A Multi-Agent System to aid the Automation of Search and Examination in Hazardous Environments

David Smyth

Supervisors: Prof. Michael G. Madden
Dr. Frank G. Glavin

College of Science and Engineering
National University of Ireland, Galway

This dissertation is submitted for the degree of
Master of Science

National University of Ireland,
Galway

September 2019

Abstract

Systems utilising autonomous agents are becoming increasingly pervasive in today's society, garnering commercial interest and research funding in a variety of domains ranging from home automation to undersea exploration. This has stemmed from a resurgence in interest in Artificial Intelligence over the last number of years. Globally, we are starting to move towards an age of automation through physical and software systems that exhibit redundancy, modularity and robustness. Research into how to induce intelligent decentralised behaviour in such systems will be key to their development.

Autonomous systems that can be operated remotely are highly suited to disaster scene management, due to their dangerous and uncertain nature. This thesis outlines the design and implementation of the software component of a team of autonomous robots that implement a multi-phase disaster scene management plan. The problem domain involves robotic aerial vehicles that have sensors and actuators to interact with their environment.

First, the design and development of a virtual high-fidelity simulation environment using a game engine is outlined. This simulation environment was used extensively in the research project, ROCSAFE, which motivated the work in this thesis. It helped to address the problem of generating data to carry out training, testing and validation of systems used to aid hazardous scene management.

We then discuss the development of an autonomous software system to aid the surveillance of a disaster scene. We first address a surveying problem, whereby a swarm of aerial vehicles need to use sensors to record data at each point in a discretised region defined by a bounding polygon. This is a standard early phase of the forensic examination of a crime scene and the data gathered can be used to guide strategies during subsequent phases of the disaster management process.

The second is a stochastic search problem, where multiple agents are used to pinpoint the location of a target. "Target" means anything that can be sensed by the agents. It is assumed that agents are fitted with sensors and actuators and can move around the bounding region freely. Sensor readings are assumed to have some inherent noise, and a stochastic approach is presented to account for this. Results and analysis are presented to give insight into how the software can be used to formulate search control strategies that achieve some realistic objectives.

Table of contents

Declaration

I declare that this thesis has been composed by me and I have not obtained a degree from the National University of Ireland, Galway, or elsewhere, on this work previously.

David Smyth
September 2019

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation and Scope

Technologies based on systems of intelligent agents have received an increasing amount of interest in both research and industry in recent years. Mobile robotics applications such as Simultaneous Localisation and Mapping (SLAM) [96] and target detection and tracking have received substantial interest in particular, with a focus on the multi-robot setting [80], [73]. Application domains have naturally gravitated towards problems that would pose hazards to physical human presence [61]. The work outlined in this thesis has been funded and motivated by a EU Horizon 2020 research project called ROCSAFE (Remotely Operated Chemical, Biological, Radiological, Nuclear (CBRN) Scene Assessment and Forensic Examination) [3]. The goal of the project is to fundamentally change how CBRNe scenes are assessed by ensuring the safety of crime scene investigators. The project uses autonomous robots to reduce the need for investigators to enter high-risk areas when their job requires the determination of threats and gathering of forensic evidence.

The scope of this thesis lies in developing intelligent autonomous agents in the realm of mobile robotics, in order to carry out tasks that are of value in scenarios that would present dangers to humans. A system of heterogeneous aerial vehicles is considered, with sensing capabilities as well as high-level navigation capabilities. The focus is primarily on designing software that implements strategies to efficiently carry out area surveillance and target localisation, independent of the hardware on board the aerial vehicles. Realistic constraints such as operational speeds and battery capacity are taken into account. The use of high-fidelity 3-D simulation environments has also been investigated, due to the highly complex nature of hazardous real-world scenarios that require surveillance. The work done in this thesis has complemented much of the research that has been carried out in parallel in the ROCSAFE project, for example image processing modules.

1.2 Research Questions

We identified two main research questions to be investigated for this thesis. They are as follows:

1. Can an agent-based software system be developed to run on a system of heterogeneous autonomous aerial vehicles to aid the tasks of scene surveying and target localisation in a hazardous environment?

2. Can a high-fidelity simulation environment be created, which can generate realistic salient data to support the process of prototyping AI systems designed to aid the management of real-world hazardous scenarios?

Both research questions were derived from the aims of the ROCSAFE project. Deliverable 2.3[1] of the project provides a gap analysis in the management of the forensic phase of a crime scene. Among the topics that were addressed were

- Scene Assessment: The document identifies a gap in hazardous scene assessment capabilities at the time of publication: "*Currently only limited means are available for remote scene assessment. These are slow and uncoordinated*". The desired solution that the ROCSAFE project was intended to provide was stated as "*To provide a drone-based overview and reconnaissance of the scene and the impacted area and to provide a means to feed that data into a CDM system*". The document states motivation for this solution: "*Scene assessment and crime scene planning is significantly aided by photography and mapping. A critical step in an evidence collection plan is rapid and safe overview of the scene*".

- Detection, Identification and Monitoring (DIM): Deliverable 2.3 outlines the requirements of DIM in relation to forensic analyses: "*The DIM concept requirements are based on out-ruling possibilities one by one. This should be relied on by ROCSAFE and followed by ROCSAFE. The DIM process is not concerned with the detection equipment – it is the process of identification that is highlighted*". It is noted that for the ROCSAFE project, carrying this out using remote vehicles is a key output: "*In order to reduce risk and to determine areas of contamination, remote means are required, and which should follow the DIM sequence of action*". The document states that a proposed solution should incorporate the following: "*A combination of first using UAVs and then deploying directed UGVs will permit the DIM sequence to be employed. This is turn will reduce the complexity and eventual cost of detection and identification*".

[1]https://www.nuigalway.ie/rocsafe/research/

The first research question is based on these identified gaps. We explore the two parts of the first question, related to scene surveying and target localisation, in Chapters 4 and 5 respectively, where we provide a technical definition of the problem. The second research question arose from the first research question, since prototyping and validating systems to be run on RAVs in hazardous environments in the real world is prohibitive due to the high cost.

1.3 Contributions

The contributions of the work done for this thesis are summarised:

- We designed a high-fidelity simulation environment, which was open-sourced for public use with an MIT Licence at the linked Github repository.[2] The development of the simulation environment is discussed in Chapter 3.
- We created a software system consisting of agents which tackle the stochastic target localisation problem, discussed in Chapter 5. The software implementation was open-sourced and can be found at the linked repository on Github.[3]
- We designed and open-sourced software that can generate routes for multiple agents to cover a uniformly-spaced grid according to various objective functions. The software implementation can be found at the linked repository on Github.[4]
- We designed and open-sourced a user interface which allows the user to specify the bounding points of a polygonal region using the WGS84 coordinate system. Using the grid generation software, the user can generate a grid over this region. Using the agent-routing software it is then possible to request for routes be generated for multiple agents to execute in a real or virtual simulation environment. If the virtual environment developed above is chosen, the user can optionally request for data-gathering tasks be carried out while visiting each grid point. The integrated software implementation can be found at the linked Github repository.[5]

Accepted publications related to this research are listed:

1. Smyth, D. L., Fennell, J., Abinesh, S., Karimi, N. B., Glavin, F. G., Ullah, I., Drury, B., and Madden, M. G. (2018b). A Virtual Environment with Multi-Robot Navigation,

[2]https://github.com/NUIG-ROCSAFE/CBRNeVirtualEnvironment/releases
[3]https://github.com/DavidLSmyth/OccupancyGridAgent
[4]https://github.com/DavidLSmyth/CBRNeMissionDesigner
[5]https://github.com/NUIG-ROCSAFE/CBRNeVirtualEnvironment/

Analytics, and Decision Support for Critical Incident Investigation. In *International Joint Conference on Artificial Intelligence*, pages 5862–5864, Stockholm. International Joint Conferences on Artificial Intelligence

2. Smyth, D. L., Glavin, F. G., and Madden, M. G. (2018c). Using a Game Engine to Simulate Critical Incidents and Data Collection by Autonomous Drones. In *2018 IEEE Games, Entertainment, Media Conference, GEM 2018*, pages 436–440, Galway. Institute of Electrical and Electronics Engineers Inc

3. Smyth, D. L., Abinesh, S., Karimi, N. B., Drury, B., Ullah, I., Glavin, F. G., and Madden, M. G. (2018a). A Virtual Testbed for Critical Incident Investigation with Autonomous Remote Aerial Vehicle Surveying, Artificial Intelligence, and Decision Support. In *IWAISe 2nd International Workshop on A.I. in Security*, pages 88–93, Dublin. Springer

1.4 Thesis Outline

This rest of this thesis is structured as follows:

1. Chapter 2 outlines some essential background knowledge related to the work discussed in this thesis. We provide sections on the ROCSAFE project, agent and multi-agent systems, Hidden Markov Models, Dynamic Bayesian Networks, sequential statistical Hypothesis Testing and high-fidelity game engine environments. We also provide a review of related work.

2. Chapter 3 provides the details of the development of the high-fidelity simulation environment mentioned above in the contributions section. We provide an exposition on the techniques that we used to achieve a high level of realism. We also outline the integration of a plugin called Airsim [81], which facilitated the use of virtual **R**emote **A**erial **V**ehicles (RAVs). We finally discuss potential future work and applications for the simulation environment.

3. Chapter 4 is concerned with outlining the solution we implemented to solve the problem of Scene Surveying and is concerned with the first research question stated in Section 1.2. A simplified version of the problem is first explored. We provide algorithms and some sample results outlining the advantages of the explored solution. We then move to the complete version of the problem, which introduces additional constraints. We discuss our implemented solution, which uses an open-source constraint-programming solver.

4. Chapter 5 addresses the target localisation problem using a system of RAVs. This pertains to the first research question stated in Section 1.2. We provide the details of the agent-based approach that we took to solve this problem. We explain how a Dynamic Bayesian Network (DBN) was used to describe the conditional relationships between relevant stochastic features of the agent's environment. Results of running simulations while perturbing user-specified parameters are provided in order to show how some configurations can lead to much more desirable results than others, which offer insight into how to calibrate the search parameters for the specific application of a potential user.

Chapter 2

Background Knowledge and Related Work

As mentioned in Chapter 1, the research outlined in this thesis was motivated and funded by an EU Horizon 2020 project called ROCSAFE, which involves investigating the automation of certain aspects of the management and examination of the forensic phase of a crime scene investigation. Accordingly, this chapter outlines existing research specifically related to the application of agent-based systems in this area, while also providing a broad overview of single and multi-agent systems and general techniques commonly used in such systems to solve abstract problems.

First, a synopsis of the ROCSAFE project, which funded and motivated this work is presented. Then a broad overview of agent-based systems is given, mainly outlining common approaches in agent and multi-agent system design. This leads to a more focused discussion on how agents and multi-agent systems have been designed in work that is tightly related to that of this thesis. Important concepts and algorithms relating to multi-agent probabilistic search are then outlined. Techniques used to reach the current state-of-the-art for these problems are identified and discussed. Work previously done which relates to the research questions stated in Chapter 1 is appraised and reviewed. Finally, previous work and available tools related to developing simulation environments are outlined.

2.1 The ROCSAFE Project

Remotely **O**perated *Chemical, Biological, Radiation, Nuclear, explosive (CBRNe)* **S**cene **A**ssessment and **F**orensic **E**xamination (ROCSAFE) is an EU Horizon 2020 project. Horizon 2020 is an EU Research and Innovation Program, with almost €80 billion in total funding allocated from 2014 to 2020. The ROCSAFE project comes under the *Secure societies - Protecting freedom and security of Europe and its citizens* programme, whose objective is *"to foster secure European societies in a context of unprecedented transformations and growing global interdependencies and threats, while strengthening the European culture of freedom and justice".* [1]

Specific details of the ROCSAFE project can be found on the Horizon 2020 website[2] and ROCSAFE website[3]. We summarise the relevant details here, based on these primary sources. The high-level goal of ROCSAFE is to fundamentally change how CBRNe events are assessed, given that current practices require personnel to enter hazardous areas with unquantified risks. The project does not aim to provide a first response to CBRNe events; rather it intends to provide support to the forensic phase of the investigation, which is chiefly concerned with the collection, preservation, and scientific analysis of evidence during the course of an investigation, ideally leading to the ability to present admissible evidence during a criminal investigation.

CBRNe events are exceedingly difficult to prepare for because they are so rare and diverse, with the consequence that limited data from real events is available to use as a reference. Despite this, many of the procedures related to forensic investigations are well-defined, in order to preserve the chain-of-custody of evidence. The work done in this thesis was identified at the inception of the project to aid:

1. The initial phase of the forensic investigation, in which a survey of the scene is carried out in order to provide high-level information related to the examination area to the crime scene manager.

2. The identification and localisation of forensic evidence that can be detected using sensors developed as part of the project.

3. Prototyping, testing and validation of some of the technologies related to this project. This was done by designing a high-fidelity simulation environment from scratch.

2.2 Agents and Multi-Agent Systems

This section provides some general background knowledge related to intelligent agents and multi-agent systems. We introduce common terminology and outline approaches that are commonly taken when designing agents.

[1]https://cordis.europa.eu/programme/rcn/664463/en
[2]https://cordis.europa.eu/project/rcn/203295/factsheet/en
[3]http://rocsafe.eu/

2.2.1 Agents

An essential concept which is repeatedly referred to in this thesis is that of an *agent*. The term "*agent*" is an abstract one with no single definition universally accepted in the literature. Most definitions agree reasonably closely with the one provided in AI: A Modern Approach by Russell and Norvig: an agent is "*anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.*" [77, p. 34]. Figure 2.1 helps to illustrate this concept. The box containing the question mark represents the agent's

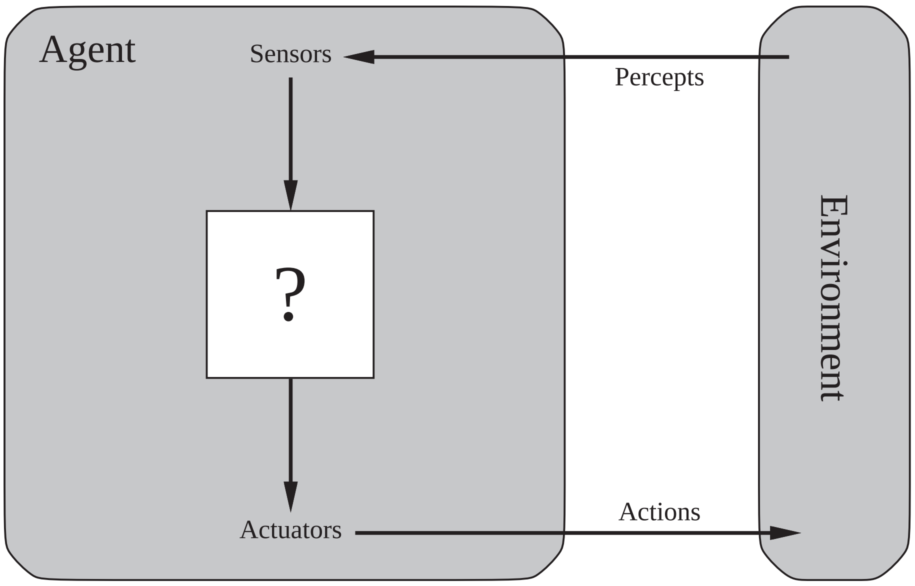


Fig. 2.1 Agent-environment interaction based on [77, p. 35]

internal decision-making process, which generally speaking involves choosing an action, given the complete history of everything that the agent has perceived. The mapping from percept sequences to actions is described as the agent function. The agent function is an abstract notion; at a lower level an agent program implements the agent function, running on some physical system. A key concept in developing an agent program is rationality: a rational agent should choose actions in order to maximise its expected performance measure, given the information provided by the sequence of percepts it has received and any other environment-related knowledge that the agent may have [77, p. 37]. This thesis is concerned with designing a system of agents that should exhibit rational behaviour.

Again following the approach taken by Russell and Norvig, agents can be grouped into four common classes based on the design of the agent program, which describe the principles which underpin almost all intelligent systems [77, p. 47]:

1. **Simple reflex agents**: These agents select actions to take by ignoring the percept history to date, with the exception of the most recent percept.

2. **Model-based reflex agents**: These agents have an internal state that depends on the percept history and is updated using new percepts. Updates occur using a model of the world.

3. **Goal-based agents**: These agents are an extension to model-based reflex agents. They have some information about whether they have reached a "goal" state, which is desirable. The agent can choose actions to take based on the model.

4. **Utility-based agents**: A utility function is used as an internal performance measure in order to select actions. This allows an agent to pick among actions that may not immediately lead to a goal state.

Recent work in the field of AI has strongly focused on *learning agent* architectures through approaches such as reinforcement learning [93], which are designed to improve the agent's performance through time, according to a fixed *performance standard*. The key difference between the agent types listed above and a learning agent is that a learning agent has the ability to improve its performance through time [77]. Its decision making process is not necessarily static and can change beyond its initial programming. An explanation of the architecture of a general learning agent can be found in Russell and Norvig [77, p. 55]. Implementing each of these different types of agents requires varying degrees of effort. Depending on the problem, different architectures may be more or less suited, which suggests that fully understanding the problem that is attempted to be solved is of fundamental importance when designing an agent.

2.2.2 Multi-Agent Systems

Matching the above, there is no universally agreed-upon definition of a *Multi-Agent System* (MAS). A definition that captures the core aspects described in most MAS-related publications is provided in a field review paper by Stone and Veloso: "*Multiagent Systems (MAS) is the subfield of AI that aims to provide both principles for construction of complex systems involving multiple agents and mechanisms for coordination of independent agents' behaviors*" [92]. While the design of individual agents tends to focus on maximising a performance measure, the design of a multi-agent system is rather more multi-faceted. Weiss [106] notes that it is almost always oriented towards answering the question of "*when and how to interact with whom*". Multi-agent systems have been proposed as a solution to many problems that modern AI attempts to tackle for some of the following reasons, which are outlined in more depth in [92]:

- Some problems by definition can be described as a multi-agent system. An example is an organization that may want to model its internal affairs with a single system. The different departments have their own sub-systems that have differing priorities and capabilities; their interactions naturally can be thought of as interactions between independent agents, in accordance with the definition provided at the start of this section.
- The accomplishment of a task can be expedited significantly by using multiple agents. Multi-agents systems are part of the field of Distributed Artificial Intelligence (DAI) and so problem domains that decompose into several independent tasks that can be handled by separate agents can benefit from their use. The problem of efficient task allocation in multi-agent systems has been well-studied in the literature [32].
- Robustness is an often-cited benefit of multi-agent systems. Distributed control means that failure of a single agent (mechanical or otherwise) may be tolerated.
- Multi-agent systems are often more scalable than single-agent systems. The necessary modularity of multi-agent systems means that adding new agents to the system can often be a solution to a more difficult problem, rather than adding new capabilities to a monolithic system.
- The modularity of multi-agent systems means that the design and programming of them may be simplified. For example, rather than solving a multi-objective problem with a single agent, a single-objective problem may be solved with multiple agents. A result of this is that multiple cheap robots may be used to outperform a single, expensive robot [34].

We use these concepts in Chapters 4 and 5.

2.3 Stochastic Target Localisation Related Literature

This section discusses the literature related to the problem of target localisation, pertaining to the first research question in Section 1.2. The necessary background knowledge to fully understand these approaches is provided in the next section (Section 2.4).

We state problem of stochastic target localisation in a general setting as follows:

Given a region of space to explore and a set of heterogeneous autonomous aerial vehicles with sensing capabilities, devise a search strategy with a search cut-off criteria which will accurately return either the locations of the targets if one or more is present, or return that no targets are present, in the shortest possible time.

The problem and its variants have been studied extensively and we survey the existing literature behind it here in chronological order. The study of problem can be traced back to a series of three papers by Koopman [46], [47], [48]. The first of the three, entitled "*Kinematic Bases*", references World War *II* as a primary motivation behind the series, citing "*detection, location, and identification of targets*" as a key problem which required an urgent solution [46]. Three key general features of search are identified, which are addressed individually in each of the three papers:

1. "*Kinematic Bases*" addresses search with a focus on the kinematic constraints involving the positions, geometric configurations and motion of the searchers and targets [46].
2. "*Target Detection*" is concerned with the stochastic nature of the sensors and instrumentation that is used when navigating and detecting [47].
3. "*The Optimum Distribution of Searching Effort*" examines the probability of contact under the stated conditions. It also provides insight on how to optimise results by choosing a search strategy which maximises the probability of contact [48].

This introduced the problem in structured manner and offered a strong insight into potential future research. In subsequent years the research branched into problems that involved the motion of both the searchers and the target [91], or just the searchers [15]. Since the work in this thesis is primarily concerned with stationary targets, we refer the reader to the PhD. thesis [53] for a broad review of the literature related to search with a moving target.

Chew explored an instance of the search problem which is very close to the version this thesis explores [15]. The following assumptions were made

- There is an object to be found in one of R boxes at discrete locations or an $R+1^{st}$ representing the absence of the object.
- Sampling locations for the objects returns values with a false negative rate α_i, which depends on the location, i.
- The boxes at discrete locations can be searched sequentially.
- All outcomes are independent, conditional on the location of the object and the search procedure used.

- A loss function introduces a cost for concluding the search when the object has not been found.

Note that no false positives were assumed to be observed. A dynamic programming approach was proposed, based on [8], which was shown to be optimal when restricted to a fixed number of inspections. In this case, "optimal" is taken to mean that the strategy minimises the searcher's total expected cost of searching and stopping. A stopping rule was also suggested, with a proof of desirable properties such as minimizing expected cost when using an optimal search procedure.

Black examined a discrete instance of the stochastic target localisation problem for a stationary target, following the similar assumptions as [15] above, with the exception of the possibility of the object not being present (represented by the $R+1^{st}$ location) and without a cost for concluding the search when the object has not been found [7]. A sampling policy was proposed that was shown to be optimal in terms of minimising the expected cost of the search, according to an arbitrary cost function. The optimal policy was summarised as "*Always look in the region for which the posterior probability (given the failure of earlier looks) of finding the object divided by the cost is maximum.*". The policy was not described in terms of a dynamic program, which the author claimed provides more transparency to the solution, as opposed to previous work.

Ross examined the search problem with the assumptions set out in [15] with the exception of the possibility of the object not being present (represented by the $R+1^{st}$ location) but did include the issue of when to conclude the search [75]. The paper showed that in general, an optimal search strategy exists for the problem with a penalty cost for stopping the search. He proved that when searching is warranted (i.e. the cost of stopping is small relative to the cost of further searching), the optimal search strategy is the same as in the no-penalty case. This means that to optimally solve the general problem of searching and stopping when a penalty cost for stopping is charged (as set out in [15]), the only additional requirement to the use of the optimal search strategy is a stopping time s* that is optimal when used with that search strategy.

Chew then iterated on his previous work [15] and outlined a best stopping rule to accompany the optimal search procedure described in [15] with the work in [16]. This went a step further than [75], since it included the possibility that the object is not present in the search region. The optimal stopping rule was found to be derived from the optimal dynamic programming strategy outlined in [15], subject to a realistic technical condition relating the search termination cost, the cost of searching each location and the probability of a missed detection in each location.

Kimeldorf and Smith studied the more general problem of locating n objects hidden among m boxes, where m is known and n is unknown [43]. They assumed that there was a cost associated with searching each box and that the distributions of p (the number of boxes) and π (the distribution objects among the boxes) were known. They also assumed that they knew when all the objects had been found and that there was no associated search termination cost. Their results showed that for special cases of the distributions of n and p, optimal search strategies can be found which are an extension of the general strategy set out in [8].

Assaf and Zamir tackled the same problem, but imposed the constraint that the distribution of π was not known exactly, but instead some partial information was known about π [2]. Their findings showed that even small random fluctuations, or statistical inaccuracies, may give rise to results that are greatly different from the results obtained when π is perfectly known. This suggested that a robust strategy in the case of a possibly inaccurate knowledge of π would be desirable.

Many recent approaches related to the problem of non-deterministic detection and localisation of objects follow the groundwork laid by Elfes [29], whereby an *occupancy grid* is used to represent the distribution of the target over the cells of a spatial lattice. Elfes describes an occupancy grid as a "*probabilistic tessellated representation of spatial information*". In simpler terms, the occupancy grid is construct that can be used to maintain probabilities of occupation of cells that partition a region of space. The approach was taken bearing in mind recent advancements in mobile robotics, with sensing and navigation abilities that allowed the processing of information from complex environments. There was an emphasis on designing the model to deal with an environment of which little may be known in advance, which is in contrast to previous approaches which assumed most aspects of the environment were known. The occupancy grids representation was defined to maintain an estimated environment state base on spatio-temporal data. A general Bayesian update rule was stated, which takes the current probabilities of cell occupation and updates them based on a new sensor reading:

$$\text{Probability of occupancy of grid cell } i \text{ given the set of observations } \{o_1, ..., o_{t+1}\} = \frac{p(o_{t+1}|cell_i occupied) \times p(cell_i occupied|o_1, ..., o_t)}{\sum_{k=1}^{\#gridcells} p(o_{t+1}|cell_k occupied) \times p(cell_k occupied|o_1, ..., o_t)}$$

This allows the agent to maintain and recursively update the occupancy grid distribution based on new observations. This update rule is discussed in greater detail in Section 2.4.3. Elfes lists sonar based mapping and path planning as major applications of the occupancy grid framework.

A decentralized Bayesian approach to coordinating multiple autonomous mobile sensors, with the goal of localising a single stationary target, was presented in [10]. The approach was implemented using Bayesian Filtering, discussed in greater depth in this thesis in Section 2.4.3. Each autonomous airborne vehicle maintains its own probability density function (PDF) which is represented using an occupancy grid. Bayesian Distributed Data Fusion (DDF) was used to update each vehicle's individual PDF. The authors cited "*scalability, modularity and adaptability*" as the motivation behind the approach. The results of running high-fidelity simulations for a target lost at sea using aerial vehicles demonstrated the feasibility of running the platform in real life.

In [18], Chung and Burdick proposed a general framework for the search problem, with a focus on the decision-making aspect of the agent implementing the search. An abstract sensor model was proposed, which assumed that observations are part of the set $\{1,0\}$, representing a positive and negative detection respectively. They introduced a sensor model which assumes a false positive rate (α) and false negative rate (β) which can be calibrated before running the search. They also utilised an occupancy grid representation, with the corresponding Bayesian update rule updating the agent's belief based on new observations. They described the use of Wald's Sequential Probability Ratio Test (SPRT) as a termination criteria for the search [103]. The mean time to decision (TTD) was reported for simulation runs which varied the search strategy for a single stationary target in a 10×10 grid, against which future results could be compared. Chung and Burdick extended this approach to multiple agents in [19], finding that the best results were achieved using a hybrid search strategy (different search strategies were used for each agents, which had a complementary effect). Finally, [20] provides some theoretical analyses on special cases of their developed framework. Useful bounds are derived, including the minimum number of observations necessary to draw a conclusion.

[102], [100] and [101] were published by authors working on the same project, named Sensing Unmanned Autonomous Aerial Vehicles (SUAAVE) [12]. The work expands on the framework described by Chung and Burdick in [18]. Methods were proposed for dealing with observations that spanned multiple cells in the occupancy grid, motivated by the case that observations in the form of camera images do not always align with the grid. They also proposed the use of a feature-matching algorithm based on the SURF [4] algorithm as a detection method based on aerial images. More search strategies were put forward and evaluated through the results of simulation, which demonstrated the effectiveness of a multi-vehicle approach.

2.4 Stochastic Target Localisation Background Knowledge

In this section we present some of the tools that are used in the methodology of Chapter 5. Many of the techniques are used in the approaches noted in the literature review.

2.4.1 Hidden Markov Models

HMMs appear frequently in AI literature as they provide an abstract framework to deal with stochastic processes, which themselves are pervasive in their use as a tool to model real-world phenomena. This section will outline what HMMs are and their use in literature describing target localisation algorithms. A general overview of HMMs and Markov Processes can be found in [62], [33], [5] and [65], on which we base the following discussion.

Markov Processes

It is instructive to understand what is meant by a stochastic process for some of the concepts mentioned in this thesis. A random process can be described as a family of random variables indexed by a set τ: $\{X_t\}_{t \in \tau}$ [5]. Commonly in AI, stochastic processes model the evolution of a random system through *discrete* time steps: τ=$\mathbb{N}$. Examples of phenomena that are frequently modelled by stochastic processes include the growth of a bacterial population and the movement of a gas molecule [5].

A first-order discrete-time Markov process is a stochastic process that describes a discrete-time stochastic process for which the first-order *Markov property* holds [33]. The first-order Markov property states that the probability distribution of the n$_{th}$ random variable in the process is conditionally independent of all previous probability distributions in the sequence but the $n-1_{st}$: $P(X_t = x_t|X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}, ..., X_1 = x_1) = P(X_t = x_t|X_{t-1} = x_{t-1})$ [33]. This is often referred to as the memoryless property of Markov processes. In order to describe a Markov process, it is therefore necessary to describe what is known as the transition function between each pair of time-steps: $P(X_t = x_t|X_{t-1} = x_{t-1})$. A common assumption is that the rules that govern state transitions are time invariant, meaning that they can be specified generally for any given pair of time-steps. This assumption will be made for the subsequent discussion. If X_t is a discrete random variable defined over S states, the transition function can be described by a stochastic matrix T, where T$_{i,j}$ = $P(X_t = j|X_{t-1} = i)$:

$$\begin{pmatrix} T_{1,1} & T_{1,2} & \dots & T_{1,j} & \dots & T_{1,S} \\ T_{2,1} & T_{2,2} & \dots & T_{2,j} & \dots & T_{2,S} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ T_{i,1} & T_{i,2} & \dots & T_{i,j} & \dots & T_{i,S} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ T_{S,1} & T_{S,2} & \dots & T_{S,j} & \dots & T_{S,S} \end{pmatrix}$$

Some obvious results are worth pointing out; as for any stochastic matrix, by the axioms of probability theory, the sum of conditional probabilities across each column sum to one: $\sum_{j=1}^{S} T_{i,j} = 1$ and the transition probabilities over k time-steps can be described by the k_{th} power of the transition matrix: $(T^k)_{i,j} = P(X_{t+k} = j|X_t = i)$.

It also is possible to calculate the probability of the process experiencing a sequence of states from time-steps 1 as far as t, using the chain rule of probability and the Markov property: $P(X_{1:t}) = P(X_1, X_2, ..., X_t) = P(X_1) \times P(X_2|X_1) \times P(X_3|X_2, X_1) \times ... \times P(X_t|X_{t-1}, X_{t-2}, ..., X_1) = P(X_1) \times \prod_{i=2}^{t} P(X_i|x_{i-1})$. Marginalization over variables in this sequence allows the calculation of many useful quantities.

Markov processes can described by graphical models. For example Figure 2.2 displays a graphical representation of first and second order Markov processes, where a directed arrow between nodes represents a directional dependence between those nodes.

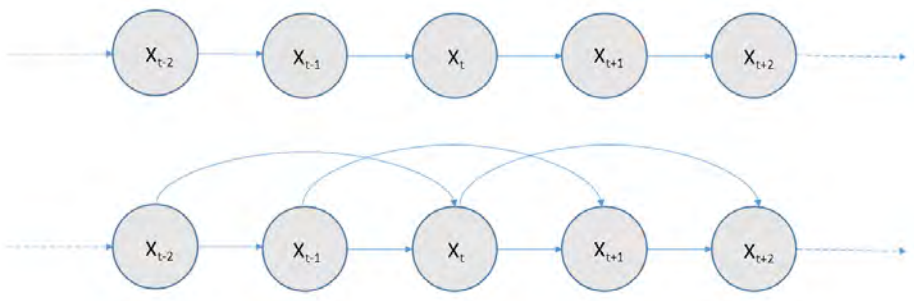


Fig. 2.2 A graphical model describing first and second order Markov processes

HMM Description

Hidden Markov Models (HMMs) are models that build on the Markov Process model, which describes the evolution of a random system in the language of probability theory. HMMs assume that the system being modelled can be described by a Markov process, but that the states of this process are unobservable [33]. This means that it is not possible to determine the state

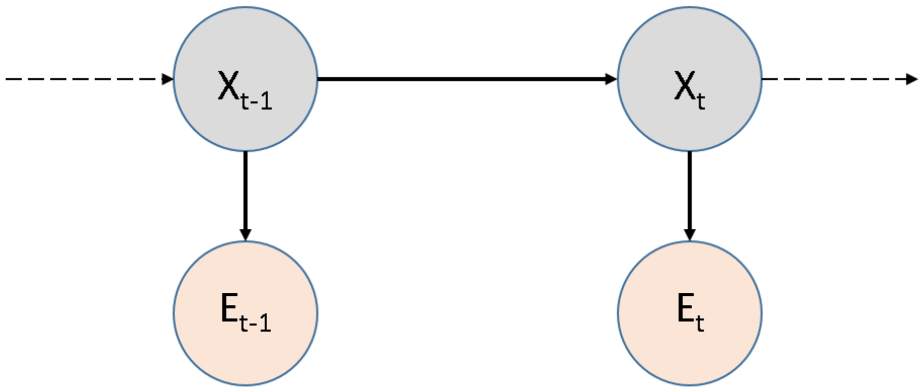


Fig. 2.3 A graphical model of a HMM

of the system exactly at any given point in time. However, it is possible to make an observation of a random variable that is related to the hidden state which yields information about the hidden state. A graphical representation of a HMM is shown in Figure 2.3, where the Markov Process is shown by the variables X_i and the observation variables are shown by variables E_i. A HMM can be specified by a triple, $\lambda = (T, O, \pi)$, where T is the stochastic transition matrix, π is the initial distribution $P(X_1)$ and O describes the conditional probability of an observation given that the system is in a certain state: $O(E_i, X_i) = P(E_i|X_i)$ [69]. Taken together, it is then possible to specify the joint distribution of the hidden state variables and the evidence variables, analogous to the Markov Process: $P(X_{1:t}, E_{1:t}) = P(X_1, X_2, ..., X_t, E_1, E_2, ..., E_t) = P(X_1) \times P(E_1|X_1) \times \prod_{i=2}^{t} (P(X_i|x_{i-1}) \times P(E_t|X_t))$. Given this representation, it is then possible to answer questions identified in [69] and [62]:

- Given the HMM, λ, what is the probability of occurrence of a particular observation sequence, $P(E_{1:t}|\lambda)$?
- Given a sequence of observations $(E_1, E_2, ..., E_n)$, what is the most likely sequence of hidden states that led to these observations? i.e. find
$$\arg\max_{X_{1:t}} P(X_{1:t}|E_{1:t})$$
- Determine the parameters of T and O, given a training set of observations, i.e. find the solution to
$$\arg\max_{\lambda} P(E_{1:t}|\lambda)$$
- *Filtering*: What is the current distribution of the hidden state given all previous evidence ("belief state") of the environment at time t: $P(X_t|E_{1:t})$?

- *Prediction*: What is the distribution of the hidden state in the future, given all evidence to date: $P(X_{t+k}|E_{1:t})$, for some k>0?
- *Smoothing*: What is the distribution of a past state given all observations up to the current point in time: $P(X_k|E_{1:t})$, for some $0 \leq$ k $<$ t?

2.4.2 Dynamic Bayesian Networks

DBNs are a generalization of HMMs and are used to model time series. The key difference between DBNs and HMMs is that DBNs are not limited in how they decompose the hidden state variables of a complex stochastic system into the variables that represent its constituent conditional distributions [77]. Rather than use a single random variable, X_t to represent the state space, a set of random variables are used [62]. DBNs can be considered as a special case of a Bayesian Network with an infinite number of nodes and repeating structure which specifies the transition probabilities between one time step and the next [45, p. 204]. The hidden state variables are represented by nodes which have an associated conditional probability distribution (CPD) which states the probability of observing a value of a random variable given its parents: $p(X_i|Parents(X_i))$, where i is the index of a node in the network [62, p. 14]. Given this representation, it is possible to calculate the joint distribution, $p(X_t|X_{t-1})$, of all N state variables at time t given the joint distribution at $t-1$ as [62, p. 14]:

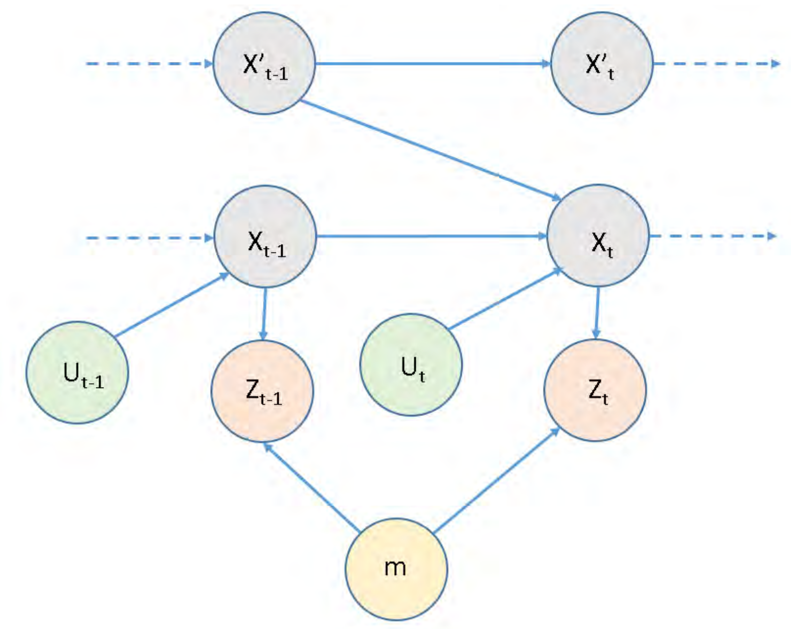


Fig. 2.4 A DBN used for **S**imultaneous **L**ocalisation **a**nd **M**apping (SLAM), based on [96, p. 311].

$$p(X_t|X_{t-1}) = \prod_{i=1}^{N} p(X_t^i|Parents(X_t^i)$$

i denotes the index of each N variables in the network. Koller and Friedman provide a technical definition of DBNs: "*A dynamic Bayesian network (DBN) is a pair $\langle B_0, B_\rightarrow \rangle$, where B_0 is a Bayesian network over $X^{(0)}$, representing the initial distribution over states, and $B_\rightarrow$ is a 2-TBN for the process. For any desired time span $T \geq 0$, the distribution over $X^{(0:T)}$ is defined as a unrolled Bayesian network, where, for any i = 1, . . . , n:*

- *The structure and CPDs of $X_i^{(0)}$ are the same as those for X_i in B_0,*
- *The structure and CPD of $X_i^{(t)}$ for $t > 0$ are the same as those for $X_i^{'}$ in $B_\rightarrow$.*

" [45, p. 204]. What this essentially means is that a DBN is a compact way of specifying an infinite set of Bayesian Networks (one for each $T > 0$) with repeating structure.

Technically, every DBN can be represented as a HMM and visa versa, however, the number of parameters that need to be determined to represent DBNs can be significantly less than that of HMMs [45]. For example, consider the case of n discrete random variables, each with support of size m. Specifying the full joint distribution without assuming any conditional independences between variables would require a number of probabilities exponential in the number of random variables, $m^n - 1$, whereas specifying the same joint distribution as factored conditional distributions may require far fewer, reaching a number of probabilities directly proportional to the number of random variables, $n \times m$ in the degenerate case [45, p. 63].

It is often convenient to categorise the random variables in the network into state variables (assumed to be hidden), control variables and observation variables [96]. An example of a Dynamic Bayesian Network can be seen in Figure 2.4. The hidden state variables are shaded in light grey, the observation variables in light orange and the control variables in light green. The directed arcs represent "instantaneous" causation, as with regular Bayesian Networks [62, p. 15]. Note that unlike HMMs, there isn't a requirement to have a single state and observation variable across each time-step and other variables can be introduced into the model.

2.4.3 Inference Algorithms

As discussed, HMMs and DBNs are useful tools for modelling complex random dynamic processes. Typically, models are used to provide some kind of inference about a process, to gain insight into its inner workings. By inference, we mean calculating the probability distributions over variables of interest. Some algorithms will be outlined here that will demonstrate how to calculate both exact and approximate inferences, using general structures for HMMs and DBNs.

First, inference algorithms commonly used with HMMs are discussed, as their representation is more rigid, meaning that it is easier to exploit their structure generally to perform inference. As previously stated, HMMs and DBNs are routinely used to model systems that have state variables that cannot be directly observed. In most cases, we are interested in inferring what the state of the system might be, given the observed evidence variables received. This is the first and arguably the most useful inference challenge that we inspect.

The actual quantity that we want to compute is

$$P(X_t|e_1,e_2,...,e_t) = P(X_t|e_{1:t})$$

that is, the probability distribution of the hidden state variable, given all previously observed evidence variables. The notation $e_{1:t}$ denotes the joint distribution of $e_1,e_2,...,e_t$. We are interested in computing this value, rather than the unconditional state distribution, $P(X_t)$, because we cannot observe the state directly, but we can observe the evidence variables. The conditional distribution $P(X_t|e_{1:t})$ is frequently referred to as the *belief state* and the process of calculation of this distribution is frequently referred to as *state estimation* or *filtering* [77], [96], [45]. The *forward algorithm* can be used to calculate the value of $P(X_t|e_{1:t})$ [77, p. 572]. It is a recursive algorithm, and takes advantage of the fact that the underlying process is Markovian. The forward algorithm for HMMs is shown in Algorithm 1 and is based on the algorithm outlined in [96, p. 27]. The derivation is useful to follow as an exercise, so we present it in the subsequent sub-section.

HMM Filtering Algorithm Derivation

Two well-known probability identities are used in the derivation:

$$(a) \quad p(A|B,C) = \frac{p(B|A,C)p(A|C)}{p(B|C)} \quad \text{and} \quad (b) \quad p(A|B) = \int_C P(A|B,C)P(C|B)dC$$

The derivation is as follows:

1. $p(X_t|e_{1:t}) = p(X_t|e_{1:t-1},e_t)$
2. applying (a) and letting $\eta = \frac{1}{p(e_t|e_{1:t-1})} \quad = \quad \eta p(e_t|e_{1:t-1},X_t)p(X_t|e_{1:t-1})$
3. by the Markov property $= \quad \eta p(e_t|X_t)p(X_t|e_{1:t-1})$
4. applying (b) $= \quad \eta p(e_t|X_t)\int_{X_{t-1}} p(X_t|e_{1:t-1},X_{t-1})p(X_{t-1}|e_{1:t-1})dX_{t-1}$

5. by the Markov property $= \quad \eta p(e_t|x_t) \int_{X_{t-1}} p(X_t|X_{t-1})p(X_{t-1}|e_{1:t-1})dX_{t-1}$

Note that η is a normalizing constant that ensures that the probability distribution integrates to 1, and that the probabilities that need to be calculated can be done so from the parameters specified in the HMM; namely $p(e_t|X_t)$, specified by the sensor model, and $p(X_t|x_{t-1})$, specified by the transition model.

Algorithm 1 Forward Algorithm for HMMs

```
Input:
    $P(x_{t-1}|e_{1:t-1}) = bel(x_{t-1})$ :   The belief distribution as far as the previous time-step
    $e_t$ :   The most recent observation
    $hmm$ :   A Hidden Markov Model specifying the transition and observation probabilities,
    $p(X_t|x_{t-1})$ and $p(E_t|x_t)$
Output:
    $P(X_t|e_{1:t}) = bel(X_t)$

1: for all $x_t$ do
2:     $\overline{bel}(x_t) = \int p(x_t|x_{t-1})p(x_{t-1}|e_{1:t-1})dx_{t-1}$
3:     $bel(x_t) = p(e_t|x_t)\overline{bel}(x_t)$
4: end for
5: $\eta = 1/\int_{x_t} bel(x_t)dx_t$
6: for all $x_t$ do: do
7:     $bel(x_t) = \eta bel(x_t)$
8: end for
9: return $bel(X_t)$
```

Algorithm 1 uses integrals to account for the fact that the distribution of the hidden state, X_t, may be continuous. The work done in this thesis only uses discrete distributions, and so summations are used instead of integrals for the subsequent discussion. A consequence of using discrete distributions is that it is possible to formulate the forward algorithm using vector and matrix notation. As outlined in Section 2.4.1, the transition model $T = p(X_t|X_{t-1})$ for a HMM can be written down in matrix form, as can the observation model $O_t = p(E_t|X_T)$. This allows for a highly compact notation: denoting $p(x_t|e_{1:t})$ as $f_{1:t}$, we can write $f_{1:t} = \eta O_t T^T f_{1:t-1}$, where $f_{1:t}$ contains the vector of values of $p(x_t|e_{1:t})$ for every possible x_t [77, p. 579]. Note that the observation model, O_t is time dependent, as opposed to the transition model, which is assumed to be stationary.

Some insight can be gained from studying Algorithm 1. There are three main steps to this algorithm, and the intuition behind them is explained based on explanations in [77], [96] and [62]:

1. The first step is often referred to as the prediction step:

$$\overline{bel}(x_t) = \sum_{x_{t-1}} p(x_t|x_{t-1})p(x_{t-1}|e_{1:t-1})dx_{t-1}$$

This step carries out the first part of the calculation of the belief value for a given state x_t, by marginalizing over all possible hidden states (x_{t-1}) that could have preceded the current one (x_t). This reflects intuition: the probability of being in state x_t should depend on the probability of transitioning from all possible previous states to x_t. Think "*the probability that I am in location x is the sum of probabilities of starting at all other possible locations and subsequently ending up in location x, weighted against my belief that I began in each other possible location*".

$\overline{bel}(x_t)$ effectively projects our current belief to the next time step by using the transition probabilities specified in the HMM.

2. The second step is often referred to as the correction step, or the measurement update step:

$$bel(x_t) = p(e_t|x_t)\overline{bel}(x_t)$$

This step takes the projected belief in step 1 and multiplies it by the probability of observing the data that we did in the given hidden state, x_t. This can be thought of as a correction because the use of the measurement data narrows the probabilities of the possible states that the system could have transitioned to.

3. The final step is to ensure that the distribution sums to 1. This is done by multiplying by a normalizing constant, η.

A key strength of this algorithm is the fact it allows updates to be performed online. The reason for this is that the information-state vector, $p(x_t|e_{1:t})$ is a sufficient statistic for the past history of observations. For a full proof of this property, the reader is referred to Appendix A of [84], which also provides insight into the update rule itself. In relation to the update rule, note that an initial distribution of the estimated state must be provided to this algorithm before the first update can be performed. The complexity of the forward algorithm is $\theta(nm^2)$, where m is the size of the joint distribution of the hidden variables, x, and n is the number of time-steps that have occurred. This is clear to see when viewing the update as the matrix-vector multiplication $f_{1:t} = \eta O_t T^T f_{1:t-1}$, as the size of T is m^2.

Evidence Likelihood Algorithm Using a HMM

The second useful quantity that we would like to calculate is the probability of observing the evidence that we did, often referred to the likelihood function of the data, as identified in Section 2.4.1. In simpler terms, this is the calculation of the probability of observing the data, given a fixed parameterised model of the data (the HMM). This value is useful as it can reveal insights into whether one model is more likely than another model, with applications including **M**aximum **a P**osteriori (MAP) parameter estimates and hypothesis testing. It is used in this thesis in Section 5.2.7. We would like to compute the value of

$$P(e_1, e_2, ..., e_t) = P(e_{1:t})$$

given the fixed set of parameters provided by the HMM. A brief outline of the derivation is shown:

HMM Evidence Likelihood Algorithm Derivation

Analogous to the filtering algorithm derivation, two well-known probability identities are used in the derivation:

$$(a) \quad p(A,B,C) = p(A|B,C)p(B,C) \quad \text{and} \quad (b) \quad p(A,B) = \sum_C p(A|B,C)p(B,C)$$

The derivation is as follows:

1. $p(e_{1:t}) = \sum_{x_t} p(e_{1:t}, x_t) \quad = \quad \sum_{x_t} p(e_{1:t-1}, e_t, x_t)$
2. applying (a) $\quad = \quad \sum_{x_t} p(e_t|e_{1:t-1}, x_t)p(x_t, e_{1:t-1})$
3. by the Markov property $\quad = \quad \sum_{x_t} p(e_t|x_t)p(x_t, e_{1:t-1})$
4. applying (b) $\quad = \quad \sum_{x_t} p(e_t|x_t) \sum_{x_{t-1}} p(x_t|e_{1:t-1}, x_{t-1})p(x_{t-1}, e_{1:t-1})$
5. by the Markov property $\quad = \quad \sum_{x_t} p(e_t|x_t) \sum_{x_{t-1}} p(x_t|x_{t-1})p(x_{t-1}, e_{1:t-1})$

The algorithm for calculating likelihoods uses the forward algorithm, α, with the replacement of $p(x_{t-1}|e_{1:t-1})$ with $p(x_{t-1}, e_{1:t-1})$ and one final summation. This means that it is possible to use the forward algorithm to calculate likelihoods: $p(e_{1:t}) = \sum_{i=1}^{n} \alpha(x_i, e_i)$, where $\alpha(x, e)$ is the result of applying the forward algorithm to the sequence of evidence variables $(e_1, ..., e_t)$.

Filtering Algorithm for DBNs

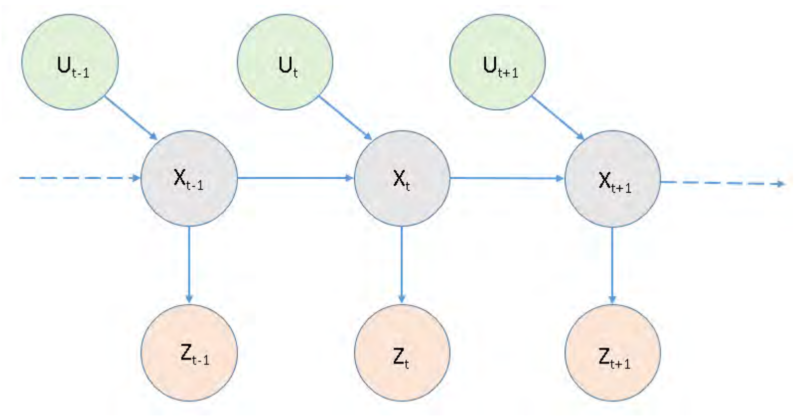


Fig. 2.5 A DBN with Hidden State variables (grey), Observation variables (orange) and Control variables (green).

The forward algorithm for state estimation was presented in Section 2.4.3. A slightly modified version of this algorithm is now described, which is very commonly used in stochastic systems that can be influenced by actions that an agent may take. Such systems have *control variables* as well as hidden state variables and observation variables. These control variables (variables which represent the actions that an agent may perform) have an influence on the transition probabilities between states. The graphical model in Figure 2.5 describes the basic case. The arrows describe the conditional independence assumptions: at time t, the hidden state (x_t) depends on the previous state (x_{t-1}), and the previous control action taken (u_t). As with the HMM, the observation e_t is conditionally dependent on the state x_t. This causes a slight change in the filtering algorithm: on line 2 of algorithm 1: $\overline{bel}(x_t) = \sum_{x_{t-1}} p(x_t|x_{t-1})p(x_{t-1}|e_{1:t-1})$ is replaced with $\overline{bel}(x_t) = \sum_{x_{t-1}} p(x_t|u_t, x_{t-1})p(x_{t-1}|e_{1:t-1}, u_{1:t-1})$, reflecting that transition probabilities now depend on the most recent control action taken, as well as the previous state.

It is worth noting that there are many filtering algorithms that deal with data that come from certain distributions and satisfy constraints related to their transition and observation models, for example the Kalman Filter [96, p. 43].

Approximate State Estimation

The state estimation algorithms mentioned so far maintain the exact values of the distribution $p(x_t|e_{1:t}, u_{1:t})$, but we will quickly mention an approximate method that is also frequently used. The computational complexity of the filtering algorithms is determined by how well the joint distribution of hidden state variables, observation variables and control action variables can be factored, but in the worst case the complexity is given by the case where the joint distribution of the hidden state variables cannot be factored at all. The forward algorithm, described in Algorithm 1 for a HMM performs the update from each time state in O(n^2), where n is the dimension of the hidden state variable [88]. In some cases, this can become intractable if the state space is large enough. In this case, approximate techniques are used.

An example is the *particle filter*. Details of the particle filter can be found in [96, p. 96] and [45, p. 665].

2.4.4 Sequential Statistical Hypothesis Testing

A statistical test is a mechanism for making quantitative decisions about a process, by determining how well the data stand in agreement with given predicted probabilities [22]. Statistical tests are usually used to draw conclusions about a *statistical hypothesis*, which is a testable hypothesis based on observations of a process that is modelled using random variables. There are many texts that outline the statistical tests that can be used to make a decision between accepting and rejecting the null hypothesis, H_0, for example [37]. The usual context in which this occurs is one in which the data has been collected in advance and the sample size is fixed and known. The standard procedure for testing a simple hypothesis involves using a Uniformly Most Powerful (UMP) Test for a fixed value of the probability of making a Type I error, α [37, p. 253]. The subsequent sections discuss how this can be extended to a sequential paradigm, where the sample size is not fixed.

There exists a branch of statistical hypothesis testing called *sequential hypothesis testing*, which is used when the sample size is not fixed in advance [37, p. 375]. Given a hypothesis to test, H_0, this means the decision process goes beyond deciding whether to accept or reject the null hypothesis, but to either

1. Accept the hypothesis being tested, H_0.
2. Reject the hypothesis being tested, H_0
3. Continue the experiment by making a further observation.

It is clear that samples are gathered as long as 1). or 2). above are not chosen, which intuitively corresponds to the notion of making an informed decision, where it is desirable to ensure that enough data has been gathered to draw a meaningful conclusion. In the sequential testing paradigm, two kinds of error may be committed, as with the non-sequential case: we may reject the null hypothesis when it is true (commit a Type I error) or we may accept the null hypothesis when some alternative hypothesis is true (commit a Type II error).

The Sequential Probability Ratio Test (SPRT), which was devised by Wald, proposes a statistical test for simple hypotheses with specified fixed values, which ensures that the probability of Type I and Type II errors do not exceed α and β respectively [103]. Wald and Wolfowitz proved that the SPRT is optimal, in the sense that of all tests with the same power, the SPRT requires on average the fewest number of observations to reach a decision [104]. A strong advantage of the test is that it can be carried out without determining any probability distributions whatsoever [103]. The SPRT can be thought of as a stopping rule for sampling in a stochastic process. For the sake of brevity, we omit the full derivation of the rule and the proof of optimality, but instead refer the reader to [103], [105] and [104]. The SPRT can be carried out following the procedure shown in Algorithm 2.

Algorithm 2 The Sequential Probability Ratio Test Algorithm

```
Input:
    $\alpha$:    The upper limit in probability for making a Type I error.
    $\beta$:    The upper limit in probability for making a Type II error.
    $H_0$:    The null hypothesis.
    $H_1$:    The alternative hypothesis.
    $(x_1,...,x_m)$:    The m observations made so far.
Output:
    A decision to accept $H_0$, accept $H_1$ or make another observation.

1: Calculate $p_{0m} = p_{0m}(x_1,...,x_m)$, the probability of observing the data under the assumption $H_0$ is true.
2: Calculate $p_{1m} = p_{1m}(x_1,...,x_m)$, the probability of observing the data under the assumption $H_1$ is true.
3: Calculate the values $A = \frac{1-\beta}{\alpha}$ and $B = \frac{\beta}{1-\alpha}$
4: Accept $H_1$ if $\frac{p_{1m}}{p_0m} \geq A$
5: Accept $H_0$ if $\frac{p_{1m}}{p_0m} \leq B$
6: Make an additional observation if $B < \frac{p_{1m}}{p_0m} < A$
```

The main idea, as is the case with simple non-sequential hypothesis tests, is based on the idea that if the likelihood ratio ($\frac{p_{1m}}{p_{0m}}$) lies in some *critical region*, C, then we may reject the null hypothesis [37]. The SPRT extends this by partitioning the m-dimensional sample space M_m into three mutually exclusive regions, R_m^0, R_m^1 and R_m. After the i_{th} observation (x_i) has been drawn, H_0 is accepted if $(x_1,...,x_i)$ lies in R_m^0, H_1 will be accepted if $(x_1,...,x_i)$ lies in R_m^1 or a $i+1_{th}$ observation will be drawn if $(x_1,...,x_i)$ lies in R_m [103]. The process of how to choose R_m^0, R_m^1 and R_m is outlined in Section 3 of [103], which goes beyond the scope of this thesis. The derivation essentially shows that $(x_1,...,x_i) \in R_m^0$ is equivalent to $\frac{p_{1m}}{p_0m} \leq B$, $(x_1,...,x_i) \in R_m^1$ is equivalent to $\frac{p_{1m}}{p_0m} \geq A$, and $(x_1,...,x_i) \in R_m$ is equivalent to $B < \frac{p_{1m}}{p_0m} < A$.

Summary

The SPRT is a sequential hypothesis test that formulates a stopping rule in order to draw conclusions based on statistical data. It has frequently been in quality control studies, where samples can be expensive to gather, but it can be applied in other scenarios where sampling can be expensive [64]. We use it in Section 5.2.7

2.5 High Fidelity Simulation Environments Background Knowledge

In this section, we discuss existing literature and common approaches related to designing and implementing high-fidelity simulation environments. This provides a backdrop for Chapter 3, which describes the work we did to implement our own custom high-fidelity simulation environment.

2.5.1 Key Aspects of Simulation Environments

We begin by discussing some key aspects of simulation environments that serve as a starting point on which more advanced techniques build upon. According to Shannon, simulation can be defined as *"the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behavior of the system and/or evaluating various strategies for the operation of the system"* [82]. Simulations have been used extensively to model complex systems since the invention of the modern computer, gaining traction since the proposal of the Markov Chain Monte Carlo method by Stansislaw Ulam and John Von Neumann in the late 1940s [72]. Simulations written with software are created in order to gain insight into the system's dynamics and to evaluate results of using a method intended for real-world use. This is usually because proposed methods may be too time-consuming or expensive to test in the real world. Software simulations are being used increasingly for a wide variety of tasks, from planning new roads to alleviate traffic [66] to developing self-driving vehicles [25].

Designing simulations allows the developer of an agent-based system to abstract away details of the real-world conditions that the agents will act in and focus on the salient aspects that the agents are concerned with, in order to prototype, train, test, analyse and validate. General advantages of simulations are listed in [82], with the most notable being:

- *"It can be used to explore operating procedures, decision rules, organizational structures, etc. without disrupting the ongoing operations."*
- *"Simulation allows us to control time. Thus we can operate the system for several months or years of experience in a matter of seconds allowing us to quickly look at long time horizons or we can slow down phenomena for study."*
- *"It allows us to gain insights into how a modelled system actually works and understanding of which variables are most important to performance."*

- *"Simulation's great strength is its ability to let us experiment with new and unfamiliar situations and to answer "what if" questions."*

In relation to designing intelligent agents, often the trade-off between exploration and exploitation is referenced. Exploration or "information gathering" [77] can be considered as the agent posing a "what if" question in order to learn the consequences of performing action, and exploitation can be considered as an agent using gathered or previous knowledge to choose actions to take that maximize its performance measure, as outlined in Section 2.2.1. Since simulation environments are highly suited to answering "what if" questions, which has motivated the use of using simulations to design systems of autonomous agents in the literature, for example the simulation environment used in the Multi-Agent Programming Contest.[4]

2.5.2 Simulation Environments Design Using Game Engines

Since computer games are effectively simulations, the technologies used to create them have also been used to create simulations for a more serious purpose, often referred to in the literature as *serious games*. Serious games usually refer to games that have been specifically designed to provide training to professionals working in an industry where extensive training is necessary but difficult, expensive or dangerous to provide [89]. An overview and taxonomy of serious games is provided by Laamarti et al. [51]. Early motivation for using games engine in scientific research is outlined in a 2002 journal article by Lewis and Jacobson, summarized by the sentence "*By elevating the problems of updating and synchronization to a primary concern, game engines provide superior platforms for rendering multiple views and coordinating real and simulated scenes as well as supporting multiuser interaction*" [57]. More recently, games have been used to train learning agents, defined in Section 2.2.1. The reasons that this is a recent phenomenon are as follows:

- As discussed in Section 2.2.1, training a learning agent usually involves defining a mapping from environment states or percept sequences to actions, in order to maximize the performance measure. Until a few years ago, state-space representations of many environments were of such a high dimension that it was prohibitive to find such a mapping. Seminal works on Neural Networks [54], [50] have allowed high-dimensional states to be compressed to lower-dimensional ones while retaining key information related to the states, simplifying the process of mapping states to actions. This has led to the development of algorithms that can utilise high-dimensional states to choose optimal actions to great effect, such as Deep Q-Learning [59].

[4]https://multiagentcontest.org/

- Writing complex software simulations rendering high-fidelity sensor outputs (e.g. images) in order to train an agent requires a lot of highly-optimised code, which takes a significant amount of effort to write. General Purpose Graphics Processing Units (GPGPUs) have significantly decreased the time required to carry out certain tasks in parallel, which has made the problem of calculating sensor outputs in a reasonable time tractable. For example, the rendering time for photo-realistic images has significantly sped up relative to the rendering time using a CPU only [78].

- Insufficient amounts of data could be generated in a reasonable amount of time for the learning agent to perform sufficiently well. Quicker CPU speeds have increased the rates at which agents can learn significantly over the last number of years, in agreement with the predictions of Moore's law [58]. Game engines tend to have highly optimised code and given the possibility to generate copious amounts of data in a small amount of time, they have significantly expedited the rate at which learning agents can be trained [41], [79].

There have been some highly notable simulation environments written with various games engines for non-gaming purposes. One of the first mature simulations written for a serious purpose using technologies usually applied to game development is the Gazebo simulator [44]. Gazebo makes use of physics engines and graphics rendering engines that are commonly used to develop games. The use of Gazebo has been reported to reduce the training time for some simple benchmark robotics tasks by as much as 33%, while retaining the same level of performance [98].

There have also been a notable number of simulators designed to aid the development of autonomous cars, trucks, RAVs and UUVs, with documentation on each commonly citing difficulties in changing configurations for physical vehicles, time taken to generate data and dangerous edge cases as motivation [25], [107], [81], [9]. Dosovitskiy et al. designed the CARLA open-source simulator [25] for autonomous driving research using Unreal Engine 4 (UE4) to provide a lot of basic functionality. They cite "*state-of-the-art rendering quality, realistic physics, basic NPC logic, and an ecosystem of interoperable plugins*" as key motivations to use the engine. AirSim [81] is also built over UE4, with their motivations including

- "*The amount of training data needed to learn useful behaviors is often prohibitively high*".

- "*Autonomous vehicles are often unsafe and expensive to operate during the training phase*".

- "*Simulated perception, environments and actuators are often simplistic and lack the richness or diversity of the real world*".
- It is "*important to develop more accurate models of system dynamics so that simulated behavior closely mimics the real-world*".

Finally, the designers of Sim4CV [60], which is also built over UE4, list the following motivations for the work done in developing a simulation environment for RAV tracking:

- "*The combination of the simulator with an extensive aerial benchmark provides a more comprehensive evaluation toolbox for modern state-of-the-art trackers and opens new avenues for experimentation and analysis.*"
- "*Although simulation is popularly used in machine learning and animation and motion planning, the use of synthetically generated video or simulation for tracker evaluation is a new field to explore.*"
- "*The Unreal Engine 4 (UE4) has recently become fully open-source and it seems very promising for simulated visual tracking due in part to its high-quality rendering engine and realistic physics library.*"
- "*Our proposed RAV simulator along with novel evaluation methods enables tracker testing in real-world scenarios with live feedback before deployment*"

2.5.3 Software Simulations Related to Hazardous Scenes

We now move the discussion to the domain of hazardous scenario management. There are a number of existing software simulations that were designed to aid the management of hazardous scenarios. In this context, the scenarios are taken to be of a very serious nature, such that there is an immediate and unquantified risk to human life in the immediate vicinity.

Jacoff et al. provide a review of the design process of physical search and rescue environments in an urban setting with the aim of aiding the design of robotic navigation systems [39]. They discuss the recommended physical configuration of simulation environments and offer technical details on how the simulation could be run, including performance metrics and details of how the robot should interact with the environment. The details are relatively high-level, but serve as a good starting point on which to develop more complex and realistic scenarios.

USARSim [13] was proposed in 2007 as a general-purpose simulation environment for Urban Search-and-Rescue (USAR) scenarios. It was built using Unreal Engine 2 and was designed to specifically facilitate the design and testing phase of autonomous robots that would be used to help deal with dangerous scenarios. The simulator was designed to be used as a companion to the National Institute of Standards' (NIST) Reference Test Facility for Autonomous Mobile Robots for Urban Search and Rescue. They describe the decision to use a game engine as their simulation platform as one that considers the strong advantages of offloading the most technical and difficult aspects of simulation to the game engine, which provides superior visual rendering and physical modelling. This allows the user to concentrate their efforts on developing the robotics-specific tasks related to the management of the scenario.

VIVID: Virtual Environment for Visual Deep Learning [52] is a set of integrated tools that is capable of generating photo-realistic quality images and sensor readings from simulated vehicles. It was designed using UE4 and ships with a number of ready-made environments. The author envisages its main applications as lying in the domains of "*deep reinforcement learning, semantic segmentation, object recognition, action recognition and video event detection.*" A number of the ready-made environments involve a hazardous component which could benefit from autonomous scene surveying and management, such as the ruined school and train station scenarios.

Chapter 3

High-Fidelity Simulation Environment

As highlighted in Chapter 1, one of the main goals of the ROCSAFE project is to avoid putting forensic investigators in potentially hazardous situations by using autonomous robots in their stead [3]. In order to design and validate systems based on this approach, it was desirable to have a low-consequence test environment. Outlined in Section 2.5.3, to the best of our knowledge, there are not many simulations in existence that specifically address these kind of hazardous scenarios. To address this, we developed a high-fidelity simulation environment, which preserves the critical aspects of CBRNe incidents without presenting any risk of exposure to the dangerous elements of such scenes in the real world. We did this work to explore the answer to the second research question stated in Section 1.2:
"*Can a high-fidelity simulation environment be created, which can generate realistic salient data to support the process of prototyping AI systems designed to aid the management of real-world hazardous scenarios?* "

The following chapter discusses work based on our accepted publications: [86], [87], [85].

3.1 Virtual Environment Design

3.1.1 Evaluation Criteria

In order to design the virtual simulation environment, it was important to first identify the criteria that would determine its usefulness. We began by identifying key physical phenomena that would need to be present, or integrated in future iterations. This was based on an operational scenario that was outlined in the ROCSAFE Deliverable 2.4: Detailed Use Cases[1]. We modelled Use Case 1, with a brief description as follows:

[1]https://www.nuigalway.ie/rocsafe/research/

"*The scene is set in a rural location, with some forest, a twin track rail line, a small town 10Km away with a small road running near to the rail tracks with access to the rail tracks. The conditions are cool and dry, with a light breeze. A train has been derailed and heavy machinery has been used to damage the tracks. Radioactive material in containers has been exposed. It is intended to use autonomous aerial and ground vehicles to remotely survey and document the scene. They will then be used to localize forensic evidence, subsequently leading to remote evidence collection.*".

The result of modelling this scenario using UE4 is shown in Figures 3.3 and 3.4.

In relation to the research question stated above, we identified the most salient data used in hazardous scene management and how it can be realistically generated:

- ROCSAFE and other CBNR scene management reference manuals frequently state the value of a visual overview of the scene [90]. In order to generate realistic simulated images, the simulation will need to have high-resolution rendering capabilities.
- Further to the above point, the images will need to be generated from the perspective of a RAV in order to be of use when prototyping object detection modules and other AI systems related to ROCSAFE.
- The scene will need to have a configurable physical layout so that occlusion, shadows and other real-world aspects will be present in images. A non-uniform topography is also important for planning potential paths to sources of evidence for sampling.
- For the scene described above, a sensor which can simulate radiation readings is necessary, which will record noisy readings that are affected by real-world phenomena such as occlusion.
- All data must have timestamps and spatial information associated with it. The spatial information should use a real-world coordinate reference system, such as GPS.

We used the above points to discern some technical requirements that the simulation environment would need to achieve, which form our evaluation criteria:

- The ability to place arbitrary realistic virtual representations of physical objects in the scenario in various configurations.
- The ability to render high-quality images from the scenario from arbitrary locations at arbitrary resolutions, with common visual phenomena present, such as shadows and lens flare.

- There must be sufficient detail in the scene to introduce some noise to image processing problems. This means avoiding a highly-uniform configuration.
- Multiple heterogeneous simulated aerial vehicles, with a high-level API for sensing and navigation for each vehicle. The API should not be platform-specific so that different types of vehicle may be considered.
- Simulated sensor readings from the aerial vehicles, including position, velocity, altitude, that depend on the state of the aerial vehicle in the environment. For example, if the aerial vehicle is close to a source of radiation, the simulated sensor reading should be high.
- Simulation software should have a permissive licence and should permit publications that include details of the software.
- The ability to run the simulations at an increased speed without corrupting the fidelity of the sensor/actuator functionality.
- The ability to quickly change the configuration of the simulation.

3.1.2 Existing Tools and Software

In order to provide this functionality, it is clear that using existing tools would be desirable, as writing low-level code for tasks such as rendering would take a prohibitive amount of time. Game engines have been increasingly used for simulations of physical phenomena, with a growing interest in niche areas. Examples include generating high-fidelity training data for computer vision algorithms, [68], deep learning algorithms [31], automated crowd size estimation algorithms [55] and target tracking algorithms [60]. An overview of literature related to the use of game engines in creating simulation software is outlined in Section 2.5.2. The overview describes how most modern simulation software that use game engine components are mature in their capabilities to model and render physical scenarios, but not all provide good support for the use of robotic vehicles. This meant that an emphasis was placed on choosing software with some capability to implement high-fidelity simulated aerial vehicles as well as physics and graphics rendering. Standalone simulation tools were considered alongside tools designed to be run using a game engine. The simulation software whose potential use for designing a custom simulation environment for disaster scene management that were investigated are shown in Table 3.1, with more detailed overviews provided in [27]. UE4, combined with the Airsim plugin, was chosen to develop the simulation since the documentation for both suggested that it could meet all of the evaluation criteria outlined above. We give more details of the integration of AirSim in Section 3.2.

3.1.3 Virtual Simulation Environment Design Using UE4

Once an environment has been built using UE4, it can be labour-intensive to radically alter it [76, p. 454]. This suggests that care should be taken to ensure to plan the design process so that consistent and efficient results can be achieved.

Some good level design practices are noted in [76], although many are not applicable to the design of a serious simulation. Notable points from [76] are summarised:

- Pencil and paper sketches of the level's general layout can be a very good idea in order to avoid the perils of "*designing yourself into a corner*". As an example, this could manifest itself as underestimating the proximity between two high-level objects (e.g. a hall and a room), which may lead to a large-scale redesign further down the design process.
- During the first pass, do not worry overly about textures and geometry, focus on ensuring that the layout is realistic.
- Refine the architecture once a realistic layout has been identified
- Add basic gameplay once the physical layout has taken shape. This avoids a tight coupling between the two.
- The final step is to refine gameplay and aesthetics.

Using these concepts as a basis, the design process for the simulation environment proceeded roughly as follows, bearing in mind the evaluation criteria identified at the start of the chapter:

1. A sketch of the layout was drawn up based on an operational scenario outlined in Deliverable 2.4 of the ROCSAFE project.
2. The landscape was sculpted and painted using UE4 in-built landscaping tools.
3. We identified the assets that would be necessary to develop the environment to our specification. We acquired these assets from websites including CGTrader[2] and Google 3D warehouse[3].

[2]https://www.cgtrader.com
[3]https://www.3dwarehouse.sketchup.com

4. We imported into *Autodesk 3DS* in order to ensure that textures were of sufficient quality to facilitate the planned image processing on collected images.

5. We then imported into UE4 as static meshes and placed into the scene according to the sketch. In order to ensure that this process can scale, we ensured that static meshes could be replaced by simply importing a new mesh which retains the position of the original in the environment.

6. We qualitatively evaluated the rendered images to determine their suitability.

7. We archived the environment before integrating the Airsim aerial vehicle simulator plugin.

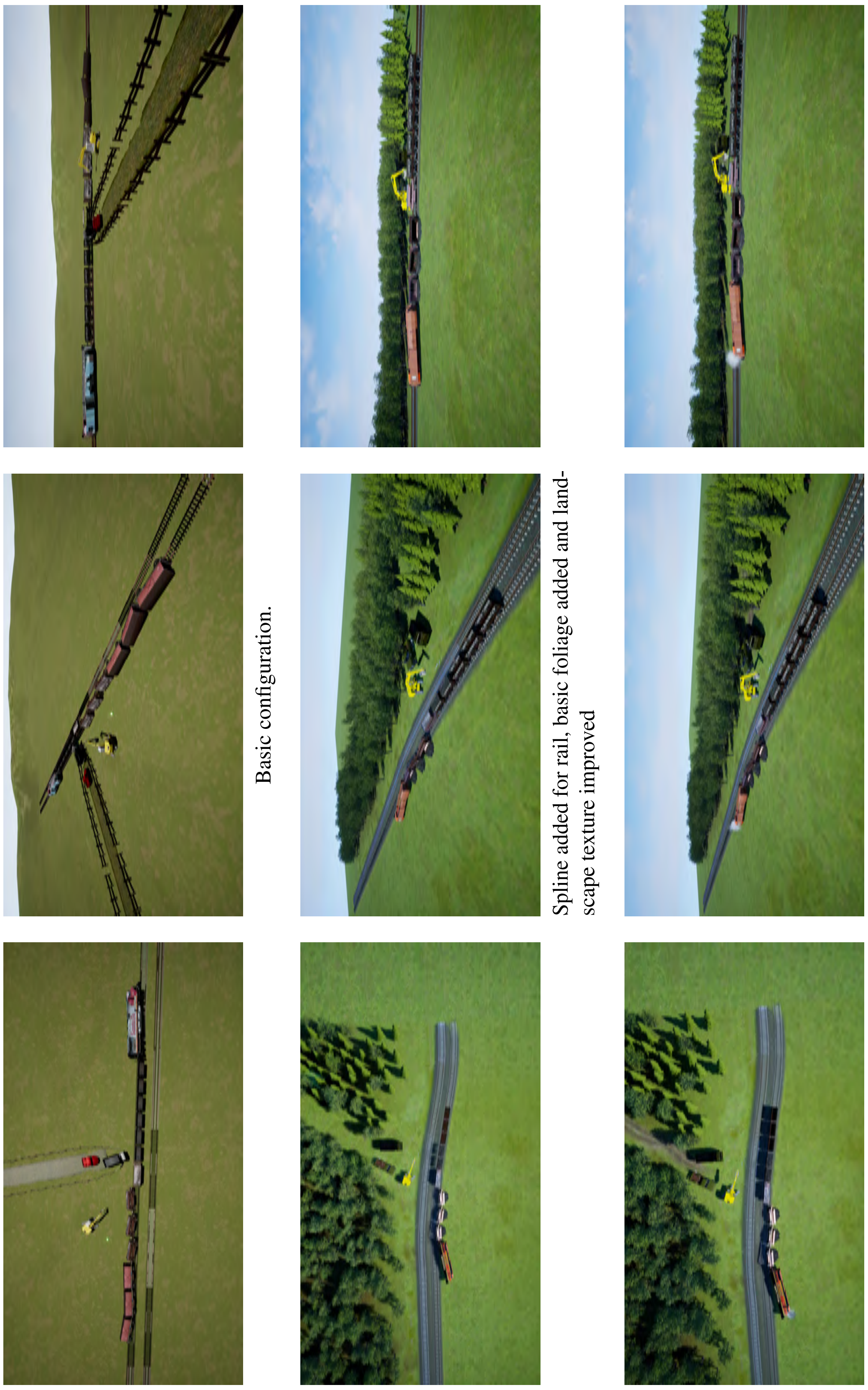
8. We then integrated the Airsim [81] with some modifications, which provided aerial vehicles simulation in UE4. This step is non-trivial and details of how this was done is outlined in Section 3.2.

3.1.4 Design Process without Dynamic Elements

This process was carried out iteratively and the results of the developed world excluding the dynamic elements are discussed here. Most changes took place directly within the UE4 Editor. The chronological development of the virtual world is shown in Figures 3.1 and 3.2. The Figures on the i_{th} row corresponds to the i_{th} distinct version of the physical layout of the virtual world. The first iteration has flaws that were improved throughout the development process. The major problems that we addressed were:

- Rendered images were of poor quality due to incorrect UV texture mappings on objects.
- Textures were of poor quality and highly uniform.
- The layout of the environment was highly uniform. Note that the rail tracks are perfectly straight.
- The scene was minimalist and lacking any convincing detail which would introduce noise to the AI algorithms being developed as part of ROCSAFE.

Although a human may recognise the semantics of the scene, it does not capture some key aspects identified in the evaluation criteria. In order to be of real value to the ROCSAFE project for tasks such as training the automation of localising an object using aerial vehicles, it was necessary to address these major problems. Improvements are shown in Figures 3.1 and 3.2 and the techniques used to make these improvements are discussed in Section 3.1.5.

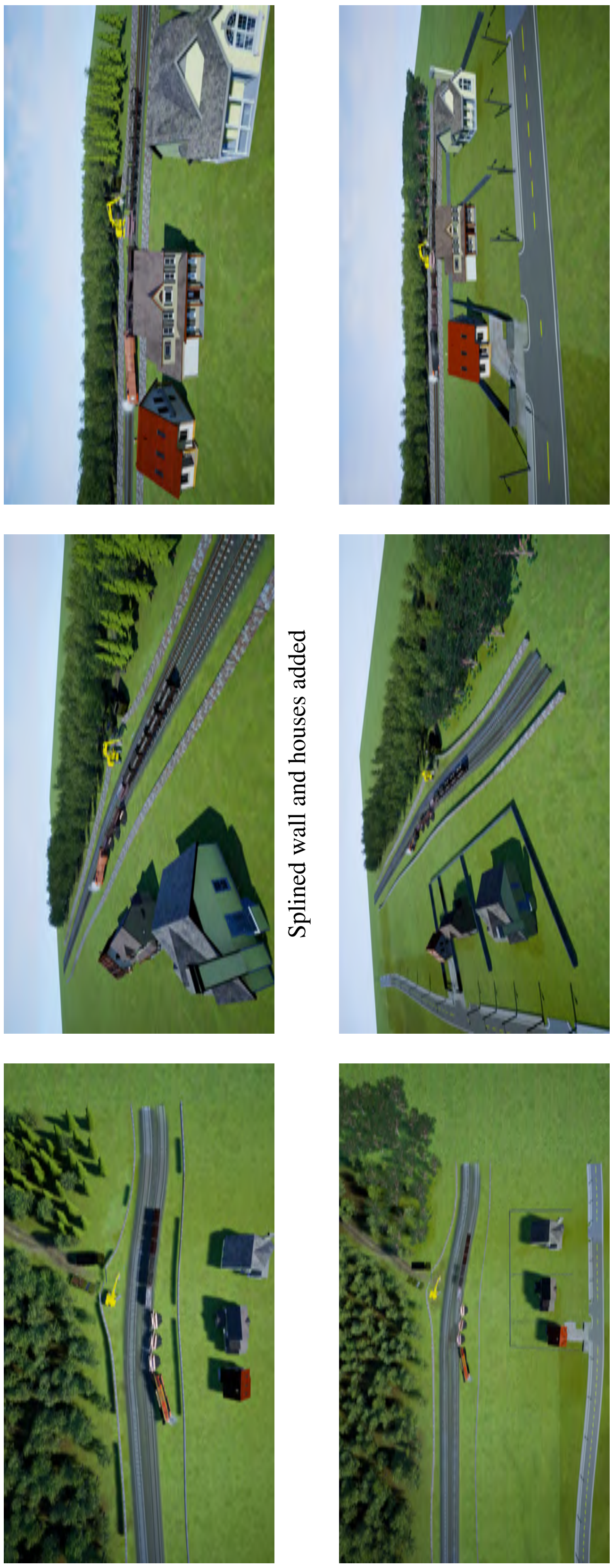


Basic configuration.

Spline added for rail, basic foliage added and landscape texture improved

Dirt track and smoke/steam effect added

Fig. 3.1 Evolution of the Simulation Environment. Each row depicts images from a subsequent iteration of the environment.



Splined wall and houses added

Road and extra walls added

Fig. 3.2 Evolution of the Simulation Environment. Each row depicts images from a subsequent iteration of the environment.

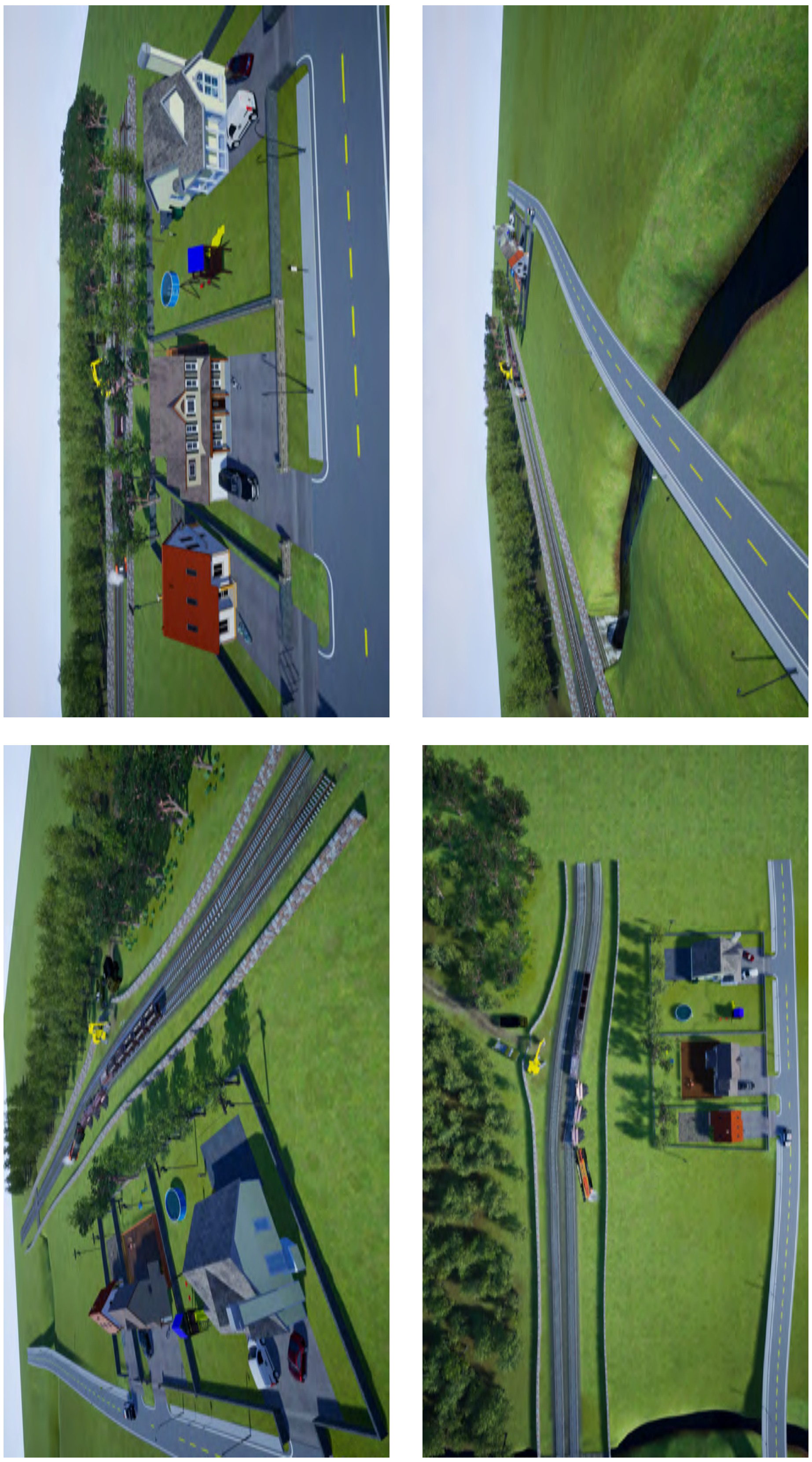


Fig. 3.3 Images From Final Version of Simulation Environment

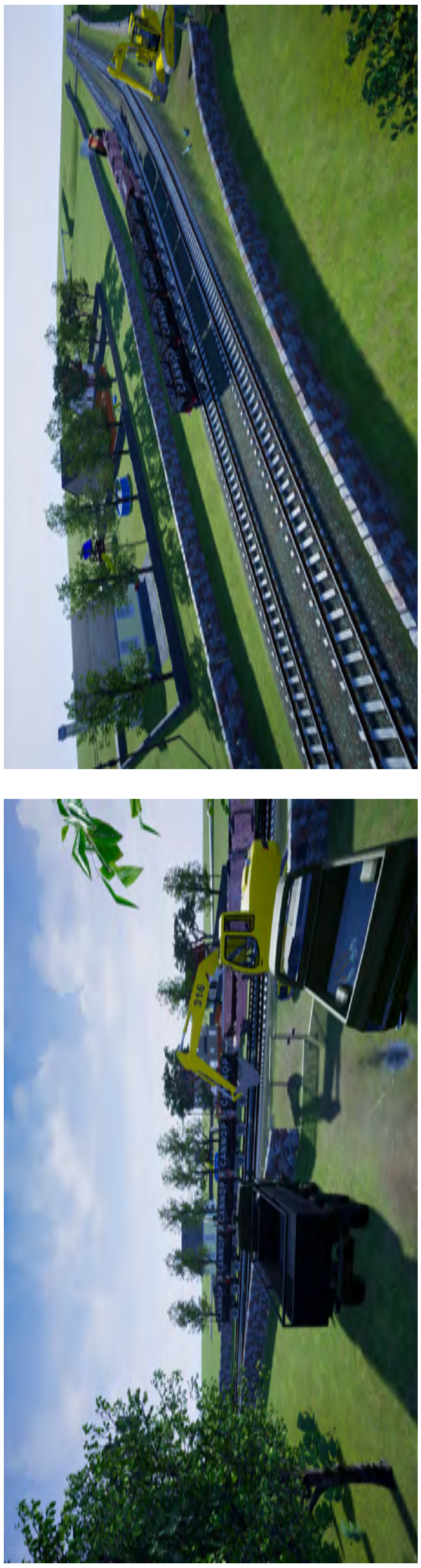


Fig. 3.4 Images From Final Version of Simulation Environment

3.1.5 Techniques used to Improve Realism

Blueprint Visual Scripting System

UE4 uses a visual scripting system to provide a lot of functionality, known as *Blueprints*. We provide a brief summary of Blueprints based on the UE4 documentation page[4] . Blueprints provide a node-based interface to create gameplay elements. To develop different aspects of a game, the system provides a visual approach to scripting, and many of the tools available in standard written scripting languages are available, such as typed variables, arrays, structs, loops, etc. Blueprints were used extensively to develop the simulation.

Materials

The details of creating materials can be found in the UE4 Documentation[5], which we use as a reference for the following paragraph. Materials are made up of a number of components in UE4, which specify aspects such as colour, opacity, roughness, specularity and emissive colours. In order to produce realistic materials, it is necessary to blend and layer different textures as well as identifying the correct parameters for surface normals and specularity, among the other features. The UE4 editor is equipped with tools for modifying materials to achieve a high-fidelity output, which we accessed through the Blueprints interface. The landscape in the first iteration

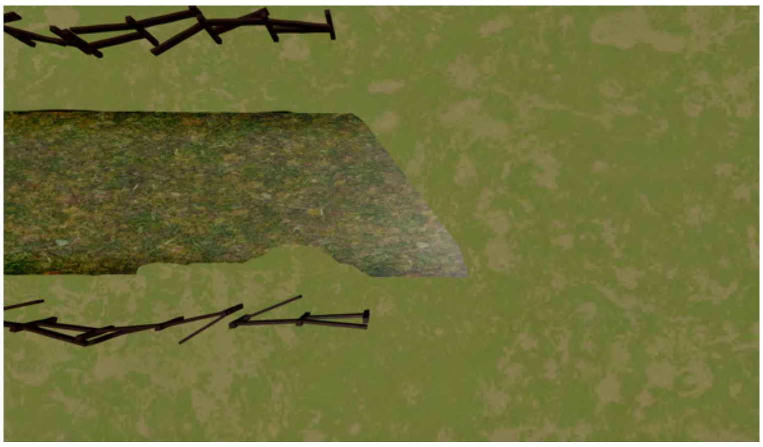


Fig. 3.5 Materials used in the first iteration

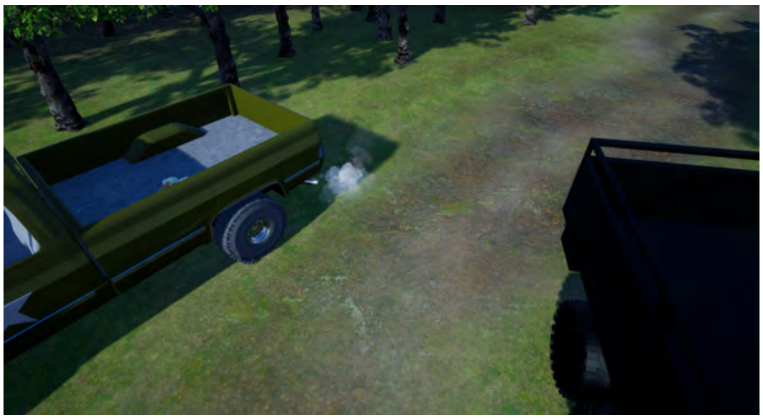
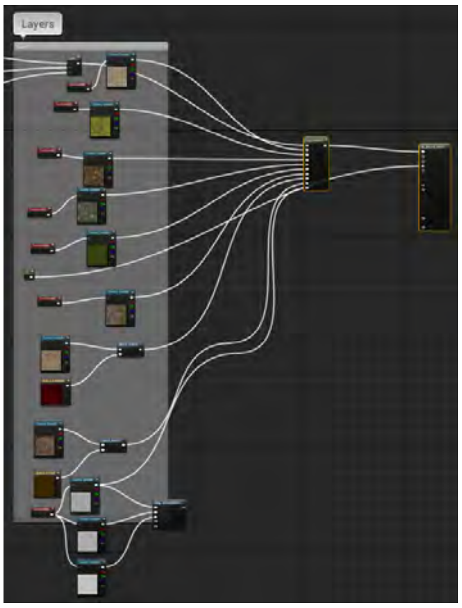
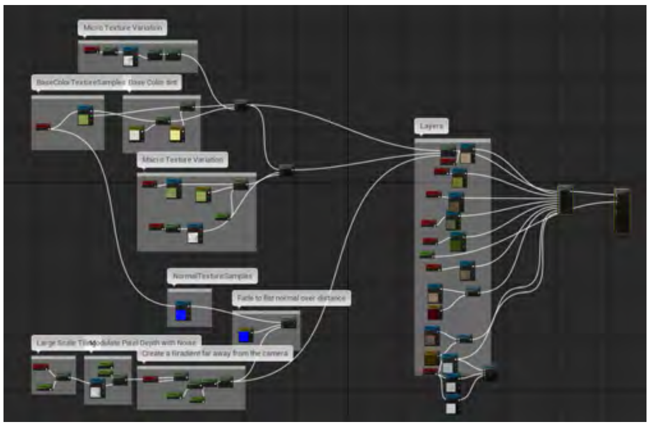


Fig. 3.6 Materials used in the final iteration

[4]https://docs.unrealengine.com/en-US/Engine/Blueprints/index.html
[5]https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/IntroductionToMaterials/index.html



(a) Layers blended into Landscape Layer Blend Node



(b) Blueprint used to create landscape material

Fig. 3.7 Blueprints used to create the landscape

of the virtual environment consisted of a single uniform texture, with none of the above parameters specified. The result is shown in Figure 3.5. Subsequent versions used multiple layers and blending in order to create a higher-fidelity material, with relevant parameters tuned. The final version is shown in Figure 3.6. Figure 3.7 shows how a Blueprint was used to create the landscape material, where a Landscape Layer Blend node is used to combine individual textures to create the material.

Splines

Uniformity tends to be rare in the real world; perfectly straight lines do not occur naturally often. For this reason UE4 offers tools to create splines, along which the terrain can be deformed. We base our explanation on the official documentation page[6]. Splines are typically used to model roads and paths, but the Landscape Spline system is very flexible and can be used to model many different phenomena. In the initial stages of the simulation environment, only a perfectly straight section of rail could be sourced for use in the environment, as shown in Figure 3.8. The spline tool allowed for much more realistic construction of the section of rail and accompanying wall, as shown in Figure 3.9. It was also used to create the road and the bridge.

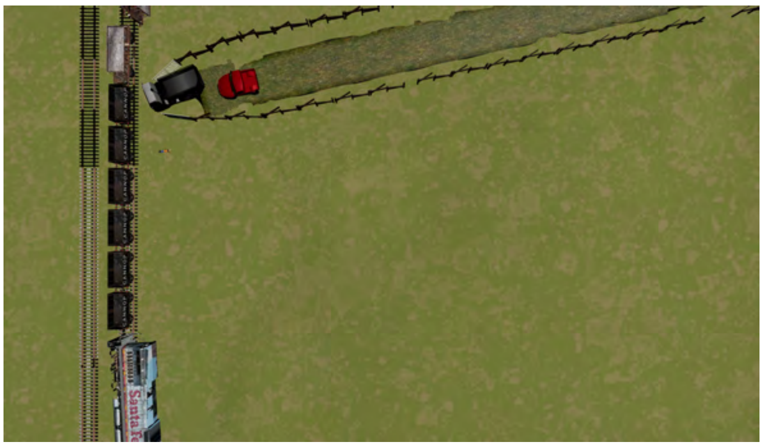


Fig. 3.8 Non-Splined rail section in first environment iteration

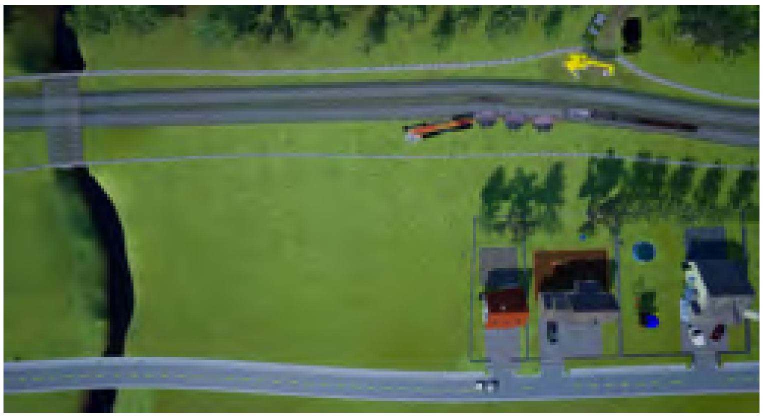


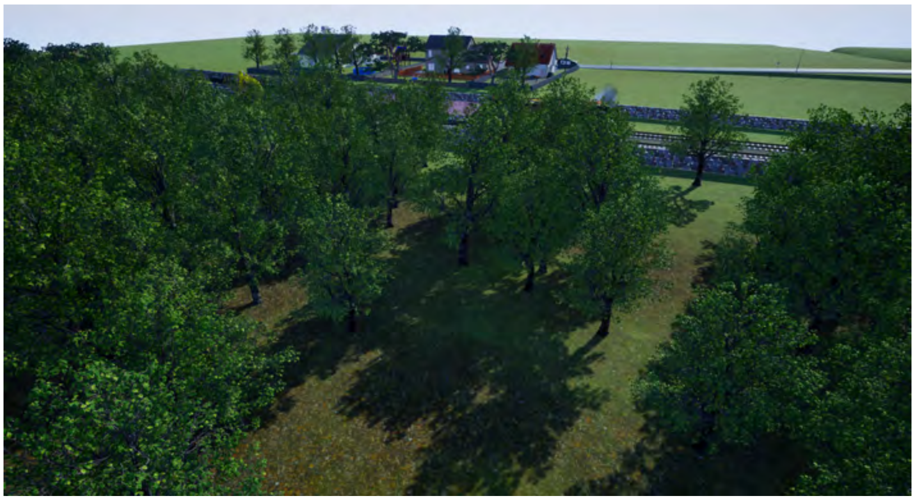
Fig. 3.9 Splined rail section in final environment iteration

Foliage

Similar to the argument made in relation to splines, it is rare to have uniformly configured foliage in the real world. In order to address this, there exist foliage generation and editing tools in UE4 editor. Version 4.18 of the editor onward contains the Procedural Foliage Tool[7], which is the most convenient way to add swathes of foliage to a scene. Since we were using other content dependent on UE 4.16, we opted to use the foliage painter tool, which allows

[6]https://docs.unrealengine.com/en-US/Engine/BlueprintSplines/index.html
[7]https://docs.unrealengine.com/en-US/Engine/OpenWorldTools/ProceduralFoliage/QuickStart/index.html

the user to effectively paint foliage directly onto a landscape. It allows the user to specify a number of parameters to achieve the required density, scaling and other relevant features. The results of applying foliage to the scene are visible in Figures 3.10a, 3.10b and 3.10c.


(a) Layers blended into Landscape Layer Blend Node

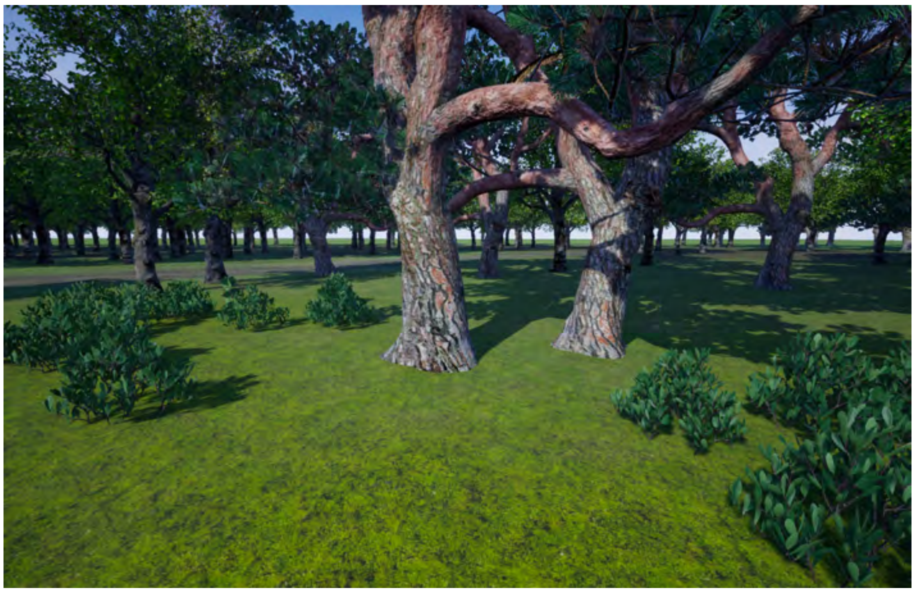
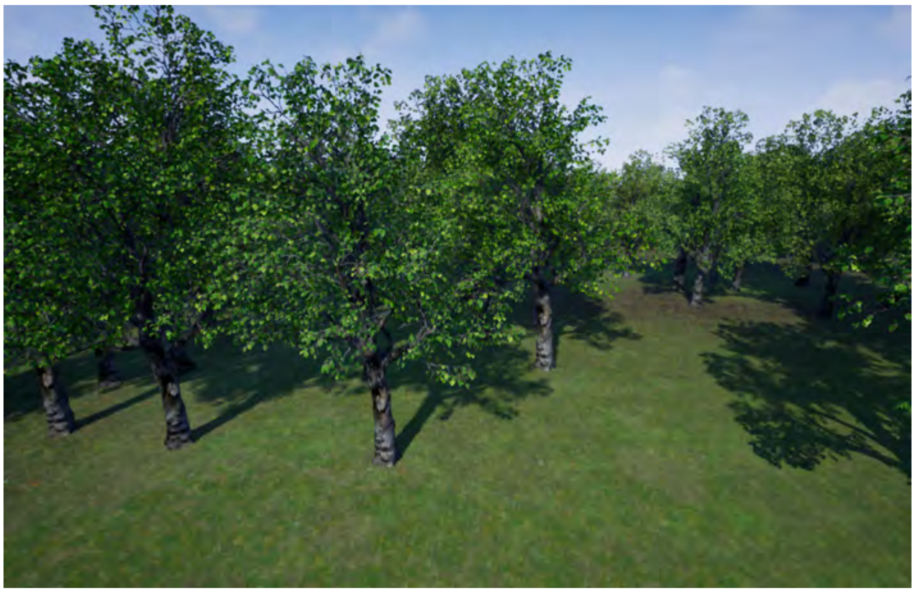

(b) Trees and shrubs generated with non-uniform configuration

Fig. 3.10 Examples of foliage in the simulation environment

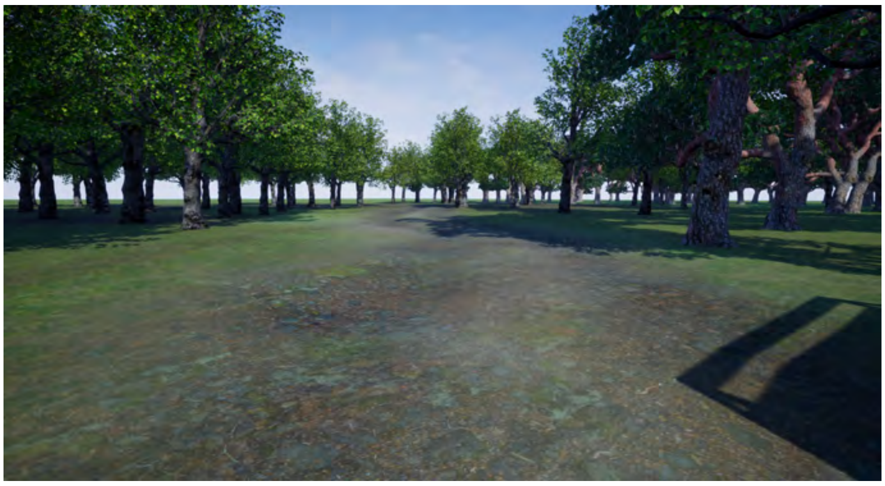


(c) Woodland area effect

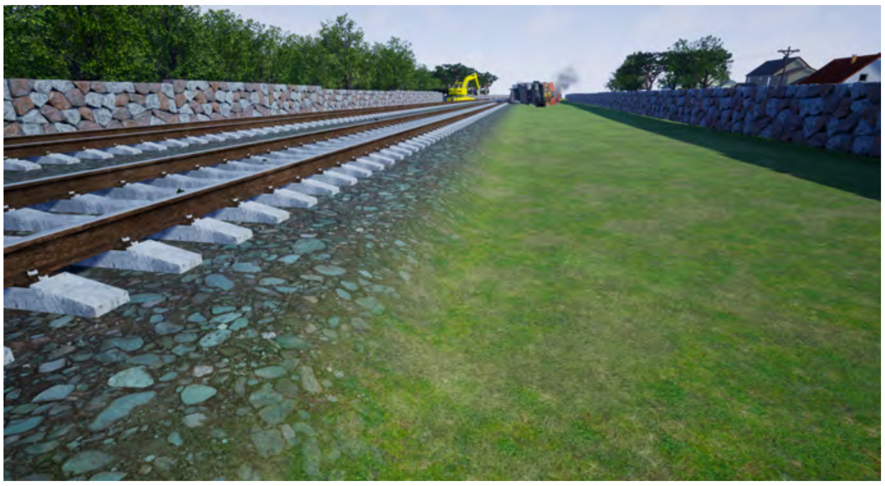
Fig. 3.10 Examples of foliage in the simulation environment

Landscape Editing

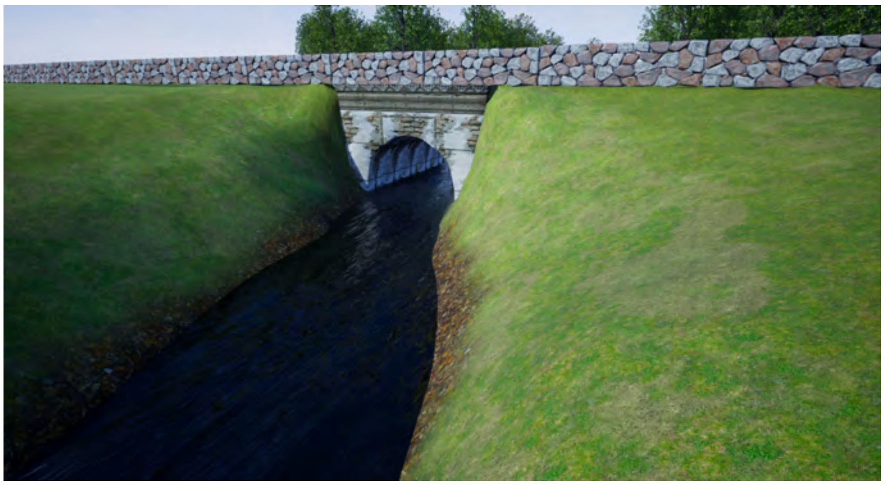
To realistically model the configuration of physical objects and their corresponding shadow and occlusion effects, it was necessary to model the terrain realistically. This was also desirable to allow the potential simulation of ground vehicles, so that difficulties that may be experienced in the real world, such as steep climbs or highly uneven surfaces, may be taken into account. UE4 provides a suite of landscaping tools that allows the user to create a highly variable landscape. The tools facilitate raising and flattening, smoothing, random noise and simulated erosion, as well as allowing for other more detailed modifications. These were used to create the railway embankment, the rutted path leading onto the rail tracks and the riverbank and riverbed, shown in Figure 3.11.



(d) Rutted Track



(e) Rail embankment



(f) Riverbank

Fig. 3.11 Results of using the UE4 landscape editing tools

Asset Sourcing

An *asset*[8] can be described as a piece of content for an Unreal Engine project, which has been serialized to a file. Assets can be re-used and modified in the UE4 editor, but are usually created using external software. UE4 uses assets that come in the Filmbox (.fbx) format, which is a proprietary file format owned by Autodesk. Searching for free assets was a labor-intensive process, which consisted of a number of steps:

1. First, identify possible candidates for a particular type of asset (e.g. a train) based on a search of asset stores that offer free assets. We mainly used CGTrader[9] , TurboSquid[10] , 3D Warehouse[11] and ShapeNet[12].

2. Once a potentially suitable asset had been identified based on its description and preview, it was downloaded in the Filmbox (fbx) format if possible. Otherwise, it was downloaded in whatever format was available.

3. The asset was opened in Autodesk and visually inspected for suitability. If the textures and geometry were not of a sufficient standard the processes was restarted.

4. If the asset was deemed suitable from the inspection in Autodesk, then it was exported in Filmbox format.

5. The asset was then imported into the UE4 editor. Problems often arose in scaling, incorrect texture mapping and one-sided materials applied to the wrong side of assets. These problems could sometimes be addressed; if not we had to restart the process.

[8]https://docs.unrealengine.com/en-US/Engine/Basics/AssetsAndPackages/index.html
[9]https://www.cgtrader.com/
[10]https://www.turbosquid.com/
[11]https://3dwarehouse.sketchup.com/?hl=en
[12]https://www.shapenet.org/

3.2 Virtual Aerial Vehicle Integration

Simulator	Implementation Language	Supported OS	Licence	Developer	High-Level Dependencies
Gazebo [44]	C++	Linux, MacOS	Apache V2.0	Open Source Robotics Foundation	
AirSim [81]	C++	Windows, Linux	MIT	Microsoft	Unreal Engine 4
jMAVSim [26]	Java	Linux, MacOS, Windows	BSD 3	DroneCode Project	Java3D
HackFlightSim (renamed MulticoptorSim) [56]	C++	Linux, Windows	GPL	Simon Levy	Unreal Engine 4
RotorS [108]	C++	Ubuntu	ASL 2.0	ETH Zurich	ROS, Gazebo
Morse [28]	Python	Linux	BSD	LAAS-CNRS	Blender Game Engine
New Paparazzi Simulator [35]	C	Linux, MacOS	GPL v2	Ecole Nationale de l' Aviation Civil	JSBSim

Table 3.1 Simulator Comparisons, based on [27]

In order to add high-level autonomous aerial vehicle support to the simulation environment, we used the AirSim [81] plugin for UE4. Table 3.1 shows candidates that were considered to provide the support for simulated RAVs. We chose to use AirSim with UE4 over other simulation technologies for the following reasons, based on the paper published by the AirSim team [81]:

- The design process of the physical environment could be decoupled from the plugin and other plugins could add further functionality independently of AirSim. This meant that new versions of AirSim could be seamlessly integrated with the existing simulation.
- UE4 has ample documentation and is highly suited to simulating physical phenomena.

- AirSim has high-level, extensible navigation and sensor APIs for multiple programming languages.
- AirSim provides the ability to run multiple RAVs and RGVs in the same environment.
- AirSim has been open-sourced with an MIT licence and was designed to be easily extended.
- AirSim provides an API to render unmodified images, depth maps and segmented images.
- AirSim is highly modular in its approach to modelling sensors and actuators, which means that it can easily be extended with domain-specific functionality.
- AirSim supports flight controller firmware such as PX4, ROSFlight and Hackflight with minimal setup.

AirSim internally models RAVs using different low-level modules outlined in [81]. Figure 3.12, published in [81], provides an overview of how the different components of the simulator work together.

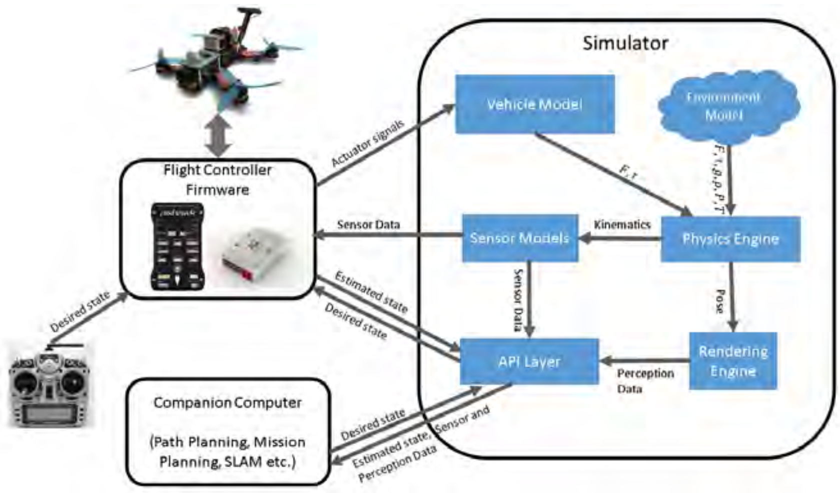


Fig. 3.12 The architecture of the AirSim simulator. Diagram taken from [81]

Once the plugin had been successfully integrated into the project, some additional functionality was added to meet some of the design objectives, described in the subsequent section.

3.2.1 Modifications and Extensions

As shown in Figure 3.12, AirSim maintains the internal state related to the vehicles that it is simulating and allows the user to modify this state through an API layer. In practice, this means that adding functionality to the API layer involves modifying a number of layers underneath the API layer. While making all the changes listed below, the steps documented by the plugin authors[13] were followed: first, the plugin code is edited in order to add the desired functionality, then the plugin is built, added to a UE4 environment and tested. This process is necessary since AirSim is a plugin - there is no easy way to run and test it as a standalone piece of code. All of the modifications were made to a fork of the main AirSim repository[14].

AirSim API Overview

In order to make modifications to the AirSim API, it is necessary to understand how the API has been designed. The API is socket-based over rpclib[15], a library that implements the **R**emote **P**rocedure **C**all (RPC) using the msgpack[16] binary serialization protocol for speed. The server contains an instance of the RPC server, and binds methods exposed to the user to internal methods responsible for interacting with the simulation. The server effectively dispatches the user's request to internal API's that are not directly exposed to the user, which are responsible for modifying the simulation parameters as the simulation proceeds.

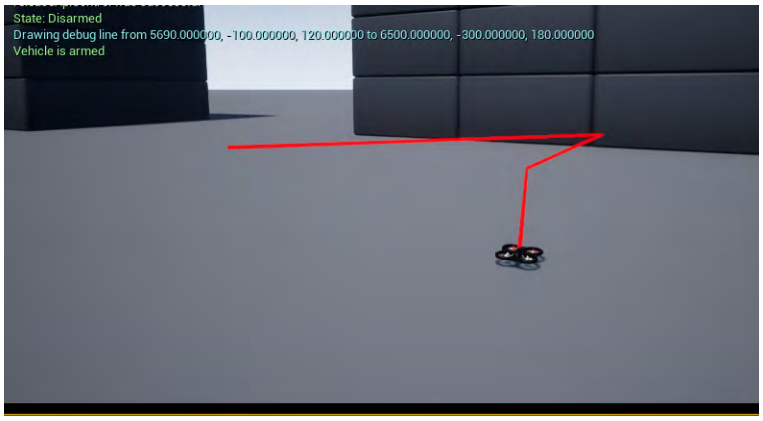
The RPCLibServerBase class uses the **P**ointer to **Impl**ementation (PImpl) design to bind the RPC server to an instance of the ApiProvider class, which is responsible for dispatching commands to the three base class defining internal APIs that are currently implemented in AirSim:

- VehicleApiBase, which is responsible for getting the state from the vehicle and sending controls commands to the vehicle. Derived classes define additional methods specific to a vehicle.
- VehicleSimApiBase, which is responsible for handling commands that aren't directly related to the vehicle's state, such as rendering the vehicle in UE4.
- WorldSimApiBase, which handles non-vehicle specific commands, for example setting the time of day in the simulation.

[13]https://microsoft.github.io/AirSim/docs/dev_workflow/
[14]https://github.com/microsoft/AirSim
[15]http://rpclib.net/
[16]https://msgpack.org/

Debugging Lines for Planned Routes

The first modification made to AirSim was to extend the API to allow the user to visualize planned routes using debugging lines. The end result can be seen in Figure 3.13. We used a top-down approach to make the appropriate changes. First, we specified the functionality that we wanted to expose through the AirSim API. This was determined to be as follows: given 2 points in the 3-Dimensional Unreal Engine Frame of Reference (NED), a line thickness, a line colour and a time which the line will be shown for, display the specified line in the simulation environment. We wanted this to be available in both the global frame of reference and the local RAV frame of reference. This was planned to allow the user to visualise a series of lines from one specified waypoint to the next, on aggregate creating a visualisation of a planned path.



```
State: Disarmed
Drawing debug line from 5690.000000, -100.000000, 120.000000 to 6500.000000, -300.000000, 180.000000
Vehicle is armed
```

Fig. 3.13 Debugging Lines Visualise planned RAV route.

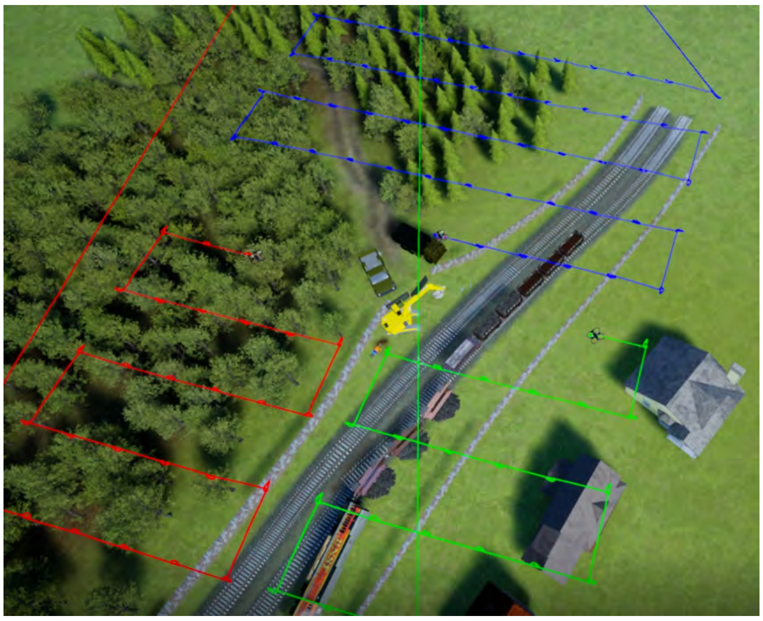


Fig. 3.14 RAV path visualised mid-way through planned mission.

First, the methods simShowDebugLines and showPlannedWaypoints were added to the client code, in RpcLibClientBase.hpp, with method signatures

```
void simShowDebugLines(double x1,
    double y1,
    double z1,
    double x2,
    double y2,
    double z2,
    double thickness,
    double lifetime,
    const std::string& debug_line_color
);

void showPlannedWaypoints(double x1,
    double y1,
    double z1,
    double x2,
    double y2,
    double z2,
    double thickness,
    double lifetime,
    const std::string& debug_line_color,
    const std::string& vehicle_name = ""
);
```

The corresponding server methods were added to RpcLibServerBase.cpp with the following signatures:

```
pimpl_->server.bind("simShowDebugLines", [&](double x1,
double y1, double z1, double x2, double y2, double z2,
double thickness, double lifetime,
const std::string debug_line_color) -> void {
        getWorldSimApi()->showDebugLine(x1, y1, z1, x2, y2, z2,
        thickness, lifetime, debug_line_color);
});

pimpl_->server.bind("simShowPlannedWaypoints", [&](double x1,
    double y1, double z1, double x2, double y2, double z2,
    double thickness, double lifetime,
    const std::string debug_line_color,
    const std::string& vehicle_name) -> void {
            getVehicleSimApi(vehicle_name)->showPlannedWaypoints(x1,
            y1, z1, x2,y2, z2, thickness, lifetime,
            debug_line_color);
});
```

The server delegates the handling of the debug line display to the worldSimApi in the case that the user wants to display in the global reference frame in the first case and the vehicleSimApi handles the local reference frame in the second case.

A concrete implementation of vehicleSimApi then transforms the coordinates to the local reference frame of the vehicle. The showDebugLine method added to AirBlueprintLib.cpp uses the DrawDebugLine method, defined in the DrawDebugHelpers library, to directly interface with UE4 to draw the debug lines.

```
void DrawDebugLine
(
    const UWorld * InWorld,
    FVector const & LineStart,
    FVector const & LineEnd,
    FColor const & Color,
    bool bPersistentLines,
    float LifeTime,
    uint8 DepthPriority,
    float Thickness
)
```

Debug lines showing where the RAV has travelled were also added to the simulation. This was done by using a Blueprint, outlined in section 3.1.5. A variable, *oldPos*, is used to store the previous position of the RAV. Every 0.2 seconds, the position is updated and a line is drawn from the RAV's old position to its new position. The user has the option to turn debugging lines on or off by setting a binary variable in the blueprint.

3.2.2 Radiation Simulation

In order to simulate radiation using UE4, it was necessary to model both the radiation emission and detection using a sensor. First, we developed a model of the level of ionizing radiation observed d meters from a point source, making the assumptions that the radiation is emitted from a point and is transmitted equally in all directions. We also assumed that the propagating medium is lossless, implying that the level of ionizing radiation is conserved. Letting d = distance from point source of radiation, the strength of the ionizing radiation in micro Sieverts per second ($\mu\mathrm{Sv\,s^{-1}}$) is determined by the equation $strength(d) \propto \frac{1}{d^2}$. Letting

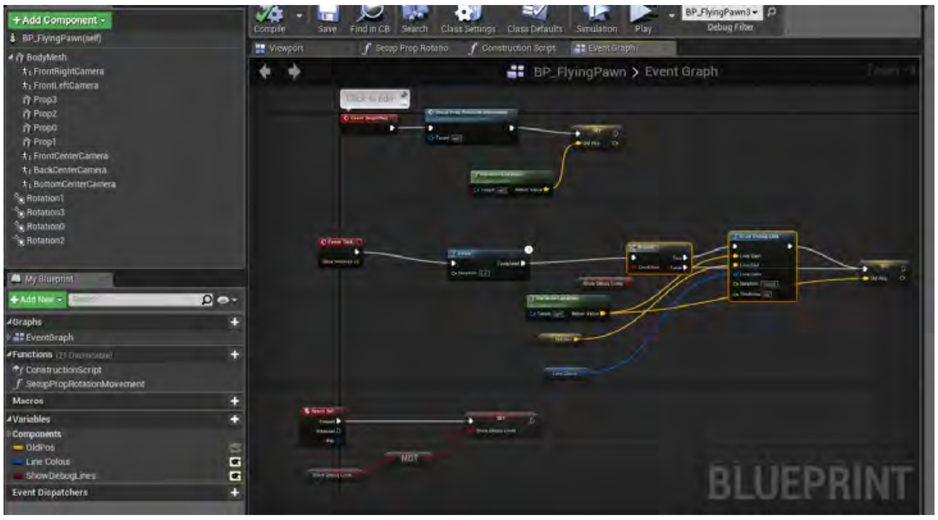


Fig. 3.15 Blueprint used to create the debugging lines tracking a RAV

the strength at a distance of 1m from the source equal to $\sigma \mu\mathrm{Sv\,s}^{-1}$, the ionizing radiation in $\mu\mathrm{Sv\,s}^{-1}$ is given by:

$$strength(d) = \frac{\sigma \times 4 \times \pi}{d^2 \times 4 \times \pi} = \frac{\sigma}{d^2} \mu\mathrm{Sv\,s}^{-1}$$

We found that the GameplayStatics library in the Engine module[17] could provide a large part of the functionality. The simulated emission of radiation was modelled in UE4 by using the built-in blueprint node, *Apply Radial Damage with Falloff*[18], since it provides a ready-made inverse-square equation that can be parameterised to match the equation identified above to give the level of ionizing radiation.

The detection of ionizing radiation strength can then be simulated using the TakeDamage C++ method, which is automatically fired when the instance of the object that implements this method takes damage in UE4. An actor class, which can be associated with a physical object in the game, was created to use this method to record the current level of radiation detected and then expose it via a method to the vehicleSimApi, so that it may be queried from the server during a simulation run.

3.3 Conclusion

In this chapter, we presented a custom-built high-fidelity simulation environment which runs on the UE4 game engine. The final results of this are visualised in Figures 3.3 and 3.4. The

[17]https://api.unrealengine.com/INT/API/Runtime/Engine/index.html
[18]https://api.unrealengine.com

solution that we outlined using UE4 and AirSim met all of the criteria set out in Section 3.1.1 and we released the simulation environment for public use on a Github repository[19]. The simulation environment has been used in the ROCSAFE project for multiple purposes and we envision that future work may be able to build on the approach that we took to generate diverse datasets that may be used to prototype, test and validate systems intended to deal with scenarios that present dangers physical human presence.

[19]https://github.com/NUIG-ROCSAFE/CBRNeVirtualEnvironment

Chapter 4

Scene Surveying

In this chapter, we provide details of the techniques and software developed to address the problem of aiding autonomous scene surveying, which was identified as part of the first research question stated in Section 1.2:

"*Can an agent-based software system be developed to run on a system of heterogeneous autonomous aerial vehicles to aid the tasks of scene surveying and target localisation in a hazardous environment?* "

As with the rest of the work in this thesis, this chapter is motivated by the context of the ROCSAFE project, but we present the work in a general setting. In the initial phase of any crime scene investigation, it is desirable for the crime scene manager to visually scan the entire crime scene area to thoroughly assess the scene as early as possible [95]. We will henceforth refer to the area to be scanned as the *region of interest* for the purpose of generality. In the ROCSAFE project, it is intended to address this problem using a fleet of RAVs, which are equipped with cameras and sensors which are highly suited to data-gathering. The RAVs can operate concurrently and can communicate with a centralised controller. We developed a system to autonomously survey a hazardous situation using the simulation environment described in Chapter 3. We published this work in a number of papers [86], [85], [87].

We began by providing a description of the two main sub-problems identified as part of autonomous scene-surveying:

1. Given a polygonal region defined by a set of points that lie on the earth's surface, generate a discrete set of cells of uniform area that partition the region of interest. The cells can be described by their centre points, as they are assumed to be of uniform height and width. We will refer to this set as R, the set of *waypoints*.

2. Find a set of routes for each of *K* RAVs to be used in the data-gathering process, such that these sets partition *R* and the cost of the system of RAVs traversing these points is minimized.

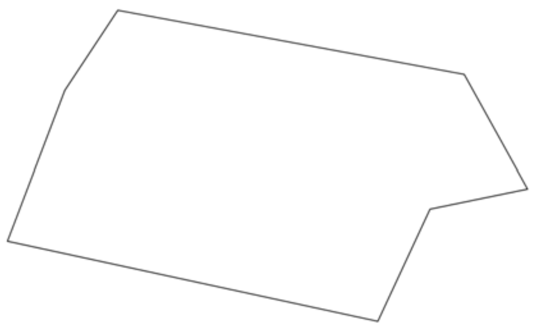
Hence there are two main goals: find a discrete representation of a continuous area which represents the region of interest, and then find routes that the RAVs can execute in order to visit each of the grid points exactly once, while minimising the cost of doing so.

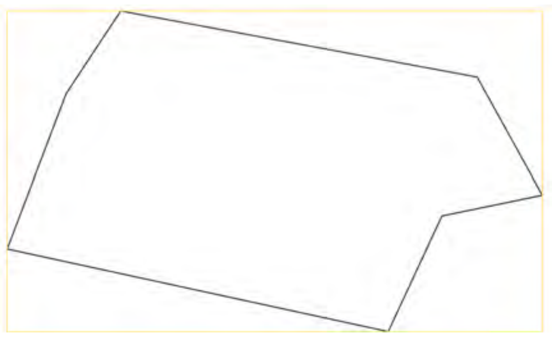
4.1 Generation of Waypoints

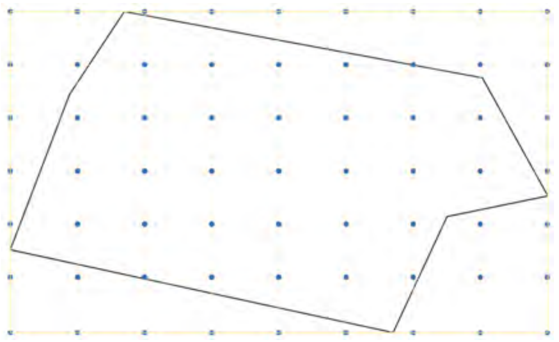
We first address the issue of finding a discrete set of waypoints, referenced above in point 1 as *R*. We chose to formulate the problem using a discrete set of waypoints, rather than deal with the case of treating the region as continuous space, since it is much easier to accurately record sensor data when the RAV has settled at a given point.

We began by assuming that the region which is to be surveyed, *R*, can be described by a polygon on the Cartesian plane, since it allows for a discrete representation by describing the coordinates of the corners of the polygon. Given this polygon, the goal is to generate a set of points, *R*, which are a uniform distance from each other in the x and y directions and lie inside the polygonal grid. The set *R* forms a regular tessellation of the region of interest. In order to do this, we employed the following methodology:

1. Find the circum-rectangle that tightly bounds the polygon, which is oriented with the x-y plane. This is shown in 4.1b
2. Generate a set of grid points in the bounding rectangle. This is shown in 4.1c
3. Prune the points that lie inside the rectangle but outside the polygon. This is shown in 4.1d


(a) Arbitrary Polygonal Region


(b) Bounding Rectangle Found


(c) Candidate Grid Points Generated in Bounding Rectangle

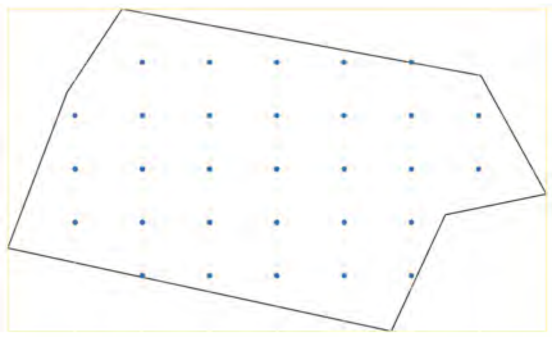

(d) Grid Points Outside of Polygon Pruned

Fig. 4.1 A visualisation of the steps involved in Algorithm 4

Algorithm 3 Algorithm to Generate a Uniformly Spaced Grid of Points in an Arbitrary Polygon

Input:

R :	The set of (x, y) points defining the polygon which covers the region of interest
x_spacing :	The desired spacing between points in the x direction
y_spacing :	The desired spacing between points in the y direction

Output:

grid_points : A set of uniformly spaced (x, y) points, which define a regular tessellation of the region of interest

```
1: grid_points ← empty array
2: max_x ← maximum x value of all points in R
3: min_x ← minimum x value of all points in R
4: max_y ← maximum y value of all points in R
5: min_y ← minimum y value of all points in R
6: bounding_rect ← The tightest bounding rectangle which contains the polygon R defined
   by the points (min_x, min_y), (min_x, max_y), (max_x, max_y),(max_x, min_y).
7: no_y_points ← $\left\lfloor \frac{max\_y - min\_y}{y\_spacing} \right\rfloor$
8: no_x_points ← $\left\lfloor \frac{max\_x - min\_x}{x\_spacing} \right\rfloor$
9: for y_spacing_index = 0 to no_y_points do
10:    for x_spacing_index = 0 to no_x_points do
11:        p←(min_x+x_spacing × x_spacing_index, min_y+y_spacing × y_spacing_index)
12:        if Point-in-Polygon(R, p) then
13:            Add p to grid_points
14:        end if
15:    end for
16: end for
17: return grid_points
```

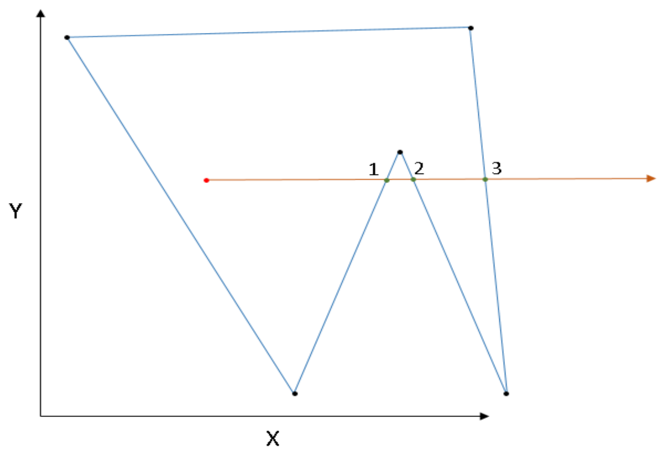
We describe the grid point generation algorithm (Algorithm 3) here. First, the bounding circum-rectangle of the polygonal region of interest is constructed from the largest and smallest x and y coordinates of the points of the polygon, outlined in lines 2-6 and shown in Figure 4.1b. Then candidate uniformly spaced grid points in the bounding rectangle are generated in lines 7-11, shown in Figure 4.1c. Finally in lines 12-13, a check is performed which adds the grid point to the set of grid point contained in the polygon if it passes a check, shown in Figure 4.1d. This is done using the Point-in-Polygon routine which is provided separately in Algorithm 4, with the details discussed in the next paragraph.

Algorithm 4 Point-in-Polygon

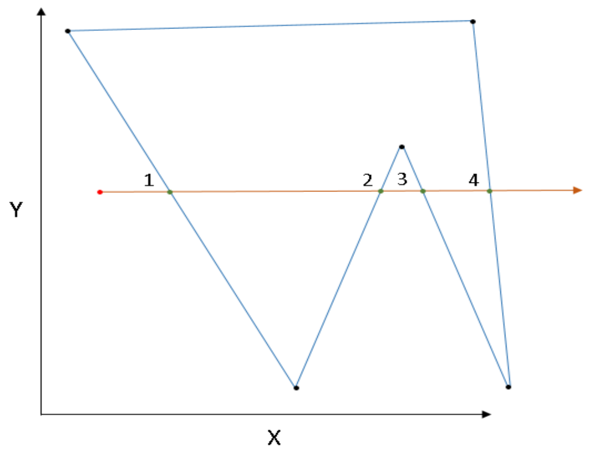
```
Input:
    R :   The polygon which covers the region of interest
    P :   A point which will be tested for containment in the polygon
Output:
    True if P is contained in R else False

 1: polygon_points ← The set of points defining the vertices of R
 2: if $P_X$ is greater than the largest value or less than the smallest value of the x coordinates
    of the points in R then
 3:     return False
 4: else if $P_Y$ is greater than the largest value or less than the smallest value of the y
    coordinates of the points in R then
 5:     return False
 6: else
 7:     point_in_polygon ← False
 8:     edges ← the set of edges defining R
 9:     for each edge in edges do
10:         P1 ← the first point of edge
11:         P2 ← the second point of edge
12:         if ($P_Y$ lies between $P1_y$ and $P2_y$) & $P_X$ is less than the x coordinate of the point of
            intersection between edge and the ray extended to $+\infty$ from P in the +x direction
            then
13:             point_in_polygon ← ¬ point_in_polygon
14:         end if
15:     end for
16: end if
17: return point_in_polygon
```

Algorithm 4 tests whether a point lies within a polygon, described by a set of edges. Its implementation uses the *crossing test* [83], which extends a ray from the point to be tested in the positive x-direction. If it crosses the boundary of the polygon an odd number of times, then it must be contained within the polygon, otherwise it must lie outside. This is illustrated



(a) Odd number of crossings



(b) Even number of crossings

Fig. 4.2 Examples of a positive and negative results when applying the Crossing Test

in Figure 4.2. The implementation is provided in the Point-In-Polygon routine shown in Algorithm 4.

4.1.1 Generating Waypoints as GPS Locations

For the sake of real-world usage, it is necessary to refer to these points using some kind of reference coordinate system that RAVs can use. We provide a brief overview of how we accomplished this, without delving too far into Geographic Coordinate Systems (GCS). Since the *Global Positioning System* (GPS) is by far the most common system that RAVs use for navigation, we chose to design an implementation based on the *World Geodetic System 1984* (WGS84) coordinate system, which is the coordinate system that GPS relies upon. The WGS84 system was created by the US Department of Defence and is officially documented in the official government report entitled "*Department of Defense World Geodetic System 1984 Its Definition and Relationships With Local Geodetic Systems*" [71]. It assumes that the datum surface is an oblate spheroid. Coordinates are referenced using latitude, longitude and altitude, where longitude measurements range between [-180°, 180°], latitude measurements range between [-90°, 90°] and altitude measurements come in the form of meters above sea level. Greenwich is used as the *prime meridian* for longitude, meaning Greenwich defines 0° longitude. The equator is used to define 0° latitude. In order to generate a grid of points in a bounding polygon as coordinates that can be referenced by the RAVs' GPS sensors, we had to make some modifications to the grid point generation algorithm outlined above. The main issue that arose was that the grid point generation algorithm assumes that the Cartesian coordinate system is used, which is not valid when assuming coordinates lie on a non-planar surface (the earth). As an example, we present results quoted from Robert G. Chamberlain, reviewed on the comp.infosystems.gis newsgroup in October 1996 [14]:

"*If the distance is less than about 20 km (12 mi) and the locations of the two points in Cartesian coordinates are X1,Y1 and X2,Y2 then using Pythogoras' theorem with Cartesian coordinates can generate errors of*

1. *less than 30 meters (100 ft) for latitudes less than 70 degrees*
2. *less than 20 meters (66 ft) for latitudes less than 50 degrees*
3. *less than 9 meters (30 ft) for latitudes less than 30 degrees*

"

This is due to the non-uniform curvature of the earth and shows that the accuracy of the using Cartesian coordinates with Pythagoras' theorem depends on the latitude at which it is applied. For the sake of guaranteed accuracy we decided to use the methods which were first described by Vincenty [99], which gave a far superior accuracy:
"*The latitudes of standpoints were from 0° to 80° in increments of 10' and the distances were in multiples of 2000 km up to 18000 km, which gave 81 test lines. The maximum disagreement was 0·01 mm.*".
Based on this, we made the following minor changes to the algorithm for use with WGS84 coordinates:

1. When calculating distances, the *inverse solution* [99] to finding geodesics (shortest paths along the earth's surface) was used. We also used the *direct solution* [99] to find a destination given a start point, bearing and distance. These algorithms take into account the non-planar nature of the WGS84 coordinate system. The inverse solution can be used in place of subtraction of Cartesian Coordinates. It would replace the

subtraction in lines 7 and 8 of Algorithm 3 and line 12 of Algorithm 4. The direct solution can be used in place of addition of distance to Cartesian coordinates and is used in line 11 of Algorithm 3.

2. We created a specific WGS84 coordinate class, which can be used in place of regular Cartesian coordinates in Algorithms 3 and 4. We uploaded this to a publicly available Github repository[1] with an MIT licence. If the user is not interested in generating grids which have very accurately spaced points, they can use the Cartesian solution which approximates the region of interest as a plane, otherwise the solution using Vincenty's equations in the point above can be applied.

4.1.2 User Interface and Results of Grid Generation Algorithm

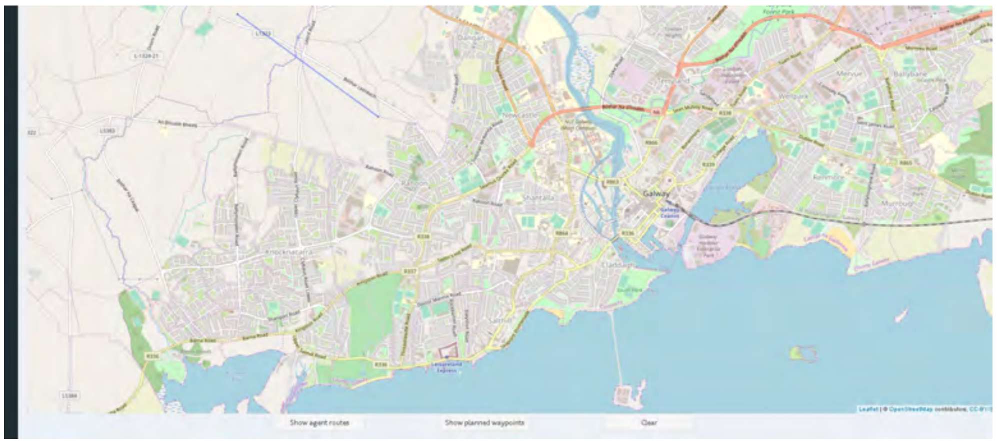


Fig. 4.3 The UI developed to quickly generate uniformly spaced grid points in an arbitrary polygonal region to survey

We developed a User Interface (UI) in order to have the ability to quickly generate grids for use in simulations, as outlined in Section 4.2. We also used the UI to visually validate that the grid generation code was performing as expected. We chose to use the Shiny framework[2], which is a web app development framework written in the R language[3]. We used the Leaflet Package[4], which provides a high-level interface to the Open Street Map[5] API to provide the interface to a map with WGS84 Coordinate reference system. We added an *observeEvent* function to record when the user clicks on points on the map, adding an edge the previously

[1]https://github.com/DavidLSmyth/WGS84Coordinate
[2]https://shiny.rstudio.com/
[3]https://www.r-project.org/
[4]https://rstudio.github.io/leaflet/
[5]https://wiki.openstreetmap.org/wiki/Develop

clicked point to the current clicked point. The process of sequentially selecting the points on the map defining a polygonal region can be seen in Figure 4.4.

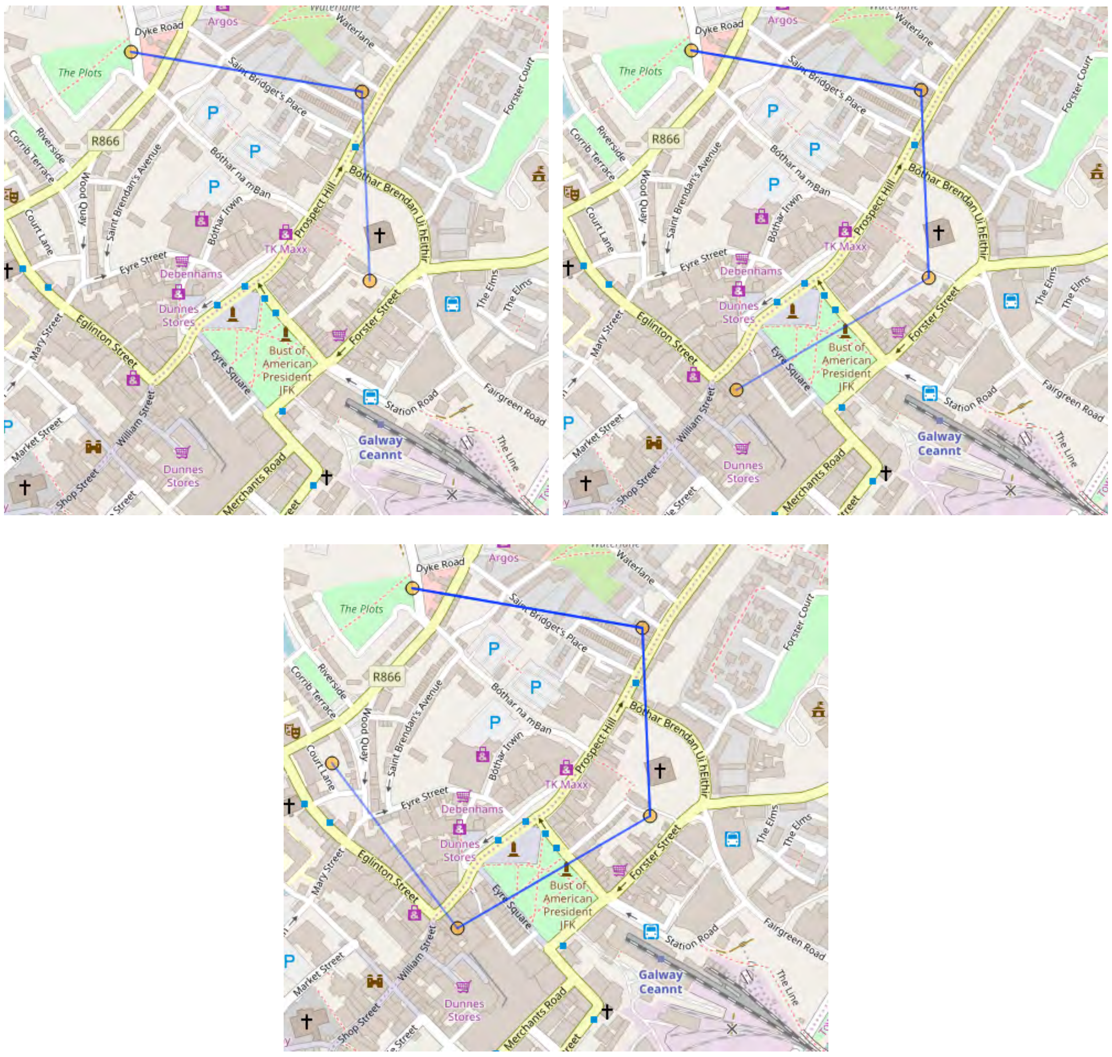


Fig. 4.4 Creating the bounding polygon using the UI

In order to actually generate and draw the grid points on the user interface, we added "Show planned waypoints" button, which can be seen in Figure 4.3. When this button is clicked, the polygon defining the region of interest that the user has selected is sent to the grid generation code, as well as the desired latitude and longitude spacing between the generated grid points. The grid points are generated and written to a file. The user interface then reads the points from the file and then overlays them on the map. Examples of visualisation of polygonal regions with grid points are shown in Figure 4.5. We open-sourced this code with an MIT licence on Github [63].

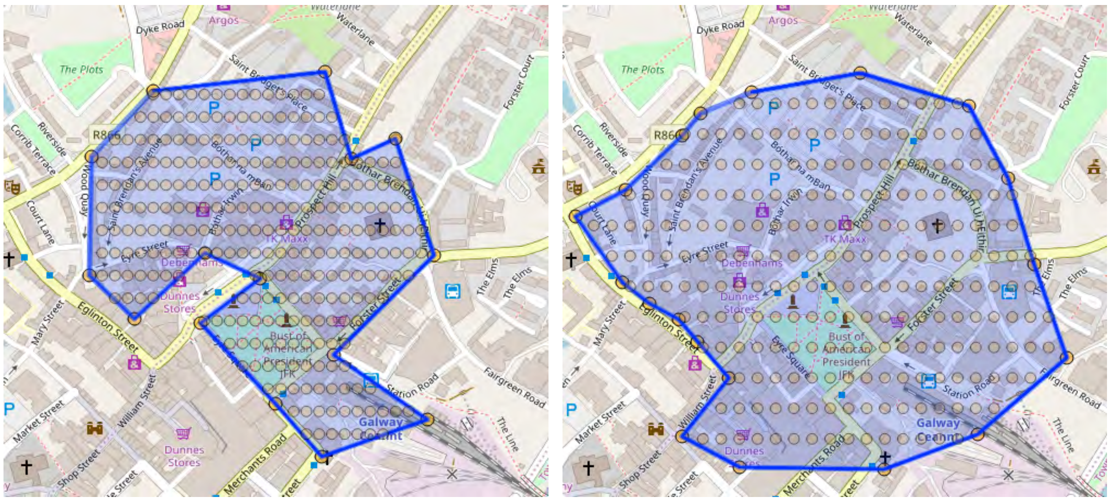


Fig. 4.5 Examples of generated grid points overlaid onto the UI

4.2 Scene Surveying

This section outlines the algorithms that were explored to generate a set of routes for the RAVs that will be used to record sensor data at each of the grid points generated in the region of interest. We gave a high-level description of this problem at the beginning of Chapter 4: "*Find a set of routes for each of K RAVs to be used in the data-gathering process, such that these sets partition R and the cost of the system of RAVs traversing these points is minimized*". We make some assumptions related to the solution of this problem:

- Each RAV is assumed to have the same internal representation of the region of interest, namely the set of uniformly spaced grid points generated by Algorithm 3.
- Each RAV is assumed to have the ability to move between any pair of grid points unobstructed using the shortest possible path.
- RAVs are assumed to move with a fixed operational velocity, which can vary between RAVs.
- Each RAV may be equipped with different sensors to the others and it is assumed that sensing times may vary among the RAVs.
- Each RAV has a finite battery capacity which implies they have a finite amount of time that they can fly for before they need to recharge.

The scene surveying problem that we would like to solve can be treated as an instance of the *Vehicle Routing Problem* (VRP), which was first described in a paper by Dantzig and

Ramser [23]. This problem is a generalisation of the classic *Travelling Salesman problem* (TSP). There are many variants and extensions to this problem, but the VRP essentially asks for a set of routes to be assigned to the RAVs such that each point in the graph is visited exactly once, which minimises the total time taken to "service" each of the points [23]. In our case, the service time is the time taken to record a sensor reading. We are interested in solving this problem where the graph is defined by the grid of points that is generated using Algorithm 3. A formal definition of the VRP and its variants can be found in [97].

4.2.1 Simplified Problem

We began by taking a simplified version of the full vehicle routing problem in order to explore possible solutions. This section expands on our published work in [87], which summarises how this simplified problem was tackled. Rather than concern ourselves with the details of sensor sampling times and battery constraints, we first focused on designing a solution that can assign a set of routes to a homogeneous set of RAVs. The problem can be described as follows:
Given a fully connected graph, G, to visit and n RAV agents, find a subtour for each agent such that each point in $P \in G$ is visited exactly once by any agent in the system, with the objective of minimizing the longest time taken for any individual agent subtour, in order to minimize the time taken to carry out the survey.
This is the *multiple Travelling Salesman Problem* (mTSP).

4.2.2 Proposed Solution for the Simplified Problem

Due to the fact that the mTSP is at least as hard as the TSP, since it is a generalisation of the TSP, finding a polynomial time solution to the mTSP is not feasible. We explored a number of sub-optimal solutions that take advantage of the highly structured nature of the uniformly spaced grid that we use in our instance of the problem. The usual trade-off between solution quality vs. time taken to find the solution motivated our choice of implemented algorithm, as we anticipated that the time taken to execute the planned routes may be comparable to the time taken to generate some solutions (the order of minutes or hours). For example, Hungerländer et al. shows the results of using a Mixed Integer Linear Program (MILP) took hours to run for grid sizes that could be considered relatively small in many real-world domains [38].

4.2.3 Nearest Neighbour Algorithm

There are four common heuristic algorithms that form the basis of most solutions to TSP and mTSP problems, as stated in [40]: the *nearest neighbour* algorithm, the *greedy algorithm*, the *Clarke-Wright* algorithm and the *Christofides* algorithm. We began by implementing an algorithm which is a modification of a TSP solution based on the nearest-neighbour heuristic. Our reasoning is based on the following premises:

1. The nearest-neighbour heuristic is a well-known heuristic that is straightforward to implement. It is known to provide good results to the TSP when the cost is defined as the Euclidean distance between points [40], as in our use case.

2. The nearest-neighbour solution to the TSP can be very easily modified to be applied to the mTSP by implementing it in a round-robin manner, as detailed in Algorithm 5.

3. The nearest-neighbour heuristic is known to have a running time which is $O(n^2)$ [74], which means it can scale well to reasonably large problem instances compared to other algorithms which have a far worse performance complexity. For example, the Christofides algorithm is known to be within a factor of $\frac{3}{2}$ of the optimal solution, but its running time is $O(n^3)$ [17], which quickly becomes prohibitively long.

The nearest-neighbour (NN) heuristic algorithm for the multiple Travelling Salesman problem is shown in Algorithm 5.

The algorithm can be seen to be a very straightforward extension of the single agent NN algorithm. The while loop iteratively cycles through the list of RAVs and assigns the nearest non-assigned neighbour to the next RAV's partially assembled route.

Since we run this algorithm over a uniformly-spaced grid, it builds agent routes in a lawn-cutting pattern, which provides good results as long as the routes do not begin to overlap. The idea is to take the NN algorithm TSP solution and apply it exhaustively in a round-robin manner, so that each RAV is encouraged to find a non-overlapping optimal sub-tour. These non-overlapping optimal sub-tours partition the region and therefore offer a solution to the mTSP. Section 4 of [38] gives an insight to how optimal solutions can be found for a uniformly spaced rectangular grid when RAVs are assumed to start in the corners of the rectangle, which provides further motivation to using Algorithm 5. In practice non-overlapping optimal sub-tours are not found, but solutions may come close, as illustrated in Figure 4.6.

Algorithm 5 The Nearest-Neighbour Solution to the mTSP Problem

```
Input:
    $k$ :   The number of RAVs in the mTSP
    $way\_points$ :   The set of points the RAVs must visit
    $cost(i, j)$ :   The function giving the cost of travelling from node i to node j
Output:
    $RAV\_routes$ :   A key-value data structure mapping RAVs to their corresponding routes.

 1: RAV_routes←empty key-value container
 2: visited_points←empty array
 3: remaining_points_to_visit←way_points
 4: for each agent in agents: do
 5:     Initialise path of agent as empty array in RAV_routes
 6: end for
    current_agent_index←0

 7: while pointsToVisit is not empty do
 8:     agent_position ← last value in agent_paths.get(current_agent_index)
 9:     nearest_neighbour← $\min_{neighbour \in points\_to\_visit}$ cost(agent_position, neighbour)
10:     Update current_agent value in agent_paths to include nearest_neighbour
11:     Add nearest_neighbour to visited_points.
12:     Remove nearest_neighbour from remaining_points_to_visit.
13:     current_agent_index←(currentAgentIndex+1) mod|List of Agents|
14: end while
15: return RAV_routes
```

4.2.4 Analysis of the NN algorithm

The results and analysis of applying the NN heuristic algorithm to the symmetric Euclidean TSP and mTSP are well documented and we will not go into great detail repeating the results here. Instead, we refer the reader to the chapter "The travelling salesman problem: A case study" of [1], written by Johnson and Mcgeoch, which provides a comparison of the NN heuristic to other heuristic algorithms. They compare solutions with the standard Held-Karp lower bound [36] using a standardised library of Travelling Salesman Problems, TSPLIB [70]. Rather than provide tables of results showing the performance of the NN heuristic algorithm on standard benchmark data sets such as TSPLIB, we instead focus on the results for the domain we are most interested in, which is a connected graph defined by a uniformly spaced set of grid points. We found that the algorithm performed best when the RAVs used regions that have multiple axes of symmetry. Rectangular regions in particular yield scalable, high-quality solutions, as shown in the figures in Table 4.1. This corresponds to the optimal

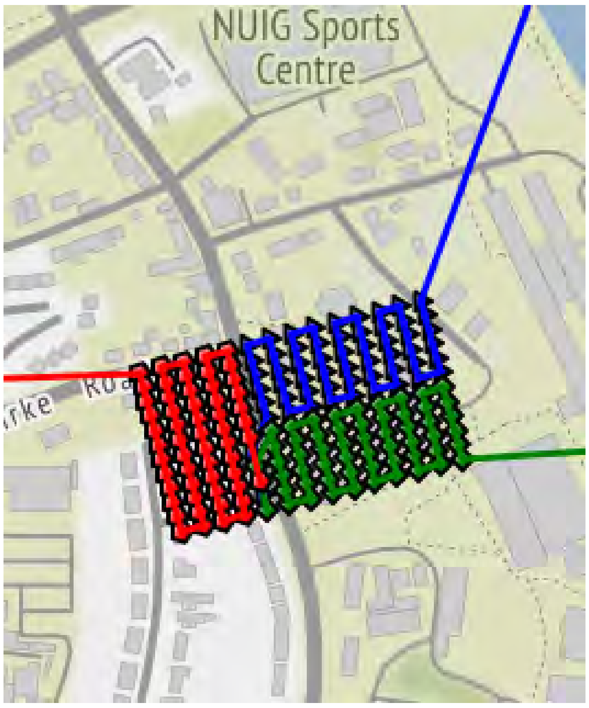


Fig. 4.6 NN heuristic encourages the creation of optimal sub-tours, which are assigned to each RAV

performance configuration suggested in [38]. We evaluated solutions both qualitatively and quantitatively. We focused on the qualitative results of the experiments, which were designed to show that the system can partition the in the region to give a reasonably balanced amount of work to each agent for regular polygons with a number of axes of symmetry. This was a proof-of-concept, with the intention for future work to provide a more thorough analysis on performance. We found that RAV starting points are a critical factor in solution quality, which again is in line with the findings in [38]. Tables 4.1, 4.2 and 4.3 illustrate some of the results, with the configuration of each listed below. The experiments are intended to demonstrate practical use cases where the NN algorithm will provide qualitatively good results.

4.2.5 Qualitative Behaviour of the NN Algorithm with a Rectangular Grid

We set up this experiment with the aim of showing that if the RAVs can be placed in the configuration suggested by Hungerländer et al., they will partition the region in a close to optimal fashion [38], minimising the longest distance any one RAV needs to travel to complete its mission. We ran the experiment using 1-4 RAVs, placing them in the corners of the rectangles. This was done in an approximate fashion by manually dragging the RAVs to their starting position using the UI mentioned in Section 4.1.2. The results in Table 4.1 show that the amount of work done by each RAV is approximately evenly balanced. Theoretically, the minimum amount of work done by each RAV would be $\frac{1}{2}$, $\frac{1}{3}$ and $\frac{1}{4}$ of the total work done by a single RAV when using 2, 3 and 4 RAVs respectively.

The results show that when using 2, 3 and 4 RAVs, the solutions found by the NN algorithm were 2.701%, 4.500% and 17.315% less efficient than evenly splitting up the solution for a single RAV ($\frac{1}{2}$, $\frac{1}{3}$ and $\frac{1}{4}$ of 3576.6m of the single-RAV route length, respectively). This was partly due to the extra distance added from the starting positions. Note that when using 4 RAVs, there was some overlap in the routes which added a significantly higher overhead to the cost of the route in relation to that given by even splitting the solution found for a single RAV.

We used the following configuration to run the experiment:
Spacing between grid points: 23m in latitude, 25m in longitude.
Bounding rectangle coordinates: (53.2781933786, -9.0671226391), (53.2803800539, -9.067182416), (53.2804392784, -9.0611222377), (53.2782526061, -9.0610624609)

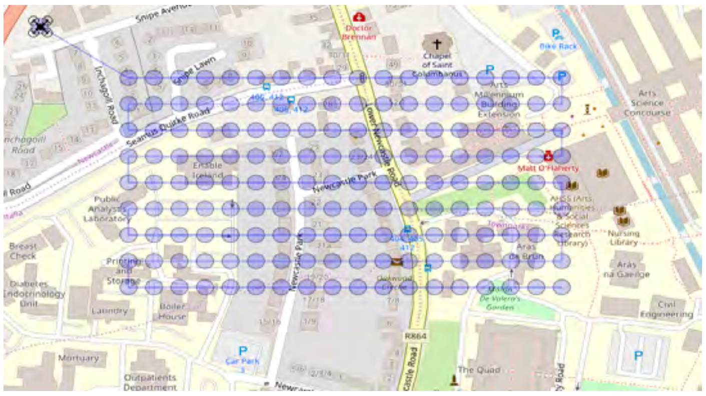
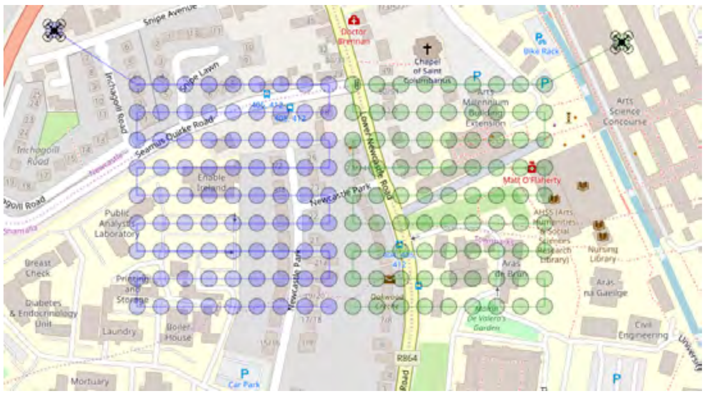
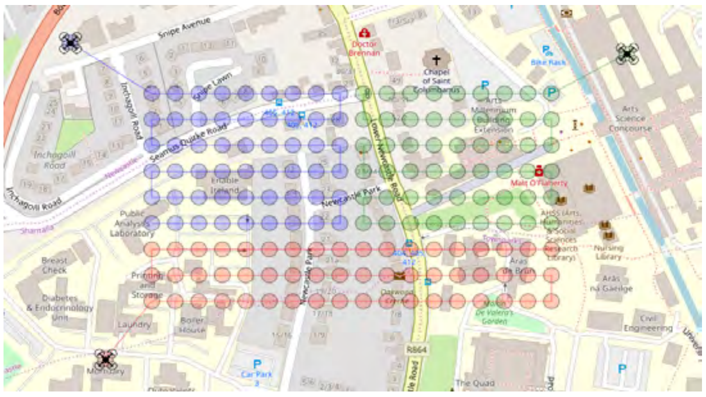
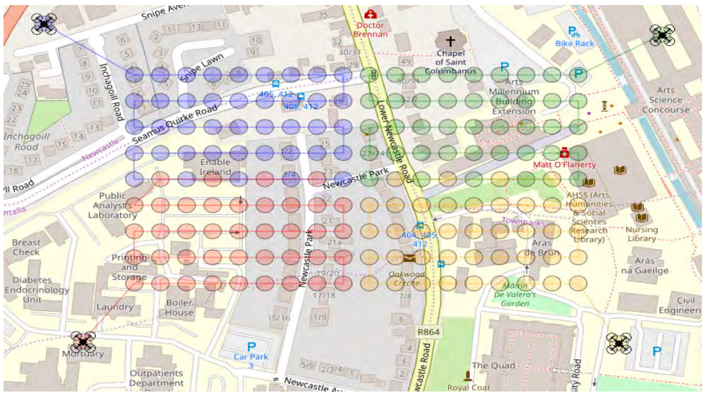
Planned RAV Routes	Route Lengths (metres)
	RAV 1 (blue): 3576.6
	RAV 1 (blue): 1836.3 RAV 2 (green): 1825.8
	RAV 1 (blue): 1245.6 RAV 2 (green): 1235.1 RAV 3 (red): 1215.9
	RAV 1 (blue): 1048.8 RAV 2 (green): 1038.3 RAV 3 (red): 994.5 RAV 4 (yellow): 990.5

Table 4.1 Results of applying NN algorithm to a rectangular region

4.2.6 Qualitative Behaviour of the NN Algorithm with a Triangular Grid

This experiment was set up in a similar manner to the first, where the RAVs were dragged to an initial starting position that should allow the NN algorithm to take advantage of the axes of symmetry of the triangle. The results in Table 4.2 show that the amount of work done by each RAV is approximately evenly balanced when using two RAVs, but not three.

The results show that when using two and three RAVs, the solutions found by the NN algorithm were 0.155% and 36.565% less efficient than that given by even splitting the solution for a single RAv ($\frac{1}{2}$ and $\frac{1}{3}$ of 1940.6, respectively). This can be explained by the corresponding figures displaying the agents' routes in Table 4.2. Clearly, the NN algorithm takes advantage of the symmetry down the vertical centre axis when two RAVs are used, but when three RAVs are used, the solution the RAV beginning at the top of the triangle skips the grid points that require a "diagonal" move, instead moving down to the closer grid point. These points are picked up by the second RAV (green) once it meets the third (red), adding a relatively large cost.

We used the following configuration to run the experiment:
Spacing between grid points: 23m in latitude, 25m in longitude.
Bounding rectangle coordinates: (53.2781933786, -9.0671226391), (53.2782526061, -9.0610624609), (53.2803800539, -9.06409255)

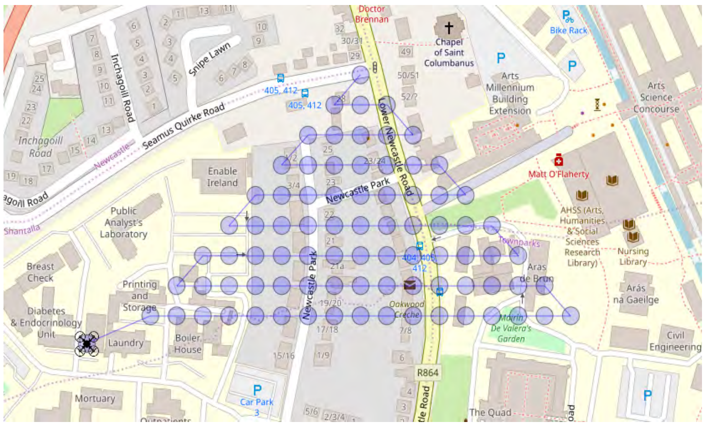
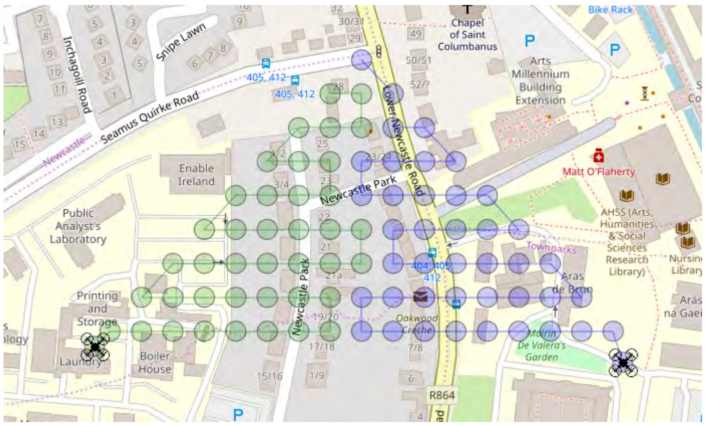
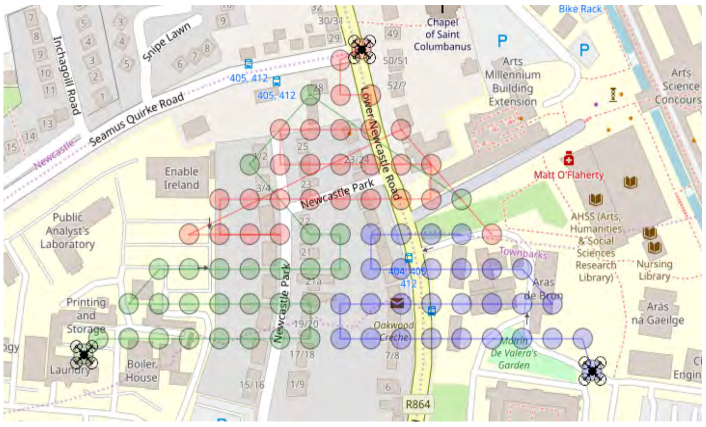
Planned RAV Routes	Route Lengths (metres)
	RAV 1 (blue): 1940.6
	RAV 1 (blue): 971.8 RAV 2 (green): 930.7
	RAV 1 (blue): 621.6 RAV 2 (green): 812.7 RAV 3 (red): 883.4

Table 4.2 Results of applying NN algorithm to a triangular region

4.2.7 Qualitative Behaviour of the NN Algorithm with a Hexagonal Grid

This experiment was again set up in a similar manner to the first, where the RAVs were dragged to starting positions that should allow the NN algorithm to take advantage of the axes of symmetry of the hexagon. In this case, we had to vary their starting locations depending on the number of RAVs used in order to encourage the generation of non-overlapping solutions. We found that the NN algorithm could find (approximately) symmetrical solutions using two or four RAVs, shown in Table 4.3.

The results show that when using two RAVs, the longest route is 2.191% shorter than half the length of the route found using just one RAV. When two RAVs are used, they move horizontally to the nearest grid location, meet in the middle of the hexagon and then move back to the outer edge. Once the reach the lower third of the hexagon, they move down diagonally and then back across horizontally. This behaviour is the same when using a single RAV. The key difference is when they move to the top third of the hexagon, they exhibit the same behaviour, whereas the single agent skips some of the "diagonal" grid points, as in the case of the triangular region using three RAVs. It must go back to visit them at a relatively large cost, since the points that it missed on each from range from both the left and right sides of the upper hexagon. This can be seen in the first figure in Table 4.3.

When using four RAVs, the longest route was 36.565% longer than $\frac{1}{4}$ of the length of the route found for a single RAV. This was mainly due to the slight imbalance in the number of points in the upper third and lower third of the hexagon and the middle third. There are 79 points in both the upper and lower third and 147 points in the middle third. This means that the two RAVs that sweep back and across the middle move to the upper and lower third to "help". This results in large jumps to finish of the final few grid points, visible in the third figure of Table 4.3.

We used the following configuration to run the experiment:
Spacing between grid points: 32m in latitude, 38m in longitude. Bounding coordinates: (53.2782526061, -9.0610624609), (53.2803800539, -9.06409255), (53.2781933786, -9.0671226391), (53.27621, -9.0671226391), (53.27402, -9.06409255), (53.27615, -9.0610624609)

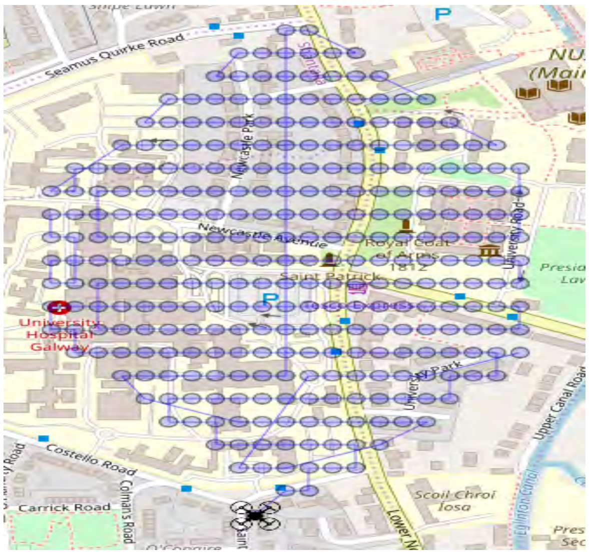
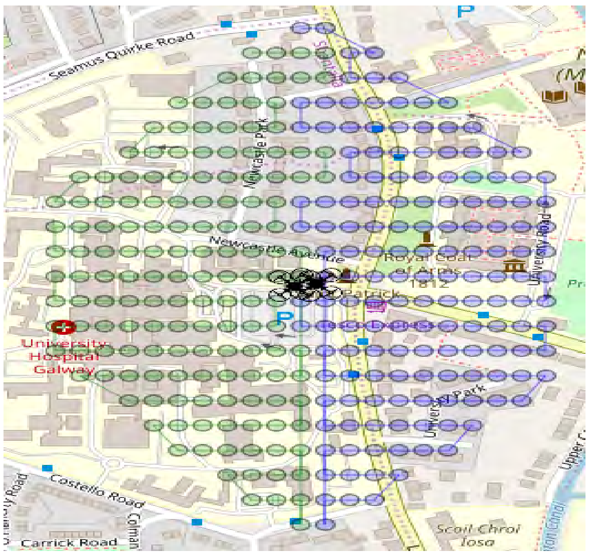
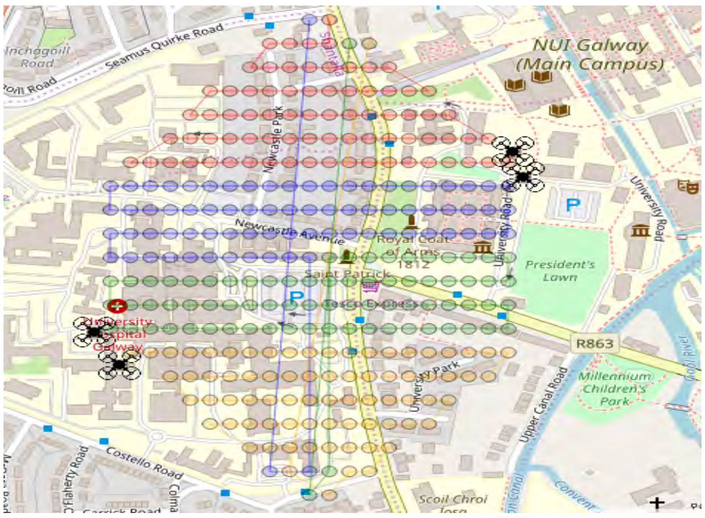
Planned RAV Routes	Route Lengths (metres)
	RAV 1 (blue): 7247.2
	RAV 1 (blue): 3587.4 RAV 2 (green): 3544.2
	RAV 1 (blue): 2435.6 RAV 2 (green): 1629.6 RAV 3 (red): 2412.0 RAV 4 (yellow): 2245.9

Table 4.3 Results of applying NN algorithm to a hexagonal region

4.2.8 Executing Agent Routes in Simulation

In order to test the algorithms developed in this chapter in a realistic scenario, we utilised the simulation environment described in Chapter 3. The AirSim plugin [81] provides a straightforward interface to be able to configure the simulation environment to run with a user specified number of RAVs, with an API to send commands to each. We successfully ran a number of tests in simulation using the UI to select the bounding area and grid spacing for the RAVs, which interfaced with the code which generated the routes for the RAVs. We then used the AirSim API to send the virtual RAVs to the generated waypoints on each route to gather images. The results of a simulation run showing the executed routes are shown in Figure 4.7

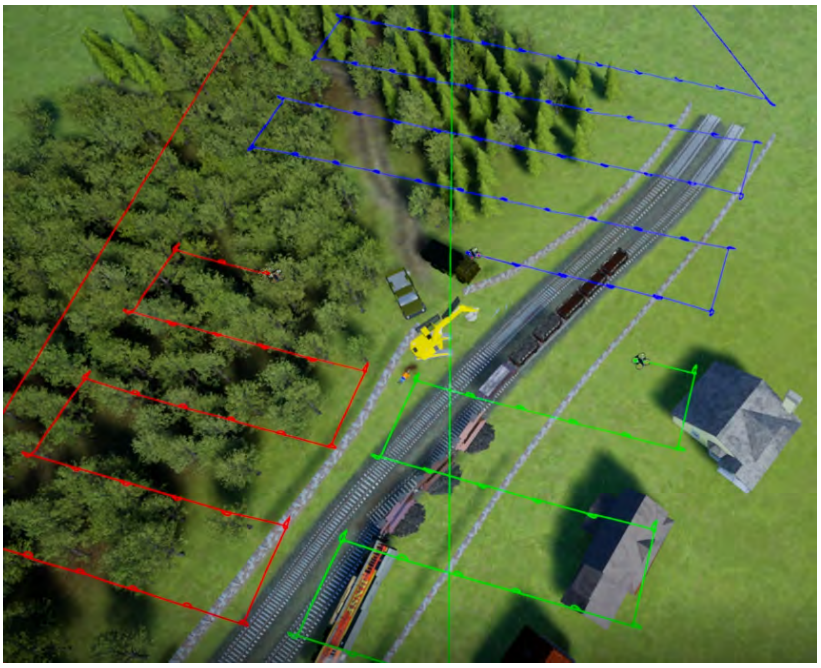


Fig. 4.7 RAVs executing their planned routes in the simulation environment

4.3 Scene Surveying With Heterogeneous Battery Constraints and Sampling Times

Once we had prototyped the NN algorithm in order to find a suitable solution to the simplified problem, we then considered the more general problem, where RAVs have heterogeneous battery constraints, sampling times and operational speeds, which is mentioned in Section 4.2. This can be described by the more general Vehicle Routing Problem (VRP). We decided to formulate the problem as a *linear program* and planned to find solutions using a linear program solver, following the approach outlined in [97]. We subsequently use terminology commonly used in convex optimisation and linear programming. We refer the reader to the text "Convex Optimization" [11] for information on linear programming and convex optimisation in general. This means that the problem's objective function and constraints must be written as linear expressions. Specifying the VRP explicitly as a linear program and then passing it to a solver is a time-consuming process, which has led to the development of a number of tools which act as wrappers for software developers to solve well-known problems without having to write excessive amounts of boiler-plate code. We chose to use Google's Apache-licensed *Operations Research*[6] (OR) repository, which contains a routing library with high-level interfaces specifically designed to allow the user to define and solve VRPs. This meant we could focus on defining the salient aspects of the problem rather than the details of how to convert the VRP into a linear program.

We began by following the example outlined in the OR tools documentation[7]. The documentation outlines the steps to solve a linear program using the repo follow the same pattern:

- Create the variables.
- Define the constraints.
- Define the objective function.
- Declare the solver — the method that implements an algorithm for finding the optimal solution.
- Invoke the solver and display the results.

4.3.1 Specifying Variables and Constraints

We first implemented a vehicle class which records the variables related to routing for each vehicle. Member variables included:

- The time taken to record data at each node
- The location of the depot, where the charging point is located
- The operational speed of the vehicle

[6]https://developers.google.com/optimization/
[7]https://developers.google.com/optimization/routing/vrp

- The estimated remaining battery life in seconds

In order to facilitate recharging, we followed the approach outlined in the documented examples and created "virtual" depot nodes, which are duplicate nodes of the recharge location and have an associated recharge time for each RAV. We force the RAVs to visit the depot to recharge once its battery level reaches below 5% by creating a cumulative variable with a range equal to the predicted time taken for the RAV battery to degrade to this level. In effect, this indicates to the solver that it must include a virtual depot node in the solution at this time.

For each RAV, we then created a lookup table of times taken to travel from one node to the next and service the second node, based on the distances between the nodes, the operational speed of the RAV and the service time. In order to communicate to the solver that it should refer to each of these lookup tables to determine the cost of adding a node to a vehicle's route, we created a *dimension* for each vehicle which refers to the lookup table. *Dimensions* are used in the OR Tools interface to represent quantities accumulated at nodes along the routes. We then specified each time dimension to contribute to the objective function by using the $setGlobalSpanCostCoefficient$ method, which notifies the solver to minimise the largest cost among all the time dimensions, which equates to minimising the longest time taken for any vehicle to complete its route. Since the solver is designed to find solutions to mixed-integer linear programs, we took the recommended approach of scaling up distance between nodes and then rounding, in order to minimise error. Once the solution had been computed, we normalised the results to their original scale. We used the default CP-SAT[8] solver to compute a solution, with the cheapest arc heuristic. The cheapest arc heuristic builds a solution by beginning with the start node and connects it to the node which produces the cheapest route segment, then extends the route by iterating on the last node added to the route.

4.3.2 Results of the Solver-Based Method

We prototyped the OR-Tools solver based method, which meant the results that we generated apply to carefully chosen regions that offer a proof-of-concept that the OR-Tools based method can provide viable solutions. We outline in Section 4.4 how we envision the prototype solution could be extended and sufficiently validated. Sample solutions computed for various configurations of the vehicles are shown in Table 4.4. For each of the solutions shown, we enforced that RAV 1 (red) and RAV 2 (green) have 1200 seconds fly time and that RAV 3 (yellow) has 2400 seconds of flying time. We assumed arbitrarily that the RAVs take 600

[8]http://google.github.io/or-tools/python/ortools/sat/python/cp_model.html

seconds to recharge to full capacity and that the RAVs will always recharge to full capacity if they return to the depot node. RAV 1, RAV 2 and RAV 3 begin with 17.5%, 33.3% and 100% of their full battery capacities respectively. RAVs start and finish at the same location.

The configurations and solutions for the individual vehicles used are given in Table 4.4

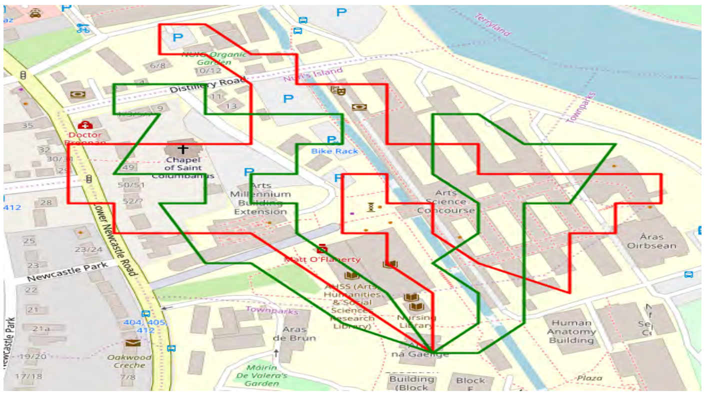
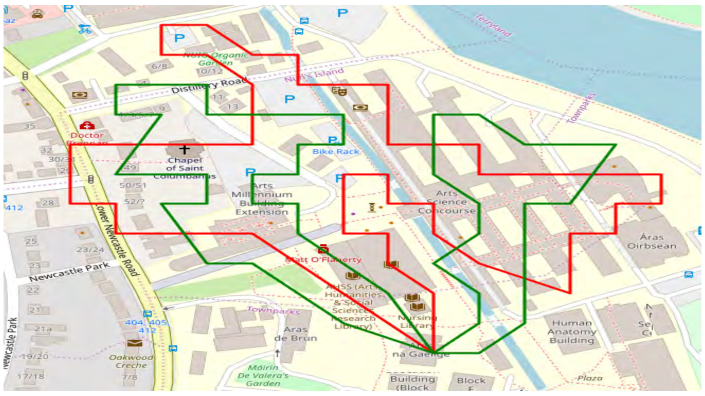
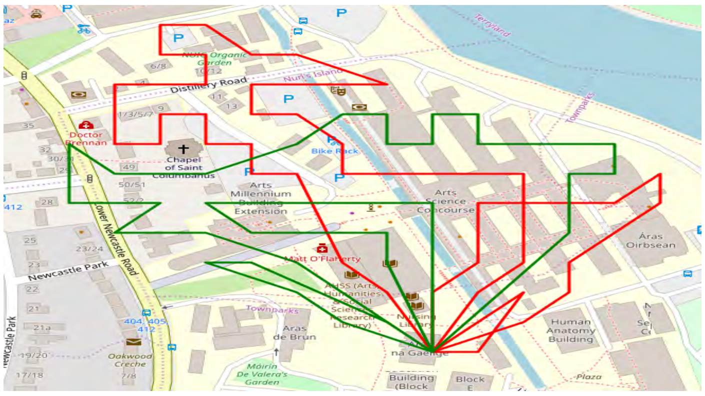
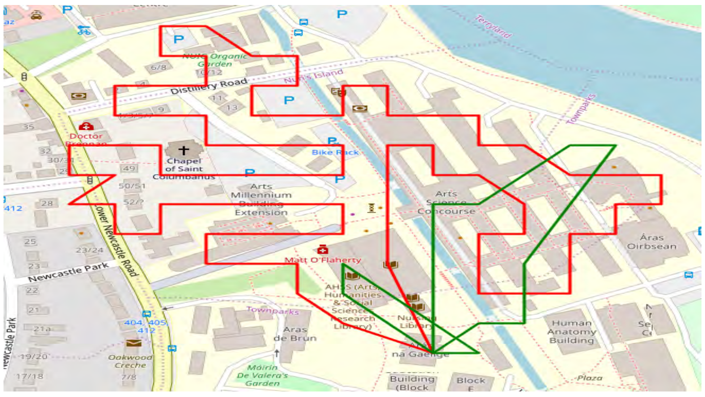
Planned RAV Routes	Solution details	RAV 1	RAV 2
	Speed	4	2
	T.T.S	0.5	0.5
	Color	Red	Green
	Dist.	1813	1813
	Time	1410	1600
	Speed	1	1
	T.T.S	0.5	0.5
	Color	Red	Green
	Dist.	1961	2141
	Time	3423	3423
	Speed	1	1
	T.T.S	2	2
	Color	Red	Green
	Dist.	2142	2150
	Time	3462	3462
	Speed	8	1
	T.T.S	1	1
	Color	Red	Green
	Dist.	2638	870
	Time	1410	1600

Table 4.4 Results of applying solution found with OR-Tools solver. T.T.S = Time to sample, Dist. = Distance

4.4 Conclusion and Future Work

In this chapter we provided the details of how a swarm of RAVs can be used to gather data in a given region described by a polygon. We first described how to discretise the region into a set of grid points. We made some minor modifications to ensure that the implementation would be able to deal with real-world usage, which in practice takes the form of using the coordinate system that GPS depends on. We then implemented a Nearest Neighbour (NN) heuristic algorithm to generate routes for the swarm of RAVs which visit each grid point in the region exactly once, which is a solution to the multiple Travelling Salesman Problem. For practical purposes in relation to ROCSAFE [3], the project that has funded this work, this offered a simple and scalable solution to the surveying problem. In order to run tests, we then used a web development framework to design a UI to interact with the grid generation code and the NN algorithm, which we used to generate some results which can be seen in Tables 4.1, 4.2 and 4.3. These experiment were purposely set up to demonstrate how regions of interest that have symmetries allow the NN algorithm to find solutions which scale well to multiple RAVs. Finally, we used the simulation environment, outlined in Chapter 3, to run further tests which offer highly realistic conditions, since the virtual RAVs run autopilot software that is used on real-world RAVs. These tests were carried out successfully and the results are illustrated in Figure 4.7.

We then used an optimisation tool to solve the more general problem of surveying with constrains such as heterogeneous RAV battery capacities, sampling times and operational speeds. We used the open-source Apache licensed Google OR-Tools repository to develop a prototype solution. This solution took orders of magnitude longer to calculate solutions compared to the NN algorithm, but offered qualitatively good solutions to the more general problem.

4.4.1 Future Work

We outline some future work that we planned but did not have the time to explore in depth. The points outlined are all taken to be specific to the Euclidean grid-based approach that was discussed throughout this chapter.

1. A major limitation to the work done in this chapter is that we assume that RAVs act deterministically. In reality, there are many sources of randomness in environments that RAVs operate in. An approach that can deal with this randomness would have a major advantage over any approach that assumes determinism.

2. Our approach is centralised, which means that RAVs do not act as independent agents. If for some reason an agent failed or communications were lost between RAVs and the central controller, the system would fail. Future work could focus on modifying the approach taken in this thesis which could allow the RAVs to re-negotiate on what work they do based on their capabilities. Re-negotiation could periodically occur in the case that a RAV is left idle when it could be participating in the task.

3. Theoretical analysis of the performing surveying tasks with a swarm of RAVs may yield some insight into how to create a scalable algorithm for certain classes of grid (e.g. some regular shapes or compositions of regular shapes).

4. The surveying task can be used to provide data to other algorithms that can provide useful tools when managing a disaster scene. For example, *structure-from-motion* is a technique for creating three-dimensional point clouds from two-dimensional images. A review of techniques used for structure-from-motion can be found in [6]. There may be potential to perform optimisations in the generation of the spacing and altitude of the grid points as well as the sequence in which the points are collected in order to generate the most accurate point cloud possible. We use the sample data collected as prior information for the stochastic target localisation problem outlined in Chapter 5

Chapter 5

Stochastic Target Localisation

This chapter outlines the approach taken to address the first research question stated in the introduction chapter:
"*Can an agent-based software system be developed to run on a system of heterogeneous autonomous aerial vehicles to aid the tasks of scene surveying and target localisation in a hazardous environment?* "
This Chapter deals specifically with the problem of *target localisation*, which we explain in precise terms in Section 5.1. The context for this problem is derived from the ROCSAFE project [3]. We first give the technical description of the problem and simplifying assumptions that we made. We then discuss the testbed that was used to prototype the solution. We used a DBN to model the agent's environment, described subsequently. Finally, we present the results of running Monte Carlo simulations to evaluate the system performance. We designed an agent-based software system to perform the search, taking a similar approach to the recent literature, noted in Section 2.3.

5.1 Problem Description

As mentioned in Chapter 1, a major problem in hazardous scene management includes localizing sources of hazardous materials and localizing potential sources of evidence. The reasons these are difficult problems, in the context of the ROCSAFE project, are:

- Hazardous materials belong to different classes of threat (chemical, biological, radiation, nuclear). If the nature of the threat is uncertain, the wrong preventative measures may be taken and personnel may be put at risk.

- Evidence localisation usually requires moving a sensor to within close proximity of the evidence. If a human is responsible for this, there is a chance that they will accidentally disturb with the evidence, possibly yielding it unusable.
- Since these scenarios are highly dangerous, the area to search may be large to avoid potentially missing important sources of evidences. This means that the process of localisation may be painstaking and time-consuming for humans.

In the ROCSAFE project, the use of RAVs is proposed to aid the execution of these tasks [3].

Spatiotemporal localisation problems have a reasonable body of literature behind them, and can be described using abstract language which allows them to be approached using a common framework, with only minor implementation details necessary to specify which instance of the problem is being addressed. The framework we have developed uses a lot of the theory outlined in Chapter 2 and builds on the literature that was reviewed there. The problem that this chapter attempts to solve can be generally described as follows:

"*Given a region of space to explore and a set of heterogeneous autonomous aerial vehicles with sensing capabilities, devise a search strategy with a search cut-off criterion which will accurately return either the locations of the targets if one or more is present, or return that no targets are present, in the shortest possible time.*"

We list some concrete versions of this problem that we envisage this approach could solve:

- *Given a system of heterogeneous autonomous aerial vehicles, some of which are equipped with radiation sensors and limited battery capacity, localize multiple sources of radioactive material in a scene.*
- *Given a system of heterogeneous autonomous aerial vehicles, some of which are equipped with high-quality cameras and limited battery capacity, localize multiple objects of a given description in a scene.*

Note that we do not give the details of how to solve these specific instances of the problem, but they merely serve as an example of envisaged real-world use cases.

5.1.1 Preliminary Assumptions

Rather than immediately attempting to tackle the full problem, we chose to initially make some simplifications in order to identify potential solution strategies that could be extended to more complex versions of the problem. At the outset, we made the following simplifying assumptions, which are in line with the assumptions made in related literature [18], [19]:

- There are either zero or one targets to be localized.
- There is only a single RAV operating in the region.
- The RAV has unlimited battery capacity.
- The region to be searched can be well approximated by a polygon.
- The sensor specificity and sensitivity are known or can be estimated for a given resolution (e.g. 1m). These are assumed to be greater than 50% for the given resolution.
- The search agent operates over a discrete spatial grid spanning the region to search, assumed to be polygonal as above, the dimensions of which are pre-determined by the sensor resolution.
- The RAV is assumed to have a GPS sensor that is accurate to beyond the sensor resolution (implying that the RAV moves to discrete grid locations without drift).
- The target is assumed to be small enough to occupy only one grid cell at a time. It is also assumed to not lie across grid cells.
- There exists a path that the RAV can follow between any two given grid cells.
- Time is discrete and the agent can choose one action to take and receives one percept on each time-step. This means that the agent can move between any two grid points in a given time-step, following the approach of previous work [18], [19], [49] and [102].

While these assumptions would not hold in real situations, they are convenient because they simplify the design of the system and subsequent analysis. In later sections in this chapter, these assumptions are relaxed and the necessary modifications for the solution strategy are discussed.

5.1.2 Experimental Testbed

Given the assumptions outlined above, rather than beginning by working on designing candidate solutions, we instead decided to set up the software that would be necessary to quickly test and evaluate a solution. This is related to the specification of the agent's environment, which is described in Section 5.2. This involved the following software components:

- A 2-Dimensional grid coordinate system which can be configured to create a grid over a polygonal region. This is outlined in greater detail in Section 4.1.
- An evidence source simulator which simulates the readings that a sensor would observe given the sensitivity and specificity of the sensor.
- A grid manager component, which manages the positions of RAVs and targets on the grid.
- A simulation manager component, which constructs the agents from their configuration files and is responsible for running the simulation using the other software components.
- Configuration files which allow the user to specify the configurations of the sensors, the agents, environment parameters and debugging/analysis files.

These components were designed in a modular fashion to distinguish the agent from its environment. The agent may have an internal representation of the grid environment in which it operates, which must be completely independent of the actual grid environment which is run in the simulation. The user can fully specify all aspects of the agent and environment (relating to the above assumptions) through configuration files.

5.2 Agent Design

When designing the agent, we adhered the approach outlined in Chapter 2 of "*Artificial Intelligence: A Modern Approach*" [77]. First, we describe four critical parts of the agent design, collectively referred to as the agent's *task environment*: the Actuators, Sensors, Environment, and Performance Element. Further discussion on specific aspects of how the agent function was implemented is then given.

5.2.1 Agent Environment

Here, we refer to conventional terms used to describe agent environments, described in [77, p. 41]. The agent's environment is *partially observable*, since it is assumed that it cannot directly observe the location of the target, but must instead use partial information related to the location of the target from noisy sensors. The outcomes of the agent's actions are assumed to be *deterministic*, meaning that if an agent chooses to move to a location, it is assumed to do so without any chance of it accidentally moving to an alternative location. The environment is *sequential*, arising from the fact that future decisions on where the agent should take a sensor reading are influenced by previous locations at which a sensor reading has been taken. The agent is assumed to operate in a 2-dimensional environment, consisting of discrete uniformly spaced grid cells overlaid onto a physical region of space. The unknown location of the target can be described by the set

$$\{x_1, x_2, ..., x_n, x_{n+1}\}$$

where x_i represents the target location being at grid cell i for $i \in \{1,2,..,n\}$, and x_{n+1} represents that the target is not present. The search status can be described by

$$\{ongoing, terminated\_x_1, terminated\_x_2, ..., terminated\_x_n, terminated\_x_{n+1}\}$$

where *ongoing* represents that the search is continuing and $terminated\_x_i$ is an absorbing terminal state that arises from the agent taking a terminal action indicating the target location, explained further in the subsequent paragraph. It is necessary to include the terminal states in the environment representation in order to specify a *performance measure* for the agent. The Cartesian product of these sets defines the environment state. For example, the environment might start in state $< x_3, ongoing >$, which represents that a target is located at grid cell 3 and the search is ongoing. $< x_3, terminated\_x_5 >$ represents that the search has terminated with the agent concluding the target is present at grid cell 5, with the target actually present at grid cell 3. The graphical model shown in Figure 5.1 depicts the conditional independence assumptions made between the hidden state variables, which is explained in Section 5.2.4.

5.2.2 Actuators and Sensors

Here we consider the actions that may be chosen to be performed by actuators and percepts that may be received by sensors. The problem of *target localisation* in the context of this chapter requires the agent to move around a discrete grid and use a calibrated sensor to record noisy readings that indicate whether the target is present or not at the location of the reading. We therefore describe the set of possible actions to be performed by the actuators by the set of all n possible grid locations that the agent can move to and take a sensor reading at, indexed by an arbitrary ordering: $\{move\_x_1, move\_x_2, ..., move\_x_n\}$. We did not restrict the agent to only move between adjacent grid cells since our use case deals with agile aerial vehicles, which can move freely between any two grid points. We add search termination actions to this set, $\{terminate\_search\_x_i\}$, for $i \in \{1,2,...,n,n+1\}$, which lead to an absorbing terminal state representing the agent's conclusion regarding whether a target is present or not, $terminated\_x_i$. To summarise, the agent may either choose a move action or a search termination action, which respectively move the agent to a new grid location at which they record a sensor reading, or conclude the search and return the most likely target location x_i.

The set of percepts that the agent will receive from its sensors come from the binary set $\{1, 0\}$, indicating the target has or has not been detected, respectively.

5.2.3 Performance Measure

The agent's performance measure maps sequences of environment states to the real numbers. Given the above definitions, we decided that environment states of the form

$$< x_i, terminated\_x_i >$$

should be of high value, as they indicate that the agent has correctly identified the location of the target in the environment. Secondary to this, sequences of environment states that take longer to end in a terminal state should be valued lower than shorter ones, reflecting our desire for the agent to terminate its search in the minimum possible amount of time. Therefore, the performance measure primarily gives high values to the agent when it correctly identifies the location of the target or correctly concludes that the target is not present, with a secondary ordering on value determined by the time taken to come to a conclusion. The actual value of the function only needs to adhere to this ordering, but we arbitrarily defined it as:

$$PerformanceMeasure(state_1, ..., state_t) = \begin{cases} \frac{1}{t} & \text{if } state_t = < x_i, terminated\_x_i > \\ -1 & \text{otherwise.} \end{cases}$$

It is worth noting that this performance measure provides goal states for the agent:

$$< x_i, terminated\_x_i >$$

5.2.4 Stochastic Environment Model

Model-based agents require a concrete implementation of a model of their world, in order to maintain their internal state [77, p. 50]. The agent's internal state can be thought of as its opinion of what state the world might be in, and its model describes how it believes the world changes over time [77]. Chapter 2 outlines the background behind potential models of the stochastic world, including both how internal state can be represented and how internal state can be updated.

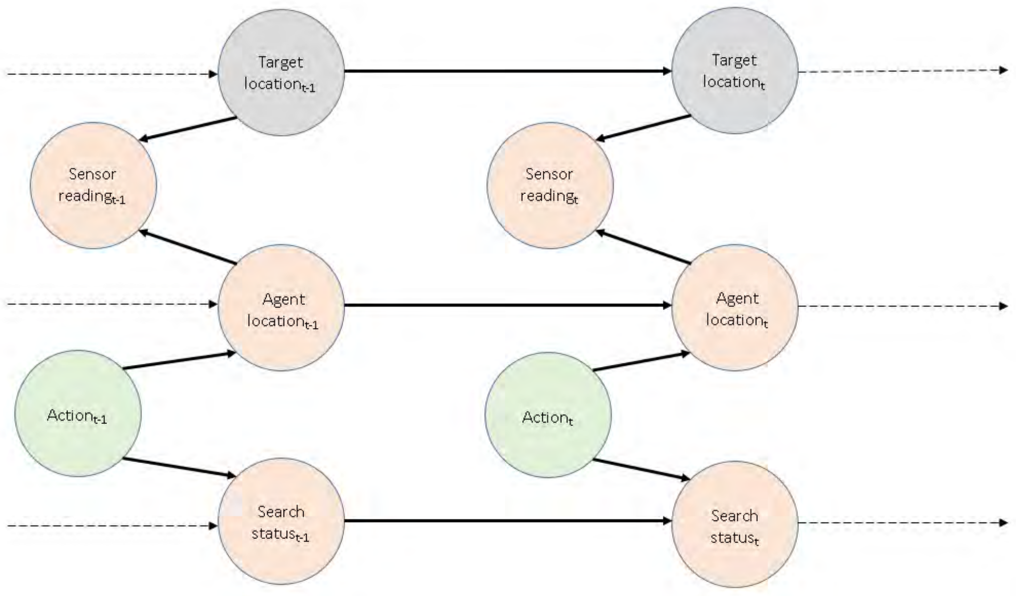


Fig. 5.1 First Version of the DBN representing the agent world model

Hidden Markov Models and Dynamic Bayesian Networks were identified as suitable models, due to the fact that they provide a succinct and flexible representation of hidden stochastic world state, as well as efficient online state estimation updating algorithms. We chose to use a Dynamic Bayesian Network (DBN), due to the fact that it can use conditional independence relations between variables to reduce the number of probabilities needed to be calculated to perform accurate inference [45, p. 63]. DBNs also facilitate the incorporation of extra variables with arbitrary conditional independence relations. This was desirable, since we plan future work to extend the model to include extra variables, such as battery level, once a basic implementation was shown to work as intended.

The 2-Time Slice DBN shown in Figure 5.1 describes the agent's world model. A first-order Markov assumption is made, where the current state only depends on the state directly preceding it. Grey coloured variables are *hidden state variables* and are assumed to not be directly observable. Green coloured variables are *control actions* taken by the agent and the peach coloured variables are *evidence variables*, which are assumed to be observable and depend on the hidden world state. A number of conditional probabilities are specified here in order to describe the factored joint distribution of the world model fully. The full tables are omitted as they are sparse. For brevity, we use the abbreviation Loc for Location.

$$p(SensorReading_t|AgentLoc_t, TargetLoc_t) = \begin{cases} \alpha & \text{if } SensorReading_t = 1 \text{ and } TargetLoc_t \neq AgentLoc_t \\ 1-\beta & \text{if } SensorReading_t = 1 \text{ and } TargetLoc_t = AgentLoc_t \\ \beta & \text{if } SensorReading_t = 0 \text{ and } TargetLoc_t = AgentLoc_t \\ 1-\alpha & \text{if } SensorReading_t = 0 \text{ and } TargetLoc_t \neq AgentLoc_t \end{cases} \tag{5.1}$$

Conditional Probability Distribution for the *Evidence* Variable.

$$p(TargetLoc_t|TargetLoc_{t-1}) = \begin{cases} 1 & \text{if } TargetLoc_t = TargetLoc_{t-1} \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

Conditional Probability Distribution for the *TargetLocation* Variable

$$p(SearchStatus_t|SearchStatus_{t-1}, Action_{t-1}) = \tag{5.3}$$

$$\begin{cases} 1 & \text{if } SearchStatus_t = terminated\_x_i \text{ and } SearchStatus_{t-1} = terminated\_x_i \\ 1 & \text{if } Action_t = terminate\_x_i \text{ and } SearchStatus_{t-1} = ongoing \text{ and } SearchStatus_t = terminated\_x_i \\ 1 & \text{if } Action_t = move\_x_i \text{ and } SearchStatus_{t-1} = ongoing \text{ and } SearchStatus_t = ongoing \\ 0 & \text{otherwise.} \end{cases} \tag{5.4}$$

Conditional Probability Distribution for the *SearchStatus* Variable

$$p(AgentLoc_t|AgentLoc_{t-1}, Action_t) = \begin{cases} 1 & \text{if } Action_t = move\_x_i \text{ and } AgentLoc_t = x_i \\ 1 & \text{if } Action_t = terminate\_x_i \text{ and } AgentLoc_t = AgentLoc_{t-1} \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

Conditional Probability Distribution for the *AgentLocation* Variable

The semantics of the equations listed above is given here:

1. Equation 5.1 uses two parameters, α and β, to represent the probability of making a false positive and false negative sensor reading, respectively. These are assumed to have been calculated by using pre-calibrated values of the sensor. This model is does not stipulate any restrictions on the sensor other than that it must return a reading indicating that the target is present or not.

2. Equation 5.2 simply states that the location of the target does not change, even though it is hidden. This could be modified to allow for mobile target detection in the case of a non-stationary target by introducing non-unity probabilities in a transition matrix for different values of $TargetLoc_t$ and $TargetLoc_{t-1}$.

3. The first line of Equation 5.3 states that once the agent terminates the search, it remains over. The second line states that if the agent requests an ongoing search to terminate, then it does so deterministically. The third line states that if the agent chooses to move in an ongoing search, then the search remains ongoing.

4. Equation 5.5 states that the agent moves to new locations deterministically. It also indicates that should the agent choose to terminate the search, then the environment enters a terminal state, indicating that the search is over.

Note that despite the fact that some hidden state variables cannot be observed directly, it is still possible to infer their value exactly based on their starting state. For example, the position of the agent is a deterministic function of its actions and previous position.

5.2.5 Estimated State

Since the agent operates in a partially observable environment, the agent's internal state representation of the environment needs to take into account the fact that the current environment state is not certain. For this reason, a probability distribution over possible environment states is used by the agent to internally represent the environment state. The internal agent state is described mathematically by

$$p(x_t | e_{1:t}, u_{1:t})$$

which is the distribution of possible world states given all evidence and control actions up to the current time step, where x_t, e_t and u_t follow the conventions of being the hidden state variables, the evidence variables and the control action variables respectively. The agent updates this state using the world model specified in the preceding chapter, using the filtering algorithm explained in Section 2.4.3. There are a couple of noteworthy points:

- A distribution that has a single sharp peak would indicate that the agent believes that the target is at a specific location with high confidence. This is clearly preferred to a flat, uniform distribution representing uncertainty in relation to where the target might be.
- The only true source of uncertainty in the world state is introduced by the sensor. Therefore, analysis of this state representation in relation to the sensor model should provide insight into how the system performs.

In order to update the estimated state of the agent, the below equations are used, which are described in their general form in Section 2.4.3. For brevity x_t denotes the hidden state variables, e_t denotes the evidence variables and u_t denotes the control action taken by the agent. The equations only describe the estimated state update for move actions, since if the agent terminates the search a terminal state is deterministically entered. The $SearchStatus$ variable is omitted from the equations since it only effects the equations if the $Action$ variable is to terminate the search.

$$p(AgentLoc_t = x, TargetLoc_t = y | e_{1:t}, u_{1:t}) =$$
$$\begin{cases} \eta \alpha p(AgentLoc_{t-1} = x, TargetLoc_t = y | e_{1:t-1}, u_{1:t-1}) \text{ if } e_t \text{ is a positive reading and } AgentLoc_t \neq TargetLoc_t \\ \eta \beta p(X_{t-1} = x_t | e_{1:t-1}, u_{1:t-1}) \text{ if } e_t \text{ is a negative reading and } Agent_loc_t = TargetLoc_t \\ \eta (1-\alpha) p(X_{t-1} = x_t | e_{1:t-1}, u_{1:t-1}) \text{ if } e_t \text{ is a negative reading and } Agent_loc_t \neq TargetLoc_t \\ \eta (1-\beta) p(X_{t-1} = x_t | e_{1:t-1}, u_{1:t-1}) \text{ if } e_t \text{ is a positive reading and } Agent_loc_t = TargetLoc_t \end{cases} \tag{5.6}$$

η is a normalising factor, α is the probability of a sensor false positive detection and β is the probability of a sensor false negative detection.

5.2.6 Action Selection Strategies

At each discrete time step the agent may either choose to terminate the search or move to a new location to record an observation. The agent may choose which of these actions to perform based on its location, the search status and its internal representation of the environment, which are outlined in the preceding section. First we discuss how move actions are chosen, if the agent decides that the search should not be terminated at the current time step. We began by implementing some of the recommended basic search strategies outlined in the related works by Chung and Burdick [18] and Waharte and Trigoni [101]. These search methods are devised in order to optimize the agent's performance measure, which is set out in Section 5.2.3. The results of applying these strategies are discussed in Section 5.4.5.

Random Search This method serves as a baseline against which similar strategies can be compared. The agent simply chooses the next grid location to explore randomly from all possible grid locations. The expected number of moves needed to take a reading at all possible grid cells is given by the solution to the *coupon collectors problem* [30], nH_n, where H_n is the n_{th} harmonic number and n is the total number of grid cells, which gives a good idea of the expected amount of time that will be taken to conclude the search.

ε-Greedy Search A simple greedy search was implemented next, where with probability $1-\varepsilon$ the agent chooses to visit the grid cell with the highest estimated probability of containing the target in a localised region around the agent:

$$Action_t = \underset{NewLoc \in N(AgentLoc)}{\arg\max} \; p(TargetLoc = NewLoc, SearchStatus, AgentLocation | e_{1:t}, u_{1:t})$$

and with probability ε, the agent moves to a random grid cell in a localised region around the agent. $N(AgentLoc)$ is a function that returns a neighbourhood of locations around the agent's location and can be calibrated to trade off the cost of saccading between grid cells that may be far from each other against the cost of limiting the agents range to a narrow and possibly "cold" region. This method is designed bearing in mind that there may be motivation to explore some areas before others based on prior knowledge.

Saccadic Search This was proposed in [18] and is a special case of ε-greedy search. The idea is to mimic the behaviour of the human eye when looking at an image, whereby it "*saccades*" from one salient feature to another. The consequence is that the most promising cells are explored at each time step, which means that the agent can be drawn to travel large distances in order to explore a peak in the spatial distribution given by the occupancy grid. Further details are given in [18].

Sweep Search This strategy sweeps the region of interest systematically, aiming to take a reading at each grid cell an equal number of times while minimizing the total distance travelled. It then traverses this same path again in reverse order. This requires planning a trajectory in advance, which is often referred to in related literature as the *complete coverage path*. Since the region to be swept is assumed to be a grid, where adjacent points are assumed to be equidistant, a number of heuristic solutions were available for use from Section 4.2.

5.2.7 Search Termination

At each discrete time-step, the agent can either choose to move to a new grid location to record a sensor measurement or it can decide to terminate the search based on its estimated state of the environment. There is a trade-off in terminating the search early, which means that less time and resources are spent on continuing the search, versus the possibility of drawing misinformed conclusions from the search due to a lack of information. For example, if the agent receives a series of false positive readings at a given location, it could mistakenly choose to conclude that the target is present at a given location rather than sample further to gain confidence that it has correctly found the location of the target. Following this line of thinking, it is clear that a strategy needs to be devised to minimize the probability of drawing false conclusions, which is described in the performance measure set out in Section 5.2.3.

Pollock outlines three commonly used criteria that can be used to make a decision whether to terminate the search or not: the *Bayes Criterion*, which minimizes the expected cost per decision, the *Minimax Criterion*, which chooses a decision which minimises the maximum expected cost and the *Neyman-Pearson Criterion*, which uses a likelihood ratio test to determine the optimal decision [67]. These tests are proposed in the context of a target that is definitely present in the search region, but can be extended to incorporate the decision problem where the target may or may not be present. In addition, related work by Chung and Burdick has addressed this problem using methods that use heuristics to make a decision whether to terminate the search or not [18].

We ultimately choose to implement the Sequential Probability Ratio Test (SPRT), which is a hypothesis-testing framework developed by Wald and Wolfowitz to optimally deal with sequential decision problems, as opposed to traditional frameworks which assume that all the necessary data has been gathered prior to analysis [105]. The background knowledge behind the SPRT can be found in Section 2.4.4. An algorithm is also provided on how to perform this test in practice. We applied the SPRT algorithm to our problem to provide a search termination criteria using the following quantities:

H_0 : The null hypothesis, the target is not present in the search region

H_1 : The alternative hypothesis, the target is present in the search region

α : The maximum probability of making a type *I* error.

β : The maximum probability of making a type *II* error.

p_{0t} : The probability of observing the data $(e_1, ..., e_t)$ under the assumption of H_0

$$p_{0t} = \sum_{loc=1}^{n} p(TargetLoc_t = loc, AgentLoc_t, SearchStatus_t | e_{1:t}, u_{1:t})$$

p_{1t} : The likelihood of observing the data $(e_1, ..., e_t)$ under the assumption of H_1

$$p_{1t} = p(TargetLoc_t = n+1, AgentLoc_t, SearchStatus_t | e_{1:t}, u_{1:t})$$

p_{0t} and p_{1t} are calculated by using the evidence likelihood algorithm, which is described in detail in Section 2.4.3. The SPRT algorithm was then used at each time-step to decide whether to

1. Terminate the search accepting H_0, that the target is not present in the search region.
2. Terminate the search accepting H_1, that the target is present in the search region. In this case, the target location with the highest estimated probability is returned as the target location.
3. Continue the search, using the Action Selection Strategy described in Section 5.2.6.

In practice, we often took logarithmic transforms of the values used in the SPRT, to simplify some calculations and to be able to look at plots that are less heavily skewed and simpler to interpret.

5.2.8 Analysis of Search Termination Criteria

α and β, the two parameters that the user needs to specify to perform the SPRT, need to be chosen carefully and depend on the context of the search. As in the standard hypothesis testing context, it is important to consider the significance level and power of the test to ensure that they reflect the severity of drawing an incorrect conclusion [37]. They can also help to perform analysis on how well the agent could perform, since setting a high threshold means that the agent may have to take a minimum number of samples at the correct target location in order to choose H_0 or H_1.

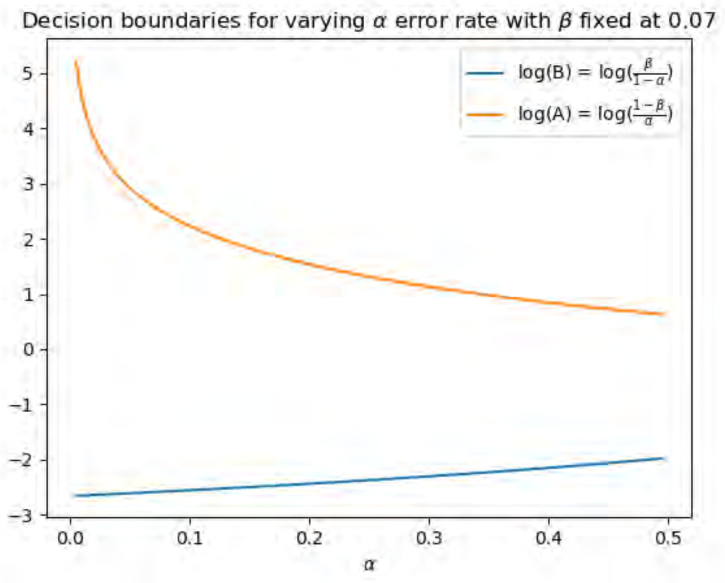
Figure 5.3 shows how A and B vary as functions of the parameters α and β on a log scale Tables of the values of A and B for varying for varying Type I rates (α) and Type II (β) error rates may be referred to in Appendix A. If the log-likelihood ratio of the data lies in between the red and blue surfaces on the graph, another sample is taken. A projection of this plot is shown in Figure 5.2:

(a) shows a plot of how varying the Type I error rate α for a fixed Type II error rate $\beta = 0.07$ affects the decision boundaries defined by A and B.

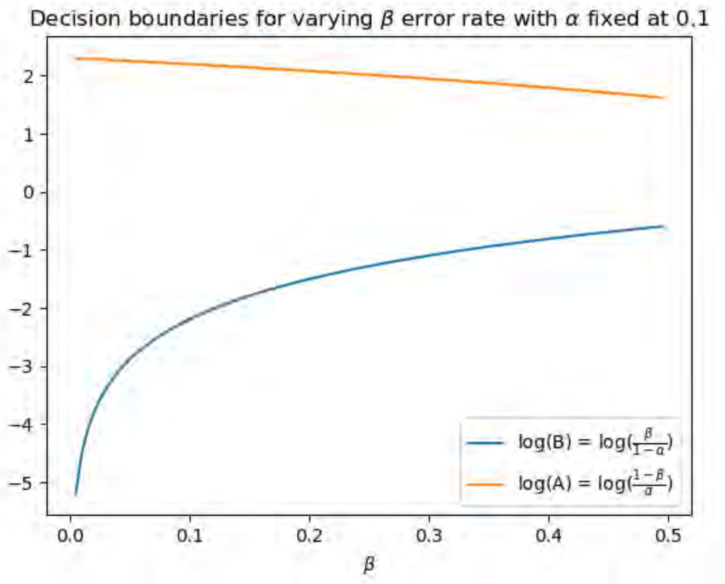
(b) shows a plot of how varying the Type II error rate β for a fixed Type I error rate $\alpha = 0.1$ affects the decision boundaries defined by A and B.

Figures 5.3 and 5.2 reflect the intuition behind the decision boundaries. The lower the probability of making either a Type I or Type II error, the further apart the decision boundary becomes, meaning the likelihood ratio must be very far apart from 1, which appears as 0 on the log scale. A likelihood ratio of 1 indicates maximum uncertainty. It is also possible to see that the surfaces meet when $\alpha = 0.5$, $\beta = 0.5$, which reflects the fact that immediately terminating the search will give a 0.5 probability of returning a false positive or false negative.

Figure 5.4 shows the regions in which the SPRT will elect to take another sample (the green shaded region), to accept H_0, the upper red shaded region, and to accept H_1, the lower red shaded region. The blue line shows the log-likelihood ratio of $\frac{(e_{1:t}|H_0)}{p(e_{1:t}|H_1)}$ as a function of the agent belief that the target is present in the search region. Note that since the values of α and β are significantly smaller in Figure 5.4 (b) than in Figure 5.4 (a), the acceptance region for H_0 and H_1 are significantly smaller in (b) than in (a).



(a) Varying TI error rate for a fixed TII error rate



(b) Varying TII error rate and fixed TI error rate

Fig. 5.2 Upper and lower values of A and B for varying error rates. Values are shown on a log scale

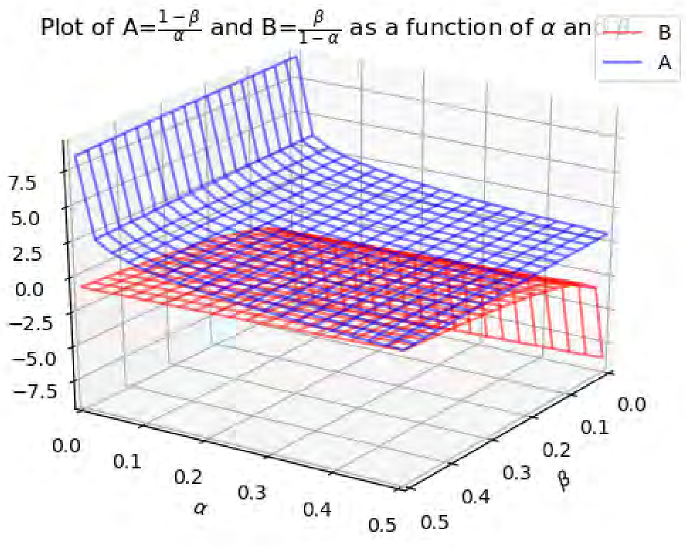
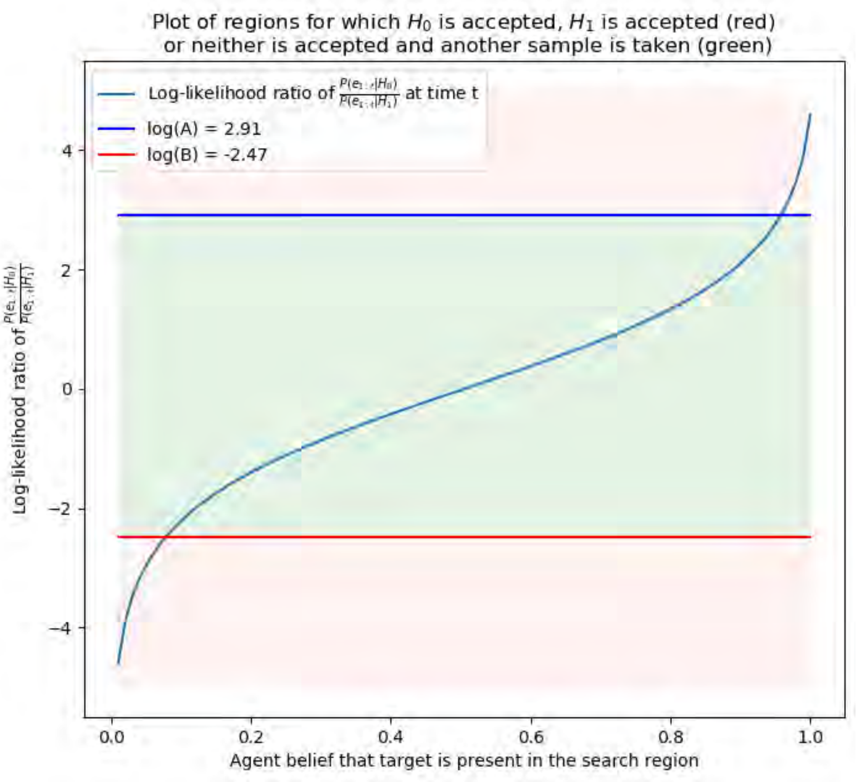
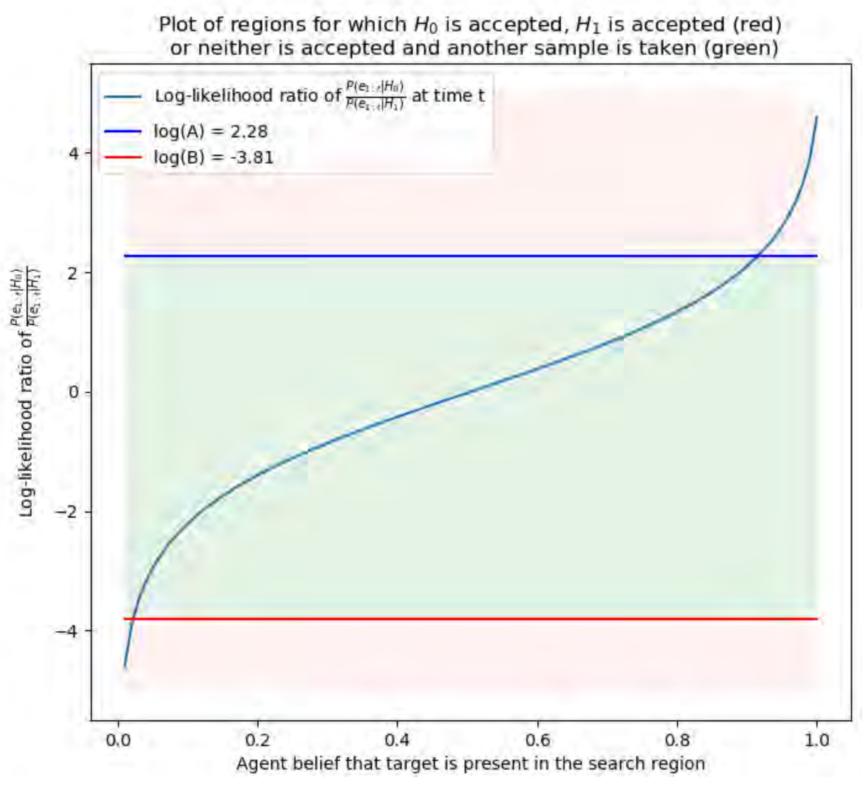


Fig. 5.3 The upper and lower SPRT bounds for acceptance and rejection of H_0, as functions of the significance and power of the test. Values are shown on a log scale.



(a) SPRT cut-off regions for $\alpha = 0.05$, $\beta = 0.08$

Fig. 5.4



(b) SPRT cut-off regions for $\alpha = 0.01$, $\beta = 0.02$

Fig. 5.4 The cut-off regions of the SPRT for (a) $\alpha = 0.05$, $\beta = 0.08$ and (b) $\alpha = 0.01$, $\beta = 0.02$ as a function of agent belief in whether target is present or not. Values are shown on a log scale.

5.2.9 Agent Function Design

Putting together the components outlined so far in this chapter, we end up with an agent function that is shown in Figure 5.5.

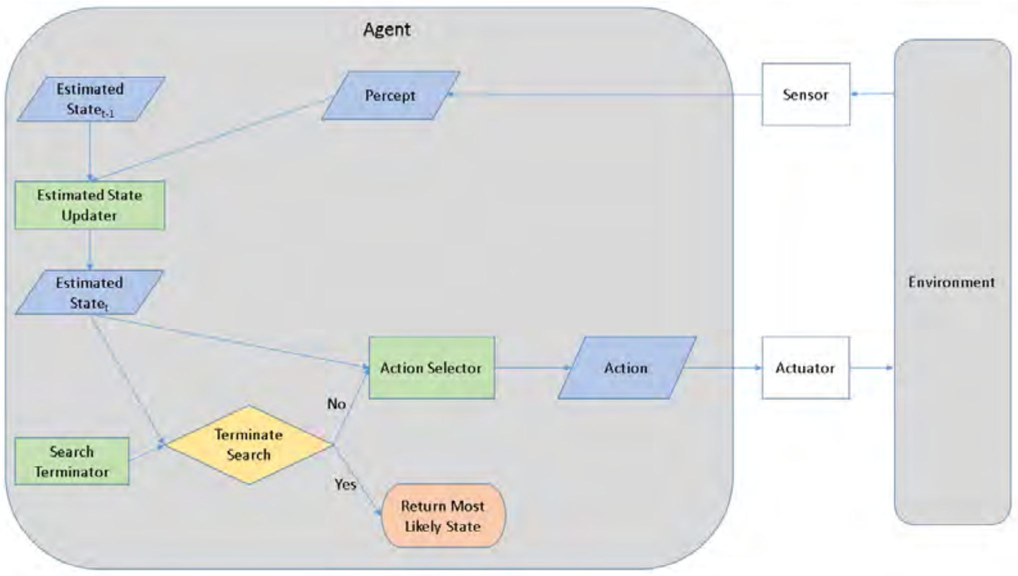


Fig. 5.5 The structure of the agent function.

The diagram outlines how the agent interacts with its environment and abstracts away technical details such as the search status. This is a concrete version of the model-based reflex agent outlined in [77, p. 51]. The estimated state update rules are applied as outlined in the previous sections in this chapter. Algorithm 6 shows the algorithm which describes the agent function.

Algorithm 6 Single Target Localisation Algorithm

Input:

$initial\_estimated\_state = P(x_0|e_0,u_0)$: The initial distribution of the estimated state
$select\_action$: Returns an action given an estimated state
$estimated\_state\_updater$: Returns the updated estimated state given the current estimated state, a percept and an action
$SPRT$: An object calibrated with a specified Type I, Type II error which has a method to implement the SPRT and an attribute which returns the result of the application of the SPRT.

Output:

The grid cell containing the source, x_i for $i \in \{1,...,N\}$, or x_{N+1}, indicating the source is not present

```
1: $estimated\_state \leftarrow initial\_estimated\_state$
2: while not $SPRT.should\_terminate\_search(estimated\_state)$ do
3:     $action \leftarrow select\_action(estimated\_state)$
4:     $perform\_action(action)$
5:     $percept \leftarrow get\_percept()$
6:     $estimated\_state \leftarrow estimated\_state\_updater(estimated\_state, percept, action)$
7: end while
8: return $\arg\max_{x_i} p(x_i|e_{1:t}, u_{1:t}) = \arg\max estimated\_state$
```

5.3 Extensions to the Basic Agent

The agent design outlined in the preceding sections of this chapter addresses the problem outlined in Section 5.1, following the assumptions outlined in Section 5.1.1. As mentioned in Section 5.1.1, some of these assumptions are not realistic, which we now address in this section.

5.3.1 Localising Multiple Targets

The first assumption that we address is that there are either zero or one targets to be localised. In many applications, this is an unrealistic assumption. For example, the ROCSAFE project which motivates this work, aims to apply object detection algorithms to aerial images in order to avoid the need to send a crime scene investigator into a hazardous scenario in order to verify the presence of a potential source of evidence [3]. An assumption that is more realistic is that there are a maximum of K unknown sources present in the search region, which addresses the general case for the ROCSAFE project.

The naive approach would be to introduce new random variables which represent the possible location of each target. Using the abbreviation T for target and Loc for location, this gives a set of random variables, whose joint distribution replaces that of TargetLocation:

$$\{TOneLoc, TTwoLoc, TThreeLoc, ..., TKLoc\}$$

Each of these random variables has support $\{x_1, x_2, ..., x_N, x_{N+1}\}$, where, as before, x_i represented the target location being at grid cell i for $i \in \{1, 2, ..., N\}$ and x_{N+1} represented that the target is not present. Assuming that targets cannot occupy the same grid cell, the update rule can be modified to assign the probability of the estimated state to be zero in the case that two target locations coincide. For example:

$$p(TOneLoc = x, TTwo = x, ..., TKLoc, SearchStatus, AgentLocation | SensorReading_{1:t}, Action_{1:t}) = 0$$

Otherwise the update rule would be slightly modified to reflect the possibility of multiple targets:

$$p(SensorReading_t | AgentLoc_t, TOneLoc_t, ..., TKLoc_t) =$$
$$\begin{cases} \alpha \quad \text{if } SensorReading_t = 1 \text{ and } AgentLoc_t \neq TOneLoc_t \text{ and } AgentLoc_t \neq TTwoLoc_t \text{ and ... and } AgentLoc_t \neq TKLoc_t \\ 1-\beta \quad \text{if } SensorReading_t = 1 \text{ and } \{TOneLoc_t = AgentLoc_t \text{ or } TTwoLoc_t = AgentLoc_t \text{ or ... or } TKLoc_t = AgentLoc_t\} \\ \beta \quad \text{if } SensorReading_t = 0 \text{ and } \{TOneLoc_t = AgentLoc_t \text{ or } TTwoLoc_t = AgentLoc_t \text{ or ... or } TKLoc_t = AgentLoc_t\} \\ 1-\alpha \quad \text{if } SensorReading_t = 0 \text{ and } TOneLoc_t \neq AgentLoc_t \text{ and } TTwoLoc_t \neq AgentLoc_t \text{ and ... and } TKLoc_t \neq AgentLoc_t \end{cases} \tag{5.7}$$

Modifications to update rule for joint distribution accounting for multiple targets.

The main issue that arises using this approach is that there is an exponential increase the dimensionality of the hidden state for each new target added. This means that maintaining an estimate of the state quickly becomes infeasible, since the run time of the forward algorithm detailed in Section 2.4.3 is dependent on the square of the size of the joint distribution of the hidden state variables, X_t and the memory requirements for maintaining the estimated state also depends on the dimensionality of the joint distribution of the hidden state. Therefore this naive approach is not feasible to implement in practice and other methods must be considered.

Section 2.3 of the literature review outlines a number of techniques that have been applied by other researchers in this domain in order to deal with the case of multiple targets which avoids the problem of creating an exponential increase in the size of the state. Waharte et al. [102] describe an approach commonly used with occupancy grids, which is that every grid cell may or may not contain a target, independent of whether or not all other grid cells contain a target. When the location of the agent is known, this technique is often referred to as *mapping*. This is described in the general case by Thrun et al. in [96, P. 284]. Rather than maintaining a distribution representing the location of the single target, $TargetLocation$, as was the case in the work outlined up to this point in this chapter, we could create new binary random variables representing a target being present or not in each grid cell, denoted m_i for $i \in \{1,...,N\}$, where N is equal to the number of grid cells. m is the joint distribution of these random variables. The DBN that would correspond to this is shown in Figure 5.6. The agent maintains the estimate of the map:

$$p(m|SensorReading_{1:t}, Action_{1:t}) = \prod_i p(m_i|SensorReading_{1:t}, Action_{1:t})$$

The update rule simply updates only the estimated state of the grid cell at which the reading was taken, as outlined in [102].

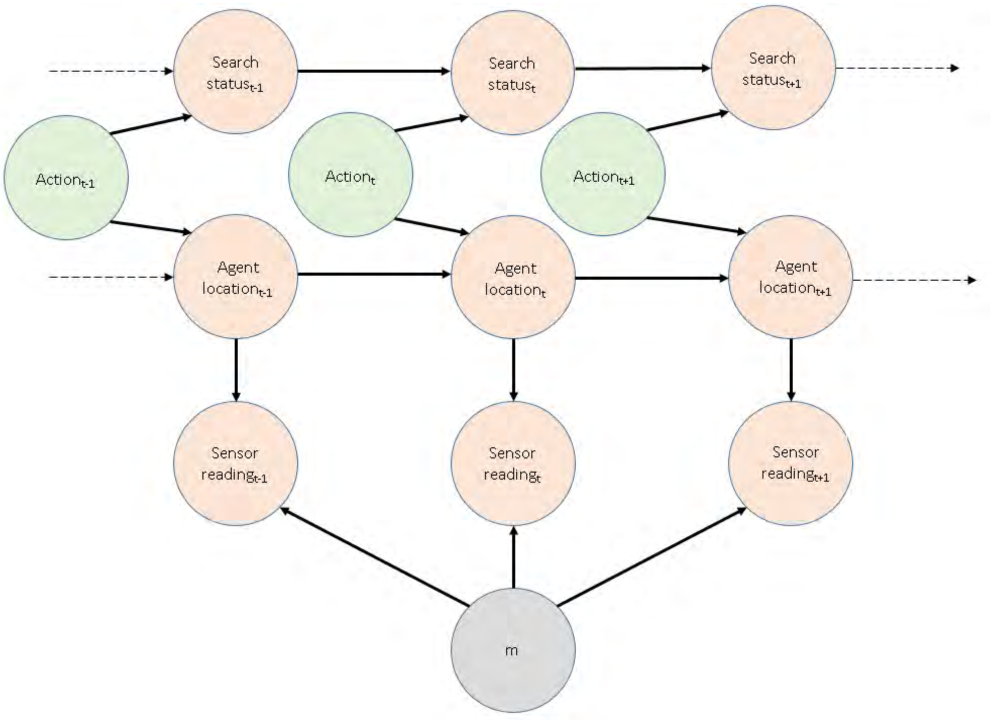


Fig. 5.6 The DBN which takes into account that there could be a target in each of the grid cells

The major drawback of this approach is that there is no clear way to apply the SPRT, which meant that our implementation for this approach required a new method for terminating the search. Instead, we chose to use a simple extension to the approach proposed in this chapter in the sections prior to this one. We chose to run a sequential search, which locates sources one-by-one, using information from the previous search to begin the subsequent search for the next target. This uses the advantages of using the sequential search procedure with the SPRT while avoiding the dimensionality problems that arise with introducing extra targets that would arise from a simultaneously searching for multiple targets. The search procedure we propose for multiple sources is given in Algorithm 7.

Algorithm 7 Multiple Target Localisation Algorithm

```
Input:
    $initial\_est\_state = p(x_0|e_0, u_0) = bel(x_0)$ :   The initial estimated state distribution
    $K$ :   The maximum number of targets to be localised.
    $find\_next\_target$ :   The procedure for locating targets described in Algorithm 6
Output:
    A set of target locations, of size $\leq$ K

 1: target_locations ← empty array
 2: estimated_state ← initial_est_state
 3: while target_locations.length ≤ K do
 4:     next_target_location ← find_next_target(estimated_state)
 5:     if next_target is $x_{N+1}$ then
 6:         break
 7:     else
 8:         target_locations.append(next_target_location)
 9:         Update the estimated_state probability where (target_location =
            next_target_location) to zero
10:         Normalize the estimated state probabilities to sum to one
11:     end if
12: end while
13: return target_locations
```

The algorithm maintains a list of located targets in the variable *target_locations* and runs the *find_next_target* procedure until the SPRT returns that there is no target present using the most recent state estimate. Lines 9 and 10 effectively restart the search procedure for a single source with the probability at the location where the last target was located set to zero. This means that it will continue to be zero, due to the multiplicative update formula, which effectively removes it from the process of localising subsequent targets. This allows us to extend the framework developed for the search procedure so far, which greatly simplifies the process of writing the implementation code to perform the search and the subsequent analysis of results.

5.3.2 Using Multiple Agents

The next assumption that we address is that only a single agent will be used to execute the search. Using a single agent which has complex mechanisms for manoeuvring and sensing poses a significant risk due to the very real chance of malfunction. For this reason, much recent research been focused on multi-agent systems which can offer robustness, redundancy and scalability where a single-agent system cannot [92]. In the context of the target localisation problem, it is intuitive to conclude that the lower bound of the search time is inversely proportional to the number of agents carrying out the search, which is further motivation to include mechanisms for using multiple agents for carrying out the search.

The ROCSAFE project [3] motivating this work assumes that the RAVs used to carry out the search procedure may lose connectivity with a centralised controller from time to time, which means that the assumption that they must be able to act autonomously without a centralised decision mechanism must be made. There are many well-researched paradigms that multi-agent systems fall into, which allow high-level goals to be achieved without relying on a centralised source of control. Due to time constraints, we did not investigate comprehensively negotiation strategies between agents or ways of inducing emergent system behaviours, but this could be a topic of future research for this area.

We assumed that the RAVs may communicate pair-wise with each other within a certain radius, with the probability of a successful transmission inversely proportional to the distance between the RAVs. We also allowed that the sensors on the RAVs may have different probabilities of reporting false positives and false negatives, which reflects that some sensors used may be more accurate than others, due to cost, size and other realistic constraints.

Since we do not have an associated cost with communication, we implemented a strategy whereby agents will attempt to communicate with all other agents if possible, after they have sensed at each time-step. As in [19] and [102], each agent keeps a local list of sensor readings made by itself and other agents. When agents communicate, the agent updates its local state estimate using Algorithm 1 with the previously unseen sensor readings communicated by the other agent and the calibrated false positive and false negative rate for the other agent's sensor. This takes advantage of the fact that the Bayesian update rule for the estimated state of each agent is composed of multiplicative terms, for which the order of multiplication does not matter. Information related to the locations of targets will naturally propagate around the system as agents come within the communication radius of each other.

5.4 Simulation Results

The results presented in this section were generated by running Monte Carlo simulations of the search procedure, since finding a closed-form solution to the mean **T**ime **T**o **D**ecision (TTD) is not readily available in the general case [20]. We simulated the grid, the agents and the targets in order to evaluate the performance of the system. We present statistics related to the Monte Carlo simulations, which reveal how modifying parameters of the search procedure affect the outcome. For each set of parameters in tables 5.2, 5.3, 5.4, 5.5, 5.6 and 5.7 we ran 5000 simulations, which finish when the agent (or agents) terminates the search. The parameters that we vary in the simulations are shown in Table 5.1:

Parameter	Configuration 1	Configuration 2	Configuration 3
Initial Distribution	Gaussian	Uniform	-
Initial Cumulative Probability	0.25	0.5	0.75
Sensor Model Parameters	α=0.05, $\beta = 0.02$	α=0.2, $\beta = 0.15$	α=0.4, $\beta = 0.4$
# of Targets Present	1	2	3
# of Agents Used	1	2	3

Table 5.1 Parameters varied for each simulation run

The meaning of each of these parameters is outlined as follows:

1. The initial belief distribution of each agent describes the distribution of $p(x_0|e_0, u_0)$, which corresponds to the probability distribution describing its initial belief of the location of the target, prior to gathering any evidence. This may come from prior information about the scene. A Gaussian distribution is peaked, meaning that prior information suggests that some grid cells are more likely to contain the target than others. This can be seen in Figure 5.7b. A Uniform distribution is not peaked, reflecting no prior information related to the location of the target. This can be seen in Figure 5.7a.

2. The initial cumulative belief that the target is present in the region is the agent's belief in whether the target is present in the region, prior to gathering any evidence. It is the cumulative sum of the probabilities of the target presence in each of the grid cells. An initial cumulative belief of 0.25 means that initially, the agent is 25% sure that the target is present in the region.

3. The sensor model false positive rate and false negative rate and the parameters α and β are set out in Section 5.2.4. These can differ from the true rate at which the sensor will

observe positive and negative observations. For example, given that the sensor actually outputs false positives and false negatives at a rate of 0.2 and 0.15 respectively, if the sensor model is calibrated with a false positive rate and false negative rate of 0.05 and 0.02 respectively, it will drastically under-estimate the rate at which false readings are recorded, which is reflected in the way it updates its estimated state.

4. # of Targets Present gives the number of distinct targets present in the search region. These are assumed to exist in distinct locations.

5. # of Agents Used is the number of agents participating in the search.

The results of the simulations show how varying these parameters and suggest how to set them to achieve a desired result. We focus on the most commonly reported metrics in the literature, which are related to the distribution of time to decision [20], [100], [101] and [53]. The time to decision is measured as the number of discrete time-steps before the search concludes. We also report on the rate at which incorrect target locations are returned and the rate at which the agents incorrectly conclude that the target is not present.

For each of the simulations, we arbitrarily chose to use the SPRT cut-off criteria with the upper Type I error probability set to 0.1 and the upper Type II error probability set to 0.15. In practice, this meant the agent would terminate the search if its cumulative belief that the target was present exceeded 0.895 or fell below 0.143. We generated simulated sensor readings using arbitrarily chosen values of the false positive rate = 0.2 and a false negative rate = 0.15. For each simulation run, we generated random starting locations for the agents and targets in a uniformly spaced 10×10 grid.

Unless specified otherwise, we use the following default parameters: sensor model false negative rate = 0.15, sensor model false positive rate = 0.2, initial belief distribution = uniform, initial cumulative belief target is present = 0.5, number of targets present = 1, number of active agents = 1, the ε-greedy search has ε=0.2 and a neighbourhood radius of 4. Histograms showing the results of running the simulation with varying parameters are shown in Appendix B.

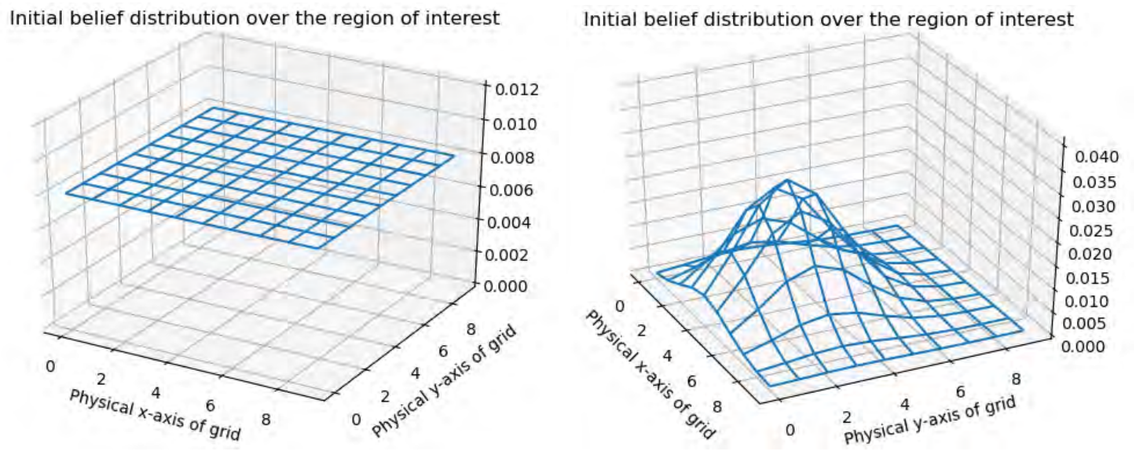
For the subsequent discussion, we use the abbreviations TTD: Time To Decision, SD: Standard Deviation, FPR: False Positive Rate, FNR: False Negative Rate.

5.4.1 Varying the initial distribution of the agent belief

This experiment explored the consequences of varying the initial distribution of the agent's belief while keeping all other parameters at their default values, outlined in 5.4.5. We chose to vary the initial distribution between a peaked Gaussian and non-peaked Uniform, as examples of two common use cases representing the presence of some prior information and no prior information respectively.

Strategy	Initial Belief Distribution	Mean TTD	Sample SD[TTD]	False Negative Rate	Proportion Incorrectly Localised
ε-Greedy	Uniform	112.9258	62.3798	0.1516	0.0398
ε-Greedy	Gaussian	21.68	20.44	0.0296	0.0118
Sweep	Uniform	601.5697	183.4529	0.1254	0.0454
Sweep	Gaussian	464.48	185.54	0.0832	0.0294
Saccadic	Uniform	98.8274	56.1298	0.1588	0.0370
Saccadic	Gaussian	14.558	18.75	0.0338	0.0114
Random	Uniform	629.5462	282.9514	0.1368	0.0366
Random	Gaussian	501.83	268.45	0.0792	0.0308

Table 5.2 Results of running the target localisation simulation with a Uniform initial belief distribution and Gaussian initial belief distribution for each implemented search strategy.



(a) Uniform initial belief distribution

(b) Gaussian initial belief distribution

Fig. 5.7 Initial belief distributions

Table 5.2 displays the results of running a simulation with a single agent and a single target while varying the initial belief distribution that the target is present in the search region.

For each search strategy, we ran the simulation with both a Uniform and Gaussian initial belief distribution. The Gaussian mean coincided with the target location and the covariance matrix was $\begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$, which is visualised in Figure 5.7. This represents a correct suspicion about the location of the target, which is the case in scenarios where some prior information may suggest clues to the target location. As expected, for all of the search strategies the expected time until a decision reduces dramatically when using the Gaussian prior relative to the Uniform prior. Since the shape of the Gaussian distribution is aligned more closely to the true distribution that the Uniform distribution, it takes fewer samples to reach a sufficiently peaked distribution to reach the acceptance region for the SPRT.

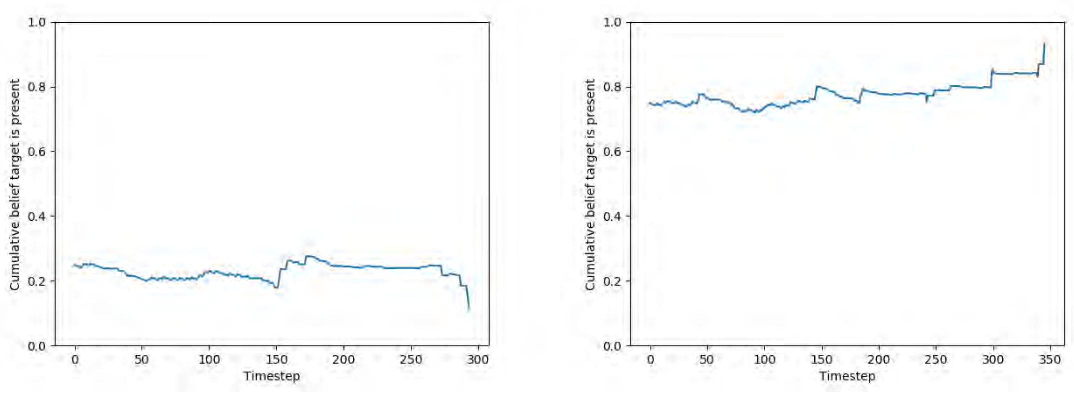
It can also be seen in Table 5.2 that the false negative rate (the rate at which the agent incorrectly concludes that the target is not present in the region) drops in the case of the Gaussian prior relative to the Uniform prior. This is also a result of using a prior distribution that matches the shape of the true distribution, since it requires a significant number of false positive/negative samples to modify the distribution enough to reach a negative conclusion.

5.4.2 Varying cumulative initial belief that the target is present

This experiment explored the consequences of varying the initial cumulative probability that the source is present. The cumulative probability of target presence reflects knowledge of likely the target presence is, prior to any evidence being gathered. We chose the parameters to reflect the cases where the search team do not believe it is unlikely that the target is present (0.25), it is equally likely that it is present as not present (0.5) and that it is likely that the target is present (0.75).

Strategy	Initial Belief Target Present	Mean TTD	Sample SD[TTD]	False Negative Rate	Proportion Incorrectly Localised
ε-Greedy	0.25	87.6214	30.9801	0.487	0.009
ε-Greedy	0.5	112.93	62.38	0.152	0.040
ε-Greedy	0.75	114.9276	81.9386	0.0396	0.1346
Sweep	0.25	520.4050	212.5122	0.4292	0.0118
Sweep	0.5	601.57	183.45	0.1254	0.0454
Sweep	0.75	485.3650	242.1377	0.0326	0.1586
Saccadic	0.25	75.8320	29.8345	0.5054	0.0064
Saccadic	0.5	98.83	56.13	0.1588	0.037
Saccadic	0.75	100.1332	74.3883	0.0392	0.1418
Random	0.25	539.3802	267.6280	0.4284	0.0074
Random	0.5	629.55	282.95	0.137	0.0336
Random	0.75	538.0904	325.6283	0.035	0.1538

Table 5.3 Results of running the target localisation simulation with a varying cumulative initial belief that the target is present in the search region.



(a) Belief for sweep search prematurely drops below the lower SPRT threshold with initial cumulative belief 0.25.

(b) Belief for sweep search does not prematurely drop below the lower SPRT threshold with initial cumulative belief 0.75.

Fig. 5.8 Sample runs showing the effect of varying the agent's initial cumulative belief that the target is present

Table 5.3 displays the results of running a simulation with a single agent and a single target while varying the agent's cumulative initial belief that the target is present in the search region. We discuss the results in relation to the adaptive and non-adaptive search methods:

1. The results show that for the non-adaptive Sweep and Random search strategies, the mean TTD decreases when the initial belief that the target is present is set to both 0.25 and 0.75. In the case of the initial 0.25 belief value, we calculated the lower bound on the number of necessary observations before a conclusion can be drawn is found to be 62 consecutive negative observations at distinct locations. This would cause the SPRT to determine the target is not present. It is not likely for this to happen for the given parameters, since the sensor will generate false positive readings at a rate of 0.2, but there are much more likely scenarios that occur which cause the search to terminate early, for example observing 88 negative observations and 16 positive observations.

   The search terminates prematurely in a high proportion of cases, before a significant number of positive observations are observed to move the cumulative belief that the target is present away from the lower decision boundary. This can be observed in Figure 5.8, where on the left hand side there are not a sufficient number of positive observations to stop the search from terminating early. In the case of the 0.75 initial belief value, fewer positive observations are required to reach the upper termination criteria set by the SPRT compared to starting with an initial belief of 0.5, with the same phenomenon occurring for the belief starting at 0.25. In the best case, a starting belief of 0.25 would need 6 consecutive positive observations at the same location to cross the upper termination threshold, whereas a starting belief of 0.5 would require 5 and a starting belief of 0.75 would only require 4. This comes at the price of making it easier to incorrectly identify the target location, increasing the proportion of runs where the target is incorrectly localised.

2. Relative to the cumulative belief starting at 0.5, the mean TTD only decreases for the adaptive ε-greedy and Saccadic search methods in the case that the initial belief is 0.25 and slightly goes up for both in the case where it is 0.75. We had expected the mean time to decision to decrease in the case of the initial belief starting at 0.75 and increase in the case of the initial belief starting at 0.25, since they are respectively closer and further away from the upper SPRT decision boundary. The reason for the reduced mean TTD with an initial belief of 0.25 for the adaptive search methods is the same as that for the non-adaptive search methods, where the relatively large number of negative samples drove the cumulative belief below the lower SPRT threshold prematurely. The answer to the question of why the mean time to decision is less than that for the non-adaptive search methods lies with the biased sampling nature of the adaptive methods. If one of the adaptive methods comes across a spurious positive, the next choice of sample location will be the same (or probably be the same in the ε-greedy case).

   For the parameters in this run of the simulation, after one positive spurious observation and one subsequent negative observation in the same location, the agent's belief that the target is present would be 0.2496. For the same scenario in the non-adaptive case, the agent would not re-sample the location with the spurious positive observation, but would continue on to sample the next location, which is likely to be a negative observation. In this case, the agent's belief that the target is present would be 0.2545, which is significantly higher than the adaptive case of 0.2496. Since the adaptive agent will re-sample spurious positives, it is able to then rule them out more quickly as potential target locations and as a result the expected time to decision is much lower than that for the non-adaptive methods. The slight increase in the TTD in the case where the initial belief is set to 0.75 is due to the large number of samples needed to break the lower threshold for the SPRT. For the cases where the initial belief set to 0.5 and 0.75, five and four consecutive positive observations at the same location are needed to cross the upper threshold respectively. 139 consecutive negative observations (100 at distinct locations, 39 at previously observed locations) would be needed to cross the lower threshold in the case of starting with an initial cumulative belief of

0.5 whereas 200 consecutive negative observations (100 distinct locations with 2 observations each) would be needed to cross the lower threshold in the case of starting with an initial cumulative belief of 0.75. The mean time to reach a negative decision was 309.1111 and 293.7143 for the ε-greedy and saccadic cases with initial belief 0.75, compared to 196.6622 and 177.3552 with initial belief 0.5. The mean time to reach a positive decision was 106.9209 and 92.2352 for the ε-greedy and saccadic cases with initial belief 0.75, compared to 97.9630 and 84.0031 with initial belief 0.5.

5.4.3 Varying sensor model parameters

This experiment explored the consequences of varying the agent's sensor model parameters while keeping all other parameters at their default values, outlined in Section 5.4.5. We varied the FPR and FNR of the agent's sensor model to reflect the cases where they underestimate (FPR=0.05, FNR=0.02), correctly estimate (FPR=0.2, FNR=0.15) and overestimate (FPR=0.05, FNR=0.02) the correct rates at which the sensor will record false positive and false negative observations. The correct rate at which the sensor makes false negative and false positive observations was set for all experiments as 0.2 and 0.15 respectively.

Strategy	Sensor Model FPR	Sensor Model FNR	Mean TTD	Sample SD[TTD]	False Negative Rate	Proportion Incorrectly Localised
ε-Greedy	0.05	0.02	67.2488	43.2448	0.1368	0.4270
ε-Greedy	0.2	0.15	112.9258	62.3798	0.1516	0.0398
ε-Greedy	0.4	0.4	197.5886	113.1707	0.0008	0.0000
Saccadic	0.05	0.02	59.9230	38.6500	0.1440	0.4298
Saccadic	0.2	0.15	98.8274	56.1298	0.1588	0.0370
Saccadic	0.4	0.4	142.2648	96.2213	0.0006	0.0
Random	0.05	0.02	167.2306	134.0652	0.015	0.7044
Random	0.2	0.15	629.5462	282.9514	0.1368	0.0366
Random	0.4	0.4	2100.5140	659.7263	0.1682	0.0
Sweep	0.05	0.02	132.1120	74.3178	0.0162	0.7932
Sweep	0.2	0.15	601.5697	183.4529	0.1254	0.0454
Sweep	0.4	0.4	2138.6002	554.5915	0.1344	0.0

Table 5.4 Results of running the target localisation simulation with varying sensor model parameters for each implemented search strategy.

Table 5.4 displays the results of running a simulation with a single agent and a single target while varying the parameters of the sensor model. The sensor model is parameterised using a false positive rate and false negative rate, detailed in Equation 5.1. For this set of simulations, we deliberately chose extreme values for these parameters to show the effects of an extreme miscalibration, which can be thought of as a worst case scenario. We discuss the results in terms of underestimating and overestimating the true sensor parameters:

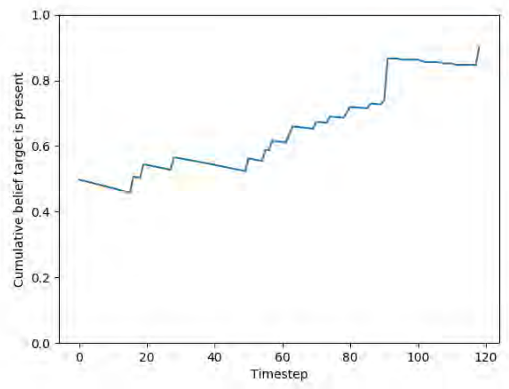
1. When the model is calibrated to underestimate the rate at which the sensor will detect false negatives and false positives (the rows which have a sensor model FPR = 0.05,

sensor model FPR = 0.02 in Table 5.4), relative to the correctly calibrated sensor model (the rows which have a sensor model FPR = 0.2, sensor model FPR = 0.15 in Table 5.4), both positive and negative observations will have a greater effect on the change in the shape of the distribution of the agent's belief, since the sensor model is more confident that the readings are correct. This means that is easier to cross the decision threshold of the SPRT due to spurious observations. For example, it would require 3, 5 and 17 consecutive positive observations at the same location to cross the upper SPRT threshold in the cases that the sensor model is calibrated with FPR=0.05/FPR=0.02, FPR=0.2/FPR=0.15, FPR=0.4/FPR=0.4 respectively.

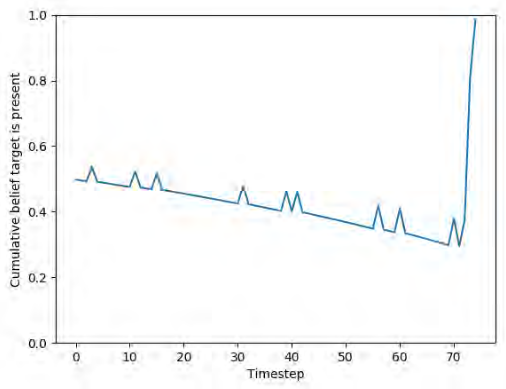
Underestimating the true sensor parameters leads to a much higher incorrect localisation rate in the adaptive search strategies in relation to the non-adaptive search strategies, which is evident in the Proportion Incorrectly Localised column of Table 5.4. The reason for this is that the non-adaptive search strategies do not follow up on positive observations. The agent belief increases a relatively large amount with every positive observation, which actually arrive at a much higher rate (0.2066) than the sensor is calibrated for (0.0593). For example, if the agent immediately makes a positive observation, its belief will jump from 0.5 to 0.5425. Figure 5.9 (a) shows the evolution of the agent's cumulative belief that the target is present in the search region. Relatively large jumps in cumulative belief due to the miscalibrated sensor model are clearly visible. This is in contrast with the adaptive strategies, where the agent re-samples positive observations, which drives the agent's belief back down in the case of spurious positives, which is visible in Figure 5.9 (b).

2. Conversely, when the model is calibrated to overestimate the rate at which the sensor will record false positives and false negatives (the rows which have a sensor model fpr = 0.4, sensor model FPR = 0.4 in Table 5.4), the belief distribution changes much more gradually given positive or negative observations. This means that crossing the decision thresholds given by the SPRT will require many more positive or negative readings, which leads to much higher mean times to decision (TTD) for all search strategies. As a lower bound, the SPRT would require 447 consecutive negative observations when running the sweep search in order to conclude that the target is not present. In the adaptive cases, once a positive observation is made at a given location, the agent's belief usually becomes highest at that location, which causes the agent to sample there again. Once it visits the true target location, usually it will receive a sufficient number of positive observations to keep sampling it, which causes its cumulative belief to steadily increase, as shown in Figure 5.10 (a). The non-adaptive methods do not exhibit this behaviour since they are unbiased sampling methods. They do not re-sample when

they come across a positive observation, which causes their cumulative belief that the target is present to gradually drop. If they do not experience consecutive positive observations when they visit the true target location, usually their belief will drop below the SPRT threshold, as can be seen in Figure 5.10 (b).

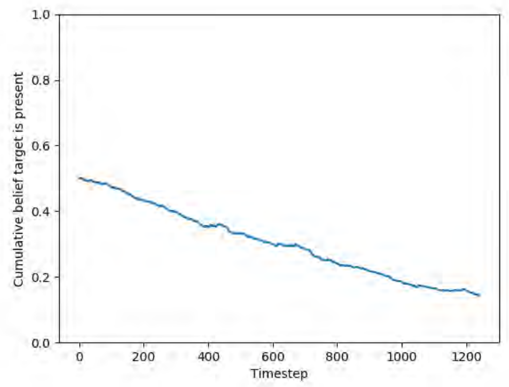


(a) Sample run using Sweep search strategy

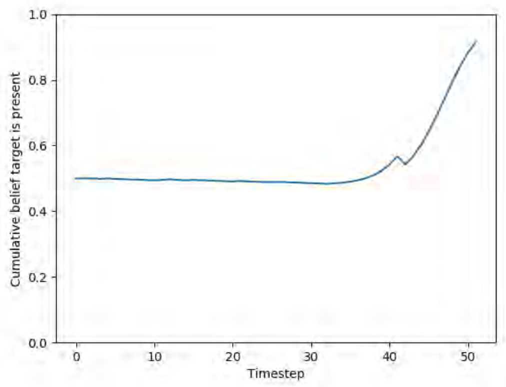


(b) Sample run using Saccadic search strategy

Fig. 5.9 Sample runs with miscalibrated sensor model parameters FPR=0.05, FNR=0.02



(a) Sample run using Random search strategy



(b) Sample run using Saccadic search strategy

Fig. 5.10 Sample runs with miscalibrated sensor model parameters FPR=0.4, FNR=0.4

5.4.4 Varying number of targets present in the region

This experiment explored the consequences of varying the number of targets present in the search region. We ran the experiment with 1-3 targets present, anticipating that the results could suggest the behaviour that would result for more than 3 targets. Tables 5.5 and 5.6 contain decision matrices which show the proportion of targets correctly localised vs. the number of returned target locations. Since the agent can return either a target location or that the target is not present, there is scope two potential sources of error: returning an incorrect target location or returning that the target is not present. For example, for the ε-Greedy search strategy with 3 targets present, 5.10% of the time, the agent made 3 guesses for the target locations and 2 of these were correct. For the same case, 31.7% of the runs ended with the agent correctly returning 2 target locations before concluding that a third target was not present (mistakenly), as seen in the cell intersecting the third row and third column.

Strategy	# Targets Present	Mean TTD	SD [TTD]	Decision Matrix Showing Proportions of # Returned Target Locations (Rows) vs. # Targets Correctly Localised (Cols).
ε -Greedy	1	112.93	62.38	$\begin{array}{ccc} & 0 & 1 \\ 0 & 0.152 & 0 \\ 1 & 0.0400 & 0.8080 \end{array}$
ε -Greedy	2	155.77	50.93	$\begin{array}{cccc} & 0 & 1 & 2 \\ 0 & 0.0218 & 0 & 0 \\ 1 & 0.0012 & 0.2644 & 0 \\ 2 & 0.0006 & 0.0474 & 0.6646 \end{array}$
ε -Greedy	3	175.84	42.78	$\begin{array}{ccccc} & 0 & 1 & 2 & 3 \\ 0 & 0.0038 & 0 & 0 & 0 \\ 1 & 0 & 0.0596 & 0 & 0 \\ 2 & 0 & 0.0064 & 0.3170 & 0 \\ 3 & 0 & 0.0002 & 0.0510 & 0.5620 \end{array}$
Sweep	1	601.57	183.45	$\begin{array}{ccc} & 0 & 1 \\ 0 & 0.1254 & 0 \\ 1 & 0.0454 & 0.8292 \end{array}$
Sweep	2	708.02	206.57	$\begin{array}{cccc} & 0 & 1 & 2 \\ 0 & 0.0106 & 0 & 0 \\ 1 & 0.0002 & 0.181 & 0 \\ 2 & 0.0018 & 0.0666 & 0.7398 \end{array}$
Sweep	3	770.00	223.96	$\begin{array}{ccccc} & 0 & 1 & 2 & 3 \\ 0 & 0.002 & 0 & 0 & 0 \\ 1 & 0.0002 & 0.022 & 0 & 0 \\ 2 & 0 & 0.0014 & 0.2104 & 0 \\ 3 & 0.0002 & 0.0034 & 0.082 & 0.6784 \end{array}$

Table 5.5 Results of running the target localisation simulation with 1,2 and 3 targets for sweep search and ε-greedy search.

Strategy	No. Targets Present	Mean TTD	SD [TTD]	Decision Matrix Showing Proportions of # Returned Target Locations (Rows) vs. # Targets Correctly Localised (Cols).
Saccadic	1	98.83	56.13	$\begin{array}{c\|c\|c} & 0 & 1 \\ \hline 0 & 0.1588 & 0 \\ 1 & 0.0370 & 0.8042 \end{array}$
Saccadic	2	135.64	45.43	$\begin{array}{c\|c\|c\|c} & 0 & 1 & 2 \\ \hline 0 & 0.0258 & 0 & 0 \\ 1 & 0.0014 & 0.2664 & 0 \\ 2 & 0 & 0.0448 & 0.6616 \end{array}$
Saccadic	3	154.59	36.62	$\begin{array}{c\|c\|c\|c\|c} & 0 & 1 & 2 & 3 \\ \hline 0 & 0.0052 & 0 & 0 & 0 \\ 1 & 0.0002 & 0.07 & 0 & 0 \\ 2 & 0 & 0.0036 & 0.3246 & 0 \\ 3 & 0 & 0.0006 & 0.0458 & 0.55 \end{array}$
Random	1	629.55	282.95	$\begin{array}{c\|c\|c} & 0 & 1 \\ \hline 0 & 0.1368 & 0 \\ 1 & 0.0336 & 0.8296 \end{array}$
Random	2	787.44	287.61	$\begin{array}{c\|c\|c\|c} & 0 & 1 & 2 \\ \hline 0 & 0.0166 & 0 & 0 \\ 1 & 0.0012 & 0.1872 & 0 \\ 2 & 0.0004 & 0.053 & 0.7416 \end{array}$
Random	3	882.74	295.05	$\begin{array}{c\|c\|c\|c\|c} & 0 & 1 & 2 & 3 \\ \hline 0 & 0.0014 & 0 & 0 & 0 \\ 1 & 0.0002 & 0.0248 & 0 & 0 \\ 2 & 0 & 0.0014 & 0.2138 & 0 \\ 3 & 0 & 0.0022 & 0.0608 & 0.6954 \end{array}$

Table 5.6 Results of running the target localisation simulation with 1,2 and 3 targets for saccadic search and random search.

Tables 5.5 and 5.6 display the results of running a simulation with a single agent with a varying number of targets in the search region. We ran algorithm 7 to carry out the localisation procedure. The results show that the mean time to decision increases sub-linearly with respect to the number of targets present. The rows of the decision matrix show the proportions of how many target locations are returned by the agent and the columns show how many targets are correctly localised. The results indicate that the more targets that are present, the less willing the agent is to conclude that the are no targets present, but that it is also less willing to conclude that the maximum number of targets are present. The reason why the agent consistently returns 0 target locations at a lower rate when there is more targets present can be explained by the miscalibrated sensor results; since there are more targets present, the proportion of observed true positives is greater than the calibrated values. This essentially means that the sensor model FNR is over-estimated, which leads to a much lower search false negative rate. This could be overcome by calibrating the sensor model with the FNR and FPR that would arise from the number of targets suspected to be present in the region, and re-calibrating it each time a target is localised by the agent.

5.4.5 Varying number of search agents

This experiment explored how varying the number of agents involved in the search impacted on the effectiveness of the search. We assumed that agents communicated their most recent observation to all other agent at each time step, which the other agents used to update their local belief that the target it present in the region.

Strategy	# Agents	Mean TTD	Sample SD[TTD]	False Negative Rate	Proportion Incorrectly Localised
ε-Greedy	1	112.9258	62.3798	0.1516	0.0398
ε-Greedy	2	65.5912	34.3248	0.1568	0.0314
ε-Greedy	3	47.5176	24.9861	0.1428	0.0262
Sweep	1	601.5697	183.4529	0.1254	0.0454
Sweep	2	303.7328	94.2748	0.1232	0.0466
Sweep	3	204.8172	65.1273	0.1252	0.0408
Saccadic	1	98.8274	56.1298	0.1588	0.0370
Saccadic	2	75.3466	39.9718	0.1520	0.0132
Saccadic	3	65.0774	33.9798	0.1598	0.0090
Random	1	629.5462	282.9514	0.1368	0.0366
Random	2	315.0082	140.3954	0.1254	0.0366
Random	3	211.4242	94.7801	0.1222	0.0448

Table 5.7 Results of running the target localisation simulation with a varying number of homogeneous agents for each implemented search strategy.

Table 5.7 shows the results of running the search with 1, 2 and 3 homogeneous agents. Each agent is initialised with the same parameters, but start at independently chosen random locations. They receive new observations from other agents at each time-step and then record their own observation. We did not assume that there could be corrupted/interrupted transmissions, although this could be investigated in future work. Figure 2 and Figure 3 of [19] show sample runs of adaptive and non-adaptive search strategies for the same simulation configuration. The observed results matched our expectations based on [19], where the largest relative reduction in the mean TTD is found in the non-adaptive cases. This is because the non-adaptive cases use unbiased sampling methods. The sweep search method samples all grid cells an equal number of times, so running multiple agents is equivalent to running a single agent with the non-adaptive strategy that can sample multiple times on each time-step rather than just once. This implies the search time is inversely proportional to the number of agents used in the non-adaptive cases, supported by Table 5.2. Since the agents first record a sensor observation and update their belief before updating their belief based on other agent observations made at the current time-step, sometimes an agent will end its search before the other agents, which means that the effect of having multiple agents is lessened. This is more prevalent when using the adaptive strategies since when the agents locate the source correctly, there is a chance that one could receive a false negative, while the others terminate with true positive observations. This leads the last agent to make extra observations, inflating the mean TTD. The unbiased methods do not exhibit this behaviour as much and usually terminate when one agent at the target location makes a positive observation which is shared among other agents who are not at the target location and who then also subsequently terminate.

5.5 Conclusion and Future Work

This chapter described a system which was designed to solve the problem of *target localisation* using RAVs. Our system design was based on previous work, outlined in Section 2.3. The system's performance was analysed using Monte Carlo simulation and we explored the trade-off between minimising Time-To-Decision (TTD) against drawing incorrect conclusions. The results largely matched what we expected to see, but they offer a solid basis on which to set system parameters in order to achieve a desired level of performance. We evaluated four search strategies. Two were adaptive, ε-greedy search and Saccadic search, meaning they sampled the search region in a biased manner in the hope of converging quickly on the target location. Two were unbiased, Sweep search and Random search, which offered a baseline for comparison and facilitated analysis which was useful in showing how the system reacts to perturbations in the values of the parameters.

5.5.1 Varying Initial Distribution

We first found that the shape of the initial belief distribution has a large bearing on the time-to-decision, where the use of an uninformed initial belief distribution causes a large increase in the search time relative to the use of an initial belief distribution that is aligned with the shape of the true distribution. The effect was greatest while using the adaptive search methods, which offered respective reductions in mean time to decision of 80.1% and 85.3% for the ε-greedy search and Saccadic search strategies. The results also showed that the rate at which the system incorrectly concluded that the target was not present also dropped significantly, with respective reduction in mean time to decision of 80.1% and 75.6.3% for the ε-greedy search and Saccadic search strategies.

5.5.2 Varying Initial Cumulative Probability of Target Presence

When the initial cumulative belief that the target was present for a uniform distribution was set to 0.25, we observed a drop in the mean time to decision for all search strategies. The adaptive strategies resampled the locations of false positives, which quickly drove the cumulative belief below the lower SPRT threshold prematurely. The non-adaptive strategies also experienced enough negative observations in relation to positives to terminate prematurely. This led to an inflated rate at which the system concluded the source was not present. In the case where the initial cumulative belief that the target was present started at 0.75, the adaptive methods saw a slight increase in the mean time to decision relative to the case where the initial cumulative belief was set to 0.5. The evidence suggested that since a comparable number of consecutive positive observations at the true target location would push the agent's belief over the SPRT upper threshold, the relatively high number of samples needed to make a negative conclusion was the cause. The non-adaptive methods saw a decrease in the mean time to decision because the initial belief started closer to the boundary and required fewer positive observations to cross the upper SPRT threshold.

5.5.3 Varying Sensor Model Parameters

When we set the sensor model parameters to under-estimate the actual false positive and false negative rates of observations, the results we observed matched our expectations. Since the model was more confident about observations, the agent's belief fluctuated more aggressively, which made it easier to cross the upper and lower decision thresholds. This led to a drop in the mean time to decision for all search strategies relative the case with the correctly calibrated sensor (a reduction in mean TTD of 40.4%, 37.4%, 71.4%, 78.0% for ε-greedy, Saccadic, Random and Sweep respectively). Conversely, when the sensor model parameters over-estimated the FPR and FNR of the sensor, the agent's belief fluctuated in a much more gentle manner, which meant it took many more observations to reach the threshold set by the SPRT compared to the case with the correctly calibrated sensor (75.0%, 43.9%, 233.7%, 255.5% for ε-greedy, Saccadic, Random and Sweep respectively). When the sensor model parameters were under-estimated, the proportion of simulation runs where the target was incorrectly localised shot up, since the relative true positive rate was lower than in the case where the sensor model FPR and FNR were set to 0.2 and 0.15 respectively.

5.5.4 Varying Number of Targets

Multiple targets caused the mean time to decision to increase sub-linearly with the number of targets present for all search strategies. This is because the agent ran a sequential procedure, which meant information gathered while localising the first target could be used to expedite the localisation of subsequent targets. The presence of two targets instead of one increased the mean time to decision by 37.9%, 17.7%, 37.3%, 25.1% for ε-greedy, Sweep, Saccadic and Random respectively. The presence of three targets instead of one increased the mean time to decision by 55.7%, 28.0%, 56.4%, 40.2% for ε-greedy, Sweep, Saccadic and Random respectively. These represent marked improvements over the naive case, where re-starting the search once a target had been localised with the same initial belief distribution would have double and tripled the mean time to decision for two and three targets. The results show that as more targets were introduced to the target region, the more likely the search procedure was to terminate prematurely, falsely concluding that there were fewer targets present than there actually were. Addressing this could be part of future work.

5.5.5 Varying Number of Search Agents

The use of multiple agents cut the mean time to decision for each of the search strategies while maintaining the rates at which an incorrect conclusion was drawn. The use of two agents cut the mean time to decision by 42.0%, 50.0%, 23.4%, 50.0% for ε-greedy, Sweep, Saccadic and Random respectively. The use of three agents cut the mean time to decision by 58.0%, 66.0%, 34.2%, 66.4% for ε-greedy, Sweep, Saccadic and Random respectively. We anticipate that introducing a maximum radius in which the agents can communicate with each other, with the probability of communicating successfully inversely proportional to the distance between agents, would mean adding extra agents would have less of an effect on the mean time to decision but would help to stabilise the rates at which the system mistakenly concludes the search.

5.5.6 Future Work

Future work for the problem of target localisation has been outlined in [18], [19], [20], [49]. We summarise the previously identified future work, along with our own suggestions, as follows:

1. We did not take into account battery limitations for the RAV agents. In reality, they would need to periodically recharge while carrying out the search, as mentioned in Section 4.3 of Chapter 4. We hypothesise that a battery model could be learned using the Expectation-Maximisation (E-M) algorithm [24]. This could then be integrated with the search strategy to prevent the RAVs from running out of power while searching.

2. In this work, all simulations were ran with a target present. Future work could investigate system performance with the absence of a target, or with the presence of false targets that could be considered as an adversarial measure taken to fool the system.

3. We assumed that agents could localise *themselves* accurately, which is not always the case. Research into how agents can perform the search without a deterministic location sensor could incorporate this extra level of uncertainty.

4. Our simulation involving multiple agents assumed that they are homogeneous. Future work could explore the use of heterogeneous agents in carrying out the target localisation task. This could involve using agents with varying sensing capabilities, operational speeds, battery lives and search strategies. Future work could also investigate how to optimise various configurations to provide a more robust and efficient system, building on [19].

5. The performance measure discussed in Section 5.2.3 only takes into account the time taken to localise the target. It could be modified to take into account other factors such as energy expended by the agents. This may allow for a more realistic trade-off than solely focusing on making the correct decision in the least possible amount of time.

6. It is becoming more common for RAVs to integrate a multitude of sensors in their platforms. For example, in the ROCSAFE project [3], advanced lightweight sensors are being developed to be used with a RAV to remotely detect potentially hazardous substances. Multi-sensor fusion techniques have been shown in certain cases to be highly effective in identifying outliers and creating more robust estimators of true environment state. The review paper [42] discusses challenges and promising future areas related to research involving the fusion of data from multiple sensors. Exploring the improvements that a multi-sensor approach could potentially provide could be a future avenue of research.

7. We followed the approach of [18] in developing a sensor model. The approach is very general, but is highly susceptible to providing poor results due to miscalibration, evident in Table 5.4. Developing a more advanced sensor model that is specific to common tasks, such as object detection, may yield a more robust and efficient system. For example, [94] outlines a sensor model which was calibrated with varying heights of the RAV and [96] outlines sensor models commonly used with robotic vehicles.

8. Running the system in a realistic setting would offer insight in the practicality of the system. For example, we ran the agent software on a desktop machine, but in reality it would likely be running on an embedded system with a reduced computational ability relative to a desktop. This may show the system is infeasible due to constraints such as memory size or processor speed. Future research could involve running the system on realistic hardware to evaluate its feasibility. In the future, we intend to run the system

using the high-fidelity simulation discussed in Chapter 3, with the agent software running on raspberry pi devices. This will give a stronger indication of how the system may potentially perform in the real world.

9. Further theoretical analysis of the system could offer insight into how it could be improved. In [20] and [21], special cases are analysed in order to elicit properties, such as the rate of change of agent belief, which can be useful in devising strategies to improve the search performance.

Chapter 6

Conclusion

At the beginning of this thesis, we identified two research questions:

- Can an agent-based software system be developed to run on a system of heterogeneous autonomous aerial vehicles to aid the tasks of scene surveying and target localisation in a hazardous environment?
- Can a high-fidelity simulation environment be created, which can generate realistic salient data to support the process of prototyping AI systems designed to aid the management of real-world hazardous scenarios?

These were identified based on the ROCSAFE project and we aimed to find answers to them with the work done as part of this thesis.

The second question was addressed in Chapter 3, where we provided the details of a custom-built high-fidelity simulation environment using the UE4 game engine. We identified realistic data that was relevant to hazardous real-world scenarios, which included images taken from the perspective of a RAV and CBRN sensor data. We simulated this data through the use of tools that are part of UE4 as well as the use of the Airsim [81] plugin for UE4. This aided the tasks of developing techniques to solve the problems of scene surveying and target localisation, identified in the second research question, by ensuring that the developed methods could be tested at low monetary cost and high speed. This allows us to answer the first question positively.

We can also answer the first research question positively, based on the work outlined in Chapters 4 and 5. We developed a software system that built on previous related work which was highly modular and could be calibrated to reflect realistic varying parameters among aerial vehicles, such as operational speed. The task of scene surveying was addressed in Chapter 4 and we devised a method of discretising the region to be surveyed along with algorithms that would set out routes for the aerial vehicles. We tested this using our developed high-fidelity simulation environment, which closely mirrored intended real-world usage. The task of target localisation was tackled in Chapter 5. We provided the details of a system that could use multiple RAVs to effectively find the hidden location of one or more targets, based on sensor data. We used the Sequential Probability Ratio Test as a cut-off criterion for the search termination. We implemented a number of sampling strategies and investigated the effects of varying key parameters in the search, such as the prior distribution and the number of targets present. The results indicate that for reasonable choices of these parameters, the search could localise one or more targets according to the error rates set out by the SPRT.

References

[1] Aarts, E. and Lenstra, J. K., editors (1997). *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[2] Assaf, D. and Zamir, S. (1985). Optimal Sequential Search: A Bayesian Approach. *The Annals of Statistics*, 13(3):1213–1221.

[3] Bagherzadeh, N., Madden, M., Drury, B., and Madden, M. G. (2017). ROCSAFE: Remote Forensics for High Risk Incidents. In *IJCAI*, Melbourne.

[4] Bay, H., Tuytelaars, T., and Gool, L. V. (2006). SURF: Speeded Up Robust Features. In *ECCV*, pages 404–417, Graz. Springer-Verlag.

[5] Bhattacharya, R. N. and Waymire, E. C. (2009). *Stochastic Processes With Applications*. Wiley & Sons, Philadelphia.

[6] Bianco, S., Ciocca, G., and Marelli, D. (2018). Evaluating the Performance of Structure from Motion Pipelines. *Journal of Imaging*, 4(8):98.

[7] Black, W. L. (1965). Discrete Sequential Search. *Information and Control*, 8:159–162.

[8] Blackwell, D. (1962). Discrete Dynamic Programming. *Annals of Mathematical Statistics*, 33(2):719–726.

[9] Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. (2016). End to End Learning for Self-Driving Cars.

[10] Bourgault, F., Furukawa, T., and Durrant-Whyte, H. (2004). Decentralized Bayesian negotiation for cooperative search. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2681–2686, Sendai. Institute of Electrical and Electronics Engineers (IEEE).

[11] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, New York.

[12] Cameron, S., Hailes, S., Julier, S., Mcclean, S., Parr, G., Trigoni, N., Ahmed, M., Mcphillips, G., De Nardi, R., Nie, J., Symington, A., Teacy, L., and Waharte, S. (2010). SUAAVE: Combining Aerial Robots and Wireless Networking Contact Details. In *25th Bristol International UAV Systems Conference*, Bristol.

[13] Carpin, S., Lewis, M., Wang, J., Balakirsky, S., and Scrapper, C. (2007). USARSim: A robot simulator for research and education. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1400–1405, Roma. IEEE.

[14] Chamberlain, R. (1996). Gisl listserver faq. https://groups.google.com/forum/#!searchin/comp.infosystems.gis/\protect\T1\textdollar20GIS\protect\T1\textdollar20FAQ\protect\T1\textdollar20List\protect\T1\textdollar20%7Csort:date/comp.infosystems.gis/Yy14nK03oIU/moxwTGYbigEJ.

[15] Chew, M. (1966). A Sequential Search Procedure. *The Annals of Mathematical Statistics*, 38(2):494–502.

[16] Chew, M. C. (1973). Optimal Stopping in a Discrete Search Problem. *Operations Research*, 21(3):741–747.

[17] Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, U.S. Office of Naval Research, Pittsburgh.

[18] Chung, T. H. and Burdick, J. W. (2007). A decision-making framework for control strategies in probabilistic search. In *IEEE International Conference on Robotics and Automation*, pages 4386–4393, Roma. Institute of Electrical and Electronics Engineers.

[19] Chung, T. H. and Burdick, J. W. (2008). Multi-agent probabilistic search in a sequential decision-theoretic framework. In *IEEE International Conference on Robotics and Automation*, pages 146–151, Pasadena. Institute of Electrical and Electronics Engineers.

[20] Chung, T. H. and Burdick, J. W. (2012). Analysis of search decision making using probabilistic search strategies. *IEEE Transactions on Robotics*, 28(1):132–144.

[21] Chung, T. H., Kress, M., and Royset, J. O. (2009). Probabilistic search optimization and mission assignment for heterogeneous autonomous agents. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 939–945, Kobe. IEEE.

[22] Cowan, G. (1998). *Statistical data analysis*. Oxford University Press, USA.

[23] Dantzig, G. B. and Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, 6(1):80–91.

[24] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.

[25] Dosovitskiy, A., Ros, G., Codevilla, F., López, A., and Koltun, V. (2017). CARLA: An Open Urban Driving Simulator. In Sergey Levine and Vincent Vanhoucke and Ken Goldberg, editor, *Proceedings of Machine Learning Research*, pages 1–16, California. PMLR.

[26] DroneCodeProject (2014). jMAVSim. https://github.com/PX4/jMAVSim. [Online; accessed 4/7/19].

[27] Ebeid, E., Skriver, M., Terkildsen, K. H., Jensen, K., and Schultz, U. P. (2018). A survey of Open-Source UAV flight controllers and flight simulators. *Microprocessors and Microsystems*, 61:11–20.

[28] Echeverria, G., Lassabe, N., Degroote, A., and Lemaignan, S. (2011). Modular Open Robots Simulation Engine: MORSE. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 46–51, Shanghai. IEEE.

[29] Elfes, A. (1989). Using Occupancy Grids for Mobile Robot Perception and Navigation. *Computer*, 22(6):46–57.

[30] Erdos, P. and Renyi, A. (1961). On a Classical Problem of Probability Theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*.

[31] Gaidon, A., Wang, Q., Cabon, Y., and Vig, E. (2016). Virtual Worlds as Proxy for Multi-Object Tracking Analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas. Institute of Electrical and Electronics Engineers.

[32] Gerkey, B. P. and Matarić, M. J. (2004). A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954.

[33] Ghahramani, Z. (2001). An Introduction To Hidden Markov Models And Bayesian Networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(1):9–42.

[34] Grabowski, R., Navarro-Serment, L. E., Paredis, C. J. J., and Khosla, P. K. (2000). Heterogeneous Teams of Modular Robots for Mapping and Exploration. *Autonomous Robots*, 8:293–308.

[35] Hattenberger, G., Bronz, M., and Gorraz, M. (2014). Using the Paparazzi UAV System for Scientific Research. In *Proceedings of the International Micro Air Vehicle Conference and Competition*, pages 247–252, Delft. Delft University of Technology.

[36] Held, M. and Karp, R. M. (1962). A Dynamic Programming Approach to Sequencing Problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210.

[37] Hogg, R., McKean, J., and Craig, A. (1978). *Introduction To Mathematical Statistics*. Macmillan Publishing Co., 866 Third Avenue, New York.

[38] Hungerländer, P., Jellen, A., Jessenitschnig, S., Knoblinger, L., Lackenbucher, M., and Maier, K. (2018). The Multiple Traveling Salesperson Problem on Regular Grids. In *Operations Research*, Brussels. Springer.

[39] Jacoff, A., Messina, E., Weiss, B. A., Tadokoro, S., and Nakagawa, Y. (2003). Test Arenas and Performance Metrics for Urban Search and Rescue Robots. In *IEEE International Conference on Intelligent Robots and Systems*, volume 4, pages 3396–3403.

[40] Johnson, D. S. and Mcgeoch, L. A. (1997). The Traveling Salesman Problem: A Case Study in Local Optimization. In *Local Seach in Combinatorial Optimization*, chapter 8, pages 215–310. Wiley, New York.

[41] Juliani, A., Berges, V.-P., Vckay, E., Gao, Y., Henry, H., Mattar, M., and Lange, D. (2018). Unity: A General Platform for Intelligent Agents. *Arxiv*.

[42] Khaleghi, B., Khamis, A., Karray, F. O., and Razavi, S. N. (2013). Multisensor data fusion: A review of the state-of-the-art. *Information Fusion*, 14(1):28–44.

[43] Kimeldorf, G. and Smith, F. H. (1979). Binomial Searching for a Random Number of Multinomially Hidden Objects. *Management Science*, 25(11):1115–1126.

[44] Koenig, N. and Howard, A. (2005). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, Sendai. IEEE.

[45] Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press.

[46] Koopman, B. O. (1956a). The Theory of Search I. Kinematic Bases. *The Journal of the Operations Research Society of America*, 4(3).

[47] Koopman, B. O. (1956b). The theory of search. ii. target detection. *Operations Research*, 4(5):503–531.

[48] Koopman, B. O. (1957). The Theory of Search. III. The Optimum Distribution of Searching Effort. *The Journal of the Operations Research Society of America*, 5(5):613–626.

[49] Kriheli, B., Levner, E., and Spivak, A. (2016). Optimal Search for Hidden Targets by Unmanned Aerial Vehicles under Imperfect Inspections. *American Journal of Operations Research*, 6:153–166.

[50] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems*, pages 1097–1105, Nevada. Curran Associates Inc.

[51] Laamarti, F., Eid, M., and El Saddik, A. (2014). An overview of serious games. *International Journal of Computer Games Technology*, 2014.

[52] Lai, K. T., Lin, C. C., Kang, C. Y., Liao, M. E., and Chen, M. S. (2018). ViviD: Virtual environment for visual deep learning. In *MM 2018 - Proceedings of the 2018 ACM Multimedia Conference*, pages 1356–1359. Association for Computing Machinery, Inc.

[53] Lau, H. (2007). *Optimal Search in Structured Environments*. PhD thesis, The University of Technology, Sydney.

[54] Lecun, Y., Bottou, E., Bengio, Y., and Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[55] Lee, K., Shmatov, A., Byrne, J., and Moloney, D. (2018). Digital Scene Augmentation Techniques for Generating Photo-Realistic Virtual Crowds. In *2018 IEEE Games, Entertainment, Media Conference, GEM 2018*, pages 299–305. Institute of Electrical and Electronics Engineers Inc.

[56] Levy, S. (2018). Multicoptersim. https://github.com/simondlevy/MulticopterSim. [Online; accessed 8/7/19].

[57] Lewis, M. and Jacobson, J. (2002). GameEnginesInScientificResearch. *Communications of the ACM*, 45(1):27–31.

[58] Mack, C. A. (2011). Fifty years of Moore's law. *IEEE Transactions on Semiconductor Manufacturing*, 24(2):202–207.

[59] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*.

[60] Mueller, M., Smith, N., and Ghanem, B. (2016). A benchmark and simulator for UAV tracking. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9905 LNCS, pages 445–461. Springer Verlag.

[61] Müller, J. P. and Fischer, K. (2014). Application Impact of Multiagent Systems and Technologies: A Survey. In *Agent-Oriented Software Engineering*. Springer, Berlin, Heidelberg, Berlin.

[62] Murphy, K. P. (1994). *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley.

[63] National University of Ireland, G. (2018). Grid generation ui. https://github.com/NUIG-ROCSAFE/CBRNeVirtualEnvironment. [Online; accessed 1/8/19].

[64] Ou, Y., Wu, Z., Chen, S., and Lee, K. M. (2010). An improved SPRT control chart for monitoring process mean. *International Journal of Advanced Manufacturing Technology*, 51(9-12):1045–1054.

[65] Papoulis, A. and Pillai, S. U. (2002). *Probability, Random Variables, and Stochastic Processes*. McGraw Hill, Boston, fourth edition.

[66] Pell, A., Meingast, A., and Schauer, O. (2017). Trends in Real-time Traffic Simulation. *Transportation Research Procedia*, 25:1477–1484.

[67] Pollock, S. M. (1971). Search Detection and Subsequent Action: Some Problems on the Interfaces. *Operations Research*, 19(3):559–586.

[68] Qiu, W. and Yuille, A. (2016). UnrealCV: Connecting Computer Vision to Unreal Engine. *European Conference on Computer Vision*.

[69] Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286.

[70] Reinelt, G. (1991). Tsplib—a traveling salesman problem library. *Orsa Journal On Computing*, 3(4):376–384.

[71] Report prepared by the DMA WGS 84 Development Committee (1991). Department of Defense World Geodetic System 1984. Technical report, U.S. Defense Mapping Agency.

[72] Robert, C. and Casella, G. (2011). A Short History of Markov Chain Monte Carlo: Subjective Recollections from Incomplete. *Statistical Science*, 26(1):102–115.

[73] Robin, C. and Lacroix, S. (2016). Multi-robot target detection and tracking: taxonomy and survey. *Autonomous Robots*, 40:729–760.

[74] Rosenkrantz, D. J., Stearns, R. E., and Lewis, P. M. (1977). An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6:563–581.

[75] Ross, S. (1969). A Problem in Optimal Search And Stop. *Operations Research*, 17(6).

[76] Rouse, R. (2004). *Game design : Theory &amp; Practice*. Jones & Bartlett Learning, 2 edition.

[77] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

[78] Ryoo, S., Rodrigues, C. I., Baghsorkhi, S. S., Stone, S. S., Kirk, D. B., and Hwu, W.-M. W. (2008). Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA. *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*.

[79] Sadeghi, F. and Levine, S. (2016). CAD 2 RL: Real Single-Image Flight Without a Single Real Image. *arXiv Preprint*.

[80] Saeedi, S., Trentini, M., Seto, M., and Li, H. (2016). Multiple-Robot Simultaneous Localization and Mapping: A Review. *Journal of Field Robotics*, 33(1):3–46.

[81] Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2017). AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In Hutter M., S. R., editor, *Field and Service Robotics*, pages 621–635, Zurich. Springer, Cham.

[82] Shannon, R. E. (1998). Introduction to the Art and Science of Simulation. In *WSC '98 Proceedings of the 30th conference on Winter simulation*, pages 7–14, California. IEEE Computer Society Press.

[83] Shimrat, M. (1962). Algorithms. *Communications of the ACM*, page 434.

[84] Smallwood, R. D. and Sondik, E. J. (1973). The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon. *Operations Research*, 21(5):1071–1088.

[85] Smyth, D. L., Abinesh, S., Karimi, N. B., Drury, B., Ullah, I., Glavin, F. G., and Madden, M. G. (2018a). A Virtual Testbed for Critical Incident Investigation with Autonomous Remote Aerial Vehicle Surveying, Artificial Intelligence, and Decision Support. In *IWAISe 2nd International Workshop on A.I. in Security*, pages 88–93, Dublin. Springer.

[86] Smyth, D. L., Fennell, J., Abinesh, S., Karimi, N. B., Glavin, F. G., Ullah, I., Drury, B., and Madden, M. G. (2018b). A Virtual Environment with Multi-Robot Navigation, Analytics, and Decision Support for Critical Incident Investigation. In *International Joint Conference on Artificial Intelligence*, pages 5862–5864, Stockholm. International Joint Conferences on Artificial Intelligence.

[87] Smyth, D. L., Glavin, F. G., and Madden, M. G. (2018c). Using a Game Engine to Simulate Critical Incidents and Data Collection by Autonomous Drones. In *2018 IEEE Games, Entertainment, Media Conference, GEM 2018*, pages 436–440, Galway. Institute of Electrical and Electronics Engineers Inc.

[88] Smyth, P., Heckerman, D., and Jordan, M. I. (1997). Probabilistic independence networks for hidden Markov probability models. *Neural Computation*, 9(2):227–269.

[89] Söbke, H. and Streicher, A. (2016). Serious games architectures and engines. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9970 LNCS, pages 148–173. Springer Verlag.

[90] Standardization Office, N. (2012). AJP-3.8(A), Allied Joint Doctrine for Chemical, Biological, Radiological and Nuclear Defence. Technical report.

[91] Stone, L. D. (1980). Optimal search for moving targets. *Advances in Applied Probability*, 12(2):279–280.

[92] Stone, P. and Veloso, M. (2000). Multiagent Systems: A Survey from a Machine Learning Perspective. *In Autonomous Robotics*, 8(3).

[93] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

[94] Symington, A., Waharte, S., Julier, S., and Trigoni, N. (2010). Probabilistic target detection by camera-equipped UAVs. In *Proceedings - IEEE International Conference on Robotics and Automation*.

[95] Technical Working Group on Crime Scene Investigation (2013). *Crime Scene Investigation A Guide For Law Enforcement*. US Department of Justice.

[96] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

[97] Toth, P. and Vigo, D. (2002). *The vehicle routing problem*. Society for Industrial and Applied Mathematics, Philadelphia, 2 edition.

[98] Vilches, V. M., Cordero, A. H., Calvo, B., Ugarte, I. Z., Kojcev, R., Robotics, E., Vitoria-Gasteiz, S. L., and Spain, A. (2018). robot gym: accelerated robot training through simulation in the cloud with ROS and Gazebo. *ArXiv*.

[99] Vincenty, T. (1975). Direct and Inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, 23(176):88–93.

[100] Waharte, S., Symington, A., and Trigoni, N. (2010). Probabilistic Search with Agile UAVs. In *IEEE International Conference on Robotics and Automation*, pages 2840–2845, Anchorage. Institute of Electrical and Electronics Engineers.

[101] Waharte, S. and Trigoni, N. (2010). Supporting search and rescue operations with UAVs. In *International Conference on Emerging Security Technologies (ROBOSEC)*, pages 142–147, Islamabad. Institute of Electrical and Electronics Engineers.

[102] Waharte, S., Trigoni, N., and Julier, S. J. (2009). Coordinated search with a swarm of UAVs. In *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops, SECON Workshops 2009*.

[103] Wald, A. (1945). Sequential Tests of Statistical Hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186.

[104] Wald, A. and Wolfowitz, J. (1948). Optimum Character of the Sequential Probability Ratio Test. *Annals of Mathematical Statistics*, 19(3):326–339.

[105] Wald, A. and Wolfowitz, J. (1950). Bayes Solutions of Sequential Decision Problems. *Source: The Annals of Mathematical Statistics*, 21(1):82–99.

[106] Weiss, G. (2000). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press.

[107] Wymann, B., Dimitrakakis, C., Sumner, A., Espié, E., and Guionneau, C. (2015). TORCS: The open racing car simulator.

[108] Zurich, E. (2015). Rotors. https://github.com/ethz-asl/rotors_simulator. [Online; accessed 1/7/19].

Appendix A

Tables

A.1 Sequential Probability Ratio Test Tables of A and B as functions of the parameters α and β

Table A.1 A table of values of A=$\frac{1-\beta}{\alpha}$, where α is the significance and $1-\beta$ is the power of the Sequential Probability Ratio Test (SPRT)

β \ α	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
0.05	19.00	18.00	17.00	16.00	15.00	14.00	13.00	12.00	11.00	10.00
0.1	9.50	9.00	8.50	8.00	7.50	7.00	6.50	6.00	5.50	5.00
0.15	6.33	6.00	5.67	5.33	5.00	4.67	4.33	4.00	3.67	3.33
0.2	4.75	4.50	4.25	4.00	3.75	3.50	3.25	3.00	2.75	2.50
0.25	3.80	3.60	3.40	3.20	3.00	2.80	2.60	2.40	2.20	2.00
0.3	3.17	3.00	2.83	2.67	2.50	2.33	2.17	2.00	1.83	1.67
0.35	2.71	2.57	2.43	2.29	2.14	2.00	1.86	1.71	1.57	1.43
0.4	2.37	2.25	2.12	2.00	1.88	1.75	1.62	1.50	1.38	1.25
0.45	2.11	2.00	1.89	1.78	1.67	1.56	1.44	1.33	1.22	1.11
0.5	1.90	1.80	1.70	1.60	1.50	1.40	1.30	1.20	1.10	1.00

Table A.2 A table of values of B=$\frac{\beta}{1-\alpha}$, where α is the significance and $1-\beta$ is the power of the Sequential Probability Ratio Test (SPRT)

β \ α	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
0.05	0.05	0.11	0.16	0.21	0.26	0.32	0.37	0.42	0.47	0.53
0.1	0.06	0.11	0.17	0.22	0.28	0.33	0.39	0.44	0.50	0.56
0.15	0.06	0.12	0.18	0.24	0.29	0.35	0.41	0.47	0.53	0.59
0.2	0.06	0.12	0.19	0.25	0.31	0.37	0.44	0.50	0.56	0.62
0.25	0.07	0.13	0.20	0.27	0.33	0.40	0.47	0.53	0.60	0.67
0.3	0.07	0.14	0.21	0.29	0.36	0.43	0.50	0.57	0.64	0.71
0.35	0.08	0.15	0.23	0.31	0.38	0.46	0.54	0.62	0.69	0.77
0.4	0.08	0.17	0.25	0.33	0.42	0.50	0.58	0.67	0.75	0.83
0.45	0.09	0.18	0.27	0.36	0.45	0.55	0.64	0.73	0.82	0.91
0.5	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00

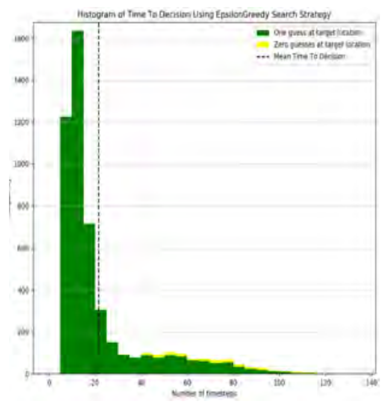
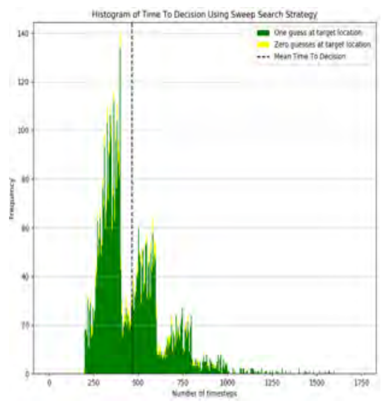
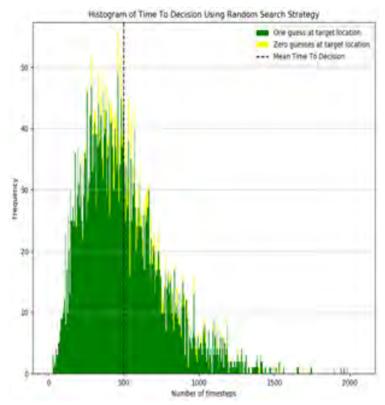
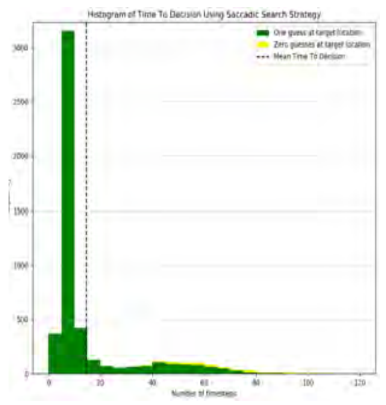
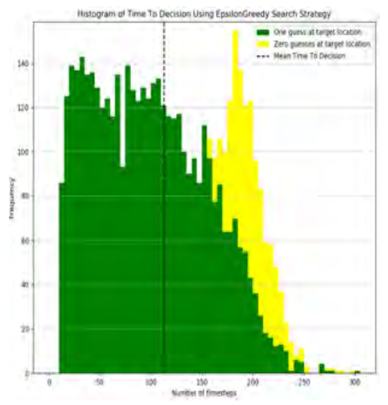
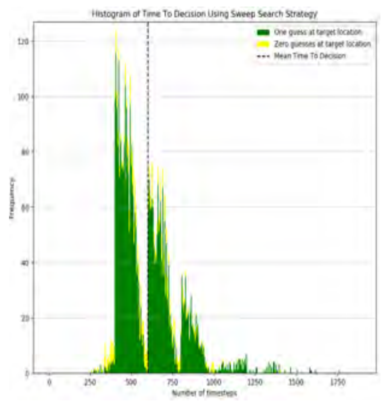
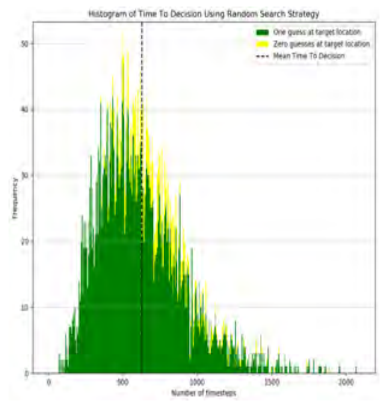
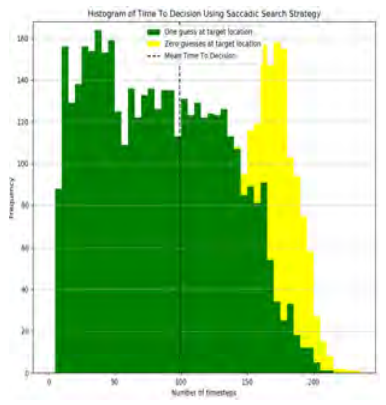
Appendix B

Histograms showing Time To Decision while varying search parameters

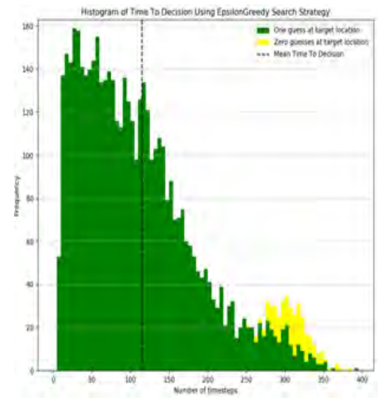
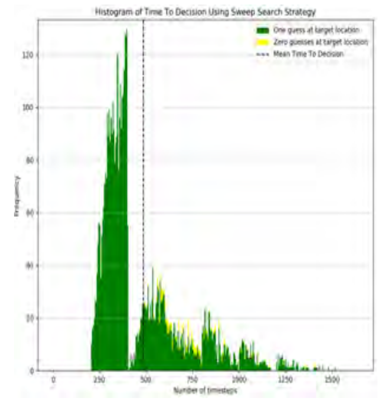
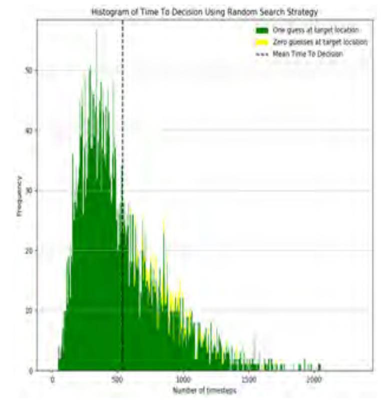
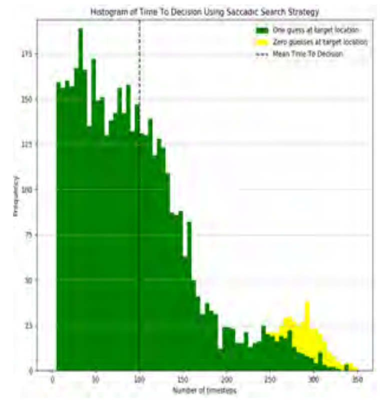
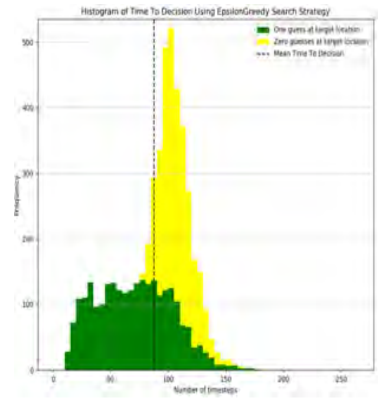
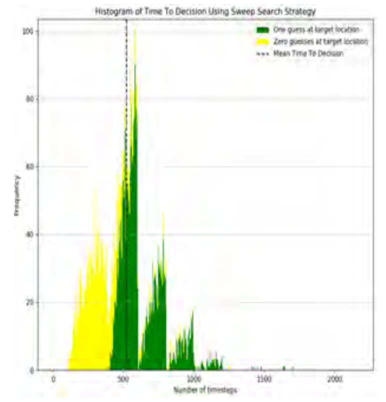
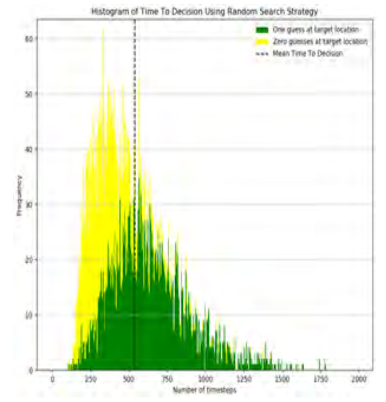
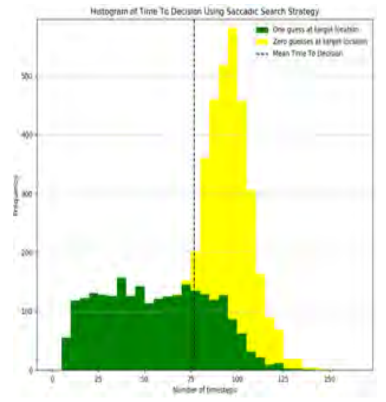
In order to generate the histograms displayed in this appendix, we used the following experimental setup:

We generated simulated sensor readings using arbitrarily chosen values of the false positive rate = 0.2 and a false negative rate = 0.15. For each simulation run, we generated random starting locations for the agents and targets in a uniformly spaced 10×10 grid. Unless specified otherwise, we use the following default parameters: sensor model false negative rate = 0.15, sensor model false positive rate = 0.2, initial belief distribution = uniform, initial cumulative belief target is present = 0.5, number of targets present = 1, number of active agents = 1, the ε-greedy search has ε=0.2 and a neighborhood radius of 4.

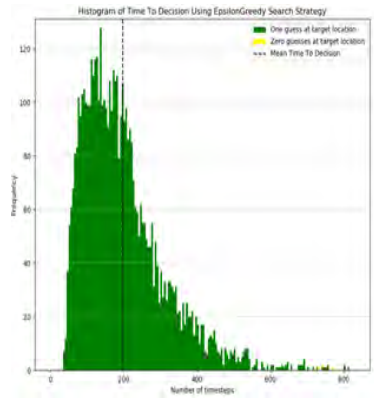
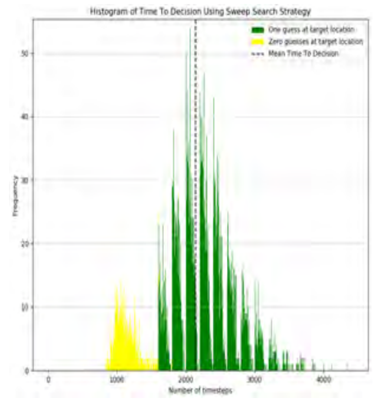
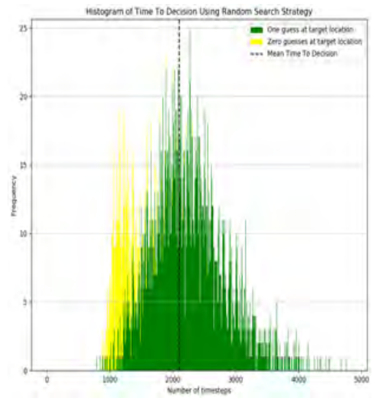
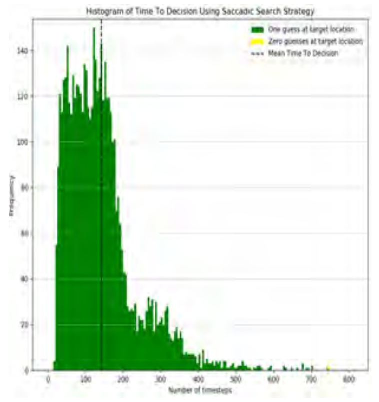
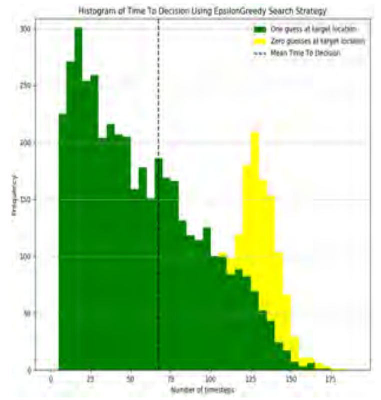
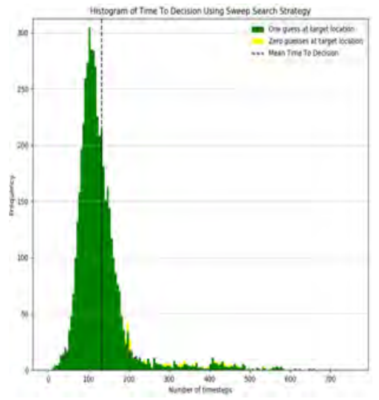
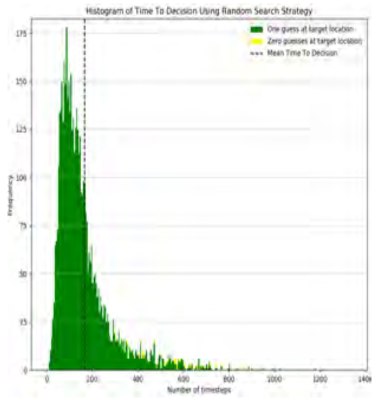
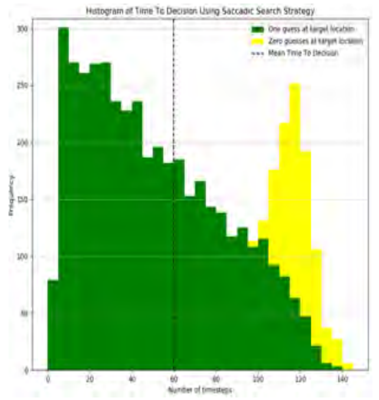
B.1 Histograms of Time To Decision for varying initial belief distributions

	ε-Greedy	Sweep	Random	Saccadic
Gaussian				
Uniform				

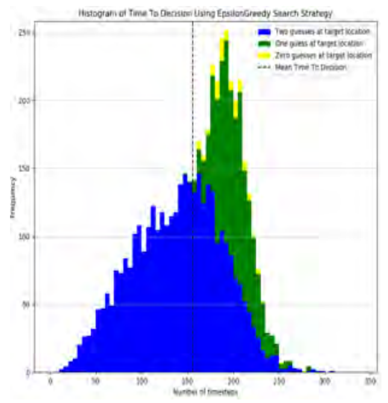
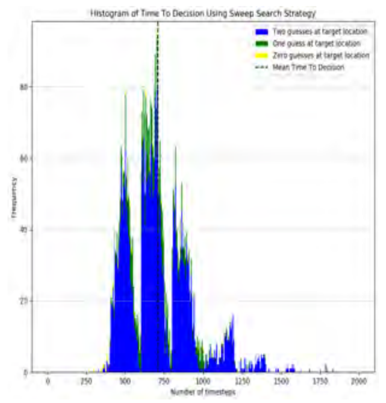
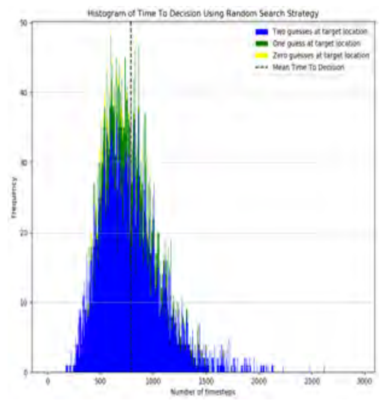
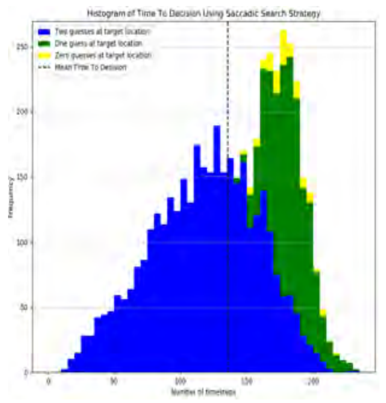
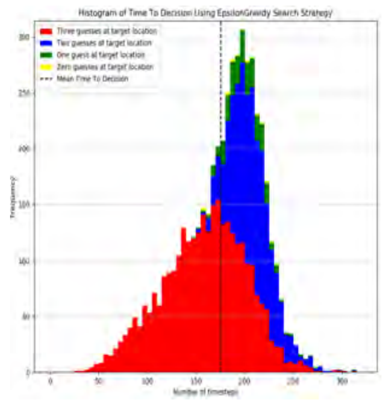
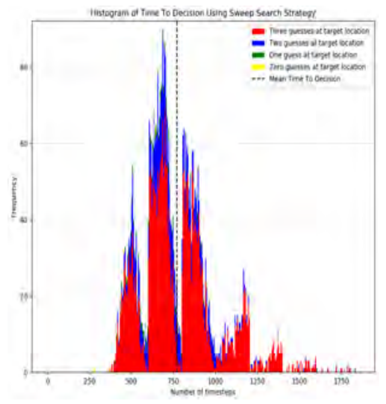
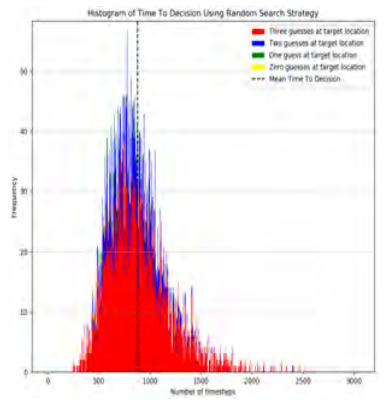
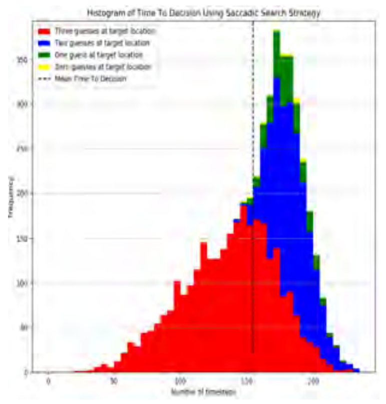
B.2 Histograms of Time To Decision (TTD) for a varying cumulative initial belief in target presence

	ε-Greedy	Sweep	Random	Saccadic
0.75				
0.25				

B.3 Histograms of Time To Decision (TTD) for varying sensor model parameters

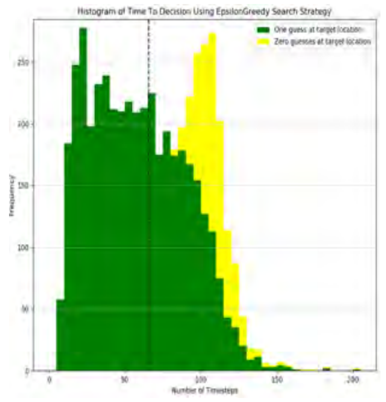
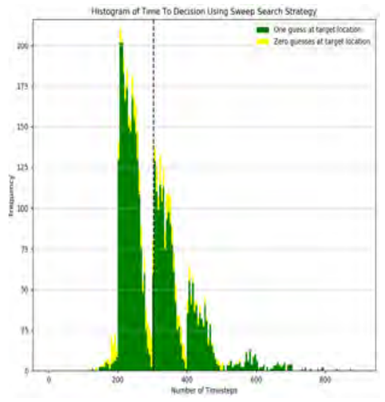
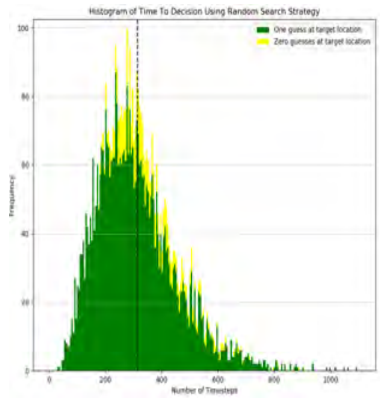
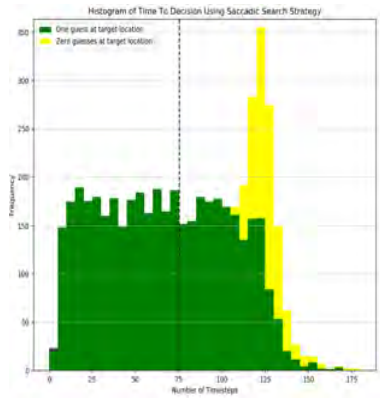
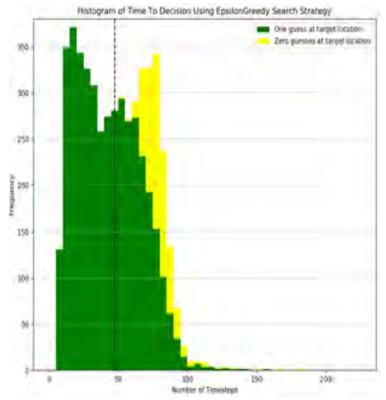
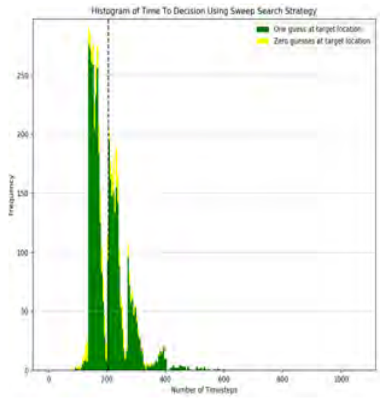
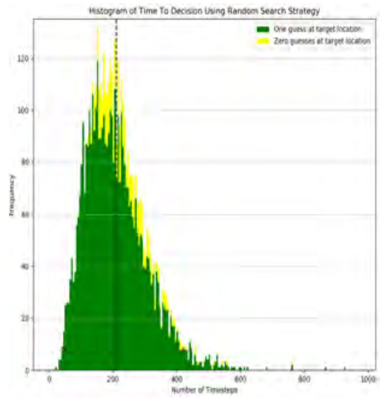
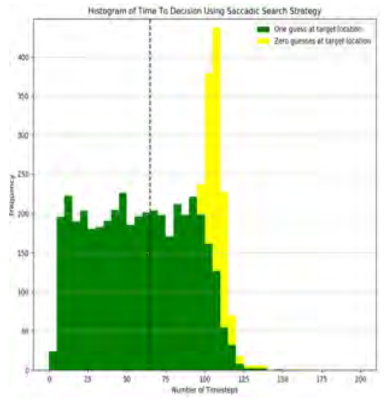
	ε-Greedy	Sweep	Random	Saccadic
0.4/ 0.4				
0.05/ 0.02				

B.4 Histograms of Time To Decision (TTD) for a varying number of targets present

	ε-Greedy	Sweep	Random	Saccadic
2				
3				

labeltable:HistVaryingNoTargetsPresent

B.5 Histograms of Time To Decision (TTD) for a varying number of search agents

	ε-Greedy	Sweep	Random	Saccadic
2				
3	 Histogram of Time To Decision Using EpsilonGreedy Search Strategy One guess at target location Zero guesses at target location Frequency Number of Timesteps	 Histogram of Time To Decision Using Sweep Search Strategy One guess at target location Zero guesses at target location Frequency Number of Timesteps	 Histogram of Time To Decision Using Random Search Strategy One guess at target location Zero guesses at target location Frequency Number of Timesteps	 Histogram of Time To Decision Using Saccadic Search Strategy One guess at target location Zero guesses at target location Frequency Number of Timesteps