Coventry University
Centre for Future Transport and Cities

Towards AI-Enabled Autonomous Aerial Vehicles for Railway Tracks Maintenance Needs Detection

Jun Jet Tai

Supervisor: Mauro Sebastián Innocente

Co-supervisors: Nadjim Horri, James Brusey

Acknowledgement

I would like to express my deepest gratitude to my primary supervisor, Dr. Mauro Sebastián Innocente, whose guidance, patience, and unwavering support have been instrumental in the completion of this work. I am equally grateful to my co-supervisors, Dr. Nadjim Horri and Dr. James Brusey, for their invaluable insights and mentorship throughout this journey.

I extend my thanks to Owais Mehmood from Omnicom Balfour Beatty for providing the initial dataset, which allowed me to carry out my first set of experiments. I would also like to acknowledge the team at BCIMO, particularly Nick Mallinson, who facilitated my work at the site in Dudley, UK, and Martin Franklin, who ensured that safety procedures were maintained during my time there. Special thanks go to Kevin Vincent for putting me in contact with BCIMO, a connection that proved vital to this project.

I am immensely grateful to the Farama Foundation for entrusting me with the opportunity to contribute to some of the world's most important open-source reinforcement learning libraries, such as Gymnasium and PettingZoo. My time working with these projects significantly shaped my engineering techniques. In particular, I would like to thank Jordan Terry, Mark Towers, Elliot Tower, and many others within the community whose collaboration and feedback were invaluable.

On a personal note, I would like to thank my older sister for being a constant source of physical familial support while in the UK, as well as my parents, younger sister, and close friends back home for their endless encouragement while being approximately 10,612 km away.

My heartfelt thanks also go to my colleagues at AVAILab Coventry — Mohammad Tavakol, Ioannis Papagianis, Daizy Rajput, Paolo Grasso, and Zaidan Zyadat — for their camaraderie and collaboration. I am also fortunate to have a supportive network of friends, including Cris Argamino and his wife Dianne, who have been a constant source of motivation and encouragement.

Lastly, I acknowledge that this project was made possible by the generous funding provided through Dr. M.S. Innocente's Trailblazers Award.

Abstract

The continued operation and reliability of critical systems like railways hinge upon effective maintenance practices. Various issues, ranging from missing fasteners and deformed track geometry to structural health insufficiencies and overgrown vegetation, can significantly impact system performance and lead to non-negligible downtimes. Traditional inspection methods, such as foot patrols, trolleys, and measurement vehicles, are comprehensive but often resource-intensive, time-consuming, and disruptive to routine traffic.

This research proposes a paradigm shift by exploring the application of autonomously operated Unmanned Aerial Vehicles (UAVs) for detecting issues requiring maintenance in railway systems. While UAVs have proven successful in other industries like bridge maintenance and building inspection, their application in railway systems spanning long distances remains largely unexplored. The investigation specifically focuses on inexpensive and agile UAVs operating autonomously, a pioneering approach in the railway maintenance domain.

The aim of this research is to identify application-based improvements and develop solutions enabled by autonomous UAVs for detecting maintenance needs. By leveraging technologies such as computer vision and machine learning, the project involves designing and implementing an autonomous UAV system, integrating the required sensors, developing novel and efficient algorithms for performing autonomous navigation and maintenance needs detection from data gathered onboard the UAV, and bridging any simulation to reality gap that may exist in the solution stack.

The outcomes of this research includes a framework for autonomous UAV navigation along railway corridors, plus techniques for performing maintenance needs detection using data gathered from the UAV itself, reducing the time and personnel manpower required for comprehensive inspections. This innovative approach not only aligns with current trends in engineering but also addresses a pressing challenge in railway maintenance. It is expected to impact society by minimising train service downtime, reducing costs, and enhancing the reliability, safety, and efficiency of both maintenance and operation of railway systems. At a more fundamental level, the contributions of this work include a hackable UAV simulation software named PyFlyt; Critic Confidence Guided Exploration — a reinforcement learning algorithm for allowing agents to bootstrap off oracle policies instead of learning from scratch; and algorithms and insights for performing anomaly detection from image data for railway systems.

Contents

Acronyms

ADeReL	Anomaly Detection by Reconstructing Labels
AI	Artificial Intelligence
AoA	Angle of Attack
API	Application Programming Interface
AWAC	Advantage Weighted Actor Critic
B2F	Bayesian to Frequentist
BCIMO	Black Country Innovative Manufacturing Organisation
BPTT	Back Propagation Through Time
CCGE	Critic Confidence Guided Exploration
CFD	Computational Fluid Dynamics
CI	Continuous Integration
CNN	Convolutional Neural Network
CV	Computer Vision
DAE	Digital Asset Exchange
DDPG	Deep Deterministic Policy Gradients
DQN	Deep Q Network
F2B	Frequentist to Bayesian
FPS	Frames per Second
FPT	First Passage Time
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
HOG	Histogram of Gradients
IoT	Internet of Things
IQL	Implicit Q Learning
IQM	Interquartile Mean
JSRL	Jump Start Reinforcement Learning
LiDaR	Light Distance and Ranging
LSTM	Long Short Term Memory
MDP	Markov Decision Process
MLP	Multilayer Perceptron
MSE	Mean Squared Error
MTBF	Mean Time Before Failure
NMT	New Measurement Train
OSA-CBM	Open System Architecture for Condition Based Monitoring

PdM	Predictive Maintenance
PID	Proportional-Integral-Derivative
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
PyPI	Python Packaging Index
RaDaR	Radio-Frequency Distance and Ranging
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RUL	Remaining Useful Life
SAC	Soft Actor Critic
SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transform
Sim2Real	Simulation to Reality Transfer
SLAM	Simulataneous Localization and Mapping
TD3	Twin Delayed Deep Deterministic Policy Gradients
TRPO	Trust Region Policy Optimization
UAV	Unmanned Aerial Vehicle
URDF	Universal Robot Definition Format
YAML	YAML Ain't Markup Language

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Critical systems such as railways must remain continuously operational, for which continued maintenance is paramount. Maintenance issues including missing fasteners, deformed track geometry, subgrade/ballast instabilities, structural health insufficiencies, obstructions and overgrown vegetation will cause slowdowns to these systems, which lead to non-negligible downtimes due to the level of interconnectedness at hand [Alrahman and Adham, 2024].

The traditional inspection method involves foot patrols or trolleys, although more modern techniques include the use of measurement vehicles [Gong et al., 2022]. These techniques can be comprehensive and accurate at the expense of significant manpower, time, and interruption of routine traffic.

The use of drones, also known as Unmanned Aerial Vehicles (UAVs), to identify maintenance needs is a technique that has seen successful deployment by many other industries such as bridge maintenance and building inspection [Chen et al., 2019b, Liu et al., 2021, Mader et al., 2016]. However, the use of such technologies for systems spanning long distances such as railways has not yet seen widespread use [Maghazei and Steinmann, 2020]. More crucially, the use of UAVs working *autonomously* has not been explored to a large extent in a traditionally risk-averse industry such as railways. In this work on Chapter 2.1.1, we estimated that the cost of monitoring the UK's railway network for issues related to maintenance costs anywhere from £19 million to £320 million. Being able to save even a small percentage of this value with investment in UAV technologies is likely to produce a significant return on investment. This project will therefore investigate the possibilities for autonomous UAVs to be used in the field of detecting maintenance needs for railway systems.

1.2 Aim and Objectives

The aim of the project is to:

Remove barriers to the successful deployment of
UAV systems for detecting maintenance needs on railway tracks

The objectives of the project are as follows:

1. To explore avenues where maintenance issues monitoring can benefit from UAV powered technologies.

2. To explore new algorithms for allowing UAVs to fly along railway tracks, testing these algorithms in simulation and the real world.

3. To develop vision-based algorithms for informing decision makers on maintenance points along the railway based on autonomously estimated track condition.

1.3 Research Questions

1. **What current practices are being used to detect maintenance issues along railway tracks and is there a gap that can be addressed with AI-powered technologies?**

   I aim to understand the current state of maintenance needs monitoring along railway tracks. Building on this, I aim to evaluate the tangible benefits of using UAVs for maintenance monitoring. This aligns with objective 1.

2. **Are machine learning methods for obstacle avoidance and navigation feasible to be used for navigating UAVs through railway corridors?**

   Modern UAV obstacle avoidance and navigation often rely on heuristic or hardcoded methods. I aim to explore whether machine learning techniques can be used instead, particularly for navigating railway corridors, while addressing the challenges of applying machine learning algorithms to real-world autonomous UAV flight. This aligns with objective 2.

3. **What current practices are being used to detect maintenance issues along railway tracks and what existing gaps can be addressed using AI-enabled UAV technologies?**

   Railway systems are complex electromechanical networks with numerous maintenance needs. I aim to identify which of these can be reliably detected using UAVs and what are the barriers limiting their broader use. This addresses objectives 1 and 3.

4. **What challenges present themselves when trying to perform maintenance monitoring on railways using machine learning from data gathered aboard a UAV and how can they be solved?**

   Knowing the maintenance needs that can be detected by a UAV-based system, I seek to understand the unique challenges associated with detecting these issues from a UAV. I also aim to develop techniques that make meaningful impact in addressing those challenges in this work. This research question addresses objective 3.

1.4 Thesis Structure

This thesis contains four primary chapters plus a concluding chapter and an appendix, this chapter is the first of those — the introduction.

Literature Review The literature review, chapter 2 on page 5, provides the required background for understanding the proceeding chapters of this thesis. In particular, it covers various techniques for performing autonomous UAV flight and past works using those techniques on tasks similar to flight along a railway. It also covers the preliminaries of computer vision and reinforcement learning — two paradigms that will be extensively studied in this thesis. The literature review also covers current techniques for the detection of maintenance needs on railways, as well as existing works detecting maintenance needs along various infrastructure, potentially using a UAV. The point of the review is to both present the state of the art and also show that there is a gap in current knowledge that this work aims to address.

Contribution Chapters This thesis contains two central contribution chapters. The chapter covering autonomous UAV flight along a railway is chapter 3 on page 51. Here, I introduce PyFlyt in section 3.4 on page 55 — an open-source contribution of a UAV simulation engine for reinforcement learning research — and its use in developing a simulated railway environment for UAVs flight in section 3.5 on page 68. Then, I introduce a novel Reinforcement Learning (RL) algorithm termed Critic Confidence Guided Exploration (CCGE) in section 3.6 on page 72 and demonstrate its use in the simulated railway environment in section 3.7 on page 88. I demonstrate how to utilise an image segmentation model to perform simulation-to-reality transfer in section 3.8 on page 91 and finally show results of real world deployment in section 3.9 on page 96.

The second work chapter is chapter 4 on page 101, where I describe the work done on the detection of maintenance needs on railways. This chapter starts with section 4.3 on page 102 introducing the various datasets that were leveraged, including those gathered during this work. It is followed by section 4.4 on page 106 documenting the detection of maintenance issues on railways using clean images with ideal viewpoints and angles. I also explore performing anomaly detection on railways using uncertainty quantification in section 4.5 on page 116, showing that existing techniques do not work on the high diversity data gathered from UAVs which comprises many viewpoints, lighting conditions, and background settings when gathered in the low data regime (< 10,000 images). This motivates the development of a new algorithm termed Anomaly Detection by Reconstructing Labels (ADeReL), which I present in section 4.6 on page 129. ADeReL's performance is still less than optimal, which I conclude in section 4.7 on page 133 by providing insights to future research in this direction.

1.5 Published Work

Publications derived from this work are as follows:

- Tai, J.J., Innocente, M.S. and Mehmood, O., 2021, September. *FasteNet: a fast railway fastener detector*. In Proceedings of Sixth International Congress on Information and Communication Technology: ICICT 2021, London, Volume 1 (pp. 767-777). Singapore: Springer Singapore.

- Tai, J.J., Terry, J.K., Innocente, M.S., Brusey, J. and Horri, N., 2022. *Some supervision required: Incorporating oracle policies in reinforcement learning via epistemic uncertainty metrics*. arXiv preprint arXiv:2208.10533.
- Tai, J.J., Wong, J., Innocente, M., Horri, N., Brusey, J. and Phang, S.K., 2023. *PyFlyt–UAV Simulation Environments for Reinforcement Learning Research*. arXiv preprint arXiv:2304.01305.

2 Literature Review

The work in this thesis focuses primarily on concepts and ideas in railway maintenance and deep learning. This section serves to first introduce the reader to the various concepts used in these fields, setting the foundation for the subsequent chapters of this thesis. Then, the preliminaries and related works of the proceeding chapters are covered. These are topics revolving around UAV navigation and the detection of maintenance needs along railways.

2.1 Background

Artificial Intelligence (AI) is the capability of a computer system to mimic human cognitive functions such as learning and problem-solving. Machine Learning is a subfield of AI that focuses on developing algorithms and models that allow computers to learn from data and make predictions or decisions. The central idea behind machine learning is to build mathematical models that can learn patterns and relationships from data and generalize those learnings to unseen data. These techniques have found applications in a wide range of domains, including image recognition, natural language processing, speech recognition, and autonomous systems.

The use of AI has seen proliferation in many fields. One up and coming field for it is in maintenance. Maintenance is a crucial activitiy in industry, with significant impact on a company's ability to maintain low costs, high performance and quality. Unplanned downtime of equipment can potentially lead to significant financial penalties and reputation loss. In some cases, maintenance plays a highly non-trivial role in a company's core business and high emphasis is placed on the drive for well-implemented and efficient maintenance strategies. For instance, the operation and maintenance costs for offshore wind turbines account for 20% to 35% of total revenue from electricity generated [Gong and Qiao, 2014]. In the oil and gas industry, 15% to 70% of total production costs is typically attributed to maintenance expenditure [Bevilacqua and Braglia, 2000]. Given the growing amount of available industrial data and the unprecedented speed of developments in AI, the space of possible applications spans all permutations of architecture, maintenance type, and field of industry.

2.1.1 Railway Maintenance

Railway systems form the backbone of modern transportation networks, ensuring the efficient movement of goods and passengers. As these systems play a critical role in global connectivity, ensuring their reliability, safety, and sustainability becomes paramount. Railway maintenance emerges as a key aspect in achieving these objectives, encompassing a range of

activities aimed at preserving and enhancing the infrastructure's operational integrity. More formally, railway maintenance is defined as the upkeep and management of railway assets by identifying where and what requires recuperative work. Traditionally, maintenance practices relied on scheduled inspections and routine interventions, often resulting in inefficiencies and unforeseen disruptions. This is known as preventive maintenance. In recent years, advancements in sensor technologies, data analytics, and machine learning have ushered in a new era of proactive and data-driven maintenance strategies.

Network Rail, the owner and infrastructure manager of most of the railway network in Great Britain, spent £13.1 billion operating, maintaining, renewing, and enhancing the national rail infrastructure in the year ended 31 March 2022. Of this amount, approximately £1.9 billion was spent on maintenance alone [Office of Rail and Road, 2022]. While the exact amount spent on monitoring tasks is not published, we can make certain educated guesses at this amount by observing adjacent fields of mass infrastructure systems. Some industries, such as aviation, nuclear power, and oil and gas, may allocate a significant portion of their maintenance budget to inspection and monitoring due to the critical nature of their equipment and the potential consequences of failure, relying mostly on preventive and predictive maintenance techniques. The International Air Transport Association [2021] estimates that approximately 16.8% of all maintenance costs are associated with inspection and monitoring. On the other hand, industries with less critical equipment or those relying on reactive maintenance approaches might allocate a smaller percentage of their maintenance budget to inspection and monitoring, focusing more on corrective maintenance when issues arise. If we assume that the minimum amount required for maintenance monitoring costs is 1%, then a fair estimate for railway maintenance monitoring in the United Kingdom is anywhere from £19 million to £320 million. Being able to save even a small percentage with minimal investment and overhead is likely to produce a significant gain.

Current Maintenance Monitoring Techniques Knowing the rough costs associated with maintenance needs monitoring, various commonplace maintenance monitoring methods done in the United Kingdom are now introduced. Some of the most prevalent methods of finding impending maintenance issues is through Track Geometry Measurement Systems (TGMS), which measure parameters such as track alignment and curvature, helping identify potential issues with track geometry [Farkas, 2020]. In addition, Wheel Impact Load Detectors (WILD) monitor the impact loads on train wheels, helping detect anomalies and issues that may affect the track or rolling stock [Stratman et al., 2007]. This is used in conjunction with vibration analysis monitoring of rail components to detect abnormalities and predict potential failures. Internal defects in homogenous material components such as railways can be detected using non-destructive ultrasonic and eddy current testing by ground crew [Bombarda et al., 2021, Thomas et al., 2007]. Satellite imaging is also often used for providing a broader perspective for monitoring large sections of rail networks, especially for monitoring vegetation encrouchment [Chang et al., 2016, 2014]. This is lately facilitated by manually piloted UAVs [Rahman and Mammeri, 2021]. Strain gauges and accelerometers are placed on rail tracks or components to measure strain, stress, and vibrations, providing

insights into the structural health [Castillo-Mingorance et al., 2020]. Thermal sensors are used in strategic areas to detect temperature variations that may indicate problems with components like bearings or electrical systems [Falamarzi et al., 2019]. Finally, cameras are often mounted on moving railway vehicles for visual inspections of tracks, switches, and other infrastructure [Balouchi et al., 2021]. This is accompanied by physical inspection by ground crew on scheduled maintenance walks.

Maintenance Types Nomenclature and definitions tend to vary across literature, but the general consensus is that there are three degrees of maintenance proactiveness: Corrective Maintenance, Preventive Maintenance and Predictive Maintenance [Gackowiec, 2019].

- **Corrective Maintenance** refers to the idea of replacing broken components or performing maintenance only when something breaks. This form of maintenance requires defects due to broken equipment to be identified on the system. The retroactive term for this is simply repairs. This model of maintenance has several benefits, namely the reduced requirement to be on top of equipment condition. However, this form of maintenance can lead to severe equipment downtime, catastrophic failure, or even loss of life[1], especially when replacement equipment is not available or restorative action takes a considerable amount of time.

- **Preventive Maintenance** is the most common form of maintenance, wherein the equipment or component is replaced or restored well in advance of the actual equipment failures. The most common technique of preventive maintenance is to perform maintenance periodically, also known as periodic maintenance. This form of maintenance typically does not require any form of system monitoring. By doing preventive maintenance, system downtime from equipment failure is avoided, and instead replaced by the amount of time it takes to perform the replacement or restoration of the component. The downsides to preventive maintenance is simply the requirement to formulate schedules for the component. When extrapolated to entire systems, this can lead to elaborate schedules for all core components of a system.

- **Predictive Maintenance** attempts to forecast when a particular asset is going to fail, before replacing it right before the expected failure time. This maximizes the equipment Mean Time Before Failure (MTBF), which is the expected working lifespan of an equipment across various life cycles. The predominant method of performing Predictive Maintenance (PdM) is by having instrumenting with sensors to predict the state of health based on auxiliary measurements such as equipment temperature or overall system load. Thus, predictive maintenance typically requires extensive amounts of system monitoring in order to assess the structural health of each component. This form of maintenance aims to minimise total system downtime by lengthening the amount of time between maintenance cycles as a consequence of longer component working

[1] https://en.wikipedia.org/wiki/Granville_rail_disaster

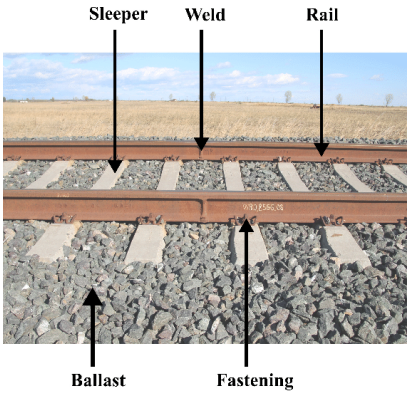

Figure 2.1: Visualization of various components essential to a railway track.

lifespan. The core barrier of entry to widespread adoption of PdM is the high upfront cost. Sensored equipment generally costs more than non-sensored equipment, and further require accurate models that can evaluate the state of health of a particular equipment based on sensor readings. In some cases, a large amount of sensor data must be processed live to evaluate equipment health. For instance, the Boeing 737 generates 20 TB of data per engine per hour of flight [Chen et al., 2016a]. To that end, fairly sophisticated data processing algorithms are required in order to make use of all the data to give an accurate analysis of engine health. Such an application requires substantial upfront cost, and has to be engineered in from the start of overall engine design. Such costs are only justified for systems at scale.

Railway Maintenance Requirements Various components of the railway track are illustrated in figure 2.1. Broadly speaking, maintenance needs of railway tracks can be classified into intrinsic and extrinsic needs. Intrinsic needs refers to maintenance requirements induced by railway track components, such as structural defects and track geometry defects [Sharma et al., 2018]. Structural defects refers to defects caused by the deterioration of rail track components. These include:

- track bed settlement leading to missing or insufficient ballast,
- broken or missing fasteners.
- cracked sleepers,
- rail head or foot deterioration/corrosion,
- rail fractures,
- tunnel defects — any defects in tunnels that trains travel in.

On the other hand, track geometry defects are defects that involve deviation of the railway geometry away from the intended geometric parameters. These include:

- rail misalignment, including gauge spread, rail twist, rail canter defects, and cross-level irregularities,
- void sleepers caused by missing or insufficient ballast.

Conversely, extrinsic needs refer to the kinds induced by external factors, such as leaf fall coverage and vegetation encroachment. It also includes unpredictable anomalies that occur which are outside the control of railway track design, such as vandalism or natural disasters.

One notes easily that, based on the defects listed, a majority of railway maintenance needs arise from defects that occur spontaneously, where components fail abruptly without clear warning signs (e.g. fasteners do not degrade gracefully, sleepers don't crack slowly). This is in contrast to the maintenance needs of, for example, aircraft engines [Khan et al., 2021], where components exhibit gradual degradation and provide indicators for maintenance timing (e.g. engine oil replacement depends on concentration of impurities in the liquid, air filter replacement depends on air flow rate efficiency, bearing replacement depends on axial play).

2.1.2 Deep Learning

Deep Learning is a subset of machine learning that focuses on using artificial neural networks to model and solve complex problems. The recipe of deep learning generally consists of gathering a dataset that aligns to the goal at hand, defining a suitable loss function, implementing an expressive function approximator such as a neural network with more than two layers, and then optimize the function approximator to minimize the loss function via gradient descent on the dataset. This paradigm allows for many problems to be reduced from finding the right algorithm for the task to finding the right network architecture, dataset and loss function. Through selection of appropriate network architectures, loss functions and datasets, various complex tasks such as natural language [Brown et al., 2020], image generation [Rombach et al., 2021] and speech processing [Radford et al., 2023] can be solved.

Supervised Learning Supervised Learning is a branch of machine learning where the algorithm learns from labeled examples. In this setting, a dataset consisting of inputs $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_n\}$ mapped to outputs $\boldsymbol{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ...\boldsymbol{y}_n\}$ is provided to the algorithm, and it learns to map inputs to corresponding outputs by minimizing a predefined loss function. There are many variants of loss function, each suited to a different task, some examples include mean squared error for continuous valued outputs $\mathcal{L} = ||\hat{\boldsymbol{y}} - \boldsymbol{y}||_2/\dim(\boldsymbol{y})$ or binary cross entropy for binary outputs $\mathcal{L} = ||\hat{\boldsymbol{y}} - \boldsymbol{y}||_2/\dim(\boldsymbol{y})$, where $\dim(\boldsymbol{y})$ represents the dimensionality of $\boldsymbol{y}$.

Unsupervised Learning Unsupervised Learning is another category of machine learning where the algorithm learns from unlabeled data. This means that only a dataset of

$\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_n\}$ is provided. In this case, the algorithm aims to discover hidden patterns, structures, or relationships within the data. One example of this is by tasking the model with reducing the dimensionality of the input $\boldsymbol{x}$ to form a new state $\boldsymbol{y}$, and then performing clustering techniques, whereby similar inputs in $\boldsymbol{X}$ are placed closed together in the $\boldsymbol{Y}$ space. An alternative form of unsupervised learning is input denoising, where a model f is trained to reverse a noising process on the input $f : \boldsymbol{x} + \epsilon \rightarrow \boldsymbol{x}$. Unsupervised learning is used extensively in the field of predictive maintenance, where a deep learning model is used to perform representation learning on data gathered from equipment under normal operating conditions. This representation is then used to inform of potential equipment failure or time-to-failure as a result of sensor readings outside of the normal range [Carvalho et al., 2019]. The benefit of unsupervised learning over supervised learning is there is no need for a labelled dataset during learning, making data gathering significantly easier.

Reinforcement Learning Reinforcement Learning (RL) is a form of learning in which a model, also known as an agent, is trained to map observations s_t of an environment into outputs a_t that enable it to take actions that influence the environment. In turn, the agent alters the state of the environment, causing it to observe a new state s_{t+1} and receive a reward r_t for its actions. Learning in reinforcement learning is underpinned by the objective of maximizing the cumulative rewards over time, thereby enabling the agent to learn from its experiences by understanding the consequences of its actions through a trial-and-error approach. This method facilitates the development of sophisticated decision-making strategies through user-defined reward systems. Given the extensive use of RL in this work, this field of deep learning is elaborated in greater detail in section 2.3 on page 23.

2.1.3 Deep Learning Architectures

The design of neural networks can be likened to building LEGO®; there are various foundational input-output blocks that may function for limited tasks standalone, but the *magic* of neural network architectures is by clever composition of various neural network building blocks into a single model. Here, we introduce several of the most important blocks serving as a waypoint for architectures that will be used in this work.

Multilayer Perceptron The Multilayer Perceptron (MLP) is a fundamental neural network architecture. It's basic unit starts with a linear layer followed by a non-linear activation function (a.k.a perceptron) $f_1(\boldsymbol{x}) = g(\boldsymbol{A}\boldsymbol{x} + b)$ where the input is $\boldsymbol{x} \in \mathbb{R}^{n_0}$, $\boldsymbol{A} \in \mathbb{R}^{n_1 \times n_0}$ and $b \in \mathbb{R}$ are learnable parameters, and g is an element wise non-linear function. Some options for g include the Rectified Linear Unit (ReLU) or tanh function. A full MLP is simply multiple layers of f, e.g. $f_n(...f_3(f_2(f_1(\boldsymbol{x}))))$. Hornik et al. [1989] has shown very early on that an infinitely stacked MLP is essentially a universal function approximator. The simplified explanation of a neural network is a function for approximating highly complex nonlinear functions.

Convolutional Neural Network The Convolutional Neural Network CNN is a class of networks designed to mimic the visual processing in the human brain [Fukushima, 1980]. They function by convolving a series of n filter kernels $\boldsymbol{h} \in \mathbb{R}^{c \times m_1 \times m_2}$ over the input image $\boldsymbol{x} \in \mathbb{R}^{c \times h \times w}$ to produce a series of feature maps $\hat{\boldsymbol{y}} \in \mathbb{R}^{n \times (h-m_1-1) \times (w-m_2-1)}$ denoting how strongly a local patch of the input matches with the pattern in the kernels. This operation allows networks to capture spatially invariant local patterns in the input data. A bias is normally then added to the output feature map before the result is passed through an element-wise non-linear function, similar to those used in MLPs. By itself, a single convolutional layer cannot accomplish much beyond matching patterns in the image to those represented in the kernels. However, naively stacking several layers of convolutional layers on top of each other results in a model with enough representational power to beat humans at image recognition [Krizhevsky et al., 2012]. This works by allowing the network to capture different image patterns and features at various scales, enabling the network to automatically extract relevant information from the input images.

Various augmentations on top of the vanilla CNN architecture are possible, the most basic is to implement any sort of pooling operation in between the layers to reduce the computational cost of each subsequent layer. Other augmentations include using residual connections, where the input of a layer is added to its output before being passed on to the next layer [He et al., 2016]. The de facto pure convolutional architecture for computer vision tasks is now the U-Net [Ronneberger et al., 2015], first designed for biomedical imaging where the residual connections visualized as a block diagram resembles a U pattern, where the last layer incorporates information from the first layer, the second last layer from the second layer, the third last layer from the third layer, and so on.

Recurrent Networks Recurrent Neural Networks (RNNs) are a type of neural network architecture specifically designed to handle sequential or time-dependent data. The base concept of the RNN is for a portion of the output, known as the hidden state $\boldsymbol{h}_*$, to be fed back into the input at the next time step, $[\boldsymbol{y}, \boldsymbol{h_1}] = f(\boldsymbol{x_0}, \boldsymbol{h_0})$. This hidden state acts as a memory, enabling the network to maintain information about the past and use it to make predictions or decisions in the current time step.

RNNs are optimized using Back Propagation Through Time (BPTT). This introduces the vanishing and exploding gradients problem, where for long sequences, the gradients can diminish or explode as they backpropagate through time, leading to difficulties in learning long-term dependencies. To address this issue, more advanced variants of RNNs, such as the Long Short Term Memory (LSTM) and Gated Recurrent Units (GRUs), have been introduced.

While these models addressed the learning instability, two major drawbacks of recurrent models are the fixed memory size and autoregressive nature. Recurrent networks only have a memory size as large as the dimensionality of the hidden state $\boldsymbol{h}$. This is an issue for long sequences because it forces the model to forget potentially important information from the beginning of the sequence. In addition, when presented with a sequence of inputs to process, recurrent models must process one token of the input sequence at a time, limiting processing

speed. This is in contrast to traditional deep learning models, where the entirety of the input is processed at once in parallel.

Self Attention Mechanism The self attention mechanism Vaswani et al. [2017], also known as the Transformer architecture, is arguably the most researched model at the time of writing. It introduces a mechanism that allows the model to focus on different parts of the input sequence while processing it. The gist of the self attention is that an n-long input sequence $\boldsymbol{x} \in \mathbb{R}^{n \times m}$ is converted into a query, key, and value $\boldsymbol{v}$ matrix $\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v} \in \mathbb{R}^{n \times p}$. Then, a weighted attention operation is done to produce a new output sequence $\hat{\boldsymbol{y}} = g(\boldsymbol{q} \times \boldsymbol{k}^\intercal / \sqrt{p}) \times \boldsymbol{v}$, where g is a softmax in the dimension of p and $\hat{\boldsymbol{y}} \in \mathbb{R}^{n \times p}$. It can easily be seen that the output $\hat{\boldsymbol{y}}$ is formulated by allowing every part of the input sequence to attend to every other part via the internal matrix multiplication $\boldsymbol{q} \times \boldsymbol{k}^\intercal$. The output sequence can then be transformed in any differentiable manner suitable for the task, such as simply taking the average if a single vector is required. Unlike traditional recurrent approaches, the self-attention mechanism can capture global dependencies in the input sequence in a single parallel operation, making it more efficient for long-range dependencies. The downside to self attention is the high memory complexity, which scales quadratically to the size of the input. This contrasts with recurrent models where memory complexity scales linearly with input size during training and is constant during inference.

2.2 UAV Autonomous Navigation

UAVs are predominantly recognized as flying robots which are capable of carrying a plethora of sensors. UAV navigation is seen as a process in which these flying robots take actions, possibly according to a plan, that allows it to safely achieve an intended goal. This goal may be to reach a specified target location, or in the case of autonomous remote sensing [Pajares, 2015], reach a series of locations in order to achieve some underlying goal such as total area coverage by a camera.

2.2.1 Sensors for UAV Navigation

UAVs rely on a variety of sensors for navigation by maintaining comprehensive awareness of their states. These states include crucial factors such as location, navigation speed, heading direction, observable obstacles, starting point, and target location. The integration of accurate and reliable sensor data is imperative for efficient route planning and safe maneuvering within the dynamic and unpredictable environment that UAVs often operate in.

One category of sensors mandatory UAV operation is inertial sensors. Inertial sensors are ego-mounted sensors that allow the UAV to perceive changes in velocity and orientation. These sensors are commonly referred to under the unified term of Inertial Measurement Unit (IMU), and typically consists of an accelerometer and gyroscopes to measure vehicle acceleration and angular velocity, allowing for the estimation of position over time [Phang et al., 2014, 2012]. However, inertial sensors are susceptible to drift errors, accumulating inaccuracies over extended durations. To address this limitation, research has focused on sensor fusion techniques, combining inertial data with other sensor modalities to enhance accuracy and mitigate drift [Li et al., 2017].

Satellite-based navigation, particularly Global Navigation Satellite Systems (GNSS) like GPS, is another pivotal component of UAV navigation systems. GNSS provides global coverage and reasonable accuracy in open-sky environments ($\tilde{3}$0 cm), enabling precise position fixes. However, GNSS signals may be compromised or completely unavailable in urban canyons, dense foliage, or indoor environments [Cui et al., 2015, Wang et al., 2014]. As such, supplementary navigation methods become essential in scenarios where satellite signals are unreliable.

Vision-based navigation via onboard cameras represents a promising avenue for UAV navigation, leveraging cameras and computer vision algorithms to interpret the surroundings. This approach enables real-time obstacle detection, terrain mapping, and localization through visual landmarks [Zhang et al., 2018, Chen et al., 2016b, Phang et al., 2010]. Despite its prevalence, vision-based systems may face challenges in adverse weather conditions, low-light environments, or scenarios with limited visual cues.

Other commonly used sensors in UAVs include Light Distance and Ranging (LiDaR) and Radio-Frequency Distance and Ranging (RaDaR) systems. LiDAR, which stands for Light Detection and Ranging, uses laser light to measure distances and create detailed, high-resolution maps of the environment. LiDaR is particularly effective for terrain mapping,

obstacle avoidance, and accurate altitude control, making it valuable for both navigation and surveying applications in various environmental conditions [Lin et al., 2019b, Wallace et al., 2012].

RaDaR, or Radio Detection and Ranging, is another sensor commonly employed in UAVs. RaDaR systems use radio waves to detect the presence and location of objects in the environment. They are useful in scenarios where visual or laser-based sensors may face limitations, such as in adverse weather conditions or situations with dust or smoke [Mullen and Contarino, 2000, Abushakra et al., 2021].

Complementary to these sensors, barometric altimeters are often integrated into UAVs to provide altitude information based on atmospheric pressure. While not as precise as other altitude sensors, barometric altimeters offer a supplementary means of altitude estimation, especially in scenarios where GPS or LiDaR may face challenges.

In practice, UAV navigation systems typically utilize a combinations of sensors to enhance accuracy, reliability, and robustness in diverse operational environments in a process known as sensor fusion. The selection and integration of these sensors depend on the specific requirements and challenges of the UAV mission, ensuring adaptability and effectiveness across various scenarios. For instance, the barometric sensor is often paired with the onboard IMU to gain a better estimate of the UAV's angular state. In other cases, laser range finders can be used with monocular vision cameras to produce a pseudo-depth map [Wang et al., 2013].

2.2.2 Path Planning vs. Obstacle Avoidance and Navigation

UAV navigation involves robots planning a safe and efficient route to a target location based on the current environment. This field can be broadly classified into several fronts:

- **Path Planning** involves the determination of an optimal or feasible route for a robot or vehicle to reach its destination, considering factors such as terrain, distance, and potential obstacles. It focuses on finding the most efficient way to navigate from point A to point B.

- **Obstacle Avoidance and Navigation** deals with real-time decision-making during the execution of the planned path. This involves dynamically adapting to unforeseen obstacles or changing environmental conditions.

Both are two distinct yet interconnected concepts in the field of robotics and autonomous systems. While path planning sets the route, obstacle avoidance and navigation enable the system to respond to dynamic challenges in real-time, ensuring safe and effective movement by adjusting the trajectory based on sensor input and environmental changes. Together, these concepts play a crucial role in enabling autonomous systems to navigate complex and dynamic environments. In the context of this thesis, we focus on the task of obstacle avoidance and navigation. The premise of this decision is that, for well defined systems such as railways, there is minimal requirement for path planning to occur as railway corridors are very well defined.

2.2.3 Heuristic Techniques for UAV Autonomous Navigation

The paradigm of UAV navigation can be extended to the overarching realm of robotics under mild assumptions, and autonomous navigation has been a long standing field in robotics research. These assumptions include similar environment principles of observability with comparable sensor regimes, physical environment predictability, limited computational resource, and three dimensional mobility characteristics. Prior to deep learning, autonomous navigation has been predominantly handled using heuristic methods. Heuristic methods in autonomous navigation refer to algorithms that rely on predefined rules and deterministic operations rather than learning from data. These methods have been historically preferred due to their simplicity, transparency, and ease of implementation. They allow for straightforward debugging and understanding of the robot's behavior since the decision-making process is explicitly coded. However, heuristic methods may struggle with complex, dynamic environments where handcrafted rules cannot anticipate every possible scenario. Their deterministic nature also makes them less adaptable to unforeseen changes compared to learning-based approaches. The work on heuristic methods for obstacle avoidance and navigation is massive and has a long standing history in the field of robotics. While many of the concepts in heuristic navigation have no parallels in deep learning techniques, we introduce them here for completeness.

Map-Based Navigation utilizes detailed maps of the environment to guide the UAV. Typically, this is done using either occupancy grids [Meyer-Delius et al., 2012] or octree maps [Meagher, 1982]. Under mild assumptions, octree maps are simply efficient methods of encoding occupancy grids, and there exists many open source tools that allow conversion from one to the other [2] [3]. The predominant method of obtaining these maps is to utilize 3D volumetric sensing to efficiently map and explore the environments while making a best guess estimate action towards the navigational goal. One method in this category include the usage of stereo vision cameras [Li and Ruichek, 2014, Yu et al., 2015, Nguyen et al., 2011], where depth information is additionally gathered using feature location disparity between both cameras. This depth information, in addition to the sensed position and orientation estimate on the UAV, allows for the estimation of the location of a feature in 3D space. However, this technique can suffer from occlusions and limited accuracy in poorly textured environments. Additionally, stereo vision systems can be computationally intensive, which may limit their applicability on resource-constrained platforms like UAVs. Another approach involves the use of LIDAR sensors which emit laser pulses and measure the time it takes for the pulses to return after hitting an object[Luo et al., 2019, Sabatini et al., 2018]. LIDAR systems can provide highly accurate distance measurements and are effective in various lighting conditions. However, they tend to be bulkier and more expensive than stereo vision systems. A third method is the integration of both stereo vision and LIDAR, leveraging the strengths of each to compensate for their individual weaknesses [Moghadam et al., 2008, Oh and Kang,

[2] nomis80.org/code/octree.html

[3] flipcode.com/archives/Octree_Implementation.shtml

2016, Li, 2013]. By combining these sensors, it is possible to achieve more robust and accurate 3D mapping and navigation capabilities. For instance, LIDAR can be used to provide precise distance measurements, while stereo vision can enhance the resolution of the environment's visual features. Once a detailed 3D map is obtained, a Simulataneous Localization and Mapping (SLAM) algorithm — a probabilistic algorithm for making best guess estimates of the robot's location and surrounding geometry — can then be used in conjunction with global heuristic path planning algorithms such as A-star search [Hart et al., 1968] can to complete the navigational task.

Geometric Methods Geometric methods work by measuring distances between the UAV and other obstacles. These techniques are also commonly termed as vector-field methods. Ha et al. [2019] proposed a method of guiding a UAV from a starting point to an ending point by segregating the flight domain into a series of high risk spheres around each obstacle. Then, tangent lines between the UAV to the sphere is used to formulate an optimal flight path between each obstacle from the start to the end. A tracking control law is then implemented onboard the UAV to ensure that the UAV stays as close to this trajectory as possible. Upon detection of a new obstacle and impending collision, the trajectory is recomputed on the fly is used as the new trajectory to be followed. Lin and Peng [2021] first plan a trajectory from the starting to ending points using path planning algorithm, before using an optical flow camera to detect obstacles and generate depth cues for a lightweight obstacle avoidance algorithm. Their method relies on using optical flow images to determine closeness of obstacles, and then relying on the density of optical flow on on a read image to take evasive actions. Zheng et al. [2019] perform a similar task, but instead opting to rely on a 2D LiDaR to measure distances to obstacles. Holmberg et al. [2022] use a slightly more advanced concept by further incorporating SLAM into the algorithm to have premonition of all obstacles present. The main trend in geometric methods is to represent obstacles in the space as a set of vectors from the UAV's position, and then utilize simple geometric rules to generate a suitable trajectory from the start to the end.

Force Field Methods Force field methods rely on an idea inspired by attractive or repulsive electric forces, also known in other circles as potential field methods. Classical force field methods are limited to simple trajectory planning, and do not have strict guarantees for obstacle avoidance. Sun et al. [2017] propose to perform post-processing to optimize a basic potential field algorithm with a distance factor and a jump strategy to address this shortcoming of classical force field methods. Van Der Veeken et al. [2021] experiment with a potential field that takes inspiration from magnetic attraction and repulsion, wherein the potential field strength scales to the square of distance to an obstacle. Cong et al. [2021] uses a normal potential field, but places narrow channels between obstacles which serve as strict fly-zones. Their simulations show that the method can perform path planning effectively, but cannot solve an issue of the UAV getting into a local optimum. The common trend of these works is that the potential field is computationally efficient and can be solved in real time. All it requires is an artificially set force field function. Some other examples of force field meth-

ods include those by Mac et al. [2016], Ma'Arif et al. [2021], Rostami et al. [2019], Bui et al. [2019] and many more. More often than not, force field methods tend to be merged together with geometric methods to leverage the strengths of both techniques. Tai et al. [2022] demonstrate one such example where a vector field is used to compute smooth flight trajectories to avoid obstacles, while a potential field is layered on top the guide the UAV away from obstacles when they are unable to closely follow the planned vector field trajectory.

Optimization Methods Optimization methods aim at finding optimal or near optimal solutions for path planning and motion characteristics of the UAV while incorporating information about all other known objects. They typically rely on the calculation of an avoidance trajectory based on geographical information. They also encompass probabilistic search algorithms which aim to provide flights through the best areas based on available uncertainty information. The Rapid Exploration Random Tree (RRT) algorithm is an algorithm proposed by LaValle et al. [2001]. It relies on a rapidly expanding tree of nodes, where at each point, the direction of branch is selected based on various heuristics such as obstacle closeness, vehicle motion constraints, and uncertainty. This has seen massive prevalence in modern robotics, and its use in UAVs is notwitheld. Meng et al. [2017] propose using an improved version of this algorithm to solve obstacle avoidance in a dynamic environment. Zekui et al. [2018] perform a similar task using additional constraints such as corner and path length constraints to restrict the turn angle and path length, before fitting a Bezier curve to the resulting path to generate a suitable flight trajectory. To accomadate for the notion of higher risk obstacles, Yafei et al. [2020] propose using an artificial potential field as a basis for risk factor calculation within the RRT algorithm. Their results show that this allows the UAV to avoid threat areas quickly and effectively. Chen and Yu [2021] further incorporate compute cost into the RRT algorithm in a manner similar to A* to further speed up convergence of the algorithm, and then applied this to a UAV platform to perform obstacle avoidance. LI HY et al. [2021] utilize a forest of RRTs, each computed by a different UAV in a swarm to further speed up the RRT computation. To handle dynamic obstacles, Chen and Wang [2022] propose a pruning and reconnection mechanism for segments where a previously moved obstacle has moved to regenerate paths without recomputation of the entire tree. Fu et al. [2022] add a bidirectional objective heuristic to the RRT objective to handle blind search, long paths, and zigzag paths in a 3D spatial environment.

Another form of optimization method include Bayesian optimization, particle swarm optimization, and greedy methods. Inspired by the ant colony algorithm, Pérez-Carabaza et al. [2019] utilize a minimum time search algorithm to perform obstacle avoidance within a multi-UAV scenario to ensure successful collision avoidance in communication constrained environments. Polvara et al. [2018] discusses collision detection and path planning methods by considering global and local path planners. They perform a comparative analysis between the most common techniques from graph search theory to evolutionary algorithms. Boivin et al. [2008] formulates optimal trajectories by taking inspiration from model predictive control to solve a cost function involving the UAV's current position and the position of all obstacles. Biswas et al. [2019] utilize a particle swarm algorithm (PSO) to assign weights

to different regions of an environment using all available sensor information. Then, an optimal path is chosen through territories of least risk. Jalal [2015] modified the conventional PSO algorithm for offline UAV navigation by incorporating an additional error factor to convert infeasible paths to feasible paths. The infeasible paths in this case are caused by motion constraints of the UAV.

Common Basis Map-based navigation is advantageous in structured environments where detailed maps are available, but it struggles in dynamic settings where obstacles can change. The reliance on accurate maps can be a significant limitation. Geometric methods excel in real-time adaptability and computational efficiency, making them suitable for dynamic environments. However, their effectiveness diminishes in highly complex or unstructured settings, and they depend heavily on accurate sensor data. Force field Methods offer simplicity and computational efficiency, making them ideal for real-time applications. Yet, their susceptibility to local minima and limited guarantees for obstacle avoidance pose challenges in complex scenarios. Combining force field methods with geometric methods can leverage the strengths of both techniques, as demonstrated by Tai et al. [2022]. Optimization methods provide robust and flexible solutions capable of handling complex and dynamic environments. However, they are computationally intensive and may require significant customization. Techniques like RRT and PSO offer promising results but can be slow to converge without careful tuning.

The recurring theme in navigation tasks is to map a robot's sensor data $\boldsymbol{s} \in \boldsymbol{\mathcal{S}}$ to some lower dimensional representation (octrees/occupancy grid/graph/tree) $\boldsymbol{z} \in \boldsymbol{\mathcal{Z}}$. This representation is the most complex for map-based navigation, requiring ideally the full map. This requirement is further relaxed for optimization methods, which can make use of any currently available information to formulate a best guess estimate of an optimal flight path using probabilistic approaches to the problem. Finally, geometric methods and force field methods utilize the smallest representation of currently known information — oftentimes only requiring knowledge of the k-nearest obstacles to the UAV and disregarding all other information. A more detailed comparison of these methods is shown in table 2.1.

Given this representation of the environment, the task of navigation is phrased as a problem of finding the optimal sequence of actions $\{\boldsymbol{a_1}, \boldsymbol{a_2}, ..., \boldsymbol{a_n}\}$ to take given some utility function $\mathcal{U}$ between a target configuration $g(\boldsymbol{z}_T)$ and the current configuration $g(\boldsymbol{z})$. In short, a navigation task aims to solve the following:

$$\{\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_n\} = \underset{\{\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_n\}}{\arg\max} \; -\mathcal{U}(g(\boldsymbol{z}), g(\boldsymbol{z}_T)) \tag{2.1}$$

Depending on context, the arg max can also be expressed as an arg min by flipping the sign of $\mathcal{U}$. In the case of geometric, force-field, and map-based methods, a best guess estimate of an action function $\mathbf{a}_t = f(\mathbf{z}_t)$ is assumed before hand, and directly applied to the representation as a programmed algorithm. This action function is simple for potential field methods; most complex for map-based navigation which typically utilize a full path-planning algorithm; and

somewhere in between for vector field methods. By contrast, optimization based methods do not utilize any predetermined action function, and instead directly solve for the action sequence $\{\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_n\}$ at runtime, allowing the algorithm designer to simply choose the choice of utility function $\mathcal{U}$ instead. This live computation of action sequence at every step contributes to the high computational complexity of optimization methods.

In learned methods, the action function $f_{\boldsymbol{\theta}}(\boldsymbol{z}_t)$ is typically represented as a learnable function parameterized by a set of weights $\boldsymbol{\theta}$. Throughout the course of learning, $\boldsymbol{\theta}$ is then updated to optimize $\mathcal{U}$, therefore learning an action function before deployment that serves as a best estimate at optimizing $\mathcal{U}$. This leads to very low computational complexity during deployment, while approximating the optimality of optimization methods. Furthermore, in many cases of learned methods, the mapping of robot sensor data to low dimensional representation is also represented as a learnable function. We term that function in this context as the representation function. Both representation function and action function can then be jointly learned end-to-end, allowing the algorithm to incorporate only necessary information into the low dimensional representation $\boldsymbol{z}$. This end-to-end nature results in the ability to phrase the problem as a single optimization task that can be solved prior to deployment. This reduction of a complex task into a single optimization objective is the core argument for learned methods.

Table 2.1: Comparative Analysis of UAV Navigation Methods

Method	Description	Pros	Cons	Examples and References
Map-Based Navigation	Applies a path finding algorithm to predefined environment layouts like occupancy grids or octree maps measured using sensors capable of depth sensing.	High interpretability; leverages existing maps; use of well known mapping tools and algorithms.	Requires accurate and updated maps; limited adaptability to dynamic environments; computationally intensive for large environments.	[Meyer-Delius et al., 2012, Meagher, 1982]
Geometric Methods	Measures distances between UAV and obstacles, uses geometric principles to compute optimal flight paths and trajectories dynamically adjusting based on sensor input.	Real-time adaptability to detected obstacles; good computational efficiency.	May struggle with dynamic obstacles; relies on continuous sensor input for distance measurements.	[Ha et al., 2019, Lin and Peng, 2021, Zheng et al., 2019, Holmberg et al., 2022]
Force Field Methods	Similar to geometric methods, but use potential fields to navigate.	Computationally efficient; easily integrated with other methods.	Susceptible to local minima, leading to potential navigation failures; limited guarantees for successful obstacle avoidance in complex scenarios.	[Sun et al., 2017, Van Der Veeken et al., 2021, Cong et al., 2021, Mac et al., 2016, Ma'Arif et al., 2021, Rostami et al., 2019, Bui et al., 2019]
Optimization Methods	Find optimal or near-optimal paths using algorithms like RRT, Bayesian optimization or PSO.	Provides optimal solutions; can handle complex environments; may require significant tuning and customization.	High computational cost; may be slow in real-time applications.	[LaValle et al., 2001, Meng et al., 2017, Zekui et al., 2018, Yafei et al., 2020, Chen and Yu, 2021, LI HY et al., 2021, Chen and Wang, 2022, Fu et al., 2022, Pérez-Carabaza et al., 2019, Polvara et al., 2018, Boivin et al., 2008, Biswas et al., 2019, Jalal, 2015]

2.2.4 Applications for UAV Navigation

Various works have involved the use of UAVs for flight over or around an entity. These entities can range from long distance pipelines to manned aircraft. Some advancements made in those fields are highlighted here.

Pipeline Tracking Shukla et al. [2016a,b], Xiaoqian et al. [2016] utilize a series of image processing tools including Gaussian filters, intensity gradients, non-maximal suppression, normal threshold, and hysteresis thresholding on the result of performing a Canny edge detection on the image. The result is then used to identify pipes, which are then used to decide the orientation of the UAV relative to the pipe. Gómez Eguíluz et al. [2020] utilize a LiDaR mounted on a UAV to formulate a point cloud of its surroundings. After that, the RANSAC algorithm is utilized to recursively identify cylinders in the environment. The presence of cylinders are then used as an indicator of pipe location. This approach is similarly done by Guerra et al. [2018]. However, in contrast to Gómez Eguíluz et al. [2020] which only utilize a LiDaR, Guerra et al. [2018] adopt a probabilistic approach to determining the location of the pipe. First, the RANSAC algorithm is used in conjunction with the LiDaR point cloud to form a priori estimate to the location of the pipe. Then, a computer vision algorithm utilizing Canny edge detection is used to develop a much more accurate posterior estimate. They demonstrate that this works and show that they reduce spurious detection rates of a pipe to below 1%. da Silva et al. [2022] presented a piece of work which utilizes the YOLO architecture to detect pipes. The pipe is then cropped out of the image, before a Canny edge detection algorithm is utilized to identify the orientation and position of the pipe within the cropped image. Finally, the detected lines of the pipe are used to inform a downstream flight control algorithm, where the goal is to keep the line in the middle of the image, signifying that the UAV is flying directly over the pipe. They demonstrate that their framework can keep the UAV to within 15 cm of the pipe when flying overhead. Roos-Hoefgeest et al. [2023] utilize a semantic segmentation pipeline to segment parts of the image corresponding to pipes. Sensor measurements from a LiDaR are then used to filter this masked image to remove pipes that are far away. A series of filters are run over the resulting image, extracting the centerline of the pipe for the UAV to follow. To detect maintenance issues, they perform k-means clustering on patches of pixels to identify patches of the pipe that have a different colour to the un-corroded pipes. They further demonstrate that this technique works reasonably well in real world scenarios.

Tunnel Navigation Tunnel navigation involves the use of UAVs flying autonomously within tunnels or tunnel-like structures. Elmokadem and Savkin [2022] utilizes a depth sensing camera and SLAM to formulate a 3D point cloud of the interior of the tunnel, then utilize a kinematic model and vector operations to formulate a feasible flight path. A sliding mode controller is used to allow the UAV to follow this flight path. This work builds on a prior algorithm by the same authors which developed an algorithm for following tunnels based on 3D point cloud information [Elmokadem, 2020]. Yan et al. [2020] utilize a system where mul-

tiple algorithms are stacked on top of each other for a highly configurable and interpretable stack. They first utilize a Hector-SLAM algorithm in conjunction with a LiDaR to construct a 3D point cloud map of the environment. This is followed by a variant of RRT to perform route planning, before this route is optimized using a cubic B-spline to ensure smooth trajectories. Then, an classical PID controller is utilized to follow the B-spline. To avoid obstacles dynamically during runtime, an artificial potential field is layered over the whole architecture at runtime based on new information gathered by the LiDaR system. They verify their system on a physical UAV and show that this form of realtime path planning works reasonably well while ensuring a mostly optimal route. An interesting implementation of tunnel following is done by Ge et al. [2021]. Instead of building a 3D map from scratch, they adopt a technique developed by line-following robots, where the UAV is meant to follow a taped line on the floor of a tunnel to traverse the tunnel. While not scalable to unseen tunnels, this highlights one potential method in which UAVs may be used to follow railway tracks, since they possess a very well defined line that can be easily observed. In dark tunnels, Mansouri et al. [2019] utilize a computer vision technique of darkness contour detection to guide a UAV. In their work, the UAV possesses a high powered light source and a camera. This light source is used to illuminate the area immediately ahead of the UAV. Darker areas of the tunnel correspond to sections where the light does not bounce off the wall, indicating an opening. The goal of the UAV is to then continually fly towards the center of this dark opening, thereby avoiding the walls and traversing the tunnel itself.

Railway Tracks The use of UAVs for tracing railway tracks is less seen in literature, however, they are still present. Xia [2023] propose using a vanishing point detection to guide the UAV for track following in one instance, and utilize a preplanned flight path in another. Guclu et al. [2021] use a similar approach where the left and right lanes of the railway track are detected using two different instances of Gabor filters. These lines lead up to a vanishing point which is detected, and then pursued using a PID controller.

Notable Fields An adjacent field to UAV's tracking and traversing entities over long distances is in autonomous automotive vehicles. This has been a very hotly researched topic in recent times, and their impact on the future of transportation cannot be understated. Several techniques and algorithms have been developed to enable vehicles to autonomously follow lanes and navigate through diverse road conditions. Pre-deep learning, the use of LiDaR, RaDaR and ultrasonic sensors was the predominant method in which autonomous driving was implemented. Nowadays, a combination of vision-based sensors, LiDaR and RaDaR sensors are used in conjunction with deep learning to formulate better-than-human level driving in complicated urban environments.

2.3 Reinforcement Learning

Reinforcement Learning (RL) stands out as a paradigm that mimics the process of learning through interaction with an environment. Unlike traditional machine learning approaches that rely on a predefined dataset, reinforcement learning is characterized by an agent that learns to make decisions by performing actions and receiving feedback in the form of rewards or penalties. This section delves into the concepts, algorithms, and challenges of reinforcement learning, serving as a precursor for applying deep learning to UAV navigation. The notation here has been borrowed from the ubiquitous book by Sutton and Barto [2018].

2.3.1 Markov Decision Process (MDP)

From a machine learning perspective, the navigation problem can be phrased as a Markov Decision Process (MDP), which is a mathematical object describing the interaction of an agent within a stochastic environment. An MDP has the following components:

- $\boldsymbol{\mathcal{S}}$: state space, the space from which environment states are derived from.
- $\boldsymbol{\mathcal{A}}$: action space, the space of possible actions the agent can take.
- $r \in \mathbb{R}$: the reward an agent receives at every state $\boldsymbol{s}_t$.
- ρ: state transition distribution of $P(r_t, \boldsymbol{s}_t | \boldsymbol{s}_{t+1}, \boldsymbol{a}_t)$, where, given an action $\boldsymbol{a}_t$ in a state $\boldsymbol{s}_t$ at timestep t, the transition probability distribution P defines the distribution over possible next states $\boldsymbol{s}_{t+1}$ and scalar rewards r_t.
- $\gamma \in [0, 1]$: the discounting factor, larger values motivate the agent to emphasize long term rewards.

> The state transition tuple $\{\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}\}$ may be written as $\{\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}'\}$ for more concise notation.

An agent which interacts with the MDP is commonly written as $\pi(\boldsymbol{a}_t | \boldsymbol{s}_t)$. The canonical definition of solving an MDP is to find the *optimal* policy $\pi^*(\boldsymbol{a}_t | \boldsymbol{s}_t)$ which optimizes an objective over the expected sequence of rewards induced by the state transition distribution ρ under all states $s \in \boldsymbol{\mathcal{S}}$:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\rho,\pi}[f(r_1, r_2, ..., r_n)] \tag{2.2}$$

Partially Observable Markov Decision Processes (POMDPs) assume the agent only has access to an observation which is derived from the current state, $\boldsymbol{o}_t \sim P(\boldsymbol{o}_t | \boldsymbol{s}_t)$. In the case of a sensored robot, the sensor observations are derived from the true state of the robot in the environment.

Heuristic methods aim to map the sensor observations and actions $\{\boldsymbol{o}_1, \boldsymbol{a}_1, \boldsymbol{o}_2, \boldsymbol{a}_2, ...\boldsymbol{o}_t, \boldsymbol{a}_t\}$ into a state representation (octrees / occupancy grid / graph / tree) $\boldsymbol{z}_t \in \boldsymbol{\mathcal{Z}}$. If we treat $\boldsymbol{z}_t$ as the system state, *the POMDP can then be written as a MDP*. The task of navigation can then be phrased as a two step process of designing a suitable mapping $f : \{\boldsymbol{o}_1, \boldsymbol{a}_1, \boldsymbol{o}_2, \boldsymbol{a}_2, ...\boldsymbol{o}_t, \boldsymbol{a}_t\} \rightarrow \boldsymbol{z}_t$, then applying a heuristic algorithm to solve the equivalent MDP based on using $\boldsymbol{z}_t$"ch as the system state.

2.3.2 Reinforcement Learning Variants

Reinforcement lerning (RL) is a zeroth order, black box optimization method for solving MDPs. This section is primarily concerned with the cumulative discounted future reward setting of RL, wherein we define state-value function - the cumulative discounted sum of rewards of being in a state $\boldsymbol{s}$ and performing according to π as:

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\rho,\pi} \sum_{t=0}^{\infty} [\gamma^t r_t | \mathbf{S}_0 = \boldsymbol{s}] \tag{2.3}$$

γ here is known as a discount factor, which controls much importance is placed on long horizon rewards, and typically has values around 0.99. The goal of RL is to find an optimal policy π^* which produces the optimal state-value function $V^{\pi^*}(\boldsymbol{s}) = V^*(\boldsymbol{s}) = \max_{\pi} V(\boldsymbol{s})$. We categorize RL algorithms into three overarching classes - value-based RL, policy-based RL, and model-based RL.

Value-Based Reinforcement Learning Value-based RL relies on the concept of an action-value function - the cumulative discounted sum of rewards from being in state $\boldsymbol{s}$ and taking action $\boldsymbol{a}$ and then performing according to π. This quantity is commonly referred to as the Q-value, defined as:

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\rho}[r|\boldsymbol{s}, \boldsymbol{a}] + \gamma \mathbb{E}_{\rho,\pi}[Q(\boldsymbol{s}', \boldsymbol{a}')] \tag{2.4}$$

The state-value function can also be written as the expectation over actions of the Q-value without loss of generality:

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{a}\sim\pi}[Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})] \tag{2.5}$$

$$V^*(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{a}\sim\pi^*}[Q^{\pi^*}(\boldsymbol{s}, \boldsymbol{a})] \tag{2.6}$$

$$= \max_{\pi}[Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})] \tag{2.7}$$

Thus, the Q-value can also be written as:

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\rho}[r|\boldsymbol{s}, \boldsymbol{a}] + \gamma \mathbb{E}_{\boldsymbol{s}'\sim\rho} V^{\pi}(\boldsymbol{s}') \tag{2.8}$$

Given the Q-value of a policy π^n, we can derive a policy π^{n+1} that performs better than the base policy π^n by simply taking actions according to a greedy policy:

$$\pi^{n+1}(\boldsymbol{a}|\boldsymbol{s}) = \begin{cases} 1, & \text{if } \boldsymbol{a} = \arg\max_{\boldsymbol{a}} Q^{\pi^n}(\boldsymbol{s}, \boldsymbol{a}) \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

Typically, an ϵ-greedy policy is used instead — with probability $(1 - \epsilon)$, the action is chosen according to (2.9); with probability ϵ, the action is chosen randomly to facilitate environment exploration. The value of ϵ is typically around 0.05.

By alternating between learning (2.4) for π, and redefining π as (2.9), we thus have a well defined method of solving MDPs. The aforementioned technique is known as Q-learning, with its basic algorithm shown in algorithm 1 [Watkins, 1989].

Algorithm 1 Basic Q-learning algorithm

```
Initialize Q(s, a) arbitrarily
for number of episodes do
    Get initial state s = s_0 from environment ρ
    for steps in episode do
        Select action a from Q(s, a) (using e.g. ε-greedy)
        Observe reward r and next state s'
        Update Q(s, a) ← r + γ max_a' Q(s', a')
        Set s ← s'
    end for
end for
```

In many but the simplest cases, algorithm 1 is unlikely to work very well because bootstrapping causes very unstable estimates of the Q-value. Instead, popular RL algorithms try to tame this instability with several engineering techniques. These include a delayed target Q-network [Mnih et al., 2015] to aid in the stability of the bootstrapped value and double Q-networks to tame overestimation bias [Van Hasselt et al., 2016]. Very recently, this instability has also been tamed in very recent research directions using heavy regularization on much larger Q-networks [Nauman et al., 2024].

For cases where sampling the max from the Q-value function is intractable (e.g. when $\mathcal{A} \in \mathbb{R}^n$), one can employ sampling techniques such as cross entropy sampling [Kalashnikov et al., 2018]. An alternative branch of methods known as actor-critic methods phrase the sampling challenge as an optimization task, wherein the policy π_θ is a separate function approximator parameterized by θ, and the policy is distilled by solving (2.9) using gradient based optimization. Algorithm 2 demonstrates the basic idea of a value-based actor-critic algorithm. Algorithms of this class were first conceived by Silver et al. [2014] and later dubbed Deep Deterministic Policy Gradients (DDPG) by Lillicrap et al. [2015], more stable and performant versions such as Twin Delayed Deep Deterministic Policy Gradients (TD3) and SAC were later derived [Fujimoto et al., 2018, Haarnoja et al., 2018b,c]. TD3 uses the delayed tar-

get network and twin Q-networks first used in normal Q-learning to stabilize training, while SAC is a version of TD3 that attempts to maximize the entropy of the action distribution, encouraging exploration and further boosting learning stability.

Algorithm 2 Basic value-based actor-critic algorithm

Initialize $Q(\boldsymbol{s}, \boldsymbol{a})$ arbitrarily
Initialize $\pi_\theta(\boldsymbol{s})$ arbitrarily
for number of episodes **do**
  Get initial state $\boldsymbol{s} = \boldsymbol{s}_0$ from environment ρ
  for steps in episode **do**
    Sample action from policy $\boldsymbol{a} \sim \pi_\theta(\boldsymbol{s})$
    Observe reward r and next state $\boldsymbol{s}'$
    Update $Q(\boldsymbol{s}, \boldsymbol{a}) \leftarrow r + \gamma \mathbb{E}_{\boldsymbol{a}' \sim \pi_\theta} Q(\boldsymbol{s}', \boldsymbol{a}')$
    Update $\theta \leftarrow \arg\max_\theta \mathbb{E}_{\boldsymbol{a} \sim \pi_\theta}[Q(\boldsymbol{s}, \boldsymbol{a})]$
    Set $\boldsymbol{s} \leftarrow \boldsymbol{s}'$
  end for
end for

The primary strength of value-based methods is their ability to leverage past experience. In practice, this involves storing all past state transitions $\{\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}'\}$ in a replay buffer $\mathcal{D}$, which continually increases in size with more environment interactions. This allows models to not fall into a phenomena known as catastrophic forgetting, a phenomena present in policy-based algorithms [Sullivan et al., 2022]. The downside of value-based methods is that the wall clock time required to train such models is much longer than policy-based methods, an attribute caused by the need to learn over the replay buffer after every environment episode interaction.

Policy-Based Reinforcement Learning Policy-based RL relies on the notion of the policy gradient. To formalize the policy gradient, we introduce the notion of policy performance as the expected state-value for all states for a policy:

$$J(\pi) = \mathbb{E}_{\boldsymbol{s} \sim \rho, \boldsymbol{a}, \sim \pi}[Q(\boldsymbol{s}, \boldsymbol{a})] \tag{2.10}$$

The policy gradient revolves around finding the gradient of policy performance and then stepping in a direction that improves that value. Thus, we intend to find the gradient $\nabla_\pi J(\pi)$:

$$\nabla_\pi J(\pi) = \tag{2.11}$$
$$= \nabla_\pi \mathbb{E}_{s_t \sim \rho, a_t \sim \pi}[Q(s_t, a_t)] \tag{2.12}$$
$$= \nabla_\pi \sum_{s, s_{t+1} \in \mathcal{S}} \rho(s_{t+1}|s_t, a_t) \sum_{a_t \in \mathcal{A}} \pi(a_t|s_t) Q(s_t, a_t) \tag{2.13}$$
$$= \sum_{s, s_{t+1} \in \mathcal{S}} \rho(s_{t+1}|s_t, a_t) \sum_{a_t \in \mathcal{A}} \pi(a_t|s_t) \nabla_\pi \log \pi(a_t|s_t) Q(s_t, a_t) \tag{2.14}$$
$$= \mathbb{E}_{s_{t+1}, s_t \sim \rho, a_t \sim \pi}[\nabla_\pi \log \pi(a_t|s_t) Q(s, a)] \tag{2.15}$$
$$= \mathbb{E}_{\rho, \pi}[\nabla_\pi \log \pi(a|s) Q(s, a)] \tag{2.16}$$

(2.13) pulls the probabilities out of the expectation and into the summation itself. (2.14) uses the identity $\nabla_x x = x \nabla_x \log x$. (2.16) is just a more concise way of writing (2.15).

The policy can then be updated in the positive direction of (2.16) to obtain a better policy. A basic policy gradient algorithm is summarized in algorithm 3.

Algorithm 3 Basic policy gradient algorithm

Initialize $\pi_\theta(s)$ arbitrarily
Select learning rate α
for number of episodes **do**
  Initialize trajectory buffer $\mathcal{D} = \{\}$
  Get initial state $s = s_0$ from environment ρ
  for steps in episode **do**
    Sample action from policy $a \sim \pi_\theta(s)$
    Observe reward r and next state s'
    Add to buffer $\mathcal{D} \cup \{\{s, a, r\}\}$
    Set $s \leftarrow s'$
  end for
  Reset policy gradient $\nabla_\theta J(\tau) = 0$
  for each $\{s, a, r\}$ in $\mathcal{D}$ **do**
    Compute $Q(s, a)$
    Compute $\nabla_\theta J(\pi)$ ▷ (2.16)
  end for
  Update $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi)$
end for

The vanilla policy gradient algorithm as shown in algorithm 3 is unlikely to work except for the most simplest of cases. This is caused by several core problems:

- **High variance updates** The gradients of the policy are influenced by the sum of discounted returns $Q(\boldsymbol{s}, \boldsymbol{a})$, this represents an unbiased and high variance estimate of the performance of the policy.
- **Distributional shift** The gradient update is only accurate under the assumption that the policy π is the same policy that is used to generate the trajectories τ. Most policy gradient methods are therefore known as on-policy algorithms, whereas value-based algorithms are commonly of the off-policy variant.

To tackle the problem of high variance updates, the simplest method would be to reduce variance by subtracting a baseline, this is known as taking an advantage estimate:

$$A(\boldsymbol{s}, \boldsymbol{a}) = Q(\boldsymbol{s}, \boldsymbol{a}) - V(\boldsymbol{s}) \tag{2.17}$$

(2.16) is thus modified to:

$$\nabla_\pi J(\pi) = \mathbb{E}_{\rho,\pi}[\nabla_\pi \log \pi(\boldsymbol{a}|\boldsymbol{s}) A(\boldsymbol{s}_t, \boldsymbol{a}_t)] \tag{2.18}$$

Doing so requires that a separate model be placed over $V(\boldsymbol{s})$ that is learned via Monte Carlo estimation from independent trajectories. Other methods of reducing variance include techniques that introduce bias to further reduce variance [Schulman et al., 2015b, Ng et al., 1999]. To tackle distributional shift, several techniques have been proposed, likely the first of which is importance sampling [Tokdar and Kass, 2010]. Alternative methods of tackling distributional shift include simply preventing the policy update step from updating the policy too much. This has resulted in very powerful RL algorithms, including Trust Region Policy Optimization (TRPO) [Schulman et al., 2015a] and Proximal Policy Optimization (PPO) [Schulman et al., 2017], the latter of which is the most popular RL algorithm in implementation.

The core benefit of policy-based methods is that they work well without requiring a replay buffer of past trajectories. This allows learning to happen quickly after each environment interaction. However, due to the algorithms not keeping track of past trajectories and the nature of preventing distributional shift, on-policy algorithms typically require an order of magnitude more samples to obtain good policies when compared to off-policy methods. This may not be a bad proposition if the simulator that is being used to represent the environment is fast, which is often easily done using hardware acceleration [Makoviychuk et al., 2021, Weng et al., 2022, Freeman et al., 2021]. Further, on-policy algorithms are also notorious for being difficult to train, requiring many engineering tricks to get going well [Andrychowicz et al., 2021].

Model-Based Reinforcement Learning A fundamental challenge with conventional RL is the inherently bad sample efficiency; a large number of environment interactions

is required to obtain a good policy. To tackle this, Sutton and Barto [2018] hypothesised using environment abstractions for better learning, wherein a learned model is used in place of the actual environment itself to allow an agent to perform abstracted planning and learning. Ha and Schmidhuber [2018] first proposed the notion of world models - models which are a drop in replacement for the environment itself within the MDP framework. The proposal is for a model f_ϕ which fulfils the following:

$$f_\phi = \arg\min_f D(f(\boldsymbol{s}, \boldsymbol{a}), \rho(\boldsymbol{s}, \boldsymbol{a}))\forall \boldsymbol{s}, \boldsymbol{a} \tag{2.19}$$

where $D(\cdot, \cdot)$ represents the notation for statistical divergence (e.g. Kullback-Leibler, Wasserstein, etc).

Using a world model, the RL agent can then execute as many as many environment interactions as required within the world model, so long as the regions explored within the world model are similar to those in the real environment. This paradigm has led to many impressive algorithms, such as Dreamer [Hafner et al., 2019], DreamerV2 [Hafner et al., 2020], DayDreamer[Wu et al., 2022], Denoised MDPs [Wang et al., 2022], etc. World models have also been used in the context of tree search for RL, where truncated Monte Carlo Tree Search is used to select the best action for a given state. Examples of this form of algorithm include AlphaZero [Silver et al., 2018], MuZero [Schrittwieser et al., 2020] and EfficientZero [Ye et al., 2021].

From a sample efficiency and performance perspective, model-based RL represent a class of models with the highest sample efficiencies and best performance. However, as lamented in Sutton [2019]'s Bitter Lesson, models of this class utilize the most amount of computational resources, as all of the conventional models of a RL algorithm are required on top of the world model itself.

2.3.3 Simulation to Reality for Reinforcement Learning

Most RL problems assume a reset-able environment from which the RL algorithm is able to interact with. This is most conveniently done as a computer simulated environment. This bodes to the low sample efficiency of RL algorithms in general, and the transfer of a policy trained in simulation to the real world is most commonly known as Simulation to Reality Transfer (Sim2Real). The main governing factor of the Sim2Real gap is triggered by the inconsistency of friction, damping, mass, collision properties, and other physical attributes. If a policy relies on visual information to act, the gap is further exacerbated by aspects such as scene lighting, occlusion, motion blur, lens profiles, and other optical aspects that cause real images to not match those in simulation. To this end, methods of solving the Sim2Real challenge can broadly be categorized into one of three techniques: system identification, domain randomization, and domain adaptation. Typically, in the Sim2Real literature, the domain in which an agent is trained in is known as the source domain, and the real environment in which the agent must act in is known as the target domain.

System Identification refers to the paradigm of building as accurate simulators as possible by identifying crucial parameters of the real model. Because system identification will not be used in the work done in this thesis, only a surfacial level of introduction will be given. Indeed, the field of system identification is huge, and requires many bookshelves to be adequately covered. The core of performing system identification to build estimating models is statistical theory. It revolves around the idea of building a model using a selection or collection of models from a model class that matches the true description of a system whilst minimizing model complexity, based on the available information about the system itself. Consider an unknown function $g_0(x)$. For a sequence of x-values $\{x_1, x_2, ..., x_N\}$, we observe a set of corresponding function values with some noise:

$$y(t) = g_0(x_t) + e(t) \tag{2.20}$$

Where $e(t)$ is noise. The basic problem is to therefore construct an estimate $\hat{g}_N(x)$ that matches $g_0(x_t)$ as closely as possible. The term 'matches' here is ambiguous, and in the field of system identification, many metrics in the field of information theory exist for measuring the closeness of two models such as Akaike's Final Prediction Error, Akaike's Information Criterion, Generalized Cross-Validation Criterion, Empirical Risk, etc. [Akaike, 1998, Sakamoto et al., 1986, Li, 1987].

Domain Randomization describes to the idea of training an agent in the source environment ρ_s with randomized properties $\xi \in \Xi$ that are randomly chosen for every trajectory, with the goal of later transferring the learned policy to a target environment ρ_t. With a reasonable definition of Ξ, the hope is that the properties of the target environment ξ_t are also in Ξ. Sim2Real can work for physical dynamics [Peng et al., 2018], as well as for image-based tasks [Loquercio et al., 2019], and has seen extensive usage across a range of robotics tasks.

Instead of randomly sampling properties from Ξ, domain randomization can instead be viewed as a bilevel optimization problem, where the space of properties Ξ_ϕ is parameterized with parameters ϕ. With access to the source domain, the optimal parameters ϕ^* can then be learned via:

$$\phi^* = \arg\max_{\phi} \mathbb{E}_{\rho_t}[\mathcal{P}(\pi_{\theta^*(\phi)})] \tag{2.21}$$

$$\text{s.t.} \quad \theta^*(\phi) = \arg\max_{\theta} \mathbb{E}_{\rho_s|\Xi_\phi}[\mathcal{P}(\pi_\theta)] \tag{2.22}$$

where $\mathcal{P}$ represents a utility function, typically the total reward per trajectory.

In short, the space of properties Ξ is continually refined by repeatedly testing the learned policy in the source domain. Such a bilevel optimization can be done in several ways not withstanding evolutionary algorithms [Yu et al., 2018], RL [Kar et al., 2019] or a neural networked version of state space identification [Chebotar et al., 2019].

Domain Adaptation refers to the notion of adapting the target environment ρ_t to match that of the source environment ρ_s. This is typically done for image-based tasks, where the state is fed to the policy as an image (or sequence of images). Since the feature space of the source and target domain are different, the spirit of domain adaptation is aimed at aligning the two feature spaces such that the policy will perceive either domains as being similar. There are primarily three variants of domain adaptation: discrepancy-based, adversarial-based and reconstruction-based.

Discrepancy-based domain adaptation typically use a single feature extraction backbone, and aim to minimize a statistical distance between extracted features from the source and target domains in order to align feature spaces. Some of the influential work in this space include Deep Domain Confusion [Tzeng et al., 2014], Deep Adaptation Networks [Long et al., 2015b], and Correlation Alignment [Sun et al., 2016].

Adversarial Domain Adaptation builds a domain classifier to identify whether the features produced by a feature extractor come from states in the source domain or target domain. The training loss is then augmented to make the feature extractor work towards fooling the domain classifier. Although in principle this is similar to discrepancy-based methods, the primary difference is that a completely separate classifier is trained over the feature space of the feature extractor; discrepancy-based domain adaptation instead use a statistical metric over the feature space which negates the training of a completely separate network. Some examples include Adversarial Training [Ganin et al., 2016, Bousmalis et al., 2017] and Simultaneous Deep Transfer [Tzeng et al., 2015].

Reconstruction-based domain adaptation leverages the field of Generative Adversarial Networks (GANs) [Gonog and Zhou, 2019], wherein two networks, a generator and a discriminator, compete in a minmax game. The generator takes states from the target domain $\mathcal{S}_t$, and converts them into states that resemble the source domain $\mathcal{S}_s$, while the discriminator takes in states from either domains and attempts to distinguish them both. The goal of the generator is to fool the discriminator that its generated states come from the source domain while retaining information from the target domain, while the discriminator aims to visually discriminate between generator-created states and true source states. Through iterative training, the generator eventually generates states that are indistinguishable from the true source states. Perhaps the most prevalent reconstruction-based domain adaptation technique is CycleGAN [Zhu et al., 2017] and derivatives.

2.3.4 Deep Learning for UAV Navigation

Some examples of off-the-shelf RL algorithms applied to obstacle avoidance and navigation include works by Zeng et al. [2021], Huang et al. [2019], Abedin et al. [2020], Oubbati et al. [2021a,b]. Various interesting implementations include one by Wang et al. [2020a] which utilized sparse rewards for training a UAV-based obstacle avoidance algorithm utilizing non-expert helpers. Sadeghi and Levine [2016] discretize the action space of the UAV as regions in an image obtained by a monocular camera, and train the agent in a pure simulation en-

vironment using domain randomization to attain autonomous navigation in the real world without training on a single real image. Gandhi et al. [2017] intentionally crash a real drone 11,500 times to generate a diverse datasets of crashes, then train learn a value function which can then be utilized to avoid crashing. Their approach shows impressive results, achieving safe flight even in extremely cluttered environments. Kouris and Bouganis [2018] predicts distance to collision using a monocular camera on a drone, and then utilizes this heuristic to perform obstacle avoidance and navigation. Loquercio et al. [2018] utilize a dataset of bicycle and car data within a city to train a UAV to fly in an urban environment. Their results show that this approach allows the UAV to navigate through indoor corridors and parking lots in a dense city. Some landmark experiments include works by Kaufmann et al. [2018, 2023, 2019], Loquercio et al. [2019], Song et al. [2023], which achieved better-than-human level drone racing by using RL and extensive training in simulation augmented by system identification in the real world.

2.3.5 Arguments for Deep Learning based Approaches

While heuristic methods have efficacy in certain scenarios, the use of deep learning-based approaches in autonomous UAV obstacle avoidance and navigation brings forth a multitude of advantages that significantly elevate the capabilities of unmanned aerial vehicles. The following arguments highlight the superiority of deep learning-enabled techniques in this context:

Direct and Effective Goal Optimization One significant advantage of deep learning approaches, particularly RL, lies in their ability to optimize a more direct and effective goal. In traditional autonomous systems, classical control methods typically involve a three-stage process—sensor fusion, planning, and optimal control. RL, on the other hand, enables the UAV to learn from its interactions with the environment, optimizing a better goal through a direct reward function. This allows for a more adaptive and context-aware decision-making process compared to the rigid structure of classical methods.

At first, this paradigm was demonstrated in well-defined games like Go [Silver et al., 2016]. Then, it was applied to highly complex games such as Defence of the Ancients 2 and Starcraft [Vinyals et al., 2019, Berner et al., 2019]. More recently, this has enabled superhuman performance to be achieved using computer algorithms, particularly in car and UAV racing scenarios [Song et al., 2023, Kaufmann et al., 2023]. This scenario has been explicitly studied, and while it was determined that classical methods can bring the performance of navigation and control algorithms to the level exhibited by RL, the ease of finding immediately optimal solutions in RL makes iterating through various design constraints much faster [Wurman et al., 2022].

Richer Feature Extraction and Recognition The trend of optimizing a better goal is not only seen for algorithms in actionable tasks, but in all aspects where deep learning can be applied. For instance, when designing obstacle detection algorithms, the classical

method may involve hard-coded rules such as Histogram of Gradientss (HOGs) [Chayeb et al., 2014] or edge-finding in LiDAR based scenarios Larson and Trivedi [2011].

In contrast, deep learning models, particularly CNNs, are employed to automatically learn hierarchical representations from raw data. This class of learning model excels at capturing intricate patterns and hierarchical features, making them well-suited for tasks like obstacle detection. By presenting the algorithm with a diverse set of labeled examples, the model learns to discern relevant features and representations that aid in accurate obstacle detection [Hu et al., 2018, He et al., 2016].

The advantage of using deep learning lies in its ability to adapt and generalize to different scenarios without the need for manual feature engineering. While traditional methods often struggle to handle variations in data, requiring explicit adjustments to accommodate different environments or conditions. Deep learning models, on the other hand, inherently learn robust representations, making them more versatile and capable of handling diverse real-world situations.

Moreover, deep learning models can be fine-tuned or extended to address specific challenges or incorporate additional information. Transfer learning, for example, allows pre-trained models on large datasets to be adapted to a particular task with limited labeled data. This facilitates the development of more efficient and effective obstacle detection systems.

2.4 Modern Computer Vision

When performing maintenance needs detection using only UAV-based sensors, the predominant method of doing so is through computer vision via onboard cameras. For that reason, this literature review section aims to provide the reader with a sufficient background of computer vision techniques before covering how they can and are currently be used for detecting maintenance needs along railways or infrastructure in . Formally, there are primarily three categories of deep learning computer vision tasks – image classification, object detection, and instance segmentation, each one being the more progressively harder task. Image classification is the notion of identifying the main object type in an image and assigning a corresponding class label to it. Object detection denotes the task of identifying one or multiple objects in the image, and then drawing bounding boxes around each object and assigning class labels to them. Instance segmentation is like object detection, but instead of bounding boxes, specific pixels of the objects are highlighted; this is known as masking.

2.4.1 Datasets for Modern Computer Vision

Modern computer vision algorithms are usually benchmarked by their performance on respective datasets. Datasets play a crucial role in advancing the field of computer vision by providing a diverse and extensive collection of annotated images that serve as training, validation, and testing data for machine learning models. These datasets serve as benchmarks for evaluating the performance of algorithms, fostering healthy competition and innovation. A well-curated dataset reflects the real-world scenarios and challenges faced by computer vision applications, enabling the development of robust models that generalize effectively. Moreover, datasets facilitate reproducibility and comparability of research results across different studies, laying the foundation for a cumulative body of knowledge. They not only drive progress in tasks like image classification, object detection, and segmentation but also support the exploration of emerging areas such as scene understanding, autonomous systems, and fine-grained analysis. Some of most important datasets in the realm of computer vision thus far include:

- **Image Recognition Datasets**
  - **ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC 2012):** This dataset consists of millions of labeled images across thousands of categories. It was used for the ILSVRC 2012 competition, a pivotal event in the development of deep learning for image classification [He et al., 2015].
  - **Modified National Institute of Standards and Technology Handwritten Digit Dataset 1998 (MNIST):** MNIST is a classic dataset widely used for handwritten digit recognition [LeCun et al., 1998a].
  - **CIFAR-10:** The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes, with 6,000 images per class [Krizhevsky et al., 2009].

  - **Fashion-MNIST:** Similar to MNIST, Fashion-MNIST consists of 28x28 grayscale images of 10 fashion categories [Xiao et al., 2017].
  - **SVHN (Street View House Numbers):** SVHN is a dataset for digit recognition in natural street-level images [Netzer et al., 2011].
  - **Caltech-256:** Caltech-256 is a collection of over 30,000 images across 256 object categories [Griffin et al., 2007].

- **Object Detection Datasets**
  - **PASCAL Visual Object Classes 2012 (PASCAL VOC 2012):** PASCAL VOC is a benchmark dataset for object detection and classification [Everingham et al., 2010].
  - **Places 2014:** This dataset focuses on scene recognition and understanding [Zhou et al., 2017a].
  - **Open Images 2017 (OICOD):** Open Images is a large-scale dataset with millions of images annotated with object bounding boxes [Kuznetsova et al., 2018].
  - **COCO-Stuff:** An extension of MS-COCO, COCO-Stuff includes rich annotations for semantic segmentation [Caesar et al., 2017].
  - **KITTI Vision Benchmark Suite:** KITTI is a dataset for autonomous driving [Geiger et al., 2012].

- **Object Detection and Instance Segmentation Datasets**
  - **Microsoft Common Object in Context 2014 (MS-COCO):** MS-COCO is a widely used dataset for object detection, segmentation, and captioning [Lin et al., 2014].
  - **ImageNet Large Scale Visual Recognition Challenge 2017 (ILSVRC 2017):** This iteration of the ImageNet challenge builds upon the success of ILSVRC 2012 [Russakovsky et al., 2015].
  - **Cityscapes:** Cityscapes is a dataset for urban scene understanding [Cordts et al., 2016].
  - **ADE20K:** ADE20K is an extensive dataset for semantic segmentation [Zhou et al., 2017b].
  - **SemanticKITTI:** SemanticKITTI is a dataset for semantic segmentation and scene understanding in autonomous driving scenarios [Behley et al., 2019].

2.4.2 Image Classification

Image classification is the basis from which object detection and instance segmentation tasks are built on. For this reason, much of the foundation in object detection and instance segmentation are attributed to lessons learnt from image classification. The first successes of

image classification using CNNs have been used to classify the handwritten digits and German traffic signs [LeCun et al., 1998a,b]. These networks are notable for outperforming their benchmarks. However, talk of Deep CNNs was not yet prevalent. Arguably the first modern success of a Deep CNNs which brought about large mainstream discussion was in 2012 when Krizhevsky et al. [2012] published AlexNet. AlexNet was designed to perform image classification in the ILSVRC 2012. While the network architecture of AlexNet was not groundbreaking by modern standards (being simply an 8 layered CNN), the computational and mathematical background and foundations that built AlexNet formulate much of the backbone for modern Deep CNNs. Several novel innovations are central to the success of AlexNet. Data augmentation techniques utilizing translations, reflections, color maps, and even principal component analysis on the pixel values are first used to expand the size of the dataset. AlexNet also implemented the ReLU function inspired by [Jarrett et al., 2009], further usage of a local response normalization improved the networks performance to the high variance and dimensionality in data, and splitting the network across two GPUs arranged in parallel similar to the work of Cireşan et al. [2011]. In addition, a powerful regularization technique known as Dropout was used, adapted from Srivastava et al. [2014]. Dropout describes a technique where the outputs of random neurons in the network are turned off during training, usually with a 50% probability Srivastava et al. [2014]. This prevents neurons from relying on the outputs of the neurons next to them, effectively forcing every neuron to stand for itself. Formally, this prevents the complex coadaptation between neuron activations. During the test time, the outputs of each neurons is then re-scaled to compensate for there twice as many neurons firing. This made the network much more effective at generalization and prevented overfitting. Since the dominance of AlexNet, every ILSVRC since 2012 has been dominated by Deep CNNs, and advancements in the field has grown by leaps and bounds.

Although image recognition is now deemed as a solved problem, some of the more modern architectures stemming from this sub-field of computer vision include VGGNet [Simonyan and Zisserman, 2014], ResNet [He et al., 2016], InceptionNet [Szegedy et al., 2015, 2016], and MobileNet [Howard et al., 2017]. VGGNet, with its simple and uniform architecture, demonstrated the importance of depth in CNNs. It introduced the concept of using very small (3x3) convolutional filters stacked on top of each other. VGGNet typically had 16 or 19 weight layers, making it deeper than AlexNet. The straightforward architecture made it easy to understand and implement. ResNet addressed the challenge of training very deep networks by introducing residual learning. It utilized residual blocks, allowing for the training of extremely deep networks without suffering from vanishing gradients. The architecture includes skip connections that bypass one or more layers, facilitating the flow of gradients during training. InceptionNet, also known as GoogLeNet, introduced the concept of inception modules, which use multiple filters of different sizes in parallel and concatenate their outputs. This helps capture multi-scale features efficiently. It is also known for its network-in-network design, making it computationally efficient. Finally, MobileNet was designed for mobile and edge devices, emphasizing computational efficiency while maintaining reasonable accuracy. It is widely used in applications with limited computational resources. MobileNet

introduced depth-wise separable convolutions, which significantly reduced the number of parameters and computations compared to traditional convolutional layers.

2.4.3 Object Detection

Object detection refers to the task of identifying and localizing objects within an image. The task of object detection is threefold; determine if there are instances of objects in an image, determine the class(es) of objects which the instances come from, and draw bounding boxes around the objects to localize the object. The requirement for bounding boxes is not strictly necessary, as there are instances that perform object detection without bounding boxes [Yang et al., 2019, Ribera et al., 2019], but most benchmarks for object detection use bounding boxes or segmentation masks. Object detection is significantly harder from image classification because the algorithms must also be robust across a range of intrinsic and extrinsic conditions such as variations in color, texture, shape, size, scale, as well as situations such as variable lighting, partial occlusions, blur, clutter, noise, and motion among others for the same object. Other artifacts of extrinsic conditions include the presence of high ISO related noise, poor resolution, or distortions and transformations of the images attributed to lens properties. Additionally, certain object detection tasks have the challenge of interclass variation. As opposed to merely distinguishing the difference between the categories of 'dog', 'cat' and 'duck', interclass variation describes the difference between 'Siberian Husky', 'Poodle' and 'Rottweiler', which are all breeds of dogs.

There are primarily two main detection frameworks for object detection – two stage detection frameworks, also known as region proposal frameworks; and one stage detection frameworks, also known as unified detection frameworks. In the first example, many regions from the image are extracted using a Region Proposal Network (RPN), then, a generic image classification algorithm is applied to the different regions. The RPN has the task of predicting the 'objectness' of various regions in the image. 'Objectness' is a measure of how much the network thinks a region of the image is a foreground object or just the background. With the exception of OverFeat [Sermanet et al., 2013], several two stage frameworks were published almost simultaneously by independent parties of the research community that started the object detection revolution.

Two Stage Detection Frameworks Two-stage detection frameworks, also known as region proposal frameworks, were among the early breakthroughs in object detection using deep networks. One of the pioneering models in this category is the Region Proposal Network (RPN), which extracts multiple regions from an image and subsequently applies a generic image classification algorithm to these regions. The RPN plays a crucial role in predicting the 'objectness' of various regions in the image, essentially determining whether a region is likely to contain an object or not [Ren et al., 2015].

Faster R-CNN, introduced by Ren et al. [2015], marked a significant breakthrough by combining RPNs with the traditional object detection pipeline. The RPN generates region proposals for potential object locations, and these proposals are then fed into the subsequent

detection network. This modular design not only improves accuracy but also streamlines the process, making it computationally more efficient. One of Faster R-CNN's strengths lies in its flexibility and ease of integration with different backbone architectures, such as ResNet and VGG. This adaptability allows researchers to leverage powerful pre-trained models, enhancing the detection performance across various datasets and domains. However, it's worth noting that while Faster R-CNN achieves impressive accuracy, its inference speed can be a limiting factor in real-time applications. This is primarily due to the two-stage nature of the framework, where region proposals are generated first and then processed through the detection network. Despite this drawback, Faster R-CNN remains a cornerstone in the evolution of object detection frameworks.

Building upon the success of Faster R-CNN, other two-stage detection frameworks emerged. Notably, the R-FCN (Region-based Fully Convolutional Networks) by Dai et al. [2016] introduced position-sensitive score maps for object localization, further enhancing the precision of object detection. Later on, He et al. [2017] introduced Mask R-CNN in 2017, extending the capabilities of Faster R-CNN to include instance segmentation. Mask R-CNN seamlessly integrates a pixel-wise segmentation branch alongside the existing object detection components. This addition enables the model not only to identify objects but also to provide precise segmentation masks for each instance within an image. The introduction of the mask branch in Mask R-CNN sets it apart from its predecessors, making it particularly valuable in applications that demand precise delineation of object boundaries, such as in medical imaging or autonomous driving scenarios. Despite this added complexity, Mask R-CNN manages to maintain competitive inference speeds, making it a versatile choice for a wide range of computer vision tasks. In terms of performance, Mask R-CNN consistently outperforms its predecessors in benchmarks for instance segmentation. The incorporation of the mask prediction does not compromise the accuracy of object detection, showcasing the robustness of this framework.

Single Stage Detection Frameworks In contrast to the two-stage frameworks, single-stage detection frameworks, also known as unified detection frameworks, aim to simplify the object detection process by directly predicting object bounding boxes and class labels in a single pass. These frameworks eliminate the need for a separate region proposal step, making them computationally more efficient.

The seminal YOLO (You Only Look Once) model, introduced by Redmon et al. [2016], revolutionized the field of object detection by presenting a single-stage architecture capable of real-time processing. YOLO divides the input image into a grid and predicts bounding boxes and class probabilities directly from each grid cell, thereby achieving remarkable efficiency by, literally, looking only once at the input image. YOLOv2, also known as YOLO9000, also introduced by Redmon and Farhadi [2017], marked a significant evolution by tackling the challenge of handling a vast number of object categories. The model incorporated a hierarchical labelling approach, leveraging an extensive dataset with a diverse range of classes. YOLO9000 implemented joint training on the COCO dataset and an auxiliary dataset, Im-

ageNet, containing a more extensive set of classes. This allowed YOLOv2 to simultaneously detect a wide range of objects, including those not present in the COCO dataset.

Following YOLOv2, YOLOv3, proposed by Redmon and Farhadi [2018], made substantial contributions to both accuracy and efficiency. YOLOv3 introduced the concept of a "darknet-53" backbone, a deeper and more complex neural network architecture compared to its predecessors. The darknet-53 backbone aimed to capture more intricate features and representations, enhancing the model's ability to discern objects in complex scenes. Additionally, YOLOv3 incorporated a feature called PANet (Path Aggregation Network), which facilitated information flow across different scales, improving the model's handling of objects at varying sizes. Joseph Redmon — the original author of the YOLO model, has famously stopped publicly available computer vision work after the release of YOLOv3, citing concerns over the misuse of these models. Despite this, his range of models have introduced many novel and groundbreaking ideas in the realm of computer vision, most notably the use of k-means clustering to determine ideal bounding box sizes from the dataset, as well as the notion that one dense network is sufficient for object detection.

Several further advancements were made after YOLOv3, such as YOLOv4 by Bochkovskiy et al. [2020] YOLOv5 by the Ultralytics team Jocher [2020], YOLOv6 by Li et al. [2022], YOLOv7 by Wang et al. [2023] and YOLOv8 by the Ultralytics team . Another notable advancement is the SSD (Single Shot MultiBox Detector), proposed by Liu et al. [2016]. SSD is an object detection algorithm that operates by using a unified, single neural network to predict object bounding boxes and class scores in a single pass through the network. SSD is designed to address the challenge of detecting objects at different scales within an image efficiently. The architecture consists of a base convolutional network, often based on a pre-trained model such as VGG or ResNet, followed by a set of additional convolutional layers of varying sizes and aspect ratios. These layers are responsible for capturing features at different scales and resolutions. For each spatial location in these layers, SSD predicts multiple bounding boxes with different aspect ratios and scales, along with confidence scores for different object classes in a similar vein to YOLO's output bounding boxes. The final predictions are obtained by combining these predictions across all spatial locations and scales through non-maximal suppression. This design allows SSD to efficiently handle objects of various sizes in a single forward pass, making it suitable for real-time applications where speed and accuracy are crucial.

2.4.4 Segmentation

In contrast to object detection, the task of detection poses much greater challenges for computer vision. Segmentation entails the task of individually assigning each object/background of the scene with a class label. There currently exist two general paradigms for segmentation, semantic and instance. Semantic segmentation involves classifying each pixel in an image into specific classes or categories. The goal is to partition the image into meaningful segments, where each segment represents a region with similar semantic content. For example, for an image containing a dog, a car, and a person, semantic segmentation would assign a label to

each pixel indicating whether it belongs to the "dog" class, the "car" class, the "person" class, or another predefined class. Instance segmentation, on the other hand, not only identifies and classifies objects in an image but also distinguishes between individual instances of each class. In an image with multiple dogs, instance segmentation would not only recognize each dog as a "dog" but also provide a separate mask for each dog, uniquely identifying and distinguishing between them.

Segmentation is not an isolated field but can be considered as a natural progression from coarse to fine detection. For this reason, a lot of the core technologies utilized for object detection is leveraged over the segmentation tasks. These include the hitchhiking off backbone networks such as AlexNet, VGG, GoogLeNet, ResNet and other famous architectures utilized for the task of object detection. Similar to object detection, transfer learning is also commonly exploited as the core datasets for image segmentation are usually not as abundant in volume as object detection datasets due to the expensive annotation cost of per-pixel labelling.

Semantic Segmentation One of the first successful semantic segmentation model is the unimaginatively-named Fully Convolutional Network (FCN) [Long et al., 2015a]. The core idea of FCN was to take existing convolutional networks and to fully convolutionalize them by removing the fully connected head and replacing it with a transposed convolution with the same output resolution as the input image. The authors experimented by performing predictions on different feature maps throughout the network as well as with different backbone networks such as VGG, GoogLeNet, and AlexNet. The work here is considered the cornerstone in all semantic segmentation tasks and shows the power of convolutional networks in predicting dense prediction masks over images. They also presented one of the main troubles of segmentation that is not present with object detection – the case of lack of global context when performing local pixel predictions.

The natural progression from FCN is for the single transposed convolution pixel wise prediction layer to be replaced by multiple layers gradually increasing in size In general, the part of the network which was taken from classification networks is often termed the encoder network, whose core functionality is to take an input distribution and encode it into a lower resolution latent space through the mapping. From here, a form of decoder network, which is simply a series of transposed convolutions, takes this latent variable and simply maps it into the intended output distribution through a series of transposed convolutional layers. A softmax classifier then takes the final output and regresses the predictions onto the final class labels for each pixel to obtain the output. One example of this decoder network is SegNet [Badrinarayanan et al., 2017]. Very soon, The residual U-Net was introduced by [Ronneberger et al., 2015]. The Residual U-Net is arguably the powerhouse of segmentation tasks as of the time of writing. The U-Net architecture itself was designed for biomedical image segmentation tasks, particularly for the segmentation of neuronal structures in electron microscopic stacks. It builds upon the original U-Net, also used in the SegNet architecture, by incorporating residual connections, inspired by the success of residual networks (ResNets) in image classification tasks. By integrating residual connections into the U-Net architec-

ture, the Residual U-Net enhances information flow during both the encoding and decoding stages while also helping to alleviate the vanishing gradient problem and facilitate the training of deeper networks. This resulted in more stable and faster training for larger and more performant networks.

Instance Segmentation In contrast to semantic segmentation, instance segmentation is a much harder task. The main reason for this is because each separate instance of a class label needs to be identified. This process is not straightforward as the number of instances of the same object is not known, and hence common techniques in deep learning where 'containers' are designated for each object cannot be applied. And unlike object detection, merely making overpredictions and then applying non-maximal suppression is not a wise move because while object detection regimes need to predict 4 values per prediction – positions plus box width and height – instance segmentation requires the prediction of easily hundreds of pixels per prediction, and there is no trivial method of suppression once the predictions are made. Like object detection however, there are primarily two variants of instance segmentation, namely the same two-stage and unified framework methods, with the latter method being the category of more state-of-the-art variants, being able to run faster and more accurately.

In large, there are two variants of two stage instance segmentation frameworks — detection followed by segmentation and pixel labelling followed by clustering. The former method is by far the most popular method as it requires minimal redesign of any network. The core principle is to take a conventional object detection framework, like YOLOv3 or Faster RCNN, and add on a mask prediction head. Conceptually, for each candidate object, predict a class label, a bounding box offset, as well as an object mask. Such an architecture was introduced by the Mask RCNN seminal work He et al. [2017]. Another popular approach to instance segmentation that takes inspiration from a Mask RCNN like architecture includes Hybrid Task Cascade (HTC) [Chen et al., 2019a]. HTC is a hugely complicated network, drawing inspiration from Mask RCNN and Cascade RCNN collectively while adding on more novel features. HTC relies on the idea that instance segmentation tasks should not be performed in parallel with object detection, but that semantic segmentation and object detection outputs can be used to boost the performance of instance segmentation in a non-trivial fusion manner. HTC uses an FPN network as the backbone network, generating an output feature map. From this feature map, a semantic segmentation branch is added on to produce a semantic segmentation mask map. Separately, an RPN extracts high-level regions from the feature map, and bounding box detection is performed on these individual regions. In contrast to vanilla Faster RCNN which takes the regions as indication of individual bounding boxes, HTC uses Cascade RCNN and performs bounding box prediction and refinement on each region. These bounding box regions are then repeatedly sent for refinement and more bounding box predictions. In addition, instance segmentation masks are also performed on each of these regions. Finally, using all the generated segmentation masks, HTC fuses this information with the semantic segmentation mask to perform the final set of instance segmentation predictions. As a result of such a large architecture, although HTC gets a few points

of accuracy over previous implementations on reference benchmarks, it also runs at less than half the inference speed of Mask RCNN, at 2.5 FPS optimally. If anything, HTC shows that allowing DCNNs to perform detection at scale through tremendous manipulation and flow of information produces good results, the compromise to computational requirement is obvious and a wall of diminishing returns is to be expected.

2.5 Detecting Maintenance Needs on Railway Tracks using UAVs and Computer Vision

Implementing deep learning models for the detection of maintenance needs on railways presents its own set of challenges. Unsurprisingly, an extensive amount of research has been done in this sector, culminating in a multitude of specialized algorithms and methods for more effectively deploying deep learning techniques in this field.

2.5.1 Railway Track Maintenance Challenges with UAVs and Computer Vision

Aside from logistics challenges relating to legal and intellectual property hurdles [Operations, 2023], attempting to build deep learning models for maintenance detection on railways presents its own suite of challenges. One may assume that it is simple to gather large amounts of data and implement a machine learning model over that data. However, reality is far from that case. Primarily, the proportion of defects to non-defected track components is extremely low. For instance, railway tension clamps or clips have a failure rate of 0.17% after about 1.5 years [Kim et al., 2022]; railway sleeper design codes (UIC-713; AREMA; EN13230-2; AS1085.14) assume a sleeper lifetime of 50 years. These statistics make it very difficult to gather balanced datasets with good coverage over all possible failure types of each component. While various sampling techniques can be used to mitigate imbalanced observations in the data, this severe lack of appropriate data may make it difficult for deep learning models to make accurate predictions, even after being trained with various sampling and data augmentation techniques. For example, a component on the railway may fail in a manner that has not been represented by the dataset, causing classical supervised learning techniques to fail.

This difficulty of generating holistic datasets is the reasoning for there being a lack of open source labelled datasets for detecting railway maintenance issues, despite the ease of large scale data collection. This is attributed to the aforementioned low amounts of defect data, but also to the labor-intensive and expensive process of image labelling. Defect detection requires extensive domain knowledge and expertise, while data labelling requires copious amounts of time and patience. These two requirements for forming high quality datasets do not commonly overlap. In addition, due to the high variance in video camera parameters, train vehicle models, railway track types, environmental setting, lighting conditions and weather changes, a high quality dataset for railway tracks in one region may not transfer well to a railway track in another, limiting value when determining whether to create the dataset in the first place.

Inline with the difficulties of dataset creation, what follows is the final hurdle — a lack of openly available benchmarks. Benchmarks are an important aspect when developing any algorithm. A lack of benchmarks makes it it virtually impossible to compare the performance of one algorithm against another in literature. From this perspective, it becomes very difficult to determine whether newly developed models for the same tasks are even worth pursuing.

2.5.2 Maintenance Needs Detection Techniques

We classify machine learning techniques that can be used for maintenance needs detection into two categories — *direct detection* and *anomaly detection*. Direct detection involves training a model in a supervised manner on a set of labelled data, where the labels are typically either "ideal" or "maintenance-required", and then directly using the model predictions on new data in inference. Anomaly detection involves training the model on only non-anomalous data in either a supervised or unsupervised manner, and then applying statistical inference techniques on model predictions made on new data to determine whether the datapoint is anomalous.

More concisely, anomaly detection refers to the task of identifying rare or unusual patterns or observations within a dataset that deviate significantly from the norm or expected behaviour. By analyzing historical sensor data or performance metrics, anomaly detection algorithms learn the normal patterns, ranges, and variations of various parameters such as temperature, pressure, vibration, or power consumption. Once the baseline behavior is established, anomaly detection algorithms continuously monitor real-time sensor data or performance metrics. Any deviation or unusual pattern that differs from the established baseline beyond a set threshold is flagged as an anomaly. These anomalies can then be passed to verification systems for evaluating the severity of potential faults or failures before they cause major breakdowns or disruptions. Due to the challenges of obtaining balanced datasets of anomalous and non-anomalous data (discussed in section 2.5.1 on the previous page), anomaly detection is of particular interest to us, since obtaining a large amount of railway data under normal operating conditions is simple.

Reconstruction-based Anomaly Detection From the perspective of over-parameterized deep learning models, anomaly detection can be classified into two variants — reconstruction based and uncertainty based. Reconstruction based anomaly detection is an unsupervised learning technique that uses the concept of reconstructing the input data to identify anomalies. Reconstruction-based anomaly detection has been successfully performed in literature for a variety of tasks [Zavrtanik et al., 2021a,b]. For railways, it has also ben applied for the task of foreign debris detection [Gasparini et al., 2020, 2021]. There are primarily three steps to this:

1. **Training Phase** A dataset of anomalous and non-anomalous data is obtained, and a reconstruction model with an information bottleneck is applied strictly to the non-anomalous dataset. This allows the model to learn patterns from normal data. The simplest example of this is to utilize a variational autoencoder [Kingma and Welling, 2014] to reconstruct input data.

2. **Characterisation Phase** Data points from the anomalous and non-anomalous data are then passed through the model. The discrepancy between the input and output data is measured. This discrepancy can be measured using various metrics, such as the

mean squared error loss, binary cross entropy loss for binary data, or statistical measures such as the Kullback-Leibler divergence measurement. By passing all the available data through the model, a characterisation of the reconstruction discrepancy between anomalous to non-anomalous data can be formed. Under ideal circumstances, the range of values for anomalous data should have a non-negligible difference when compared to those of non-anomalous data. More realistically, however, the difference in reconstruction discrepancy between anomalous and non-anomalous data is dependent on the complexity of the data and the model, and having a noticeable difference between the two modes of data is not always guaranteed. In this case, the characterisation phase is used to study the distributions of reconstruction discrepancies between the anomalous and non-anomalous data. If no noticeable difference can be measured, it is likely that the model used is over-parameterized or that there may be insufficient complexity in the data. Both cases would warrant going back to the training phase.

3. **Inference Phase** New data is passed through the model. Given a suitable reconstruction discrepancy threshold obtained during the characterization phase, any data point that causes the model to produce a reconstruction discrepancy beyond the selected threshold has the potential to be anomalous, which warrants a human or additional validation to be classified as anomalous.

Uncertainty-based Anomaly Detection Meanwhile, uncertainty-based anomaly detection relies on machine learning models performing an auxiliary predictive task on data while also emitting an *epistemic* uncertainty measure about said detection. This uncertainty measure is then directly utilized as a flag for the presence of anomalies. The two primary paradigms of uncertainty estimation can be classed into frequentist and Bayesian methods.

> Note we refer to *epistemic* uncertainty in this case, which refers to 'uncertainty in the model'. This form of uncertainty stems from insufficient support in the training data and/or insufficient representational power in the learning model. Epistemic uncertainty differs from *aleatoric* uncertainty, which refers to variability in the data itself.

Frequentists Methods for Uncertainty Quantification Frequentists methods are extensively studied algorithms for uncertainty estimation. The core idea involves utilizing models where the default, untrained output support corresponds to a uniform distribution over the entire output space. The models are then iteratively updated to concentrate the output support in areas where actual data reside. More formally, let $\mathbf{x}$ be an input datapoint and $\mathbf{y} \in \mathbb{Y}$ be the corresponding label. The model is described as a probability distribution over the output space given an input, denoted as $f_\theta(\mathbf{x}) = P(\mathbf{y})$. Frequentist methods aim to train models where the distribution over output labels for unseen inputs approximates $f_\theta(\mathbf{x}) \approx U(\mathbb{Y})$. During training, the model is updated to make $f_\theta(\mathbf{x})$ closer to 1.

This can be achieved through various means. One approach is to represent the model as an ensemble of predictors $f_\theta = \{f_{\theta_1}, f_{\theta_2}, ...\}$, each predictor being different or having a different set of initialization weights. Each predictor is then trained in a normal supervised learning manner. During inference, the model uncertainty is then represented by the disagreement between each model. Various measures of disagreement include Kullback-Leibler divergence, prediction entropy, or even the standard deviation across each predictor. A slight variation of initializing an ensemble of models is to utilize Monte Carlo dropout on a single large model. In this case, multiple predictions are made with the same model, with random weights being set to zero in each prediction.

Bayesian Methods for Uncertainty Quantification On the other hand, Bayesian Methods are a less studied form of uncertainty estimation in the context of deep learning, but much more explored in the context of classical machine learning. In contrast to Frequentist methods, these methods aim to model evidential priors over the predictive distribution. For tasks involving multiple possible outcomes, you can represent the predictions using a Dirichlet distribution, which consists of a set of positive values. The number of these values matches the number of outcomes, and they depend on the model's input. When making predictions, the results are normalized to create probabilities over outputs. The uncertainty of the model's predictions can be measured as the inverse of the sum of these values, indicating how spread out the probabilities are. This technique for uncertainty estimation was made popular the context of deep learning by Sensoy et al. [2018]. For predictions with continuous outputs, one can simply model the prediction as a Gaussian distribution, where the mean of the distribution is sampled from a conjugate prior (also a Gaussian) while the variance is sampled from a Gamma distribution [Amini et al., 2020]. The conjugate prior and the Gamma distribution are the targets to be learned during training.

Another form of Bayesian methods include Bayesian neural networks, which are a type of neural network that incorporates Bayesian principles into their architecture and training process. In a Bayesian neural network, instead of having fixed values for the network's weights, these parameters are assigned prior distributions. These prior distributions capture the beliefs about the possible values of the weights before seeing any data. During training, the network is updated to find the posterior distribution over the weights given the observed data.

Given any new datapoint, Bayesian neural networks allow for a comprehensive view of the predictive distribution by propagating the predictive distribution through each weight's distribution, in a manner similar to variational inference. In practice, there may not be closed forms for this to happen for complex networks, and computationally expensive sampling techniques are often used.

2.5.3 Types of Data for the Detection of Maintenance Needs on Railway Tracks

There exist two predominant modes of data used in the detection of maintenance needs for railways — image and time-series.

Image data is typically gathered using visual sensors or cameras, and the data is represented as a 2-dimensional grid of vectors $x \in \mathbb{R}^{C \times H \times W}$ where C represents the channel dimension, H the height of the image in pixels, and W the width of the image in pixels. The channels of an image are typically of the RGB form, where the three values of each image corresponds to the intensity of red, green and blue colours. However, other formats for visual information also exist, such as HSV for hue, saturation and value. Image data can also be used to represent hyperspectral images, which may have hundreds of channels per pixel. Hyperspectral images are gathered from hyperspectral cameras — cameras that can see electromagnetic wavelengths beyond the ranges of visible light.

Time-series data represents a sequence vectors arranged temporally. The most predominant form of time-series data in the field of railway maintenance are frequency against time data for vibration sensors placed at various key points along a railway. That said, other forms such as train velocity against time or temperature of track components against time also tend to be used. From a technical perspective, a recorded video can be considered time-series data, where the base vectors represent image data.

2.5.4 Deep Learning Applied to Detecting Maintenance Needs on Railways

Numerous works exist on using deep learning models to assist in railway track maintenance. We focus on those where applicability to UAV platforms are probable. This limits the scope to systems which primarily utilize some form of image-based sensor. Broadly speaking, these techniques can then be classified into two classes — supervised and unsupervised techniques. Supervised techniques are those which require a dataset of labelled data for the model to be trained on. Conversely, unsupervised techniques are those which simply require a dataset of unlabelled data.

Computer Vision via Supervised Learning We focus on maintenance needs detection using deep learning, for which the lowest hanging fruit is direct supervised learning. Some of the most popular examples of deep supervised learning applied to railway defect detection utilize the tried-and-trusted YOLO [Redmon and Farhadi, 2018] architecture. For example, Hovad et al. [2021] utilize YOLO v3 as a supplementary method to existing eddy current and ultrasonic sensing methods to detect various surface defects which include various squats and welding defects. They utilized a dataset of 2,155 images collected from a Eurailscout UST02 recording car in Denmark where images were collected with a line scan camera. On the other hand, Cui et al. [2023] utilize a modified YOLO architecture with a custom backbone to perform defect detection on a railway noise barrier in China trained on 2,000 images collected using a manually flown DJI Matrice 300 RTK UAV. The novelty in their work involves a new backbone utilizing a model with extensive use of skip connections called SCYNet, as well as data augmentation techniques to expand the training set. Zheng et al. [2020] also utilizes the YOLO architecture — specifically the v3 variant, but this time

for defect detection on railway fasteners. This time, a labelled dataset was collected from Metro Line 8 of Beijing Subway using a downward facing, monochrome camera mounted on a moving vehicle. A total of 1,800 images were collected at a resolution of 2,048 × 1,024. Similar experiments of using a YOLO or YOLO-esque object detection framework for detecting maintenance issues in a supervised manner on a labelled dataset were also done by Lin et al. [2019a] on the national railway line in Taiwan; Hsieh et al. [2022, 2020] used various YOLO models and a dataset augmented using a Cyclic GAN architecture, also on the national railway line in Taiwan; Arrosida et al. used YOLO v3 for railway track damage detection in Indonesia; Zhang et al. [2023] with YOLO v8 and multi-scale heads to detect cracks on a railway line in China; Wang et al. [2020b] using modified YOLO v3 for detection of catenary split pins on overhead components, followed by segmentation using Google's Deeplabv3 semantic segmentation model, before the result is given to a look-up-table style classifier to identify faulty pins; and Feng et al. [2020] YOLO with a custom backbone for railway track surface defect detection; and others.

Outside of object detection, James et al. [2018] utilizes a multistage framework termed TrackNet for detection of surface defects on railway tracks. The architecture comprises a segmentation head which first segments out relevant sections of a railway track. Then, snippets of these segments are fed into a binary classifier to determine if a defect exists. A classical U-Net is used for the semantic segmentation component, while a traditional ResNet is used to make binary predictions on the resulting segmented image. They test their work on a dataset of images collected using commercial off the shelf vision track inspection system in Singapore. Chen et al. [2019a] utilize a similar idea, first identifying components then classifying defects, on the issue of component defect on moving trains. The defects they address include loose locking plates, closed valves, missing bolts, and eroded bearings on train bogey systems in China. Yang et al. [2021] explore the idea of using a non-deep learning image segmentation algorithm to identify potential defects, before using the YOLO algorithm to localize them. Precise segments of railway track are first extracted using Canny edge detection to identify the edges of the railway. Then, the GrabCut algorithm is used to semantically segment potential surface defects. Finally, the YOLO algorithm is used to localize those defects on the railway. Their work shows impressive results, achieving upwards of 97% precision and recall on a collection of 4,000 images collected from Jilin, China. Pahwa et al. [2019] aim to detect faulty valves on a carriage vehicle's bogey system using a two stage system. They first utilize a U-Net to semantically segment out valves with pixel precision, and then utilize conventional computer vision techniques and heuristics to identify whether the valve is faulty. More precisely, they take the semantically labelled valve binary image and split it into a handle and valve body, before measuring the angle between both components to identify faults. This architecture, coined FaultNet, achieves a 97.26% accuracy on their testing dataset of 73 images. He et al. [2022]'s work aims to identify track defects on MagLev rail systems. Their dataset consists of images of the MagLev system's stator coils captured using an on-vehicle high-speed monochrome camera. To handle the issue of insufficient data, they utilize a GAN to generate new defects from existing defect images to artificially increase the size of

the dataset. This dataset is then used to train a modified, lightweight Faster RCNN object detection framework to identify various faulty components.

This section highlights many landmark works within the field of railway maintenance needs detection using computer vision. However, they do not cover all possible works out there given the vastness of the field and high levels of similarity between adjacent implementations. In the current research, it is consistently observed that when provided with a well-labeled dataset of images from a specific domain, such as a particular railway track, most supervised learning methods achieve at least 80% precision and recall in challenging scenarios, often exceeding 90%. Unsurprisingly, with the representational power of modern deep learning computer vision models built with modern techniques and architectures, when presented with a dataset of well cropped, clean, and sufficiently labelled images of railway tracks, defect detecting poses minimal challenge.

Computer Vision via Unsupervised Learning Methods The predominant method of employing unsupervised learning is for anomaly detection. The most predominant form of anomaly detection for railway applications arise from the use of track side or carriage riding sensors which detect anomalies in the form of adverse vibrations or delayed operation indicating a case of impending component failure Rabatel et al. [2009], Shim et al. [2022], Kang et al. [2018], Bao et al. [2019]. These techniques tend to utilize auto-encoders or classical machine learning methods such as k-means clustering in order to detect if a reading is outside the norm. Less common methods utilize some form of generative modelling to reconstruct input signals. The difference in actual signals to the reconstructed signal is then used as a proxy metric to detect anomalies.

The most interesting applications of this sort of learning is for use with camera data. For example, to detect the presence of foreign objects on the railway [Gasparini et al., 2020, 2021], detection of faults in overhead components [Lyu et al., 2019, Chen et al., 2022], or to detect the presence of discontinuous rails [Jahan et al., 2021]. These applications utilize very similar approaches which primarily include isolating patches in the image containing components of concern, and then reconstructing the patch using some form of generative model. This approach has a high level of feasibility due to the structured method in which the data is presented to the trained models — they are either images gathered from an ego perspective view of a forward facing train carriage, or well cropped images of railway line components. In practice, this has seen a high success rates, with Jahan et al. [2021] in particular quoting a 100% precision and recall on detecting the presence of anomalies on the rail lines themselves (although their validation and test set utilized less than 1% of the total data available). In contrast to methods which utilize track side or carriage riding sensors, methods relying on computer vision pose a unique challenge due to sheer variance of image data. Everything that a learned model has not seen before may be classed as an anomaly, despite the visual anomaly never actually being on the effective components of the track itself.

A very interesting use-case of images across the temporal domain is done by Jang et al. [2019]. In their work, a line scan camera is utilized to capture images of railway track pantographs over a range of time. Since the camera is attached to the roof of a moving vehicle, the

images of the components are always similar across time. Various components in the image are first identified using a simplified SSD object detection framework. Then, similar components over time are lined up imagewise using the speeded-up robust feature (SURF) and andom sample consensus (RANSAC) algorithms. Finally, the difference between the two aligned images are used to identify the presence of defects. Their framework works very well, achieving up to 97% accuracy on detecting surface defects.

Other Deep Learning Techniques Outside of computer vision, maintenance needs detection on railway systems have predominantly utilized audio or vibration information to inform of ill-functioning components. This is particularly true in predictive maintenance regimes, where one source of very effective data for predicting remaining useful life is vibration data from onbaord sensors. For example, Davari et al. [2021] utilize an autoencoder to reconstruct vibration signals from onboard air compressor systems. Differences between the reconstructed signal and the actual signal indicate deterioration in the component. For a more comprehensive overview of the current field, De Donato et al. [2022] have published a very extensive survey of existing deep learning methods applied to railway maintenance detection utilizing video and audio sensors. On another front, Shimizu et al. [2022] utilize a similar autoencoder structure to reconstruct railway vehicle door closing speed profiles. This profile is then compared with the actual profile, before being fed into a support vector machine to identify failing components. Audio/vibration signals can also be utilized to perform anomaly detection using computer vision models. In one instance, Shim et al. [2022] construct a spectogram using measured vibration at vehicle wheels. This spectogram image is then fed to a LeNet model to identify instances of wheels with surface corrosion defects. This approach is similarly adopted by Ye et al. [2023], which demonstrated that, by using a more powerful computer vision model, such an architecture can be used to accurately detect the location of the defect on the wheel. Circumventing this two step approach altogether, Shaikh et al. [2023] instead aim to perform direct detection by passing the audio signal to a dense neural network, and shows that this can also produce favourable results, achieving upwards of 90% accuracy and precision on a synthetic dataset.

2.5.5 Summary

Detection methods for railway defect detection range from direct supervised learning to anomaly detection. These methods effectively utilize various data types, including RGB images, hyperspectral images, time-series data, and audio/vibration signals, applying deep learning models such as YOLO and U-Net for supervised tasks or autoencoders and generative models for unsupervised tasks. However, railway maintenance defect detection faces significant challenges, primarily due to data imbalance from the rarity of defects (e.g., tension clamp failures at 0.17% after 1.5 years), high labeling effort requiring expertise, and variability in camera settings and environmental conditions, which reduces dataset transferability. Furthermore, the lack of standardized benchmarks hinders the fair comparison of detection algorithms, slowing progress.

3 UAV Autonomous Navigation

This chapter explores the application of reinforcement learning methods to UAV navigation, contrasting them with traditional heuristic approaches. Reinforcement learning introduces a novel approach to solving complex challenges such as autonomous navigation. The contribution of this chapter includes the development of PyFlyt, a specialized UAV simulation software designed to facilitate the training of reinforcement learning agents. Additionally, this chapter further contributes CCGE, a reinforcement learning algorithm that enables actor critic reinforcement learning algorithms to more efficiently find solutions under the presence of a suboptimal oracle policy. Furthermore, Domain Adaptation is investigated as a promising technique to bridging the gap between simulation-trained RL algorithms, such as CCGE in PyFlyt, and real-world UAV navigation tasks.

3.1 Core Challenges

The objective of autonomously navigating a UAV along a railway corridor, while intelligently avoiding obstacles through the application of RL, is a complex task with multiple challenges. The limiting factor to success is the limited training and interaction time available on an actual railway corridor. RL relies on an agent repeatedly interacting with an environment, at first at random, in order to exploit an intrinsic reward function that is typically hand designed. Doing so on an actual railway is infeasible due to safety and regulatory constraints. In fact, training RL agents from scratch in the real world is only done in very rare laboratory settings. Therefore, to that end, the predominant way to obtaining competent RL agents for real world application is through training in simulation, and then performing a Sim2Real transfer.

In addition, RL relies on effective reward structures for the environment at hand. However, in the context of flying a UAV along a railway, reliable reward signals may be sparse, particularly when considering safety-critical events such as obstacle avoidance. One potential solution to this challenge is to devise a reward function built on the sparse reward environment that facilitates meaningful learning through techniques such as reward shaping [Ng et al., 1999]. Another solution to this challenge involves devising better learning algorithms that can bootstrap off heuristic information.

The end goal to this section of work was to successfully fly a UAV autonomously along a section of railway at BCIMO's private railway facility at Dudley, United Kingdom, under the permission of BCIMO themselves. This section of railway stretches approximately 250 meters, with a path that is shown overlayed over a satellite map in figure 3.1. The venue itself

Figure 3.1: The section of railway track where this section of work concerns.

was chosen due to constraints relating to regulation and accessibility. While the section itself is short and does not possess much in the way of variance, the techniques and algorithms developed in this chapter are meant to scale to larger domains assuming sufficient data and range.

3.2 Approach

One key insight when coming up with a solution to this challenge is that all active railway corridors must be at least the width and height of a freight car to be considered functional. If this minimum dimension is not met, any maintenance needs detected along the section is meaningless as this signals that bigger problems are at hand. This is a rarely occurring scenario in the context of simple maintenance needs detection. Because of this factor, it is reasonable for us to design a system that aims to fly a UAV along a section of railway, with a hard-coded start and end point. At any point during the flight, if the RL agent deems that forward flight is no longer possible, it is a reasonable thing to terminate the flight and signal the requirement

for human insight. Given this design envelope, there remains two primary design choices — sensor suite and algorithm.

Sensor Suite Given the proof-of-concept nature of the design, we aim to only utilize monocular RGB vision, effectively a digital camera. If this single sensor is sufficient to accomplish the task at hand, adding more sensors simply allows the implementation of more robust and powerful algorithms. For instance, adding LiDaR or RaDaR only serves to provide more information about obstacles.

Algorithm The challenge of bridging the Sim2Real gap is particularly pronounced for vision-based algorithms designed for real-world deployment. To address this disparity, one can employ either domain randomization or domain adaptation. Domain randomization, considered a straightforward end-to-end approach for training RL agents, entails creating a simulated railway corridor and continually altering textures across surfaces while the RL agent learns to navigate the railway to optimize rewards. The underlying expectation is that the RL agent becomes indifferent to variations in texture, color, and lighting, focusing solely on the essential element — the railway. Although technically viable, the task of designing the simulation's railway corridor is intricate, demanding extensive hours of 3D modeling. Consequently, we have chosen a domain adaptation approach.

Our proposed methodology, illustrated in figure 3.2, seeks to train the RL agent using semantically labeled images from simulation. Subsequently, a separate model is trained to convert real-world images into semantically segmented counterparts using real-world images gathered from manual UAV flights. The integration of these two models end-to-end facilitates autonomous flight. This approach allows us to forgo modeling extraneous components irrelevant to railway flight, such as grass, gravel, and distant objects, as they have no impact on UAV flight. Consequently, the simulation environment only needs to incorporate the railway and potential obstacles, streamlining the simulation-building process.

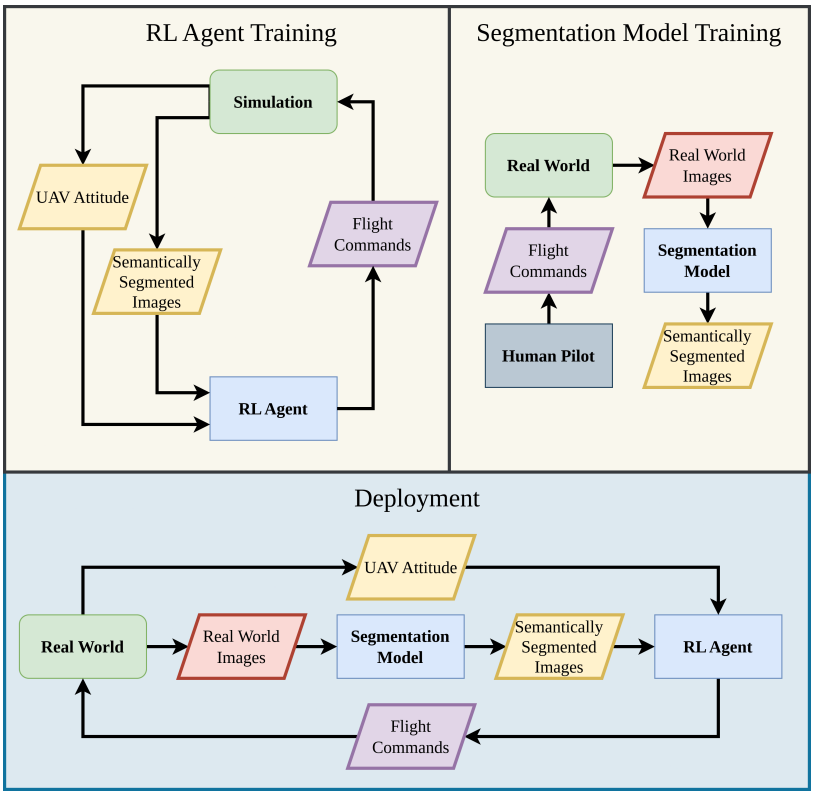

Figure 3.2: Top level proposed architecture for flying a UAV along a railway corridor separated into two separately trained components — RL agent training in the top left and segmentation model training in the top right. The trained models from both training instances (blue blocks) are combined during deployment in the manner illustrated by the block diagram in the bottom half of the image.

Figure 3.3: An image of the UAV used in all tests.

3.3 Hardware Setup

The hardware setup for this project is built for rapid iteration, having a high payload capacity to potentially carry additional sensors. In actual deployment, the system does not need to be so large or overbuilt and can be tremendously streamlined. That said, the actual hardware setup has been included here for sake of reproducibility. An image of the built UAV is shown in figure 3.3.

- **Frame Class**: Tarot 650
- **Motors**: Tarot 4008 330 kV brushless motors
- **Power System**: 6S 6200 mAh lithium polymer main battery, stepped-down to 12V for digital electronics
- **Companion Computer**: Nvidia Jetson Orin Nano Developer Kit
- **Flight Controller**: Pixhawk 6C
- **Remote Controller Link**: Express LRS 2.4 GHz
- **Camera**: Runcam 5 Orange

3.4 PyFlyt - A Python-based Hackable Framework for UAV Simulation

In the previous section, it was established that training in simulation is the most reliable and scalable method for training RL agents. Therefore, this section details the development of such a simulation enviornment for training our RL agent. Some popular open-source UAV simulation tools include Gazebo [Koenig and Howard, 2004], FlightGear [Perry, 2004], X-Plane [Garcia and Barnes, 2010], AirSim [Shah et al., 2018], UE4Sim [Mueller et al., 2017]

	Gazebo	FlightGear	X-Plane	AirSim	UE4Sim	Flightmare	PyFlyt
Multiagent Capable	Y	N	N	Y	Y	N	Y
Multiple Vehicle Offerings	Y	Y	Y	Y	Y	Y	Y
Custom Vehicles	Y	Y	N	Y	Y	Y	Y
Custom Environments	Y	Y	Y	Y	Y	Y	Y
Installation Scriptable	Y	N	N	N	N	Y	Y
First Party Python Support	Y	N	N	Y	Y	Y	Y

Table 3.1: Comparison of PyFlyt against other popular UAV simulation tools. "Installation Scriptable" refers to the ability to install using essentially a single command (possibly using a bash file) without external credential systems, a particularly important trait considering remote computing cluster systems.

and Flightmare [Song et al., 2021]. However, a majority of them make it very difficult to modify the underlying environment or don't have the required modification fidelity for the actual UAV. For instance, FlightGear and X-Plane are fixed-wing only UAV simulation tools. AirSim and UE4Sim lack a physics model and are kinematics-only. Modifying the underlying UAV model in Flightmare is non-trivial. One exceptionally suitable tool for the task is the Gazebo simulation tool. However, the highly involved structure of installation and robot construction limits its portability and modularity when running experiments.

To that end, a UAV simulation tool, termed PyFlyt[1], was built to fulfill this section of work. A visual description of the PyFlyt simulation environment is shown in figure 3.4, with a comparative table of its offerings against other simulation tools in table 3.1. In particular, the decisions around PyFlyt specifically focuses around the following key features:

- *Ease of Use*: installation using a single line of code, physics simulation, variable control and agent loop rates, collisions, multiple UAVs in the same environment, implementation of various classical control algorithms for low level flight control and extensible dynamics via the Bullet Physics Engine Coumans [2015].

- *Configurability*: unlike other tools, users can construct UAVs of any configuration using modular implementations of basic components such as motors, boosters, lifting surfaces, gimbals and cameras, all done using a Python interface, wihtout having to modify PyFlyt's source code.

- *Standardisation*: pre-implemented designs for various UAVs along with corresponding environments compatible with the Gymnasium API [Towers et al., 2023], tested with various RL algorithms.

- *Quality of Life Improvements*: deterministic simulation depending on initialisation seed, Continuous Integration (CI) testing on the master repository, a packaging and

[1]Source code hosted at https://github.com/jjshoots/PyFlyt/tree/master

release cycle to the Python Packaging Index (PyPI) and autogenerated documentation website at https://jjshoots.github.io/PyFlyt/documentation.html.

For a more in depth analysis of UAV simulators available at the time of writing, we refer readers to the excellent paper by Dimmig et al. [2024].

Gymnasium[a] [Towers et al., 2023] is an API standard for RL research.

[a] https://gymnasium.farama.org/

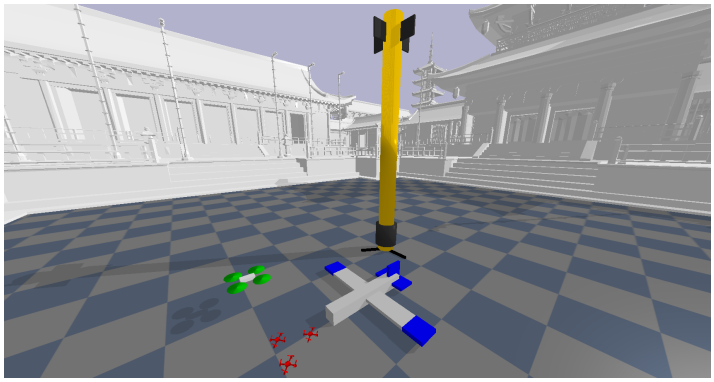

Figure 3.4: The PyFlyt simulator, showing some prebuilt UAV - one Rocket, one Fixedwing, one generic QuadX, and a cluster of three Crazyflie QuadXs.

3.4.1 Overall Architecture

From a design standard, PyFlyt was conceived to have the following core requirements:

1. User friendly installation
2. Time steppable discrete physics time steps
3. Support for user-configurable low-level control for each UAV
4. Variable looprates for physics, control, and RL
5. Automatic physics and collision tracking
6. First party support for arbitrary UAV architectures
7. Easily configured environments
8. Completely Python-based frontend

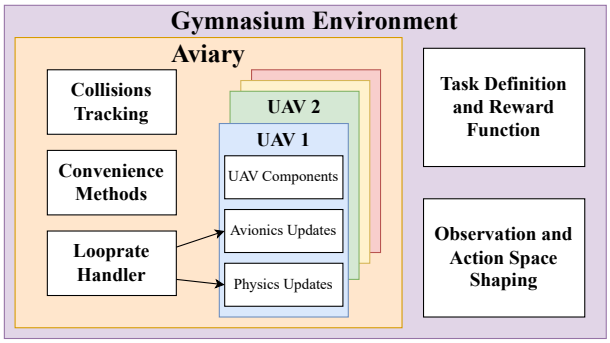

Figure 3.5: Top level overview of the PyFlyt architecture, showing the composition of classes that make up core UAV flight engine and how it integrates into a Gymnasium environment. Coloured boxes describe a class definition, while white boxes describe various functionality of the classes.

PyFlyt was designed for the development and testing of RL algorithms for UAVs. As such, these basic requirements reflect what is required for applied RL research, where rapid iteration and ease of use are paramount. The support of arbitrary UAV configurations is done through a modularized software interface, where multiple basic components such as motors and lifting surfaces can be attached together via a coded interface. The components also include the cameras — gimballed or fixed — with RGB, depth maps, and segmentation maps, allowing for the development of vision-based RL algorithms.

Since RL is likely only used for high level control instead of having access to low level actuators, low-level control schemes are supported natively, allowing the mapping of RL agent actions to setpoints instead of directly controlling raw actuator outputs. Looprates for RL, control, and physics can also run at different frequencies, allowing the RL loop to operate at a slower rate than the internal control loop — mimicking what is likely to happen in the real world.

From the design standard, the Bullet Physics Engine was chosen as a physics engine for its open-source nature, fast C++ backend, flexible rendering options, time-discrete steppable physics, Python bindings, and support for various robot file definitions. To avoid obfuscating usage patterns, PyFlyt was designed using composition as a priority, allowing users to customize the simulation environment without relying on inheritance-based approaches [Alam and Kienzle, 2012, Bitman, 1997].

The library's architecture is composed of various classes, shown in figure 3.5. At the core lies the Aviary (defined in `PyFlyt/core/aviary.py`), serving as a domain for all UAVs. The Aviary handles collision tracking between entities and performs step scheduling according to the various looprates. It also includes convenience functions for setting setpoints, arming status, and flight modes for all UAVs.

The Aviary can accommodate any number of UAVs, each defined by a combination of a Universal Robot Definition Format (URDF) file and a Python class. The URDF file de-

Listing 1 Code snippet of interacting with the PyFlyt library. This spawns three separate drones with different flightmodes, two with optional parameters, with the QuadX model (drone index 1) commanded to a new position at timestep 500.

```python
import numpy as np
from PyFlyt.core import Aviary

# the starting position and orientations
start_pos = np.array([[0.0, 5.0, 5.0], [3.0, 3.0, 1.0], [5.0, 0.0, 1.0]])
start_orn = np.zeros_like(start_pos)

# individual spawn options for three different drones
rocket_options = dict()
quadx_options = dict(use_camera=True, drone_model="primitive_drone")
fixedwing_options = dict(starting_velocity=np.array([0.0, 0.0, 0.0]))

# environment setup
env = Aviary(
    start_pos=start_pos,
    start_orn=start_orn,
    render=True,
    drone_type=["rocket", "quadx", "fixedwing"],
    drone_options=[rocket_options, quadx_options, fixedwing_options],
)

# set quadx to position control and fixedwing as nothing
env.set_mode([0, 7, 0])

# simulate for 1000 steps (1000/120 ~= 8 seconds)
for i in range(1000):
    states = env.all_states
    aux_states = env.all_aux_states

    if i == 500:
        env.setpoint(1, [1.0, 5.0, 4.0])

    env.step()
```

scribes the geometrical and physical properties of the UAV, including its mass, inertia, as well as visual and collision meshes, which can be imported as Digital Asset Exchange (DAE) files from standard CAD software. The Python class associated with each UAV inherits from a common Drone Class (defined in `PyFlyt/core/abstractions/base_drone.py`), which defines only the core functionality shared by all UAVs, necessary for interfacing with the Aviary. Each UAV defines its own base control laws, state spaces, actuator mappings, and physical dynamics interactions. The use of Python classes allows for flexibility in defining complex interactions between the UAVs and their environment. In pseudocode, the PyFlyt's simulation sequence is shown in Algorithm 4. Example code for interacting with the PyFlyt library is provided in Listing 1.

Algorithm 4 PyFlyt simulation order of operations

```
Initialize and spawn in UAVs
(Optional) Initialize and spawn in other entities
(Optional) Define wind field
for as many simulation steps as desired do
    (Optional) Receive control setpoints for all UAVs
    for all UAVs do
        if control loop then
            Update low-level controllers, send command to actuators
        end if
    end for
    for all UAVs do
        Update actuator forces and torques
    end for
    Step physics simulation
    Detect collisions
    Update camera components if any
end for
```

3.4.2 UAV Components

The field of engineering often shares common mathematical models across similar components. To provide a clear abstraction, PyFlyt offers configurable implementations of common components used in UAV construction. These components can be imported through the Python import statement `from PyFlyt.core.abstractions import {component}`. At the time of writing, there are six basic components:

1. Motors (defined in `PyFlyt/core/abstractions/Motors.py`)
2. Boosters (defined in `PyFlyt/core/abstractions/boosters.py`)
3. Gimbals (defined in `PyFlyt/core/abstractions/gimbals.py`)

4. Lifting Surfaces (defined in `PyFlyt/core/abstractions/lifting_surfaces.py`)

5. Boring Body (defined in `PyFlyt/core/abstractions/boring_body.py`)

6. Camera (defined in `PyFlyt/core/abstractions/camera.py`)

MOTOR A brushless DC motor driven propeller, simply termed motor here, is a propulsion unit consisting of a fixed pitch propeller paired to a brushless DC electric motor, commonly seen on various UAVs as a means of propulsion. We model the motor's RPM ω as a function of the throttle input $\zeta \in [-1, 1]$, using a first-order transfer function plus noise depending on the motor time constant τ_m, maximum RPM k_ω and motor fluctuation standard deviation σ_m:

$$\omega_{t+1} = \left(\frac{1}{\tau_m r_p} (\zeta k_\omega - \omega_t) + \omega_t \right) (1 + \sigma_m \omega_t \eta) \tag{3.1}$$

Where the subscript t denotes the physics timestep, r_p denotes the physics looprate in Hz, and η is noise sampled from a standard Normal. Depending on the thrust coefficient k_F and torque coefficient k_T, this gets turned into a torque and force on the motor:

$$T_m = k_T \omega^2 \tag{3.2}$$

$$F_m = k_F \omega^2 \tag{3.3}$$

BOOSTERS Unlike electric motors, fueled boosters are propulsion units that produce no meaningful torque around their thrust axis and have limited throttleability. More crucially, they depend on a fuel source that depletes with usage, changing the mass and inertia properties of the UAV. Additionally, some boosters, typically of the solid fuel variety [Abdelraouf et al., 2022], cannot be extinguished and reignited, a property we call reignitability.

In PyFlyt, the booster is parameterized with total fuel mass $m_{b_{max}}$, maximum fuel mass flow rate $\dot{m}_{b_{max}}$, the diagonal elements of the moment of inertia matrix at full fuel load $J_{xx_{max}}$, $J_{yy_{max}}$, $J_{zz_{max}}$, minimum thrust $T_{b_{min}}$, maximum thrust $T_{b_{max}}$, a boolean for reignitability $\mathbb{I}_b$, the booster ramp time constant τ_b and a thrust output fluctuation standard deviation σ_b. The booster component also allows placement of the fuel tank at an arbitrary location on the UAV, useful for simulating jet fuel powered turbojet UAVs which have fuel stored in the main body and wings.

Fueled boosters have complex mathematical models that drastically change depending on altitude, temperature, total fuel remaining, and throttle setting [d'Agostino et al., 2001, Barato et al., 2015, Greatrix, 1995]. In favour of simplicity, we adopt a simpler model for the booster in PyFlyt, useful as a baseline for developing more complex boosters should the user see necessary. The control inputs to the booster include a boolean ignition state ι and

a normalized throttle setting $\beta \in [0, 1]$. The minimum duty cycle is defined as the ratio of minimum thrust to maximum thrust:

$$\lambda_{min} = \frac{k_{t_{min}}}{k_{t_{max}}} \tag{3.4}$$

Then, the current duty cycle of the booster $\lambda \in [0, 1]$ changes according to the throttle setting through a first order transfer function, plus some noise modelled similarly to that of the motor:

$$\lambda_{t+1} = \mathbb{I}_\lambda \mathbb{I}_{m_b>0} \left(\frac{1}{\tau_b r_p} (\beta(1 - \lambda_{min}) - \lambda_t) + \lambda_t \right) (1 + \sigma_b \lambda_t \eta) \tag{3.5}$$

Where $\mathbb{I}_\lambda$ is an indicator on whether the booster is lit while $\mathbb{I}_{m_b>0}$ indicates the availability of remaining fuel.

The fuel consumption $\dot{m}_b$ of the booster depends on the duty cycle:

$$\dot{m}_b = -\lambda k_{fr} \mathbb{I}_\lambda \mathbb{I}_{m_b>0} \tag{3.6}$$

To prevent reignition of un-reignitable boosters, the booster lit indicator is recomputed at every timestep:

$$\mathbb{I}_{\lambda,t+1} = (\neg \mathbb{I}_b \wedge \mathbb{I}_{\lambda,t}) \vee \iota_t \tag{3.7}$$

In short, a booster that is not reignitable cannot be extinguished, while one that is reignitable can be extinguished and reignited at will.

From here, the thrust output depends on the duty cycle:

$$T_b = \lambda T_{b_{max}} \tag{3.8}$$

And inertia properties of the fuel tank depend on the remaining fuel balance:

$$J_i = \frac{m_b}{m_{b_{max}}} J_{i_{max}}; i \in \{xx, yy, zz\} \tag{3.9}$$

There are several deficiencies with this model. First, it does not model the sloshing of liquid fuel within the fuel tank. Second, it does not model engine deterioration with the number of reignitions. Third, solid rocket fuel boosters can have non-linear thrust profiles depending on the amount of fuel remaining [Püskülcü and Ulas, 2008], this is not modelled in the default booster model. Lastly, there is presently no limit on the number of times a booster can be reignited, unlike most liquid-fueled rocket engines [Ma et al., 2018]. That said, it is very difficult to have a parameterised booster model that suits the wide range of possible boosters in literature. For that reason, the booster component is designed with generalizability and simplicity in mind; interested users are encouraged to modify the model depending on their needs.

GIMBALS An electric gimbal is a powered joint that rotates around two axes. The first servo motor rotates the joint about the first axis, $\mathbf{c}_a$, referenced to the body frame. Similarly, the second servo motor rotates the joint about the second axis, $\mathbf{c}_b$, also referenced to the body frame. Typically, the two rotational axes are orthogonal to each other, although this is not a necessity. There are several uses for such gimballing, but the most common application is for thrust vectoring of propulsion systems [Cen et al., 2015, Kumar et al., 2020, Papachristos et al., 2014], especially in scaled model rocketry [Jain et al., 2020, Kamath, 2021].

The gimbal is controlled using a normalized input $\xi_i \in [-1, 1]; i \in a, b$ and produces three dimensional rotation matrix about the UAV's body frame. Configurable parameters include the unit vectors for the two rotational axes $\mathbf{c}_a, \mathbf{c}_b$, maximumum gimballing angles k_{g-a}, k_{g-b}, and actuation time constant τ_g. We utilize the same first order transfer function when converting input setpoint to actual gimbal angles ϵ:

$$\epsilon_{i_{t+1}} = \frac{1}{\tau_g r_p}(\xi_i k_{g-i} - \epsilon_{i_t}) + \epsilon_{i_t} \tag{3.10}$$

Where $i \in \{a, b\}$ is an axis index. To obtain the final rotation matrix, Rodrigues' rotation formula is used [Valdenebro, 2016]. Let the unit vector for an axis of rotation be $\mathbf{c}_i = \langle u_x, u_y, u_z \rangle$. We then define $\mathbf{W}$ as:

$$\mathbf{W}_i = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} \tag{3.11}$$

The Rodrigues' rotation matrix is then constructed as:

$$\mathbf{R}_i = \mathbf{I} + \sin \epsilon_i \mathbf{W}_i + \left(2 \sin^2 \frac{\epsilon_i}{2}\right) \mathbf{W}_i^2 \tag{3.12}$$

Where $\mathbf{I}$ is the identity matrix. The final rotation matrix is thus:

$$\mathbf{R} = \mathbf{R}_a \times \mathbf{R}_b \tag{3.13}$$

The term multiplying $\mathbf{W}_i^2$ is typically written as $1 - \cos(\epsilon_i)$. However, the sine variant is more numerically stable for values near $2n\pi; N \in \mathbb{Z}$. Since $\mathbf{W}_i$ and $\mathbf{W}_i^2$ only depend on the axis of rotation which is predetermined at initialization, they can be precomputed and stored, resulting in a very fast computation for the rotation matrix around each axis.

LIFTING SURFACES Lifting surfaces are effectively flat plates with an aerofoil cross section that travel through the air. In doing so, they experience a net force resulting in a lifting force $F_{s_{lift}}$, a drag force $F_{s_{drag}}$ and a net moment T_s about the aerodynamic center.

The forces and moments that act on them depend on a number of factors, most notably the angle-of-attack α and freestream velocity V_∞ according to the equation:

$$\begin{bmatrix} F_{s_{lift}} \\ F_{s_{drag}} \\ T_s \end{bmatrix} = \begin{bmatrix} C_L \\ C_D \\ C_M \end{bmatrix} \frac{1}{2} \rho A_s V_\infty^2 \tag{3.14}$$

Where A_s is the area of the lifting surface and C_L, C_D and C_M represent dimensionless lift, drag, and pitching moment coefficients, forming what we term here as a coefficient vector.

The coefficient vector can be captured in a lift, drag, and polar curve — graphical representations of how the values vary with Angle of Attack (AoA). These curves, also known as a performance profile, are unique to each aerofoil and are often obtained through extensive Computational Fluid Dynamics (CFD) simulation or wind tunnel testing.

In our approach, we adopt the method proposed by [Khan and Nahon, 2015a] to obtain the performance profile. Specifically, a lifting surface can be parameterized by its zero-lift AoA α_0, positive stall AoA α_{stall_+}, negative stall AoA α_{stall_-}, zero-lift drag coefficient C_{D_0}, lift coefficient slope C_{L_α}, and a dimensionless viscosity correction factor k_{s_η} [2].

The lifting surfaces in PyFlyt can have actuated flaps, hence the term flapped lifting surface. This flap is parameterized with a flap-to-chord ratio — a ratio of how much of the lifting surface is movable, a flap deflection limit δ_{max}, and an actuation time constant τ_s. The flap allows the lifting surface to change its aerofoil geometry, consequently altering its performance profile curves. We employ an approximation by [Khan and Nahon, 2015b] to model this modification and utilize the familiar first order transfer function to model the flap deflection δ as a function of time and normalized actuation input $\nu \in [-1, 1]$:

$$\delta_{t+1} = \frac{1}{\tau_s r_p}(\nu \delta_{max} - \delta_t) + \delta_t \tag{3.15}$$

Our approximation is generally valid for normal operating conditions, but may not accurately model the aerodynamic behavior of the lifting surface in extreme conditions, such as when experiencing major side slip, highly turbulent air, or supersonic flows.

Boring Bodies The boring body is a body that exhibits no other properties except aerodynamic drag. In general, it is parameterized by the normal areas $\mathbf{A} = (A_x, A_y, A_z)$ and drag coefficients $\mathbf{C}_d = (C_{d_x}, C_{d_y}, C_{d_z})$ for the 3 body axes. When in motion, the boring body then experiences a force of $\mathbf{F} = -0.5\rho \mathbf{v}_B^2 \mathbf{C}_d \mathbf{A}$ where $\mathbf{v}_B$ is a vector of local body velocities and ρ is the air density, fixed at 1.225 kg m^{-3}.

Camera The camera component allows the development of UAVs with vision-based capabilities. To further expand on this idea, there are no limits to the number of cameras a UAV can have, and no restrictions on their locations relative to the UAV. This flexibility allows

[2] Page 112 of [McCormick, 1994].

simulating stereo vision for depth perception [Iyengar et al., 2021], multiple viewpoints for collision avoidance [Akinola et al., 2021], or multi-camera resolution [Lee et al., 2022].

The camera in PyFlyt is parameterized with an image pixel height h and width w, field of view in degrees, camera orientation and pose relative to the UAV's body frame, and whether the camera is gimbal stabilized. The option for gimbal stabilization forces the camera to utilize rotation about the ground frame X and Y axes, providing a locked horizon view with the viewpoint following the body frame yaw.

When queried, each camera returns a set of 3 images readily obtained from the Bullet Physics Engine — an RGB image in $\mathbb{R}^{w \times h \times 4}$, a depth map in $\mathbb{R}_{+}^{w \times h \times 1}$, and a segmentation map in $\mathbb{Z}^{w \times h \times 1}$ with values indicating the ID of the entity from which the pixel came from.

3.4.3 Example UAV Models

UAVs are defined using 3 core files — a URDFs file defining the geometrical and inertial properties of the UAV, a YAML Ain't Markup Language (YAML) file defining component parameters, and a Python file defining how the components are attached, low-level control schemes and input mappings to the different actuators. PyFlyt ships with several default vehicles that users may use to build their own simulation environments, each defined under `PyFlyt/core/drones/*.py`. In this section, we detail the design of three such models - a quadrotor, a fixedwing, and a rocket.

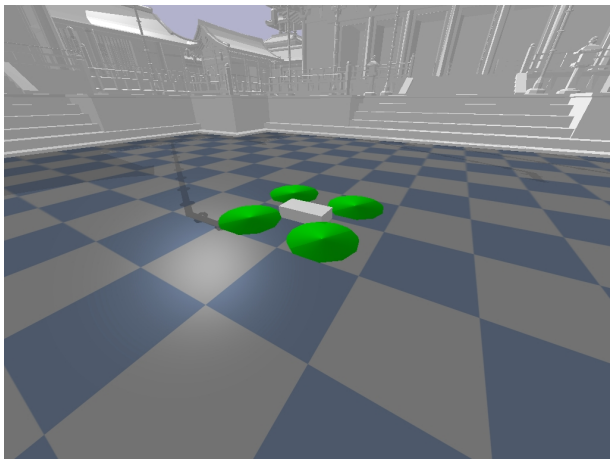

Figure 3.6: The base *QuadX* model within PyFlyt.

QuadX The *QuadX* UAV describes a multirotor UAV in the Quad-X configuration as described by ArduPilot [3] and PX4 [4]. It is constructed using four motors attached to four edges of a boring body. Visually, the *QuadX* models is shown in figure 3.6.

The QuadX UAV comes with various default control modes that users can build off using the native `register_controller` method. These control modes utilize a cascaded Proportional-Integral-Derivative (PID) architecture [Phang et al., 2012] to achieve increasingly higher levels of attitude control. The implemented controllers allows for the following setpoint definitions:

- Raw motor outputs
- Angular velocity control and normalized thrust output
- Angular position control and climb rate
- Linear velocity control and yaw rate
- Linear velocity control and yaw angle
- Position control and yaw rate
- Position control and yaw angle

In general, all quadrotor UAVs share a common mathematical model, albeit with varying physical and geometric parameters, such as mass, inertia, and size, as well as corresponding control parameters. The QuadX implementation in PyFlyt is generic and allows users to modify the parameters by passing a URDF and YAML file pair to the underlying QuadX class. This allows all mass, geometric, and control parameters of a QuadX UAV to be reconfigured easily.

As a reference, we provide two different QuadX models in PyFlyt. The first is the model of a Bitcraze Crazyflie 2.x, leveraging the extensive amounts of system identification work available [Landry et al., 2015, Luis and Ny, 2016, Förster, 2015]. The second model is that of a generic 1 kg quadrotor UAV with parameters mimicking a generic F450 quadrotor UAV [5], meant to serve as a starting point for users implementing models of their own without using a DAE mesh file.

[3] https://ardupilot.org/copter/docs/connect-escs-and-motors.html#quadcopter
[4] https://dev.px4.io/v1.10/en/airframes/airframe_reference.html#quadrotor-x
[5] https://www-v1.dji.com/flame-wheel-arf/feature.html

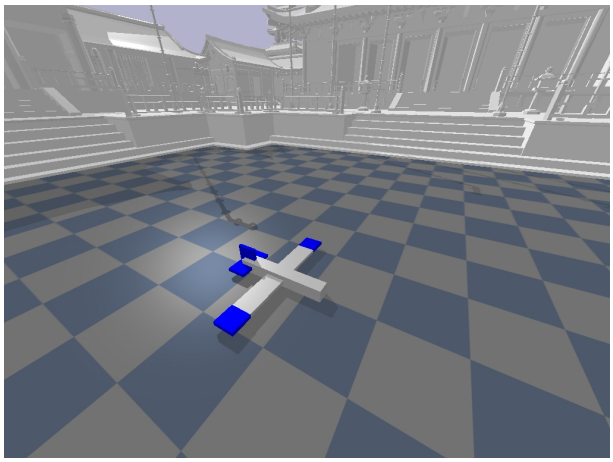

Figure 3.7: The base *Fixedwing* model within PyFlyt.

FIXEDWING The *Fixedwing* UAV in PyFlyt, shown in figure 3.7, is a tube-and-wing aircraft consisting of five lifting surface components—main wing, left aileron, right aileron, horizontal tail, vertical tail—and a motor. All of the lifting surfaces except the main wing are flapped, and they have parameters of an ideal 2D aerofoil. The ailerons have a maximum flap deflection of 30°, while the rudder and elevators are limited to 20°. It has a wingspan of 2.4 m, a nose to tail length of 1.55 m, with a 0.8 thrust to weight ratio. The total mass of the UAV is 2.35 kg, with 1 kg uniformly distributed in the fuselage, 0.5 kg in the main wing and the remaining mass distributed among the other lifting surface components.

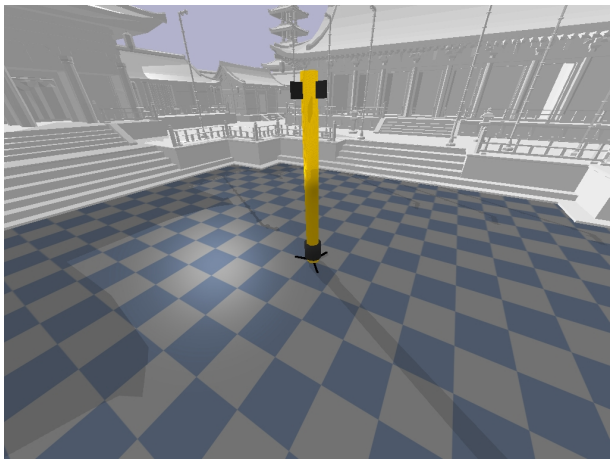

Figure 3.8: The base *Rocket* model within PyFlyt.

Rocket The *Rocket* UAV is a 1:10th scale rocket closely modeled after the design of SpaceX's Falcon 9 [6] v1.2 first stage and interstage. The model of the *Rocket* UAV is shown in figure 3.8. Mass and geometry properties were extracted from a datasheet by Space Launch Report [7]. It consists of booster, gimbal, and lifting surface components. The SpaceX Falcon 9 features grid fins [Washington and Miller, 1993] as supplementary control actuators. However, at the time of writing, limited studies on the aerodynamic properties of the Falcon 9's grid fins were made available to the public [Lee et al., 2020]. As a consequence to this shortcoming, the Rocket's control fin properties were simply approximated. In addition, the Bullet Physics Engine approximating the world as a flat 2D surface, it is not feasible to simulate the orbital mechanics of the Rocket within the PyFlyt framework.

3.5 PyFlyt Rail Environment

Having developed a flexible, configurable, and highly scriptable environment for simulating UAVs, the PyFlyt Rail Environment was developed to facilitate the training of RL agents for UAV navigation along a railway. A render from the environment itself is shown here in figure 3.9. The nature of using only semantically segmented images as inputs to training allows the simulation environment to forego any form of visual realism, relying only on semantic similarity between the simulation environment and the real world. The environment itself adheres to Gymnasium's environment creation Application Programming Interface (API), following the standard rules for designing RL training environments.

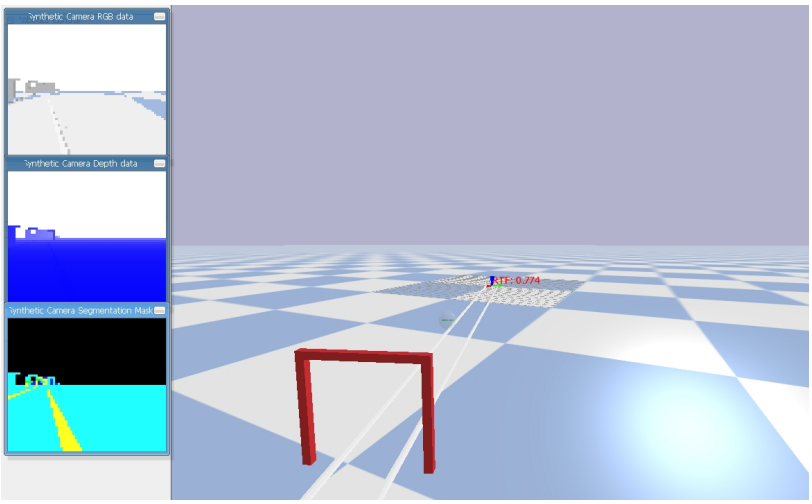

Figure 3.9: A render of the PyFlyt Rail Environment, showing the segmentation image on the bottom of the three views shown on the left of the figure.

[6] https://www.spacex.com/vehicles/falcon-9/

[7] https://web.archive.org/web/20170825204357/spacelaunchreport.com/falcon9ft.html#f9stglog

3.5.1 Environment Description

The PyFlyt Rail Environment consists of a single agent-controlled UAV navigating through a procedurally generated railway. The railway adheres to the standard dimensions of the UK's railway, with a width of approximately 1.4 meters, this mirrors what the agent will encounter in the real world. The agent interacts with the environment at a rate of 10 Hz, while the underlying flight controller operates at 60 Hz and physics runs at 240 Hz. The low agent interaction frequency strikes a balance between capturing meaningful state differences and preventing states from becoming excessively non-Markovian, a phenomena where adjacent states become too similar preventing the agent from being overwhelmed by simulation noise.

The railway in the environment is practically infinite, relying on a procedural generation mechanism to extend the railway based on the Euclidean distance between the UAV and the end of the railway. Various obstacles, including archways, tunnels, and trees, are strategically placed within 2 meters of the railway to simulate intrusive obstacles along the UAV's path. Archways and tunnels are positioned to necessitate the UAV's navigation beneath them, enhancing the complexity of the environment. Obstacles are spawned densely within the environment, forcing the agent to learn effective obstacle avoidance strategies. The intentional oversaturation of obstacles challenges the agent to navigate with precision and adaptability.

The environment itself has the following configurable parameters:

- Simulation time before truncation
- Agent control loop rate
- UAV spawn altitude
- Target flight altitude
- Camera resolution
- Camera field of view
- Camera tilt angle
- Maximum yaw rate
- Maximum climb rate

While it is possible to hard code each parameter and train an RL agent in the environment, we found much better success performing a form of domain randomization on these parameters to allow the resulting trained RL agent to be robust to environmental uncertainties in the real world. More precisely, the simulation time before truncation was been set as 60 seconds and camera resolution was set to 64×64, but every other parameter was uniformly sampled from the values shown in table 3.2.

Parameter	Value Range	Units
Agent control loop rate	{1, 2, 4, 10, 20}	Hz
UAV spawn altitude	[0.5, 1.5]	meters
Target flight altitude	[0.5, 1.5]	meters
Camera field of view	[60, 150]	degrees
Camera tilt angle	[-15, 30]	degrees downwards from horizon
Maximum yaw rate	[1.0, 2.0]	radians per second
Maximum climb rate	[0.5, 2.0]	meters per second

Table 3.2: Results of training the evidential model on the two datasets.

3.5.2 Observation and Action Spaces

The RL agent receives two types of observations from the environment structured as a Gymnasium `Dict` space:

- Semantically Segmented Images: These 64×64 images, captured from a forward-looking stabilized camera on the underbody of the UAV, depict the railway and obstacle pixels. Railway pixels correspond to pixels belonging to the rails on the railway (this excludes the sleepers themselves). Obstacle pixels correspond to pixels belonging to any object that does not represent the floor, UAV itself, or railway. The image has two channels — the first represents the pixels that belong to the railway, and the second channel represents pixels belonging to obstacles. An example image is shown in figure 3.10.
- UAV Attitude Information: This includes the unnormalized values for the UAV's altitude, three body-frame linear velocities, and the agent's past action. The choice to use a subset of the UAV attitude, specifically linear velocity, is a deliberate decision to balance information richness with learning efficiency. This choice, combined with a 10 Hz interaction frequency, facilitates smoother learning without overwhelming the agent with excessive state information.

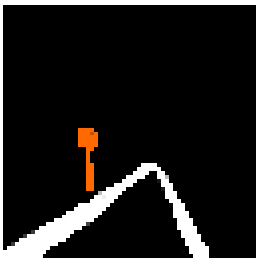

Figure 3.10: An example of the semantically segmented images that the RL agent sees as an input. Colors here are only for visualization as the actual agent receives a binary bitmap with 2 channels. White pixels here correspond to railway pixels, while orange pixels correspond to obstacles such as signs or overhead arches.

The action space comprises of a discrete braking action and three continuous actions (body-frame sideways velocity, yaw angular velocity, climb rate), $\boldsymbol{a}_t = \{\mathbb{I}_{\text{brake}}, v_y, v_\psi, v_z\}$. In the environment, the agent has no control over the body-frame forward velocity of the UAV outside of just braking. This ensures that the UAV continuously moves forward over the railway at a specified speed, reducing the need to optimize the reward structure for forward flight speed.

3.5.3 Reward Structure

The environment employs a dense reward structure, encouraging the agent to follow an ideal trajectory directly over the railway at a specified altitude. Deviations from this trajectory result in decreasing rewards. Collisions with obstacles or loss of sight of the railway incur significant penalties, leading to episode termination.

More specifically, at each time step t, the agent receives a reward $r_t = R(t)$:

$$R(t) = c_1 - c_2\mathbb{I}_{\text{collision}} - c_3\mathbb{I}_{\text{rail-lost}} - c_4\mathbb{I}_{\text{too-low}} - c_5\mathbb{I}_{\text{stop}} - c_6 d_t^2 - c_7\psi_t^2 - c_8 h_t^2 \quad (3.16)$$

Where c_i, $i \in \{1, 2, 3, 4, 5, 6, 7\}$ represent adjustable coefficients, and:

- $\mathbb{I}_{\text{collision}}$ is a binary indicator on whether a collision has happened
- $\mathbb{I}_{\text{rail-lost}}$ is a binary indicator when the railway is out of sight of the UAV
- $\mathbb{I}_{\text{too-low}}$ is a binary indicator when the UAV goes too low
- $\mathbb{I}_{\text{stop}}$ is a binary indicator when the UAV slows to a stop
- d_t is the distance between the UAV and the closest point on the railway
- ψ_t is the deviation between the UAV's heading and the railway's heading.
- h_t is the deviation between the UAV's current altitude and the specified ideal altitude.

Optimizing the ideal ratios of c_1 to c_8 is important. c_1 serves as a bonus for not terminating the environment, and in general must have a cumulative value more than all the dense penalties combined. Without c_1, the UAV will be incentivised to prematurely end the environment — either by collision or other means — in order to not accumulate excessive penalty. Meanwhile, c_5 must be set lower than all the other sparse penalties, as the penalty for stopping the UAV safely must not be higher than other more catastrophic penalties.

A number of experiments were done to identify good coefficients for this environment. The process involved first selecting a set of coefficients as a starting point (all 10.0 was selected as a starting point). A new model was then trained using these coefficients, its behaviour observed visually, and then tweaked by human intuition to steer the behaviour towards certain desired criteria, such as avoiding obstacles and maintaining view of the track. In our experiments, we found that a good set of numbers for c_1 to c_8 was:

- $c_1 = 10.0$
- $c_2 = 500.0$
- $c_3 = 500.0$
- $c_4 = 500.0$
- $c_5 = 100.0$
- $c_6 = 3.0$
- $c_7 = 3.0$
- $c_8 = 3.0$

3.6 Reinforcement Learning using Critic Confidence Guided Exploration (CCGE)

Selecting the appropriate RL algorithm for training RL agents depends on several factors. Key among those include deciding whether sample efficiency is a priority and what action space the target environment uses. For a simulation environment, PyFlyt is not a "fast" environment. For instance, Mujoco environments [Todorov et al., 2012] can operate at more than 4,000 times real time speed, facilitating the training of high speed, low sample efficiency RL algorithms like PPO. By contrast, due to the high fidelity physics simulation of PyFlyt which incorporates multiple low-level PID controllers in the simulation and the low agent-interaction frequency of PyFlyt Rail Environment, the simulation environment operates at about 30 times real time speed. Due to this shortcoming, high sample efficiency algorithms such as SAC are preferred.

One interesting caveat to training in the PyFlyt Rail Environment is that, in general, a suboptimal but not-completely-random policy is to simply stay above the railway at a fixed height while moving forward at the fixed specified speed. Such behaviour is easily implemented, especially in simulation. If RL algorithms are allowed to learn from this suboptimal policy and then learn obstacle avoidance by itself, dramatic improvements to training sample efficiency can be achieved. This thought process is the driving factor of an algorithm termed Critic Confidence Guided Exploration (CCGE) — a new algorithm developed as part of this research work.

3.6.1 Concept

RL seeks to learn a policy that maximizes the expected discounted future rewards for MDPs [Sutton and Barto, 2018]. Unlike supervised learning, which learns a function to map data to labels, RL involves an agent interacting with an environment to learn how to make decisions that optimize a reward signal. While RL has shown great promise in a wide range

of applications, it can be challenging to explore complex environments to find optimal policies. This is because complex environments can exhibit very sparse reward signals (such as the popular Montezuma's Revenge environment [Salimans and Chen, 2018]) or have many local minima (such as the extensively studied Mountain Car problem [Sutton, 1995]). Most popular RL methods employ stochastic actions to explore the environment, which can be time-consuming and lead to suboptimal solutions if the agent gets stuck in local minima, which can be true due to the curse of dimensionality or in complex environments where the reward signal is sparse. In addition, when multiple optimal behaviors are possible within the reward landscape, it is impossible to guide learning policies towards one particular optimal behaviour in the context of from-scratch RL.

One technique to circumvent this issue is by incorporating prior knowledge into the learning process via the use of an oracle policy — a policy that takes better-than-random actions when given a state observation. Such an oracle policy can be obtained from demonstration data through behavioral cloning [Bain and Sammut, 1995], pretrained using a different RL algorithm, or simply hard-coded. That said, it may not be clear how best to incorporate this oracle policy into the learning policy — and more crucially, when to wean the learning policy off the oracle policy. In this context, CCGE was developed as an algorithm that seeks to address these challenges by using uncertainty to decide when to use the oracle policy to guide exploration versus aiming to simply optimize a reward signal.

The proposed CCGE algorithm builds upon the actor critic framework for RL, where an actor learns a policy that interacts with the environment while the critic aims to learn the value function [Grondman et al., 2012]. Many of the most widely used methods in deep RL use the actor critic framework. For instance, they comprise four out of twelve algorithms in OpenAI Baselines Dhariwal et al. [2017] and four out of seven algorithms in Stable-Baselines3 [Raffin et al., 2019]. CCGE works by using the critic's prediction error or variance as a proxy for uncertainty. When uncertainty is high, the actor learns from the oracle policy to guide exploration, and when uncertainty is low, the actor aims to maximize the learned value function. Our intuition is that it is more beneficial to allow the learning policy to decide when to follow the oracle policy, as opposed to competing approaches that dictate when this switching happens.

3.6.2 Notation

We use the standard MDP definition [Sutton and Barto, 2018] defined by the tuple $\{\mathbf{S}, \mathbf{A}, \rho, r, \gamma\}$ where $\mathbf{S}$ and $\mathbf{A}$ represent the state and action spaces, $\rho(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ represent the state transition dynamics, $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ represents the reward function and $\gamma \in (0, 1)$ represents the discount factor. An agent interacts with the MDP according to the policy $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$, and during training, transition tuples of $\{\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}\}$ are stored in a replay buffer $\mathcal{D}$. The goal of RL is to find the optimal policy π^* that maximizes the cumulative discounted rewards $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$ for any state $s_0 \in \mathbf{S}$, where $\boldsymbol{a}_t \sim \pi(\boldsymbol{s}_t)$ and $\boldsymbol{s}_{t+1} \sim \rho(\boldsymbol{s}_t, \boldsymbol{a}_t)$.

Bellman Error Loss Function For Q-networks, the Bellman error is minimized via the loss function:

$$\mathcal{L}_Q(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbb{E}[l(Q^\pi_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - \gamma Q^\pi_\phi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}))] \tag{3.17}$$

where $l(\cdot)$ is usually the squared error loss and the expectation is taken over $\{\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, r_t\} \sim \mathcal{D}, \boldsymbol{a}_{t+1} \sim \pi(\cdot|\boldsymbol{s}_{t+1})$. In Deep Q Network (DQN), the optimal policy is defined as $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t) = \delta(\mathbf{a} - \arg\max_{\boldsymbol{a}_t} Q^\pi_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t))$, where $\delta(\cdot)$ is the Dirac function. This loss function in (3.17) will be used in derivations later in this work.

Soft Actor Critic The SAC algorithm is an entropy regularized actor critic algorithm introduced by Haarnoja et al. [2018a]. This work specifically refers to SAC-v2 [Haarnoja et al., 2018b], a slightly improved variant with twin delayed Q networks and automatic entropy tuning. SAC has a pair of models parameterized by θ and ϕ that represent the actor π_θ and the critic Q^π_ϕ. In a similar grain to DQN, the critic aims to learn the state-action value function via Q-learning. However, here, the critic is modified to include entropy regularization to promote exploration and stability for continuous action spaces. More concisely, the actor is driven to take actions that can afford imprecise actions taken in the environment (the actions commanded by the actor can have variance), thus avoiding cliff-walking scenarios [Sullivan et al., 2022]. The actor is then optimized by minimizing the following loss function via the critic:

$$\mathcal{L}_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = -\mathbb{E}_{\substack{\boldsymbol{s}_t \sim \mathcal{D} \\ \boldsymbol{a}_t \sim \pi_\theta}}[Q^\pi_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t)] \tag{3.18}$$

Note that we have dropped the entropy term for brevity as it is not relevant to our derivations, but it can be viewed in the formulation by Haarnoja et al. [2018b].

3.6.3 Epistemic Uncertainty

When optimizing a model, having a measure of distance to model optimality is useful to understand whether the model needs to be improved or whether the data exhibits too much variance. In other words, having a sense of model uncertainty versus data uncertainty can help identify reasons for incorrect predictions. In statistical learning theory, this quantity is referred to as *epistemic uncertainty* [Matthies, 2007] or model uncertainty, of which there are multiple formal quantifications. This quantity is different from *aleatoric uncertainty* or uncertainty in the data, which generally refers to the variability in the desired target prediction conditioned on a data point [Hüllermeier and Waegeman, 2021].

A naive approach to estimating epistemic uncertainty in machine learning models is to use the variance of model predictions as a proxy metric [Lakshminarayanan et al., 2017, Gal and Ghahramani, 2016]. Epistemic uncertainty estimation is widely known in Bayesian RL, which use Monte Carlo Dropout or model ensembles [Ghosh et al., 2021, Osband et al., 2018, Osband, 2016, Lütjens et al., 2019]. Other examples use Gaussian processes or deep kernel learning methods to explicitly store aleatoric uncertainty estimates within the replay buffer [Kuss and Rasmussen, 2003, Engel et al., 2005, Xuan et al., 2018] or distributional

models aiming to learn a distribution of returns [Bellemare et al., 2017, Dabney et al., 2018]. Such methods allow a direct capture of aleatoric uncertainty, allowing epistemic uncertainty to be estimated as a function of model variance.

Evidential regression [Amini et al., 2020] aims to estimate epistemic uncertainty by proposing evidential priors over the data likelihood function. This works very well for data with stationary distributions, notably in supervised learning [Charpentier et al., 2020, 2021, Malinin and Gales, 2018]. However, this technique does not extend to non-stationary distributions, a staple in RL.

More recently, Jain et al. [2021] modelled aleatoric and epistemic uncertainties by relating them to the expected prediction errors of the model. This relation is intuitive as it directly predicts how much improvement can be gained given more data and learning capacity. They further demonstrated how to use this prediction in a limited case of curiousity-driven RL, in a similar grain to Moerland et al. [2017], Nikolov et al. [2018]. We adopt the formalism for epistemic uncertainty presented by Jain et al. [2021] and present its notation here.

Consider a learned function f that tries to minimize the expected value of the loss $l(f(x) - y)$ where $y \sim P(\mathcal{Y}|x)$.

Definition 1 The *total uncertainty* $\mathcal{U}_f(x)$ of a function f at an input x, is defined as the expected loss $l(f(x) - y)$ under the conditional distribution $y \sim P(\mathcal{Y}|x)$.

$$\mathcal{U}_f(x) = \mathbb{E}_{y \sim P(\mathcal{Y}|x)}[l(f(x) - y)] \tag{3.19}$$

This expected loss stems from the random nature of the data $P(\mathcal{Y}|x)$ (aleatoric uncertainty) as well as prediction errors by the function due to insufficient knowledge (epistemic uncertainty).

Definition 2 A *Bayes optimal predictor* f^* is defined as the predictor f of sufficient capacity that minimizes $\mathcal{U}_f$ at every point x.

$$f^* = \arg\min_f \mathcal{U}_f(x) \tag{3.20}$$

Definition 3 The *aleatoric uncertainty* $\mathcal{A}(\mathcal{Y}|x)$ of some data $y \sim P(\mathcal{Y}|x)$ is defined as the irreducible uncertainty of a predictor. It can also be viewed as the total uncertainty of a Bayes optimal predictor.

$$\mathcal{A}(\mathcal{Y}|x) = \mathcal{U}_{f^*}(x) \tag{3.21}$$

Note that the aleatoric uncertainty is defined over the conditional data distribution, and is not conditioned on the estimator. By definition, $\mathcal{A}(\mathcal{Y}|x) \leq \mathcal{U}_f(x), \forall f \forall x$.

When $P(\mathcal{Y}|x)$ is Gaussian and $l(\cdot)$ is defined as the squared error loss, an optimal predictor predicts the mean of the conditional distribution, $f^*(x) = \mathbb{E}_{y \sim P(\mathcal{Y}|x)}[y]$. As a result, the total uncertainty of an optimal predictor is equivalent to the variance of the conditional

distribution. Thus, by extension, the aleatoric uncertainty of any predictor on Gaussian data trained using the squared error loss is equal to the variance of the data itself.

$$\begin{aligned}\mathcal{A}(\mathcal{Y}|x) &= \mathcal{U}_{f^*}(x) \\ &= \mathbb{E}_{y\sim P(\mathcal{Y}|x)}[l(y - \mathbb{E}_{y\sim P(\mathcal{Y}|x)}[y])] \\ &= \mathbb{E}_{y\sim P(\mathcal{Y}|x)}[l(y)] - l(\mathbb{E}_{y\sim P(\mathcal{Y}|x)}[y]) \\ &= \sigma^2_{|x}(\mathcal{Y})\end{aligned} \tag{3.22}$$

Note that we have chosen to write $\sigma^2(\mathcal{Y}|x)$ as $\sigma^2_{|x}(\mathcal{Y})$ for brevity.

Definition 5 The *epistemic uncertainty* $\mathcal{E}_f(x)$ is defined as the difference between total uncertainty and aleatoric uncertainty. This quantity will asymptotically approach zero as the amount of data goes to infinity for a predictor f with sufficient capacity.

$$\mathcal{E}_f(x) = \mathcal{U}_f(x) - \mathcal{A}(\mathcal{Y}|x) = \mathcal{U}_f(x) - \mathcal{U}_{f^*}(x) \tag{3.23}$$

The right hand side of (3.23) provides an intuitive view of epistemic uncertainty — it is the difference in performance between the current model and a model that perfectly reconstructs the conditional distribution of data.

3.6.4 Using Uncertainty for Guidance

Our method of improving sample efficiency with an oracle policy is loosely inspired by the Upper Confidence Bound (UCB) Bandit algorithm. Assume that we have a critic Q^π_ϕ and means of estimating its epistemic uncertainty $\mathcal{E}_\phi$. For any state and action $\{\boldsymbol{s}_t, \boldsymbol{a}_t\}$, we can assign an upper-bound to the true Q^π value:

$$Q^\pi_{UB}(\boldsymbol{s}_t, \boldsymbol{a}_t) = Q^\pi_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t) + \mathcal{E}_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t) \tag{3.24}$$

In a state $\boldsymbol{s}_t$, the potential improvement of following the oracle policy's suggested action $\bar{\boldsymbol{a}}_t$ can be then canonically written as:

$$\Delta(\boldsymbol{s}_t, \boldsymbol{a}_t, \bar{\boldsymbol{a}}_t) = Q^\pi_{UB}(\boldsymbol{s}_t, \bar{\boldsymbol{a}}_t) - Q^\pi_{UB}(\boldsymbol{s}_t, \boldsymbol{a}_t) \tag{3.25}$$

The term $Q^\pi_{UB}(\boldsymbol{s}_t, \bar{\boldsymbol{a}}_t)$ refers to the upper-bound Q-value when taking the action $\bar{\boldsymbol{a}}_t$ and then acting according to the learning policy π_θ. Now, the decision to learn from the oracle policy versus the critic can be made by defining the actor's loss function as either a supervisory signal $\mathcal{L}_{sup}$ or a reinforcement signal from (3.18) $\mathcal{L}_\pi$:

$$\mathcal{L}_{CCGE}(\boldsymbol{s}_t, \boldsymbol{a}_t, \bar{\boldsymbol{a}}_t) = \begin{cases} \mathcal{L}_{sup}(\boldsymbol{s}_t, \boldsymbol{a}_t, \bar{\boldsymbol{a}}_t), & \text{if } k \geq \lambda \\ \mathcal{L}_\pi(\boldsymbol{s}_t, \boldsymbol{a}_t), & \text{otherwise} \end{cases} \tag{3.26}$$

where k is computed with:

$$k = \frac{\Delta(\boldsymbol{s}_t, \boldsymbol{a}_t, \tilde{\boldsymbol{a}}_t)}{Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t)} \tag{3.27}$$

and λ is a constant which we term *confidence scale*. To put simply, the choice between imitating the oracle policy versus performing reinforcement learning is chosen according to the normalized potential improvement of doing the former versus the latter. In general, CCGE is flexible to choices of λ, and we show preliminary results of different values of λ in Appendix 3.6.9.

During training policy rollout, the learning policy may also choose to take the action from the oracle policy:

$$\boldsymbol{a}_t \leftarrow \begin{cases} \tilde{\boldsymbol{a}}_t, \text{if } k \geq \lambda \\ \boldsymbol{a}_t, \text{otherwise} \end{cases} \tag{3.28}$$

This allows the learning policy to quickly see the effect of the oracle policy's suggestions, helpful when the learning policy is completely uncertain about its environment at the beginning of training. We do not do this step at inference or policy evaluation.

3.6.5 Supervision Signal Definition

There are various ways to define the supervision loss $\mathcal{L}_{\text{sup}}$. For continuous action spaces, we can simply use the squared error loss:

$$\mathcal{L}_{\text{sup}}(\boldsymbol{s}_t, \boldsymbol{a}_t, \tilde{\boldsymbol{a}}_t) = \mathbb{E}_{\substack{\boldsymbol{a}_t \sim \pi_\theta(\cdot|\boldsymbol{s}_t) \\ \tilde{\boldsymbol{a}}_t \sim \tilde{\pi}(\cdot|\boldsymbol{s}_t)}}[\|\boldsymbol{a}_t - \tilde{\boldsymbol{a}}_t\|_2^2] \tag{3.29}$$

That said, any other loss function in similar contexts can be used, such as minimizing the negative log likelihood $-\log \pi_\theta(\tilde{\boldsymbol{a}}_t)$, L1 loss function $|\boldsymbol{a}_t - \tilde{\boldsymbol{a}}_t|$, or similar. Discrete action space models can instead utilize the cross entropy loss $-\tilde{\boldsymbol{a}}_t \log(\pi_\theta(\boldsymbol{a}_t))$.

3.6.6 Epistemic Uncertainty Metrics for a Critic

We present two metrics for quantifying the epistemic uncertainty of a critic — one based on Q-network ensembles in a similar grain to past work [Osband, 2016, Clements et al., 2019, Festor et al., 2021] which we call Implicit Epistemic Uncertainty, and one based on the DEUP technique [Jain et al., 2021] which we call Explicit Epistemic Uncertainty. We do not argue which metric provides a more holistic estimate of epistemic uncertainty, interested readers are instead directed to work by Charpentier et al. [2022] for a more detailed study. Instead, CCGE simply assumes a means of evaluating the epistemic uncertainty of the Q-value estimate given a state action pair, and these are two such examples.

Implicit Epistemic Uncertainty

We adopt the simplest form of estimating epistemic uncertainty in this regime using an ensemble of Q-networks. In SAC, an ensemble of $n(=2)$ Q-networks are used to tame overestimation bias [Hasselt, 2010, Haarnoja et al., 2018b]. For every state action pair, a set of Q-value estimates, $\{Q^{\pi}_{\phi_1}, Q^{\pi}_{\phi_2}, ...Q^{\pi}_{\phi_n}\}$ are obtained. We utilize the variance of these Q-values as a proxy metric for epistemic uncertainty $\mathcal{E}_{\phi} = \sigma^2(\{Q^{\pi}_{\phi_1}, Q^{\pi}_{\phi_2}, ...Q^{\pi}_{\phi_n}\})$ and set $Q^{\pi}_{\phi} = \min(Q^{1}_{\phi}, Q^{2}_{\phi}, ...Q^{n}_{\phi})$ as is done in the original SAC implementation to obtain k.

Explicit Epistemic Uncertainty

Adopting the framework for epistemic uncertainty from Jain et al. [2021], we derive an estimate for epistemic uncertainty based on the Bellman residual error. More formally, we first define single step epistemic uncertainty for a Q-value estimate as:

$$\begin{aligned}\delta_t(\boldsymbol{s}_t, \boldsymbol{a}_t) &= \mathcal{U}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - \mathcal{A}(Q^{\pi}_{\phi}|\{\boldsymbol{s}_t, \boldsymbol{a}_t\}) \\ &= l\big(\mathbb{E}[Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - \gamma Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})]\big)\end{aligned} \tag{3.30}$$

This is simply the expected Bellman residual error projected through the squared error loss. The exact derivation is available in Appendix 2.1.

Then, we propose using the root of the discounted sum of single step epistemic uncertainties as a measure for total epistemic uncertainty given a state and action pair:

$$\mathcal{E}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \left[\mathbb{E}_{\pi,\mathcal{D}}\left[\sum_{i=t}^{T} \gamma^{i-t}|\delta_i(\mathbf{s}_i, \mathbf{a}_i)|\right]\right]^{\frac{1}{2}} \tag{3.31}$$

The intuition here is that δ_t is a biased estimate of epistemic uncertainty. To obtain a more holistic view of epistemic uncertainty, we instead take the discounted sum of all single step epistemic uncertainty estimates as the true measure. In our experiments, we found that taking the root of this value results in more stable training dynamics.

The value of $\mathcal{E}_{\phi}$ can be estimated on an auxiliary output of the Q-network itself, and learned by minimizing the residual loss in (3.32):

$$\mathcal{L}_{\mathcal{E}}(\boldsymbol{s}_t, \boldsymbol{a}_t) = l(\mathcal{E}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - \left(\delta_t(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathcal{E}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})^2\right)^{\frac{1}{2}}) \tag{3.32}$$

The resulting loss function for the Q-value network for a single state action pair can then be derived from (3.17) and (3.32):

$$\mathcal{L}_{\mathcal{E},Q}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathcal{L}_{\mathcal{E}}(\boldsymbol{s}_t, \boldsymbol{a}_t) + \mathcal{L}_{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t) \tag{3.33}$$

3.6.7 Algorithm

We present the full pseudocode for CCGE in Algorithm 5. The standard parameters for an actor-critic reinforcement learning algorithm are first initialized. Then, we select a confidence scale that controls how confident the algorithm should be about the learning policy's actions. For most tasks, a value of 1 has been found to work well, while smaller values near 0.1 may result in better performance for harder exploration tasks. During environment rollout, actions are taken in the environment according to the oracle policy or learning policy depending on k. Similarly, during policy optimization, the learning policy either performs supervised learning against the oracle policy's suggested actions or standard reinforcement learning depending on k. Algorithm 5 illustrates the pseudocode for CCGE. The deviations of CCGE beyond the standard SAC algorithm is highlighted in blue text. `UpdateCritic` is the standard reinforcement learning value function update, usually done by minimizing (3.17). CCGE modifies this update to (3.33) using either Implicit Epistemic Uncertainty or Explicit Epistemic Uncertainty methods.

Algorithm 5 Critic Confidence Guided Exploration (CCGE)

Select discount factor γ, learning rates η_ϕ, η_π
Select size of Q-value network ensemble n
Select confidence scale $\lambda \geq 0$

Initialize parameter vectors θ, ϕ
Initialize actor and critic networks π_θ, Q_ϕ
Initialize or hardcode oracle $\bar{\pi}$

for number of episodes **do**
  Initialize $\boldsymbol{s}_t = \boldsymbol{s}_{t=0} \sim P(\cdot)$
  while env not done **do**
    Sample $\boldsymbol{a}_t$ from $\pi_\theta(\cdot|\boldsymbol{s}_t)$
    Sample $\bar{\boldsymbol{a}}_t$ from $\bar{\pi}(\cdot|\boldsymbol{s}_t)$
    Compute k using $Q_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t), Q_\phi(\boldsymbol{s}_t, \bar{\boldsymbol{a}}_t), \mathcal{E}(\boldsymbol{s}_t, \boldsymbol{a}_t), \mathcal{E}_\phi(\boldsymbol{s}_t, \bar{\boldsymbol{a}}_t)$
    Override $\boldsymbol{a}_t \leftarrow \bar{\boldsymbol{a}}_t$ if $k \geq \lambda$
    Sample $r_t, \boldsymbol{s}_{t+1}$ from $\rho(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t)$
    Store transition tuples $\{\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}, \bar{\boldsymbol{a}}_t\}$ in $\mathcal{D}$
  end while

  for $\{\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}, \bar{\boldsymbol{a}}_t\}$ **in** $\mathcal{D}$ **do**
    Update critic $Q_\phi \leftarrow$ `UpdateCritic`$(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, Q_\phi)$
    Update actor parameters $\theta \leftarrow \theta - \eta_\phi \nabla_\phi \mathcal{L}_{CCGE}$
  end for
end for

3.6.8 Experiments

We implement CCGE on SAC, specifically SACv2 [Haarnoja et al., 2018b], and then evaluate its behaviour and performance on a variety of environments against a range of other algorithms. We use environments from Gymnasium (formerly Gym) [Brockman et al., 2016], D4RL derived environments from Gymnasium Robotics [Fu et al., 2020], and waypoint environments from PyFlyt [Tai and Wong, 2023]. Following guidelines from Agarwal et al. [2021], we report 50% Interquartile Means (IQMs) with bootstrapped confidence intervals. For each configuration, we train for 1 million environment timesteps and aggregate results based on 50 seeds per configuration and calculate evaluation scores based on 100 rollouts every 10,000 timesteps. The results of our experiments are shown in figure 3.11 and figure 3.12 with all relevant hyperparameters and network setups recorded in Appendix 2.4, 2.5, and 2.6. The experiments are aimed at answering the following questions:

1. Does CCGE benefit most from the supervision signal or the guidance from actions taken by the oracle?
2. How does CCGE compare over the baseline algorithm with no supervision?
3. How important is the performance of the oracle policy towards the performance of CCGE?
4. Is CCGE sensitive to different methods of epistemic uncertainty estimation?
5. Can we bootstrap the oracle using the learning policy continuously?
6. How does CCGE perform on hard exploration tasks in comparison to other state of the art algorithms?

To answer questions 2, 3, and 4, we use the continuous control, dense reward, robotics tasks from the MuJoCo suite of Gymnasium environments [Brockman et al., 2016, Todorov et al., 2012]. More concisely, we use Hopper-v4, Walker2d-v4, HalfCheetah-v4 and Ant-v4. These environments are canonically similar to those used by Haarnoja et al. [2018b] and various other works. The resulting learning curves for these experiments are shown in figure 3.11. For each environment, we obtain two oracles — Oracle 1 and Oracle 2 — using SAC trained for 250×10^3 and 500×10^3 respectively. Both oracles have different final performances that are generally lower than what is obtainable using SAC trained to convergence. For each oracle, we evaluate the performance of CCGE using two separate methods of estimating epistemic uncertainty — the implicit and explicit methods detailed in Sections 3.6.6 and 3.6.6. This results in a total of four CCGE runs per environment, which we use to compare results against SAC. We label CCGE with the first oracle as `CCGE_1` and with the second, better performing oracle as `CCGE_2`. To denote between different forms of epistemic uncertainty estimation, we further append either `_Im` or `_Ex` to the algorithm names to denote usage of either the implicit or explicit forms of epistemic uncertainty estimation.

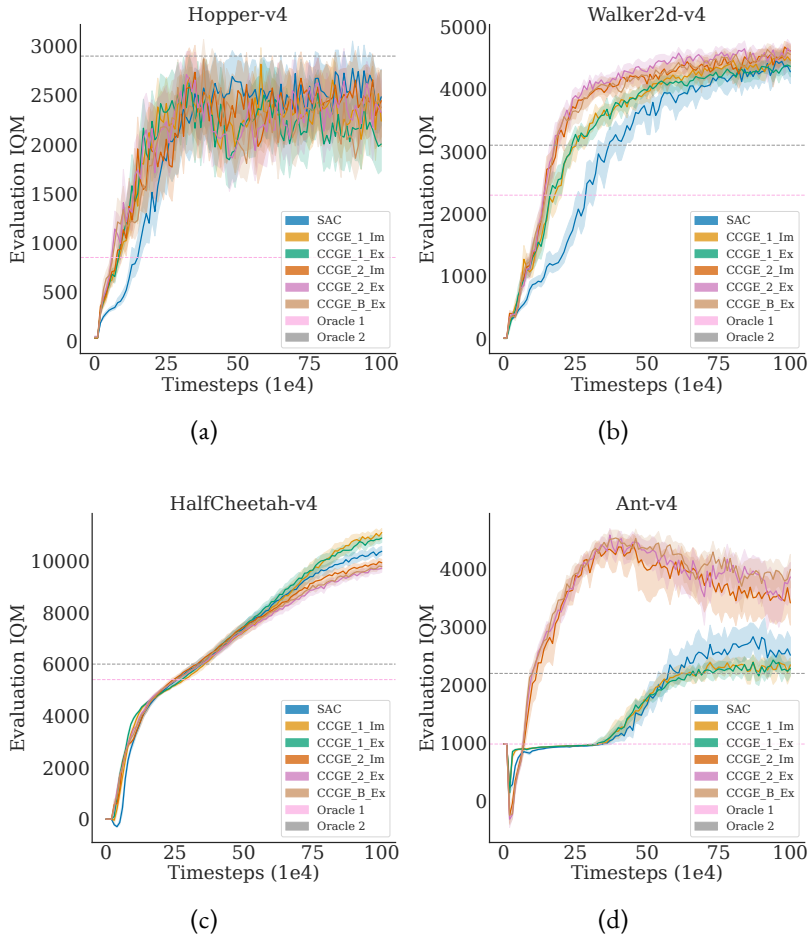

Figure 3.11: Learning curves of CCGE and SAC on Gym Mujoco environments. For the CCGE runs, the run names are written as `CCGE_{Oracle Number}_{Epistemic Uncertainty Estimation Type}`. The oracle policy labelled `B` denotes the bootstrapped oracle policy.

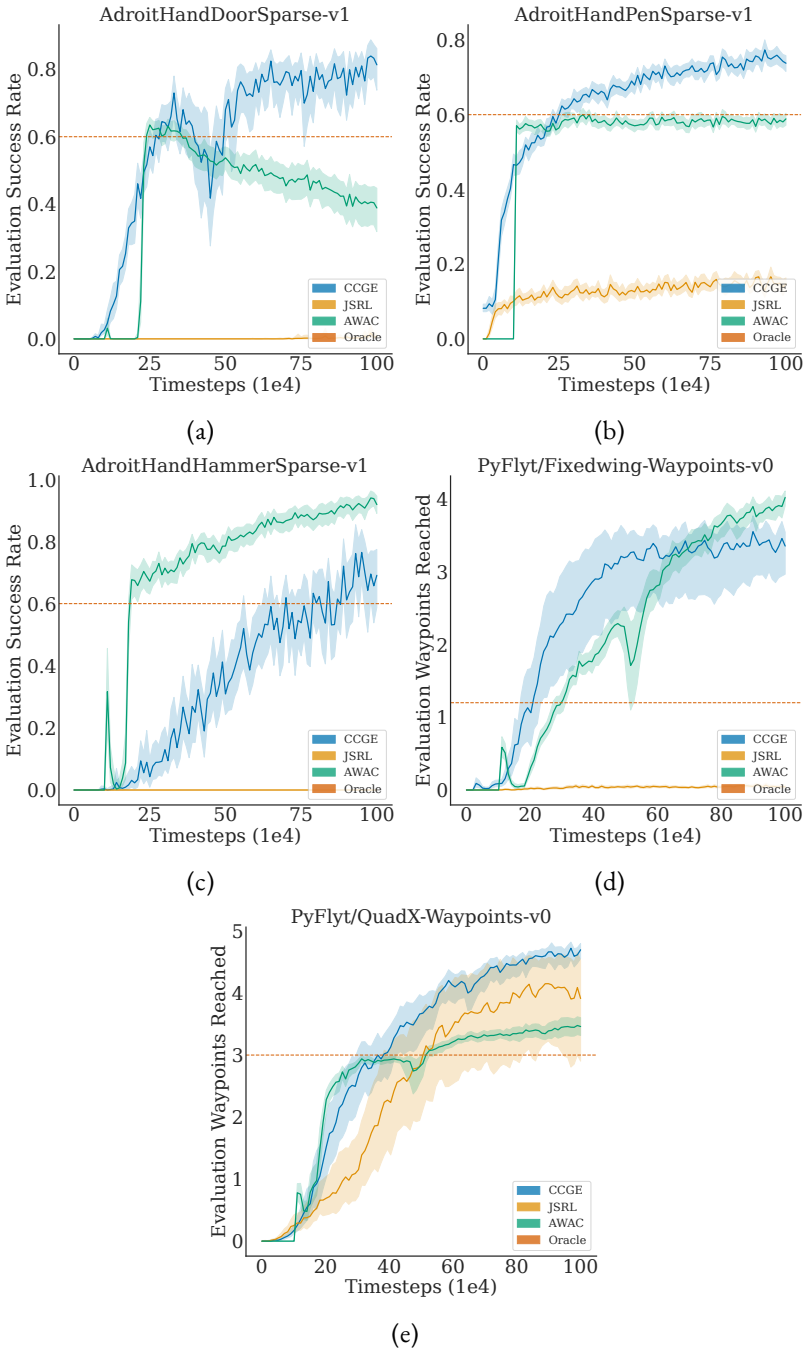

Figure 3.12: Learning curves of CCGE, AWAC, and JSRL on three Gymnasium Robotics and two PyFlyt environments.

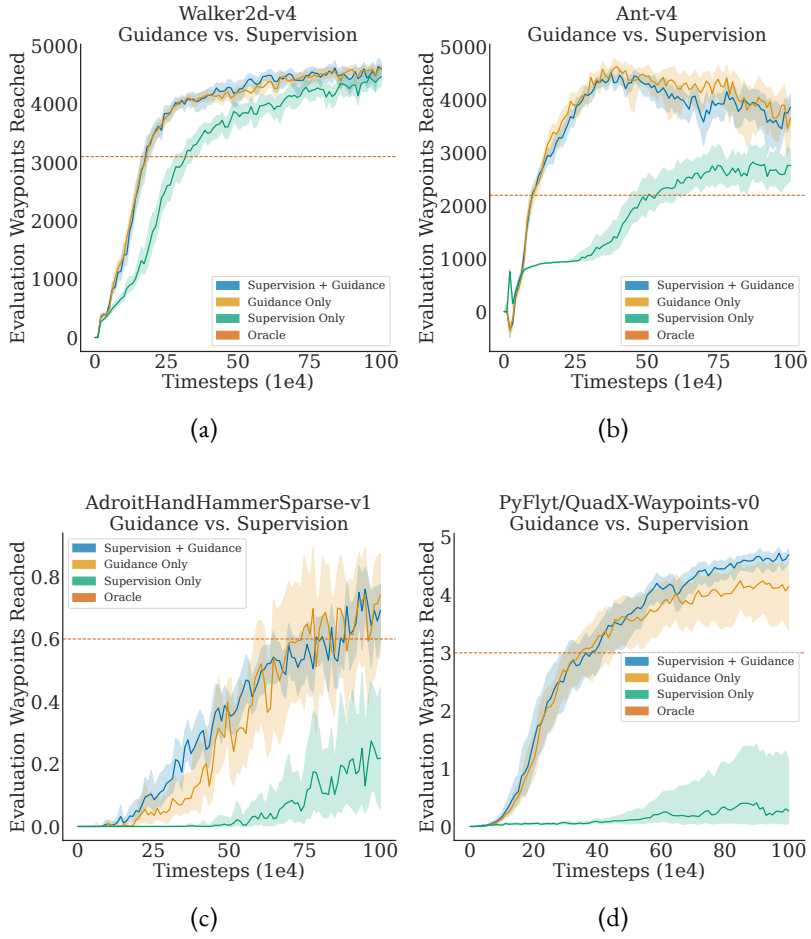

Figure 3.13: Learning curves of CCGE in various modes of supervision on four sparse reward environments.

CCGE Results

CCGE vs. no CCGE Compared against the baseline SAC algorithm, CCGE outperforms SAC in all cases using the right oracle policy. In some cases, CCGE significantly outperforms SAC by leveraging the oracle policy to escape local minima early on. The obvious example here is in the Ant-v4 environment in figure 3.11d. Here, SAC tends to learn a standing configuration very early on, achieving an evaluation score of about 1000. It takes approximately 300×10^3 more transitions before the algorithm achieves a stable walking policy that does not fall. Using the right oracle policy — in this case a mostly walking oracle with an evaluation score of 2100, CCGE learns a successful walking configuration in approximately 15% the number of transitions that it takes for SAC. The performance of CCGE in the Ant-v4 environment seems to degrade over time. One possible reason for this could be catastrophic forgetting due to the limited capacity replay buffer [Isele and Cosgun, 2018], and this is explored more in Appendix 2.3.

Oracle Performance Sensitivity Most reinforcement learning algorithms that bootstrap off imitation learning are improved by having higher quality data. CCGE is no exception to this dependency, as the change in performance from having different oracle policies directly dictates the learning performance of the algorithm. In particular, a different oracle policy for HalfCheetah-v4 results in CCGE either performing better or worse than SAC, this behaviour is shown in figure 3.11c. Similarly, when initialized with an oracle policy stuck in a local minima in Ant-v4 (learning curves starting with `CCGE_1` in figure 3.11d), CCGE causes the learning policy to learn slightly slower than SAC due to it repeatedly reverting to the suboptimal oracle policy when uncertain. Despite this, the learning policy escapes the local minima in about the same time as SAC. This suggests that while CCGE with a bad oracle policy can hamper learning performance, it does not limit the exploration rate of the learning policy in general.

Different Epistemic Uncertainty Estimation Techniques We use the same confidence scale $\lambda = 1.0$ for both implementations. The results from figure 3.11 show that while choice of epistemic uncertainty estimation technique does impact learning performance, the impact is far smaller than a change of oracle policy. More concretely, implicit epistemic uncertainty estimation works slightly better in Walker2d-v4 and HalfCheetah-v4, but performs worse in Ant-v4. This is shown in the learning curves ending with `Im` for implicit method and `Ex` for explicit method in figure 3.11b, figure 3.11c and figure 3.11d). As long as CCGE has access to a reasonable measure of epistemic uncertainty, CCGE would benefit much more from better performing oracle policies than better epistemic uncertainty estimation techniques.

Supervision or Guidance

Algorithmically, CCGE uses two techniques to incorporate the oracle policy — a supervision signal induced during the policy improvement phase, and guidance via taking the oracle's ac-

tions directly during policy rollout. We conduct a series of experiments using Walker-v4, Ant-v4, PyFlyt/QuadX-Waypoints-v0 and AdroitHandHammerSparse-v1 with either only guidance, only supervision, or both enabled to determine the effect that either component has on CCGE's performance. More concisely, guidance can be disabled by disabling (3.28), and supervision signal can be disabled by setting (3.29) to 0. The results are shown in figure 3.13. In all cases, using the supervision only variant of CCGE produces the worst performance, with performance similar to the base SAC algorithm in the case of Walker2d-v4 (figure 3.13a) and Ant-v4 (figure 3.13b). CCGE performance benefits most from oracle guidance, with the guidance only algorithm performing similarly to the full algorithm in all dense reward environments. For sparse reward environments, the supervision component in the full algorithm produces a minimal but non-negligible learning improvement in terms of final performance in PyFlyt/QuadX-Waypoints-v0 (figure 3.13d) and initial rate of improvement in AdroitHandHammerSparse-v1 (figure 3.13c).

Bootstrapped Oracle

One advantage of CCGE's formulation is that it does not make any assumptions about the oracle policy. In fact, it does not even require the state-conditioned distribution of actions from the oracle policy to be stationary as required by algorithms such as Implicit Q Learning (IQL) [Kostrikov et al., 2023]. Therefore, CCGE can function even when the oracle policy is continuously improving. To test this theory, we evaluated a variant of CCGE where the oracle policy's weights are updated to match the learning policy's weights whenever the learning policy's evaluation performance surpasses that of the oracle policy. We use CCGE with explicit epistemic uncertainty estimation, with the same MuJoCo suite setup as done previously and the oracle policy initialized using Oracle 2. The resulting runs are labelled as `CCGE_B_Ex` in figure 3.11. Surprisingly, we found that using a bootstrapped oracle in this scenario did not provide any meaningful improvement in performance.

Experiments on Hard Exploration Tasks

We evaluate the performance of CCGE on hard exploration tasks against Advantage Weighted Actor Critic (AWAC) and Jump Start Reinforcement Learning (JSRL) using an SAC backbone — two algorithms suitable for this setting of learning from prior data in conjunction with environment interaction which represent the state of the art at the time of writing. Their performances are evaluated on the AdroitHandDoorSparse-v1 (figure 3.12a), AdroitHandPenSparse-v1 (figure 3.12b) and AdroitHandHammerSparse-v1 (figure 3.12c) tasks from Gymnasium Robotics, as well as the Fixedwing-Waypoints-v0 (figure 3.12d) and QuadX-Waypoints-v0 (figure 3.12e) tasks from PyFlyt. Hard exploration tasks describe environments where rewards are flat everywhere except when a canonical goal has been achieved. For example, in AdroitHandDoorSparse-v1 from Gymnasium Robotics, the reward is -0.1 at every timestep and 10 whenever the goal of opening the door completely has been achieved. Similarly, in PyFlyt the reward is -0.1 at every timestep and 100 when the agent has reached a waypoint.

The oracles for the AdroidHand tasks were obtained using SAC trained in the dense reward setup until the evaluation performance reached 0.6, requiring about 200×10^3 to 400×10^3 environment steps in the dense reward setup. The oracle for Fixedwing-Waypoints-v0 was obtained similarly — using SAC until the agent reaches 3 waypoints during evaluation, which happened after approximately 300×10^3 environment steps. For QuadX-Waypoints-v0, a cascaded PID controller [Kada and Ghazzawi, 2011] which partially solves this environment is deployed as the oracle policy. The variant of CCGE here uses explicit epistemic uncertainty estimation.

When compared with other competitive methods on the Gymnasium Robotics and PyFlyt tasks, CCGE demonstrates competitive sample efficiency and final performance on most tasks. This implies that CCGE can effectively explore and learn from environments with sparse rewards, which can be challenging for traditional RL algorithms. Notably, CCGE excels on environments that require more exploration than incremental improvement, such as AdroitHandPenSparse-v1 (figure 3.12b), AdroitHandDoorSparse-v1 (figure 3.12a), and PyFlyt/QuadX-Waypoints-v0 (figure 3.12e). Its underlying SAC's maximum entropy paradigm allows for aggressive exploration, which enables CCGE to quickly surpass the performance of the oracle policy. This is in contrast to AWAC which struggles to perform much better than the oracle policy due to more conservative exploration. However, on environments with a more narrow range of optimal action sequences, such as AdroitHandHammerSparse-v1 (figure 3.12c) and Fixedwing-Waypoints-v0 (figure 3.12d), AWAC's more conservative policy updates result in better overall learning performance and final performance. Interestingly, JSRL fails to perform well in most tasks except one. This could be due to a few factors, such as its reliance on the underlying IQL backbone or the potential idea that simply starting from good starting states is insufficient in guaranteeing good performance.

3.6.9 Choosing Confidence Scale

CCGE introduces one hyperparameter — the confidence scale, λ — on top of the base RL algorithm. We perform a series of experiments to study the effect that this hyperparameter has on the learning behaviour of the algorithm. To do so, we perform 100 different runs with $\lambda \in [0, 5]$, and compute mean values over 200k timestep intervals. We plot the guidance ratio — the proportion of transitions where the learning policy requests guidance from the oracle policy — as well as the average evaluation performance for the PyFlyt/QuadX-Waypoints-v0 and Walker2d-v4 environments. The results are shown in figure 3.14.

The choice of environment for these sets of experiments, while not exhaustive, were chosen to study the effect of λ on CCGE in both a dense (figure 3.14a and figure 3.14b) and sparse reward environment (figure 3.14c and figure 3.14d). As expected, low values of λ result in a higher guidance ratios, and this is especially true early on in training (before 400k timesteps). At later stages in training, the effect of λ is almost nullified, likely due to the learning policy's performance surpassing that of the oracle policy in all states. In sparse reward environments,

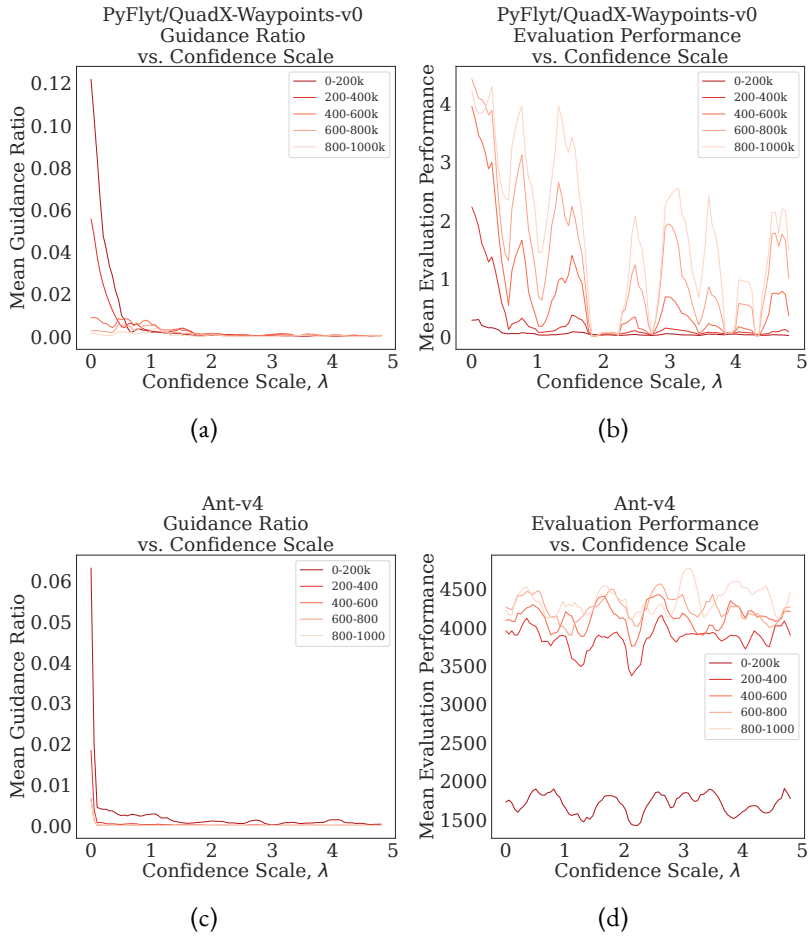

Figure 3.14: Mean guidance ratio and mean evaluation performance of CCGE on one sparse (`PyFlyt/QuadX-Waypoints-v0`) and one dense reward environment (`Ant-v4`). The results are averaged over 200k timestep intervals, each line corresponds to 100 different runs using a range of values for λ.

a high guidance ratio leads to better evaluation and learning performance; while in dense reward settings, this effect seems to not be present.

As an overarching recommendation, we suggest $\lambda \gtrsim 0$ (like 0.01) for more sparse reward settings, higher values seem to have no effect on performance for well-formed dense reward environments.

3.6.10 Summary

In this section, a new approach for RL called Critic Confidence Guided Exploration (CCGE) is introduced. This algorithm seeks to address the challenges of exploration during optimization. The approach bootstraps from an oracle policy and uses the critic's prediction error or variance as a proxy for uncertainty to determine when to learn from the oracle and when to optimize the learned value function. We empirically evaluate CCGE on several benchmark robotics tasks from Gymnasium, evaluating its learning behaviour under different oracle policies and different methods of measuring uncertainty. We further test CCGE's performance on various sparse reward environments from Gymnasium Robotics and PyFlyt and find that CCGE offers competitive sample efficiency and final performance on most tasks against other, well performing algorithms, such as Advantage Weighted Actor Critic (AWAC) and Jump Start Reinforcement Learning (JSRL). Notably, CCGE excels on environments that require more exploration than incremental improvement, while other methods perform better on tasks with a more narrow range of optimal action sequences.

3.7 CCGE on PyFlyt Rail Environment

The past three sections have introduced the PyFlyt UAV simulator, the follow up PyFlyt Rail Environment, and CCGE — a new algorithm for allowing RL algorithms to bootstrap off oracle policies using uncertainty as a proxy metric. In this section, we detail the design choices and procedure used to learn a viable policy within the PyFlyt Rail Environment using CCGE.

3.7.1 Suboptimal Oracle Policy

CCGE relies on having an oracle policy serving as a jumping off point for learning policies. This oracle policy can either be hard coded or learned. In our particular case, hard-coding an oracle policy provides the most straightforward method of obtaining a competent oracle policy. We term the hard-coded policy as the heuristic policy in this section of work. In particular, our heuristic policy aims to maintain the UAV directly over the railway and ignores the presence of all obstacles — developing a heuristic policy that has built in obstacle avoid-

ance is possible, but is also something that we can motivate the learning policy to figure out. Simply put, the heuristic policy follows the following pure-proportional control scheme:

$$\mathbb{I}_{\text{brake}} = 1 \tag{3.34}$$
$$v_y = -k_y d_y \tag{3.35}$$
$$v_\psi = k_\psi d_p si \tag{3.36}$$
$$v_z = k_z d_z \tag{3.37}$$
$$\tag{3.38}$$

Where k_* represents control gains, d_y represents the distance along the body-frame y-axis of the UAV to railway, v_ψ represents the difference in angle between the UAV's the railway at the closest point to the UAV and d_z represents the difference between the UAV's current altitude and the target altitude (1.5 meters by default). The various parameters, k_y, k_ψ, and k_z were all tuned by visual verification, and were set to $\{v_y, v_\psi, v_z\} = \{0.5, 0.25, 1.0\}$ respectively. Modelling the transfer function of the UAV under the influence of the low-level cascaded PID control scheme in order to find the control time constant is irrelevant to this piece of work, and hence is not conducted.

3.7.2 Model Architecture

CCGE's underlying SAC algorithm uses two models - an actor model which takes in observations and produces actions, and a critic model which takes in observations and actions and produces a predicted Q value. In our implementation, we use a single neural network for the actor, and a twin delayed Q-network for the critic. The critic consists of two networks of identical architectures with a delayed clone each, resulting in 4 networks for the critic and 1 network for the actor, totalling 5 networks. Since both networks use fundamentally the same observations, all networks consist of backbones of an identical architecture plus a small neural network prediction head. In the case of the actor, the prediction head takes in the state representation computed by the backbone and produces an action prediction. In the case of the critic, the prediction head takes in the state representation as well as the actor's action to produce a predicted Q value. A visual diagram of both models are shown in figure 3.15.

The CNNs utilize the same network architecture, shown in table 3.3. The actor and critic architectures differ by a single parameter, n_e, which is the embedding size. This number denote the layer sizes of all single-layer neural networks used for both models. All images and actions were normalized network-side to be in $[-1, 1]$ via a linear scaling. UAV attitude inputs were not normalized.

3.7.3 Learning Results

Here, we compare the performance of CCGE against SAC on the PyFlyt Rail Environment. Both algorithms were trained for 200×10^3 total environment steps, taking approximately 10 GPU hours per training session. 10 independent models were trained in the environment

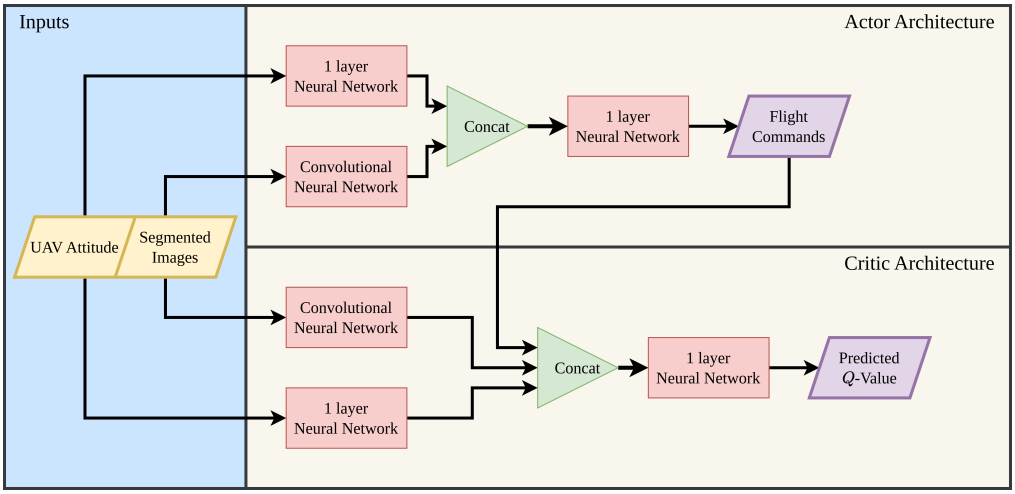

Figure 3.15: Architecture of both the actor and critic models for processing various inputs from the PyFlyt Rail Environment. Before both concatenation blocks, the CNNs perform a flatten operation on their outputs.

Table 3.3: CNN design parameters for both backbones in the CCGE algorithm for training a flying policy.

Layer #	Operation	Input Size	Output Size	Kernel Size	Pad Size	Pool Size	Activation
1	Convolution	2@64×64	16@32×32	3×3	1×1	2×2	ReLU
2	Convolution	16@32×32	32@16×16	3×3	1×1	2×2	ReLU
3	Convolution	32@16×16	64@8×8	3×3	1×1	2×2	ReLU
4	Convolution	64@8×8	n_e/4@4×4	3×3	1×1	2×2	ReLU

using a different set of initialization seed each. One evaluation step was run every 20,000 environment steps, where an evaluation score equates to the mean cumulative reward over 20 environment episodes. The IQM evaluation performance for each experiment is then plotted and shown in figure 3.16.

From the plots, CCGE is able to learn much faster than SAC, showing about double the evaluation performance before 5×10^3 at total environment time steps. Beyond that, CCGE demonstrates similar final performance to SAC, albeit with higher learning variance. Upon visual inspection of the policies, this is attributed to the learning behaviour of the underlying algorithm. Because CCGE starts from a suboptimal oracle policy, the final obtained policies tend to land in one of two camps — overly conservative leading to low evaluation performance by favouring braking more often, or overly aggressive, leading to very high evaluation scores but higher collision rates. Higher variance results in general may not be a great thing, but it is possible to play this to our advantage because the resulting policies will exhibit a large range of behaviours. For instance, the presence of many emergent policies means that decision makers can then decide the nature of policy that is desired for the task at hand.

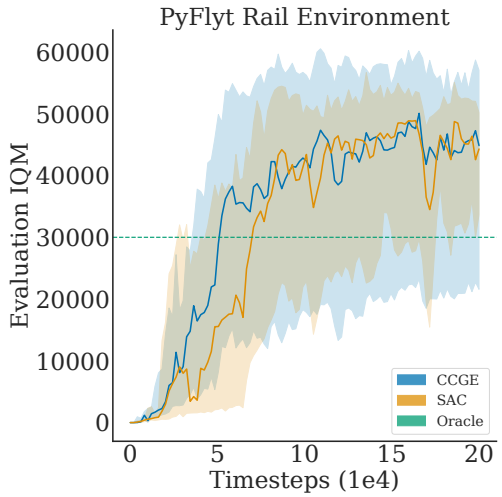

Figure 3.16: Results of training CCGE and SAC on the PyFlyt Rail Environment.

3.8 Image Segmentation Model

Part of the roadmap to deploying an algorithm that can fly a UAV along a railway is a computer vision model for taking real images and converting them to semantically segmented images. This section goes over the data collection scheme, model architecture, and learning results of applying the model. The work here was supported by BCIMO, who have graciously allowed us access to their testing railway at Dudley, United Kingdom.

Figure 3.17: Sample images when manually flying a UAV over BCIMO's test railway track in Dudley, United Kingdom.

3.8.1 Data Collection

For data collection, the UAV described in section 3.3 on page 55 was manually flown over the railway in a variety of angles and heights. The base idea is to generate a diverse set of images that simulate what the UAV may actually see in autonomous flight. Manual data collection and labelling cannot happen perpetually as we extend the domain that the UAV is supposed to fly over — one scalable way of handling this issue is through active learning, which is the idea of leveraging model uncertainty for the purpose of continuously scaling datasets automatically. Instead, manual data collection serves as a reasonable starting point before more scalable and autonomous methods can take over.

The images were gathered using the onboard Runcam 5 Orange camera at a resolution of 3840×2160 in RGB at a rate of 30 Frames per Second (FPS). A sample range of images that were collected are shown in figure 3.17. They were then labelled using SAM Tool, a tool developed as part of this section of work that provides a web interface to Meta's Segment Anything Model Kirillov et al. [2023], shown in figure 3.18. Every 10th frame from the recorded videos were added to the dataset, resulting in 1,515 images in total. During labelling, any entity that could pose as an obstacle to the UAV was labelled as an obstacle. This included signs, bridges, and outreaching tree branches.

3.8.2 Model Design

Design Criteria Computer vision, in general, is a very well established and successful field. Our goal is to design a model for segmenting the railways and obstacles out of an image so that the UAV may fly along the railway track while avoiding obstacles. For modern deep

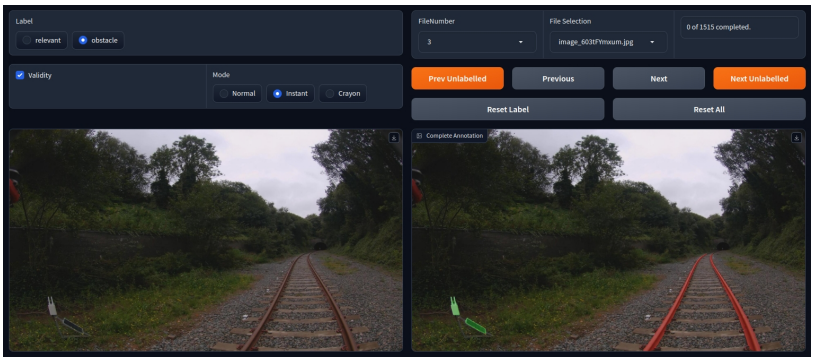

Figure 3.18: An overview of SAM Tool, a tool used to label the railway images gathered.

computer vision networks, this is generally not a difficult task. The challenge when applying it for a UAV flying along the railway is to design a model that can comfortably reside on the companion computer onboard the UAV alongside the RL agent. This is to allow the entire software stack to operate at a reasonable frequency of operation.

To decide on the ideal model architecture for the task at hand, the data was first split into an approximately 66.6:33.3 train:test split, resulting in 1,000 images for training and 500 images for testing. As an evaluation metric, a pixel-wise F1 score was utilized:

$$\mathrm{F1} = \frac{\mathrm{TP}}{\mathrm{TP} + 0.5(\mathrm{TP} + \mathrm{FP})} \tag{3.39}$$

Where TP stands for true-positives or the total amount of pixels correctly identified, and FP stands for false-positives or the total amount of pixels wrongly identified. The F1 score has a range of $[0, 1]$ and is the equivalent of the harmonic mean of precision and accuracy, where 0 corresponds to having 0% precision and accuracy, and 1 corresponds to having 100% precision and accuracy. Scores in the middle correspond to varying levels of precision and accuracy, e.g.: a score above 0.9 corresponds to approximately 90% precision and accuracy; 90% accuracy and 100% recall results in an F1 score of about 0.92.

Model Architecture The Residual Attention U-Net is a CNN with residual connections, with an attention layer embedded in the deepest layer, a block diagram of the architecture is shown in figure 3.19. This architecture is a natural departure from the U-Net architecture, and has seen extensive success in various computer vision fields for its high semantic depth and long range relational capabilities [Chen et al., 2020, Dong et al., 2022, Ni et al., 2019].

Very early on, we discovered that this architecture yielded very good results on the task at hand - achieving F1 scores of more than 0.92 in pixel-wise F1-score easily with less than 0.3M parameters and an inference speed of less than 30 milliseconds on an Intel i5-8265U CPU. Our focus then shifted from maximizing F1 score to trying to shrink model complexity while

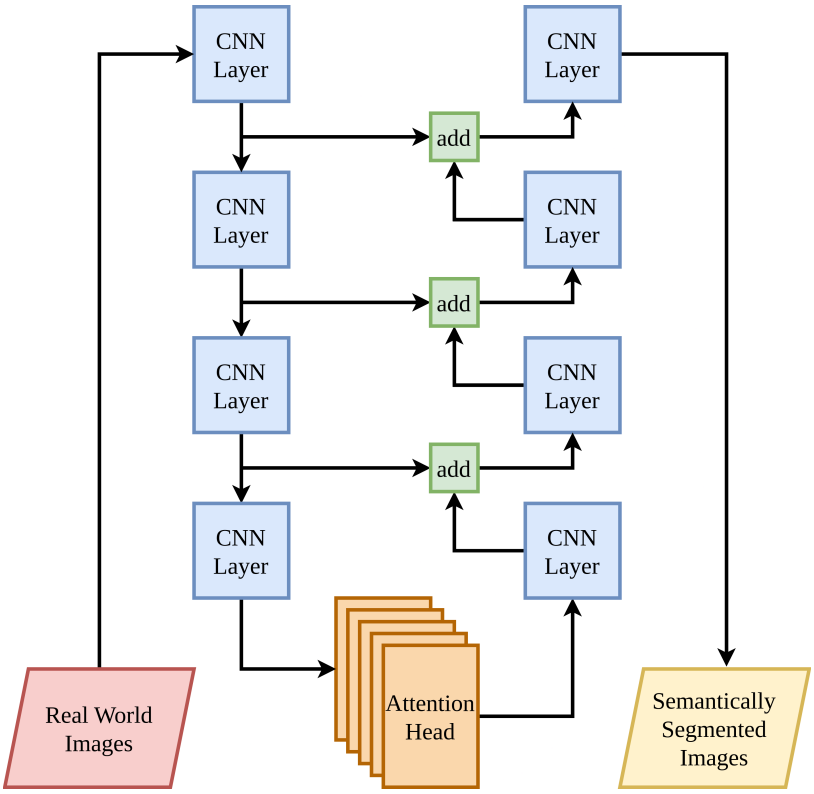

Figure 3.19: A block diagram overview of the Residual Attention U-Net. The green `add` blocks indicate an element-wise addition operator.

maintaining the same F1 score. In other words, how much can we prune down the model without hurting performance too much?

To begin this search, the following configuration was chosen as a starting point:

- CNN channel count, $[k_1, k_2, k_3, k_4]$: $[16, 32, 64, 128]$
- Attention embedding size: equal to number of channels in deepest CNN layer
- Number of attention heads, k_5: 4
- Number of attention modules in series, k_6: 3

> k_1 to k_4 were kept at numbers divisible by 2.

The model was trained for 300 epochs over the training set with a batch size of 32, sufficient for convergence of the F1 scores. The models were optimized with the AdamW [Kingma and Ba, 2015] optimizer using a maximum weight step of 0.001 with the `ReLU` activation function. Random flips and rotations were applied to the training images at a uniform rate, no random cropping was done — this would result in non-quantizable inputs to the model unless the cropped images were upscaled which causes the images to not be representative of what is seen in deployment. The pruning was done using a Bayesian optimization procedure which recursively reduces k_1 through k_6 from the starting point to obtain a decision boundary of parameters that satisfy the original criteria of a 0.92 F1 score. The Bayesian optimization procedure used follows the utility, expected improvement, and optimization functions described in (3.41), (3.42), and (3.8.2).

$$\mathbf{x} = [k_1, k_2, ...k_n] \tag{3.40}$$

$$f(\mathbf{x}) = \begin{cases} \sum \mathbf{x} & \text{if } F1 \geq 0.92 \\ \infty & \text{else} \end{cases} \tag{3.41}$$

$$EI(\mathbf{x}) = \mathbf{E}\left[\min\left(g(\mathbf{x}_{best}) - g(\mathbf{x}), \infty\right)\right] \tag{3.42}$$

$$\mathbf{x}_{next} = \arg\max_{\mathbf{x}} EI(\mathbf{x}) \tag{3.43}$$

Where f represents the utility function used, EI represents the expected improvement function and g represents the surrogate Gaussian process for modelling f. The top 10 models were then isolated, and evaluated for their inference time. The parameter set producing the lowest inference time was chosen as the optimized model. The optimization task was run over the following range of values:

- $k_1 \in [4, 8, 16]$
- $k_2 \in [4, 8, 16, 32]$
- $k_3 \in [4, 8, 16, 32, 64]$
- $k_4 \in [4, 8, 16, 32, 64, 128]$
- $k_5 \in [1, 2, 3, 4]$
- $k_6 \in [0, 1, 2, 3, 4]$

Final Model Architecture The results from the Bayesian optimization procedure are shown in figure 3.20. The finalized model architecture utilizes has a 4 millisecond inference loop time with the following design configuration:

- $k_1 = 8$
- $k_2 = 32$
- $k_3 = 4$
- $k_4 = 4$
- $k_5 = 4$
- $k_6 = 1$

3.9 Chapter Results and Lessons Learned

In this chapter, we detailed steps taken towards deploying an RL algorithm on a real UAV in the real world. The overall approach first involved the construction of a flexible and hackable UAV simulation environment named PyFlyt using the Bullet physics engine. The simulation tool enabled the construction of a railway track environment on which a UAV was able to fly in. Then, a novel reinforcement learning algorithm known as Critic Confidence Guided Exploration was conceived for bootstrapping off a suboptimal oracle policy was introduced. Many ablation tests were done on this algorithm in a multitude of environments to assess its performance, before it was used to train an agent to fly a UAV within a simulated environment of a railway track. A visual segmentation model was then trained to perform domain adaptation in the visual space to deploy the trained agent to the real world. The final trained image segmentation and RL model was deployed on the hardware setup specified in section 3.3 on page 55. The entire software stack comfortably operates on the chosen Nvidia Jetson Orin Nano Development Kit, with the segmentation and RL pipeline operating at more than 10 Hz consistently. The flight tests were done on BCIMO's experimental railway track, as eluded to in the beginning of the chapter. A total of 5 flight tests have been done,

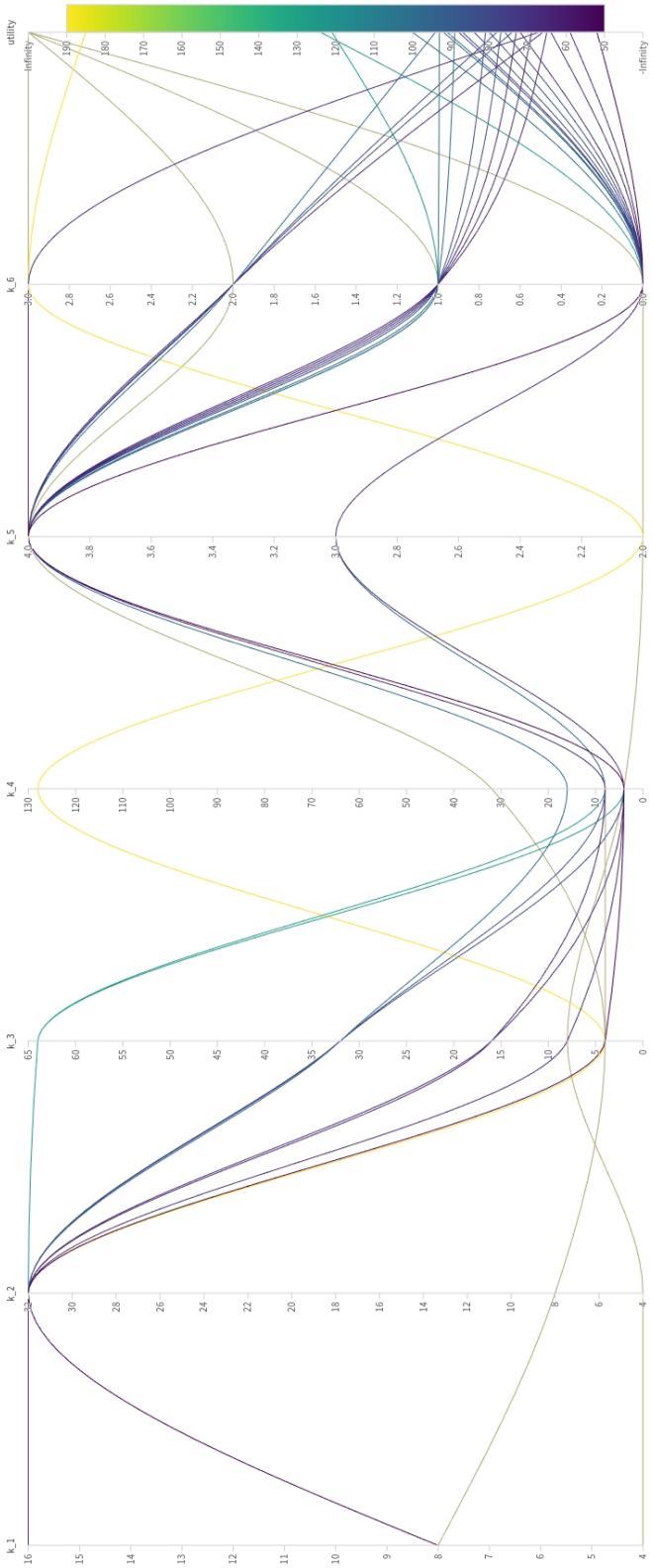

Figure 3.20: Results of Bayesian optimization over the various model parameters. Each spline represents one experiment, with the various axes it passes through indicating the hyperparameter values used for that experiment. The final axes at the top with name `utility` represents the final utility score achieved for the experiment utilizing that set of hyperparameters.

with an example of a run indicating the UAV successfully tracking the railway autonomously is shown in an unlisted YouTube video [8]. In multiple test scenarios on the section of track, the system operated successfully by correctly flying along the railway track without colliding with obstacles.

3.9.1 Noteworthy Observations

Trained RL agents have one-side biased actions An interesting trait of using RL on a task such as navigation along a railway is that the agent has asymmetrical actions. More concisely, when given a image of a railway viewed from the UAV $\mathbf{i}$ and a state $\mathbf{x}$, the RL agent may output action $\mathbf{a}$. Flipping the image left to right, represented here as $\bar{\mathbf{i}}$ would yield essentially another image of a railway. Additionally, it is also possible to flip all sideways components of the $\mathbf{x}$ as $\bar{\mathbf{x}}$. More concisely, this means we flip the sign of components such as UAV side velocity and UAV current yaw rate. One would assume that passing both $\bar{\mathbf{i}}$ and $\bar{\mathbf{x}}$ would yield exactly $\bar{\mathbf{a}}$ which is simply $\mathbf{a}$ with all sideways components having a flipped sign. However, because of the nature of RL, this tends to not be the case. As a result, the UAV has a tendency to cope with left turns better than right turns, or vice versa. At the time of this work, this is still an open research question.

3.9.2 Non-trivial Lessons

Contrary to what was depicted in this chapter, the path from simulation, novel RL algorithm training and characterization, hardware setup, and Sim2Real, was far from trivial. Many non-theoretical and far-from-obvious insights have been gathered throughout the course of its execution and these are detailed here for any future researchers hoping to carry on research in this direction.

Hardware robustness is the most important thing to software success. Regardless of the level of domain adaptation, control prowess, and algorithm sophistication, hardware robustness is the number one contribution to the success of a project of this scale. This is simply due to the ease of experiment implementation and iteration. When the hardware cannot be trusted, debugging software that controls the UAV's motions is virtually impossible. As an example, a software defined forward command may produce a backward motion despite the entire software stack being correctly implemented, simply because the UAV's underlying flight controller is poorly tuned resulting in the physical system not performing as expected. In this scenario, one may opt to flip a sign within software as an easy way of fixing what was perceived as a bug, thus introducing a new bug that wasn't there in the first place. This was evidently the limiting factor to our success. Another instance experienced during this project is when the UAV's laser altimeter and compass are externally influenced by the presence of magnetic and metallic elements on account of the railway tracks. Flying in this configuration results in a case of confusion on whether the RL agent is giving incorrect

[8] https://youtu.be/tGpJc04ecKI

commands, or whether the UAV is perceiving its state wrongly. Ultimately, we found that reducing the action space down to a single degree of freedom in the real world (yaw rate control) yielded the most reassuring results, simply because everything else could be debugged easier.

Deploying an RL algorithm from simulation to the real world will most likely result in an underdamped system. For robotics tasks, an unintended consequence of RL is that trained agents tend to always achieve perfectly damped response. When this is taken to the real world, latency, varying loop rates, and environmental disturbance cause there to be cases where the overall system time constant is much higher. The result of increasing the system time constant on a tuned control system is very well established — it leads to an underdamped, oscillating system. This is the exact phenomena observed here, resulting in undesirable and even dangerous execution. The recommendation is to therefore lower all gains of a trained agent in the real world. One example of this is to reduce the action space scaling — reducing the maximum yaw rate of the agent from 3 radians per second to 1 radian per second. This is easily implemented as a scalar after the output of the neural network.

4 Detection of Maintenance Needs on Railways

In the previous chapter, the problem of UAV navigation down a railway corridor was phrased as a reinforcement learning problem. Then, we introduced a new reinforcement learning problem termed Critic Confidence Guided Exploration which pushes the state of the art by allowing reinforcement learning agents to learn from oracle policies. We further introduced a new UAV simulator specifically catered for machine learning workflows termed PyFlyt. PyFlyt has a completely Python-driven user experience and is built to be reconfigured for specific use cases. A railway corridor environment was then built in PyFlyt, where Critic Confidence Guided Exploration was used to train an agent to fly a simulated UAV down this simulated railway corridor. Domain randomization and adaptation was then done to adapt this agent to function in the real world, successfully demonstrating autonomous flight using reinforcement learning at BCIMO's test track in Dudley, UK. In this chapter, we study feasibility and challenges when attempting to perform maintenance needs detection from such a UAV platform. Then, we propose several methods in which computer vision can be utilized to perform maintenance needs detection using classical supervised learning, before exploring concepts surrounding anomaly detection using uncertainty quantification. We run our experiments on a variety of datasets, and demonstrate that, for the task of railway monitoring, regularity in patterns is paramount for detecting maintenance needs.

4.1 Core Challenges

Many of the core challenges for performing maintenance needs detection from data gathered onboard a UAV are not immediately obvious. First and foremost, the main form of data gathered onboard a UAV platform is visual data in the form of camera-based images. Second, data gathered along a railway corridor contains a high amount of variance due to varying time of day, weather (whether it has rained prior to data collection or not), and surrounding scenery. This alone should generally not prove to be a challenge for deep learning models as long as a sufficient amount of data is used. However, to compound on this factor, while images gathered from a UAV platform are useful for providing a variety of view angles, introduces additional noise to the process due to that very fact. In addition, the ability to fly at varying heights causes similar components to be captured at varying scales, increasing the scale of training data required to develop reasonable deep learning models.

4.2 Approach

I have established that there are a plethora of maintenance needs that can be detected along railway corridors. I have also established that there are two different methods of detecting anomalies along railway corridors — classic direct detection via supervised learning and anomaly detection via mostly unsupervised learning methods. We aim to explore both directions in this work by looking at the problem through two different angles. The first is assuming that the UAV is able to fly fixed paths along the railway corridor, generating highly structured data over the railway in a variety of lighting and environment scenarios but mostly from a fixed viewpoint. In this instance, performing the actual UAV flights is a trivial matter — simply fly the UAV at a fixed height following waypoints that form a railway network [Bull, 2021]. Under this assumption, maintenance needs detection via classical supervised learning is possible by exploiting the inherent structure in the railway.

However, there is a possibility that flights are done with ill-defined flight envelopes, leading to images that are gathered with a variety of viewpoints. Here, we aim to explore using anomaly detection using uncertainty models instead. The goal is to collect a dataset of railway corridor images gathered from a UAV — a task not done before as of the time of writing. Then, using this gathered dataset, we aim to explore various techniques for anomaly detection.

4.3 Datasets

Here, various datasets relevant to railway tasks are described. In particular, the focus is on datasets primarily with image data, as that is the domain we aim to address. There currently exist several publicly available datasets, each defined for a different task, typically with varying viewpoints, lighting conditions, and camera intrinsic and extrinsic parameters. The high variance betwen each dataset makes it difficult for any one model to leverage all datasets to tackle a single task. Two additional datasets that have been gathered during the time in this research work will also be introduced here. One such dataset has been gathered using existing railway infrastructure, while the other represents a novel dataset gathered from the viewpoint of a UAV.

4.3.1 Publicly Available

RailSem19 [Zendel et al., 2019] RailSem19 presents 8,500 unique images of a railway track taken from the ego-perspective of a railway vehicle. Each image contains semantic segmentation annotations that are compatible with many Cityscapes-compatible road labels — Cityscapes being a class of datasets meant for urban environment scene understanding tasks. This compatibility means that RailSem19 doesn't contain labels pertaining to railway components, and instead uses the same labels as a Cityscapes-style dataset, segmenting entities such as greenery, people, road, railway, and signage.

RailEnV-PASMVS [Broekman and Gräbe, 2021] This dataset has been described as a perfectly accurate, synthetic, path-traced dataset featuring a virtual railway environment for multi-view stereopsis (PASMVS) training and reconstruction applications. It consists of images of a digitally rendered railway across 40 different scenes, totalling 79,800 images paired with ground truth depth maps, intrinsic and extrinsic camera parameters, pseudo-geolocation metadata, and binary segmentation masks of all relevant track components. Each scene is rendered from a set of 3 cameras, positioned for optimal 3D reconstruction of the rail profile. This dataset aims to serve as a resource for performing 3D railway reconstruction through metrology techniques.

Rail-DB [Li and Peng, 2022] Rail-DB is a dataset of 7432 pairs of images and annotations of rail lines on a railway corridor. The images are collected from different situations in lighting, road structures, and views. Similarly to RailSem19, the images are collected from the ego-perspective of a railway vehicle. Each image contains exactly 4 railway lines.

4.3.2 Obtained During This Work

New Measurement Train Captured Railway Image Dataset This is a dataset of nadir view railway images, captured from a NMT (shown in figure 4.4). The NMT is a 5-carriage train equipped with a plethora of sensors, one of those sensors is a downward facing camera intended for capturing high definition images of a rail, its clips, and some section of sleepers. The data itself was provided by infrastructure company Balfour Beatty during a collaborative partnership during the duration of the research. The dataset consists of 997 images, some examples of the gathered images are shown in figure 4.2. The fasteners in the images include fasteners which are occluded, damaged or missing. Furthermore, the fasteners are exposed under various lighting conditions due to time of day; no artificial illumination is used. The images also contain fasteners of various types - PR clip, E clip, Fast clip, C clip, and J clip. Finally, the images are solely black and white, making detection harder by removing colour information. We refer to this dataset as NMTFastenerSet2020.

UAV-Captured Railway Image Dataset Outside of performing maintenance detection on structured camera data, it is important to visit the idea of maintenance needs detection using information gathered from a flying UAV — the goal of this research work. To address this challenge, a dataset of railway images gathered from a UAV platform was created by manually flying the UAV platform described in section 3.3 on page 55 along the railway corridor owned by BCIMO. We address this dataset as UAVRailSet2023, and as of the time of writing, such a dataset is not known to us to be available publicly.

The images were gathered by flying the UAV in a Stabilized flight mode, with the gimballed camera facing anywhere from 70 to 90 degrees downwards from the horizon. Two separate flights were conducted across two separate days with fairly favourable weather conditions. In total, approximately 30 minutes of flight footage was gathered at a 3840×2160 resolution, from which one frame was gathered every 2 seconds, resulting in 930 images. These images

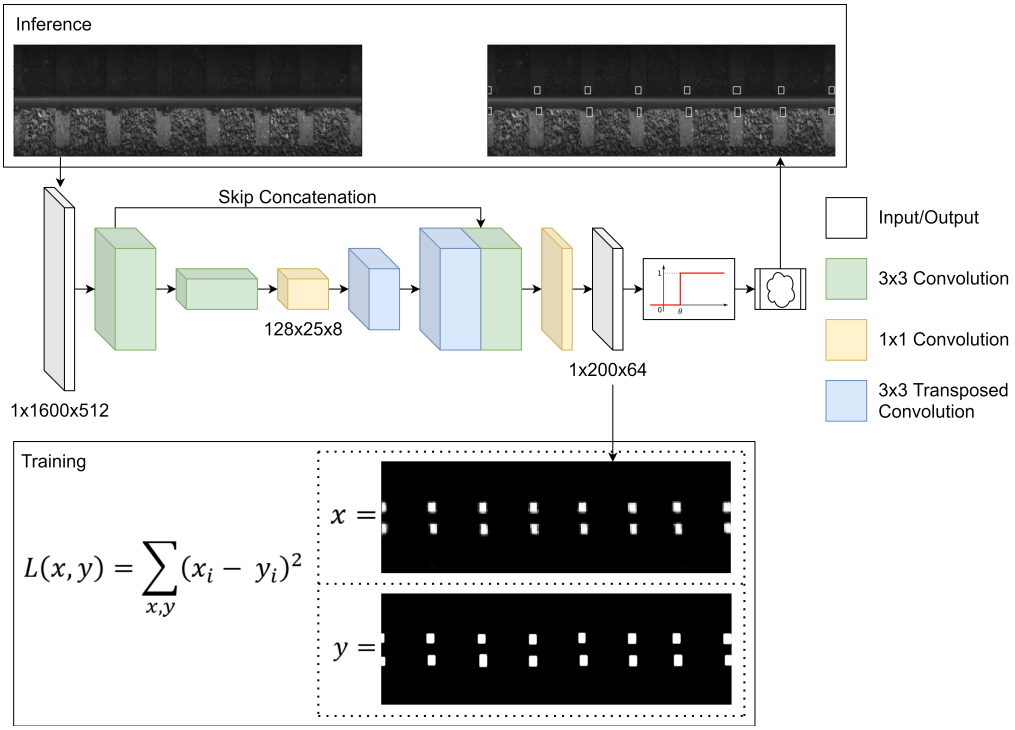

Figure 4.1: Top level view of FasteNet. During training, the network trains against segmentation masks. During inference, the network outputs are thresholded before a contour finding algorithm is applied to find individual fasteners.

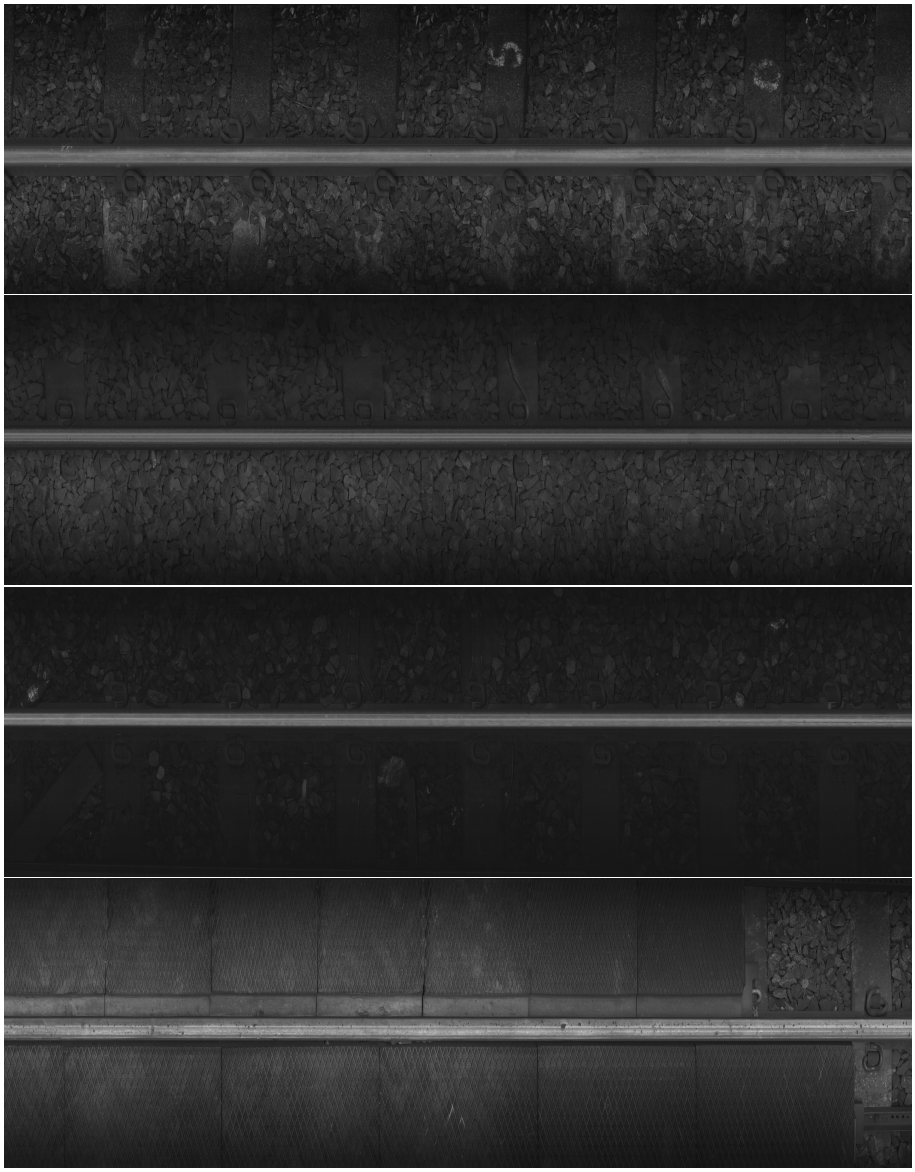

Figure 4.2: Sample images gathered using the NMT. Supplied by Balfour Beatty.

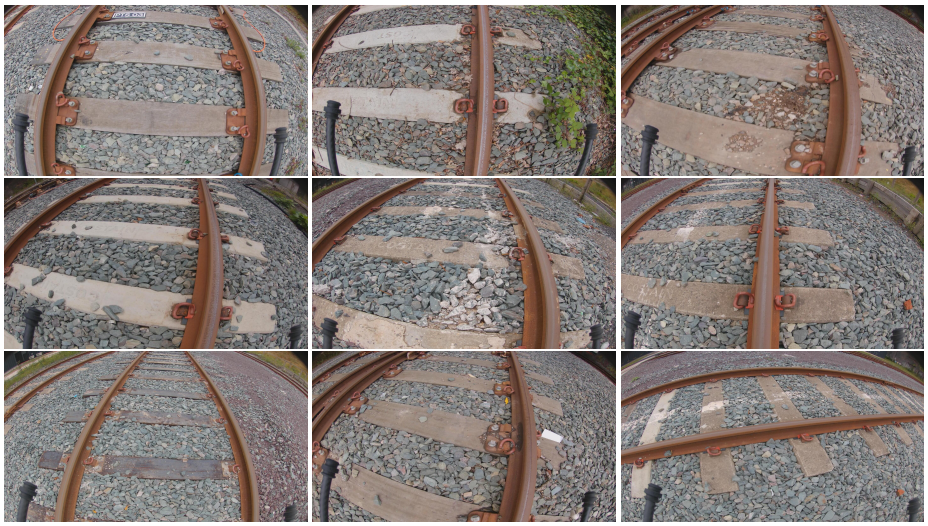

Figure 4.3: Example images from UAVRailSet2023.

were then hand labelled using the SAMTool, described earlier in section 3.8.1 on page 92. The labels that were chosen were rail, sleeper, clip and anomalies which encompasses foreign debris and greenery over the operational sections of the image. Examples of the gathered images are shown in figure 4.3.

The goal of UAVRailSet2023 was to gather a collection of images forming a dataset of highly unstructured image data for a railway corridor. As opposed to the datasets used in earlier experiments, these images were not gathered from a fixed viewpoint, so many of the exploits used in earlier experiments such as counting number of fasteners per image and polynomial regression for sunkink detection are not applicable. In addition, lens distortion and the variety of heights and angles make it harder for any structured analysis to be done. The goal of this dataset is to motivate the study of anomaly detection given arbitrary viewpoints using deep networks.

4.4 Computer Vision under Ideal Angles

We use the term "ideal angles" to denote cases where the images of the railway come from a consistent viewpoint. For instance, images of the railway gathered from the consistent ego-driver viewpoint of a railway carriage, or overhead views of the railway from a fixed height and offset from the railway. In these cases, we can exploit the structure of the image to identify any issues with regards to the railway corridor itself. Here, we propose two algorithms for detecting maintenance needs on railway tracks. The first is on fastener detection, and the second is on identifying sun kinks.

Figure 4.4: New Measurement Train operated in the United Kingdom.

4.4.1 Fastener Detection

As a starting point, we aimed to validate whether deep learning models are suitable for maintenance needs detection. The scope of the project was simple, we aimed to count the number of fasteners in an image using the NMTFastenerSet2020 dataset, described in section 4.3.2 on page 103. The act of being able to count fasteners enables the possibility of identifying missing fasteners. The guiding principle is that there are an even number of fasteners per length of railway track. This was not a novel task — past examples include the Viola-Jones algorithm [Xia et al., 2010] and symmetry-based pyramid histogram of oriented gradients (PHOG) algorithm [Liu et al., 2015]. Gibert et al. [2015] used support vector machines (SVMs) to perform fastener and defect detection. The work was also extended to fully convolutional neural networks [Gibert et al., 2016]. Much later on, Song et al. [2019] utilized a ResNet18 [He et al., 2016] backbone network on top of Faster-RCNN [Ren et al., 2015]. Their implementation achieved a recall and precision of more than 99% but was capped at 35 FPS on an Nvidia Titan X for 210 fasteners detected per second. Wang et al. [2020c] proposed using the popular YOLOv2 and YOLOv3 architectures to perform fastener component detection. In our case, we aimed for the network to perform fastener detection in a completely learned manner, and to do it utilizing modern architectures to achieve much faster fastener detection rates. The lessons here will be used in downstream research.

FasteNet Our solution to this problem is FasteNet - a purely convolutional architecture that uses contour finding to localize fasteners in an image. figure 4.1 provides a top level overview of FasteNet's architecture, as well as training and inference regimes. The input to the network is a black-and-white image defined here as $w_i \in \mathbb{R}^{1\times m\times n}$ where $m \hookleftarrow 64|m$ and $n \hookleftarrow 64|n$. It outputs a saliency map of $x_i \in \mathbb{R}^{1\times \frac{m}{8}\times \frac{n}{8}}$. The backbone network of FasteNet has just-enough layers to provide it with a roughly 64×64 ERF. This is a slightly hand-wavy method of calculating the ERF; in reality, the ERF more closely represents a Gaussian distribution on the true receptive field Araujo et al. [2019]. This ERF is roughly 1.5× the size of a fastener in the image. However, using only convolutional and pooling layers to achieve this ERF would translate to a very poor output resolution of 1/64th the input resolution. Many implementations of object detection work well with low output resolutions by using multiple bounding boxes per spatial element, and then suppressing repeat predictions

Table 4.1: Network architecture of FasteNet. The outputs from layer 3 are concatenated channel wise to outputs of layer 10 to form the input to layer 11.

Layer #	Operation	Input Size	Output Size	Kernel Size	Pad Size	Pool Size	Activation
1	Convolution	1@ 1600×512	32@ 800×256	3×3	1×1	2×2	Leaky ReLU
2	Convolution	32@ 800×256	32@ 400×128	3×3	1×1	2×2	Leaky ReLU
3	Convolution	32@ 400×128	64@ 200×64	3×3	1×1	2×2	Leaky ReLU
4	Convolution	64@ 200×64	64@ 100×32	3×3	1×1	2×2	Leaky ReLU
5	Convolution	64@ 100×32	128@ 50×16	3×3	1×1	2×2	Leaky ReLU
6	Convolution	128@ 50×16	128@ 25×8	3×3	1×1	2×2	Leaky ReLU
7	Convolution	128@ 25×8	64@ 25×8	1×1	-	-	Leaky ReLU
8	Transposed convolution	64@ 25×8	64@ 50×16	4×4	1×1	-	Leaky ReLU
9	Transposed convolution	64@ 50×16	64@ 100×32	4×4	1×1	-	Leaky ReLU
10	Transposed convolution	64@ 100×32	64@ 200×64	4×4	1×1	-	Leaky ReLU
11	Convolution	128@ 200×64	1@ 200×64	1×1	-	-	Sigmoid

Redmon and Farhadi [2017] Fu et al. [2017]. This approach relies on very deep networks to provide the the output layers with enough semantic value, and then uses bounding boxes to handle localization precision loss.

Our approach takes a different look at the problem versus previous work. FasteNet uses transposed convolutions to increase the output spatial resolution to 200×64 (1/8th the input size), and then predicts an objectness score at each output location. To reduce information loss and alleviate the vanishing gradient problem Hochreiter [1998], feature maps from an earlier layer are concatenated to layers before the output layer. 1×1 convolutions are then used to produce the output saliency map. This allows up to 30 predictions per fastener. Other tidbits that boost FasteNet's performance include batch normalization to combat internal covariate shift during training and to provide regularization. Leaky ReLU units with a negative gradient of -0.1 are used to prevent dead neurons. The overall network architecture is shown in table 4.1, note that the spatial size depends on the input size; any input size divisible by 64 can be computed by FasteNet - our implementation uses an input size of 1600×512 during inference. This varying input size during training (for highly parallel training) and full resolution images for inference is unique to fully-convolutional architectures, a strong suit of our implementation.

Training and Evaluation The network is trained with 1,000 256×256 random crops per image for 700 out of 997 available images. The mean squared error (MSE) objective is used as a loss function, with the Adam optimizer Kingma and Ba [2015] with a learning rate of 1e-6 and weight decay of 1e-2 is set as the optimizer. The network converges after the first 20 epochs with a precision of 96% and a recall of 89% when validated on the remaining 297 images. Hard negative mining was then used to train the network on difficult training images (more than 1 FP or FN). The network was then explicitly trained on these examples for 2 epochs, before continuing on the whole 700 images for another 2 epochs. This was repeated 3 times. In the end, a precision of 99% and a recall of 90% was accomplished on the

validation dataset. Some examples of predictions against ground truth labels are shown in figure 4.5.

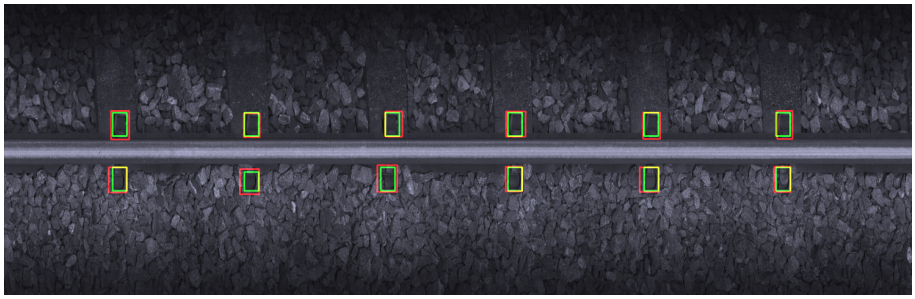

Figure 4.5: Green boxes are FasteNet predictions, red boxes are ground truth annotations. Some examples of FasteNet predictions against ground truth labels.

Insights As a preliminary experiment, FasteNet provided lots of lessons. First and foremost, while FasteNet performed well for a localization task, it showed that perfect image segmentation on objects with arbitrary boundaries is a very difficult task. This is attributed to the imprecision of human labellers as well as the ambiguity in pixel perfect labels. This is clearly seen in figure 4.5. While most of the predicted fasteners were localized correctly, their prediction boundaries never matched up with the ground truths. While this may not be a problem for localizing fasteners, when extrapolated to more precise objects such as rail lines, pixel perfect precision starts to become important.

Second, the goal of FasteNet was to initially detect missing fasteners in an image of a railway. However, from the get-go, this proved difficult — there were less than 20 instances of broken or missing fasteners across all 997 images. Specifically training FasteNet to identify these missing instances proved impossible due to the low data regime as well as the diversity in which these fasteners can be missing. Instead, it is much easier to detect the presence of fasteners, and then count the expected number of fasteners per image as a proxy for missing fasteners. In this manner, FasteNet is able to "identify" a large majority of missing fasteners because of its ability to correctly localize what a properly fastened fastener looks like — better than human annotators as shown in figure 4.6. This implies that it's much easier to detect the absence of normally occurring information in an image as a maintenance requirement than it is the opposite.

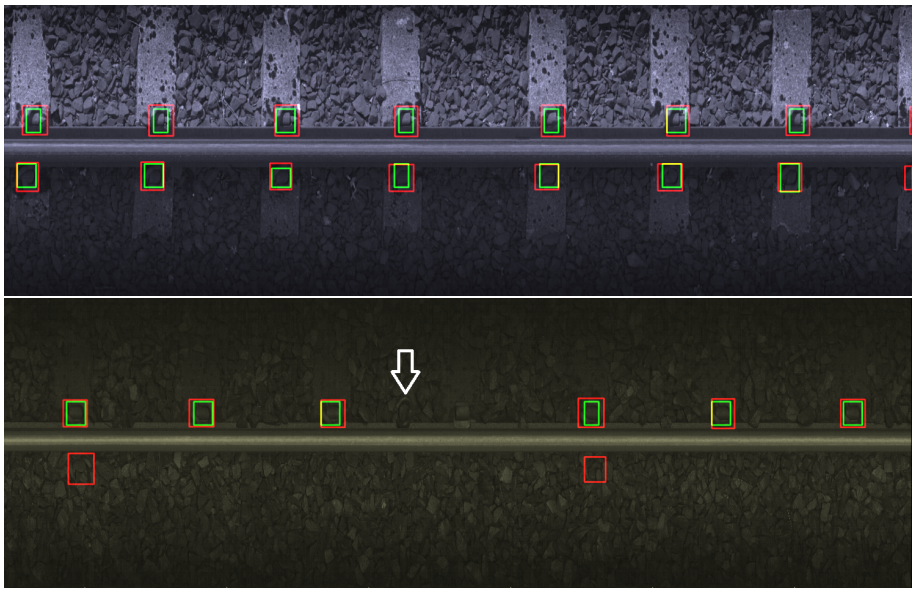

Figure 4.6: Examples of FasteNet being remarkably good at detecting the presence of properly fastened fasteners. Top: Fastenet properly identifies correctly fastened fasteners at the edges of the image, despite the ground truth not being labelled by human annotators. Bottom: Fastenet correctly hedges against a broken or missing fastener illustrated with the white arrow. However, it does make false positive predictions in the bottom half of the image.

4.4.2 Sun Kink Detection

Another case where we can exploit the structure of the images is in sun kink detection, using a rail segmentation network as a backbone. Railways are pre-tensioned system, in that the individual rail tracks are stretched during construction, such that under high temperatures, the expansion caused by temperature fluctuation serves to relieve the compressive stresses in the track. However, under particularly hot weather or use conditions, extreme temperature variations under the influence of direct sunlight, can cause the metal rails to expand or contract leading to warping beyond specification, especially when the rails are used repeatedly without cooling. This expansion and contraction might lead to bending or warping of the tracks if they are not properly maintained or if there are issues with the track structure. An example of a track sun kink is shown in figure 4.7 Some ways of circumventing this issue include novel paints on the railway to reflect heat [Li et al., 2019, Johnson, 2022]. Conventionally, sun kink detection has been done using acoustic and ultrasonic data [Phillips, 2012, Di Summa et al., 2023, Rustam et al., 2023]. One example recently utilized an region-of-interest identifier into object detection to identify sun kinks using augmented data [Mittal and Rao, 2017].

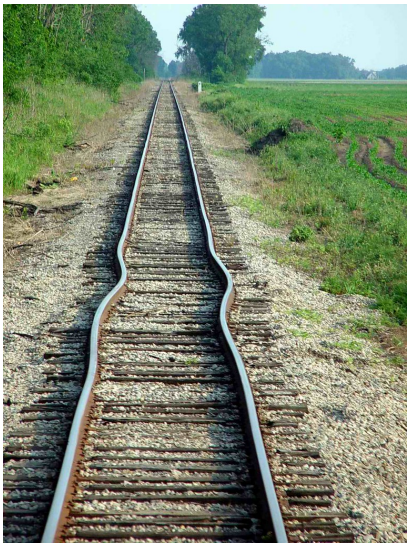

Figure 4.7: An example of rail track warp caused by excessive heat, also known as a sun kink.

Table 4.2: Network architecture of SunkinkNet. The outputs from layer 3 are concatenated channel wise to outputs of layer 10 to form the input to layer 11.

Layer #	Operation	Input Size	Output Size	Kernel Size	Pad Size	Pool Size	Activation
1	Convolution	1@ 1600×512	32@ 800×256	3×3	1×1	2×2	Leaky ReLU
2	Convolution	32@ 800×256	32@ 400×128	3×3	1×1	2×2	Leaky ReLU
3	Convolution	32@ 400×128	64@ 200×64	3×3	1×1	2×2	Leaky ReLU
4	Convolution	64@ 200×64	64@ 100×32	3×3	1×1	2×2	Leaky ReLU
5	Convolution	64@ 100×32	128@ 50×16	3×3	1×1	2×2	Leaky ReLU
6	Convolution	128@ 50×16	128@ 25×8	3×3	1×1	2×2	Leaky ReLU
7	Convolution	128@ 25×8	64@ 25×8	1×1	-	-	Leaky ReLU
8	Transposed convolution	64@ 25×8	64@ 50×16	4×4	1×1	-	Leaky ReLU
9	Transposed convolution	64@ 50×16	64@ 100×32	4×4	1×1	-	Leaky ReLU
10	Transposed convolution	64@ 100×32	64@ 200×64	4×4	1×1	-	Leaky ReLU
11	Convolution	128@ 200×64	1@ 200×64	1×1	-	-	Sigmoid

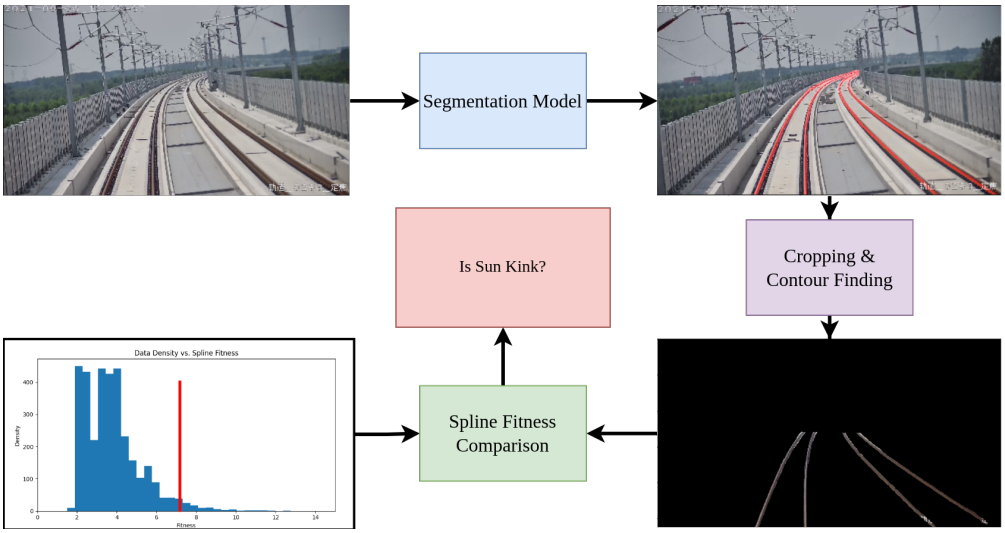

Figure 4.8: Top level view of SunkinkNet. During training, the model is trained to perform vanilla image segmentation on the rail tracks. At inference, we fit a second order polynomial to the contours segmented by the model. Comparing the level of fitness of the curve to a distribution of past fits allows us to identify outliers.

SunkinkNet Our strategy for performing sun kink detection carries a similar theme to that of fastener detection — utilize a deep learning model to segment relevant components, then exploit the structure of the image to detect what obtain what's required. In this case, we aim to obtain segmentation maps of the rail tracks, and then fit a second order polynomial to each segmented contour. This relies on the assumption that the camera view relative to the rail track is mostly unchanged. Finally, we compare the curve fitness to a distribution of global curve fitnesses over observed data. The curve fitness refers to some measurement of how well the second order polynomial fits the contour, whether it be mean squared error between the contour points and the curve, or maximum deviation. Curves that produce values in the far outer ranges of the fitness distribution are marked for further investigation into a potential sunkink. A graphical top level view of this frameowrk is shown in figure 4.8. More concisely, SunkinkNet exploits the same purely convolutional architecture as FasteNet. The exact architecture is shown in table 4.2. The main change is that the input to the network is a full RGB image, and the output size of the network is similar to the input size.

Training and Evaluation For this experiment, we utilize the openly available RailDB [Li and Peng, 2022]. RailDB is convenient because each segment of rail does not cross other sections of rail, which is relevant for the task at hand — sun kinks never occur at rail intersections since relief points are built there. We put all railway segmentations from RailDB into the same channel, and split it into 75% training set, and 25% test set. This results in 5,574 training images and 2,160 testing images in various lighting and weather conditions. RailDB

Figure 4.9: Examples of digitally modified images of rail tracks with sun kinks.

itself doesn't come with sun kink data. In fact, no publicly available dataset has a statistically significant amount of sun kink data. This is an expected finding since railway defects, as established, are a rare occurence. To solve this issue, we adopt the approach used by [Mittal and Rao, 2017] where image manipulation techniques are utilized to draw sun kinks on existing images. While this does not yield perfectly representative images of sunkinks in the real world, at the resolution that SunkinkNet is trained at, the segmentation of the rail tracks still worked well. Some examples of these modified images are shown in figure 4.9.

All images were rescaled to a size of 270×480, or about 1/8th the original image size. This size was chosen through visual inspection of the images to not lose too much detail. The binary cross entropy loss function was used as an objective to the model, with the AdamW optimizer [Kingma and Ba, 2015] using a learning rate of 1e-3 with a weight decay of 1e-2 (the default parameters in PyTorch 2.0). The network converges after the first 50 epochs with a pixel-wise precision of 82% and a recall of 85% when validated on the remaining 2,160 images.

Curve Fitting Several preprocessing steps must first happen after the image is segmented for curve fitting to happen. To hedge against spurious detections, contours with a pixel count less than k_{thresh} are eliminated. Visually, the railway track merges together at the far end of the image due to the vanishing point effect. This is visually observable in the top right image of figure 4.8. Since the goal is to isolate individual track lines, they must be separated from each other. This cannot be fixed with simple image distortion correction since the actual pixels are a cohesive contour. Instead, we opt to simply crop out this section of pixels. The process used here is simple — detect, from the top of the image, when the first railway track pixel is detected, then crop a fixed percentage down from that. In our experiments, we found a value of 15% of total image height works well. These two processes yield a collection of n contours, where each contour is made of a series of pixel locations $\{(x_1, y_1), (x_2, y_2), ...(x_m, y_m)\}$. Finally, For each contour in the image (bottom right image of figure 4.8, we swap the x and y image axes, and fit a second-order polynomial $f_i(\cdot)$ for $i \in \{1, 2, ...n\}$ to the points within the contour by minimizing the mean squared error between each point and their estimated location on the line $E_i = 1/n \sum_{j=1}^{n} (x_j - f_i(y_j))$ (we fit f_i using y_j as the input variable).

Several methods can be used to then evaluate the fitness of each curve. One may naturally approach the idea of using the mean squared error as a fitness function. After all, it is already

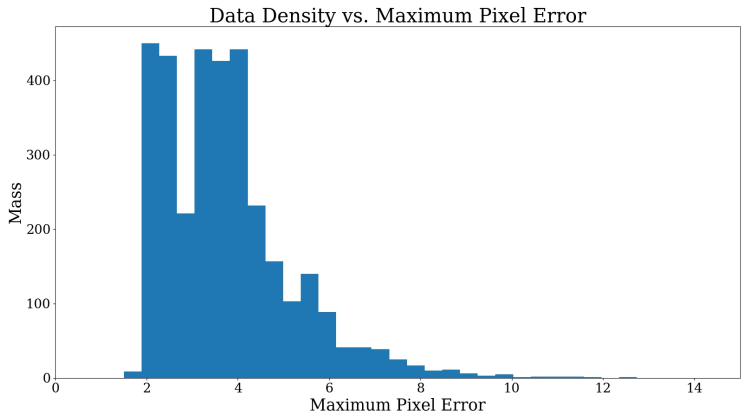

Figure 4.10: Distribution of maximum pixel errors between segmented railway tracks and the fitted line across images in the RailDB dataset.

being used to fit the curve itself. However, for long segments of railway track, it was observed that the error produced by a sun kink gets averaged out by the mean squared error operator. This is expected — the mean squared error has resilience to outliers. Instead, a better fitness function would be and maximum absolute error between all pixels in the contour and a given curve, ie: $F_i = \max(\{|x_j - f_i(y_j)| \forall j \in \{1, 2, ...m\}\})$ which we term here as maximum pixel error.

Insights The result of running the aforementioned curve-fitting procedure on all the railway track images in the evaluation dataset yields the distribution of maximum pixel errors shown in figure 4.10, generated by binning all pixel errors into 100 separate bins. This distribution reflects the distribution of pixel errors for normal railway tracks without sun kinks. Thus, we can draw a threshold on this distribution for values that may represent sun kinked tracks. Given that we do not have images drawn from the source domain of sun kinked railways, digitally manipulated images are used instead, and their maximum pixel errors are plotted relative to this distribution, shown in figure 4.3. While we cannot yet draw a probability value that these images contain sunkinks based on the inferred values for maximum pixel error, one can imagine a framework where a threshold for images with a maximum pixel error more than a certain threshold could be flagged for further analysis, either by a separate model trained specifically to identify sun kinks based on an image recognition framework, or a human expert.

			
6.5	8.5	7.3	10.2

Table 4.3: Digitally manipulated railway track images and their maximum pixel error values when using the SunkinkNet model.

4.4.3 Other Potential Avenues for Performing Maintenance Detection Under Ideal Angles

Outside of fastener (or missing thereof) detection and sun kink detection, various other avenues for railway maintenance needs detection can be addressed using computer vision by exploiting the structure in the data produced by consistent camera viewpoints. This section aims to outline several avenues where a similar approach can be used, highlighting avenues for future research.

One prominent example includes insufficient ballast detection. Track ballast is the material which forms the track bed upon which sleepers are laid. It is packed between, below, and around the sleepers with the primary goal of preventing the sleepers from shifting under the loads and forces induced by a moving overhead train. Insufficient ballasts can cause a variety of issues in the railway track system, including track instability where sleepers shift and sink under the weight of a train, increased vibration, and insufficient drainage leading to vegetation growth within the railway tracks. In the past, Sabato and Niezrecki [2017] have utilized a 3D-image correlation technique using moving cameras to evaluate the health of track ballast without any machine learning techniques. On the other hand, Mittal and Rao [2017] utilize a single object detection framework to perform insufficient ballast on a dataset of track images with insufficient ballast. Our proposal for performing insufficient ballast detection is to train a segmentation model to segment exposed sleeper sides, such as that shown in figure 4.11. This total area of exposed sleeper sides can then be compared against a distribution of nominal exposure amount to detect the presence of insufficient ballast. Such methods can also be used to detect defects in railway tunnel walls, as done by various past works [Li et al., 2021, Farahani et al., 2020, Jenkins et al., 2017], where the regular structure of railway tunnel walls are used to detect the presence of damage or missing components such as missing fasteners for wall mounted conduits or cracked bricks.

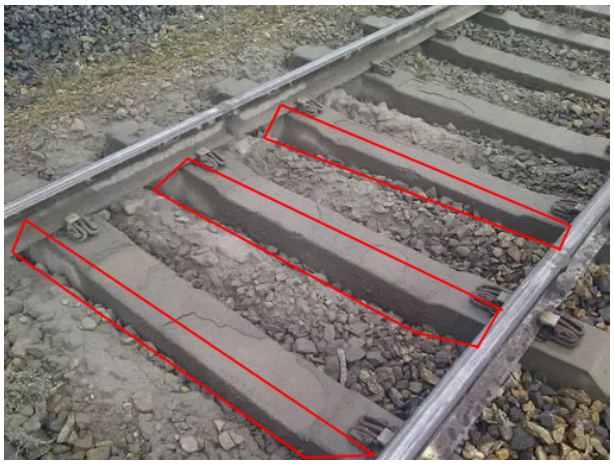

Figure 4.11: Railway track sleepers supported by insufficient ballast. One defining trait of cases of insufficient are exposed sleeper sides.

4.5 Finding Anomalies with Uncertainty Quantification

Performing anomaly detection using generative model reconstruction on the raw image has been extensively studied in literature. While this can work, we highlight one glaring argument for why this may not be the ideal approach when doing anomaly detection using images gathered from a UAV.

4.5.1 Why Reconstruction Error is a Bad Metric

The main downside to using reconstruction error as a proxy for anomalies is the breakdown of high reconstruction error on high texture areas. To highlight this scenario, we perform a toy experiment using a simple autoencoder. This experiment carries the same structure as other reconstruction-based models:

1. Train model to take in an original image x and then reproduce the same image x', the model incorporates some form of information bottleneck such that it cannot carry all the information from x to x' losslessly.

2. Select a suitable reconstruction error threshold, $\tau \in \mathbb{R}$ from which to flag anomalous images, where reconstruction error is defined as $e = |x - x'|$.

3. Deploy model, any region of the image which produces a reconstruction error higher than the selected threshold, $e > \tau$ is flagged as an anomaly.

Figure 4.12: Two examples of the images used in the experiment.

Table 4.4: The architecture used in the image reconstruction experiment.

Layer #	Operation	Input Size	Output Size	Kernel Size	Pad Size	Pool Size	Activation
1	Convolution	1@ 64×64	8@ 16×16	3×3	1×1	2×2	ReLU
2	Convolution	8@ 16×16	16@ 8×8	3×3	1×1	2×2	ReLU
3	Convolution	16@ 8×8	4@ 4×4	3×3	1×1	2×2	Tanh
4	Transposed Convolution	4@ 4×4	16@ 8×8	4×4	1×1	-	ReLU
5	Transposed Convolution	16@ 8×8	8@ 16×16	4×4	1×1	-	ReLU
6	Transposed Convolution	8@ 16×16	1@ 64×64	4×4	1×1	-	Sigmoid

Dataset The dataset includes artificially created images such as the ones shown in figure 4.12. They are grayscale images with a texture-less left half and a textured right half. The colour on the left half is exactly gray, having a value of 0.5 throughout. The right half of the image is white (with a value of 1.0), with pixels being set to black (a value of 0.0) randomly at a rate of 30%. One can imagine that the left half represents a sleeper on a railway — texture-less and mostly a single colour; the right half represents the ballast — high texture and with a high luminance range.

Model Architecture The model is a simple 6 layer convolutional neural network, with an architecture described in table 4.4. The `Tanh` activation function in the middle of the architecture aims to serve as a sort of information bottleneck without requiring the usage of variational bayes loss functions [Kingma and Welling, 2014].

Training Setup The training objective is to minimize the Mean Squared Error (MSE) loss between the reconstruction and original image. The model is trained for 1×10^4 epochs and a batch size of 32 — enough for model convergence to happen, while completing training in less than 10 seconds. It is optimized using the AdamW optimizer [Kingma and Ba, 2015] with a learning rate of 0.001.

Insights Examples of generated images, their reconstruction, and the reconstruction error for the trained model are shown in figure 4.13. Under normal scenarios, the reconstruction on the left half is faithful to the original image. However, the right half tends to be a

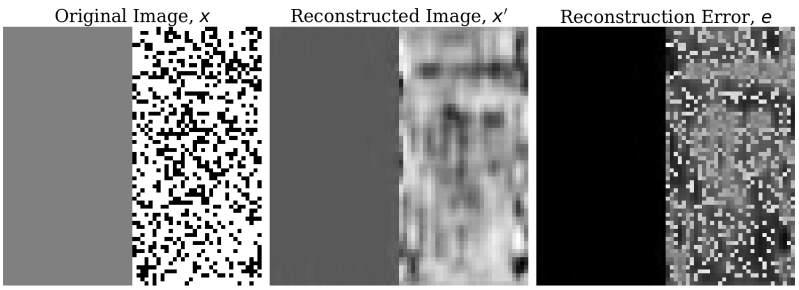

Figure 4.13: Example of image reconstruction — left: original image; middle: reconstructed image; right: reconstruction error, brighter parts indicate higher error.

blurred version of the texture — essentially the best decoded image which produces the lowest loss on that half of the image given the information bottleneck. Very frequently, the peak pixel-wise reconstruction error on the right half of the image is more than 0.9, or more than 90% the maximum theoretical error. Setting $\tau = 0.9$ is an unrealistic setting, and will cause all anomalies on the left to be ignored since the maximum theoretical error on the left half of the image is 0.7.

Obviously, there are several ways around this problem. One solution would be to run a Gaussian filter or mean filter over the image of the reconstruction error, this will have the effect of looking at localized areas of high uncertainty instead of pixel-level uncertainty. The immediate downside to this is that small actual uncertainties can now be ignored. Another solution in this toy experiment is to have a position-aware threshold, i.e.: instead of selecting a scalar threshold $\tau \in \mathbb{R}$ for the whole image, we instead select a threshold for each location in the image $\tau \in \mathbf{R}^{H \times W}$. One way of viewing this is to canonically ask "which parts of the image do we *really* care about?". This way, we can simply select a lower threshold for the left half of the image than the right half. This can work for very structured images such as the one in the toy example, or images contained in datasets utilized earlier gathered from the ego-perspective of the train in earlier sections. However, for unstructured images gathered from a UAV that can contain a railway viewed from many different angles, this is no longer trivial. While reconstruction-based anomaly detection has seen its successes — usually through careful engineering of the information bottleneck and loss function — these techniques tend to rely on exploitable traits within the domain, such as a regular background, or repeating patterns in features.

4.5.2 Two Methods for Uncertainty Quantification

Instead of using reconstruction error on the raw image as a proxy for the presence of anomalies, a better metric is to directly use some form of model uncertainty over the image. In supervised learning for classification tasks, a model is trained with the goal of mapping input data to a series of labels. In this setting, it is possible to construct models that provide some idea of model uncertainty. Generally, in addition to the standard supervised learning goal, the model is also constructed such that the model predictions provide some indication of model uncertainty or epistemic uncertainty to the user.

There are two primary methods this can be done — ensemble deep learning or evidential deep learning [Amini et al., 2020, Sensoy et al., 2018]. Ensemble deep learning falls under Frequentist methods introduced earlier. This technique is similar to bagging in classical machine learning, where M similar models with different initializations are trained to make predictions with same inputs and outputs. Each model f_j outputs a probability distribution over N labels for an in put $\mathbf{x}$ as $f_j(\mathbf{x}) = \mathbf{y}_j$ where $\dim(\mathbf{y}_j) = N, y_{j,i} \in \mathbb{R}_+$ and $\|\mathbf{y}\|_1 = 1$, the final constraint can be enforced by taking a softmax over model output logits. The penultimate probability distribution can be represented in several manners, the most convenient being the normalized sum over each label $p(i) = \sum_j^M \mathbf{y}_{j,i}/M$. In this setting, model uncertainty can be easily represented as Shannon entropy $\eta = \sum_i^N \mathbf{p}(i) \log \mathbf{p}(i)$. The loss function used in this setting is the classic binary cross entropy loss function for each model:

$$\mathcal{L}_{CE} = -\frac{1}{N} \sum_i^N ((1 - \mathbb{I}_i) \log(1 - p(i)) + \mathbb{I}_i \log p(i)) \tag{4.1}$$

Here, $\mathbb{I}_i$ is an indicator on whether the ground truth label belongs to the class i or not. When applied to image segmentation tasks, one independent prediction is done on each output pixel, classifying each pixel as belonging to a particular label class.

On the other hand, evidential deep learning falls under Bayesian methods introduced earlier, where the outputs of the model are modelled as a Dirichlet distribution. The Dirichlet distribution is probability density function that is commonly utilized as a conjugate prior to the multinomial distribution. This property makes it convenient for representing both model uncertainty and predictions. The Dirichlet distribution Dir is represented by an evidence vector $\boldsymbol{\alpha} \in \mathbb{R}^N_{>1}$[1] for N prediction targets. The evidence vector represents the belief that a model thinks a prediction belongs to a label, with higher values indicating higher belief. The probability mass for a given label $i \in N$ during prediction is then computed as $p(i|\boldsymbol{\alpha}) = \alpha_i/\|\boldsymbol{\alpha}\|_1$. When applied to image segmentation tasks, each output pixel represents its own predictive Dirichlet distribution. When pixels are allowed to overlap, $N = 2$, and M distributions are placed over each output pixel for M possible labels, such that the probability of a pixel belonging to any label $j \in M$ is $p(j|\boldsymbol{\alpha}_j) = \alpha_{1,j}/\|\boldsymbol{\alpha}_j\|_1$ resulting in

[1] In classical statistics, the Dirichlet distribution actually has a range of values $\boldsymbol{\alpha} \in \mathbb{R}^N_+$ instead of $\boldsymbol{\alpha} \in \mathbb{R}^N_{>1}$, however, we present the definition in the context of uncertainty quantification

a vector of probabilities $\mathbf{p} = \{p(1|\boldsymbol{\alpha}_1), p(2|\boldsymbol{\alpha}_2), ...p(M|\boldsymbol{\alpha}_M)\}$. The epistemic uncertainty on any label is then represented the lack of evidence over all labels, where one common representation is $\mathcal{E} = \dim(\boldsymbol{\alpha})\|\boldsymbol{\alpha}\|_1^{-1}$ for non-overlapping prediction labels and $\mathcal{E} = \sum_j^M \|\boldsymbol{\alpha}_j\|_1^{-1}$ for overlappable prediction labels. There are multiple loss functions possible in this setting, the easiest to apply being the Type II Maximum Likelihood loss function, which is a modification over the standard binary cross entropy loss. We describe the variant for non-overlappable labels here, but the application to overlappable labels is just an extension over the dimension of M:

$$\mathcal{L}_{T2ML} = \frac{1}{N}\sum_i^N \mathbb{I}_i(\log\|\alpha\|_1 - \log\alpha_i) \tag{4.2}$$

$$= -\frac{1}{N}\sum_i^N (-\mathbb{I}_i \log\|\alpha\|_1 + \mathbb{I}_i \log\alpha_i) \tag{4.3}$$

The first difference between Eq. (4.1) and Eq. (4.2) is that Eq. (4.1) operates on probability distributions, providing an overall stronger signal as gradients of $\log x$ are higher for smaller values of x, while Eq. (4.2) operates on unnormalized evidence. The second difference is the penalization on non-labels, where Eq. (4.1) aims to minimize $(1-\mathbb{I}_i)\log(1-p(i))$, which is once again a stronger signal than that in Eq. (4.2) which aims to minimize $-\mathbb{I}_i \log\|\alpha\|_1$. The overall goal is that evidential uncertainty methods favour models that are not overly confident, and hence can provide a meaningful insight into their levels of uncertainty. In contrast, ensemble methods tend to produce an ensemble of models that are each very confident in their predictions, and have to resort to group predictions for a sense of overall confidence. At optimality, both methods described here functionally provide the same result, while they vary only in methods of representing uncertainty.

From a pragmatic point of view, one may immediately see that the Bayesian method for uncertainty quantification is superior to the Frequentist one — it uses only one model instead of having a train a group of similar models. However, in practice, it has been frequently observed that evidential deep learning, in many cases, can fail to produce reliable uncertainty metrics [Ulmer et al., 2020, Ulmer and Cinà, 2021, Van Landeghem et al., 2022]. The reasoning for this is simply that it is impossible to guarantee certain outputs for *very* out-of-distribution data in deep learned models. However, guaranteeing a range of predictions for an ensemble of models on out of distribution data is much easier, especially for large ensembles.

4.5.3 Uncertainty Quantification for Image Segmentation

Section 4.5.1 naively established that using reconstruction error for anomaly detection in the image space is not the best idea. Two most popular branches for uncertainty quantification in classification tasks have also been introduced in Section 4.5.2 (image segmentation can be seen as a simple extension of classification). Here, we explore using the described uncertainty quantification techniques on some sample datasets for image segmentation, and demonstrate

many flaws that arise from this application. The goal of this section of work is to decide if uncertainty quantification on classification tasks scales appropriately to image segmentation. Image segmentation can be generalized into a classification task at every output pixel. However, one key difference is that image segmentation generally incorporates a "background" class — a label for pixels that do not belong to any important asset but are normal within an image. This label is not usually present in classic classification tasks. Thus, when trying to use current uncertainty quantification algorithms on image segmentation without modifications, we immediately run up against the issue of trying to discern the difference between anomalies and the background. In addition, we also face the problem of high uncertainty at the edges of segmented object instances. We term this the "uncertain edge" problem. This is because it is very difficult to get consistent pixel-level accuracy for the exact boundary between an object and its surrounding. The following experiments demonstrate the shortcomings described, and proposes one solution to uncertain edges.

Normalizing Uncertainty Scores In the all cases, we normalize the uncertainty to lie in $[0, 1]$ for much easier visualization. In the case of ensembling, an ensemble of predictions is defined as $\mathbf{Y} \in \{0, 1\}^{M \times N}$ where M is the size of the ensemble and N is the number of labels, and each N long vector for M vectors making up $\mathbf{Y}$ is a one-hot vector. We define the final prediction $\mathbf{y}$ as mean vector over M predictions, that is: $\mathbf{y}_i = [M^{-1} \sum_j^M \mathbf{Y}_{j,i}]$ where $i \in \{1, 2, ...N\}$. The maximum uncertainty in this case happens for a uniform distribution over N labels, ie: $\mathcal{E}_{max} = -\log(1/N) = \log N$. Thus, we defined the normalized uncertainty measure as $\mathcal{E}_{norm} = -\sum_i^N \mathbf{y}_i \log \mathbf{y}_i / \log N$

In evidential learning, a prediction is an N long vector $\mathbf{y} \in \mathbb{R}^N_{>1}$. Conveniently, the uncertainty measure is defined as $\mathcal{E} = \mathcal{E}_{norm} = N/(\sum_i^N \mathbf{y}_i)$, which is already normalized to $[0, 1]$.

Dataset We utilize the NMTFastenerSet2020 dataset defined earlier in figure 4.1. The datasets does not inherently have an anomaly class. Instead, we manipulate the images digitally to produce a set of anomalous images for each dataset that are used outside of training to visualize the effectiveness of the two uncertainty quantification techniques. More concisely, we do the following:

- **Artificial Drawing** — This involves artifially drawing on the base image to produce completely out of distribution samples.
- **Cut & Paste** — This involves copying out certain square segments of the image and pasting it onto other parts. The intention here is to evaluate whether the model pays attention to local features or the overall structure of the image
- **Foreign Entity Insertion** — This involves inserting random foreign entities into the image. The intention here is to see if the model is able to flag previously unseen entities as anomalous, such as foreign debris on the railway tracks.

Some examples of these manipulations are shown in figure 4.14, figure 4.15 and figure 4.16 respectively.

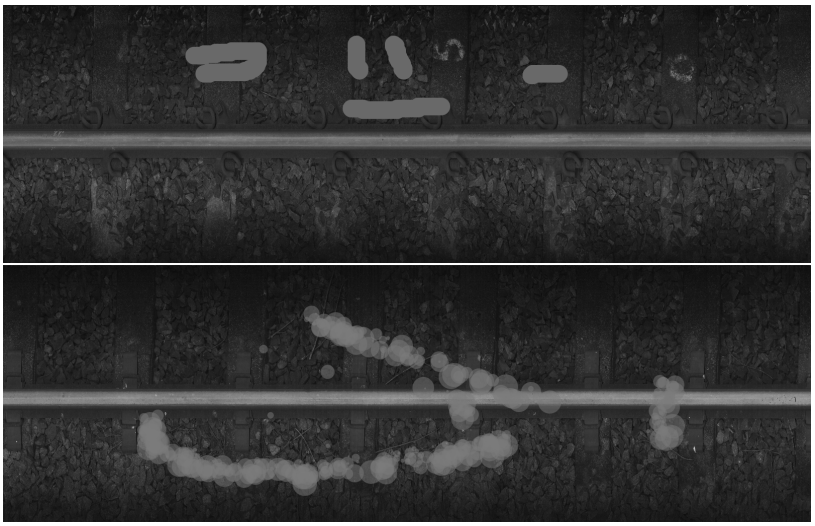

Figure 4.14: Images from NMTFastenerSet2020 that have been digitally altered by drawing on them.

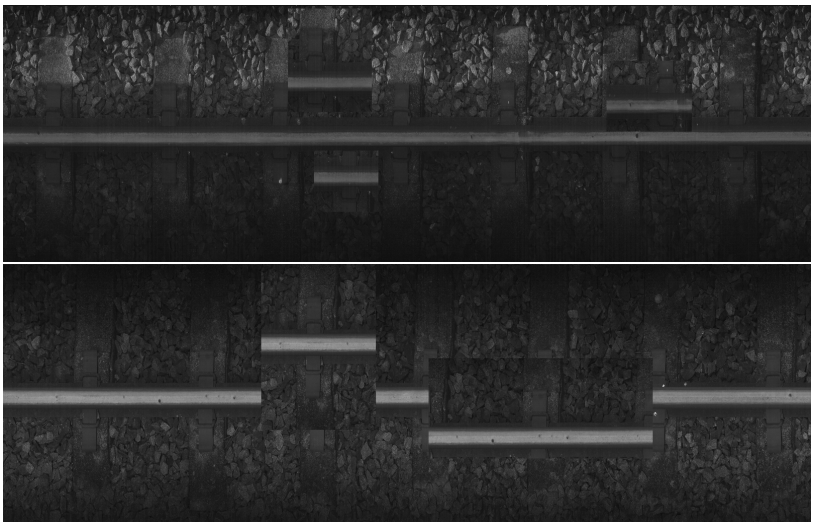

Figure 4.15: Images from NMTFastenerSet2020 that have been digitally altered by cropping sections from one part of the image onto other sections.

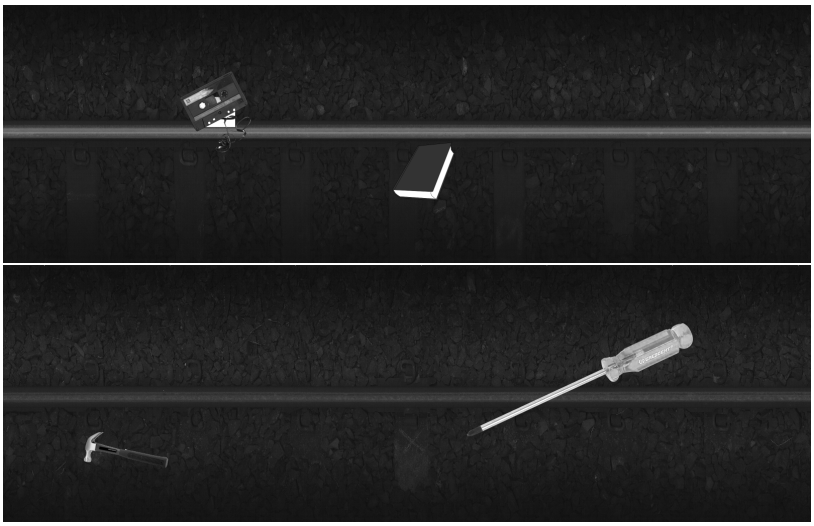

Figure 4.16: Images from NMTFastenerSet2020 that have been digitally altered by inserting foreign entities into the image

Model Architecture and Training Setup We utilize a generic Residual U-Net for all experiments. For the evidential model, the architecture follows the description in table 4.5 exactly. The outputs of the model are then activated using $(\text{SoftPlus}(x) + 1)$ where x is the output from the final layer. For the ensemble, we utilize 4 identical models of that described in table 4.5. The output from each model is activated using the `softmax` operator during training and then trained using the binary cross entropy loss described in Eq. (4.1). At inference, we first perform an `argmax` over the label dimension of each model output, then we take the mean over the ensemble dimension. All training instances were trained with a batch size of 32 using the AdamW optimizer [Kingma and Ba, 2015] with a peak learning rate of 0.003. During training, all images were subjected to random cropping to a size of 256×256, as well as random rotation and random flipping. Each training instance was trained to convergence, this was identified when the precision and recall metrics on the test set for each dataset fluctuated by less than 1% across three epochs, all models reached convergence before 200 epochs.

Table 4.5: The architecture used in the image reconstruction experiment. Because it is a residual architecture, we adopt an element wise addition across the convolution and transposed convolution layers. Therefore, layer 1's output is copied and added to the input of layer 10, layer 2's output is copied and added to the input of layer 9, etc. .

Layer #	Operation	Input Channels	Output Channels	Kernel Size	Pad Size	Pool Size	Activation
1	Convolution	3	16	3×3	1×1	2×2	ReLU
2	Convolution	16	16	3×3	1×1	2×2	ReLU
3	Convolution	16	32	3×3	1×1	2×2	ReLU
4	Convolution	32	32	3×3	1×1	2×2	ReLU
5	Convolution	32	64	3×3	1×1	2×2	ReLU
6	Transposed Convolution	64	32	4×4	1×1	-	ReLU
7	Transposed Convolution	32	32	4×4	1×1	-	ReLU
8	Transposed Convolution	32	16	4×4	1×1	-	ReLU
9	Transposed Convolution	16	16	4×4	1×1	-	ReLU
10	Transposed Convolution	16	num. labels	4×4	1×1	-	ReLU

Uncertainty under Normal Conditions The results of training both methods of uncertainty aware models on are shown in table 4.6. For these scores, we utilize a fixed train/test 3:1 split for NMTFastenerSet2020. The goal of these results is not to demonstrate that the model architecture chosen works on both datasets (their performances can definitely be improved with a better architecture with more parameters), but to show that they still work even with different representational heads meant to quantify uncertainty. Using a cross validation technique such as K-fold may produce different results, but in our experiments we found very little (< 1%) run-to-run variation.

	Evidential	Ensemble
Precision	92.9%	97.2%
Recall	93.8%	90.4%
Accuracy	99.3%	99.2%

Table 4.6: Results of training both evidential and ensemble models on NMTFastenerSet2020.

Some examples of the uncertainty maps over the images are shown in figure 4.17. One immediately obvious problem is the issue with segmentation outlines, where the outlines on each object instance — fasteners in this case — are of very high uncertainty. This effect is caused by the notion that having pixel perfect segmentation outlines is very difficult, especially for learned models. Therefore, the labels around the edges of individual objects carries very high uncertainty because the model can never be 100% certain in those areas. One easy solution to this problem is to run a Gaussian or mean filter over the uncertainty map. This will immediately crush low thickness traces of uncertainty, leaving only large mass high uncertainty areas remaining. The effect of this is shown in figure 4.19 for the evidential method. Note that the uncertainty at the edges has been completely removed. A similar result can be reproduced for the ensembled model, the result of which is not shown here because it looks the same.

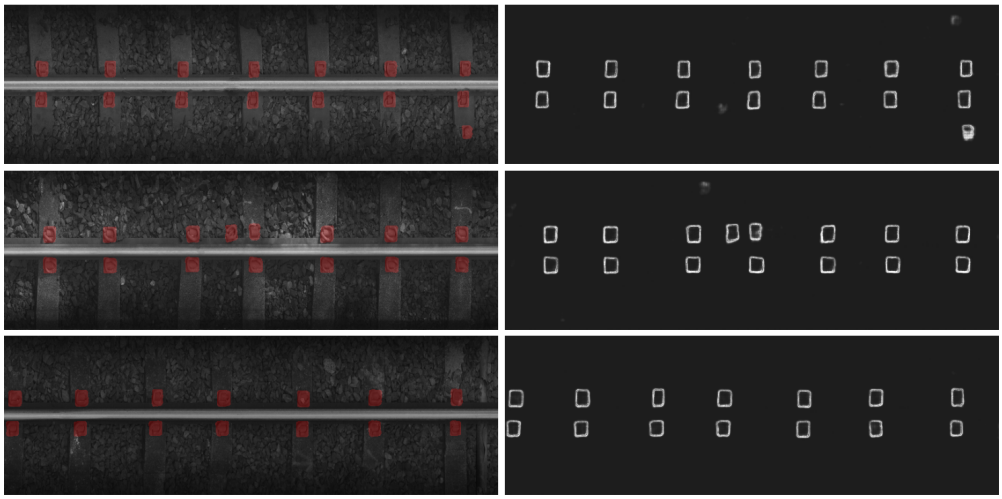

Figure 4.17: Evidential learning results on the NMTFastenerSet2020 dataset. Left: predicted fastener locations in red; right: uncertainty maps, brighter colours indicate higher uncertainty.

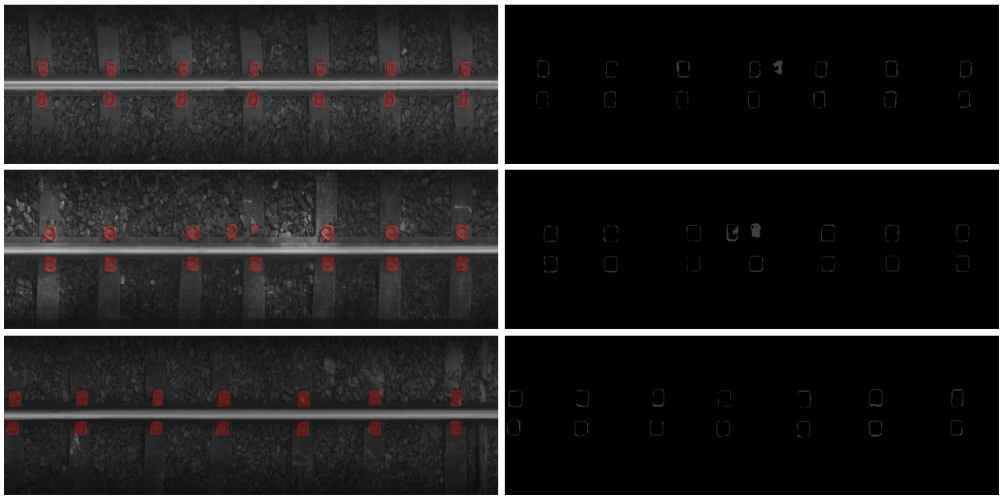

Figure 4.18: Ensemble learning results on the NMTFastenerSet2020 dataset. Left: predicted fastener locations in red; right: uncertainty maps, brighter colours indicate higher uncertainty.

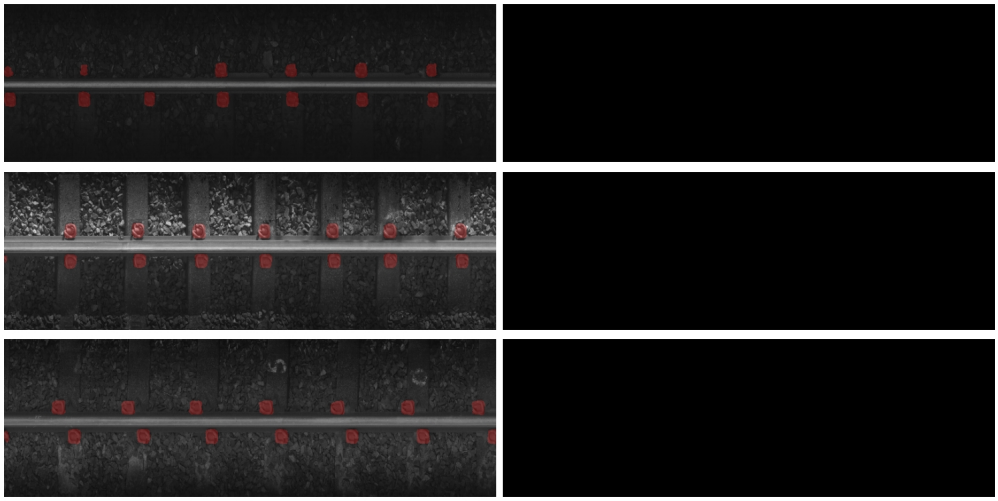

Figure 4.19: Running a filter over the uncertainty map produced on NMTFastenerSet2020 and thresholding the filtered result results in the total removal of uncertainty around the edges. Note that the right map is blank because running a mean filter over the uncertainty map removes all thin lines of uncertainty.

Uncertainty under Anomalous Conditions We digitally augment the images to deliberately have defects as introduced in section 4.5 on page 116. In these instances, we would like the models to produce high uncertainty in areas that have been altered. figure 4.20 to figure 4.22 demonstrates some predictions under these scenarios. Qualitatively, the uncertainty maps under these scenarios look similar to those under normal scenarios with the exception of the images with foreign entities inserted. In particular, artificial drawings on the base image are outright ignored into the background class. This is likely because during training, everything that is not a component of significance (a fastener in this case), is classed as the background class. As a result, anything that does not classify a fastener is identified as background by the model.

A similar story can be said for images altered via the cut & paste technique. In this case, the models performed uncertainty quantification in the normal fashion *within* the cropped sections of the image, essentially ignoring that the image structure was never before seen within the training data. The one case where uncertainty quantification worked exceptionally well is in images altered with foreign entity insertion. In particular, the uncertainty maps on these images show very obviously where the foreign objects are. The main difference between this case and the case of drawing over the main image is that the entities inserted here appear to be higher in luminance values compared to the rest of the image, potentially resulting in the textures that make up the entities being truly out of distribution (far enough from the background class) for the trained model.

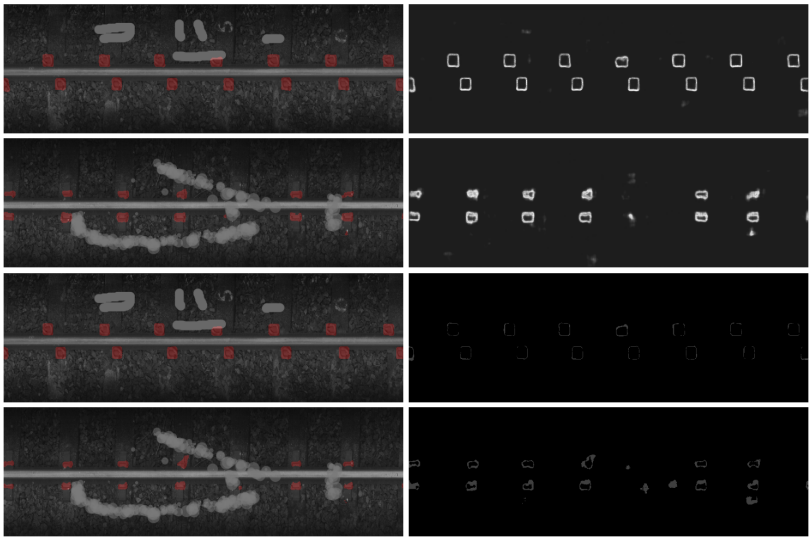

Figure 4.20: Uncertainty quantification results on NMTFastenerSet2020 that have been altered by drawing over them. Top two rows: evidential learning model; Bottom two rows; ensemble-based model. Left: predicted fastener locations in red; right: uncertainty maps, brighter colours indicate higher uncertainty.

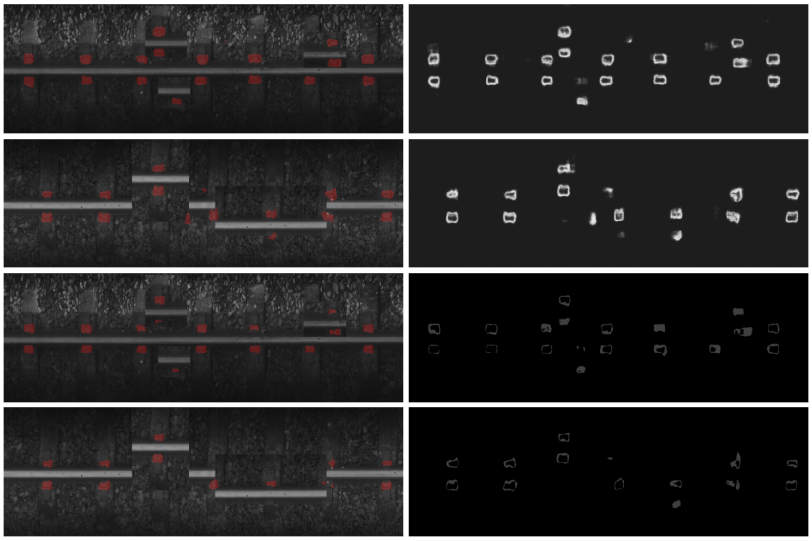

Figure 4.21: Uncertainty quantification results on NMTFastenerSet2020 that have been altered by cropping certain sections of the image over other sections. Top two rows: evidential learning model; Bottom two rows; ensemble-based model. Left: predicted fastener locations in red; right: uncertainty maps, brighter colours indicate higher uncertainty.

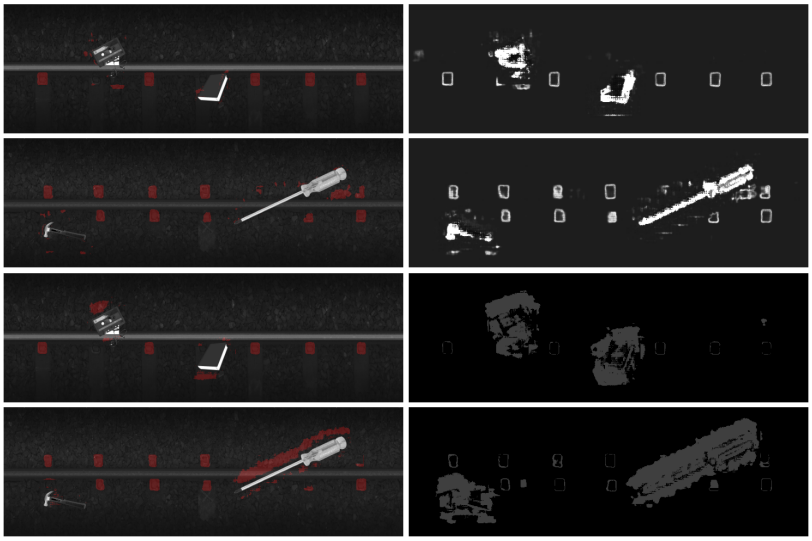

Figure 4.22: Uncertainty quantification results on NMTFastenerSet2020 that have been altered by adding foreign objects into the image. Top two rows: evidential learning model; Bottom two rows; ensemble-based model. Left: predicted fastener locations in red; right: uncertainty maps, brighter colours indicate higher uncertainty.

Insights There are several important takeaways from these experiments. The first is that when applying uncertainty quantification models to image segmentation, there tends to be a large amount of uncertainty at the edges of segmented objects. This problem appears to be an inherent problem, and is a result of the difficulty of having pixel-perfect segmentation in most cases. Fortunately, the fairly easy solution to this problem is to filter the uncertainty map, essentially running a low-pass filter to remove high frequency noise in the uncertainty. Uncertainty quantification can work in some but not all cases of anomaly detection. In particular, the model can appear to be blind to certain textures, such as when images are altered with digital drawings; but can work when image anomalies are distinct enough, such as when foreign entities are inserted into the base image. More crucially, models trained using uncertainty quantification techniques using the conventional attention U-Net seem to ignore structural anomalies in the image. This is evident in the case where the images are altered by cropping out certain sections of the image to be pasted in other sections. This is a fairly alarming finding because in the the context of railway corridor anomaly detection, not all anomalies present themselves as foreign debris. Anomalies can present themselves in cases where fasteners are broken off their intended position or when sleepers are cracked — both anomalies manifesting as structural deviations from the expected non-anomalous case.

4.6 Anomaly Detection by Reconstructing Labels

Prior sections have identified two fundamental issues with conventional anomaly detection techniques for image segmentation tasks - uncertainty at the edges and disambiguity between background and anomalies. The former can be circumvented by first running a form of filter over the uncertainty map, and then thresholding the result. For example, we can take the mean filter with a kernel size of 5×5 pixels, and then treshold the result by zero-ing out all pixels with an uncertainty below 0.5. This produces the result shown in figure 4.19. On the other hand, solving the disambiguity between background and anomalies requires analyzing the problem in more detail.

We conjecture that deep networks fail to identify anomalies because training causes them to reject all instances of non-relevant information and only pick out relevant information. One supporting piece of research is the Dimpled Manifold Model Hypothesis [Shamir et al., 2021]. When a section of image fails to resemble any relevant label, it is automatically regarded as background, and only sections of image that carry some semblance of relevant objects carry sufficient evidence to not belong to the background class – resulting in anomalies only when the anomaly does not visually stray too far from relevant objects like a railway or a fastener, but also does not completely resemble them.

To solve this challenge, we propose a framework that exploits this trait of neural networks to perform anomaly detection only in the label space. We call this algorithm Anomaly Detection by Reconstructing Labels (ADeReL), with a proposed architecture shown in figure 4.23, uses a backbone network to semantically label images, then utilizes a reconstruction model to reconstruct labels using a heavily masked version of the predicted segmentation label. The architecture operates on the idea that the reconstruction step relies on models learning about how labels are structured over the image. This structure is well defined when there is a lack of anomalies in the image, and all relevant parts of the image is labelled properly by the prediction model. In the presence of anomalies, the predicted label in the prediction step strays from the well defined structure, which causes the reconstructed segmentation labels to not line up with the predicted segmentation labels.

Mathematically, we first predict the segmentation label $\mathbf{y} \in \mathcal{Y}$ using the predictor model $\mathbf{y} = f_p(\mathbf{x})$ where $\mathcal{Y}$ represents the space of all reasonable segmentation labels. Then, we mask the result to produce a segmentation mask that is outside of the domain $\mathbf{y}_{\text{mask}} \in \mathcal{Y}_{\text{mask}}$, where $\mathcal{Y}_{\text{mask}}$ represents the space of all feasible masked segmentation labels. Finally, we train a model to bring the masked segmentation mask from the masked domain back to the segmentation label domain $\mathbf{y}' = f_r(\mathbf{y}_{\text{mask}})$ where $\mathbf{y}' \in \mathcal{Y}'$ and $\mathcal{Y}' \approx \mathcal{Y}$. The principle idea is when that when an image has an actual anomaly, the predicted segmentation label $\mathbf{y}$ will lie in a space far from $\mathcal{Y}$, resulting in $\mathbf{y}_{\text{mask}}$ lying outside of $\mathcal{Y}_{\text{mask}}$, and the resulting reconstruction $\mathbf{y}'$ being far from $\mathcal{Y}$, ultimately flagging an anomaly.

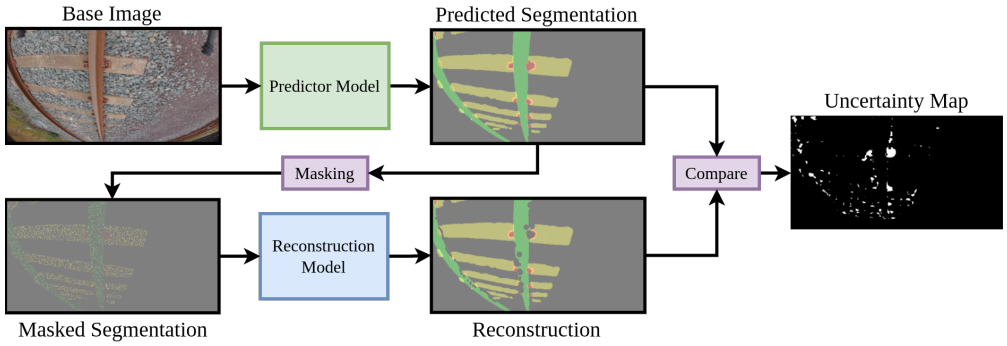

Figure 4.23: Our proposed model, Anomaly Detection by Reconstructing Labels (ADeReL), for performing anomaly detection on image segmentation tasks. In this image, the sampling probability for masking is set at 0.2 $p_{mask} = 0.2$ for visibility. In our experiments, we set it at 0.03.

Table 4.7: The architecture used in the image reconstruction experiment. It is a residual architecture where element wise addition across the convolution and transposed convolution layers are done.

Layer #	Operation	Input Channels	Output Channels	Kernel Size	Pad Size	Pool Size	Activation
1	Convolution	3	num. labels	3×3	1×1	2×2	ReLU
2	Convolution	16	16	3×3	1×1	2×2	ReLU
3	Convolution	16	32	3×3	1×1	2×2	ReLU
4	Convolution	32	32	3×3	1×1	2×2	ReLU
5	Transposed Convolution	32	32	4×4	1×1	-	ReLU
6	Transposed Convolution	32	16	4×4	1×1	-	ReLU
7	Transposed Convolution	16	16	4×4	1×1	-	ReLU
8	Transposed Convolution	16	num. labels	4×4	1×1	-	ReLU

4.6.1 Implementation Specifics

Network Architectures The prediction network is responsible for taking actual images of the railway and producing segmentation labels. We utilize the same network as in table 4.5 for the main prediction network. It is trained using the binary cross entropy loss as in Eq. (4.1). For the reconstruction networks, we utilize a trimmed down model of the prediction network, shown in table 4.7. The main reasoning is that the reconstruction network operates in the low-texture, well defined domain of segmentation labels, having models with high representational power in this domain is not necessary.

Masking Procedure The goal of masking for reconstruction is to present the reconstruction model with sparse information about the original segmentation, such that each reconstruction model formulates its own reconstruction task. Various masking techniques can be chosen, such as the technique from Liu et al. [2023], where an $n \times n$ binary checkerboard is used as a mask. Here, we experiment with the simplest — the input to the reconstruction

model is 0 everywhere, but we randomly sample pixels from the predicted segmentation label with probability p_{samp} (set to 0.03 in our experiments) to produce $\mathbf{y}_{\text{mask}}$, which is then used to reconstruct $\mathbf{y}'$.

Anomaly Identification The uncertainty map $\boldsymbol{\epsilon}$ in our experiments is the difference in prediction between the reconstructed segmentation label and the predicted segmentation label, $\boldsymbol{\epsilon} = (\mathbf{y} - \mathbf{y}')^{|\cdot|}$, where the $^{|\cdot|}$ operator denotes an element-wise absolute. Given this map, there are then several ways to flag an image as having an anomalous instance. Taking the mean uncertainty value over the whole image will not yield a good metric since for large images with smaller anomalies, the outliers will not be reliably picked up. In addition, segmentation labels with many lines and edges will get flagged more often than segmentation labels with low number of edges due to the issue described in section 4.5.3 on page 120. Instead, we flag images as highly uncertainty using the maximum uncertainty patch contour size, where images producing an uncertainty map with a contour size larger than a certain value are flagged instead, resulting in a technique that is more robust to outliers.

Datasets and Model Training We evaluate the performance of the framework on both structured and unstructured data. More concisely, we evaluate its performance on RailDB with one prediction label (the railway track), before evaluating its performance on UAVRailSet2023. We then trained each model with a batch size of 32 using the AdamW optimizer [Kingma and Ba, 2015] with a peak learning rate of 0.001. During training, all images were subjected to random cropping to a size of 256×256, as well as random rotation and random flipping. The model was trained to convergence, taking approximately 50 epochs.

4.6.2 Insights

Performance on RailDB RailDB represents a relatively easier task for ADeReL, as the source images always come from a similar viewpoint, leading to segmentation masks in the label space following a common theme. The dataset serves as a litmus test for ADeReL, and to simulate anomalous data, random sections of the original image, no larger than 5% width and height, are artificially cropped and pasted over sections of the railway tracks in the images. figure 4.24 showcases various examples of images from RailDB with these modifications. This 5% value represents the rough size of a concrete sleeper on the railway track, and emulates the best-case-scenario of an anomaly with the largest scale on the railway track. Smaller anomalies would no doubt be harder to detect, however, simply using higher resolution imagery or more close up images would alleviate this issue. We leave the study of smaller anomalies using higher resolution images to future work and instead focus on feasibility the algorithm at hand instead. The resulting distributions of the maximum uncertainty patch size between normal and anomalous images are illustrated in figure 4.25, clearly demonstrating a distinct difference between anomalous and clean images, underscoring the effectiveness of ADeReL for structured images.

Figure 4.24: Various examples of images from RailDB where random sections of the image are pasted over the railway tracks.

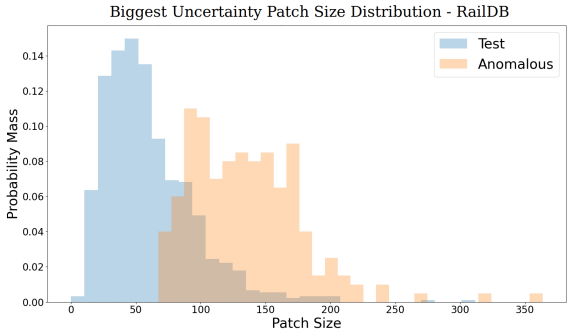

Figure 4.25: Distribution of maximum uncertainty patch size for both testing and anomalous data on the RailDB dataset.

Performance on UAVRailSet2023 Conversely, while ADeReL performs well on the RailDB dataset, its performance on UAVRailSet2023, a dataset of railway images collected from a UAV, is considerably less satisfactory. The dataset is divided into anomalous and non-anomalous data, with the anomalous data comprising images depicting foreign debris occluding crucial railway components, such as ballast rocks on fasteners or vegetation on the railway track, with some examples shown in figure 4.27. figure 4.26 demonstrates the resulting distributions for anomalous and non-anomalous data.

One key factor to the suboptimal performance is the comparatively low volume of data available in this dataset, having less than 1,000 samples, which may hinder the model's ability to generalize effectively across diverse scenarios. Moreover, the dataset's predominant unstructured nature poses a significant challenge. The variability in lighting conditions, camera angles, and environmental factors specific to UAV-captured images introduces complexity not fully accounted for during model training on more structured datasets, as a result, this requires even more data for the reconstruction model to obtain a good representation of what a complete segmentation label should look like from different viewpoints. This lack of environmental diversity may result in a limited ability of ADeReL to discern and adapt to the nuanced characteristics of anomalous instances in the UAVRailSet2023 dataset.

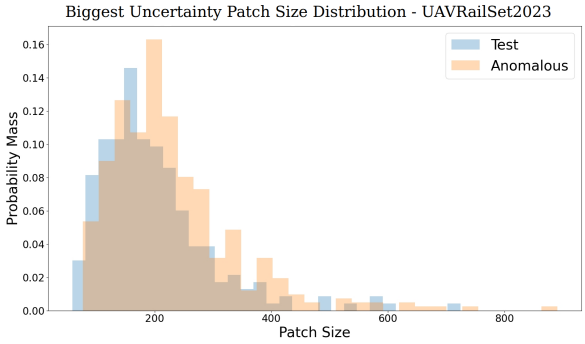

Figure 4.26: Distribution of maximum uncertainty patch size for both testing and anomalous data on the UAVRailSet2023 dataset.

Figure 4.27: Various examples of images of anomalous data from the anomalous set of UAVRailSet2023.

4.7 Chapter Results and Lessons Learned

In this chapter, we started off exploring the detection of specific maintenance needs (e.g. missing fasteners, kinked rail) using computer vision when presented with images of the railway under ideal angles. Given the extensive amount of existing work in this direction, it is unsurprising that our various suggested approaches also work reasonably well. Unfortunately, images captured from a UAV platform are unlikely to have consistent angles. In addition, maintenance needs on mass infrastructure systems such as railways tend to be few and far between, resulting in a low number of known examples in contrast to the many ways that components can fail. This makes it very difficult to design a catch-almost-all maintenance needs detector using heuristic methods. To this end, we introduced the notion of anomaly detection under the premise that this is the better approach for maintenance needs detection on railways. One common trend for performing anomaly detection in literature is to use simple pixel-level reconstruction. We demonstrate, using a toy example, how this can fail in a very trivial manner when attempting to apply it to a railway. This result motivates the push for uncertainty quantification techniques instead.

The chapter then explores two principle methods for performing uncertainty quantification on images in the pixel space on two different datasets. Despite seemingly performing well, we demonstrate that these uncertainty quantification techniques are paying no heed to the overall structure of the image, and are instead relying on local textural information at inference. This motivates the chapter into presenting ADeReL, an algorithm for performing anomaly detection using uncertainty quantification in the label space. Our experiments show

that ADeReL can work reasonably well for datasets with reasonably structured data, but fail on our own collected dataset of UAV collected railway images. One reason this may be the case is that the size of the dataset used to train ADeReL is insufficient for the underlying semantic segmentation model to learn the structure of the images.

At the time of writing, the notion of performing anomaly detection on railways using multiple viewpoint images gathered from a UAV is still unsolved. Anomaly detection for domains with very diverse data, particularly in the domain of deep learning, poses unique challenges due to the scarcity of labeled anomalous data. Traditional supervised learning approaches are hindered by the limited availability of such data, leading to sub-optimal model performance. This leads to difficulty in models differentiating between anomalous data and previously unseen transformations of normal data. This is especially evident in the domain of maintenance needs detection in spaces like railway monitoring. One promising direction for future research to solve this problem is active learning. Active learning emerges as a promising solution to address this issue by strategically selecting and labeling the most informative instances. The core idea of active learning relies on the model automatically labelling instances of low uncertainty to be used in further iterative training, while flagging instances of high uncertainty for expert labelling, where the expert here usually refers to a human. More precisely, active learning starts off with an initial dataset of images $\mathcal{D}_1$ with samples $\{(x_1, y_1), (x_2, y_2), ...\}$ on which an uncertainty aware model $f_\theta : x \rightarrow (\hat{y}, \epsilon)$ is trained on. The model is then deployed to the field, where it interacts with new, unseen data $\{x_a, x_b\} \notin \mathcal{D}_1$. For instances of data producing uncertainty scores lower than a set threshold, the labels produced by the model are accepted as valid ground truths to produce a set of new training instances $\mathcal{D}_{\text{certain}} = \{(x_a, \hat{y}_a), (x_b, \hat{y}_b), ...\}$. Instances of data that produce uncertainty scores higher than said threshold are put into a separate dataset, where an external expert (a human) is used to label the data to create new training instances $\mathcal{D}_{\text{uncertain}} = \{(x_p, \bar{y}_p), (x_q, \bar{y}_q), ...\}$. All instances are then added to the initial dataset to form a larger training set $\mathcal{D}_2 = \mathcal{D}_1 \cup \mathcal{D}_{\text{certain}} \cup \mathcal{D}_{\text{uncertain}}$. Finally, the original trained model can then be fine-tuned on this new model, or a completely new model retrained on this dataset. This process is then repeated as long as the model is in deployment. The idea here is that as time progresses and the training dataset gets larger, the developing dataset $(\mathcal{D}_1, \mathcal{D}_2, ...)$ slowly encompasses all data available from the deployed setting. More interestingly, active learning would be the foundation for building foundational models, which would unquestionably be a building block for performing much more complex tasks around railway infrastructure.

In the future, we propose to take inspiration from the realm of language modelling and foundational models. ADeReL relies on a model learning the general structure of normal data under many different image transformations or camera viewpoints. Given sufficient data, it is possible to construct foundational vision models that can label railway track components under any viewpoint, and by extension, it is possible to construct foundational models that can perform reconstruction of image labels for railway images under any viewpoint. Following this train of thought, it is then possible to detect anomalies using the combination of these two models in the way that ADeReL proposes.

5 Conclusion and Future Work

This research is motivated by the importance of maintaining operational reliability of key systems like railways, which face various challenges such as missing fasteners, deformed track geometry, structural health issues, and overgrown vegetation that can lead to substantial downtime. Traditional inspection methods, though comprehensive, are resource-intensive and disruptive. Recognizing the potential for innovation, this study investigates the application of autonomously operated Unmanned Aerial Vehicles (UAVs) for railway maintenance needs detection, and addresses various challenges to its success. This chapter summarizes the key contributions made by each chapter in this thesis, points out the existing limitations, and discusses the possible future research directions.

5.1 Conclusion

In this thesis, the current methodologies and status quo for detecting maintenance needs on railway tracks was explored. The idea of performing the same detection of maintenance issues using AI-powered UAVs was then explored. This started in chapter 3 on page 51, where a proposal for performing UAV navigation utilizing reinforcement learning was introduced. Reinforcement learning is a learned technique, which differs from current methods which primarily utilize heuristic, hard-coded techniques for navigation in the real world. Although this paradigm is not new, its application to railway tracks presents its own sets of challenges, asserting its novelty. Some of these challenges include the need for better simulation tools catered to reinforcement learning software stacks, bridging the Sim2Real gap, and better reinforcement learning algorithms to speed up the bootstrapping phase of classical reinforcement learning. To this end, this thesis proposed a new reinforcement learning algorithm termed Critic Confidence Guided Exploration (CCGE) for allowing agents to bootstrap off a suboptimal oracle policy using uncertainty metrics in section 3.6 on page 72. This effectively allows the reinforcement learning agent to learn much faster by mimicking the oracle early on in training, circumventing much of early exploration steps. A new UAV simulation tool — PyFlyt — has also been developed and released as part of this work in section 3.4 on page 55. PyFlyt is specifically catered to the reinforcement learning research community, and integrates well with many existing tools. In addition, due to the Python interface, this allows users to rapidly construct custom environments for their research. Solving the navigation issue is only solving half the challenge, as just flying along a railway track does not automatically allow one to identify maintenance needs. In chapter 4 on page 101, we explored how such a platform can enable the automatic detection of maintenance needs. We investigated possi-

ble maintenance needs that can be detected using a UAV, and found that the main strength of a UAV lies in its ability to provide high resolution imagery of the railway from many different points of view. Detecting well-defined maintenance issues from a fixed point of view can be done by leveraging the overall image structure and carefully crafted algorithms. We demonstrated this by utilizing deep learning backbones and heuristics to detect fasteners (or the lack thereof), and sun kinks on railway tracks. While this worked, we discovered that the real challenge of railway track maintenance needs detection lies in the detection of previously unseen component failures, which are few and far in between. This necessitated more sophisticated algorithms than ones performing defect detection via pure supervised learning. We proposed anomaly detection, but immediately discovered that the classical method of performing anomaly detection via reconstruction fails trivially on a task as diverse as railway track defect detection. We further discovered how attempting to perform anomaly detection via uncertainty quantification also fails as a result of models being sensitive to texture instead of image structure. These two lessons led to the proposal of a new algorithm, which we call ADeReL, for performing anomaly detection by reconstructing image labels. The algorithm uses two networks, one to predict labels, and another to reconstructed the corrupted predicted labels. In principle, such a framework allows for the prediction of anomalies without explicit anomaly labels within the training dataset. Applying this algorithm to a custom dataset of railway images reveals limited successes due to the limited size of the dataset, but we are optimistic in its performance in the future.

5.2 Contributions

We propose four novel contributions to the field towards autonomous UAVs for maintenance needs detection on railways based on work done in this thesis.

1. *PyFlyt*

   PyFlyt is a UAV flight simulator for RL research. Its conception was motivated by the lack of easy-to-use, hackable and standardized UAV simulators in the field that specifically cater for the needs of RL research and machine learning at large. To this end, PyFlyt is written using the tried-and-trusted Bullet physics engine with a Python interface. It provides various default vehicles, such as two variants of quadrotor UAVs, a fixedwing aircraft, and a rocket model. Each of these models are built using modularized components, such as motors, gimbals, and lifting surfaces to name a few. PyFlyt also caters for running RL agents at different looprates to the underlying UAV's control loop, which is again at a different looprate to the physic's simulator's stepping rate. Various default Gymnasium environments exist within PyFlyt, both with sparse and dense reward structures. The design of PyFlyt is used to build the Rail Environment, on which a RL algorithm was trained to fly a UAV along a railway.

2. *Critic Confidence Guided Exploration*

A large part of modern RL involves training agents from scratch on carefully crafted dense reward functions. However, in some scenarios, it is relatively easy to design a sub-optimal hard-coded policy. In other cases, sparse reward functions may be too difficult to learn but produce optimal policies; while dense reward functions are easier to learn and produce sub-optimal policies. We propose a novel RL algorithm, termed Critic Confidence Guided Exploration (CCGE), that effectively allows RL agents to leverage prior information from an oracle policy to effectively explore their environment, bypassing the requirement to learn from scratch. When evaluated on several standardized RL benchmarks, we demonstrated that CCGE performs comparatively to de factor RL algorithms in some cases, while outperforming them in others.

3. *UAV Autonomous Navigation*

   Combining the work from the prior two contributions, we develop a framework for flying a UAV down a railway corridor using mostly learned methods. We propose training a semantic segmentation model trained on real world images to segment out relevant components in images of a railway scene gathered from a UAV. The semantic segmentation labels we chose in the instance included railway track lines and obstacles, serving as a healthy start to possibly more complex architectures. Then, we train an agent using CCGE to navigate a UAV down a railway corridor within PyFlyt based on semantic segmentation images obtained directly from the simulation. This architecture is then ported into a real UAV and tested on a railway track at BCIMO's facility in Dudley, United Kingdom. Our experiments show that this framework works reasonably well, and serves as a good starting point for possibly more complex learned behaviours.

4. *Maintenance Needs Detection — Structured Problems*

   We demonstrate several algorithms for performing basic maintenance needs detection on railways. Most notably, these first few techniques rely on the structure of the problem space to motivate solutions. For instance, for finding missing fasteners, we propose a framework where the number of fasteners per distance is computed, and areas where the number deviates from the intended value indicates a missing fastener. For sun kink detection, we propose having a semantic segmentation model segment railway tracks within an image, and then fitting a second order curve to the railway tracks. Any fit that exhibits higher than expected fitness indicates a possible kink.

5. *Maintenance Needs Detection — Anomaly Detection*

   Fortunately for public transportation systems and unfortunately for machine learning algorithms, defects along railway corridors are rare. Unfortunately, because of this, classical supervised learning for defect detection is difficult to perform for railway systems because crafting a well balanced dataset that encompasses all modes of component failures is not easy. In addition, components may also fail in ways not

initially captured in the training dataset. To this end, we propose that anomaly detection is a more suitable technique for maintenance needs detection. In the context of a railway corridor, we demonstrated why the usual route of reconstruction-based anomaly detection may not work for the environment of a railway corridor, and further experiment with two popular methods for performing anomaly detection using uncertainty-driven frameworks. Our experiments show that, for an environment with a high degree of variance, both techniques tend to not work because the underlying models are basic semantic segmentation models. In this case, the models adopt a dimpled manifold principle approach to semantic segmentation, and simply ignore patches of the image that it doesn't recognize, instead of attempting to label it based on surrounding context. To alleviate this issue, we proposed a framework where we perform image reconstruction within the label space, which we term Anomaly Detection by Reconstruction Labels (ADeReL). Our idea stems from the notion that, for the task of anomaly detection along a railway corridor, the label space is fairly structured — fasteners are placed at intersections of sleepers and tracks; tracks tend to be continuous segments; and sleepers tend to be evenly spaced. Our experiments show that, for a dataset with a sufficiently structured viewpoint, this can work. However, on a smaller dataset of UAV collected imagery, the images exhibit too much variance for this ADeReL to show any meaningful use for anomaly detection, and the pursuit of anomaly detection on this dataset is left to future work.

5.3 Addressing Research Questions

This thesis aimed to answer various research questions surrounding the topic of autonomous UAVs for railway maintenance monitoring with the aid of artificial intelligence systems. These questions were first listed on section 1.3 on page 2. Here, we address these questions using the presented work and propose several new ones for future research.

What current practices are being used to detect maintenance issues along railway tracks and what existing gaps can be addressed using AI-enabled UAV technologies? In section 2.5 on page 43, we observed that the status quo of current maintenance methods involved sensor instrumented carriages going down the railway, in order to perform maintenance needs detection, or for ground sensors to be placed around railway infrastructure to monitor track conditions. In the case of running a carriage down the railway track, inspection can only happen as many times as the carriage goes down the track. In the case of ground sensors, costs scale linearly with the distances that the railway track covers. UAVs offer a good middle ground where maintenance can happen as many times as required, while not requiring extensive sensor networks to be placed all over the track. However, we also observed that not all maintenance needs can be detected from a UAV platform due to the nature of sensors required.

Are learned obstacle avoidance and navigation methods feasible to be used for navigating UAVs through railway corridors? This research question was answered in chapter 3 on page 51. The answer is yes, however, modern deep learning algorithms have reached a point where the various technologies and techniques are sufficient for deploying a reinforcement learning agent to the real world aboard a UAV. The main drawback of RL is the requirement for an agent to continuously interact in an environment. We show, in section 3.4 on page 55 to section 3.7 on page 88, one feasible approach is to design an environment in simulation for an agent to fly a UAV along a railway corridor. Then, in section 3.8 on page 91, we demonstrate how image segmentation pipelines can be used to reduce the gap between simulation and reality, sufficient for allowing an RL agent to be adapted for the real world after being trained in simulation. Our results in section 3.9 on page 96 are limited, but demonstrate a promising approach forward for learned algorithms to be used for autonomous flight of UAVs.

What challenges present themselves when trying to perform maintenance monitoring on railways using machine learning from data gathered aboard a UAV and how can they be solved? The main challenge of performing maintenance detection is the imbalanced dataset problem. While it is easy to gather a lot of normal data (data where there are no broken missing components) for mass infrastructure systems, getting an equal amount of abnormal data (data with broken or missing components) for the purpose of detecting abnormality is hard. This is primarily down to the fact that mass infrastructure systems are designed with robustness and low failure rates in mind. For this reason, we argue in section 4.5 on page 116 and proceeding subsections that a better way of detecting maintenance needs on railway systems is to instead perform anomaly detection. The classical technique of measuring pixel-level reconstruction error for anomaly detection was first attempted, and we demonstrate through a toy example that this is unlikely to work for images gathered from a UAV. We also attempted to use techniques from the field of uncertainty quantification to perform anomaly detection and demonstrate its failure modes for this space of problems. Finally, in section 4.6 on page 129, we introduce a new method for performing anomaly detection via reconstruction in the label space and demonstrate some promise in performing anomaly detection on railway infrastructure. Despite this, the field of anomaly detection, and therefore the detection of maintenance needs, from a UAV is still an open problem.

5.4 Limitations and Future Work

Our work here represents a small glimpse into the possibilities of autonomous UAVs for automated infrastructure maintenance needs detection and monitoring. It exhibits multiple limitations and avenues for improvement before a completely autonomous UAV for infrastructure maintenance needs detection is possible.

PyFlyt While suitable as a quick environment for training UAV-based RL agents for a multitude of tasks, PyFlyt's main downside is due to its simplistic visual design. Because of this reason, photorealistic environments are not possible within the simulation, and we recommend using the native segmentation image instead when training computer vision models. In addition, while PyFlyt incorporates a multitude of dynamics modelling taken from past literature, many of these components are uncalibrated with real world results, and performing such calibration would be paramount for taking PyFlyt from a toy simulation tool to a full Software-In-The-Loop (SITL) simulator.

Critic Confidence Guided Exploration While our work on CCGE represents a novel idea of using uncertainty to guide exploration, the results can be viewed by some to be lacklustre when compared to existing vanilla RL algorithms. This is especially true in light of other RL algorithms that have been released during the time of CCGE work, such as the many algorithms in the realm of offline RL or hybrid offline-online RL. In the future, we would like to explore the idea of incorporating offline data into CCGE using uncertainty metrics, and perhaps also incorporate distributional value functions as a critic for a more holistic view of action-space uncertainty.

UAV Autonomous Navigation Arguably the biggest improvement to this research work can be had in this area. Our experiments represent a very small subset of what's possible using autonomous RL algorithms for UAV navigation. For instance, our algorithm utilizes a very narrow action space, with no notion of allowing the UAV to control its own velocity outside of a stop-go command. In addition, the reward function we utilized for training the RL agent does not incorporate the view-quality of the dataset. Incorporating these aspects into the reward function will be an interesting research avenue, allowing the UAV to autonomously backtrack to have a better look at certain components. Our implementation also does not look at cases where the railway track splits into multiple directions; how to handle many electro-mechanical aspects such as flight time, charging plans, and weather resistant operation; operating during live railway operations with oncoming trains; and multi-UAV scenarios. All of these fall under the long tail of issues that will have to be solved before such a system will be deemed suitable for real world deployment.

Maintenance Needs Detection The biggest limitation in our work on structured problem maintenance needs detection was the lack of openly available large scale datasets. There have been many works on utilizing deep learning for maintenance needs detection on railways. However, despite the extensive network of railways around the world, almost all of these works utilize proprietary datasets. This is a huge limiting factor, not only because benchmarking of algorithms with each other is not feasible in this manner, but also because no algorithm can claim to work well in almost all scenarios due to the wide range of environments that railway corridors exist in — the United Kingdom's environments are typically tracks on sleepers held with fasteners set in forested regions; those in China tend to be steel tracks held by fasteners on concrete bases; some in regions of Europe tend to be grooved

tracks on city streets. To compound on this factor, the task of collecting a holistic dataset can be exacerbated by the variety of camera settings that can be used, not considering the range of camera angles and positions relative to the railway possible. The presence of good datasets is the grounding factor for good machine learning models. This is evidenced by the surge of foundational Large Language Models and computer vision models. Tackling this issue for the railway maintenance industry would undoubtedly bring a lot of value for the field.

5.5 Closing Remarks

This research work involved the study of artificial intelligence enabled autonomous UAV's to perform maintenance needs detection on railways. In many ways, this research work is highly exploratory, and many non-technical issues must first be solved before a future of autonomous, maintenance monitoring UAVs are deployed in the real world. However, this research work shows that there are definite benefits to the idea, and that there are a multitude of interesting challenges to be solved. By pursuing this idea, we have also presented a number of novel algorithms which are potentially interdisciplinary to research, and not solely confined to railway monitoring using UAVs. We hope that this inspires more research in this direction, and eagerly await the future of faster, more efficient and safer maintenance monitoring schemes for mass infrastructure systems.

Bibliography

AM Abdelraouf, OK Mahmoud, and MA Al-Sanabawy. Thrust termination of solid rocket motor. In *Journal of Physics: Conference Series*, volume 2299, page 012018. IOP Publishing, 2022. 61

Sarder Fakhrul Abedin, Md Shirajum Munir, Nguyen H Tran, Zhu Han, and Choong Seon Hong. Data freshness and energy-efficient uav navigation optimization: A deep reinforcement learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 22(9): 5994–6006, 2020. 31

Feras Abushakra, Nathan Jeong, Deepak N Elluru, Abhishek K Awasthi, Shriniwas Kolpuke, Tuan Luong, Omid Reyhanigalangashi, Drew Taylor, and S Prasad Gogineni. A miniaturized ultra-wideband radar for uav remote sensing applications. *IEEE Microwave and Wireless Components Letters*, 32(3):198–201, 2021. 14

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021. 80

Hirotogu Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected papers of hirotugu akaike*, pages 199–213. Springer, 1998. 30

Iretiayo Akinola, Zizhao Wang, and Peter Allen. Clamgen: Closed-loop arm motion generation via multi-view vision-based rl. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2376–2382. IEEE, 2021. 65

Omar Alam and Jörg Kienzle. Designing with inheritance and composition. In *Proceedings of the 3rd international workshop on Variability & Composition*, pages 19–24, 2012. 58

Zena Abd Alrahman and Ali Adham. A review on existing technologies used in the maintenance of railway infrastructure. In *AIP Conference Proceedings*, volume 3079. AIP Publishing, 2024. 1

Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. *Advances in Neural Information Processing Systems*, 33: 14927–14937, 2020. URL https://proceedings.neurips.cc/paper/2020/file/aab085461de182608ee9f607f3f7d18f-Paper.pdf. 46, 75, 119

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. In *ICLR 2021-Ninth International Conference on Learning Representations*, 2021. 28

André Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 4(11):e21, 2019. 107

Hanum Arrosida, Indonesia Agus Susanto, Adiratna Ciptaningrum, Tyan Rudianti, Masayu Nazar Surya Kencana, and Rizal Mahmud. Rail line surfaces defect monitoring using yolo architecture: Case study on madiun-magetan track, east java. 48

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. 40

Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995. 73

Farouk Balouchi, Adam Bevan, and Roy Formston. Development of railway track condition monitoring from multi-train in-service vehicles. *Vehicle System Dynamics*, 59(9):1397–1417, 2021. 7

Yuequan Bao, Zhiyi Tang, Hui Li, and Yufeng Zhang. Computer vision and deep learning–based data anomaly detection method for structural health monitoring. *Structural Health Monitoring*, 18(2):401–421, 2019. 49

F Barato, N Bellomo, M Faenza, M Lazzarin, A Bettella, and D Pavarin. Numerical model to analyze transient behavior and instabilities on hybrid rocket motors. *Journal of Propulsion and Power*, 31(2):643–653, 2015. 61

A. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. *arXiv preprint arXiv:1904.01416*, 2019. 35

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017. 75

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław D kebiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 32

Maurizio Bevilacqua and Marcello Braglia. The analytic hierarchy process applied to maintenance strategy selection. *Reliability Engineering & System Safety*, 70(1):71–83, 2000. 5

Sumana Biswas, Sreenatha G Anavatti, and Matthew A Garratt. A particle swarm optimization based path planning method for autonomous systems in unknown terrain. In *2019 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, pages 57–63. IEEE, 2019. 17, 20

William R Bitman. Balancing software composition and inheritance to improve reusability, cost, and error rate. *Johns Hopkins APL Technical Digest*, 18(4):485–500, 1997. 58

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020. 39

Eric Boivin, André Desbiens, and Eric Gagnon. Uav collision avoidance using cooperative predictive control. In *2008 16th Mediterranean Conference on Control and Automation*, pages 682–688. IEEE, 2008. 17, 20

Davide Bombarda, Giorgio Matteo Vitetta, and Giovanni Ferrante. Rail diagnostics based on ultrasonic guided waves: an overview. *Applied Sciences*, 11(3):1071, 2021. 6

Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3722–3731, 2017. 31

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. 80, 179

André Broekman and Petrus Johannes Gräbe. Railenv-pasmvs: A perfectly accurate, synthetic, path-traced dataset featuring a virtual railway environment for multi-view stereopsis training and reconstruction applications. *Data in Brief*, 38:107411, 2021. 103

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020. 9

Duc Hong Phuc Bui, Sung Kyung Hong, et al. Nonlinear control for autonomous trajectory tracking while considering collision avoidance of uavs based on geometric relations. *Energies*, 12(8):1–20, 2019. 17, 20

Jonathan Bull. Latitude and longitude of all railway stations, 2021. URL `https://www.whatdotheyknow.com/request/latitude_and_longitude_of_all_ra`. Accessed: 12 Dec 2022. 102

H. Caesar, J. Uijlings, and V. Ferrari. Coco-stuff: Thing and stuff classes in context. *arXiv preprint arXiv:1612.03716*, 2017. 35

Thyago P Carvalho, Fabrízzio AAMN Soares, Roberto Vita, Roberto da P Francisco, João P Basto, and Symone GS Alcalá. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 137:106024, 2019. 10

Juan Manuel Castillo-Mingorance, Miguel Sol-Sánchez, Fernando Moreno-Navarro, and María Carmen Rubio-Gámez. A critical review of sensors for the continuous monitoring of smart and sustainable railway infrastructures. *Sustainability*, 12(22):9428, 2020. 7

Zhaohui Cen, Tim Smith, Paul Stewart, and Jill Stewart. Integrated flight/thrust vectoring control for jet-powered unmanned aerial vehicles with acheon propulsion. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 229(6): 1057–1075, 2015. 63

L Chang, RPBJ Dollevoet, and RF Hanssen. Railway infrastructure monitoring using satellite radar data. *Int. J. Railw. Technol*, 3:79–91, 2014. 6

Ling Chang, Rolf PBJ Dollevoet, and Ramon F Hanssen. Nationwide railway monitoring using satellite sar interferometry. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(2):596–604, 2016. 6

Bertrand Charpentier, Daniel Zügner, and Stephan Günnemann. Posterior network: Uncertainty estimation without ood samples via density-based pseudo-counts. *Advances in Neural Information Processing Systems*, 33:1356–1367, 2020. 75

Bertrand Charpentier, Oliver Borchert, Daniel Zügner, Simon Geisler, and Stephan Günnemann. Natural posterior network: Deep bayesian predictive uncertainty for exponential family distributions. *arXiv preprint arXiv:2105.04471*, 2021. 75

Bertrand Charpentier, Ransalu Senanayake, Mykel Kochenderfer, and Stephan Günnemann. Disentangling epistemic and aleatoric uncertainty in reinforcement learning. *arXiv preprint arXiv:2206.01558*, 2022. 77

A Chayeb, Noureddine Ouadah, Z Tobal, Mustapha Lakrouf, and Ouahiba Azouaoui. Hog based multi-object detection for urban navigation. In *17th international IEEE conference on intelligent transportation systems (ITSC)*, pages 2962–2967. IEEE, 2014. 33

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019. 30

Jianqing Chen and Jiyan Yu. An improved path planning algorithm for uav based on rrt. In *2021 4th international conference on advanced electronic materials, computers and software engineering (AEMCSE)*, pages 895–898. IEEE, 2021. 17, 20

Jin Chen, Zhi Lyu, Yuyuan Liu, Jiayang Huang, Guigang Zhang, Jian Wang, and Xi Chen. A big data analysis and application platform for civil aircraft health management. In *2016 IEEE Second International Conference on Multimedia Big Data (BigMM)*, pages 404–409. IEEE, 2016a. 8

Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4974–4983, 2019a. 41, 48

Richeng Chen, Yunzhi Lin, and Tao Jin. High-speed railway pantograph-catenary anomaly detection method based on depth vision neural network. *IEEE Transactions on Instrumentation and Measurement*, 71:1–10, 2022. 49

Siyuan Chen, Debra F Laefer, Eleni Mangina, Iman Zolanvari, and Jonathan Byrne. Uav bridge inspection through evaluated 3d reconstructions. *Journal of Bridge Engineering*, 24, 2019b. 1

Xiaocong Chen, Lina Yao, and Yu Zhang. Residual attention u-net for automated multi-class segmentation of covid-19 chest ct images. *arXiv preprint arXiv:2004.05645*, 2020. 93

Xudong Chen, Swee King Phang, Mo Shan, and Ben M Chen. System integration of a vision-guided uav for autonomous landing on moving platform. In *2016 12th IEEE International Conference on Control and Automation (ICCA)*, pages 761–766. IEEE, 2016b. 13

Yicheng Chen and Lingling Wang. Adaptively dynamic rrt*-connect: Path planning for uavs against dynamic obstacles. In *2022 7th International Conference on Automation, Control and Robotics Engineering (CACRE)*, pages 1–7. IEEE, 2022. 17, 20

Dan C Cireşan, Ueli Meier, Luca M Gambardella, and Jürgen Schmidhuber. Handwritten digit recognition with a committee of deep neural nets on gpus. *arXiv preprint arXiv:1103.4487*, 2011. 36

William R Clements, Bastien Van Delft, Benoît-Marie Robaglia, Reda Bahi Slaoui, and Sébastien Toth. Estimating risk and uncertainty in deep reinforcement learning. *arXiv preprint arXiv:1905.09638*, 2019. 77

Y Cong, Z Zhao, C Xing, and Z Wang. Dynamic obstacle avoidance path planning of uav based on improved artificial potential field. *Journal of Ordnance and Equipment Engineering*, 42:170–176, 2021. 16, 20

M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. *arXiv preprint arXiv:1604.01685*, 2016. 35

Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, page 1. 2015. 56

Jin Q Cui, Swee King Phang, Kevin ZY Ang, Fei Wang, Xiangxu Dong, Yijie Ke, Shupeng Lai, Kun Li, Xiang Li, Feng Lin, et al. Drones for cooperative search and rescue in post-disaster situation. In *2015 IEEE 7th international conference on cybernetics and intelligent systems (CIS) and IEEE conference on robotics, automation and mechatronics (RAM)*, pages 167–174. IEEE, 2015. 13

Jing Cui, Yong Qin, Yunpeng Wu, Changhong Shao, and Huaizhi Yang. Skip connection yolo architecture for noise barrier defect detection using uav-based images in high-speed railway. *IEEE Transactions on Intelligent Transportation Systems*, 2023. 47

Yago MR da Silva, Fabio AA Andrade, Lucas Sousa, Gabriel GR de Castro, João T Dias, Guido Berger, José Lima, and Milena F Pinto. Computer vision based path following for autonomous unmanned aerial systems in unburied pipeline onshore inspection. *Drones*, 6(12):410, 2022. 21

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR, 2018. 75

L d'Agostino, L Biagioni, and G Lamberti. An ignition transient model for solid propellant rocket motors. In *37th Joint Propulsion Conference and Exhibit*, page 3449, 2001. 61

Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016. 38

Narjes Davari, Bruno Veloso, Rita P Ribeiro, Pedro Mota Pereira, and João Gama. Predictive maintenance based on anomaly detection using deep learning for air production unit in the railway industry. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2021. 50

Lorenzo De Donato, Francesco Flammini, Stefano Marrone, Claudio Mazzariello, Roberto Nardone, Carlo Sansone, and Valeria Vittorini. A survey on audio-video based defect detection through deep learning in railway maintenance. *IEEE Access*, 2022. 50

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines, 2017. 73

Maria Di Summa, Maria Elena Griseta, Nicola Mosca, Cosimo Patruno, Massimiliano Nitti, Vito Renò, and Ettore Stella. A review on deep learning techniques for railway infrastructure monitoring. *IEEE Access*, 2023. 110

Cora A Dimmig, Giuseppe Silano, Kimberly McGuire, Chiara Gabellieri, Wolfgang Hšnig, Joseph Moore, and Marin Kobilarov. Survey of simulators for aerial robots: An overview and in-depth systematic comparisons. *IEEE Robotics & Automation Magazine*, 2024. 57

Fangfang Dong, Dengyang Wu, Chenying Guo, Shuting Zhang, Bailin Yang, and Xiangyang Gong. Craunet: A cascaded residual attention u-net for retinal vessel segmentation. *Computers in Biology and Medicine*, 147:105651, 2022. 93

Taha Elmokadem. A 3d reactive navigation method for uavs in unknown tunnel-like environments. In *2020 Australian and New Zealand Control Conference (ANZCC)*, pages 119–124. IEEE, 2020. 21

Taha Elmokadem and Andrey V Savkin. A method for autonomous collision-free navigation of a quadrotor uav in unknown tunnel-like environments. *Robotica*, 40(4):835–861, 2022. 21

Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 201–208, 2005. 74

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. 35

Amir Falamarzi, Sara Moridpour, and Majidreza Nazem. A review on existing sensors and devices for inspecting railway infrastructure. *Jurnal Kejuruteraan*, 31(1):1–10, 2019. 7

Behzad V Farahani, Francisco Barros, Pedro J Sousa, Paulo J Tavares, and Pedro MGP Moreira. A railway tunnel structural monitoring methodology proposal for predictive maintenance. *Structural Control and Health Monitoring*, 27(8):e2587, 2020. 115

András Farkas. Measurement of railway track geometry: A state-of-the-art review. *Periodica Polytechnica Transportation Engineering*, 48(1):76–88, 2020. 6

Jiang Hua Feng, Hao Yuan, Yun Qing Hu, Jun Lin, Shi Wang Liu, and Xiao Luo. Research on deep learning method for rail surface defect detection. *IET Electrical Systems in Transportation*, 10(4):436–442, 2020. 48

Paul Festor, Giulia Luise, Matthieu Komorowski, and A Aldo Faisal. Enabling risk-aware reinforcement learning for medical interventions through uncertainty decomposition. *arXiv preprint arXiv:2109.07827*, 2021. 77

Julian Förster. System identification of the crazyflie 2.0 nano quadrocopter. B.S. thesis, ETH Zurich, 2015. 66

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL http://github.com/google/brax. 28

Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017. 108

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020. 80

Qiang Fu, Xinghui Lan, Yuanfa Ji, Xiyan Sun, and Fenghua Ren. Heuristic rrt fusion a* for 3d path planning of uav. In *2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC)*, volume 6, pages 1433–1443. IEEE, 2022. 17, 20

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018. 25

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4): 193–202, 1980. 11

Paulina Gackowiec. General overview of maintenance strategies–concepts and approaches. *Multidisciplinary Aspects of Production Engineering*, 2(1):126–139, 2019. 7

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016. 74

Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3948–3955. IEEE, 2017. 32

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016. 31

Richard Garcia and Laura Barnes. Multi-uav simulator utilizing x-plane. In *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009*, pages 393–406. Springer, 2010. 55

Riccardo Gasparini, Stefano Pini, Guido Borghi, Giuseppe Scaglione, Simone Calderara, Eugenio Fedeli, and Rita Cucchiara. Anomaly detection for vision-based railway inspection. In *Dependable Computing-EDCC 2020 Workshops: AI4RAILS, DREAMS, DSOGRI, SERENE 2020, Munich, Germany, September 7, 2020, Proceedings 16*, pages 56–67. Springer, 2020. 44, 49

Riccardo Gasparini, Andrea D'Eusanio, Guido Borghi, Stefano Pini, Giuseppe Scaglione, Simone Calderara, Eugenio Fedeli, and Rita Cucchiara. Anomaly detection, localization and classification for railway inspection. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 3419–3426. IEEE, 2021. 44, 49

Shengyang Ge, Feng Pan, Dadong Wang, and Pu Ning. Research on an autonomous tunnel inspection uav based on visual feature extraction and multi-sensor fusion indoor navigation system. In *2021 33rd Chinese Control and Decision Conference (CCDC)*, pages 6082–6089. IEEE, 2021. 22

A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. The kitti vision benchmark suite. *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 846–853, 2012. 35

Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. Why generalization in rl is difficult: Epistemic pomdps and implicit partial observability. *Advances in Neural Information Processing Systems*, 34:25502–25515, 2021. 74

Xavier Gibert, Vishal M Patel, and Rama Chellappa. Robust fastener detection for autonomous visual railway track inspection. In *2015 IEEE Winter Conference on Applications of Computer Vision*, pages 694–701. IEEE, 2015. 107

Xavier Gibert, Vishal M Patel, and Rama Chellappa. Deep multitask learning for railway track inspection. *IEEE transactions on intelligent transportation systems*, 18(1):153–164, 2016. 107

Augusto Gómez Eguíluz, Julio Lopez Paneque, José Ramiro Martínez-de Dios, and Aníbal Ollero. Online detection and tracking of pipes during uav flight in industrial environments. In *Robot 2019: Fourth Iberian Robotics Conference: Advances in Robotics, Volume 1*, pages 28–39. Springer, 2020. 21

Wendong Gong, Muhammad Firdaus Akbar, Ghassan Nihad Jawad, Mohamed Fauzi Packeer Mohamed, and Mohd Nadhir Ab Wahab. Nondestructive testing technologies for rail inspection: a review. *Coatings*, 12(11):1790, 2022. 1

Xiang Gong and Wei Qiao. Current-based mechanical fault detection for direct-drive wind turbines via synchronous sampling and impulse detection. *IEEE Transactions on Industrial Electronics*, 62(3):1693–1702, 2014. 5

Liang Gonog and Yimin Zhou. A review: generative adversarial networks. In *2019 14th IEEE conference on industrial electronics and applications (ICIEA)*, pages 505–510. IEEE, 2019. 31

D Greatrix. Acceleration-based combustion augmentation modelling for noncylindrical grain solid rocket motors. In *31st Joint Propulsion Conference and Exhibit*, page 2876, 1995. 61

G. Griffin, A. Holub, and P. Perona. The caltech-256 object category dataset. In *Computer Vision and Pattern Recognition Workshop (CVPRW)*, pages 1–8, 2007. 35

Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012. 73

Emre Guclu, Ilhan Aydin, and Erhan Akin. Development of vision-based autonomous uav for railway tracking. In *2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pages 120–125. IEEE, 2021. 22

Edmundo Guerra, Rodrigo Munguía, Yolanda Bolea, and Antoni Grau. Detection and positioning of pipes and columns with autonomous multicopter drones. *Mathematical Problems in Engineering*, 2018, 2018. 21

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018. 29

Le Nhu Ngoc Thanh Ha, Duc Hong Phuc Bui, and Sung Kyung Hong. Nonlinear control for autonomous trajectory tracking while considering collision avoidance of uavs based on geometric relations. *Energies*, 12(8):1551, 2019. 16, 20

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018a. 74

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018b. 25, 74, 78, 80

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018c. 25

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2019. 29

Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2020. 29

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4 (2):100–107, 1968. 16

Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010. 78

K. He, X. Zhang, S. Ren, and J. Sun. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 59(5):84–90, 2015. 34

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 11, 33, 36, 107

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 38, 41

Yongxiang He, Jun Wu, Yaojia Zheng, Yuxin Zhang, and Xiaobo Hong. Track defect detection for high-speed maglev trains via deep learning. *IEEE Transactions on Instrumentation and Measurement*, 71:1–8, 2022. 48

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998. 108

Max Holmberg, Oskar Karlsson, and Michael Tulldahl. Lidar positioning for indoor precision navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 359–368, 2022. 16, 20

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 10

Emil Hovad, Thomas Wix, Maxim Khomiakov, Georgios Vassos, André Filipe da Silva Rodrigues, Alejandro de Miguel Tejada, and Line H Clemmensen. Deep learning for automatic railway maintenance. *Intelligent quality assessment of railway switches and crossings*, pages 207–228, 2021. 47

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *arXiv preprint arXiv:1704.04861*, 2017. 36

Chen-Chiung Hsieh, Ya-Wen Lin, Li-Hung Tsai, Wei-Hsin Huang, Shang-Lin Hsieh, and Wei-Hung Hung. Offline deep-learning-based defective track fastener detection and inspection system. *Sensors & Materials*, 32, 2020. 48

Chen-Chiung Hsieh, Ti-Yun Hsu, and Wei-Hsin Huang. An online rail track fastener classification system based on yolo models. *Sensors*, 22(24):9970, 2022. 48

Chaowei Hu, Yunpeng Wang, Guizhen Yu, Zhangyu Wang, Ao Lei, and Zhehua Hu. Embedding cnn-based fast obstacles detection for autonomous vehicles. Technical report, SAE Technical Paper, 2018. 33

Hongji Huang, Yuchun Yang, Hong Wang, Zhiguo Ding, Hikmet Sari, and Fumiyuki Adachi. Deep reinforcement learning for uav navigation through massive mimo technique. *IEEE Transactions on Vehicular Technology*, 69(1):1117–1121, 2019. 31

Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506, 2021. 74

International Air Transport Association. Airline maintenance cost executive commentary. Technical report, International Air Transport Association, 2021. 6

David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 84, 183

Rajan Iyengar, Jack Zhu, and Bryan P Tripp. Simulator for the oreo robotic head. *Journal of Computational Vision and Imaging Systems*, 7(1):13–15, 2021. 65

Kanwal Jahan, Jeethesh Pai Umesh, and Michael Roth. Anomaly detection on the rail lines using semantic segmentation and self-supervised learning. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE, 2021. 49

Moksh Jain, Salem Lahlou, Hadi Nekoei, Victor Butoi, Paul Bertin, Jarrid Rector-Brooks, Maksym Korablyov, and Yoshua Bengio. Deup: Direct epistemic uncertainty prediction. *arXiv preprint arXiv:2102.08501*, 2021. 75, 77, 78, 176

Umang Jain, Harshit Shukla, Sashant Kapoor, Amit Pandey, and Harsh Nirwal. Design and analysis of 2-axis rocket motor stand for thrust vectoring. In *AIAA propulsion and energy 2020 forum*, page 3920, 2020. 63

Lana Dalawr Jalal. Three-dimensional off-line path planning for unmanned aerial vehicle using modified particle swarm optimization. *International Journal of Aerospace and Mechanical Engineering*, 9(8):1579–1583, 2015. 18, 20

Ashish James, Wang Jie, Yang Xulei, Ye Chenghao, Nguyen Bao Ngan, Lou Yuxin, Su Yi, Vijay Chandrasekhar, and Zeng Zeng. Tracknet-a deep learning based fault detection for railway track inspection. In *2018 International Conference on Intelligent Rail Transportation (ICIRT)*, pages 1–5. IEEE, 2018. 48

Jinbeum Jang, Minwoo Shin, Sohee Lim, Jonggook Park, Joungyeon Kim, and Joonki Paik. Intelligent image-based railway inspection system using deep learning-based object detection and weber contrast-based image comparison. *Sensors*, 19(21):4738, 2019. 49

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision*, pages 2146–2153. IEEE, 2009. 36

Mark David Jenkins, Tom Buggy, and Gordon Morison. An imaging system for visual inspection and structural condition monitoring of railway tunnels. In *2017 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS)*, pages 1–6. IEEE, 2017. 115

Glenn Jocher. YOLOv5 by Ultralytics, May 2020. URL https://github.com/ultralytics/yolov5. 39

Sabrina Johnson. Network rail is painting tracks white to cool them down because they're 48°c. *Metro*, July 18 2022. URL https://metro.co.uk/2022/07/18/network-rail-is-painting-tracks-white-to-cool-them-down-in-the-heat-17021528/. 110

Belkacem Kada and Y Ghazzawi. Robust pid controller design for an uav flight control system. In *Proceedings of the World congress on Engineering and Computer Science*, volume 2, pages 1–6, 2011. 86

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR, 2018. 25

Abhinav Girish Kamath. *Robust thrust vector control for precision rocket-landing*. University of California, Davis, 2021. 63

Seungmin Kang, Sravana Sristi, Jabir Karachiwala, and Yih-Chun Hu. Detection of anomaly in train speed for intelligent railway systems. In *2018 International Conference on Control, Automation and Diagnosis (ICCAD)*, pages 1–6. IEEE, 2018. 49

Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4551–4560, 2019. 30

Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. In *Conference on Robot Learning*, pages 133–145. PMLR, 2018. 32

Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 690–696. IEEE, 2019. 32

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023. 32

Khalid Khan, Muhammad Sohaib, Azaz Rashid, Saddam Ali, Hammad Akbar, Abdul Basit, and Tanvir Ahmad. Recent trends and challenges in predictive maintenance of aircraft's engine and hydraulic system. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 43:1–17, 2021. 9

Waqas Khan and Meyer Nahon. Real-time modeling of agile fixed-wing uav aerodynamics. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1188–1195, 2015a. doi: 10.1109/ICUAS.2015.7152411. 64

Waqas Khan and Meyer Nahon. Real-time modeling of agile fixed-wing uav aerodynamics. In *2015 international conference on unmanned aircraft systems (ICUAS)*, pages 1188–1195. IEEE, 2015b. 64

Ji-Hyeon Kim, Yeun-Chul Park, Mancheol Kim, and Hyoung-Bo Sim. A fatigue reliability assessment for rail tension clamps based on field measurement data. *Applied Sciences*, 12 (2):624, 2022. 43

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. 95, 108, 113, 117, 123, 131

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. 44, 117

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023. 92

Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004. 55

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2023. 85

Alexandros Kouris and Christos-Savvas Bouganis. Learning to fly by myself: A self-supervised cnn-based approach for autonomous navigation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018. 32

A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research). Technical Report TR-2009-148, Technical Report, 2009. 34

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25: 1097–1105, 2012. 11, 36

Rumit Kumar, Mahathi Bhargavapuri, Aditya M Deshpande, Siddharth Sridhar, Kelly Cohen, and Manish Kumar. Quaternion feedback based autonomous control of a quadcopter uav with thrust vectoring rotors. In *2020 american control conference (acc)*, pages 3828–3833. IEEE, 2020. 63

Malte Kuss and Carl Rasmussen. Gaussian processes in reinforcement learning. *Advances in neural information processing systems*, 16, 2003. 74

A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, T. Duerig, and V. Ferrari. The open images dataset v4. *arXiv preprint arXiv:1811.00982*, 2018. 35

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017. 74

Benoit Landry et al. *Planning and control for quadrotor flight through cluttered environments*. PhD thesis, Massachusetts Institute of Technology, 2015. 66

Jacoby Larson and Mohan Trivedi. Lidar based off-road negative obstacle detection and analysis. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 192–197. IEEE, 2011. 33

Steven M LaValle, James J Kuffner, BR Donald, et al. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, 5:293–308, 2001. 17, 20

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998a. 34, 36

Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998. *URL http://yann. lecun. com/exdb/mnist*, 10(34):14, 1998b. 36

HyeongJin Lee, WooSung Cho, SangHo Ko, and Yeol Lee. Aerodynamic characteristics of the grid fins on spacex falcon 9. *Journal of the Korean Society for Aeronautical & Space Sciences*, 48(10):745–752, 2020. 68

Junyong Lee, Myeonghee Lee, Sunghyun Cho, and Seungyong Lee. Reference-based video super-resolution using multi-camera video triplets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17824–17833, 2022. 65

Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, et al. Yolov6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*, 2022. 39

Dawei Li, Qian Xie, Xiaoxi Gong, Zhenghao Yu, Jinxuan Xu, Yangxing Sun, and Jun Wang. Automatic defect detection of metro tunnel surfaces using a vision-based inspection system. *Advanced Engineering Informatics*, 47:101206, 2021. 115

Ker-Chau Li. Asymptotic optimality for cp, cl, cross-validation and generalized cross-validation: discrete index set. *The Annals of Statistics*, pages 958–975, 1987. 30

Maohong Li, Yuanxiao Hong, Hong Yu, Shuxin Qu, and Ping Wang. A novel high solar reflective coating based on potassium silicate for track slab in high-speed railway. *Construction and Building Materials*, 225:900–908, 2019. 110

Xinpeng Li and Xiaojiang Peng. Rail detection: An efficient row-based network and a new benchmark. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 6455–6463, 2022. 103, 112

You Li. *Stereo vision and Lidar based dynamic occupancy grid mapping: Application to scenes analysis for intelligent vehicles*. PhD thesis, Université de Technologie de Belfort-Montbeliard, 2013. 16

You Li and Yassine Ruichek. Occupancy grid mapping in urban environments from a moving on-board stereo-vision system. *Sensors*, 14(6):10454–10478, 2014. 15

Zhenming Li, Mingjie Lao, Swee King Phang, Mohamed Redhwan Abdul Hamid, Kok Zuea Tang, and Feng Lin. Development and design methodology of an anti-vibration system on micro-uavs. In *International micro air vehicle conference and flight competition (IMAV)*, pages 223–228, 2017. 13

RENHR LI HY et al. Cooperative indoor path planning of multi-uavs for high-rise fire fighting based on rrt-forest algorithm. *Acta Automatica Sinica*, 45:1–12, 2021. 17, 20

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. URL `http://arxiv.org/abs/1509.02971`. 25

Huei-Yung Lin and Xin-Zhong Peng. Autonomous quadrotor navigation with vision based obstacle avoidance and path planning. *IEEE Access*, 9:102450–102459, 2021. 16, 20

T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. *European Conference on Computer Vision (ECCV)*, pages 740–755, 2014. 35

Ya-Wen Lin, Chen-Chiung Hsieh, Wei-Hsin Huang, Sun-Lin Hsieh, and Wei-Hung Hung. Railway track fasteners fault detection using deep learning. In *2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, pages 187–190. IEEE, 2019a. 48

Yi-Chun Lin, Yi-Ting Cheng, Tian Zhou, Radhika Ravi, Seyyed Meghdad Hasheminasab, John Evan Flatt, Cary Troy, and Ayman Habib. Evaluation of uav lidar for mapping coastal environments. *Remote Sensing*, 11(24):2893, 2019b. 14

Donghai Liu, Xietian Xia, Junjie Chen, and Shuai Li. Integrating building information model and augmented reality for drone-based building inspection. *J. Comput. Civ. Eng*, 35(2):04020073, 2021. 1

Jiajia Liu, Bailin Li, Ying Xiong, Biao He, and Li Li. Integrating the symmetry image and improved sparse representation for railway fastener classification and defect recognition. *Mathematical Problems in Engineering*, 2015, 2015. 107

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 39

Zhenzhen Liu, Jin Peng Zhou, Yufan Wang, and Kilian Q Weinberger. Unsupervised out-of-distribution detection with diffusion inpainting. *arXiv preprint arXiv:2302.10326*, 2023. 130

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015a. 40

Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105. PMLR, 2015b. 31

Antonio Loquercio, Ana I Maqueda, Carlos R Del-Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018. 32

Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2019. 30, 32

Carlos Luis and Jérôme Le Ny. Design of a trajectory tracking controller for a nanoquadcopter. *arXiv preprint arXiv:1608.05786*, 2016. 66

Zhongzhen Luo, Martin V Mohrenschildt, and Saeid Habibi. A probability occupancy grid based approach for real-time lidar ground segmentation. *IEEE Transactions on Intelligent Transportation Systems*, 21(3):998–1010, 2019. 15

Björn Lütjens, Michael Everett, and Jonathan P How. Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8662–8668. IEEE, 2019. 74

Yang Lyu, Zhiwei Han, Junping Zhong, Changjiang Li, and Zhigang Liu. A gan-based anomaly detection method for isoelectric line in high-speed railway. In *2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1–6. IEEE, 2019. 49

Lin Ma, Kexin Wang, Zuhua Xu, Zhijiang Shao, and Zhengyu Song. Trajectory optimization for powered descent and landing of reusable rockets with restartable engines. In *Proceedings of the International Astronautical Congress, IAC, Bremen, Germany*, pages 1–5, 2018. 62

Thi Thoa Mac, Cosmin Copot, Andres Hernandez, and Robin De Keyser. Improved potential field method for unknown obstacle avoidance using uav in indoor environment. In *2016 IEEE 14th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 345–350. IEEE, 2016. 17, 20

D Mader, R Blaskow, P Westfeld, and C Weller. Potential of uav-based laser scanner and multispectral camera data in building inspection. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 41:1135, 2016. 1

Omid Maghazei and Matthias Steinmann. Drones in railways: Exploring current applications and future scenarios based on action research. *European Journal of Transport and Infrastructure Research*, 20(3):87–102, 2020. 1

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu based physics simulation for robot learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. 28

Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. *Advances in neural information processing systems*, 31, 2018. 75

Sina Sharif Mansouri, Miguel Castaño, Christoforos Kanellakis, and George Nikolakopoulos. Autonomous mav navigation in underground mines using darkness contours detection. In *International conference on computer vision systems*, pages 164–174. Springer, 2019. 22

Hermann G Matthies. Quantifying uncertainty: modern computational representation of probability and applications. In *Extreme man-made and natural hazards in dynamics of structures*, pages 105–135. Springer, 2007. 74

Alfian Ma'Arif, Wahyu Rahmaniar, Marco Antonio Márquez Vera, Aninditya Anggari Nuryono, Rania Majdoubi, and Abdullah Çakan. Artificial potential field algorithm for obstacle avoidance in uav quadrotor for dynamic environment. In *2021 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*, pages 184–189. IEEE, 2021. 17, 20

Barnes W McCormick. *Aerodynamics, aeronautics, and flight mechanics*. John Wiley & Sons, 1994. 64

Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982. 15, 20

Li Meng, Song Qing, and Zhao Qin Jun. Uav path re-planning based on improved bidirectional rrt algorithm in dynamic environment. In *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*, pages 658–661. IEEE, 2017. 17, 20

Daniel Meyer-Delius, Maximilian Beinhofer, and Wolfram Burgard. Occupancy grid models for robot mapping in changing environments. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012. 15, 20

Shruti Mittal and Dattaraj Rao. Vision based railway track monitoring using deep learning. *arXiv preprint arXiv:1711.06423*, 2017. 110, 113, 115

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 25

Thomas Moerland, Joost Broekens, and Catholijn Jonker. Efficient exploration with double uncertain value networks. In *NIPS 2017: Thirty-first Conference on Neural Information Processing Systems*, pages 1–17, 2017. 75

Peyman Moghadam, Wijerupage Sardha Wijesoma, and Dong Jun Feng. Improving path planning and mapping based on stereo vision and lidar. In *2008 10th International Conference on Control, Automation, Robotics and Vision*, pages 384–389. IEEE, 2008. 15

Matthias Mueller, Vincent Casser, Jean Lahoud, Neil Smith, and Bernard Ghanem. Ue4sim: A photo-realistic simulator for computer vision applications. 2017. 55

Linda J Mullen and V Michael Contarino. Hybrid lidar-radar: seeing through the scatter. *IEEE Microwave magazine*, 1(3):42–48, 2000. 14

Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample-efficient continuous control. *arXiv preprint arXiv:2405.16158*, 2024. 25

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. The street view house numbers (svhn) dataset. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, 2011. 35

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer, 1999. 28, 51

Thien-Nghia Nguyen, Bernd Michaelis, Ayoub Al-Hamadi, Michael Tornow, and Marc-Michael Meinecke. Stereo-camera-based urban environment perception using occupancy grid and object tracking. *IEEE Transactions on Intelligent Transportation Systems*, 13(1): 154–165, 2011. 15

Zhen-Liang Ni, Gui-Bin Bian, Xiao-Hu Zhou, Zeng-Guang Hou, Xiao-Liang Xie, Chen Wang, Yan-Jie Zhou, Rui-Qi Li, and Zhen Li. Raunet: Residual attention u-net for semantic segmentation of cataract surgical instruments. In *International Conference on Neural Information Processing*, pages 139–149. Springer, 2019. 93

Nikolay Nikolov, Johannes Kirschner, Felix Berkenkamp, and Andreas Krause. Information-directed exploration for deep reinforcement learning. In *International Conference on Learning Representations*, 2018. 75

Office of Rail and Road. Annual efficiency and finance assessment of network rail 2021-22, 2022. 6

Sang-Il Oh and Hang-Bong Kang. Fast occupancy grid filtering using grid cell clusters from lidar and stereo vision sensor data. *IEEE Sensors Journal*, 16(19):7258–7266, 2016. 15

Network Rail Air Operations. Operational guidance for drone users: flying a drone near a railway track. *Network Rail*, 2023. 43

Ian Osband. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS workshop on bayesian deep learning*, volume 192, 2016. 74, 77

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018. 74

Omar Sami Oubbati, Mohammed Atiquzzaman, Abdullah Baz, Hosam Alhakami, and Jalel Ben-Othman. Dispatch of uavs for urban vehicular networks: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 70(12):13174–13189, 2021a. 31

Omar Sami Oubbati, Mohammed Atiquzzaman, Abderrahmane Lakas, Abdullah Baz, Hosam Alhakami, and Wajdi Alhakami. Multi-uav-enabled aoi-aware wpcn: A multi-agent reinforcement learning strategy. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2021b. 31

Ramanpreet Singh Pahwa, Jin Chao, Jestine Paul, Yiqun Li, Ma Tin Lay Nwe, Shudong Xie, Ashish James, Arulmurugan Ambikapathi, Zeng Zeng, and Vijay Ramaseshan Chandrasekhar. Faultnet: Faulty rail-valves detection using deep learning and computer vision. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 559–566. IEEE, 2019. 48

Gonzalo Pajares. Overview and current status of remote sensing applications based on unmanned aerial vehicles (uavs). *Photogrammetric Engineering & Remote Sensing*, 81(4): 281–330, 2015. 13

Christos Papachristos, Kostas Alexis, and Anthony Tzes. Efficient force exertion for aerial robotic manipulation: Exploiting the thrust-vectoring authority of a tri-tiltrotor uav. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 4500–4505. IEEE, 2014. 63

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018. 30

Sara Pérez-Carabaza, Jürgen Scherer, Bernhard Rinner, José A López-Orozco, and Eva Besada-Portas. Uav trajectory optimization for minimum time search with communication constraints and collision avoidance. *Engineering Applications of Artificial Intelligence*, 85:357–371, 2019. 17, 20

Alexander R Perry. The flightgear flight simulator. In *Proceedings of the USENIX Annual Technical Conference*, volume 686, pages 1–12, 2004. 55

Swee King Phang, Jun Jie Ong, Ronald TC Yeo, Ben M Chen, and Tong H Lee. Autonomous mini-uav for indoor flight with embedded on-board vision processing as navigation system. In *2010 IEEE Region 8 International Conference on Computational Technologies in Electrical and Electronics Engineering (SIBIRCON)*, pages 722–727. IEEE, 2010. 13

Swee King Phang, Chenxiao Cai, Ben M Chen, and Tong Heng Lee. Design and mathematical modeling of a 4-standard-propeller (4sp) quadrotor. In *Proceedings of the 10th world congress on intelligent control and automation*, pages 3270–3275. IEEE, 2012. 13, 66

Swee King Phang, Kun Li, Ben M Chen, Tong H Lee, Kimon P Valavanis, and George J Vachtsevanos. Systematic design methodology and construction of micro aerial quadrotor vehicles. *Handbook of Unmanned Aerial Vehicles*, pages 181–206, 2014. 13

Robert Ronald Phillips. *Ultrasonic methods for rail inspection*. University of California, San Diego, 2012. 110

Riccardo Polvara, Sanjay Sharma, Jian Wan, Andrew Manning, and Robert Sutton. Obstacle avoidance approaches for autonomous navigation of unmanned surface vehicles. *The Journal of Navigation*, 71(1):241–256, 2018. 17, 20

G Püskülcü and A Ulas. 3-d grain burnback analysis of solid propellant rocket motors: Part 1–ballistic motor tests. *Aerospace Science and Technology*, 12(8):579–584, 2008. 62

Julien Rabatel, Sandra Bringay, and Pascal Poncelet. So_mad: Sensor mining for anomaly detection in railway data. In *Advances in Data Mining. Applications and Theoretical Aspects: 9th Industrial Conference, ICDM 2009, Leipzig, Germany, July 20-22, 2009. Proceedings 9*, pages 191–205. Springer, 2009. 49

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518. PMLR, 2023. 9

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3, 2019. 73

Md Atiqur Rahman and Abdelhamid Mammeri. Vegetation detection in uav imagery for railway monitoring. In *Vehits*, pages 457–464, 2021. 6

Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017. 38, 108

Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 39, 47

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 38

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 37, 107

Javier Ribera, David Guera, Yuhao Chen, and Edward J Delp. Locating objects without bounding boxes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6479–6489, 2019. 37

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021. 9

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241. Springer, 2015. 11, 40

Sara Roos-Hoefgeest, Jonathan Cacace, Vincenzo Scognamiglio, Ignacio Álvarez, Rafael C González, Fabio Ruggiero, and Vincenzo Lippiello. A vision-based approach for unmanned aerial vehicles to track industrial pipes for inspection tasks. In *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1183–1190. IEEE, 2023. 21

Seyyed Mohammad Hosseini Rostami, Arun Kumar Sangaiah, Jin Wang, and Xiaozhu Liu. Obstacle avoidance of mobile robots using modified artificial potential field algorithm. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–19, 2019. 17, 20

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 35

Furqan Rustam, Abid Ishaq, Muhammad Shadab Alam Hashmi, Hafeez Ur Rehman Siddiqui, Luis Alonso Dzul López, Juan Castanedo Galán, and Imran Ashraf. Railway track fault detection using selective mfcc features from acoustic data. *Sensors*, 23(16):7018, 2023. 110

Stefano Sabatini, Matteo Carno, Simone Fiorenti, and Sergio Matteo Savaresi. Improving occupancy grid mapping via dithering for a mobile robot equipped with solid-state lidar sensors. In *2018 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1145–1150. IEEE, 2018. 15

Alessandro Sabato and Christopher Niezrecki. Feasibility of digital image correlation for railroad tie inspection and ballast support assessment. *Measurement*, 103:93–105, 2017. 115

Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016. 31

Yosiyuki Sakamoto, Makio Ishiguro, and Genshiro Kitagawa. Akaike information criterion statistics. *Dordrecht, The Netherlands: D. Reidel*, 81(10.5555):26853, 1986. 30

Tim Salimans and Richard Chen. Learning montezuma's revenge from a single demonstration. *arXiv preprint arXiv:1812.03381*, 2018. 73

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020. 29

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015a. 28

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *International conference on learning representations*, 2015b. 28

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347. 28

Murat Sensoy, Lance Kaplan, and Melih Kandemir. Evidential deep learning to quantify classification uncertainty. *Advances in neural information processing systems*, 31, 2018. 46, 119

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 37

Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics: Results of the 11th International Conference*, pages 621–635. Springer, 2018. 55

Khurram Shaikh, Imtiaz Hussain, and Bhawani Shankar Chowdhry. Wheel defect detection using a hybrid deep learning approach. *Sensors*, 23(14):6248, 2023. 50

Adi Shamir, Odelia Melamed, and Oriel BenShmuel. The dimpled manifold model of adversarial examples in machine learning. *arXiv preprint arXiv:2106.10151*, 2021. 129

Siddhartha Sharma, Yu Cui, Qing He, Reza Mohammadi, and Zhiguo Li. Data-driven optimization of railway maintenance for track geometry. *Transportation Research Part C: Emerging Technologies*, 90:34–58, 2018. 8

Jaeseok Shim, Jeongseo Koo, Yongwoon Park, and Jaehoon Kim. Anomaly detection method in railway using signal processing and deep learning. *Applied Sciences*, 12(24):12901, 2022. 49, 50

Minoru Shimizu, Suresh Perinpanayagam, and Bernadin Namoano. A real-time fault detection framework based on unsupervised deep learning for prognostics and health management of railway assets. *IEEE Access*, 10:96442–96458, 2022. 50

Amit Shukla, Huang Xiaoqian, and Hamad Karki. Autonomous tracking of oil and gas pipelines by an unmanned aerial vehicle. In *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4. IEEE, 2016a. 21

Amit Shukla, Huang Xiaoqian, and Hamad Karki. Autonomous tracking and navigation controller for an unmanned aerial vehicle based on visual data for inspection of oil and gas pipelines. In *2016 16th international conference on control, automation and systems (ICCAS)*, pages 194–200. IEEE, 2016b. 21

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014. 25

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 32

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 29

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 36

Qing Song, Yao Guo, Lu Yang, Jianan Jiang, Chun Liu, and Mengjie Hu. High-speed railway fastener detection and localization system. *arXiv preprint arXiv:1907.01141*, 2019. 107

Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. In *Proceedings of the 2020 Conference on Robot Learning*, pages 1147–1157, 2021. 56

Yunlong Song, Angel Romero, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 8(82):eadg1462, 2023. 32

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 36

Brant Stratman, Yongming Liu, and Sankaran Mahadevan. Structural health monitoring of railroad wheels using wheel impact load detectors. *Journal of failure analysis and prevention*, 7:218–225, 2007. 6

Ryan Sullivan, Justin K Terry, Benjamin Black, and John P Dickerson. Cliff diving: Exploring reward surfaces in reinforcement learning environments. *arXiv preprint arXiv:2205.07015*, 2022. 26, 74

Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. 31

Jiayi Sun, Jun Tang, and Songyang Lao. Collision avoidance for cooperative uavs with optimized artificial potential field algorithm. *IEEE Access*, 5:18382–18390, 2017. 16, 20

Richard Sutton. The bitter lesson. *Incomplete Ideas (blog)*, 13:12, 2019. 29

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 8, 1995. 73

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 23, 29, 72, 73

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 36

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016. 36

Jun Jet Tai and Jim Wong. Pyflyt - uav flight simulator gymnasium environments for reinforcement learning research, 2023. URL http://github.com/jjshoots/PyFlyt. 80

Jun Jet Tai, Swee King Phang, and Felicia Yen Myan Wong. Coaa*—an optimized obstacle avoidance and navigational algorithm for uavs operating in partially observable 2d environments. *Unmanned Systems*, 10(02):159–174, 2022. 17, 18

Hans-Martin Thomas, Thomas Heckel, and G Hanspach. Advantage of a combined ultrasonic and eddy current examination for railway inspection trains. *Insight-Non-Destructive Testing and Condition Monitoring*, 49(6):341–344, 2007. 6

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012. 72, 80

Surya T Tokdar and Robert E Kass. Importance sampling: a review. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):54–60, 2010. 28

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. URL https://zenodo.org/record/8127025. 56, 57

Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014. 31

Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE international conference on computer vision*, pages 4068–4076, 2015. 31

Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, et al. Jump-start reinforcement learning. *arXiv preprint arXiv:2204.02372*, 2022. 189

Dennis Ulmer and Giovanni Cinà. Know your limits: Uncertainty estimation with relu classifiers fails at reliable ood detection. In *Uncertainty in Artificial Intelligence*, pages 1766–1776. PMLR, 2021. 120

Dennis Ulmer, Lotta Meijerink, and Giovanni Cinà. Trust issues: Uncertainty estimation does not enable reliable ood detection on medical tabular data. In *Machine Learning for Health*, pages 341–354. PMLR, 2020. 120

Angel G Valdenebro. Visualizing rotations and composition of rotations with the rodrigues vector. *European Journal of Physics*, 37(6):065001, 2016. 63

Stefan Van Der Veeken, Jamie Wubben, Carlos T Calafate, Juan-Carlos Cano, Pietro Manzoni, and Johann Marquez-Barja. A collision avoidance strategy for multirrotor uavs based on artificial potential fields. In *Proceedings of the 18th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks*, pages 95–102, 2021. 16, 20

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016. 25

Jordy Van Landeghem, Matthew Blaschko, Bertrand Anckaert, and Marie-Francine Moens. Benchmarking scalable predictive uncertainty in text classification. *Ieee Access*, 10:43703–43737, 2022. 120

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 12

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019. 32

Luke Wallace, Arko Lucieer, Christopher Watson, and Darren Turner. Development of a uav-lidar system with application to forest inventory. *Remote sensing*, 4(6):1519–1543, 2012. 14

Chao Wang, Jian Wang, Jingjing Wang, and Xudong Zhang. Deep-reinforcement-learning-based autonomous uav navigation with sparse rewards. *IEEE Internet of Things Journal*, 7(7):6180–6190, 2020a. 31

Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7464–7475, 2023. 39

Fei Wang, Jinqiang Cui, Swee King Phang, Ben M Chen, and Tong H Lee. A mono-camera and scanning laser range finder based uav indoor navigation system. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 694–701. IEEE, 2013. 14

Fei Wang, Kangli Wang, Shupeng Lai, Swee King Phang, Ben M Chen, and Tong H Lee. An efficient uav navigation solution for confined but partially known indoor environments. In *11th IEEE International Conference on Control & Automation (ICCA)*, pages 1351–1356. IEEE, 2014. 13

Jian Wang, Longfu Luo, Wei Ye, and Shenglan Zhu. A defect-detection method of split pins in the catenary fastening devices of high-speed railway based on deep learning. *IEEE Transactions on Instrumentation and Measurement*, 69(12):9517–9525, 2020b. 48

Tiange Wang, Fangfang Yang, and Kwok-Leung Tsui. Real-time detection of railway track component via one-stage deep learning networks. *Sensors*, 20(15):4325, 2020c. 107

Tongzhou Wang, Simon Du, Antonio Torralba, Phillip Isola, Amy Zhang, and Yuandong Tian. Denoised mdps: Learning world models better than the world itself. In *International Conference on Machine Learning*, pages 22591–22612. PMLR, 2022. 29

William Washington and Mark Miller. Grid fins-a new concept for missile stability and control. In *31st Aerospace Sciences Meeting*, page 35, 1993. 68

Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989. 25

Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, et al. Envpool: A highly parallel reinforcement learning environment execution engine. *arXiv preprint arXiv:2206.10558*, 2022. 28

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In *6th Annual Conference on Robot Learning*, 2022. 29

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022. 32

Yiqi Xia, Fengying Xie, and Zhiguo Jiang. Broken railway fastener detection based on adaboost algorithm. In *2010 International Conference on Optoelectronics and Image Processing*, volume 1, pages 313–316. IEEE, 2010. 107

Yuwei Xia. Autonomous railway defect detection. Master's thesis, University of Waterloo, 2023. 22

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 35

Huang Xiaoqian, Amit Shukla, and Hamad Karki. Autonomous ground piipelines tracking via an uav. In *2016 13th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 372–375. IEEE, 2016. 21

Junyu Xuan, Jie Lu, Zheng Yan, and Guangquan Zhang. Bayesian deep reinforcement learning via deep kernel learning. *International Journal of Computational Intelligence Systems*, 2018. 74

Lu Yafei, Wu Anping, Chen Qingyang, and Wang Yujie. An improved uav path planning method based on rrt-apf hybrid strategy. In *2020 5th International Conference on Automation, Control and Robotics Engineering (CACRE)*, pages 81–86. IEEE, 2020. 17, 20

ZHOU Yan, CAI Chenxiao, YAO Juan, ZOU Yun, and YANG Yi. Design of autonomous navigation system for quadrotor in subway tunnel. In *2020 IEEE 16th International Conference on Control & Automation (ICCA)*, pages 187–192. IEEE, 2020. 21

Hao Yang, Hao Wu, and Hao Chen. Detecting 11k classes: Large scale object detection without fine-grained bounding boxes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9805–9813, 2019. 37

Hongfei Yang, Yanzhang Wang, Jiyong Hu, Jiatang He, Zongwei Yao, and Qiushi Bi. Deep learning and machine vision-based inspection of rail surface defects. *IEEE Transactions on Instrumentation and Measurement*, 71:1–14, 2021. 48

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34:25476–25488, 2021. 29

Yunguang Ye, Caihong Huang, Jing Zeng, Yichang Zhou, and Fansong Li. Shock detection of rotating machinery based on activated time-domain images and deep learning: An application to railway wheel flat detection. *Mechanical Systems and Signal Processing*, 186: 109856, 2023. 50

Chunlei Yu, Veronique Cherfaoui, and Philippe Bonnifait. Evidential occupancy grid mapping with stereo-vision. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 712–717. IEEE, 2015. 15

Wenhao Yu, C Karen Liu, and Greg Turk. Policy transfer with strategy optimization. In *International Conference on Learning Representations*, 2018. 30

Vitjan Zavrtanik, Matej Kristan, and Danijel Skočaj. Reconstruction by inpainting for visual anomaly detection. *Pattern Recognition*, 112:107706, 2021a. 44

Vitjan Zavrtanik, Matej Kristan, and Danijel Skočaj. Draem - a discriminatively trained reconstruction embedding for surface anomaly detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8330–8339, October 2021b. 44

Qin Zekui, Wang Rui, Dong Xiwang, Li Qingdong, Fang Dongyang, and Ren Zhang. Three-dimensional path planning for unmanned aerial vehicles based on the developed rrt algorithm. In *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*, pages 1–5. IEEE, 2018. 17, 20

Oliver Zendel, Markus Murschitz, Marcel Zeilinger, Daniel Steininger, Sara Abbasi, and Csaba Beleznai. Railsem19: A dataset for semantic rail scene understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019. 102

Yong Zeng, Xiaoli Xu, Shi Jin, and Rui Zhang. Simultaneous navigation and radio mapping for cellular-connected uav with deep reinforcement learning. *IEEE Transactions on Wireless Communications*, 20(7):4205–4220, 2021. 31

Lele Zhang, Fang Deng, Jie Chen, Yingcai Bi, Swee King Phang, Xudong Chen, and Ben M Chen. Vision-based target three-dimensional geolocation using unmanned aerial vehicles. *IEEE Transactions on Industrial Electronics*, 65(10):8052–8061, 2018. 13

Yan Zhang, Kefeng Li, Guangyuan Zhang, Zhenfei Wang, Zhenfang Zhu, Chen Fu, and Guangyuan Jiang. Msca-yolo: Accurate detection of railway track anomalies using multi-scale features. 2023. 48

Lanxiang Zheng, Ping Zhang, Jia Tan, and Fang Li. The obstacle detection method of uav based on 2d lidar. *IEEE Access*, 7:163437–163448, 2019. 16, 20

Yingjie Zheng, Songrong Wu, Dong Liu, Ruoyu Wei, Shuting Li, and Zhenwei Tu. Sleeper defect detection based on improved yolo v3 algorithm. In *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 955–960. IEEE, 2020. 47

B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1452–1464, 2017a. 35

B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Ade20k: A large-scale dataset for semantic segmentation in adverse conditions. *arXiv preprint arXiv:1708.02383*, 2017b. 35

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017. 31

Appendix

1 PyFlyt Tested Gymnasium Environments

`PyFlyt/QuadX-Hover-v0` To facilitate reproducibility and usability of the library, PyFlyt provides several well-tested and well-documented Gymnasium environments for reinforcement learning research. These default environments are described in this section.

The task of this environment is for an agent to hover the QuadX UAV indefinitely. In the dense setting, the reward of the environment is described as follows:

$$r = \begin{cases} -100 & \text{if crash,} \\ -||[\theta, \phi]|| - ||[x, y, z-1]|| & \text{otherwise} \end{cases} \tag{1}$$

Where θ and ϕ are the pitch and roll angles in radians, x, y and z describe the linear position of the UAV.

In the sparse reward variant, the reward is simply -0.1 for each timestep and -100 for crashing.

`PyFlyt/QuadX-Waypoints-v0` This environment describes the task of getting a QuadX UAV to reach a series of randomly generated waypoints in 3D space in a specific order. By default, there are five waypoints in total, but this number is reconfigurable. The dense reward function of the environment is as follows:

$$r = \begin{cases} -100 & \text{if crash,} \\ c_a \delta^{-1} - c_b \dot{\delta} & \text{if } \dot{\delta} < 0, \\ 100 & \text{if waypoint reached,} \\ \delta^{-1} & \text{otherwise} \end{cases} \tag{2}$$

Where δ is the displacement from the UAV to the next target, and c_a and c_b are scaling coefficients, set at 0.1 and 3 respectively. To avoid reward function blowup caused by the δ^{-1} component, a waypoint is considered reached when the UAV's center of mass is some distance d away from the waypoint. This distance is a reconfigurable parameter that is set at 0.2 m by default. In the sparse reward setting, the reward is simply -100 for crashing and 100 for reaching a waypoint.

`PyFlyt/Fixedwing-Waypoints-v0` This environment is similar to `PyFlyt/QuadX-Waypoints-v0` but is modified to use the Fixedwing UAV. Due to the increase in

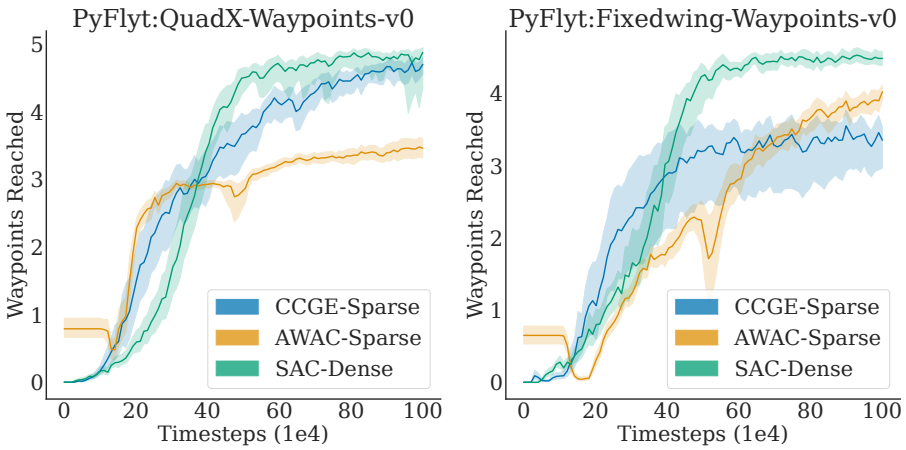

Figure 1: Learning curves of three different reinforcement learning algorithms on two core environments within PyFlyt.

size of the UAV model, d is set at 2, c_a is set at 1 and c_b is set at 3. In addition, the waypoints are also generated much further apart, at a scale of 20 times that of the QuadX environment.

Environment Learnability We test all `*-Waypoints-v0` environments using two existing and one new RL algorithm. The results, shown in Figure 1, are meant to show the feasibility of training agents in these environments across multiple different algorithms. The dense reward version of both environments were tested using the SAC algorithm, while the sparse reward environments have been tested using AWAC and CCGE. CCGE is a reinforcement learning algorithm that was developed as part of this research work, and its exact formulation is described in section 3.6 on page 72. Both AWAC and CCGE require the existence of an oracle policy to train, this oracle policy was obtained by using a hard-coded carrot-chasing algorithm for `QuadX-Waypoints-v0`, while the experiments for `Fixedwing-Waypoints-v0` utilized an oracle policy trained using SAC. Examples of flight trajectories for each environment are also shown in Figure 2.

2 CCGE Details

2.1 Explicit Epistemic Uncertainty

In this section, we derive epistemic uncertainty in terms of Q-value networks for any $\{\boldsymbol{s}_t, \boldsymbol{a}_t\}$ pair. This derivation follows the formalism taken from Jain et al. [2021] briefly covered in section 3.6.3 on page 74. Unless otherwise specified, all expectations in this section are taken over $\boldsymbol{s}_{t+1}, r_t \sim \rho_{env}(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t), \boldsymbol{a}_{t+1} \sim \pi_\theta(\cdot|\boldsymbol{s}_{t+1})$.

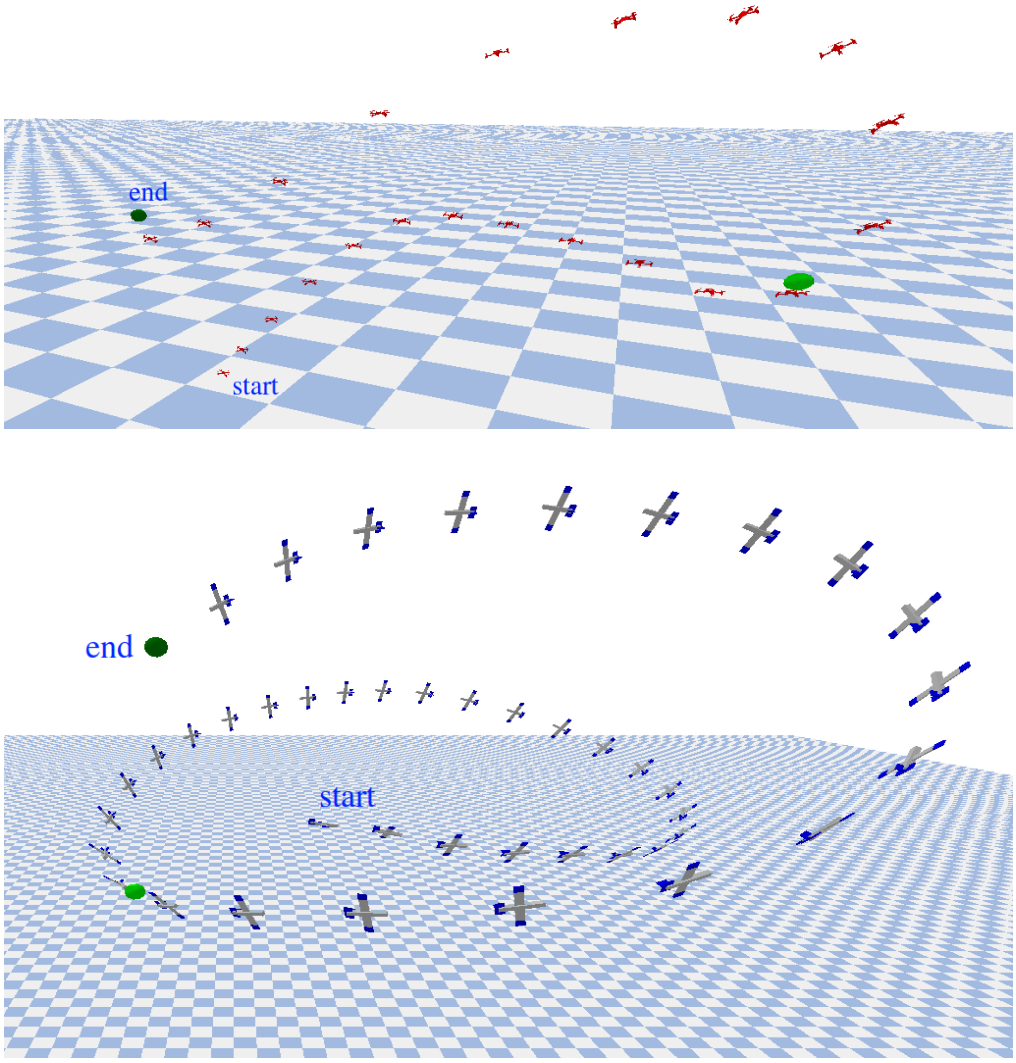

Figure 2: Example flight trajectories generated using trained agents in `QuadX-Waypoints-v0` and `Fixedwing-Waypoints-v0`. In this instance, both environments only use two waypoints for easier visualization.

Single Step Epistemic Uncertainty

For a Q-value estimator, following the definition in (3.19), the *single step total uncertainty* can be written as the expected total loss for Q^{π}_{ϕ}:

$$\mathcal{U}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbb{E}\left[l(Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - \gamma Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}))\right] \tag{3}$$

The expectation here is taken over $\{\boldsymbol{s}_{t+1}, r_t\} \sim \mathcal{D}|\{\boldsymbol{s}_t, \boldsymbol{a}_t\}, \boldsymbol{a}_{t+1} \sim \pi(\cdot|\boldsymbol{s}_{t+1})$. Likewise, we can define a *single step aleatoric uncertainty* for an estimated Q-value by extending (3.21) and (3.22). Recall that the aleatoric uncertainty is defined over the target distribution (in this case $\mathbb{E}[r_t + \gamma Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})]$), and assuming that the target distribution is Gaussian, the aleatoric uncertainty simply becomes the variance in the data. Using this assumption, we can write the aleatoric uncertainty as the variance of the target Q-value estimate.

$$\mathcal{A}(Q^{\pi}_{\phi}|\{\boldsymbol{s}_t, \boldsymbol{a}_t\}) = \sigma^2_{|\boldsymbol{s}_t, \boldsymbol{a}_t}\left(r_t + Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})\right) \tag{4}$$

Finally, we denote the epistemic uncertainty for Q-value estimates across one time step for a given state and action as $\delta_t(\boldsymbol{s}_t, \boldsymbol{a}_t)$. Following equation (3.23) and using (3), this is defined as:

$$\begin{aligned}
&\delta_t(\boldsymbol{s}_t, \boldsymbol{a}_t) \\
&= \mathcal{U}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - \mathcal{A}(Q^{\pi}_{\phi}|\{\boldsymbol{s}_t, \boldsymbol{a}_t\}) \\
&= \mathbb{E}\left[l(Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - \gamma Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}))\right] - \sigma^2_{|\boldsymbol{s}_t, \boldsymbol{a}_t}\left(r_t + Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})\right)
\end{aligned} \tag{5}$$

Taking the definition of variance as $\sigma^2(x) = \mathbb{E}[l(x)] - l(\mathbb{E}[x])$, the first term in (5) can be expanded into:

$$\begin{aligned}
&\mathbb{E}\left[l(Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - \gamma Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}))\right] \\
&= \sigma^2_{|\boldsymbol{s}_t, \boldsymbol{a}_t}(Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})) + l\left(\mathbb{E}[Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - \gamma Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})]\right)
\end{aligned} \tag{6}$$

Since $\sigma^2(Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t)) = 0$, $\sigma^2(x + C) = \sigma^2(x)$ for constant C, and $\sigma^2(-x) = \sigma^2(x)$, (6) evaluates to:

$$\begin{aligned}
&\mathbb{E}\left[l(Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - \gamma Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}))\right] \\
&= \sigma^2_{|\boldsymbol{s}_t, \boldsymbol{a}_t}(Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})) + l\left(\mathbb{E}[Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - \gamma Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})]\right) \\
&= \sigma^2_{|\boldsymbol{s}_t, \boldsymbol{a}_t}(-r_t - Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})) + l\left(\mathbb{E}[Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - \gamma Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})]\right) \\
&= \sigma^2_{|\boldsymbol{s}_t, \boldsymbol{a}_t}(r_t + Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})) + l\left(\mathbb{E}[Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - \gamma Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})]\right)
\end{aligned} \tag{7}$$

Putting (7) into (5), we obtain:

$$\begin{aligned}
\delta_t(\boldsymbol{s}_t, \boldsymbol{a}_t) &= \mathcal{U}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - \mathcal{A}(Q^{\pi}_{\phi}|\{\boldsymbol{s}_t, \boldsymbol{a}_t\}) \\
&= l\left(\mathbb{E}[Q^{\pi}_{\phi}(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_t - \gamma Q^{\pi}_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})]\right)
\end{aligned} \tag{8}$$

Simply put, the single step epistemic uncertainty for a Q value estimator is simply the expected Bellman residual error projected through the loss function.

N-Step Epistemic Uncertainty

The single step epistemic uncertainty is only a measure of epistemic uncertainty of the Q-value predicted against its target value. For RL methods which rely on bootstrapping, the target value consists of a sampled reward and an estimated Q-value of the next time step. To obtain a more reliable estimate for epistemic uncertainty, it is important to account for the epistemic uncertainty in the target value itself. This train of thought leads very naturally to estimating epistemic uncertainty using the discounted sum of single step epistemic uncertainties, which we refer to as $\mathcal{E}_\phi$.

$$\mathcal{E}_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbb{E}_{\pi,\mathcal{D}}\left[\sum_{i=t}^{T} \gamma^{i-t} |\delta_i(\mathbf{s}_i, \mathbf{a}_i)|\right] \tag{9}$$

Learning the N-Step Epistemic Uncertainty

In our experiments, we found that having neural networks learn (9) leads to very unstable training due to value blowup. Instead, we propose simply learning its root, resulting in much more stable training:

$$\mathcal{E}_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t) = \left[\mathbb{E}_{\pi,\mathcal{D}}\left[\sum_{i=t}^{T} \gamma^{i-t} |\delta_i(\mathbf{s}_i, \mathbf{a}_i)|\right]\right]^{\frac{1}{2}} \tag{10}$$

This quantity can be learnt via bootstrapping, in the normal fashion that Q-value estimators are learnt using the following recursive sum:

$$\mathcal{E}_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t) = \left(\delta_t(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma(\mathcal{E}_\phi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}))^2\right)^{\frac{1}{2}} \tag{11}$$

Equation 11 provides a learnable metric for epistemic uncertainty that can be learned via bootstrapping for Q-value networks.

One detail is that (8) requires taking the expectation of the Bellman residual error projected through the squared error loss. In practice, except for the simplest of environments, this is not entirely possible as it requires access to the reward and next state distribution for given state and action pair. In our experiments, we take the expectation of single state transition samples, and note that this results in inflated estimates of epistemic uncertainty. More concisely, the resulting learnt quantity is much closer to (3).

Explicit Epistemic Uncertainty Estimates on Gymnasium Environments

We implement DQN with explicit epistemic uncertainty estimation on four Gym environments with increasing difficulty and complexity [Brockman et al., 2016]: CartPole, Acrobot,

MountainCar, and LunarLander. The goal is to study how this measurement behaves throughout the learning process. Note that **we do not use this measurement of epistemic uncertainty to motivate exploration or mitigate risk as in other works**, the goal is to simply study its behaviour during a standard training run of DQN. We aggregate results over 150 runs using varying hyperparameters shown in Table 1.

On CartPole in Fig. 3, the epistemic uncertainty starts small, increases and then decreases, while evaluation performance exhibits a mostly upward trend. This is perhaps expected, since state diversity –and therefore reward diversity– starts small and increases during training, model uncertainty follows the same initial trend. Eventually, model uncertainty falls as Q-value predictions get more accurate through sufficient exploration and Q-value network updates, leading to better performance and lower model uncertainties. This trend is similarly observed in Acrobot and MountainCar, albeit to a lesser extent. Conversely, it is not observed for the more challenging LunarLander, where the trend of the F-value increases monotonically and plateaus out in aggregate.

Analyzing results from individual runs reveals that in environments with rewards that vary fairly smoothly such as CartPole and LunarLander (Figure 4(a)), the F-value can show either performance collapse as in the example shown in CartPole, or indicate state exploration activity as in LunarLander. In environments with sparse rewards such as Acrobot and MountainCar (Figure 4(b)), the F-value is indicative of agent learning progress: we observe spikes in uncertainty when the model discovers crucial checkpoints in the environment. For Acrobot, this occurs when the upright position is reached and the reward penalty is stopped. In MountainCar, this occurs when the agent first reaches the top of the mountain, which completes the environment. More examples of similar plots are shown in Appendix 2.2.

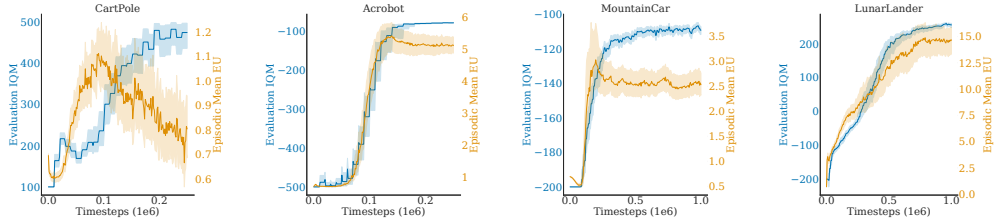

Figure 3: Aggregate episodic mean epistemic uncertainty and evaluation scores across four environments using DQN.

Table 1: DQN Hyperparameters for CartPole, Acrobot, MountainCar and Lunarlander

Parameter	Value
Constants	
optimizer	AdamW
number of hidden layers (all networks)	2
number of neurons per layer (all networks)	64
non-linearity	*ReLU*
number of evaluation episodes	50
evaluation frequency	every 10×10^3 steps
total environment steps	
CartPole	250×10^3
Acrobot	250×10^3
MountainCar	1×10^6
LunarLander	1×10^6
replay buffer size	
CartPole	100×10^3
Acrobot	100×10^3
MountainCar	200×10^3
LunarLander	200×10^3
Ranges	
minibatch size	{128, 256, 512}
max gradient norm	[0.25, 1.00]
learning rate	[0.0001, 0.001]
exploration ratio	[0.05, 0.15]
discount factor	[0.980, 0.999]
gradient steps before target network update	[500, 2000]

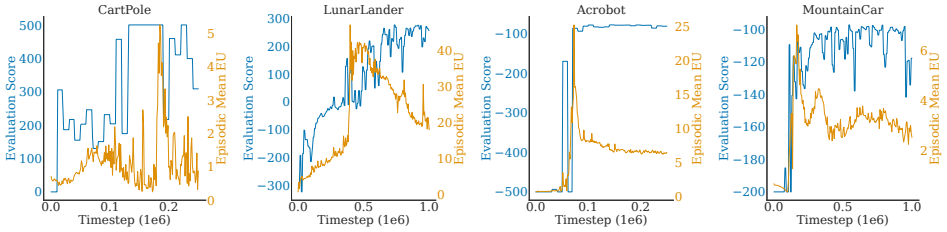

Figure 4: Examples of episodic mean epistemic uncertainty behaviour on non-sparse reward environments using DQN.

2.2 Examples of Epistemic Uncertainty Behaviour on Individual Runs of Gym Environments

2.3 Exploring Performance Degradation in Ant-v4

The performance of CCGE starts degrading after about 300k timesteps. We suspect that this is not an issue of instability in CCGE, but a result of catastrophic forgetting due to the limited capacity of the FIFO replay buffer. Several additional experiments were performed to pinpoint this reasoning. Here, we train both SAC and CCGE in Ant-v4 for 2 million timesteps, with a replay buffer size of 3×10^5 and 5×10^5. In addition, we also implement a global distribution matching replay buffer [Isele and Cosgun, 2018] for both replay buffer sizes for both learning algorithms. The IQM results for each configuration, displayed in Fig. 9, were aggregated using 20 random initial seeds.

From the results, while the performance of CCGE in the default configuration does degrade to the peak level of SAC, the performance of SAC also ends up degrading a significant amount when training is continued for an extended amount of time. Utilizing a larger replay buffer size does aid in reducing this performance degradation in CCGE, but seems to hurt performance in SAC. When using global distribution matching — a technique for circumventing catastrophic forgetting — the performance degradation of both CCGE and SAC is much less severe, inline with the results obtained by Isele and Cosgun [2018].

The results here are interesting, posing an interesting question for future research on whether CCGE can be used to reduce the replay buffer size through an oracle policy. That said, the results also suggest that the performance degradation is not necessarily instability, nor is it an artifact caused by CCGE's implementation alone as it is also present in SAC.

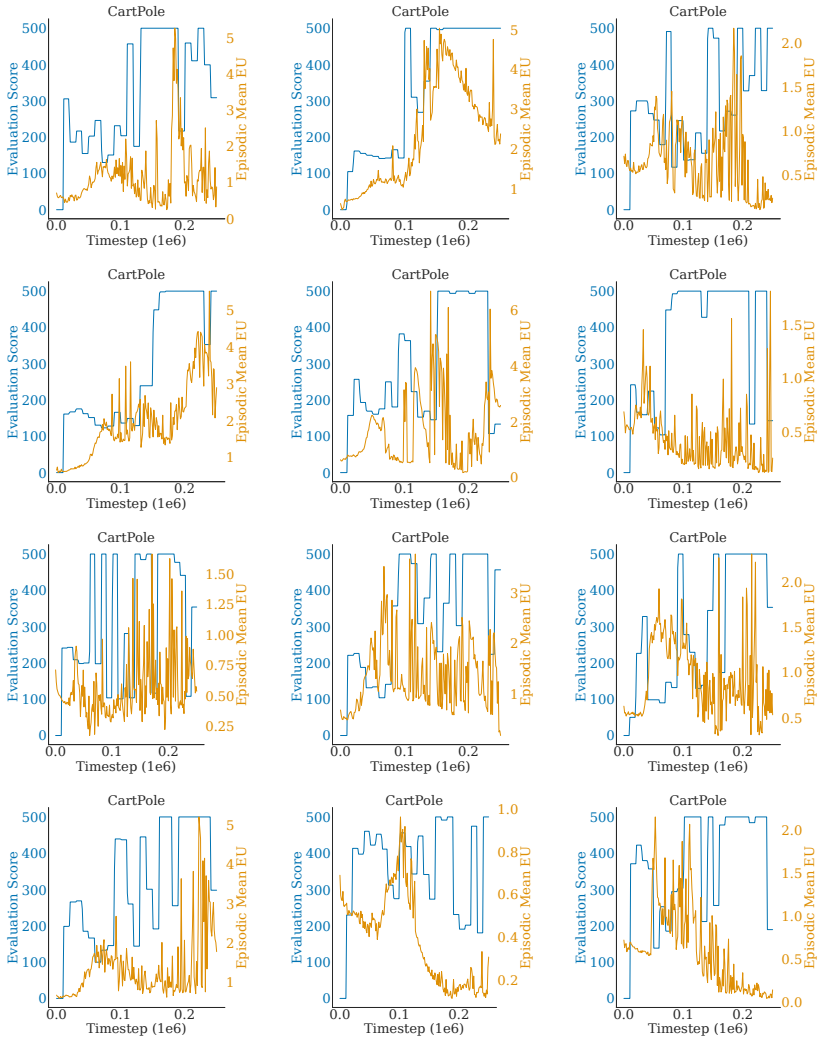

Figure 5: Episodic mean epistemic uncertainty and evaluation performance curves on various runs of CartPole.

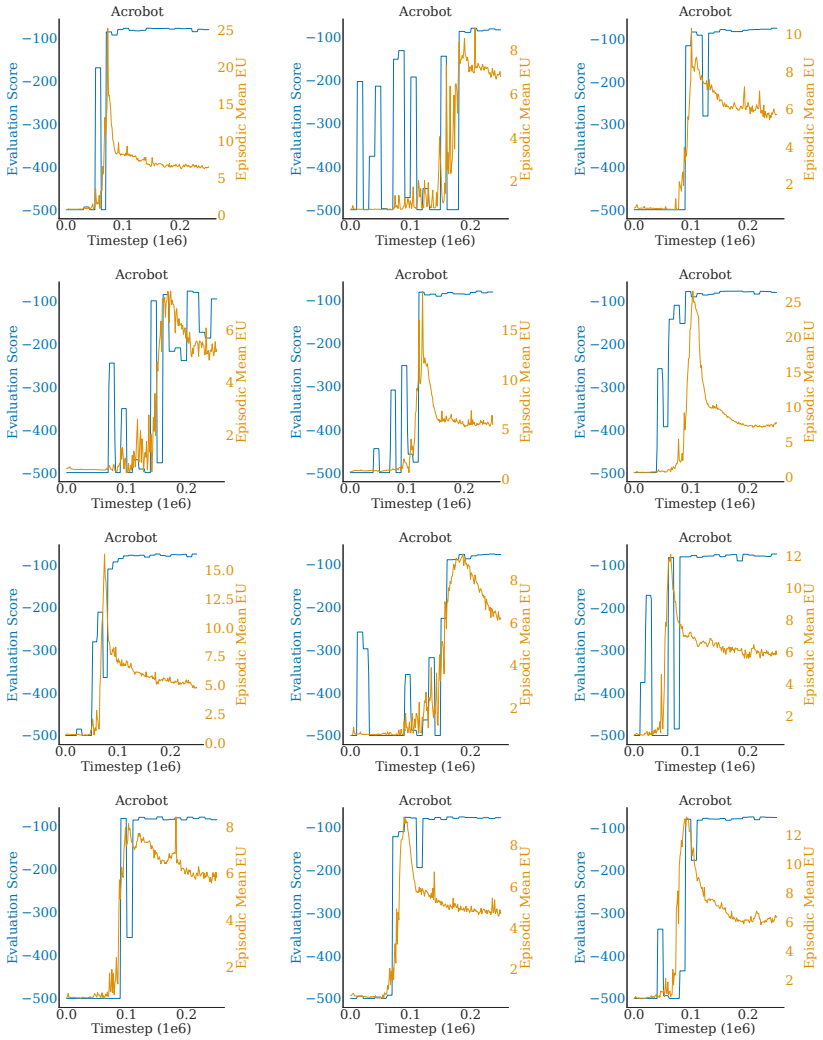

Figure 6: Episodic mean epistemic uncertainty and evaluation performance curves on various runs of Acrobot.

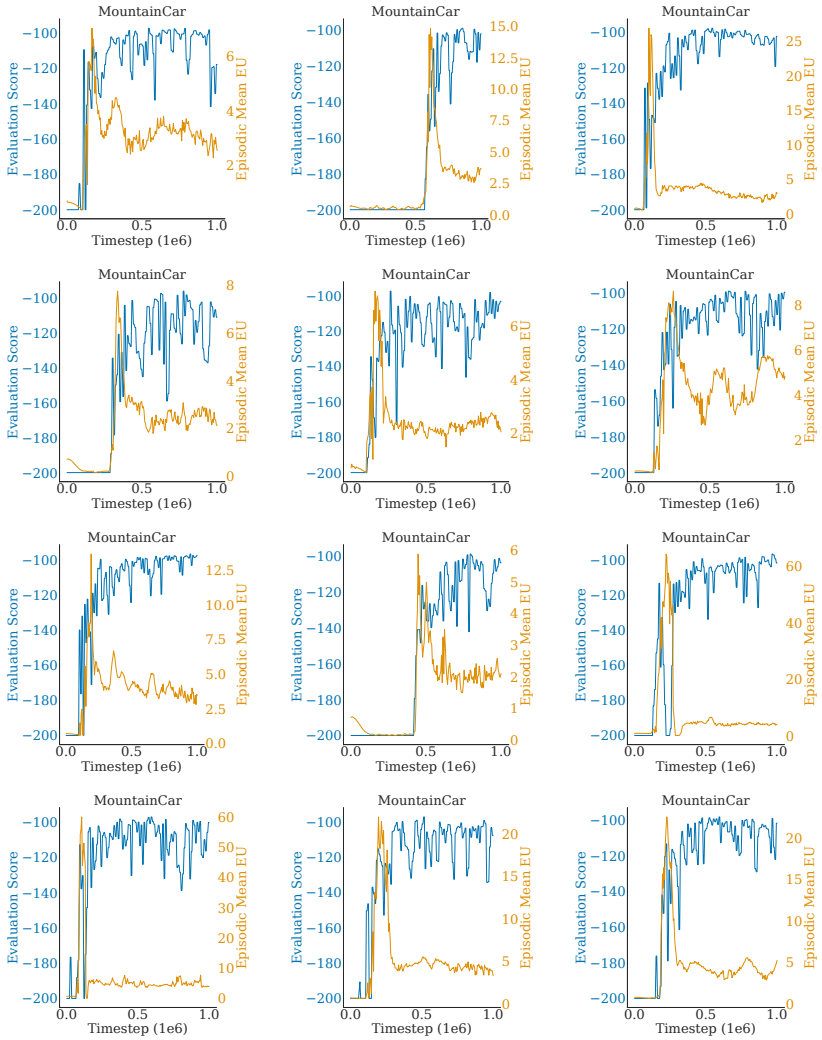

Figure 7: Episodic mean epistemic uncertainty and evaluation performance curves on various runs of MountainCar.

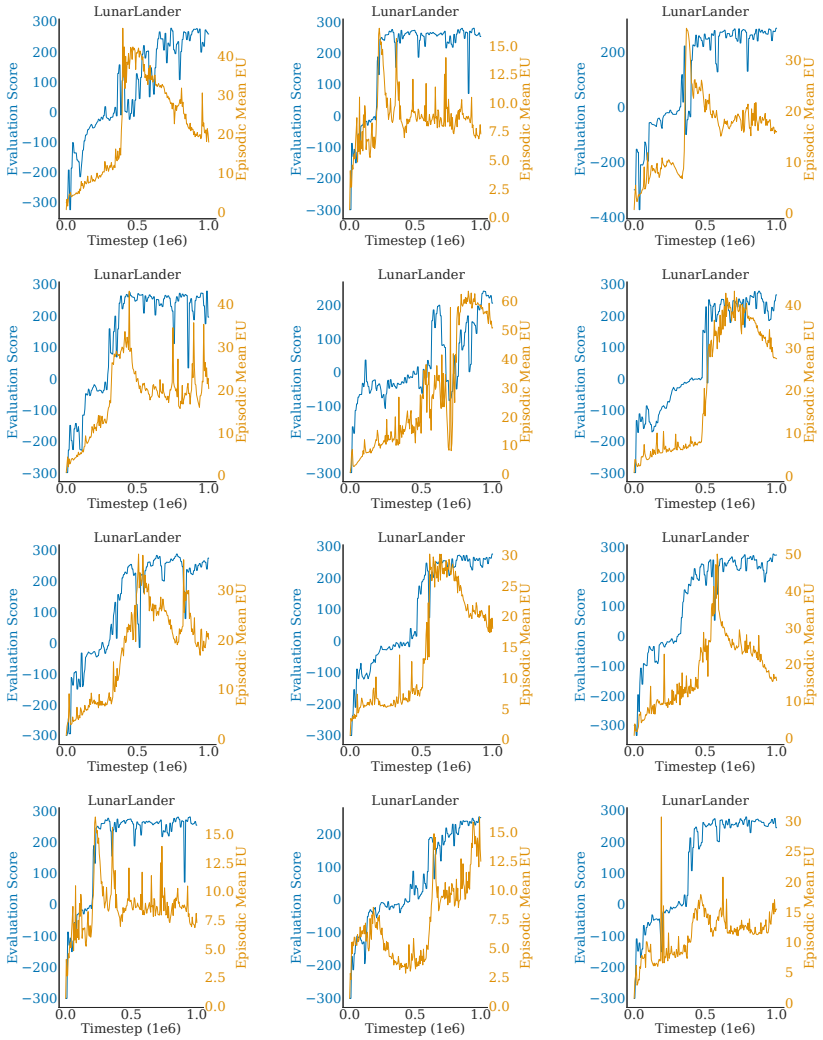

Figure 8: Episodic mean epistemic uncertainty and evaluation performance curves on various runs of LunarLander.

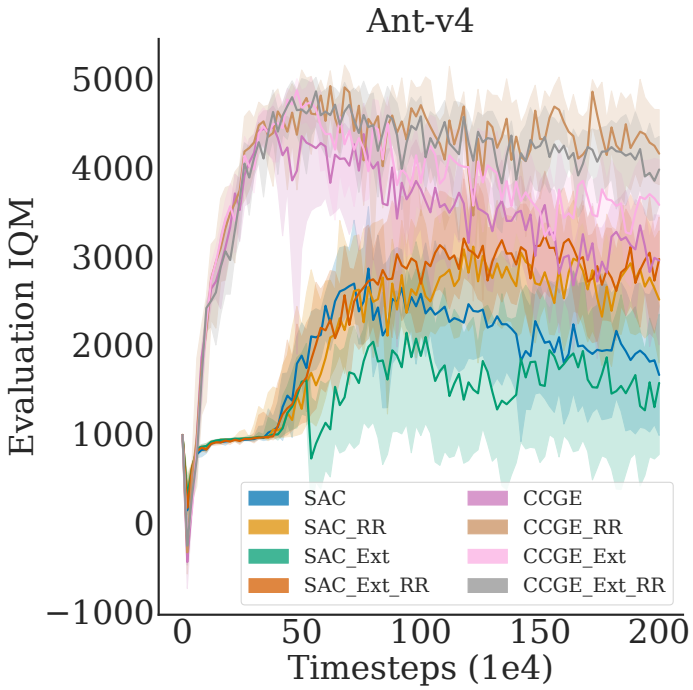

Figure 9: Learning curves of CCGE and SAC on the Ant-v4 environment, using different replay buffer sizes and different replay buffer forgetting techniques, trained for 2 million timesteps. Runs with the `_Ext` extension denote experiments done with a replay buffer size of 5×10^5, and runs with the `_RR` extension denote experiments where global distribution matching was used as a forgetting technique.

2.4 SAC and CCGE Hyperparameters for Mujoco Tasks

Table 2: SAC and CCGE Hyperparameters for Hopper-v4, Walker2d-v4, HalfCheetah-v4 and Ant-v4

Parameter	Value
Constants	
optimizer	AdamW
learning rate	4e-4
batch size	256
number of hidden layers (all networks)	2
number of neurons per layer (all networks)	256
non-linearity	*ReLU*
number of evaluation episodes	100
evaluation frequency	every 10×10^3 environment steps
total environment steps	1×10^6
replay buffer size	300×10^3
target entropy	$-\dim(\mathcal{A})$
discount factor (γ)	0.99
For CCGE only	
confidence scale (λ)	1.0

2.5 AWAC, JSRL, and CCGE Hyperparameters for AdroitHand Tasks

We utilize the same SAC backbone for all three algorithms. For JSRL, we utilize JSRL Random as described in the original paper [Uchendu et al., 2022], supposedly a more performant version of JSRL which allows the oracle policy to act for a random amount of timesteps in each episode before the learning policy takes over.

2.6 AWAC, JSRL, and CCGE Hyperparameters for PyFlyt Tasks

Part of the observation space in the PyFlyt Warpaint environments uses the `Sequence` space from Gymnasium. As a result, using a vanilla neural network to represent the actor and critic is insufficient due to the non-constant observation shapes. We utilize the network architectures illustrated in Figure 10, which takes inspiration from graph neural networks to process the waypoint observations and agent state separately. All algorithms utilize the same architecture. For JSRL, we use JSRL Random — the more performant version of JSRL as described in the original paper[Uchendu et al., 2022].

Table 3: AWAC, JSRL, and CCGE Hyperparameters for AdroitHandDoorSparse-v1, AdroitHandPen-v1 and AdroitHandHammer-v1.

Parameter	Value
Constants	
optimizer	AdamW
learning rate	4e-4
batch size	512
number of hidden layers (all networks)	2
number of neurons per layer (all networks)	128
non-linearity	*ReLU*
number of evaluation episodes	100
evaluation frequency	every 10×10^3 environment steps
total environment steps	1×10^6
replay buffer size	300×10^3
target entropy	$-\dim(\mathcal{A})$
discount factor (γ)	0.92
For CCGE only	
confidence scale (λ_{CCGE})	1.0
For AWAC only	
Number of demonstration transitions	100×10^3
Pretrain epochs	10
Lagrangian multiplier (λ_{AWAC})	0.3

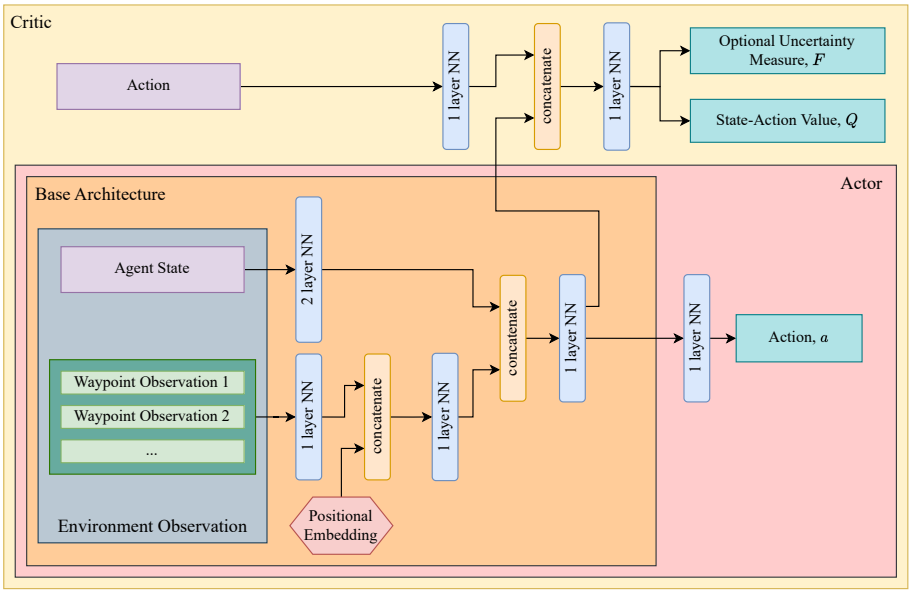

Figure 10: Block diagram of the architectures of the actor and critic used for PyFlyt experiments.

Table 4: AWAC, JSRL, and CCGE Hyperparameters for PyFlyt/Fixedwing-Waypoints-v0 and PyFlyt/QuadX-Waypoints-v0.

Parameter	Value
Constants	
optimizer	AdamW
learning rate	4e-4
batch size	1024
number of neurons per layer (all networks)	128
non-linearity	*ReLU*
number of evaluation episodes	100
evaluation frequency	every 10×10^3 environment steps
total environment steps	1×10^6
replay buffer size	300×10^3
target entropy	$-\dim(\mathcal{A})$
discount factor (γ)	0.99
For CCGE only	
confidence scale (λ_{CCGE})	0.1
For AWAC only	
Number of demonstration transitions	100×10^3
Pretrain epochs	10
Lagrangian multiplier (λ_{AWAC})	0.3