Research Progress Update for Student Research Project 2018

Tutor: Hadi Samer Jomaa

Team:
Benjamin Hogstad
Krithika Murugesan
Ramya Raghuraman
Michael Ruchte
Mohana Nyamanahalli Venkatesha

June 7th, 2018

Contents

Problem Setting

Our goal is to apply current state of the art techniques to train a team of autonomous drones (multi-agent) to explore an unknown environment while avoiding obstacles as well as other drones as efficiently as possible in order to find a given target. This setting leads to a wide range of problems as it encompasses both Deep Learning (Neural Networks) and Reinforcement Learning.

Cooperative exploration using Deep Learning for multi-agent systems can be broken into four parts {Network Structure, Update Technique, Action-selection technique, and reward function}.

Network Structure:
A neural network can come in various structures and impacts the convergence of the algorithm.

Update Technique:
The update technique allows the neural network to adjust its weights based on the outcome of an agent's action, current and previous state.

Action-selection technique:
The process to promote early exploration in each agent's navigation and slowly pursuing exploitation of the learned optimal solution.

Reward/cost Function:
The goal or result in which the network is evaluated against.

In a cooperative multi-agent setting, the largest problem is synchronisation and coordination. Every agent in the team must synchronize with other drones to avoid work that has already been done. This implies that the drone’s individual decision-making, i.e. path planning depends not only on its own past decisions and its current environment, but also on decisions made by other drones. At the budding stage of the project, working with a single agent and target and later developing the algorithm for a multi agent setting would provide efficient results.

The primary focus right now is incorporating Asynchronous Advantage Actor – Critic (A3C) which replaced the previously existing deep – Q learning technique by replacing the experience replay with a series of actors.

The most important part of the algorithm is the state, action and reward based on which the agent is penalized if it comes in contact with a drone/obstacle and if it goes out of bounds (i.e out of grid/map) or rewarded if it captures the location of the target. There are four possible actions {Left, Right, Up, Down}. All agents/agent will be made to initialize in any of the 4 corners of the map. The episode will end when an agent arrives at the target location. The algorithm will be optimized for finding the target as quickly as possible. A series of scenarios will be posed to the proposed techniques such as varying the number of agents, randomizing the position and number of obstacles.

Currently a working model of Actor Critic along with deep Q algorithm has been developed and also various hyper parameters are tuned in order to minimize the loss. A simple and clear visualization is also achieved to capture the movement of the drone as it navigates through the obstacles and reaches the target.

Assumptions:

- Search area is limited beforehand by the operator
- Target is easy to identify – if a drone “sees” the target location, it automatically discovers it

- Exploration is time-critical
- All drones are able to communicate to all other drones at all times(not yet implemented)
- Drones are able to fly stable in the given direction and are already able to avoid small obstacles (trees, power poles, etc.) by themselves.

Out of scope:

- Technical details about drone hardware and communication
- Optimization with regard to most efficient use of hardware resources

Visualization and Simulator:

The training environment is a basic simulator implemented using python, which aids in data collection for training. Each cell in the grid can be an empty space or an obstacle that has to be avoided, the drones should navigate to reach a cell in the grid where the target is positioned. In the initial stage of implementation, we consider the surrounding to be a 2D grid and the drone is capable of making one of the four actions namely, up, down, right and left with no obstacles.

The grid is represented as a 2D numpy array, the obstacles and the target are placed in the grid such that there is at least one possible path to reach the target by the drones. To ensure this a flood fill is implemented initially, also random seeds are set to simulate the same environment for different training iterations. Each drone is allowed movement across one of the four available states, at each episode of training the current grid is returned to the visualization block. The drones can see only the consecutive elements in the directions it can move, that is the corresponding right, left, top and bottom cells of the drone is it's surrounding. A flattened grid is used as input to the neural network.

The visualization block uses Tkinter python package to show the navigation of the drone across the grid. The obstacles are represented as black color, the path traversed by the drone in yellow and the empty spaces in white. It shows the how the drone steers towards the target by making decisions based on the learning algorithm. It also incorporates the exploration map of the drones.

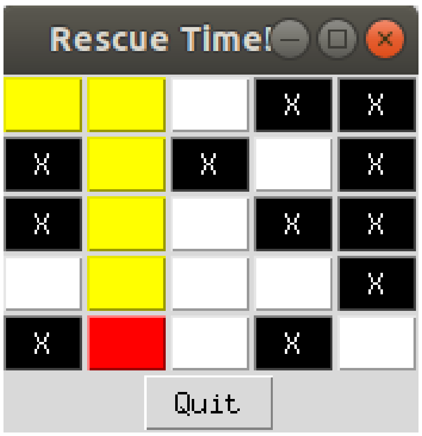


Figure 1 : Snapshot of the visualization

For more realistic environments, two simulators are commonly used in robotics simulation: Gazebo and AirSim. We will focus on the latter which is a free to use, open source and a well drone simulator by Microsoft based on the Unreal engine and it is under active development. It exposes APIs to control a drone and to evaluate its state (e.g. position, velocity, GPS location, camera view, etc) using Python and other languages. We have not implemented this in this phase and may be implemented at a later phase.

The gazebo simulator on the other hand is too complex for our approach, as it allows for designing a complete robot and test firmware against it. This is meant to be used by robot firmware developers, which want to simulate an indoor environment for their future robots rather than just simulating an environment for Reinforcement Learning. Furthermore it is very badly documented and cumbersome to set up.

Hyperparameters:
The hyperparameters regarding the simulator are common for the following implementation and used as specified here.

Variable	Value	Module	Explanation
gridsize	(5, 10)	simulator	The size of the world (the grid in the simulator
obstacles	0	simulator	Simulation runs without obstacles
max_iterations	25	learning	Maximum number of steps the drone is allowed to do in each epoch. Obtained by gridsize.x * gridsize.y / 2. Must be high enough to find the target.

State of the Art

As this is a new and quickly changing field, the state of art is evolving rapidly. Below lists the current state of art for each aspect of the Cooperative exploration with a multi-agent system.

Network Structure

[4] were the first ones who were able to use one Algorithm and a unique set of hyperparameters to successfully play seven different Atari games. They developed a variant of the Q-Learning approach where the Optimal-Action-Values (Q-Values) are estimated by a deep convolutional neural network based on raw pixel input (*Deep-Q-Network*). This way, they are able to generalize from the training because the CNN works as a function approximation, instead of calculating and updating the Q-Values directly.

Deep Q Learning (implementation)

Deep Q-Learning is the use of a neural network to approximate the value function used in Q-learning. The network inputs are states of the simulator, which consist of the drone location and a discovery map (refer to section XX). Evaluating the inputs, the network estimates the value of each possible action. This estimate is a combination of the reward as well as a factor of the next expected reward. The action with the maximum value is chosen and the next state is evaluated.

To create a baseline for the experiment, a simple one-step Q-learning is implemented which utilizes the following loss function:

$$L_i(\theta_i) = (r + \gamma\, max\ Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))^2 + \lambda\, log(Q(s; \theta_i)) \cdot Q(s; \theta_i)$$

Entropy regularization was also applied by taking the softmax of the action values, multiplying them by the log of themselves and summing them.

The process to learn the network uses the current q-value to train updating its estimate to correctly predict the reward and a factor of the next q-value. One-step refers to how many future states, the loss function takes into consideration and the current q-value needs to estimate.

The implemented one-step Deep Q-network, it has the following structure:
inputs: [100,1] integer array

hidden layer 1: 100 nodes, activation function: ReLU
hidden layer 2: 50 nodes, activation function: ReLU
outputs: 4 (one for each action)

Hyperparameters:
The hyperparameters apart from network structure are specified as follows:

Variable	Value	Module	Explanation
e_greedy	**0.8**	**learning**	**Probability of choosing an action randomly during learning to increase exploration of the map**
learning_rate	**0.0001**	**learning**	**Learning rate to the optimizer**
EPISODES	**100,000**	**learning**	**Number of epochs performed**
val_decay	**0.9**	**model**	**Importance of next value on network output**
reg_factor	**0.2**	**model**	**Importance of regularization of the policy using entropy**

As the values of the network are only updated for a given action-state pair, other pairs are learned indirectly. This leads to a slow learning process as each action-state pair needs to be seen to properly evaluate it and train on it. To encourage exploration, a simple E-greedy strategy was deployed. For action-states that are often seen (**Figure 2**), the defined Q-values can be learned quickly, however, for those seen less often, it will take longer (**Figure 3**). Note that the states themselves vary beyond that of the drone location, the number of times a cell has been visited also adds variation.

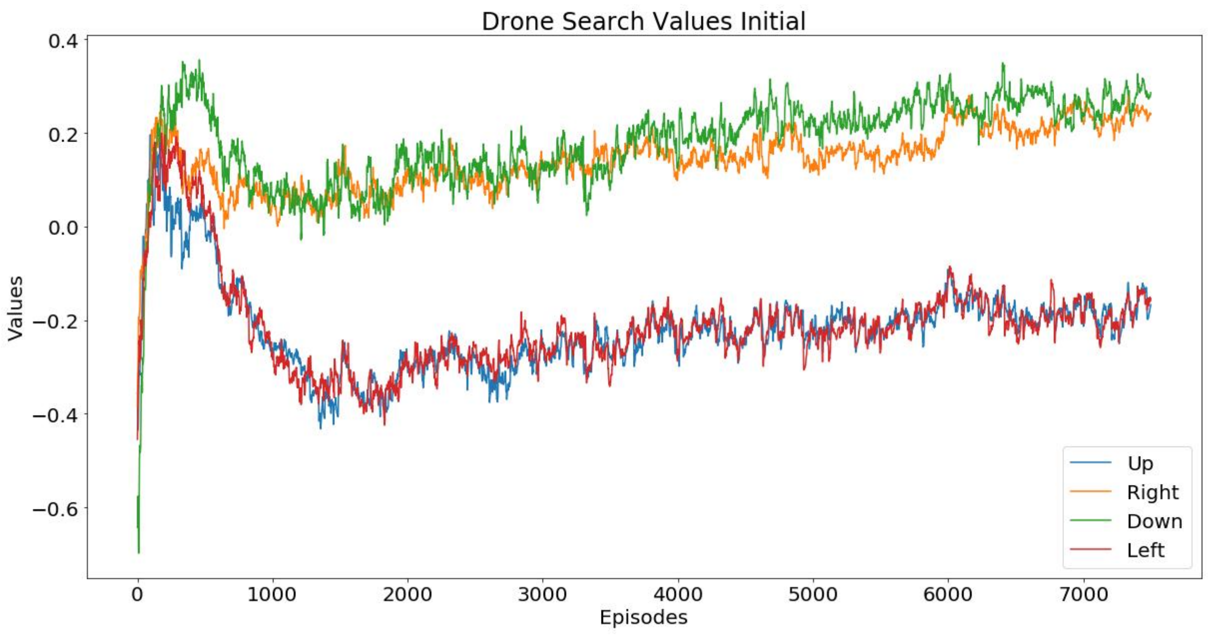


Figure 2: The state is in the starting location in the top left corner of the grid. The UP and LEFT q-values should be equally low as they lead to an out of bounds reward (highest penalty) and the RIGHT and DOWN Q-values should be equally higher (both within bounds).

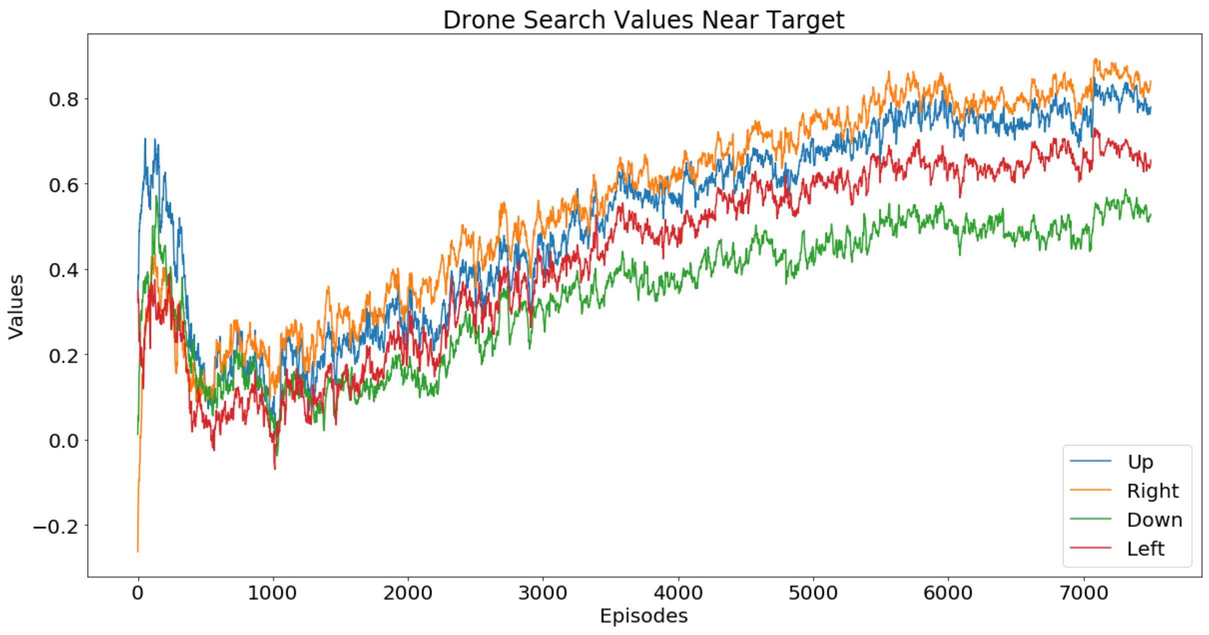
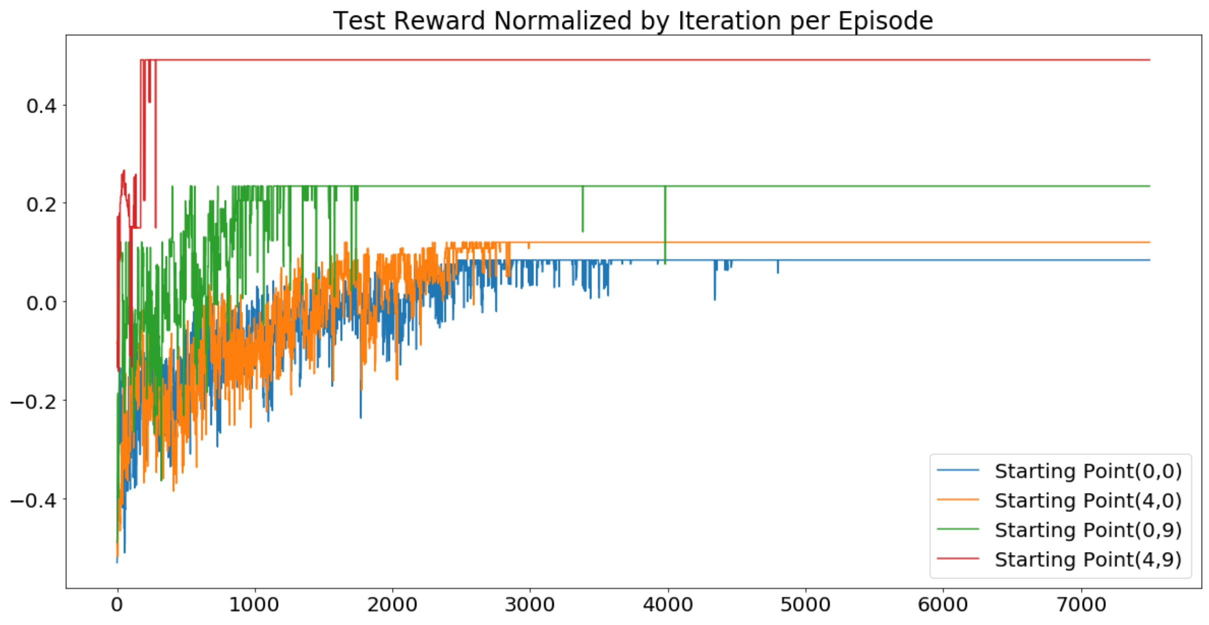


Figure 3: The state is near the Target location and is infrequently visited during training. The UP and RIGHT q-values should be equal and high as they lead to discovery of the target. The DOWN q-value should be the lowest as it leads to an out of bounds reward (highest penalty).



After 5000 episodes, the network can be seen to converge at all starting locations.
Figure 4. Convergence of Test simulations without obstacles.

When tested with 8 obstacles, the network was still able to find the optimal path.

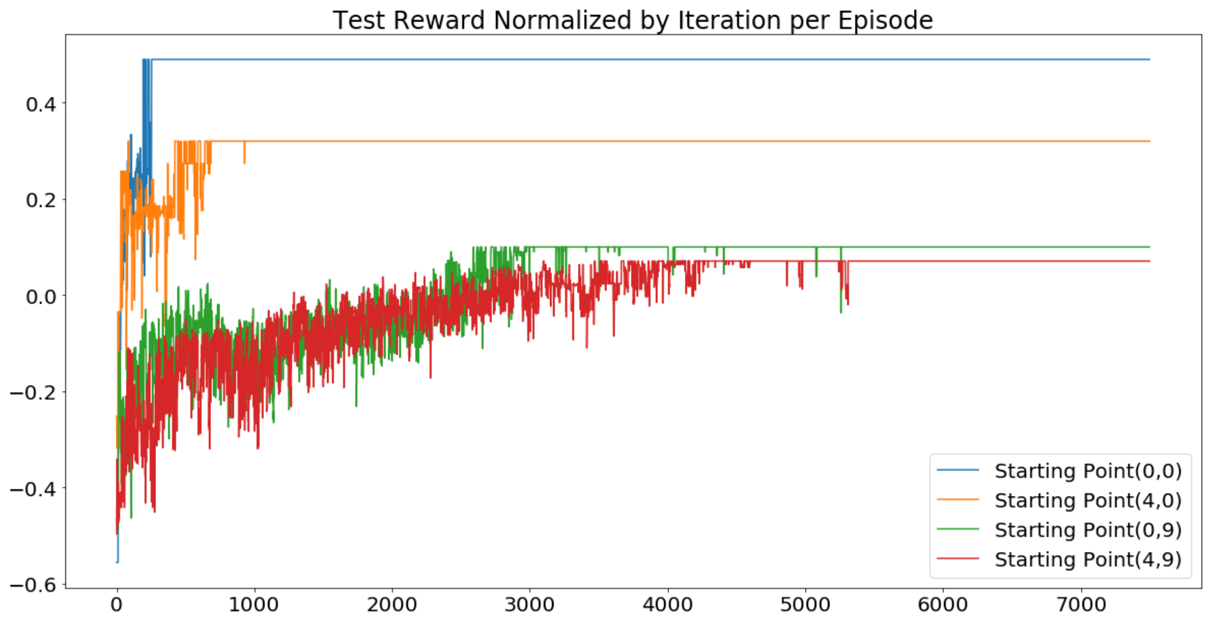


Figure 5. Convergence of Test simulations with obstacles.

Asynchronous Actor Critic

[7] introduced asynchronous variants of one-step Sarsa, one-step Q-Learning and advantage actor-critic (A3C) with lesser resources. This is achieved by using multiple CPU threads on same machine and multiple actor-learners which explore different parts of environment. Each actor-learner calculates its update gradient and sends to a master worker which averages the gradients and applies the update to the global network. The global network is then distributed out to all actor-learners for another episode of learning.

Different exploration policies may be used for each actor-learner, replay memory is not applied because parallel updates by multiple learner-actors is less likely to be correlated in time than a single agent applying online updates. All the techniques perform better on Atari domain with A3C outperforming the others by a huge margin.

Actor critic implementation

As basis for A3C, a normal actor critic implementation is needed.

Currently we are working on a rather simplified network for the actor critic approach to analyse its behaviour. Instead of having one network for each of the critic and the policy, we have a combined representation as commonly used in the literature [7].

Therefore we have one hidden layer and one output layer, where the softmax of 4 nodes represents the policy (probabilities for up, right, down, left) and the fifth node represents the critic output. All layers are fully connected layers with *relu* activation function.

Hidden layer:

- 100 Input nodes (the features delivered by the simulator)
- 30 Output nodes. This has been chosen arbitrarily

Policy output layer

- 30 Input nodes

- 4 Output nodes

Critic output layer
- 30 Inputs
- 4 Outputs

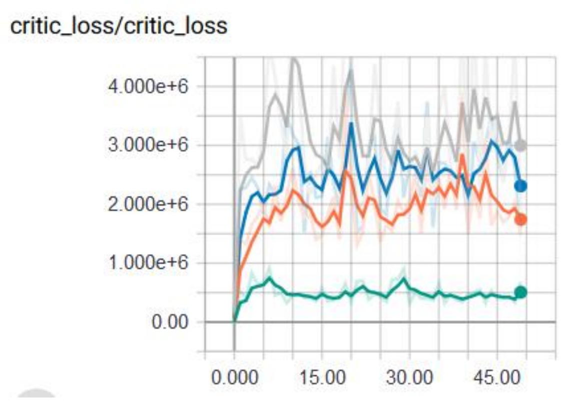
Hyperparameters:
The hyperparameters apart from network structure are specified as follows:

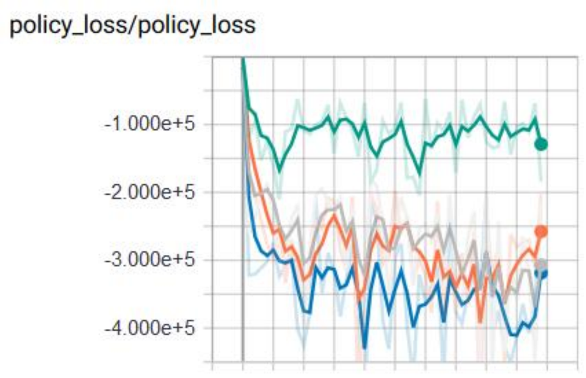
Variable	Value	Module	Explanation
e_greedy	0.4	learning	Probability of choosing an action randomly during learning to increase exploration of the map
learning_rate	0.0001	learning	Learning rate to the optimizer
EPISODES	50-500	learning	Number of epochs performed
reward_decay	0.99	model	The factor to discount future rewards with
entropy_factor	0.2	model	Importance of regularization of the policy using entropy

Convergence:
The model is currently not stably converging. This might be due to the following reasons:
- Network is too simple / too complex
- Hyperparameters are wrongly specified
- Reward is not optimal

We are currently working on having a more stable convergence but because the state space is huge (due to the exploration map) this is difficult. On the other hand we cannot remove the exploration map as it will be used later when tackling the multi-agent problem.

Currently, the policy loss and the critic loss are minimized independently as suggested in[7](Algorithm S3). In tensorboard it can be seen, that these losses are contradicting each other.





If the critic loss is high, the policy loss is low and the other way around. This is possible as they share the same hidden layer.

To have a better solution it might be useful to combine the losses and balance them by another hyperparameter.

Experimental Findings:
One crucial factor is the percentage on how often the drone as seen the target during learning. We can say that if it is lower than 50 %, the drone will most likely not find the target in the exploitation scenario.

Example output of the simulator after 40 episodes training:

```
Target Found in:  30 % of Episodes
Total training discovery map:
 [[78.   69.    75.    82.    50.    45.    43.    48.    50.    141.]
  [13.   14.    26.    30.    28.    25.    25.    30.    38.    98.]
  [2.    5.     5.     3.     6.     8.     7.     11.    12.    25.]
  [0.    0.     2.     2.     2.     2.     2.     0.     1.     0.]
  [0.    0.     1.     1.     3.     5.     2.     1.     0.     0.]]


---------EXPLOITATION--------

Exploitation discovery map:
 [[ 1.   1.    1.    1.    1.    1.    1.    1.    1.    16.]
  [ 0.   0.    0.    0.    0.    0.    0.    0.    0.    0.]
  [ 0.   0.    0.    0.    0.    0.    0.    0.    0.    0.]
  [ 0.   0.    0.    0.    0.    0.    0.    0.    0.    0.]
  [ 0.   0.    0.    0.    0.    0.    0.    0.    0.    0.]]

 Simulator grid with size (5, 10):
[[None None None None None None None None None   99]
 [None None None None None None None None None None]
 [None None None None None None None None None None]
 [None None None None None None None 'T'  None None]
 [None None None None None None None None None None]]
```

Here we can see how the drone explored the grid during training, but did not learn the optimal path. It seems the top right corner was not visited often enough to learn to go down or back.
Another relevant finding is that the initialization of weights and biases is crucial. If initialized with a distribution centered at zero the calculation of the gradients explodes producing NaN.

Update Technique
Regarding the update technique, we did not implement anything regarding the papers in the proposal. But we still consider Lenient DQN as the optimal way to tackle the problem of correlated states in a multi agent scenario. We consider this approach when we are implementing the multi-agent scenario.

Action-selection Technique
For now we are only using e-greedy for the simple implementations. We regard noisy net as the most advanced action selection technique which we are currently planning to implement at a later stage.

Reward/Cost
The reward function does not play a key role in state of the art papers. For instance, [3] uses for the localization task a reward function as simple as just counting the number of robots who have already found the target combined with an action punishment to introduce some efficiency.

While implementing baseline papers we found that the reward function affects convergence by a strong degree. Therefore we have limited the rewards/cost between -1 and 1.

Tangible Outcomes

A paper will be written to summarize the experiment and key research findings for the purpose of submitting for publication. In addition to this, we will also be including a demo or a tutorial in the form a video representation. The video will depict the simulation of the drone navigating through a unknown path in search of the target, while avoiding obstacles on the way.

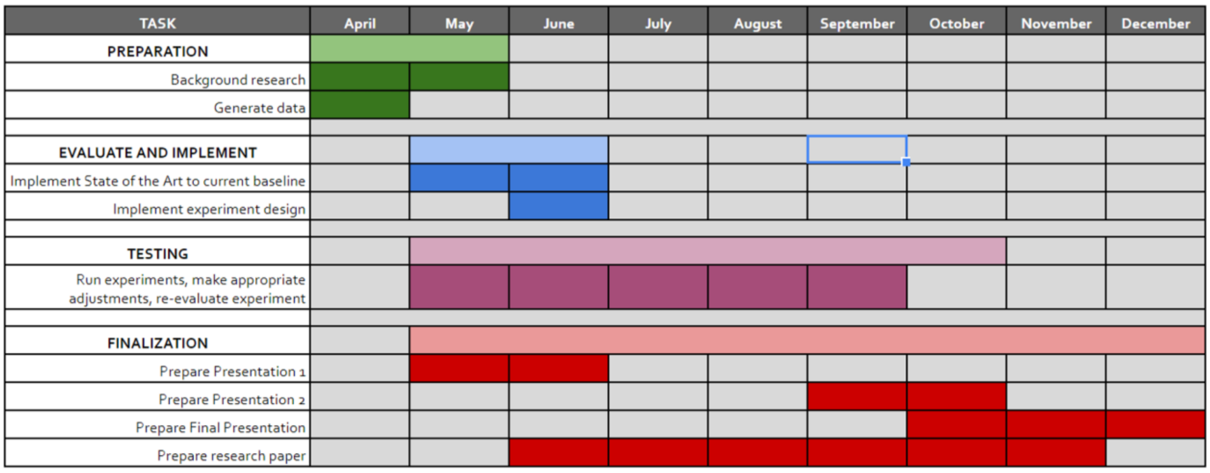
Work Plan

Our initial has to change slightly as the implementation is more complicated as expected.

Goals until second presentation

1.	Actor-Critic converging	by June
2.	Implementation of A3C	by July
3.	Multiple drones	by July
4.	Random starting of drones	by June
5.	Implement NoisyNet and Lenient	by September
6.	Visualization of simulator	by June
7.	Save drone test states for visualization	by June
8.	Research	ongoing

This is our original plan.



TASK	April	May	June	July	August	September	October	November	December
PREPARATION	🟩	🟩							
Background research	🟩	🟩							
Generate data	🟩								
EVALUATE AND IMPLEMENT		🟦	🟦						
Implement State of the Art to current baseline		🟦	🟦						
Implement experiment design			🟦						
TESTING		🟪	🟪	🟪	🟪	🟪	🟪		
Run experiments, make appropriate adjustments, re-evaluate experiment		🟪	🟪	🟪	🟪	🟪			
FINALIZATION		🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
Prepare Presentation 1		🟥	🟥						
Prepare Presentation 2						🟥	🟥		
Prepare Final Presentation							🟥	🟥	🟥
Prepare research paper			🟥	🟥	🟥	🟥	🟥	🟥	

References

[1]	Foerster (2017) Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning
[2]	Fortunato (2018) Noisy Network fo Exploration
[3]	Hüttenrauch et al (2017) Guided Deep reinforcement learning for Swarm systems

[4] Mnih et al. (2013) Playing Atari with Deep Reinforcement Learning
[5] Palmer (2018) Lenient Multi-Agent Deep Reinforcement Learning
[6] Zhang (2017) A Deeper Look at Experience Replay
[7] Mnih (2016) Asynchronous Methods for Deep Reinforcement Learning
[8] Hessel et al. (2017) Rainbow: Combining Improvements in Deep Reinforcement Learning
[9] Lisa Lee(2015) Robotic Search & Rescue via Online Multi-task Reinforcement Learning