Vision-Based Drone Flight Report

Tian Yi Lim
ETH Zurich
tialim@ethz.ch

Hanyu Wu
ETH Zurich
hanywu@ethz.ch

Feng Tian
ETH Zurich
fetian@ethz.ch

Tingyu Wang
University of Zurich
tingyu.wang@uzh.ch

***Abstract*—Vision-based drone flight is a challenging task due to the low cost and common nature of camera-equipped quadrotor drones. In this project, we developed a reinforcement learning based control policy to allow a camera drone to track another drone in 3D. Our policy was deployed in a real-world setting. In this report, we demonstrate the robustness and performance of our approach through simulations and real-world test analyses.**

***Index Terms*—reinforcement learning, autonomous robots, drone flight, control, object tracking**

I. Introduction

With the widespread use of camera-equipped quadrotor drones, a common use-case is visually tracking an object of interest. While previous works consider tracking an object on the ground 2D plane [1, 6], it is also interesting to track another object in 3D.

This report details the design and implementation of a Deep Reinforcement Learning (RL) policy that controls a quadrotor drone to visually track an actor drone. Two versions of the policy are detailed: One with global position information of the actor drone, and another which takes the bounding box of the actor drone as detected by an external image-based detector as input. Both policies have access to the full state information of the ego- (or "camera") drone.

A. Related Work

1) Deep Reinforcement Learning: Deep RL has demonstrated significant success in various control tasks, such as drone racing [4, 11] and quadrupedal locomotion [9]. Among the many RL algorithms, Proximal Policy Optimization (PPO) [10] has gained widespread popularity due to its stable performance and simple implementation. PPO operates within the actor-critic framework [5], where the actor network receives the state space $\mathbf{s} \in \mathbb{R}^N$ as input and outputs actions $\mathbf{a} \in \mathbb{R}^M$. Meanwhile, the critic network estimates the value function $V(\mathbf{s})$ of the given states. During evaluation, the actor network $\boldsymbol{\pi}(\mathbf{s})$ is deployed, and a forward pass through this network determines the control policy.

2) Quadrotor Simulation and Deployment: These advancements in Deep RL are impressive, but would not be able to translate to real-world deployment without a realistic simulator. The *Flightmare* [12] simulator is tailored towards autonomous, agile quadrotor flight. This enables the RL policy trained in simulation to transfer to the real world without a crippling domain gap – without which, sim-to-real transfer would not be possible. Furthermore, the *Agilicious* [2] framework allows for convenient deployment of trained policies in simulated and real-world environments.

3) Agile Quadrotor Flight: Quadrotor control is an interesting use-case for Deep RL approaches, as "classical" model-based control methods like Model Predictive Control may be difficult to apply. This is due to the underactuated nature of the platform and its highly nonlinear dynamics. Furthermore, it is challenging to accurately identify model parameters, especially in extreme scenarios.

Song et al. [11] presents a Deep RL approach based on PPO for time-optimal trajectory planning in autonomous drone racing. The focus is on overcoming computational challenges of traditional optimization-based approaches and achieving real-time flight at high speeds.

Closer to the subject of agile control of quadrotor drones, [4] details how it is possible to races drones competitively against human world champions with a Deep RL controller. The paper introduces Swift, an advanced drone system comprises a perception system that processes visual and inertial data to estimate the drone's states and a control policy implemented via Deep RL that commands the drone's maneuvers based on perceived environmental states. The model is trained in simulation environments with empirical data augmentation from real-world settings to handle real-time physical constraints and unpredictable conditions typical of drone racing.

II. Task Requirements

Our team is tasked with developing a reinforcement learning-based control system that uses the state of the camera drone and the bounding box of the target drone to track it in real-time while maintaining a specified distance ($1\mathrm{m}$ in our state-based tracking setup and $1.5\mathrm{m}$ in the vision-based tracking setup). The system is launched with ROS [13], read bounding box data from a ROS topic, and publish control commands, ensuring operation at $60\mathrm{Hz}$.

III. General Methodology

This section details the process of training the control policy to fulfill task requirements. As a stepping stone towards full *vision-based* tracking, a *state-based* tracking policy was developed. In vision-based tracking, the only observation of the actor drone are its bounding box in normalized image coordinates. In state-based tracking, the problem is simplified as the state of the actor drone in a world reference system is known. This is much easier, as the control policy has knowledge of the position of the actor drone even if the actor drone is not within its field of view.

In both state and vision-based tracking, the actor network is a $[N, 256, 256, 4]$ multi-layer perceptron (MLP) network with input dimension N as the dimension of observations, and output dimension 4 equal to the dimension of the actions.

The Flightmare [12] simulator was used to train policies for quadrotor control using reinforcement learning and to test their performance in complex 3D environments. The Agilicious [2] framework was used for the deployment of our policy, realistically showing how the policy would track changes in the position of an actor drone in simulation. Additionally, it could use "rosbags" captured from real-world environments to further test and validate the policy's tracking performance. A modified Proximal Policy Optimization (PPO) network [10] from *Stable Baselines 3* [8] was used.

For both state- and vision-based tracking, we identify the following objectives for the policies.

- **Tracking:** The camera drone should keep the actor drone within the field of view of its camera.
- **Smoothness:** The camera drone should avoid excessively jerky movements, to allow for smooth footage of the actor drone to be captured, as well as to preserve the hardware.
- **Safety:** The camera drone should not crash into the environment boundaries or the actor drone.

In addition, both tracking tasks have the same action space. The action $\mathbf{a}_t \in \mathbb{R}^4$ at time t contains body rates $\mathbf{a}_t^{\omega} \in \mathbb{R}^3$ and mass-normalized collective thrust $\mathbf{a}_t^{\mathbf{T}} \in \mathbb{R}$.

IV. State-based Tracking

A. Observations

The observation $\mathbf{o}_t \in \mathbb{R}^{25}$ at time step t used in state-based tracking consists of:

1) States $\mathbf{s}_t^{cam} \in \mathbb{R}^{15}$, which are: The position $p \in \mathbb{R}^3$, orientation $\theta \in \mathbb{R}^6$, linear velocity $v \in \mathbb{R}^3$, and angular velocity $\omega \in \mathbb{R}^3$ of the camera drone in the world frame. Inspired by [15], the first two rows of the rotation matrix are used to represent the camera drone orientation to reduce redundancy.
2) Position $\mathbf{p}_t^{actor} \in \mathbb{R}^3$ of the actor drone in the world frame at the current time step.
3) Position $\mathbf{p}_{t-1}^{actor} \in \mathbb{R}^3$ of the actor drone in the world frame at the previous time step.
4) Action $\mathbf{a}_{t-1} \in \mathbb{R}^4$ applied in the previous time step.

All observations are normalized before being passed to the network.

B. Rewards

Our reward function r_t at time t is in the form:

$$r_t = r_t^{track} + r_t^{dist} + r_t^{smooth} + r_t^{crash} + r_t^{exist}$$

1) Tracking Task: For the *tracking* task to be achieved, the following rewards are set:

Firstly, r^{track} is a positive reward that encourages the camera drone to face the actor drone. To encode this, the angle between the position vector of the actor drone in the camera drone's frame ${}_{cam}\mathbf{r}_{cam,act}$ and the x- basis vector $\mathbf{e}_x^{cam}$ is computed. This angle will be labelled $\theta_{cam,act}$.

$\theta_{cam,act}$ is then converted into a smooth positive reward by the *scaled Gaussian* function, $\exp(-(x-\mu)^2 \cdot c)$. This formulation is similar to the Gaussian probability density function, except that it has a maximum value of 1 where $x = \mu$, making it more intuitive to scale relative to the other rewards.

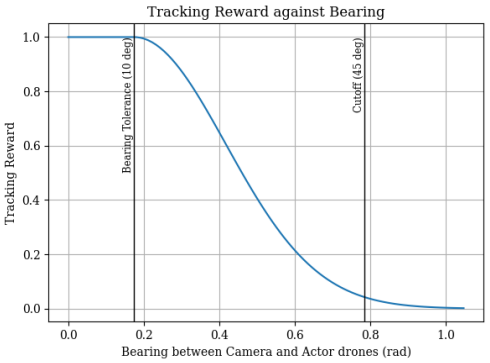


Fig. 1: r^{track} against Bearing

Figure 1 shows how the tracking reward changes as $\theta_{cam,act}$ varies. If it is within the bearing tolerance $\mu = 10°$, it takes its maximum value of 1. This allows the attitude of the camera drone to vary slightly, so long it is pointed roughly at the actor drone. This is important as the underactuated nature of a quadrotor drone means that in order for it to translate, its orientation has to change. This "play" in r^{track} enables the policy not to conflict between maximizing r^{track} and r^{dist}. Furthermore, the scale of the Gaussian $c = 8.5$ is chosen such that at roughly $45°$, the policy no longer gets any tracking reward.

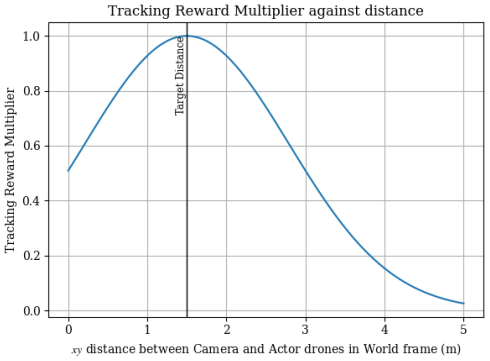


Fig. 2: r^{track} Multiplier against xy-Distance

In addition, r_{track} is multiplied by the xy-distance between the camera and actor drones in the world frame. As the bearing between drones does not encode distance information, r_{track} may return a high value even when the camera drone is far away from the actor drone. This may cause the policy to reach a local minima in training, where the camera drone merely points towards the actor drone without moving towards it.

To prevent this, r_{track} is further multiplied by another scaled Gaussian, centred around a target distance of 1.5m. Thereby, the maximum reward of r_{track} is only assigned when the camera drone is appropriately far from the camera drone.

Secondly, r^{dist} is split into two sub-rewards: $r^{dist_{xy}}$ and r^{dist_z}. These are the distances between the camera and actor drones in the *world* frame, or in other words, their lateral distances and their difference in height.

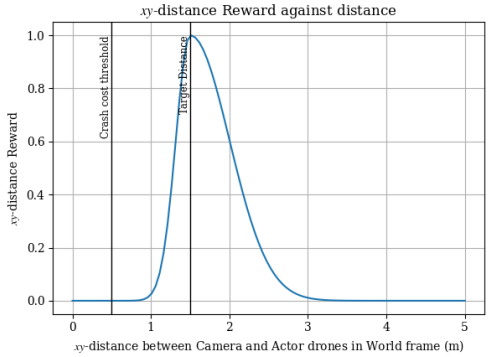


Fig. 3: $r^{dist_{xy}}$ against Distance

Figure 3 shows how $r^{dist_{xy}}$ varies. It is an asymmetric scaled Gaussian with different coefficients depending on whether the drone is closer or further from the target tracking distance 1.5 m (the same as specified in fig. 2). This gives the policy a strong signal not to be too close to the actor drone for safety. However, there is a wider tolerance of acceptable xy-distances if the camera drone is not too close to the target drone. Therefore, the Gaussian is set with a shallower gradient. This also gives the policy a better learning signal to follow.

In particular, the parameters for the scaled Gaussians are $\mu_{near} = 1.5$, $c_{near} = 15.0$, $\mu_{far} = 1.5$, $c_{far} = 2.0$.

r^{dist_z} is a scaled Gaussian on the difference in z-distance between the two drones in world frame. It encourages the camera drone to remain at the same height as the actor drone. While r_{track} does implicitly encode this requirement, we find that providing r^{dist_z} gives the policy a "hint" to first hover near the actor drone in the first few epochs of training.

2) Smoothness Task: r_t^{smooth} encodes the requirement of the policy to control the camera drone smoothly. It is implemented as negative rewards for deviating from the requirements. These are inspired by those from [4]. This serves to regularize the policy, discouraging "bang-bang" policies which oscillate wildly. These are needed as the positive rewards in section IV-B1 do not explicitly discourage this behaviour.

Firstly, we penalize the deviation from level hovering, as r_t^{hover}. This is the sum of $r_t^{hover,\omega}$, the 2-norm of the current body-rate command $||\mathbf{a}_t^{\omega}||_2$, and $r_t^{hover,\mathbf{T}}$, the magnitude of the difference between collective thrust and g, $|g - \mathbf{a}_t^{\mathbf{T}}|$.

Secondly, we penalize large differences between consecutive control commands as $r_t^{\Delta\mathbf{a}}$. For body-rates, $r_t^{\Delta\mathbf{a},\omega}$ is the 2-norm of the difference in consecutive body-rate commands $||\mathbf{a}_t^{\omega} - \mathbf{a}_{t-1}^{\omega}||_2$.

For collective thrust, the cost $r_t^{\Delta\mathbf{a},\mathbf{T}}$ is a clipped exponential. The exponential term is $e = (\exp(\Delta\mathbf{T} \cdot c_2) - 1) \cdot c_1$, where $\Delta\mathbf{T}$ is the difference in thrust. $c_2 = 8.0$ scales the slope of the exponential, while $c_1 = 0.05$ performs overall scaling and shifting such that the cost is never negative and equals to zero when $\Delta\mathbf{T} = 0$. Lastly, the exponential term e is clipped as $\min(e, 5.0)$ to prevent any exploding gradients and to allow the policy to initially learn how to fly stably. In this way, large changes in collective thrust are heavily penalized.

To conclude this section:

$$r_t^{smooth} = -r_t^{hover,\omega} - r_t^{hover,\mathbf{T}} - r_t^{\Delta\mathbf{a},\omega} - r_t^{\Delta\mathbf{a},\mathbf{T}}$$

3) Safety Task:

$$r_t^{crash} = \begin{cases} -5.0, & \text{if exceeds the safety boundary} \\ 0, & \text{otherwise} \end{cases}$$

Lastly, to avoid drone crash, we define a safety reward. This reward is only active when the camera drone is outside a safety boundary or is too close to the actor drone. Due to the presence of environmental randomness, aligning the safety boundary with the map boundary is insufficient to ensure that the drone will not crash. The safety margin is 0.1m. If the camera drone's distance from the map boundary is less than this, the drone will receive a large negative reward, which "reminds" the drone to maintain a safe distance from the environment boundaries and the actor drone, thereby reducing crashes.

4) Existence Reward: Initially, the policy weights are randomized, and the policy does not know how to avoid crashing. It hence may only accumulate the negative rewards in section IV-B2 and section IV-B3, without any ability to accumulate the positive rewards in section IV-B1. These grow larger as the episode length increases. The policy may then choose to enter a terminal state as quickly as possible and settle into a local minima, which avoids the accumulation of negative reward as much as possible. However, this prevents the policy from learning anything "useful".

To prevent this, a reward r^{exist} is given for each time step where the policy is not in the terminal state. This aids in stabilizing training, as the policy is incentivized to stay in non-terminal states for longer to continue accumulating r^{exist}, which then allows it to maximize the positive rewards in section IV-B1.

	r_t^{track}	$r_t^{dist_{xy}}$	$r_t^{dist_z}$	r_t^{smooth}	r_t^{exist}	r_t^{crash}
weight	0.15	0.15	0.04	0.02	0.05	1.0

TABLE I: Reward Table with Different Weights

5) Relative Reward Weights: Table I shows how each reward is weighted relative to others. The weighted sum of the rewards is used to train the value function.

C. Training

We train our agent across 200 environments in parallel, achieving over 50,000 transitions per second using an i7-14700HX CPU. To ensure the agent can handle diverse scenarios, we initialize the states of both the camera and actor drones

randomly, including their positions and orientations within the arena boundary. The actor drone's trajectory is generated randomly using cubic splines, with its speed uniformly sampled from the range of 0 to $1ms^{-1}$.

A key challenge in RL is addressing the sim-to-real gap. To mitigate this, we adopt domain randomization, where we randomize key dynamics of the camera drone—such as drag force, moment of inertia, mass, and thrust—making the control policy more robust to differences between simulation and real-world environments. We train the agent for 2,000 iterations (100 million steps).

V. Vision-based Tracking

As mentioned in section III, the vision-based task is much harder than the state-based one. Ideally, the policy should know what to do when the actor drone is in the field of view, and when the actor drone is not. Furthermore, a camera model needs to be implemented to simulate the bounding-box detections.

A. Double Sphere Camera Model

The double sphere model [14] is a camera model commonly used for fish-eye lenses. It describes how points in 3D are projected to the 2D image plane while taking distortion into account for large field-of-view lenses. It is computationally efficient, allowing for closed-form projection and unprojection, along with unified projection and distortion correction.

There are six parameters for the double sphere model, $f_x, f_y, c_x, c_y, \xi, \alpha$. The pixel coordinates u, v are:

$$u = f_x \frac{x}{\alpha d_2 + (1-\alpha)(\xi d_1 + z)} + c_x$$

$$v = f_y \frac{x}{\alpha d_2 + (1-\alpha)(\xi d_1 + z)} + c_y$$

Where x, y, z is the point in 3D, d_1 is the distance from the camera center to the 3D point, d_2 adjusts the distance by introducing the distortion parameter ξ from the Double Sphere Model.

$$d_1 = \sqrt{(x^2 + y^2 + z^2)}$$

$$d_2 = \sqrt{(x^2 + y^2 + (\xi d1 + z)^2)}$$

Double sphere model is used to generate training data. By using the actor drone's known size, we get the coordinates of the bounding cuboid's vertices, transform them into the camera frame, and project them onto the 2D plane, obtaining their u, v coordinates. These 2D points are then used to calculate the drone's bounding box in the image for training purposes.

B. Observations

For vision-based tracking, the observation $\mathbf{o}_t \in \mathbb{R}^{43}$ includes:

1) States $\mathbf{s}_t^{cam} \in \mathbb{R}^{15}$, which are the same as those used in state-based tracking.
2) Action $\mathbf{a}_{t-1} \in \mathbb{R}^4$ applied in the previous time step.
3) Action $\mathbf{a}_{t-2} \in \mathbb{R}^4$ applied two time steps ago. We utilize one more historical action information to address the increased difficulty of vision-based tasks.
4) Bounding box parameters $\{\mathbf{b}_i \in \mathbb{R}^5\}_{i=t-3}^{t}$ from the past 3 time steps and the current time step. Each bounding box consists of four position parameters $\{u_{min}, v_{min}, u_{max}, v_{max}\}$, and a binary indicator $\sigma \in \{1, -1\}$. If a bounding box is detected, $\sigma = 1$; otherwise, $\sigma = -1$.

Similar to state-based tracking, these observations are normalized before being passed to the network as well.

C. Rewards

The rewards for vision-based tracking are the same as for state-based tracking, as the underlying task is the same.

D. Training

Initially, we used the same training settings as in state-based tracking. However, we found that the agent struggled to learn an effective policy. This may be due to the completely random initialization of the position and orientation, where the actor drone often ended up outside the camera drone's FoV. As a result, the camera drone was unable to learn how to locate and track the actor drone.

To address this issue, we designed an easier task, considering also that the detection module is likely to fail if the distance to the actor drone is too large in real world. In the modified task, the actor drone is always initialized at the center of the arena, while the camera drone's initial position is set at a distance and yaw difference sampled from the ranges $(0.25\text{m}, 2.5\text{m})$ and $(-0.25\text{rad}, 0.25\text{rad})$, respectively. This ensures that the camera drone can consistently keep the actor drone within its FoV.

In addition to the domain randomization applied in state-based tracking, we also introduced jitter to the bounding box in the observation by adding random noise in the range of $(-0.005, 0.005)$ to the normalized coordinates of the bounding box. This further enhances the robustness of the tracking policy. For this task, we trained the policy over 1,000 iterations, after which the training environment was set to the one previously described in section IV-C.

VI. Experimental Results

Trained policies were first evaluated in a simulated scenario using Agilicious. If the policy displayed adequate performance, it was deployed in a real-world scenario.

A. State-Based Tracking

Our state-based policy was deployed in a real-world scenario. Here, we show the output of the policy in several different situations[1].

1) Tracking a stationary actor drone: Figure 4 shows how the position of the camera drone varies as it moves toward a stationary actor drone far away. The camera drone quickly moves to a target distance to the actor, and settles to its desired position without oscillations. This policy was trained with

[1]Videos of the policy can be seen at https://youtu.be/OHuGVk8cu0M

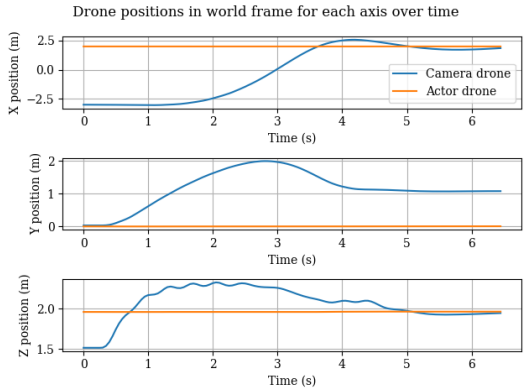


Fig. 4: Positions against time tracking a stationary actor drone

$r^{dist_{xy}}$ centred around 1.0m. In later vision-based task, the target distance was increased to 1.5m for safety.

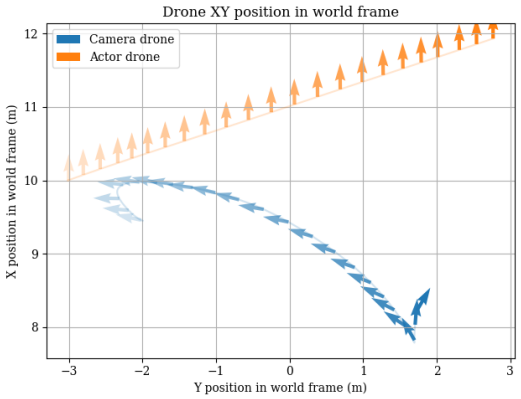


Fig. 5: xy pose against time tracking a moving actor drone

2) Tracking a moving actor drone: Figure 5 shows the camera drone move to track a moving actor drone. The initial pose of both the camera and actor drones have the highest opacity, and the opacity decreases as time increases.

Initially, the actor drone moves perpendicular to the camera drone, so the camera drone simply turns around the yaw axis, keeping the actor drone in its FoV. However, as the actor drone moves further away beyond the target distance, the camera drone moves towards the actor to close the gap.

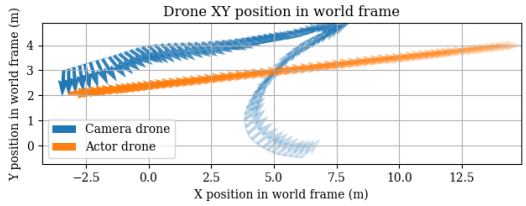


Fig. 6: xy pose against time tracking while avoiding boundaries

3) Tracking while avoiding boundaries: Figure 6 shows the camera drone tracking while avoiding the boundary of the arena at $y = 5$. The camera drone initially attempts to track the actor to the actor's left, but soon reaches the boundary. The policy has learned to avoid crashing, and therefore switches to the actor's right and resumes tracking from there.

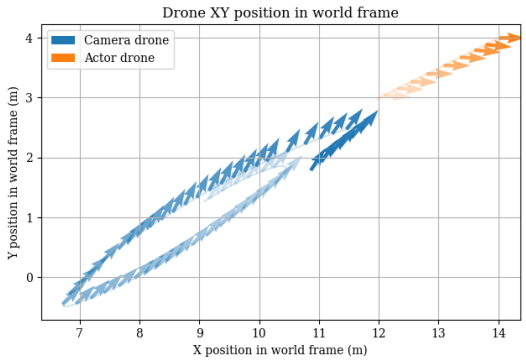


Fig. 7: xy pose against time tracking despite collisions

4) Robustness to mid-air collision: Figure 7 shows the policy's robustness to mid-air collisions. In this sequence, the actor drone moved directly towards the camera drone. Unable to avoid the collision directly, the camera drone was bumped mid-air by the actor. Surprisingly, the policy was able to recover from this and resume tracking.

B. Vision-Based Tracking

Though our vision-based policy was deployed in a real-world experiment, our group was unable to ensure that the inputs were normalized between $(-1, 1)$. Therefore, we only report results on the policy based on training output.

Figure 8 shows training metrics from *Tensorboard*. Before reaching 1000 iterations (50 million steps), the agent successfully learns the easy task. The mean episode length is around 1500 out of 1800 steps, showing that the agent survives. The mean episode reward keeps increasing, showing the agent's performance increases.

However, when the environment is completely randomly initialized, as in state-based tracking, the agent's performance noticeably declines.

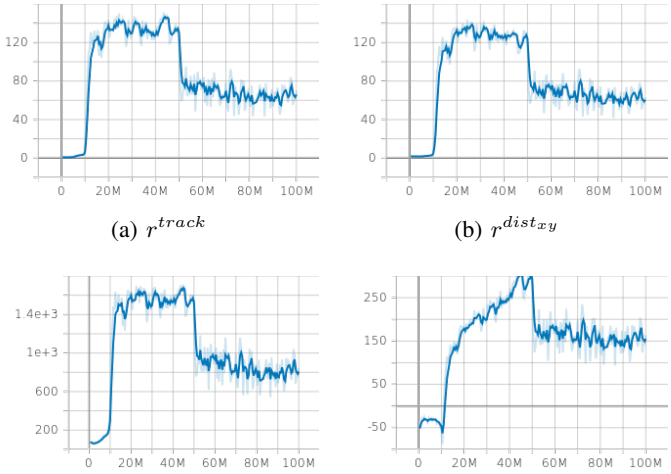
Figure 9 shows an evaluation from *Flightplot* at iteration 1000. We can see that the camera drone tracks the actor drone successfully. As mentioned earlier, after 1000 iterations, performance degrades.

VII. Discussion

A. Tradeoffs between rewards

It can be seen that the rewards described in section IV-B are somewhat conflicting. A few examples that were observed are enumerated:

1) Any movement may need a change in thrust, which may then result in changing z height. To maintain a steady height, the policy may end up rapidly "toggling" thrust on and off. This needs to be regularized with r^{smooth}.
2) To move closer to the actor drone and maximize r^{dist}, the attitude of the camera drone must change. This then



(a) r^{track} (b) $r^{dist_{xy}}$

(c) Episode length (d) Episode reward

Fig. 8: Plots of mean training metrics against iteration count

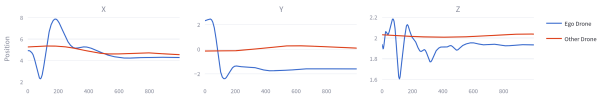


Fig. 9: Evaluation of tracking performance in *Flightplot*

results in temporarily less r^{track}. The policy may choose to simply rotate to face the actor drone and not move towards it.

3) While useful as an initial training signal, the policy may settle into a local minimum, only maximizing r^{dist_z}. Simply matching the height of the actor drone will minimize smoothness and crashing costs.
4) For vision-based tracking, r^{dist} may confuse the policy if the actor drone is not in FoV, as the relative pose of the actor drone is unobservable. Especially for r^{dist_z}, the policy may instead learn to simply hover in the middle of the arena at $z = 2.0$, or the mean initial z position of the actor drone seen during training.

B. Observation Choice

The observations we chose did not include accelerations. This meant that the policy had to infer our r^{smooth} penalties in part through the difference in previous control commands.

C. Curriculum Choice

The implemented curriculum in section V-D immediately jumped to a very difficult scenario based on the iteration count. However, this meant that the curriculum was made more difficult before the policy had "converged", which resulted in training diverging. In the future, we could implement curriculum advancement with heuristics based on episode length (policy does not prematurely crash) and on reward (policy is not still learning from current improvement).

VIII. Conclusion and Future Work

In conclusion, a state- and vision-based tracking policy was developed. The state-based policy achieved good success, showing safe and performant results in real-world trials. On the other hand, there was insufficient time to adequately train a vision-based policy, though our training statistics displayed promising results.

While an interesting example, the assumption of externally-provided state estimation is likely unrealistic. Future work can incorporate on-board state estimation, for instance from a Visual-Inertial odometry (VIO) source, e.g. SVO [3] or VINS [7]. This would require the modelling of odometry drift in the simulator as well.

Another interesting extension would be to perform obstacle avoidance while tracking an object of interest. This is because real-world environments are unlikely to be as free of obstacles as the scenario described in this report. If done purely visually, this is a highly non-trivial task, requiring several extensions:

- Differentiation of the object of interest from other environmental obstacles, perhaps with instance segmentation.
- Determining free space around the camera drone for manoeuvre, perhaps with monocular depth estimation.
- Determining the motion of all objects in the scene, perhaps with segmentation + optical flow.

References

[1] Sarthak Bhagat and PB Sujit. "UAV target tracking in urban environments using deep reinforcement learning". In: *2020 International conference on unmanned aircraft systems (ICUAS)*. IEEE. 2020, pp. 694–701.

[2] Philipp Foehn et al. "Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight". In: *Science robotics* 7.67 (2022), eabl6259.

[3] Christian Forster et al. "SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems". In: *IEEE Trans. Robot.* 33.2 (2017), pp. 249–265. DOI: 10.1109/TRO.2016.2623335.

[4] Elia Kaufmann et al. "Champion-level drone racing using deep reinforcement learning". In: *Nature* 620.7976 (2023), pp. 982–987.

[5] Vijay Konda and Vijaymohan Gao. "Actor-critic algorithms". In: (Jan. 2000).

[6] Sedat Ozer et al. "Visual object tracking in drone images with deep reinforcement learning". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 10082–10089.

[7] Tong Qin, Peiliang Li, and Shaojie Shen. "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator". In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 1004–1020.

[8] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[9] Nikita Rudin et al. "Learning to walk in minutes using massively parallel deep reinforcement learning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 91–100.

[10] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

[11] Yunlong Song et al. "Autonomous Drone Racing with Deep Reinforcement Learning". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 1205–1212. DOI: 10.1109/IROS51168.2021.9636053.

[12] Yunlong Song et al. "Flightmare: A flexible quadrotor simulator". In: *Conference on Robot Learning*. PMLR. 2021, pp. 1147–1157.

[13] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. Version ROS Melodic Morenia. May 23, 2018. URL: https://www.ros.org.

[14] Vladyslav Usenko, Nikolaus Demmel, and Daniel Cremers. "The double sphere camera model". In: *2018 International Conference on 3D Vision (3DV)*. IEEE. 2018, pp. 552–560.

[15] Yi Zhou et al. "On the Continuity of Rotation Representations in Neural Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.