Counter a Drone via Deep Reinforcement Learning

ENDER ÇETIN
M.S. Aerospace Engineering

Advisors
DR. CRISTINA BARRADO MUXÍ

Doctorate program in Aerospace Science and Technology
Department of Computer Architecture - Aerospace Engineering Division
Technical University of Catalonia - BarcelonaTech

A dissertation submitted for the degree of
Doctor of Philosophy
June 2023

Counter a Drone via Deep Reinforcement Learning

Author
Ender Çetin

Advisors
Dr. Cristina Barrado Muxi

Reviewers
Dr. Juan Jose Ramos
Dr. Murat Bronz

Thesis committee
Dr. Juan Jose Ramos
Dr. Murat Bronz
Dr. Esther Salamí

Doctorate program in Aerospace Science and Technology
Technical University of Catalonia - BarcelonaTech
June 2023

PhD. Thesis made in:
Department of Computer Architecture - Aerospace Engineering Division
Esteve Terradas, 5
08860 Castelldefels (Barcelona), Spain

To those who believed in me and made this journey possible,

Contents

List of Figures

Figures in Appendices

List of Tables

List of Publications

Following is the inverse chronological order of the publications resulting from this PhD thesis:

Related Journal Papers

- ÇETIN, ENDER, BARRADO, CRISTINA & PASTOR, ENRIC. 2022. Countering a Drone in a 3D Space: Analyzing Deep Reinforcement Learning Methods. *MDPI Sensors*. D.O.I.: 10.3390/s22228863. **22**, 8863.
- ÇETIN, ENDER, BARRADO, CRISTINA & PASTOR, ENRIC. 2021. Improving real-time drone detection for counter-drone systems. *THE AERONAUTICAL JOURNAL*. D.O.I.: 10.1017/aer.2021.43. **1292**, 1871–1896.
- ÇETIN, ENDER, BARRADO, CRISTINA & PASTOR, ENRIC. 2020. Counter a Drone in a Complex Neighborhood Area by Deep Reinforcement Learning. *MDPI Sensors*. D.O.I.: 10.3390/s20082320. **8**, 2320.
- MUÑOZ, GUILLEM, BARRADO, CRISTINA, ÇETIN, ENDER & SALAMÍ, ESTHER. 2019. Deep Reinforcement Learning for Drone Delivery. *MDPI Drones*. D.O.I.: 10.3390/drones3030072. **3**, 72.

Conference Proceedings

- ÇETIN, ENDER, BARRADO, CRISTINA & PASTOR, ENRIC. 2021 (Oct.). Counter a Drone and the Performance Analysis of Deep Reinforcement Learning Method and Human Pilot. *In: 2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*.
- ÇETIN, ENDER, BARRADO, CRISTINA, MUÑOZ, GUILLEM, MACIAS, MIQUEL & PASTOR, ENRIC. 2019 (Sep.). Drone Navigation and Avoidance of Obstacles Through Deep Reinforcement Learning. *In: 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)* ***Best of Session Award***.
- ÇETIN, ENDER, MUÑOZ, GUILLEM, PASTOR, ENRIC & BARRADO, CRISTINA . 2018 (Dec.). Deep reinforcement learning for multiple drones air traffic management. *In: 8th SESAR Innovation Days* ***Poster Session***.

Other Journal Papers

- ROYO, PABLO, ASENJO, ÀLEX, TRUJILLO, JUAN, CETIN, ENDER & BARRADO, CRISTINA. 2022. Enhancing Drones for Law Enforcement and Capacity Monitoring at Open Large Events. *MDPI Drones*. D.O.I.: 10.3390/drones6110359. **6/11**, 359.

- ÇETIN, ENDER, CANO, ALICIA, DERANSY, ROBIN, TRES, SERGI & BARRADO, CRISTINA . 2022. Implementing Mitigations for Improving Societal Acceptance of Urban Air Mobility. *MDPI Drones*. D.O.I.: 10.3390/drones6020028. **6**, 28.

Acknowledgements

I would like to express my deepest gratitude and appreciation to all those who have supported the successful completion of my Ph.D. study, which have spanned nearly five years. This academic journey has been a remarkable experience, and I am sincerely grateful to the numerous individuals and institutions that have supported and guided me throughout this challenging journey. Their tireless encouragement, invaluable expertise, and unwavering commitment have played a crucial role in shaping my research, enhancing my knowledge, and finally helping me to reach this significant milestone in my academic career.

I would like to express my sincerest thanks to my supervisor, Dr. Cristina Barrado Muxí, for her exceptional guidance, endless support, and invaluable tutoring throughout my research journey. Her profound knowledge, insightful feedback, and never-ending encouragement have been a key in shaping the direction and quality of my research.

I also thank my research group Intelligent Communications and Avionics for Robust Unmanned Aerial Systems (ICARUS). Specifically, I would like to thank Dr. Enric Pastor Llorens, Dr. Esther Salamí San Juan, and Dr. Pablo Royo Chic for their collaboration and fellowship. The collective expertise and diverse profiles within the group have greatly improved my understanding of the academic knowledge. I would like to express my full appreciation to each of the members of the group for their valuable contributions that have made a significant impact on the outcome of my research.

I would also like to thank Dr. Ramos from Universitat Autònoma de Barcelona - UAB and Dr. Bronz from Ecole Nationale de l'Aviation Civile (ENAC) for agreeing to review this PhD dissertation and being members of the committee. I really appreciate your time and commitment, which resulted in valuable comments to improve the quality of the final document. I must also acknowledge Dr. Salamí, from Universitat Politècnica de Catalunya (UPC), for agreeing to be part of the committee.

I would like to dedicate a special paragraph to my beloved parents, my dear father Neşat Çetin, my dear mother Fatma Çetin, and my beloved sister Merve Çetin. They have been an endless source of love, encouragement and support throughout this journey. Their sacrifices, understanding and faith in my abilities have been the cornerstone of my personal support, perseverance and determination. Your continued presence, patience and understanding during the uncountable hours spent researching have been really amazing. I am eternally grateful to them for their unconditional support and continuous motivation. This achievement would not have been possible

without them by my side, and I feel truly blessed to have them as my pillars of strength. Thank you for your endless support and always believing in me.

At this moment of sincere gratitude, I would like to dedicate a special paragraph to express my deepest thanks to Elisabet Massó and her family. Their ongoing support and guidance have been essential in helping me overcome the challenges of adapting to a new country, learning its language, and integrating myself into its vibrant culture. Elisabet Massó's patience, kindness and tireless efforts to teach me the language and introduce me to local customs have been invaluable. I am forever grateful to both her and her family for their continued support, which has enriched my life immeasurably. Thank you for being the guiding light during my journey and for making me feel at home in this new land.

I am also thankful to the administrative staff and technical support team at UPC for their assistance, guidance, and smooth functioning of the facilities and resources essential for my research work. Their efficiency and professionalism have aided the successful completion of my study.

Lastly, I would like to express my heartfelt gratitude to friends for their strong support, understanding, and patience throughout this exciting journey. Their confidence in my abilities and their continuous encouragement have been a source of motivation and inspiration.

Barcelona, June 2023
Ender Çetin

Abstract

Unmanned aerial vehicles (UAV) also known as drones have been used for a variety of reasons such as surveillance, reconnaissance, shipping and delivery, etc. and commercial drone market growth is expected to reach remarkable levels in the near future. However, drones can accidentally or intentionally violate the air routes of major airports, flying too close to commercial aircraft or invading the privacy of someone. In order to prevent these unwanted events to happen, counter-drone technology is needed to eliminate the threats coming from drones and hopefully the drones can be integrated into the skies safely. A number of counter-drone solutions are being developed, but the cost of drone detection ground systems can also be very high, depending on the number of sensors deployed and powerful fusion algorithms.

Counter-drone system supported by an artificial intelligence (AI) method can be an efficient way to fight against drones instead of human intervention. Considering the recent advances in AI, counter-drone systems with AI can also be very accurate. The time required to engage with the target can be less than other methods based on human intervention such as bringing down a malicious drone by a laser gun. Also, AI can identify and classify the target with a high precision in order to prevent a false interdiction with the targeted object. Counter-drone technology with AI will bring important advantages to the threats coming from some drones and will help the skies to become safer and more secure. AI has been used in different research areas in aerospace to create an intelligent system. Especially, a drone can be controlled by AI methods such as deep reinforcement learning (DRL) in different purposes. With the support of DRL, drones can become more intelligent and eventually they can be fully autonomous.

The main objective of this PhD thesis is to develop an artificial intelligence approach based on deep reinforcement learning to counter drones that may pose a threat to safety or security. AI agents can continuously learn and adapt to new threats and countering drones with DRL has several advantages. One of the most important advantages is autonomous decision-making which enables AI agents to make autonomous decisions based on their environment and the situation. In this way, drone threats can be countered quickly and effectively, even in vulnerable environments. Additionally, AI agents can be trained in simulation, allowing for safe experimentation, testing, and validation before deployment.

Firstly, DRL architecture is proposed to make drones behave autonomously inside a suburb neighborhood environment. Secondly, a state-of-the-art object detection algorithm for drone detection is also added to the counter drone solution. The construction of drone detection models

involves transfer learning and training a state-of-the-art object detection algorithm. After achieving fully autonomous drone which can avoid obstacles in an environment, a deep reinforcement learning method to counter a drone in a 2D space in an environment is presented. In this way, drone can maintain its current altitude, and it can try to catch another drone without crashing any obstacle in the environment. Finally, a deep reinforcement learning model is developed to counter a drone in a challenging 3D space in an environment. The learner drone is not only moving in a 2D space but also changing altitudes to eliminate the target drone.

DRL is a promising approach for countering drones. It involves training AI models but there are certain challenges that need to be addressed. One of the biggest challenges is that training DRL algorithms require a lot of computational power. If the training is carried out without a simulation environment, training DRL models can be computationally intensive which can make them impractical for some applications where resources are limited. Furthermore, there are concerns about the actions AI agents could take in sensitive environments. It is important to ensure that AI agents are properly trained and validated so that they can make safe and responsible decisions. Without proper testing and validation, there is a risk that AI agents in sensitive areas such as airports or critical infrastructure might perform actions that could be dangerous or violate regulations. By addressing these challenges, DRL-based counter-drone solutions can be made more practical, efficient, and secure for future use.

Resumen

Los vehículos aéreos no tripulados (UAV), también conocidos como drones, se han utilizado por una variedad de razones, como vigilancia, reconocimiento, envío y entrega de paquetes, etc. y se espera que el crecimiento del mercado de drones comerciales alcance niveles notables en un futuro próximo. Sin embargo, los drones pueden violar accidental o intencionadamente las rutas aéreas de los principales aeropuertos, volando demasiado cerca de aviones comerciales o invadiendo la privacidad de alguien. Para evitar que sucedan estos eventos no deseados, se necesita tecnología contra drones para eliminar las amenazas provenientes de los drones para que éstos pueden integrarse en los cielos de manera segura. Se han desarrollado varias soluciones contra drones, pero el costo de los sistemas terrestres de detección de drones puede ser muy alto, según la cantidad de sensores desplegados y los potentes algoritmos de fusión.

El sistema contra drones, respaldado por un método de inteligencia artificial (IA), puede ser una forma eficiente de luchar contra los drones si necesidad de intervención humana. Teniendo en cuenta los avances recientes en IA, los sistemas contra drones con IA también pueden ser muy precisos. El tiempo requerido para neutralizar al objetivo puede ser menor que con los métodos basados en la intervención humana, como por ejemplo derribar un dron malicioso con una pistola láser. Además, la IA puede identificar y clasificar el objetivo con alta precisión para evitar un error al neutralizar a un objetivo equivocado. La tecnología contra drones con IA brindará ventajas importantes a las amenazas que suponen algunos drones y ayudará a que los cielos se vuelvan más seguros. La IA se ha utilizado en diferentes áreas de investigación en la industria aeroespacial para crear sistemas de decisión inteligente. Especialmente, un dron puede ser controlado por métodos de IA, como el aprendizaje profundo por refuerzo (DRL), para diferentes propósitos. Con el apoyo de DRL, los drones pueden volverse más inteligentes y eventualmente pueden ser completamente autónomos.

El objetivo principal de esta tesis doctoral es desarrollar un enfoque de inteligencia artificial basado en el aprendizaje profundo por refuerzo para contrarrestar los drones que pueden representar una amenaza para la seguridad. Los agentes basados en IA pueden aprender y adaptarse continuamente a las nuevas amenazas, ya que contrarrestar los drones con DRL tiene varias ventajas. Una de las ventajas más importantes es la toma de decisiones autónoma que permite a los agentes tomar decisiones autónomas en función de su entorno y la situación. De esta forma, las amenazas de los drones se pueden contrarrestar de forma rápida y eficaz, incluso en entornos complejos. Además, los agentes inteligentes pueden entrenarse con simulaciones, lo que permite una experimentación, prueba y validación seguras antes de la implementación.

En primer lugar, se propone una arquitectura DRL para hacer que los drones se comporten de forma autónoma dentro de un entorno de barrio suburbano. En segundo lugar, se propone una solución contra drones apoyado por un algoritmo de detección de objetos de última generación modificado para la detección de drones. La construcción de modelos de detección de drones usa el aprendizaje por transferencia además del algoritmo de detección de drones. Después de lograr un dron completamente autónomo que puede evitar obstáculos en un entorno, se presenta un método de aprendizaje profundo por refuerzo para contrarrestar un dron en un espacio 2D. De esta manera, el dron puede mantener su altitud actual y puede intentar atrapar a otro dron sin chocar con ningún obstáculo en el entorno. Finalmente, se desarrolla un modelo de aprendizaje profundo por refuerzo para contrarrestar un dron en un espacio 3D, un entorno de complejidad desafiante. El dron aprende no sólo a moverse en un espacio 2D, sino que también cambia de altura para capturar al dron malicioso.

DRL es un enfoque prometedor para tareas contra drones. Implica entrenar modelos de IA y hay ciertos desafíos que deben abordarse. Uno de los mayores desafíos es que entrenar algoritmos DRL requiere mucha potencia computacional. El entrenamiento de los modelos DRL suele ser computacionalmente intensivos, lo que puede hacerlo poco práctico para algunas aplicaciones donde los recursos son limitados. Además, existen preocupación sobre las acciones que los agentes de IA podrían tomar en entornos sensibles. Es importante asegurarse de que los agentes de IA estén debidamente capacitados y validados para que puedan tomar decisiones seguras y responsables. Sin las pruebas y la validación adecuadas, existe el riesgo de que los agentes de IA en áreas sensibles, como aeropuertos o infraestructura crítica, puedan realizar acciones que podrían ser peligrosas al violar las normas establecidas. Al abordar estos desafíos, las soluciones contra drones basadas en DRL se pueden hacer más prácticas, eficientes y seguras para uso futuro.

Resum

Els vehicles aeris no tripulats (UAV) també coneguts com a drons s'han utilitzat per diverses raons, com ara vigilància, reconeixement, enviament i lliurament, etc. i s'espera que el creixement del mercat de drons comercials assoleixi nivells notables en un futur proper. Tanmateix, els drons poden violar accidentalment o intencionadament les rutes aèries dels principals aeroports, volar massa a prop d'avions comercials o envair la privadesa de les persones. Per tal d'evitar que es produeixin aquests esdeveniments no desitjats, és necessària la tecnologia contra-drones, per tal d'eliminar les amenaces que provenen d'alguns drons i fer que els drons es puguin integrar a l'espai aéri de manera segura. S'estan desenvolupant diverses solucions contra drons, però el cost dels sistemes terrestres de detecció de drons també pot ser molt elevat, depenent del nombre de sensors desplegats i d'algoritmes de fusió potents.

El sistema contra drons recolzat per un mètode d'intel·ligència artificial (IA) pot ser una manera eficient de lluitar contra els drons en lloc de la intervenció humana. Tenint en compte els avenços recents en IA, els sistemes de contra-drones amb IA també poden ser molt precisos. El temps necessari per neutralitzar l'objectiu pot ser inferior al d'altres mètodes basats en la intervenció humana, com ara fer caure un drone maliciós amb una pistola làser. A més, la IA pot identificar i classificar l'objectiu amb una alta precisió per tal d'evitar un error en la selecció d'aquest objectiu. La tecnologia contra drons amb IA aportarà avantatges importants a les amenaces que provenen d'alguns drons i ajudarà a que els cels siguin més segurs i segurs. La IA s'ha utilitzat en diferents àrees de recerca en l'aeroespacial per crear sistemes intel·ligents. Especialment, un drone es pot controlar mitjançant mètodes d'IA com ara l'aprenentatge profund per reforç (DRL) amb diferents finalitats. Amb el suport de DRL, els drons poden ser més intel·ligents i, finalment, poden ser totalment autònoms.

L'objectiu principal d'aquesta tesi doctoral és desenvolupar un enfocament d'intel·ligència artificial basat en l'aprenentatge profund per reforç per contrarestar els drons que poden suposar una amenaça per a la seguretat. Els agents d'IA poden aprendre i adaptar-se contínuament a noves amenaces. Contrarestar els drons amb DRL té diversos avantatges. Un dels avantatges més importants és la presa de decisions que permet als agents d'IA prendre decisions autònomes en funció del seu entorn i de la situació. D'aquesta manera, les amenaces dels drons es poden contrarestar de manera ràpida i eficaç, fins i tot en entorns complexos. A més, els agents d'IA es poden entrenar en simulació, cosa que permet experimentar, provar i validar de manera segura abans del desplegament.

En primer lloc, es proposa l'arquitectura DRL per fer que els drons es comportin de manera autònoma dins d'un entorn d'un barri suburbà. En segon lloc, també s'afegeix un algorisme de detecció d'objectes d'última generació per a la detecció de drons com a part de la solució contra drons. La construcció de models de detecció de drons pot ser recolçada per l'aprenentatge per transferència, a mes de per un algorisme de detecció d'objectes d'última generació. Després d'aconseguir un dron totalment autònom que pugui evitar obstacles en un entorn, es presenta un mètode d'aprenentatge profund per reforçament per neitralitzar un dron en un espai 2D. D'aquesta manera, el drone pot mantenir la seva altitud, i pot intentar captura l'altre dron sense xocar amb cap obstacle a l'entorn. Finalment, es desenvolupa un model d'aprenentatge profund per reforçament per contrarestar un dron en un espai 3D, un entorn desafiant. El dron intelligent no només es mou en un espai 2D, sinó que també canvia d'altitud per eliminar el dron objectiu.

DRL és un enfocament prometedor per combatre els drons. Implica entrenar models d'IA, però hi ha certs reptes que cal abordar. Un dels majors reptes és que entrenar algorismes DRL requereix molta potència computacional. L'entrenament dels models DRL pot ser computacionalment intensiu, cosa que pot fer-lo poc pràctic per a algunes aplicacions on els recursos són limitats. A més, hi ha preocupacions sobre les accions que els agents d'IA podrien prendre en entorns sensibles. És important garantir que els agents d'IA estiguin degudament formats i validats perquè puguin prendre decisions segures i responsables. Sense les proves i validacions adequades, hi ha el risc que els agents d'IA en àrees sensibles com aeroports o infraestructures crítiques puguin dur a terme accions que podrien ser perilloses o infringir la normativa. En abordar aquests reptes, les solucions contra-drones basades en DRL es poden fer més pràctiques, eficients i segures per a un ús futur.

List of Acronyms

AI Artifical Intelligence

BVLOS Beyond Visual Line of Sight

C-UAS Counter Unmanned Aerial System

CNN Convolutional Neural Network

DDQN Double Deep Q-network

DQfD Deep Q-learning from Demonstrations

DQN Deep Q-network

DRL Deep Reinforcement Learning

GNSS Global Navigation Satellite System

NED North East Down

NN Neural Networks

PER Prioritized Experience Replay

RGB Red, Green, and Blue

RL Reinforcement Learning

TL Transfer Learning

UTM Unmanned Aircraft System Traffic Management

XAI Explainable Artificial Intelligence

İstikbal Göklerdedir.
[The future is in the skies.]

— Mustafa Kemal Atatürk

When once you have tasted flight, you will forever walk the earth with your eyes turned skyward, for there you have been, and there you will always long to return.

— Leonardo DaVinci

I Introduction

Drone industry has been improving itself over the years and drones market is growing drastically. According to counter unmanned aerial system (C-UAS) and unmanned aircraft system traffic management (UTM) market analysis Unmanned-Airspace (2017), commercial drone market is estimated to be worth United States Dollar (USD) 6510.8 Million in 2022 and it is projected to reach USD 34500 Million by 2028 with a Compound Annual Growth Rate (CAGR) of 32.0% during the review period. Drones are used by professionals and hobbyists for different purposes such as delivery, wildlife monitoring, reconnaissance, inspection and surveillance. Federal Aviation Administration (FAA) from United States published a report FAA (2022) that 865,505 drones registered. In this report it is stated that 314,689 of them are commercial drones and 538,172 of them are recreational drones or hobby use of drones. These numbers are quickly increasing all over the world. This chapter provides a brief overview of artificial intelligence and its application to drones, as well as counter drone technology.

I.1 Drones and Artificial Intelligence

An unmanned aerial vehicle known as a drone is an aircraft without human pilot and it is guided autonomously by using remote control and onboard sensors and electronic transmitters.

Drones are flying machines ranging from insect-sized flapping crafts to large airplanes the size of a commercial airline jet Palmer & Clothier (2013). Their capabilities are also wide-ranging: some drones are capable of flying for only a few minutes, while others can fly for days at a time. The applications of drones are also diverse. Drones can be used for a variety of civil and military

purposes. The classification of the applications of drones is presented in Figure I-1. The drone can operate in challenging environments both outside and indoors. While the initial applications of drones were mainly for military purposes, and later for recreational purposes, drones are used today in many civil applications and in public spaces. Some of the most common commercial applications and uses for drones include agriculture (crop spraying, crop monitoring, etc.), live streaming events, emergency response, search and rescue, firefighting, disaster zone mapping, mapping and surveying and artificial intelligence applications Kugler (2019); Chew *et al.* (2020). More recently, the societal utility of drones has been further enhanced in the management of the global COVID-19 pandemic, with use cases such as aerial spraying of public areas to disinfect streets, the surveillance of public spaces, and monitoring local authorities during lockdowns and quarantine Restás *et al.* (2021).

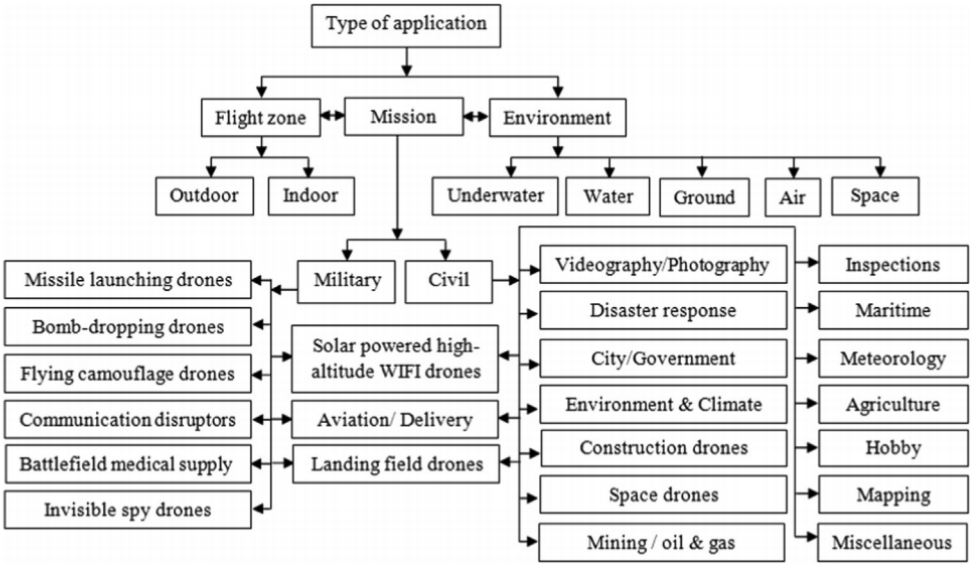


Figure I-1: *Applications of drones Hassanalian & Abdelkefi (2017)*

There are many types of drones available and drones have been categorized in different ways in terms of their weight, size, autonomy level and the usage area. For instance, European Union Aviation Safety Agency (EASA) EASA (2022) defines drones in different groups presented in Table I-1 depending on the actual weight of the drone considering leisure drone activities and low-risk commercial activities. Additionally, drones with different design are presented in Figure I-2. Drones can be fixed-wing, helicopter, ducted shape or in a bird shape design such as smart bird. Also, the first stratospheric UAS of its kind, high altitude platform station (HAPS), called Zephyr from Airbus and Space (2023) is flying continuously for months at around 70,000ft altitude. This UAS demonstrated day/night longevity in the stratosphere.

Table I-1: *The type of drones.*

UAS	Subcategory	Operational Restriction
< 500 g	A1	fly over people but not over assemblies of people
< 2 kg	A2	fly close to people
< 25 kg	A3	fly far from people

Artificial intelligence (AI) has been utilized in different purposes to support Unmanned air vehicles (UAV). For example, a drone supported with AI can navigate in an unknown environment by detecting and avoiding the obstacles by using object detection algorithms. Moreover,

(a) Fixed wing Northrop-Grumman-Corporation's (2022b)

(b) Helicopter Northrop-Grumman-Corporation's (2022a)

(c) V-bat Shield-AI (2023)

(d) Smart bird Festo (2022)

(e) Parrot Anafi USA Parrot (2023)

(f) Volodrone Volocopter-GmbH (2023)

(g) Matrice 300 DJI (2023)

(h) Zephyr and Space (2023)

Figure I-2: *Different types of drones.*

a drone can deliver medicine or any kind of materials by operating autonomously using an AI method. Reinforcement learning (RL) which is an AI method based on trial and error experiences, is also used in drones in different scenarios. Drones supported with RL can operate autonomously to deliver goods, navigate in an environment, or even in drone racing tournaments where drones race against human pilots. Reinforcement learning methods showed promising results in many areas such as gaming which requires a lot of experiences to achieve successful results. This shows that a drone supported with RL can be used to counter drones in an effective way. Fighting against unknown and malicious drones can be very accurate and efficient by implementing an AI method in counter-drone technology. AI methods can speed up the time to engage with the target compared to other methods based on human intervention. A drone with AI can identify and classify the target with a high precision. It is also possible to prevent a false interdiction with the targeted object by using an AI. Countering a drone in 2D space can be an easy way since the drone and the target move without changing altitude. However, if the target changes an altitude and moves in 3D space, which is highly expected in real world, an AI method such as reinforcement learning can be an efficient method thanks to perception and interpretation of environment by RL agent drone since RL models can learn by interacting in an environment by trial and error experiences. This is an important advantage to be used against drones in 3D space.

In recent years, researchers proposed some studies in the area of deep reinforcement learning (DRL) and UAV. In this context, the studies are mostly focused on the topics of drone detection and of navigation of drones in an unknown environment, avoiding obstacles. Akhloufi *et al.* (2019) propose deep reinforcement learning and deep search areas for drones' pursuit-evasion problems. Firstly, DRL is used to follow a target drone, by predicting its actions to follow the target. Also, supervised learning is applied by using a large dataset of drone images. Another example is to predict the position of the target drone using deep object detector and search area proposal. YOLO-v2 Redmon *et al.* (2016) is used as an object detector. A drone can also be navigated with the help of deep reinforcement learning, using the sensor data. Hodge *et al.* (2021) proposed a generic navigation algorithm that uses sensor data. Authors state that locating problems rapidly

and accurately in hazardous situations is vital. Proximal policy optimisation (PPO) DRL algorithm coupled with incremental curriculum learning and long short-term memory neural networks are implemented. The algorithm acts as a recommender for autonomous drone and human pilot if it is applicable. DRL is also used against jamming. In a research by Lu *et al.* (2018) DRL methods were applied to choose the relay policy by using a drone as part of a cellular communication framework against jamming. In this method, the cellular systems can resist the jamming without knowing the jamming model and the network model. In this article, it is stated that the optimal performance can be achieved by adequately interacting with the jammer. Rodriguez-Ramos *et al.* (2019) proposed a DRL for the autonomous landing of drones on a moving platform. The drone control during landing is performed using the deep deterministic policy gradients (DDPG) algorithm and tested over a simulator interface. Reinforcement learning is also used in Bertoin *et al.* (2021) to focus on automated anti-collision systems. In this study, it is stated that training Reinforcement Learning agents can deflect a drone equipped with an automated anti-collision system. The effectiveness of reinforcement learning in finding security holes for the autonomous systems is also highlighted. Moreover, in a study by Lee (2021), tracking and capturing an invading drone using a vision-based drone to defend it is presented. Firstly, researchers developed a deep learning-based detection algorithm which is applied to detect a drone and estimate its position. Secondly, a deep reinforcement learning algorithm is introduced to find the optimal behavior to track a drone. Reinforcement learning can be combined with another deep learning method called imitation learning. For example, a deep reinforcement learning method is proposed in He *et al.* (2020) to navigate an UAV in an unknown environment using demonstration data. Researchers presented that expert demonstrations can speed up the training process and both the policy and Q-value network are pre-trained in the imitation phase. Simulation results show that UAV can avoid obstacles in an unknown 3D environment.

I.2 Counter Drone Systems

Counter drone system is an emerging necessity to detect and eliminate a malicious drone or any kind of UAV which threatens public security or individual's privacy. In order to eliminate threats to public security and privacy because of these misused drones, counter drone solutions have been proposed by researches using different methods and tools. In Figure I-3, steps to follow when countering a drone are explained. Firstly, drones are detected identified and tracked. Secondly, the decision is made whether the drone detected is friendly or malicious. Later, the interdiction methods are applied to counter a drone. Finally, the drone is disarmed, isolated or terminated. The technologies for the detection, localization, and identification of small UAVs include infrared sensors, laser devices, optical surveillance aids and devices, acoustic devices, LiDAR (Light Detection and Ranging) sensors, equipment operating with image recognition technology, devices capable of detecting and localizing UAV remote control signals, and human air observers Kratky & Farlik (2018). After the target drone is detected, the elimination methods such as laser guns, water cannons, birds trained for catching drones, jamming can be applied. More details on detecting drones can be seen in a survey by Chiper *et al.* (2022). In this survey, the different drone detection and defense systems based on different types of methods which were proposed in the literature are presented. In the real world, the target position can be detected in many different sensing technologies such as radar Drozdowicz *et al.* (2016); Semkin *et al.* (2021), acoustical Bernardini *et al.* (2017a); Mezei *et al.* (2015), radio frequency (RF) Nguyen *et al.* (2016), optical Opromolla *et al.* (2018), lidar de Haag *et al.* (2016a), or a deep learning-based solution Çetin *et al.* (2021); Aker & Kalkan (2017).

There are many study cases which investigate countering drones. In a study presented in defence science journal Kratky & Farlik (2018), UAV detection and elimination are discussed. The

terminology used in this journal is based on The North Atlantic Treaty Organization (NATO) . According to this terminology, the problem of defence is divided into 3 main aspects: Air surveillance, command and control and elimination. Air surveillance is used for detecting and identifying UAV. Command and control collects data from sensors and acts as a decision maker mechanism when the actions are taken against the aerial object. Elimination is a collection of methods for interdiction of the threats. Devices to detect UAV, such as radar systems, sensors or acoustic devices, are existing technologies used for countering drones. To eliminate these threats several methods are proposed. These methods include shotguns, laser guns, nets, water cannons, birds trained for catching drones, jamming the command and control radio signals and jamming the global navigation satellite system signals. Detailed explanation about conter drone systems and the methods used to counter drones presented in surveys Park *et al.* (2021) and Michel (2018). Tables I-2 and I-3 present the methods used in counter drone systems and their definitions.

In a survey Lykou *et al.* (2020), the limitations of available counter-drone technologies and the advantages of using them are explained in detail. In this study, it is stated that the threats coming from drones in airports are not easy to deal with. However, methods such as geofencing, multiple radars with different detection ranges and a combination of radio-frequency sensors with visual detection sensors can be implemented to defend airports against unwanted drones. It is also highlighted that airfield operators must remain within the law when using disruptive technologies, and the risks to the wider community should be fully assessed and understood. Researchers Watkins et al. proposed a blueprint Watkins *et al.* (2020) which offers a design for a novel autonomous counter drone tool based on the weaponization of "hard-to-patch" vulnerabilities. The paper highlights the problem of privacy violation due to drones and presents a counter-drone tool which breaks the drone's autonomy code. In another study Barišic *et al.* (2022), it is presented that the system developed can extract target UAV trajectory which is enough to intercept an intruder drone. The research states that with a priori knowledge of the shape of the target trajectory, they managed to track and intercept an intruding drone 30% faster than their sentry vehicle in more than half of the software in the loop (SITL) experiments conducted. The system is also tested in an outdoor unstructured environment and the drone successfully intercepts in 9 out of 12 experiments.

In the literature, the researchers also studied different instruments to detect unmanned aerial vehicles. Choi *et al.* (2018) propose a radar system to detect drones such as quadcopters from long distances. The drone detection system has also been experimented in outdoor environments to verify the long range drone detection. In another research by Bernardini *et al.* (2017b), acoustic drone detection method is presented. A machine learning based warning system is developed to detect the drones by using the drone audio fingerprint. The effectiveness of the sensing approach is supported by preliminary experimental results. de Haag *et al.* (2016b) presented LiDAR and radar sensors to detect small unmanned aerial systems platforms. The position and average velocity of the target can also be determined very accurately by applying motion compensation and target tracking techniques thanks to the LiDAR's high update rate and high ranging accuracy. A full counter drone system with several types of sensors and several level of prediction and fusion is also presented by Samaras *et al.* (2019).

I.3 Motivation of this PhD

This section focuses on the importance of counter-drone systems to deal with the unwanted or intruder drones. Sections I.3.1 and I.3.2 present the growing trend for drone market and the drone incidents occurred lately in Europe and in all over the world. These future needs drive the motivation of this PhD thesis.

Table I-2: *Detection, Tracking and Identification.*

Methods	Feature	Definition
Radar	Physical object	Long range ($1-20$ km) Less affected by the weather High expense
Radio-frequency (RF) receiver	RF signal	Long range ($3-50$ km) Detect the drone operator Obstacle-free
Electro-optical (EO)	Visibility	Short range ($0.5-3$ km) Miniaturized Low expense
Infrared (IR) camera	Heat	Long range ($1-15$ km) Less affected by the weather Low accuracy
Acoustic receiver	Acoustic signal	Extremely low detection range (<0.2 km) Miniaturized Low accuracy
Hybrid	Mixed features	Combination of different technologies

Table I-3: *Interdiction methods.*

Methods	Destructive	Definition
RF Jamming	Non-destructive	Effective for drones using unknown protocols Not effective for autonomous drones Instant procedure
GNSS Jamming	Non-destructive	Can effect nearby friendly drones Available for follow-up investigation Instant procedure
Spoofing	Non-destructive	Wide availability Difficult to control Includes autonomous and manual flight
Geofencing	Non-destructive	Only available for communicable drones Simultaneous response Easy to extend
Capture	Non-Destructive	Difficult to target and hit Possible damage after Available for follow-up investigation
Laser	Destructive	Long range Confirmatory destruction High expense
High Power Microwave	Destructive	Long range Confirmatory destruction High expense
Anti-aircraft weapons	Destructive	Long range Confirmatory destruction High expense
Collision Drone	Destructive	Hard to target and hit Possible ground hit Low cost

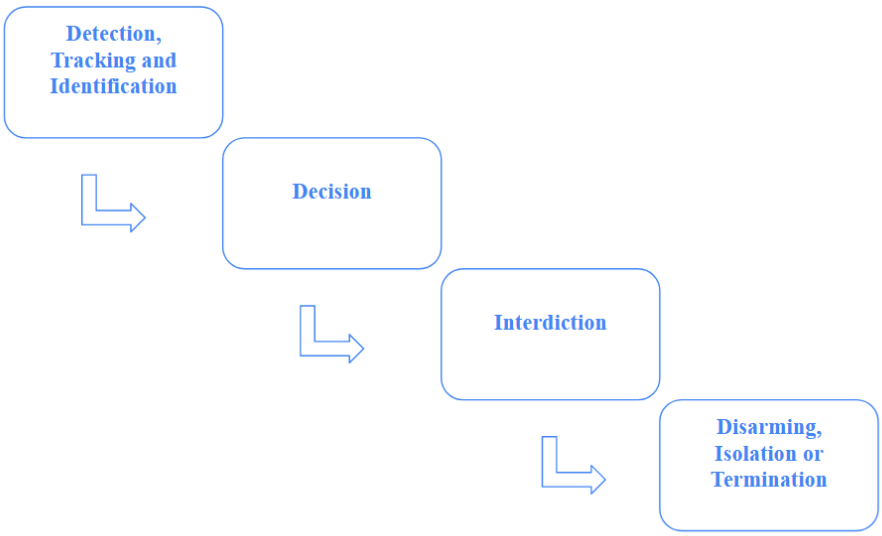


Figure I-3: *Steps to Counter-Drone*

I.3.1 Flourishing Drone Market

Increasing use of commercially available drones and the capabilities of them are posing a threat to safety of skies if they are misused. It is expected that the unmanned aerial Vehicles(UAV), also known as drones, will scale their flights and their operations beyond visual line of sight (BVLOS) will cover most of the air traffic by 2035 Sesar-JU (2023a). However, this increase on the number of drones in the airspace worldwide can increase the risk of misusing drones. To allow drones to operate in low-level airspace, U-space project is initiated by the European Commission. U-space is a set of new services relying on a high level of digitalisation and automation of functions and specific procedures designed to support safe, efficient and secure access to airspace for large numbers of drones Sesar-JU (2023b). Figure I-4 shows airspace volumes from U-space and the airspace is partitioned in X (low-risk), Y (medium-risk) and Z (high-risk) airspace Barrado *et al.* (2020). Airspace X has basic requirements from the operator, the pilot, and the drone. In airspace X, the pilot remains responsible for seperation at all times and visual-line-of-sight (VLOS) operations are allowed. Airspace Y requires an approved flight plan and VLOS and beyond-visual-line-of-sight BVLOS flight operations are allowed. Airspace Z has higher density operations than airspace Y and it also requires an approved operation plan.

The 2016 European Drones Outlook Study SESAR (2016) forecasts a promising economical growth fostered by the emerging drone market. Unmanned aircraft will be part of everyday life in most of the economic sectors, but will have a greater impact on air travel, utilities, entertainment and media, logistics, and agriculture. Indeed, the number of drones flying in the European airspace is expected to increase from a few thousand to several hundred thousand by 2050, most notably in government and commercial activities. The annual economic benefit could exceed EUR 10 billion by 2035 in Europe and create 100,000 new direct jobs to support drone-related operations. An example of this growth is illustrated by the agricultural sector, where authors estimate that 150,000 drones will be operated by 2035. The same is true in the fields of utilities and security, where around 60,000 unmanned aircraft will be used to assist in natural disaster management or traffic control, among other tasks.

A summary of the drone demand outlook per mission type by 2025 and 2050 is presented in Figure I-5 by SESAR JU Undertaking *et al.* (2017). In this figure. drones are divided into two parts such as specific drones and certified drones under EASA framework. Specific drones are drones which are below 25 kilograms and flying near or below 150 meters. Certified drones are

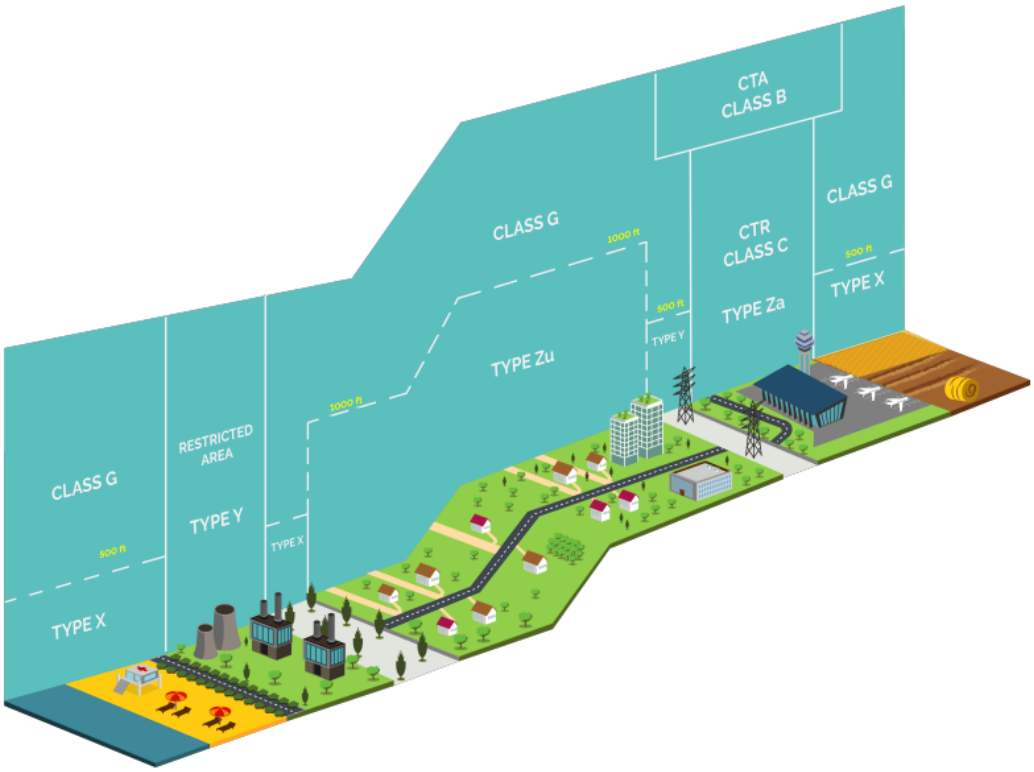


Figure I-4: *Example of airspace volumes.* Barrado et al. *(2020)*

used for drones flying well above 150 meters. beyond visual line of sight (BVLOS) with light load includes agriculture spraying and delivery drones. In addition, BVLOS surveying includes more automatic long range surveillance drones such as centralized police drones, agriculture remote sensing, monitoring of pipeline , power-lines and railway. Localized visual line of sight (VLOS) surveying represent inspections for energy sites (solar,oli & gas, etc.), telecommunication towers , mining & construction sites, etc., and in-vehicle police & fire response units. It is seen that the majority of potential demand is for drones expected to perform BVLOS missions.

Moreover, in figure I-6 provided by SESAR JU Undertaking *et al.* (2017), it is shown that the forecasted economic impact of drones by 2035 and 2050 in different sectors such as agriculture, energy, public safety and security, delivery, mobility and other sectors. Many businesses will benefit from drones in terms of product, service, and other support capability values at-stake for European demand.

I.3.2 Drone incidents

Under the latest drone regulation drones should not enter restricted zones, such as airports. Although there are many counter-drone solutions available in the literature, each solution targets special cases. In 2018, a drone caused a huge problem in Gatwick London Airport. The flights were canceled and around 140,000 passengers were affected BBC-UK (2019). However, methods such as geofencing, multiple radars with different detection ranges and a combination of radio-frequency sensors with visual detection sensors can be implemented to defend airports against unwanted drones. It is also highlighted that airfield operators must remain within the law when using disruptive technologies, and the risks to the wider community should be fully assessed and understood. Drone incidents happened in the World and in Europe are mapped by (Dedrone, 2022) and they are shown in Figures I-7 and I-8. In addition. Table I-4 presents the list of drone incidents reported in different parts of Europe. It is seen that airports are exposed to the drone in-

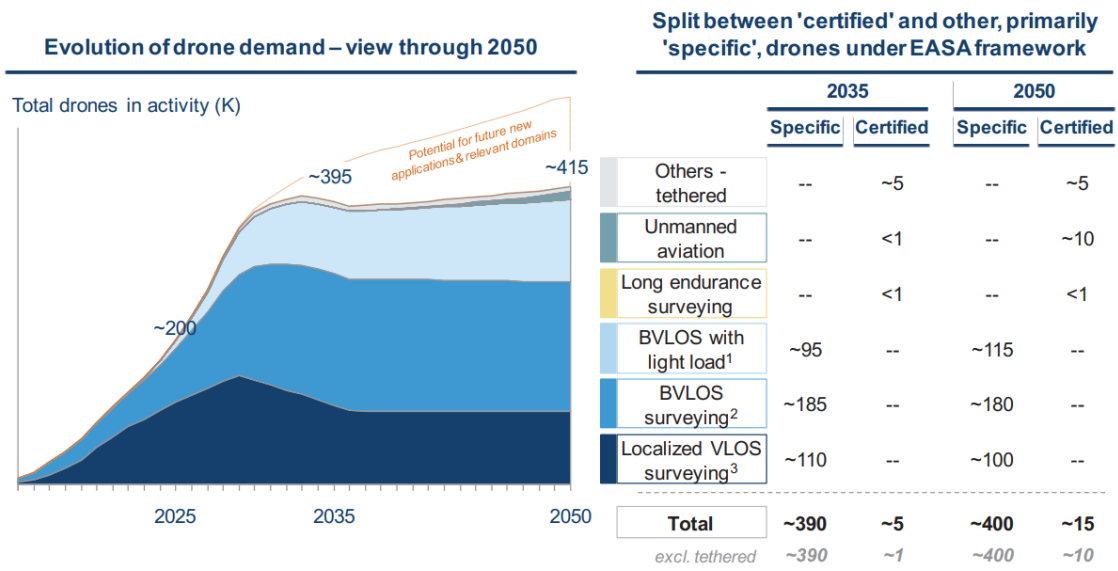


Figure I-5: *Drone demand outlook by type of mission.* Undertaking et al. (2017)

Figure I-6: *Industry view of forecasted economic impact.* Undertaking et al. (2017)

truders and results in delays, costs, and traffic rerouting for passengers, airlines, and the airports. However, intruder drones are not limited to airports and there are many locations and many ways to utilize drones. Law enforcement or first responders, stadiums, energy infrastructure, holiday locations etc. can be affected by drones.

I.4 Objectives of this PhD Thesis

The main objective of this PhD thesis is to demonstrate the capability of an artificial intelligence method, deep reinforcement learning, to counter drones and provide safe integration of drones into the public areas. Currently drones must be supervised by a human, but many of their phases are completely automated. Recent advances in artificial intelligence and advance flight control

Table I-4: *Published drone incidents in Europe.*

Incident	Place	Location	Date
easyJet plane comes within 10 feet of drone in close counter Sky-News (2022)	Airports	London United Kingdom	24/10/2022
Flights to Glasgow Airport forced to divert after suspected drone sighting GlasgowLive (2022)	Airports	Glasgow United Kingdom	23/09/2022
Norway's airport owner Avinor reports 50 drone incursions per month at Oslo airport alone NRK (2022)	Airports	Oslo Norway	13/09/2022
Five flights scheduled to land at Adolfo Suárez-Madrid Barajas airport were diverted and airport operations were disrupted for one hour due to the presence of unauthorized drones 20minutos (2022)	Airports	Madrid Spain	29/08/2022
Several sightings' of drone flown illegally near city airport Stv-News (2022)	Airports	Aberdeen United Kingdom	06/07/2022
Drones spy on private homes of Borussia Dortmund football stars Bild (2022)	Law Enforcement/ First Responders	Dortmund Germany	09/07/2022
Mossos report man for flying a drone over Barcelona without permission Elperiódico (2022)	Law Enforcement/ First Responders	Barcelona Spain	27/06/2022
Glastonbury: Police detect illegal drone flight over festival Thefestivals (2022)	Law Enforcement/ First Responders	Glastonburry United Kingdom	25/06/2022
Drone crash during Bayern training Kronen-Zeitung (2021)	Stadiums	Munich Germany	31/03/2021
Spanish league responds to Kicker (2021) drone incident in Bilbao	Stadiums	Bilbao Spain	26/03/2021
Swedish Security Service investigates Reuters (2022) drones at three nuclear plants	Energy/Utilities	Oskarshamn Sweden	14/01/2021
Loss of control following bird attack GOV.UK (2020)	Private/ Non-Coporate	Stranraer United Kingdom	08/10/2020

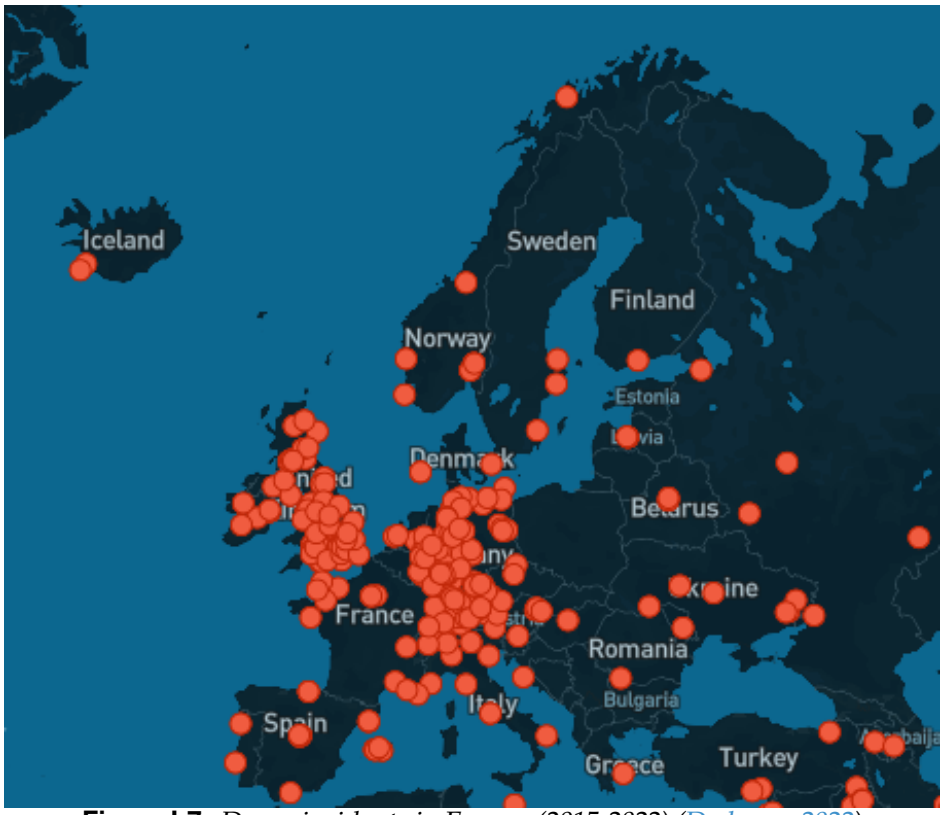


Figure I-7: *Drone incidents in Europe (2015-2022) (Dedrone, 2022)*

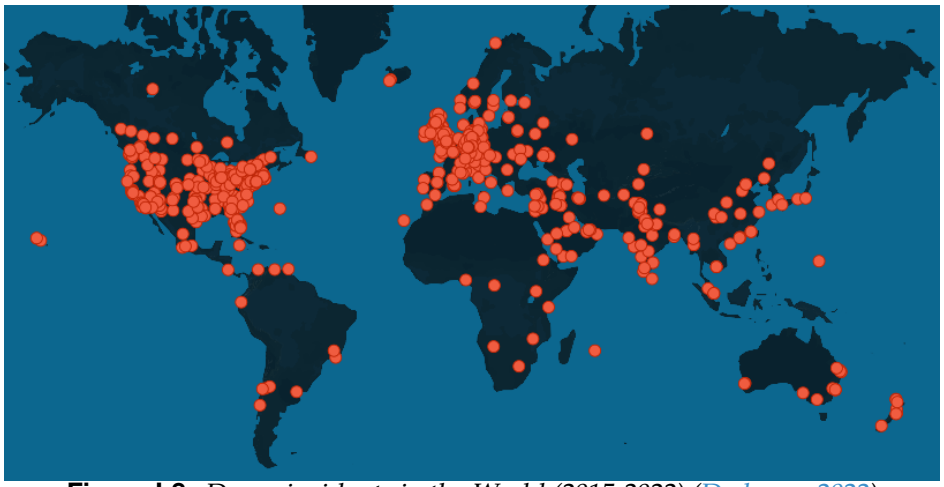


Figure I-8: *Drone incidents in the World (2015-2022) (Dedrone, 2022)*

techniques present new opportunities to fully automate the drones. This research has a futuristic view and thus addresses the long-term evolution of these flying vehicles.

The specific objectives of this PhD thesis can be outlined as follows:

- Create a deep reinforcement learning model that enables a drone to navigate through an urban environment without crashing on any stationary and non-stationary obstacles such as houses, trees, cars, electric wires, drones etc. and geofences which are virtual walls. It is aimed that the learner drone (the agent) will follow the other drone in the environment and crash on it. This idea is under a subject of counter drone technology which is used to detect the unmanned aircraft and take action about it. In this PhD thesis, there will be the learner drone and the random drones which move randomly in the environment. In other words,

there will be prey and predator relationship between the learner drone and the target drone which is one of the random drones in the environment. The purpose of the learner drone is to crash on the target drone.

- Investigate and develop a drone detection method to implement in counter-drone systems. Counter-drone systems include different mechanisms to work together. Drone detection and identification are the first phase of countering a drone. In this PhD thesis, state of the art object detection algorithm Redmon *et al.* (2016) is adapted to develop a drone detection model. Drone detection model is used as an input parameter of deep reinforcement learning algorithm neural network.

- Propose a method by deep reinforcement learning to counter a drone in a 2D space in an environment. The drone learns to navigate in a geofenced environment and heads towards the target drone. However, the actions are only in a 2D space such as moving forward and yawing left and right. In other words, the target drone is assumed to be at the same altitude as the learner drone which is trying to catch the target.

- Investigate further the most challenging counter-drone problems and develop a deep reinforcement learning model to counter a drone within a 3D space in a certain environment. A deep reinforcement learning method supported a drone detection model proposed so that the drone can catch the target drone in a 3D space. The learner drone is not only moving in a 2D space but also changing altitudes to eliminate the target drone. Optimizing both the time to interact with the target and the actions performed by the counter-drone system is essential for successful counter-drone operations in a 3D space. Thus, counter-drone system needs to be able to provide fast and accurate responses to neutralize the target.

- Design and implement transfer learning algorithm to increase the speed of learning progress of the drone. Filtering algorithm applied during transfer learning. This consists of preprocessing the previous experiences and eliminating those considered as bad experiences. Transfer learning is an important part of deep reinforcement learning training sessions when countering a drone in a 3D space. The time to train a drone is reduced dramatically and drone learns faster and in an efficient way.

- Propose graphical methods to present the explainability of deep reinforcement learning. The figures which represent the rewards, drone locations, crash positions and the action distribution during training and testing are analyzed and compared with different scenarios and parameters. In other words, the agent behavior is observed and the modifications are done accordingly in training and testing sessions.

I.5 Scope and Limitations of this PhD Thesis

To achieve the objectives of this PhD thesis, several assumptions have been made. The limitations that define its scope:

- This thesis is limited to simulation and the real flight tests are not performed due to their associated costs. Nevertheless, open source flight simulator used in this PhD thesis is very realistic and aircraft performance parameters can be adjusted to different vehicles. The interactions between the simulation and DRL models have been made without any human interactions and the agent drone is fully-autonomous.

- It is assumed that the solution proposed in this PhD thesis is part of a full solution where real-time position information of malicious drones is acquired through external means and

communicated to the counter drone. The related target data such as the target drone attributes and positions in the environment are already available thanks to the flight simulation used to train and test the deep reinforcement learning methods.

- None of the drones involved in the tests are subject to traffic management separation services, neither ATM nor U-space.
- The environmental disturbances such as the weather effects (wind, wind gust, wind shear) around the agent drone are not considered in the simulations during training and testing sessions. The drone operates normally without expecting any technical errors such as propeller failures or communication problems. The aerodynamic changes that occur during the drone's flight are not considered, and instead, the default conditions provided by the simulator are used. For example, as a drone agent changes speed or altitude, its aerodynamic characteristics, such as lift and drag, will also change. In other words, all the experimental conditions are considered as normal and there are no disturbances other than aerodynamic effects during take-off and maneuvers of the agent drone.

I.6 Outline of this PhD Thesis

It is worth noting that a broad state-of-the-art of the main topics addressed in this PhD thesis has been presented before. A deep and more detail state-of-the-art for each subject is included at the beginning of the chapter that addresses it. The present thesis is organized in eight chapters, which are summarized as follows:

- **Chapter** II presents the reinforcement learning fundamental theory and methods to establish core concepts of deep reinforcement learning. Drone detection for counter drone system main concepts are also introduced to be used in the later implementations. In addition, previous studies related to drone navigation and obstacle avoidance as well as counter-drone solutions are addressed.
- **Chapter** III gives the details of experimental setup. The setup includes the software framework which is used to communicate between the flight simulation and the deep reinforcement learning models. Moreover, the technical details of simulation and the drone used in the learning process are also explained in detail. In addition, it is also presented the drone detection model used in deep reinforcement learning models. The details of the drone detection model and the auto-labeling images which are used during training and testing of the detection model are introduced. The communication between the drone detection model and the deep reinforcement learning main program is also presented.
- **Chapter** IV proposes deep reinforcement learning model to navigate the drone in the environment without crashing any stationary and non-stationary obstacles.
- **Chapter** V proposes deep reinforcement learning models for countering a drone in 2D space. in this chapter, neural network details of the deep reinforcement learning models, the agent actions, the agent state and the reward function are presented.
- **Chapter** VI presents the deep reinforcement learning models for countering a drone in a challenging 3D space. In addition to the details of deep reinforcement learning models, the analysis of different scenarios are also discussed to present the explainability of deep reinforcement learning methods.
- **Chapter** VII analyzes the performance of human pilots, direct solution method, and deep reinforcement learning methods presented in Chapters V and VI.

- **Chapter VIII** presents conclusions of this thesis. Achievements of the work performed during writing the thesis are discussed. Potential future work is also described in details.

The significant problems we have cannot be solved at the same level of thinking with which we created them.

— Albert Einstein

The only stupid question is the one you were afraid to ask but never did.

— Rich Sutton

II

State of the Art

Artificial intelligence (AI) is the simulation of human intelligence in machines that are designed to think and act like humans. AI and reinforcement learning (RL) are two closely related fields that have greatly impacted each other's development and application. RL is a subfield of machine learning that focuses on how an agent can learn to make decisions in an environment by performing actions and receiving rewards. RL has proven to be a powerful approach for solving complex decision-making problems and has been successfully applied in various domains, including robotics, gaming, and finance. AI, on the other hand, provides the infrastructure for building complex systems that can learn from data and make decisions in real-world scenarios. When combined, AI and RL have the potential to create intelligent systems that can adapt to changing environments and make decisions based on learned experience. Deep reinforcement learning, which combines deep learning techniques with reinforcement learning, has achieved breakthrough results in areas such as game playing, where agents have surpassed human-level performance. In this chapter, deep neural networks, the fundamentals of reinforcement learning, deep reinforcement learning methods used in this PhD thesis are presented. In addition, this chapter presents state of the art drone detection methods for counter drone systems, as well as previous works on drone navigation, obstacle avoidance, and counter drone techniques in both 2D and 3D spaces.

II.1 Background

This section introduces the concept of deep neural networks and it explains the key principles of reinforcement learning. Also it provides a basis for understanding the following chapters and the

use of deep reinforcement learning for the thesis context.

II.1.1 Reinforcement Learning

Reinforcement learning (RL) is an approach to artificial intelligence inspired by a human's way of learning, similar to what a baby experiences when learning how to walk. In RL, the agent is a term used for any kind of object such as a drone, a robot, or an algorithm that receives inputs and react to them with outputs. The agent gains knowledge by continuously interacting with the environment to optimize its behavior. Reinforcement learning is a complex and challenging field, and the design of an effective reinforcement learning algorithm often requires a deep understanding of the problem and careful tuning of hyperparameters. However, it has shown great potential in solving a wide range of problems and has received increasing attention in recent years.

Some of the examples of RL are as follows:

- **Walking robot**: The agent can control a walking robot. Haarnoja *et al.* (2018a)
- **PC gamer**: The agent can be the program controlling a game character in Pac-Man game. Gnanasekaran *et al.* (2017)
- **Board gamer**: The agent can play a board game against human players or another AI models. Silver *et al.* (2018)
- **Smart home**: The agent can control the temperature in a house during the day. Yang *et al.* (2021)
- **Financial advisor**: The agent can observe stock market prices and decide how much to buy or sell. Cong *et al.* (2021)

II.1.2 Concepts and the terminology

II.1.2.1 The agent-environment interactions

In RL, agent makes a decision and takes an action. The agent interacts with the environment. The environment works in a similar way like the agent by reacting to inputs with outputs. The environment provides states, which is information about the current status of the environment. Each action updates the environment and its state. Finally, a reward is submitted by the environment informing about the benefit of using the action in that moment. The objective of the agent is to maximize the final reward value. The interaction between the agent and the environment is shown in Figure II-1. State is represented as S_t and the State Space is represented as S. The interaction between the agent and the environment is in discrete time steps t. Action and Action Space are represented as A_t and $A(S_t)$ respectively. Reward values are updated in each time, R_{t+1}, and a new state becomes S_{t+1}.

II.1.2.2 States and observations

A state s, the current status of the environment, is available in the environment and there is no information hidden from the state. However, an observation o is the partial description of the state and observations can have less information. For example, fully observable environment means that the agent can observe the complete state of the environment. On the other hand, if the agent can have only the part of the the observation, the environment is called partially observed. A state can be represented in different ways. For instance, an agent such as walking robot can be in a state by its joint angles and velocities. A state of a drone agent can be its positions, velocities,

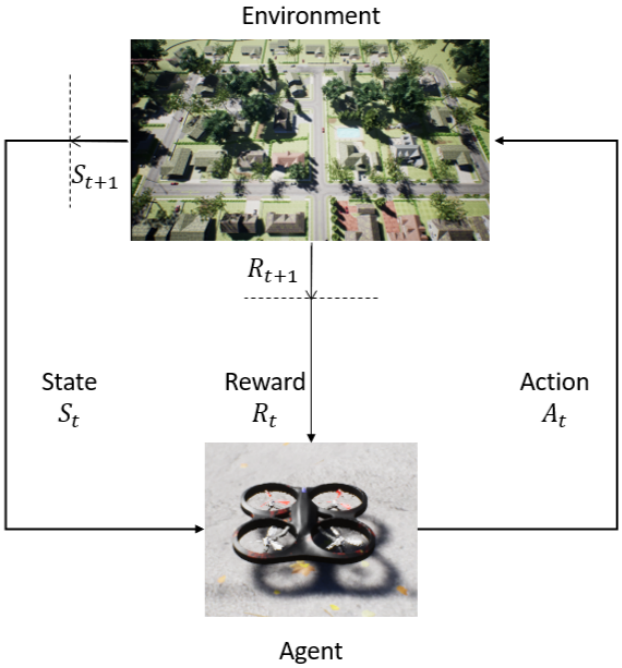


Figure II-1: *The agent-environment interaction in reinforcement learning.*

or if the drone has optical sensor, the state can be a visual representation by the RGB matrix of its pixel values.

II.1.2.3 Actions

In reinforcement learning, the agent is able to use different kind of actions depending on the environments. In other words, different environments provide different types of actions. The set of actions available in an environment is called the action space. There are two types of action spaces: discrete action space and continuous action space. In discrete action spaces, there is only a finite number of movements available to the agent in the environment such as in a Go game Britannica (2022) environment. However, in continuous action spaces, real valued vectors are the actions available to the agent. For example, the agent that controls a robot in an environment has continuous action spaces.

II.1.2.4 Reward

After an agent selects an action, the environment provides a reward. Reward is usually a scalar number which tells the agent how good or bad the current world state is. The reward function R is one of the important element in reinforcement learning. Reward function is presented in Equation II.1 and R depends on the current state, the action, and the next state of the agent. Reward function can be simplified as current state $r_t = R(s_t)$ or state action pair $R(s_t, a_t)$.

$$r_t = R(s_t, a_t, s_{t+1}) \tag{II.1}$$

The goal of the agent is to maximize its cumulative reward over time and it is called return OpenAI (2022). The reward return can be without discounts and it is called finite-horizon undiscounted return. This return is just a sum of the rewards obtained in a fixed window of steps and shown in Equation II.2. In this equation τ is a sequence of states and actions $(s_0, a_0, s_1, a_1, \dots)$. It is also frequently called episodes.

$$R(\tau) = \sum_{t=0}^{T} r_t \tag{II.2}$$

Return can also be discounted and it is called infinite-horizon discounted return presented in Equation II.3. In this type of return, the rewards obtained by the agent is summed up and they are discounted to determine how much the agent cares about rewards in the distant future relative to those in the immediate future.

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{II.3}$$

where $\gamma \in [0, 1]$ is the discount factor. The discount factor γ determines the importance of future rewards. A factor of 0 will make the agent short-sighted by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward.

II.1.2.5 Policies

In RL, states are mapped to the probability of the possible actions in each time step t and this is called policy denoted π. The policy is chosen to maximize the sum of the discounted rewards over time shown in Equation (II.4). This means maximizing not the immediate rewards R_{t+1}, but the expected discounted return G_t Sutton & Barto (1998).

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{II.4}$$

The agent's long term rewards can be maximized by estimating their expected discounted returns G_t and using different policies. A policy can be deterministic or stochastic.

- **Deterministic Policies:** A deterministic policy is a function from the set of states of the environment, S, to the set of actions, A. For example, in a Gridworld, the cells of the grid correspond to the set of states of the environment and the set of actions which is composed of four actions: north, south, east, and west. Given a state $s \in S$, $\pi(s)$ with probability 1, deterministically cause the agent to move one cell in the respective direction on the grid.
- **Stochastic Policies:** A stochastic policy is a family of conditional probability distributions from the set of states, S, to the set of actions, A. For example, the probability of taking an action number 1 from state S is 10%, 20% for taking action number 2, and there are 70% chance of taking action number 3.

II.1.2.6 The reinforcement learning problem

The reinforcement learning problem can be defined by assuming the stochastic environment transitions and the policy OpenAI (2022). RL aims to select a policy that maximizes expected return. The first step towards determining expected return is to look at probability distributions over the trajectory. The probability of a T-step trajectory becomes:

$$P(\tau|\pi) = \rho(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{II.5}$$

The expected return denoted $J(\pi)$ is:

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)] \tag{II.6}$$

So, the optimization problem in RL can be expressed by an optimal policy:

$$\pi^* = \underset{\pi}{argmax} J(\pi) \tag{II.7}$$

To understand the basic terms in reinforcement learning for the rest of the sections, glossary terms Google (2023b) are listed as follows:

- **Agent** is the entity that interacts with the environment and it uses a policy to maximize the expected return.
- **Environment** is the world that contains the agent and it allows the agent to observe that world's state.
- **Policy** is an agent's probabilistic mapping from states to actions.
- **Action** is the mechanism taken by the agent based on the state of the environment. The agent chooses the action by using the policy.
- **Reward** is the feedback given to the agent by the environment for taking an action in a state.
- **Return** is the sum of all rewards that the agent expects to receive.
- **State** is the parameter value that describe the current situation returned by the environment.
- **Episode** is the agent's repeated attempts to learn the environment. It is in between an initial state and a terminal-state.
- **Trajectory** is a sequence of tuples that represent a sequence of state transitions of the agent, where each tuple corresponds to the state, action, reward, and next state for a given state transition.
- **Annealing** is the amount of random exploration to increase the speed of learning progress.
- **Exploration** is a situation that the agent gathers information about the environment.
- **Exploitation** is that the agent exploits its knowledge to learn the environment.
- **Time Limit** is the maximum time that the agent is allowed in an episode to learn the environment.

The following sections, inspired by the work of Sutton & Barto (1998) and the work of Szepesvári (2010), provide the fundamental theory behind the RL agent.

II.1.3 Markov decision processes

The general RL problems are modeled according to the dynamic programming(DP) which formulates the RL process as a Markov decision process (MDP). An MDP is defined as a discrete time stochastic control framework which dedicates to address sequential decision making problems where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards. Earlier the agent-environment is shown in Figure II-1. The learner and decision maker is called the agent and it interacts with is the environment. The agent and the

environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, ...$ These interactions are the agent selecting actions and the environment responses according to these actions and presenting new situations to the agent. The environment also outputs rewards which the agent aims to maximize over time by selecting actions.

Specifically, MDPs can be defined as a triplet consisting of (S, A, R) and each component is explained as follows:

At each time step t,

- the environment's state $S_t \in S$, finite set of states available in the environment
- the agent's action $A_t \in A(s)$, finite set of actions available and these actions for an agent often relies upon the state s, $A(s)$
- the reward $R_{t+1} \in R$, numerical reward after the agent finds itself in a new state S_{t+1}

Given any particular state and action, the probability of each possible pair of next state and reward for all $s', s \in S, r \in R$ and $a \in A(s)$:

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{II.8}$$

$p(s', r|s, a)$ is the probability denoted by p of reaching state s' and receiving reward r given that we were in state s and took action a. $Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$ represents the same probability in a more descriptive form.

The dynamics function $p : S \times R \times S \times A \to [0, 1]$ is an ordinary deterministic function of four arguments. In Equation II.8, p defines the dynamic of the MDP. In MDP, the probabilities given by this equation completely characterize the environment's dynamics. The state must include information about all aspects of the agent-environment interaction which contributes to the future and then the state can be said to have the Markov property. From all these arguments, the state transition probabilities can also be calculated as follows:

$$p(s'|s, a) \doteq Pr\{S_t = s'|S_{t-1} = s', A_{t-1} = a\} = \sum_{r \in R} p(s', r|s, a) \tag{II.9}$$

The MDP framework can be implemented in many different problems to some extent. For example, the time steps can be referred to arbitrary successive stages of decision making and acting. Also, the actions can be categorized as high level control such as a decision to pursue a post graduate level degree, and low level control such as inputs to a industry robot. Moreover, states can be defined in many different forms. For instance, the states can be directly from the sensor readings which are low level sensations, or they can be more high level such as symbolic descriptions of objects in a room. They can be based on past readings or they can be entirely subjective. For example, the state of the agent can be in a situation in which the agent does not know where an object is or the agent can be in a state of clearly defined sense.

II.1.3.1 Value functions

A policy is described as a mapping from states to probabilities of selecting each possible action. The value function is a measure of a state in terms of how good it is for the agent to perform a given action in a particular state. The term "How good" is in terms of future rewards or expected return. Formally, value functions are defined with respect to the policies.

The value function of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter. Formally a value function is defined as:

$$v_\pi(s) \doteq E_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \tag{II.10}$$

where E_π denotes the expected value of a random variable and t represents the current time step. In other words, v_π is called the state-value function for policy π. Equation II.10 defines the state-value function as the expected return when starting in a particular state s and following a specific policy π. It considers all possible future rewards, weighted by their discount factor γ, to estimate the expected value of the total return.

Also, action-value function for policy π for taking action a in state s under policy π can be presented as:

$$q_\pi(s, a) \doteq E_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{II.11}$$

A fundamental property of value functions satisfy recursive relationships between the value of a state and the value of its successor states induced by the Markov property. For any policy π and any state s, this relationship is expressed for $v_\pi(s)$ as:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in S \tag{II.12}$$

where the actions a, state s, next state s', and the rewards r.

Equation II.12 is the Bellman equation for v_π which expresses a relationship between the value of a state and the values of its successor states. Bellman equation averages over all the possibilities and weights each of them by the probability of occurrence. Figure II-2 shows that the value of the start state must equal the (discounted) value of its expected next state plus the reward. Each open circle represents a state, the arrows represent the reward and each solid circle represents a state–action pair. The value function v_π is the unique solution to its Bellman equation.

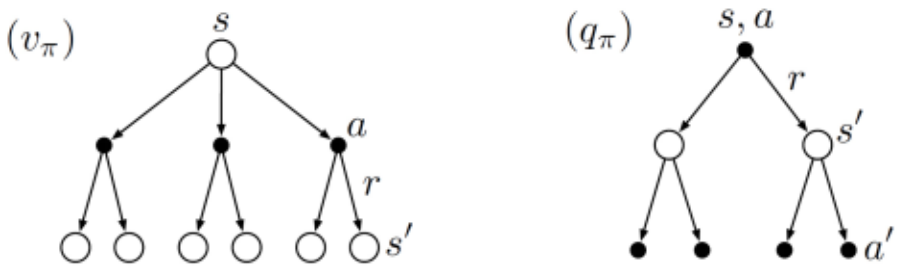


Figure II-2: *The representation diagram of value function and action-value function.Sutton & Barto (1998)*

II.1.3.2 The optimal value functions

A policy in reinforcement learning task aims to achieve the most reward over the long run. There is always at least one policy that is better than or equal to all other policies. This is an optimal policy. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states.

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \forall s \in S \tag{II.13}$$

The optimal policy is denoted π^* and the optimal state-value function is denoted v_* and can be defined as:

$$v_* \doteq \max_{\pi} v_\pi(s) \forall s \in S \tag{II.14}$$

Optimal policies also share the same optimal action-value function, denoted q_* and it is defined as:

$$q_*(s,a) \doteq \max_{\pi} q_\pi(s,a) \forall s \in S \quad and \quad a \in A(s) \tag{II.15}$$

In addition, the self-consistency condition given by the Bellman equation for state values, II.12 must be satisfied because $v_*(s)$ is the value function for a policy. The value of a state under optimal policy must equal the expected return for the best action from that state. This is represented in Bellman optimality equation and Equation II.12 can be rewritten as:

$$\begin{aligned}
v_*(s) &= \max_{a \in A(s)} q_{\pi^*}(s,a) && \text{(II.16a)}\\
&= \max_a \mathbb{E}_{\pi^*}[G_t | S_t = s, A_t = a] && \text{(II.16b)}\\
&= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] && \text{(II.16c)}\\
&= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] && \text{(II.16d)}\\
&= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_*(s')] && \text{(II.16e)}
\end{aligned}$$

Equations II.16d and II.16e are the two forms of the Bellman optimality equations for v_*. For the optimal action-value function q_* the Bellman optimality equations are:

$$\begin{aligned}
v_*(s) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] && \text{(II.17a)}\\
&= \sum_{s',r} p(s', r | s, a)[r + \gamma \max_{a'} q_*(s', a')] && \text{(II.17b)}
\end{aligned}$$

Bellman optimality equations for v_* and q_* are summarized in backup diagrams in Figure II-3. This is the same diagrams for v_π and q_π presented earlier in Figure II-2. However, the difference is that arcs have been added at the agent's choice points to represent that the maximum over that choice is taken rather than the expected value given some policy.

In this sense, a one-step search is performed and for optimal value functions, the actions that appear best after that one-step search will be the optimal actions. Therefore any policy that behaves greedy considering v_* is an optimal policy. Notably, this consideration of short-term consequences is also optimal in the long-term, since v_* already takes into account the reward consequences of all possible future behavior. Thereby, long-term optimal actions are brought down to a one-step look-ahead.

Moreover, an important branching point in finite MDPs is whether the agent learns a model of the environment. Finite MDPs are categorized according to model-based and model-free environments. In Model-based problem, states S, actions A, state-transition probabilities $p(s'|s,a)$ and immediate rewards $r(s,a,s')$ are fully available. In this problem, the agent is allowed to plan by thinking ahead and the agent can see what would happen for possible some choices and it can decide between its options. On the other hand, in model-free problem, there is no prior information about the environment and thus the agent needs to learn it and gain some experiences.

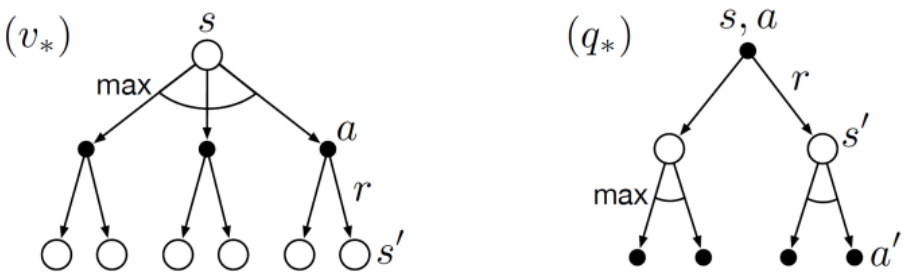


Figure II-3: *The representation diagram of optimal value and action-value functions.*Sutton & Barto (1998)

II.1.3.3 Model-based problem: Dynamic Programming

Dynamic programming (DP) is a collection of algorithms that can be used to find the optimal policies without an actual agent/environment interaction since the perfect model of the environment is already given. Classical DP algorithms have limited usage in RL because of the assumption of a perfect model and the great computational expense.

The key point in DP is to search for good policies by organizing value functions. As discussed before in section II.1.3.2, optimal policies can be obtained after the optimal value functions are found.

The policy iteration is the most important method in dynamic programming and it has two phases: policy evaluation (prediction) and policy improvement. In policy evaluation, the state-value function v_π for an arbitrary policy π is computed. This is also called as the prediction problem. In this case the Bellman equation II.12 for v_π is used as an update rule at each iteration k:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \forall s \in S \tag{II.18}$$

where $\pi(a|s)$ is the probability of taking action a in state s under policy π and t is discrete time step.

Equation II.18 is called iterative policy evaluation and it applies to same operation to each state s such that at the first iteration, the states immediate reward R_{t+1} is known and at the second iteration the method knows the state values and the expected immediate rewards along all the one-step transitions possible. On the other hand, the new policy π_{t+1} is estimated with the new value function by selecting the action which is the best according to $q_\pi(s, a)$ and the new greed policy π' becomes:

$$\pi_{'} = \underset{a}{argmax} \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a] \tag{II.19}$$

In this equation II.19, $\underset{a}{argmax}$ denotes the value of a which maximizes the expression that follows.

The Bellman optimality equation is satisfied if the optimal value function and optimal policy are found. The value function stabilizes only when it is consistent with the current policy, and the policy stabilizes only when it is greedy with respect to the current value function. Finally, policy and value function stabilize only when a policy has been found that is greedy with respect to its own evaluation function. This process is called as generalized policy iteration (GPI) and it is illustrated in Figure II-4.

The policy evaluation and improvement processes in GPI can also be represented in terms of two constraints or goals. For example, in Figure II-5, each process drives the value function or

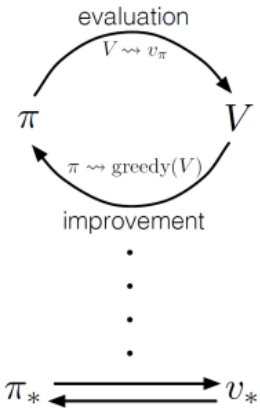


Figure II-4: *The representation diagram of generalized policy iteration (GPI).Sutton & Barto (1998)*

policy toward one of the lines representing a solution to one of the two goals. The goals interact because the two lines are not orthogonal. Driving directly toward one goal causes some movement away from the other goal. Finally, optimality can be found. The arrows in this figure correspond to the behavior of policy iteration in that each takes the system all the way to achieving one of the two goals completely.

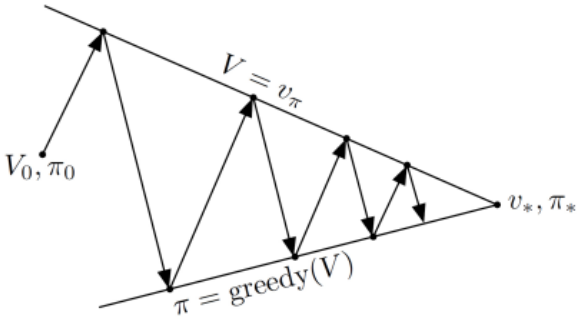


Figure II-5: *Representation diagram of generalized policy iterations (GPI).Sutton & Barto (1998)*

II.1.3.4 Model-free problem Applicable in Reinforcement Learning

The main downside of the dynamic programming is that it assumes the agent has complete knowledge of the environment. The agent can perform well with respect to the learned model but the performance can be sub-optimal in real environment. However, model-free methods enable to estimate value function and learn an optimal policy of a finite MDP without having a model available to the agent. These methods have the advantage of the sample efficiency from using a model. In this section, the learning methods for estimating value functions and discovering optimal policies are discussed and summarized on the basis of a textbook Sutton & Barto (1998) and lecture notes by Silver (2022).

- **Monte Carlo (MC) methods** can learn value functions and find optimal policies from experiences by sampling sequences of states, actions, and rewards from actual or simulated interaction with an environment. This method is a way of solving the RL problem based on averaging sample returns. Also, the environment is sampled with a number of episodes. It is assumed that experience is divided into episodes, and they eventually terminate regardless of the actions selected. If an episode is completed, value estimates and policies are changed.

In Figure II-6, it is shown a basis to estimate the value function after the terminal state. The return over the entire episode and the distribution of states encountered are used as backup. For example, Monte Carlo Tree Search can be implemented on a game called tic-tac-toe and it is represented in Figure II-7. Every state is represented as a board and it starts with an empty board. Arrows in the Figure II-7 represent a move which is a transition from one node to another and game ends in a terminal node.

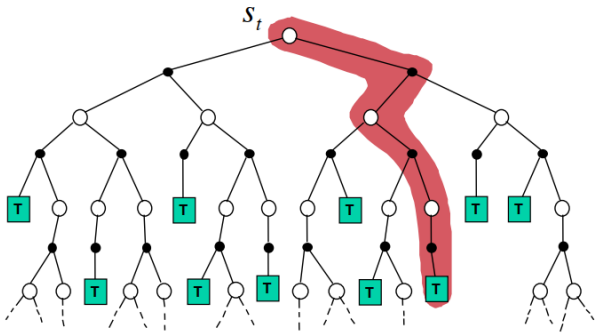


Figure II-6: *Monte Carlo methods backup diagram.Silver (2022)*

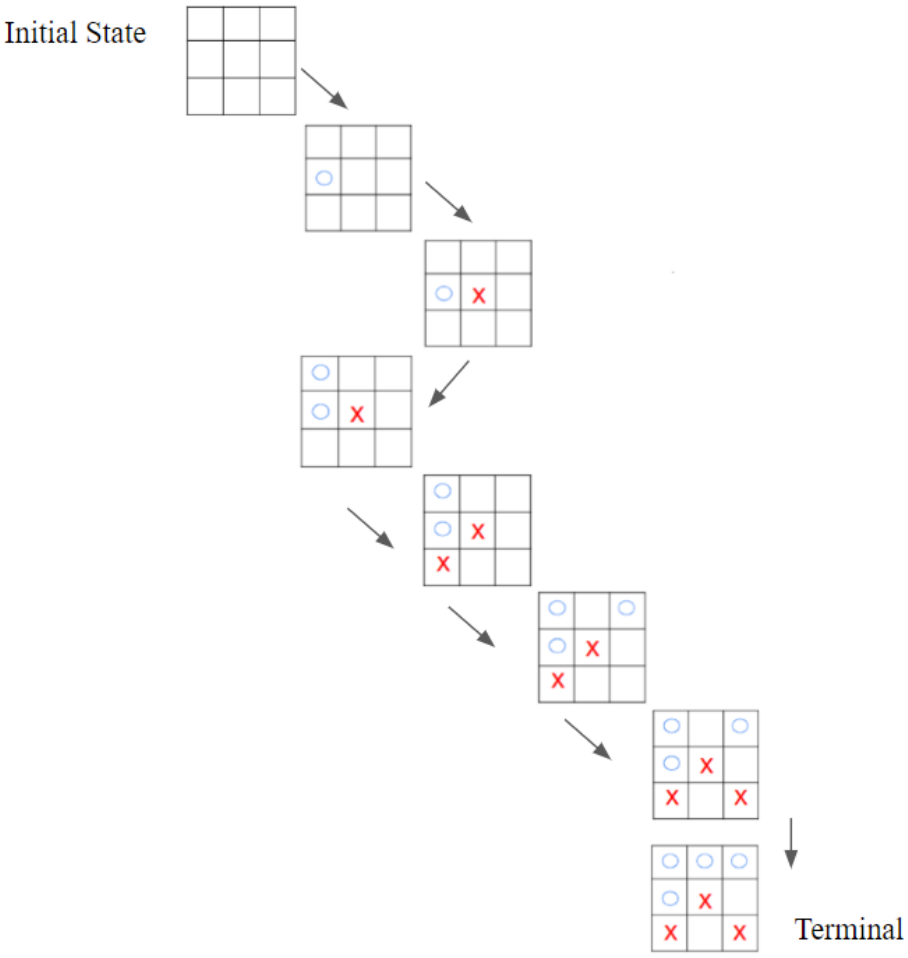


Figure II-7: *Monte Carlo Tree Search for Tic-Tac-Toe Game.*

The value functions v_π and q_π can be estimated from experiences rather than using a model to compute the value of each possible state by following policy π and maintaining an average, for each state encountered, of the actual returns G_t that have followed that specific

state. The average will then converge to the state's value, $v_\pi(s)$, as the number of times that specific state is encountered approaches infinity. In addition, separate averages are kept for each action taken in a specific state, then these averages will converge to the action values, $q_\pi(s, a)$. Equation II.20 expresses that the state-value function $V(S_t)$ can be updated toward the actual return G_t each time the state is encountered. In this equation a small positive fraction called the step-size parameter is denoted by α and it influences the rate of learning which is a positive scalar determining the size of the step Goodfellow *et al.* (2016b). The value functions are estimated by sampled returns without computing them. Monte Carlo methods do not update their value estimates on the basis of the value estimates of successor states but use real sampled experience. Therefore no bootstrapping is performed. In bootstrapping, updating estimates involves estimates and the idea refers to the process of estimating future values based on the agent's current estimates.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{II.20}$$

- **Temporal Difference (TD) Learning** is the core part of the reinforcement learning problem. TD learning includes the ideas presented in DP and MC methods. TD learning can learn from sampled experience without requiring an environment model. Also, TD methods update estimates based on other learned estimates without waiting for a final outcome like done in dynamic programming and thus they bootstrap. In other words, TD methods learn a guess from a guess. Equation II.21 presents that unlike MC methods, TD methods wait until the next time step and use an estimated return $R_{t+1} + \gamma V(S_{t+1})$. TD methods directly form a target and update it with the immediate reward R_{t+1} and the estimate for $V(S_{t+1})$.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{II.21}$$

In this equation, the error calculated between the estimated value and the better estimate based on the agent's immediate experience is called as temporal difference error, δ_t and expressed in equation II.22.

$$\delta_t = [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{II.22}$$

The difference between the Monte Carlo methods and Temporal difference learning methods are shown in backup diagram II-8. MC methods only work for episodic environments which terminate but TD methods can work with environments that do not terminate. MC methods require a larger number of sample episodes for their convergence than TD methods Sutton & Barto (1998).

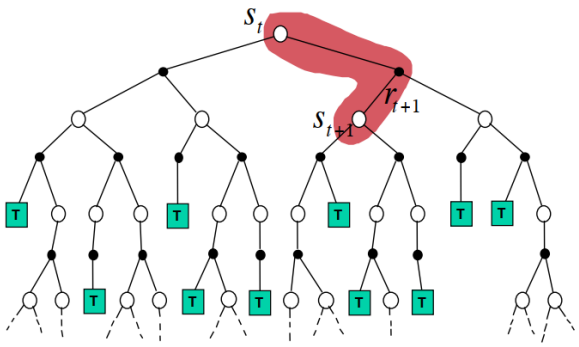


Figure II-8: *Temporal difference learning methods backup diagram.Silver (2022)*

- **Q-Learning** is the off-policy TD control algorithm. This algorithm is the early breakthroughs in reinforcement learning Watkins & Dayan (1992) and expressed in Equation II.27. In this equation, the learned action-value function, Q, directly approximates the optimal action-value function, q_π, which is no matter what the policy being followed. The term, Off-policy, is used to call this algorithm since the agent does not always have an exploration and the agent learns from experience. The agent in RL relies on its experience and the actions taken to maximize the cumulative reward while the good actions need to be discovered by trying the actions which are not selected before. The agent learning is in balance between what it already knows to obtain reward, exploit, and exploration which is to find possible better future action. To manage the amount of exploration globally, a simple but an effective behavior policy method is used and it is called as $\epsilon - greedy\ policy$. The parameter ϵ determines the randomness in action selections. In this method presented in II.23, memorization of exploration specific data is not necessary.

$$\mu(S_t) = \begin{cases} argmax_a Q(S_t, A) & \text{at probability } 1-\epsilon \\ \text{random } A & \text{at probability } \epsilon \end{cases} \tag{II.23}$$

Off-policy method has an approach that one can be learned about called target policy π and the other one behaviour policy μ is that the optimal policy is used to choose the actions. On the other hand, on-policy methods ensure the agent always have an exploring part and thus the best policy that still explores can be found. In the off-policy approach, the actual action A_t is chosen with respect to the behavior policy μ. On the other hand, Q-learning also considers the alternative successor action $A^{'}$, which would have been selected with respect to the target policy π. Accordingly, the action value for the starting state with action A_t is updated towards the alternative action and it is expressed in equation II.24.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A^{'}) - Q(S_t, A_t)] \tag{II.24}$$

Thus, The target policy π improves greedy with respect to $Q(s, a)$ and the behaviour policy μ improves with exploratory $\epsilon - greedy$ with respect to $Q(s, a)$.

$$\mu(S_{t+1}) = \max_a Q(S_{t+1}, a) \tag{II.25}$$

Then the estimated return, learning target y_t becomes:

$$y_t = R_{t+1} + \gamma Q(S_{t+1}, A^{'}) \quad = R_{t+1} + \gamma \underset{a}{argmax} Q(S_{t+1}, a) = R_{t+1} + \max_a \gamma Q(S_{t+1}, a) \tag{II.26a}$$

Finally the update of the TD control algorithm known as Q-learning can be expressed by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{II.27}$$

As discussed earlier in II.22, the TD error δ_t becomes:

$$\delta_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \tag{II.28}$$

The Q-learning method evaluates if the action took by the exploratory policy was better or worse than the target policy, which approximates the optimal action-value function q^*. Q has been shown to converge to q^* with probability 1 Sutton & Barto (1998).

The Q-learning algorithm is shown in II.1 in procedural form. The Q-learning reinforcement learning method can also be classified as an adaptive control algorithm that converges online to the optimal control solution for completely unknown systems Lewis *et al.* (2012).

Algorithm II.1: *Q-learning: An off-policy TD control algorithm*

Initialize $Q(s, a)$ randomly
Initialize discount factor γ
Initialize step-size parameter α
repeat(for each episode):
    Observe initial state S_1
    repeat(for each step of episode):
        Choose action A_t using policy derived from Q (e.g. ϵ-greedy)
        Take action A_t
        Observe reward R_{t+1} and new state S_{t+1}
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, a) - Q(S_t, A_t)]$
        $S_t \leftarrow S_{t+1}$
    until S_t is terminal
until end of learning

II.1.4 Deep Neural Networks

Tabular methods which are discussed in section II.1.3.4, such as Monte Carlo, temporal difference learning, and Q-learning, have limitations when compared to function approximation and deep neural networks. These limitations include slow convergence, issues with high-dimensional state and action spaces. These constraints are addressed in deep neural networks to support efficient scalability and handle complex environments. In tabular methods, every state has an entry $V(s)$ or every state-action pair is represented as $Q(S, A)$ and the value functions are presented in a lookup table for all the states. These methods are most suitable for games such as casino card game of blackjack, and tic-tac-toe which are the examples of environments with small size states and actions. However, these tabular methods might not be suitable for the complex environments with large state-action pairs. There are certain limitations which can challenge the tabular case. One of the important challenges is the memory. Also, online updates can be challenging since in tabular case, firstly the state must be located to find the value for that certain state. Moreover, the learning process can be slow because of large states. In this case, the value of each state is to be learned individually. Because of these challenges it is harder to find optimal policy with a limited data and in a limited time. For infinite markov decision problems, generalization becomes an important subject which can relate different states with previous ones. This term generalization in RL is called function approximation which takes data samples from the desired function $Q(s, a)$ and generalize from them to produce an approximation of the entire function. For instance, in supervised learning, labeled dataset is provided to the agent and the main goal is to obtain an approximation of a function given the examples. Deep neural networks(DNN) proved that they can outperform the other machine learning methods and become the core part in artificial intelligence developments Schmidhuber (2015). Recent developments in deep neural networks Mnih *et al.* (2015) Silver *et al.* (2017) show that superhuman performances can be achievable with an algorithm that learns superhuman proficiency in challenging domains. In the next sections, the summary of neural networks based on Goodfellow *et al.* (2016a) and Sutton & Barto (1998) is discussed. Neural network architecture, algorithms used, and finally training and testing the network are among the main topics covered in next sections.

II.1.4.1 Neural Networks

A neural network can be used to approximate a value function or a policy function and it can nonlinearly maps input vectors x to a category y by learning the value of the parameters θ and thus a mapping $y = f(x : \theta)$. Goodfellow *et al.* (2016a). This is accomplished by applying a sequence of parameterized layers to the input signal. For example, in equation II.29, the functions f^1, f^2, and

f^3 are composed together.

$$f(x;\theta) = f^1(f^2(f^3(x))) \tag{II.29}$$

The common architecture of concatenating the different layer parameter vectors is called feed forward neural network. Feed forward neural networks play a very important role in machine learning. For example, convolutional neural network (CNN), used for object recognition from photos, is a kind of feed forward neural network. More details will be given in the next section about the CNN. Figure II-9 shows a generic feedforward neural network with four input units, two output units, and two hidden layers which are the layers that are neither input nor output layers. A real-valued weight is associated with each link.

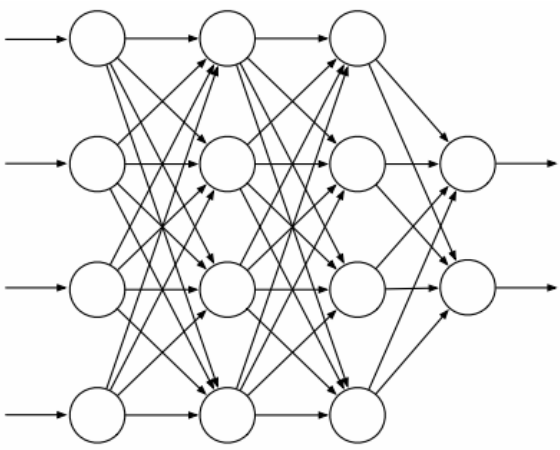


Figure II-9: *A generic feed forward neural network Sutton & Barto (1998).*

In this equation II.29, f^1 is called the first layer of the network, f^2 is called the second layer and son on. The overall length of these layers gives the depth of the model. The term "deep" comes from this terminology. The final layer of a feed forward neural network is called the output layer. In neural networks each layer f^i consists of neurons which are the application of an affine function to the input. If the affine function connects the input to all neurons, the neural network layer is called fully-connected layer and the linear operation is followed by a non-linear transformation g shown in equation II.30.

$$f^i(x) = g(W^i x + b) \tag{II.30}$$

In this equation, W and b represent the weights and the bias respectively. g is the activation function which inserts the non-linearity between the layers. The activation function can be S-shaped , or sigmoid functions such as the logistic function $f(x) = 1/(1 + e^{-x})$. The most famous activation function used in most feed forward neural networks is the rectified linear activation function and it is called RELU Nair & Hinton (2010) defined in equation II.31. After RELU is applied to the output of a linear transformation, it yields a nonlinear transformation. The function is a piecewise linear function with two linear pieces. Thus, Linear models are easy to be optimized with gradient-based methods.

$$g(W^i x + b) = \max(0, W^i x + b) \tag{II.31}$$

In this equation II.31, the nonlinear function is element-wise applied. The function consists of a simple max-operator that sets the particular neuron output to 0, if the value is below 0 and otherwise applies a linear output with slope 1. This is represented in Figure II-10.

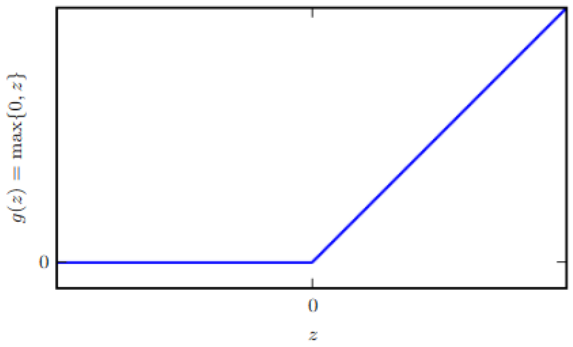


Figure II-10: *The rectified linear activation function Goodfellow et al. (2016a).*

II.1.4.2 Training and Testing the network

The main challenge of deep learning methods is to find the network parameters such as weights and biases. To optimize the parameters the loss function $J(\theta)$ is used a measure of how far off the prediction was to the target. Here θ is the network parameter. Thus, the purpose is to minimize the loss by means of pushing the prediction towards the target.

There are many optimization techniques available in the literature but the gradient descent algorithm is one of the most commonly used computational optimization technique. In this technique, the gradient of $J(\theta)$ is computed iteratively with respect to θ and then the loss related to each network parameter can be calculated. With this method, the magnitude and direction can be found by changing each parameter in order to minimize the loss. Gradient descent update rule is defined in equation II.32 where α is the specified learning rate of the network. Learning rate scales the size of the updates.

$$\theta \rightarrow \theta - \alpha \nabla_{\theta} J(\theta). \tag{II.32}$$

There exist optimizers for minimizing the loss such as stochastic gradient descent (SGD) LeCun *et al.* (2012) which utilizes the average gradient over a batch of randomly selected samples called mini-batch and Adam optimizer Kingma & Ba (2014). Adam optimization algorithm is an extension of the SGD and it is a popular optimization algorithm for neural networks. Adam optimization is short for Adaptive momentum. Introducing the concept of momentum, the algorithm can increase the convergence speed facing narrow cliffs and helps the algorithm to overcome shallow local optima. Another advantage of the Adam optimization algorithm is that it is insensitive to the choice of the learning rate α.

The output layer has a desired target value and the loss can be calculated at this layer and then the loss passes backwards layer by layer in the network from output layer to the hidden layers. Since the flow of loss is backpropagated into the network, the algorithm is called backpropagation Chauvin & Rumelhart (2013).

II.1.4.3 Convolutional Neural Networks

Convolutional Neural Networks LeCun *et al.* (1995) known as CNN are a specialized kind of neural network for processing data that has a known grid-like topology such as the image data as 2-D grid of pixels and time-series data as a 1- grid taking samples at regular time intervals. A mathematical operation called convolution is adapted by the network and convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers Goodfellow *et al.* (2016b).

Generally, convolution is an operation on two functions of a real-valued argument. Equation II.33a shows how the convolution operation is performed. The convolutional operation is also denoted with an asterisk shown in equation II.33b. The first argument the function x to the convolution is often referred to as the input and the second argument, the function w, as the kernel. The output is also called the feature map. If we now assume that x and w are defined only on integer t, the discrete convolution can be defined as in equation II.33c

$$s(t) = \int x(a)w(t-a)da \tag{II.33a}$$

$$s(t) = (x * w)(t) \tag{II.33b}$$

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{II.33c}$$

The input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters. These multidimensional arrays are called tensors. For example, a 2-D image I is used as an input and 2-D kernel K is moved over the input image step by step. The input of a neuron in the convolutional layer is calculated as an inner product of the kernel with the currently underlying image section. This operation is illustrated in Figure II-11. The output, known as feature map is expressed in equation II.34. Convolution is commutative and the only reason to flip the kernel is to obtain the commutative property.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{II.34}$$

In figure II-11, $3x4$ input image, the $2x2$ kernel size across the input image data results $2x3$ output values (feature map) or 6 distinct activations. The weights of a neuron are the filter values such as x, y, z, w. The feature detected by a neuron is the kind of input pattern that will cause the neuron to activate. Goodfellow *et al.* (2016b).

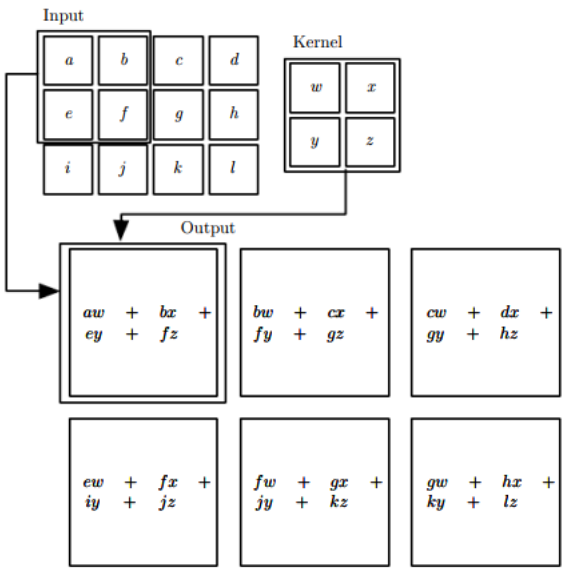


Figure II-11: *An example of 2-D convolution without kernel-flipping Goodfellow et al. (2016b).*

In addition, convolutional layers can reduce the size of the parameters. In Figure II-11, the kernel is moved by one, and the amount the filter shifts called striding. The length of the stride

can be chosen depending on the result expected. The larger the sizes of strides, the lower the computational and statistical burden of the next layer becomes. The larger size of strides enables the model parameters to shrink in size. Equation II.34 can be expanded with the striding parameter λ_t and it is expressed in equation II.35.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i \cdot \lambda_t + m, j \cdot \lambda_t + n) K(m,n) \tag{II.35}$$

For example, the architecture of a deep convolutional network is presented in Figure II-12. LeCun *et al.* (1998) proposed it to recognize hand-written characters. This network consists of alternating convolutional and subsampling layers and then it follows fully connected final layers. Each convolutional layer produces a number of feature maps which are a pattern of activity over an array of units.

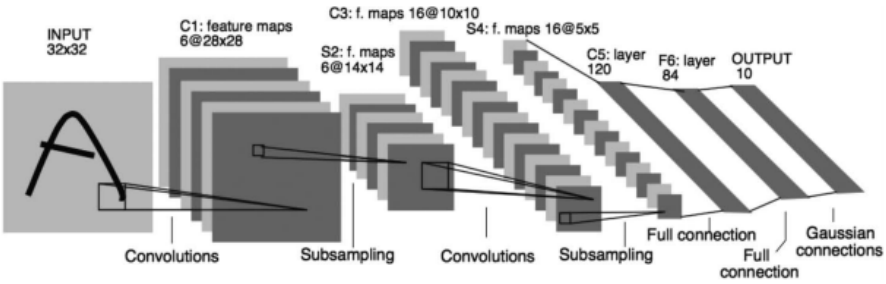


Figure II-12: *Deep Convolutional Network Sutton & Barto (1998) LeCun et al. (1998).*

II.1.5 Transfer Learning

Deep reinforcement learning is capable of handling difficult complex problems. However, learning can be too slow or even infeasible. For this reason, researchers in DRL have focused on improving the time spent on learning by implementing various approaches. The most successful is **transfer learning** (TL). The main purpose of TL is to improve the learning performance by using the experience from successfully pre-trained models Taylor & Stone (2009). Recent transfer learning approaches are systematically investigated in the context of DRL by Zhu *et al.* (2020). Transfer learning approaches are categorized by considering: the knowledge transferred, RL frameworks compatible with the TL approach, differences between the source and the target domain, information available in the target domain, sample-efficient in TL approach, and the goals of transfer learning.

Transfer learning can be used for different goals and in different situations. Several evaluation metrics can be addressed in order to evaluate the TL algorithms. Although there is no single metric can summarize the efficacy of a TL approach, common parameters for measuring TL performance are shown in Figure II-13. The difference between the initial reward values, with and without TL, is called jump-start. The final performance of the agent is named as asymptotic performance and the time required to achieve a pre-defined level is called time to Threshold. These metrics are explained in detail in Taylor & Stone (2009). Transfer learning is applied in different applications of DRL for UAV tasks. For instance, Anwar & Raychowdhury (2020) studied the DRL for autonomous navigation. Transfer learning is applied to reduce the training computation load. The environment is designed in Unreal Engine EpicGames (2019) and tested in the real world, by using a low-cost drone (a DJI Tello), and the similar results obtained.

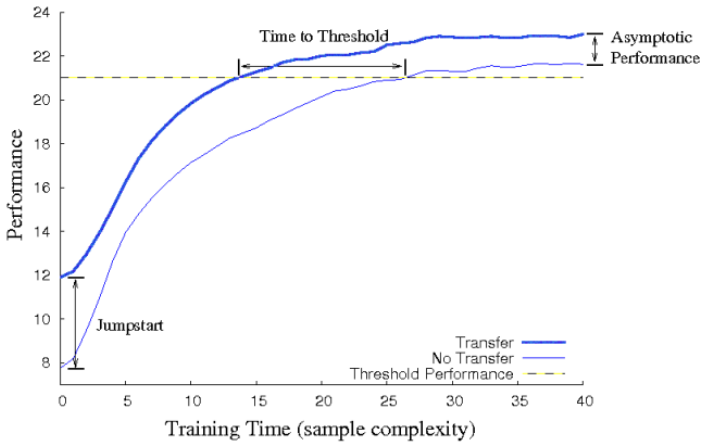


Figure II-13: *Different metrics for measuring TL. Taylor & Stone (2009)*

II.1.6 Deep Reinforcement Learning

Deep reinforcement learning (DRL) combines reinforcement learning and deep neural networks as function approximator and DRL helps the agents learn how to achieve their goals. Werbos P.J. Werbos (1989) studied neural networks as function approximation for reinforcement learning. This was an important research for the neural network as a function approximator after the tabular representations of values. Neural networks trained by error backpropagation to learn policies and value functions using TD-like algorithms. Nevertheless, Tsitsiklis & Van Roy (1996) showed that the combination of nonlinear function approximation, off-policy, and bootstrapping may lead to instability and divergence . This is also called the deadly triad issue Sutton & Barto (1998). The danger is neither a control or to generalized policy iteration nor learning, uncertainties about the environment. The instability can also be avoided if any two elements of the deadly triad are present. One can also do without bootstrapping but at the cost of computational and data efficiency.

Deep RL proposes the use of deep neural networks as the agent's decision algorithm. In conjunction with the experience replay memory, deep RL has been able to achieve super-human level when playing video and board games. For instance, DQN method is published by DeepMind Mnih *et al.* (2015) and the main goal of DQN is to use a deep convolutional neural network to approximate the optimal action-value function. DQN provides updating action values and target values iteratively. The deep RL solution is also based in double deep Q-network (DDQN) Van Hasselt *et al.* (2016a), an extension of the DQN implementation Mnih *et al.* (2013). DDQN selects from the state the action of the agent which maximizes the Q-value. Q-values are estimations of the future reward of an action executed in a given state. Van Hasselt *et al.* (2016b), Mnih *et al.* (2016), and Silver *et al.* (2016) are also focused on deep reinforcement learning after the original DQN research and these contributions help DRL evolve in time.

II.1.6.1 Reinforcement learning algorithms

Reinforcement Learning algorithms differ based on their characteristics, such as their approach to solving an RL problem, the type of information they use, or the requirements they have. RL can be broadly categorized into three main branches: the value-based approach, policy-based approach, and model-based approach. In value-based RL, the agent seek the policy which maximizes a value function which describes the long-term reward of a particular state or action. On the other hand, in policy-based approach, the agent focuses on directly learning a policy, which is a mapping from states to actions. Moreover, in model-based reinforcement learning, a model is provided to the agent for the environment or the agent is to learn an environment model to perform the

certain tasks in the environment. It is claimed that model-based learners are typically much more efficient in terms of experience Poole & Mackworth (2010) and model-free methods require more experiences but use less memory and often use less computation time. The agent's policy can be affected by some inaccuracies and imprecision if the agent is asked to learn the environment model itself. Therefore, many approaches were proposed to integrate the model-free approaches with the model-based ones François-Lavet *et al.* (2018). In addition to these main categories, there are also hybrid methods that combine elements of multiple approaches, and meta-RL methods that focus on learning to learn.

Reinforcement learning algorithms are also classified from different perspectives: model-based and model-free methods, value-based and policy-based methods (or combination of the two), Monte Carlo (MC) methods and temporal-difference methods (TD), on-policy and off-policy methods. In Figure II-14 the taxonomy of reinforcement learning algorithms is presented and they are as follows: DP (Dynamic Programming), TD (Temporal Difference), MC (Monte Carlos), Monte Carlo Tree Search (MCTS) Browne *et al.* (2012), Q-learning Watkins & Dayan (1992), I2A (Imagination-Augmented Agent) Racanière *et al.* (2017), DQN (Deep Q-Network) Mnih *et al.* (2015), Quantile QT-Opt Kalashnikov *et al.* (2018), C51 Categorical 51 Bellemare *et al.* (2017), TRPO (Trust Region Policy Optimization)Schulman *et al.* (2015), ACKTR (Actor Critic using Kronecker-Factored Trust Region)Wu *et al.* (2017), AC (Actor-Critic)Konda & Tsitsiklis (1999), A2C (Advantage Actor Critic) Mnih *et al.* (2016), A3C (Asynchronous Advantage Actor Critic)Mnih *et al.* (2016), DDPG (Deep Deterministic Policy Gradient)Lillicrap *et al.* (2015), TD3 (Twin Delayed DDPG)Fujimoto *et al.* (2018), SAC (Soft Actor-Critic)Haarnoja *et al.* (2018b), REINFORCE Williams (1988), world models Ha & Schmidhuber (2018).

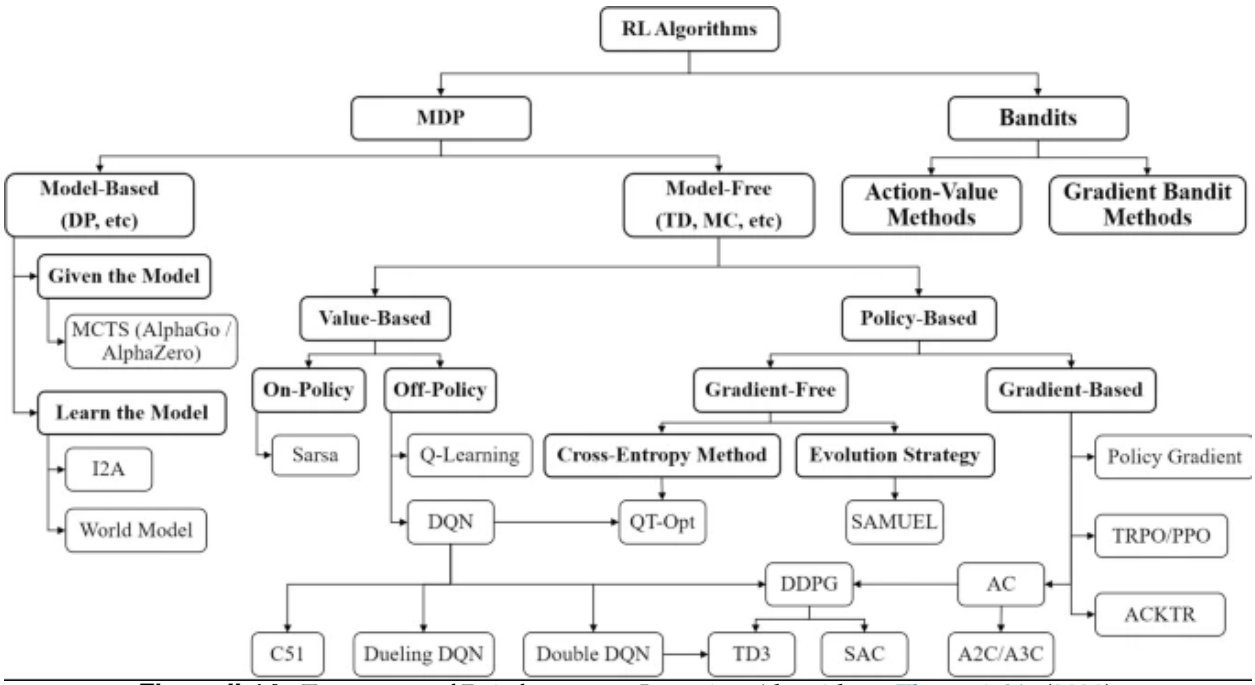


Figure II-14: *Taxonomy of Reinforcement Learning Algorithms Zhang & Yu (2020).*

II.1.6.2 Deep-Q Network (DQN)

Q-learning explained in section II.1.3.4 is a well-known method for reinforcement learning when there is no knowledge of the environment or no model is available Watkins & Dayan (1992). Mnih *et al.* (2015) was successful in combining Q-learning with neural networks and named the method deep Q-Network (DQN). The overall goal of Deep Q-Network (DQN) is to use a deep convolutional neural network to approximate the optimal action-value function, defined as:

$$\theta_\pi(s,a) = \max_\pi \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s, a_t = a, \pi] \tag{II.36}$$

The optimal action-value function represents the maximum of the sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy μ = P(a|s), after making an observation (s) and taking an action (a).

The release of the DQN (Deep Q-Network) paper by DeepMind Mnih *et al.* (2015) noticeably changed Q-learning introducing a novel variant with two key ideas.

The first idea was using an iterative update that adjusted the action-values (Q) towards target values ($\gamma \max_a$ Q(s_{t+1},a)) that were only periodically updated, thereby reducing correlations with the target.

The second one was using a biologically inspired mechanism named experience replay that randomizes the data removing correlations in the observations of states and enhancing data distribution, with a higher-level demonstration and explanation by previous research in McClelland (1995) , Riedmiller (2005b) and Lin (1993). The use of the experience replay encourages the choice of an off-policy type of learning, such as Q-learning, because if not, past experiences would have been obtained following a different policy from the current one.

Two huge advances can be taken out from this, one is that each training batch consists of samples of experience obtained randomly from the stored samples and current experience, so temporal correlation is clearly avoided. The other one is that each step in the agent's experience can be used in many weight updates, so a significant gain in efficiency is obtained in learning from the environment.

The whole process consists in characterizing an approximate value function $Q(s,a;\theta_t)$ using the CNN shown in eqn. II.38, in which θ_t are the weights of the Q-network at iteration t. For the experience replay, agent's experiences e_t which consist in the tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ are stored at each time-step t in the replay memory $e_1, \cdots, e_N$, where N sets the limit of entries, with the possibility of replacing older experiences for new ones when the limit of the memory is reached.

The standard Q-learning update for network parameters θ after taking action A_t in state S_t and observing the immediate reward R_{t+1} and resulting state S_{t+1} is:

$$\theta = \theta_t + \alpha[y_t^Q - Q(S_t, A_t; \theta_t)]\nabla_{\theta_t} Q(S_t, A_t; \theta_t), \tag{II.37}$$

where the estimated return as defined as Q-target y_t^Q:

$$y_t^Q = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta) \tag{II.38}$$

This update resembles stochastic gradient descent, updating the current value $Q(S_t, A_t; \theta_t)$ over the TD (Temporal Difference) error towards a target value y_t^Q.

However, the algorithms such as Q-learning described previously are inefficient when the experiences which are obtained by trial and error are used to adjust the networks only once and then discarded. Although some experiences can be ignored, there might be rare and very good experiences and it is wasteful if they are thrown away. To solve this issue, experience replay Lin (1992) is introduced. Experience replay (ER) is described as a way of reusing experiences. The RL agent can remember its past experiences and the experiences are presented to its learning algorithm repeatedly with the help of ER. Thus, the agent experiences what it had experienced before again and again and the agent can refresh what it has learned before.

Moreover, **Prioritized Experience Replay (PER)** is introduced by Schaul *et al.* (2015) to make the agent learn faster. Previously, experiences are sampled uniformly from a replay memory and

the transitions are replayed without considering their significance. However, PER prioritizes the experiences and important transitions are replayed more frequently. In this way, the agent learns efficiently.

The earlier research by Lange & Riedmiller (2010) uses the action as an input for the neural network and the history of all state-action transitions such as $Q(s, a)$ which maps the history of state-action pairs to scalar estimates of actions in a particular state. However, in this type of architecture, the Q-value of each possible action is computed in a separate forward pass and thus it is scaled linearly with the number of actions used. On the other hand, Deepmind researchers proposed an architecture Mnih *et al.* (2015) which separates the output unit for each possible actions and the state passes through the neural network as an input. In other words, the output layer corresponds to the predicted Q-values of each action for the input state and then the action with the highest value can be chosen by the policy. The convolutional neural network with state input and action output layer is shown in Figure II-15. In this figure, the input to the neural network consists of an $84X84X4$ image, followed by three convolutional layers and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier non-linearity (that is, $max(0, x)$).

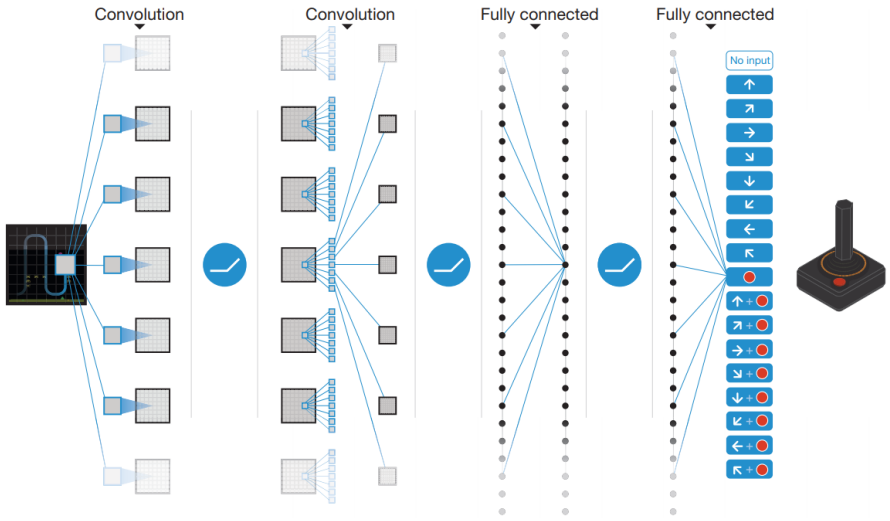


Figure II-15: *Schematic illustration of the convolutional neural network Mnih et al. (2015).*

Furthermore, in Q-learning, Q-function is updated and the target values shifts along with it, but this can cause divergence when neural networks are used to approximate the entire Q-function. To avoid the divergence, neural fitted Q-iteration is proposed by Riedmiller (2005a) to make the algorithm more stable compared to standard Q-learning algorithm. The target network parameters θ_- are only updated with the Q-network parameters θ every C steps and are held fixed between individual updates shown in Algorithm II.2 line 15. The target used by DQN is shown in Equation II.39.

$$y_t^{DQN} = R_{t+1} + \gamma \underset{a}{max} Q(S_{t+1}, a; \theta^-) \tag{II.39}$$

The full algorithm for training deep Q-networks is presented in II.2. The agent selects and executes actions according to an $\epsilon - greedy$ policy based on Q.

In addition, instead of updating the target network every C steps seen in Algorithm II.2 Line 15, soft target updates are proposed by Lillicrap *et al.* (2015). The weights of the target network are updated by having them slowly follow the learned networks at each step as a function of the target factor $\tau \ll 1$. The line 15 in Algorithm II.2 is extended to implement the soft target update

Algorithm II.2: *Deep Q-Network (DQN) with Experience Replay and Neural fitted Q-iteration*

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with two random sets of weights θ
Initialize target action-value function $\hat{\theta}$ with weights $\theta^{-} = \theta$
repeat(for each episode):
  Observe initial state S_1
  repeat(for each step of episode):
    Select action A_t with probability ε.
    Otherwise, select A_t
    Observe reward R_t and next state S_{t+1}
    Store the transition (S_t, A_t, R_t, S_{t+1}) in $\mathcal{D}$
    Sample random mini-batch of transitions from $\mathcal{D}$ ▷ Experience Replay
    Calculate target for each transition:
    $y_j = \begin{cases} R_j & \text{if } S_{j+1} \text{ is terminal} \\ R_j + \gamma Q(S_{j+1}, A|\theta^{-}) & \text{otherwise} \end{cases}$
    Update network parameters θ with loss $\mathcal{L} = \mathbb{E}[(y_j - Q(S_j, A_j; \theta))^2]$
    Every C steps, reset $\theta^{-} = \theta$ ▷ Neural fitted Q-iteration
  until S is terminal
until end of training

shown in Equation II.40. Authors showed that stability of learning is improved by constraining the target values to change slowly.

$$\theta^{-} = \tau\theta + (1 - \tau)\theta^{-} \tag{II.40}$$

II.1.6.3 Double Deep-Q Network (DDQN)

Using the Q-learning algorithm results in a positive bias by definition, since the maximum of the estimates is used as an estimate of the maximum of the true values. This makes it likely select overestimated values using a greedy policy as a target policy. The idea proposed in Hasselt (2010) and named as Double Q-learning is basically based in decoupling action selection from evaluation.

Two action-value functions Q_1 and Q_2 are learned by assigning each experience randomly to update one of the two function with the two sets of weights, θ and $\theta^{'}$ in Double Q-Learning, one set of weights is used to determine the greedy policy and the other its value.

$$y_t^{Q} = R_{t+1} + \gamma\theta(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t) \tag{II.41}$$

And the two Double Q-learning targets can then be written as

$$y_t^{DoubleQ_1} = R_{t+1} + \gamma\theta_2(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a; \theta_t); \theta_t^{'}) \tag{II.42}$$

$$y_t^{DoubleQ_2} = R_{t+1} + \gamma\theta_1(S_{t+1}, \arg\max_a Q_2(S_{t+1}, a; \theta_t^{'}); \theta_t) \tag{II.43}$$

Q_1 is used to determine the maximizing action $A_* = \arg\max_a Q_1(a)$ and Q_2 is used to provide the estimate of its value with Q_2 (A_*) = Q_2 $(\arg\max_a Q_1(a))$ shown in equation II.42. The second set of weights can be updated symmetrically by switching the roles of θ and $\theta^{'}$ into equation II.43, achieving unbiased estimates.

As only one estimate is updated per step in a random selection, but two estimates are learned, it doubles the memory requirements but not the computational effort made at each step. The Double Q-learning was extended for the DQN-algorithm in Van Hasselt *et al.* (2016a). Furthermore, the DQN-algorithm provides with the target network θ^- a natural candidate for the second value function, without having to introduce additional networks. The Double DQN algorithm remains the same as the original DQN-algorithm, the architecture for DQN and DDQN shown in Figure II-16, except replacing the target y_{DQN} explained in Kersandt *et al.* (2018) due to the limited space with

$$y_t^{DoubleDQN_1} = R_{t+1} + \gamma\theta_2(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a; \theta_t); \theta_t^-) \tag{II.44}$$

$$y_t^{DoubleDQN_2} = R_{t+1} + \gamma\theta_1(S_{t+1}, \arg\max_a Q_2(S_{t+1}, a; \theta_t^-); \theta_t) \tag{II.45}$$

where the weights of the second network $\theta^{'}$ of double Q-learning in equations II.42 and II.43 are replaced with the weights of the target network θ^-, performing the update to target network as in neural fitted Q-iteration. DDQN Algorithm is presented in Algorithm II.2.

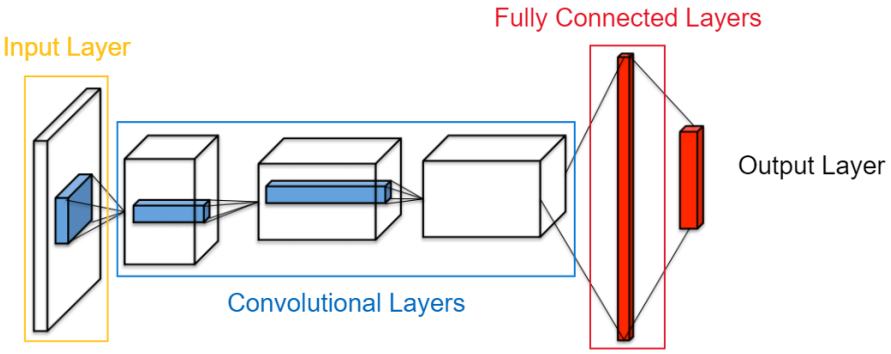


Figure II-16: *DQN and DDQN Architecture* Wang et al. *(2016).*

II.1.6.4 Dueling Network Architecture

In the dueling network architecture presented in Wang *et al.* (2016) there is no need to estimate the value of each action choice as it is calculated in DQN and Double-DQN. Instead of following the convolutional layers with a single sequence of fully connected layers, the dueling network has two new streams. One of the streams estimates state-value $V(s;\theta,\beta)$ and the other stream estimates the advantage for each action and output an $|A|$ dimensional vector $A(s,a;\theta,\alpha)$. θ is the parameters of the convolutional layers, while α and β are the parameters of the two streams of fully-connected layers. The lower layers of the dueling network are as in the original DQN. Finally, the two streams are combined to produce a single output Q function shown in Equation (II.46) as it is done in DQN Mnih *et al.* (2015). The agent dueling architecture can be seen in Figure II-17.

$$\theta(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) \tag{II.46}$$

The advantage of using dueling architecture is that the agent can learn which states are more valuable without learning each action at each state. In other words, there is no need to calculate the value of each action at that state value if the state is not good.

Algorithm II.3: *Double DQN*

Inputs: $\mathcal{D}$: Empty replay buffer, θ: initial network parameters, θ^{-}: copy of θ, N_r: replay buffer maximum size, N_b: training batch size, N^{-}: target network replacement frequency.
for episode $e \in 1, 2, ...M$ **do**
    Initialize frame sequence $x \leftarrow ()$
for $t \in 0, 1, ...$ **do**
    Set state s $s \leftarrow x$, sample action $a \sim \pi_B$
    Sample next frame x^t from environment e given (s, a) and receive reward r, and append x^t to x
    if $|x| > N_f$ **then**
        Delete oldest frame $x_{t_{min}}$ from x
    Set $s^{'} \leftarrow x$, and add transition tuple $(s, a, r, s^{'})$ to $\mathcal{D}$, replacing the oldest tuple if $|\mathcal{D}| \geq N_r$
    Sample a minibatch of N_b tuples $(s, a, r, s^{'}) \sim Unif(\mathcal{D})$
    Construct target values, one for each of the N_b tuples:
    Define $a^{max}(s^{'}; \theta) = argmax_{a^{'}} Q(s^{'}, a^{'}; \theta)$
    $$y_j = \begin{cases} r & \text{if } s^{'} \text{ is terminal} \\ r + \gamma Q(s^{'}, a^{max}(s^{'}; \theta); \theta^{-}) & \text{otherwise} \end{cases}$$
    Do a gradient descent step with loss $\|y_j - Q(s, a; \theta)\|^2$
    Replace target parameters $\theta^{-} \leftarrow \theta$ every N^{-} steps

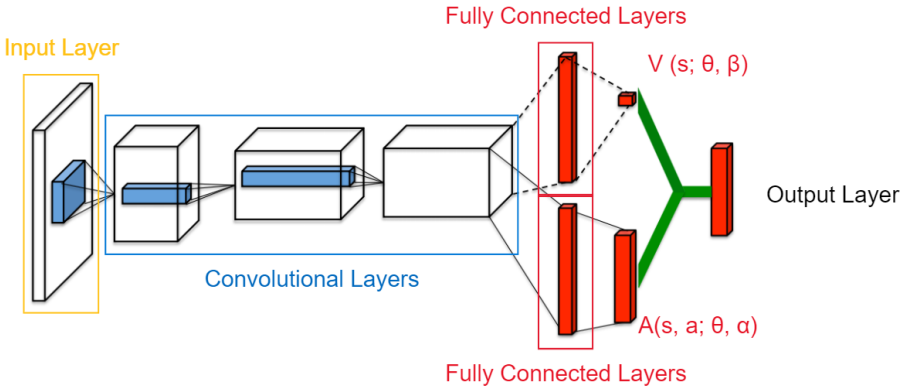


Figure II-17: *The dueling Q-network Wang et al. (2016).*

II.1.6.5 Deep Q-learning from demonstrations (DQfD)

Learning from scratch can be time consuming in some real-world applications such as counter-drone systems in a 3D space. Experiences from an expert can be used to accelerate the learning speed and the agent can learn in an efficient way. For instance, an AI method called imitation learning has been used to teach an agent to mimic the behavior of an expert. In imitation learning, the labeled data are used as an input and the agent imitates the actions from the recorded data. However, the data are limited to the expert data. On the other hand, a deep reinforcement learning algorithm called Deep Q-learning from Demonstrations Hester *et al.* (2018) is introduced to combine imitation learning and reinforcement learning. In DQfD the agent continues learning by sampling from both its self-generated data as well as the demonstration data. For instance, a spacecraft learns to land by DQfD algorithm in a target area surrounded by randomly generated terrain Grigsby (2023). DQfD can quickly learn the expert's policy and the spacecraft is landing successfully by avoiding the obstacles in the terrain.

The main purpose of this algorithm is to remove the limitations of the applicability of DRL to real-world tasks where the agent must learn in the environment. DQfD provides the agent with

data from previous control of the system. Thanks to a prioritized experience replay mechanism, DQfD can access the demonstration data to accelerate the learning process even if the agent has a small amount of demonstration data. Authors Hester *et al.* (2018) showed that DQfD can perform better and learn to out-perform the best demonstration given in 14 out of 42 games. In addition, DQfD has already achieved state-of-the-art results on 11 Atari games.

The DQfD algorithm has two phases of training. Firstly, DQfD pretrains on the expert demonstration data using a combination of temporal difference (TD): 1-step TD, n-step TD, supervised, and regularization losses. The TD loss enables the algorithm to learn a self-consistent value function from which it can continue learning with RL. In addition, the supervised loss is used to learn to imitate the demonstrator. After pre-training, the agent starts interacting with the domain with its learned policy. The agent updates its network with a mix of demonstration and self-generated data. The combination of demonstration data and self-generated data is automatically controlled by a prioritized-replay mechanism. The overall loss is presented in Equation (II.47). In this equation, in addition to TD losses, a margin classification loss J_E Piot *et al.* (2014) and L2 regularization loss J_{L2} are implemented. λ parameter is added to control the weighting between the losses. L2 regularization loss is applied to the weights and biases of the network to help prevent it from over-fitting on the relatively small demonstration dataset. A margin classification loss is added to make the greedy policy induced by the value function imitate the demonstrator by forcing the values of the other actions to be at least a margin lower than the value of the demonstrator's action. After pre-training, the agent starts interacting with the environment, collecting self-generated data and adding it to the replay buffer until it is full, but the demonstration data are never over-written.

$$J(Q) = J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q) \tag{II.47}$$

$$J_E(Q) = \max_{a \in A}[Q(s,a) + lQ(a_E, a) - Q(s, a_E)] \tag{II.48}$$

$$J_{L2}(Q) = \|\theta\|^2 \tag{II.49}$$

where a_E is the action the expert demonstrator took in state s, $l(a_E, a)$ is the margin function and θ represents weights described in Algorithm II.4.

II.1.7 Drone Detection for Counter Drone System

The technologies for the detection, localization and identification of drones, and the interdiction methods are explained in detail in section I.2. In this PhD thesis, drone detection is an important part of the counter drone solution to indicate where the target drone is located. This information is very valuable for learner drone, the agent in reinforcement learning, to improve the learning progress. How the agent uses drone detection information is explained in detail chapters IV, V and VI where the deep reinforcement learning models are proposed to counter a drone. Figure II-18 shows how drone detection model works with the environment and the DRL model. Drone detection model works as an improvement of the image and it adds an information on image where the target drone is.

II.1.7.1 Object detection models

Object detection is one of the core techniques in computer vision and image processing. There are several fast and powerful real-time object detection systems such as Fast r-CNN Girshick (2015), Faster r-CNN Ren *et al.* (2015), Xception Chollet (2017), Yolo Redmon *et al.* (2016) or EfficientNet Tan & Le (2019) Tan *et al.* (2020). These methods have been updated and improved considerably with respect to former CNN models. For example, Fast r-CNN Girshick (2015) is an evolution of the VGG16 network, a region-based convolutional neural network, using the Caffe framework

Algorithm II.4: *Deep Q-learning from Demonstrations (DQfD)*

Inputs: $\mathcal{D}^{replay}$: Initialized replay memory with demonstration data set, θ: weights for initial behavior network (random), $\theta^{'}$: weights for target network (random), τ: frequency at which to update target net, k: number of pre-training gradient updates.
for steps $t \in 1, 2, ...k$ **do** ▷ Pre-training
    Sample a mini-batch of n transitions from $\mathcal{D}^{replay}$ with prioritization
    Calculate loss $J(Q)$ using target network
    Perform gradient descent step to update θ
    if $t \bmod \tau = 0$ **then**
        $\theta^{'} \leftarrow \theta$
for steps $t \in 1, 2, ...$ **do** ▷ Post-training
    Sample action from behavior policy $a \sim \pi^{\epsilon Q_\theta}$
    Play action a and observe $(s^{'}, r)$
    Store $(s, a, r, s^{'})$ into $\mathcal{D}^{replay}$, overwriting oldest self-generated transition if over capacity
    Sample a mini-batch of n transitions from $\mathcal{D}^{replay}$ with prioritization
    Calculate loss $J(Q)$ using target network
    Perform gradient descent step to update θ
    if $t \bmod \tau = 0$ **then**
        $\theta^{'} \leftarrow \theta$
    $s \leftarrow s^{'}$

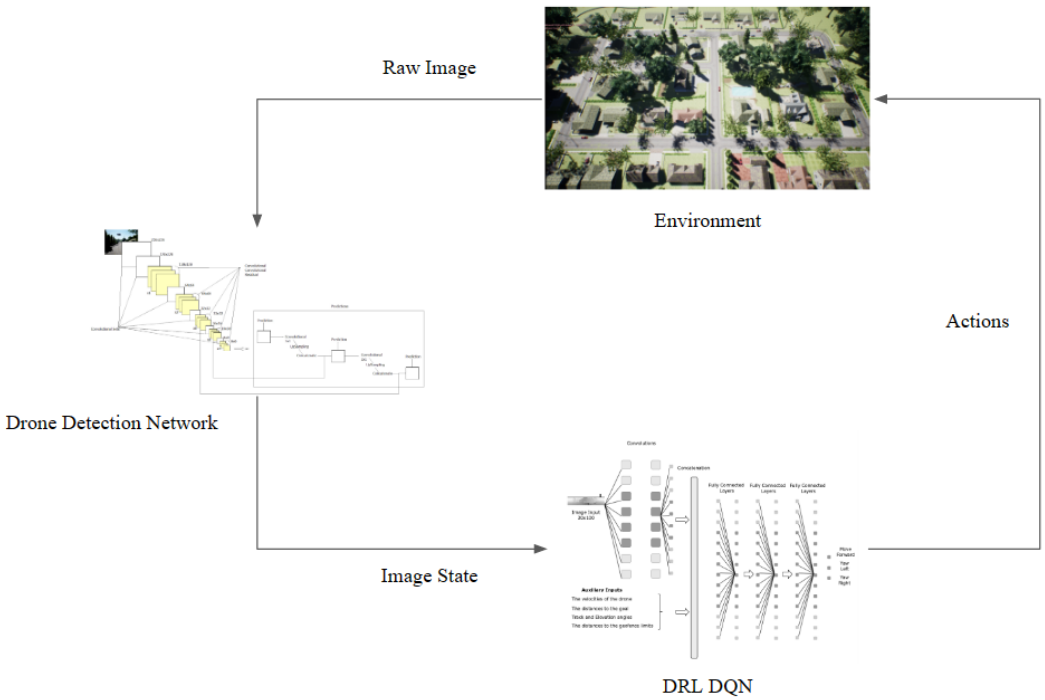


Figure II-18: *Drone Detection Model and DRL Diagram.*

with multitasking, in which training is done in a single-stage and avoiding any disk storage. Faster r-CNN Ren *et al.* (2015) builds on top of Fast r-CNN by introducing a Region Proposal Network (RPN) with the aim to propose regions of interest at the same time that feature maps are being generated.

Xception Chollet (2017) is also an evolution of the VGG16 network that uses inception layers Szegedy *et al.* (2015). These are neural network layers that independently look correlations at cross-channel and at spatial pixels. Xception is a linear pipeline of depth-wise separable convolu-

tion layers, efficiently implemented over TensorFlow. Xception is the base of the Facebook object detector software.

Yolo Redmon *et al.* (2016) is a family of algorithms used by many research works on object detection. Proposed by Redmon et al. Yolo, called after "You Look Only Once", frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. Yolo processes images by firstly resizing the input image. Secondly, a single convolutional neural network runs on the image. Finally, the resulting detections are thresholded by the model's confidence. Any newer version of Yolo proposes a larger neural network than the precious version and/or shows faster execution. Yolo predicts bounding boxes using dimensional clusters as anchor boxes Redmon & Farhadi (2017). Yolo has the advantage of allowing multi-label predictions, this is, an object can be detected as two (or more) different labels at the same time. In this way, a friendly drone and a malicious drone can be detected and labeled both as drones, and, at the same time, if visual appearances are known, they could be also labeled as threat or non-threat.

EfficientNet Tan & Le (2019) is a brand-new state of the art object detection model, that together with its recent evolution, EfficientNet Tan *et al.* (2020), has become very popular in a short time thanks to it's accuracy and efficiency. One of the key improvements is its novel bi-directional feature network (BiFPN) which allows information to flow in different directions: top-down and bottom up. Secondly, EfficientNet uses a fast normalized fusion technique, which adds an additional weight for each input feature, identifying the importance of each input features. Finally, it introduces a scaling method, which jointly resizes the resolution/depth/width of the model to better fit with the different resource constraints. EfficientNet-B0 is the version of EfficientNet adapted for small-size object.

II.1.7.2 Object detection models for on-board UAV processing

Object detection methods have been also implemented to detect objects from the UAV for different purposes such as surveillance and disaster management. Royo *et al.* (2022) enhanced the capabilities of drones used for the surveillance of large events. Authors presented a methodology which integrates efficiently deep learning algorithms in drone avionics. In addition, Salamí *et al.* (2019) proposed a parallel architecture which includes an UAV and cloud services provides useful real-time information directly to the end-users. In this research, olive trees which are regularly spaced from each other are counted in a crop field. They demonstrate near real-time results obtained from Unmanned aerial systems usage. Also, real time object detection has been performed to detect humans by using videos captured from UAV Bhattarai *et al.* (2018). Constraints such as computation time, scale of viewing and altitude are addressed. The results are also visualized in a Geographic Information Systems platform by geo-localizing objects to world coordinates. In other research, a technique is proposed Rudol & Doherty (2008) to detect humans at a high frame rate by using an autonomous UAV. A map of points of interest is built by geo-locating the positions of the detected humans. The video sequences, which streams from two video sources, thermal and color, are collected on-board the UAV and fused to increase the successful detection rate. Furthermore, Yang D. et al. Yang *et al.* (2020) proposed a real time detection and warning system, based on artificial intelligence (AI), to monitor the social distancing between people during the COVID-19 WHO (2021). A fixed monocular camera is used to detect individuals in a region of interest (ROI) and the distances between them are measured in real time without data recording. The proposed method is tested across real world datasets to measure its generality and performance.

II.1.7.3 Drone detection models

In 2017 the European SafeShore project launched, in parallel with the IEEE AVSS conference, the *Drone-vs-Bird Challenge* in order to improve the methods for detecting UAV close to coastal borders, where they can be easily get confused with birds. The challenge aims to correctly label

drones and birds appearing in a video stream. From the papers presented in the first and second editions, in 2017 and 2019 respectively, we can conclude that new advances are driven by more and more layers of the CNN, larger training datasets and improved implementations. In addition, the exploitation of temporal information is a key issue in differentiating the drones from the birds. Best paper of 2019 Craye & Ardjoune (2019) proposed a 110-layers CNN, based on a semantic segmentation U-Net network originally designed for medical image processing and adding some dilation layers to improve small objects detection. A posterior spatial-temporal filtering allowed an obtained F1-score of 0.73. The set of training and test images of these works, although challenging, are from ground and have the sky as background.

Drone-Net Lin (2020) is a deep learning model, available as open source, trained with $2,664$ real drone images. It contains 24 convolutional layers in total and two of them are detection layers called as Yolo layers. The details of the Drone-Net CNN can be seen in Appendix Fig. A-10. This model will be used as the state-of-the-art model for drone detection, and accuracy results of this model will be used to compare with the new models presented in Appendix A.

Xiaoping L. et al. Xiaoping *et al.* (2019) proposed a dynamic drone detection method, based on two consecutive inter-frame differences. They combine a Support Vector Machines classifier (SVM) with the traditional Histogram of Oriented Gradient (HOG) detection algorithm, and add an intermediate step, a Fisher Linear Discriminant (FLD), to reduce the dimensionality of the HOG features. Using a dataset of 500 images their results show an accuracy above 90%, similar to other drone detection algorithms, but with improvements in terms of detection time allows to process up to 10 images per second. Since this method is based on the difference of consecutive images, it is not suitable for a moving camera.

Hu Q. et al. Hu *et al.* (2019) proposed an object detection method, called DiagonalNet, by using an improved hourglass CNN as its backbone network and generating confidence diagonal lines as detection result. A large dataset ($10,974$ sample images) is created by processing videos and photos with different backgrounds and lights and augmented by rotating and flipping each image with random angles. The images contain six types of UAVs, including multirotor and helicopter, and are labeled manually. The proposed algorithm detects UAV quickly (at 31 images per second) and accurately (with a mean average precision greater than 90%). The experiments were all indoors and in close proximity with the target drone.

With the objective to improve a swarm cooperative flight and low-altitude security, Jin R. et al. Jin *et al.* (2019) proposed a 6D pose estimation algorithm for quadrotors. The algorithm proposed, based in the Xception network and pre-trained with ImageNet, includes a novel relational graph network (RGN) to improve the performance of the drone pose detection. The pose is obatined by recognizing eight key points of a quadrotor (nose, rotors, etc.). After training with 340 images the simulation experiments show a mean average precision of 0.94 in position and 0.74 in velocity, which compared to the baseline network is an enhancement of 9.7%. Given the real time capabilities of the algorithm (30 frames per second) the algorithm was tested also with real flights. However these new results dropped to mean average precision of $0.65 - 0.75$. Moreover, the accuracy of 6D pose decreased for small-sized drones.

Carrio A. et al. Carrio *et al.* (2020) use Airsim to train a CNN detection network of 16-layers by automatically labeling depth maps. Depth maps are obtained by stereo matching of the RGB image pairs of the virtual ZED stereo camera on the Airsim drone. The ground truth labels of the depth maps are generated automatically by color segmentation of the visual image. After training with 470 images the detection system is integrated on board a small drone and tested while the drone navigates in the environment. Results show that the system can simultaneously detect drones of different sizes and shapes mean average precision of $0.65 - 0.75$, and localize them with a maximum error of 10% of the truth distance when flying in linear motion encounters. The solution is limited to maximum distance of 8 meters and relative speeds up to 2.3 m/s.

Wyder P. M. et al. Wyder *et al.* (2019) present a UAV platform to detect and counter a small UAV in an indoor environment where GPS is not available. An image dataset is used to train a Tiny Yolo object detection algorithm. This algorithm, combined with a simple visual-servoing approach, is also validated in a physical platform. It successfully tracked and followed a target drone, even with an object detection accuracy limited to 77%.

II.1.8 Summary of Background

In this thesis, DRL methods are the most promising method againts drones and DRL algorithms are utilized to create autonomous and intelligent drones that can learn and adapt to their surroundings. DRL offers several advantages in countering drones. Firstly, DRL models can handle noisy or inaccurate sensor inputs and adapt to uncertain or dynamically changing conditions. They can learn to make robust decisions even in the presence of sensor noise or environmental disturbances. Secondly, DRL allows autonomous decisions to be made without requiring explicit programming. The agent can learn to make optimal decisions in complex and uncertain situations. Moreover, DRL algorithms can adapt and learn from their environment, which makes them suitable for dynamic and changing drone scenarios. Furthermore, a state-of-the-art object detection algorithm is also implemented to identify drones in the environment to add a valuable information to the agent's observation of the target drone. In summary, the DRL and object detection methods together provide a complete solution to address the challenges of drone navigation, drone detection, tracking and countering drones.

II.2 Previous Works

This section presents previous studies to review and show the existing research and findings in drone navigation, obstacle avoidance and counter drone techniques that exist in both 2D and 3D spaces.

II.2.1 Drone Navigation and Avoidance of Obstacles

In this section literature works that are related to the drone navigation by using reinforcement learning methods are reviewed. Duo & Zhao (2017) proposed UAV autonomous navigation system for GNSS (Global Navigation Satellite System) invalidation and it is stated that the GNSS can help navigating an UAV in most of the application scenarios. However, when the GNSS is not available, for example passing through the sheltered areas, it is necessary to use other methods to aid inertial navigation. To overcome this problem, the researchers used proper visual sensors for navigation. In another research Pham *et al.* (2018), reinforcement learning algorithm is used to navigate an UAV in an unknown environment. In this study it is shown that the quadrotor can successfully learn how to navigate through an unknown environment by using Q-learning methods combined PID (Proportional-Integral-Derivative) controller. Also, Hu et al. Hu *et al.* (2020) proposed to address the UAV's autonomous motion planning problem by implementing a method based on a DRL algorithm known as deep deterministic policy gradient (DDPG). The method is called multiple experience pools (MEP)–DDPG algorithm. The researchers stated that UAVs trained with MEP-DDPG algorithm can complete successfully the experimental tasks in an unknown environment. Moreover, the geofence technology is used in drone navigation in order to create constraints for drones. A geofence is defined as a technology which creates a virtual barrier around a geographical area. The purpose of using geofence is to keep the drones out of or within the predefined area von Bothmer (2018). If the geofenced area is violated, the user can be notified to a pre-programmed entity and send warning signals to the operator to prevent the device

entering this geofenced area. In addition, European Commission has released a regulation Commission (2023) on the rules and procedures for the operation of unmanned aircraft. According to this regulation drones are restricted to be used in certain airspaces. If it is detected a potential breach of airspace limitations, the remote pilots are alerted so that they can take immediate and effective action to prevent that breach. In a study Gurriet & Ciarletta (2016) about a generic and modular geofencing strategy for civilian UAVs, geofence is used to avoid collisions with the environment such as controlled airspace areas, people, or other flying vehicles. Additionally, a deep reinforcement learning solution is proposed to teach a drone how to reach a non-visible goal without crashing into any block in an environment which has columns inside as obstacles Kersandt (2018) Kersandt *et al.* (2018). The results show that a fully autonomous drone can be achieved by deep reinforcement learning. Furthermore, deep reinforcement learning method where the states containing image and several scalars are processed by a joint neural network (JNN) is proposed to use drones in an urban environment to deliver goods without crashing on obstacles and geofence Muñoz *et al.* (2019). Recent studies demonstrated the applicability of the usage of deep reinforcement learning methods to navigate autonomously drones in an environment. However, due to the limitations in the environment and the time required to train the DRL agent, previous works implemented DRL algorithms did not consider a new environment with moving obstacles, feasible geofencing concept and improved drone movements.

II.2.2 Counter a drone in a 2D space

In the literature, UAV detection, localization and neutralization have been studied by using different methods including artificial intelligence solutions. Previously, counter drone systems are explained and summarized in I.2. In addition, several current technologies are reviewed in a research Besada *et al.* (2021) for the non-collaborative tracking and detection of UAVs. The authors propose a collection of simulation models which are composed by integrating preexisting models of radar and acoustic sensing. Using these models, the most important aspects of detection and estimation performance of C-UAS sensors and sensor networks can be simulated. Moreover, Rudys *et al.* (2022) proposes the concept of the airborne counter-UAV platform with radar. Authors use a low-cost marine radar with a high resolution $2m$ wide antenna which is embedded into the wing and aircraft heading is changed to enable radar scanning. It has been demonstrated that the platform is capable of detecting drones, determining their coordinates, and neutralizing them using a "hunter" drone in order to accomplish the intended tasks. Also, Watkins *et al.* (2020) illustrated how to design a powerful counter autonomous drone tool that can be used against a single drone or a group of drones by using hard-to-patches vulnerabilities. Authors claim that this counter autonomous drone tool fills a critical need for mitigation of risks due to privacy violations such as recording video by drones. Furthermore, Chen *et al.* (2022) proposes a countermeasure to disrupt drone swarm coordination and clustering by splitting them into several unconnected parts in a short period of time. Authors observe that the proposed two algorithms such as a genetic algorithm and particle swarm optimization can find the optimal solution with high accuracy and efficiency. Recent studies show that counter drone technologies utilizes radar systems, different kind of sensors and cameras to detect and mitigate security threats posed by drones. However, these methods are not cost effective considering the equipment used and they did not include an autonomous drone which can navigate in an environment without colliding with obstacles and then can attempt to catch the target drone.

II.2.3 Counter a drone in a 3D space

Drones equipped with DRL have been proposed by researchers and their main focus is on avoiding obstacles in an environment and navigating them safely to their destination. For instance, Polvara et al. Polvara *et al.* (2017) introduces a method based on DRL to land an UAV on a ground

marker. A hierarchy of DQN are used for high level navigation policies. The network generates in output 7 actions including vertical descent. Results show that DQN can be applied to complex problems such as landing on a ground marker. However, it is suggested that further research is needed to reach stable policies for different kind of scenarios such as sea and wind currents etc. Moreover, Anwar et al. Anwar & Raychowdhury (2020) presented a deep reinforcement learning method for resource constraint edge nodes using transfer learning to reduce the on-board computation required to train a deep neural network. In other words, the domain knowledge is transferred to test environments and training the last few fully connected layers only. A low-cost drone is also tested by using this approach and the results showed similar performance in different baselines. In another research by Kouris et al. Kouris & Bouganis (2018), it was proposed that a self-supervised Convolutional Neural Network (CNN)-based approach can be used to navigate a drone autonomously and to avoid collisions in a 3D world. A regression CNN is used to predict the distances between the agent and the obstacles. The distances to the closest obstacle in different directions are estimated. The drone flight parameters in a 2D space, such as the linear velocity and the yaw angle of the drone, are changed according to the predictions made through the deep neural network. Furthermore, reinforcement learning is implemented to land a fixed wing aircraft autonomously in simulation environment Matúš (2019). Author defined the actions such as discrete control inputs and throttle input to perform the landing. However, it is shown that the policies proposed do not reach a pilot-like performance and the autonomous aircraft failed to reach the runway. In addition, Hovell *et al.* (2022) studied multi-agent real-time guidance strategy for quadrotors to perform aircraft runway inspection. Authors implemented a deep reinforcement learning method and quadrotors autonomously and cooperatively learn to perform a runway inspection. These studies mostly focused on 2D space and the action space did not include any actions that corresponds to changing the drone altitude. However, in chapter VI, the action which can change the drone altitude is also included to counter a drone. Besides, the experiences are pre-processed from previous training to improve learning on a related but different task such as hunting the target drone in an environment with moving obstacles.

Furthermore, EfficientNet-B0, a sub version of EfficientNet Tan *et al.* (2020), is used to detect drones in chapter VI. EfficientNet is a popular state-of-the-art object detection model thanks to its accuracy and efficiency. EfficientNet-B0 is adapted for small-size objects. A detailed explanation of our drone detection model is provided in section III.4 and appendix A.

Les coses importants son les que no ho semblen.
[The important things are the things that don't seem so.]

— Mercè Rodoreda

Before anything else, preparation is the key to success.

— Alexander Graham Bell

III

Experimental Setup

In this chapter, the tools required for training and testing deep reinforcement learning algorithms in this thesis are explained. Reinforcement learning algorithms can be trained in many different platforms that ensure agent-environment interaction. The experimental setup provides an overview of various software platforms and simulators that can be used for training and testing deep reinforcement learning algorithms. In this thesis, the AirSim simulator is used to provide a realistic and dynamic environment for training and testing DRL algorithms. Moreover, the OpenAI Gym library and Python toolkits like Keras-rl and TensorFlow are used for the development and evaluation of algorithms. Additionally, a state-of-the-art object detection algorithm is improved for detecting drones and the details are explained in detail. Finally, explainable artificial intelligence and reinforcement learning section is added to explain and justify the agent actions by using graphical methods.

This chapter is based on the following publications:

- Çetin E, Barrado C, & Pastor E.. 2021. Improving real-time drone detection for counter-drone systems. *In: The Aeronautical Journal*. 125(1292), 1871-1896. D.O.I: 10.1017/aer.2021.43.

III.1 Reinforcement Learning Software Platforms and Flight Simulators

Software platforms are utilized to execute certain tasks for reinforcement learning (RL) algorithms. In this thesis, one of most popular platform, OpenAI gym Brockman *et al.* (2016) is used to communicate, train and test deep reinforcement learning algorithms. However, many platforms are introduced in the literature and most of them are open-source. These platforms can be used for many areas such as robotics, trading, autonomous vehicles, etc.. Table III-1 presents the environments and simulations for reinforcement learning algorithms and applications.

In the literature, there are many simulators to train RL algorithms. For example, X-Plane X-Plane (2023) is one of the most advanced flight simulator. It provides many different kinds of aircraft and it is based on first-principles physics and real-world data with realism. Also, Gazebo Gazebo (2023) is an open source robotics simulator and it allows users to access to high fidelity physics, rendering, and sensor models. QPlane Richter & Calix (2021) is an open-source RL toolkit for autonomous fixed wing aircraft simulation. MujoCo MuJoCo (2023) is open source physics engine and it can be used in robotics, biomechanics, graphics and animation. Furthermore, DeepMind (2023d), DeepMind (2023e), DeepMind (2023c), DeepMind (2023a), and DeepMind (2023e) provide different frameworks for RL in games. Also, Microsoft' The Malmo platform Microsoft (2023b) is designed to support fundamental research in artificial intelligence. It is a sophisticated AI experimentation platform built on top of Minecraft game. ReAgent Meta-Research (2023) is an open source end-to-end platform for applied RL. Also, Tensor Trade TensorTrade-org (2023) allows users to build, train, evaluate and deploy trading algorithms using RL and it is an open source. Another framework called Ns3-gym integrates both OpenAI Gym and the network simulator it encourages the usage of RL in networking research. Text World Microsoft (2023c) is a text-based game generator and RL agent can be trained and tested in this learning environment. Furthermore, Reco Gym Criteo-Research (2023) is a OpenAI gym RL environment for the problem of product Recommendation in online advertising. OpenSim Stanford-NMBL (2023) is a biomechanical physics environment for musculoskeletal simulations and it allows users to develop a controller for a physiologically plausible 3D human model. VIZDoom Farama-Foundation (2023) is primarily intended for research in machine visual learning, and deep RL and it allows developing AI bots that play Doom using only the visual information. In addition, there is another environment, Gym Trading Ingargiola (2023) for reinforcement-learning algorithmic trading models. Webots Cyberbotics (2023) is an open-source robot simulator to model, program and simulate robots, vehicles and mechanical systems. PyBullet Physics (2023) is a real-time collision detection and multi-physics simulation for virtual reality, games, robotics, and machine learning. AWS DeepRacer Amazon (2023) allows users to get started with machine learning by training reinforcement learning models and test them in an autonomous car racing.

Airsim Shah *et al.* (2017) flight simulator among the simulators listed in Table III-1 is selected for this PhD thesis to simulate DRL algorithms. The main reason is that Airsim is developed as a platform for AI research to experiment with deep learning, computer vision and reinforcement learning algorithms for autonomous vehicles. In addition, the flight characteristic of a quadrotor flying in a real-world is very close to the quadrotor provided in the Airsim simulation. Thus, the model error between the real world and the simulation can be minimized.

III.2 AirSim

AirSim Shah *et al.* (2017) is a platform for AI research to experiment with deep learning, computer vision and reinforcement learning algorithms for autonomous vehicles, and it is built on Unreal Engine EpicGames (2019). Unreal Engine provides ultra realistic rendering and strong graphic

Table III-1: *RL Software Platforms.*

Platform	Implementation
Airsim Microsoft (2023a)	Autonomous Vehicles
X-Plane X-Plane (2023)	Autonomous Vehicles
Gazebo Gazebo (2023)	Robotics and Autonomous Vehicles
Q-plane Richter & Calix (2021)	Autonomous Vehicles
OpenAI Gym Brockman *et al.* (2016)	Robotics and Custom Environments
MujoCo MuJoCo (2023)	Robotic and Biomechanics
DeepMind OpenSpiel DeepMind (2023d)	Gaming - Search and Planning
DeepMind Control Suite DeepMind (2023b)	Infrastructure for Physics-Based Simulation
DeepMind Lab DeepMind (2023c)	3D navigation and puzzle-solving
Deepmind PySC2 DeepMind (2023e)	Gaming
AWS DeepRacer Amazon (2023)	Autonomous Vehicles
Project Malmo Microsoft (2023b)	Gaming
ReAgent Meta-Research (2023)	Building Products and Services for Large-scale
AI Safety Gridworlds DeepMind (2023a)	Identifying AI Safety
Tensor Trade TensorTrade-org (2023)	Trading Strategies
Ns3 Gym TKN-TUBerlin (2023)	Networking
Text World Microsoft (2023c)	Gaming
Reco Gym Criteo-Research (2023)	Product Recommendation in Online Advertising
OpenSim Stanford-NMBL (2023)	Biomechanics
VIZDoom Farama-Foundation (2023)	Gaming
Gym Trading Ingargiola (2023)	Trading Strategies
PyBullet Physics (2023)	Robotics
Webots Cyberbotics (2023)	Robotics

features for the Airsim. Airsim has a lot of environments available to be used in reinforcement learning. These environments are mountains, blocks, neighbourhood, city environment etc.. Also, Airsim provides different types of vehicles such as drones and cars. In this thesis, the neighbourhood environment is selected for training and testing. Environment and quadcopter used in the Airsim simulator can be seen in Figures III-3 and III-2. Airsim supports sensors such as camera, Global Positioning System (GPS), LIDAR sensors. AirSim API provides different type of images such as Scene, DepthPlanar, DepthPerspective, DepthVis, DisparityNormalized, Segmentation, SurfaceNormals, and Infrared. Additionally, camera sensor on quadcopter provides images in different angles such as front center, front right, front left, bottom center and back center of the quadcopter. In Figure III-1 shows a snapshot from Airsim simulator where a quadcopter is flying in an urban environment. The inset in this figure shows the depth, object segmentation and front camera streams generated in real time. In this thesis, Scene and DepthPerspective image types are utilized.

The urban neighborhood is chosen to counter a drone because of the similarity in real-life experiences such as a high number of drones in urban areas. Quadcopter used in the Airsim simulator and the environment in Airsim can be seen in Figures III-2 and III-3.

It is important to mention that AirSim is not deterministic. The simulator has its own physics and dynamics, which can be affected by simulated environmental conditions, such as wind, but

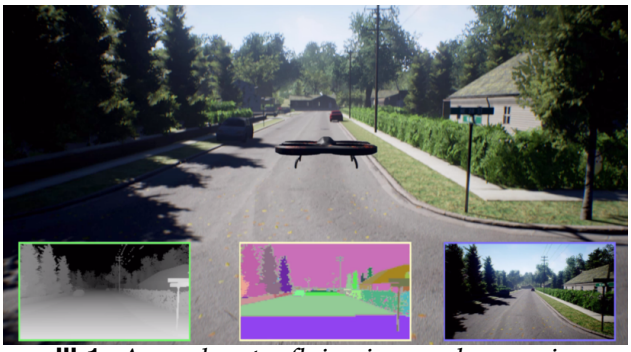


Figure III-1: *A quadcopter flying in an urban environment.*

it is also affected by some random white noise. By including white noise, AirSim aims to create a more realistic and dynamic simulation environment. For instance, in Airsim, random fluctuations or disturbances introduced into the simulation environment to mimic real-world conditions. For this reason, as in real life, the same actions give not exactly the same response when applied in different simulations.

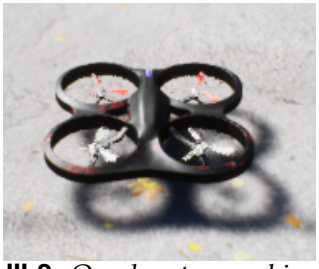


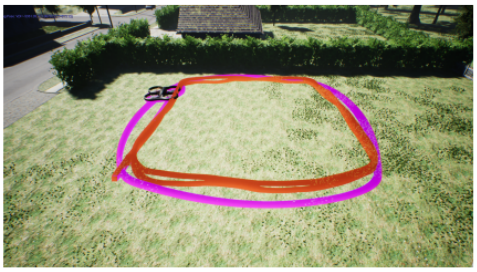
Figure III-2: *Quadcopter used in AirSim.*

Figure III-3: *The neighbourhood environment.*

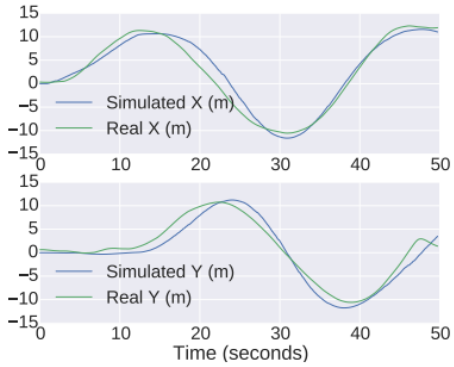
In addition, the flight characteristics of the quadcopter in Airsim simulator are evaluated by Microsoft research team Shah *et al.* (2017). They also evaluated some of the sensor models in Airsim against the real-word sensor models. Pixhawk v2 flight controller together with a Gigabyte 5500 Brix running Ubuntu 16.04 are utilized for the evaluations. They recorded the sensor measurements on the Pixhawk device itself. They also configured the simulated quadrotor in AirSim using the measured physical parameters and simulated sensor models configured using sensor data sheets. To perform repeatable offboard control for both real-world and the simulated flights, they used an application called AirSim MavLinkTest. Afterwards, they fly the quadrotor in the simulator in two different patterns such as a trajectory in a circle shape with 10m long radius and a trajectory in a squared shape with each side being 5m long. Same commands are used to fly the real vehicle and location of the vehicle in local NED coordinates are collected for both the simulation and real-world flights. For example, Figure III-4 presents the differences between the

simulated and the real-world flight. Figures III-4(a) and III-4(b) present visual comparison for the circle and the square patterns respectively. In these figures the simulated trajectory is represented by a purple line while the real trajectory is shown with a red line. It is observed that both the real-world and the simulated vehicle trajectories are close but there are small differences caused by some factors such as vehicle model approximations, integration errors and random winds. Moreover, the time series of locations in simulated flight and the real flight are shown in Figures III-4(c) and III-4 respectively. Researchers also computed the symmetric Hausdorff distance between the real-world track and the track in simulation. It is found that the simulation and real-world tracks were fairly close both for the circle ($1.47m$) as well as the square ($0.65m$).

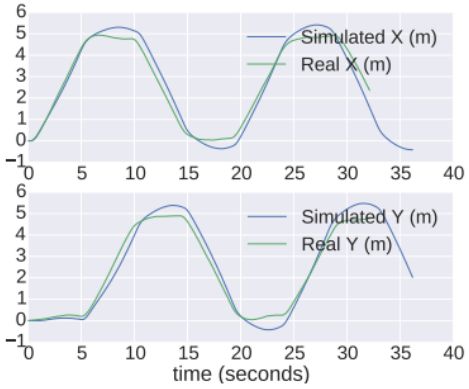
(a) Circle maneuver



(b) Square maneuver



(c) Space-Time Plot for Circle



(d) Space-Time Plot for Square

Figure III-4: *Evaluating the differences between the simulated and the real-world flight.* Shah et al. (2017)

III.3 OpenAI Gym and Python Toolkits

OpenAI-Gym is an open source interface to facilitate reinforcement learning tasks. It is a toolkit for developing and comparing reinforcement learning algorithms. It is a well known library for low level implementation of neural networks and it is compatible with TensorFlow Abadi *et al.* (2015). OpenAI-Gym library has a collection of environments to test reinforcement learning algorithms. These environments have a shared interface which allows to write general algorithms. Custom environments can also be created by the users to train and test RL algorithms or these environments can be connected to external simulation tools. Moreover, Keras-rl Plappert (2016) implements some state-of-the art deep reinforcement learning algorithms in Python and seamlessly integrates with the deep learning library Keras. Keras is a high-level neural network library and Keras functions as a wrapper to TensorFlow's framework. Keras can work with OpenAI-Gym and it is built according to the developer needs, giving the ability to define own callbacks and metrics.

In Figure III-5, core components of the experimental setup and the interactions between the simulation and the DRL tools such as Tensorflow, Keras and OpenAI Gym, drone detection model via python pipe. The use of a Python pipe enables parallel processing, allowing for efficient and concurrent execution of tasks. The drone detection model assists in identifying drones within the simulation environment. In this setup the DRL model constructed in python interacts with the simulation environment provided by OpenAI Gym and uses Tensorflow and Keras to process observations and generate actions. These actions are then executed within the simulation environment and the agent receives rewards.

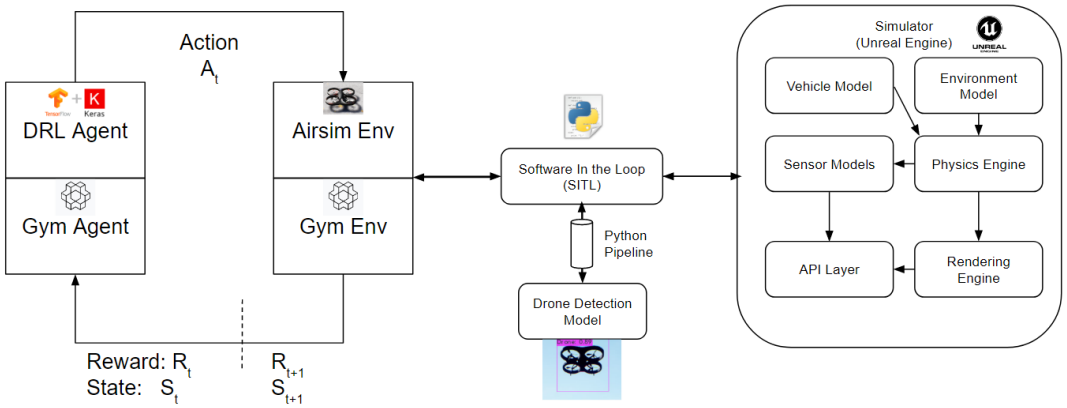


Figure III-5: *Experimental Setup.*

III.4 Drone Detection Platform

A number of counter-drone solutions are being developed but the cost of drone detection ground systems can also be very high depending on the number of deployed sensors and powerful fusion algorithms. In this section, a state-of-the-art object detection algorithm presented in section II.1.7 is used to train for detecting drones. Three existing object detection models are improved by transfer learning and tested for real-time drone detection. The drone detection model is trained with different kind of images of drones to obtain a more robust drone-detector. The main contribution in this section is that images captured from the Airsim simulator are automatically labeled with a bounding box around the drone. Training of this drone detection model is done with a new dataset of drone images, constructed automatically from Airsim. The guard-drone flies capturing random images of the area, while, at the same time, a malicious drone is flying too. The drone images are auto-labeled using the location and attitude information available in the simulator for both drones. The world coordinates of the malicious drone position have to be projected into image pixel coordinates. One of the advantages is that the time to label each image captured in the simulator is reduced compared to labeling them manually. Besides, the images can be used directly for training without needing third party applications for labeling. It is considered that the auto-labeling introduced here can be time saving for many researchers who work in object detection subject. The training and test results show a minimum 22% improvement in accuracy with respect to state of the art object detection models. In this PhD thesis, the drone detection algorithm is applied to image seen by the drone and before sending it to the agent as part of the state of the RL model. The drone detection algorithm adds a valuable information to the state by indicating were the target drone is.

III.4.1 Tools and Methods

In this section the tools and methods which are used for developing, training and testing drone detection models are discussed.

III.4.1.1 Training and Testing Framework

DarknetRedmon (2013); Bochkovskiy (2020) is a framework for the training and testing of the neural networks written in C language that provides an efficient solution for general object detection in real-time. We will use Yolo-V3 Redmon & Farhadi (2018) with Darknet-53, the originally 53-layered CNN shown in Figure A-9, that achieves the highest measured floating point operations per second. In addition to the algorithms implementation the Darknet framework provides also several CNN pre-trained models.

III.4.1.2 Drone Image Dataset and Auto-Labeling

Training a CNN needs a large dataset of labeled images. In this section the new drone images for the training dataset are captured by using the Airsim simulator. Airsim provides a public Application Programming Interface (API) for receiving parameters related to the drone and environment. We created a dataset of 2000 images for training, 1280×960 in size, captured in Airsim by using a drone flying randomly in the environment. Then, the images captured are auto-labeled by mapping drone position in world coordinates to the image coordinates. Some of the image samples used in training can be seen in Figure III-6.

(a) (b) (c) (d)

Figure III-6: *Airsim Training Images.*

The procedure of projection from 3D world coordinates to the image plane includes few steps and they are explained in detail below.

- The Pinhole Camera Model
  The pinhole camera model defines the geometric relationship between a 3D point in the scene and its 2D corresponding projection onto the image plane. This geometric mapping from 3D to 2D is a perspective projection. In Airsim simulator, pinhole camera model is available and it is mounted to the drones for capturing images. The pinhole camera models in Airsim don't include geometric distortion which are caused by lenses. Figure III-7 shows a schematic view of the pinhole camera projection. Following paragraphs will explain more in detail the different coordinate systems.

- Forward Projection
  The order of the forward projection is shown in Figure III-8. Firstly, world coordinates of the drone which is found in the image are converted to camera coordinates by using quaternions and rotation matrix. The rotations are described as a yaw-pitch-roll sequence and the rotation matrix can be obtained by using the Euler angles which are available in the simulator and it is shown in Equation III.1.

  Quaternions are applied to coordinate rotations and related them to the Euler angles Stevens *et al.* (2015). Quaternion for a yaw-pitch-roll sequence are presented in Equation III.2.

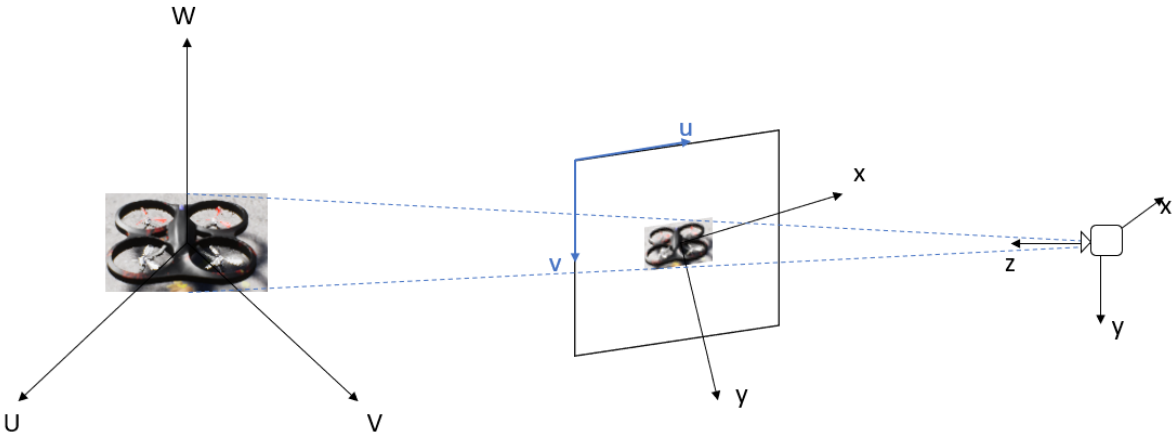


Figure III-7: *Pinhole Camera Projection Visualization*

$$R = \begin{bmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ -\cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) & \cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\sin(\psi) & \sin(\phi)\cos(\theta) \\ \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi) & -\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) & \cos(\phi)\cos(\theta) \end{bmatrix} \tag{III.1}$$

$$q = \begin{bmatrix} \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) - \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) - \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) \end{bmatrix} \tag{III.2}$$

Drone captures the image of the other drone which is visible in the camera in certain angles such that when the field of view of camera is less than 60 degrees. Secondly, image coordinates are obtained by using the perspective projection of the camera which uses the camera matrix received from the simulator. Finally, the pixel values are calculated by moving the origin to the upper-left corner of the screen. The mapping geometry is presented in Figure III-7.

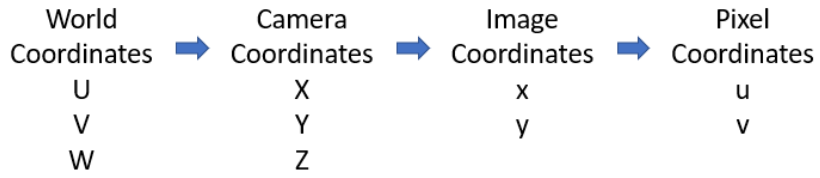


Figure III-8: *Projection Summary*

III.4.1.3 Transfer Learning and Proposed Models

The main purpose of transfer learning (TL) is to improve the learning performance by using the experience from successfully pre-trained models Taylor & Stone (2009). Transfer learning can be used for different goals and in different situations. For instance, the Drone-Net is built by

using transfer learning to refine a Darknet model for detecting drones. In this section new models proposed to improve the current results of Drone-Net are presented.

Table III-2: *Backbone Models used in Training*

Model	# of CNN Layers	# of Classes
Drone-Net Lin (2020)	24	1
Darknet-53 Redmon & Farhadi (2018)	107	80
EfficientNet-B0 Tan & Le (2019)	145	80

Table III-3: *Model Details for Training (all 6,000 iterations)*

Model	Backbone (# Pre-trained Layers)	Train Datasets	# of the Images
Model-1	Drone-Net (16)	Airsim + Drone-Net	3000
Model-2	Darknet-53 (74)	Airsim + Drone-Net	3000
Model-3	EfficientNet-B0 (132)	Airsim + Drone-Net	3000

- **Model-1** uses the same CNN architecture which Drone-Net has, shown in Appendix Figure A-10. From the pre-trained Yolo network, up to 16 convolutional layers are transferred. After transferring these convolutional layers, the network is trained with 2000 new images obtained from the airsim simulator and another 1000 images taken from the Drone-Net training set. The main purpose is to use the pre-trained weights from Drone-Net with the hope that the new model can detect the real drones and the drones from airsim images too.

- **Model-2** is built by using the pre-trained weights from Darknet-53 for the neural network model based on Yolo-v3 Redmon & Farhadi (2018) architecture. In this model, Darknet-53 model is implemented and the default Yolo-v3 network is modified to detect only the drone class. The Yolo-v3 network shown in Figure III-9 contains 107 layers: 75 convolutional layers, 23 shortcut layers, 4 routes, 2 upsamples and 3 Yolo detection layers. Predictions in Figure III-9 show that Yolo-v3 detects objects in 3 different layers. The model summary can be seen in Appendix Figure A-11. In this model up to 74 convolutional layers from Darknet-53 model are transferred and then the training has been extended with the 3000 images: 2000 images from the Airsim simulator as before, plus another 1000 images taken from the Drone-Net training set.

- **Model-3** is constructed and optimized by using EfficientNet-B0 object detection algorithm to detect a drone. EfficientNet-B0 contains 145 layers and 2 detection layers. Up to 132 convolutional layers from Efficient-D0 model are transferred. The EfficientNet-B0 model summary used in training can be seen in Appendix Figure A-12. Training set includes the 3000 images: 2000 images from the Airsim simulator and 1000 images taken from the Drone-Net training set.

  The models proposed here have backbone networks such as Drone-Net, Darknet-53 and EfficientNet-B0 backbone models. The details of these backbones are presented in Table III-2. In addition, in Table III-3, the model details are explained. For instance, model-1 uses up to 16 Drone-Net backbone network convolutional layers and model-2 has backbone network from darknet-53 up to 74 convolutional layers. Model-3 has more layers in total than the other models thanks to EfficientNet-B0 model which has 145 layers total and it uses up to 132 convolutional layers from EfficientNet-B0.

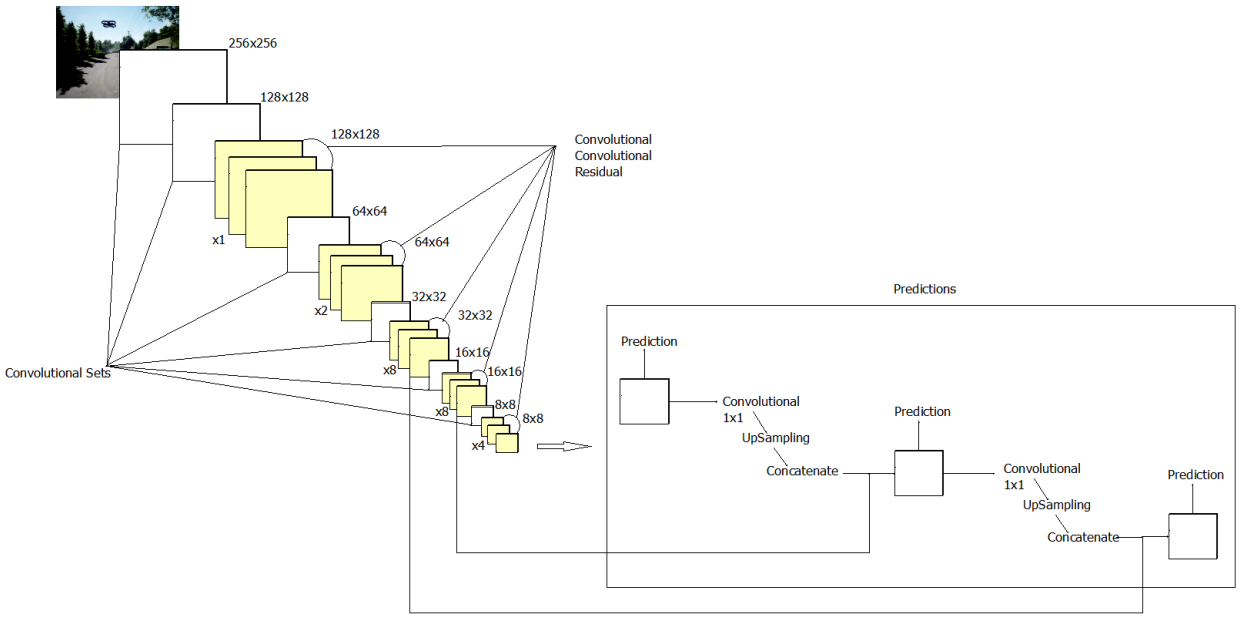


Figure III-9: *Darknet-53 CNN used as backbone for Model-2*

III.4.2 Quality of Drone Detection Model

In appendix A.1, the CNN model summaries of the drone detection model and the training and test results are presented. The overall test results show that proposed model here has acceptable accuracy above 85 % to detect only drones. These are promising results compared to former state of the art drone detection models. The good results are a step forward the construction of a full-autonomous counter-drone system.

III.5 Local Desktop Computer & Google Cloud Platform

The proposed models are trained on a desktop with a graphical processing unit (GPU), NVIDIA GeForce GTX 1060 with 6 GB RAM graphic co-processor and Intel i7 processor, 16 GB of memory. In addition, Google cloud platform provides colaboratory Bisong (2019), shortly “Google Colab" which allows you to write and execute python in internet browser. Google Colab allows executing codes on Google’s cloud serves and there are powerful hardware including graphical processing unit (GPU) and tensor processing unit (TPU) in these servers. Neural networks are also tested on a Google cloud server with a GPU Tesla T4-16GB. Recent DRL Models which uses drone detection models are trained on a desktop PC with NVIDIA GeForce RTX 3060 Ti with 8 GB VRAM graphics co-processor.

III.6 Explainable AI and Reinforcement Learning

Recent developments in AI technology demonstrated the significant power of AI algorithms. In addition, AI models have been continuously evolving to become more and more complex. As a result, users and even engineers or data scientists who created the AI algorithm have difficulties to explain how the AI model gives the specific output as it is represented in Figure III-10. This phenomenon is also called black box and this makes it difficult for end-users to trust these AI systems.

Explainable artificial intelligence (XAI) is defined as the set of processes and methods that make possible for humans to understand and trust machine learning results and artificial intelligence (AI) models IBM (2023). XAI has been also considered in a research by Johnson W.L. Johnson (1994). In this research, it is described an approach for intelligent artificial agent to explain and justify their action. This approach is implemented in an artificial fighter pilot and it can explain the reasons why it has chosen the actions. Area of research with XAI grows and there are many reviews that have been published Anjomshoae *et al.* (2019). XAI becomes one of the most important parts of AI systems which include autonomous driving, weather simulations, facial recognition, business optimization and security Wells & Bednarz (2021). The transparency and trustworthiness are crucial for these AI systems to build trust and confidence when it is decided to use them in real world applications. In aeronautics, it is mandatory to certify any software or model created with AI methods.

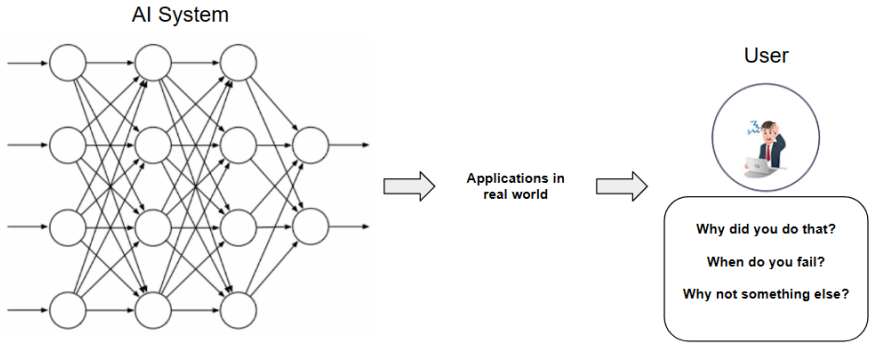


Figure III-10: *The need for explainable AI*

XAI has been investigated by researchers in many organizations DARPA (2023) Google (2023a) to produce more explainable AI models without changing the learning performance and to enable users to understand and trust AI models. XAI can also help developers to ensure that the AI system works as expected and it can have an impact on safety of the machines which uses AI models such as autonomous vehicles or robotics Araiza-Illan & Eder (2019).

In this PhD thesis, the explainability of deep reinforcement learning is also investigated. The figures which represent the rewards, drone locations, crash positions and the action distribution during training and testing are analyzed and compared with different scenarios and parameters. In other words, the agent behavior is observed and the modifications are done accordingly in training and testing sessions. To understand how agent behaves in an episode, some figures can be investigated. For example, drone positions for a successful episode during training is shown in Figure III-11. It is seen that the agent moves forward at the beginning and then moves up in few steps. Later the agent executes left and right turns before climbing back to the altitude where the target drone is located. Finally the agent execute his final steps: moves forward and turns right. The actions agent took in this episode are illustrated in Figure III-12 with colors to compare it with action frequencies during training shown in Figure III-13. The expected behavior can be seen in actions. For instance, it is expected that the agent needs to move forward and move up in some steps because the target is at an altitude higher than the agent is at when the episode starts. Action-4 is used frequently during the training to move up the agent and then the second mostly used action, Action-0, is selected to move forward. The actions and the movements are described in Table III-4.

Table III-4: *Actions.*

Action	Movement
0	in +x direction
1	yaw left
2	yaw right
3	+z direction
4	−z direction

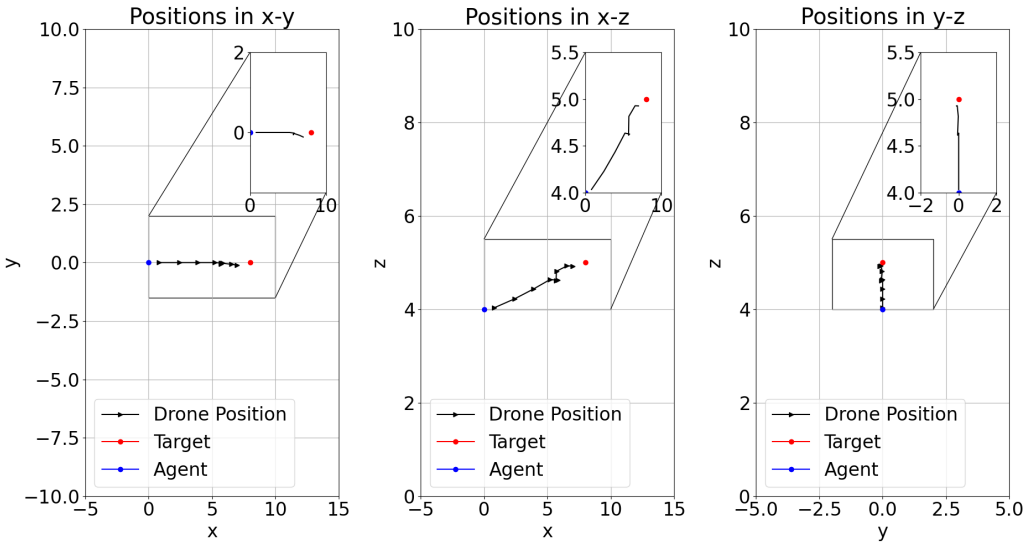


Figure III-11: *Drone positions.*

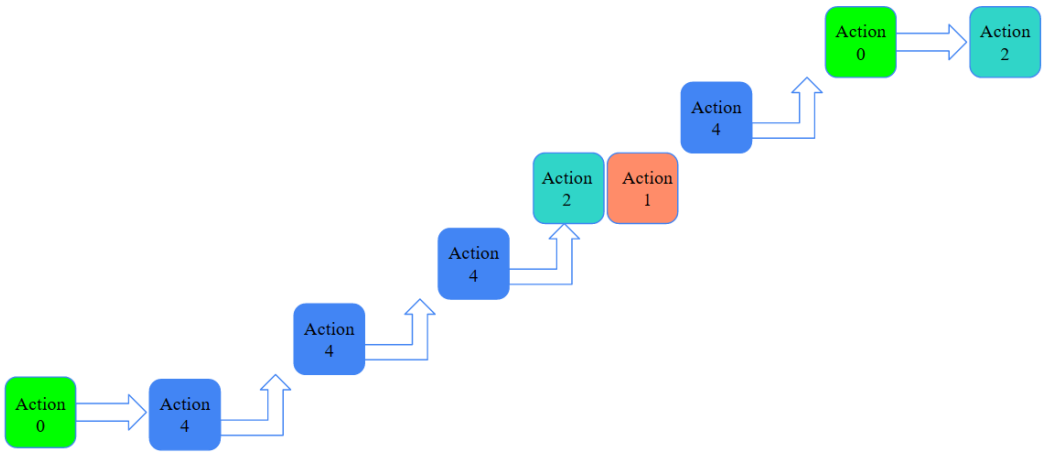


Figure III-12: *Actions.*

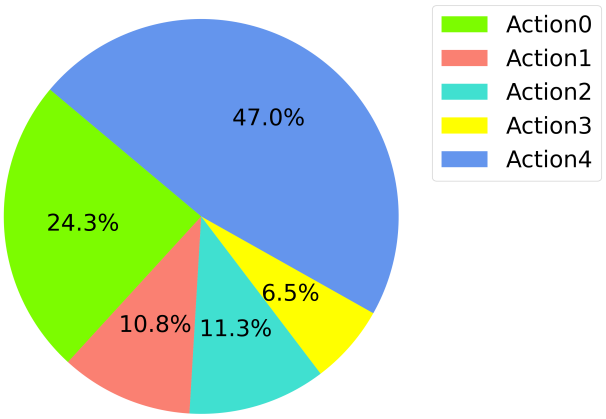


Figure III-13: *Action Frequencies during training.*

One, remember to look up at the stars and not down at your feet. Two, never give up work. Work gives you meaning and purpose and life is empty without it. Three, if you are lucky enough to find love, remember it is there and don't throw it away.

— Stephen Hawking

Je donnerais tout ce que je sais pour la moitié de ce que je ne sais pas.
[I would give everything I know for half of what I don't know.]

— René Descartes

IV

Drone Navigation and Avoidance of Obstacles

In this chapter, deep reinforcement learning (DRL) architecture is proposed to make drones behave autonomously inside a suburb neighborhood environment. Drones are trained to reach target locations in an environment with obstacles. The environment in the simulator has plenty of obstacles such as trees, cables, parked cars and houses. In addition, there are also other drones, acting as moving obstacles, inside the environment, while the learner drone has a goal to achieve. In this way the drone can be trained to detect stationary and moving obstacles inside the neighborhood and so the drones can be used safely in a public area in the future. The drone has a front camera and it can capture continuously depth images. Every depth image is part of the state used in DRL architecture. Also, another part of the state are the distances to the geofence (a virtual barrier on the environment) which are added as scalar values. The agent will be penalized when it tries to overpass the geofence limits and the episode will be terminated. In addition, angle to goal and elevation angle between the goal and the drone will be used as information to be added to the state. It is considered that these scalar values will improve the DRL performance and also the reward obtained. The drone is trained using Deep Q-Network and its convergence and final reward are evaluated. Results from three scenarios show promising outcomes, with the learner drone reaching its destination with a success rate of 100% in the first two tests and 98% in the third test, which involved three drones. The outcomes of this chapter that helped in the counter-drone solutions in terms of using geofence, reward system, avoiding obstacles in the environment.

This chapter is based on the following publications:

IV.1 Environment

In this section, Airsim release v1.2.3 is used and small urban neighbourhood is selected as an environment. Airsim provides APIs to retrieve data and control vehicles autonomously. Python script is created to interact with Airsim API layer to communicate and exchange data. The neighbourhood environment is shown in Figure IV-1 which includes the drone which is the agent, start point, destination, coordinates, obstacles in the environment and geofenced region which covers an area shaded blue. DepthImage and the boolean landing and collision information are also collected. Initially, the drone agent is facing in x-direction.

Figure IV-1: *The neighbourhood environment*

IV.2 Drone Agent States

Deep reinforcement learning model has a double state composed by an image and scalar values. The part of the state containing the front depth image is set as 20x100 pixels image by using the middle section of the full image and the scalar values of the state are: the current velocity of the drone, the distance to the goal, angle to goal (track angle), elevation angle between the goal and the drone and the distance to the geofence limits shown in Table IV-1.

In Table IV-1 the data proposed as state is shown. The data is received at every time-step from the Airsim. The current velocities of the drone is a pair of scalars (v_x, v_y) providing the x and y components of the speed vector and the units are in meters per seconds. The distance between the goal and the drone in two directions composoed of (d_x, d_y) and the total distance to the goal (d_t) in the environment. The distance to the geofence limits contains $(dg_{xmin}, dg_{xmax}, dg_{ymin}, dg_{ymax})$. The geofence limits are shown in shaded blue region in Figure V-2. Also, drone yaw angle (ψ) relative to the initial orientation is received.

- Çetin E, Barrado C, Munoz G, Macias M, & Pastor E.. 2019 (Sep.). Drone navigation and avoidance of obstacles through deep reinforcement learning. *In: IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. San Diego, CA, USA : DASC. D.O.I: 10.1109/DASC43569.2019.9081749
- Muñoz, G., Barrado, C., Çetin, E., & Salami, E.. 2019. Deep reinforcement learning for drone delivery. *In: MDPI Drones*. 2019, 3(3), 72. D.O.I: 10.3390/drones3030072.

Table IV-1: *Data received from the environment*

Data	Meaning
$v_x v_y$	agent's velocities in x and y directions
$d_x d_y d_t$	agent's distances to goal in x and y and Euclidian
dg_{xmin} dg_{xmax}	
dg_{ymin} dg_{ymax}	agent's distances to geofence limits
ψ	yaw angle relative to initial orientation
DepthImage	depth image in camera plan (256 x 144)
arrived	boolean landing info
collided	boolean collision info

IV.3 Agent's Neural Network

Double-DQN (DDQN) algorithm is used to train and the resulting model is used to navigate the drone in this chapter, and a number of improvements are added considering a new environment with moving obstacles, geofencing concept, and new scalar values which are added to the joint and the smoothing of the drone movements.

Several scalars and one image are combined by a neural network in order to process the states. The image is the input of a convolutional neural network (CNN) and then a concatenation layer joints the flatten output of the CNN with the reshaped scalar values of the state. The concatenated tensor becomes the input of three RELU layers with the following characteristics: each has activated consecutive 256 kernel dense layers. The output layer is a dense layer and the outputs are the action values. The deep neural network (DNN) architecture is shown in Figure IV-2.

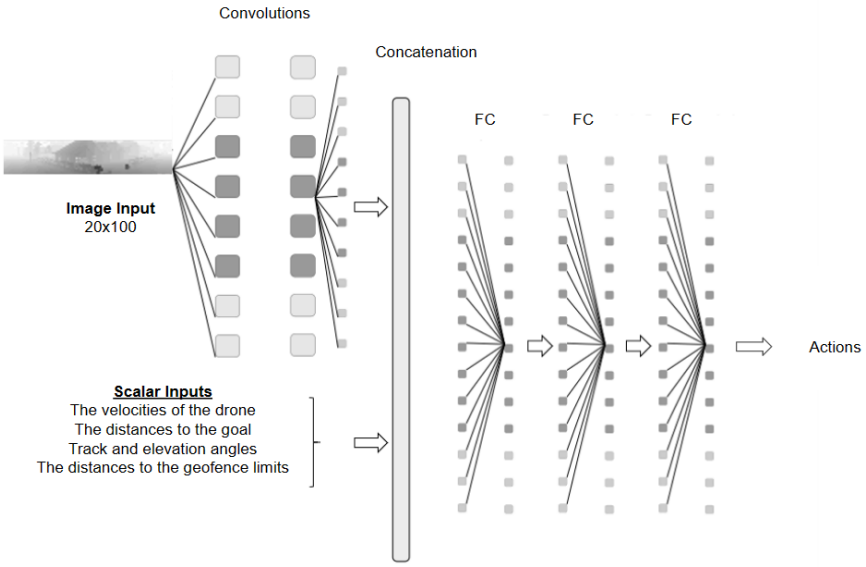


Figure IV-2: *The DNN architecture of the drone agent.*

IV.4 Drone Control Actions

In this section the actions that drone agent can select are explained. After the take-off no vertical movements will be allowed, thus the drone will fly at fixed altitude. It is aimed that the drone

collision can be observed by moving only along two dimensions (x and y). The output of the neural network architecture contains 5 different options: 4 actions which are the modification of the velocity in the two dimensions, the equivalent of $\pm 0.5 m/s$, and no speed modification option which the drone keeps its current velocity. In other words, the drone movements are continuous. Figure IV-3 shows the 2D representation of the actions.

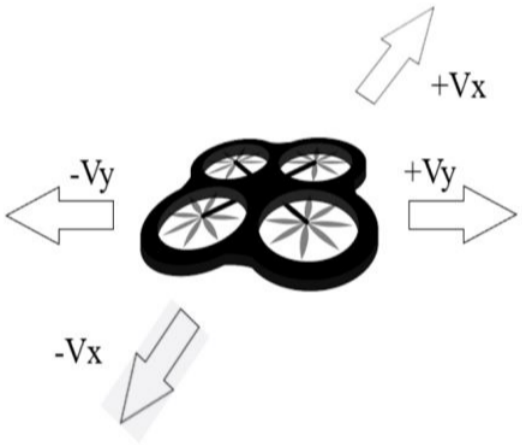


Figure IV-3: *Action Space*

IV.5 Reward Function

The reward system is very relevant for the success of a DRL training. The reward function is shown in Table IV-2. The agent is rewarded $+100$ if the episode is successful by reaching the destination. On the other hand, the agent is penalized and it is given -100 if there is a collision or a violation of the geofence. As previously described, the geofence is used as a virtual barrier for the drone and the drone is penalized if the predefined area is violated in order to increase the safe integration of drones into the environment. This is accomplished by calculating the distance to the geofence limits. Some collisions may not be caused by the learning drone, but the others running over it. In this case, the agent is innocent but it is penalized with a reward of -10. We consider the learner drone to be innocent when it can not see the random drone and it is the random drone movement which produces the collision. Independent of its source, a collision of the learner drone always ends the episode. Intermediate steps return a reward of -1 (to penalize delays) plus Δdg which is distance-to-the-goal difference with respect to the previous step. Δdg is used to stimulate actions that approach to the goal. Additionally, maximum time to reach the goal was set and the episode run was stopped after the given time limit.

Table IV-2: *Rewards given at the end of each step/episode*

Reward	The Reason
$+100$	Goal reached
-100	Collision: Obstacle (stationary or moving) or geofence
-10	Innocent Collision: By random drone
$-1 + \Delta dg$	Otherwise

IV.6 Results

The results are given in next two subsections: Training results presented in section IV.6.1 and Testing results shown in section IV.6.2.

IV.6.1 Training Results

In this section, training results are given separately for three experiments: training number 1 including only learner drone, training number 2 including learner drone and random drone 1, and finally training number 3 having the learner drone and random drones 1 & 2 which move randomly in the environment. The learner drone starts an episode with different yaw angles in order to increase the exploration capabilities of the learning.

In Figure IV-4 the initial positions of the random drones and the learner drone in the environment are shown. The random drones are used as moving obstacles and each random drone moves randomly but restricted to stay along the road. For example, in Figure IV-4, the area shaded with blue, and cyan colors represent the regions of random drones 1 & 2 respectively. Their duty is to block the movement of the drone trained by deep reinforcement learning.

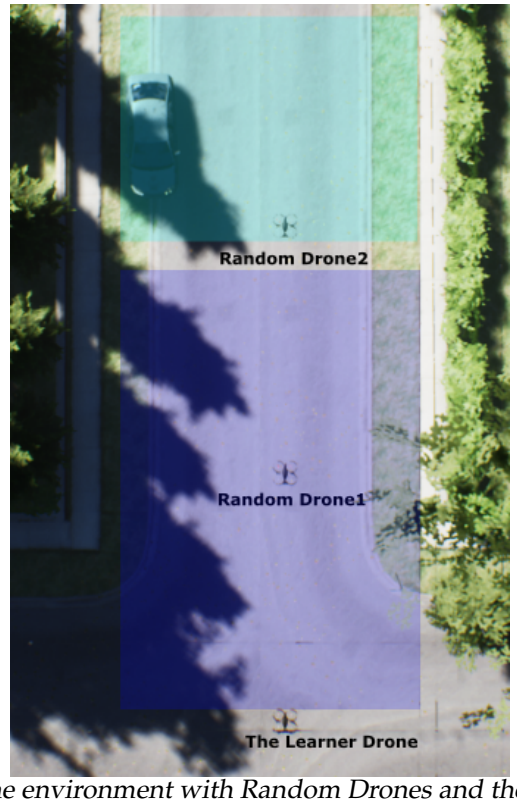


Figure IV-4: *The environment with Random Drones and the Learner Drone*

In Figure IV-5, Figure IV-6 and Figure IV-7, the training results with 125000 steps and accumulated rewards are shown. The light blue represents the actual reward value of each step and the dark blue represents the mean rewards of the every 100 steps. The vertical dotted line represents the end of the annealing training part marking the 50000 step point. The training episodes are finished when the drone reaches the destination or when the learning drone collides with any stationary or moving obstacle (random drone) or when the learning drone overpasses the geofence limits.

The training results obtained by using only the learner drone and without any moving obstacles in the environment are shown in Figure IV-5. It is seen that at the beginning of the training

the rewards are below the zero when the percentage of randomness in selecting the action is high. Although there are some successful episodes, the reward values are lower than the -100 which means the drone crashed in most of the periods. As the random behavior decreases over time, it is seen that the reward values are starting to increase and the trend of training curve is moving up and having more positive rewards. After annealing the reward values becomes more stable towards the end of the training.

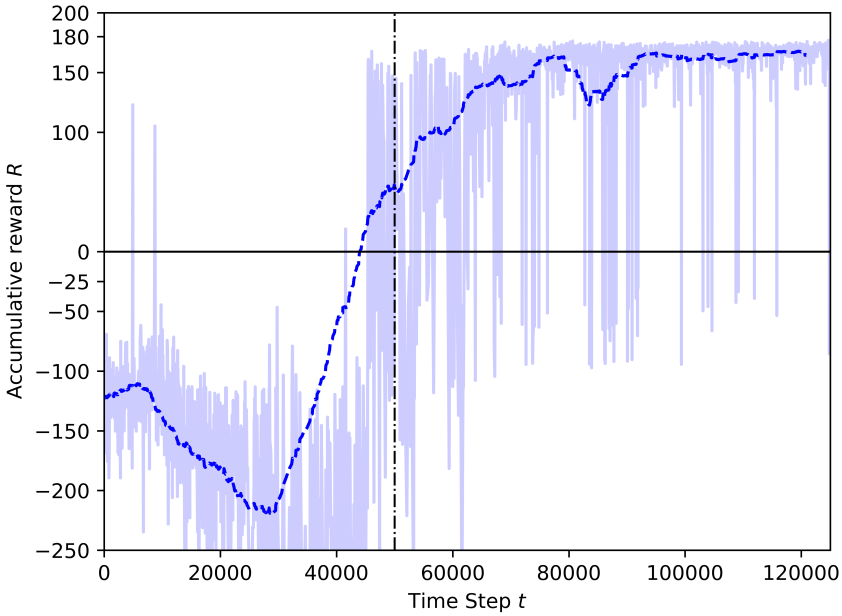


Figure IV-5: *Training results with only learner drone*

In Figure IV-6 the training results of the learner drone under scenario "random drone 1" are shown. The training curve shows similar trend as in Figure IV-5. For example, after around 25000 time steps, the learner drone starts learning how to avoid obstacles and the mean reward line goes up, the accumulative reward (R) increases. However, there exist episodes in which the learning drone crashed mostly at the beginning of the training and the reward values are between 0 and (-25). There are 135 episodes which the agent is hit by the "random drone 1". This random drone is the reason of the collision and the learning drone is innocent and then the learner drone penalized as (-10) and the simulation is reset. These crashes are represented with red cross in Figure IV-6. The crashes are mostly happened in the beginning of the training where the random behavior is high and also before the annealing point. As the training proceeds, the number of crashes caused by a randomly moving obstacle decreases and after some point there are no crashed episodes because of the random drone.

In Figure IV-7 the training results of the learner drone with 2 random drones are shown. The training curve shows similar trend as in Figure IV-6. However, there exist 79 episodes which crashed because of the random drones. The learner drone is hit by the random drones in these crashes. Most of the crashes occur at the beginning of the training where the random behavior is high. For example, almost half of the crashes happened before 20000 steps. After this point the learner drone starts exploring different parts of the environment where the random drones do not exist and thus there are no random crashed episodes until around 40000 steps. After that, the learner drone is hit by the random drones couple of times again but less than the beginning of the training. After a short time, the learner drone is starting to learn again to avoid obstacles and the mean reward values increase. The crashes caused by the random drones decrease through the end of the training and after some point the learner drone does not collide with anything.

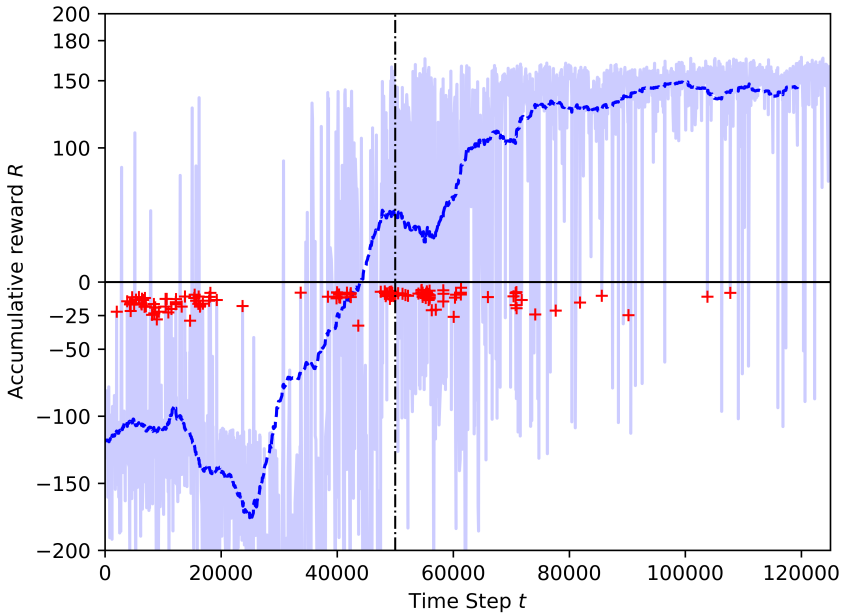


Figure IV-6: *Training results with random drone 1*

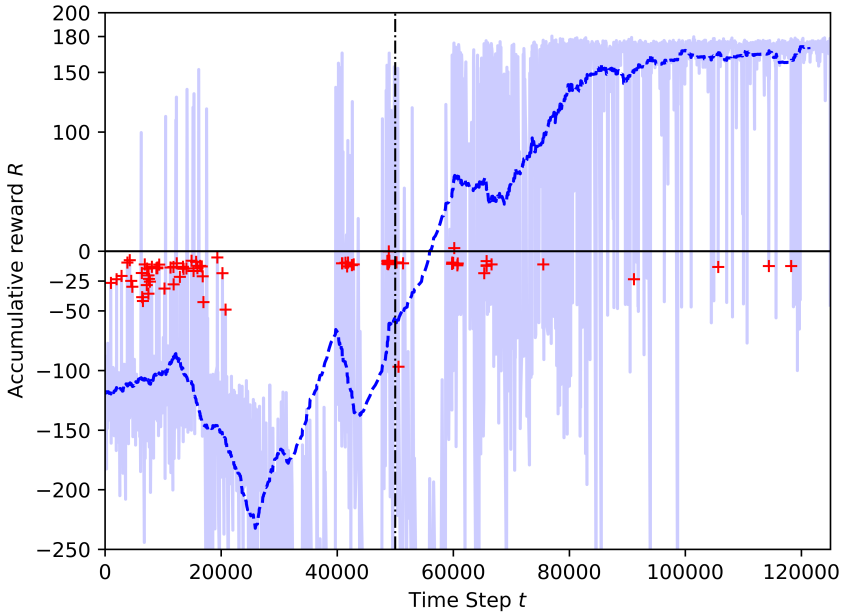


Figure IV-7: *Training results with random drones 1 & 2*

IV.6.2 Test Results

In this part, tests results of each trained model are shown. The tests are performed by using models created in each training section to observe how the agent can learn to avoid obstacles and the performance of the agent is assessed. These tests are made of 100 episodes starting from the take-off and ending by landing to the destination, by collision or by the time limit.

Test results are shown in Figure IV-8 and Figure IV-9. In Figure IV-8 two scenarios are shown. First scenario has only learner drone in the environment and the second scenario includes the learner drone with "random drone 1" in the environment. In these tests, the learner drone has reached the destination without crashing anything. As it is seen in these figures, the cumulative reward values follow almost a straight line and having few oscillations. The details about the reward values of the tests and the number of successful episodes can be seen in Table IV-3. The mean reward value for the model including the learner drone and the "random drone 1" is 163.49, lower than the mean reward for only one drone,learner drone, model, 172.63. This is because the learner drone has to deal with the moving obstacle which makes the movements more conservative but also slower in the second training session and the model has a reward lower than the reward in the first training model.

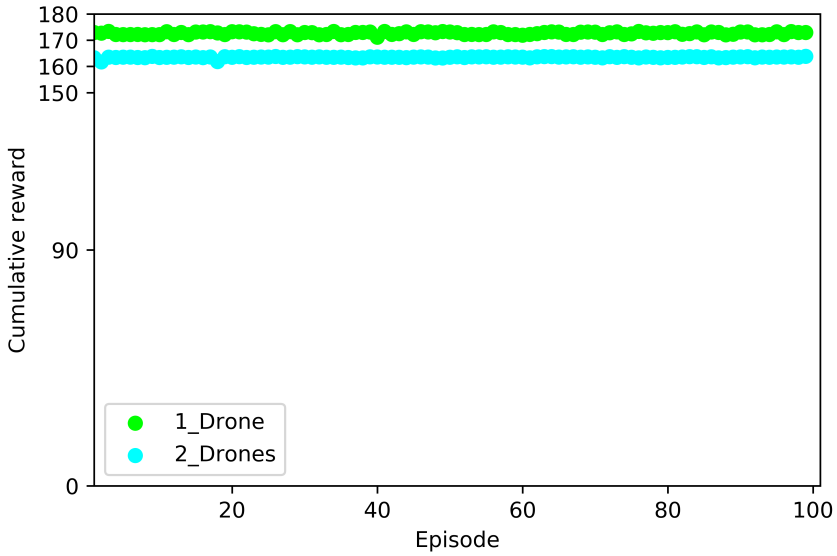


Figure IV-8: *Test results for only 1 drone and 2 drones*

In Figure IV-9 the test consists of 3 drones, the learner drone trained in the environment and random drones 1 & 2. In Figure IV-9, green dots represent the successful episodes, red dot represents the crash and yellow dots which represent the timeout limit meaning there is no crash or successful episode, but only reached a timeout. There are 98 successful episodes out of 100 episodes and only one crash and one timeout happened in the test. The reward values can be seen in Table IV-3. The mean reward value which is 153.69 is the lowest one when we compare with the other two tests. The reason can be more than one random drones exist in environment and the learning drone try to avoid them. Surprisingly, the maximum reward is the highest in all three tests.

The random behavior of the random drones which exist in the environment can change the path followed by the learner drone. For this reason, there are also successful episodes having reward values range between 90 and 150. The learner drone tries to avoid the random drones and the other stationary obstacles in the environment. Even if the reward values are lower, the learner drone successfully reaches the destination.

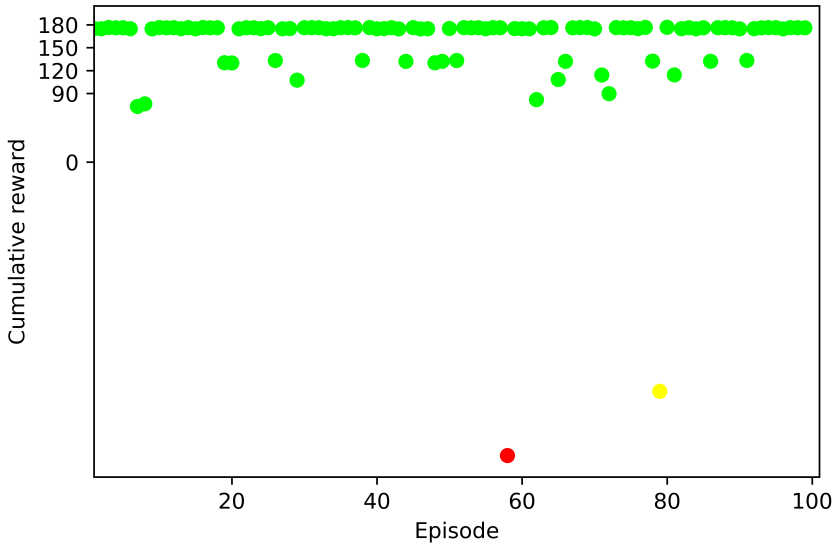


Figure IV-9: *Test results for 3 drones*

Table IV-3: *Model Tests Comparison*

Model	Mean reward	Max reward	Min reward	Success rate
1 Drone	172.63	173.35	171.13	100%
2 Drones	163.49	164.02	161.72	100%
3 Drones	153.69	176.76	-385.21	98%

IV.7 Discussion of Experimental Outcomes

A number of other research studies have proposed architectures that resemble the Joint Neural Network presented in this chapter, but they have not been used for RL state. For instance, the hybrid reward architecture in Van Seijen *et al.* (2017) used separate value functions and trained them separately, resulting in faster convergence but using separate neural networks. Srouji *et al.* (2018) tested a decomposition of the neural network into two streams in various environments including an urban driving simulator, but the training was not done jointly in a single network. The closest architecture to JNN is in Dosovitskiy & Koltun (2016), but it is applied to supervised learning instead of RL. This network combines three inputs - an image processed by a CNN, scalar measurements processed by two dense networks in parallel, and a goal - and merges the outputs into a single vector state.

DQN relies on a trade-off between exploration and exploitation, i.e., finding the optimal balance between trying new actions and exploiting the actions that have already been learned to be effective. This balance can be difficult to achieve in real-world environments. As the size and complexity of the environment increase, the computational requirements of DQN can become prohibitively expensive. This can limit the ability to use DQN in large-scale drone navigation applications.

A man who dares to waste one hour of time has not discovered the value of life.

— Charles Darwin

It's the job that's never started as takes longest to finish.

— J. R. R. Tolkien

V

Counter a drone in a 2D space

In this chapter, a deep reinforcement learning (DRL) architecture is proposed to counter a drone by using another drone in a 2D space. The defense drone will be called the learning drone and it will autonomously avoid all kind of obstacles inside a suburban neighborhood environment. The environment is in a simulator and has stationary obstacles such as trees, cables, parked cars, and houses. In addition, another non-malicious third drone, acting as moving obstacle inside the environment was also included. In this way, the learning drone is trained to detect stationary and moving obstacles, and to counter and catch the target drone without crashing with any other obstacle inside the neighborhood. The learning drone has a front camera and it can capture continuously depth images. Every depth image is part of the state used in DRL architecture. There are also scalar state parameters such as velocities, distances to the target, distances to some defined geofences, track angle, and elevation angle. The state image and scalars are processed and concatenated by a neural network in a similar way to what it is done in chapter IV. Moreover, transfer learning is tested by using the weights of the first full-trained model. With transfer learning, one of the best jump-starts achieved higher mean rewards (close to 35 more) at the beginning of training. Transfer learning also shows that the number of crashes during training can be reduced, with a total number of crashed episodes reduced by 65%, when all ground obstacles are included.

This chapter is based on the following publications:

- Çetin E, Barrado C, & Pastor E.. 2020. Counter a drone in a complex neighborhood area by deep reinforcement learning. *In: MDPI Sensors*. 20(8), 2320. D.O.I: 10.3390/s20082320.

V.1 Environment

From the number of AirSim environments available for AI research to experiment with deep reinforcement learning algorithms, a small urban neighbourhood is selected. The reason is that currently some drones operators are starting to operate in similar environments, which may be used by malicious drones to enter the area too, and to become a thread for its inhabitants. A counter-drone system is needed to avoid this not desired incomers. The tested neighborhood environment is shown in Figure V-2. A two-dimensional representation of the environment is shown by using the x and y axis at the origin point of the agent.

Figure V-1 shows the environment in the simulation, with the starting location of the three involved drones: The agent, also known as learning drone, in in the bottom of the image; On the top of the image we find the target drone, this is, the malicious drone that the agent has to catch; In between both some simulations include a third drone, named as random drone. This is used as a moving obstacle that the agent shall avoid. The target and the random drones move randomly from their starting point, inside the shaded areas of Figure V-1: red for the target drone (sized $25x8m$), yellow for the random drone (sized $10x8m$). Target drone and random drone can change positions up to 1 meter in each step. The learning drone is always started in the same location, $(0, 0, 0)$ in the NED coordinate system, but its yaw angle is random. This aims to increase the exploration capabilities of the learning drone from the first step.

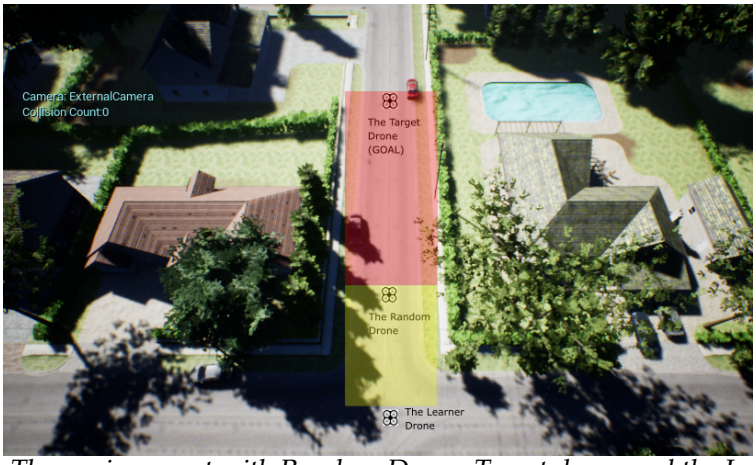


Figure V-1: *The environment with Random Drone, Target drone and the Learner Drone*

It is assumed that the counter-drone system must remain always within the neighbourhood that has contracted it. The limits of the area of the contract are given in the form of a geofence which limits ($155x90$ meter square) are shown in shaded blue region of Figure V-2.

Figure V-2: *Geofences in the environment.*

V.2 Drone Agent States

The state of the model is composed by an image and several auxiliary scalar values as it is explained previously in section IV.2. However, the image in this chapter has been improved and it includes additional information. The states are as follows:

- Image State
  Most drones have one or more cameras facing front, able to capture the objects situated in the flying direction. AirSim provides three virtual front cameras: visual, thermal, and depth. For the state we selected the image received from the depth camera. According to the AirSim Microsoft (2023a), the depth camera output is received by using Airsim API "Depth Perspective" image type, which simulates the return of a projection ray that hits its pixels. The depth image received from the camera is a $256x144$ pixels image as shown in Figure V-3. From this image we crop a central part and create the state image. This state image is set as $30x100$ pixels and is shown in Figure V-4. The bottom of the image includes the cropped central part of the depth image ($20x100$ pixels). Then, the top 10 rows of the state image is white (no obstacles), except for the $3x10$ pixels black line. This black line is used to represent the track angle, this is, the suitable direction to find the target drone Kersandt (2018). This $3x10$ pixels black line moves left and right according to the relative movements of the target and the catching drones.

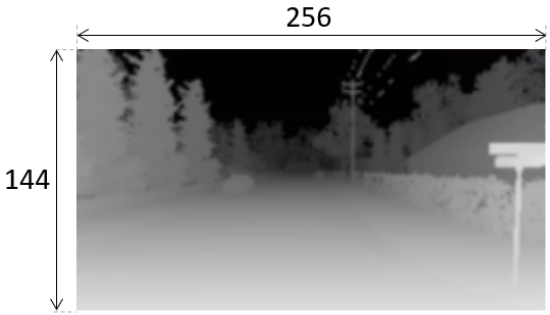


Figure V-3: *The Depth Image.*

Additionally, as Figure V-5 shows, a grid is drawn on top of the state image when the drone is close to cross the geofence. The thickness of the grid increases as the drone moves towards

the geofence limits. The grid appears when the separation distance between the drone and the geofence limits becomes lower or equal to 4 meters.

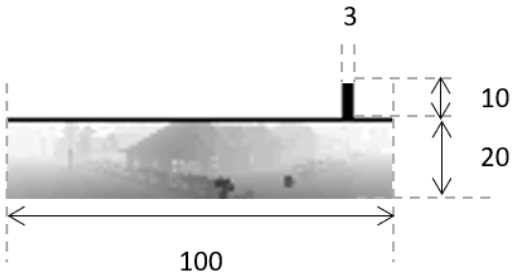


Figure V-4: *The State Image and Encoded Section*

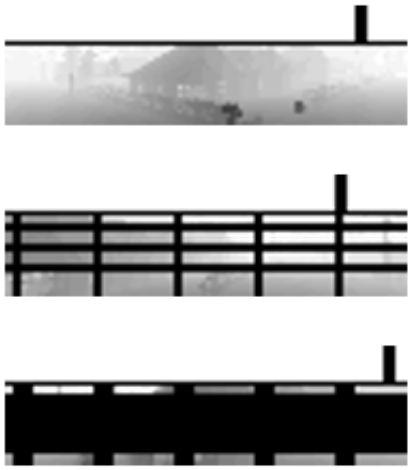


Figure V-5: *Fences drawn on the State Image.*

- Auxiliary Inputs
  Airsim capability to retrieve data of the environment is explained in Section III.2. For the purposes of our agent the following data is collected: the Euclidean distance, the track angle and the elevation angle of the target drone from the current position of the training drone. Other auxiliary data collected for the state is the distances to geofence limits. To summarize, the auxiliary data, aggregated to the state image as part of the agent state, is:

  – The velocity of the agent in x and y directions: $v_x v_y$
  – The distance from the agent to the goal in x and y directions and the Euclidean distance: $d_x d_y d_t$
  – Track and the elevation angles between the agent and the goal: $\psi\zeta$
  – The distances to the four geofence limits: (dg_{xmin}, dg_{xmax}, dg_{ymin}, dg_{ymax})

V.3 Agent's Neural Network

The full state, composed by the image and the auxiliary data, is processed with a neural network. The architecture of this neural network is shown in Figure V-6. The image is the input of a convolutional neural network (CNN), followed by a flatten layer. Then a concatenation layer joints the flatten output of the CNN with the scalar auxiliary data of the state.

The first layer of the CNN consists of RELU activated 32 kernel $4x4$ with stride 4. This layer is followed by RELU activated 64 kernel $3x3$ with stride 2. The output of the sequential CNN model is concatenated with the reshaped scalar values and the concatenated tensor becomes the input of three RELU activated consecutive 256 kernel dense layers. The output layer is a dense layer and the outputs are the action values. Neural network model summary can be seen in Figure V-6 and with more detail about the layers and their parameters in the Appendix Figure B-1.

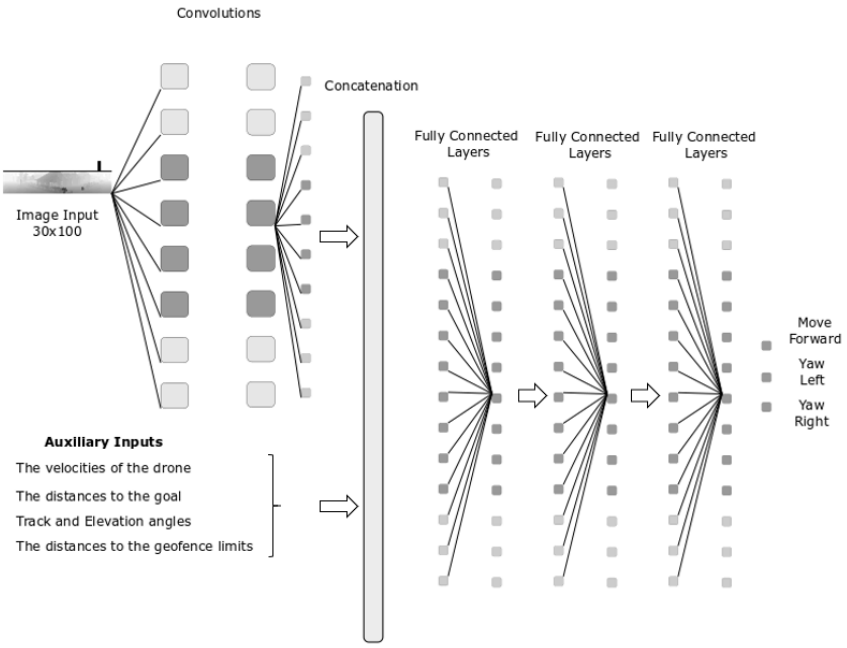


Figure V-6: *The Agent.*

V.4 Drone Control Actions

The drone agent is in the same plane in which the target drone is found, this is, without changing altitude during the training. In other words, target drone and the learner drone are in a 2D space. Figure V-7 illustrates the representation of the three actions and the description of the actions available are as follows:

- Straight: Straight movement in direction of the heading with speed equal to 4 m/s
- Yaw left: Rotate clockwise around z axis with a 30deg/s angular speed
- Yaw right: Rotate counter-clockwise around z axis with a 30deg/s angular speed

As a consequence, the output layer of the agent's neural network consists of three activation values, one for each possible action. The neural network will predict which of the three actions has a higher probability of obtaining the maximum cumulative reward.

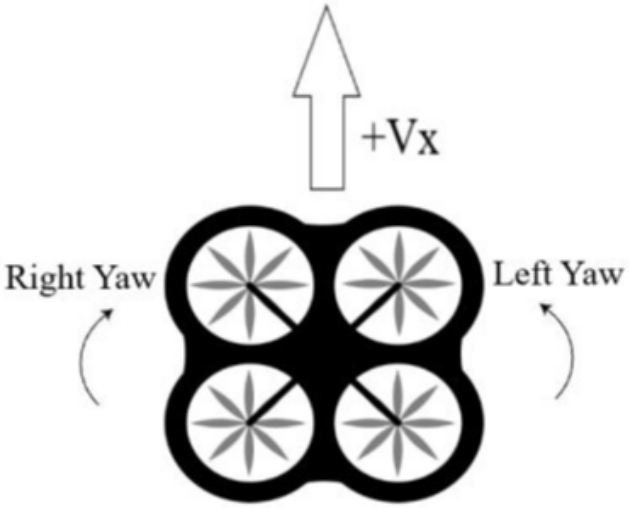


Figure V-7: *Action Space.*

V.5 Reward Function

Reward function is configured to allow agents learn and explore the environment and thus success in catching the target. Previously in section IV.5, reward function includes the innocent collisions which may not be caused by the learning drone. However, in this chapter, the agent is penalized with the minimum reward -100 if the collision occurs. The proposed reward function is shown in Table V-1. The agent is rewarded $+100$ if the episode is successful and the episode is terminated as it has ended by catching the target drone. On the other hand, the agent is penalized and it is given a reward -100 if the episode was unsuccessful, this is, it has ended because the agent had a collision with a visible obstacle of the environment or because had violated the geofence. Additionally, every intermediate step returns a reward of -1 to penalize delays on achieving the agent's objective. The reward of an intermediate step has two bonus: plus $\Delta Distance$, which is distance-to-the-goal reduction with respect to the previous step, and plus $trackangle$ which represents the zero-deviation towards the target direction.

Table V-1: *Rewards*

Reward	The Reason
+ 100	Goal reached
− 100	Collision: Obstacle (stationary or moving) or geofence
-1 + Δ Distance + TrackAngle	Otherwise

V.6 Definition of Training Cases

Several training experiments are defined and categorized into two groups. The first group of training experiments are performed at 30 meters height, above trees, houses and cars. In this first group the only obstacles in the environment are the geofence, the random drone and the target drone. The second group of training experiments are performed at low altitude, at an altitude of four meters. The main reason is that the drone can interact with all kind of obstacles such as trees, houses, and electrical wires found at this level. More detail about the obstacles

in the environment can be seen in videos from youtube[1]. In each group of training, cases with and without transfer learning are implemented to analyze and compare the performance of the models. All transfer learning models use a unique pre-trained model. The pre-trained model is built in the first presented case and named Baseline. A full training phase for the case named Baseline, lasted for $125,000$ steps, and spent around 48 to 56 hours. The same resources are used in both training and testing. The tests, run much faster, and are used to evaluate the learned capabilities of the agent training.

Table V-2 summarizes the different cases explained through the section.

Table V-2: *Training cases summary*

CASE	TRAINING	STEPS	ANNEALING	GEOFENCE	OBSTACLES
Baseline	FULL	125K	50K	YES	NONE
Case 1.1	FULL	75K	50K	YES	stationary 3rd drone
Case 1.2	Transferred	50K	25K	YES	stationary 3rd drone
Case 1.3	FULL	75K	50K	YES	non-stationary 3rd drone
Case 1.4	Transferred	50K	25K	YES	non-stationary 3rd drone
Case 2.1	FULL	125K	50K	YES	houses, trees, electrical, etc.
Case 2.2	Transferred	50K	10K	YES	houses, trees, electrical, etc.

V.7 Results

The following two subsections present the results: Training results in section V.7.1 and Testing results in section V.7.2.

V.7.1 Training Results

This section presents and analyzes the results of the experiments described in Table V-2.

V.7.1.1 Case 1: Training at 30 meters height

The following figures show the training performance relating the number of step (in the x-axis) with its cumulative reward (in the y-axis). The light blue represents the actual reward value of the step and the dark blue represents the mean rewards of the every 100 steps. The time steps are discrete and equal to one second. The vertical dotted line represents the end of the annealing training part. There is a linear epsilon-greedy exploration before the annealing points, starting from full random down to 10% random. After the annealing point, the 10% random is maintained until the end of the training.

The training episodes are finished when the drone catches the target or when the learning drone collides with any stationary or non-stationary obstacle (random drone) or when the learning drone overpasses the geofence limits.

- Baseline: Training including geofence and target drone
  In Figure V-8, the training results for 125K steps are shown. This training is set at 30 meters altitude, where both the learner drone and target drone are flying. This training is known as "Baseline" because it is used as a pre-trained model for training by transfer learning which

[1]https://www.youtube.com/watch?v=wFDGZANAcfQ&feature=youtu.be

is described in Table V-2. At the beginning of the training, the learner drone explores the environment and the rewards are mostly around -300 which is a very low reward. The main reason is that at the beginning of the training the random behavior is very high and the drone has not yet knowledge of how to catch the target, thus, it exceeds the episode time limit. However, the learning curve sets a higher slope at 20K steps and the cumulative reward reaches the highest values around 40K steps, before the annealing point. Although there are 61 episodes crashed during training, there is only one unsuccessful episode after annealing. This training shows that after a certain time the drone learns how to avoid geofence limits and how to catch the target as soon as possible.

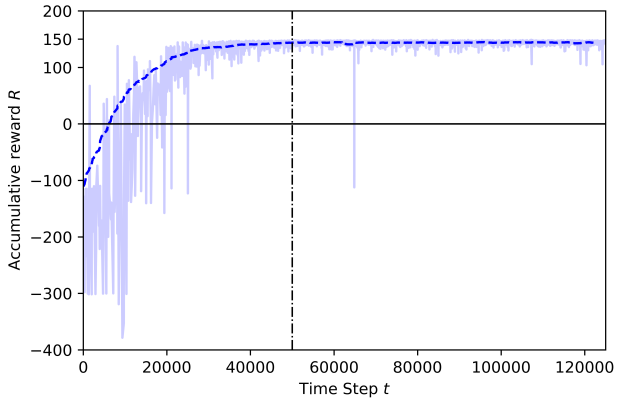


Figure V-8: *Training result for Baseline.*

- Case 1.1: adding a stationary third drone
  This training is set at 30 meters height with a stationary drone is placed in the environment at this same altitude, same as the learner drone and target drone. In Figure V-9 the full training results for 75K steps can be seen. At the beginning of the training, the learner drone explores the environment as it is seen in the previous training and the rewards are mostly around -300. The main reason is the same as before, a very high initial random behavior. However, this case has the stationary drone placed just 5 meters away from the learning drone. As a result, it is observed that the behavior of the learning drone is different than the training shown in Figure V-8. For example, in Figure V-9, it is observed that there are accumulative rewards around -100 and these are crashed episodes in this training because of the stationary drone placed in the environment. The learning drone eventually learns how to avoid this stationary drone but it takes a while. For example, the number of crashes against the random drone decreases after 65K steps. The cumulative reward reaches the highest values around 70K steps.

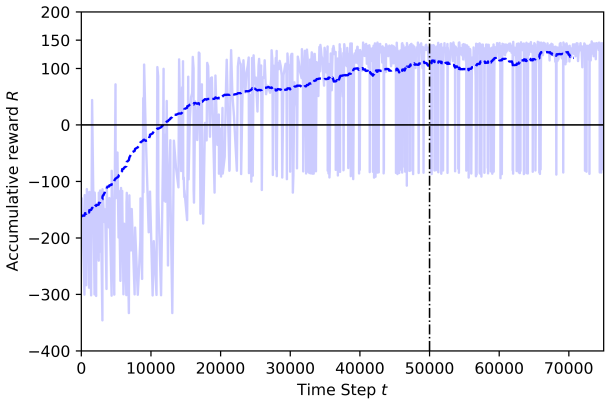


Figure V-9: *Training result for Case 1.1.*

- Case 1.2: adding a stationary third drone and using pre-trained model from Baseline
  In Case 1.2 the training is performed by using transfer learning. This training the Baseline pre-trained model, whose training results are shown in Figure V-8, is already trained to catch the target while avoiding the geofences, and the new training focus only on the new knowledge: the avoidance of the stationary drone. The training time is finished after 50K steps and annealing point is set at 25K steps in this training. The last layer of the model is frozen and the other layers are trained. In Figure V-10 the training results for transfer learning are shown. As it is seen in this figure, at the beginning of the training, the cumulative reward reaches positive values very fast, if compared to the training seen in Figure V-9. There are still some crashes but these are caused by the stationary drone and the high random behavior of the learning drone before the annealing point. After the annealing point, the learning drone is in general able to catch the target drone and to successfully avoid the stationary drone and the geofences.

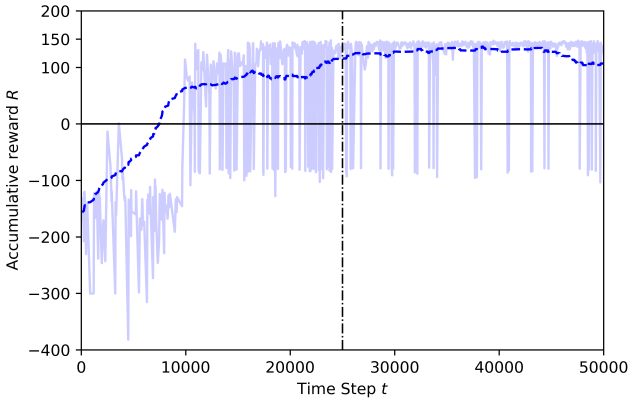


Figure V-10: *Training result for Case 1.2 by Transfer Learning.*

Figure V-11 shows the transfer learning metrics: there is no jump-start, but the threshold time can be observed. Both training start their curve with -160 mean reward. However, during the training with TL, the agent reaches a pre-specified performance level faster (at around 30K steps) than the model without TL. The asymptotic performance level is zero at the end of 50K steps.

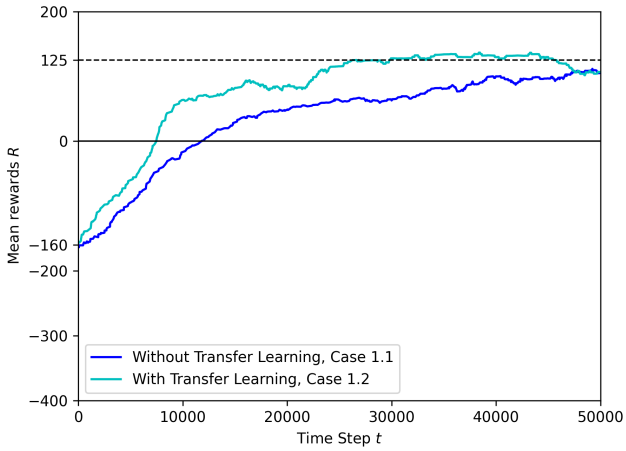


Figure V-11: *Training mean rewards for Case 1.1 and Case 1.2.*

- Case 1.3: adding movement to the third drone
  In this case, a third drone moving randomly is placed into the environment, at the same altitude than the learner drone and target drone. In Figure V-12 the training results for 75K steps can be seen. At the beginning of the training, the rewards are mostly around -300 because the learning drone still explores the environment. It is observed how the behavior of the drone changes before and after identifying the non-stationary third drone. Before, the mean rewards curve goes up until 20K steps since the drone seems to learn how to avoid geofence and the non-stationary drone at the beginning, but after the curve starts going down for a while and the rewards are mostly around -300 which are mostly considered time-limit. The main reason is that the drone starts exploring again until catching the target drone. After annealing, the mean rewards looks stable, but there are still episodes that crash because of the non-stationary drone moving randomly.

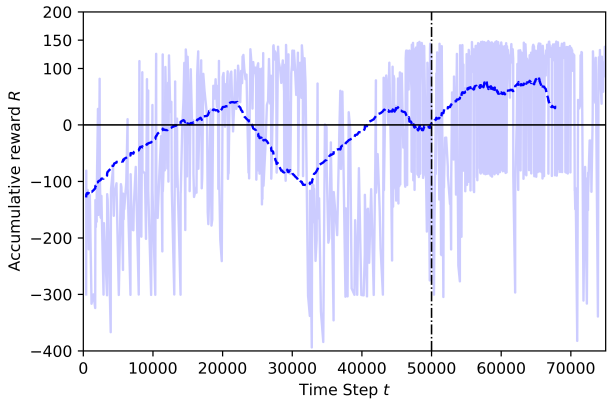


Figure V-12: *Training result for Case 1.3.*

- Case 1.4: adding movement to the third drone and using pre-trained model from Baseline
  In this training, the Baseline model, which is shown in Figure V-8, is used as the pre-trained model of transfer learning, to train the learning drone to avoid geofences and the non-stationary drone and to catch the target drone. The last layer of the model is frozen and the other layers are trained. The training time is 50K steps and annealing point is at 25K steps. In Figure V-13, the training result for transfer learning is shown. At the beginning of

the training, the cumulative rewards start at -115 and but reach high values after the annealing point. The cumulative reward is more stable during the training with TL compared to the training seen in Figure V-12. This is because the crashes caused by the non-stationary drone after annealing point in the second case. Thanks to transfer learning, the number of crashes with the geofence are reduced by almost 75%. The non-stationary drone is a hard challenge for the agent, because it can hit the agent during the training and thus the learning takes longer.

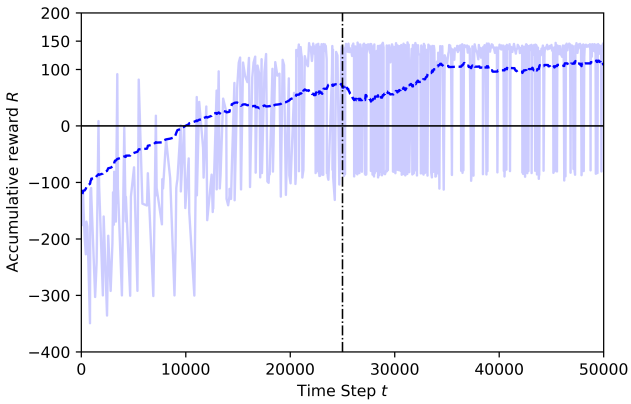


Figure V-13: *Training result for Case 1.4 by Transfer Learning.*

Figure V-14 compares the reward curves for training with and without the transfer learning. Training without transfer learning is not smooth. Both curves start the training with mean reward around -115. However, with TL, the agent reaches a pre-specified performance level, which is at 100 mean reward, and does it faster (at around 35K steps), as expected. On the contrary, without TL the curve is not stable, with many up and downs caused by the unexpected random movement of the non-stationary drone. Since the target drone have the same image and also moves randomly, the situation creates confusion and the full training is not successful.

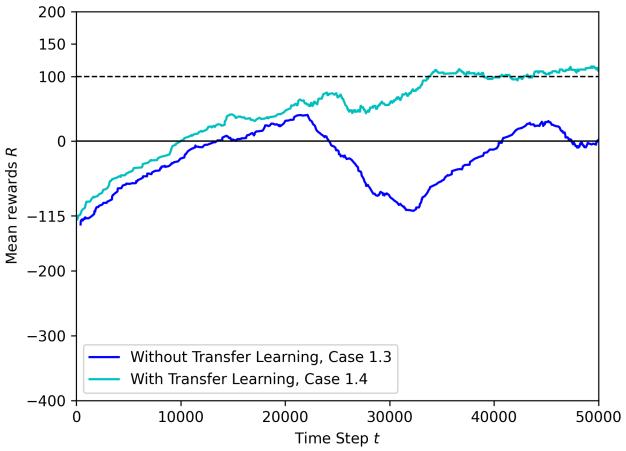


Figure V-14: *Training mean rewards for Case 1.3 and Case 1.4.*

V.7.1.2 Case 2: Training at low altitude, with many obstacles

- Case 2.1: without Transfer Learning
  In Figure V-15 the results for 125K steps full training are shown. The training is set at 4 meters altitude and in addition to the obstacles, such as trees, houses, power cables and cars, the learning drone and target drone are added to the environment. At the beginning of the training, the learning drone explores the environment and the rewards are mostly around -300. This is a very low reward during all training session in this case. However, the learning curve goes up after 25K steps and the cumulative reward reaches higher values around 35K steps, before the annealing point. This training shows that after a certain time, the drone learns how to avoid all kind of obstacles including geofence limits, and how to catch the target as soon as possible.

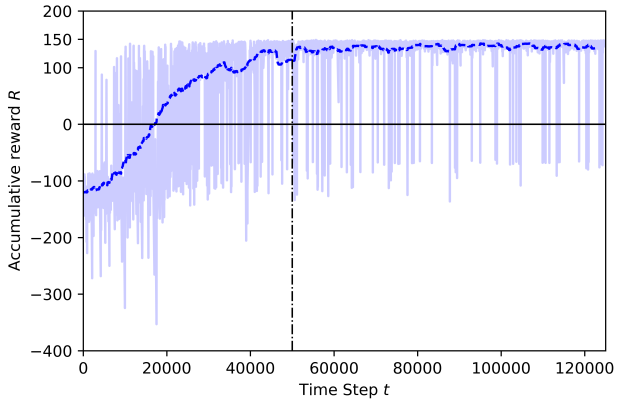


Figure V-15: *Training result for Case 2.1.*

- Case 2.2: with Transfer Learning, using pre-trained model from Baseline
  In this training, the Baseline model, shown in Figure V-8, is used as pre-trained to transfer to the learning drone the knowledge on how to catch the target drone. The last layer of the model is frozen and the other layers are trained. The training time is set to 50K steps and the annealing point to 10K steps. In Figure V-16 the training results for transfer learning are shown. After the annealing point, as shown in Figure V-15, the training with TL shows better results compared to the training without TL.

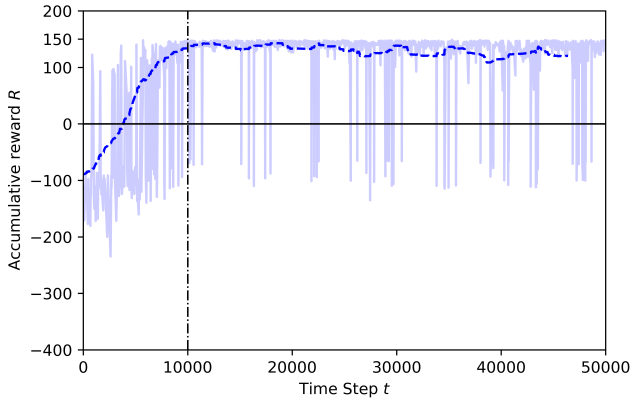


Figure V-16: *Training result for Case 2.2 by Transfer Learning.*

In Figure V-17 the transfer learning metrics are shown in terms of jump-start and threshold time. TL starts training with mean reward around -85 while the training without TL is

worse, around -120. The jump-start achieved is almost 35 more mean reward with transfer learning. Moreover, the TL also reaches a pre-specified performance level faster (at around 10K steps), while the model without TL reaches the threshold point just after 40K steps.

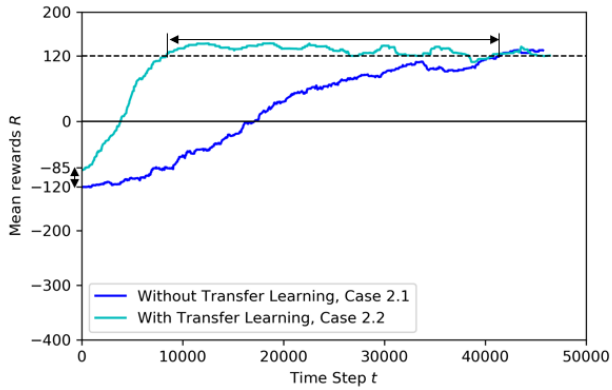
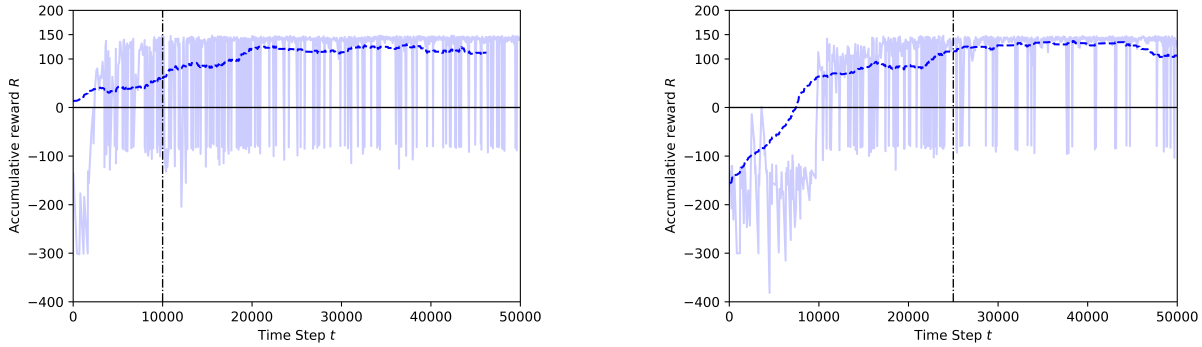


Figure V-17: *Training mean rewards for Case 2.1 and Case 2.2.*

V.7.1.3 Training Results in Different Annealing Points

Transfer learning training with different annealing lengths are compared for Case 1.2. The linear annealing policy is the same in all cases (from 1 to 0.1 randomness). However, different responses are found for different annealing points. For example, in Figure V-18(a), it is seen that the annealing starts at 10K steps and the total training covers 50K steps. Before the annealing, the agent learns slowly and reaches higher rewards in 20K steps, but there are still crashed episodes. However, in Figure V-18(b) the annealing is set at 25K steps. Although mean rewards are very low at the beginning of the training, after the agent explores more the environment, for around 15K steps, and it is able to reach high reward values. After the annealing, there are still crashes but the number of crashed episodes is lower than the crashed episodes compared to Figure V-18(a).



(a) Training result for Case 1.2, Annealing at 10K steps.

(b) Training result for Case 1.2, Annealing at 25K steps.

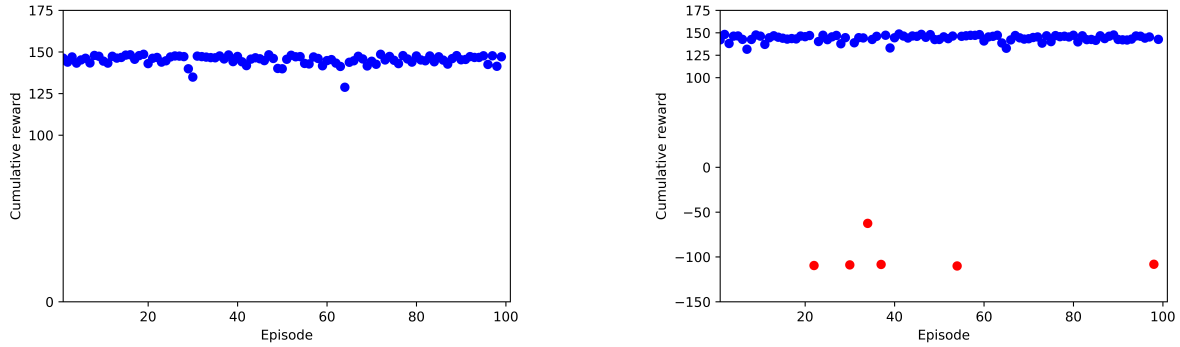
Figure V-18: *Training Results for Different Annealing*

V.7.2 Test Results

V.7.2.1 Testing the Models at Low Altitude

In this section the tests results for Case 2.1 and Case 2.2, and are discussed. During a test, the agent is not learning anymore, but applies the learnt model with no more random behaviour. For this reason in this section we will not call the agent the learning drone, but the agent drone.

Each test set is made of 100 episodes, starting from the take-off and ending by catching the target drone (successful), by colliding (failure) or by time-out. Failures can happen by crashing with a visible ground obstacle (tree, house, wires, poles, etc.) or with the virtual geofence.



(a) Test Result for Case 2.1 (Without TL)

(b) Test Result for Case 2.2 (With TL)

Figure V-19: *Test Results*

Test results are shown in Figure V-19(a) for full training (case 2.1), and in Figure V-19(b) for transfer learning (case 2.2). In all 100 episodes of case 2.1, the agent drone was successful and able to catch the target drone without any crash. The cumulative reward plot is almost a straight line, with a few oscillations. On the contrary, with transfer learning (case 2.2) the agent drone crashed 6 times out of 100 episodes. The main reason for these crashes is that the agent drone had learned only some of the obstacles in the environment, and thus, it missed some of the distant obstacles. However, even if there were failure episodes, the transfer learning showed a 94% success rate, which is good performance when considering the short time spent on training.

The results demonstrated how the learning process can be improved by step-by-step learning. Initially, the drone learns the basic objective of its mission: head towards the target drone while moving inside an invisible geo-cage. Then, new secondary objectives can be further introduced using transfer learning. Our additional objectives were to avoid colliding with another (non-malicious) drone, or to avoid multiple but fixed obstacles (houses, trees, electrical wires, etc). Transfer learning showed much better performance that starting a longer full training: It was faster in reaching a threshold reward and it did with a higher asymptotic performance.

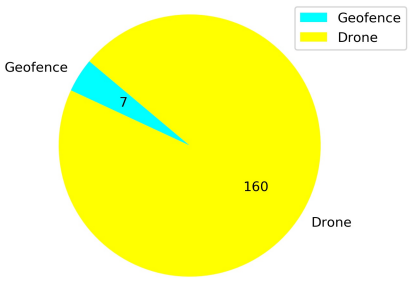
With the expansion of drones flying in the airspace, the availability of an effective counter-drone technology is a must. This counter-drone technology consists on several systems, on ground and on air, from which the solution presented in this thesis is just one part. For this to work, it is necessary to have a support system that detects the target drone, classifies its activity as malicious, and estimates the position to the target drone. The results also show that deep reinforcement learning is a promising approach for the interception of the target drone moving randomly. However, the full system is still to be developed. In particular the final interception method, which is here achieved by crashing into the target drone, could use a more sophisticated approach to capture it.

V.8 Discussion of Experimental Outcomes

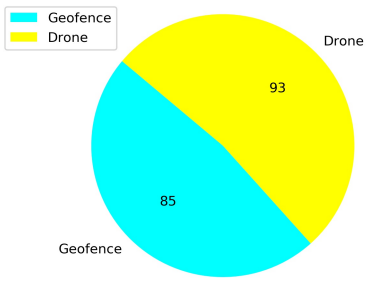
In this section, the explainability of deep reinforcement learning is focused and the charts related to drone positions and crash reports are discussed.

V.8.1 Comparison of the effects of different annealing points in TL

Previously, training results for different annealing points are presented in Figure V-18. In this section, crash reports are analyzed and discussed further with explainable AI. Figure V-20(a) shows that from a total of 167 crashed episodes by annealing at 10K steps, only seven episodes crashed with the geofence while 160 episodes crashed with the stationary third drone. In Figure V-20(b) the number of crashed episodes and their crashed obstacles are shown for annealing at 25K steps. There are 178 crashed episodes in total, 85 episodes crashed on geofence and 93 episodes crashed on drone. The total number of crashed episodes are slightly higher than before, but the number of episodes crashed on stationary drone are reduced in half. The main reason is that a longer annealing allows the agent to explore more at the beginning of the training, learning about both type of obstacles at the same time.



(a) Crash report chart for Case 1.2, Annealing at 10K steps.



(b) Crash report chart for Case 1.2, Annealing at 25K steps.

Figure V-20: *Crash Report Charts for Different Annealing*

V.8.2 Comparison of the explored areas with or without TL

In this section, a map was built to show the drone positions during all the training steps. As in the previous comparison, the details of the crashed obstacles are also given. The map of the drone positions is limited by the coordinates of the geofence, these limits are [-5, 150] for the x-axis and [-70, 20] for the y-axis. The blue dots represent the agent position in each time step during training. The red dot is the initial position of the target drone.

In Figure V-21 the learning drone position map for Baseline is shown. It can easily be seen that the agent drone learns how to focus on targeting the goal, avoiding exploring areas that do not face the target. Also it is observed that geofence limits are not approached.

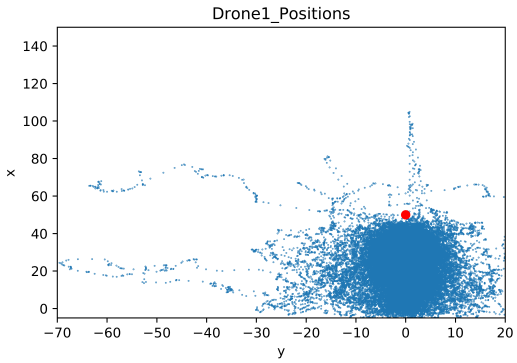
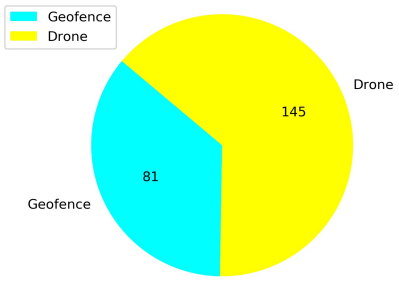


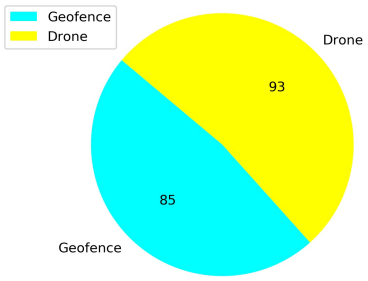
Figure V-21: *Drone Position Map for Baseline*

In Figure V-22(a) the number of total crashes and the crashed obstacles for the Case 1.1 are shown. As it is seen in Figure V-22(a), there are 145 episodes crashed against the stationary drone. Moreover, it is observed that the number of crashes with the geofence are also higher in this training. The geofence limits are exceeded 81 times in this training, although it was 61 in the training without adding the stationary drone shown in Baseline.

In Figure V-22(b) the number of total crashes and the crashed obstacles for Case 1.2 are shown. As it is seen in Figure V-22(b), there are 93 episodes crashed against the stationary drone. Moreover, it is observed that the number of crashes with the geofences is 85. The geofence limits are violated four times more with transfer learning compared with the full training of Case 1.1.



(a) Crash report chart for Case 1.1.



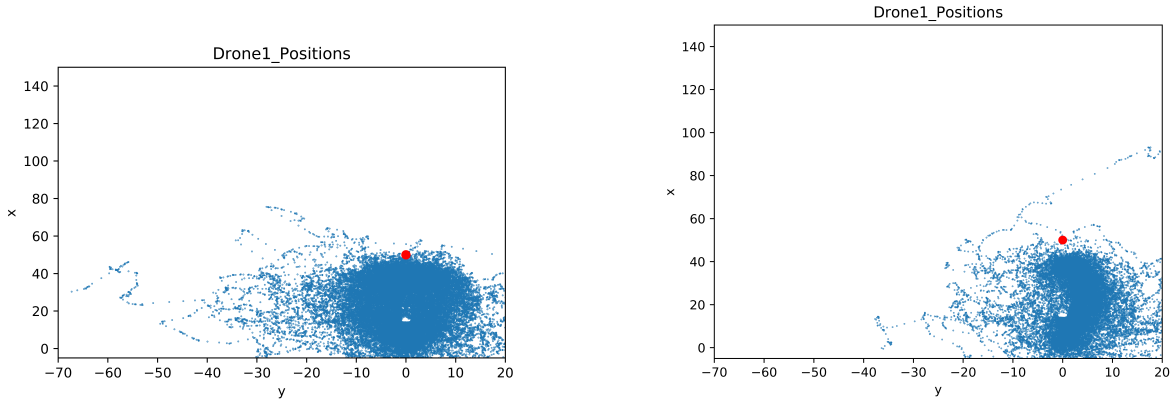
(b) Crash report chart for Case 1.2 by Transfer Learning.

Figure V-22: *Crash Report Charts for Cases 1.1 and 1.2*

Figures V-23(a) and V-23(b) show maps of the learning drone position for Case 1.1 and Case 1.2 respectively. Observe that with transfer learning (Figure V-23(b)) the drone is mostly directed to the target which is visually indicated by a red-colored dot, while without transfer learning, the drone is distracted and moves far away from the goal (see Figure V-23(a)).

In Figure V-24(a) the number of the total crashes and the crashed obstacles for Case 1.3 are shown. There are more crashed episodes in Case 1.3 compared to the other cases because of the non-stationary drone. There are 193 episodes crashed this drone. Also, the number of episodes of crashing with the geofence(150) is high compared to the cases before. In summary, the learning drone does not learn how to avoid this non-stationary drone in a considerable time as expected in this case.

In Figure V-24(b) the number of total crashes and the crashed obstacles for Case 1.4 are shown. A similar number of episodes failures (200) are also due to the non-stationary drone. How-



(a) Drone Position Map for Case 1.1.

(b) Drone Position Map for Case 1.2 by Transfer Learning.

Figure V-23: *Drone Position Maps for Cases 1.1 and 1.2*

ever, the crashes with the geofence (45) is lower than in Figure V-24(a) without transfer learning.

(a) Crash report chart for Case 1.3.

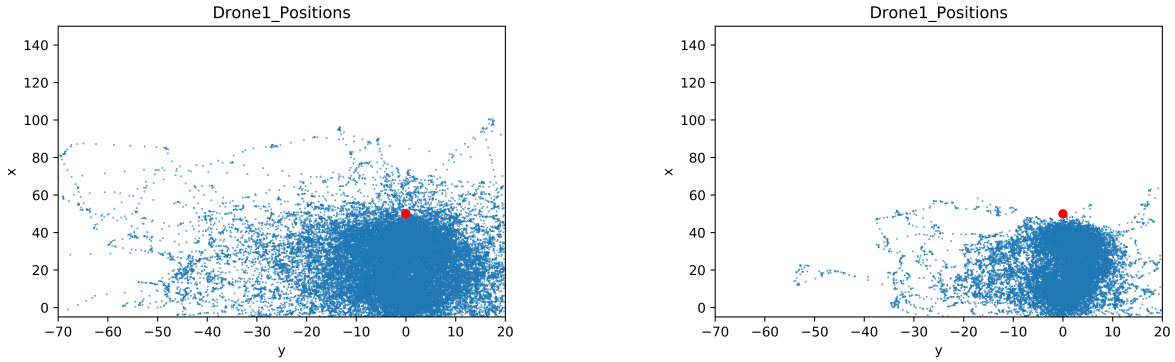
(b) Crash report chart for Case 1.4 by Transfer Learning.

Figure V-24: *Crash Report Charts for Cases 1.3 and 1.4*

In Figure V-25(a) and Figure V-25(b) the maps of the learning drone positions during the training sessions can be seen. With transfer learning (Figure V-25(b)) the drone is better focused on the target, while without transfer learning, the drone is moving left and right side of the environment in order to find a way to avoid non-stationary drone (Figure V-25(a)) but fails in reaching its goal.

In Figure V-26(a) and Figure V-26(b) the number of total crashes and the crashed obstacles for Case 2.1 and Case 2.2 are presented respectively. Although the number of crashed episodes is 177 for the transfer learning Case 2.2, the number is even higher (503 crashed episodes) in Case 2.1. The total number of crashed episodes was reduced by 65% with transfer learning. The crashed obstacles are categorized as geofences, trees, power lines, and houses.

In Figure V-27(a) and Figure V-27(b) the learner drone position maps in the environment during the training sessions are shown. In both figures, the shape of obstacles can be observed. For example, in Figure V-27(a), the learner drone tries to explore the environment by moving around the obstacle, as seen with a white rectangular shape positioned in the y-direction within the range of $[-30, -10]$. In addition, one of the trees (in front of the house) can also be seen as a white area surrounded by many blue dots. In Figure V-27(b) the shape of the house can also be observed, but with transfer learning the learning drone does not need to explore all around the



(a) Drone Position Map for Case 1.3

(b) Drone Position Map for Case 1.4 by Transfer Learning.

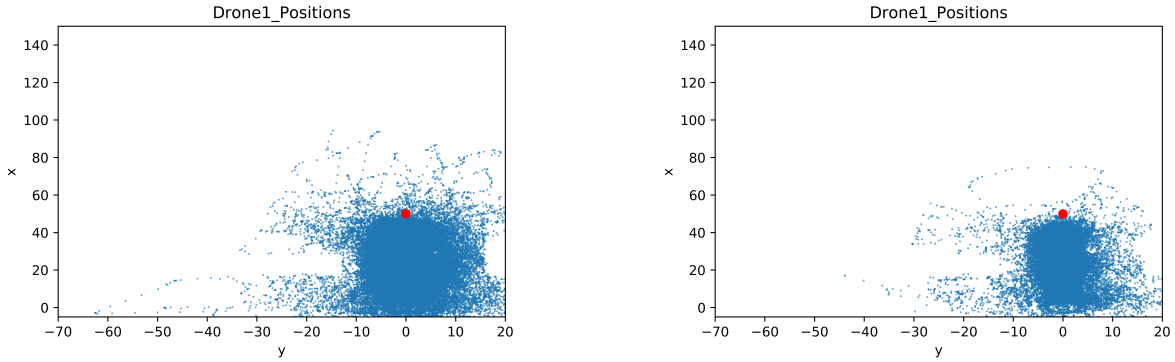
Figure V-25: *Drone Position Maps for Cases 1.3 and 1.4*

(a) Crash report chart for Case 2.1.

(b) Crash report chart for Case 2.2 by Transfer Learning.

Figure V-26: *Crash Report Charts for Cases 2.1 and 2.2*

house.



(a) Drone Position Map for Case 2.1

(b) Drone Position Map for Case 2.2 by Transfer Learning.

Figure V-27: *Drone Position Maps for Cases 2.1 and 2.2*

The good thing about science is that it's true whether or not you believe in it.

— Neil deGrasse Tyson

Sorprenderse, extrañarse, es comenzar a entender.
[To be surprised, to be amazed, is to begin to understand.]

— José Ortega y Gasset

VI

Counter a drone in a 3D space

In a 2D space it is already shown that the deep reinforcement learning method is an effective way to counter a drone. However, countering a drone in a 3D space with another drone is a very challenging task considering the time required to train and avoid obstacles at the same time. In this chapter, a Double Deep Q-Network (DDQN) algorithm with dueling network architecture and prioritized experience replay is presented to catch another drone in the environment provided by an Airsim simulator. The models have been trained and tested with different scenarios to analyze the learning progress of the drone. Experiences from previous training are also transferred before starting a new training by pre-processing the previous experiences and eliminating those considered as bad experiences. The drone detection model running in the background is also implemented in most of the models, and one of the best models is obtained by using the drone detection model running in the background. The results show that the best models are obtained with transfer learning and the drone learning progress has been increased dramatically.

This chapter is based on the following publications:

- Çetin E, Barrado C, & Pastor E.. 2022. Countering a Drone in a 3D Space: Analyzing Deep Reinforcement Learning. *In: MDPI Sensors*. 22(22), 8863. D.O.I: 10.3390/s22228863.
- Çetin E, Barrado C, & Pastor E.. 2021. Improving real-time drone detection for counter-drone systems. *In: The Aeronautical Journal*. 125(1292), 1871-1896. D.O.I: 10.1017/aer.2021.43.

VI.1 Environment

The urban neighborhood is chosen as it is done in previous chapters IV and V to counter a drone because of the similarity in real-life experiences in urban areas. The details of the environment is explained in Section III.2. Also, all the models starts at the same location in $y - direction$ and it is at $y = 0$. Target locations in $z - direction$ are also same for all the models and it is fixed at 5 meters above the ground $z = -5$ but the target locations in $x - direction$ have been changed to challenge the agent during training and test. The orientation of x-y-z directions are presented in Figure VI-1.

Figure VI-1: *Environment Setup x-y-z Directions.*

VI.2 Drone Agent States

Agent states consist of images and scalar input values which are concatenated later. However, different image states are used in two different DRL models.

VI.2.1 Drone Agent Scalar State Inputs

Scalar inputs contain the agent's distances to the goal in x, y and z directions and the Euclidean distance $d_x d_y d_z d_t$. The scalar inputs are concatenated with an image state input with or without drone detection.

VI.2.2 Image State without Drone Detection

The depth image seen in Figure VI-2, with 256 $\times$ 144 pixels captured continuously. This image is the default size that Airsim can output.

Figure VI-2: *Depth Image.*

VI.2.3 Image State with Drone Detection

Depth image shown in Figure VI-3(a), 84 × 84 pixels, and scene image, 256 × 144 pixels, are captured by using a drone onboard cameras. The scene image seen is processed by the drone detection model to create bounding boxes when the target drone is detected on the image as shown in Figure VI-3(b). After merging the images, the bounding box region in the depth image is masked shown in Figure VI-3(c). Previously, the circles are drawn in black in chapter V. Nevertheless, in this chapter bounding box are replaced with white to help the agent find the target easily since dark colors in depth images can be mistaken for obstacles. The final image used in the DRL model can be seen in Figure VI-3(d).

(a) Depth Image (b) Prediction (c) Image Masked (d) Final State

Figure VI-3: *Drone Detection and Image Processing.*

VI.2.4 Geofences on Image State

In addition, the grid is drawn on the image in both cases VI.2.2 and VI.2.3 if the drone comes closer to the geofence limits in all directions. The grids start to be drawn on the image when the distance between the drone and the geofence limits lower than or equal to 1 m in steps. The thickness of the grid increases as the drone moves towards the geofence limits. The grid which is drawn on the image is illustrated in Figure V-5 when the drone moves in a 2D Space. Additionally, the grid is drawn for 3D space if the agent is closer to the top of the geofences or closer to the ground. In Figure VI-4(a), the grid is drawn if the agent is closer to the geofences on top. On the other hand, in Figure VI-4(b), the grid is drawn on the bottom of the image if the agent is closer to the ground.

(a) (b)

Figure VI-4: *Fences in Image State. (**a**) Grid on Top of the Image. (**b**) Grid on the Bottom of the Image.*

VI.3 Agent's Neural Network

The deep reinforcement learning model is constructed by using dueling network architecture and trained with DDQN including prioritized experience replay. The image is an input of a convolutional neural network (CNN), followed by a flatten layer and then a concatenation layer joints the flatten output of the CNN with scalar inputs. Figure VI-5 shows the neural network model representation including dueling architecture. In this figure, only simple state image is shown. Also, dueling architecture has outputs: state-value V and the advantage for each action A.

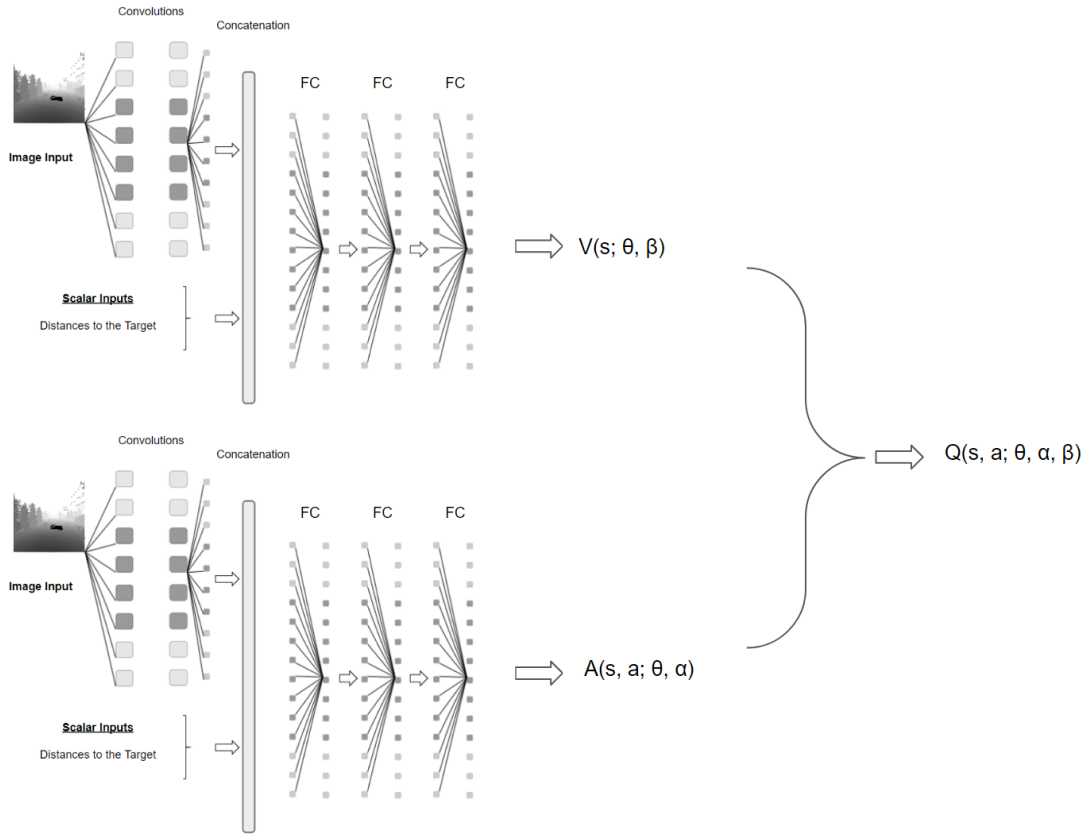


Figure VI-5: *Agent Dueling Architecture.*

VI.4 Drone Control Actions

The agent can select five different actions such as moving forward, yawing left and right, and going up and down. The actions are represented in detail in Table VI-1 and in Figure VI-6.

Table VI-1: *Actions.*

Action	Movement
0	2 m/s in +x direction
1	30 deg yaw left
2	25 deg yaw right
3	0.25 m/s in +z direction
4	0.25 m/s in −z direction

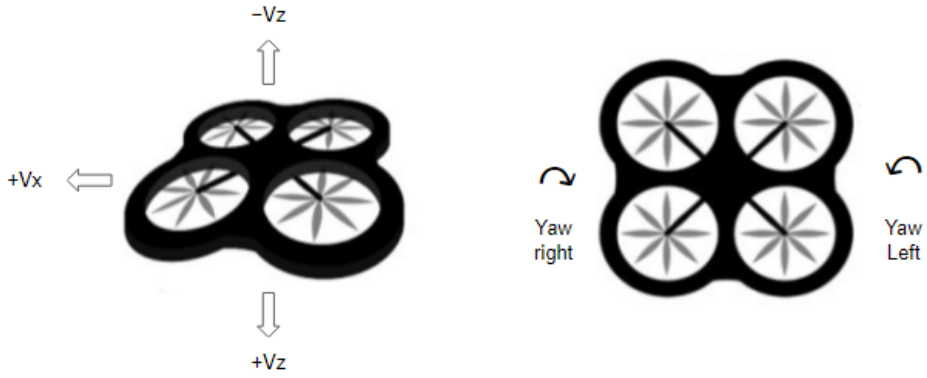


Figure VI-6: *Agent Actions.*

VI.5 Reward Function

The reward function includes incremental rewards which penalize the agent during the episode and the reward giving a successful episode. Incremental rewards shown in Table VI-2 are given to the agent such as $[-1 : -4]$ during an episode in every 50 steps. In addition, an intermediate step reward is added: $\Delta Distance$ which represents the change of distance to the target between the current step and the previous step. In other words, if the $\Delta Distance$ is higher, the agent is penalized more. Finally, the time limit is also set to restart the episode after 180 steps and the reward of the agent becomes the the sum of rewards at the end of the 180th step. In this paper, collision penalization for colliding with any obstacle in the environment is not implemented. The reward function is shown in Table VI-2.

Table VI-2: *Rewards.*

Reward	The Reason	Step Interval
$+100$	Target Caught	End of the Episode
$-1 + \Delta Distance$	Incremental Advancement	Episode steps between 0–50
$-2 + \Delta Distance$	Incremental Advancement	Episode steps between 50–100
$-3 + \Delta Distance$	Incremental Advancement	Episode steps between 100–150
$-4 + \Delta Distance$	Incremental Advancement	Episode steps between 150–200

VI.6 Definition of DRL Models

Nine models are trained by implementing different scenarios such as different target drone locations, teleportation and random heading at the beginning of each episode during training. Teleportation is that the agent starts the episode in different locations around the target to enhance the exploration. All models described in Table VI-3, are trained with DDQN algorithm with prioritized experience replay. In addition, some models have been trained by implementing transfer learning and using different network architectures such as a dueling network. Different annealing sections and the total training times are also investigated and shown in Table VI-3. State of the DRL models are already explained in section VI.2.1 and all models have the same kind of scalar states such as distances to the target.

Table VI-3: *Setup DRL Models.*

Models	Teleportation and Random Heading	Transfer Learning	Dueling Network Architecture	Image State	Annealed Steps	Training Steps	Drone Detection
Model-1	NO	NO	YES	(256,144)	15,000	48,540	NO
Model-2	NO	YES	YES	(84,84)	20,000	55,032	YES
Model-3	NO	NO	YES	(84,84)	15,000	24,795	YES
Model-4	NO	NO	YES	(84,84)	20,000	72,634	YES
Model-5	NO	NO	YES	(84,84)	15,000	49,999	YES
Model-6	NO	NO	NO	(84,84)	15,000	45,253	YES
Model-7	YES	NO	NO	(84,84)	100,000	42,060	YES
Model-8	YES (Random Heading)	NO	NO	(84,84)	50,000	103,947	YES
Model-9	NO	NO	NO	(84,84)	15,000	33,901	YES

VI.7 Results

In this section training and test results are presented. Models are trained on a desktop PC with NVIDIA GeForce RTX 3060 Ti with 8 GB VRAM graphics co-processor. Previously in chapter III in Figure III-5, core components of the experimental setup and the interactions between the DRL tools such as Tensorflow, Keras and OpenAI Gym, drone detection model via python pipe which accomplishes the parallel processing and the simulation are presented. The linear Epsilon-greedy policy is applied during the training. Different training steps and the annealed part of the training section are implemented to train DRL models. In addition, some of the models are also trained by loading experiences from another training. In general, full training with 75,000 steps can take approximately 48 h but training time can vary depending on the model. A summary of the models is presented in Table VI-3 and Hyperparameters of the training and tests are presented in Table C-1.

VI.7.1 Training Results

In order to observe each model separately in Appendix C, all the training curves are presented individually. These individual training curves can be found in Figure C-1. Mean rewards of DRL models are presented together in Figure VI-7. It is seen that only model-2 has positive mean rewards at the beginning thanks to transfer learning by loading experiences from the previous training. On the other hand, it can be seen in this figure that Model-7 is very slow and does not reach a positive mean reward in training while the other models reach positive mean rewards after some time in training.

In Table VI-4 maximum, minimum and average cumulative rewards of DRL models are presented. Success rates during training are also shown. Model-1 and model-2 have the maximum

success rates and maximum average cumulative results. Model-3 and Model-7 has the minimum success rates.

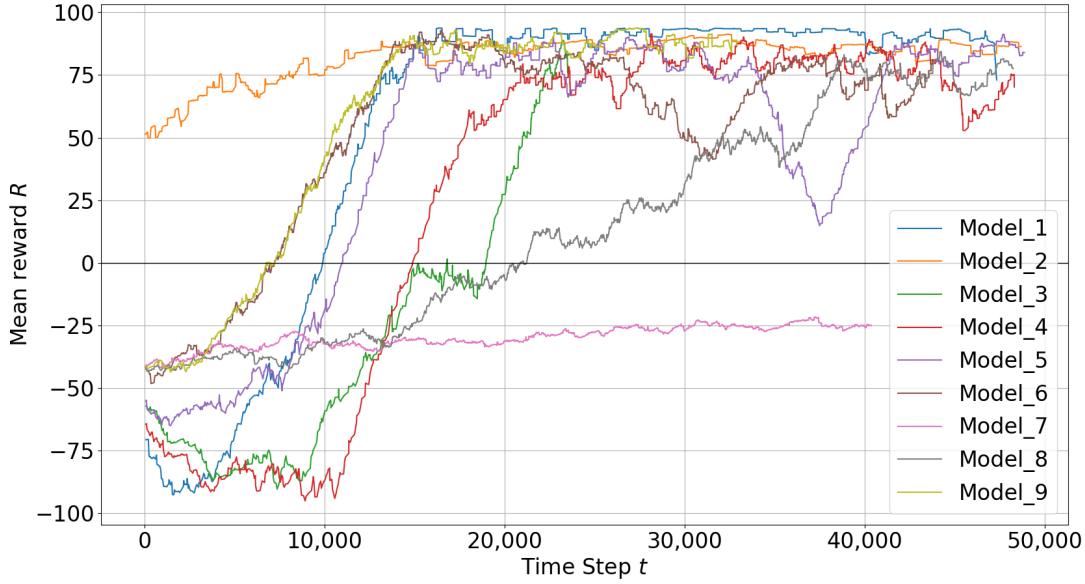


Figure VI-7: *Training Results ALL Models.*

Table VI-4: *DRL Models Training Rewards Statistics.*

Models	Average Cumulative Reward	Max. Cumulative Reward	Min. Cumulative Reward	Success Rates
Model-1	83.11	94.48	−429.27	95%
Model-2	83.82	94.12	−420.99	98%
Model-3	−3.73	97.33	−426.16	47%
Model-4	65.91	98.09	−427.04	88%
Model-5	65.24	96.16	−424.30	88%
Model-6	64.49	98.21	−416.40	83%
Model-7	−29.40	96.25	−252.38	5%
Model-8	71.80	97.16	−421.06	86%
Model-9	72.08	97.14	−418.69	85%

VI.7.1.1 Selection of Best Models Candidates

Best models are chosen according to training and test performances. If the training has more successful episodes with less crashes and it is stable during the training, the model is considered to be a good model. Model-1 and Model-2 are selected as best models and the mean rewards are compared and presented in Figure VI-8. Although both models have dueling network architecture and prioritized experience replay, model-1 has no drone detection model and no transfer learning, as indicated in Table VI-3. On the other hand, model-2 includes a drone detection model running and the experiences from previous training are transferred. As is seen in Figure VI-8, model-2 starts the training with positive rewards and reaches its maximum levels in a short time. It

is seen that transferring experiences from a previous training speeds up the learning process. However, model-1 starts the training from scratch but can reach the high rewards like model-2, whereas model-1 has more crashes at the beginning of the training. Likewise, in Figure VI-9, the differences between model-1 and model-2 during training are shown in terms of the usage of transfer learning. In this figure, cumulative rewards in each episode are presented and the episodes are color-coded according to agent' success in blue, crash (failure) in red or time limit in cyan. At the beginning of the training, it is seen that model-1 without transfer learning has many episodes failed to catch the target and crashed on obstacles. However, model-2 takes an advantage of transfer learning and it has less episodes failed.

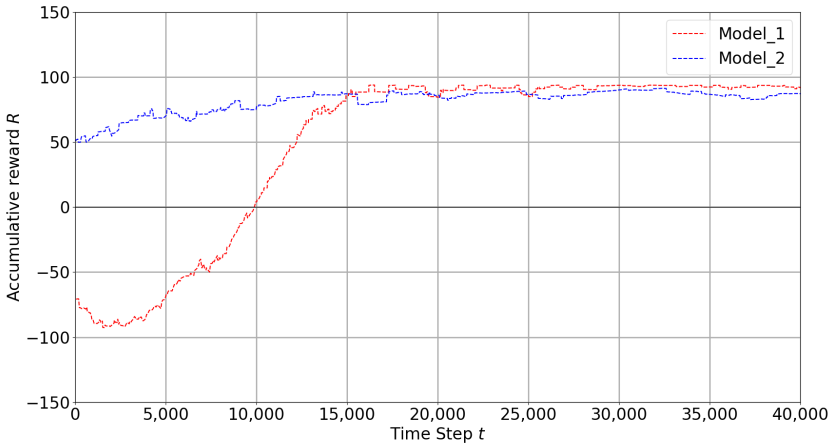
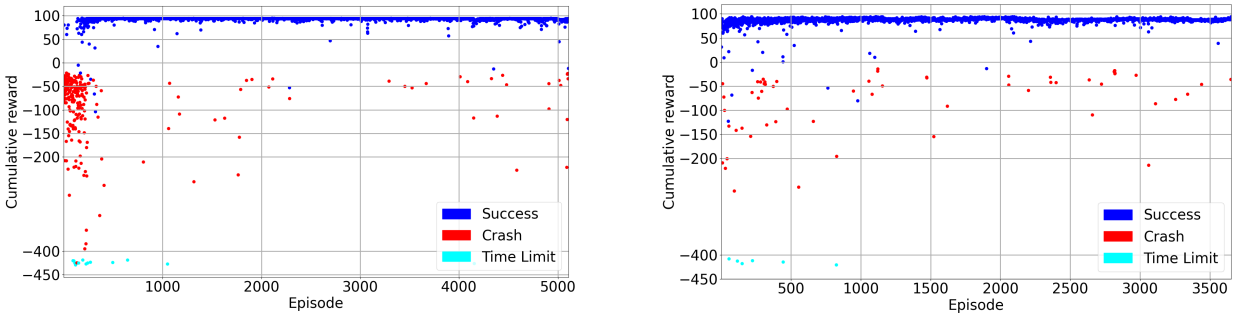


Figure VI-8: *Best Models.*



(a) without Transfer Learning (Model-1) (b) with Transfer Learning (Model-2)

Figure VI-9: *Comparison of Models with or without Transfer Learning*

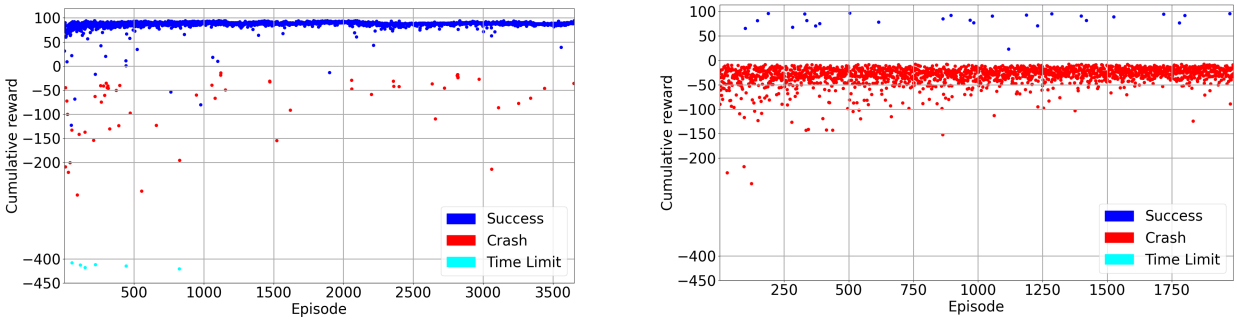
VI.7.1.2 Analysis of Models with Different Scenarios

In this section, the training results of the models are compared in terms of different annealing parts, teleporting and starting an episode with random heading.

- Different Annealing Points and Teleporting Feature:

  At the beginning of the training, it is expected to start with higher randomness in action selection to explore and try new things. This helps to balance between exploration and exploitation. In order to implement this scenario, different annealing stages are applied on the models to investigate the training progresses. For example, models with different

annealing parts and teleporting feature are investigated to analyze the learning progress of the agent. Teleporting is a feature at the moment of the start of training to locate the agent in different coordinates in the environment to increase exploration. Training results for model-2 and model-7 are selected due to their significantly different annealing sections, as specified in Table VI-3 and Figure VI-10 presents the difference between these two models. In Figure VI-10(b), it is seen that the model with high annealing part which ends after 100K steps and teleporting option has many episodes failed to catch the target and crashed on obstacles in the environment. However, in Figure VI-10(a) it is shown that the model which has less annealing part and without teleporting option has more successful episodes and less crashes. Although model-7 has more exploration than model-2 during training, the agent learns very slowly.

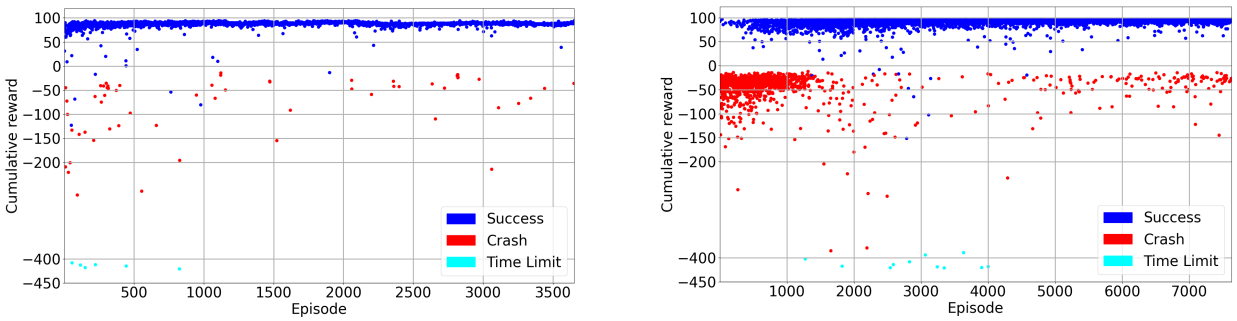


(a) Annealing=20K and without Teleporting Model-2 (b) Annealing=100K and with Teleporting Model-7

Figure VI-10: *Comparison of Models with Different Annealing Parts and Teleporting.*

- Random Heading at the beginning of an Episode:

  In normal scenario, the agent starts an episode by facing the target drone. However, exploration can be limited and the learning progress is not efficient. To avoid this situation, the agent starts each episodes with random yaw angle to explore different areas in the environment and to avoid obstacles during searching for the target. For example, Model-8 which starts an episode with random heading is compared with the model-2 facing the target at the beginning of an episode. Figure VI-11 presents both cases to highlight the differences in training between these models. Without random heading, the model seen in Figure VI-11(a) can crash less at the beginning of training and converges faster. However, as shown in Figure, VI-11(b) Model with random heading starts training with many crashes over a long period and then the training starts converging.



(a) without Random Heading (Model-2) (b) with Random Heading (Model-8)

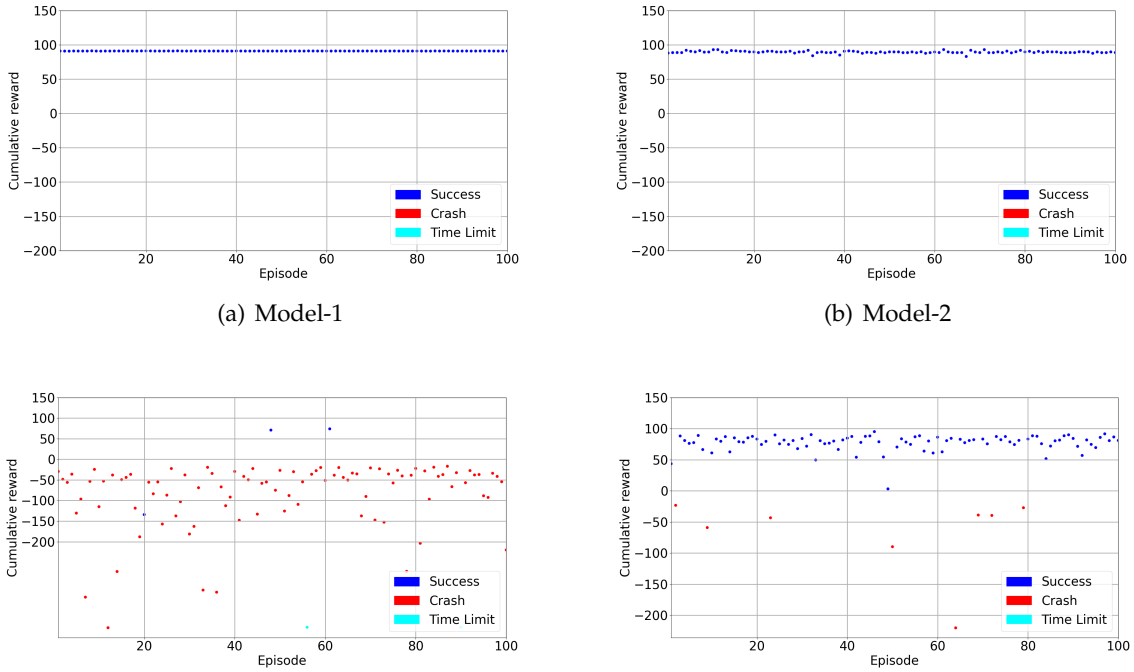
Figure VI-11: *Comparison of Models with or without Random Heading.*

VI.7.2 Test Results

After training the DRL models described in Table VI-3, the models are tested in the environment with the best checkpoint weights obtained during the training. The models presented which includes best and worst models are tested and presented in Figure VI-12. The test results are shown in Table VI-5. In this table, average cumulative rewards, minimum and maximum rewards are compared. The success rates represent on how many episodes the learner drone catches the target drone in a test out of 100 episodes. In addition, the average steps in each episode in tests are also presented. Model-1 and model-2 show the best performance as expected since they are selected as the best models and shown in Section VI.7.1.1. Average cumulative rewards are 90.86 and 89.38, the highest for model-1 and model-2 respectively. The highest minimum cumulative rewards show the precision of these models. Model-1 and model-2 spend less time to catch the target with average time steps 8.98 and 11.92 respectively. However, model-7 and model-8 are not as successful as expected. Although model-8 has a better success rate (92%), it has higher average steps compared to the best models, and model-7 fails to catch the target drone.

Table VI-5: *DRL Models Test Statistics.*

Models	Average Cumulative Reward	Max. Cumulative Reward	Min. Cumulative Reward	Success Rates	Average Steps
Model-1	90.86	91.12	90.69	100%	8.98
Model-2	89.38	92.93	82.93	100%	11.92
Model-7	−87.55	73.78	−408.58	3%	56.97
Model-8	65.73	95.06	−220.29	92%	32.01



(a) Model-1 (b) Model-2

(c) Model-7 (d) Model-8

Figure VI-12: *Test Results.*

VI.8 Discussion of Experimental Outcomes

In this section some plots are shown and analyzed to understand the relation of the models characteristics and the behaviour of the agents during their training process. The position of the learner drone, the agent, in episodes during the training of DRL models have shown different flight paths in order to catch the target in a 3D space. Figures VI-13–VI-15 present 3 projections of the 3D space (x-y, x-z and y-z) and the positions during a particular, but late, episode, in particular 2285 of the total of 5107 episodes for model-1, episode 3560 of the total of 3653 episodes for model-2, and 2666 of the total of 7631 episodes for model-8. It is found that the learner drone can use different actions to catch the drone in different episodes and different models. For instance, model-2 takes 58 steps to catch the target and the positions are shown in Figure VI-14. The learner drone moves up and goes forward without changing position in the y-direction. Figure VI-13 shows that model-1 can take many action steps to catch the target. Firstly, the learner drone moves up and goes forward. After the learner drone passes the target, it starts going up and down to search for the target and finally it catches it and the total steps are greater than 100 steps. Moreover, model-1 has shown an interesting approach in that the learner drone spends time in the y-direction such as going left and right and going up and down at the same time. However, although model-8 is declared as one of the worst models, there are also successful episodes in which the learner drone catches the target. For example, the learner drone position is presented in Figure VI-15. In this figure, it is seen that the learner drone spends a lot of time to find the target and uses different kinds of actions including going forward and backwards. The episode in this approach takes 84 steps to catch the target. In counter-drone systems, catching the target as soon as possible is expected. Otherwise, the target can be lost in a short time.

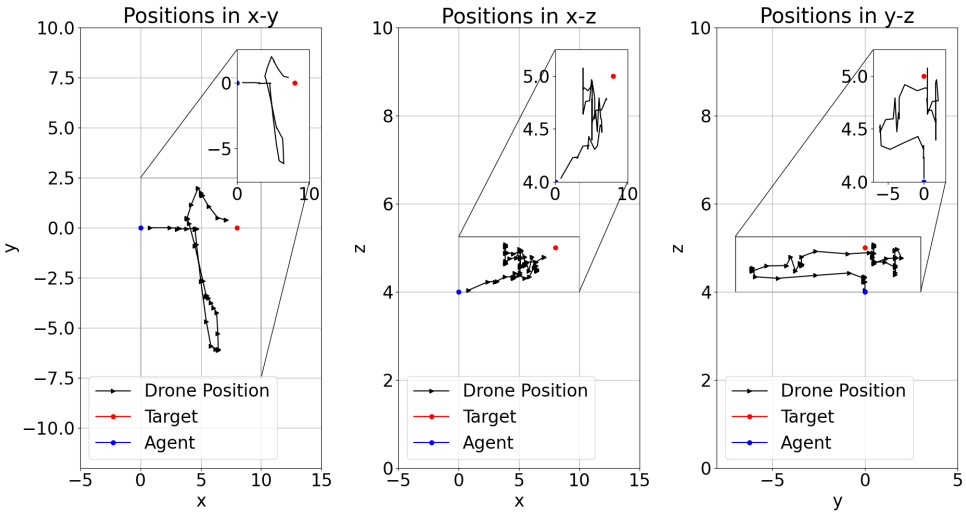


Figure VI-13: *Model-1 Training Episode 2285 Drone Position.*

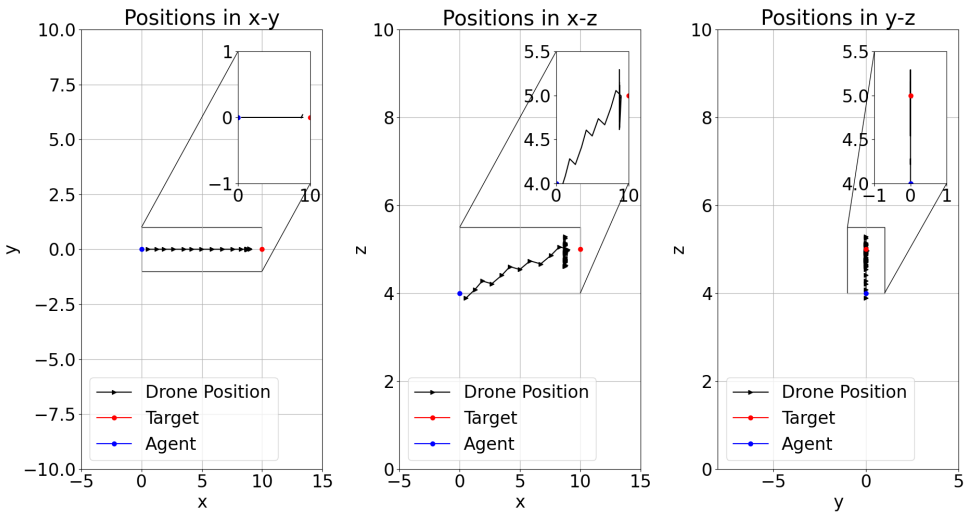


Figure VI-14: *Model-2 Training Episode 3560 Drone Position.*

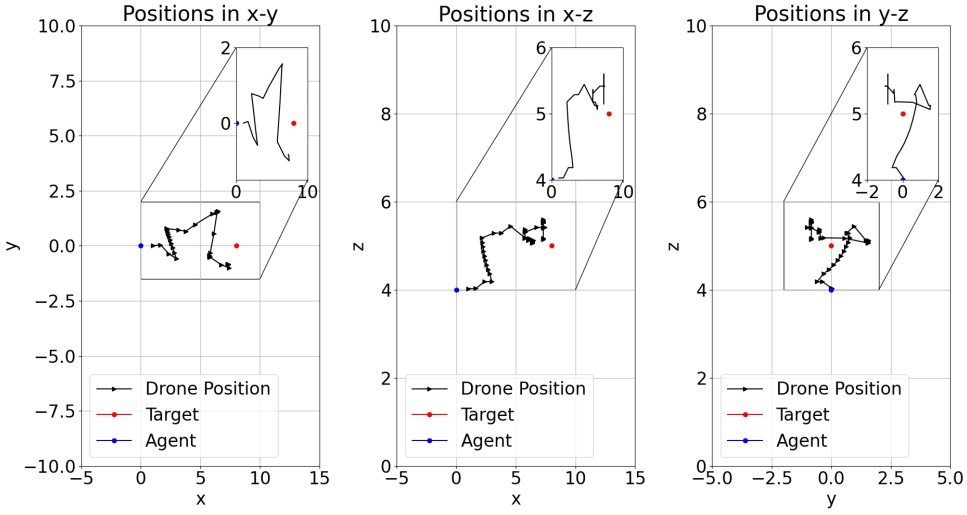
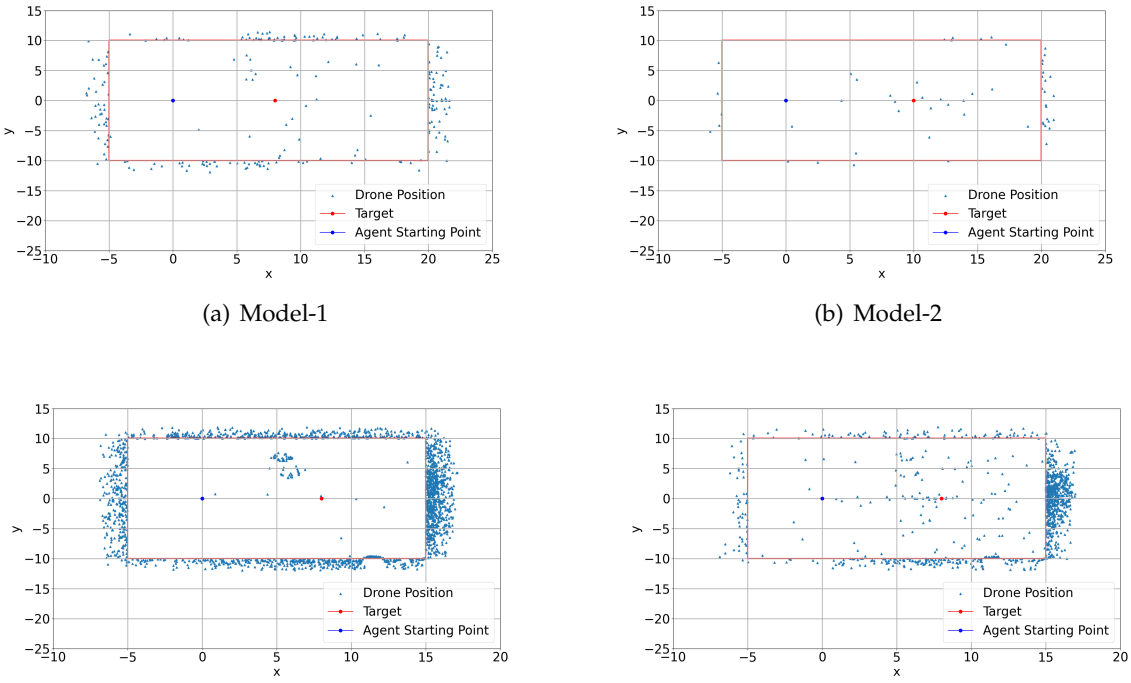


Figure VI-15: *Model-8 Training Episode 2666 Drone Position.*

Additionally, explainability of deep reinforcement learning models is investigated. The crashed episodes are analyzed by checking the drone crash locations in the environment. Figure VI-16 shows crash positions in the environment in x-y directions for four different models. Red rectangle lines represent the geofences in the environment in x-y directions. It is clearly seen that model-7 and model-8 crash a lot of times on the right side of the geofenced location in $\times$ direction. However, among all the models, model-2 has minimum crashes. This can also be seen in Figure C-4. Model-1 performs better than model-7 and model-8 but not as well as model-2. Moreover, model-7 crashes on each side of the geofenced area. The long annealing part also contributes to this situation because the random behavior is high in the annealing part and the learner drone tries to explore more in the environment, but even after a long time, there are no improvements in this model.



(a) Model-1 (b) Model-2

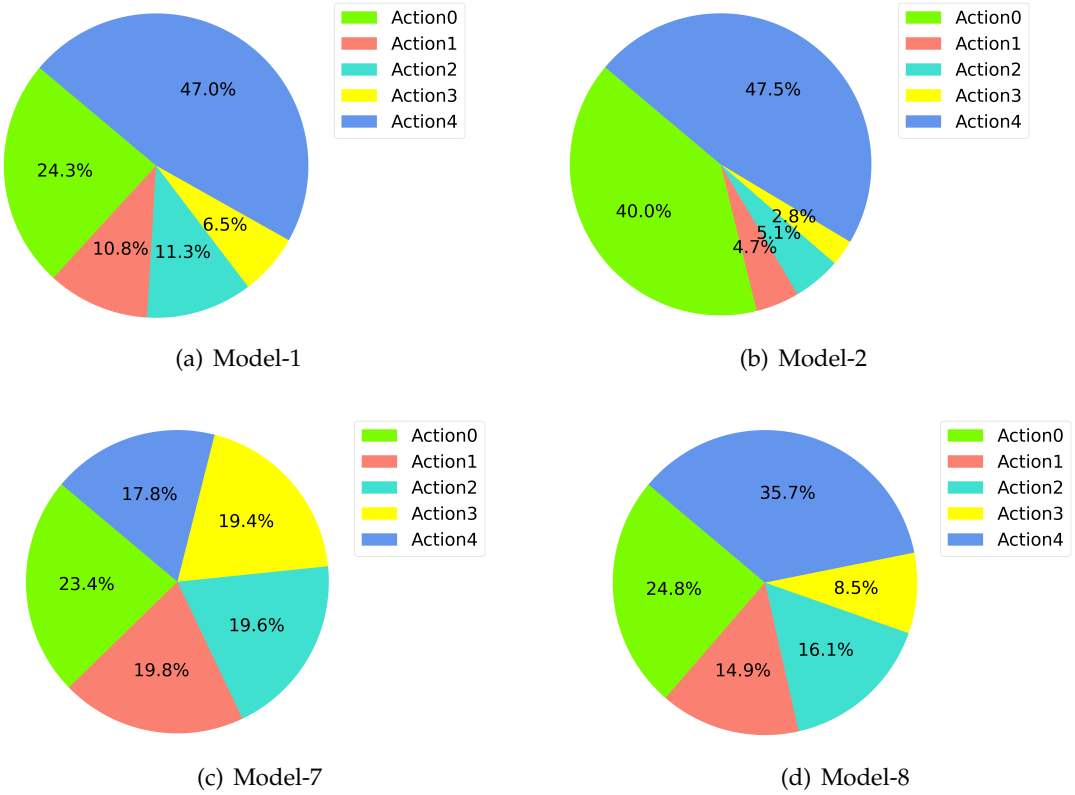
(c) Model-7 (d) Model-8

Figure VI-16: *Crash positions.*

Actions for these four models are also presented in Figure VI-17. Expected behavior of the models which does not have random heading at the beginning of an episode is that the drone should go up and move forward since the target is in front of the learner drone and the vertical distance is 1 m. Model-1 and model-2 perform as expected but model-1 spends more time on turning left and right. Model-2's performance shows a better result and it spends less time on turning but focuses on going up and moving forward. Model-8 fails to do the expected behavior and sometimes spends a lot of time finding the target. However, model-7's actions show no learning at all. The actions the drone uses in this model are almost distributed equally among the five actions and the drone has no idea where the target is and where it should move to catch it. All the actions that the DRL models used can be seen in Appendix C, Figure C-5. The actions and the movements are described in Table VI-6.

Table VI-6: *Actions.*

Action	Movement
0	Forward
$1-2$	Lateral
$3-4$	Vertical



(a) Model-1 (b) Model-2

(c) Model-7 (d) Model-8

Figure VI-17: *Action Frequencies.*

Your time is limited, don't waste it living someone else's life. Don't be trapped by dogma, which is living the result of other people's thinking. Don't let the noise of other opinions drown your own inner voice. And most important, have the courage to follow your heart and intuition, they somehow already know what you truly want to become.

— Steve Jobs

Life is like riding a bicycle. To keep your balance, you must keep moving.

— Albert Einstein

VII

Performance Analysis of DRL Agents and Human Pilots

In this chapter, the performance of human pilots are compared with the performance of deep reinforcement learning method and direct solution method to counter a drone in the simulation environment. The results are analyzed by comparing the time to catch the target drone in seconds between DRL methods for 2D space and 3D space, human pilot and direct method, an algorithm, which directs the drone towards the target position without using any AI method. The main idea is to catch a drone in an environment as fast as possible without crashing any obstacles inside the environment. The training and test results show that the agent drone learns to catch target drone which can be a stationary and a non-stationary. In addition, the agent avoids crashing any obstacles in the environment with a minimum success rate of 93%. Also, DRL model performance is compared with the human pilot performances and the agent with DRL model shows better time to catch the target drone. Human pilots struggle to control the drone by using remote controller when catching the target in simulation. However, the agent with DRL model is rarely missing the target when trying to catch the target. Direct method is also tested to catch the target drone and it has the fastest results if there is no obstacle between the target and the agent. However, when the performances compared with one of the best DRL models in a 3D space and the direct method after introducing the obstacle between the agent and the target, direct method always failed to catch the target since it crashes on the obstacle, while DRL model catches the target with a minimum success rate of 72%.

This chapter is based on the following publications:

VII.1 Description of the Methods

In this section, DRL methods, human pilots and the direct method are described. In all of these approaches, the agent is a quadcopter drone and the environment setup is already explained in chapter III. In DRL method, the agent drone is trained to catch the target drone and it is rewarded in each time step by the environment provided by Airsim flight simulator. However, human pilots and direct method do not have a DRL algorithm running on background but direct method, as an algorithm, simply follows the target and crash on it. The description of the methods are presented in Table VII-1. In this table, the name of the methods, the number of actions used by each method, and whether they have DRL algorithm and reward function are shown. Details of the methods are explained in the following sections.

Table VII-1: *Description of the Methods.*

Models	DRL Algorithm	Human in the Loop	Number of Actions	Reward Function	State Input
DRL-2D Agent	DDQN	NO	3	YES	Image + Scalars
DRL-3D Agent	DDQN+Dueling+PER	NO	5	YES	Image + Scalars
Human Pilots	-	YES	4	NO	Image
Direct Method	-	NO	1	NO	Target Location

VII.1.1 DRL model for a 2D Space

In this section, the DRL model which is presented in Chapter V is used. However, several improvements have been made in states, reward function and actions that the agent can take. One of the most important improvements in this DRL model in a 2D space is in image state which contains the drone detection information on the image input. The improvements of this model are presented in the following sections in detail.

VII.1.1.1 Drone Agent States

The state of the DRL model is constructed as it is explained previously in section V.2. Neural network model is also explained in detail in section V.3. The image in this section includes geofence information by drawing fences on the image and an additional information about the target drone's location. Drone detection model processes the raw image before sending it to the agent as an input to the DRL network. This process is presented in Figure II-18. The states of the DRL model in this section are as follows:

- Image State
  The drone continuously captures 256 x 144 pixels size depth image and scene image by using its onboard camera. The scene image seen in Figure VII-1(a) is processed by drone detection model to create bounding boxes when the target drone is detected on the image. The world coordinates of the target drone position are projected into image pixel coordinates by using the quaternions and rotation matrix. Image processing and the drone detection process can be seen in detail in section III.4.

  Depth image seen in Figure VII-1(b) is used in DRL model for detecting obstacles. After processing the images, a bounding box region in depth image is filled with white color and circles like a target in the dart game are created inside the white bounding box region as it is shown in Figure VII-1(b).

- Çetin E, Barrado C, & Pastor E.. 2021 (Oct.). Counter a Drone and the Performance Analysis of Deep Reinforcement Learning Method and Human Pilot. *In: IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*. San Antonio, TX, USA : DASC. D.O.I: 10.1109/DASC52595.2021.9594413 Hybrid event: DASC 2021.

(a) Drone Detection Image (b) Image After Drone Detection

Figure VII-1: *Drone Detection and Image Processing*

The image state shown in Figure VII-2 is set as 30 x 100 pixels which contains 3 × 10 pixels black line used to represent the track angle. This line moves left and right while the track angle changes. This line can be slightly in different places on the image as in the location of the circles on the image because the position of the line is a linear correlation between the angle value and the width of the image Kersandt (2018). Moreover, the grid is drawn on the image when the distance between the drone and the geofence limits becomes lower or equal to 4 m. The thickness of the grid increases as the drone moves towards the geofence limits. The grid is drawn as it was in V-5.

Figure VII-2: *Image State*

- Scalar Inputs
  Scalar inputs contain $v_x v_y$ agent's velocities in x and y directions $d_x d_y d_t$ agent's distances to goal in x and y directions and euclidean distance, dg_{xmin} dg_{xmax} dg_{ymin} dg_{ymax} agent's distances to geofence limits, track and elevation angles between the agent and the target. These inputs are received by using APIs provided by the simulation for both the agent and the target drones. In real world, localization and tracking the target drones are not directly available as explained in section I. There are tools and methods to calculate these angles in real world counter drone systems.

VII.1.1.2 Drone Control Actions

The actions used by the agent includes 3 different options as explained in section V.4. However, the agent will have different yaw rates in this section to allow the agent explore different parts of the environment: rotate by 30 degree on the left, rotate by 25 degree on the right and going forward 4 m/s. Also, there is an additional task called attack mode which is added to the actions during testing. If the drone is very close to the target drone and the drone detection model detects a drone, then activates the attack mode to terminate the target. The attack mode is a high speed termination task which is only available in close distances to the target. It is observed that the agent with deep reinforcement learning model moves to the target drone position and reduces the distances to target until the attack mode is activated.

VII.1.1.3 Reward Function

The reward function is improved by adding an information about drone detection. Reward function is formulated in Table VII-2. Drone detection reward is one of the important factor in this study to achieve better results considering the accuracy of state of the art real time object detection algorithm. The reward function is improved by adding a drone detection information value to the reward calculation. If a drone is detected and the target drone is on the visibility of the agent drone, the agent is rewarded. Track angle calculated by using the positions of target and agent, yaw angle. In other words, track angle is at the relative to the heading angle towards the goal.

Table VII-2: *Rewards*

Reward	The Reason
$+100$	Target Catched
-100	Collision: Obstacle or geofence
$+1$	Drone Detection
$+\zeta$	Track Angle
$-1+\Delta dg$	Otherwise

VII.1.2 DRL model for a 3D Space

In this section, one of the best DRL models proposed in Chapter VI, model-1, is used to counter a drone in 3D space. The details of this model can be seen in Tables VI-3 and VI-4. The actions and reward function are described in VI.4 and VI.5 respectively. The differences between this model and the rest of the methods in this chapter are explained previously in Table VII-1.

VII.1.3 Experimental Setup for Human Pilots

Human pilots performed the same task as the agent does in DRL method to catch the target drone. Two pilots with different skills are selected. One of the pilots has a drone pilot license and the other pilot is "SIM pilot" with more experience on flight simulation but without drone pilot license. Human pilots control the drone by using the remote controller, FrSky Taranis X9D Plus FrSky (2021). This is a real UAV remote controller and it can be directly connected to PC via USB port. Pilot actions shown in Figure VII-3 include throttle input, as well as inputs for roll, pitch and yaw control. DRL agents indirectly control these inputs and instead they use simulation APIs to interact with vehicle programmatically.

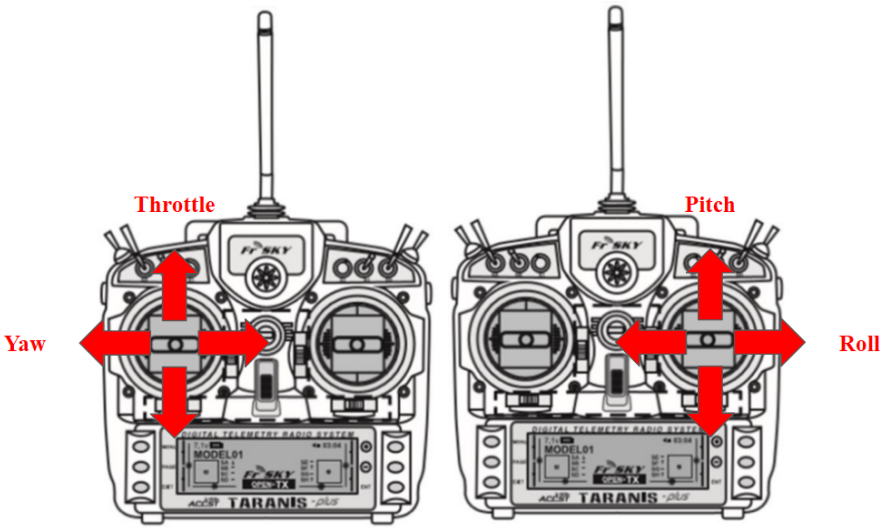


Figure VII-3: *Actions for Human Pilots*

The state input of the pilots seen in Figure VII-4 is the first person view of the neighborhood environment and the pilots performed the tests to catch stationary drone and non-stationary drone. Licensed drone pilot has trained almost 2 hours in the simulation to catch the target drone. In real life, human pilots have flight modes such as altitude-hold mode available to them in RC. However, they don't have an access to these modes in the simulation and they need to control the throttle input as well as roll, pitch and yaw control inputs. To compare human pilot performances with the deep reinforcement learning agents, 10 best episodes are selected for each task.

Figure VII-4: *State Input for Human Pilots*

VII.1.4 Direct Solution Method

An algorithm directs the drone towards the target position without using any AI method or traditional navigation and guidance method. The algorithm which directs the agent to the target is presented in Algorithm VII.1. This is the fastest method and it is considered as an ideal condition since the drone has all the necessary parameters and flight data such as target drone flight attitude angles, target identity, target GPS positions, no obstacles around the target drone. In this method, the agent has available positional information about the target. The agent drone keeps

front pointing ahead and it rotates to specific angle (i.e. yaw) and keep that angle while moving. In direct method there is no available detection and avoidance system. Although direct method is the fastest method, it is not an adequate solution due to low flexibility to detect and avoid obstacles.

Algorithm VII.1: *Direct Solution Method Algorithm*

1: Initialize drones
   a: reset ▷ Resets the drones to its original starting state.
   b: enableApiControl -> True ▷ API control for drones is enabled.
   c: armDisarm -> True ▷ Drones are armed.
   d: moveToZAsync ▷ Drones move to their initial positions.
2: **for** episode $e \in 1, 2, ...M$ **do**
3: $t_0 = time.time()$ ▷ Start time.
4: **repeat**(for each step of episode):
5: Move the agent drone toward target location in x, y and z direction with certain speed.
6: **if** The agent catches the target **then**
7: $t_1 = time.time()$ ▷ Finish time.
8: $t = t_0 - t_1$ ▷ Total time spent to catch the target.
9: Save the time t.
10: Target is catched ▷ Episode ends
11: **else if** The agent crashes on any obstacle. **then**
12: There is a collision ▷ Episode ends
13: **until** Target is catched or there is a collision

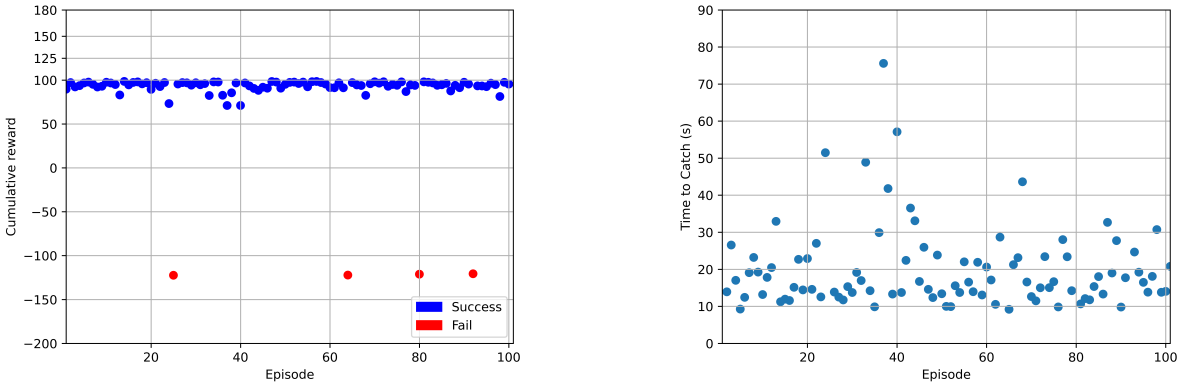
VII.2 Results

In this section, the methods are tested with different kind of targets such as a stationary drone target or a drone target which moves randomly in the environment. Each test contains 100 episodes. The test results are presented in the following sections.

VII.2.1 DRL Method in a 2D Space Test Results

DRL method in 2D Space test results are discussed. In Figure VII-5(a) and Figure VII-5(b), cumulative rewards and time to catch the non-stationary target drone are represented respectively. In Figure VII-5(a), it is seen that the agent catches the target with a success rate of 96%, but in four episodes which have rewards in between -100 and -150 the agent failed to catch the target and they are penalized. In addition, the time spent during each episodes are generally around 25 time steps in seconds. There are also episodes that take longer time to catch the target. For example, between episodes number 20 and 40, few episodes spent more than 50 time steps to catch the target. The main reason is that the target drone moves randomly and the agent can miss the target in first try. However, the agent returns back to catch the target, although the agent is not trained to catch a non-stationary drone.

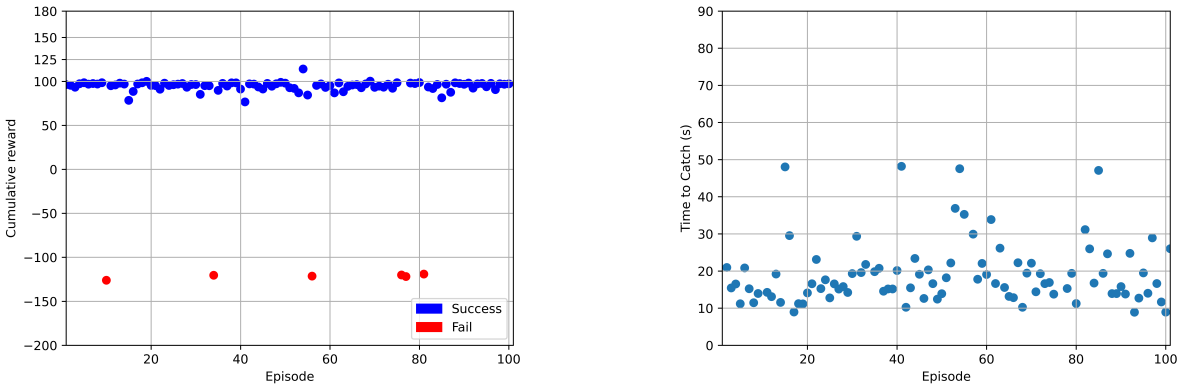
The agent spent less time when the target is a stationary drone. In Figure VII-6(b), it is shown that the agent spends less than 25 time steps in general to catch the target drone, although some episodes are longer than expected such as 50 time steps. However, it does not mean the agent is failed. On the other hand, the agent successfully catches the target after coming back to face the target drone. The cumulative rewards shown in FigureVII-6(a) are around 100 which shows similar trend seen in Figure VII-5(a) except for episode number 54 which has the highest reward



(a) Rewards (b) Time to Catch (s)

Figure VII-5: *DRL Method in 2D Space Test Results for Non-Stationary Target*

value, 114. The agent catches the target with a success rate of 94%.



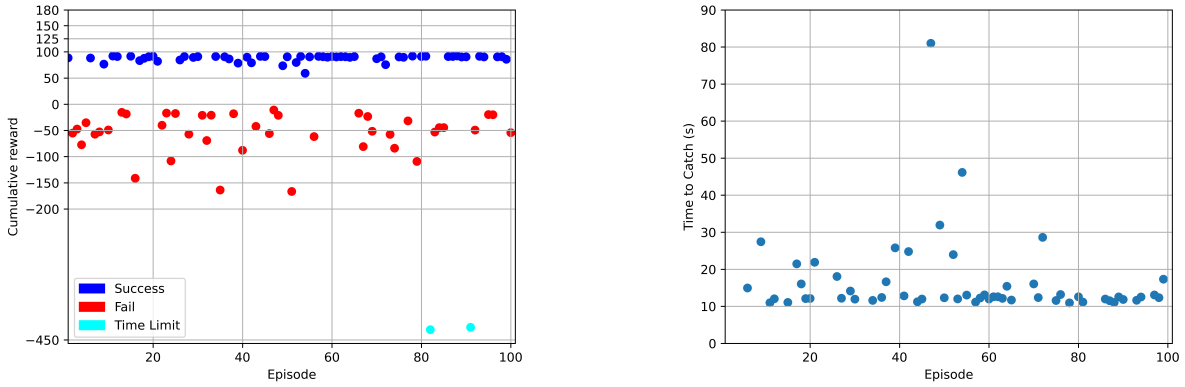
(a) Rewards (b) Time to Catch (s)

Figure VII-6: *DRL Method in 2D Space Test Results for Stationary Target*

VII.2.2 DRL Method in a 3D Space Test Results

The target drone in this section moves randomly without changing altitude but it starts 1 meter above the the agent start altitude. In Figures VII-7(a) and VII-7(b), cumulative rewards and time to catch the non-stationary target drone are represented respectively. It is seen that DRL model in 3D space has mixed results presented in Figure VII-7(a). There are also time limits occurred in between episode numbers 80 and 100. However, the time spent during each episodes are mostly around 25 time steps in seconds. There are also episodes which take longer time to catch the target such as episodes between number 40 and 60 and one of episodes the agent spent more than 75 time steps to catch the target. When the agent misses the target in first try, the agent returns back to catch the target, although the agent is not trained to catch a non-stationary drone. The results show that even in challenging scenario such as countering a drone in a 3D space, the agent can follow and catch the target in a short time as it was in VII.2.1.

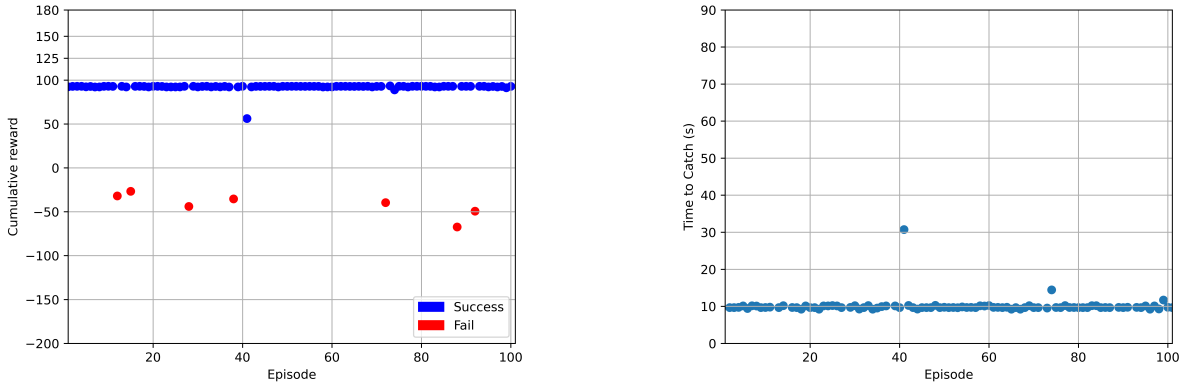
FigureVII-8(a) shows that the agent successfully catches the stationary target drone with a success rate of 93%. The cumulative rewards are mostly around 93 and stable compared to the cumulative rewards shown in Figure VII-7(a). Moreover, the agent catches the stationary target



(a) Rewards (b) Time to Catch (s)

Figure VII-7: *DRL Method in 3D Space Test Results for Non-Stationary Target*

in around 9 time steps in seconds but one of the episodes which took more than 25 time steps in seconds presented in Figure VII-8(b). This is because the agent misses the target in first try but it catches the target after coming back to face the target drone.



(a) Rewards (b) Time to Catch (s)

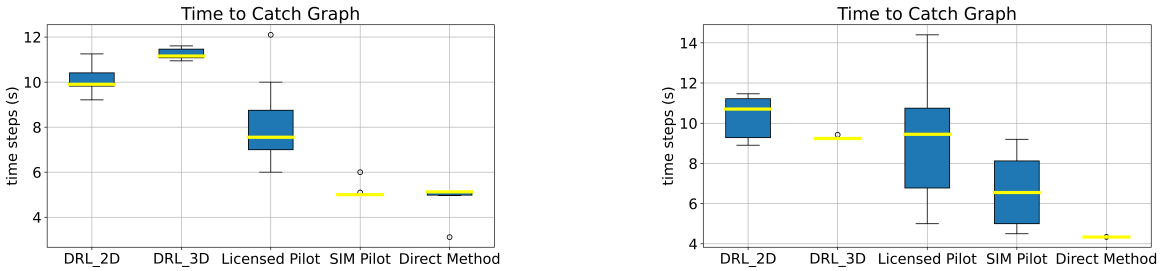
Figure VII-8: *DRL Method in 3D Space Test Results for Stationary Target*

VII.2.3 Comparison of the methods with stationary and non-stationary targets

Figures VII-9(a) and VII-9(b) present the time spent to catch the target drone for both pilots and DRL agents. In FigureVII-9(a) the time distribution to catch non-stationary target by each pilot and DRL agents is presented. Simulation pilot as known as "SIM pilot" is the fastest compared to licensed pilot and the DRL agents. DRL agents spends more time to catch the target because they are not trained to catch the non-stationary target but they can catch it in a longer time than expected average time steps in seconds.

However, the pilots have struggled to control the drone. Most of the problems in test flights are in altitude control of the drone. The median values are represented in horizontal yellow lines in FigureVII-9(a) and FigureVII-9(b). As it is seen in these figures, human pilot median values out of 10 best episodes are lower than the AI median values but DRL agent in 3D space in Figure VII-9(b) has almost the same median value as licensed pilot has. In FigureVII-9(b), it is seen that there are big gaps between the whiskers in human pilot performance results compared to DRL

agents. On the other hand, the minimum time to catch a non-stationary drone target or stationary target can be lower in human pilot time results. In other words, human pilots can be faster than DRL method to catch the target drone. However, human pilots are not stable in each test episode. They can miss the target and it can take longer than DRL agent time to come back to catch the target drone.



(a) Results for Non-Stationary Target

(b) Results for Stationary Target

Figure VII-9: *Results for Time to Catch*

Furthermore, the time results for DRL method, human pilots, and direct method are presented in Table VII-3 in terms of success rate which accounts for first approach catch, best and worst timing values. Human pilots achieved the best time when catching stationary and non-stationary target. However, human pilots also scored the worst time compared the DRL agents. In Direct solution method explained in section VII.1.4, the drone catches the stationary target in 4.3 time steps in seconds. However, when catching non-stationary target, the drone spends 3.1 time steps in seconds, but its worst time is higher, 5.8 time steps in seconds compared to targeting stationary drone which is 4.7 seconds.

- **DRL agents vs human pilots**: DRL has worse performance in general, but it is cheaper. Also, DRL agents are more stable and make the worst humans cases better. DRL needs to improve, but it is a good start.
- **DRL agents vs direct method**: Direct method is cheaper and better. Direct method is almost as good as humans best cases, and much better in their worse cases. However, direct solution method is not capable of avoiding obstacles. The following section compares one of the best DRL models with the direct solution method to test its capabilities to catch the target drone if an obstacle presents between the agent and the target.

Table VII-3: *Comparison of Results*

	Stationary Target			Non-Stationary Target		
	Success (%)	Best (s)	Worst (s)	Success (%)	Best (s)	Worst (s)
DRL-2D Agent	94	8.9	48.2	96	9.2	75.6
DRL-3D Agent	93	9.2	30.7	56	10.9	81.0
Human Pilots	100	4.5	156.5	100	5.0	143.1
Direct Method	100	4.3	4.7	100	3.1	5.8

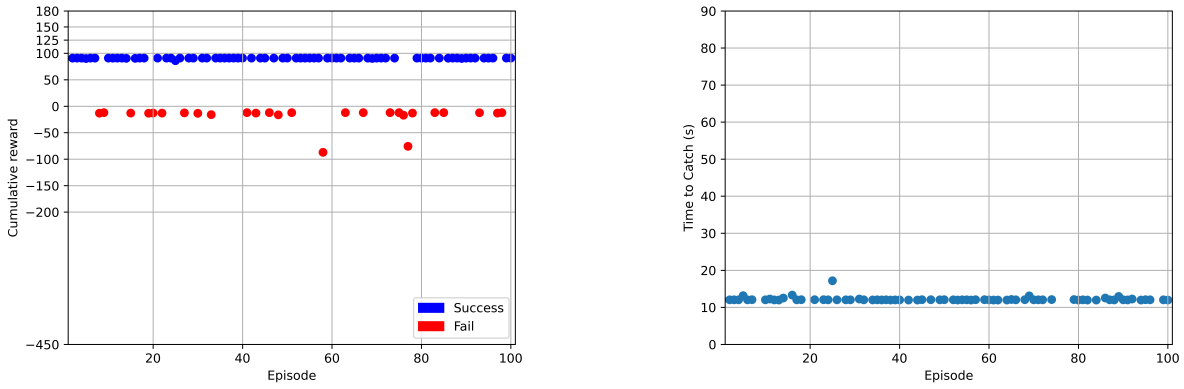
VII.2.4 Test Results with an obstacle

In this section, the methods presented in Table VII-1 are challenged with another interesting test. One of the best DRL method, DRL model in a 3D space, and the direct solution method are tested with an obstacle located between the target and the agent. This is illustrated in Figure VII-10. In Figures VII-10(a) and VII-10(b), top view and front view of the initial positions of the drones in the environment are presented respectively. An obstacle which is the identical to the target drone is located in between the target and agent drones.

The results shows that DRL model in a 3D space catches the target in 72 out of 100 episodes and DRL model catches the target in 12 seconds. However, direct method fails in all episodes during testing because the direct solution method lacks the ability to avoid obstacles. The results are shown in Table VII-4.

(a) Top View (b) Front view

Figure VII-10: *Initial positions of the drones*



(a) Rewards (b) Time to Catch (s)

Figure VII-11: *DRL Method in 3D Space Test Results with an obstacle*

Table VII-4: *Test Results with an obstacle*

Method	Success (%)	Best (s)	Worst (s)
DRL-3D Agent	72	11.9	17.2

A person who never made a mistake never tried anything new.

— Albert Einstein

Nothing in life is to be feared, it is only to be understood. Now is the time to understand more, so that we may fear less.

— Marie Salomea Skłodowska-Curie

VIII

Concluding Remarks

Counter-drone systems to fight against intruder drones can benefit from artificial intelligence methods. With the expansion of drones flying in the airspace, the availability of an effective counter-drone technology is a must. Counter-drone technology consists on several systems deployed on the ground and in the air. The solution proposed in this PhD thesis focuses on the capability to track and neutralize the target drone and it is just one part of the counter drone solution. In order for this solution to be effective, it is necessary to have a support system that detects the target drone, classifies its activity as malicious, and estimates the position to the target drone. The main objective of this PhD thesis was the development of an artificial intelligence method, deep reinforcement learning, to counter a drone in an urban environment and to contribute to the safe integration of drones into public areas.

While deep reinforcement learning showed that it is a promising approach for the interception of the target drone, there is still much research to be done to determine its effectiveness. DRL may not yet be mature enough for counter drone systems but as research and development continuously evolve, DRL may become more accessible and effective in countering drones. Moreover, it may be legally possible to fly drones with DRL, but the specific regulations and requirements that apply will depend on the country and the purpose of the flight. It is important to investigate and observe all relevant artificial intelligence and drone regulations to guarantee the safety of drone operations based on DRL algorithms.

As a result of this research, several questions were raised that need further examination. Some of these questions remain open and could be explored further. The following is a brief summary and conclusion of the achieved results, as well as suggestions for possible future research.

VIII.1 Summary of contributions

The main contributions of this PhD thesis are summarized as follows:

- Drone detection model by improving state of the art object detection algorithm and auto-labeling images are defined in chapter III. It is shown that drones can be detected with high accuracy by using powerful real-time object detection algorithms available in the literature since drone detection is a critical element of counter-drone systems, which also include other subsystems such as drone type classifiers (malicious or friendly) and neutralization subsystems. The efficiency of drone detection models is improved by training them using different categories of drone images and the images are automatically labeled. This automatic labeling approach is considered to be an efficient method for advancing drone detection models in the future.

- Deep reinforcement learning model using DDQN algorithm with uniform experience replay is addressed in chapter IV. DRL model is implemented to navigate the drone in an urban environment without crashing any stationary and non-stationary obstacles such as drones moving randomly. DRL model states include image state and the scalar states and these states are concatenated by a neural network called JNN. The results in different scenarios are very promising and the learner drone reaches its destination successfully.

- Explainability of deep reinforcement learning is also investigated to understand the black-box .The reasons why the agent has chosen its actions has been investigated to understand the DRL model results. Graphical methods are utilized to understand the behavior of the agent drone and the modifications on DRL model are made by carefully observing the agent's reactions after each episode during training. This allows to improve the models and to explain why the AI model gives a particular result.

- A deep reinforcement learning architecture is utilized and enhanced to counter a drone with another drone in 2D space in chapter V. Transfer learning approach is implemented by using the baseline pre-trained model weights. The performances of training results with or without transfer learning are compared by using the common transfer learning metrics and it is seen that training time is reduced and the cumulative reward is more stable during the training with transfer learning.

- An important breakthrough has been achieved in solving the challenging task of countering a drone in a 3D space. The state-of-the-art DQN algorithm, without any improvements, is trained earlier and the results show that the agent had difficulties to catch the target and it crashed many times during the training. However, the best models trained with DDQN with dueling architecture and prioritized experience replay presented in chapter VI, shows better performance and much faster learning.

- The first and more significant contribution is the filtering algorithm applied during transfer learning. This consists of pre-processing the previous experiences and eliminating those considered as bad experiences. The proposed solution is to be the first DRL solution that successfully solves the counter-drone challenge in 3D. There are studies presenting how a drone lands on a moving platform or a drone chases an object in the environment in 2D space by using reinforcement learning but the time to interact with the target and the actions which the drone can use are not focused on. These details are the critical part of the counter-drone systems to eliminate the target.

- Finally, deep reinforcement learning agents, the human pilot performances and direct solution method are analyzed by comparing their times spent on catching the target drone. It is observed that human pilots struggle to control the drone by using the remote controller

when catching the target in the simulation. The training and test results showed that the agent can catch the target drone successfully and the agent with DRL model shows better time to catch the target drone and the agent rarely misses the target.

VIII.2 Future research

This PhD thesis raised new questions and led to new research directions.

- Deep reinforcement learning algorithms are developing and there will be challenges in the future. Identification of the target drone's mission or intent with the help of drone detection systems is a very important part of counter-drone solutions since the state of the target drone is the main part of the deep reinforcement learning algorithm. Advanced drone surveillance technologies including improved sensors, communication and networking can make it easier and more efficient to fight against malicious drones by classifying the target as a threat or non-threat based on its behavior.
- Target drone used in the training and testing of DRL agents is not realistic. The realism of the target drone can be increased by improving their sensors, using machine learning and artificial intelligence.
- Artificial intelligence improves itself very quickly and new methods and tools are being introduced at every moment. However, there is still little knowledge about how the predictions of artificial intelligence models work. In other words, it is not clear what makes them to choose the most convenient action. In the future, the methods for visualizing, explaining and interpreting deep reinforcement learning models need to be investigated.
- Graph neural networks (GNN) can be implemented to improve the interpretability of DQN agents and understand the reason behind the decision agent made in graph-structured environments. GNN are designed to work with graph-structured data and it can be combined with a DQN algorithm. For example, a DQN agent could be trained to navigate in a transportation network or a maze, by using GNN to process the graph data and make decisions. RL and GNN can also be combined in a different way such as using GNN to model the policy or the value function of RL agents in graph-structured environments.
- Deep reinforcement learning has the potential to enable drones to perform a wide range of tasks in the real world. However, applying DRL algorithms to drones in real-world environments poses several challenges. One major challenge is that DRL algorithms must be able to make decisions based on sensor data and their environment which may require efficient algorithms and hardware in real world applications.
- Deep Q-learning from Demonstrations algorithm which uses expert data should also be investigated in the future. Recent studies show that human expert data can be useful to reduce training time and it can be improved in counter-drone solutions, especially countering a drone in a 3D space which is a challenging task compared to a 2D space counter-drone solution.

A

Drone Detection Model

A.1 Results

A.1.1 Training Results

Training has been accomplished in 6000 steps for all the models. During the training the mean average precision (mAP) value is calculated and the best weights (those which give the highest mAP value) is saved. The mAP is the mean value of the average precision (AP) for each class, being the average precision the area of the Precision-Recall curve Bochkovskiy (2020). Training results of all models are summarized in Table A-1 showing the mean average precision (mAP) metric and the training time for every models. Mean average precision is calculated for an Intersection Over Union (IoU) threshold of 0.5 and represented as mAP@0.5.

Results show that Model-3 has the highest mAP value, 89.77 %, while the Model-1 has the lowest mAP value, 83.63 %. Model-2 has an intermediate mAP@0.5 equal to 84.93% and also took an intermediate training time between Model-1 and Model-3. Training times can be affected by batch sizes and subdivisions which is set in configuration of every models for training. Model-1 has the lowest training time thanks to its neural network size which has totally 24 convolutional layers.

Table A-1: *Training Results of the models after 6000 steps*

Model	Best mAP@0.5 %	Training Time (Hours)
Model-1	83.63	2.5
Model-2	84.93	14
Model-3	89.77	18

Training plots for every models are also presented in Figures A-1, A-2 and A-3. Red line represents calculated mAP values and blue line shows the loss value during training. Observe that the loss value drops dramatically after 600 iterations in all models .

In Figure A-1 the Model-1 mAP value starts with 71% and fluctuates around 80%. After 3420 iterations the best mAP value, 83.63%, is recorded. In the meantime, the average loss value stays at minimum and stable around 0.5 which is the highest in all models.

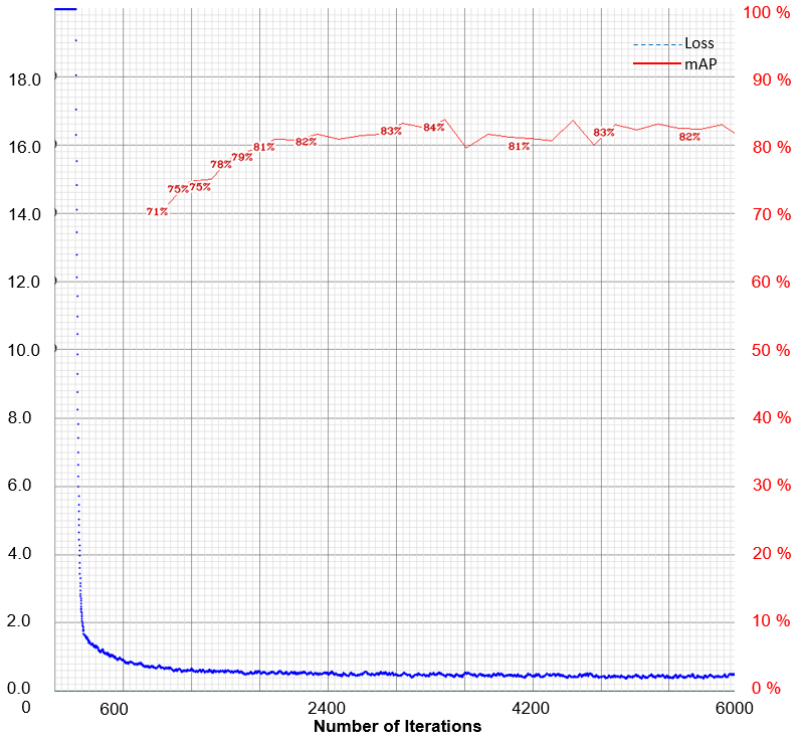


Figure A-1: *Loss & mAP(%) chart for training Model-1*

Moreover, in Figure A-2 training progress is shown for Model-2. In this training, mAP value starts with 43% and it jumps to 76 % level. After 3600 iterations the best mAP@0.5 value, 84.93, is calculated. At the end of the training the mAP value is calculated, but we saved the weights of the model for the first highest mAP value to avoid over-fitting.

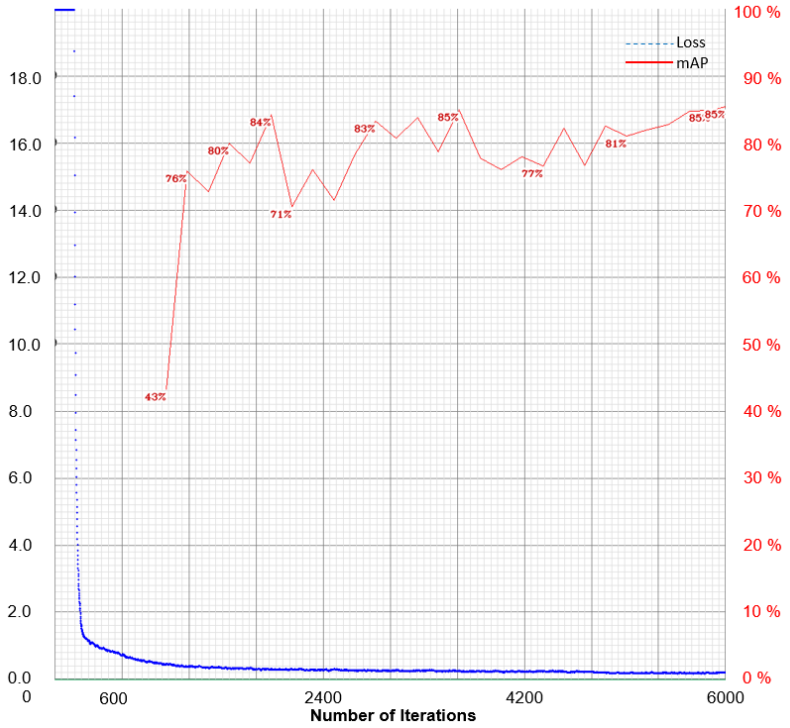


Figure A-2: *Loss & mAP(%) chart for training Model-2*

Finally, training progress for Model-3 is presented in Figure A-3. As it is done in the other model trainings, mAP values are calculated during training and the best weights are saved. In this training, best weights are obtained at around 4000 iterations. Model-3 mAP value shows the highest mAP with a 89.77% in respect to the two models presented before. The mAP values of Model-3 start with higher values, 68%, and the loss settles down at 0.25 in 6000 iterations.

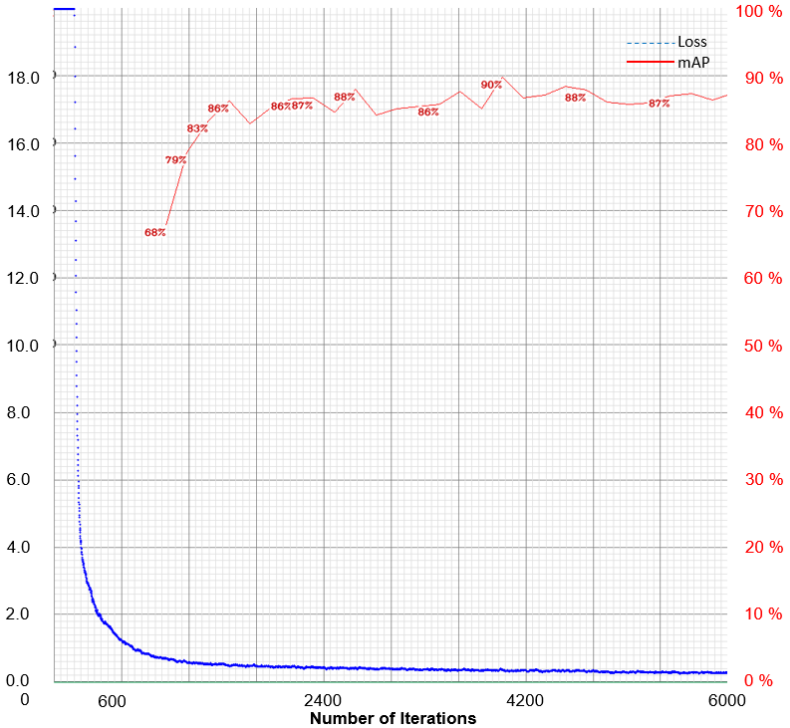


Figure A-3: *Loss & mAP(%) chart for training Model-3*

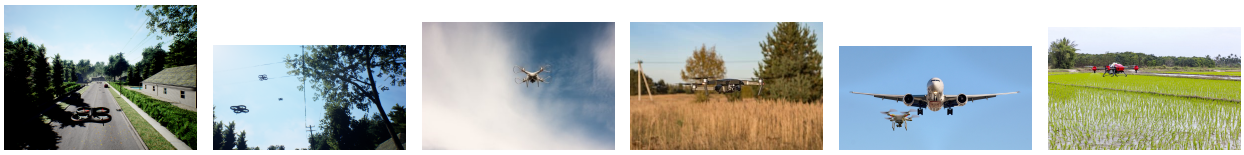
A.1.2 Test Results

Once the models are trained, we feed them with new not-seen images for the test evaluation. State of the art object detection models, shown as backbone models in Table III-2, and the three proposed models are tested with those different groups of images. The detail of the number of test images from each group is given in Table A-2. Four sources of images are proposed with a balanced distribution of 20 images for each source. Some of the images are shown in Figure A-4.

Table A-2: *Test Images Example*

Image Source	# of Images
Airsim Test Images	20
Drone-Net Test Images	20
Web Drone Images	20
No-Drone Images	20

Observe in Figure A-4 the first two images A-4(a) and A-4(b) are from Airsim simulator, the images A-4(c) and A-4(d) are obtained from the Drone-Net test set, the images A-4(e) and A-4(f) are random but challenging drone images, with noisy background, found in the Internet. Finally, the models are also tested with images, such as A-4(g) and A-4(h), which do not include any drones to capture potential errors in prediction. Observe that, unlike the other two sets, the Airsim images are drone images taken from another flying drone, not from ground as the others.



(a) Airsim #1 (b) Airsim #2 (c) DroneNet #1 (d) DroneNet #2 (e) Web #1 (f) Web #2

(g) NoDrone #1 (h) NoDrone #2

Figure A-4: *Test images from four different sets.*

The evaluation metrics used for measuring the neural network test performance and for comparing the models' results with each other are the following:

- **Accuracy**: (TP+TN)/Total Predictions
- **Precision**: TP/(TP+FP)
- **Recall**: TP/(TP+FN)
- **F1-Score**: 2×(Precison× Recall)/(Precison + Recall)

where TP: True Positives, FP: False Positives (or detection errors), FN: False Negatives (or omissions), and TN: True Negatives.

While Accuracy measures the goodness of the models, the Precision and the Recall measure the errors , one in false detection rates and the other in omissions. F1-Score, which is harmonic mean between precision and recall, is a measure of the robustness of a model.

The overall test results including all test set (80 images) are shown in Table A-3. The test results by each set of test images are also analyzed in detail and the results are presented in Tables A-5, A-6, A-7, and A-8.

Table A-3 shows that Darknet-53 and EfficientNet-B0 are less accurate and, as expected, their F1-scores are very low compared to Drone-Net which is a state of the art drone detection model. Darknet-53 and EfficientNet-B0 models are already trained by using COCO Lin *et al.* (2014) image set to detect up to 80 classes. These classes are different kind of objects such as humans, cars, trees, birds, dogs, bags, trains etc. And all kind of air vehicles are labeled under the same class: aeroplanes. A detailed look at the number of true positive predictions on Table A-3 shows that Darknet-53 and EfficientNet-B0 are missing most of the drone detections. For these reason both models have also high number of false negative predictions and, thus, a low F1-score compared to Drone-Net. As a direct conclusion for counter drone systems, it is not feasible to use directly these generic models to detect only drones. For this reason, the new models proposed here provide an improved way to detect only drones with an acceptable accuracy. For example, model-1, model-2 and model-3 have high accuracies, reaching 85%, 91% and 86% respectively. Equivalently, the models have high F1-scores, 90%, 94% and 91% respectively, and model-2 is the one with highest scores compared to the other models. This is very important because in counter drone systems, it is expected to detect a drone precisely. In other words, the number of false detections are expected to be zero or as low as possible, avoiding failing in the detection of non-expected intruder drones. As it is seen in Table A-3, model-2 has the lowest false detections compared to the other models.

Table A-3: *Overall Test Results*

Model	TP	TN	FP	FN	Accuracy	F1-Score
Drone-Net	37	18	7	18	69 %	0.75
Darknet-53	14	20	7	39	43 %	0.38
EfficientNet-B0	18	20	5	37	48 %	0.46
Model-1	52	16	6	6	85 %	0.90
Model-2	54	19	2	5	91 %	0.94
Model-3	53	16	5	6	86 %	0.91

Real-time performance of the proposed models are compared in Table A-4. The evaluation metrics such as inference time in milli-seconds (ms) and BFLOPS (Billions of floating-point operations required per second) are used to compare each models. The models are tested on Tesla T4, 16GB GPU which is commonly used GPU among the researchers. Model-1 has the fastest inference time with 4.838 milli-seconds. Model-3 performs slightly faster than model-2, although model-3 has the highest number of layers, but it has lowest BFLOPS 3.670.

Table A-4: *Comparison of Real Time Performance of the models*

Model	Inference Time (ms)	BFLOPS
Model-1	4.838	5.448
Model-2	40.756	65.304
Model-3	38.222	3.670

To better understand how the models perform predictions, we do a deeper analysis of the results for each test dataset.The separated Airsim images results are presented in Table A-5. We

see that the performance of the models are satisfactory in detecting Airsim images. Model-1 and model-3 have an accuracy of 70% while the model-2 has a higher rate of correct detection 80%. There is only one FP detection in model-1 but model-2 and model-3 have no FP detections. The F1-Score is also calculated for every models. Model-2 has the highest F1-score, 0.89 compared to model-1 and model-3, 0.82 and 0.83 respectively.

Table A-5: *Airsim Images Test Results*

Model	TP	TN	FP	FN	Accuracy	F1-Score
Model-1	14	0	1	5	70 %	0.82
Model-2	16	0	0	4	80 %	0.89
Model-3	14	0	0	6	70 %	0.83

Table A-6 has the partial results for brand new images taken from the Internet. Observe that the model-1, model-2 and model-3 have very good rate of detections, 90%, 90% and 95% respectively. Additionally, models achieved the higher F1-scores such that model-1, model-2 and model-3 have very promising F1-Scores, 0.95, 0.95 and 0.98 respectively.

Table A-6: *Web Images Test Results*

Model	TP	TN	FP	FN	Accuracy	F1-Score
Model-1	18	0	1	1	90 %	0.95
Model-2	18	0	1	1	90 %	0.95
Model-3	19	0	1	0	95 %	0.98

When looking at the partial results of the models tested with Drone-Net images, model-1, model-2 and model-3 detect the images with higher accuracy, 100% as it is seen in Table A-7 . Also, the models have higher F1-Scores, 1.

Table A-7: *Drone-Net Images Test Results*

Model	TP	TN	FP	FN	Accuracy	F1-Score
Model-1	20	0	0	0	100%	1
Model-2	20	0	0	0	100%	1
Model-3	20	0	0	0	100%	1

Finally, the partial models test results are shown for the images which do not include drones in Table A-8. It is expected that the models shall not detect any object, including images which have similar drone shapes. All the models have higher accuracy (above 80%), although model-1 and model-3 have few FP detections. The F1-Score is undefined given that the number of TP is zero.

Table A-8: *No-Drone Images Test Results*

Model	TP	TN	FP	FN	Accuracy	F1-Score
Model-1	0	16	4	0	80%	NA
Model-2	0	19	1	0	95%	NA
Model-3	0	16	4	0	80%	NA

Figure A-5 shows a same Airsim test image to compare the detection results of the four models able to predict only the class drone. Drone-Net model prediction shown in Figure A-5(a) fails

detecting drones in Airsim test images. However, Model-1 which has the same configuration with Drone-Net model but trained with Airsim images successfully detected a drone (see Figure A-5(b)). Model-2 and Model-3 detection tests shown in Figure A-5(c) and Figure A-5(d) are both correct drone detections.

(a) Drone-Net (b) Model-1 (c) Model-2 (d) Model-3

Figure A-5: *Airsim Image Test Detection Results*

Figure A-6 shows the test results of models for a challenging image of the web image set. All new models have successfully detected the drone in the image. However, the state of the art drone detection model Drone-Net failed to detect the drone for such a noisy background. In addition, bounding box sizes can be in different sizes. For example in Figure A-6(c), bounding box is larger than the expected size which can just cover drone predicted. However, this is not a general case in all test images. Different size of bounding boxes can be the result of background noise and the scale of the drone dimensions.

(a) Drone-Net (b) Model-1 (c) Model-2 (d) Model-3

Figure A-6: *Web Source Image Test Detection Results*

A.1.3 Discussion

In this section further analysis is discussed.

Both state of the art object detection algorithms Darknet-53 (Model-2) and EfficientNet-B0 (Model-3) have shown similar results. However, in real world applications such as counter drone systems, an object detection method is needed to be operated with limited resources. EfficientNet-B0 provides state of the art accuracy with 9 times smaller neural network size and it consumes significantly less computation compared to other state-of-the-art object detectors. As future work the EfficientNet-B0 drone detection model tested here can be the part of our counter drone system in which, using deep reinforcement learning methods, a guardian drone can be able to detect and counter malicious drones, while respecting and avoiding obstacles and legal drones. The accuracy of object detection and the fast response time are very important challenges to track and catch the drones.

A.1.3.1 Inaccurate Bounding Boxes of the Auto-labeling Process

In auto-labeling, it was observed that there were inaccurate bounding boxes, and they had to be removed from the training set. Almost 10% of the auto-labeled images had been detected as inaccurate. In Figure A-7 some of the inaccurate labeled images are presented. For instance, in Figure A-7(a) there is a bounding box on top left of the image and there is no drone shown inside it. The drone position at the moment of the image capture was very close to the other drone and the mapping from world coordinates to image coordinates was not correct. We believe that this

issue was caused by the processing time between capturing the image and obtaining its world position from the simulator. The processing time does not cause problems when drones are far from each other, but when they are too close their relative speed is high and the retrieved drone location is already obsolete. However, if the drone stays stationary, the problem disappears and none of the bounding boxes are inaccurate. Other erroneous and disregarded bounding boxes, not centered correctly in the images, are shown in Figures A-7(b), A-7(c) and A-7(d), all of them refer to a drone at the border or outside of the captured image.

(a) (b) (c) (d)

Figure A-7: *Inaccurate Bounding Boxes in Auto-labeling*

A.1.3.2 False Positive Detections

In this section False Positive (FP) detections by the models are discussed and analyzed. These are error cases where another object has been mistakenly labeled as a drone. The FP images can be seen in Figure A-8.

As the origin of this research we tested Drone-Net to detect drones on some images and found that most of them were FP, specially when dealing with Airsim images. The Figures A-8(a), A-8(b), A-8(c), A-8(d) and A-8(e) are the Airsim test images detected as FP. These figures show that Drone-Net is not accurate enough to detect a drone in these images. The detections are bounded to the objects such as trees, wires or covering the all image. In Figure A-8(f), Drone-Net detects a drone, but it also detects the commercial aircraft as drone. Drone-Net also detects two of the images from No-Drone test image set as false positives. Figures A-8(g) and A-8(h) show that objects such as human and direction sign are detected as FP by Drone-Net.

Model-1 has also FP images. For example, in Figure A-8(i) large part of the image detected a drone similar to the Drone-Net model. However, most of the FP detections exist in No-Drone test images. Figures A-8(j), A-8(k), A-8(l), and A-8(m) show that the drone kind of shapes could be detected as a drone.

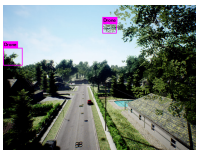
In addition, model-2 has two FP images and one of them is from No-Drone test image set. A noisy image from No-Drone image set is tested and then model-2 detects a smoke as a drone. This FP image seen in A-8(o) can be caused by shape of the smoke which appears as a drone in the image.

As it is observed previously in other models, Model-3 has also detected objects which are not drone. A drone is detected in few of the test images from the No-Drone test set seen in Figures A-8(q), A-8(r), A-8(s), and A-8(t). However, there are no drones in these images.

One of the common FP detections among the models are shown in Figures A-8(n), A-8(p), and A-8(u). In these figures, it is seen that a drone is detected in one of the test images from the web. However, another object, a commercial aircraft, is detected as a drone which is FP instead of a drone at bottom left of the aircraft.


(a) Drone-Net FP-1


(b) Drone-Net FP-2


(c) Drone-Net FP-3


(d) Drone-Net FP-4


(e) Drone-Net FP-5


(f) Drone-Net FP-6

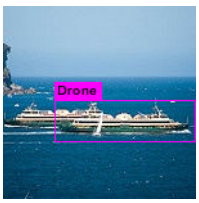
(g) Drone-Net FP-7

(h) Drone-Net FP-8


(i) Model-1 FP-1

(j) Model-1 FP-2

(k) Model-1 FP-3



(l) Model-1 FP-4

(m) Model-1 FP-5

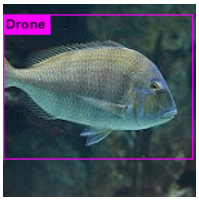

(n) Model-1 FP-6

(o) Model-2 FP-1


(p) Model-2 FP-2

(q) Model-3 FP-1

(r) Model-3 FP-2



(s) Model-3 FP-3

(t) Model-3 FP-4


(u) Model-3 FP-5

Figure A-8: *FP Images*

A.2 Darknet-53 CNN Summary

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure A-9: *Darknet-53 Redmon & Farhadi (2018)*

A.3 Drone-Net & Model-1 CNN Summary

```
mini_batch = 1, batch = 8, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 conv     16       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  16 0.150 BF
   1 max                2x 2/ 2    416 x 416 x  16 ->  208 x 208 x  16 0.003 BF
   2 conv     32       3 x 3/ 1    208 x 208 x  16 ->  208 x 208 x  32 0.399 BF
   3 max                2x 2/ 2    208 x 208 x  32 ->  104 x 104 x  32 0.001 BF
   4 conv     64       3 x 3/ 1    104 x 104 x  32 ->  104 x 104 x  64 0.399 BF
   5 max                2x 2/ 2    104 x 104 x  64 ->   52 x  52 x  64 0.001 BF
   6 conv    128       3 x 3/ 1     52 x  52 x  64 ->   52 x  52 x 128 0.399 BF
   7 max                2x 2/ 2     52 x  52 x 128 ->   26 x  26 x 128 0.000 BF
   8 conv    256       3 x 3/ 1     26 x  26 x 128 ->   26 x  26 x 256 0.399 BF
   9 max                2x 2/ 2     26 x  26 x 256 ->   13 x  13 x 256 0.000 BF
  10 conv    512       3 x 3/ 1     13 x  13 x 256 ->   13 x  13 x 512 0.399 BF
  11 max                2x 2/ 1     13 x  13 x 512 ->   13 x  13 x 512 0.000 BF
  12 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  13 conv    256       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 256 0.089 BF
  14 conv    512       3 x 3/ 1     13 x  13 x 256 ->   13 x  13 x 512 0.399 BF
  15 conv     18       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x  18 0.003 BF
  16 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
  17 route  13                                     ->   13 x  13 x 256
  18 conv    128       1 x 1/ 1     13 x  13 x 256 ->   13 x  13 x 128 0.011 BF
  19 upsample                2x    13 x  13 x 128 ->   26 x  26 x 128
  20 route  19 8                                   ->   26 x  26 x 384
  21 conv    256       3 x 3/ 1     26 x  26 x 384 ->   26 x  26 x 256 1.196 BF
  22 conv     18       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x  18 0.006 BF
  23 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
```

Figure A-10: *Drone-Net and Model-1 CNN Summary*

A.4 Model-2 CNN Summary

```
mini_batch = 1, batch = 16, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF
   2 conv     32       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  32 0.177 BF
   3 conv     64       3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  64 1.595 BF
   4 Shortcut Layer: 1,  wt = 0, wn = 0, outputs: 208 x 208 x  64 0.003 BF
   5 conv    128       3 x 3/ 2    208 x 208 x  64 ->  104 x 104 x 128 1.595 BF
   6 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
   7 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
   8 Shortcut Layer: 5,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
   9 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
  10 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
  11 Shortcut Layer: 8,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
  12 conv    256       3 x 3/ 2    104 x 104 x 128 ->   52 x  52 x 256 1.595 BF
  13 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  14 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  15 Shortcut Layer: 12,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  16 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  17 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  18 Shortcut Layer: 15,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  19 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  20 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  21 Shortcut Layer: 18,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  22 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  23 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  24 Shortcut Layer: 21,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  25 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  26 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  27 Shortcut Layer: 24,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  28 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  29 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  30 Shortcut Layer: 27,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  31 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  32 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  33 Shortcut Layer: 30,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  34 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  35 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  36 Shortcut Layer: 33,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  37 conv    512       3 x 3/ 2     52 x  52 x 256 ->   26 x  26 x 512 1.595 BF
  38 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  39 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  40 Shortcut Layer: 37,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
```

```
41 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
42 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
43 Shortcut Layer: 40,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
44 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
45 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
46 Shortcut Layer: 43,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
47 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
48 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
49 Shortcut Layer: 46,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
50 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
51 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
52 Shortcut Layer: 49,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
53 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
54 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
55 Shortcut Layer: 52,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
56 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
57 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
58 Shortcut Layer: 55,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
59 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
60 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
61 Shortcut Layer: 58,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
62 conv   1024       3 x 3/ 2     26 x  26 x 512 ->   13 x  13 x1024 1.595 BF
63 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
64 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
65 Shortcut Layer: 62,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
66 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
67 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
68 Shortcut Layer: 65,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
69 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
70 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
71 Shortcut Layer: 68,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
72 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
73 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
74 Shortcut Layer: 71,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
75 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
76 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
77 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
78 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
79 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
80 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
81 conv     18       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x  18 0.006 BF
82 yolo
```

```
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
  83 route  79                                     ->   13 x  13 x 512
  84 conv    256       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x 256 0.044 BF
  85 upsample                2x    13 x  13 x 256 ->   26 x  26 x 256
  86 route  85 61                                  ->   26 x  26 x 768
  87 conv    256       1 x 1/ 1     26 x  26 x 768 ->   26 x  26 x 256 0.266 BF
  88 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  89 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  90 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  91 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  92 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  93 conv     18       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x  18 0.012 BF
  94 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
  95 route  91                                     ->   26 x  26 x 256
  96 conv    128       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x 128 0.044 BF
  97 upsample                2x    26 x  26 x 128 ->   52 x  52 x 128
  98 route  97 36                                  ->   52 x  52 x 384
  99 conv    128       1 x 1/ 1     52 x  52 x 384 ->   52 x  52 x 128 0.266 BF
 100 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 101 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 102 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 103 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 104 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 105 conv     18       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x  18 0.025 BF
 106 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
```

Figure A-11: *Model-2 CNN Summary*

A.5 Model-3 CNN Summary

```
mini_batch = 1, batch = 8, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 conv     32       3 x 3/ 2    416 x 416 x   3 ->  208 x 208 x  32 0.075 BF
   1 conv     32       1 x 1/ 1    208 x 208 x  32 ->  208 x 208 x  32 0.089 BF
   2 conv     32/  32  3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  32 0.025 BF
   3 avg                           208 x 208 x  32 ->     32
   4 conv      8       1 x 1/ 1      1 x   1 x  32 ->    1 x   1 x   8 0.000 BF
   5 conv     32       1 x 1/ 1      1 x   1 x   8 ->    1 x   1 x  32 0.000 BF
   6 scale Layer: 2
   7 conv     16       1 x 1/ 1    208 x 208 x  32 ->  208 x 208 x  16 0.044 BF
   8 conv     96       1 x 1/ 1    208 x 208 x  16 ->  208 x 208 x  96 0.133 BF
   9 conv     96/  96  3 x 3/ 2    208 x 208 x  96 ->  104 x 104 x  96 0.019 BF
  10 avg                           104 x 104 x  96 ->     96
  11 conv     16       1 x 1/ 1      1 x   1 x  96 ->    1 x   1 x  16 0.000 BF
  12 conv     96       1 x 1/ 1      1 x   1 x  16 ->    1 x   1 x  96 0.000 BF
  13 scale Layer: 9
  14 conv     24       1 x 1/ 1    104 x 104 x  96 ->  104 x 104 x  24 0.050 BF
  15 conv    144       1 x 1/ 1    104 x 104 x  24 ->  104 x 104 x 144 0.075 BF
  16 conv    144/ 144  3 x 3/ 1    104 x 104 x 144 ->  104 x 104 x 144 0.028 BF
  17 avg                           104 x 104 x 144 ->    144
  18 conv      8       1 x 1/ 1      1 x   1 x 144 ->    1 x   1 x   8 0.000 BF
  19 conv    144       1 x 1/ 1      1 x   1 x   8 ->    1 x   1 x 144 0.000 BF
  20 scale Layer: 16
  21 conv     24       1 x 1/ 1    104 x 104 x 144 ->  104 x 104 x  24 0.075 BF
  22 dropout    p = 0.000        259584  ->   259584
  23 Shortcut Layer: 14,  wt = 0, wn = 0, outputs: 104 x 104 x  24 0.000 BF
  24 conv    144       1 x 1/ 1    104 x 104 x  24 ->  104 x 104 x 144 0.075 BF
  25 conv    144/ 144  5 x 5/ 2    104 x 104 x 144 ->   52 x  52 x 144 0.019 BF
  26 avg                           52 x  52 x 144 ->    144
  27 conv      8       1 x 1/ 1      1 x   1 x 144 ->    1 x   1 x   8 0.000 BF
  28 conv    144       1 x 1/ 1      1 x   1 x   8 ->    1 x   1 x 144 0.000 BF
  29 scale Layer: 25
  30 conv     40       1 x 1/ 1     52 x  52 x 144 ->   52 x  52 x  40 0.031 BF
  31 conv    192       1 x 1/ 1     52 x  52 x  40 ->   52 x  52 x 192 0.042 BF
  32 conv    192/ 192  5 x 5/ 1     52 x  52 x 192 ->   52 x  52 x 192 0.026 BF
  33 avg                           52 x  52 x 192 ->    192
  34 conv     16       1 x 1/ 1      1 x   1 x 192 ->    1 x   1 x  16 0.000 BF
  35 conv    192       1 x 1/ 1      1 x   1 x  16 ->    1 x   1 x 192 0.000 BF
  36 scale Layer: 32
  37 conv     40       1 x 1/ 1     52 x  52 x 192 ->   52 x  52 x  40 0.042 BF
  38 dropout    p = 0.000        108160  ->   108160
  39 Shortcut Layer: 30,  wt = 0, wn = 0, outputs:  52 x  52 x  40 0.000 BF
  40 conv    192       1 x 1/ 1     52 x  52 x  40 ->   52 x  52 x 192 0.042 BF
```

```
41 conv    192/ 192  3 x 3/ 1     52 x  52 x 192 ->   52 x  52 x 192 0.009 BF
42 avg                            52 x  52 x 192 ->    192
43 conv     16       1 x 1/ 1      1 x   1 x 192 ->    1 x   1 x  16 0.000 BF
44 conv    192       1 x 1/ 1      1 x   1 x  16 ->    1 x   1 x 192 0.000 BF
45 scale Layer: 41
46 conv     80       1 x 1/ 1     52 x  52 x 192 ->   52 x  52 x  80 0.083 BF
47 conv    384       1 x 1/ 1     52 x  52 x  80 ->   52 x  52 x 384 0.166 BF
48 conv    384/ 384  3 x 3/ 1     52 x  52 x 384 ->   52 x  52 x 384 0.019 BF
49 avg                            52 x  52 x 384 ->    384
50 conv     24       1 x 1/ 1      1 x   1 x 384 ->    1 x   1 x  24 0.000 BF
51 conv    384       1 x 1/ 1      1 x   1 x  24 ->    1 x   1 x 384 0.000 BF
52 scale Layer: 48
53 conv     80       1 x 1/ 1     52 x  52 x 384 ->   52 x  52 x  80 0.166 BF
54 dropout    p = 0.000        216320  ->   216320
55 Shortcut Layer: 46,  wt = 0, wn = 0, outputs:  52 x  52 x  80 0.000 BF
56 conv    384       1 x 1/ 1     52 x  52 x  80 ->   52 x  52 x 384 0.166 BF
57 conv    384/ 384  3 x 3/ 1     52 x  52 x 384 ->   52 x  52 x 384 0.019 BF
58 avg                            52 x  52 x 384 ->    384
59 conv     24       1 x 1/ 1      1 x   1 x 384 ->    1 x   1 x  24 0.000 BF
60 conv    384       1 x 1/ 1      1 x   1 x  24 ->    1 x   1 x 384 0.000 BF
61 scale Layer: 57
62 conv     80       1 x 1/ 1     52 x  52 x 384 ->   52 x  52 x  80 0.166 BF
63 dropout    p = 0.000        216320  ->   216320
64 Shortcut Layer: 55,  wt = 0, wn = 0, outputs:  52 x  52 x  80 0.000 BF
65 conv    384       1 x 1/ 1     52 x  52 x  80 ->   52 x  52 x 384 0.166 BF
66 conv    384/ 384  5 x 5/ 2     52 x  52 x 384 ->   26 x  26 x 384 0.013 BF
67 avg                            26 x  26 x 384 ->    384
68 conv     24       1 x 1/ 1      1 x   1 x 384 ->    1 x   1 x  24 0.000 BF
69 conv    384       1 x 1/ 1      1 x   1 x  24 ->    1 x   1 x 384 0.000 BF
70 scale Layer: 66
71 conv    112       1 x 1/ 1     26 x  26 x 384 ->   26 x  26 x 112 0.058 BF
72 conv    576       1 x 1/ 1     26 x  26 x 112 ->   26 x  26 x 576 0.087 BF
73 conv    576/ 576  5 x 5/ 1     26 x  26 x 576 ->   26 x  26 x 576 0.019 BF
74 avg                            26 x  26 x 576 ->    576
75 conv     32       1 x 1/ 1      1 x   1 x 576 ->    1 x   1 x  32 0.000 BF
76 conv    576       1 x 1/ 1      1 x   1 x  32 ->    1 x   1 x 576 0.000 BF
77 scale Layer: 73
78 conv    112       1 x 1/ 1     26 x  26 x 576 ->   26 x  26 x 112 0.087 BF
79 dropout    p = 0.000        75712  ->   75712
80 Shortcut Layer: 71,  wt = 0, wn = 0, outputs:  26 x  26 x 112 0.000 BF
81 conv    576       1 x 1/ 1     26 x  26 x 112 ->   26 x  26 x 576 0.087 BF
82 conv    576/ 576  5 x 5/ 1     26 x  26 x 576 ->   26 x  26 x 576 0.019 BF
83 avg                            26 x  26 x 576 ->    576
84 conv     32       1 x 1/ 1      1 x   1 x 576 ->    1 x   1 x  32 0.000 BF
85 conv    576       1 x 1/ 1      1 x   1 x  32 ->    1 x   1 x 576 0.000 BF
86 scale Layer: 82
87 conv    112       1 x 1/ 1     26 x  26 x 576 ->   26 x  26 x 112 0.087 BF
```

```
 88 dropout    p = 0.000        75712  ->   75712
 89 Shortcut Layer: 80,  wt = 0, wn = 0, outputs:  26 x  26 x 112 0.000 BF
 90 conv    576       1 x 1/ 1     26 x  26 x 112 ->   26 x  26 x 576 0.087 BF
 91 conv    576/ 576  5 x 5/ 2     26 x  26 x 576 ->   13 x  13 x 576 0.005 BF
 92 avg                           13 x  13 x 576 ->    576
 93 conv     32       1 x 1/ 1      1 x   1 x 576 ->    1 x   1 x  32 0.000 BF
 94 conv    576       1 x 1/ 1      1 x   1 x  32 ->    1 x   1 x 576 0.000 BF
 95 scale Layer: 91
 96 conv    192       1 x 1/ 1     13 x  13 x 576 ->   13 x  13 x 192 0.037 BF
 97 conv    960       1 x 1/ 1     13 x  13 x 192 ->   13 x  13 x 960 0.062 BF
 98 conv    960/ 960  5 x 5/ 1     13 x  13 x 960 ->   13 x  13 x 960 0.008 BF
 99 avg                           13 x  13 x 960 ->    960
100 conv     64       1 x 1/ 1      1 x   1 x 960 ->    1 x   1 x  64 0.000 BF
101 conv    960       1 x 1/ 1      1 x   1 x  64 ->    1 x   1 x 960 0.000 BF
102 scale Layer: 98
103 conv    192       1 x 1/ 1     13 x  13 x 960 ->   13 x  13 x 192 0.062 BF
104 dropout    p = 0.000        32448  ->   32448
105 Shortcut Layer: 96,  wt = 0, wn = 0, outputs:  13 x  13 x 192 0.000 BF
106 conv    960       1 x 1/ 1     13 x  13 x 192 ->   13 x  13 x 960 0.062 BF
107 conv    960/ 960  5 x 5/ 1     13 x  13 x 960 ->   13 x  13 x 960 0.008 BF
108 avg                           13 x  13 x 960 ->    960
109 conv     64       1 x 1/ 1      1 x   1 x 960 ->    1 x   1 x  64 0.000 BF
110 conv    960       1 x 1/ 1      1 x   1 x  64 ->    1 x   1 x 960 0.000 BF
111 scale Layer: 107
112 conv    192       1 x 1/ 1     13 x  13 x 960 ->   13 x  13 x 192 0.062 BF
113 dropout    p = 0.000        32448  ->   32448
114 Shortcut Layer: 105,  wt = 0, wn = 0, outputs:  13 x  13 x 192 0.000 BF
115 conv    960       1 x 1/ 1     13 x  13 x 192 ->   13 x  13 x 960 0.062 BF
116 conv    960/ 960  5 x 5/ 1     13 x  13 x 960 ->   13 x  13 x 960 0.008 BF
117 avg                           13 x  13 x 960 ->    960
118 conv     64       1 x 1/ 1      1 x   1 x 960 ->    1 x   1 x  64 0.000 BF
119 conv    960       1 x 1/ 1      1 x   1 x  64 ->    1 x   1 x 960 0.000 BF
120 scale Layer: 116
121 conv    192       1 x 1/ 1     13 x  13 x 960 ->   13 x  13 x 192 0.062 BF
122 dropout    p = 0.000        32448  ->   32448
123 Shortcut Layer: 114,  wt = 0, wn = 0, outputs:  13 x  13 x 192 0.000 BF
124 conv    960       1 x 1/ 1     13 x  13 x 192 ->   13 x  13 x 960 0.062 BF
125 conv    960/ 960  3 x 3/ 1     13 x  13 x 960 ->   13 x  13 x 960 0.003 BF
126 avg                           13 x  13 x 960 ->    960
127 conv     64       1 x 1/ 1      1 x   1 x 960 ->    1 x   1 x  64 0.000 BF
128 conv    960       1 x 1/ 1      1 x   1 x  64 ->    1 x   1 x 960 0.000 BF
129 scale Layer: 125
130 conv    320       1 x 1/ 1     13 x  13 x 960 ->   13 x  13 x 320 0.104 BF
131 conv   1280       1 x 1/ 1     13 x  13 x 320 ->   13 x  13 x1280 0.138 BF
132 conv    256       1 x 1/ 1     13 x  13 x1280 ->   13 x  13 x 256 0.111 BF
133 conv    256       3 x 3/ 1     13 x  13 x 256 ->   13 x  13 x 256 0.199 BF
134 Shortcut Layer: 132,  wt = 0, wn = 0, outputs:  13 x  13 x 256 0.000 BF


135 conv     18       1 x 1/ 1     13 x  13 x 256 ->   13 x  13 x  18 0.002 BF
136 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
137 route  133                                           ->   13 x  13 x 256
138 conv    128       1 x 1/ 1     13 x  13 x 256 ->   13 x  13 x 128 0.011 BF
139 upsample                2x     13 x  13 x 128 ->   26 x  26 x 128
140 Shortcut Layer: 90,  wt = 0, wn = 0, outputs:  26 x  26 x 128 0.000 BF
(  26 x  26 x 128) + (  26 x  26 x 576)
141 conv    128       3 x 3/ 1     26 x  26 x 128 ->   26 x  26 x 128 0.199 BF
142 Shortcut Layer: 139,  wt = 0, wn = 0, outputs:  26 x  26 x 128 0.000 BF
143 Shortcut Layer: 90,  wt = 0, wn = 0, outputs:  26 x  26 x 128 0.000 BF
(  26 x  26 x 128) + (  26 x  26 x 576)
144 conv     18       1 x 1/ 1     26 x  26 x 128 ->   26 x  26 x  18 0.003 BF
145 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, cls_norm: 1.00, scale_x_y: 1.00
```

Figure A-12: *Model-3 CNN Summary*

B

NN Model Summary in Chapter V

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 1, 30, 100)	0	
conv1 (Conv2D)	(None, 32, 7, 25)	544	input_1[0][0]
conv2 (Conv2D)	(None, 15, 3, 64)	14464	conv1[0][0]
vel (InputLayer)	(None, 1, 2)	0	
dst (InputLayer)	(None, 1, 3)	0	
geo (InputLayer)	(None, 1, 2)	0	
ae (InputLayer)	(None, 1, 2)	0	
flat1 (Flatten)	(None, 2880)	0	conv2[0][0]
reshape_1 (Reshape)	(None, 2)	0	vel[0][0]
reshape_2 (Reshape)	(None, 3)	0	dst[0][0]
reshape_3 (Reshape)	(None, 2)	0	geo[0][0]
reshape_4 (Reshape)	(None, 2)	0	ae[0][0]
concatenate_1 (Concatenate)	(None, 2889)	0	flat1[0][0] reshape_1[0][0] reshape_2[0][0] reshape_3[0][0] reshape_4[0][0]
dense_1 (Dense)	(None, 256)	739840	concatenate_1[0][0]
dense_2 (Dense)	(None, 256)	65792	dense_1[0][0]
dense_3 (Dense)	(None, 256)	65792	dense_2[0][0]
dense_4 (Dense)	(None, 3)	771	dense_3[0][0]

Total params: 887,203

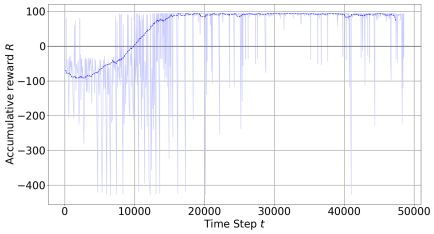
Figure B-1: *Neural Network Model Summary*

C

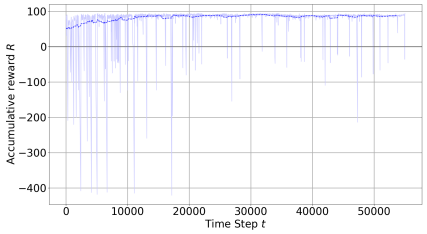
Supplemental Data for Chapter VI

Table C-1: *Hyperparametres of the training.*

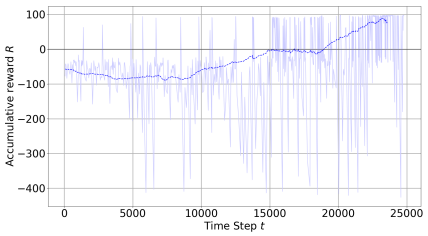
Hyperparameter	Value	Observations
Training steps	75,000	Changes in different scenarios (50,000–75,000)
Annealing length	15,000	Changes in different scenarios (15,000–45,000)
Annealing interval ϵ	[1–0.1]	Linear Annealed Policy (can be [0.1–0.01])
Steps to warm-up	180	Number of random steps to take before learning begins
Prioritized experience replay, memory limit	100,000	
Prioritized experience replay, alpha	0.6	Decides how much prioritization is used
Prioritized experience replay, beta		Decides how much we should compensate for the non-uniform probabilities
Prioritized experience replay, start-beta	0.4	
Prioritized experience replay, end-beta	0.4	
Pretraining steps	1000	Length of 'pretraining'
Large margin	0.8	Constant value
Lam_2	1	Imitation loss coefficient
Dueling type	'avg'	A type of dueling architecture
Target model update τ	0.001	Frequency of the target network update
Discount factor γ	0.99	The discount factor of future rewards in the Q function
Learning rate α	0.00025	Adam optimizer Kingma & Ba (2014)



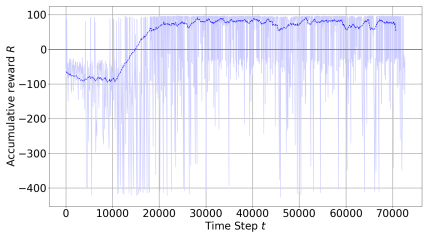
(a) Model-1



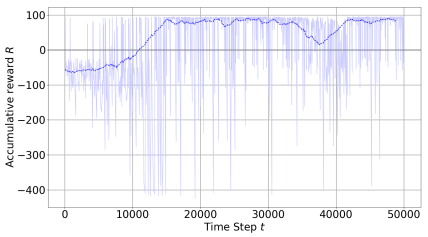
(b) Model-2



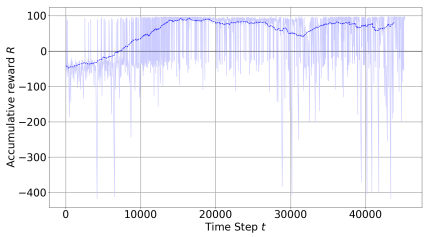
(c) Model-3



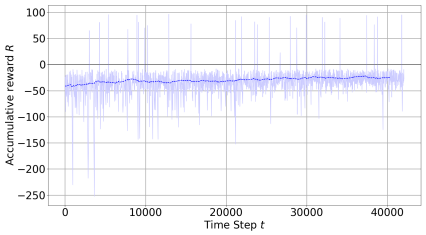
(d) Model-4



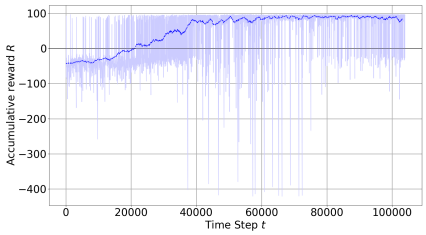
(e) Model-5



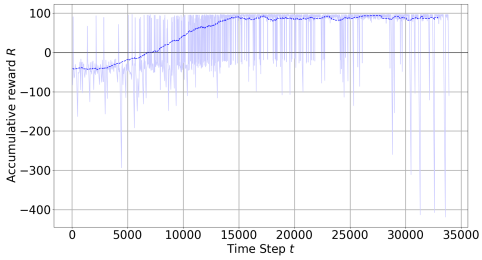
(f) Model-6



(g) Model-7

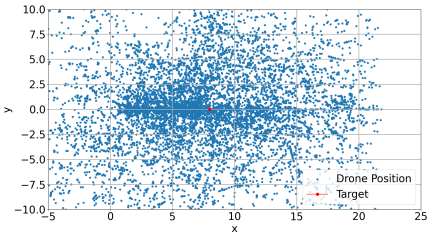


(h) Model-8

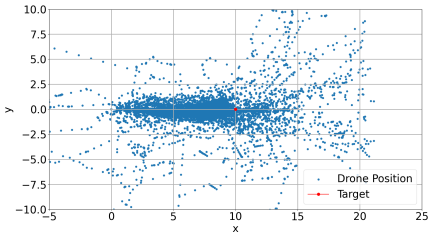


(i) Model-9

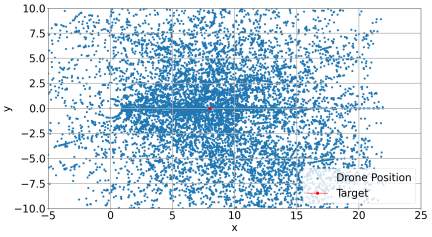
Figure C-1: *Training Results.*



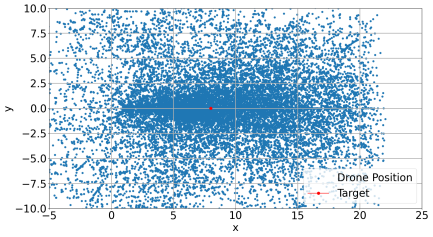
(a) Model-1



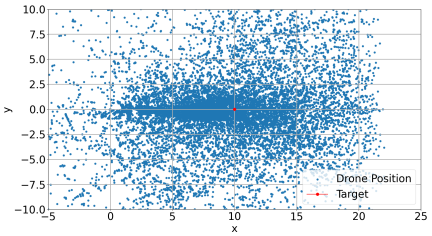
(b) Model-2



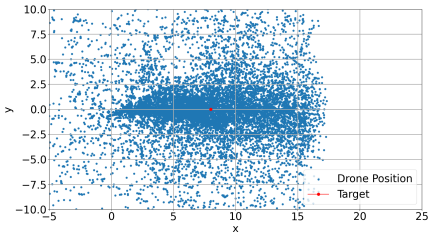
(c) Model-3



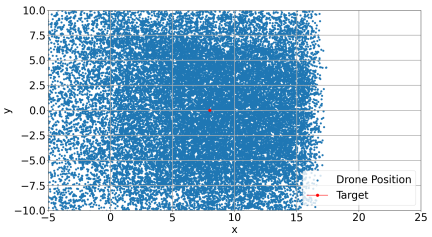
(d) Model-4



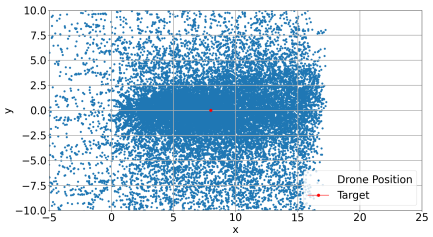
(e) Model-5



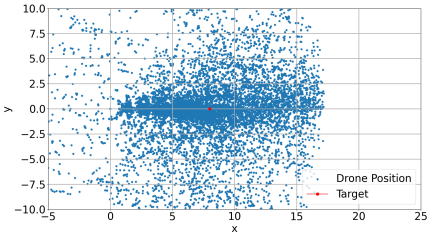
(f) Model-6



(g) Model-7

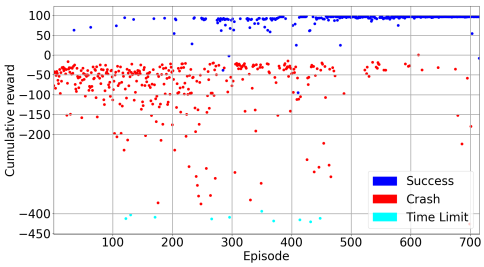


(h) Model-8

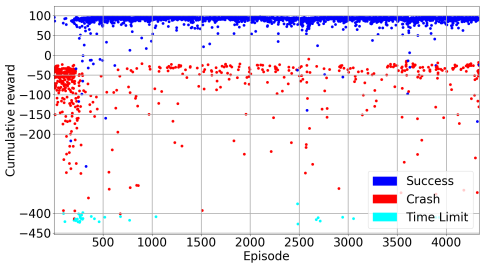


(i) Model-9

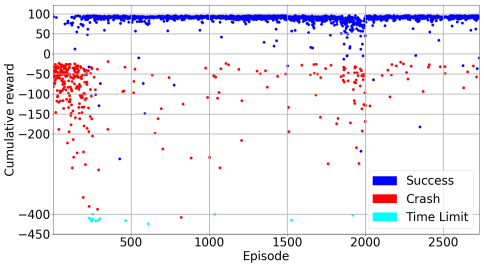
Figure C-2: *Drone Positions.*



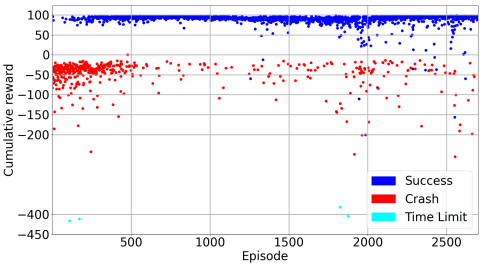
(a) Model-3



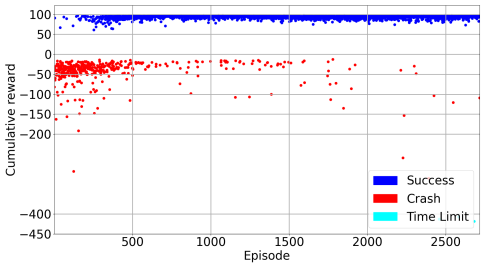
(b) Model-4



(c) Model-5

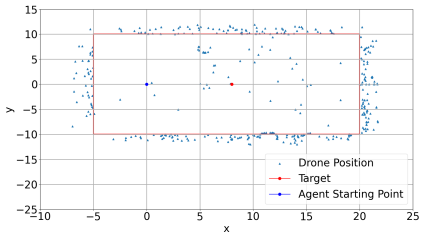


(d) Model-6

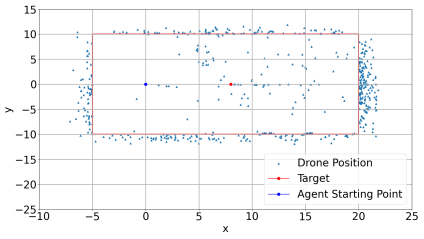


(e) Model-9

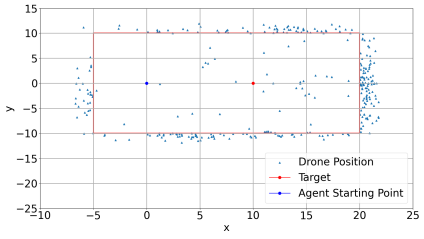
Figure C-3: *Alternative Training Results.*



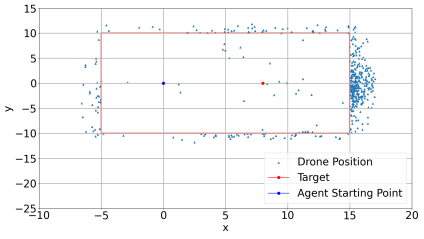
(a) Model-3



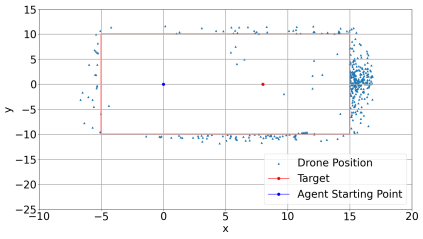
(b) Model-4



(c) Model-5

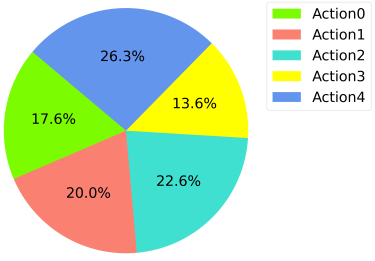


(d) Model-6

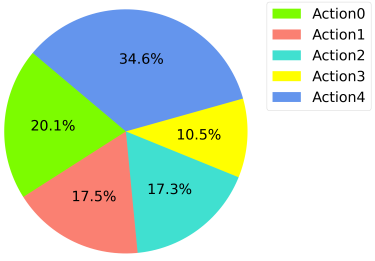


(e) Model-9

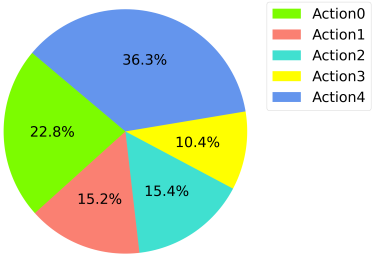
Figure C-4: *Crashed Episodes.*



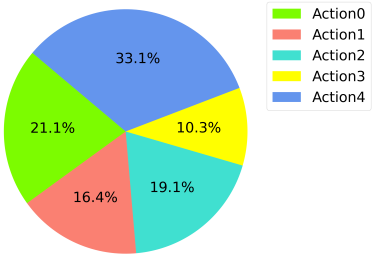
(a) Model-3



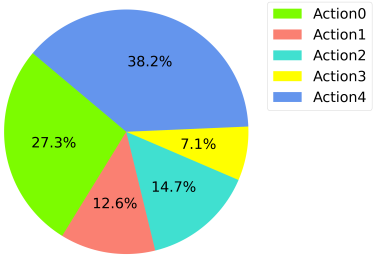
(b) Model-4



(c) Model-5



(d) Model-6



(e) Model-9

Figure C-5: *Actions Frequency.*

Bibliography

20MINUTOS. 2022. *Aena desvía cinco aviones que iban a aterrizar en barajas por la presencia de drones*. https://www.20minutos.es/noticia/5045830/0/aena-desvia-aviones-aterrizar-barajas-presencia-drones/. Last Accessed: 2022-11-29. 10

ABADI, MARTIN, AGARWAL, ASHISH, BARHAM, PAUL, BREVDO, EUGENE, CHEN, ZHIFENG, CITRO, CRAIG, CORRADO, GREG S, DAVIS, ANDY, DEAN, JEFFREY, DEVIN, MATTHIEU, *et al.* 2015. Tensorflow: Large-scale machine learning on heterogeneous systems, software available from tensorflow. org (2015). *Url https://www. tensorflow. org*. 51

AKER, CEMAL, & KALKAN, SINAN. 2017. Using deep networks for drone detection. *Pages 1–6 of: 2017 14th ieee international conference on advanced video and signal based surveillance (avss)*. IEEE. 4

AKHLOUFI, MOULAY A, AROLA, SEBASTIEN, & BONNET, ALEXANDRE. 2019. Drones chasing drones: Reinforcement learning and deep search area proposal. *Drones*, **3**(3), 58. 3

AMAZON. 2023. *Aws deepracer*. https://aws.amazon.com/deepracer/. Last Accessed: 2023-01-03. 48, 49

AND SPACE, AIRBUS DEFENCE. 2023. *Pioneering the stratosphere*. https://mediacentre.airbus.com/mediacentre/media?mediaTitle=title_Pioneering+the+Stratosphere&mediaId=546046. Last Accessed: 2023-02-07. 2, 3

ANJOMSHOAE, SULE, NAJJAR, AMRO, CALVARESI, DAVIDE, & FRÄMLING, KARY. 2019. Explainable agents and robots: Results from a systematic literature review. *Pages 1078–1088 of: 18th international conference on autonomous agents and multiagent systems (aamas 2019), montreal, canada, may 13–17, 2019*. International Foundation for Autonomous Agents and Multiagent Systems. 57

ANWAR, AQEEL, & RAYCHOWDHURY, ARIJIT. 2020. Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning. *Ieee access*, **8**, 26549–26560. 32, 46

ARAIZA-ILLAN, D, & EDER, K. 2019. Safe and trustworthy human-robot interaction. *Humanoid robotics: A reference, eds a. goswami and p. vadakkepat (dordrecht: Springer netherlands)*, 2397–2419. 57

BARIŠIC, ANTONELLA, PETRIC, FRANO, & BOGDAN, STJEPAN. 2022. Brain over brawn: Using a stereo camera to detect, track, and intercept a faster uav by reconstructing the intruder's trajectory. 5

BARRADO, CRISTINA, BOYERO, MARIO, BRUCCULERI, LUIGI, FERRARA, GIANCARLO, HATELY, ANDREW, HULLAH, PETER, MARTIN-MARRERO, DAVID, PASTOR, ENRIC, RUSHTON, ANTHONY PETER, & VOLKERT, ANDREAS. 2020. U-space concept of operations: A key enabler for opening airspace to emerging low-altitude operations. *Aerospace*, **7**(3), 24. ix, 7, 8

BBC-UK. 2019. *Flights diverted after gatwick airport*. https://www.bbc.com/news/uk-england-sussex-48086013. Last Accessed: 2019-08-23. 8

BELLEMARE, MARC G, DABNEY, WILL, & MUNOS, RÉMI. 2017. A distributional perspective on reinforcement learning. *Pages 449–458 of: International conference on machine learning*. PMLR. 34

BERNARDINI, ANDREA, MANGIATORDI, FEDERICA, PALLOTTI, EMILIANO, & CAPODIFERRO, LICIA. 2017a. Drone detection by acoustic signature identification. *Electronic imaging*, **2017**(10), 60–64. 4

BERNARDINI, ANDREA, MANGIATORDI, FEDERICA, PALLOTTI, EMILIANO, & CAPODIFERRO, LICIA. 2017b. Drone detection by acoustic signature identification. *Electronic imaging*, **2017**(10), 60–64. 5

BERTOIN, DAVID, GAUFFRIAU, ADRIEN, GRASSET, DAMIEN, & GUPTA, JAYANT SEN. 2021. Autonomous drone interception with deep reinforcement learning. 4

BESADA, JUAN A, CAMPAÑA, IVAN, CARRAMIÑANA, DAVID, BERGESIO, LUCA, & DE MIGUEL, GONZALO. 2021. Review and simulation of counter-uas sensors for unmanned traffic management. *Sensors*, **22**(1), 189. 45

BHATTARAI, NEMI, NAKAMURA, TAI, & MOZUMDER, CHITRINI. 2018. Real time human detection and localization using consumer grade camera and commercial uav. 42

BILD. 2022. *Dürfen sich die bvb-stars wehren?* https://www.bild.de/sport/fussball/fussball/bvb-drohnen-gefahr-am-phoenixsee-duerfen-sich-die-dortmund-stars-wehren-80644868.bild.html. Last Accessed: 2022-11-29. 10

BISONG, EKABA. 2019. Google colaboratory. *Pages 59–64 of: Building machine learning and deep learning models on google cloud platform*. Springer. 56

BOCHKOVSKIY, ALEXEY. 2020. *Yolo-v4 and yolo-v3/v2 for windows and linux*. https://github.com/AlexeyAB/darknet note = Last Accessed: 2020-05-27. 53, 117

BRITANNICA. 2022. *Go-game*. https://www.britannica.com/topic/go-game. Last Accessed: 2022-12-07. 17

BROCKMAN, GREG, CHEUNG, VICKI, PETTERSSON, LUDWIG, SCHNEIDER, JONAS, SCHULMAN, JOHN, TANG, JIE, & ZAREMBA, WOJCIECH. 2016. *Openai gym*. 48, 49

BROWNE, CAMERON B, POWLEY, EDWARD, WHITEHOUSE, DANIEL, LUCAS, SIMON M, COWLING, PETER I, ROHLFSHAGEN, PHILIPP, TAVENER, STEPHEN, PEREZ, DIEGO, SAMOTHRAKIS, SPYRIDON, & COLTON, SIMON. 2012. A survey of monte carlo tree search methods. *Ieee transactions on computational intelligence and ai in games*, **4**(1), 1–43. 34

CARRIO, ADRIAN, TORDESILLAS, JESUS, VEMPRALA, SAI, SARIPALLI, SRIKANTH, CAMPOY, PASCUAL, & HOW, JONATHAN P. 2020. Onboard detection and localization of drones using depth maps. *Ieee access*, **8**, 30480–30490. 43

ÇETIN, E, BARRADO, C, & PASTOR, E. 2021. Improving real-time drone detection for counter-drone systems. *The aeronautical journal*, 1–26. 4

CHAUVIN, YVES, & RUMELHART, DAVID E. 2013. *Backpropagation: theory, architectures, and applications*. Psychology press. 30

CHEN, WU, MENG, XUE, LIU, JIAJIA, GUO, HONGZHI, & MAO, BOMIN. 2022. Countering large-scale drone swarm attack by efficient splitting. *Ieee transactions on vehicular technology*, **71**(9), 9967–9979. 45

CHEW, ROBERT, RINEER, JAY, BEACH, ROBERT, O'NEIL, MAGGIE, UJENEZA, NOEL, LAPIDUS, DANIEL, MIANO, THOMAS, HEGARTY-CRAVER, MEGHAN, POLLY, JASON, & TEMPLE, DOROTA S. 2020. Deep neural networks and transfer learning for food crop identification in uav images. *Drones*, **4**(1), 7. 2

CHIPER, FLORIN-LUCIAN, MARTIAN, ALEXANDRU, VLADEANU, CALIN, MARGHESCU, ION, CRACIUNESCU, RAZVAN, & FRATU, OCTAVIAN. 2022. Drone detection and defense systems: Survey and a software-defined radio-based solution. *Sensors*, **22**(4), 1453. 4

CHOI, BYUNGGIL, OH, DAEGUN, KIM, SUNWOO, CHONG, JONG-WHA, & LI, YING-CHUN. 2018. Long-range drone detection of 24 g fmcw radar with e-plane sectoral horn array. *Sensors*, **18**(12), 4171. 5

CHOLLET, FRANÇOIS. 2017. Xception: Deep learning with depthwise separable convolutions. *Pages 1251–1258 of: Proceedings of the ieee conference on computer vision and pattern recognition*. 40, 41

COMMISSION, THE EUROPEAN. 2023. *Commission implementing regulation (eu) 2019/947 of 24 may 2019 on the rules and procedures for the operation of unmanned aircraft*. https://www.easa.europa.eu/en/document-library/regulations/commission-implementing-regulation-eu-2019947. Last Accessed: 2023-03-02. 45

CONG, LIN WILLIAM, TANG, KE, WANG, JINGYUAN, & ZHANG, YANG. 2021. Alphaportfolio: Direct construction through deep reinforcement learning and interpretable ai. *Available at ssrn*, **3554486**. 16

CRAYE, CELINE, & ARDJOUNE, SALEM. 2019. Spatio-temporal semantic segmentation for drone detection. *Pages 1–5 of: 2019 16th ieee international conference on advanced video and signal based surveillance (avss)*. IEEE. 43

CRITEO-RESEARCH. 2023. *Reco gym*. https://github.com/criteo-research/reco-gym. Last Accessed: 2023-01-03. 48, 49

CYBERBOTICS. 2023. *Webots: open-source robot simulator*. https://github.com/cyberbotics/webots. Last Accessed: 2023-01-03. 48, 49

DARPA. 2023. *Explainable artificial intelligence (xai)*. https://www.darpa.mil/program/explainable-artificial-intelligence. Last Accessed: 2023-01-16. 57

DE HAAG, MAARTEN UIJT, BARTONE, CHRIS G, & BRAASCH, MICHAEL S. 2016a. Flight-test evaluation of small form-factor lidar and radar sensors for suas detect-and-avoid applications. *Pages 1–11 of: 2016 ieee/aiaa 35th digital avionics systems conference (dasc)*. IEEE. 4

DE HAAG, MAARTEN UIJT, BARTONE, CHRIS G, & BRAASCH, MICHAEL S. 2016b. Flight-test evaluation of small form-factor lidar and radar sensors for suas detect-and-avoid applications. *Pages 1–11 of: 2016 ieee/aiaa 35th digital avionics systems conference (dasc)*. IEEE. 5

DEDRONE. 2022. *Worldwide drone incidents*. https://www.dedrone.com/resources/incidents/all. Last Accessed: 2022-11-29. ix, 8, 11

DEEPMIND. 2023a. *Ai safety gridworlds*. https://github.com/deepmind/ai-safety-gridworlds. Last Accessed: 2023-01-03. 48, 49

DEEPMIND. 2023b. *Deepmind control suite*. https://github.com/deepmind/dm_control. Last Accessed: 2023-01-03. 49

DEEPMIND. 2023c. *Deepmind lab*. https://github.com/deepmind/lab. Last Accessed: 2023-01-03. 48, 49

DEEPMIND. 2023d. *Deepmind openspiel*. https://github.com/deepmind/open_spiel. Last Accessed: 2023-01-03. 48, 49

DEEPMIND. 2023e. *Deepmind pysc2*. https://github.com/deepmind/pysc2. Last Accessed: 2023-01-03. 48, 49

DJI. 2023. *Matrice 300 rtk*. https://store.dji.com/es/product/matrice-300-rtk-and-dji-care-plus?from=store-nav&vid=111261. Last Accessed: 2023-02-07. 3

DOSOVITSKIY, ALEXEY, & KOLTUN, VLADLEN. 2016. Learning to act by predicting the future. *arxiv preprint arxiv:1611.01779*. 69

DROZDOWICZ, JEDRZEJ, WIELGO, MACIEJ, SAMCZYNSKI, PIOTR, KULPA, KRZYSZTOF, KRZONKALLA, JAROSLAW, MORDZONEK, MAJ, BRYL, MARCIN, & JAKIELASZEK, ZBIGNIEW. 2016. 35 ghz fmcw drone detection system. *Pages 1–4 of: 2016 17th international radar symposium (irs)*. IEEE. 4

DUO, JINGYUN, & ZHAO, LONG. 2017. Uav autonomous navigation system for gnss invalidation. *Pages 5777–5782 of: 2017 36th chinese control conference (ccc)*. IEEE. 44

EASA. 2022. *Urban air mobility (uam by easa)*. https://www.easa.europa.eu/en/light/topics/urban-air-mobility. Last Accessed: 2022-11-28. 2

ELPERIÓDICO. 2022. *Los mossos denuncian a un hombre por hacer volar un dron sobre barcelona sin permiso*. https://amp-elperiodico-com.cdn.ampproject.org/c/s/amp.elperiodico.com/es/barcelona/20220207/mossos-denuncian-hombre-volar-dron-13202461. Last Accessed: 2022-11-29. 10

EPICGAMES. 2019. *Unreal engine 4*. https://www.unrealengine.com/en-US/what-is-unreal-engine-4. Last Accessed: 2019-01-29. 32, 48

FAA. 2022. *Drones by the numbers*. https://www.faa.gov/uas/resources/by_the_numbers/. Last Accessed: 2022-08-04. 1

FARAMA-FOUNDATION. 2023. *Vizdoom*. https://github.com/mwydmuch/ViZDoom. Last Accessed: 2023-01-03. 48, 49

FESTO. 2022. *Smartbird*. https://www.festo.com/es/es/e/sobre-festo/investigacion-y-desarrollo/bionic-learning-network/lo-mas-destacado-de-2010-2012/smartbird-id_33686/. Last Accessed: 2022-11-29. 3

FRANÇOIS-LAVET, VINCENT, HENDERSON, PETER, ISLAM, RIASHAT, BELLEMARE, MARC G, PINEAU, JOELLE, *et al.* 2018. An introduction to deep reinforcement learning. *Foundations and trends® in machine learning*, **11**(3-4), 219–354. 34

FRSKY. 2021. *Taranis x9d plus*. https://www.frsky-rc.com/product/taranis-x9d-plus-2/. Last Accessed: 2021-06-24. 106

FUJIMOTO, SCOTT, HOOF, HERKE, & MEGER, DAVID. 2018. Addressing function approximation error in actor-critic methods. *Pages 1587–1596 of: International conference on machine learning*. PMLR. 34

GAZEBO. 2023. *Gazebo sim : A robotic simulator*. https://github.com/gazebosim/gz-sim. Last Accessed: 2023-01-03. 48, 49

GIRSHICK, ROSS. 2015. Fast r-cnn. *Pages 1440–1448 of: Proceedings of the ieee international conference on computer vision*. 40

GLASGOWLIVE. 2022. *Flights to glasgow airport forced to divert after suspected drone sighting*. https://www.glasgowlive.co.uk/news/glasgow-news/flights-glasgow-airport-forced-divert-25092938. Last Accessed: 2022-11-29. 10

GNANASEKARAN, ABEYNAYA, FABA, JORDI FELIU, & AN, JING. 2017. Reinforcement learning in pacman. *See nalso url http://cs229. stanford. edu/proj2017/final-reports/5241109. pdf*. 16

GOODFELLOW, IAN, BENGIO, YOSHUA, & COURVILLE, AARON. 2016a. *Deep learning*. MIT Press. http://www.deeplearningbook.org. ix, 28, 30

GOODFELLOW, IAN, BENGIO, YOSHUA, & COURVILLE, AARON. 2016b. *Deep learning*. MIT press. ix, 26, 30, 31

GOOGLE. 2023a. *Explainable ai*. https://cloud.google.com/explainable-ai. Last Accessed: 2023-01-16. 57

GOOGLE. 2023b. *Machine learning glossary: Reinforcement learning*. https://developers.google.com/machine-learning/glossary/rl#e. Last Accessed: 2023-03-03. 19

GOV.UK. 2020. *Loss of control following bird attack*. https://www.gov.uk/aaib-reports/aaib-investigation-to-dji-matrice-m200-uas-registration-n-a-080720?utm. Last Accessed: 2022-12-01. 10

GRIGSBY, JAKE. 2023. *lunardqfd*. https://github.com/jakegrigsby/lunarDQfD. Last Accessed: 2023-02-23. 39

GURRIET, THOMAS, & CIARLETTA, LAURENT. 2016. Towards a generic and modular geofencing strategy for civilian uavs. *Pages 540–549 of: 2016 international conference on unmanned aircraft systems (icuas)*. IEEE. 45

HA, DAVID, & SCHMIDHUBER, JÜRGEN. 2018. World models. *arxiv preprint arxiv:1803.10122*. 34

HAARNOJA, TUOMAS, HA, SEHOON, ZHOU, AURICK, TAN, JIE, TUCKER, GEORGE, & LEVINE, SERGEY. 2018a. Learning to walk via deep reinforcement learning. *arxiv preprint arxiv:1812.11103*. 16

HAARNOJA, TUOMAS, ZHOU, AURICK, HARTIKAINEN, KRISTIAN, TUCKER, GEORGE, HA, SEHOON, TAN, JIE, KUMAR, VIKASH, ZHU, HENRY, GUPTA, ABHISHEK, ABBEEL, PIETER, *et al.* 2018b. Soft actor-critic algorithms and applications. *arxiv preprint arxiv:1812.05905*. 34

HASSANALIAN, MOSTAFA, & ABDELKEFI, ABDESSATTAR. 2017. Classifications, applications, and design challenges of drones: A review. *Progress in aerospace sciences*, **91**, 99–131. ix, 2

HASSELT, HADO. 2010. Double q-learning. *Advances in neural information processing systems*, **23**. 37

HE, LEI, AOUF, NABIL, WHIDBORNE, JAMES F, & SONG, BIFENG. 2020. Deep reinforcement learning based local planner for uav obstacle avoidance using demonstration data. *arxiv preprint arxiv:2008.02521*. 4

HESTER, TODD, VECERIK, MATEJ, PIETQUIN, OLIVIER, LANCTOT, MARC, SCHAUL, TOM, PIOT, BILAL, HORGAN, DAN, QUAN, JOHN, SENDONARIS, ANDREW, OSBAND, IAN, *et al.* 2018. Deep q-learning from demonstrations. *In: Proceedings of the aaai conference on artificial intelligence*, vol. 32. 39, 40

HODGE, VICTORIA J, HAWKINS, RICHARD, & ALEXANDER, ROB. 2021. Deep reinforcement learning for drone navigation using sensor data. *Neural computing and applications*, **33**(6), 2015–2033. 3

HOVELL, KIRK, ULRICH, STEVE, & BRONZ, MURAT. 2022. Learned multiagent real-time guidance with applications to quadrotor runway inspection. 46

HU, QINTAO, DUAN, QIANWEN, MAO, YAO, ZHOU, XI, & ZHOU, GUOZHONG. 2019. Diagonalnet: Confidence diagonal lines for the uav detection. *Ieej transactions on electrical and electronic engineering*, **14**(9), 1364–1371. 43

HU, ZIJIAN, WAN, KAIFANG, GAO, XIAOGUANG, ZHAI, YIWEI, & WANG, QIANGLONG. 2020. Deep reinforcement learning approach with multiple experience pools for uav's autonomous motion planning in complex unknown environments. *Sensors*, **20**(7), 1890. 44

IBM. 2023. *Explainable ai (xai)*. https://www.ibm.com/watson/explainable-ai. Last Accessed: 2023-01-16. 57

INGARGIOLA, TITO. 2023. *Gym trading*. https://github.com/hackthemarket/gym-trading. Last Accessed: 2023-01-03. 48, 49

JIN, REN, JIANG, JIAQI, QI, YUHUA, LIN, DEFU, & SONG, TAO. 2019. Drone detection and pose estimation using relational graph networks. *Sensors*, **19**(6), 1479. 43

JOHNSON, W LEWIS. 1994. Agents that learn to explain themselves. *Pages 1257–1263 of: Aaai*. Palo Alto, CA. 57

KALASHNIKOV, DMITRY, IRPAN, ALEX, PASTOR, PETER, IBARZ, JULIAN, HERZOG, ALEXANDER, JANG, ERIC, QUILLEN, DEIRDRE, HOLLY, ETHAN, KALAKRISHNAN, MRINAL, VANHOUCKE, VINCENT, *et al.* 2018. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arxiv preprint arxiv:1806.10293*. 34

KERSANDT, KJELL. 2018. *Deep reinforcement learning as control method for autonomous uavs*. M.Phil. thesis, Universitat Politècnica de Catalunya. 45, 73, 105

KERSANDT, KJELL, MUÑOZ, GUILLEM, & BARRADO, CRISTINA. 2018. Self-training by reinforcement learning for full-autonomous drones of the future. *Pages 1–10 of: Digital avionics systems conference (dasc), 2018 ieee/aiaa 37th*. IEEE. 38, 45

KICKER. 2021. *Spanish league responds to drone incident in bilbao*. https://www.kicker.de/spanische-liga-reagiert-auf-drohnenvorfall-in-bilbao-800825/artikel. Last Accessed: 2022-12-01. 10

KINGMA, DIEDERIK P, & BA, JIMMY LEI. 2014. Adam: Amethod for stochastic optimization. *In: Proc. 3rd int. conf. learn. representations*. 30, 136

KONDA, VIJAY, & TSITSIKLIS, JOHN. 1999. Actor-critic algorithms. *Advances in neural information processing systems*, **12**. 34

KOURIS, ALEXANDROS, & BOUGANIS, CHRISTOS-SAVVAS. 2018. Learning to fly by myself: A self-supervised cnn-based approach for autonomous navigation. *Pages 1–9 of: 2018 ieee/rsj international conference on intelligent robots and systems (iros)*. IEEE. 46

KRATKY, MIROSLAV, & FARLIK, JAN. 2018. Countering uavs–the mover of research in military technology. *Defence science journal*, **68**(5), 460–466. 4

KRONEN-ZEITUNG. 2021. *Drone crash during bayern training*. https://www.krone.at/2379205. Last Accessed: 2022-12-01. 10

KUGLER, LOGAN. 2019. Real-world applications for drones. *Communications of the acm*, **62**(11), 19–21. 2

LANGE, SASCHA, & RIEDMILLER, MARTIN. 2010. Deep auto-encoder neural networks in reinforcement learning. *Pages 1–8 of: The 2010 international joint conference on neural networks (ijcnn)*. IEEE. 36

LECUN, YANN, BENGIO, YOSHUA, *et al*. 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, **3361**(10), 1995. 30

LECUN, YANN, BOTTOU, LÉON, BENGIO, YOSHUA, & HAFFNER, PATRICK. 1998. Gradient-based learning applied to document recognition. *Proceedings of the ieee*, **86**(11), 2278–2324. ix, 32

LECUN, YANN A, BOTTOU, LÉON, ORR, GENEVIEVE B, & MÜLLER, KLAUS-ROBERT. 2012. Efficient backprop. *Pages 9–48 of: Neural networks: Tricks of the trade*. Springer. 30

LEE, HANSEOB. 2021. Development of an anti-drone system using a deep reinforcement learning algorithm. 4

LEWIS, FRANK L, VRABIE, DRAGUNA, & VAMVOUDAKIS, KYRIAKOS G. 2012. Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *Ieee control systems magazine*, **32**(6), 76–105. 27

LILLICRAP, TIMOTHY P, HUNT, JONATHAN J, PRITZEL, ALEXANDER, HEESS, NICOLAS, EREZ, TOM, TASSA, YUVAL, SILVER, DAVID, & WIERSTRA, DAAN. 2015. Continuous control with deep reinforcement learning. *arxiv preprint arxiv:1509.02971*. 34, 36

LIN, DAVID CHUAN-EN. 2020. *Drone-net*. https://github.com/chuanenlin/drone-net note = Last Accessed: 2020-06-02. 43, 55

LIN, LONG-JI. 1992. *Reinforcement learning for robots using neural networks*. Carnegie Mellon University. 35

LIN, LONG-JI. 1993. *Reinforcement learning for robots using neural networks*. Tech. rept. Carnegie-Mellon Univ Pittsburgh PA School of Computer Science. 35

LIN, TSUNG-YI, MAIRE, MICHAEL, BELONGIE, SERGE, HAYS, JAMES, PERONA, PIETRO, RAMANAN, DEVA, DOLLÁR, PIOTR, & ZITNICK, C LAWRENCE. 2014. Microsoft coco: Common objects in context. *Pages 740–755 of: European conference on computer vision*. Springer. 121

LU, XIAOZHEN, XIAO, LIANG, DAI, CANHUANG, & DAI, HUAIYU. 2018. Uav-aided cellular communications with deep reinforcement learning against jamming. *arxiv preprint arxiv:1805.06628*. 4

LYKOU, GEORGIA, MOUSTAKAS, DIMITRIOS, & GRITZALIS, DIMITRIS. 2020. Defending airports from uas: A survey on cyber-attacks and counter-drone sensing technologies. *Sensors*, **20**(12), 3537. 5

MATÚŠ, TÓTH. 2019. *Učení přistávání samořiditelnỳch letadel*. M.Phil. thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum. 46

MCCLELLAND, J. L., MCNAUGHTON B. L. & O'REILLY R. C. 1995. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Pages 419–457 of: Psychological review, 102(3)*. 35

META-RESEARCH. 2023. *Reagent*. https://github.com/facebookresearch/ReAgent. Last Accessed: 2023-01-03. 48, 49

MEZEI, JÓZSEF, FIASKA, VIKTOR, & MOLNÁR, ANDRÁS. 2015. Drone sound detection. *Pages 333–338 of: 2015 16th ieee international symposium on computational intelligence and informatics (cinti)*. IEEE. 4

MICHEL, ARTHUR HOLLAND. 2018. *Counter-drone systems*. Center for the Study of the Drone at Bard College. 5

MICROSOFT. 2023a. *Airsim documentation*. https://microsoft.github.io/AirSim. Last Accessed: 2023-02-18. 49, 73

MICROSOFT. 2023b. *Project malmo*. https://www.microsoft.com/en-us/research/project/project-malmo/. Last Accessed: 2023-01-03. 48, 49

MICROSOFT. 2023c. *Text world*. https://github.com/microsoft/TextWorld. Last Accessed: 2023-01-03. 48, 49

MNIH, VOLODYMYR, KAVUKCUOGLU, KORAY, SILVER, DAVID, GRAVES, ALEX, ANTONOGLOU, IOANNIS, WIERSTRA, DAAN, & RIEDMILLER, MARTIN A. 2013. Playing atari with deep reinforcement learning. *Corr*, **abs/1312.5602**. 33

MNIH, VOLODYMYR, KAVUKCUOGLU, KORAY, SILVER, DAVID, RUSU, ANDREI A, VENESS, JOEL, BELLEMARE, MARC G, GRAVES, ALEX, RIEDMILLER, MARTIN, FIDJELAND, ANDREAS K, OSTROVSKI, GEORG, *et al*. 2015. Human-level control through deep reinforcement learning. *Nature*, **518**(7540), 529. ix, 28, 33, 34, 35, 36, 38

MNIH, VOLODYMYR, BADIA, ADRIA PUIGDOMENECH, MIRZA, MEHDI, GRAVES, ALEX, LILLICRAP, TIMOTHY, HARLEY, TIM, SILVER, DAVID, & KAVUKCUOGLU, KORAY. 2016. Asynchronous methods for deep reinforcement learning. *Pages 1928–1937 of: International conference on machine learning*. PMLR. 33, 34

MUJOCO. 2023. *Mujoco advanced physics simulation*. https://mujoco.org/. Last Accessed: 2023-01-03. 48, 49

MUÑOZ, GUILLEM, BARRADO, CRISTINA, ÇETIN, ENDER, & SALAMI, ESTHER. 2019. Deep reinforcement learning for drone delivery. *Drones*, **3**(3), 72. 45

NAIR, VINOD, & HINTON, GEOFFREY E. 2010. Rectified linear units improve restricted boltzmann machines. *In: Icml*. 29

NGUYEN, PHUC, RAVINDRANATHA, MAHESH, NGUYEN, ANH, HAN, RICHARD, & VU, TAM. 2016. Investigating cost-effective rf-based detection of drones. *Pages 17–22 of: Proceedings of the 2nd workshop on micro aerial vehicle networks, systems, and applications for civilian use*. 4

NORTHROP-GRUMMAN-CORPORATION'S. 2022a. *Mq-8b fire scout*. https://www.northropgrumman.com/what-we-do/air/fire-scout/. Last Accessed: 2022-11-29. 3

NORTHROP-GRUMMAN-CORPORATION'S. 2022b. *Rq-4d phoenix global hawk*. https://news.northropgrumman.com/news/releases/nato-rq-4d-phoenix-achieves-major-milestone-with-full-system-handover. Last Accessed: 2022-11-29. 3

NRK. 2022. *Dronetrøbbel på norske flyplasser: Alarmen går hver dag*. https://www.nrk.no/norge/dronetrobbel-pa-norske-flyplasser_-alarmen-gar-hver-dag-1.16093172. Last Accessed: 2022-11-29. 10

OpenAI. 2022. *Key concepts in rl*. https://spinningup.openai.com/en/latest/spinningup/rl_intro.html. Last Accessed: 2022-12-13. 17, 18

Opromolla, Roberto, Fasano, Giancarmine, & Accardo, Domenico. 2018. A vision-based approach to uav detection and tracking in cooperative applications. *Sensors*, **18**(10), 3391. 4

Palmer, Jennifer, & Clothier, Reece. 2013. Analysis of the applicability of existing airworthiness classification schemes to the unmanned aircraft fleet. *Pages 228–244 of: Proceedings of 15th australian international aerospace congress (aiac15-aero)*. RMIT University. 1

Park, Seongjoon, Kim, Hyeong Tae, Lee, Sangmin, Joo, Hyeontae, & Kim, Hwangnam. 2021. Survey on anti-drone systems: Components, designs, and challenges. *Ieee access*, **9**, 42635–42659. 5

Parrot. 2023. *Parrot anafi usa*. https://www.parrot.com/us/drones/anafi-usa. Last Accessed: 2023-02-07. 3

Pham, Huy X., La, Hung M., Feil-Seifer, David, & Nguyen, Luan Van. 2018. Autonomous UAV navigation using reinforcement learning. *Corr*, **abs/1801.05086**. 44

Physics, Bullet. 2023. *Pybullet*. https://github.com/bulletphysics/bullet3. Last Accessed: 2023-01-03. 48, 49

Piot, Bilal, Geist, Matthieu, & Pietquin, Olivier. 2014. Boosted and reward-regularized classification for apprenticeship learning. *Pages 1249–1256 of: Proceedings of the 2014 international conference on autonomous agents and multi-agent systems*. 40

Plappert, Matthias. 2016. *keras-rl*. https://github.com/keras-rl/keras-rl. 51

Polvara, Riccardo, Patacchiola, Massimiliano, Sharma, Sanjay, Wan, Jian, Manning, Andrew, Sutton, Robert, & Cangelosi, Angelo. 2017. Autonomous quadrotor landing using deep reinforcement learning. *arxiv preprint arxiv:1709.03339*. 45

Poole, David L, & Mackworth, Alan K. 2010. *Artificial intelligence: foundations of computational agents*. Cambridge University Press. 34

Racanière, Sébastien, Weber, Théophane, Reichert, David, Buesing, Lars, Guez, Arthur, Jimenez Rezende, Danilo, Puigdomènech Badia, Adrià, Vinyals, Oriol, Heess, Nicolas, Li, Yujia, *et al.* 2017. Imagination-augmented agents for deep reinforcement learning. *Advances in neural information processing systems*, **30**. 34

Redmon, Joseph. 2013. Darknet: Open source neural networks in c. h ttp. *pjreddie. com/darknet*, **2016**. 53

Redmon, Joseph, & Farhadi, Ali. 2017. Yolo9000: better, faster, stronger. *Pages 7263–7271 of: Proceedings of the ieee conference on computer vision and pattern recognition*. 42

Redmon, Joseph, & Farhadi, Ali. 2018. Yolov3: An incremental improvement. *arxiv preprint arxiv:1804.02767*. xi, 53, 55, 126

Redmon, Joseph, Divvala, Santosh, Girshick, Ross, & Farhadi, Ali. 2016. You only look once: Unified, real-time object detection. *Pages 779–788 of: Proceedings of the ieee conference on computer vision and pattern recognition*. 3, 12, 40, 42

Ren, Shaoqing, He, Kaiming, Girshick, Ross, & Sun, Jian. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Pages 91–99 of: Advances in neural information processing systems*. 40, 41

Restás, Ágoston, Szalkai, István, & Óvári, Gyula. 2021. Drone application for spraying disinfection liquid fighting against the covid-19 pandemic—examining drone-related parameters influencing effectiveness. *Drones*, **5**(3), 58. 2

REUTERS. 2022. *Swedish security service investigates drones at three nuclear plants*. https://www.reuters.com/world/europe/swedish-security-service-investigates-drones-three-nuclear-plants-20 Last Accessed: 2022-12-01. 10

RICHTER, DAVID J, & CALIX, RICARDO A. 2021. Qplane: an open-source reinforcement learning toolkit for autonomous fixed wing aircraft simulation. *Pages 261–266 of: Proceedings of the 12th acm multimedia systems conference*. 48, 49

RIEDMILLER, MARTIN. 2005a. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. *Pages 317–328 of: European conference on machine learning*. Springer. 36

RIEDMILLER, MARTIN. 2005b. Neural fitted q iteration-first experiences with a data efficient neural reinforcement learning method. *Pages 317–328 of: Ecml*, vol. 3720. Springer. 35

RODRIGUEZ-RAMOS, ALEJANDRO, SAMPEDRO, CARLOS, BAVLE, HRIDAY, DE LA PUENTE, PALOMA, & CAMPOY, PASCUAL. 2019. A deep reinforcement learning strategy for uav autonomous landing on a moving platform. *Journal of intelligent & robotic systems*, **93**(1-2), 351–366. 4

ROYO, PABLO, ASENJO, ÀLEX, TRUJILLO, JUAN, ÇETIN, ENDER, & BARRADO, CRISTINA. 2022. Enhancing drones for law enforcement and capacity monitoring at open large events. *Drones*, **6**(11), 359. 42

RUDOL, PIOTR, & DOHERTY, PATRICK. 2008. Human body detection and geolocalization for uav search and rescue missions using color and thermal imagery. *Pages 1–8 of: 2008 ieee aerospace conference*. Ieee. 42

RUDYS, SAULIUS, LAUČYS, ANDRIUS, RAGULIS, PAULIUS, ALEKSIEJŪNAS, RIMVYDAS, STANKEVIČIUS, KAROLIS, KINKA, MARTYNAS, RAZGŪNAS, MATAS, BRUČAS, DOMANTAS, UDRIS, DAINIUS, & POMARNACKI, RAIMONDAS. 2022. Hostile uav detection and neutralization using a uav system. *Drones*, **6**(9), 250. 45

SALAMÍ, ESTHER, GALLARDO, ANTONIA, SKOROBOGATOV, GEORGY, & BARRADO, CRISTINA. 2019. On-the-fly olive tree counting using a uas and cloud services. *Remote sensing*, **11**(3), 316. 42

SAMARAS, STAMATIOS, DIAMANTIDOU, ELENI, ATALOGLOU, DIMITRIOS, SAKELLARIOU, NIKOS, VAFEIADIS, ANASTASIOS, MAGOULIANITIS, VASILIS, LALAS, ANTONIOS, DIMOU, ANASTASIOS, ZARPALAS, DIMITRIOS, VOTIS, KONSTANTINOS, *et al.* 2019. Deep learning on multi sensor data for counter uav applications—a systematic review. *Sensors*, **19**(22), 4837. 5

SCHAUL, TOM, QUAN, JOHN, ANTONOGLOU, IOANNIS, & SILVER, DAVID. 2015. Prioritized experience replay. *arxiv preprint arxiv:1511.05952*. 35

SCHMIDHUBER, JÜRGEN. 2015. Deep learning in neural networks: An overview. *Neural networks*, **61**, 85–117. 28

SCHULMAN, JOHN, LEVINE, SERGEY, ABBEEL, PIETER, JORDAN, MICHAEL, & MORITZ, PHILIPP. 2015. Trust region policy optimization. *Pages 1889–1897 of: International conference on machine learning*. PMLR. 34

SEMKIN, VASILII, YIN, MINGSHENG, HU, YAQI, MEZZAVILLA, MARCO, & RANGAN, SUNDEEP. 2021. Drone detection and classification based on radar cross section signatures. *Pages 223–224 of: 2020 international symposium on antennas and propagation (isap)*. IEEE. 4

SESAR. 2016. *European drones outlook study. unlocking the value for europe*. http://www.sesarju.eu/sites/default/files/documents/reports/European_Drones_Outlook_Study_2016.pdf. Last Accessed: 2021-11-29. 7

SESAR-JU. 2023a. *European atm master plan: Roadmap for the safe integration of drones into all classes of airspace*. https://www.sesarju.eu/node/2993. Last Accessed: 2019-05-26. 7

SESAR-JU. 2023b. *U-space*. https://www.sesarju.eu/U-space. Last Accessed: 2023-03-01. 7

SHAH, SHITAL, DEY, DEBADEEPTA, LOVETT, CHRIS, & KAPOOR, ASHISH. 2017. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *In: Field and service robotics*. x, 48, 50, 51

SHIELD-AI. 2023. *V-bat aircraft*. https://shield.ai/v-bat/. Last Accessed: 2023-01-02. 3

SILVER, DAVID. 2022. *Ucl course on rl*. https://www.davidsilver.uk/teaching/. Last Accessed: 2022-12-21. ix, 24, 25, 26

SILVER, DAVID, HUANG, AJA, MADDISON, CHRIS J, GUEZ, ARTHUR, SIFRE, LAURENT, VAN DEN DRIESSCHE, GEORGE, SCHRITTWIESER, JULIAN, ANTONOGLOU, IOANNIS, PANNEERSHELVAM, VEDA, LANCTOT, MARC, *et al.* 2016. Mastering the game of go with deep neural networks and tree search. *nature*, **529**(7587), 484–489. 33

SILVER, DAVID, SCHRITTWIESER, JULIAN, SIMONYAN, KAREN, ANTONOGLOU, IOANNIS, HUANG, AJA, GUEZ, ARTHUR, HUBERT, THOMAS, BAKER, LUCAS, LAI, MATTHEW, BOLTON, ADRIAN, *et al.* 2017. Mastering the game of go without human knowledge. *nature*, **550**(7676), 354–359. 28

SILVER, DAVID, HUBERT, THOMAS, SCHRITTWIESER, JULIAN, ANTONOGLOU, IOANNIS, LAI, MATTHEW, GUEZ, ARTHUR, LANCTOT, MARC, SIFRE, LAURENT, KUMARAN, DHARSHAN, GRAEPEL, THORE, *et al.* 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, **362**(6419), 1140–1144. 16

SKY-NEWS. 2022. *easyjet plane comes within '10 feet' of drone in 'close encounter'*. https://news.sky.com/story/easyjet-plane-comes-within-10-feet-of-drone-in-close-encounter-12729113. Last Accessed: 2022-11-29. 10

SROUJI, MARIO, ZHANG, JIAN, & SALAKHUTDINOV, RUSLAN. 2018. Structured control nets for deep reinforcement learning. *Pages 4749–4758 of:* DY, JENNIFER, & KRAUSE, ANDREAS (eds), *Proceedings of the 35th international conference on machine learning*. Proceedings of Machine Learning Research, vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR. 69

STANFORD-NMBL. 2023. *Opensim*. https://github.com/stanfordnmbl/osim-rl. Last Accessed: 2023-01-03. 48, 49

STEVENS, BRIAN L, LEWIS, FRANK L, & JOHNSON, ERIC N. 2015. *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons. 53

STV-NEWS. 2022. *'several sightings' of drone flown illegally near city airport*. https://news.stv.tv/north/police-appeal-for-information-after-drone-spotted-flying-illegally-near-aberdeen-airport Last Accessed: 2022-11-29. 10

SUTTON, RICHARD S, & BARTO, ANDREW G. 1998. *Reinforcement learning: An introduction*. MIT press Cambridge. ix, 18, 19, 21, 23, 24, 26, 27, 28, 29, 32, 33

SZEGEDY, CHRISTIAN, LIU, WEI, JIA, YANGQING, SERMANET, PIERRE, REED, SCOTT, ANGUELOV, DRAGOMIR, ERHAN, DUMITRU, VANHOUCKE, VINCENT, & RABINOVICH, ANDREW. 2015. Going deeper with convolutions. *Pages 1–9 of: Proceedings of the ieee conference on computer vision and pattern recognition*. 41

SZEPESVÁRI, CSABA. 2010. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, **4**(1), 1–103. 19

TAN, MINGXING, & LE, QUOC. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *Pages 6105–6114 of: International conference on machine learning*. PMLR. 40, 42, 55

TAN, MINGXING, PANG, RUOMING, & LE, QUOC V. 2020. Efficientdet: Scalable and efficient object detection. *Pages 10781–10790 of: Proceedings of the ieee/cvf conference on computer vision and pattern recognition*. 40, 42, 46

TAYLOR, MATTHEW E, & STONE, PETER. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of machine learning research*, **10**(Jul), 1633–1685. ix, 32, 33, 54

TENSORTRADE-ORG. 2023. *Tensor trade*. https://github.com/tensortrade-org/tensortrade. Last Accessed: 2023-01-03. 48, 49

THEFESTIVALS. 2022. *Glastonbury: Police detect illegal drone flight over festival*. https://thefestivals.uk/glastonbury-police-drone-detected/. Last Accessed: 2022-11-29. 10

TKN-TUBERLIN. 2023. *Ns3 gym*. https://github.com/tkn-tub/ns3-gym. Last Accessed: 2023-01-03. 49

TSITSIKLIS, JOHN, & VAN ROY, BENJAMIN. 1996. Analysis of temporal-diffference learning with function approximation. *Advances in neural information processing systems*, **9**. 33

UNDERTAKING, SESAR JOINT, *et al.* 2017. European drones outlook study: unlocking the value for europe. ix, 7, 8, 9

UNMANNED-AIRSPACE. 2017. *Anti-drone market "to be worth usd1.5 billion" by 2023 – new report*. https://www.unmannedairspace.info/utm-and-c-uas-market-analysis/anti-drone-market-to-be-worth-usd1-5-billion-by-2023-new-report/. Last Accessed: 2019-08-21. 1

VAN HASSELT, HADO, GUEZ, ARTHUR, & SILVER, DAVID. 2016a. Deep reinforcement learning with double q-learning. *Page 5 of: Aaai*, vol. 2. Phoenix, AZ. 33, 38

VAN HASSELT, HADO, GUEZ, ARTHUR, & SILVER, DAVID. 2016b. Deep reinforcement learning with double q-learning. *In: Proceedings of the aaai conference on artificial intelligence*, vol. 30. 33

VAN SEIJEN, HARM, FATEMI, MEHDI, ROMOFF, JOSHUA, LAROCHE, ROMAIN, BARNES, TAVIAN, & TSANG, JEFFREY. 2017. Hybrid reward architecture for reinforcement learning. *Pages 5392–5402 of:* GUYON, I., LUXBURG, U. V., BENGIO, S., WALLACH, H., FERGUS, R., VISHWANATHAN, S., & GARNETT, R. (eds), *Advances in neural information processing systems 30*. Curran Associates, Inc. 69

VOLOCOPTER-GMBH. 2023. *Volodrone*. https://www.volocopter.com/solutions/volodrone/. Last Accessed: 2023-02-07. 3

VON BOTHMER, FREDRIK. 2018. *Missing man: Contextualising legal reviews for autonomous weapon systems*. Ph.D. thesis, Universität St. Gallen. 44

WANG, ZIYU, SCHAUL, TOM, HESSEL, MATTEO, HASSELT, HADO, LANCTOT, MARC, & FREITAS, NANDO. 2016. Dueling network architectures for deep reinforcement learning. *Pages 1995–2003 of: International conference on machine learning*. PMLR. ix, 38, 39

WATKINS, CHRISTOPHER JCH, & DAYAN, PETER. 1992. Q-learning. *Machine learning*, **8**(3-4), 279–292. 27, 34

WATKINS, LANIER, SARTALAMACCHIA, SHANE, BRADT, RICHARD, DHARESHWAR, KARAN, BAGGA, HARSIMAR, ROBINSON, WILLIAM H, & RUBIN, AVIEL. 2020. Defending against consumer drone privacy attacks: A blueprint for a counter autonomous drone tool. *In: Workshop on decentralized iot systems and security (diss)*. 5, 45

WELLS, LINDSAY, & BEDNARZ, TOMASZ. 2021. Explainable ai and reinforcement learning—a systematic review of current approaches and trends. *Frontiers in artificial intelligence*, **4**, 550030. 57

WERBOS, PAUL J. 1989. Neural networks for control and system identification. *Pages 260–265 of: Proceedings of the 28th ieee conference on decision and control,*. IEEE. 33

WHO. 2021. *Coronavirus disease (covid-19) pandemic*. https://www.who.int/emergencies/diseases/novel-coronavirus-2019. Last Accessed: 2021-01-10. 42

WILLIAMS, RONALD J. 1988. On the use of backpropagation in associative reinforcement learning. *Pages 263–270 of: Icnn*. 34

WU, YUHUAI, MANSIMOV, ELMAN, GROSSE, ROGER B, LIAO, SHUN, & BA, JIMMY. 2017. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *Advances in neural information processing systems*, **30**. 34

WYDER, PHILIPPE MARTIN, CHEN, YAN-SONG, LASRADO, ADRIAN J, PELLES, RAFAEL J, KWIATKOWSKI, ROBERT, COMAS, EDITH OA, KENNEDY, RICHARD, MANGLA, ARJUN, HUANG, ZIXI, HU, XIAOTIAN, *et al*. 2019. Autonomous drone hunter operating by deep learning and all-onboard computations in gps-denied environments. *Plos one*, **14**(11). 44

X-PLANE. 2023. *X-plane 12 flight simulator*. https://www.x-plane.com/. Last Accessed: 2023-02-18. 48, 49

XIAOPING, LI, SONGZE, LEI, BOXING, ZHANG, YANHONG, WANG, & FENG, XIAO. 2019. Fast aerial uav detection using improved inter-frame difference and svm. *Page 032082 of: Journal of physics: Conference series*, vol. 1187. IOP Publishing. 43

YANG, DONGFANG, YURTSEVER, EKIM, RENGANATHAN, VISHNU, REDMILL, KEITH A, & ÖZGÜNER, ÜMIT. 2020. A vision-based social distance and critical density detection system for covid-19. *arxiv preprint arxiv:2007.03578*. 42

YANG, TING, ZHAO, LIYUAN, LI, WEI, WU, JIANZHONG, & ZOMAYA, ALBERT Y. 2021. Towards healthy and cost-effective indoor environment management in smart homes: A deep reinforcement learning approach. *Applied energy*, **300**, 117335. 16

ZHANG, HONGMING, & YU, TIANYANG. 2020. Taxonomy of reinforcement learning algorithms. *Pages 125–133 of: Deep reinforcement learning*. Springer. ix, 34

ZHU, ZHUANGDI, LIN, KAIXIANG, & ZHOU, JIAYU. 2020. Transfer learning in deep reinforcement learning: A survey. *arxiv preprint arxiv:2009.07888*. 32