

AFRL-AFOSR-UK-TR-2024-0010

Real-time Decision Making for Autonomous Systems using AI Methods

Shin, Hyo-Sang
CRANFIELD UNIVERSITY
COLLEGE ROAD
BEDFORD, , MK43 0AL
GBR

12/12/2023
Final Technical Report

DISTRIBUTION A: Distribution approved for public release.

Air Force Research Laboratory
Air Force Office of Scientific Research
European Office of Aerospace Research and Development
Unit 4515 Box 14, APO AE 09421

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
20231212	Final	START DATE 20190715	END DATE 20230714

4. TITLE AND SUBTITLE
Real-time Decision Making for Autonomous Systems using AI Methods

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
	FA9550-19-1-7032	
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)
Hyo-Sang Shin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
CRANFIELD UNIVERSITY
COLLEGE ROAD
BEDFORD MK43 0AL
GBR

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
EOARD UNIT 4515 APO AE 09421-4515	AFRL/AFOSR IOE	AFRL-AFOSR-UK-TR-2024-0010

12. DISTRIBUTION/AVAILABILITY STATEMENT
A Distribution Unlimited: PB Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In the last performance period, we performed an extensive review on the current status of MAS technologies, and developed new task allocation algorithms, focusing on the first four capabilities above – providing certain optimality guarantee, reducing computation, and improving the adaptability and robustness. First two baseline algorithms were developed – Decentralised decreasing Threshold Task Allocation (DTTA) and Decentralised Sample Threshold Task Allocation (STTA) – and then improved to more practical versions
– Lazy Decreasing Threshold Task Allocation (LDTTA) and Threshold Bundle Task Allocation (TBTA). It was validated that the proposed algorithms achieve solution quality which is comparable to state-of-the-art algorithms in the monotone case, and much better quality in the non-monotone case with significantly less computational and communication complexity.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	SAR	86

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
FELIX KNUTSON	314-235-6474

Standard Form 298 (Rev.5/2020)
Prescribed by ANSI Std. Z39.18

DISTRIBUTION A: Distribution approved for public release.





Award number: FA9550-19-1-7032

Real-time Decision Making for Autonomous Systems using AI Methods

PI: Prof Hyo-Sang Shin

Head of Autonomous and Intelligent Systems Group
Centre for Autonomous and Cyber-Physical Systems
SATM
Cranfield University

Performance Period: 15 July 2022 – 12 October 2023

DISTRIBUTION A: Distribution approved for public release.

Summary

This research project aims to develop a modular, flexible and robust decision-making algorithm that can handle the representative mathematical properties of the decision-making problem for Multi-Agent System (MAS). The main objective to achieve the aim is to develop a decentralised or distributed online optimisation algorithm that is:

- Capable of providing quality solutions, i.e. guaranteeing a certain level of optimality;
- Computationally efficient and scalable to be applicable to large-size swarms;
- Adaptive to dynamic changes in the system, communication network and environment;
- Robust to uncertainties with incomplete and corrupted information;
- Capable to learn and improve.

In the last performance period, we performed an extensive review on the current status of MAS technologies, and developed new task allocation algorithms, focusing on the first four capabilities above – providing certain optimality guarantee, reducing computation, and improving the adaptability and robustness. First two baseline algorithms were developed – Decentralised decreasing Threshold Task Allocation (DTTA) and Decentralised Sample Threshold Task Allocation (STTA) – and then improved to more practical versions – Lazy Decreasing Threshold Task Allocation (LDTTA) and Threshold Bundle Task Allocation (TBTA). It was validated that the proposed algorithms achieve solution quality which is comparable to state-of-the-art algorithms in the monotone case, and much better quality in the non-monotone case with significantly less computational and communication complexity.

As a continuation of the last report, this report focuses on the last remaining capability above, which is the capability to learn and improve. Two Reinforcement Learning (RL) based schemes are proposed: self-play methods and adaptive laziness allocation. The proposed self-play methods are proposed for usage in mixed competitive cooperative environments, under the Multi-Agent Reinforcement Learning (MARL) framework, that can be used as approximations of task allocation of MAS applications. Adaptive laziness allocation method is proposed for a more specific and practical scenario, which is Unmanned Aerial Vehicle (UAV) flocking. The details of the problem formulation, models, and training environments are presented. Simulation results reveal that the proposed algorithms achieve better performance by learning from the environment.

The main works were conducted by Cranfield University and additional works performed by University of Patras will be presented in a separate document as an appendix.

DISTRIBUTION A: Distribution approved for public release.

Contents

Acronyms **6**

1 Introduction **8**

2 Multi-Agent Reinforcement Learning-Based Self-Play Method **10**

2.1 Preliminaries . . . 10

2.1.1 Reinforcement Learning Techniques . . . 10

2.1.2 Multi-Agent Reinforcement Learning . . . 11

2.1.3 Stochastic Games . . . 11

2.1.4 Self-Play Based Techniques . . . 12

2.2 Problem Formulation . . . 14

2.2.1 Rock Paper Scissors Environment . . . 14

2.2.2 Simple Tag Environment . . . 15

2.2.3 Waterworld Environment . . . 15

2.3 Methods . . . 16

2.3.1 Self-Play Schemes . . . 16

2.3.1.1 Naive or Plain Self-Play . . . 16

2.3.1.2 Maximum Reward Self-Play . . . 16

2.3.1.3 Sorted Plain Self-Play . . . 17

2.3.2 Sampling Distributions . . . 17

2.3.2.1 Probability Distribution 1 . . . 18

2.3.2.2 Probability Distribution 2 . . . 18

2.3.2.3 Probability Distribution 3 . . . 18

2.3.2.4 Probability Distribution 4 . . . 18

2.3.2.5 Exponential probability Distribution 1 . . . 18

2.3.2.6 Exponential probability Distribution 2 . . . 19

2.4 Results and Discussions . . . 19

2.4.1 Rock Paper Scissors Environment . . . 19

2.4.2 Simple Tag Environment . . . 22

2.4.3 Waterworld Environment . . . 26

2.4.4 Discussion . . . 26

DISTRIBUTION A: Distribution approved for public release.

3 Deep Reinforcement Learning-Based Adaptive Laziness Allocation **27**

3.1 Preliminaries . . . 28

3.1.1 Flocking Algorithm . . . 28

3.1.2 Inactiveness in Flocking . . . 29

3.2 Problem Formulation . . . 29

3.2.1 Scenario and Assumptions . . . 29

3.2.2 Mathematical Formulation . . . 30

3.3 Methods . . . 30

3.3.1 Deep Reinforcement Learning Environment . . . 31

3.3.1.1 State and Observation . . . 31

3.3.1.2 Reward Function . . . 31

3.3.2 Policy and Value Models . . . 32

3.3.3 Training Details . . . 33

3.4 Results and Discussions . . . 34

3.4.1 Tested Models . . . 34

3.4.1.1 Fully Active Model . . . 34

3.4.1.2 Heuristic Model . . . 34

3.4.1.3 Heuristic Variant Model . . . 35

3.4.1.4 Neural Network Model . . . 35

3.4.2 Validation Environment . . . 35

3.4.3 Test Environments . . . 37

3.4.3.1 Control Algorithm . . . 37

3.4.3.2 Observation . . . 39

3.4.3.3 The Number of Agents . . . 42

4 Conclusion **46**

DISTRIBUTION A: Distribution approved for public release.

List of Figures

1 The "Rock Paper Scissors" environment . . . 14
2 The "Simple Tag" environment . . . 15
3 The "Waterworld" environment . . . 16
4 Average mean episode reward for all proposed methods . . . 20
5 Average episode reward and standard deviation for "Maximum reward self-play" method . . . 20
6 Average episode reward and standard deviation for "Sorted plain self-play" method . . . 21
7 Average episode reward and standard deviation for "Naive self-play" method . . . 21
8 Average mean episode reward for all proposed methods . . . 22
9 Average episode reward and standard deviation for "Maximum reward self-play" method . . . 23
10 Average episode reward and standard deviation for "Sorted plain self-play" method . . . 23
11 Average episode reward and standard deviation for "Naive self-play" method . . . 24
12 Average episode reward and standard deviation for "Naive self-play" method . . . 25
13 Average episode reward and standard deviation for "Sorted plain self-play" method . . . 25
14 A sample scenario with 20 UAVs . . . 30
15 The four models tested in the validation environments with 200 different initialisations . . . 36
16 Performance in the test environment with $\lambda = 2.5$ and $\sigma = 2$ . . . 38
17 Performance in the test environment with $\lambda = 10$ and $\sigma = 0.5$ . . . 39
18 Performance in the test environment with $L_{bound} = 500m$ . . . 40
19 Performance in the test environment with $L_{bound} = 125m$ . . . 40
20 Performance in the test environment with $V = 10m/s$ . . . 41
21 Performance in the test environment with $V = 20m/s$ . . . 42
22 Performance in the test environment with 100 UAVs . . . 43
23 Performance in the test environment with 400 UAVs . . . 43
24 Performance in the test environment with 5 UAVs . . . 44
25 Flocking behaviour of each model with 400 UAVs . . . 44
26 Standard deviation histories of each model with 400 UAVs . . . 45

List of Algorithms

1 (POSG) RL loop with self-play . . . 13
2 Naive or plain self-play . . . 17
3 Maximum reward self-play . . . 17
4 Sorted plain self-play . . . 17

DISTRIBUTION A: Distribution approved for public release.

Acronyms

C-S	Cucker-Smale.
DDPG	Deep Deterministic Policy Gradient.
DRL	Deep Reinforcement Learning.
DTTA	Decentralised decreasing Threshold Task Allocation.
FF	Feed Forward.
FSP	Fictitious Self Play.
FXP	Fictitious Cross Play.
ISR	Intelligence, Surveillance, and Reconnaissance.
LDTTA	Lazy Decreasing Threshold Task Allocation.
LN	Layer Normalisation.
MARL	Multi-Agent Reinforcement Learning.
MAS	Multi-Agent System.
MCTS	Monte-Carlo Tree Search.
MDP	Markov Decision Process.
MHA	Multi-Head Attention.
MMDP	Multi-Agent Markov Decision Process.
MRTA	Multi-Robot Task Allocation.
POMDP	Partially Observable Markov Decision Process.
POSG	Partially Observable Stochastic Game.
PPO	Proximal Policy Optimisation.
PSRO	Policy-Space Response Oracles.
RL	Reinforcement Learning.
RUSP	Randomised Uncertain Social Preferences.
SL-PSO	Social-Learning Particle Swarm Optimisation.
STTA	Decentralised Sample Threshold Task Allocation.
TBTA	Threshold Bundle Task Allocation.
TRPO	Trust Region Policy Optimisation.
UAV	Unmanned Aerial Vehicle.

DISTRIBUTION A: Distribution approved for public release.

vPN Variant Pointer Network.

DISTRIBUTION A: Distribution approved for public release.

1 Introduction

Effective and autonomous decision making is one of the most important enablers to successfully operate MAS. This research aims to develop efficient and robust decision-making algorithms, especially focusing on Multi-Robot Task Allocation (MRTA). The main objective to achieve the aim is to develop a decentralised or distributed online optimisation algorithm that is:

- Capable of providing quality solutions, i.e. guaranteeing a certain level of optimality;
- Computationally efficient and scalable to be applicable to large-size swarms;
- Adaptive to dynamic changes in the system, communication network and environment;
- Robust to uncertainties with incomplete and corrupted information;
- Capable to learn and improve.

As discussed in the Year 1 project report, the fundamental capabilities that help to assure other capabilities above in a certain level are optimality and computational efficiency of the algorithm. Therefore, we first developed new algorithms that can provide a certain level of theoretical guarantee on the first and second capabilities in the Year 2 project report. We further improved the proposed algorithms to reduce communication burden in the Year 3 project report, which contributes to enhance adaptability and robustness. In this report, we investigate the capability to learn and improve, using RL techniques.

In RL the environment is usually modelled as a Markov Decision Process (MDP) and the agents learn how to act by interacting with the environment and taking positive or negative rewards as feedback, aiding them to learn the best actions to choose. Therefore, the agents aim to optimise this reward function, thus improving their performance. RL-based methods provide plenty of advantages, like handling disturbances and uncertainties of the system, making it suitable for dynamic task allocation, as well as, dealing with complex tasks and in cases, being real time implementable [1]. However, the performance of the RL based methods heavily depends on their problem formulation and training methods. Generally RL techniques are known to provide better performance than some metaheuristics like simulated annealing and greedy algorithms and optimisation based techniques like the Hungarian method, but especially with the increase in the dimension, the performance is not guaranteed regardless of the high computational complexity [2].

To this end, this report introduces two RL based schemes for MRTA: self-play methods and adaptive laziness allocation. First, self-play methods are proposed for usage in mixed competitive cooperative environments, under the MARL framework, that can be used as approximations of task allocation of MAS applications. These include general MAS task allocation problems such as search and rescue missions, Intelligence, Surveillance, and Reconnaissance (ISR), target attacking, mobile edge computing, environment surveillance and monitoring, products transfer, and navigating in urban environments. The proposed self-play methods are applied to general stochastic games with different degrees of cooperation and competition between the agents.

Second, adaptive laziness allocation method is proposed to encapsulate the advantages in a more specific and practical scenario, which is UAV flocking. UAVs have gained increasing attention due to the nature

DISTRIBUTION A: Distribution approved for public release.

of their flexibility, ability to boost mission outcomes, and inherent self-sufficiency. The synergy within the system arises from the shared understandings and the cooperative interactions among individual agents [3]. This type of collaborative behaviour reflects the inherent coordination observed in nature. In these regards, many scholars have been inspired by the living beings in nature, studying different self-organisation methods that enhance the operational efficiency of a multi-agent system [4, 5, 6, 7]. In a practical application such as UAV operation, the cooperative interactions fall into consensus-based behaviours and one typical example is flocking task. We consider the UAV flocking as a task where the UAVs follow a certain rule to *loosely* form a pattern, not a pre-defined formation, such as fish-schooling. We propose a Deep Reinforcement Learning (DRL)-based adaptive laziness allocation method, with the main idea to refine and incorporate the concept of inactiveness into UAV flocking problems. Instead of building a new control algorithm, we formulate this as a MRTA problem where the task refers to simply reducing the control input of the existing flocking controller, by determining which UAVs should be lazy.

The details of the problem formulation, models, and training environments are presented for each of the proposed algorithms. To validate the proposed algorithms, numerical simulations are conducted with various MAS applications for the self-play methods, and different flocking scenarios for the adaptive laziness allocation method. Simulation results reveal that the proposed algorithms achieve better performance by learning from the environment.

The rest of the report is organised with two main parts: Chapter 2 for the self-play methods and Chapter 3 for the adaptive laziness allocation method. Basic concepts and definitions are presented in Sections 2.1 and 3.1, the problem formulations are given in Sections 2.2 and 3.2, and the proposed algorithms are detailed in Sections 2.3 and 3.3. The performance of the proposed algorithms is numerically validated with different scenarios and parameters in Sections 2.4 and 3.4. A summary of contributions and future research directions are offered in Chapter 4.

DISTRIBUTION A: Distribution approved for public release.

2 Multi-Agent Reinforcement Learning-Based Self-Play Method

In this chapter we introduce new self-play methods for usage in mixed competitive cooperative environments, under the MARL framework, that can be used as approximations of MRTA applications. In a MARL setting, there are a lot of frameworks that can be used like having independent learning, centralised learning, or centralised learning with decentralised execution, which is the framework used in this work. One of the most commonly used paradigm of MARL is a self-play method, that has so far been used mostly in zero-sum games, achieving very good results. The general idea entails the usage of old policies, with which the agents play for example in zero-sum games, with the agent or group of agents, beating their past selves, thus achieving continuous performance improvement [8]. In contrast to the common practice of using self-play techniques on zero-sum games only, we expand its usage to general sum games. We also propose specific methods for curating the pool of previous policies and novel probability distributions for sampling from this pool, improving the agents' performance. To the best of our knowledge, this is the first attempt to research the role of various policy sampling distributions on different types of environments.

2.1 Preliminaries

2.1.1 Reinforcement Learning Techniques

RL algorithms can be model-based, in the case there is a model of the environment providing the transition probability of states and rewards, or model-free where there is no model of the environment and is also the focus of this work [9]. The environment is usually formed as a MDP and the agents' goal is to optimise a reward function, improving their overall performance.

The model-free methods can be divided into value-based methods, policy-based methods and actor-critic methods. The value-based methods try to compute an estimate of the value faction of the state $V(s)$ or the state-action value function $Q(s, a)$ and use a fixed rule to create a mapping from these value functions to actions that need to be chosen. A well known algorithm belonging to this category is the Q-learning method. Policy-based methods do not need estimates of the value functions, but instead they compute probability distributions of actions over states $\pi_\theta = Pr[a|s]$, by using the parameterised policy π_θ and neural networks. Known policy-based algorithms include policy gradients and REINFORCE. The actor-critic methods are a combination of the aforementioned methods, with the critic estimating a value function and the actor using a parameterised policy to update the action distribution by the critic, using a policy gradient. Some actor-critic methods include Trust Region Policy Optimisation (TRPO), Deep Deterministic Policy Gradient (DDPG), and Proximal Policy Optimisation (PPO), which is the method used in this report [9].

RL methods are very good in dealing with disturbances, something crucial for dynamic environments like most of the real world applications. They generally have high efficiency, might be online implementable (for well trained networks) and have good behaviour to environmental disturbances. On the other side they can be computationally expensive for large scale systems [10, 11, 12]. Generally RL techniques can have better performance than some metaheuristics like simulated annealing and greedy algorithms and optimisation based techniques like the Hungarian method. Nevertheless an existing problem remains the increase in the dimension and the computational cost of some methods [2].

DISTRIBUTION A: Distribution approved for public release.

The method used in this report is PPO algorithm, which is an actor-critic method, on policy method originated in [13]. PPO is based on the same principle as TRPO, aiming to find the biggest improvement step on a policy, using on policy data, but without getting too far, causing a performance deterioration. PPO achieves this aim in a simpler way than TRPO, using the first-order optimisation only. Generally PPO has the stability and reliability of TRPO, but it is simpler to implement and can have better overall performance.

2.1.2 Multi-Agent Reinforcement Learning

Learning in a multi-agent environment is much more complex and difficult than in the single agent case. There are a lot of problems like the non-stationarity of the environment, since the other agents policies are changing and affect the environment, the curse of dimensionality as the number of agents increases, multi-agent credit assignment, and global exploration [14].

In MARL there are a lot of frameworks for learning. Independent learners or decentralised learning and execution uses single agent algorithms, even though the Markov property is violated, assuming the other agents as part of the environment and it is a method used frequently, mainly for scalability purposes. In centralised learning and execution a central agent is assumed that gathers all the observations, actions and rewards from all others agents, while all the agents take the same actions, which is not suitable to larger environments [15].

The most common paradigm is the centralised learning with decentralised execution, where a central agent is assumed that gathers information from all others agents, but the execution is decentralised, meaning that the policies use local agents' information only. A common approach is through a centralised critic and a decentralised actor or parameter sharing. In this report we use parameter sharing [16, 17]. Parameter sharing can use single agent RL methods in a multi-agent setting, by using a single shared policy for all the agents, using the experiences of all the agents simultaneously. Nevertheless, even though the learning is centralised the execution is not, because each of the agents has its own observations, including their respective index.

2.1.3 Stochastic Games

Even though Multi-Agent Markov Decision Process (MMDP) extends the MDP formulation to a multi-agent setting, they assume that all agents receive the same reward. Therefore a wider extension of MDP to a multi-agent setup is stochastic games that permit each agent to have a unique reward function [17]. A stochastic game is defined by the tuple $\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}$ where:

- $\mathcal{N}$ is the number of agents in the environment
- $\mathcal{S}$ is the number of the states
- $\mathcal{A}_i$ is the set of possible actions of agent i
- $\mathcal{P} : \times \Pi_i \in_{[\mathcal{N}]} \mathcal{A}_i \times \mathcal{S} \rightarrow [0, 1]$, is the transition function and $|\mathcal{N}|$ is the set of agents
- $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \ldots \mathcal{A}_N \times \mathcal{S} \rightarrow \mathbb{R}$, is the reward function for agent i

DISTRIBUTION A: Distribution approved for public release.

In a partially observable stochastic game the agents have not access to the exact state of the environment, but only to an observation of that state through an observation function [18]. Therefore, a partially observable stochastic game is defined except for aforementioned terms of the stochastic game, with the terms Ω_i ,$\mathcal{O}$, where:

- Ω_i is the set of possible observations for agent i
- $\mathcal{O}_i : \mathcal{A}_i \times \mathcal{S} \times \Omega_i \rightarrow [0, 1]$, is the observation function

2.1.4 Self-Play Based Techniques

The self-play scheme is a MARL framework that has shown very good results in two-player zero-sum games. In this scheme the agent or agents are trained with their past selves or policies that have been generated, during training, in the past. In recent approaches it is not necessary to use only the last policy, but a set of previous policies can be created, from which the self-play policy is sampled. In zero-sum games, the agents with this scheme try to defeat their older selves (or policies), achieving continual learning [19].

In general, we can say that the self-play scheme envelops the general MARL loop and extends it by adding extra features in the beginning and the end of the loop. Defining as $\Pi = \Pi_1 \times \ldots \times \Pi_n$ the joint policy space, where Π_i is the policy space for agent i and π is the current policy. Following the definition of self play for general sum Partially Observable Stochastic Game (POSG) in [19] the main components of self play are the *menagerie* $\boldsymbol{\pi^o}$, the *policy sampling distribution* Ω, and the *gating function* G. Specified by the tuple $< \Omega(\cdot|\cdot, \cdot), G(\cdot|\cdot, \cdot) >$:

- $\boldsymbol{\pi^o} \subseteq \Pi_i$; The **menagerie** is the set of the previous policies that can be used.
- $\Omega(\boldsymbol{\pi'} \in \Pi_{-i}|\boldsymbol{\pi^o} \subseteq \Pi_i, \pi \in \Pi_i) \in [0, 1]$; where $\boldsymbol{\pi'} \subseteq \boldsymbol{\pi^o}$; The **policy sampling distribution**, is a statistical distribution that chooses a policy from the menagerie that will be used in self play.
- $G(\boldsymbol{\pi^{o\prime}} \subseteq \Pi_i|\boldsymbol{\pi^o} \subseteq \Pi_i, \pi \in \Pi_i) \in [0, 1]$; The **curator** or **gating function**, of the menagerie, that is a function that chooses with what criterion a policy enters or is discarded from the menagerie.

In Algorithm 1 we can see the RL training loop with the addition of the self-play scheme, in magenta colour, for general sum games [19]. Moreover, in the next paragraph follows a description of some characteristic self-play based methods found in literature.

In [20] the authors introduce a self-play variation called other-play, used in cases where agents need to zero-shot coordinate with unknown agents. The algorithm is used on cooperative games and works by finding a maximally robust strategy o breaking the symmetries existing on the positive sum game. In [21] neural Fictitious Self Play (FSP) is proposed, that is based on FSP but with neural network approximation and is used on zero-sum imperfect information games, converging at approximate Nash equilibria. Also in [22] the authors use self-play in a 3D competitive environment with simulated physics, using a distributed version of PPO having decentralised training, The usage of an exploration curriculum helps agent learn plenty of emergent dexterous locomotion skills that will be difficult to create a reward for. Another approach is [23], where the multi-agent environment augmentation with Randomised Uncertain Social Preferences (RUSP)

DISTRIBUTION A: Distribution approved for public release.

Algorithm 1 (POSG) RL loop with self-play

Require: *Environment*: $(S, A, O, \mathcal{P}(\cdot,\cdot|\cdot,\cdot), \mathcal{R}(\cdot,\cdot), \rho_0)$
Require: *Self-Play Scheme*: $(\Omega(\cdot|\cdot,\cdot), G(\cdot|\cdot,\cdot))$
Require: *Policy to be trained*: $\pi \in \Pi_i$

$\boldsymbol{\pi^o} = \{\pi\}$ ▷ Menagerie initialization
for $e = 0, 1, 2, \ldots$ **do**
  $\boldsymbol{\pi'} \sim \Omega(\boldsymbol{\pi^o}, \pi)$ ▷ Sample from menagerie
  $\boldsymbol{\pi} = \boldsymbol{\pi'} \cup \{\pi\}$
  $s_0, \boldsymbol{o_0} \sim \rho_0$
  for $t = 0, \ldots,$ termination **do**
    $\boldsymbol{a_t} \sim \boldsymbol{\pi}(\boldsymbol{o_t})$ ▷ Classical MARL loop
    $s_{t+1}, \boldsymbol{o_{t+1}} \sim P(s_t, \boldsymbol{a_t})$
    $\boldsymbol{r_t} \sim \boldsymbol{R}(s_t, \boldsymbol{a_t})$
    $t \leftarrow t + 1$
  end for
  $\pi \leftarrow$ update(π)
  $\boldsymbol{\pi^o} \sim G(\boldsymbol{\pi^o}, \pi)$ ▷ Curate menagerie
end for
return π

is introduced. It leads agents in achieving more cooperative equilibria in various different social dilemma environments, leading to both reciprocity and team formation. In [24] the authors use self-play in a multi-agent hide and seek environment using centralised learning and decentralised execution. They proved that multi-agent self-play can help agents learn complex skills even with simple game rules, leading to emergent auto-curricula. In [25] they propose Fictitious Cross Play (FXP), where aspects from both fictitious self-play method and Policy-Space Response Oracles (PSRO) are used, aiming to learn global equilibria in mixed cooperative-competitive games. The algorithm is tested in matrix and gridworld games and and converges fast to global NE performing better than series of popular baseline methods. Another approach is [26] where they use self-play in combination with supervised learning using human experts data to train the deep neural networks and Monte-Carlo Tree Search (MCTS) in the game of Go and achieving human level performance. An improvement of the previous work is [27] the authors propose an improved version of AlphaGo with pure RL in training and no human knowledge. that uses self-play with an improved MCTS based policy in the game of Go. The algorithm achieved superhuman performance, without human examples. This approach requires a b it more time to train, but has better performance than the previous AlphaGo method that used human expert data.

As mentioned before, the RL method used in this experiment is the PPO with parameter sharing. The hyperparameters of the PPO algorithm used in the simulations are shown in Table 1.

DISTRIBUTION A: Distribution approved for public release.

Hyperparameter	Value
sample_batch_size	200
train_batch_size	4000
sgd_minibatch_size	128
lambda	1
kl_coeff	0.2
entropy_coeff	0
num_sgd_iter	30
vf_clip_param	10.0
clip_param	0.3
batch_mode	truncate_episodes

Table 1: Hyperparameters for PPO algorithm



Figure 1: The "Rock Paper Scissors" environment

2.2 Problem Formulation

Three simulation environments were used to cover different degrees of cooperation and competition between the agents. One fully competitive, zero-sum environment and two mixed competitive cooperative, general sum ones, with different degrees of cooperation, were used. All the environments were derived from the Pettingzoo API from Farama foundation [28]. The simulation framework used is the RLlib framework [29] and Python.

2.2.1 Rock Paper Scissors Environment

The fully competitive environment is the "Rock, Paper, Scissors" environment (Fig. 1). This is a two-player zero-sum game, where each player chooses one of three moves (rock paper or scissors) and the players have to show their chosen action simultaneously. Generally, rock beats scissors, scissors beat paper, and paper beats rock and if the players have the same choice of action, then it is a draw. The winner takes a reward of +1, while the loser takes a reward of -1. In the case of a draw both agents are rewarded with 0.

DISTRIBUTION A: Distribution approved for public release.

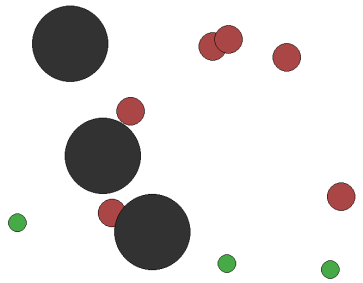


Figure 2: The "Simple Tag" environment

Agent	Number	reward
predator	6	+10
prey	3	-10

Table 2: Hyperparameters of Simple Tag environment

2.2.2 Simple Tag Environment

The first mixed competitive cooperative environment is the "Simple tag" one (Fig. 2) originated in [30] This is a predator-prey environment, where there are two types of agents, with different goals and speeds. The good agents (green) are faster than the adversaries (red) and they receive a negative reward of -10 when they are hit by the adversaries. The adversaries, that are slower receive a reward of +10, whenever they hit the good agents. There are also some large obstacles that block the way of the agents. Because the good agents, should not run to infinity to avoid the adversaries they are penalised by exiting the environment area. The setting used is 3 good agents, 6 adversaries and 3 obstacles. Because adversaries are slower they have to implement cooperation strategies in order to find and hit the prey. This environment can approximate task allocation applications of MAS like, search and rescue, targets ISR and attack of moving targets.

2.2.3 Waterworld Environment

The second mixed competitive cooperative environment used is the "Waterworld" environment from [31] originated in [16]. This is a partially observable continuous environment where the agents that are the pursuers have to cooperate to capture some moving targets (food), while avoiding some other moving entities (poison) and an obstacle. As we can see in Fig. 3 the agents (in purple) have 20 range limited sensors that measure the velocity and distance of objects in their range. Two pursuers have to cooperate to catch the targets, while avoiding poison, creating a cooperative dynamic, that has competitive elements as well. The rewards are both distributed locally to the pursuers doing a specific action scaled by 0.5, as well as globally to all the pursuers scaled by 0.5 again. When two agents capture food together they each receive a reward of 10, while there is also a smaller reward of 0.01 if they do not cooperate to catch food but they act alone. Also, if the agents touch the poison the receive a reward of -1. Moreover large actions are penalised by $-0.5||action||$, where $||action||$ is the Euclidean norm of the action velocity.

DISTRIBUTION A: Distribution approved for public release.

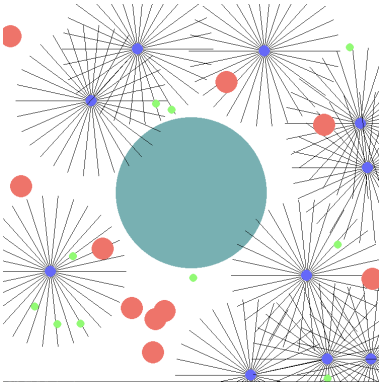


Figure 3: The "Waterworld" environment

The problem can be modelled as a decentralised Partially Observable Markov Decision Process (POMDP). This environment can be regarded as an approximation to MAS task allocation applications, like search and rescue, having moving targets and agents that need to cooperate to reach them. Also, it could be assumed as an ISR application, having surveillance and tracking of moving targets. We use 5 agents with 5 targets, that the agents have to catch, while avoiding other 10 moving objects and the obstacle.

2.3 Methods

We propose two new self-play algorithms, called "Maximum reward self-play" and "Sorted plain self-play", depending on the method using to curate the algorithm's menagerie, along with the "Naive self-play" method that is used as baseline.

2.3.1 Self-Play Schemes

Let us assume that, $R(s_t, \pi(s_t))$ is the reward of policy $\pi(s_t)$, written for simplicity as $R(\pi)$ and also, R and π are the current reward and policy, respectively. The proposed self-play schemes are explained in detail in the following sections:

2.3.1.1 Naive or Plain Self-Play

The "Naive self-play" scheme, uses a first in, first out logic for curating the menagerie. In every algorithm iteration, the current policy enters the menagerie while the oldest policy of the menagerie is removed, having a rolling window on the training policies. Algorithm 2 summarises the process of curating the menagerie in "Naive self-play".

2.3.1.2 Maximum Reward Self-Play

In the "Maximum reward self-play" scheme, the criterion used to curate the menagerie is the mean episode reward. In every iteration the policy already in the menagerie with the lowest reward, compared with the other menagerie policies, is found and the current training policy enters the menagerie only if its mean

DISTRIBUTION A: Distribution approved for public release.

Algorithm 2 Naive or plain self-play

$\pi^o = \pi^o - \pi_1^o$
for $i = 2$ to $M + 1$ **do**
    $\pi_i^o = \pi_{i-1}^o$
end for
$\pi_o = \pi_o \cup \pi$

reward is higher than this policy' reward. Also, afterwards the menagerie entries are sorted in ascending order with regards to their mean reward. Algorithm 3 summarises the process of curating the menagerie in "Maximum reward self-play".

Algorithm 3 Maximum reward self-play

find $R_{min}(\pi_{min}) : R_{min} = \min_{R_j} R(\pi^o)$
if $R > R_{min}$ **then**
    $\pi^o = \pi^o - \pi_{min}$
    $\pi^o = \pi^o \cup \pi$
    sort $\pi^o : R(\pi_{j+1}^o) > R(\pi_j^o)$
else
    sort $\pi^o : R(\pi_{j+1}^o) > R(\pi_j^o)$
end if

2.3.1.3 Sorted Plain Self-Play

The other framework used is the "Sorted plain self-play" one, that is a mixture of the other two frameworks. In this scheme the gating function of the plain self-play method is merely used for curating the menagerie, so the current policy is added to the menagerie every time regardless of its reward, but the policy with the lowest reward is discarded and the menagerie is sorted before the policy selection. Algorithm 4 summarises the process of curating the menagerie in "Sorted plain self-play".

Algorithm 4 Sorted plain self-play

find $R_{min}(\pi_{min}) : R_{min} = \min_{R_j} R(\pi^o)$
$\pi^o = \pi^o - \pi_{min}$
$\pi_o = \pi_o \cup \pi$
sort $\pi^o : R(\pi_{j+1}^o) > R(\pi_j^o)$

2.3.2 Sampling Distributions

A lot of different sampling distributions were used for sampling the policy that will be used from the menagerie. The simplest case is random sampling, where one of the policies inside the menagerie is chosen randomly as the training policy. Since the menagerie has M entries the probability of every entry to be

DISTRIBUTION A: Distribution approved for public release.

sampled is $P = (P_i, i = 1, ..., M)$. Initially we try some empirically engineered distributions, as well as two exponential ones, where $\pi \sim Exp(\lambda)$ and the probability density function is:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{for } x \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

2.3.2.1 Probability Distribution 1

In this scheme the menagerie is sampled with probability of:

$$P = (0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.1, 0.1, 0.25, 0.25)$$

2.3.2.2 Probability Distribution 2

In this scheme the menagerie is sampled with probability derived from:

$$\frac{1}{|\pi^o||(\pi^o - e)|^2}, \tag{2}$$

where e is the menagerie's policy index. This sampling distribution is derived from [19] and is used as a baseline for comparison with the other proposed distributions. Here the sampling probability is:

$$P = (0.006, 0.008, 0.01, 0.01, 0.02, 0.03, 0.04, 0.07, 0.16, 0.65)$$

2.3.2.3 Probability Distribution 3

In this scheme the menagerie is sampled with probability of:

$$P = (0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.82)$$

2.3.2.4 Probability Distribution 4

In this scheme the menagerie is sampled choosing the policy that has the highest reward of the sorted menagerie. This distribution does not apply to other self-play schemes, but only on the maximum reward self-play method.

2.3.2.5 Exponential probability Distribution 1

In this scheme the menagerie is sampled with an exponential distribution $\pi \sim Exp(5)$. Here the sampling probability is:

$$P = (0.003, 0.009, 0.01, 0.03, 0.07, 0.11, 0.14, 0.18, 0.2, 0.25)$$

DISTRIBUTION A: Distribution approved for public release.

2.3.2.6 Exponential probability Distribution 2

In this scheme the menagerie is sampled with an exponential distribution $\pi \sim Exp(1.11)$. Here the sampling probability is:

$$P = (0.002, 0.01, 0.03, 0.04, 0.05, 0.07, 0.08, 0.12, 0.14, 0.45)$$

2.4 Results and Discussions

Initially we simulate our proposed self-play schemes to the zero-sum "Rock paper scissors" environment, since self-play has so far been used on such types of environments and then we generalise this approach and use it to the other two mixed competitive cooperative environments. The baseline used in every environment, with which the proposed methods are compared is the parameter sharing PPO method, originated in [16]. Also, in every method we have the naive self-play with random sampling and maximum reward with random sampling from the menagerie, originated in our previous work [32] as an extra baseline for highlighting the role of the different probability distributions. Therefore when no sampling distribution is mentioned, it is implied that random sampling was used for this specific method. Moreover, another baseline for the probability distributions is the second probability distribution, as mentioned before. All the graphs below show the average mean reward and are averaged over 10 trials of the specific method, while the shaded region represents the standard deviation of the mean reward.

2.4.1 Rock Paper Scissors Environment

In the "Rock paper scissors" environment, there is one trainable policy for the first agent. The first agent at every iteration plays with one of the policies chosen from the menagerie, according to the methods explained before. In that way the first agent plays with previous version of himself, achieving continuous learning. For the baseline there is one trainable policy for every agent, where parameter sharing PPO is used.

In Fig. 4 we see all the methods for the "Rock paper scissors" environment. Generally, the methods based on the maximum reward scheme have better performance, followed by the sorted plain self-play and naive self-play methods. The baseline generally achieves performance close to zero, meaning that no agent performs better than the other. The maximum reward method with the exponential probability distribution 1 has the best performance overall.

In Fig. 5 we see all the plots for the maximum reward method. We notice that the exponential probability distribution 1 has the best overall reward and the probability distribution 4 is the most stable one, having a bit lower, but comparable reward. Also, distribution 1 seems to have the worst performance compared with the other distributions.

For the "sorted plain self-play" method we see in Fig. 6 that all the probability distributions have lower reward than the maximum reward with random sampling method, confirming the superiority of the maximum reward method. One exemption is the probability distribution 2 that is comparable to the maximum reward with random sampling technique and the best performance, compared with the other probability distributions of the "sorted plain self-play" technique. The distribution 1 seems to have the worst performance from all the other probability distributions, as is the case for the maximum reward method as well.

DISTRIBUTION A: Distribution approved for public release.

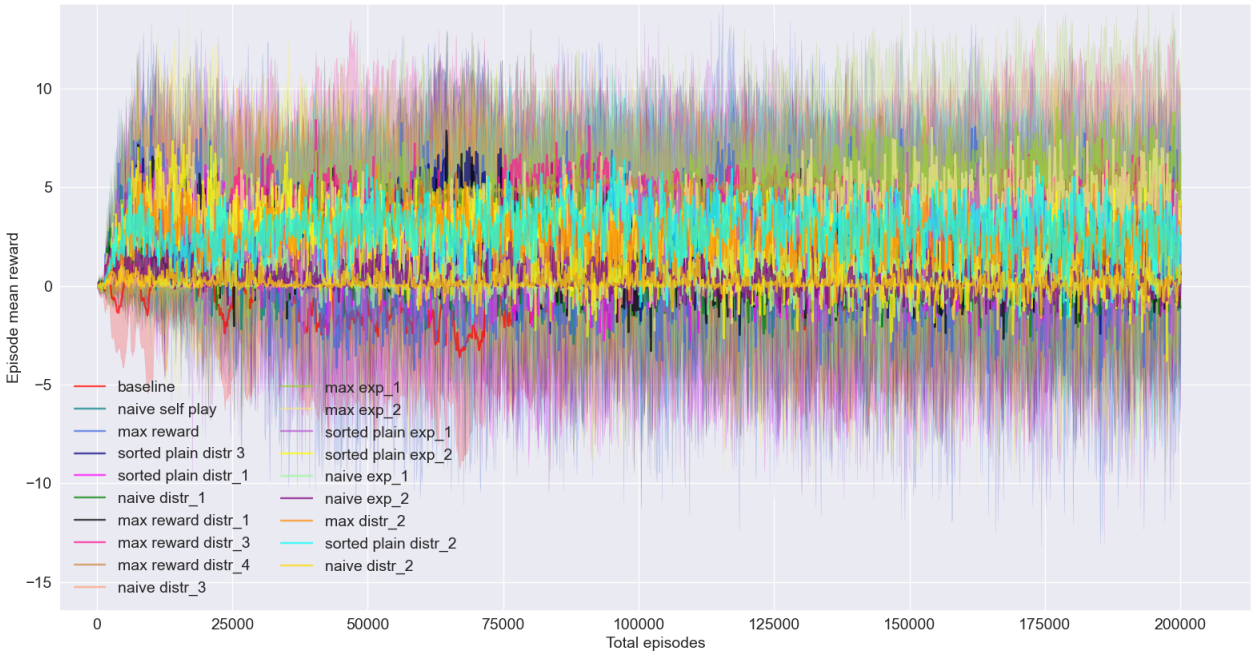


Figure 4: Average mean episode reward for all proposed methods

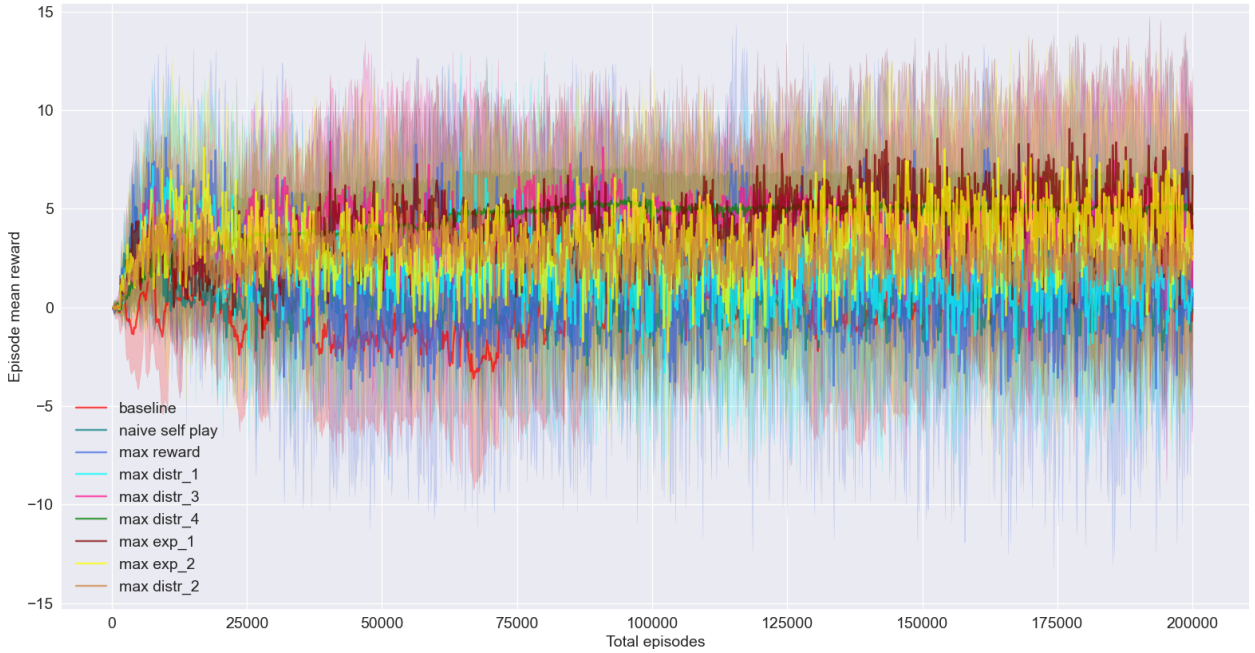


Figure 5: Average episode reward and standard deviation for "Maximum reward self-play" method

DISTRIBUTION A: Distribution approved for public release.

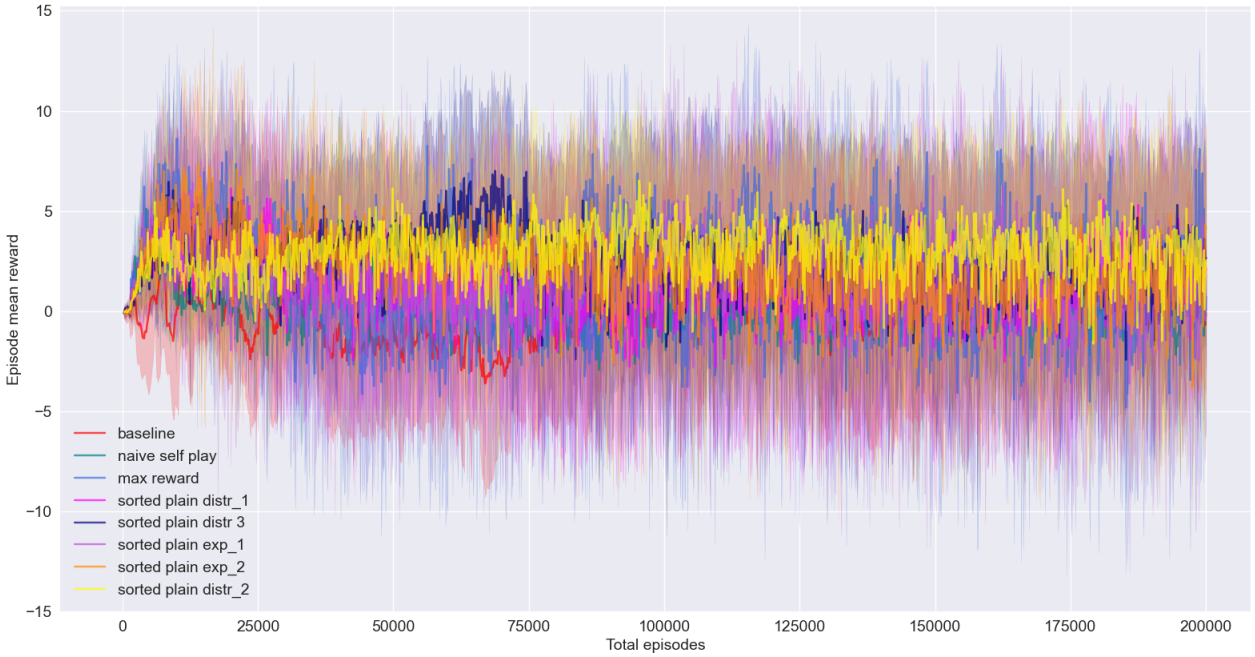


Figure 6: Average episode reward and standard deviation for "Sorted plain self-play" method

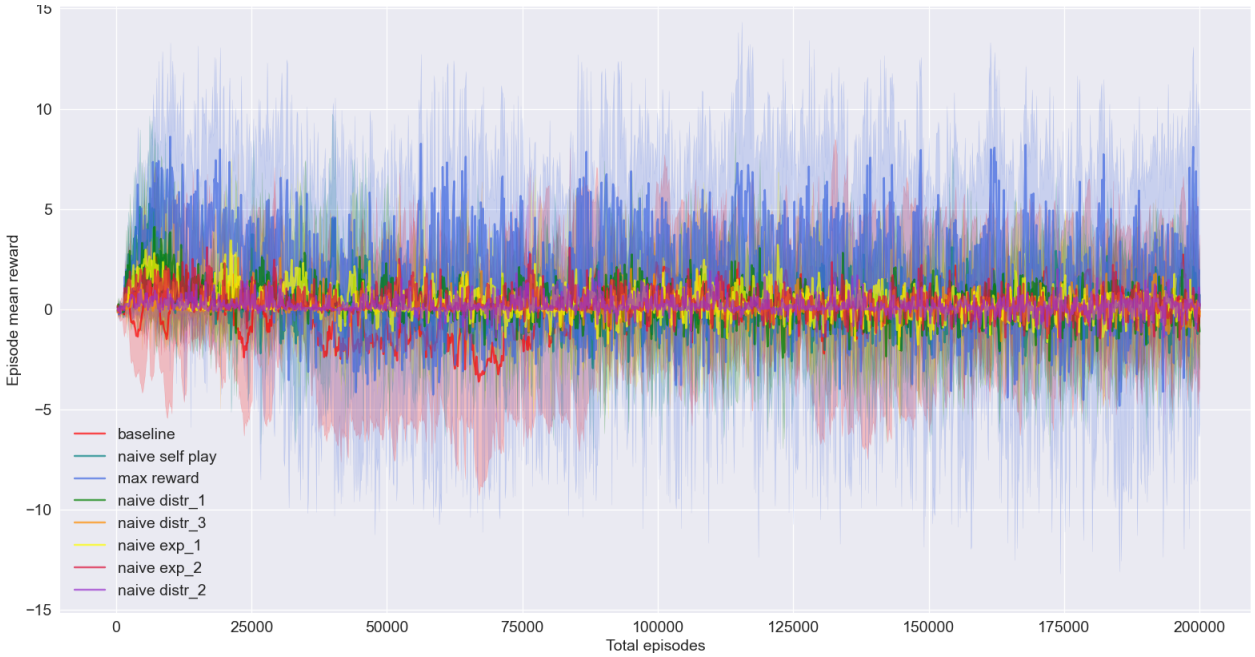


Figure 7: Average episode reward and standard deviation for "Naive self-play" method

DISTRIBUTION A: Distribution approved for public release.

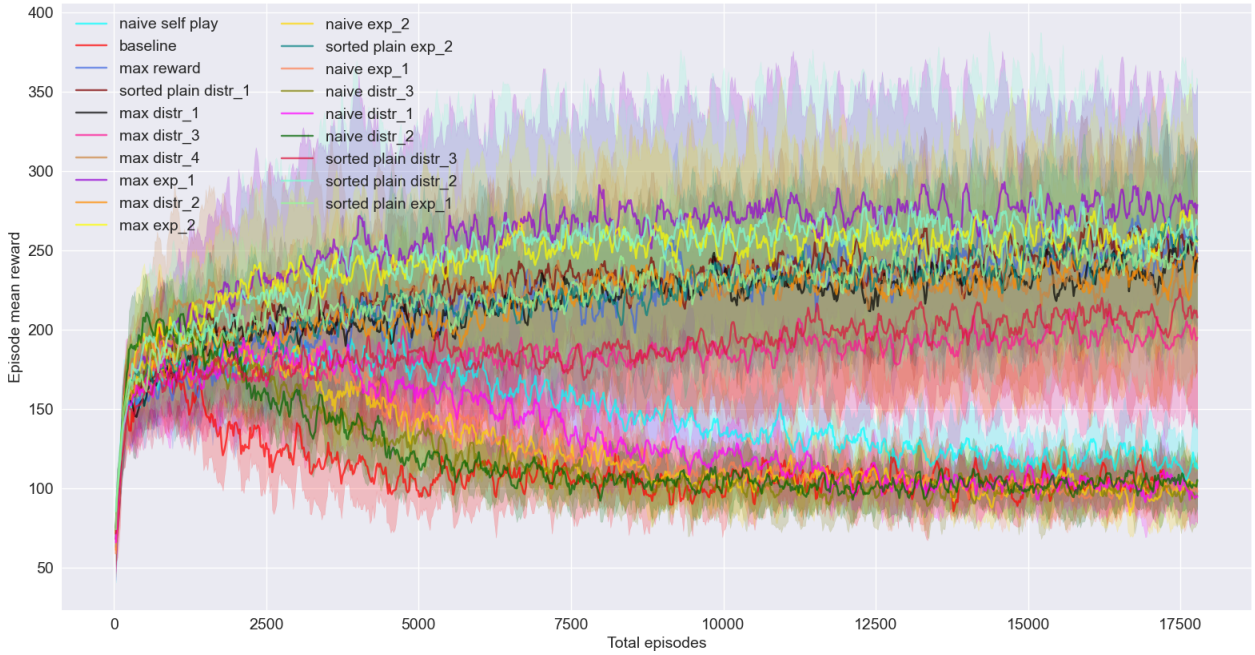


Figure 8: Average mean episode reward for all proposed methods

In the "naive self-play" method in Fig. 7, we see that using different probability distributions does not drastically improve the performance, highlighting the role of sorting the menagerie of policies. The exponential probability distribution 1 seems to have slightly better performance than the baseline and plain self-play method with random sampling at some points of the learning procedure, but still without achieving an efficient and permanent advantage for one of the agents.

2.4.2 Simple Tag Environment

Generally it is not straightforward to implement self-play in a mixed competitive cooperative environment. In this environment there are two trainable parameter sharing policies, one for the predators and another one for the prey agents. To expand the self-play schemes to this general sum environment, in every iteration, each one of the policies sample regarding to the described methods a policy from their own menagerie and these are the policies being trained, which populate the menagerie, regarding the criteria set by every method.

In Fig. 8 we can see the mean reward for all the aforementioned self-play methods, regarding the simple tag environment. Initially, we notice that the maximum reward method with random sampling has higher reward and thus better performance than the naive self-play method with random sampling and the parameter sharing PPO baseline. These two methods, also, experience catastrophic forgetting with their reward starting to decline after the 3000 episode in contrast to the maximum reward based methods that keep improving or remain stable throughout all the learning episodes.

For the "maximum reward self-play" method we notice in Fig. 9 that the exponential probability distribution 1 method has the best performance, followed by the second exponential distribution and that both of the exponential methods perform better than the second probability distribution. The probability distribution 3

DISTRIBUTION A: Distribution approved for public release.



Figure 9: Average episode reward and standard deviation for "Maximum reward self-play" method

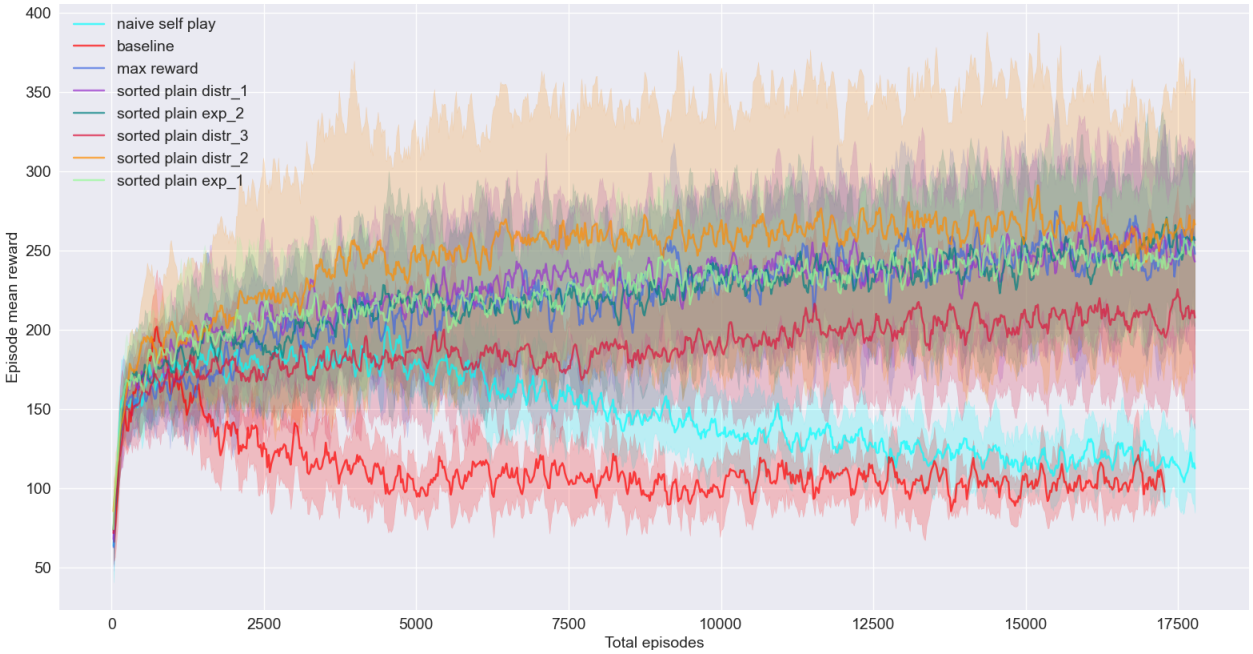


Figure 10: Average episode reward and standard deviation for "Sorted plain self-play" method

DISTRIBUTION A: Distribution approved for public release.

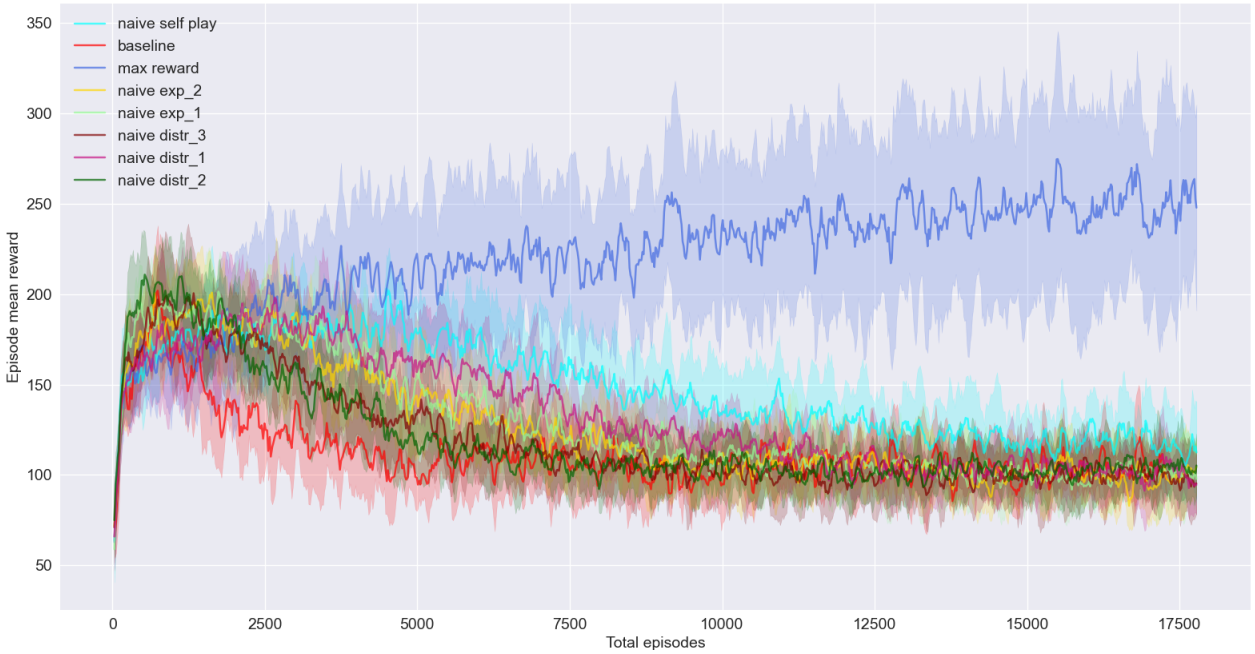


Figure 11: Average episode reward and standard deviation for "Naive self-play" method

has the worst performance with the fourth probability distribution having comparable performance with the second one.

In the "sorted plain self-play" method we notice in Fig. 10 that, as in the zero-sum environment, the second probability distribution has the best performance for most of the learning episodes, but has comparable performance with the second exponential distribution, as well as, the first distribution on the last learning episodes. Also, it has better performance than the maximum reward with random sampling for most of the initial part of the learning, leading to almost identical performance by the end of the learning, making it difficult to come to a certain conclusion regarding the role of the second probability distribution on the sorted self-play scheme.

In the "naive self-play" method in Fig. 11 we see that all the methods demonstrate catastrophic forgetting, having worse performance than naive self-play with random sampling or even the baseline in some cases. Also there is no improvement with the usage of probability distributions, highlighting again the role of sorting the menagerie.

In Fig. 12 we can see the performance of the naive self-play as well as the maximum reward self-play schemes, while increasing the menagerie size from 10 to 30 and then to 50. We notice that by increasing the menagerie size the performance is not improved, but rather slightly deteriorated, in the maximum reward scheme. Nevertheless, learning is a bit faster on the first 2500 episodes, since the same reward is achieved earlier, than when using the maximum rewards scheme with a menagerie of 10 policies. We, also, noticed that performance improves in the naive self-play scheme, achieving higher reward and less catastrophic forgetting, but, nevertheless, as the learning procedure continues catastrophic forgetting is still an issue. We believe that the reason for improvement in the naive self-play case is that by using more policies in the menagerie, the real policy space is approximated better by the menagerie, since in that case the menagerie is curated in a first in first out sense. On the other hand, on the maximum reward scheme, increasing the size

DISTRIBUTION A: Distribution approved for public release.



Figure 12: Average episode reward and standard deviation for "Naive self-play" method

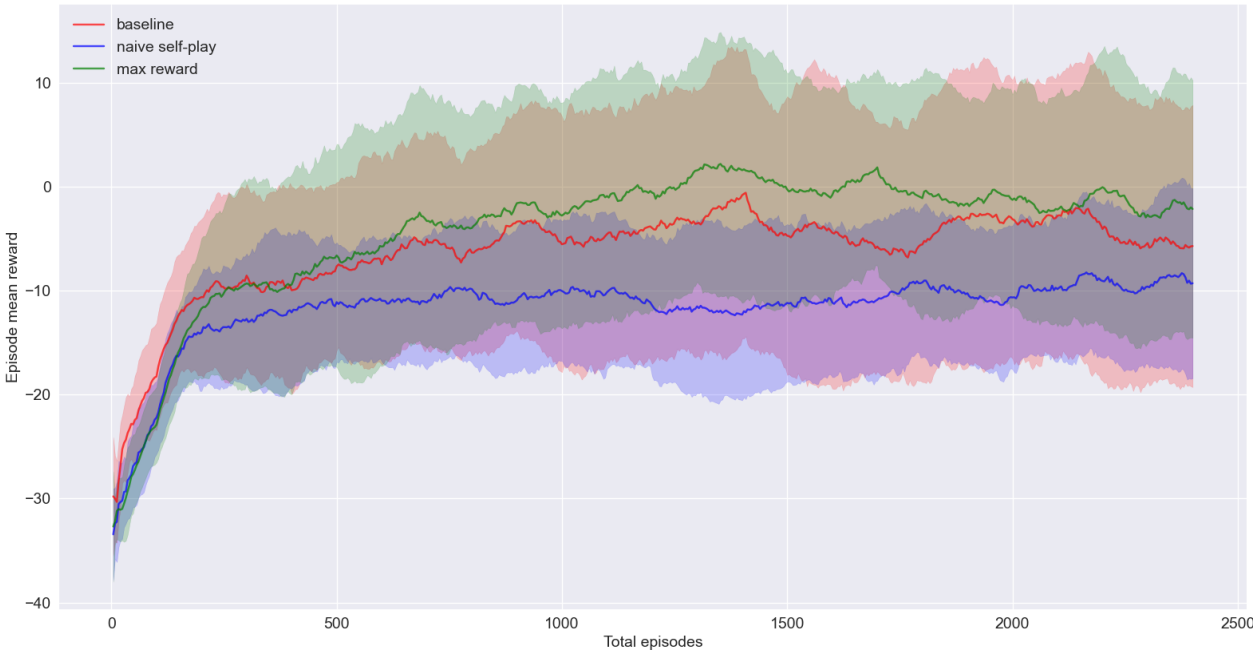


Figure 13: Average episode reward and standard deviation for "Sorted plain self-play" method

DISTRIBUTION A: Distribution approved for public release.

seems to hinder exploring efficiently other states, since the menagerie is, in that case, a sorted collection of the the policies with the highest reward.

2.4.3 Waterworld Environment

The "Waterworld" environment is more cooperative than the "Simple tag" one and therefore making it more difficult for self-play to work and achieve efficient learning performance. Nevertheless, in Fig. 13 we notice that the maximum reward scheme has better performance than the baseline method, while the naive self-play is worst than the baseline. Therefore, we conclude that by proposing new techniques for curating the menagerie using the policy reward as a criterion, self-play can improve the performance even in more cooperative complex environments. The influence of different sampling probability distributions to the performance will be researched on future work.

2.4.4 Discussion

Generally, we see that the maximum reward scheme has the best performance followed by the sorted plain self-play scheme, while the naive self-play scheme demonstrates catastrophic forgetting deteriorating performance a lot. It is apparent from the results that sorting the menagerie leads to improved performance, as well as, that choosing probability distributions that favour the policies that have the highest reward, in the already sorted menagerie, improve performance. The reason could be that using more the policies with the maximum reward, approximates better the whole policy space, while the policies do not dominate the policy space hindering continual learning.

As we can see the first exponential probability distribution that works the best, samples the latest entries of the menagerie, meaning the policies with the highest mean reward, more balanced, increasing gradually the selection probability from around 10% for the 5th entry to 25% for the last entry. The first probability distributions that sample with higher probability than the first exponential one, the initial entries of the menagerie that have lower reward demonstrate worse performance possibly due to not exploiting the highest reward policies as much as the other distributions. The distributions 2, 3 and 4 that sample the last entry, which has the highest reward, with a much bigger probability than the others, probably leading the policy with the best reward to dominate the policy space resulting to inferior performance, since less states that could improve performance are visited.

DISTRIBUTION A: Distribution approved for public release.

3 Deep Reinforcement Learning-Based Adaptive Laziness Allocation

In this chapter, we propose a DRL-based adaptive laziness allocation method for UAV flocking. The main idea is to refine and incorporate the concept of inactiveness into UAV flocking problems.

A few studies [33, 34, 35] have explored similar ideas and show valuable contributions to the field. However, they exhibit certain limitations in practical applications and differ from our approach. [33, 34] utilised Social-Learning Particle Swarm Optimisation (SL-PSO) [36], a meta-heuristic optimiser, to determine the inactivity level (i.e. inactiveness) of a UAV flock. The use of SL-PSO requires a considerable amount of computation, making it intractable. SL-PSO is an iterative optimiser designed for single-objective functions. In this context, an initial set of solutions, or particles, undergoes continuous update over many iterations. The studies used a maximum iteration count of 5000, with 100 particles evaluated in each iteration by simulating the flocking task until the UAVs show the desired flocking behaviour. Additionally, the inactivity level remains constant during the flocking; it is determined prior to the flocking mission, adding to the impracticality to the method. [35] proposes a heuristic approach that adapts the inactivity level of the selected agent according to its position and heading relative to the flock centre. It is worth noting that the selected agent and the inactivity level are termed the *lazy agent* and the *laziness*, respectively. This approach does not require any optimisation process and can be implemented in real time. However, there are several limitations to the approach. The lazy agents need to be determined prior to a flocking mission. Even if we are somehow able to determine which agents should be lazy, such as selecting the lazy agents every time step based on the same selection method, the number of lazy agents has to be still predetermined based on a posteriori insights. The authors argued that applying laziness onto only a single agent resulted in the best performance based on their tests where 20, 30, or 40 UAVs were deployed. This empirical result is violated as the number of UAVs increases, which is presented in a later section. In addition, the heuristic approach requires the user to manually design the rules. Using the same may degrade the performance until the user redesigns the rules when problem conditions change.

Since we cannot construct hand crafted laziness allocation rules for every problem condition, we can utilise a data-driven method: neural networks learn complex patterns. At first glance, supervised learning is straightforward and suitable for this approach, but extensive amount of data and the corresponding labels must be collected for supervised learning, which is a quite difficult job. Thus, we employ DRL, in particular the PPO [13]. The actor receives observations including position and velocity information of the flock and outputs the laziness of all agents in the flock. As such, the trained model determines which agents should be lazy and how much inactivity level should be applied to each lazy agent. Besides, once changes in flocking problem conditions are included in the training data set from the DRL environment, our approach can inherently address such changes, which is implemented automatically unlike the heuristic approach. Theses are the fundamental differences from the heuristic approach in [35].

We present the details of how the model is built and trained and provide numerical simulation results to evaluate our approach in various environments in comparison with the vanilla flocking algorithm (i.e. fully active flocking), the heuristic method, and an extension of the heuristic method. The simulation results demonstrate that our approach can outperform the others in various conditions.

DISTRIBUTION A: Distribution approved for public release.

3.1 Preliminaries

3.1.1 Flocking Algorithm

Inspired by bird clustering, Reynolds flocking rule [37] is a popular method for UAV flocking due to the simplicity and the convergence guarantee where Vicsek showed a group of agents can always form an angle alignment given arbitrary initial heading angles of the agents in [38]. Based on the Vicsek model, Cucker and Smale developed another flocking rule, called the Cucker-Smale (C-S) model in [39]. It enables velocity consensus of the agents and used in practical applications, such as spacecraft formation control [40]. In [41], Park *et al.* proposed the augmented C-S model as an extension of the C-S model. The study applies inter-particle bonding forces to preserve relative distances among the agents. However, the C-S model-based approaches may cause the flock to use significant amount of time and energy although they provide theoretical guarantees of velocity consensus under some conditions. In flocking tasks with fixed-wing UAVs, it is likely that the disadvantage enlarges because the maximum turn rate and acceleration causes harder convergence and this in turn leads to a large amount of energy and time consumption or divergence in the worst case.

We consider a group of agents and it follows a certain flocking rule. Here, an agent also means a UAV and the group of UAVs is also called as a flock. The terms are used interchangeably. Each agent is assumed to move a simple 2-D kinematics at a constant speed as follows:

$$\dot{x}_i = V cos\theta_i, \tag{3}$$

$$\dot{y}_i = V sin\theta_i, \tag{4}$$

$$\dot{\theta}_i = u_i, \quad \forall i = \{1, 2, ..., N\}, \tag{5}$$

where x_i and y_i are displacements in each axis and V, θ_i, u_i, and N denote the constant speed, the heading angle, the control input as the heading rate, and the flock size, respectively. Also, i indicates the agent index. The control input is constrained by the maximum turn rate: $|u_i| \leq u_{max}$.

$$u_i = \begin{cases} u_{\max} & \text{if } \tilde{u}_i > u_{\max} \\ -u_{\max} & \text{if } \tilde{u}_i < -u_{\max} \\ \tilde{u}_i & \text{otherwise,} \end{cases} \tag{6}$$

where $\tilde{u}_i$ is the unsaturated raw control input.

We utilise the augmented C-S model with the saturation constraint for the flocking control algorithm as in [35]. The control input is decomposed into the alignment term u_i^{CS} and bonding term $u_i^{bonding}$:

$$\tilde{u}_i = u_i^{CS} + u_i^{bonding}. \tag{7}$$

The angle alignment control of the i-th agent is calculated as:

$$u_i^{CS} = \frac{\lambda}{N} \sum_{j=1}^{N} \frac{\sin(\theta_j - \theta_i)}{(1 + r_{ij}^2)^{\beta}}, \tag{8}$$

where r_{ij} denotes the relative distance between the i-th and j-th agents and constants $\lambda > 0$, $\beta \geq 0$. The second term is responsible for the relative distance control as in cohesion and separation. Let $\mathbf{p}_i = [x_i, y_i]^T$

DISTRIBUTION A: Distribution approved for public release.

and $\mathbf{v}_i = [\dot{x}_i, \dot{y}_i]^T$. The bonding force term is calculated as:

$$u_i^{bonding} = \frac{\sigma}{NV} \sum_{j=1}^{N} \left(\frac{K_1}{2r_{ij}^2} \langle \mathbf{v}_i - \mathbf{v}_j, \mathbf{p}_i - \mathbf{p}_j \rangle + \frac{K_2}{2r_{ij}^2} (r_{ij}^2 - R) \right) \langle [-sin\theta_i, cos\theta_i]^T, \mathbf{p}_j - \mathbf{p}_i \rangle, \quad (9)$$

where R denotes a distance metric for desired relative distance, $\langle \cdot, \cdot \rangle$ represents inner product operation, and σ, K_1, and K_2 are positive constants.

3.1.2 Inactiveness in Flocking

Let us introduce a concept of inactiveness for the UAV flocking tasks to alleviate the issue in the C-S model-based methods. Inactiveness is a method that reduces the control inputs in a given control algorithm, aiming to improve the group performance such as the convergence time or the total energy consumption. As such, some agents use the reduced control inputs while the others use the control inputs from the algorithm. The former agents are termed inactive agents and the latter are termed active agents. Inactiveness was inspired by some phenomena in nature. In fact, in numerous species, a significant number of colony workers (i.e. agents) appear to remain inactive according to [42]. For instance, some individuals are relatively inactive while the others are active in an ant colony [43], and this improves a certain type of long-term performance of the group at the cost of short-term productivity [44]. Wang *et al.* showed that inactive workers reduce the convergence time and the total energy consumption in a circular formation problem in [45].

The inactiveness concept is introduced simply as discounting the control input, meaning that (5) should be modified as follows:

$$\dot{\theta}_i = C_{lazy,i} u_i, \quad (10)$$

where $C_{lazy,i} \in [0, 1]$. The discount factor $C_{lazy,i}$ is termed inactiveness; inactivity level, inactivity, and laziness are also interchangeably used. We think it must be noted that $C_{lazy,i} = 0$ indicates the agent to be (fully) inactive and $C_{lazy,i} = 1$ indicates the agent to be fully active. For example, when an agent has inactivity level of 0.99, it is almost fully active. In addition, $\mathbf{C}_{lazy}$ is the inactiveness of a flock and also called as laziness vector.

3.2 Problem Formulation

3.2.1 Scenario and Assumptions

We consider a scenario as follows. In a square area with a side length of L_{bound}, N UAVs are initialised at random positions at time $t = 0$. The UAVs moves with the dynamics of (3), (4), and (10). Here, we aim to find a good laziness vector $\mathbf{C}_{lazy}$ at every time step until the flock is converged. The terminal condition is met at time $t = t_c$ when the standard deviation of the flock position and velocity and their rate go below certain thresholds (see the details in [35]). Fig. 14 shows an example scenario considered.

For simplicity, we consider homogeneous agents, fully connected communication, and a centralised laziness vector computation. However, they can be readily extended as the flocking algorithm and the neural network model supports the extensions. We assume fully observable state across all agents, communication without delays, and discrete time control.

DISTRIBUTION A: Distribution approved for public release.

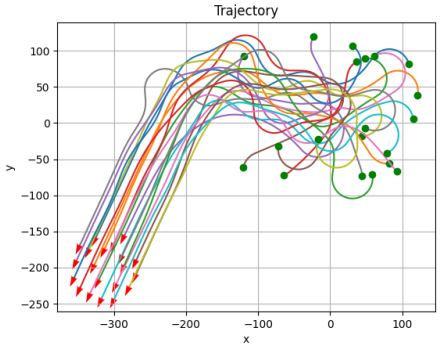


Figure 14: A sample scenario with 20 UAVs

3.2.2 Mathematical Formulation

State of i-th agent is defined as:

$$s_i = [x_i, y_i, \theta_i]^T. \tag{11}$$

The state of the flock is denoted as $\mathbf{s} \in \mathbb{R}^{(3\times N)}$. Let $\mathcal{F} : \mathbb{R}^{(3\times N)} \rightarrow \mathbb{R}^N$ be the mapping from the state to the control input by the flocking algorithm.

$$\mathcal{F}(\mathbf{s}) = \mathbf{u}. \tag{12}$$

Agent-wise mapping can also be denoted as $\mathcal{F}_i(s_i) = u_i$. A cost function is established as:

$$J\left(\mathbf{s}(t, t_c), \mathbf{C}_{lazy}\right) = \frac{V}{N} \sum_{i=1}^{N} \int_t^{t_c} |C_{lazy,i} \mathcal{F}_i(s_i(\tau))| d\tau + \rho(t_c - t), \tag{13}$$

where $\mathbf{s}(t, t_c)$ denotes the trajectory of the state from t to t_c and ρ is a positive weighting constant scalar. So, cost of the flocking task is $J(\mathbf{s}(0, t_c), \mathbf{C}_{lazy})$. For example, the cost of fully active flocking is $J(\mathbf{s}(0, t_c), \mathbf{1})$. The first term in the cost function represents the control cost, which only accounts for the heading angle control. The other term represents time cost. In this study we aim to find the optimal laziness vector. Thus, the formulation of the problem is given as follow:

$$\mathbf{C}^*_{lazy} = \underset{\mathbf{C}_{lazy}}{\arg\min}\, J\left(\mathbf{s}(0, t_c), \mathbf{C}_{lazy}\right). \tag{14}$$

3.3 Methods

In this section, we introduce the methodology used to incorporate the concept of inactivity into the UAV flocking mission using DRL. This includes establishing a DRL environment for the flocking tasks, defining the policy and value networks, and illustrating our training methods.

DISTRIBUTION A: Distribution approved for public release.

3.3.1 Deep Reinforcement Learning Environment

3.3.1.1 State and Observation

The RL environment state at time step k is defined as:

$$\mathbf{s}^{RL}(k) = [(\mathbf{s}(\mathbf{k}))^T, \mathcal{F}(\mathbf{s}(\mathbf{k}))]^T. \tag{15}$$

Note that $\mathbf{s}^{RL}(k) \in \mathbb{R}^{(4\times N)}$. This can be rewritten as an agent-wise form; the RL environment state of the i-th UAV is defined as:

$$s_i^{RL}(k) = [x_i(k), y_i(k), \theta_i(k), u_i(k)]^T. \tag{16}$$

It is crucial to point out that the flocking agent (i.e. UAV) is distinct from the RL agent as we use single agent RL in our approach, in which the two terminologies may sound confusing. When the concept of RL agents needs to be mentioned in this study, RL must hold onto 'agent'. Once an action is received, the current state is updated based on the UAV flock dynamics and the flocking algorithm $\mathcal{F}$ shown in (3), (4), (6), and (10).

We assume that the environment is fully active as the environment state is composed of information we can access. The observation is defined as follows:

$$o_i(k) = [x_i(k), y_i(k), \dot{x}_i(k), \dot{y}_i(k), u_i(k)]^T, \tag{17}$$

$$\mathbf{o}(k) = \{o_1(k), o_2(k), \cdots o_N(k)\}, \tag{18}$$

$$o^{flatten}(k) = [o_1^T(k), o_2^T(k), \cdots o_N^T(k)]^T. \tag{19}$$

This allows us to avoid discontinuity in the heading angle as a state element. In addition, we define the action:

$$a = \mathbf{C}_{lazy} \tag{20}$$

Although the (fully active) control input $\mathbf{u}(\mathbf{k})$ can be determined by the UAV agent states, we included it into the observation to guide the RL agent to understand how much it should reduce the control input. Empirically speaking this approach slightly worked better.

3.3.1.2 Reward Function

Let us revisit the problem formulation (14) and the cost function as in:

$$\mathbf{C}^*_{lazy} = \arg\min_{\mathbf{C}_{lazy}} \frac{V}{N} \sum_{i=1}^{N} \int_t^{t_c} |C_{lazy,i}\mathcal{F}_i(s_i(\tau))| d\tau + \rho(t_c - t). \tag{21}$$

Slightly modifying it:

$$\mathbf{C}^*_{lazy} = -\arg\max_{\mathbf{C}_{lazy}} \frac{V}{N} \sum_{i=1}^{N} \int_t^{t_c} (|C_{lazy,i}\mathcal{F}_i(s_i(\tau))| + \rho\,)\, d\tau. \tag{22}$$

In a discrete form, it can be rewritten as:

$$\mathbf{C}^*_{lazy} = -\arg\max_{\mathbf{C}_{lazy}} \frac{V}{N} \sum_{i=1}^{N} \sum_{k=t}^{t_c} (C_{lazy,i}(k)u_i(k) \cdot \Delta t + \rho\Delta t), \tag{23}$$

DISTRIBUTION A: Distribution approved for public release.

where Δt is the time step length. Rearranging it results in:

$$\mathbf{C}^*_{lazy} = -\arg\max_{\mathbf{C}_{lazy}} \sum_{k=t}^{t_c} \left(\sum_{i=1}^{N} \frac{V\Delta t}{N} \left(C_{lazy(k),i} u_i(k) + \rho \right) \right). \tag{24}$$

Removing the summation over the agents,

$$\mathbf{C}^*_{lazy} = \arg\max_{\mathbf{C}_{lazy}} \sum_{k=t}^{t_c} -\frac{V\Delta t}{N} \left(\langle \mathbf{C}_{lazy}(k),\, \mathbf{u}(k) \rangle + \rho N \right). \tag{25}$$

Equation (25) inspires us to establish the following reward function:

$$R(\mathbf{s}^{RL},\, a) = \begin{cases} -\frac{\Delta t}{N} \left(\langle \mathbf{C}_{lazy}(t),\, \mathbf{u}(t) \rangle + \rho N \right), & \text{if } 0 \le t < t_c \\ 0, & \text{if } t \ge t_c \end{cases}. \tag{26}$$

If we substitute (25) with (20) and (26),

$$a^* = \arg\max_{a} \sum_{t=0}^{t_c} \gamma^t R(t), \tag{27}$$

where $\gamma = 1$. This can also be written as:

$$a^* = \arg\max_{a} \sum_{t=0}^{\infty} \gamma^t R(t), \tag{28}$$

which clearly falls into an RL problem.

3.3.2 Policy and Value Models

In this study, the policy model is based on the transformer model [46] because of the two reasons. First, it can receive an input as a set, not a sequence. Second, it can receive arbitrary size of sets as its input. Making a set input requires to ignore positional encoding in transformer and enhances the sample efficiency significantly. For example, as shown in (18) and (19), when considering the sequence of o_i, $N!$ different $o^{flatten}$ are considered identical in $\mathbf{o}$. Receiving arbitrary size of sets potentially enables the model to inherently address the agent gain or loss. Even if the number of agents does not vary during the mission, it allows the model to learn from different number of agents and the user to use a single model for different agent number configurations. When an observation is received, each agent-wise observation (o_i) is considered as a token. Each token is passed through an embedding layer to represent the token into a higher dimension. In particular, the dimension of each token was enlarged from 5 to 128 in our policy model. Also, a linear layer was used as the embedding layer.

$$h_i^0 = W o_i + b, \tag{29}$$

where W and b denotes weight and bias of the embedding layer. And it was further processed by the encoder. An encoder is often composed of multiple encoder blocks, each of which is comprised of a Multi-Head Attention (MHA) layer, a node-wise fully connected Feed Forward (FF) layer, Layer Normalisation (LN) layer. In b-th encoder block out of n blocks ($n = 3$ in the study),

$$\begin{aligned} \tilde{h}_i &= LN^b \left(h_i^{b-1} + MHA_i^b(h_1^{b-1}, h_2^{b-1}, \cdots, h_N^{b-1}) \right), \\ h_i^b &= LN^m \left(\tilde{h}_i + FF(\tilde{h}_i) \right). \end{aligned} \tag{30}$$

DISTRIBUTION A: Distribution approved for public release.

In the decoder, a single decoder block was used for efficiency as we do not address generative outputs. In the MHA layer, the context embedding h_c^n was used as the query, where

$$h_c^n = \sum_{i=1}^{N} h_i^n. \tag{31}$$

In the decoder block similar to the encoder blocks,

$$\begin{aligned} \tilde{h}_c &= LN\left(h_c^n + MHA_c(h_1^n, h_2^n, \cdots, h_N^n)\right), \\ h_c^{n+1} &= LN\left(\tilde{h}_c + FF(\tilde{h}_c)\right). \end{aligned} \tag{32}$$

The output of the decoder goes an our extension of pointer network, Variant Pointer Network (vPN). Pointer networks [47] are a type of neural network and introduced to learn the conditional probability of output sequence in each element of which the network outputs the probability distribution over the input tokens to point. It was designed for categorical outputs, but we replaced the `softmax` layer with a `tanh` layer to put the output into a desired range. So, the last layer in the policy network,

$$out = \mathbf{C}_{lazy} = \frac{1 + \texttt{tanh}\left(vPN\left(h_c^{n+1}, (h_1^n, h_2^n, \cdots, h_N^n)\right)\right)}{2} \tag{33}$$

This ensures to place $out \in [0, 1]$. Note that we use deterministic action as it empirically worked better, resulting in direct output as the laziness vector.

As PPO was used as the RL algorithm in this study, a state value network was also needed. The value network simply shares most of the layers with the policy network. It diverges from the policy network at the decoder output h_c^{n+1} and passes thorough a fully connected FF layer. As such,

$$value_{out} = FF(h_c^{n+1}). \tag{34}$$

3.3.3 Training Details

To train the models effectively, the observations were preprocessed in two ways. One preprocessing is normalisation. It was carried out as follow:

$$o_i^{normalised}(k) = \left[\frac{x_i(k)}{L_{bound}}, \frac{y_i(k)}{L_{bound}}, \frac{\dot{x}_i(k)}{V}, \frac{\dot{y}_i(k)}{V}, \frac{u_i(k)}{u_{max}}\right]^T. \tag{35}$$

Placing neural network inputs within $[-1, 1]$ is generally desired for better and efficient training. The other preprocessing was to make the x-y-θ coordinate translation and rotation invariant. The flock's average heading and the centre was always put at the origin.

An auxiliary reward function was used during the training. As shown in Section 3.3.1.2, the reward function aligns with the cost function of the original problem. However, the training was not successful; the RL agent barely learned and did not outperform the fully active control most of the time. This might be because the landscape of the problem under the cost function was too tough, such as too many local optima. When we tried to find the laziness vector by employing the optimisation-based method in [33], the SL-PSO did not find any good laziness vector unless injecting the fully active control behaviour in the particles. In fact,

DISTRIBUTION A: Distribution approved for public release.

the author also incorporated the fully active particle as described in the report, which supports the complex landscape idea. So, another alternative reward function was used as shown below:

$$R^{new} = w_1 \frac{(\text{std}(x_i, y_i) - 0.75R - c_{11})^2}{c_{12}} + w_2 \frac{(\text{std}(\dot{x}_i, \dot{y}_i) - c_{21})^2}{c_{22}} + w_3 \frac{\Delta t}{N} \sum_{i=1}^{N} u_i^2, \tag{36}$$

where std$(\cdot, \cdot)$ computes the standard deviation of all arguments together for all agents and $c_{..}$ and $w_.$ are tunable parameters. c_{11} and c_{21} are offset parameters of the standard deviation errors; c_{12} and c_{22} are normalisation parameters to place the squared error terms around [0,1]; w_1, w_2, and w_3 weighting parameters to relatively prioritise certain errors. It should be noted that this reward function was inspired by the OpenAI's `Pendulum-v0` environment. They accounted for not only the control cost but also the state errors in the reward function. In this regard, the state error-related terms were added. Note that the constant penalty term was removed.

3.4 Results and Discussions

In this section, the proposed model is tested in various environments and the performance of it is compared with those of fully active model, the heuristic model, and the same heuristic model but with a slight modification.

3.4.1 Tested Models

The four models to test are introduced and presented below to describe how each model works.

3.4.1.1 Fully Active Model

It refers to a model that always generate the inactivity level vector as ones vector.

$$\mathbf{C}_{lazy} = \mathbf{1}, \quad \forall t, \tag{37}$$

where t represents the time step.

3.4.1.2 Heuristic Model

The heuristic model calculates the laziness vector at every time step. However, the index of lazy agent i_{lazy} is determined at $t = 0$. The inactivity level is determined by a concept of fitness of the agent $f_{i_{lazy}}(t)$ and a given gain G. Let us define the centroid of the flock, the position of the lazy agent, and the angle between the heading of the of the lazy agent and the vector from the agent position to the flock centroid at time t as $\mathbf{p}_c(t)$, $\mathbf{p}_{i_{lazy}}(t)$, and $\varphi_{i_{lazy}}(t)$. The fitness of the lazy agent is calculated as follows:

$$\alpha_i(t) = \begin{cases} \|\bar{\mathbf{r}}_i(0)\| & \text{if } t = 0 \\ \frac{\|\bar{\mathbf{r}}_i(t)\|}{\max_{\tau \in [0,t]} \|\bar{\mathbf{r}}_i(\tau)\|} & \text{if } t > 0, \end{cases} \tag{38}$$

DISTRIBUTION A: Distribution approved for public release.

$$\gamma_i(t) = \frac{1}{2}(1 - cos\varphi_i(t)), \tag{39}$$

$$f_i(t) = \alpha_i(t)\gamma_i(t), \tag{40}$$

$$i_{lazy} = \text{argmax}_j\, \alpha_j(0), \tag{41}$$

where $\bar{\mathbf{r}}_i(t) = \mathbf{p}_c(t) - \mathbf{p}_i(t)$ and $i = i_{lazy}$. The inactivity level of the lazy agent is calculated as:

$$C_{lazy,i}(t) = 1 - Gf_i(t), \quad t > 0, \tag{42}$$

where $i = i_{lazy}$, $G \in (0, 1]$, and $G = 0.8$ is used for the heuristic in [35]. Note that all elements in $\mathbf{C}_{lazy}(t)$ are 1 except $i = i_{lazy}$.

3.4.1.3 Heuristic Variant Model

The heuristic model use a predetermined lazy agent index, but we extend the approach to an adaptive one. Note that this model also allows a single lazy agent, termed heuristic variant model.

$$\alpha_i(t) = \begin{cases} \|\bar{\mathbf{r}}_i(0)\| & \text{if } t = 0 \\ \frac{\|\bar{\mathbf{r}}_i(t)\|}{\max_{\tau\in[0,t]} \|\bar{\mathbf{r}}_i(\tau)\|} & \text{if } t > 0, \end{cases} \tag{43}$$

$$\gamma_i(t) = \frac{1}{2}(1 - cos\varphi_i(t)), \tag{44}$$

$$f_i(t) = \alpha_i(t)\gamma_i(t), \tag{45}$$

$$C_{lazy,i}(t) = 1 - Gf_i(t), \quad t > 0, \tag{46}$$

$$i_{lazy}(t) = \text{argmax}_j\, \alpha_j(t). \tag{47}$$

Equations (43)–(45) show how to compute the fitness $f_i(t)$. Everything is the same but the lazy agent index as shown in (47).

3.4.1.4 Neural Network Model

In this case, it is straightforward: the neural network, the proposed policy network, outputs

$$\mathbf{C}_{lazy}(t), \tag{48}$$

which includes inactivity levels of all agents at time t.

3.4.2 Validation Environment

The validation environment refers to the type used in DRL training but introduces different initial states. This environment evaluates the base performances of the models. As illustrated in Table 3, the parameters align with those in the heuristic [35], ensuring fair comparison.

We created 200 different validation environments to evaluate the four models. In each episode, all four models initiate the flocking task with an identical environment state. The performance test outcomes for

DISTRIBUTION A: Distribution approved for public release.

Table 3: Environment parameters

Parameter	**Value**	**Unit**
Number of UAVs, N	20	
Speed, V	15	m/s
Predefined distance, R	60	m
Energy coefficient, ρ	1	
Inter agent strength, λ	5	
Bonding force strength, σ	1	
Constant gains, $(K-1, K_2)$	(1,3)	
Inactiveness gain, G	0.8	
Maximum turn rate, u_sat	8/15	rad/s
Initial location bound, L_{bound}	250	m
Communication decay rate, β	1/3	

Table 4: Results in the validation environment

Model	Episode reward		Flocking time (s)	
	Mean	Standard deviation	Mean	Standard deviation
Fully active	-244.32	134.98	591.04	237.83
Heuristic	-149.59	63.79	443.01	126.47
Heuristic variant	-159.95	66.82	482.27	154.87
Neural network	-147.38	43.06	452.92	99.94

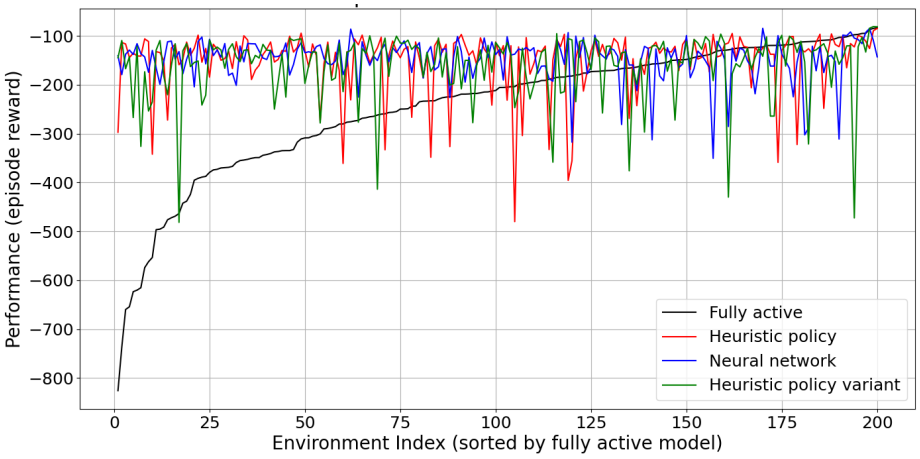


Figure 15: The four models tested in the validation environments with 200 different initialisations

DISTRIBUTION A: Distribution approved for public release.

Table 5: Performance in the test environment with $\lambda = 2.5$ and $\sigma = 2$

Model	Episode reward		Flocking time (s)	
	Mean	Standard deviation	Mean	Standard deviation
Fully active	-882.24	428.28	1424.54	587.36
Heuristic	-351.01	381.18	716.61	521.29
Heuristic variant	-250.36	176.85	604.88	292.13
Neural network	-287.46	220.14	665.82	345.02

these models in the validation environment are illustrated in Table 4 and Fig. 15. For every model, metrics such as episode reward and flocking time were assessed; these metrics are further divided into mean and standard deviation in Table 4. Given that the episode reward is the problem objective, its our primary focus; unless stated otherwise, any model's performance denotes its episode reward. The inactivity-based models(i.e. the heuristic, heuristic variant, neural network models) demonstrated superiority over the fully active model. Although neural network model slightly averagely-outperforms the heuristics, when accounting for variance, their average performance is nearly identical. Fig. 15 visually represents each model's performance with respect to the tested environment indices. It is essential to note that these indices are sorted by the performance of the fully active model, as the other three models derive from the same flocking control algorithm. Furthermore, the performance of the fully active model might correlate loosely with the extent of the flocking task executed up to that point. The neural network model displays relatively consistent performance across different environments. This consistency could be attributed to its more adaptive nature. Consequently, the learned inactivity not only improves average performance and convergence time but also the performance stability.

3.4.3 Test Environments

In real-world scenarios, RL agents often encounter unexpected scenarios. Such scenarios may result in unseen interactions between the RL agent and the environment in the training. To ensure the models can adapt to and perform under new environments, it is crucial to test their robustness and generalisation capabilities. For our assessment, we focus on five critical attributes: bonding force strength, inter-agent strength, speed, initial location bounds, and the number of UAVs. Bonding force strength and inter-agent strength are parameters of the flocking algorithm. They are grouped under the category of the control algorithm. Speed and initial location bounds are part of the RL environment state. We classify them as observations. By altering these parameters, we can evaluate the robustness of our models. Finally, varying the number of agents offers insights into how well the models can generalise across different scenarios.

3.4.3.1 Control Algorithm

In the base environment, the inter agent strength λ and the bonding force strength σ are 5 and 1, respectively. Here, we alter the flocking control algorithm parameters and test the models with them. Considering the fact that the control input is saturated in the fixed-wing UAV, the two parameters λ and σ are jointly altered. As

DISTRIBUTION A: Distribution approved for public release.

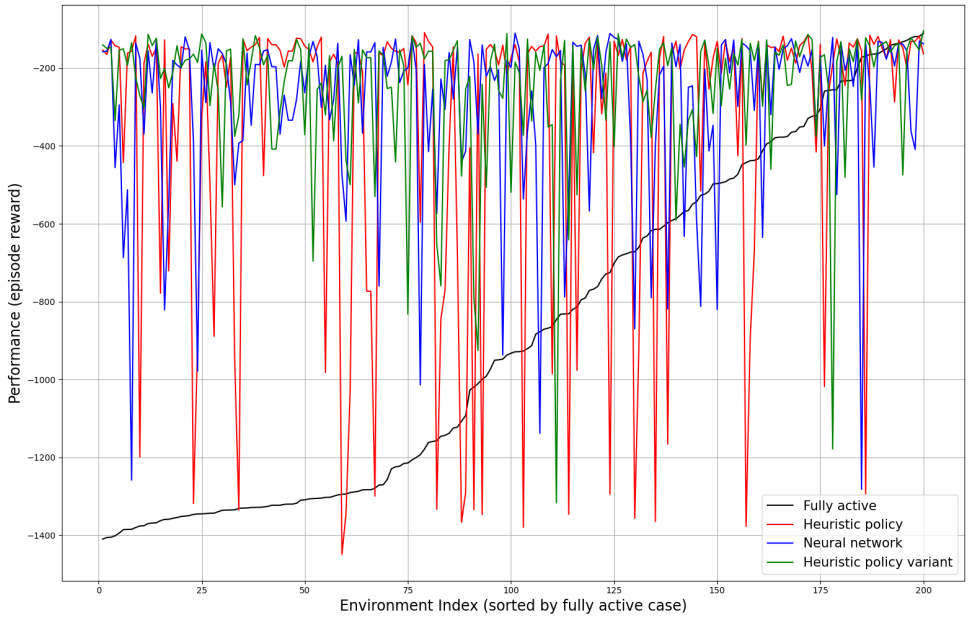


Figure 16: Performance in the test environment with $\lambda = 2.5$ and $\sigma = 2$

Table 6: Performance in the test environment with $\lambda = 10$ and $\sigma = 0.5$

Model	Episode reward		Flocking time (s)	
	Mean	Standard deviation	Mean	Standard deviation
Fully active	-104.12	30.50	349.23	186.51
Heuristic	-101.31	18.07	342.56	74.27
Heuristic variant	-109.39	25.49	376.61	144.94
Neural network	-108.80	23.73	383.68	105.52

a first experiment, $(\lambda, \sigma) = (2.5, 2)$ was tested; λ halves and σ doubles. The results are shown in Table 5 and Fig. 16. The fully active model significantly deteriorated. It did not even converge within the horizon in many cases. It should be noted that the other models must use this fully active model (i.e. the flocking algorithm) as they work based on the flocking algorithm. Nevertheless, the inactivity-based models managed to refine such control inputs. In fact, the model improvement rates from fully active control increased in the this test environment, although the performance of the lazy models decreased. In this aspect, the neural network model could be viewed robust against the environment changes.

This time λ doubles and σ halves. $(\lambda, \sigma) = (10, 0.5)$. Interestingly, fully active model performs really well in the setting. This may be due to the stronger alignment control inputs from the enlarged inter agent strength λ. A primary way that the lazy models work is to take detours that are more efficient in the long run. However, since the fully active model converges quickly as shown in Table 6, it is plausible that the lazy models have lower chance to enhance the fully active actions. So, it is hard to mention robustness of the models.

DISTRIBUTION A: Distribution approved for public release.



Figure 17: Performance in the test environment with $\lambda = 10$ and $\sigma = 0.5$

Table 7: Performance in the test environment with $L_{bound} = 500m$

Model	Episode reward		Flocking time (s)	
	Mean	Standard deviation	Mean	Standard deviation
Fully active	-258.57	124.38	673.33	228.07
Heuristic	-214.42	105.62	606.25	189.75
Heuristic variant	-210.00	77.18	617.54	149.99
Neural network	-231.29	99.67	662.59	179.66

3.4.3.2 Observation

This time the initial location bound L_{bound} is altered from 250 m to either 500 m or 125 m. It does not significantly change the fully active model performance as shown in Tables 7 and 8. That may be because this change does not change the fundamental mechanism of swarm evolution; it just changes degree of dispersion in their initial positions. The heuristic models do not change their performance much either whereas the neural network model relatively does. This is mainly because the higher initial dispersion in $L_{bound} = 500m$ largely promotes the initially large bonding force and gradually increasing angle alignment control inputs in the system. It may not leave enough room where the laziness works. On the other hand, the dense initial positions in $L_{bound} = 125m$ produces more emphasis on angle alignment instead of the bonding force. In this case, the ability of the heuristic variant (as well as the neural network) to vary lazy

Table 8: Performance in the test environment with $L_{bound} = 125m$

Model	Episode reward		Flocking time (s)	
	Mean	Standard deviation	Mean	Standard deviation
Fully active	-249.82	118.66	598.64	212.29
Heuristic	-204.93	93.98	545.84	178.79
Heuristic variant	-191.77	72.47	528.72	162.51
Neural network	-176.43	67.05	493.76	156.55

DISTRIBUTION A: Distribution approved for public release.

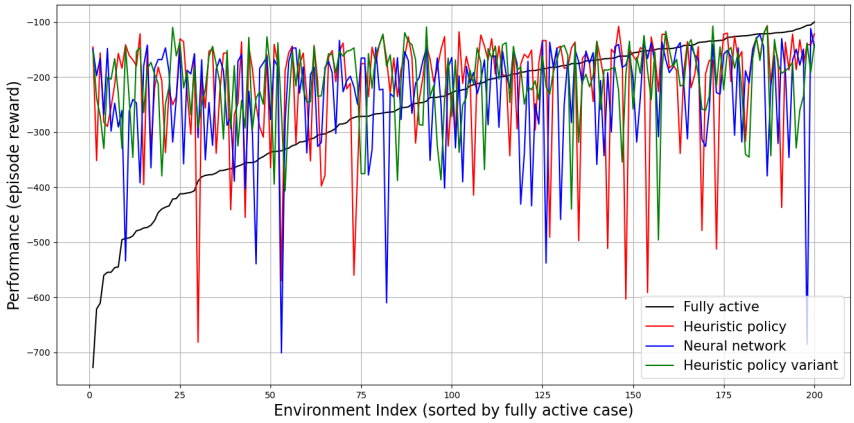


Figure 18: Performance in the test environment with $L_{bound} = 500m$

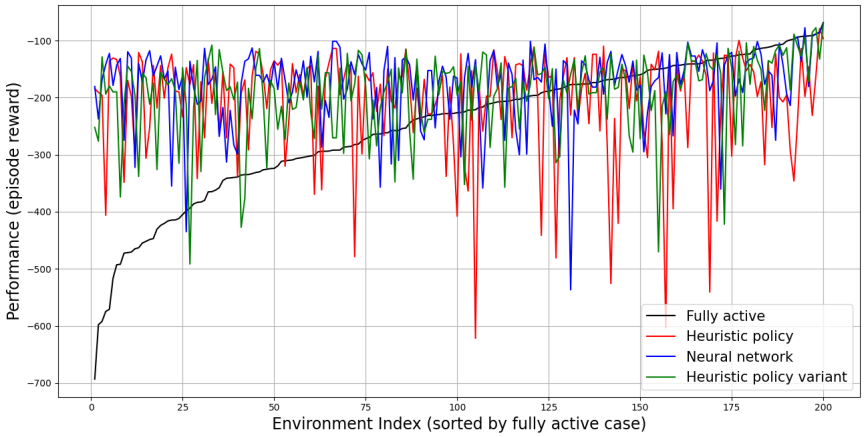


Figure 19: Performance in the test environment with $L_{bound} = 125m$

DISTRIBUTION A: Distribution approved for public release.

Table 9: Performance in the test environment with $V = 10m/s$

Model	Episode reward		Flocking time (s)	
	Mean	Standard deviation	Mean	Standard deviation
Fully active	-101.21	23.32	371.87	66.75
Heuristic	-91.32	20.86	354.99	65.06
Heuristic variant	-95.91	18.46	368.30	59.67
Neural network	-95.92	16.51	389.29	60.19

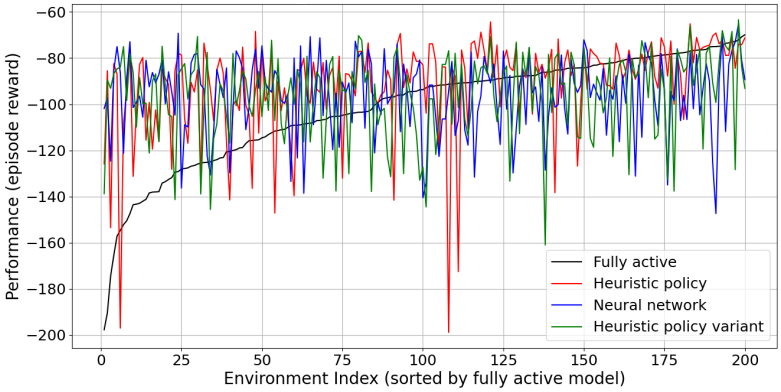


Figure 20: Performance in the test environment with $V = 10m/s$

agent index contributes to more adaptively find effective lazy behaviour. Also, the same idea is applied to the neural network model considering that it is able to have arbitrary number of lazy agents.

We also have tried to adjust the speed of UAVs. With the lower speed $V = 10m/s$, the fully active model performs very well and the things are working as same as control algorithm with $\lambda = 10$ and $\sigma = 0.5$. This is shown in Table 9 and Table 20. On the other hand, when the speed increased $V = 20m/s$, the problem gets very difficult. In most of the cases, the fully active model did not converge as shown in Table 10 and Fig. 21. However, the neural network model showed relatively higher and stable performance. One interesting fact to note is that the neural network model is not capable to observe speed because the observation normalisation reduces the speed information.

Table 10: Performance in the test environment with $V = 20m/s$

Model	Episode reward		Flocking time (s)	
	Mean	Standard deviation	Mean	Standard deviation
Fully active	-1575.65	302.11	1915.81	277.47
Heuristic	-1108.10	600.72	1501.93	599.90
Heuristic variant	-769.14	444.34	1296.10	533.16
Neural network	-592.68	298.72	1251.91	511.97

DISTRIBUTION A: Distribution approved for public release.

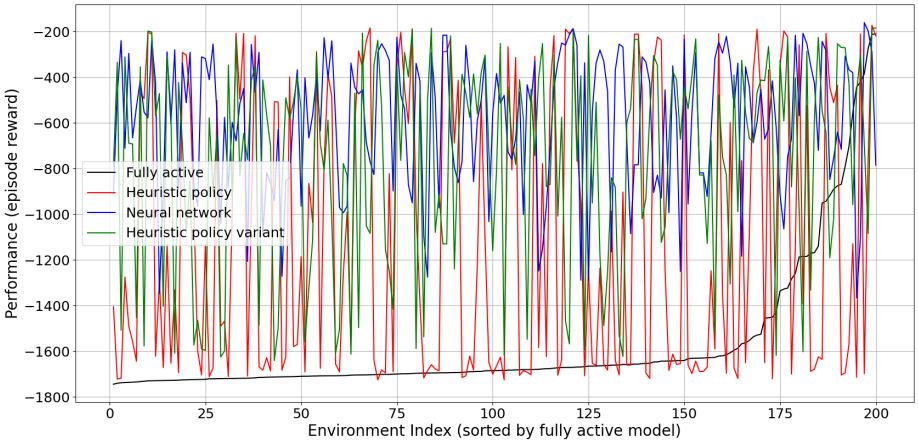


Figure 21: Performance in the test environment with $V = 20m/s$

Table 11: Performance in the test environment with 100 UAVs

Model	Episode reward		Flocking time (s)	
	Mean	Standard deviation	Mean	Standard deviation
Fully active	-408.65	143.92	947.51	257.10
Heuristic	-315.71	149.22	791.06	276.06
Heuristic variant	-377.43	158.17	903.08	289.11
Neural network	-147.89	25.01	509.18	94.91

3.4.3.3 The Number of Agents

We also have tested different number of UAVs. When the number of UAVs increased, it was quite interesting. The models were tested with 100 UAVs and it is presented in Table 11 and Fig. 22. The neural network model clearly outperformed the others. It maintained its performance and the stability compared with the model in the base environment (i.e. 20 UAVs) whereas the other models significantly deteriorated. This discrepancy enlarged when more agents were used. Table 12 and Fig. 23 show the performance of the models in the environment with 400 UAVs. The neural network model displays the best performance in every environment initial conditions out of the 300 different ones. Thus, we assume the neural network model to generalise in the number of UAVs, especially a large number of UAVs.

Table 12: Performance in the test environment with 400 UAVs

Model	Episode reward		Flocking time (s)	
	Mean	Standard deviation	Mean	Standard deviation
Fully active	-479.82	134.97	1095.25	251.76
Heuristic	-478.02	128.71	1091.02	251.38
Heuristic variant	-499.88	140.28	1167.85	287.20
Neural network	-152.10	19.60	533.73	86.18

DISTRIBUTION A: Distribution approved for public release.

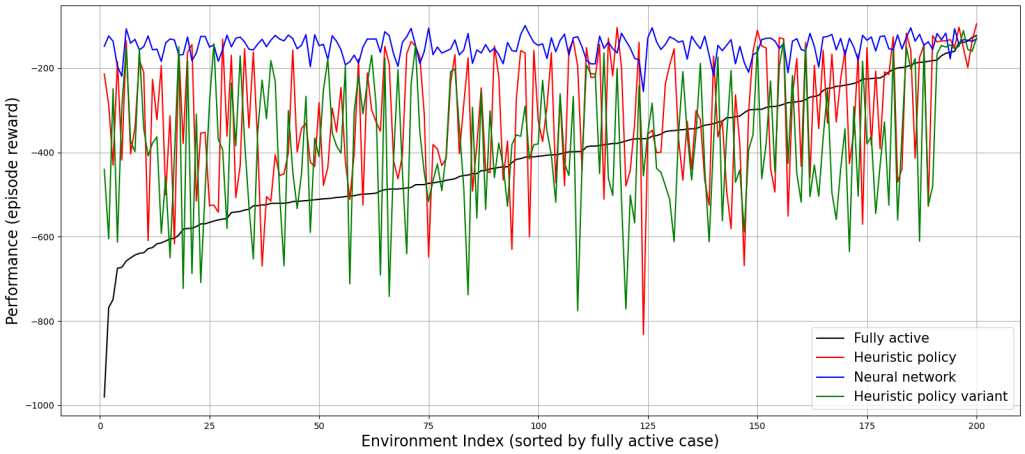


Figure 22: Performance in the test environment with 100 UAVs

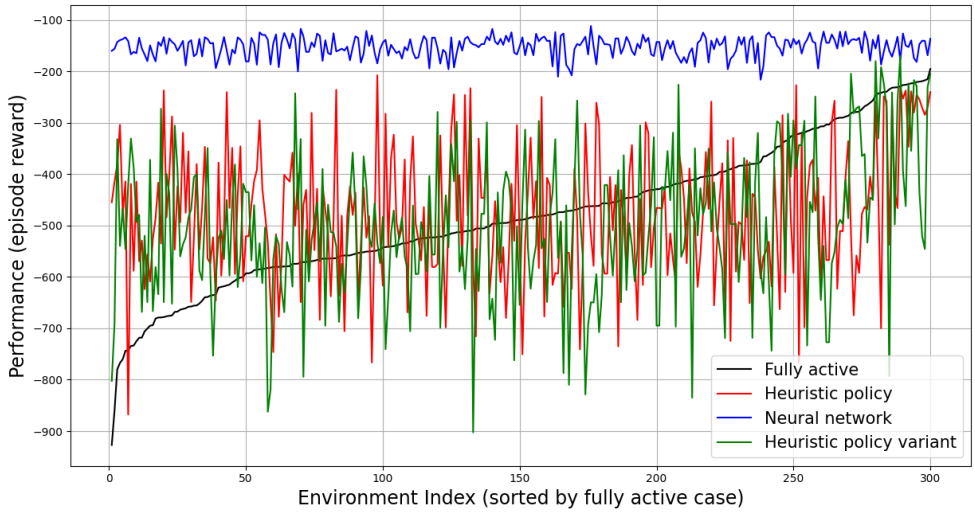


Figure 23: Performance in the test environment with 400 UAVs

Table 13: Performance in the test environment with 5 UAVs

Model	Episode reward		Flocking time (s)	
	Mean	Standard deviation	Mean	Standard deviation
Fully active	-186.92	80.38	469.52	161.11
Heuristic	-178.23	80..50	476.92	143.04
Heuristic variant	-161.54	46.99	509.24	166.11
Neural network	-187.50	80.48	491.52	181.27

DISTRIBUTION A: Distribution approved for public release.

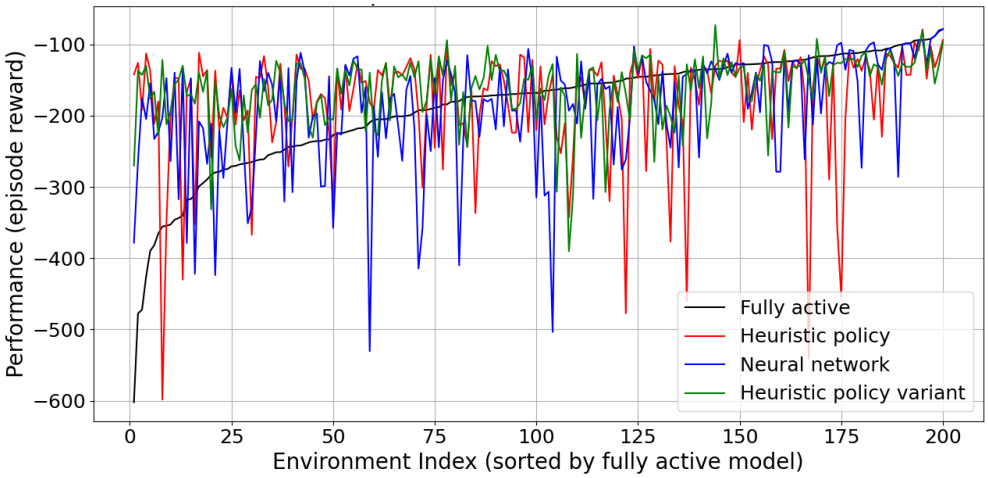


Figure 24: Performance in the test environment with 5 UAVs

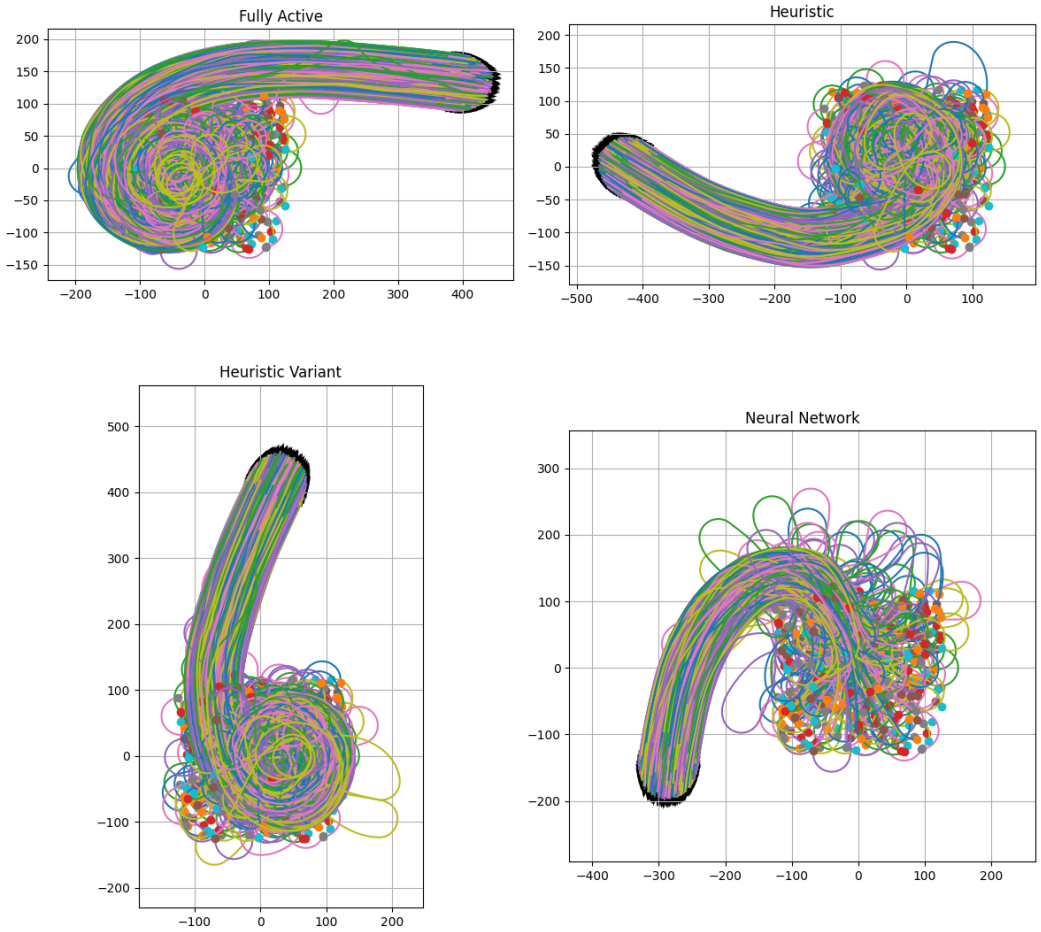


Figure 25: Flocking behaviour of each model with 400 UAVs

DISTRIBUTION A: Distribution approved for public release.

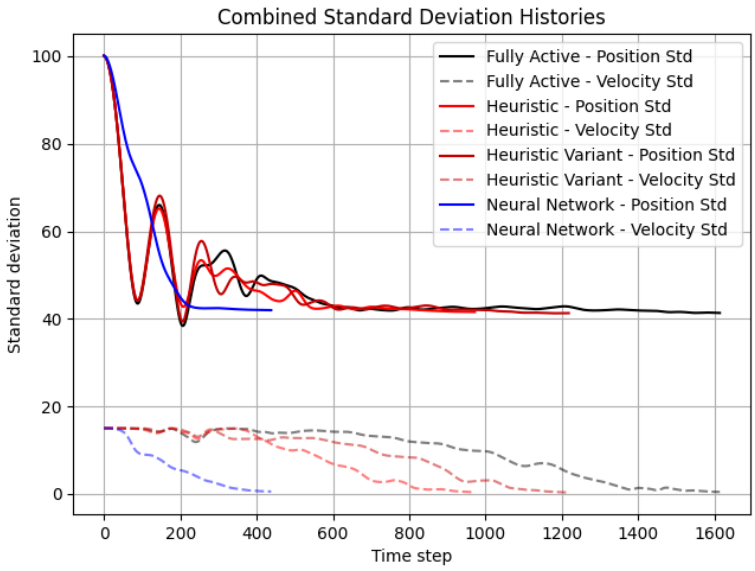


Figure 26: Standard deviation histories of each model with 400 UAVs

However, it goes quite different story when a few UAVs are used in the environment. As shown in Table 13 and Fig. 24, the neural network model did not show remarkable performance there. One possible explanation is that this flocking task does not strongly or consistently require multiple lazy agents when the number of agents are not large enough. In fact, [35] argues a single lazy agent was the best number of (highly) inactive agents for 20–40 UAVs. On the contrary, larger number of UAVs can require multiple number of inactive agents, which is a distinct attribute of the proposed neural network approach. This can be found in Fig. 25 and led to a great performance in Fig. 26. In Fig. 25, there is one lazy detour of the blue line in the heuristic trajectory whereas there are multiple lazy detours in the neural network trajectory.

DISTRIBUTION A: Distribution approved for public release.

4 Conclusion

In this performance period, we proposed some new self-play schemes, extending the usage of self-play to general sum games. We also highlighted the role of different methods for curating the menagerie, as well as, the role of different sampling distributions for choosing a policy from the menagerie, that is something researched for the first time, to the best of our knowledge.

Also, we proposed a task allocation algorithm for a UAV flocking control. Instead of building a new control algorithm, we developed a task allocation algorithm on top of an existing flocking control algorithm. The task refers to simply reducing the control input of the existing flocking controller, namely an answer to who should be lazy. In order to learn a complex pattern from the multiple agents, a DRL was utilised. The numerical simulation demonstrated that the proposed task allocation algorithm outperforms some existing approaches in some conditions. Also, robustness and generalisation capability was also tested under a few certain changes in the environment. In particular, the proposed algorithm showed significantly robust against the number of UAVs.

Our aim for future research is to find more efficient and complex heuristics for curating the menagerie, as well as, to research more the role of probabilities distributions on different types of environments. Another interesting topic is the effect of the size of the menagerie to the system's performance, as well as the effectiveness of self-play algorithms to more cooperative settings.

Summarising the accomplishments of this performance period and comparing with the last year's reports, the developed algorithms are expected to contribute to the research objectives especially on:

- Adaptive to dynamic changes in the system, communication network and environment;
- Robust to uncertainties with incomplete and corrupted information;
- Capable to learn and improve.

DISTRIBUTION A: Distribution approved for public release.

Reference

[1] G. M. Skaltsis, H.-S. Shin, and A. Tsourdos, "A survey of task allocation techniques in mas," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2021, pp. 488–497.

[2] ——, "A survey of task allocation techniques in mas," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2021, pp. 488–497.

[3] X. Lang and A. de Ruiter, "A control allocation scheme for spacecraft attitude stabilization based on distributed average consensus," *Aerospace Science and Technology*, vol. 106, p. 106173, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1270963820308555

[4] H. Oh, A. Ramezan Shirazi, C. Sun, and Y. Jin, "Bio-inspired self-organising multi-robot pattern formation: A review," *Robotics and Autonomous Systems*, vol. 91, pp. 83–100, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889016300185

[5] M. Ballerini, N. Cabibbo, R. Candelier, A. Cavagna, E. Cisbani, I. Giardina, V. Lecomte, A. Orlandi, G. Parisi, A. Procaccini, M. Viale, and V. Zdravkovic, "Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1232–1237, 2008. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.0711437105

[6] P. Leitão, J. Barbosa, and D. Trentesaux, "Bio-inspired multi-agent systems for reconfigurable manufacturing systems," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 5, pp. 934–944, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197611001886

[7] J. Choi, S. Oh, and R. Horowitz, "Distributed learning and cooperative control for multi-agent systems," *Automatica*, vol. 45, no. 12, pp. 2802–2814, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000510980900435X

[8] G. M. Skaltsis, H.-S. Shin, and A. Tsourdos, "Self play with parameter sharing in n-player mixed competitive-cooperative games," in *AIAA SCITECH 2022 Forum*, 2022. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2022-2498

[9] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, "Multi-agent reinforcement learning: A review of challenges and applications," *Applied Sciences (Switzerland)*, vol. 11, no. 11, 2021.

[10] Y. T. Tian, M. Yang, X. Y. Qi, and Y. M. Yang, "Multi-robot task allocation for fire-disaster response based on reinforcement learning," *Proceedings of the 2009 International Conference on Machine Learning and Cybernetics*, vol. 4, no. July, pp. 2312–2317, 2009.

[11] D. B. Noureddine, A. Gharbi, and S. B. Ahmed, "Multi-agent deep reinforcement learning for task allocation in dynamic environment," *ICSOFT 2017 - Proceedings of the 12th International Conference on Software Technologies*, no. Icsoft, pp. 17–26, 2017.

[12] T. S. Dahl, M. J. MatariĆ, and G. S. Sukhatme, "A Machine Learning Method for Improving Task Allocation in Distributed Multi-Robot Transportation," in *Complex Engineered Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, vol. 2006, pp. 307–337. [Online]. Available: http://link.springer.com/10.1007/3-540-32834-3{_}14

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[14] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, *A survey and critique of multiagent deep reinforcement learning*. Springer US, 2019, vol. 33, no. 6. [Online]. Available: https://doi.org/10.1007/s10458-019-09421-1

[15] A. Feriani and E. Hossain, "Single and Multi-Agent Deep Reinforcement Learning for AI-Enabled Wireless Networks: A Tutorial," *IEEE Communications Surveys and Tutorials*, vol. 23, no. 2, pp. 1226–1252, 2021.

[16] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*. Springer, 2017, pp. 66–83.

[17] J. K. Terry, N. Grammel, A. Hari, L. Santos, and B. Black, "Revisiting parameter sharing in multi-agent deep reinforcement learning," 2021.

[18] Y. Yang and J. Wang, "An Overview of Multi-Agent Reinforcement Learning from Game Theoretical Perspective," *arXiv*, pp. 1–129, 2020. [Online]. Available: http://arxiv.org/abs/2011.00583

[19] D. Hernandez, K. Denamganaï, S. Devlin, S. Samothrakis, and J. A. Walker, "A comparison of self-play algorithms under a generalized framework," *IEEE Transactions on Games*, vol. 14, no. 2, pp. 221–231, 2022.

[20] H. Hu, A. Lerer, A. Peysakhovich, and J. Foerster, ""Other-Play"for zero-shot coordination," *37th International Conference on Machine Learning, ICML 2020*, vol. PartF16814, pp. 4349–4360, 2020.

DISTRIBUTION A: Distribution approved for public release.

[21] J. Heinrich and D. Silver, "Deep Reinforcement Learning from Self-Play in Imperfect-Information Games," *arXiv*, 2016. [Online]. Available: http://arxiv.org/abs/1603.01121

[22] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=Sy0GnUxCb

[23] B. Baker, "Emergent reciprocity and team formation from randomized uncertain social preferences," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[24] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SkxpxJBKwS

[25] B. Baker, "Emergent reciprocity and team formation from randomized uncertain social preferences," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[26] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[27] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017. [Online]. Available: http://dx.doi.org/10.1038/nature24270

[28] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente *et al.*, "Pettingzoo: Gym for multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 032–15 043, 2021.

[29] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2018.

[30] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6382–6393.

[31] J. K. Terry, B. Black, M. Jayakumar, A. Hari, L. Santos, C. Dieffendahl, N. L. Williams, Y. Lokesh, R. Sullivan, C. Horsch, and P. Ravi, "Pettingzoo: Gym for multi-agent reinforcement learning," *arXiv preprint arXiv:2009.14471*, 2020.

[32] G. M. Skaltsis, H.-S. Shin, and A. Tsourdos, "Self play with parameter sharing in n-player mixed competitive-cooperative games," in *AIAA SCITECH 2022 Forum*, 2022. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2022-2498

[33] Y. Song, J. Choi, H. Oh, M. Lee, S. Lim, and J. Lee, "Improvement of decentralized flocking flight efficiency of fixed-wing uavs using inactive agents," in *AIAA SciTech 2019 Forum*, 2019, p. 0391.

[34] S. Lim, Y. Song, J. Choi, H. Myung, H. Lim, and H. Oh, "Decentralized hybrid flocking guidance for a swarm of small uavs," in *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, 2019, pp. 287–296.

[35] Y. Song, M. Gu, J. Choi, H. Oh, S. Lim, H.-S. Shin, and A. Tsourdos, "Using Lazy Agents to Improve the Flocking Efficiency of Multiple UAVs," *Journal of Intelligent & Robotic Systems*, vol. 103, no. 3, p. 53, 2021.

[36] R. Cheng and Y. Jin, "A social learning particle swarm optimization algorithm for scalable optimization," *Information Sciences*, vol. 291, pp. 43–60, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025514008366

[37] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 25–34. [Online]. Available: https://doi.org/10.1145/37401.37406

[38] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet, "Novel type of phase transition in a system of self-driven particles," *Phys. Rev. Lett.*, vol. 75, pp. 1226–1229, Aug 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.75.1226

[39] F. Cucker and S. Smale, "Emergent behavior in flocks," *IEEE Transactions on Automatic Control*, vol. 52, no. 5, pp. 852–862, 2007.

DISTRIBUTION A: Distribution approved for public release.

[40] L. Perea, G. Gómez, and P. Elosegui, "Extension of the cucker-smale control law to space flight formations," *Journal of guidance, control, and dynamics*, vol. 32, no. 2, pp. 527–537, 2009.

[41] J. Park, H. J. Kim, and S.-Y. Ha, "Cucker-smale flocking with inter-particle bonding forces," *IEEE Transactions on Automatic Control*, vol. 55, no. 11, pp. 2617–2623, 2010.

[42] D. Charbonneau and A. Dornhaus, "Workers 'specialized'on inactivity: behavioral consistency of inactive workers and their role in task allocation," *Behavioral ecology and sociobiology*, vol. 69, pp. 1459–1472, 2015.

[43] N. Pinter-Wollman, J. Hubler, J.-A. Holley, N. R. Franks, and A. Dornhaus, "How is activity distributed among and within tasks in temnothorax ants?" *Behavioral Ecology and Sociobiology*, vol. 66, pp. 1407–1420, 2012.

[44] E. Hasegawa, Y. Ishii, K. Tada, K. Kobayashi, and J. Yoshimura, "Lazy workers are necessary for long-term sustainability in insect societies," *Scientific reports*, vol. 6, no. 1, p. 20846, 2016.

[45] C. Wang and G. Xie, "Lazy workers benefit group performance in circle formation tasks**this work was supported in part by grants from the national natural science foundation of china (nsfc, no. 61503008, 91648120, 51575005, 61633002) and the china postdoctoral science foundation (no. 2015m570013, 2016t90016)," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 10 383–10 388, 2017, 20th IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S240589631732308X

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[47] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf

DISTRIBUTION A: Distribution approved for public release.

Appendix: Real-time Decision Making for Autonomous Systems using AI Methods

Prof Vaios Lappas

Ioannis Daramouskas

Dimitrios Meimetis

Niki Patrinopoulou

Introduction

The decision-making capability is an important attribute, essential for designing autonomous and intelligent systems. Agent-based real-time decision-making based on the data collected by the swarm is proven that can increase the efficiency of the solution and remain robust to dynamic changes and uncertainties.

Multiple-object detection, localization, and tracking are desirable in many areas and applications, as the field of deep learning has developed and has drawn the attention of academics in computer vision, having a plethora of networks now achieving excellent accuracy in detecting multiple objects in an image. Tracking and localizing objects still remain difficult processes which require significant effort.

The aim of this work is to examine the efficiency of a decision-making algorithm for the swarm in detecting desired objects in an area of interest and evaluate the methods with a series of metrics in different scenarios ensuring that the swarm can operate autonomously and safely regarding the inter-agent collisions.

The purpose of our work is to create a system for detecting, localizing, tracking, and counting objects in a specific area using drones. We created a global tracker system capable of localizing, tracking, and distinguishing objects detected from different sources (drones) during their autonomous flights and based on that information making informed decision regarding their next task.

Literature

DISTRIBUTION A: Distribution approved for public release.

In the field of computer vision and more specifically object detection, there has been immense research over the past decade. Authors in (Palak, Anshika, Shailendra, & Pradeep, 2016), have analysed the object recognition methods in both static images and videos. Object recognition, according to their research, may be applied in a variety of sectors including robot navigation, medical diagnosis, security, industrial inspection and automation, human-computer interaction, and information retrieval. Authors in (Wang, Shi, Song, & Shen, 2013), have developed an object detection method combining top-down recognition, there are two main steps in their method: a hypothesis generation step and a verification step. They improve the Shape Context feature in the top-down hypothesis generation stage, making it more resistant to object deformation and backdrop clutter. The authors of (Meera & Mohan, 2016)have divided an object detection method into two phases. A classifier was used to classify the query image in the first step. They used two types of classifiers for classifier optimization: a Support Vector Machine (SVM) classifier that uses GIST features and a k nearest neighbor (kNN) classifier that uses Scale Invariant Feature Transform (SIFT). Different kernels, such as linear, polynomial, and Gaussian kernels, were used in GIST-based SVM classification. Coil-20P and Eth80 were the image datasets used in this study providing state of the art results. In (Sanchez-Matilla, Poiesi, & Cavallaro, 2016), the authors offer an online multi-target tracker that uses a Probability Hypothesis Density Particle Filter architecture to exploit both high-confidence and low-confidence target detections. After the prediction step, the authors develop an early association method between trajectories and detections, which allows target estimation and state labeling to be performed without the need of any extra processes. The authors of (Bewley, Ge, Ott, Ramos, & Upcroft, 2016)propose a method for multi-object tracking that focuses on effectively associating items for online and real-time applications. To this aim, detection quality has been found as a significant influencing element in tracking performance, with altering the detector improving tracking by up to 18.9%. The solution delivers accuracy similar to state-of-the-art online trackers while just utilizing a simple mix of common techniques such as the Kalman Filter and Hungarian algorithm for the tracking components. The authors in (Chu & Ling, 2019)offer an end-to-end model, dubbed FAMNet, in which feature extraction, affinity estimation, and multi-dimensional assignment are all improved in one network. FAMNet's layers are all meant to be differentiable, allowing them to be tuned together to learn discriminative features and a higher-order affinity model. To recover false negatives and suppress noisy target candidates generated by the external detector, the authors include a single object tracking approach and a dedicated target management scheme into the FAMNet-based tracking system. Lastly, authors in (Meimetis, Daramouskas, Perikos, & Hatzilygeroudis, 2021)have thoroughly evaluated the performance of different YOLO models and a modified Deep Sort algorithm in regard to real time detection and tracking on pedestrian and car traffic datasets.

Problem Formulation

World

Description

The area of interest, in which the swarm operates and must provide surveillance is described as a rectangle with the size of 250x250 meters on the 2D horizontal plane. An infrastructure of interest (security asset) is located in the center of the area of interest. The drones are used to detect, identify and report people present in the area close to this asset before they reach the center of the world.

DISTRIBUTION A: Distribution approved for public release.

The area of interest includes some spots with dense flora (trees). When a person is situated in one of those spots, the drone's capability to detect them is decreased as the trees act as visual obstacles, creating occlusions, and the person has a better chance of not being detected. For this reason, those are described as hide spots for the proposed system. Parts of the world can be seen in Figure 1 and Figure 2.

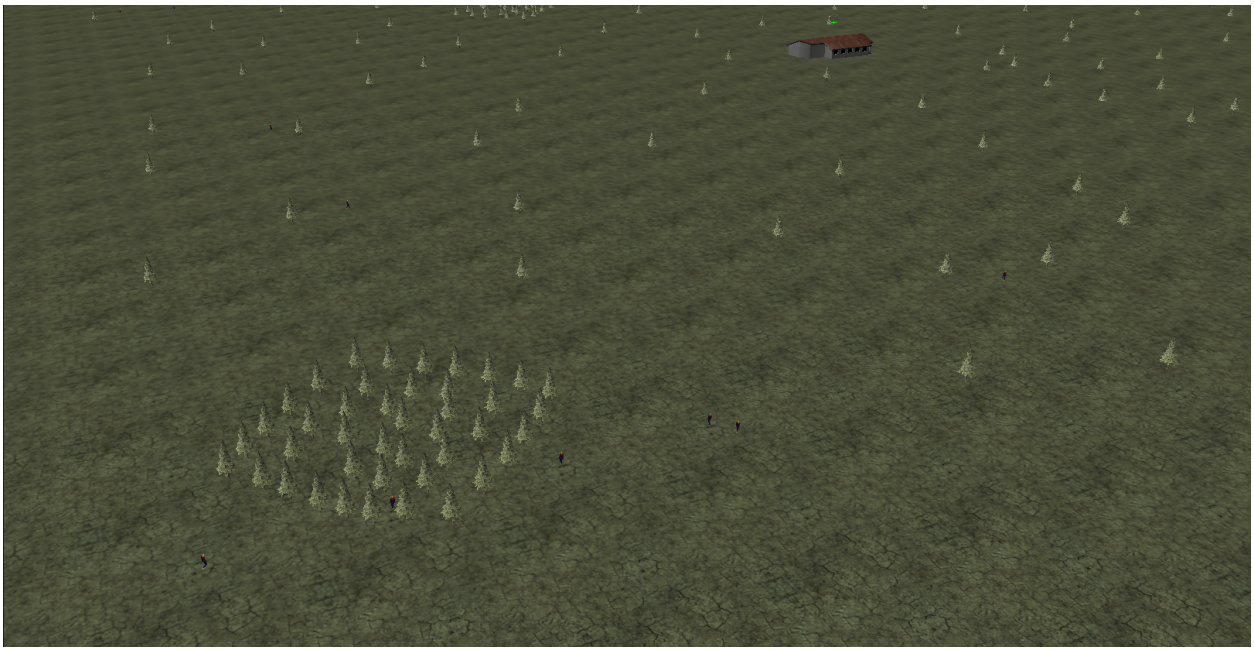


Figure 1: Hide Spots in the world with dense trees.

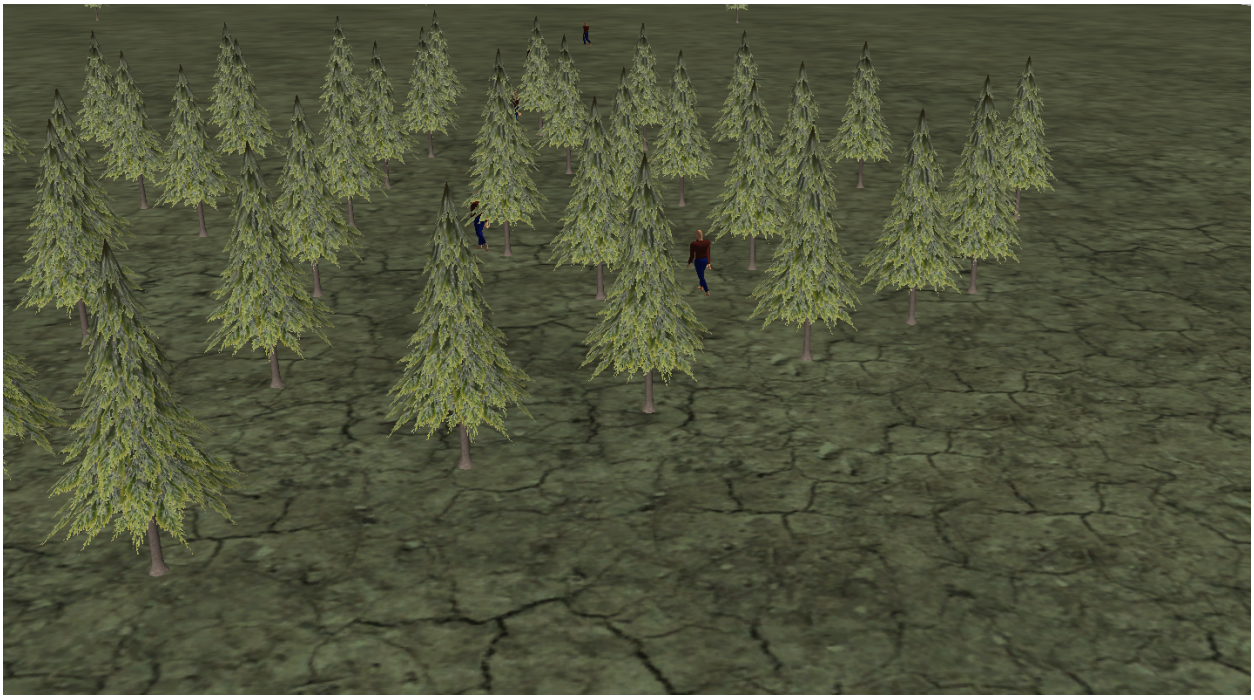


Figure 2: Hide Spots in the world with dense trees.

DISTRIBUTION A: Distribution approved for public release.

Representation

For the purpose of the proposed surveillance system, the area of interest is discretized into a 2D grid of cells of 10 meters in length. This allows each cell to be translated into one task from the swarm of drones. Additionally, this representation actually is a matrix with 50 rows and 50 columns and as many desirable channels, allowing us to values to each cell. The age is the first value we have which is used to describe the time elapsed since it was last surveyed for a specific cell. In addition to cell ages, the applied decomposition into a grid allows to associate more values with each cell. The focus of this work is not to cover the entirety of the area of interest in the highest frequency, but to detect the people in the shortest time and if necessary, perform target tracking to acquire more information. Based on this scope, a probability was added to the cells, in addition the cell's age, to signify the chance of a person currently being located in each cell. Another information is used is if the specific cell is an obstacle, so it can be used by the drone's path planning algorithm to avoid it. So, each cell contains the following information:

- gridX : the x location of the cell in array mapping (0-50)
- gridY: the y location of the cell in array mapping (0-50)
- timeOfLastVisit: the time elapsed since it was last surveyed for a specific cell
- importance: Boolean value about the presence of people in this specific cell
- isObstacle: Boolean value represent if the grid is an obstacle

In Figure 3: The cells of the world in a 2d plt image.Figure 3 the cells of the world are presented. We can see that we have four drones (colored big dots), and the colored dots represents the path that the drone has, the yellow rectangles represent the actors in the world, the red rectangles represent the hide spots and grey rectangles represents obstacles. In this case these obstacles are actually the target cell and its 3x3 neighbor that each drone has decided, and we are using this to ensure that each drone will not select the same cell and cells close to it.

DISTRIBUTION A: Distribution approved for public release.

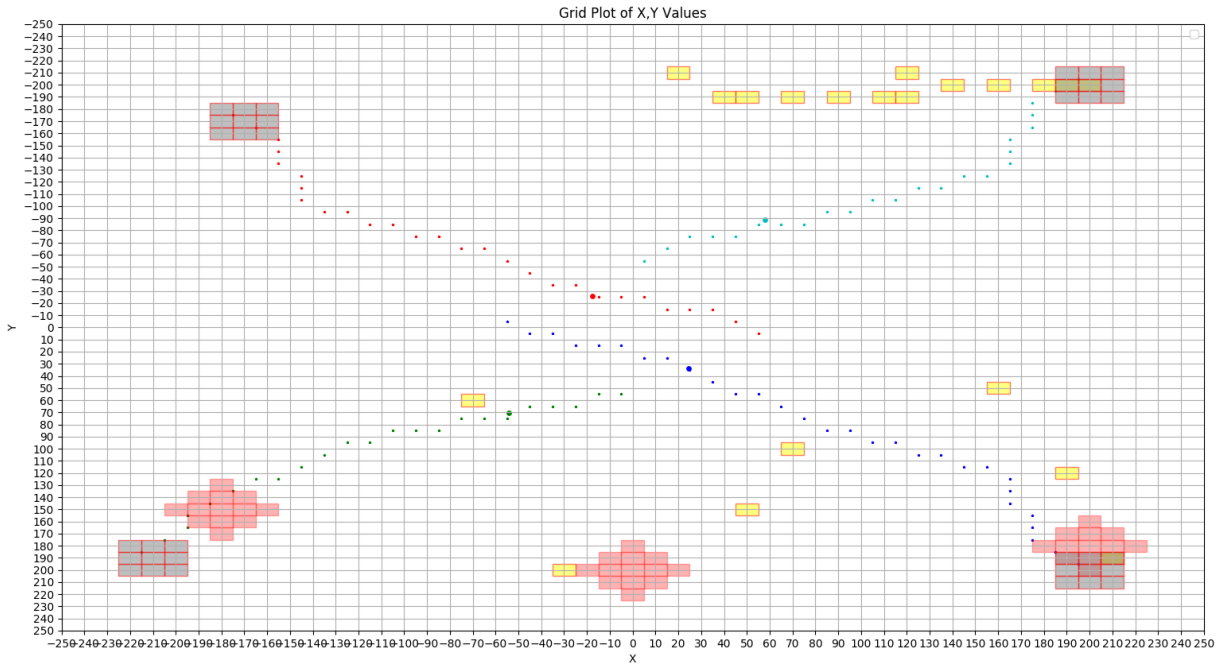


Figure 3: The cells of the world in a 2d plt image.

UAVs

Iris drone

For the purposes of the simulation, which takes place inside the Gazebo simulator, 4 Iris drones are used. Each Iris drone is controlled using velocity and yaw commands through mavros and the PX4 autopilot. Each drone is equipped with a single camera with fixed pitch. The camera has a refresh rate of 60hz and a resolution of 640x360 pixels. Moreover, the UAVs are maintaining a fixed altitude during flight. An Iris drone can be seen in the figure below:

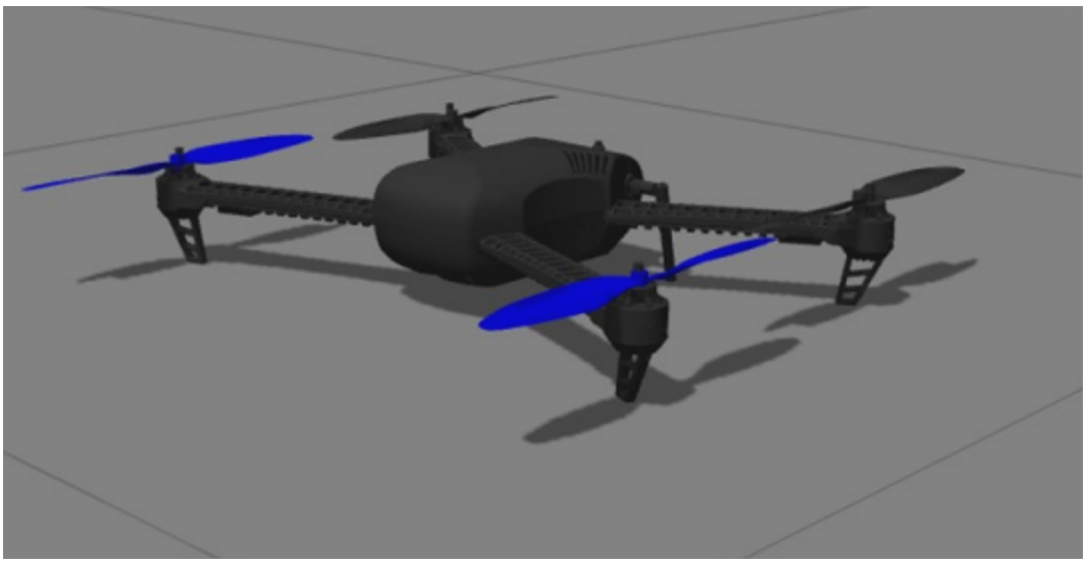


Figure 4: The Iris drone

DISTRIBUTION A: Distribution approved for public release.

Actor Strategies

The aim of the actors is to reach the center of the world. Thus, the most distance-efficient strategy would be to move as individuals from their point of entry towards the center of the world following a straight line. In this work, it is assumed that the actors are capable of observing a nearby drone and may apply more elaborate strategies and cooperative behaviour with other actors, attempting to maximise the chances of at least some actors reaching the security assets in the center of the the world. The actors exploit two techniques to achieve their goal: (1) formation, (2) hiding and (3) fleeing.

The formation strategy of actors is built upon cooperative behaviour among actors, who aim to increase the chances of any actors reaching the center of the world, even if that may cause their own chances to reduce. Actors create a formation consisting of two columns and adjustable number of rows, depending on the number of actors in the formation. They adjust and synchronise their velocities. The formation enters the area of interest and move towards the center of the world following a straight line. When the actors of the formation observe a drone in a close distance, they assume that the drone may detect them, and the formation is split into two formations. The two formations start moving in different directions, drawing a vertical angle between their directions. After a predefined distance the formation start moving again towards the center of the world. The process is repeated each time a drone is observed by the actors. If another formation (or an individual actor) is in the observation range of the formation, the two formations merge.



Figure 5: The formation of the actors

DISTRIBUTION A: Distribution approved for public release.

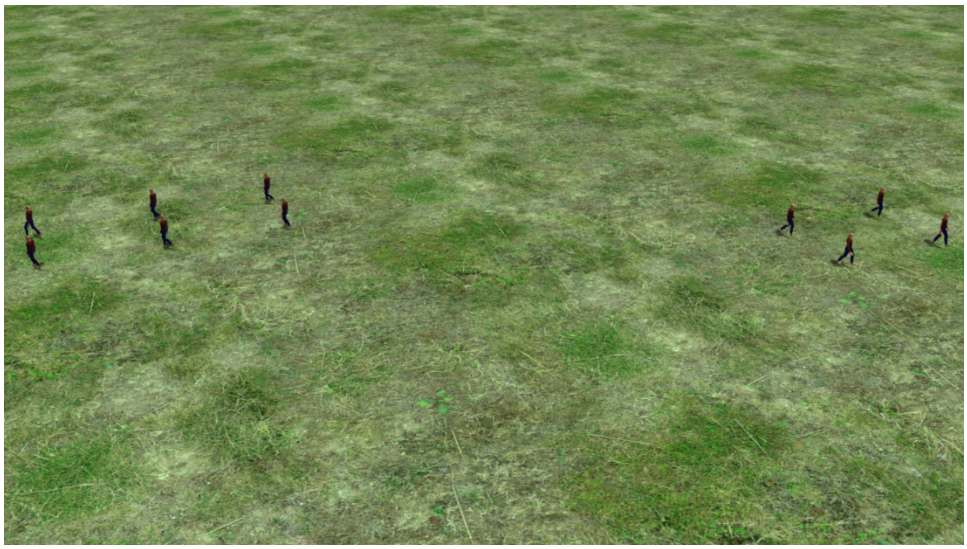


Figure 6: Formation splitted into two

In the hiding strategy, actors act as individuals and do not cooperate with other actors. In this strategy, actors take advantage of the existence of hide spots and use them to avoid being caught.

1. Each actor enters the area of interest and generates a path to reach the center of the world. This path is generated by taking into account hide spots to be visited, while minimising the travelled distance outside of hide spots. In this strategy it assumed that actors are aware of the locations and sizes of all hide spots.
2. Each actor enters the area of interest and generates a path to reach the center of the world. It is assumed that the actor does not have global knowledge of the locations and sizes of the hide spots and discovers hide-spots, as they travel. When a new hide-spot is discovered, the actor may use a path planning algorithm to update their path and minimise the travelled distance outside of hide spots.
3. Each actor enters the area of interest and starts moving towards the center of the world using a generated path. If a drone is nearby, the actor checks if there are any hide spots in close distance and travels and then stays in the hide spot until the drone has left.

The fleeing strategy also involves individual actor decision making. Actors during the execution of the path they have generated, are able to deviate from it if a drone is nearby by trying to move away from it. Fleeing takes place throughout the mission and acts as additional logic when the formation or hiding strategy is not active. When the drone is not in close proximity, the actor will continue the path he generated initially.

System design and algorithms

Multiple Object Detection, Tracking and Localisation System

This work describes an optical camera-based target detection, tracking, and localization solution for Unmanned Aerial Vehicles (UAVs). In this work we examined the well-known network YOLO7-tiny as an object detector whilst in our previous deliverable we compared YOLOv4, YOLOv4-tiny with a custom YOLOv4 implementation.

The target tracking algorithm we use is based on Deep SORT, providing cutting-edge tracking. The proposed localization approach can accurately determine the position of ground targets identified by the custom object detection model as presented in our previous deliverable.

Yolo7

In this section, we explain the YOLO architectures we use in our Multiple Object Detection system. In particular, we deal with the state-of-the-art architecture YOLOv7-Tiny (Wang, Bochkovskiy, & Liao, 2022)

YOLOv7-Tiny which is based on the newest YOLOv7 (Wang, Bochkovskiy, & Liao, 2022) algorithm has been implemented by the same developers and surpasses all previous object detection models and YOLO versions in both speed and accuracy as claimed by the authors. By making optimization changes to the backbone, they opt for layer aggregation using E-ELAN, an extended version of the ELAN computational block. Moreover, YOLOv7 employs re-parameterization planning techniques on the backbone to improve the ability of the model to generalize. To identify which modules should use such a strategy, gradient flow propagation paths are used. Lastly, when it comes to the head of the model, an auxiliary head using features roughly from the middle of the backbone is supervised during the training process and fine-tuning is performed by experimenting with different magnitudes of supervision on it. These, along with other changes, have produced state-of-the-art detection results along with great performance scaling as the model input increases. It requires cheaper hardware than other well-known architectures and requires less time in training since it can be trained on small datasets without any pre-trained weights. The major changes this architecture introduces regarding previous versions are the extended efficient layer aggregation network, model scaling for concatenation-based models, planned re-parameterized convolution and coarse for auxiliary and fine for lead loss.

We develop datasets by flying drones and recording camera frames throughout their flights, as well as having individuals perform random walks in the area, and then annotating each image with the necessary information (label, x-min, y-min, x-max, y-max) for each object in the image. For the training dataset, 14 instances with varying views and altitudes were gathered.

DISTRIBUTION A: Distribution approved for public release.

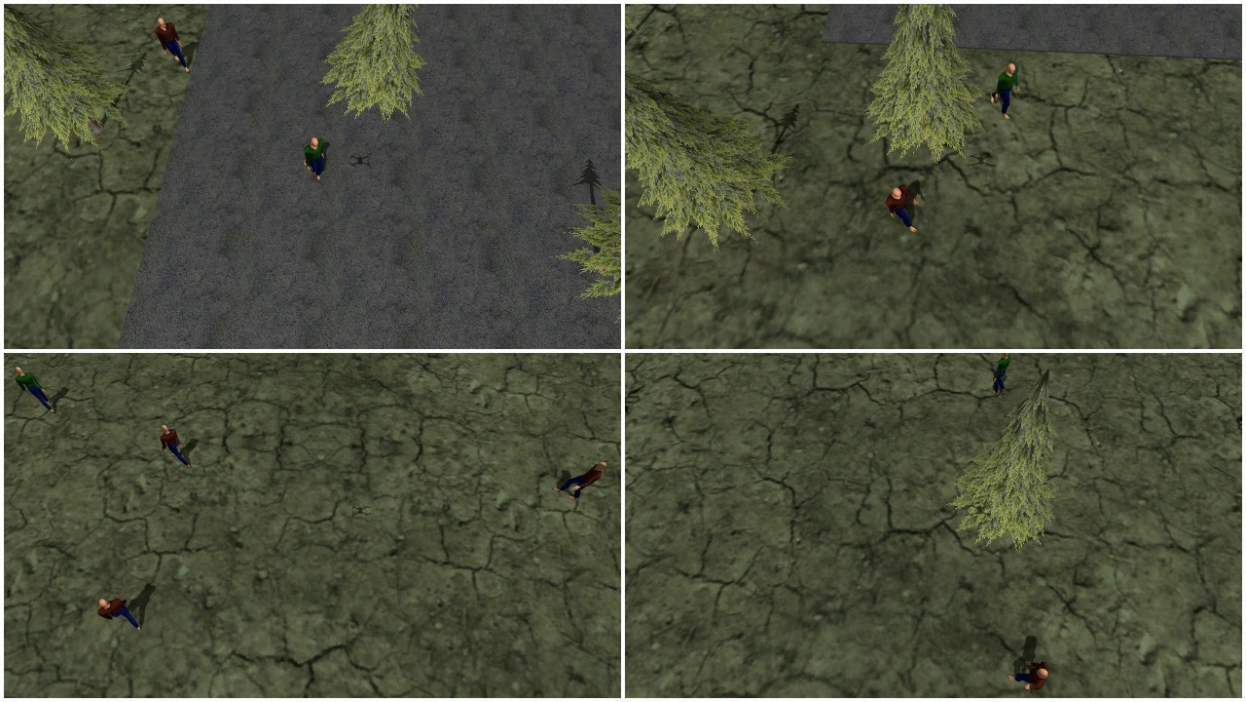


Figure 7: Train Dataset

Finally, 872 images with a completely different background were collected for the validation dataset so that we could assess how well the object detector generalizes. Except for the images made via mosaic augmentation, all images are 1280 pixels wide by 720 pixels in height. Our train dataset has 3160 photos, whereas our test dataset contains 872 images. In Figure 8 images from the test dataset are presented.

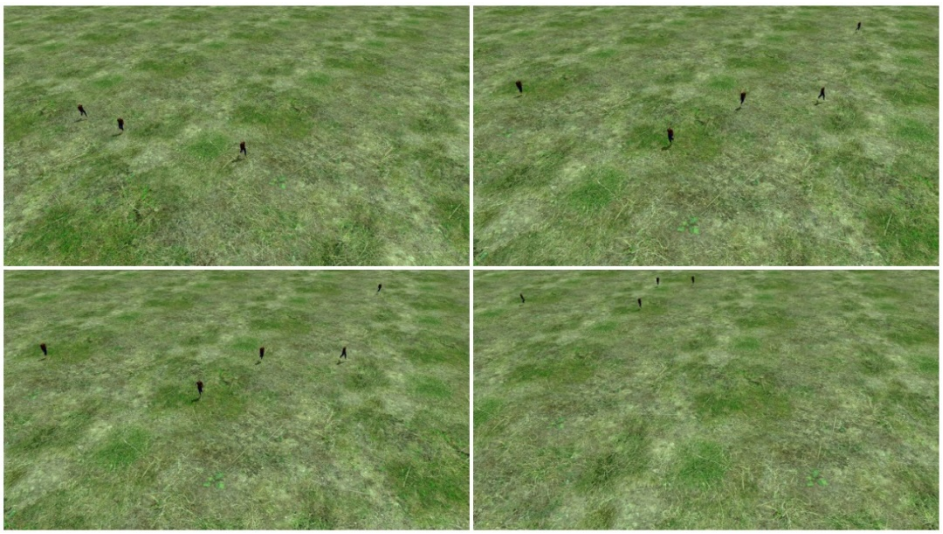


Figure 8: Test Dataset

DISTRIBUTION A: Distribution approved for public release.

Tracking

Another benefit of using these schemas is that their outputs (i.e., bounding boxes) can be used for tracking objects and deriving speed information relative to the drone's position. For this task, there is a very well-known algorithm called Deep SORT (Wokje, Bewley, & Paulus, 2017). One of the most popular object-tracking frameworks is Deep SORT, which improves tracking and reduces the number of identity changes by integrating an appearance feature vector for the tracks, which, in this example, is produced by a pretrained CNN using the bounding boxes detected by YOLO. There is a demand to develop new ways to keep track of them and accurately identify them because basic detection models are extremely likely to fail at identifying many objects sequentially as the frames pass. Deep SORT is utilized here to provide a good MOT (multiple object tracking) (Bernardin & Stiefelhagen, 2008)foundation.

The Kalman filter is an important part of Deep SORT. Because we use a basic linear velocity model, the state simply stores the absolute position and velocity components. The Kalman filter helps us account for noise in detection and predicts a suitable match for future bounding boxes based on past states. The next difficulty is linking new detections that are coming from YOLO with the predictions that have been made from the Kalman filter. Because they are handled separately, a technique to link track i to incoming detection k is required. Deep SORT provides two things to address this; (1) an efficient method to associate the data and (2) a distance metric to quantify the relationship. To include the uncertainty from the Kalman filter, the authors choose to utilize the squared Mahalanobis distance (an effective measure for dealing with distributions). This threshold can provide us with a good estimation of the real linkages. Since we are measuring the distance between two distributions, this metric is more accurate than, for example, the Euclidean distance. The Hungarian method is utilized for the data association component. Finally, the feature vector serves as the object's ``appearance descriptor''.

It is possible to lose track during tracking owing to obstruction, sudden changes in measured velocities, or poor detector performance. We have modified SORT settings to remedy this situation. The parameters we modified are the maximum age of a track has been raised from thirty frames to sixty frames, while the minimum number of frames necessary for a track to be considered active has been raised from three to seven. Finally, while tracking, we save the image coordinates of the bounding box, and if it is lost, we use the closest() method, which will try to identify and follow the track that is closest to the lost one and no more than 10 percent off on each axis. If the original track and a track similar to it are not found for the maximum-age frames, we consider the track lost. If closest() located a track and did not miss it for max-age frames, we switch to that track at the conclusion of max-age frames and consider it an identity switch. This approach works effectively in areas with a moderate invader density.

Global Tracker

In this section, the global tracker system is presented. The global tracker is an entity capable of localizing, tracking, and distinguishing objects detected from different sources. As presented in Figure 9, in our system we have multiple agents, detecting and localizing objects as described above. Each agent passes its object localization outcome to the global tracker, to produce the desired output regarding information about the objects in the area. The tracker processes this information using a custom algorithm to distinguish the unique observations produced by the cameras of the four agents. The output of the localization system of each agent is fed to the global tracker system. The observations are

processed and then passed to the well-known object tracking algorithm SORT, but instead of giving the camera's pixel coordinates of a single agent to the SORT algorithm we provided the processed observations of the agents in Cartesian coordinates and fine-tune the SORT algorithm regarding specific parameters like the max-age of detection that can be used in memory and the IOU threshold.

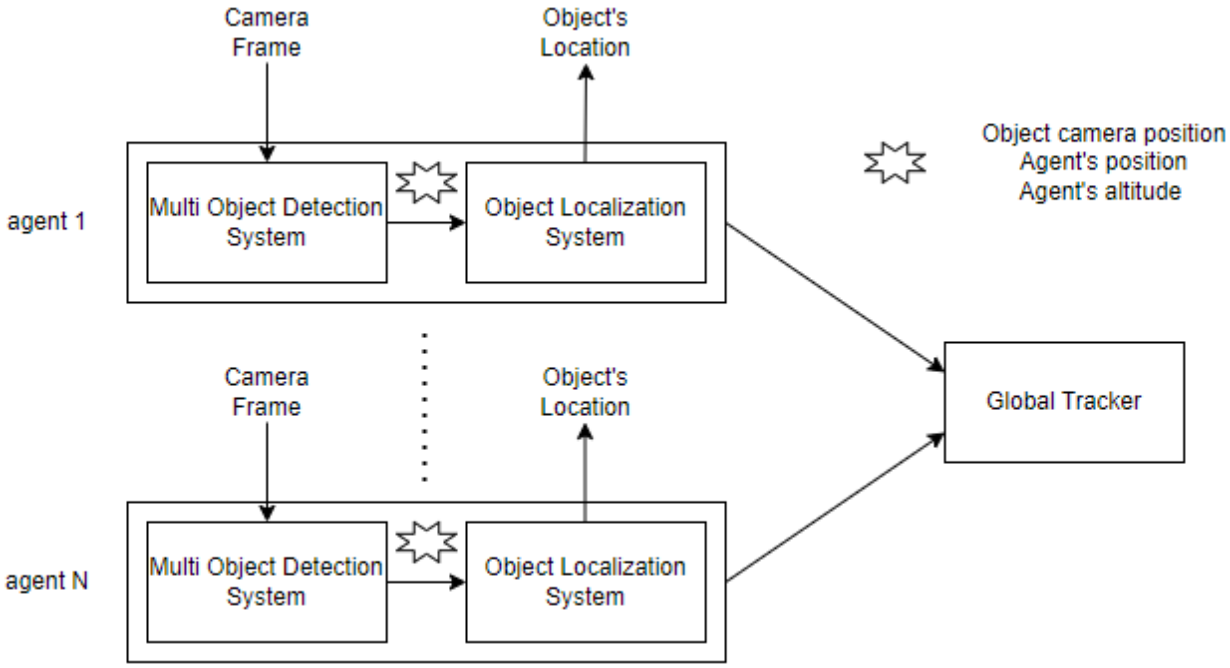


Figure 9: Global Tracker System

After having performed the pre-processing step seen Figure 10, the detected objects are then associated with tracks based on the location and size of the objects in the current and previous frames. The algorithm uses a Kalman filter to predict the location of each track in the next frame and then uses a Hungarian algorithm to associate the predicted tracks with the detected objects in the next frame. This process is repeated for each timestep during execution, allowing the algorithm to track multiple objects as they move over time.

DISTRIBUTION A: Distribution approved for public release.

Algorithm 1 Detections Selection Algorithm

```
for Detections in a particular timestep do
    while len(Detections) > 0 do
        Calculate distance matrix for all detections
        Find the pair of detections with the minimum distance
        Using that pair, create a cluster with the nearby detections having a distance
<2m
        Find the camera with the most detections in that cluster
        if Another camera has the same number of detections then
            Find the closest camera of that detection
            Keep the detections from the closest camera and remove the others in that
cluster
        else
            Keep those detections and remove the others in that cluster
        end if
    end while
end for
```

Figure 10: Detections Selection Algorithm

The SORT algorithm has several advantages over other visual tracking algorithms in our use case. It is relatively simple to implement and can run in real-time even with a high number of detections, making it suitable for use in applications with strict performance requirements. It also uses a combination of detection and data association, which allows it to handle occlusions, false detections, and other common challenges in visual tracking.

Decision Making

The decision-making Capabilities of the system is based on the individual decision of each member of our swarm. Hence the decision making process is conducted in a decentralized manner based on perception information shared among the members of the swarm regarding the detections of actors and their estimated locations and the location of the drone in each timestep.

A star

The A* algorithm is a popular and widely used pathfinding algorithm in computer science and artificial intelligence. It is used to find the shortest path between two points on a graph or grid while considering the cost associated with moving from one point to another.

The algorithm starts with an open set containing the starting node and a closed set initially empty.

While the open set is not empty:

- Select the node with the lowest f-score (f = g + h),
    - where: g is the cost of the path from the start node to the current node.
    - h is the heuristic estimate of the cost from the current node to the goal node.
- If the selected node is the goal node, the path has been found.
- Move the selected node from the open set to the closed set.

DISTRIBUTION A: Distribution approved for public release.

- For each neighbor of the current node:
    - If the neighbor is in the closed set, skip it.
    - If the neighbor is not in the open set, calculate its f-score and add it to the open set.
    - If the neighbor is already in the open set, compare the new f-score to the old one. If the new f-score is lower, update the f-score and change the parent to the current node.
- If the open set becomes empty, and the goal node has not been reached, no path exists between the start and goal nodes.
- Once the goal node is reached, the algorithm can backtrack from the goal node using the parent pointers to reconstruct the shortest path.

A* uses a heuristic function to estimate the cost from the current node to the goal node. The heuristic is problem-specific and should be admissible (never overestimating the true cost) to ensure the optimality of the path found by A*.

The heuristic function we used for path planning is:

$$calculatedHeuristic = ManhattanDistance - \left(1 + 0.3 * \frac{objectDetectedValue}{5000}\right) *$$

$$(2 * (1 + improtanceValue) * timeOfLastVisitFactor)$$

Where:

- ManhattanDistance is the Manhattan Distance of the current location to Goal location
- objectDetectedValue = A boolean value representing object detection information for the specific node
- importanceValue = Value representing importance factor, whether the node is located inside a hide spot or not
- timeOfLastVisitValue = Value representing time of last visit normalized by 50
- timeOfLastVisitFactor = $1 - e^{-timeofLastVisitValue}$

This equation calculates the heuristic value used in the A* algorithm for pathfinding. The heuristic combines the Manhattan distance between the current position and the goal with factors related to object detection information, the importance, and the recency of visit. The goal is to find the optimal path by minimizing this heuristic value during the A* search and make the drones navigate to their destination by selecting the nodes with the best score, resulting in visiting the most valuables areas instead of applying a straight-line movement to reach their destination.

To ensure safe navigation and that our drones don't collide with each other, we applied the cells that are in the 3x3 neighbor of the target location and of the current location to be considered as obstacles to the other drones of the swarm.

We have added a replanning condition on which if the condition is satisfied the drone will replan it path with the same target and is about whether on our current neighbor there is a presence of an obstacle.

Next Mission

Each drone to decide its next task will perform the following calculations:

- take a random number in range of [0,1]
- if the random number is bigger than 0.2
    - then find your next target
    - else select a hide spot and go patrol it

In this way we can ensure that our drones are going to patrol in a sequence all the hide spots of the world and be able to cover the difficult areas, and in other cases they select their next target based on the next target selection algorithm.

Next Target

Each drone to decide its next target location will perform the following calculations:

- Iterate through all the cells of the world
- If the cell is an obstacle
    - assign a value of -100
- else
    - calculate the Manhattan distance (manhattanDist) between the current cell's center and the drone's position.
    - normalize this distance relative to a maximum distance (maxDist) to make it in range of [0,1].

DISTRIBUTION A: Distribution approved for public release.

- compute a variable result value based on a random condition involving randomValue.
    - If randomValue is less than 0.5, then result is calculated as 2 * (1 - normalizedDist).
    - Otherwise, if randomValue is greater than or equal to 0.5,
    - then result is calculated as 2 * normalizedDist.

Then heuristic we used for selecting the next target is

$$calculatedHeuristic = 0.5 * result + ManhattanDistance - \left(1 + 0.3 * \frac{objectDetectedValue}{5000}\right) * (2 * (1 + improtanceValue) * timeOfLastVisitFactor)$$

Where:

- ManhattanDistance is the Manhattan Distance of the current location to Goal location
- objectDetectedValue = A boolean value representing object detection information for the specific node
- importanceValue = Value representing importance factor, whether the node is located inside a hide spot or not
- timeOfLastVisitValue = Value representing time of last visit normalized by 50
- timeOfLastVisitFactor = $1 - e^{-timeofLastVisitValue}$
-

Patrol Hide Spot

When a next mission decision is made, there is a 20% chance that a drone will decide to Patrol a nearby hide spot. Initially the drone is tasked with finding the closest hide spot that is available. This is done by calculating the Euclidean distance between the position of the drone and the center of all of the available hide spots.

The simulated world is occupied by five hide spots all of which the locations are known beforehand. Hide spot Patrolling consists of three major algorithms.

- Convex Hull generation based on the location of the highly dense area consisting of trees, thus generating the hide spot.
- Trajectory generation for the patrol based on camera FOV with waypoint interpolation to smooth out yaw and velocity changes.
- Guidance and waypoint execution logic.

In Figure 11, three examples of generating a trajectory based on camera FOV by offsetting the trajectory outwards of the Convex Hull given by the hide spot.

1. The yellow arrow represents the drone location and yaw.
2. The black dots represent the trees that the hide spot consists of.
3. The purple stars represent the generated trajectory as well as the convex hull center
4. The Cyan polygon represents the generated Convex Hull
5. The filled trapezoid in light blue represents the drone FOV from its camera

We offset the waypoints outwards following the same line generated by the Convex Hull center and the corresponding vertex with the goal of maintaining roughly 20% of the vertical FOV looking at the outer area closest to the drone. Moreover, while the drone is performing its trajectory through two vertices, we apply a yaw offset of 25 degrees in the direction of its heading. We do so to have better area coverage and avoid missing any actors that will exit the hiding spot during the patrol.

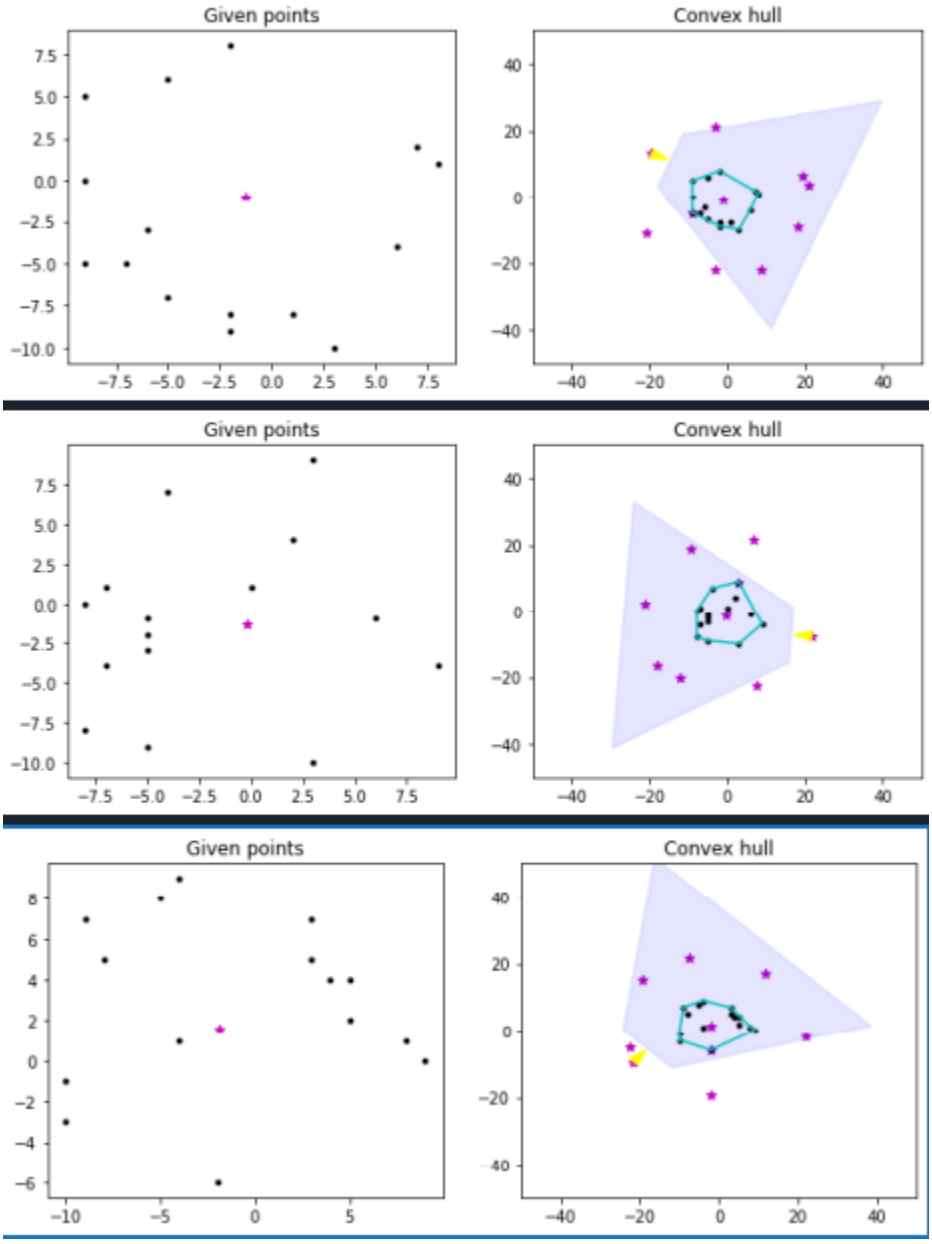


Figure 11. Convex Hull and trajectory generation for a hiding spot

Once the trajectory has been created, we perform interpolation of the given points to smooth out both the yaw and the velocity commands issued by the drone to better aid actor detection. After testing multiple methods as seen in Figure 12, we found that cubic interpolation provided the best results for our scenarios. Lastly, the augmented waypoint list is then executed by the guidance and waypoint

execution algorithms of the drone, where the drone first finds the waypoint closest to it and then starts from there.

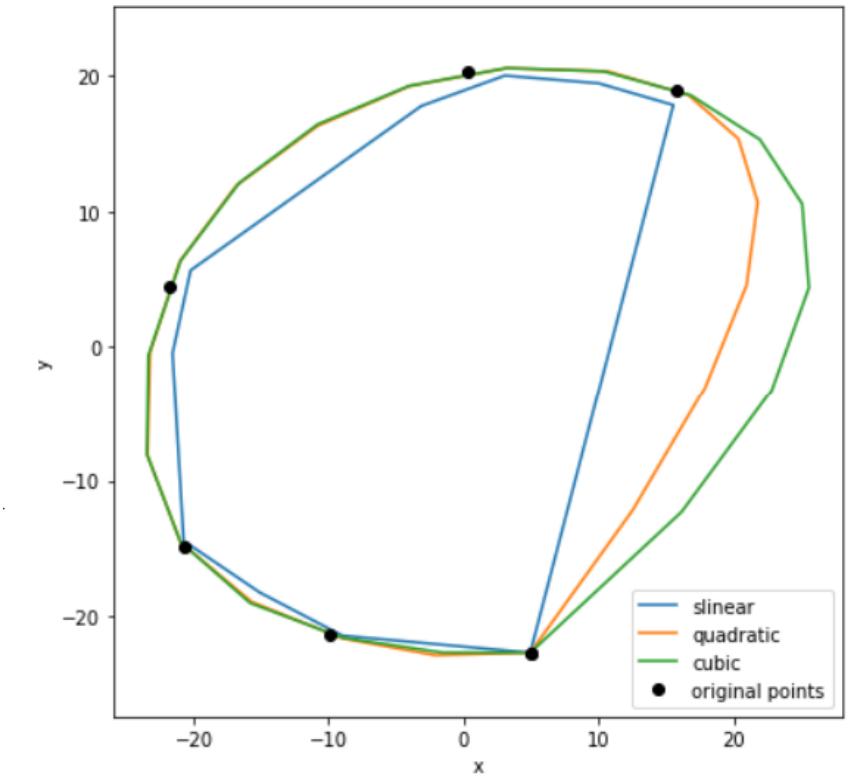


Figure 12. Interpolation methods for the generated trajectory

Overall Architecture

In this section, we explain the overall architecture of the system. The system is created using ROS, and for the needs of our development, we created several custom topics and messages, to ensure that the necessary data are exchanged to the drones to achieve their purpose. The system involves the following subsystems:

- DroneController
  - Is responsible for handling all drone operations regarding guidance, navigation and path planning
- GroundController
  - Is the central agent, receiving data from all the drones of the swarm, updating the world info, and logging data to produce the evaluation metrics
- PerceptionController
  - Is responsible for the detection and tracking of the objects, the detected objects are then localized and then the global tracker is activated to provide the fused info about the total number and the locations of the actors in the world.

These three subsystems communicate by exchanging messages through topics.

DISTRIBUTION A: Distribution approved for public release.

The topics we use in our system are:

- mavros/local_position/pose
    - When we subscribe to the "mavros/local_position/pose" topic, you are receiving information about the current local position and orientation of the drone
- mavros/global_position/pose
    - When we subscribe to the "mavros/global_position/pose" topic, you are receiving data that tells you where the drone or robot is located on the Earth's surface and how it is oriented in the global coordinate frame.
- mavros/setpoint_raw/local
    - When we publish messages to the "mavros/setpoint_raw/local" topic, you are sending control commands to the drone or robot in its local coordinate frame. These commands can include desired roll, pitch, yaw, and thrust values,
- mavros/setpoint_position/local
    - When we publish messages to the "mavros/setpoint_position/local" topic, you are sending commands to the drone or robot to move to specific positions or waypoints in its local coordinate frame. These setpoints can include desired x, y, z positions (often in meters) relative to the vehicle's current location
- camera3/iris_red3/image_raw
    - When we subscribe to the "camera3/iris_red3/image_raw" topic, you are receiving the unprocessed image data from the specified camera.
- PathData
    - When we publish to the PathData topic we are sending information about the id of the drone, the path generated from our path planning module, and information whether the drone decided to patrol a hide spot
- FirstTarget
    - When we publish to the FirstTarget topic we are sending information about the id of the drone, and the first waypoint assigned as a target for that particular drone
- ActorPos
    - When we subscribe to the ActorPos topic we are receiving information about the id of the actor, and its location. Only the detected actor’s information contained in that topic
- World
    - When we subscribe to the Word topic we are receiving information about the cells of the world as described in world representation chapter.

In Figure 13 we present an image containing the nodes of our system and their communication. We can see that the PerceptionControllerNode is receiving the images from the four drones from the camera(X)/iris_red(X)/image_raw topic, is performing object detection and tracking, provide the detections of the drones to the global Tracker and its output provides the information about the detected actors to ActorPos topic, where the GroundControllerNode can receive it. The GroundControllerNode, when the mission starts provides to the drones their first target locations using the FirstTarget topic and then it sequentially receives PathData from the drones and ActorPos topics to

DISTRIBUTION A: Distribution approved for public release.

update the world and provide the fused information to the World topic, so each drone will use it to perform Path Planning.

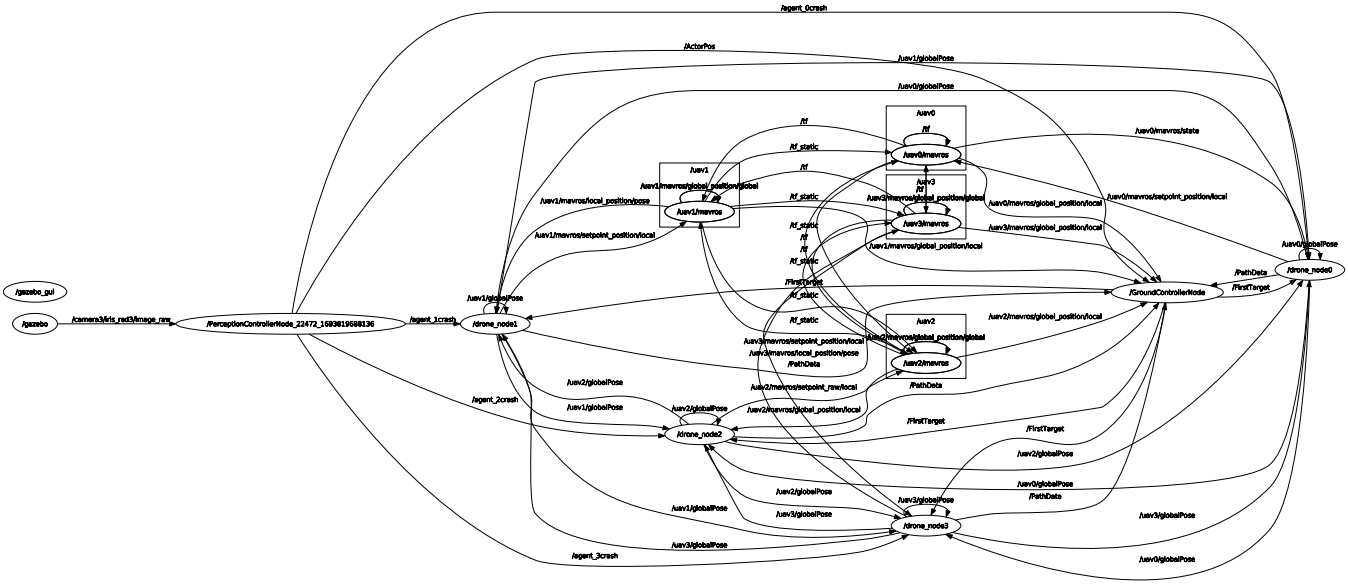


Figure 13. A graph showing the nodes in our system and their communication

Experiment set-up

Simulation Parameters

Number of Drones	Number of Actors	World Size	Grid Size	Hide Spots
4	20	500x500	10x10	5

Results

Object Detection

Our object detector, YOLOv7-Tiny have been trained with our custom train dataset and tested on the same test dataset mentioned in the previous section to evaluate their performance. For the object detection subsection, we provide the following metrics:

- Total positives

- True positives
- False positives
- False negatives
- Recall
- Precision
- F1-Score
- Average precision

Total positives	True positives	False positives	False negatives	Recall	Precision	F1-Score	Average precision
4186	4108	11	0.981	0.997	0.989	0.989	98.04%

We can observe that YOLOv7-Tiny outperforms all other models that we tested previously with an average precision of 98.04%.

To help in the evaluation of the four different networks we produced the ROC curves for our network. In Figure 14 for the YOLOv7-Tiny, another important metric to evaluate the four networks is the capability of running in real-time, hence the inference time for every model was measured and YOLOv7-Tiny was measured to have 10.9 ms inference.

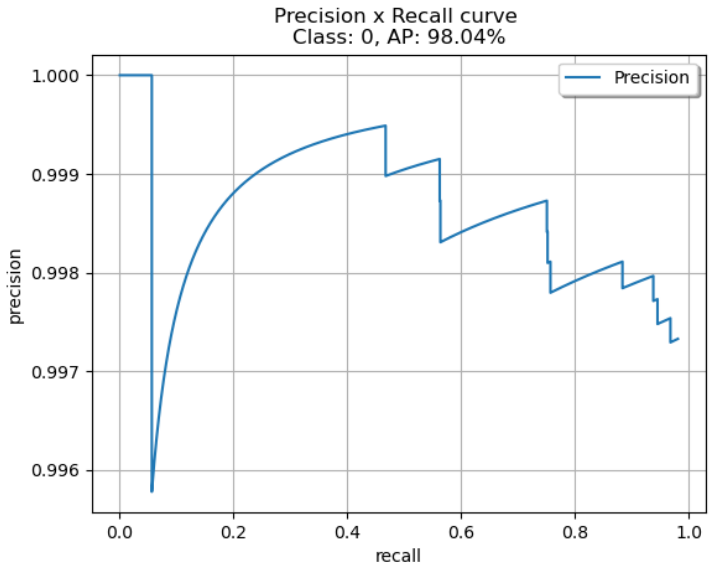


Figure 14: YOLOv7-tiny Roc Curve

Object Tracking

To assess tracking performance, we also issue MOTA, MOTP, and MOTAL scores along with the number of identity switches during tracking as seen in [20]. The MOTP measure is defined as the total position error for matched object hypothesis pairings over all frames, multiplied by the total number of matches made. It demonstrates the tracker's capacity to predict accurate item locations, regardless of its ability to recognize object combinations, maintain consistent trajectories, and so on. Out of all frames, MOTA

DISTRIBUTION A: Distribution approved for public release.

accounts for all object configuration mistakes produced by the tracker, including false positives, misses and mismatches. It provides an intuitive assessment of the tracker's ability to maintain precise trajectories, regardless of its accuracy in calculating object locations.

IDs	MOTA	MOTP	MOTAL
6	95.7	94.7	95.9

To validate our best model YOLOv7-Tiny, we decided to test it in real-world videos, the "MOT20-01" scene and the rural road scene. To do so, we trained the model using the well-known VisDrone [27], a large-scale drone-based dataset containing 8.599 photos (6.471 for training, 548 for validation, and 1.580 for testing) capable of multiple object detection for the following 10 object categories (pedestrian, person, vehicle, van, bus, truck, motor, bicycle, awning-tricycle, and tricycle). The "MOT20-01" scene from the MOT20 [28] dataset is used, which is another commonly known dataset for object tracking algorithms. It is an indoor scene in a packed railway station. It's a very challenging scenario, with 25 frames per second and 1080p resolution. The duration is 17 s and 90 people in total in this video. The "rural road" is a scene captured by a drone on a public road with real-world traffic. We used the DJI Phantom 3 drone to record a 15 min video of public road traffic at 1080 p and 25 fps. We removed the areas of the video where there was no traffic, and we reduced it to one minute and six seconds. The video contains objects of automobiles, vans, and pickup trucks. In Figure 15 and Figure 16 the two test scenes for real-world evaluation are depicted while in Table 4 the metrics for these two scenarios are presented.



Figure 15: Rural Road Video



Figure 16: Mot 20 Video

Metric	MOT20-01	Rural Road
IDs	601	8
MOTA	45.6	71.0
MOTP	51.1	71.3
MOTAL	48.6	71.5

In both scenarios, we used the same parameters as in the simulated ones. Regarding the MOT20-01 scenario, we observe a lot of identity switches and it is caused by our decision to keep the IOU threshold the same for all the tests we performed. The high detection density of this scene causes a lot of occlusions and overlaps between the detected bounding boxes. Regarding the rural road scene, the combination of YOLO7-Tiny and DeepSort performed really well, having a small number of identity switches and a score that indicates near-perfect tracking.

Global Tracker

In this section, the results of the Global Tracker are presented. The system is tested in several scenarios varying in the movement of the objects, the camera pitch degrees, and the movement of the cameras. The cameras in these scenarios are located in a rectangular shape and the actors are moving within this area. All scenarios had a pitched camera of 57~degrees except the ones that mentions a different value. The scenarios are the following:

- Cross path with no localization error: This scenario included six actors starting at positions around the centre of the world and all moving towards the centre, causing their paths to cross. After reaching the centre of the world the actors move back to their original positions. For this

scenario, no localisation error was considered and the global tracker was fed the ground truth concerning the actor's positions.

- Straight line: In this scenario, the six actors were placed in a formation and performed a straight-line path.
- Random movement: This scenario included six actors performing a random path, crossing paths with another actor in several cases.
- Random movement with a pitched camera: In this scenario, the actor's behaviour is shared with the one in the ``random movement'' scenario, but the UAV's camera is pitched at 25 degrees.
- Cross path: In this scenario, the actors' behaviour is the same as in the previous scenario, but in this case, localisation error is considered.
- Static objects moving drones: In this scenario the actors are static and the cameras are moving aiming to detect the total number of actors in the area.

In Figure 17, we see six different instances of global tracker presenting the locations of the detected objects and their unique id in the cross-path scenario.

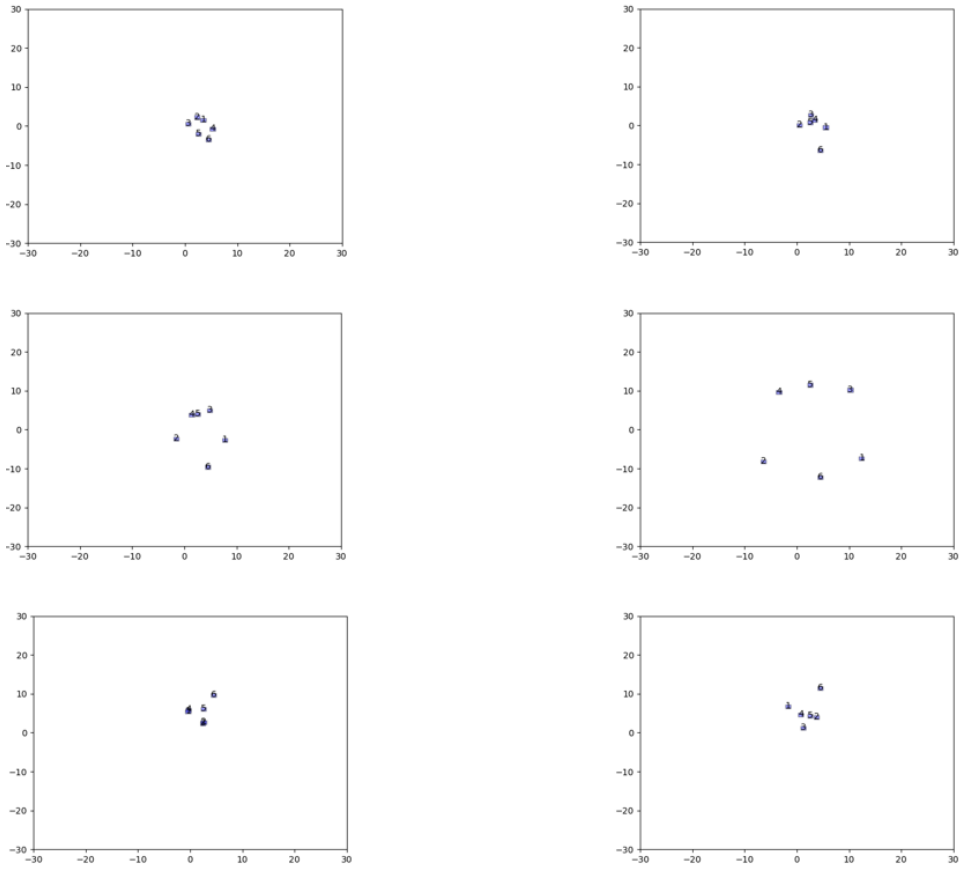


Figure 17: Instances from different timesteps.

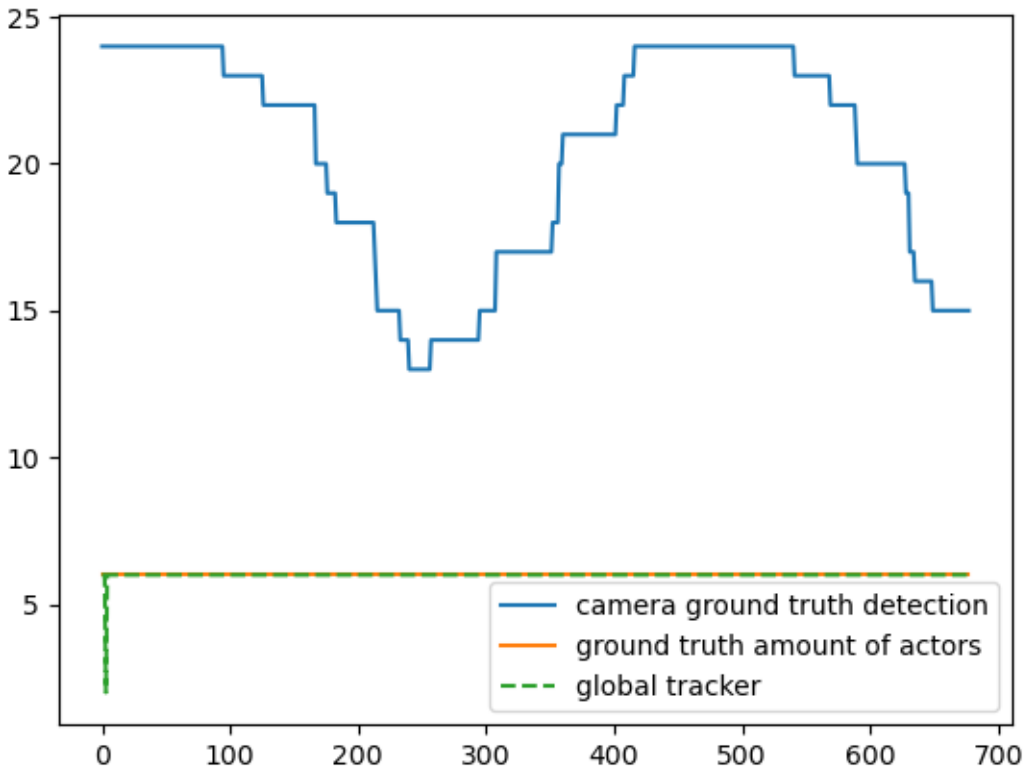
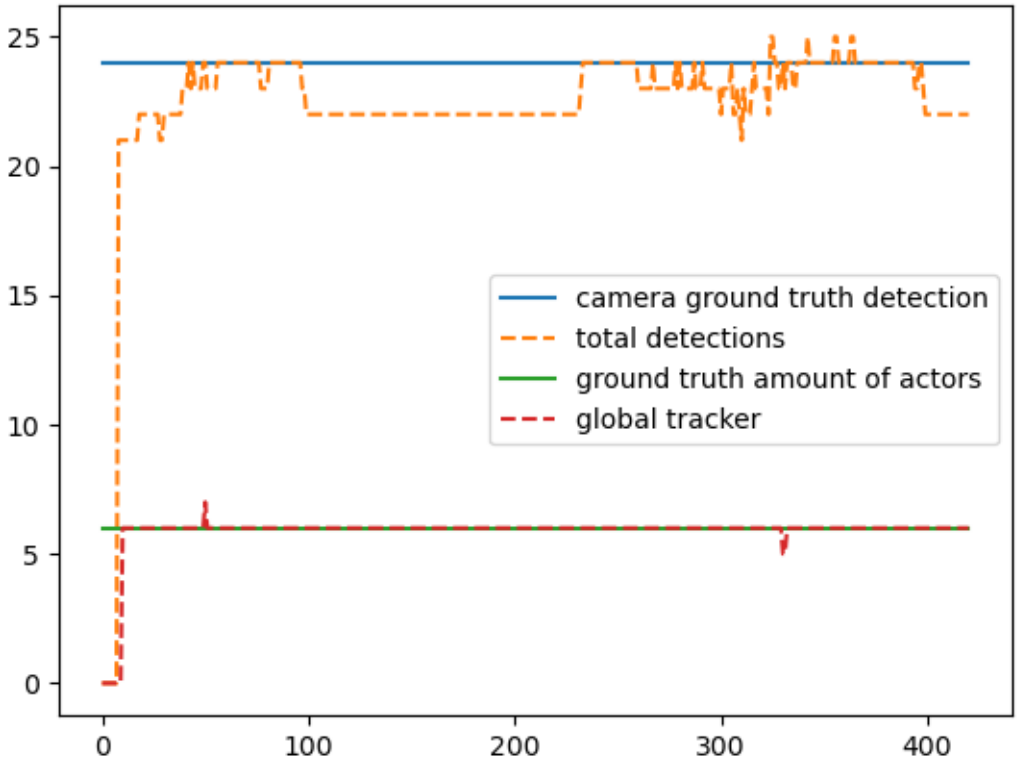


Figure 18: Graph showing the global tracker results in comparison to the ground truth for the Cross Path with no localization error scenario. The vertical axis represents the number of actors, while the horizontal axis the frames.



DISTRIBUTION A: Distribution approved for public release.

Figure 19: Graph showing the global tracker results in comparison to the ground truth for the Straight Line scenario. The vertical axis represents the number of actors, while the horizontal axis the frames.

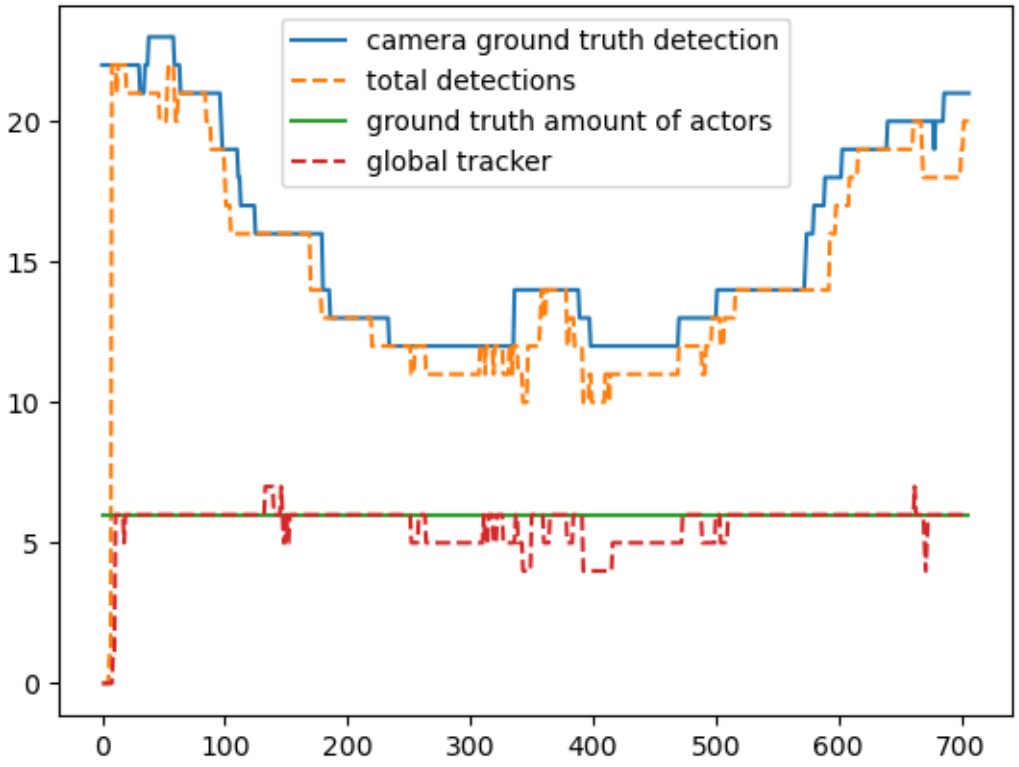


Figure 20: Graph showing the global tracker results in comparison to the ground truth for the random movement scenario. The vertical axis represents the number of actors, while the horizontal axis the frames.

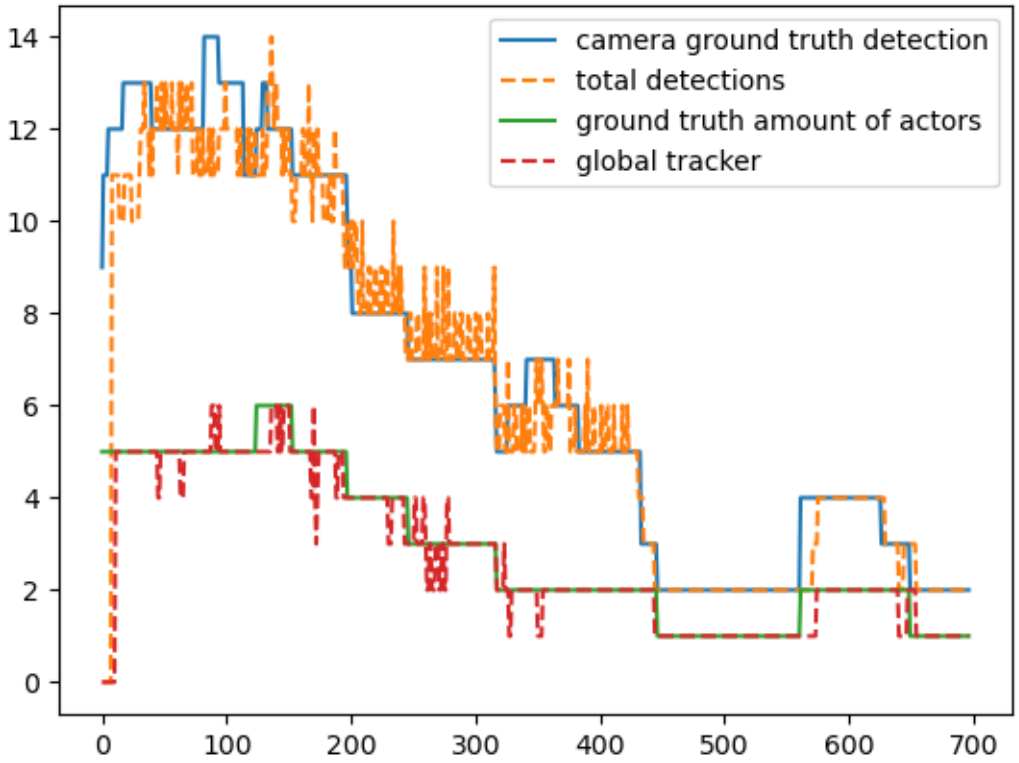


Figure 21: Graph showing the global tracker results in comparison to the ground truth for the random movement with pitched camera scenario. The vertical axis represents the number of actors, while the horizontal axis the frames.

In Figure 18, we see the results of the cross-path scenario. In this scenario, the actors move towards the centre of the rectangular area and then back again to their original positions with measurements with no localization error, having crossed paths on several occasions with 3 and more actors participating in the occlusion, hence some cameras are seeing fewer actors than the actual number present in the field of view. When no localisation error exists, the proposed global tracker performs nominally with its results coinciding with the global truth.

It is clear from Figure 17 and Figure 18 that the global tracker system is performing perfectly when the input contains perfect observations regarding the localization error despite having cases where all cameras detect all actors.

In Figure 19 we see the results of the Straight Line scenario. In this scenario, the actors were placed in a straight-line formation and moved in the rectangular area defined by the cameras. The results indicate that in this scenario the global tracker performed exceptionally having a 96.9% of correct calculations in all the frames of this scenario.

In Figure 20 we see the results of the Random scenario. In this scenario, the actors were making random movements in the rectangular area, having crossed paths on several occasions. The results indicate that in this scenario the global tracker performed exceptionally having 71% of correct calculations in all the frames of this scenario. We can observe that the error pattern of the global tracker matches the error pattern of the total detections. This means that the total detections were underestimating the total number of actors and hence the global tracker was getting an underestimated input and created a similar pattern to its output.

In Figure 21, we see the results of the Random Pitch scenario. In this scenario, the actors were making random movements in the rectangular area, having crossed paths on several occasions but the camera

pitch, in this case, was 25 degrees. The results indicate that in that scenario the global tracker performed exceptionally having 84.6% of correct calculations in all the frames of this scenario.

In Figure 22, we see the results of the cross-path scenario. In this scenario, the actors move towards the centre of the rectangular area and then back again to their original positions, having crossed paths on several occasions with three and more actors participating in the occlusion. The results indicate that in that scenario the global tracker performed really well, in a scenario where there were too many occlusions having 82.7% of correct calculations in all the frames of this scenario.

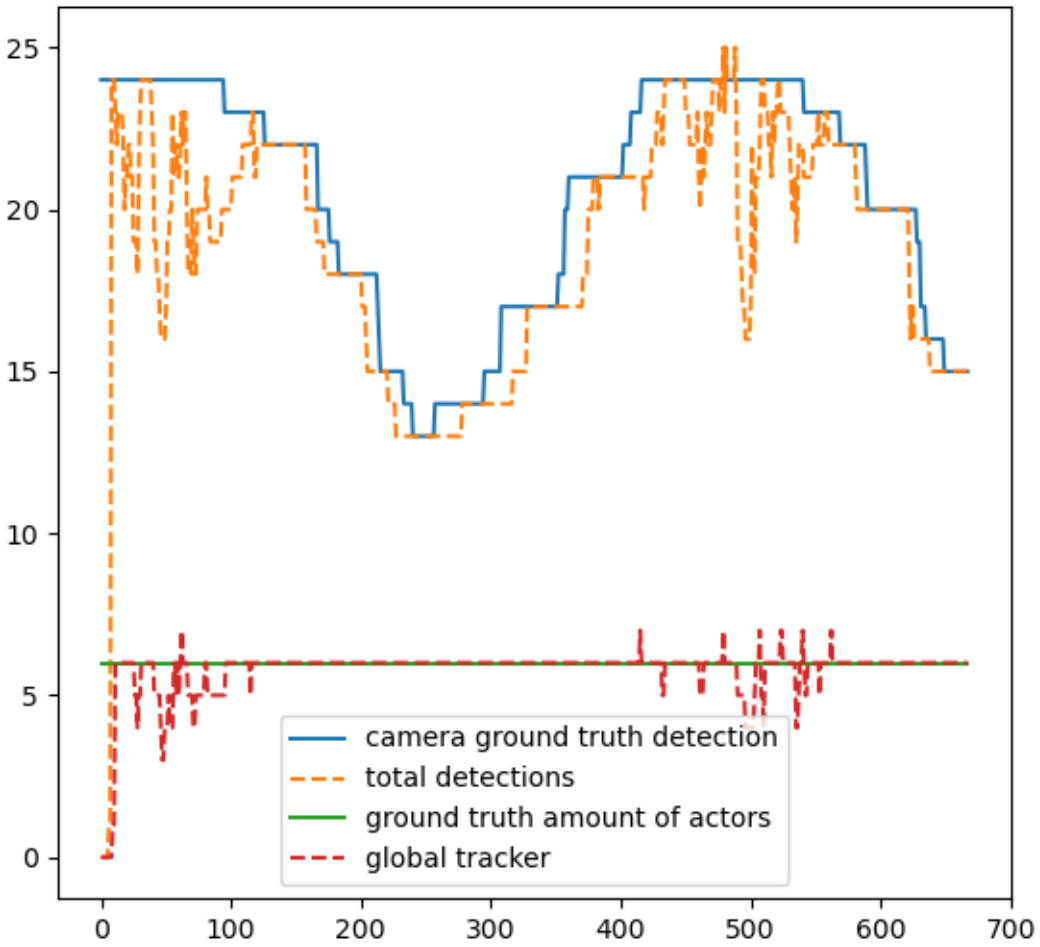


Figure 22: Graph showing the global tracker results in comparison to the ground truth for the cross path scenario. The vertical axis represents the number of actors, while the horizontal axis the frames.

In Figure 23, we see the results of the static objects moving cameras scenario. In this scenario, the actors remain static and the drones are moving aiming to detect the total number of actors in the area. The blue circles represent the predictions of the global tracker regarding the positions of the actors and the orange ones the actual location of the actors. The results indicate that in that scenario the global tracker performed really well, managing to distinguish correctly the observation and provide accurate results regarding the location of the actors.

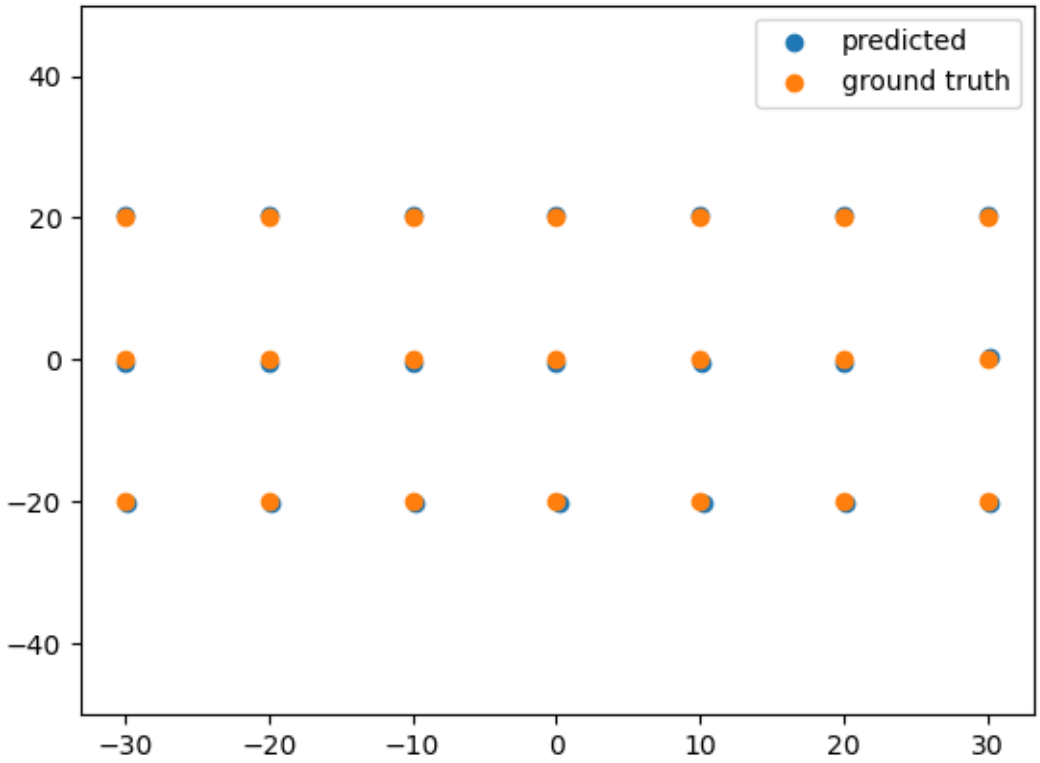


Figure 23: Graph showing the global tracker results in comparison to the ground truth for the static objects moving cameras scenario. The vertical axis represents the number of actors, while the horizontal axis the frames.

We can observe that in all scenarios, the misdetection of the number of actors by the global tracker is caused either by a lack of detections in that case (maybe occlusion) or some few outliers in our dataset based on the way we created the ground truth for our scenarios.

Overall System

To evaluate the performance of the system we have developed six different metrics which are described below:

Detections inside the field of view

The first metric showcases the performance of the object detection and tracking system along with the global tracker. Here, for every timestep by knowing the position of each UAV and the camera pitch we can identify the area visible to the camera and if any actors are part of it. This value is labeled as ground truth tracks. For the same timestep, we compare the number of tracks presented by the global tracker. A big disparity between these two metrics would mean that our global tracker system is to be blamed for poor overall system performance and not our decision making and path planning algorithms.

In Figure 24, we compare these two metrics and can see that our object detection and tracking system performs well as it follows ground truth to a large extent. It is worth noting that we notice sharp changes in the number of actors present in the FOV per timestep and this is attributed to the quick 90 degree turns the drones make during flight. Our primary goal is to cover as much area as possible and find the actors quickly so sweeping motions are beneficial. In the end of the simulation, we can also see

a high number of actors inside the camera frame but only a few detections by the tracker which is caused by the actors being stacked near each other at the center of the map.

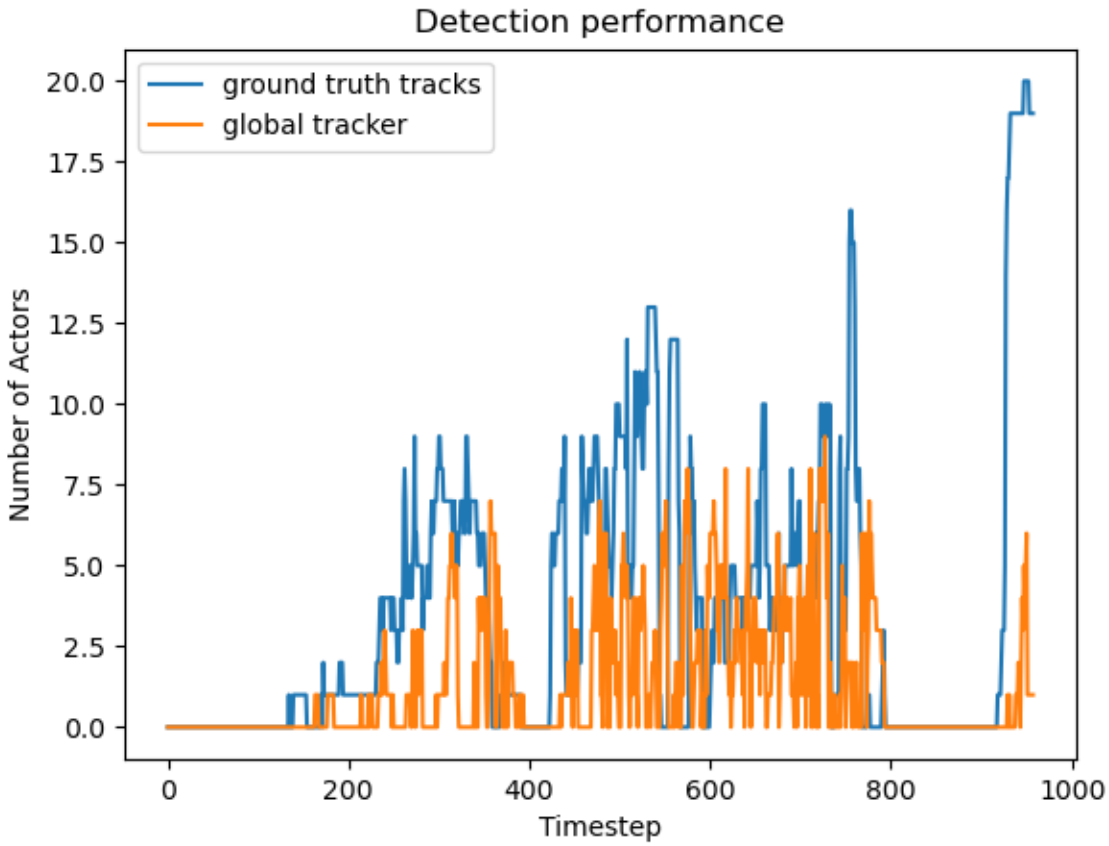


Figure 24. Detection performance under simulation of the system

Actor trajectories

The second metric showcases the trajectories executed by the actors towards the center of the world along with the positions of the detected actors by the UAVs during their search. The detected positions are shown as light grey. In Figure 25, we can see grey dots landing on the exact trajectories of the actors which indicated good localization performance as well as detection performance. In this simulation scenario, all actors were detected.

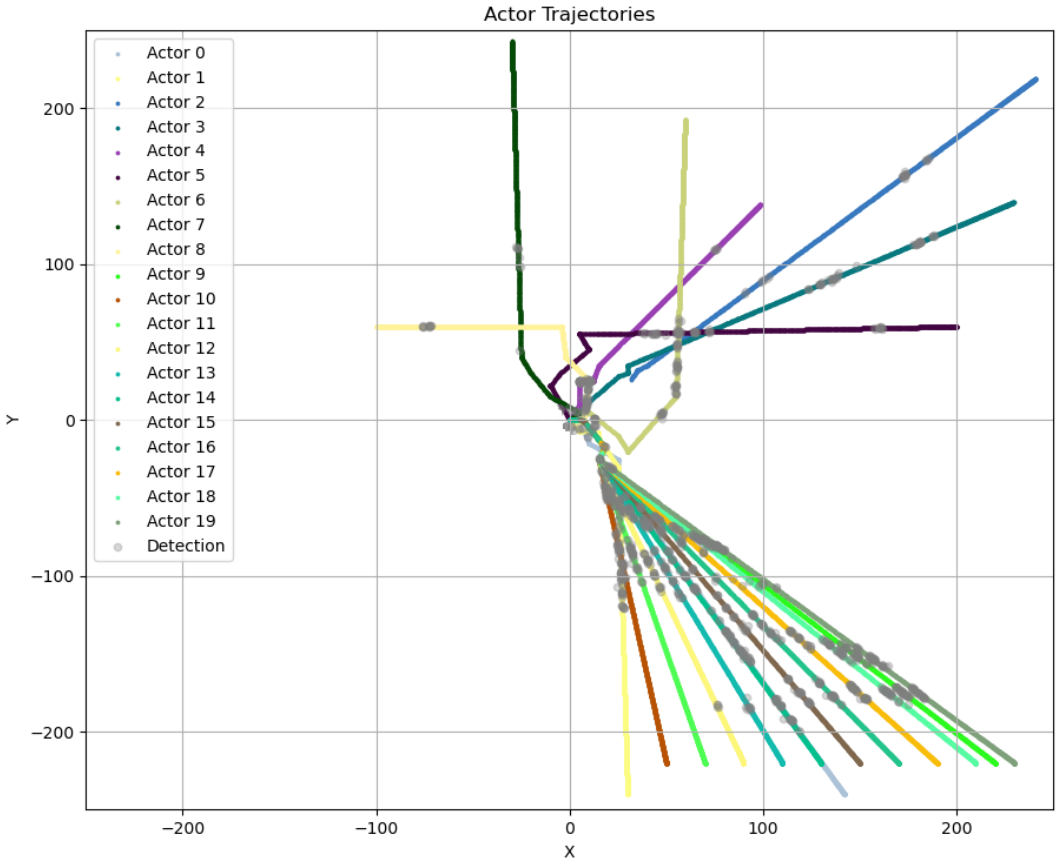


Figure 25. Ground truth actor trajectories and our detection

Map coverage

The third metric identifies the map coverage during simulation. The FOV captured by the 4 drones is logged on every timestep and then plotted on a black and white plot. The white part of the plots indicated an area that has been covered by the drones. We have split the map coverage into five evenly distributed parts time wise to better showcase the decision-making choices made by the system. Lastly, we show the map coverage for the whole simulation in the last plot. In Figure 26, we can see how map coverage alters in such a way that uncharted territories are sequentially navigated thus resulting in nearly complete coverage through optimal navigation as seen on the last plot which includes data from the whole simulation.

DISTRIBUTION A: Distribution approved for public release.

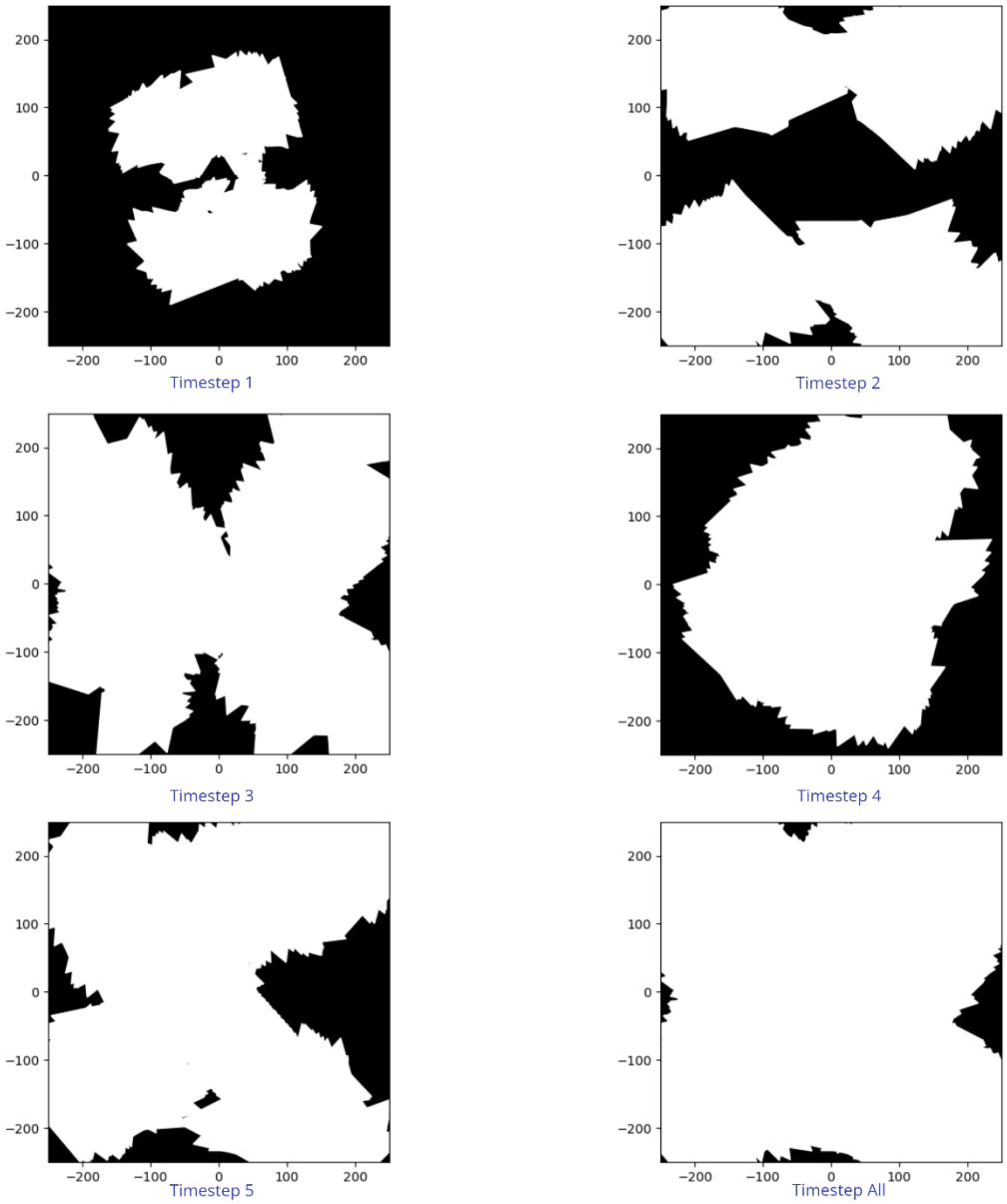


Figure 26. Map coverage based on drone FOV

Percentage of actors under detection

The fourth metric showcases the percentage of actors under detection for each timestep. In Figure 27, we can see the percentage reaching upwards of 45% at a time which would translate to 9 actors at once. This graph follows the same trend as our graph from the first metric which showcased detection performance. Our intention is not to constantly keep track of a detected actor due to the need of finding all actors as quickly as possible.

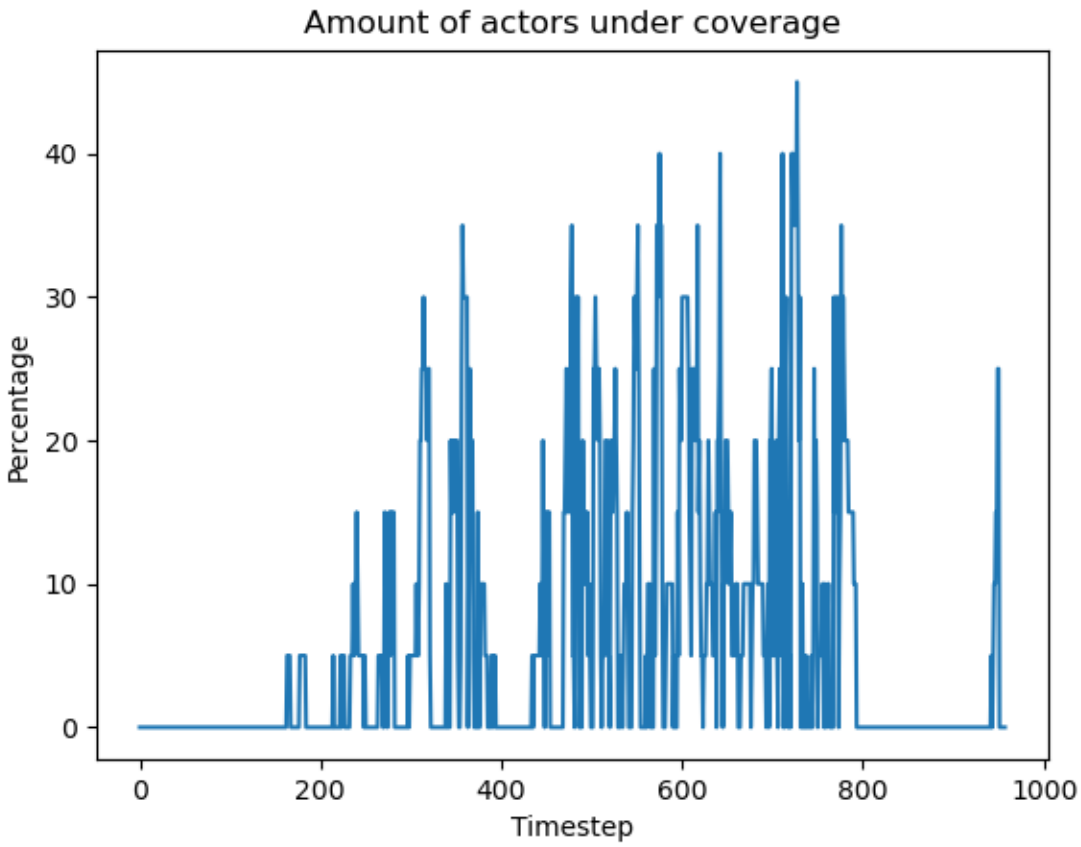


Figure 27. The percentage of actors under coverage per timestep

Percentage of time under coverage

The fifth metric showcases the percentage of time under coverage for each actor. Here we compare the total simulation time to the time each actor was under coverage to come up with the percentage of time. For this metric, an even distribution of coverage times is preferred as this would suggest the decision and path planning algorithm is not biased. In Figure 28, the results from the simulation show this to be the case as only actor 2 and 7 show comparatively less time under coverage compared to the rest.

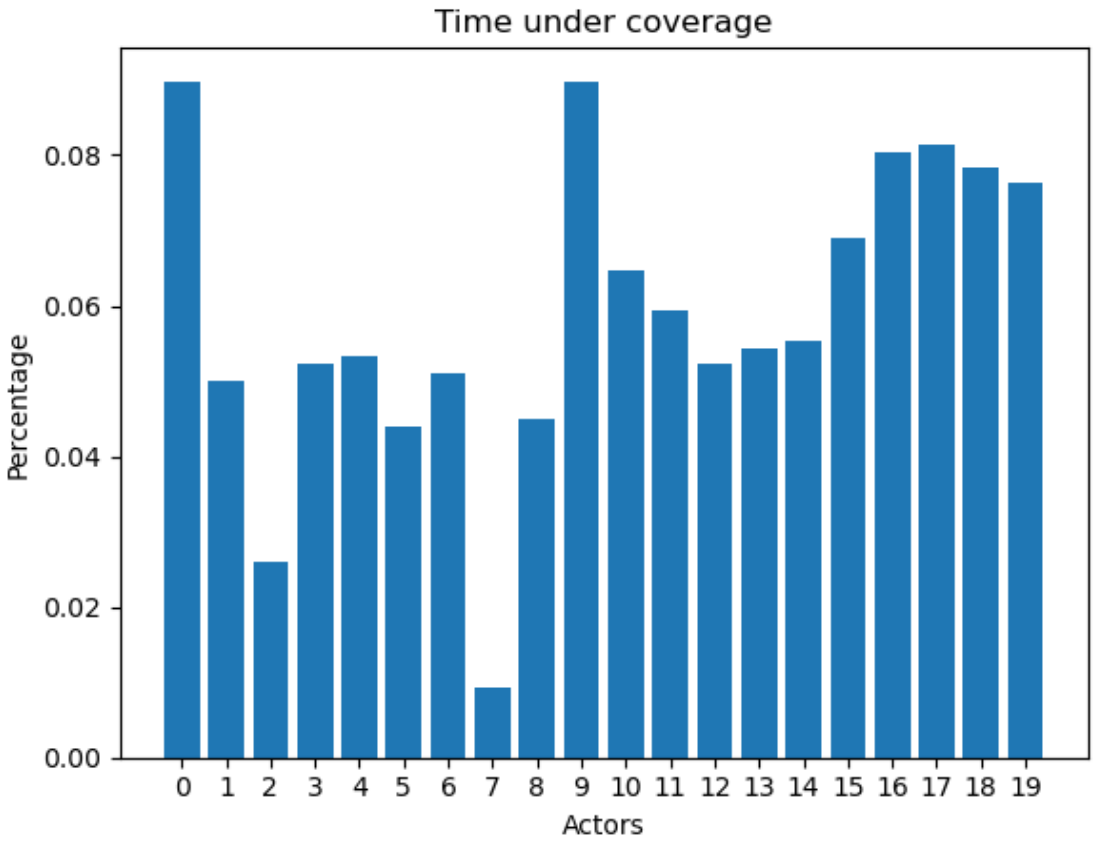


Figure 28. Individual actor time under coverage

Actor alive until found.

The sixth metric showcases the time in seconds until an actor has been caught. If the time is equal to the simulation time then the actor was never caught. The lower the better as this indicates quick detection of actors. In Figure 29, we can tell that all actors were detected within a reasonable amount of time with the longest duration being just 52.8 seconds.

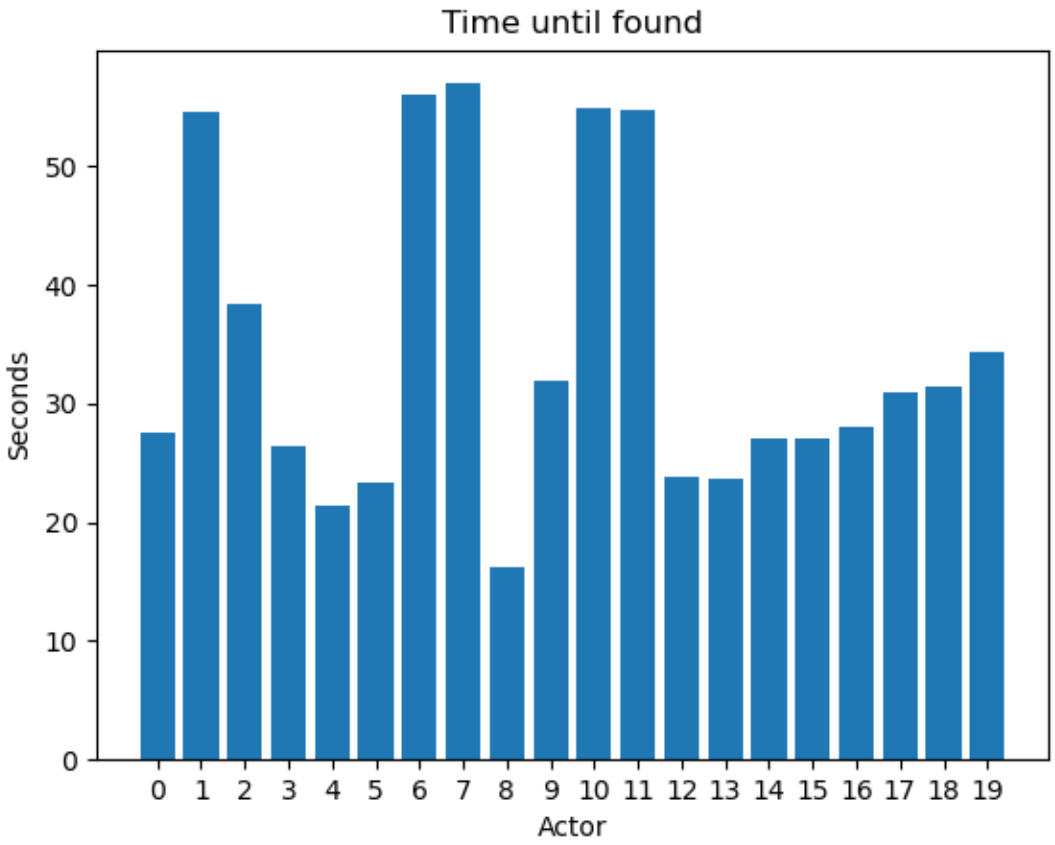


Figure 29. Individual actor time spent before found.

Discussion

In this work, we present a system with a group of UAVs equipped with cameras capable of detecting and tracking multiple objects alongside localizing the detected objects. Furthermore, a global tracker algorithm utilizing information from object detection, and localization is proposed and a system for utilizing all these modules and perform area coverage in a 500x500 world containing actors. The object localization method's performance is proven to be efficient to be used for additional target tracking and mapping applications. Nevertheless, there are certain things affecting the performance like object detection bounding box accuracy in respect of its size, and the placement orientation of the camera in the drones. An accountable error in the size of the bounding box affects the position used in the algorithm. In an image, a pixel point that is close to the centre of the image is considered to be close, despite a pixel that is located away from the centre being considered to be located further away. Furthermore, if the camera is pitched in opposite to the top-down view, objects detected at large distances are slightly distorted causing an increase in the localization error. Similar to the object localization method, the predicted bounding box accuracy is crucial to the proposed guidance method, since it uses the centre point of the bounding box to calculate the velocities and desired yaw commands

DISTRIBUTION A: Distribution approved for public release.

to be used, and it can result in losing track of the desired object. The global tracker algorithm is proven to depend on the quality of the object detection algorithm regarding the size of the predicted bounding box since the accuracy of the localization algorithm is highly affected, and the ability of the object detection algorithm to provide a few amounts of false positives or false negatives. A meaningful way to evaluate the global tracker algorithm is by comparing in every timestep the actual number of the objects in the world with the estimated objects provided by the algorithm. The results have shown that in this case, in all scenarios the algorithm performs exceptionally and in almost all cases where the objects are distinguishable, it manages to perform perfectly. Some outliers in the scenarios were found regarding the predicted detections and the ground truth. An example is that there were cases where in a frame a small percentage of an actor appears, and still the object detection algorithm manages to detect it but this specific actor was not in the annotations. Regarding the ability of the global tracker to keep the tracking id of the detected objects in all the frames the object participates in, it is proven that if the localization error is small, i.e., less than 0.5 m then the algorithm can keep the id even when multiple occlusions occur when the localization error exceeds that threshold then when an occlusion occur regarding of the overlap there might happen identity switch. Another reason causing identity switches is the transition between cameras when selecting the camera closest to the detected object.

Finally, the overall system, that utilizes all the modules and provide path planning and task allocation capabilities seems to be able to cover the area achieving high percentage coverage over the area, and is able to detect all the actors in the world through each experiment. On average, each actor is under detection for 5% of its lifetime in the simulation by a drone, and has 30 seconds average time until is detected.

Conclusions

In this work, we present a system with a group of UAVs equipped with cameras capable of detecting and tracking multiple objects alongside localizing the detected objects, and a system to perform global tracking of people in an area using information from different cameras. Regarding the multiple object detection system, we used and YOLOv7-Tiny in a custom dataset. The results indicate YOLOv7-Tiny achieved a recall of 0.981 and precision of 0.997, and mAP 98.04, while in tracking metrics it had a MOTA of 95.7, an MOTP of 94.7, and a MOTAL of 95.9 proving that indeed it can provide better results with a reduction of parameters in the range of 40%. Furthermore, the object localization method's performance is high enough for it to be used for additional target tracking and mapping applications having a mean squared error of 0.205 m. The guidance algorithm for the drones is capable of tracking an object with great accuracy while maintaining the desired constant distance from it and keeping the object in the centre of the camera frame. The global tracker algorithm proved to perform with high-efficiency counting and tracking of the detected objects in an area from different cameras as shown by the results in all scenarios. In particular, in random and straight-line scenarios the global tracker manages to distinguish the objects in 71 and 96.9% of frames, respectively, and similarly and most importantly in the most difficult scenario with multiple cross paths and occlusions the global tracker achieved to distinguish the objects in 82.7% of frames.

DISTRIBUTION A: Distribution approved for public release.

Our future work will focus on improving the identity switches occurring in scenarios with a lot of occlusions by using DeepSORT instead of SORT and using features from the detected bounding boxes of every multiple object detection module. Another element of our future work is to perform simulations using a higher fidelity simulator such as Sphinx and then proceed to fully utilize our system in real-world scenarios. To conclude, the presented work is a fully applicable system to many domains capable of providing object detection, tracking, localization services, and global tracking utilities, using optical camera-equipped UAVs.

References

Bernardin, K., & Stiefelhagen, R. (2008). Evaluating multiple object tracking performance: The CLEAR MOT metrics. *EURASIP Journal on Image and Video Processing*.

Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. *2016 IEEE International Conference on Image Processing (ICIP).*

Chu, P., & Ling, H. (2019). FAMNet: Joint Learning of Feature, Affinity and Multi-dimensional Assignment for Online Multiple Object Tracking. *CoRR.*

Meera, M., & Mohan, S. (2016). Object recognition in images. *2016 International Conference on Information Science (ICIS).*

Meimetis, D., Daramouskas, I., Perikos, I., & Hatzilygeroudis, I. (2021, August 24). Real-time multiple object tracking using deep learning methods. *Neural Computing and Applications*. doi:10.1007/s00521-021-06391-y

Palak, K., Anshika, S., Shailendra, N., & Pradeep, K. (2016). A survey on object recognition and segmentation techniques. *3rd International Conference on Computing for Sustainable Global Development (INDIACom).*

Sanchez-Matilla, R., Poiesi, F., & Cavallaro, A. (2016). Online Multi-target Tracking with Strong and Weak Detections. *ECCV Workshops 2016.*

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. (2022). {YOLOv7}: Trainable bag-of-freebies sets new state-of-the-art.

Wang, L., Shi, J., Song, G., & Shen, I.-f. (2013). Object detection combining recognition and segmentation. *Computer Vision – ACCV 2007. ACCV 2007. Lecture Notes in Computer Science*, 1258-1276.

Wokje, N., Bewley, A., & Paulus, D. (2017). Simple Online and Realtime Tracking with a Deep Association Metric. *CoRR.*

DISTRIBUTION A: Distribution approved for public release.

DISTRIBUTION A: Distribution approved for public release.