**Fyra**
Mishari Aliesa
Pushpreet Singh Hanspal
Amy Puente
Shilpa Thomas

# Fyra: Quadrotor Path Following in Simulation

**28th April 2020**

## TABLE OF CONTENTS

## INTRODUCTION

This semester, we sought to create a robust and generalized quadrotor control policy that allows a simulated quadrotor to follow a trajectory in a near-optimal manner.  We focused on developing a unified control policy that controls a quadrotor both low level (mapping environment states to rotor thrusts) and high level (allowing the quadrotor to effectively follow a trajectory).  Our approach utilized reinforcement learning and an OpenAI Gym simulation environment.

## PROJECT GOALS

1. Explore different learning methods, including Proximal Policy Optimization, Deep Deterministic Policy Gradient, curriculum learning and supervised learning.
2. Learn a unified quadrotor control policy, which controls the vehicle both on a high and low level.  On a low level, the controller should stabilize the quadrotor and map the current state to rotor thrusts. Additionally, it should guide the quadrotor through a given trajectory in a near-optimal manner.

## RELATED WORK

### Machine Learning for Quadrotor Control

Machine learning and reinforcement learning in particular have widely been applied to the task of quadrotor control. However, many previous applications of machine learning to the problem of quadrotor control policy generation rely on the presence of previously-tuned low-level controllers or treat the generation of a low-level controller as a separate task.  In Hwangbo et. al. [1] a PD controller is used during the learning process. Sadeghi et al. [2] learns a high-level control policy, which makes use of a separately tuned low-level controller.  Kang et al. [3] utilizes real world data to train a control subsystem, while separately learning a high-level yaw control policy.  We seek to eliminate the need for a separately tuned low-level controller, instead focusing on the generation  of a unified policy network as a single learning task.

### Reward Shaping in Reinforcement Learning

Reinforcement learning has extensively and long been used in robotics and has been shown to be well-suited to problems with continuous state and action spaces, Hwangbo et. al. [1]. However, developing an effective reward function which leads to the desired behavior for robotic control and manipulation tasks proves challenging.  Tasks with sparse reward settings, in which the agent receives a reward only upon reaching a goal state, may prevent the agent from ever learning the desired behavior.  On the other hand, shaping a reward function in a way that effectively allows the agent to learn the desired behavior involves considerable human effort and experimentation Ng et al. [4].

## Alternatives to Reward Shaping

In curriculum-based learning, an agent first learns simpler scenarios and gradually learns more complex tasks until the final task is learned.  This provides an effective alternative to reward function engineering, lowering the amount of human effort needed to learn an effective policy. In Florensa et al. [5], an agent first learns to reach a goal state from nearby start states and then gradually learns to reach the goal state from farther and farther start states.  Molanchov et al. [6] takes a similar approach, growing reachability regions in which the learning agent has mastered transitions between each pair of states.

## Robotic Embedded Systems Laboratory

Our work is an extension of research being done in the Robotic Embedded Systems Laboratory (RESL) at the University of Southern California.  Molchanov et al [6]. learned a unified quadrotor control network to stabilize a quadrotor and produce hovering behavior at a specified point. However, our project will extend the previous work by focusing more on path following than on hovering behavior.

In order to reduce the burden of developing an accurate simulation of quadrotor dynamics, we chose to leverage the OpenAI Gym quadrotor environment developed by RESL.  Overall, we decided to work off of the Github repository created by RESL for Molchanov et al. [6]. Making use of the existing environment and quadrotor dynamics simulations has allowed us to focus on exploring in depth the applications of machine learning to creating a robust control network.

## OVERVIEW

### Simulation Environment

As stated above, our work extends research currently being done in RESL. We leveraged RESL's quadrotor simulation environment, which is compatible with Open AI Gym. The environment simulates a quadrotor in an x configuration, as shown in Figure 1, and thoroughly models its dynamics.

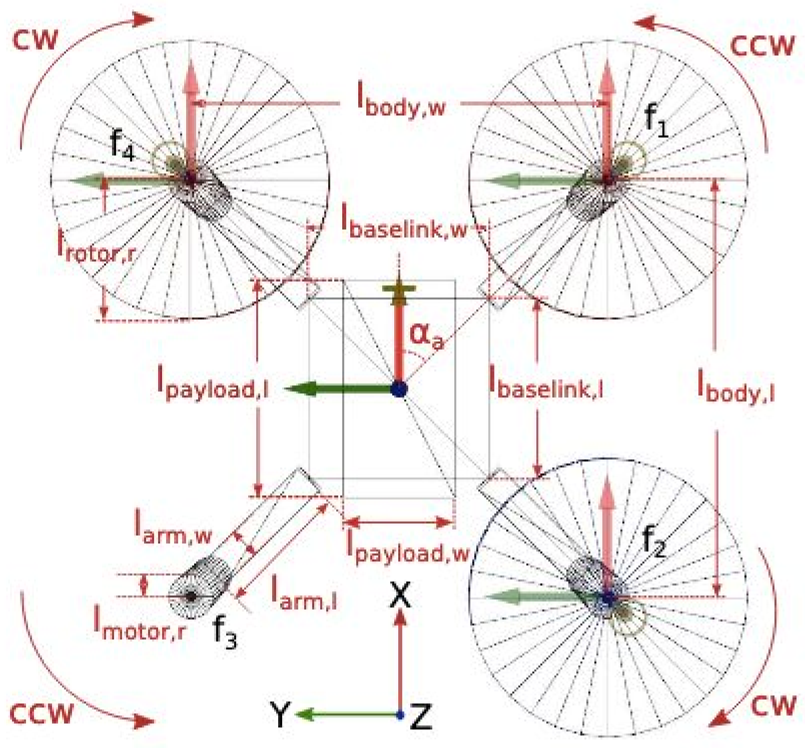


Figure 1: X configuration of quadrotor

## Quadrotor Visualization

Additionally, we've chosen to use a visualization tool also created by RESL that shows the quadrotor's position in its environment and the goal point it is attempting to reach. Along with TensorBoard, the visualization software allows us to analyze and better understand the quadrotor's performance. Figure 2 shows the visualization environment below.

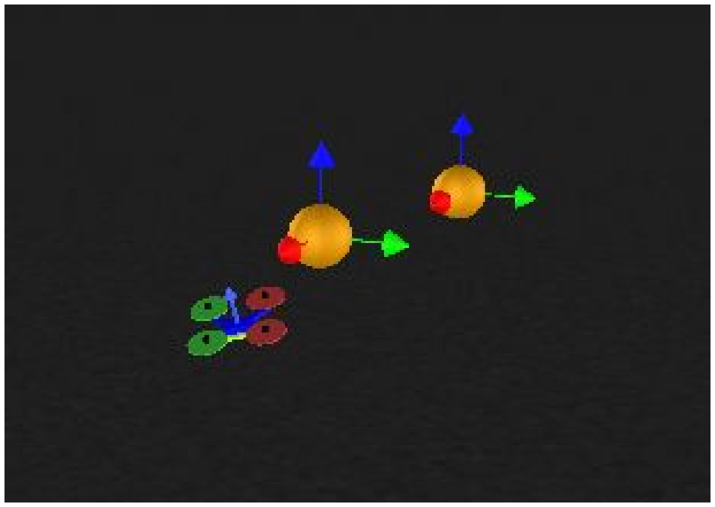


Figure 2: Quadrotor visualization tool. The quadrotor appears in the bottom left along with 2 goal points.

## Control Policy

As a final product, our software outputs a neural network which both stabilizes the quadrotor and guides it along specified trajectories. Though we have not yet attempted to run the control policy on a real quadrotor, Molchanov et al [6] had success in doing so on multiple real quadrotors. Because our methodology does not rely on specially tuned low-level controllers and only on the quadrotor simulation environment, it would likely generalize well to distinct quadrotor platforms with some changes to the simulation environment variables.

## METHODOLOGY

RESL previously explored using OpenAI's implementations of various algorithms but had the most success with Proximal Policy Optimization (PPO). Thus, as we began development, we chose to further explore PPO.

Prior to learning a control policy, we made updates to RESL's simulation environment and visualization software. Then, to begin leading the quadrotor to effectively perform path following, we first focused on directly extending RESL's PPO work by evaluating different reward functions. Next, we explored curriculum learning and Deep Deterministic Policy Gradient (DDPG).

### Restructuring Goal Representation

RESL previously trained the quadrotor to hover at a specific point with its z orientation perpendicular to the xy-plane. As a result, the existing OpenAI Gym quadrotor simulation environment only handled singular goals during training. We added support for a dynamic number of goals, allowing the learning agent to learn to follow entire trajectories rather than learning to hover at a single goal state.

### RESL Visualization Tool Updates

In keeping with our aim of trajectory following, we updated RESL's visualization tool to display a dynamic number of goals rather than a single goal. In order to show the quadrotor's progress as it follows a trajectory, we now alter the color of each goal when the quadrotor reaches it. Figure 3 shows two goals in the tool and Figure 4 shows the quadrotor during a test trajectory..

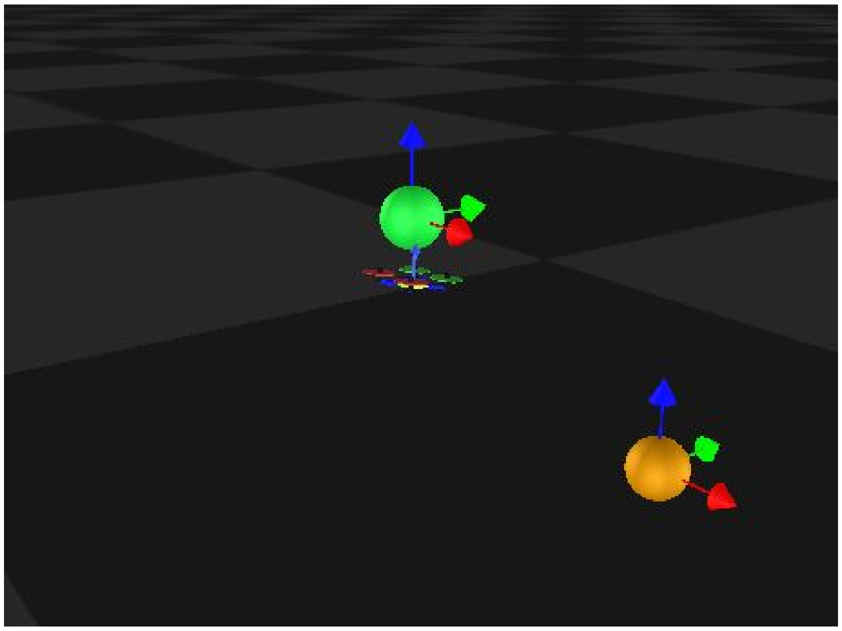


Figure 3: Visualization tool displaying 2 goals. The right quadrotor has reached the right goal, as evidenced by its golden color. Note that the quadrotor attempts to maintain its z orientation perpendicular to the xy-plane.

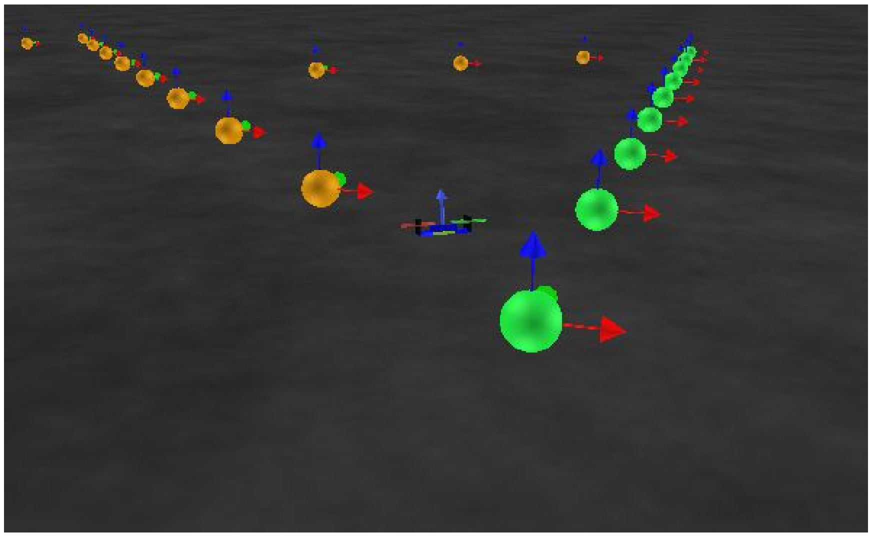


Figure 4: Visualization tool displaying the quadrotor executing a test trajectory.

## Reward Shaping

To begin leading the quadrotor to the desired path following behavior, we began by evaluating a number of different reward functions. Below is an outline of the original reward function and a selection of alternative functions we evaluated.

### Original Reward Function

Originally, the reward function penalized the quadrotor based on a number of different factors, including

- Distance between quadrotor's position and goal position
- The amount of control effort exerted by the quadrotor
- Whether the quadrotor crashed or not
- Orientation along the Z axis
- Yaw
- Uncontrolled rotation about the Z axis
- How drastically the rotor thrusts in the current action differed from those of the previous action
- Velocity
- Attitude.

### Activate loss to next goal when the previous is reached

**Description**

In this approach, we disregarded the error in position between the quadrotor and goals farther along in the given trajectory until the quadrotor came within a certain radius of the preceding goals. This is the most straightforward reward which we could use for multiple goals. Since the loss to each subsequent goal is scaled by 2, the quadrotor should care more about getting to this next goal then staying at the last one.

**Result**

The controller doesn't perform well. As soon as the quadrotor reaches the first goal, the reward function adds in the loss to the second goal. This means that it gets a sudden negative reward if it reaches a goal, so the controller just learns to stop right outside the tolerance where it reaches the goal.

### Only current goal is active

**Description**

Since the quadrotor does not need to return to previously reached goals, the reward function only factors in loss for the subsequent goal.

**Result**

As the quadrotor reaches a goal point, it again gets a negative reward since the loss to the subsequent goal gets factored into the reward function, so it again learns to hover outside of the radius of tolerance and does not learn to navigate through the given goal points.

### Add a constant negative reward for each goal which is not yet reached

**Description**

In the earlier reward functions the quadrotor received a negative reward for the following goal when it reaches the current one.  To address this issue, we add a constant negative reward for each goal which is not yet reached.  When the quadrotor reaches a goal, the constant negative reward is removed and replaced with the loss to the next goal.

**Result**

This reward function worked well and the quadrotor was finally able to reach the second goal, as it was not penalized for reaching each of the target goal points like before by simply the addition of loss to the next goal point. The addition of a constant negative reward for each goal not reached and the removal of this loss when it reached the point rewarded the quad well and helped it learn to follow the given path.

## Epsilon

**Description**

Our vanilla reward function (described above) simply activated losses for the next goal once a goal had been reached. We came up with an alternative approach where instead of losses being turned on or off using flags, the weight of the loss decreased the closer the quadrotor came to that goal. In other words, the error (loss) in position for the current goal would be defined by:

$$loss_{position} = \varepsilon_{min} E_{dist}$$

$$\varepsilon_{min} = min(\varepsilon_{min}, E_{dist})$$

Where $E_{dist}$ is simply the norm of the difference between the position of the goal and the position of the quad. Having the reward scale in this fashion effectively allowed the reward function to “remember” how close the quad got to the goal and assign a reward based on that value.

**Result**

Although this reward type seemed promising, the plots of average reward and average discounted reward were noisy and failed to converge even after prolonged training. We believe the nonlinearity introduced by $\varepsilon_{min}$ primarily caused this issue.

## Simplified Epsilon

**Description**

The motivation for the previous reward function was to scale the loss for the first goal proportionally based on the quadrotor’s proximity to the goal. This also altered the reward’s sensitivity to the change in position since the position loss now depends on both the distance to the goal, and the scaling factor (which again is a function of the distance to the goal). For this reward, the intuition was to remove this effect while still being able to incorporate the minimum distance to goal. For this, instead of multiplying the minimum distance to the current distance to goal, we just used the minimum distance itself to calculate the position loss.

**Result**

The quadrotor reaches both goals but in a suboptimal manner.

## Velocity in direction of the goal

The intuition for this reward type stems from the notion that an optimal trajectory would be one that gets to both goals the fastest. In order to reward this behavior, we added a velocity component to the loss. Specifically, we assigned a continuous loss of $loss_{velocity} = \alpha V_{proj}$, where $\alpha$ denotes the weight of the losses relative to the other losses (set to 1 in this case), and $V_{proj}$ is the velocity of the quadrotor in the direction of the next goal.

**Result**

This reward function ultimately proved to be unsuccessful.

## Z-Axis Orientation Loss

**Description:**

We found that the quadrotor continually attempted to remain vertically upright, keeping its z-axis orientation perpendicular to the floor, which prevented it from reaching the goal states in an optimal manner. We first attempted to reduce the degree to which the z-axis orientation loss factored into the reward function. This had an insubstantial effect on the quadrotor's behavior. Additionally, we trained the quadrotor to align its z-axis orientation in the direction of the next goal position. The following equation defines the Z angle to the goal:

$$Z_{angle\ to\ goal} = arccos(dot(goal - pos_{curr},\ pos_{curr}) / (norm(goal - pos_{curr}) * norm(pos_{curr})))$$

**Result:**

This performed well until the quad reached the first goal. After reaching the first goal, the quadrotor did not learn to properly align itself to the second goal.  We initially believed that adding the error of the orientation with respect to the second goal upon reaching the first goal caused this issue. So a value of pi was added and then disabled after reaching the first goal, when the error to the second goal was added. But the quadrotor still failed to align itself to the second goal even in this situation.

We also attempted to remove the z-axis loss from the reward function after a certain number of training iterations, but this also had little to no effect on the quadrotor's behavior.

## Curriculum Learning

Taking inspiration from Florensa et al. and Molanchov et al., we employed the assumption that the task of reaching goal states in smaller environments is simpler than in more spacious environments. We first trained the quadrotor to reach multiple goal states in a reduced-volume environment. After the quadrotor mastered the smaller task, we gradually grew the size of the environment, allowing the quadrotor to begin its trajectory from farther start states from the goal positions. Though further exploration of this approach would likely yield positive results, it was not as successful as the reward function we developed.

## Deep Deterministic Policy Gradient

In addition to PPO, we trained the quadrotor using DDPG. However, this yielded few results as the quadrotor did not learn to reach the goals. The model would likely perform better with further hyperparameter tuning.

## Supervised Learning

In addition to standard reinforcement learning, we hoped to explore the use of supervised learning to create a unified control policy.  To do so, we planned to fly the quadrotor using a game controller and train the quadrotor using these human demonstrated trajectories.  However, creating trajectories for training would have required a fine-tuned low-level controller, which we were unable to develop due to time constraints.

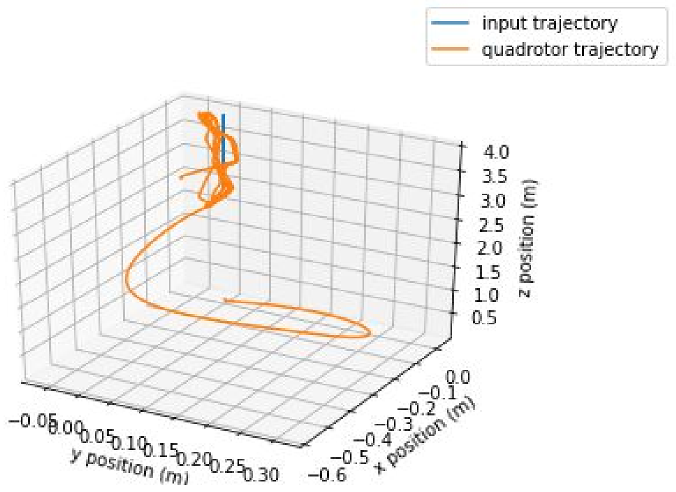
## ANALYSIS

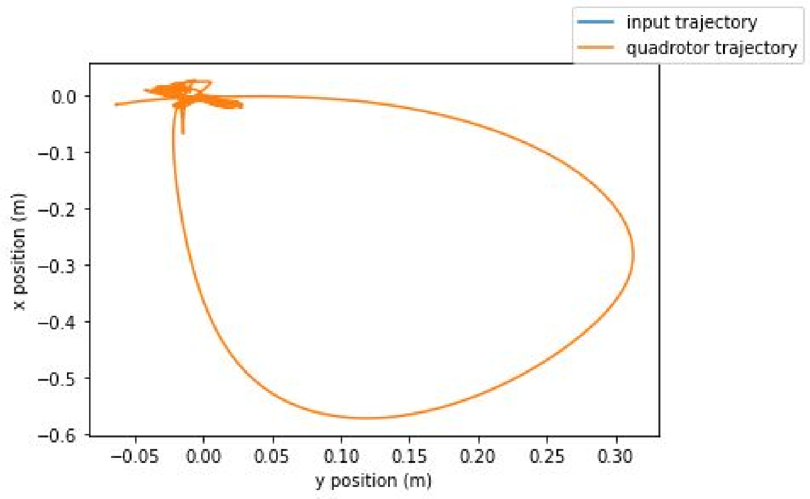
We tested each of our controllers with a variety of different trajectories. Below, we show a sampling of our test trajectories with the controller generated from the simplified epsilon reward function.

### Test Trajectories

#### Vertical Trajectory

In the vertical trajectory, the quadrotor moves up and down between heights of 2 meters and 4 meters. It does not move in the xy-plane, instead maintaining the same xy position. Below are two and three dimensional views of the quadrotor executing the trajectory. Aside from moving from the starting point to the first point in the test trajectory, the quadrotor deviates by less than .1 meter from the trajectory.

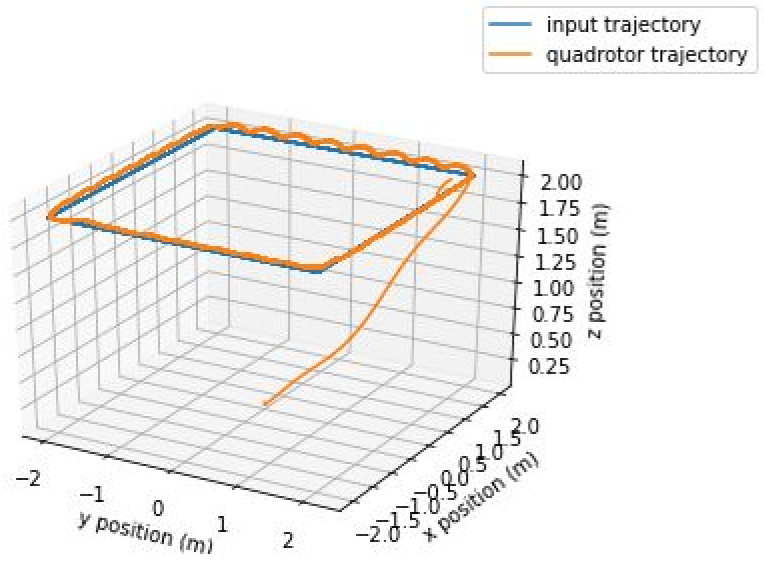


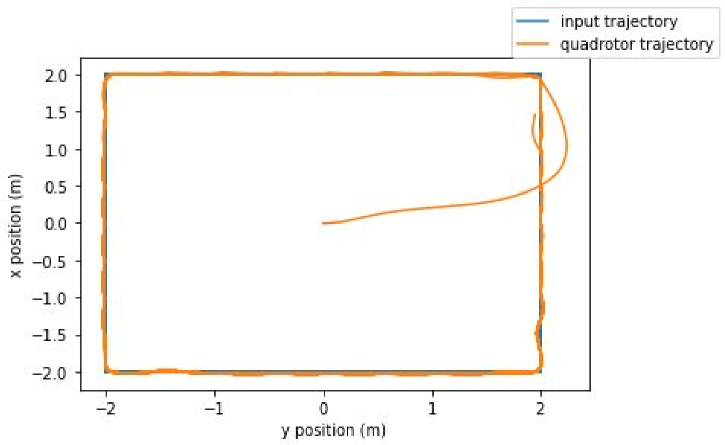


## Square Trajectory

In the square trajectory, the quadrotor moves through a square shape. It maintains a z position of 2 meters. Below, the two and three dimensional graphs show the controller results.

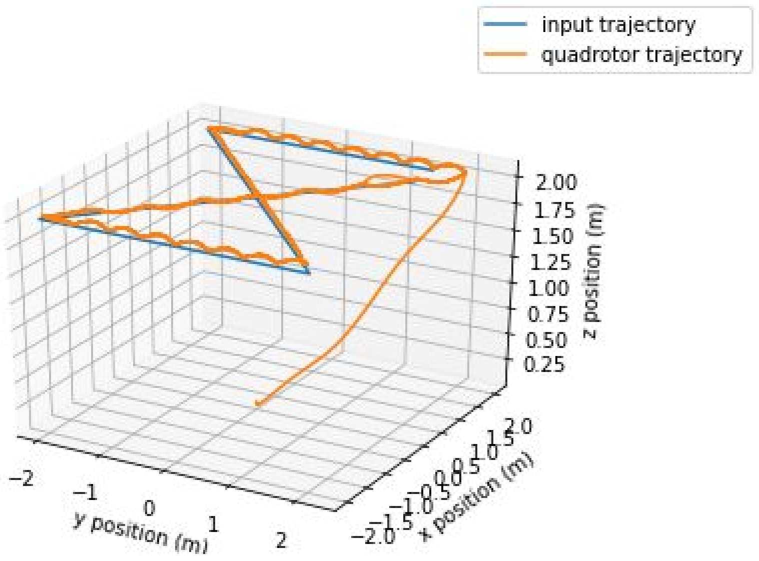
Again, the quadrotor remains within .1 meter of the trajectory. In spite of the quadrotor's overall ability to follow the trajectory, the oscillations evident in both graphs show that the controller has not reached true optimality.

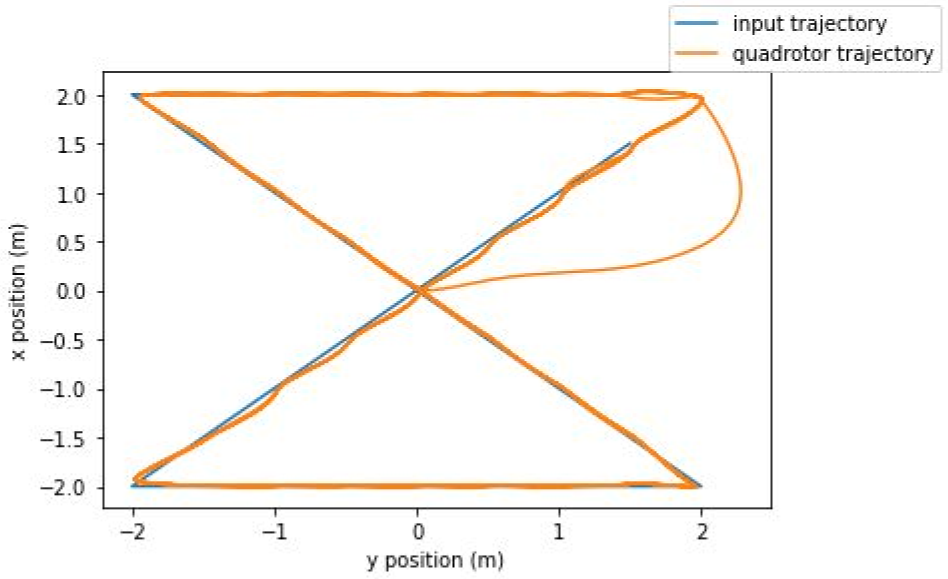




## Square Trajectory with Diagonals

In the third and final test trajectory, the quadrotor follows two edges and both diagonals of a square, while maintaining a height of 2 meters. The quadrotor again stays within .1 meter of the trajectory while oscillating.





### Results

Through feeding the test trajectories to each of our controllers, we found that the most successful controllers remained within .1 meter of each trajectory.  However, the quadrotor oscillated while executing the trajectories, potentially indicating that the quadrotor is still learning to reach singular goal states rather than to execute the trajectory in a fluid and optimal manner.

## LIMITATIONS

### Lack of generality

After having trained several models and distilling them to ones that worked reasonably well, we realized that our models did not generalize well to goals of varying distances from each other. We had initially assumed that training using goals that were always a fixed distance apart would generalize well, seeing as the error to the goal was part of the observation. To our surprise, this was not the case. When using models trained on goals a distance of 0.5m apart in an environment where the inter-goal distance was randomized, we saw that the quadrotor would slow to a hover after moving a distance of 0.5m towards the goal.

However, we found that the time needed to train a consistent, robust model increased greatly when the distances between goals are randomized. Though performance did improve steadily while training with randomized distances, the added training time meant that this was far less practical, and so we elected to limit the scope of training to fixed-distance goals.

### Reward Shaping

We found reward shaping a more difficult task than anticipated, as the quadrotor continually learned to behave in ways we didn't expect.  Creating a reward that both led the quadrotor to behave in a desired manner that would also generalize to different situations, such as goal states with varying distances between them, proved challenging.  Overall, effective reward shaping necessitated more man hours than predicted.

## CONCLUSIONS

Our work shows that reinforcement learning can effectively generate unified quadrotor controllers that are performant both in terms of low-level control and with respect to trajectory tracking. The problem of time-optimal trajectory tracking, however, has proven to be a more complex issue. Our results show that while the quadcopter accurately tracks a given trajectory, it struggles to find a balance between accuracy and speed. Instead of visiting each point while maintaining momentum, it slows to a hover before reaching a goal in order to ensure accuracy. We suspect that relaxing the tolerance radius at which a goal is considered to be reached might address this issue, as this would decrease the reward sparsity in rollouts where speed is explored, thereby incentivizing a faster approach.

## FUTURE WORK

There are innumerable options for future work in this problem space.  Further exploration of curriculum learning and supervised learning could yield positive results. Additionally, other reinforcement learning algorithms could present solutions.  In particular, DDPG and Advantage Actor Critic could create effective controllers.  In the case of DDPG, we believe that further hyperparameter tuning would likely yield a more optimal controller than that which we achieved.

## REFERENCES

[1] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, “Control of a quadrotor with reinforcement learning,” IEEE Robotics and Automation Letters (RA-L), vol. 2, no. 4, pp. 2096–2103, 2017.

[2] F. Sadeghi and S. Levine, “CAD2RL: real single-image flight without a single real image,” Robotics: Science and Systems, 2017.

[3] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine, “Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight,” IEEE Intl Conf. on Robotics and Automation (ICRA), 2019.

[4] A. Y. Ng, D. Harada, and S. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping,” International Conference in Machine Learning, volume 99, pages 278–287, 1999.

[5] C. Florensa, D. Held, M. Wulfmeier, and P. Abbeel, “Reverse curriculum generation for reinforcement learning,” in CoRL, 2017.

[6] Molchanov, A., Hausman, K., Birchfield, S., Sukhatme, G., “Region Growing Curriculum Generation for Reinforcement Learning,” ArXiv e-prints, 2018.

[7] Molchanov, A., Chen, T., Honig, W., Preiss, J. A., Ayanian, N., and Sukhatme, G. S., “Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors,” arXiv preprint arXiv:1903.04628, 2019.