Autonomous Drone Navigation for Landmark Position Estimation using Reinforcement Learning

Michalis Galanis
Technical University of Crete

Thesis Committee
Associate Professor Michail G. Lagoudakis (ECE)
Professor Michalis Zervakis (ECE)
Associate Professor Panagiotis Partsinevelos (MRE)

1 INTRODUCTION

1.1 – Problem Statement

1 2 3 4 5

1.1 PROBLEM STATEMENT

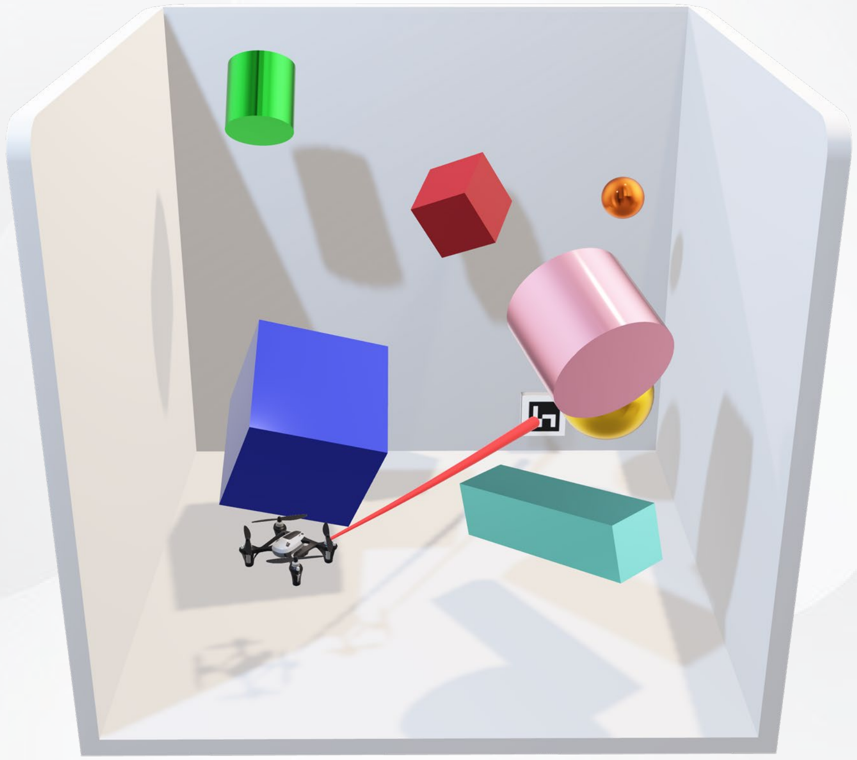
What problem are we trying to solve?

Let's assume a 3D environment, filled with **obstacles** and **markers** (targets).

A UAV (drone) is spawned in a random location inside the environment.

OBJECTIVE: The UAV's goal is to **approach** the marker in the environment without **crashing** into any obstacle.

CAUTION: **No rules are given** to the UAV about its **objective** or its **environment** whatsoever.



1.1 PROBLEM STATEMENT

How do we solve this problem?

REQUIREMENTS: The agent must be able to **safely navigate** through the environment, detect and approach a predefined set of ArUco markers (landmarks).

RELATED WORK: Several approaches have been investigated to create autonomous navigation systems such as **Simultaneous Localization and Mapping** (SLAM).

IDEA: **Reinforcement leaning** (RL) is a promising alternative that focuses on learning by **trial-and-error** procedure, in which an agent interacts with its environment and receives continuous feedback based on each action.

THESIS CONTRIBUTION: This thesis explores a **mapless approach** to UAV autonomous navigation in completely unknown 3D environments using **deep reinforcement learning** (DRL), a reinforcement learning subfield that incorporates **deep learning techniques** (deep neural networks) to overcome dimensionality limitations.

2 BACKGROUND

2.1 REINFORCEMENT LEARNING

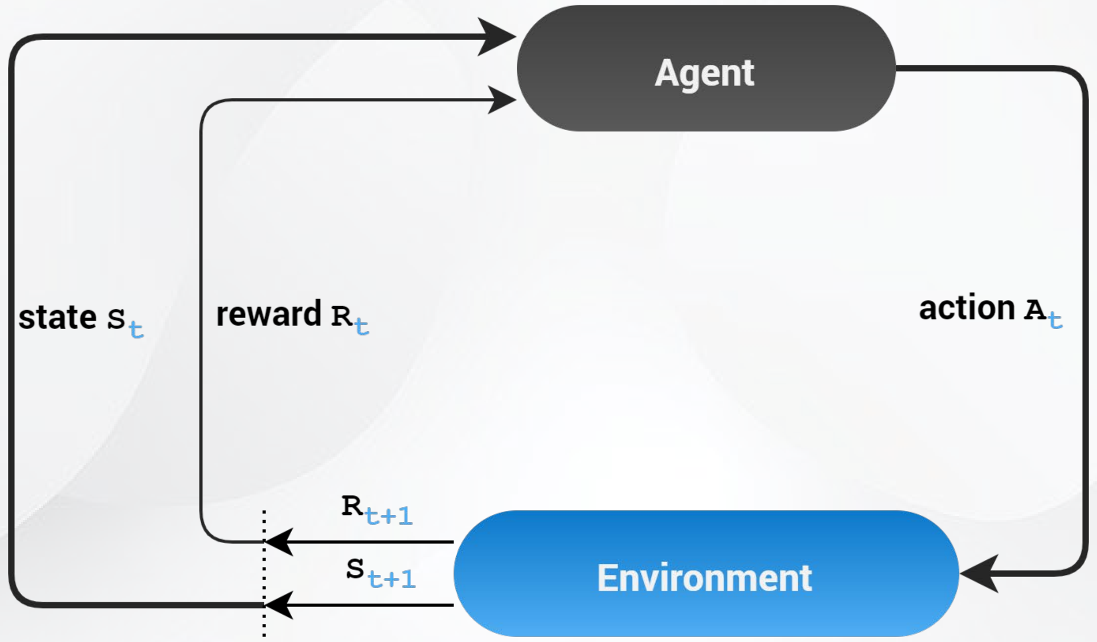
Overview

In RL, the agent learns from its own experience by **interacting** with the environment.

The agent performs an **action** in the environment.

The performed action yields a **new state**.

The environment evaluates the agent's action through a **reward**.



2.1 REINFORCEMENT LEARNING

General Terms

Step: smallest possible unit of time.

Episode: collection of steps (from an initial state to a terminal state).

State: collection of variables and conditions that describe a situation of the environment at a specific point in time.

- fatal conditions lead to **terminal states** → Episode ends, environment needs to be reset
- **State space**: set of every possible state in the environment (discrete / **continuous**)

Action: the agent's method of interaction with the environment.

- **Action space:** set of every possible action the agent can make (discrete / **continuous**)

Policy: strategy to determine the next action based on the current state.

Reward: immediate feedback returned to the agent by the environment evaluating its previous action.

Q-value: overall expected reward for an agent (when being in a current state s, performing an action a and obeying a policy π).

2.1 REINFORCEMENT LEARNING

Goal & Rewards

RL GOAL: find an **optimal sequence** of **actions** that lead to the **maximum cumulative reward** in search for a goal (**optimal policy**).

REWARD IMPORTANCE: Rewards are an integral part of a reinforcement learning problem.

- **higher** rewards motivate the agent to repeat an action in a similar situation
- **lower** rewards teach the agent to avoid a certain behavior
- Subsequent rewards are increasingly **discounted**, to **reduce their importance** (future rewards hold less accurate reward information due to uncertainty)

2.1
1 2 3 4 5

2.1 REINFORCEMENT LEARNING

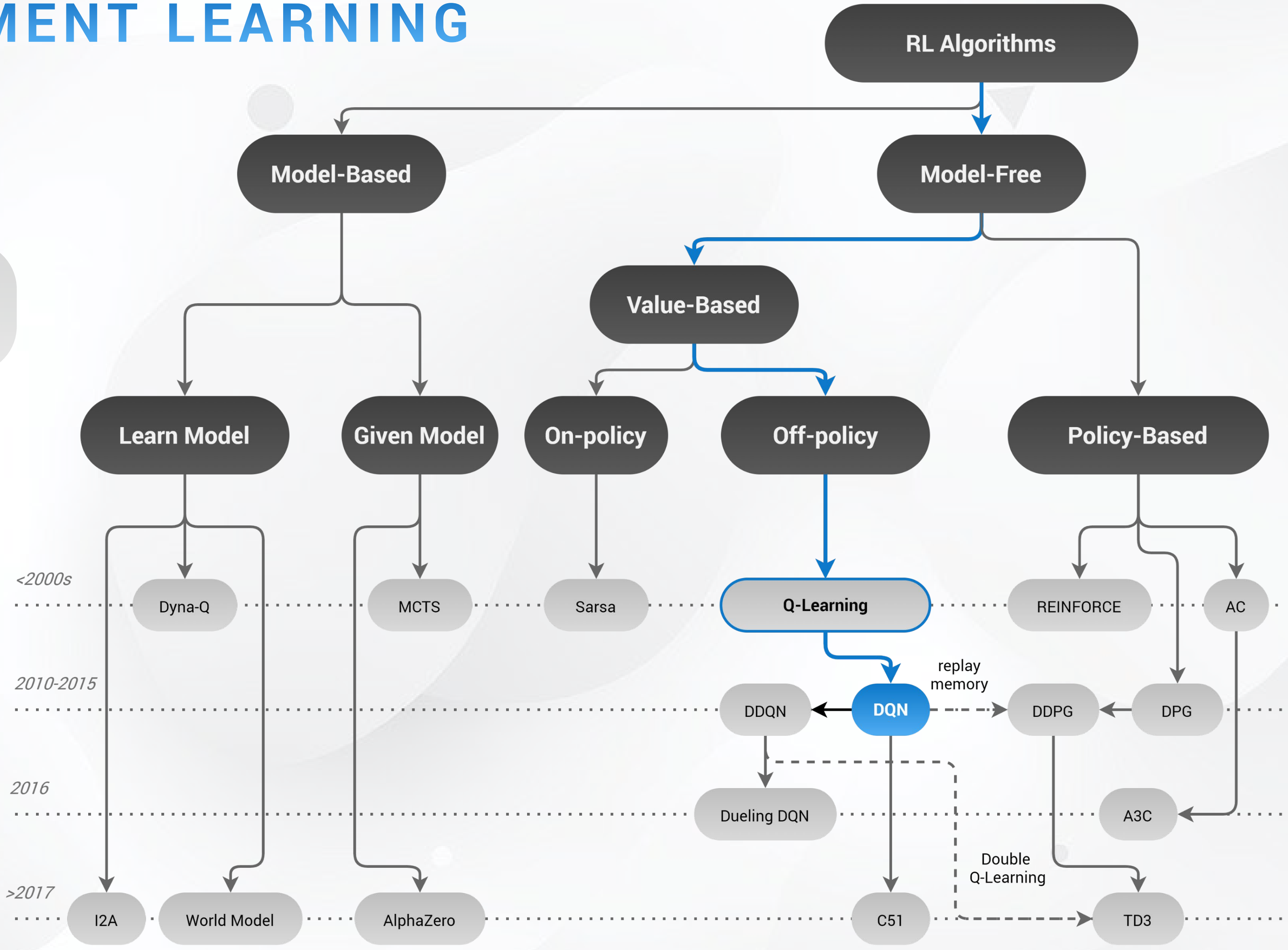
Algorithm Taxonomy

There is currently a large variety of reinforcement learning algorithms to choose from.

This thesis focuses on **model-free**, **value-based**, **off-policy** approaches.

Q-Learning is an algorithm to learn the Q-value of an action in a particular state.

Q-Learning seems promising. Let's take a look.



2.1 REINFORCEMENT LEARNING

The Q-Learning Algorithm: Overview & Limitations

OVERVIEW: Q-Learning is a TD off-policy algorithm, finds optimal policy using a Q-function.

- **SUMMARY**: Iteratively updates Q-Values for each state-action pair using the bellman equations until the q-function converges to the optimal q-value.
- Q-Values are stored in a **Q-table**, a table of states and actions in which Q-values are stored for each pair.

PROBLEM: **Curse of dimensionality**. Q-Learning only works well with small discrete state and action spaces. Larger spaces render the tabular approach incredibly inefficient.

SOLUTION: **Value function approximation,** generalize value estimation of similar states to reduce complexity. DQN uses neural networks to approximate the value function.

2.2 DEEP REINFORCEMENT LEARNING

The DQN Algorithm: Overview & Major Features

DQN CONCEPT: DQN incorporates deep learning techniques (deep neural networks) for value function approximation.

- Practically, it can handle very large state spaces.

MAJOR FEATURE 1: Experience Replay, breaks sample correlation → improves performance

MAJOR FEATURE 2: Target Network, uses 2nd network for loss calculation with frozen weights improves stability

MINOR FEATURE 1: Clipping Rewards, cuts off extreme reward values → solves instability

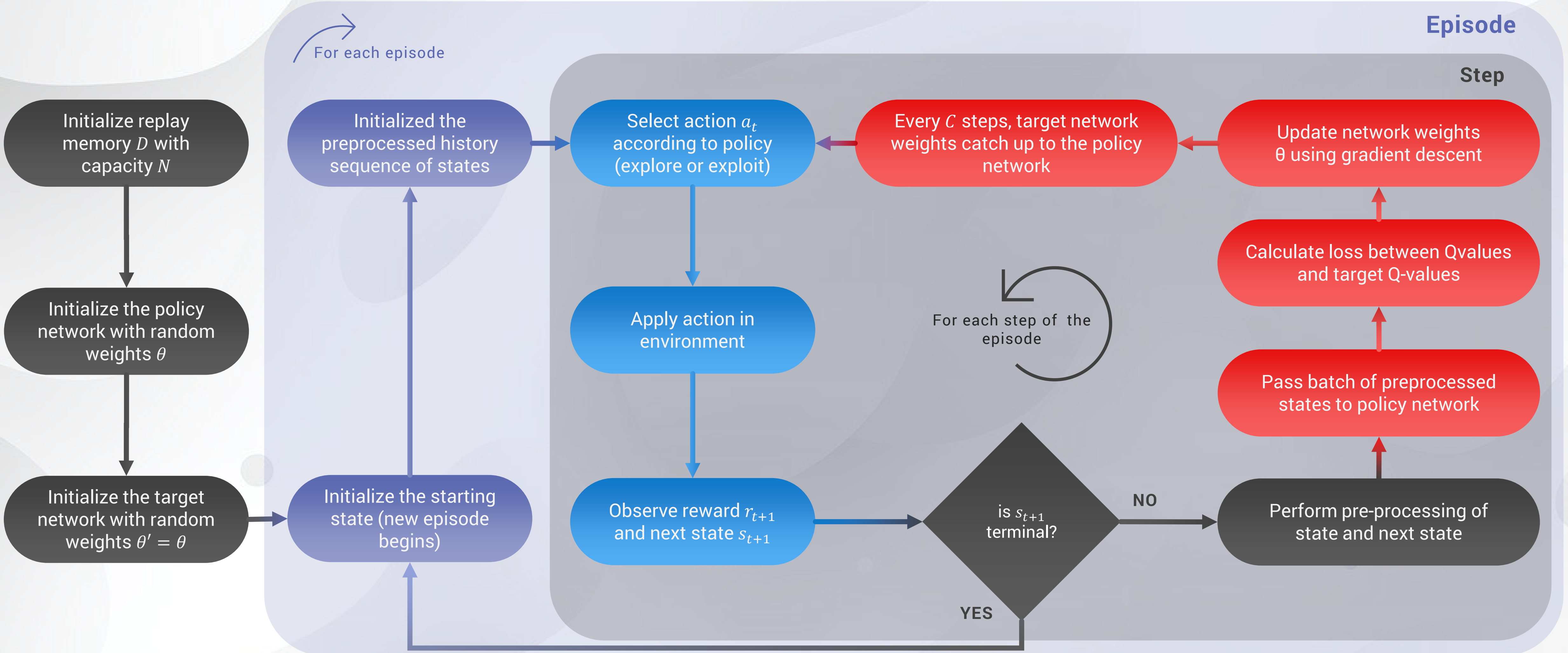
MINOR FEATURE 2: Skipping Frames, consecutive frames contain overlapping information → increases training speed

MINOR FEATURE 3: History Preprocessing, train neural network with a stack of last several frames

2.2
1 2 3 4 5

2.2 DEEP REINFORCEMENT LEARNING

The DQN Algorithm: Processing Diagram

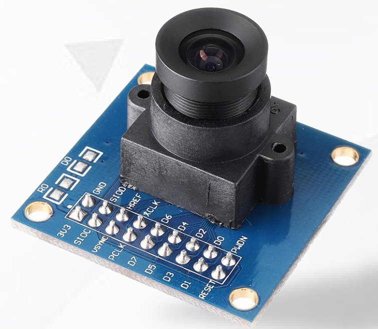


Episode
For each episode
Step
Initialize replay memory D with capacity N
Initialize the policy network with random weights θ
Initialize the target network with random weights $\theta' = \theta$
Initialized the preprocessed history sequence of states
Initialize the starting state (new episode begins)
Select action a_t according to policy (explore or exploit)
Apply action in environment
Observe reward r_{t+1} and next state s_{t+1}
is s_{t+1} terminal?
YES
NO
Perform pre-processing of state and next state
Pass batch of preprocessed states to policy network
Calculate loss between Qvalues and target Q-values
Update network weights θ using gradient descent
Every C steps, target network weights catch up to the policy network
For each step of the episode

2.4 SENSORS

Optical Camera, 2D LIDAR, SONAR

Optical Camera: Sealed Box with small hole (aperture) that allows light to reach a light-sensitive sensor. Captures RGB image frames.



LIDAR Technology: Pulses of light moving outwards, reaching objects and reflecting the light back to the receiver.

- Calculates distance by bouncing time difference
- Creates 2D map of estimated distances

SONAR Technology: Similar to LIDAR, except it uses sound waves instead of light pulses.

- Lower Accuracy, lower Cost than LIDAR.

3 IMPLEMENTATION

3.1 – UAV & Sensors

3.2 – Environment

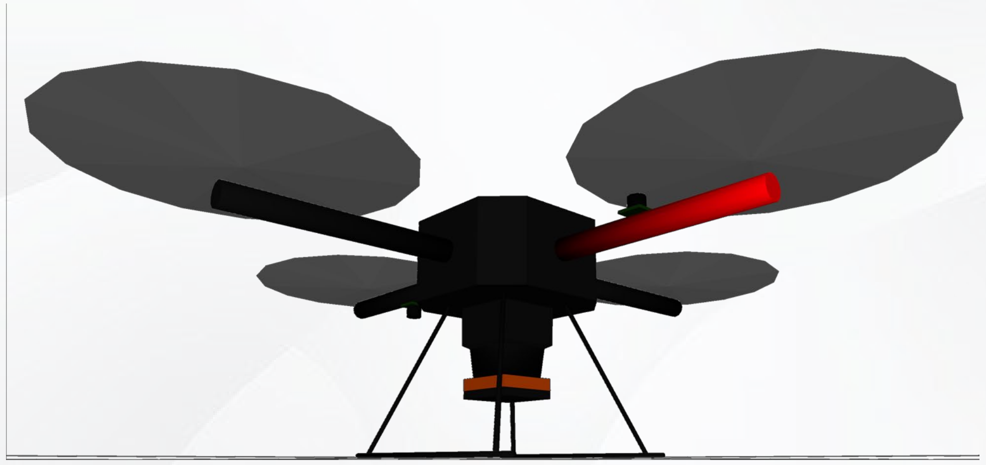
3.3 – Deep Reinforcement Learning Pipeline

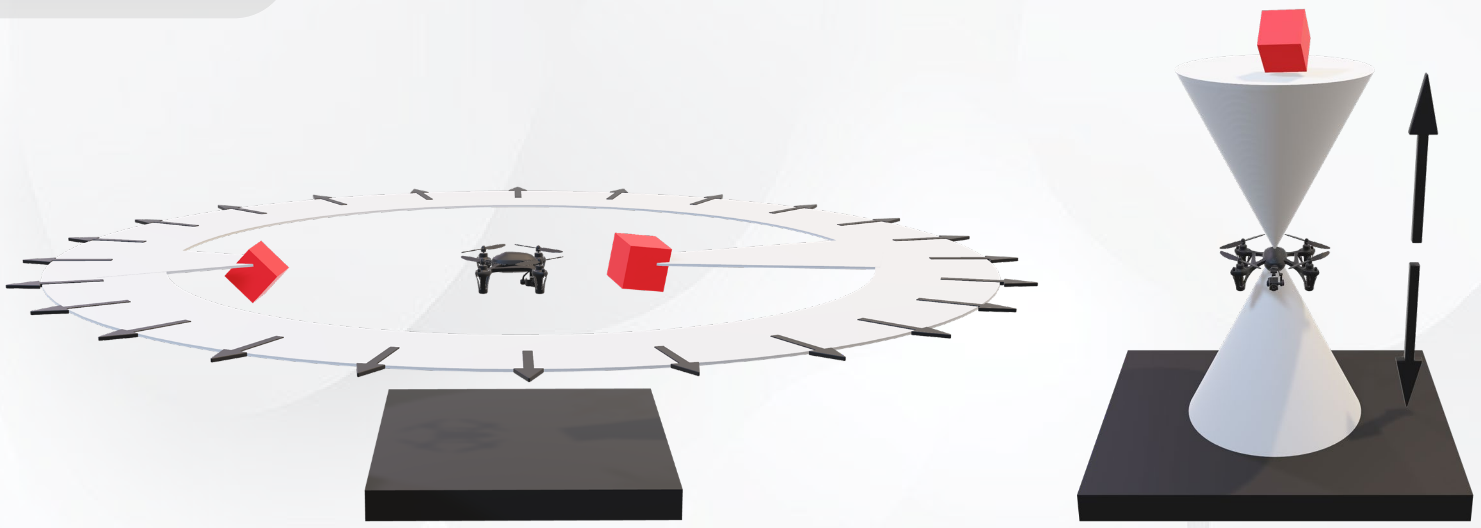
3.1 UAV MODEL & SENSOR

UAV Model Overview

MODEL: "hector_quadrotor" (category of general-purpose ROS packages related to modeling, control and simulations of UAV quadcopter systems). It contains:

- **Optical Camera**
  Marker detection
- **2D Laser Rangefinder (LIDAR)**
  Horizontal distance measurements
- 2 x **Ultrasonic Sensor (SONAR)**
  Vertical distance measurements





3.2 ENVIRONMENT

Environment Requirements

REQUIREMENTS

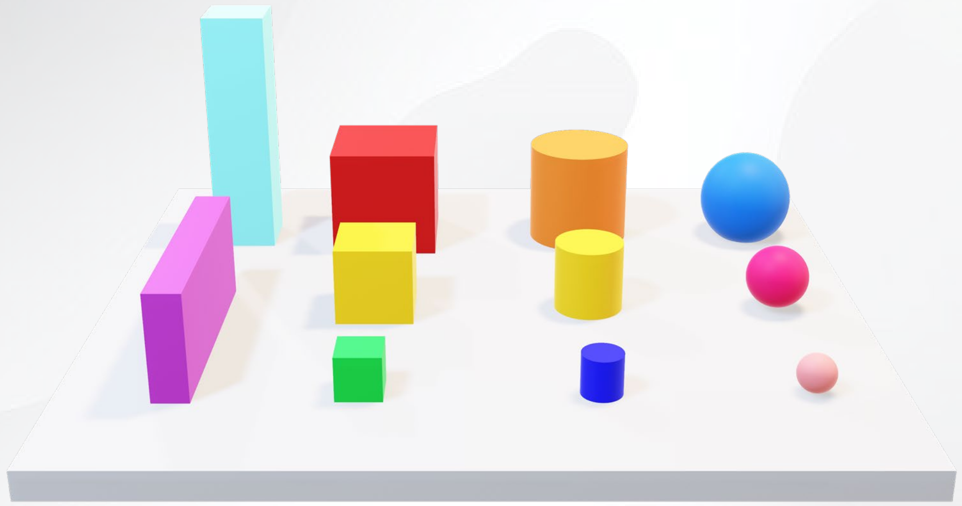
- **TRAINING SPEED:** Training process of a deep reinforcement learning algorithm requires many iterations in order to converge. **Training speed is a very important factor to consider.**
- **ROBUSTNESS:** We need a robust enough environment which offers a **variety of situations** for our agent to experience, without being overly complicated and thus affecting training speed.

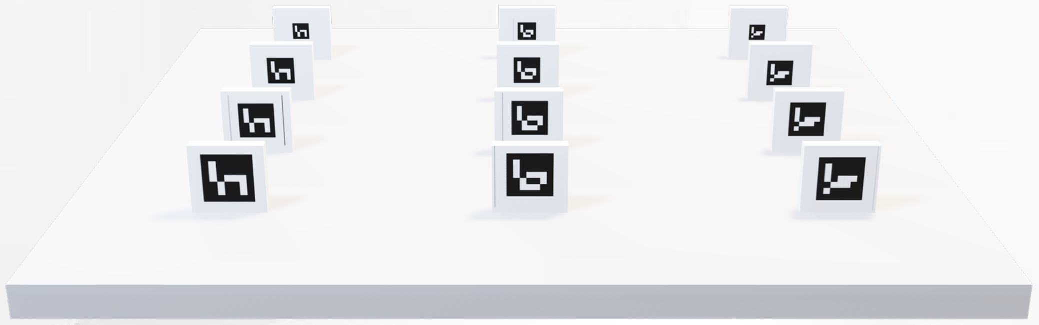
IMPLEMENTATION: a highly-configurable custom world generation system was created

- variable **obstacle** and **marker generation**
- Enables creation of **difficulty levels** for benchmarking

3.2 ENVIRONMENT

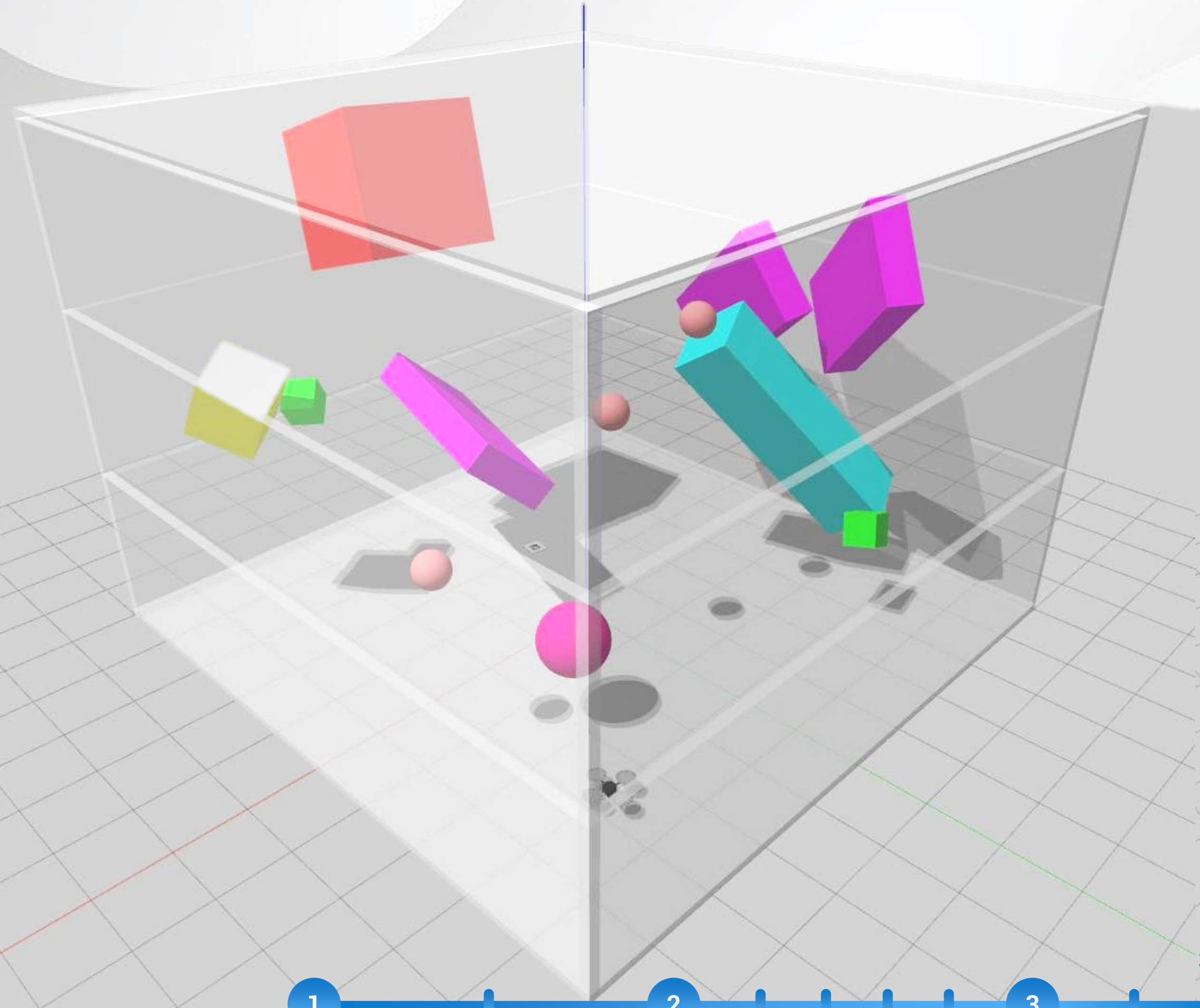
Environment **Variety**

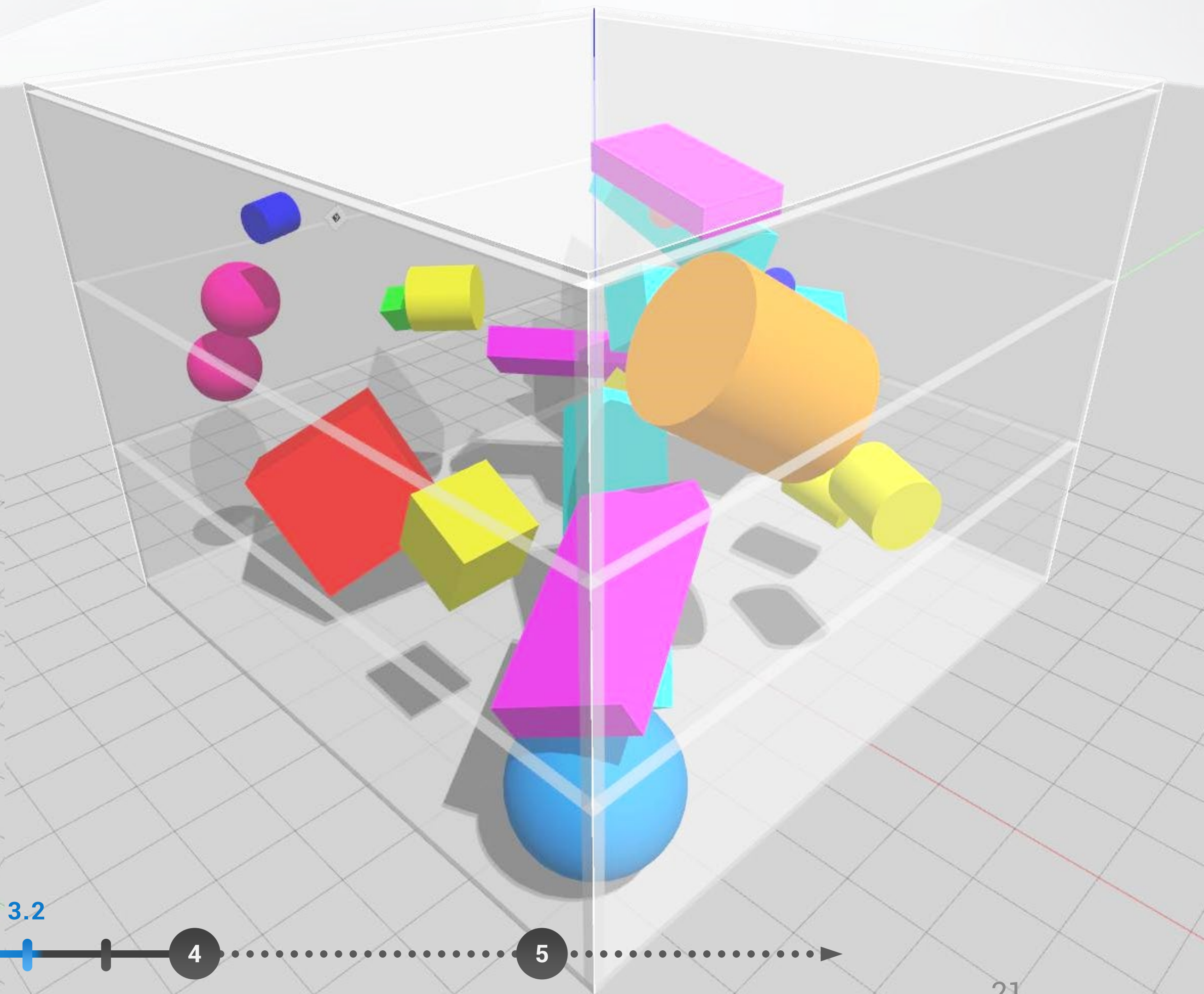




3.2 ENVIRONMENT

Environment **Examples** (Gazebo Images)





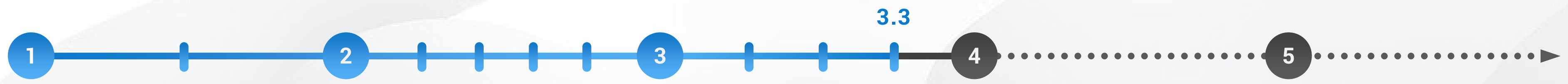
3.3 DEEP REINFORCEMENT LEARNING PIPELINE

States & Observations: Overview

OBSERVATION: Snapshot of the UAV's sensor measurements

OBSERVATION REQUIREMENTS: Two types of observations are required!

- **Target Information:** required to **find and approach** the target
- **Surroundings Information:** required to **avoid collisions** with obstacles and walls **(situational awareness)**



3.3 DEEP REINFORCEMENT LEARNING PIPELINE

States & Observations: Target Information

TARGET INFORMATION: Can be achieved in two ways!

APPROACH 1: Optical Camera (Real-World Scenario)

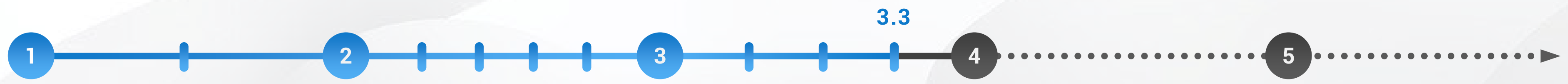
- **Scan Image Plane** for **ArUco Markers** using OpenCV detection algorithm.
- The algorithm estimates the **marker's pose** (position + rotation) relative to UAV's camera.
- Estimate final **relative pose** between UAV and marker using **Transformations**.

APPROACH 2: Simulator Information (Training purposes only)

- **Skip camera setup,** extract information **from the simulator itself**!
- **Isn't this Cheating?** → **No!** From RL perspective, we are only interested in training our neural network.

Marker position is now estimated. UAV's optimal approach point is defined as 2 meters (arbitrarily selected) in front of the marker!

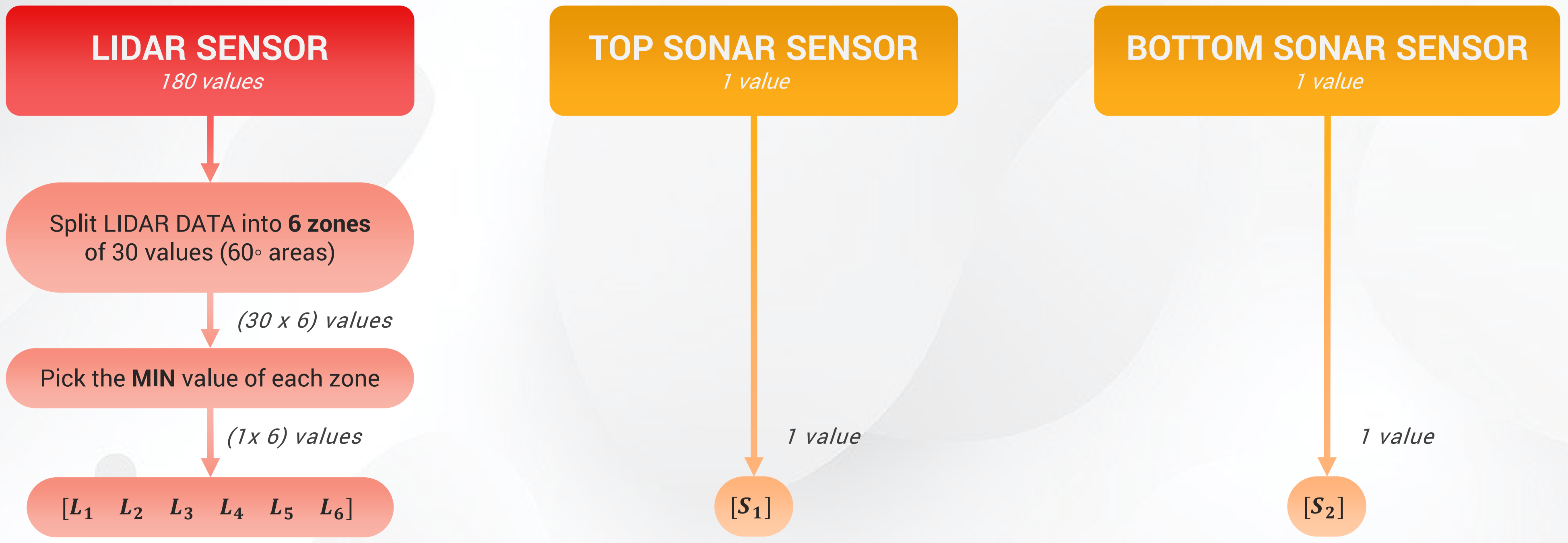
Target Information Vector: $T = [x_{delta} \quad y_{delta} \quad z_{delta} \quad yaw_{delta} \quad yaw_{global}]$ (5 values)



3.3 DEEP REINFORCEMENT LEARNING PIPELINE

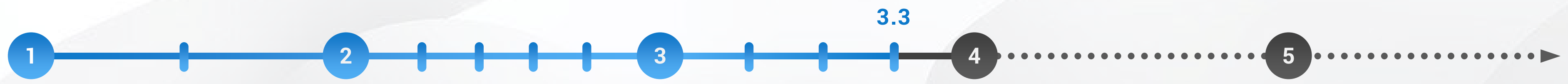
States & Observations: Surroundings Information

SURROUNDINGS INFORMATION: Contains **LIDAR** and **SONAR** data!



LIDAR SENSOR
180 values
Split LIDAR DATA into **6 zones** of 30 values (60◦ areas)
(30 x 6) values
Pick the **MIN** value of each zone
(1x 6) values
$[L_1 \quad L_2 \quad L_3 \quad L_4 \quad L_5 \quad L_6]$
TOP SONAR SENSOR
1 value
1 value
$[S_1]$
BOTTOM SONAR SENSOR
1 value
1 value
$[S_2]$

Surroundings Information Vector: $D = [L_1 \quad L_2 \quad L_3 \quad L_4 \quad L_5 \quad L_6 \quad S_1 \quad S_2]$ (8 values)



3.3 DEEP REINFORCEMENT LEARNING PIPELINE

States & Observations: Terminal States

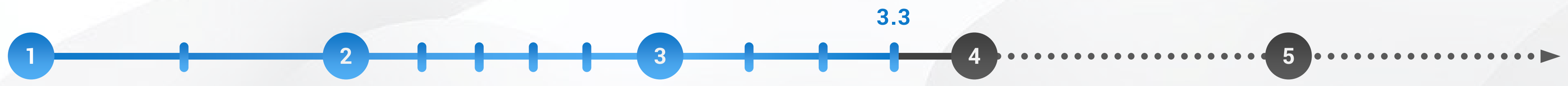
If a terminal state occurs, the environment is reset, and a new episode begins.

- **Mission Fatal:** if any distance measurement is less than 0.4 meters, we presume a collision has occurred!
  $Fatal = \exists D_i < 0.4, D_i \in D$

- **Mission Complete:** If the maximum distance of every dimension between the drone and the target point is less or equal than 0.8 meters, we presume a successful target point approach!
  $Complete = max(x_{rel}, y_{rel}, z_{rel}) \leq 0.8m$

A state is terminal if the drone's mission is either fatal or complete.
$Terminal = Fatal \vee Complete$

Time Expired: a cap of 5.000 maximum steps for each episode was implemented in order to avoid endless episodes.

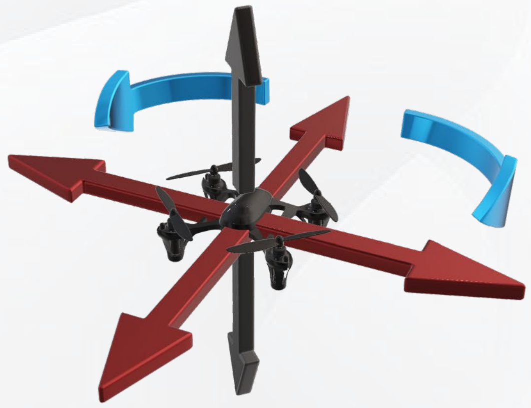


3.3 DEEP REINFORCEMENT LEARNING PIPELINE

Actions

ACTION SPACE: ~~Continuous~~ **Discrete** (a manual flight would involve analog velocity input of the UAV)

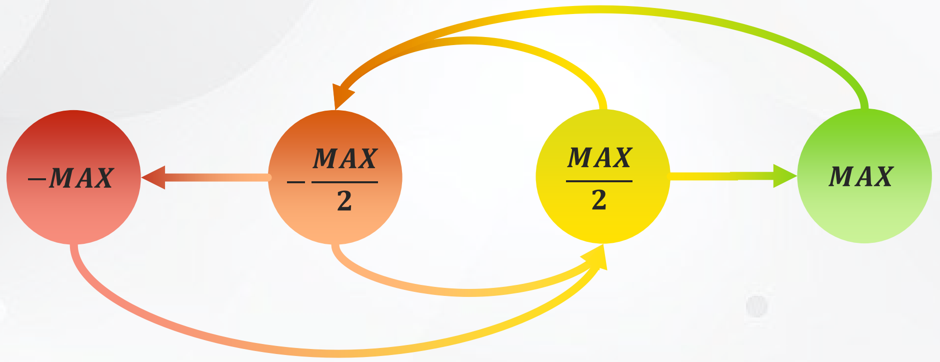
- **PROBLEM:** **DQN** and other algorithms **do not** usually **support continuous action spaces**.
- **SOLUTION:** **Classify actions** into eight (8) **discrete** options to control velocity of each axis.

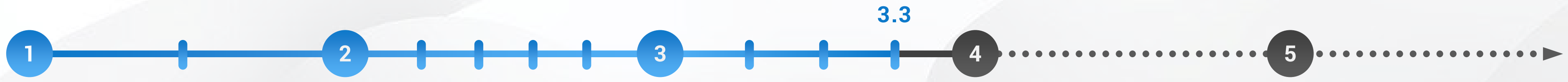


#	Action
0	Increase pitch
1	Decrease pitch
2	Increase Roll
3	Decrease Roll
4	Increase Throttle
5	Decrease Throttle
6	Increase Yaw
7	Decrease Yaw

FOUR-STAGE VELOCITY SYSTEM

- Each axis has a minimum and a maximum velocity. **Every action results in an increase/decrease in velocity of a particular axis by a specific increment.**
- **Four** (4) total **increment stages** for each axis. This allows the drone to delicately **fine-tune its velocity when it's near the target** for increased stability.





3.3 DEEP REINFORCEMENT LEARNING PIPELINE

Reward System: Overview

REWARD SYSTEM REQUIREMENTS: An effective reward system is crucial for training.

- **INFO:** Usually, the **trickiest** part of a RL problem, since there are no rules, no absolute restrictions and it is tied up with an environment's state space. Although there are some general guidelines to follow.
- ! We need to avoid sparse rewards.
- ! Rewards need to be clipped to a range of $[-1,1]$ to avoid instabilities to the neural network.

REWARD SYSTEM: Two (2) reward functions, each focusing on a specific aspect of the UAV's mission.

- **Wall Distance Reward (WDR)** → Surroundings Information
- **Velocity Direction Reward (VDR)** → Target Information

Total reward is produced by just summing up both reward functions: $\boldsymbol{R_t = WDR + VDR}$

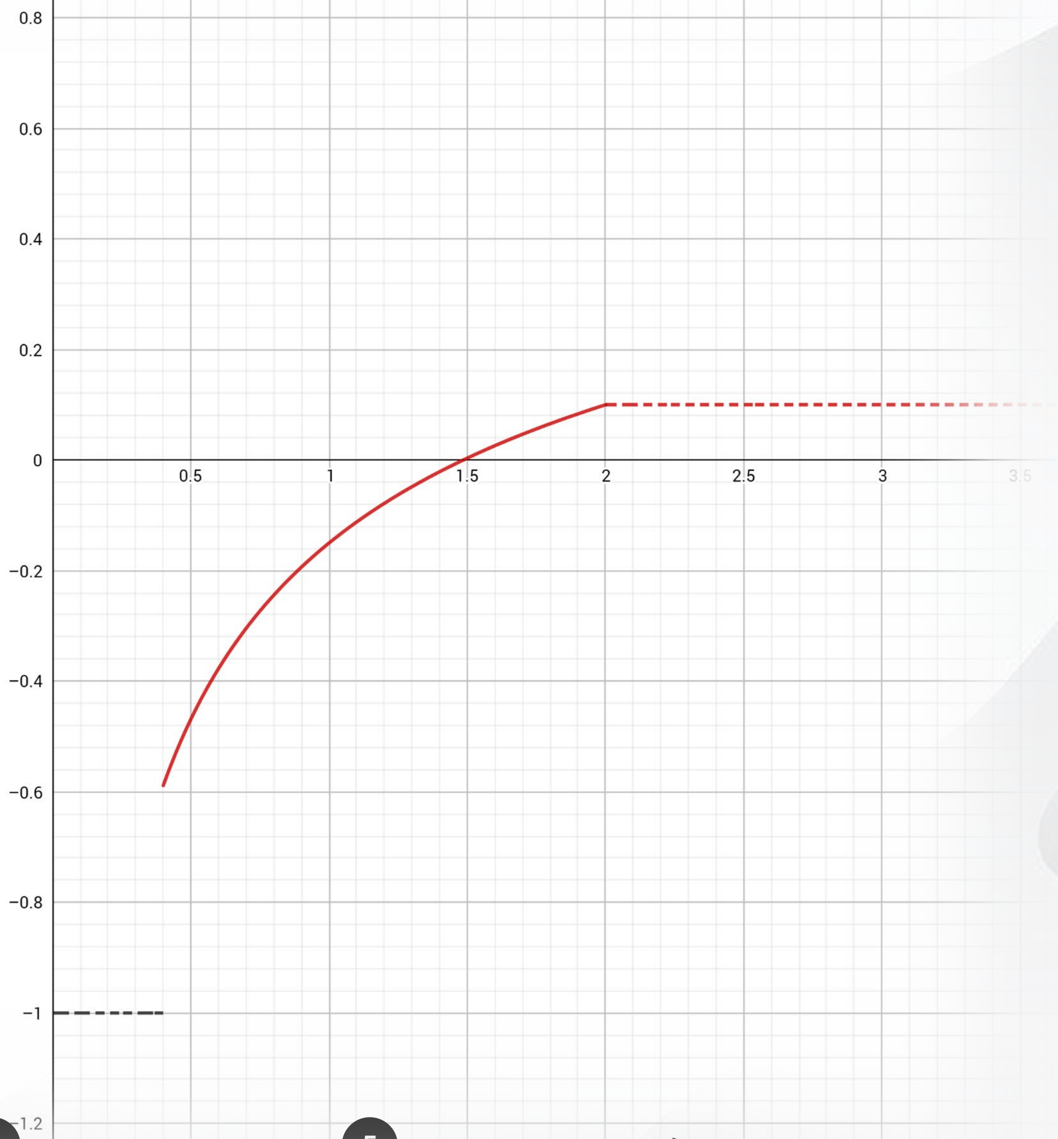
3.3 DEEP REINFORCEMENT LEARNING PIPELINE

Reward System: Wall Distance Reward

WALL DISTANCE REWARD: penalizes the agent for being close to obstacles and walls.

$$\boldsymbol{WDR} = \begin{cases} 0.1, & x > 2 \\ 1.01 - 1.16x^{-0.35}, & 0.4 \le x \le 2 \\ -1, & x < 0.4 \end{cases}$$

$$\boldsymbol{x} = \min(D), \ \boldsymbol{D} = [L_1 \quad L_2 \quad L_3 \quad L_4 \quad L_5 \quad L_6 \quad S_1 \quad S_2]$$



3.3 DEEP REINFORCEMENT LEARNING PIPELINE

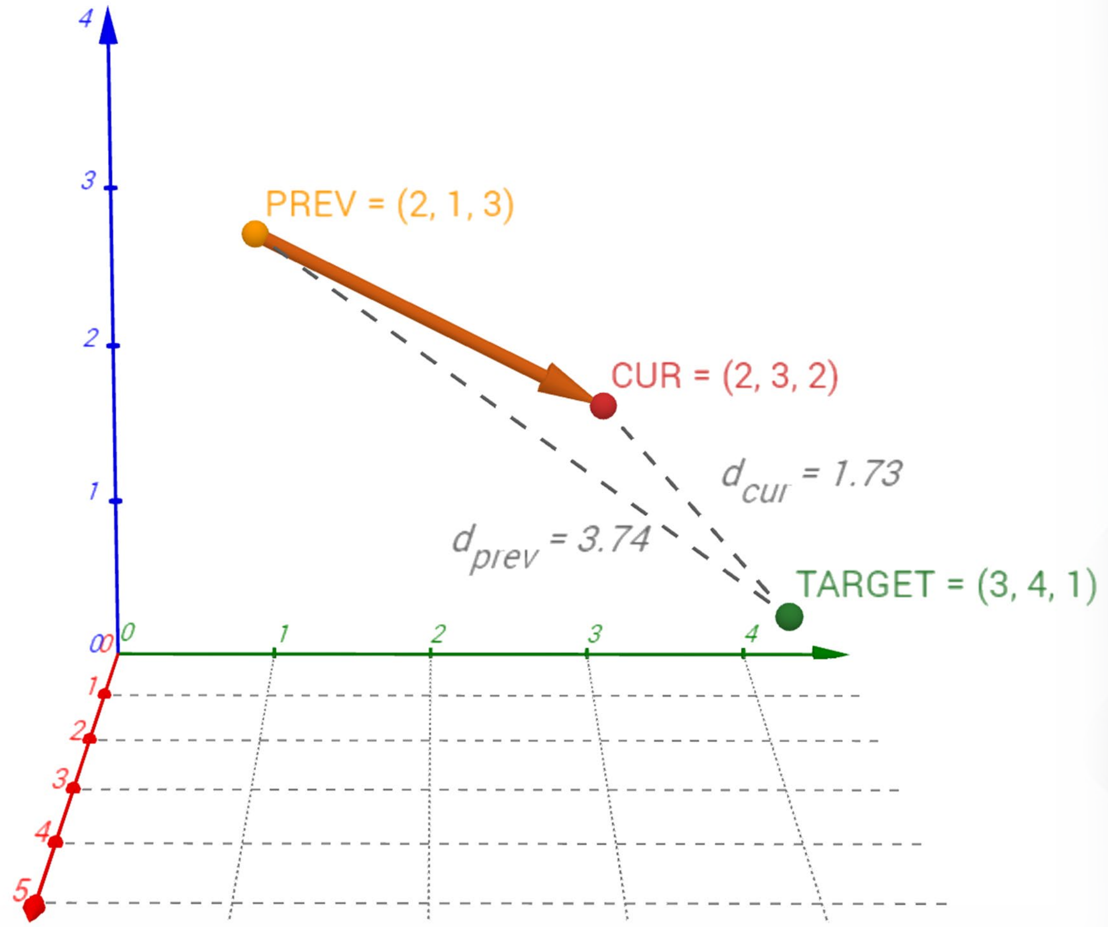
Reward System: Velocity Direction Reward

VELOCITY DISTANCE REWARD: rewards the agent for heading towards the target.

$$\boldsymbol{VDR} = 50\ (d_{delta,prev} - d_{delta,curr}) \qquad \textit{where}$$

$$\boldsymbol{d_{delta,prev}} = \sqrt{x_{delta,prev}^2 + y_{delta,prev}^2 + z_{delta,prev}^2}$$

$$\boldsymbol{d_{delta,curr}} = \sqrt{x_{delta,curr}^2 + y_{delta,curr}^2 + z_{delta,curr}^2}$$



3.3 DEEP REINFORCEMENT LEARNING PIPELINE

Deep Neural Network

NEURAL NETWORK REQUIREMENTS & OBSERVATIONS:

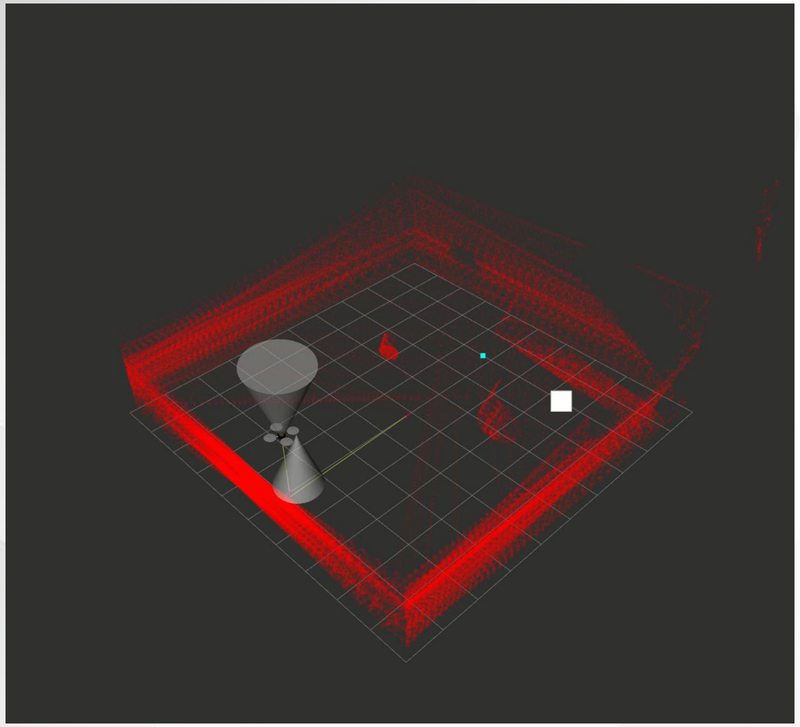
- The original DQN paper used entire RGB image frames as input.
- Our input data only consists of a small number of parameters (13 scalar values x 4 frames = 52 total parameters).
- Our data is unrelated & tabular (lacks structure). → **Best Option:** Fully Connected Layers (Dense)

Layer Name	Layer Type	Input Shape	Output Shape	Activation Function	Initializer	Parameters
input_1	InputLayer	(-, 4, 13)	(-, 4, 13)	-	-	0
flatten	Flatten	(-, 4, 13)	(-, 52)	-	-	0
dense	Dense	(-, 52)	(-, 256)	relu	he_uniform	13.568
dense_1	Dense	(-, 256)	(-, 512)	relu	he_uniform	131.584
dense_2	Dense	(-, 512)	(-, 128)	relu	he_uniform	64.664
dense_3	Dense	(-, 128)	(-, 8)	linear	he_uniform	1.032

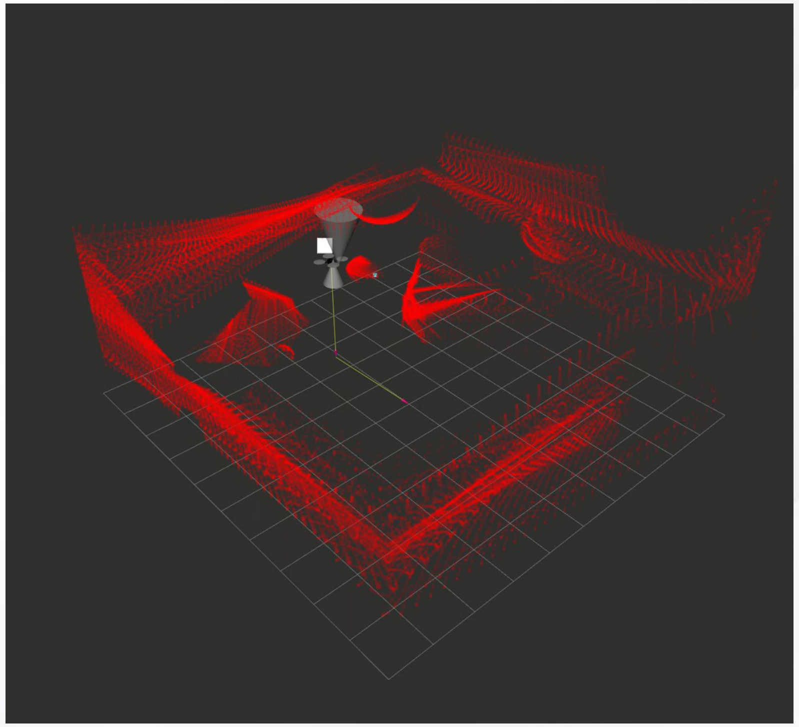
3.3 DEEP REINFORCEMENT LEARNING PIPELINE

Demo

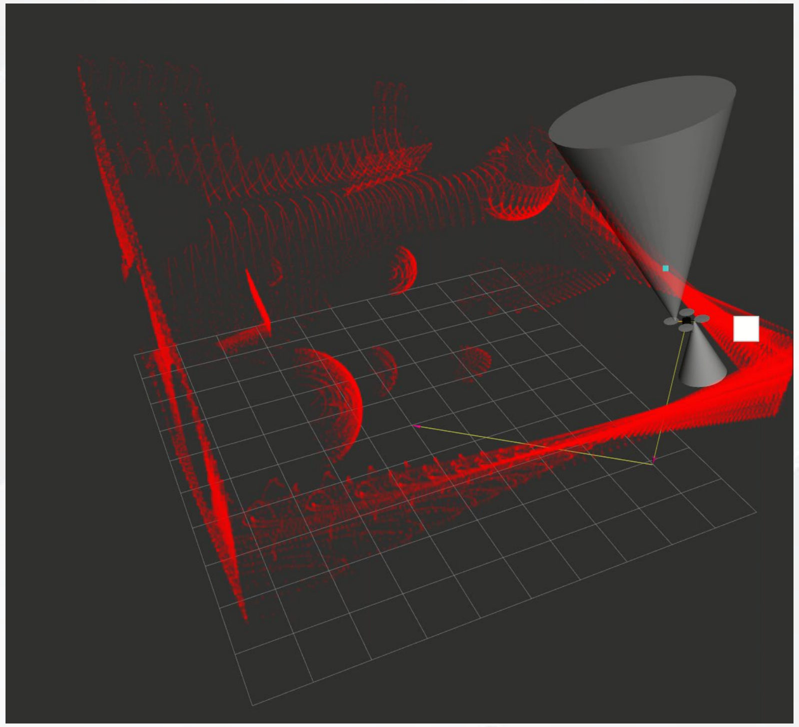
Before Training



After Training



Endless Episode Example



4 EXPERIMENTS

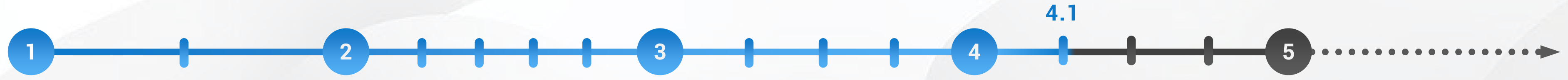
4.1 EXPERIMENTAL SETUP

Timing Configurations

TIMING CONFIGURATIONS: Six (6) training configurations were created based on training length

- Several parameters are training-time-dependent and require adjustments in order to become more suitable.
- The two largest configurations "**Large**" and "**Marathon**" are utilized. Shorter are preferred for **internal use and debugging**.
- Having discrete timing options can help standardize benchmarking results.

#	Configuration Name	Total Steps	Total Duration	Time Multiplier	Epsilon Decay Steps	Memory Size	Start Train Steps	Train Frequency	Target Update Frequency
1	Demo	5K	5m	1	1.9K	1K	200	20	500
2	Instant	30K	25m	6	25.5K	15K	5K	100	5K
3	Quick	125K	1h, 30m	25	100K	50K	15K	250	25K
4	Standard	400K	5h	80	320K	150K	30K	500	80K
5	**Large**	**1.5M**	**16h**	**300**	**1.2M**	**600K**	**80K**	**750**	**187.5K**
6	**Marathon**	**3M**	**32h**	**600**	**2.4M**	**1M**	**100K**	**1K**	**500K**



4.1 EXPERIMENTAL SETUP

World Profiles

One of the goals of this project is to evaluate the agent's behavior over a number of scenarios with varying difficulty.

WORLD PROFILES: Five (5) **world profiles of increasing difficulty** were created.

- World difficulty is dependent on UAV / marker spawning locations and obstacle quantity.

#	World Profile Name	UAV Spawn Location	Marker Spawn Location	Obstacle Spawn Location	Obstacle Quantity	Timing Configuration Used
1	**Ridiculous**	fixed	fixed	-	0	"Large"
2	**Easy**	random	fixed	-	0	"Large"
3	**Medium**	random	random	-	0	"Large"
4	**Hard**	random	random	random	6	"Marathon"
5	**Extreme**	random	random	random	12	"Marathon"

4.1 EXPERIMENTAL SETUP

Metrics & Charts

unreliable & non-practical 🔴
useful information but not always practical 🟡
useful & very practical information 🟢

TOTAL REWARD (TR): Total accumulated reward during training. Most popular metric. Not always reliable.

TOTAL MARKERS FOUND (TMF)

AVERAGE SAMPLE REWARD

LOSS (LO): loss occurred during training

RELATIVE MARKER APPROACH (RMA): When collision occurred, how close was it? $\frac{\textit{distance from target at death}}{\textit{distance from target at spawn}}$

EPISODE OUTCOMES OF LAST 100 EPISODES:

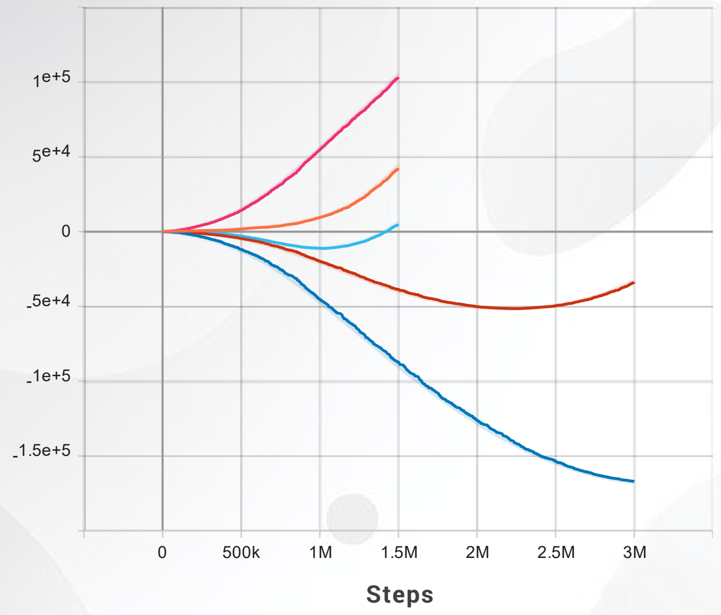
- **Collision Rate Last 100 (CR100):** measures how frequently collisions occur (mission fatal).
  $\frac{\textit{total collisions last } 100}{100}$
- **Markers Found Rate Last 100 (MFR100):** measures how frequently markers are found (mission complete).
  $\frac{\textit{total markers found last } 100}{100}$
- **Episode Expire Rate Last 100:** measures how frequently endless episodes occur. $\frac{\textit{total expired episodes last } 100}{100}$

4.2 TRAINING RESULTS

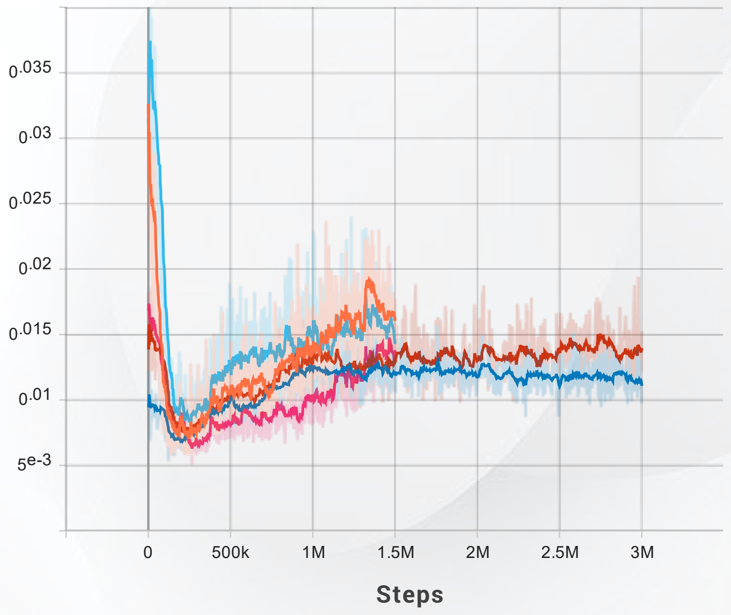
Chart Highlights

Ridiculous Difficulty, Large Training ●
Easy Difficulty, Large Training ●
Medium Difficulty, Large Training ●
Hard Difficulty, Marathon Training ●
Extreme Difficulty, Marathon Training ●

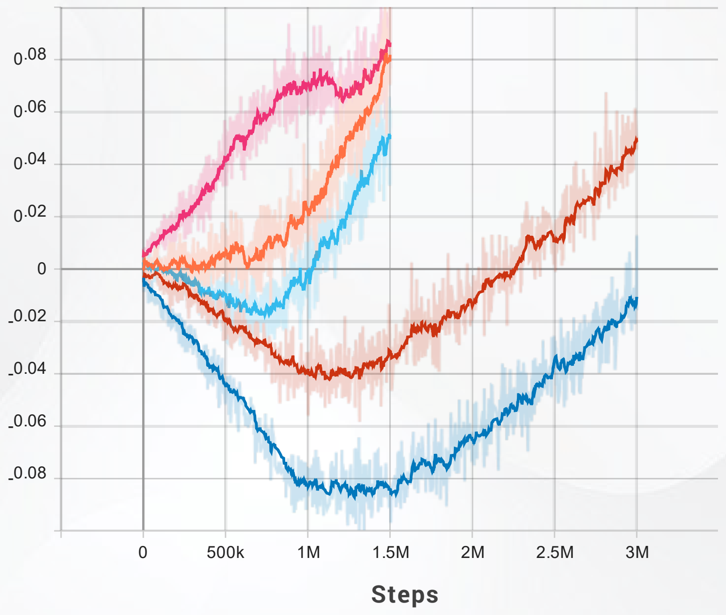
TOTAL REWARD (TR)



LOSS (LO)



AVERAGE SAMPLE REWARD



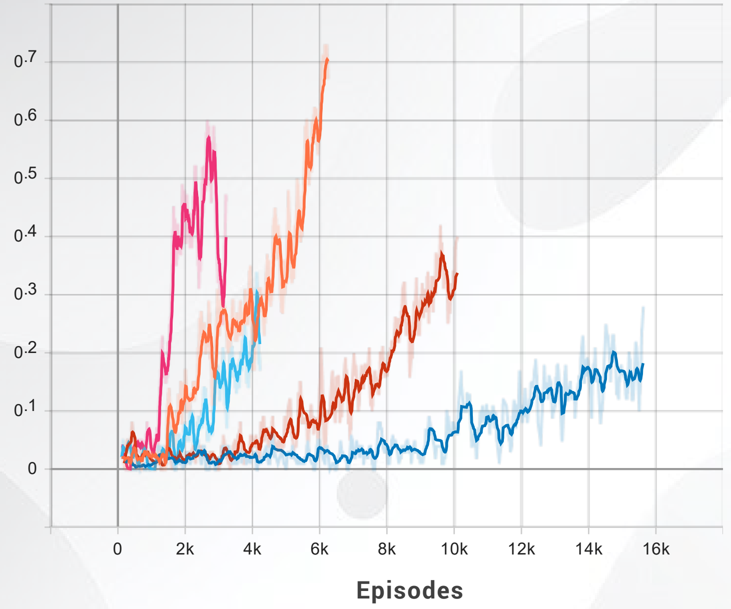
1 2 3 4 4.2 5

4.2 TRAINING RESULTS

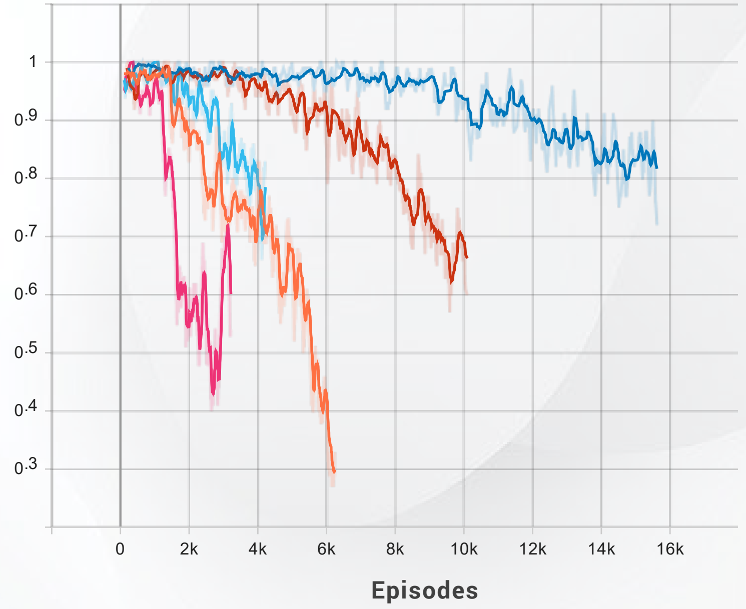
Chart Highlights

Ridiculous Difficulty, Large Training ●
Easy Difficulty, Large Training ●
Medium Difficulty, Large Training ●
Hard Difficulty, Marathon Training ●
Extreme Difficulty, Marathon Training ●

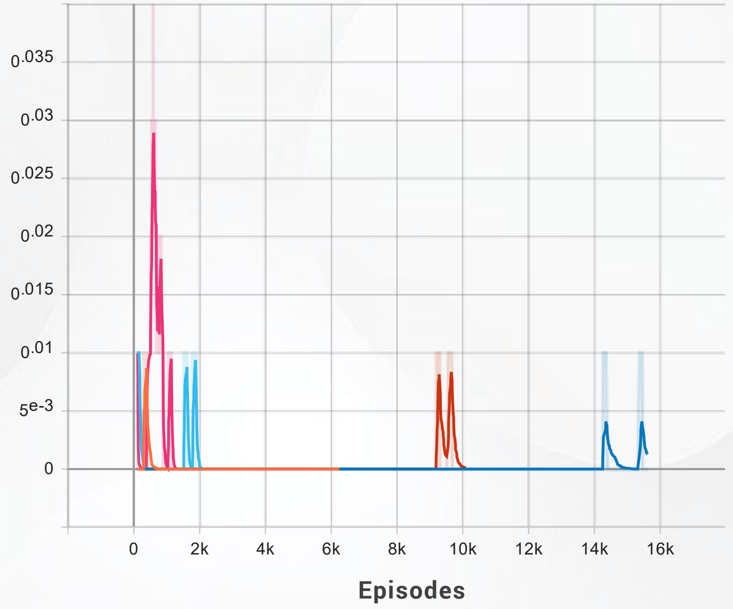
Marker Found Rate Last 100 Episodes (MFR100)



Collision Rate Last 100 Episodes (CR100)



Episode Expired Rate Last 100 Episodes

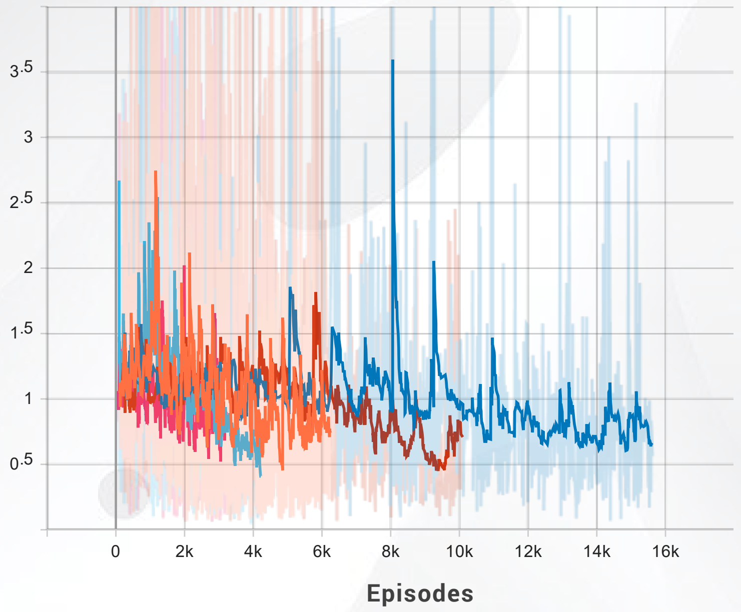


4.2 TRAINING RESULTS

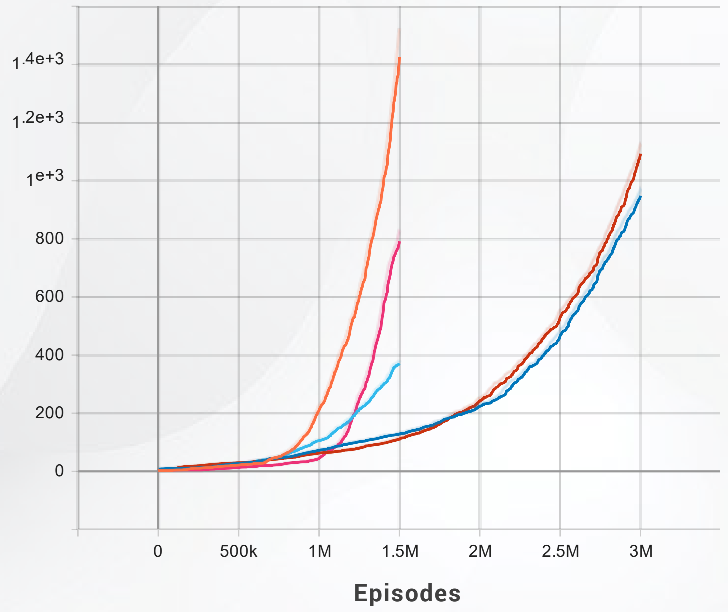
Chart Highlights

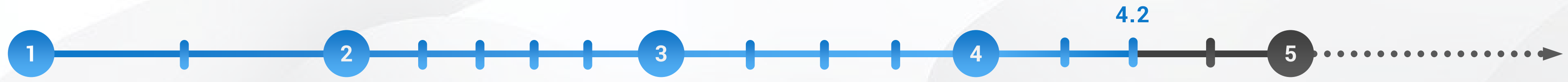
Ridiculous Difficulty, Large Training ●
Easy Difficulty, Large Training ●
Medium Difficulty, Large Training ●
Hard Difficulty, Marathon Training ●
Extreme Difficulty, Marathon Training ●

Average Relative Marker Approach (RMA)



Total Markers Found





4.3 RESULT DISCUSSION

Is our agent really learning?

World Difficulty Level	Training Steps	Marker Found Rate Last 100 (MFR100)	Collision Rate Last 100 (CR100)	Average Relative Marker Approach (RMA)	Total Reward
Ridiculous	1.5M	0.46	0.54	0.79	105.450
Easy	1.5M	0.71	0.29	0.76	44.446
Medium	1.5M	0.22	0.78	0.64	6.145
Hard	3M	0.4	0.6	0.65	-32.784
Extreme	3M	0.28	0.72	0.77	-167.170

OBSERVATION 1: as training progresses, collision rate (CR100) always decreases, marker found rate (MFR100) always increases. Increasing rate of MFR100 means exponential increase of total markers found (TMF).

OBSERVATION 2: Every training session concluded with an average RMA of <0.8 which indicates that the agent was consistently approaching the target before colliding.

OBSERVATION 3: ”Ridiculous” performed surprisingly poor. “Easy” managed an impressive MFR100 of 71% completing its mission more than 1500 times. “Medium” was slow to catch up, required more training and was outperformed by “Hard” and “Extreme”.

RESULTS OVERALL: Results appear to be promising. In every difficulty, metrics show a **poor initial agent behavior** and a **significantly improved version by the end** of each training session. Results can be improved with longer training sessions.

5 CONCLUSION

5.1 – Summary

5.2 – Limitations & Future Work

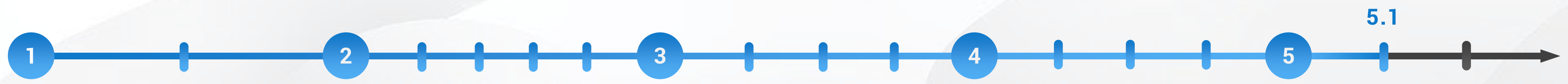
5.1 SUMMARY

What did we learn today?

SUMMARY: Proof-of-concept mapless approach to UAV **autonomous navigation** tasks in **fully unknown 3D environments**. DQN **incorporates deep learning techniques** into a well-defined reinforcement learning problem (MDP) and integrates several key features including a **replay memory** and a **target network**.

RESULTS: Five (5) experiments were conducted to evaluate agent's performance.

- Results suggest that the agent can successfully learn to navigate in the environment and avoid obstacles.
- Proof that DQN can be applied in a large variety of custom environments (not just ATARI games).



5.2 LIMITATIONS & FUTURE WORK

How can we improve our results?

TRAINING SPEED OPTIMIZATIONS:

- **Prioritized Experience Replay:** In this DQN implementation, experience tuples are uniformly sampled from the replay memory. Prioritizing samples with higher loss values will result in faster network convergence.
- **Drone Swarms:** Multiple parallel drone training will increase training speed. Each UAV would separately send input parameters to a shared neural network, allowing for faster exploration and sample extraction.
- **Better Pause-Resume Gazebo System:** Current implementation relies on services to pause and resume the simulation. Slows down training process by 30%.

TRAINING QUALITY OPTIMIZATIONS:

- **Environment Representation:** Current obstacle variation is limited to primitive shapes to save computer resources. This is a trade-off between higher-quality complex objects and training speed.

COST-SAVING OPTIMIZATIONS:

- **Replace LIDAR:** In current implementation, LIDAR is overkill. Replace LIDAR with six SONAR sensors around UAV's frame.

THE END

Thank you for your time