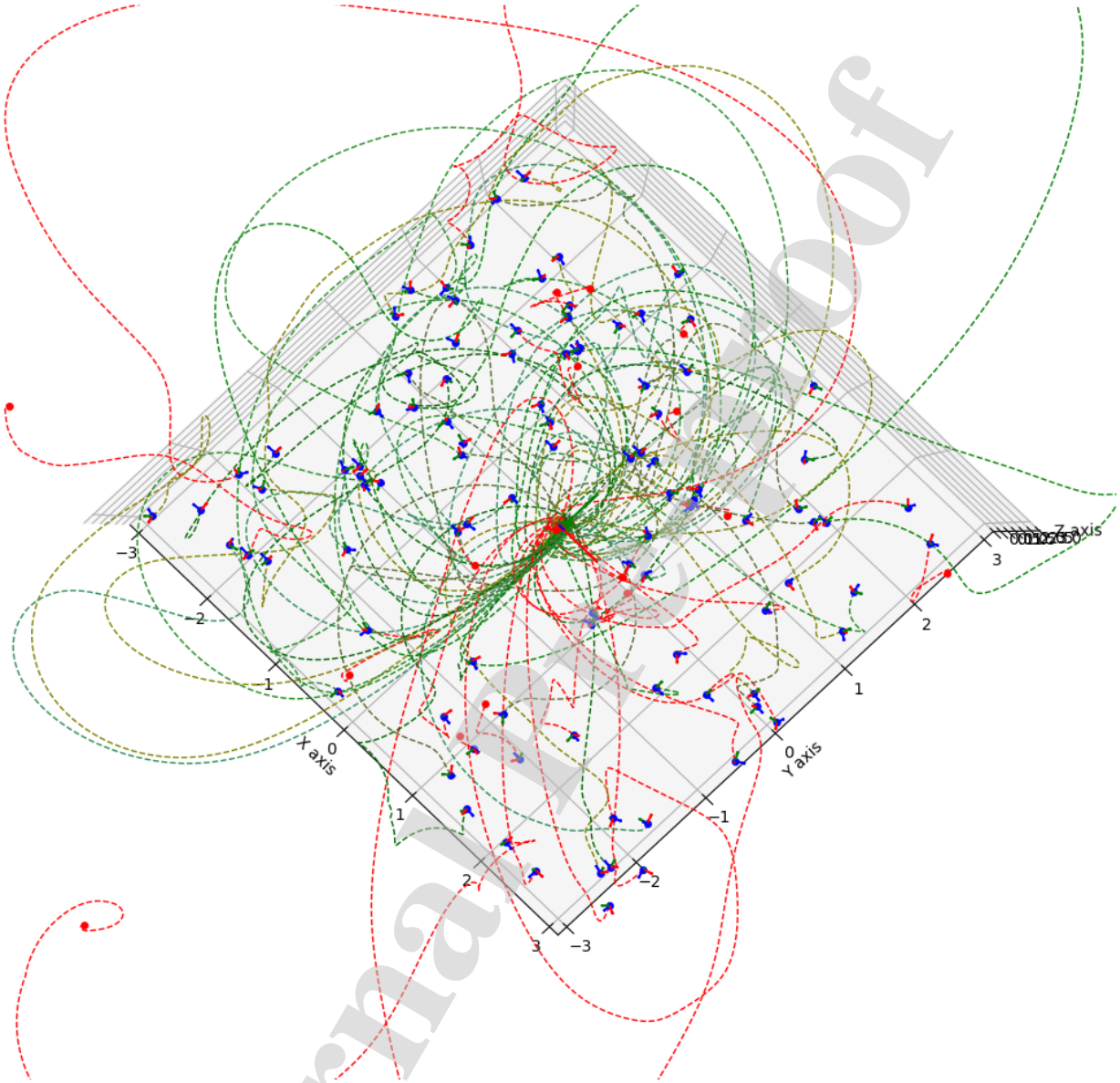


Figure A.29: Hundred sample flight trajectory using E-PPO (JD-CSS) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in starting step of each trajectory. (top-view).

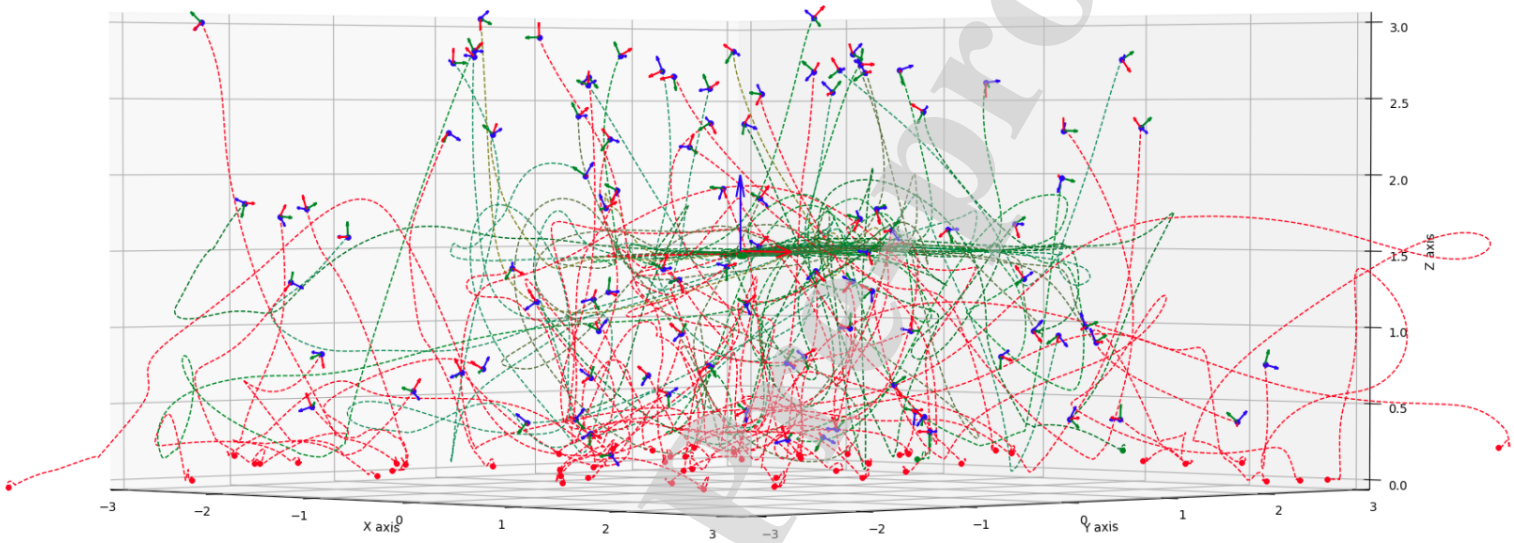


Figure A.30: Hundred sample flight trajectory using E-PPO (JD-TDTS) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in starting step of each trajectory (side-view).

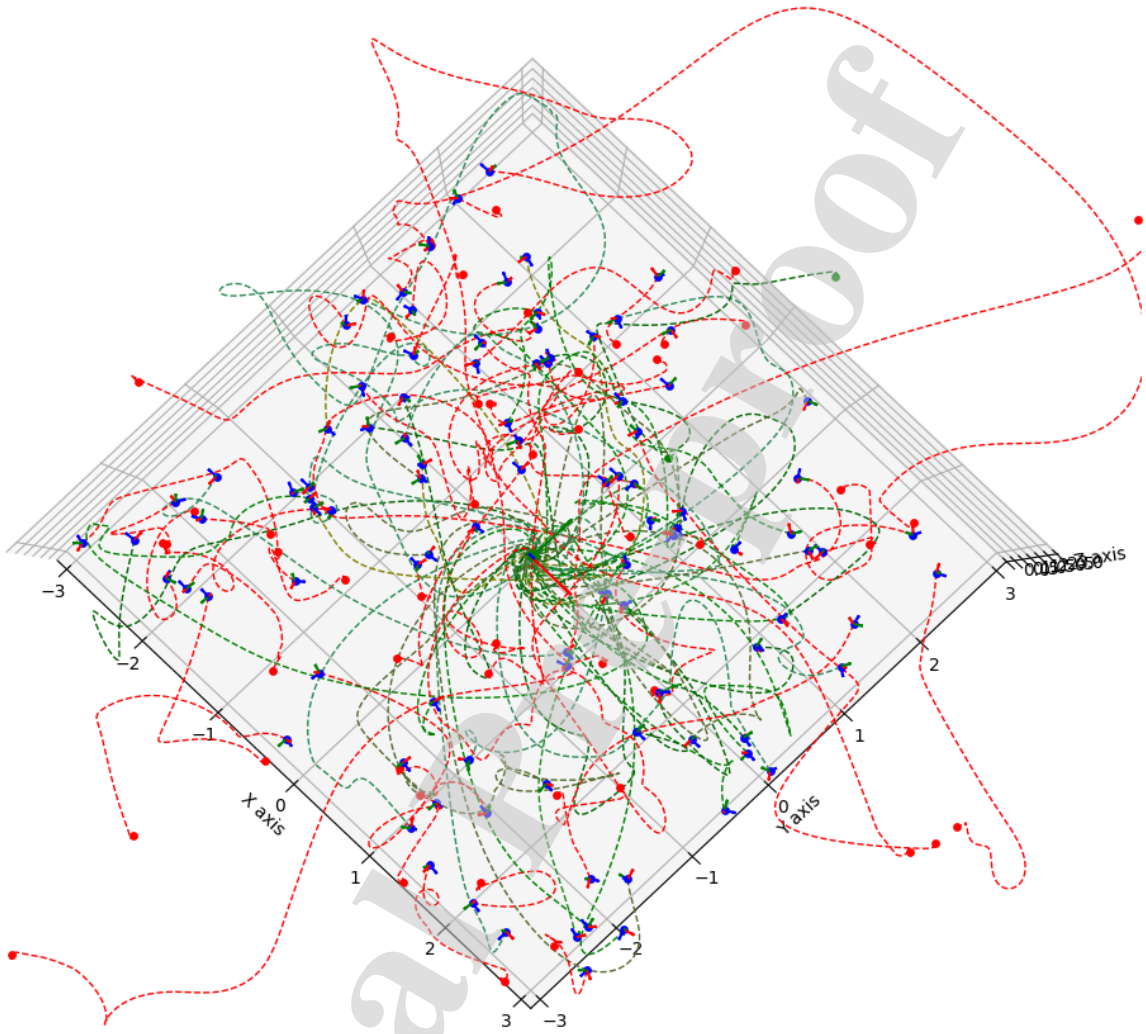


Figure A.31: Hundred sample flight trajectory using E-PPO (JD-TDTS) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in starting step of each trajectory. (top-view).

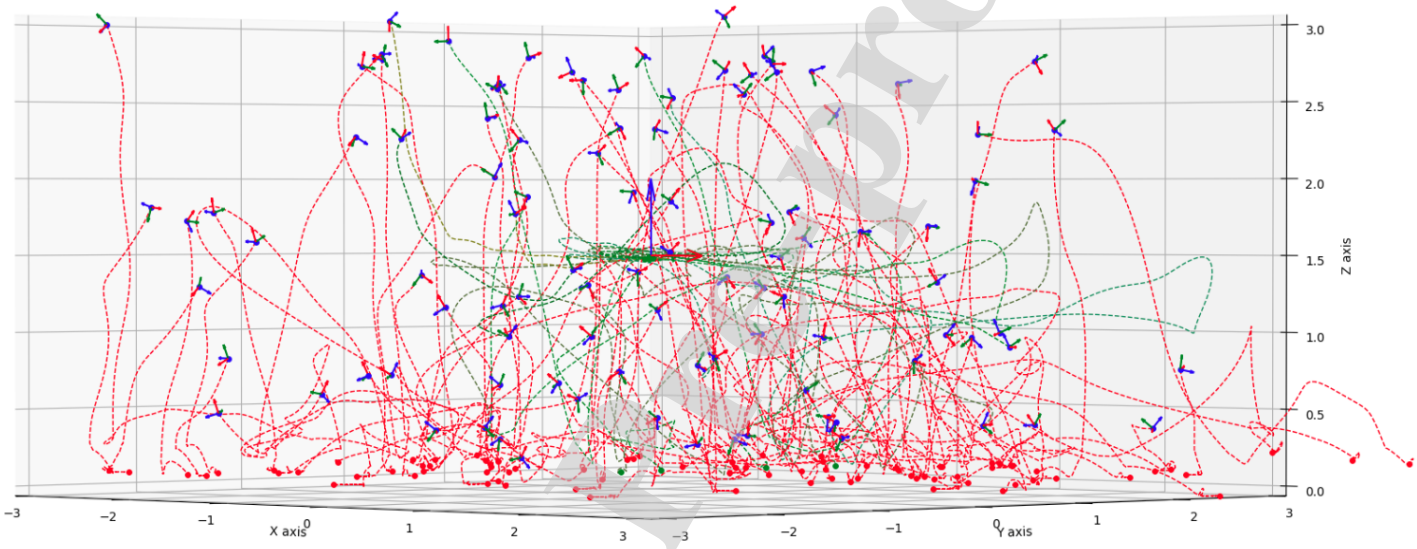


Figure A.32: Hundred sample flight trajectory using E-PPO (JD) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in starting step of each trajectory (side-view).

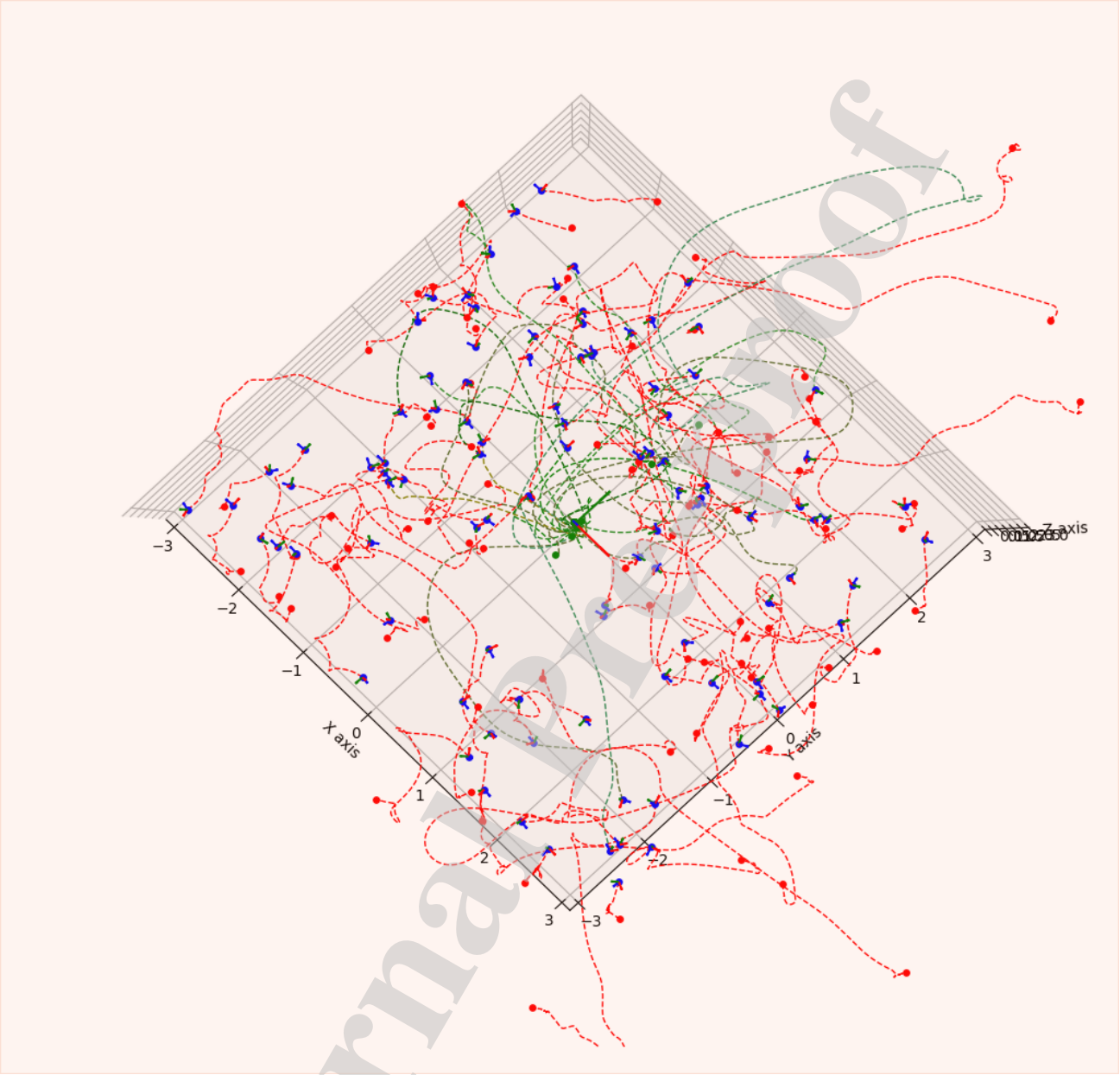


Figure A.33: Hundred sample flight trajectory using E-PPO (JD) where the quad-copter is started at random position and attitude, and flew to the desired point using BFC. A right-hand coordinate system is used to show the attitude of the drone in starting step of each trajectory. (top-view).

Highlights:

- Intelligent machines are capable of learning to master their capabilities by trial and error.
- Learning to control the quad-copter by having no knowledge regarding its mathematical or physical models.
- Reinforcement learning based algorithms are capable of creating intelligent systems that act autonomously.
- Quad-rotor autonomously learns maneuvers that surpasses traditional flight controllers in some cases.
- A system that learns to control itself is not limited to the imagination of a traditional controller designer.

Amir Ramezani Dooraki received his M.Sc. in software engineering from Asia Pacific University of Innovation and Technology in 2013. He worked for the Quatriz Co. in the area of computer vision and machine learning from 2013 to 2015. He started his Ph.D. in the field of mechanical engineering in 2016, where he is doing research in the areas of reinforcement learning, deep learning, bio-inspired artificial intelligence and autonomous systems.

Deok-Jin Lee received the Ph.D. degrees in Aerospace Engineering from Texas A\&M University in May 2005. He worked for Agency for Defense Development (ADD) from 2006 to 2007, and Korean Air R\&D Center from 2009 to 2011. He was also a research professor at the Center for Autonomous Vehicle Research (CAVR), Naval Postgraduate School, Monterey, CA, U.S.A. Currently, he is an associate professor at the School of Mechanical & Automotive Engineering, and also the director of the center of Artificial Intelligence & Autonomous Systems(CAIAS) in Kunsan National University, South Korea. His research interests include intelligent autonomous systems, robotics learning, deep reinforcement learning, sensor fusion, adaptive estimation and control, integrated navigation, and multi-agent control.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: