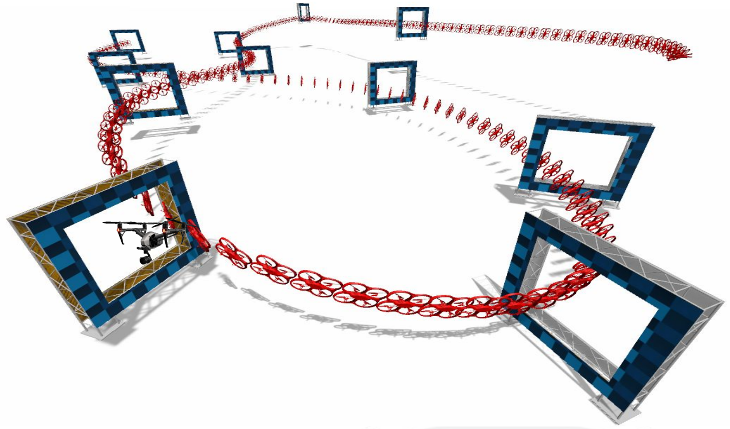


RBE 595 - Reinforcement Learning

Vision Based Quadrotor Obstacle Avoidance

Uday Girish Mardana
Jeel Chatrola

Instructor: Navid Dadkhah Tehrani

Objective

- To perform **end-to-end motion planning** for a quadrotor, emphasizing the use of raw depth images for reinforcement learning.
- We deploy a single, reliable module using **deep reinforcement learning**. A quad-rotor is deployed for navigation in a partially known environment.
- A straight path to the goal position is known **without any obstacle location** information on it.
- The quad-rotor is supposed to generate local motion plans using this information and its online sensory data for **safe and quick navigation**.

Approach

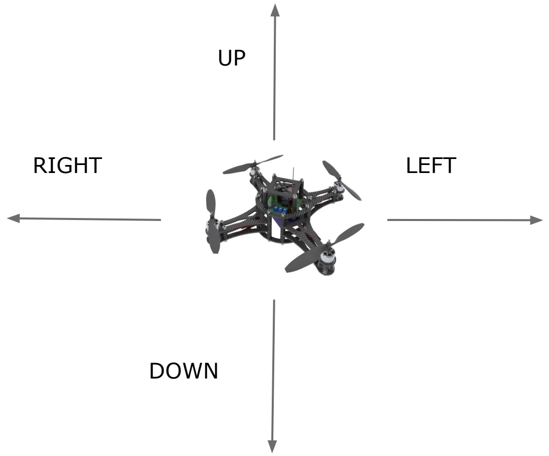
- **State** - Depth image (84x84) + position (3x1).

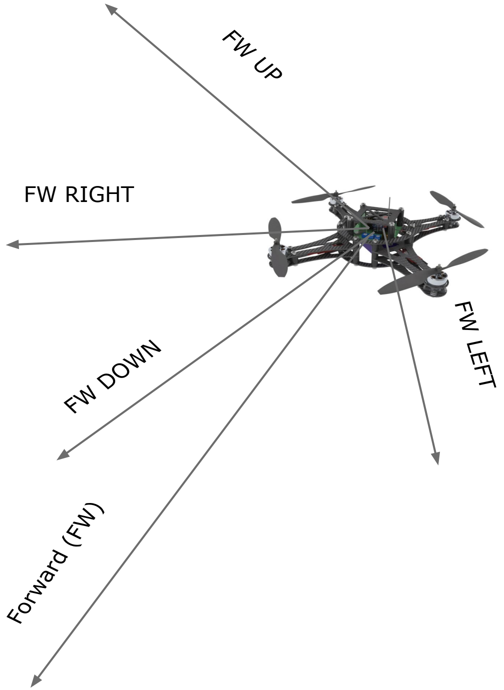
- **Action** - Respective move at each time step following some policy to increase rewards (among 9 motion primitives).

- **Reward** - Signal which assesses the quality of the agent’s actions.

- **Agent (DQN)** - Trained which takes a raw depth image and relative position information as its input, and yields a motion primitive selection as its output We want to Estimate Action-Value Function Q(s,a).

Motion Primitives





Reward Design

- **Distance Reward (R_d)**: It encourages the drone to move closer to the goal by providing a higher reward for smaller distances. d - distance between of current state to goal state

$$R_d = e^{-0.1 \cdot d_x} + 0.75 \cdot e^{-0.1 \cdot d_y} + 0.5 \cdot e^{-0.1 \cdot d_z}$$

- **Angle Reward (R_α)** : This is the reward based on the angle between the current position and the start position with respect to the goal. It encourages the drone to move close to global path that reduces this angle, effectively guiding reducing deviation from path. G = Goal State, S = Start State, C = Current State.

$$R_\alpha = \begin{cases} 1 & \alpha < \frac{\pi}{4} \\ 1 - \dfrac{2\alpha}{\pi} & \alpha > \frac{\pi}{4} \end{cases} \qquad \alpha = \arccos\left(\frac{(S_g - S_s) \cdot (S_c - S_g)}{\|S_g - S_s\| \cdot \|S_c - S_g\|}\right)$$

Continued ...

- **Collision Penalty (R_{cp})**: It discourages the drone from colliding with obstacles by providing a significant penalty for collisions.

- **Smoothness Penalty (R_ϕ)** : It discourages abrupt changes in the drone's position, promoting a smoother and more stable flight.

$$R_\phi = -0.01 * ||S_c - S_p||$$

- **Speed Reward (R_s)** : It encourages the drone to maintain a higher speed, which could help it reach the goal faster.

$$R_s = \sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2}$$

Continued …

- **Nearest Neighbours Reward (R_{nn}):** This is the reward based on the nearest neighbors on the global path to the current. It encourages the drone to stay close to the global path by providing a higher reward for being closer to the points on this path.

$$\begin{aligned} pts &= [S_s : 100 : S_g] \\ S_n &= \text{nearest\_point}(pts, R) \\ d_{\text{closest}} &= ||S_n - S_g|| \\ R_{\text{nn}} &= \exp(-0.1 \cdot d_{\text{closest}}) \end{aligned}$$

- **Total Reward R** :

$$R = R_d + R_\alpha + R_s + R_{cp} + R_\phi + R_{nn}$$

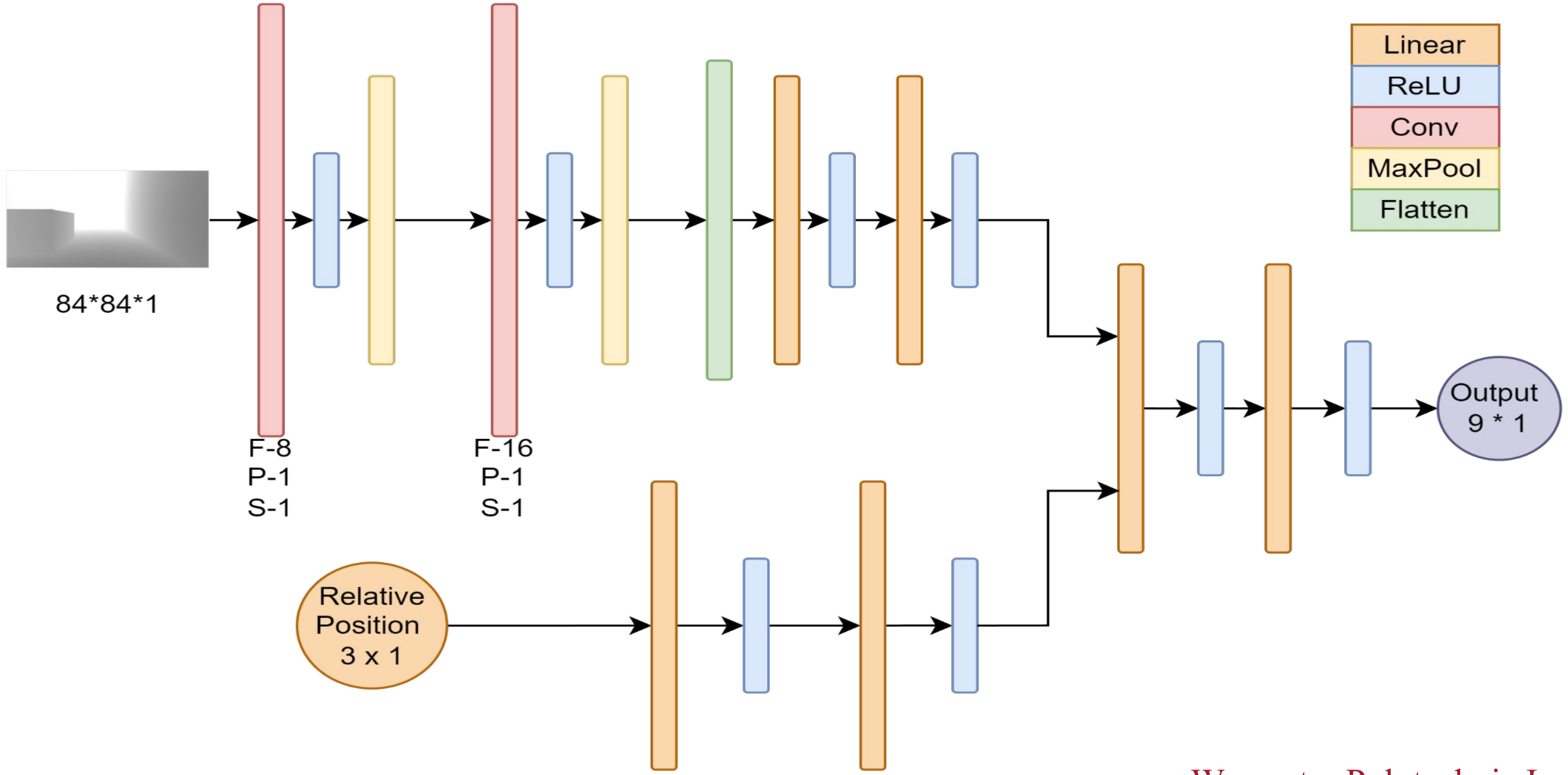
Algorithmic Details

- Main aim of Reinforcement learning algorithm for this task is to estimate the action-value function Q(s,a).

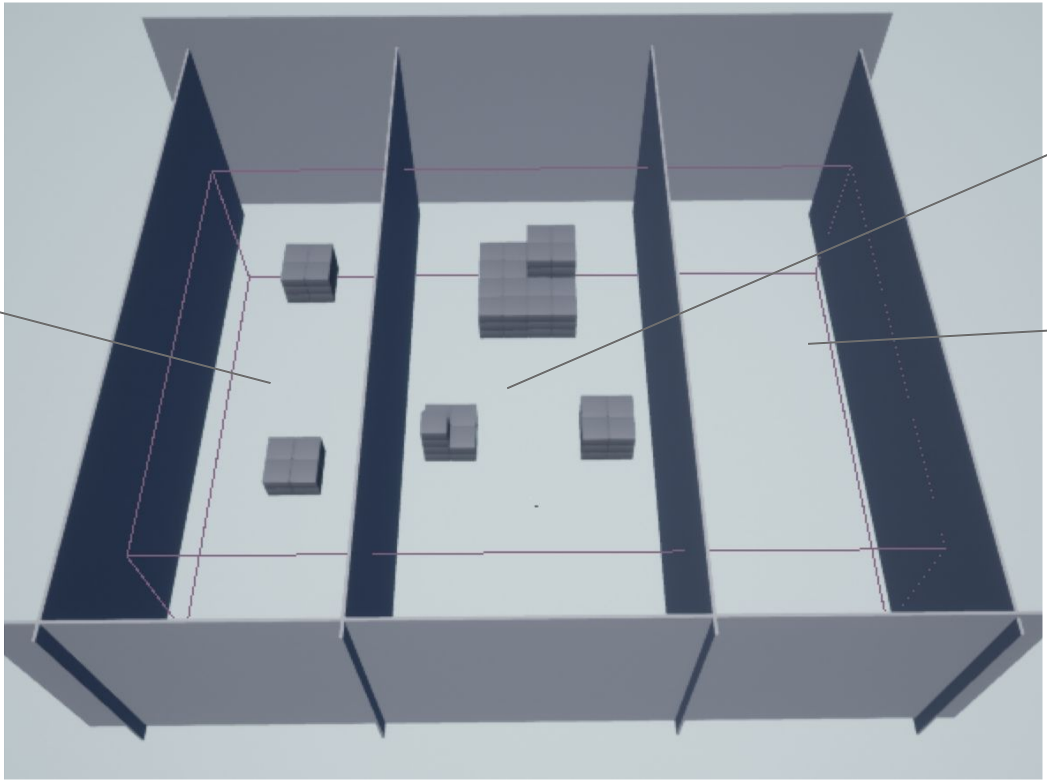
$$Q(s,a) = \mathbb{E}\left(R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') \middle| s_t = s, a_t = a\right)$$

- For this project we select Deep-Q-Network for learning the complex action value function.
- **Notation:**
  - s_t is current state of agent at time t.
  - s_{t+1} state of agent at time t+1.
  - a_t is action taken at time t.
  - R_{t+1} is reward agent obtains at time t+1 after taking action a_t.
  - γ is discount factor.

Network Architecture (Custom Multi Input)



Experimental Setup (Custom)

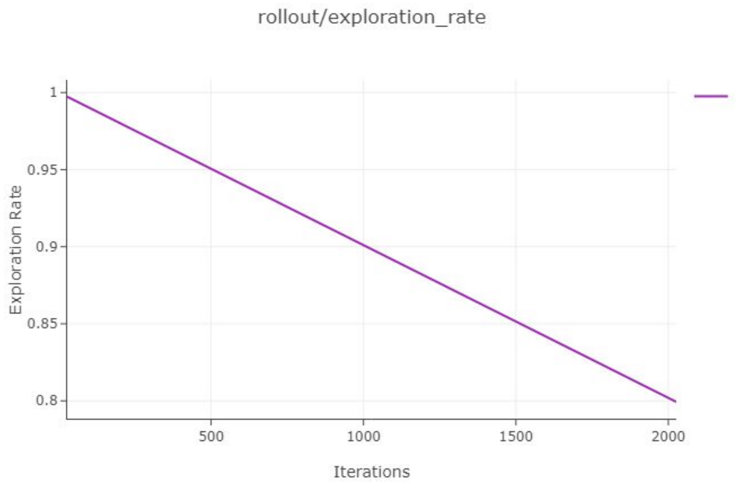


Test on both 1 & 2 Obstacle Environments

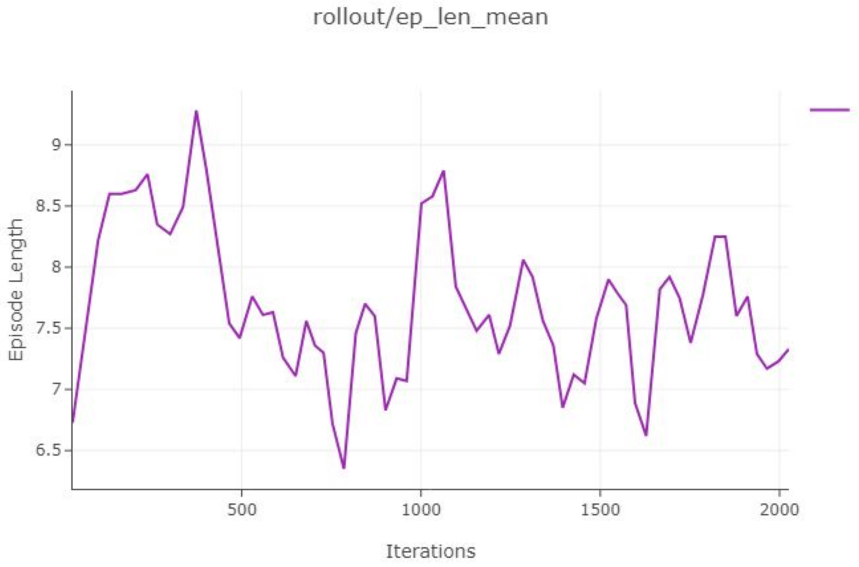
Complex Multi Obstacle Env

No Obstacle Env

Experiments

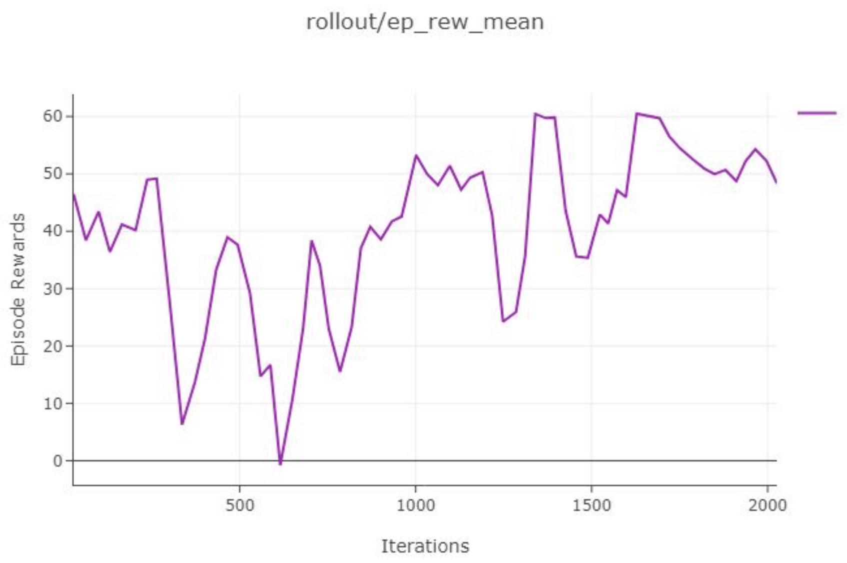


Episode Exploration Rate

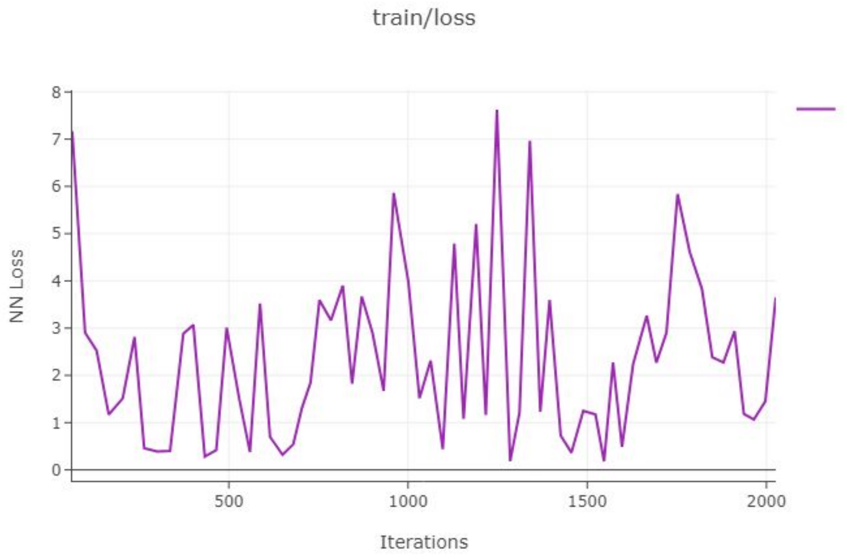


Episode Length Mean

Experiments

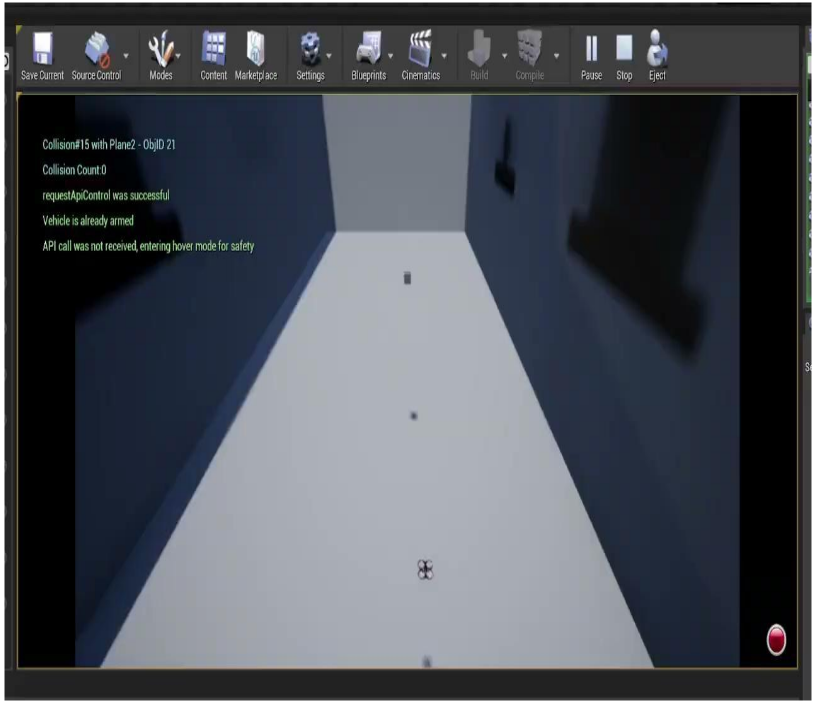


Episode Reward Plot



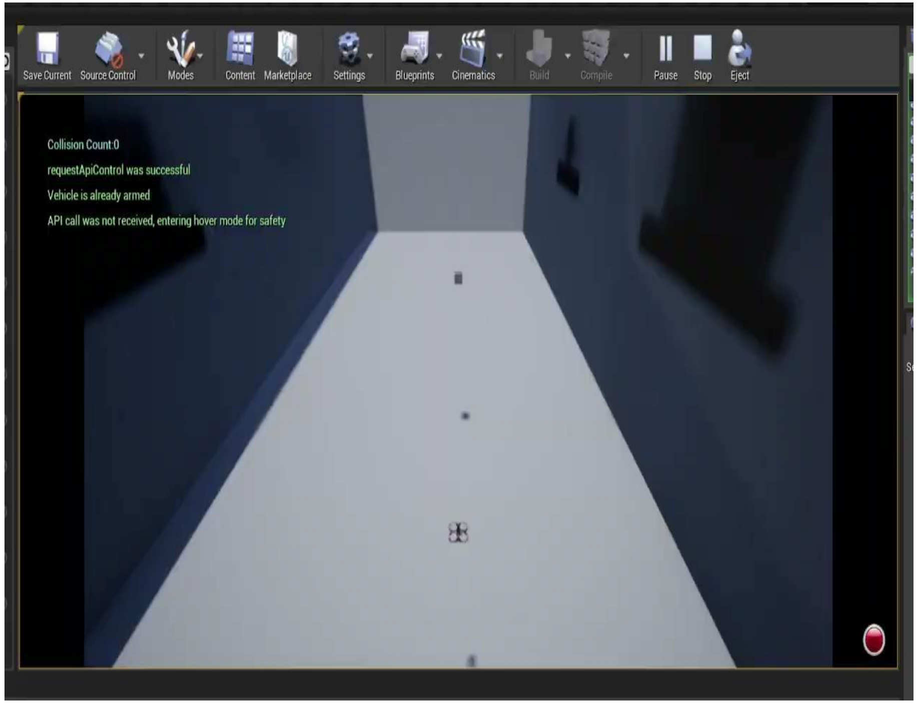
Episode Loss

Results - No Obstacle



```
Collision#15 with Plane2 - ObjID 21
Collision Count:0
requestApiControl was successful
Vehicle is already armed
API call was not received, entering hover mode for safety
```

Partial Successful - Initial iterations (Drift at the End)

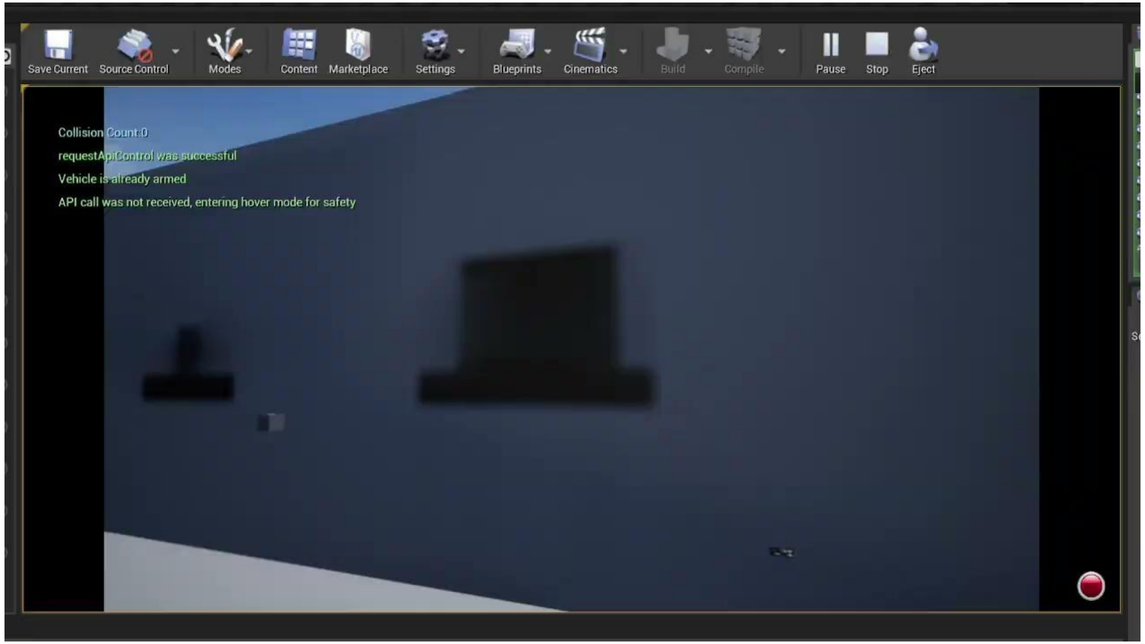


```
Collision Count:0
requestApiControl was successful
Vehicle is already armed
API call was not received, entering hover mode for safety
```

Successful Task Completion

Worcester Polytechnic Institute

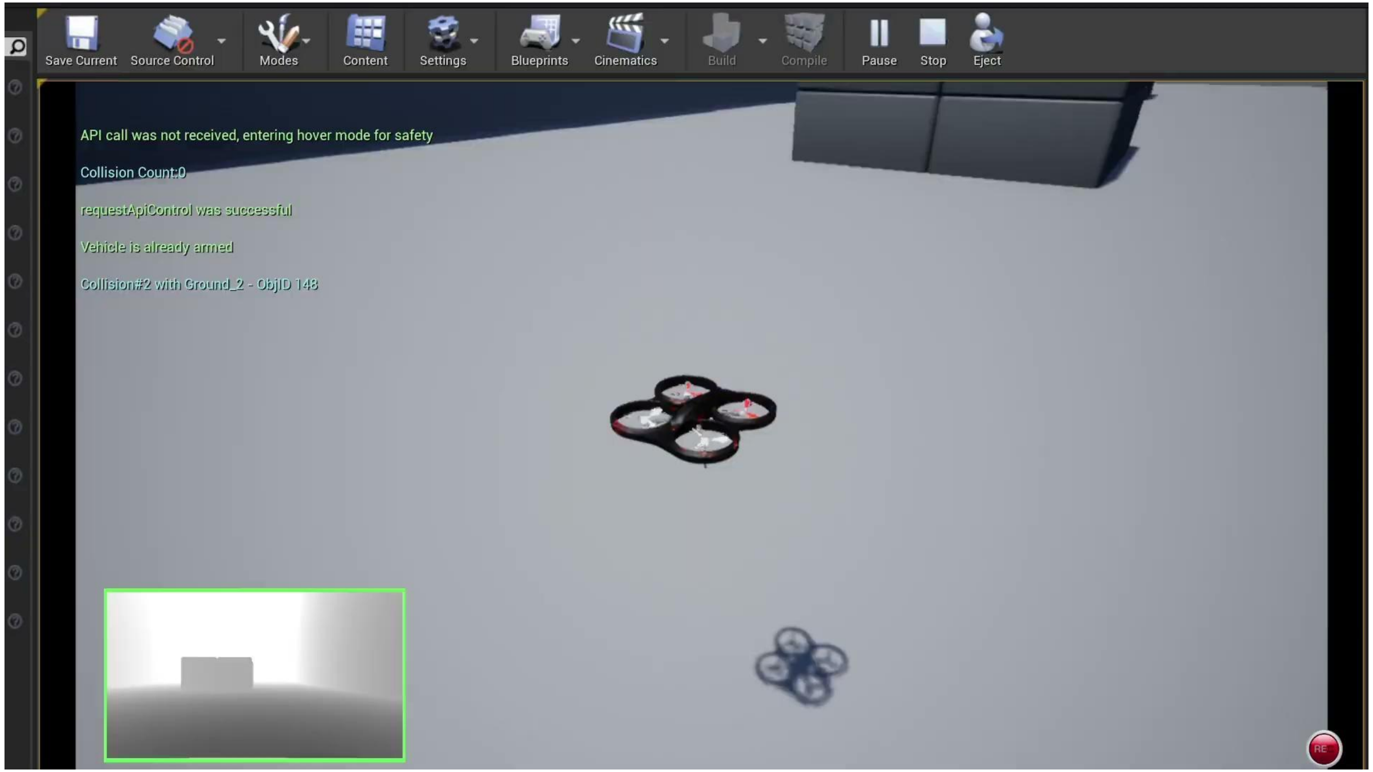
Results - No Obstacle (Side View)



```
Collision Count:0
requestApiControl was successful
Vehicle is already armed
API call was not received, entering hover mode for safety
```

Successful Task Completion - Side View

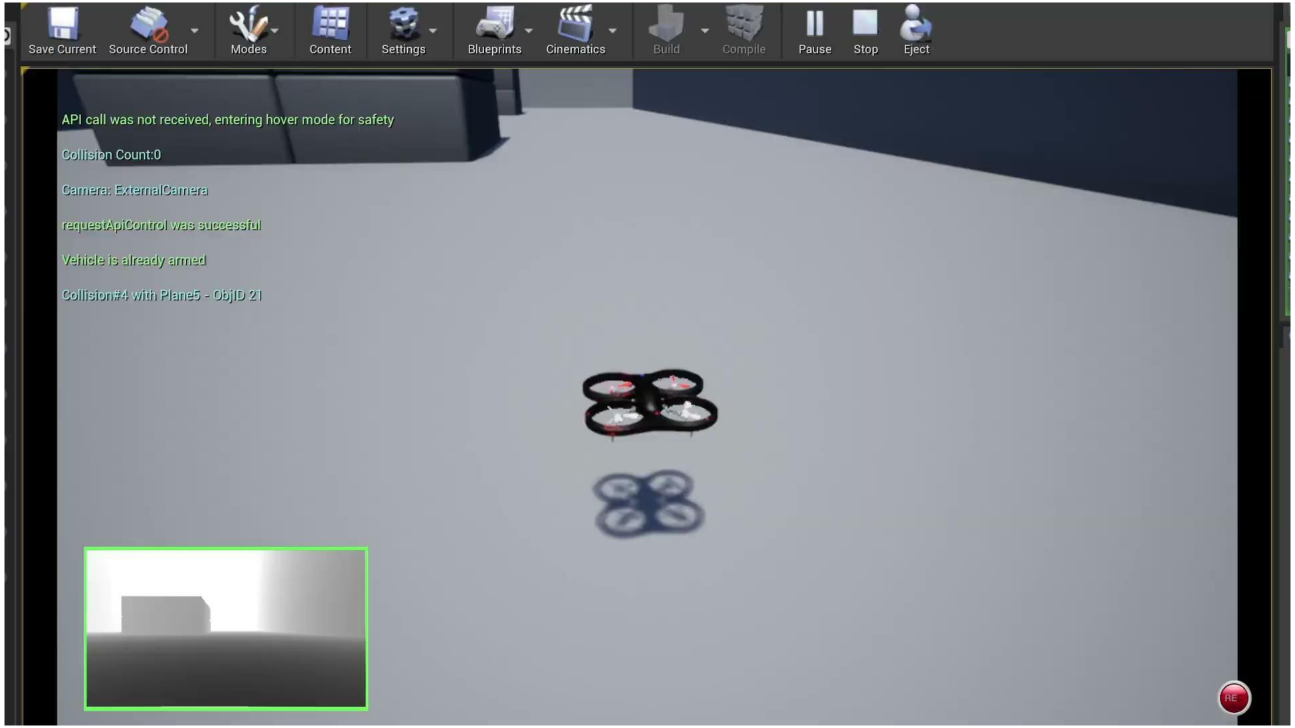
Results - Single Obstacle



```
API call was not received, entering hover mode for safety
Collision Count:0
requestApiControl was successful
Vehicle is already armed
Collision#2 with Ground_2 - ObjID 148
```

Successful Task Completion - Single Obstacle

Results - Double Obstacle



API call was not received, entering hover mode for safety

Collision Count:0

Camera: ExternalCamera

requestApiControl was successful

Vehicle is already armed

Collision#4 with Plane5 - ObjID 21

Double Obstacle - Drift at the Second Obstacle

Further Training Iterations might resolve this

Challenges

- The training involves a lot of hyperparameters which took time and we are not yet sure about their optimality.

- The results were based on less number of iteration as experimenting different rewards and also equally training involved significant amount of compute and time which we were not able to afford.

- We are limited to testing DQN only at this point because of time constraints. Further testing and experimenting with different Neural Network architectures such as **Attention Models** which can understand the **relationship between Depth and the Relative position** or any additional information for Nav would be helpful.

Conclusion

- During our literature review we found QR-DQN (Distributional Reinforcement Learning with Quantile Regression) it is extension of DQN, it models the full distribution of the Q-values instead of just the expected value.
  - **Better Uncertainty Estimation:**
  - **Improved Exploration**
  - **Robustness to Noise**
  - **Better Handling of Multi-modal Returns**

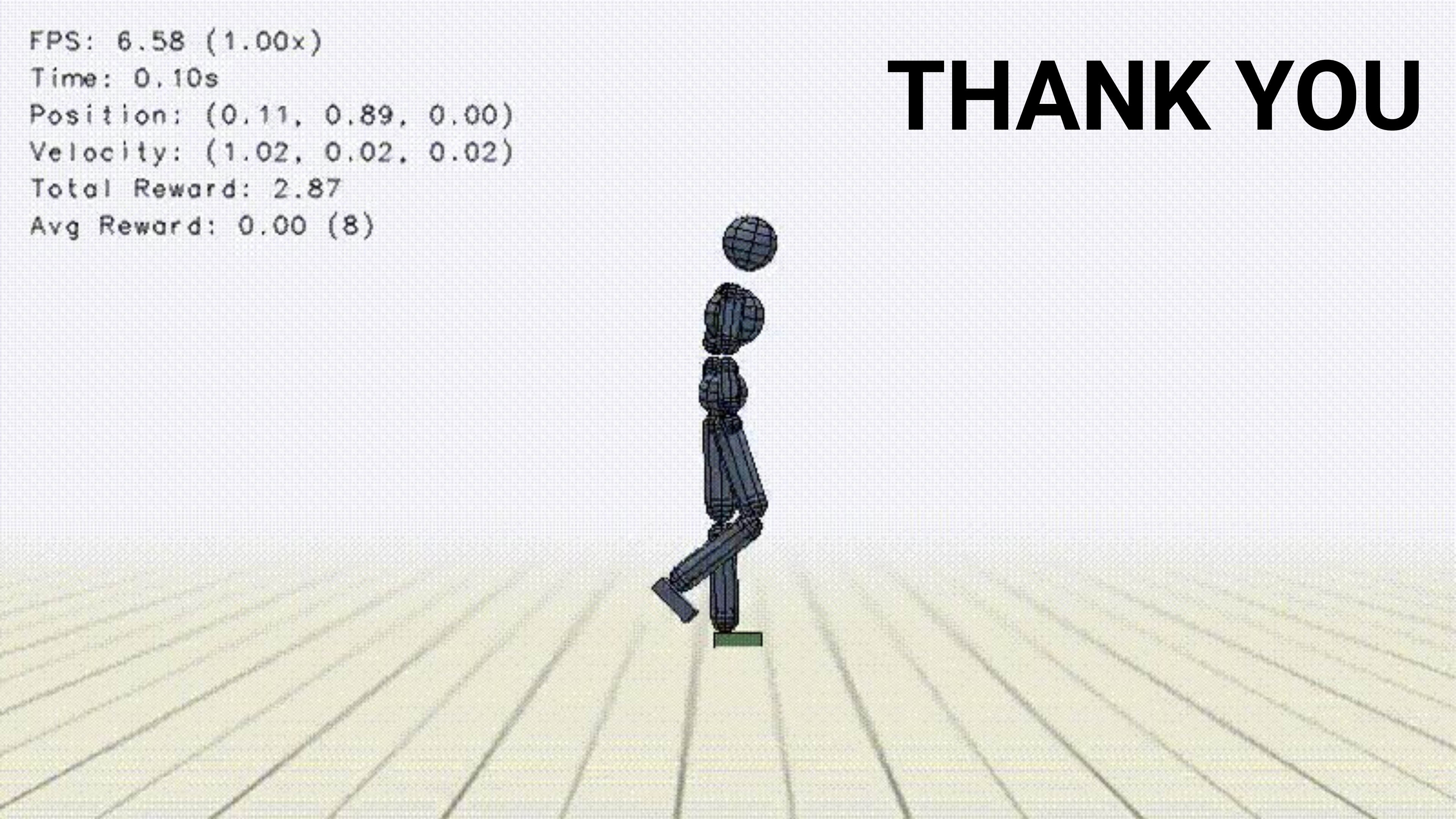
- During out experimentation we found that using motion primitive is a good idea compared to mapping the images to thrust vectors of drone. But if motion primitives are not well defined or doesn’t contain primitives for complex maneuvers drone might not be able to solve the navigation task.

- It took a good amount of time, integrating Airsim and Unreal Engine in linux.

References

1. Distributional Reinforcement Learning with Quantile Regression
   Link: https://arxiv.org/abs/1710.10044
2. End-to-End Motion Planning of Quadrotors Using Deep Reinforcement Learning
   Link: https://arxiv.org/pdf/1909.13599
3. Stable Baseline 3 Documentation
   Link: https://github.com/DLR-RM/stable-baselines3/tree/master
4. Airsim Documentation
   Link: https://airsim-fork.readthedocs.io/en/docs/
5. Reinforcement Learning- second edition Richard S. Sutton and Andrew G. Barto
   Link: http://incompleteideas.net/book/RLbook2020.pdf

THANK YOU



```
FPS: 6.58 (1.00x)
Time: 0.10s
Position: (0.11, 0.89, 0.00)
Velocity: (1.02, 0.02, 0.02)
Total Reward: 2.87
Avg Reward: 0.00 (8)
```