Research Proposal for Student Research Project - 2018

International Masters in Data Analytics

ISMLL, University of Hildesheim.

Tutor: Hadi Samer Jomaa

Team:

Benjamin Hogstad
Ramya Raghuraman
Krithika Murugesan
Mohana Nyamanahalli Venkatesha
Michael Ruchte

Contents

Problem Setting

Our goal is to train a team of autonomous drones (multi-agent) to explore an unknown area 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. Additionally, each agent is acting independently and thus views other agents as part of the environment. This causes corrections to become more irrelevant as time passes.

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.

1. **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.

[3] tackled the problem of limited communication range in a swarm of robots. They trained the swarm to both stay connected to other robots and find a target. They focus strongly on the problem of partial observations for each single robot and use a modification of DDPG (*Deep Deterministic Policy Gradient*) to use the *actor-critic* approach and continuous action spaces. During learning they use a central "critic" which knows the global state to overcome the problem of partial observations.

[7] introduced asynchronous variants of ***one-step Sarsa, one-step Q-Learning*** and advantage actor-critic(A3C) with lesser resources. This is acheived by using multiple CPU threads on same machine and multiple actor-learners which explore different parts of environment. Different exploration

policies are used explicitly 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.

[9] To remedy the problem of cost due to computation of RL, uses the ***Policy Gradient Efficient Lifelong Learning Algorithm (PG-ELLA)***, an online multi-task RL algorithm that enables the robot to efficiently learn multiple consecutive tasks by sharing knowledge between these tasks to accelerate learning and improve performance. They evaluated three RL methods—Q-learning (single –task RL) against policy gradient RL and PG-ELLA(both multi-task RL)— to find a target object in an environment under different surface conditions.

2. Update Technique

DQN assume uncorrelated training samples in order to converge. Therefore [4] introduced experience replay to uncorrelate the training samples. This approach stores each (state, action, reward, next_state)-set into a big table and for learning, training samples are randomly drawn.

[1] proposed two approaches for incorporating experience replay into the ***multi-agent RL***: Each tuple in the ER buffer is augmented with the probability of the joint action in that tuple; leading to lower importance weights for older data. The other approach estimates each other agent's behaviour using a low-dimensional fingerprint.

[5] introduced ***Lenient DQN*** (LDQN) to enhance experience replay mechanism to incorporate leniency to decay the probability that the state-action pair will be used in updates. This decay is based on the frequency in which the state-action pair has been evaluated by any agent. This approach outperforms all other approaches so far.

[6] evaluated three different update techniques to enhance experience replay (ERM): ***Online-Q, Buffer-Q, and Combined-Q***. Online-Q used just most recent action and no ERM, Buffer-Q used only ERM, and Combined used both. Additionally it evaluated variances in the buffer size. All three techniques were capable of completing the given task, however Combined was quickest to converge. Buffer size was determined to be a critical hyperparameter to evaluate as it can cause the algorithm to diverge.

- Actor-Critic -> A3C

3. Action-selection Technique

[5] also introduces the ***T-greedy*** exploration that uses the temperature of the state to replace the E in ***E-greedy***. The temperature function counts how often one cell has been visited and thereby decays over training time to transition the agent from explorer to exploiter.

NoisyNet introduced by [2] employs a noise function to the weights within the neural network to promote exploration. The noise function is minimized using gradient descent. This removes the need for a complex exploration strategy embodying randomness as it is already incorporated into the network itself. One could say the greedy action-selection method is used. When combined with ***DQL, Dueling, and A3C***, it improves the baseline of all algorithms.

Integrate all these components ***Double Q-learning, Prioritized replay, Dueling networks, Multi-step learning, Distributional RL, Noisy Nets*** into a single integrated agent called ***Rainbow***, Rainbow introduced by [8] combine the multi-step distributional loss with double Q-learning by using the greedy action in S_{t+n}(value distribution) selected according to the online network as the bootstrap

action $a*_{t+n}$ and evaluating such action using the target network. Distributional Rainbow variants prioritize transitions by the ***KL loss***. The KL loss as priority might be more robust to noisy stochastic environments because the loss can continue to decrease even when the returns are not deterministic. Within these noisy linear layers we use ***factorised Gaussian noise*** to reduce the number of independent noise variables.

4. **Reward/Cost**

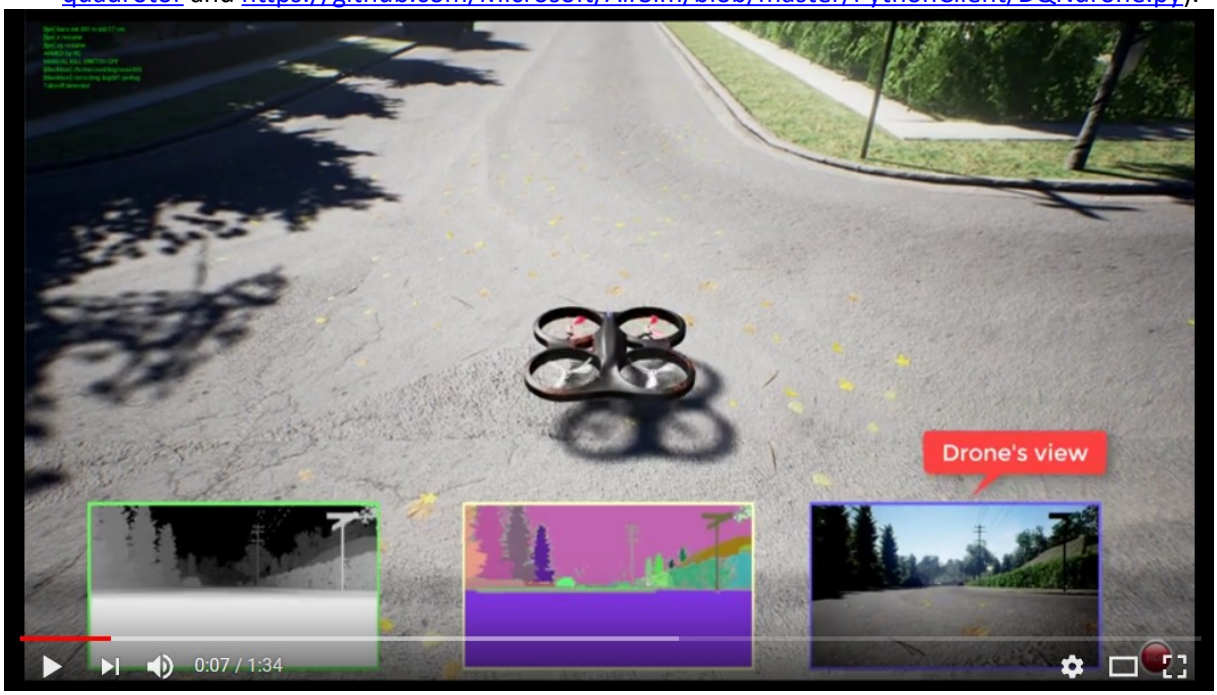
The reward function does not play a key role in state of the art papers. For instance [3] uses for the localization task a reward function as simple as just counting the number of robots who have already found the target combined with an action punishment to introduce some efficiency.

Data Foundation

For the training environment we will use a simulator in order to create the training data.

For early experiments we will develop our own simple **2D chess grid environment**, which allows for only four actions: up, down, left and right. Each cell can be blank (free space), a drone, an obstacle or the target.

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 documented 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 (see: https://github.com/Microsoft/AirSim/blob/master/PythonClient/AirSimClient.py#L487). It has been used for Reinforcement Learning already, documentation and Python source code is available (see: https://github.com/Microsoft/AirSim/blob/master/docs/reinforcement_learning.md#rl-with-quadrotor and https://github.com/Microsoft/AirSim/blob/master/PythonClient/DQNdrone.py).



It is setup for one drone by default, however it can be tweaked to work for several drones as well (See: https://github.com/Microsoft/AirSim/blob/master/docs/multi_vehicle.md, https://github.com/Microsoft/AirSim/issues/560).

With the Unreal engine as the backend, several different readily available worlds may be loaded as surrounding for the search and rescue scenario. This way we can create very realistic settings for

training. In case the Unreal Engine is computationally too demanding, its graphic settings can be scaled down to the bare minimum (see: https://docs.unrealengine.com/latest/INT/Engine/Performance/Scalability/ScalabilityReference/).

In order to simulate the victim, we can use a simple GPS location as the target which needs to be found by the team of drones.

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 baldy documented and cumbersome to set up.

Research Idea

The purpose of the experiment is to evaluate current state of art techniques and the combination of them to establish a generalized and scalable algorithm to be applied for search drones in an unknown environment.

Asynchronous Advantage Actor – Critic (A3C) replaced the ground-breaking Deep Q-Learning technique by replacing the Experience Replay with a series of parallel actors. A common problem with using Experience Replay in multi-agent systems is that it converges to sub-optimal results. A solution to this is implementing Lenient Experience Replay which uses a decaying temperature metric to reduce the impact of repeated state-actions as they have already been incorporated into the network's weights.

After the state-actions are evaluated, a selection must be made; this process is called the ***action-selection technique***. NoisyNet is the most recent proposal to incorporate a noise function into the weights of the network itself. The noise function is then minimized using gradient descent to allow the action-selection to go from exploration to exploitation. Another technique is to use the same temperature parameter as in Lenient Experience Replay and randomly select an action based on the temperature of the state.

These combined techniques A3C with NoisyNet, Lenient Experience Replay and T-greedy, and Lenient Experience Replay and Noisy Net will be evaluated in a two-dimensional setting where a multi-agent system must explore an unknown environment and discover a target at an unknown random location. There are four possible actions {Left, Right, Up, Down}. If an agent bumps into an obstacle, it will remain in place. All agents will initialize in the same location on the edge of the map. The episode will end when an agent reaches the target. The algorithm will be optimized for finding the target as quickly as possible. A series of scenarios will be posed to the proposed techniques; including: varying number of agents, randomized obstacles, and varying amounts of obstacles.

The following scenarios may be implemented and included in the evaluation:

- Time-constraint – the agents have a limited number of actions to reach the target
- Randomized Obstacles – the environment will have obstacles to avoid
- Variety of amount of obstacles – based on percentage of obstacles in the map
- Number of drones – the number of drones will vary

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

- 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:

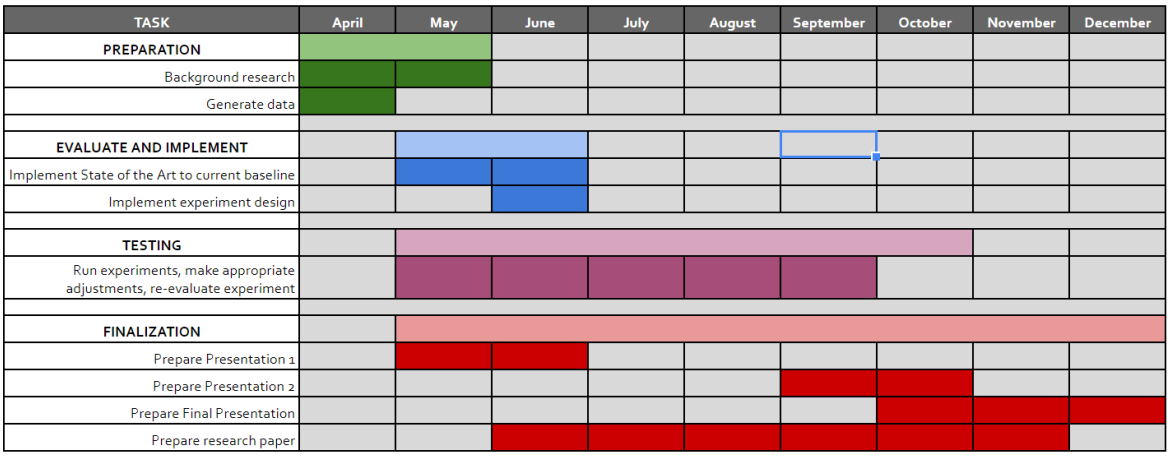
- Technical details about drone hardware and communication
- Optimization with regard to most efficient use of hardware resources

Tangible outcomes

The overall outcome would be:

- Reduction of the cost.
- We will be making use of the open source software for our project and provide results via software documentation (story about requirements, design, implementation).
- In addition to this, we will also write a research paper to summarize our experiments, key research findings(*Results indicate strong support for the process as a planning/review tool for projects required to deliver a combination of tangible outcomes*), practical implication, originality/value, design/methodology/approach.
- 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 known/unknown path in search of a particular object/person, while avoiding obstacles on the way. The data regarding the location and optimal path will be transmitted to the central hub after which the rescue mission can take place.

Work Plan



Team

Our team consists of:

- Researcher/Project Manager: Benjamin Hogstad (277456)
  - Received a Bachelors of Science in Mathematics with a Chemistry minor from the University of Oregon.
  - Prior experience in a regulated chemical laboratory as quality assurance and project manager.
  - United States of America

- Researcher/Writer: Ramya Raghuraman (277488)
  - Received a bachelor degree in Electronics and Communication.
  - Experience in Accenture as an associate software developer.
  - India

- Researcher/Presenter: Krithika Murugesan (277537)
  - Received a bachelor degree in Electrical and Electronics Engineering
  - Prior experience in Web and Click Stream Analytics for Search Engine Ad monetization and Optimization.
  - India

- Researcher/Coder: Mohana Nyamanahalli Venkatesha(276833)
  - Received a bachelor degree in Computer Science and Engineering.
  - Experience in Database Administration
  - India

- Writer/Coder: Michael Ruchte (277530)
  - Received a bachelor degree in Political Science and Economics.
  - Experience in Computer Vision, working experience in software development, database administration and Linux server administration.
  - Germany

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] Rainbow: Combining Improvements in Deep Reinforcement Learning

[9] Lisa Lee(2015) Robotic Search & Rescue via Online Multi-task Reinforcement Learning