Informative Path Planning for the Monitoring of Pathogens and Weeds

van Esch, Hilde

Award date:
2023

Master Thesis Report

Informative Path Planning for the Monitoring of Pathogens and Weeds

University supervisor:	Dr. D. J. G. T. Antunes
University supervisor:	Ir. R.M. Beumer
External supervisor:	Ir. N.L.M. Jeurgens

H. van Esch - 1306219

MSc Systems & Control
45 ECTS

Control Systems Technology
Department of Mechanical Engineering

Eindhoven, August 16, 2023

Abstract

The increasing global population coupled with the climate crisis necessitates innovative agricultural approaches to ensure food security. To address resource-intensive farming practices and make agriculture more ecologically sustainable, the Synergia project aims to enhance efficiency and inclusivity while promoting ecological balance. The project partners, Eindhoven University of Technology and Avular, collaborate to tackle challenges through technology-driven precision farming.

Precision farming is a key strategy in addressing these challenges, enabling enhanced production yield with limited resources. Autonomous robots form part of this solution, offering lightweight alternatives that prevent soil compression and reduce farmer workload. Precision monitoring through informative path planning forms an essential part of precision farming, facilitating accurate world modelling necessary for precision control, while minimizing costs by being applicable to a single autonomous mobile robot with limited battery capacity.

This thesis focuses on a path planning algorithm informed by a spreading model for monitoring pathogens and weeds in agricultural fields. The research presents a modular software architecture that enables adaptable and interdisciplinary development. The spreading module leverages spread models and predictive entropy maps to guide the robot's trajectory. The core path planning module incorporates informed path generation to optimize monitoring efficiency while adhering to budget constraints. The monitoring module utilizes the observed data to refresh the world model.

Simulations evaluate the algorithm's performance, comparing it to an uninformed planner. The results demonstrate the informed planner's substantial improvement in monitoring efficiency, confirming its effectiveness in reducing entropy and enhancing accuracy. Moreover, the research addresses additional aspects, including budget levels, kinematic constraints of varying robot types and non-greedy horizon planning, which are addressed through development of new approaches and analysis providing insights into their effects on monitoring performance.

Future research directions include exploring alternative planners for structured fields and real-life testing on actual agricultural fields. The thesis underscores the importance of bridging the gap between agricultural and technical domains and suggests further advancements in spreading models to enhance disease monitoring accuracy. This work establishes a foundation for path planning in pathogen and weed monitoring, contributing to precision farming and sustainable agriculture practices.

Contents

1 Introduction

The growing world population combined with the climate crisis requires innovation in agriculture to be able to feed all people. Current resource-intensive farming methods are required to change such that crop yield increases while minimizing the use of resources such as water, herbicides and pesticides (Cobbenhagen et al., 2021). The goal of the Synergia project is to make agriculture more ecological and sustainable, such that it is efficient and nature-inclusive (Wageningen University & Research, 2019). Currently, most Dutch agriculture consists of monocrop fields, which are prone to diseases and pests. Additionally, mono-cropping leads to soil depletion and thus lower crop yield. A solution to this problem is intercropping, where different types of crops grow on the same field, creating a more sustainable and resilient system with higher crop yield through natural symbiosis (Cobbenhagen et al., 2021). However, this scenario requires more knowledge and more physical work for employees in agriculture. Moreover, arable cropping often still employs heavy machinery that compacts the soil, leading to reduced fertility, increased risk of erosion and complex recovery (Hameed, 2018). Replacing the heavy machinery such as tractors with smaller machines leads to longer operating times to cover the same area, increasing labor costs. The Synergia project includes Eindhoven University of Technology and Avular as partners to address both of these problems. Using technological innovations to apply precision-farming, physical work can be reduced while agriculture becomes more resilient and sustainable.

One specific topic in this challenge is the ability to monitor and treat undesired activity on the field that forms a hazard for the crops, including pathogens and weeds (including fungi, bacteria, viruses and pests). For readability and conciseness, in the rest of the thesis all of these hazards are referred to as pathogens. Generally, it requires much attention, knowledge and physical work to keep these pathogens under control. Instead, employing robotics and intelligent systems can provide a solution. To enable precision treatment, a precise mapping of the pathogens must be created through monitoring processes. Planning an effective path for monitoring by autonomous robots is a complicated task, due to the changing information and stochastic processes, combined with limited path length budget.

This thesis is focused on creating an informed path planning algorithm based on a spreading model for monitoring pathogens. Around the path planning algorithm itself, a software architecture was created. This architecture enables usage of a modular system where each module can be adapted by experts based on the applicable knowledge, without requiring knowledge of all modules of the system. First of all, this architecture supports development and testing of the path planner in a constructive way throughout the project. Secondly, this architecture allows experts from different disciplines to complement each other and stimulates state-of-the-art developments and use-case specific adaptations. Aside from the planning module, this architecture includes a spreading module and a monitoring module. The architecture enables realistic design, simulation and verification of the planner.

The path planning applies to a mobile robot in a field that can be either unstructured or structured (with predefined rows). In unstructured fields, the robot is allowed to drive in any direction and location, as long as it does not cross obstacles or the field borders. In structured fields, the robot is only allowed to move in the direction of the rows. The goal of the planning algorithm is to find an efficient path to monitor the field for pathogen spread. To enable informed path planning, the growth and spread of pathogens and weeds are modelled based on approaches proposed in the literature and taken into account to predict the current state of the pathogens. A spatial map describing the entropy is derived from the stochastic spreading model, which characterizes the uncertainty or variation there may be in the spread. The models for spread are applied in the spreading module. The planned path maximizes the information function based on the monitoring goal while remaining within a budget of a maximum path length, which can be based on battery capacity or maximum available time. The monitoring goal is to reduce the entropy of the world model, thus yielding an accurate map of the pathogens, which could be used for precision treatment. The planning algorithm will be tested in a simulation environment. The planner is compared to an informed planner in terms of performance, which shows the positive effect of informed planning. The uninformed planner has no input from the current state of the spread of the pathogen and the corresponding entropy.

The larger outlook is to apply this algorithm to different kinds of fields, both structured with rows or completely unstructured. It can then be applied both to strip cropping, the form of intercropping where the division in rows still exists, and full intercropping. Additionally, the algorithm can be applied to fields of any shape and form, i.e. including obstacles, and fields which either a convex or concave shape. Furthermore, the kinematic constraints of different types of robots are taken into account for the path planning, such as the driving platform.

Moreover, from a high-level perspective, the main goal the thesis takes a step towards is: "Designing a planning agent that takes into account information from spreading models such that it can monitor the spread of pathogens and weeds in crop fields while minimizing entropy". Additionally, as a consequence of applying an informed planner in a modular fashion, the following goal is pursued: "Creating a modular software architecture for autonomously monitoring the spread of pathogens and weeds in crop fields". However, this goal serves mainly as a means to an end, in order to properly construct, simulate and test the planning agent.

The report is divided into the following chapters: background, architecture, spreading module, path planning module, monitoring module, results and conclusion. The background (Chapter 2) describes the context of the project and the state-of-the-art literature. The broad structure of the project and the corresponding modules are introduced in the architecture chapter (Chapter 3). The chapter of the spreading module (Chapter 4) describes the implemented spreading models and their relevance for the informed planner. The chapter of the path planning module (Chapter 5) describes the basis of the path planner and the contributions made to make the planner efficient and applicable to the researched topic. The functions within the planning module are described in most detail, since this forms the core of the project and contains the most significant work related to the research question. The work on the planner forms a contribution to the current literature, as it improves upon the current algorithms available and solve the unique challenges of this project. The effect of the planned path on the world model is explained in the monitoring chapter (Chapter 6). The chapter on updating the world model (Chapter 7) describes how the modules together form a full cycle which can be repeated to form a multi-day simulation. The results (Chapter 8) describe the testing cases and outcomes of the simulation tests on the planner. Lastly, the conclusion (Chapter 9) forms an answer to the research questions and lists suggestions for future research.

2 Background

In this chapter, the context of the project and the relevant literature is presented. The context consists of the Synergia project and the project partners. The literature review dives into state-of-the-art research on several aspects of the research topic.

2.1 Context of the Project

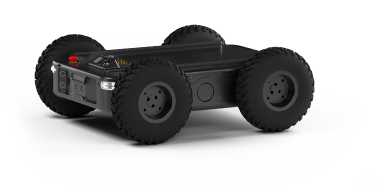
The thesis takes place within the Synergia project. The Synergia project is an interdisciplinary project that includes five Dutch universities, including Wageningen University & Research and Eindhoven University of Technology. Additionally, it includes several partners in research and business. It was funded by NWO subsidy in 2019 (NWO, 2019a).

Synergia stands for 'SYstem change for New Ecology-based and Resource efficient Growth with high tech In Agriculture'. The project is aimed to develop production systems in agriculture that form a solution to the ecological and social challenges that are faced NWO, 2019b). As the name indicates, the project is based on the concept of 'technology-for-ecology (T4E) based farming' (NWO, 2019a).

In total, the project contains three use cases: horticulture, dairy farming and arable crops. This project takes part in the latter. The arable crops use case entails open field farming, where robotics is supposed to play a role in the form of precision operations (spatial and temporal) (van Mourik, 2019).

Avular plays an important role as a partner of Synergia in this use case. Avular is an innovative mobile robotics company with a focus on custom mobile robots, robotic platforms and robotic software (Avular, 2023). In collaboration with Eindhoven University of Technology they develop new technologies in the form of software and hardware that can be applied to the challenge. Previously, research on localization and driving through field rows was executed through this collaboration.

Furthermore, Avular has had developments independent of Synergia that are applicable to the project. This includes the design of their ground robot, the Origin One, which has a use case to detect and treat diseases in potato fields (see Figure 1a). The Origin One is a driving platform, specifically designed for autonomous ground operations. The robot is built in a modular fashion, making it easily adaptable for specific purposes (Avular, 2023). One of the specific purposes led to the Potato Origin One, where the Origin One has been put on a high frame with outdoor wheels such that it can drive through potato fields (see Figure 1b). Additionally, it is by standard applicable to outdoor purposes due to the four motor-driven wheels with all-terrain tires.


(a) Origin One

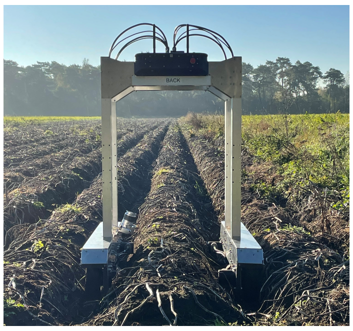

(b) Origin One for potato treatment

Figure 1: Avular robots (Avular, 2023)

2.2 Literature Review

There are different aspects within the research topic that require insights from previous literature. Literature on growth and spread of pathogens and weeds enables establishment of a model that serves both as information to the planner and as a basis for testing the planner in simulation experiments. The established model is not meant to be necessarily the most precise model possible for each species of pathogens or weeds in itself, but should represent a broad range of species where the model can be adapted according to specific species characteristics. The relevant literature on topics related to the spread of pathogens and weeds are discussed in Subsection 2.2.1. In Subsection 2.2.2, the literature review delves into the path planning algorithm, examining the specific type of problem being addressed and exploring the applications of various related algorithms in previous studies.

2.2.1 Spread of Pathogens and Weeds

A model that predicts and simulates the spread of pathogens and weeds should be generalizable to many species to enable wide applicability. Additionally, it should be structured in a way that facilitates its utilization by a path planner for informative purposes, encompassing both spatial and temporal aspects. The spatial simulation determines the distance and direction of spread, and allows for generation of a map which is employable by the path planner. The temporal simulation determines the speed of reproduction, and allows for prediction of spread over time.

It is vital to differentiate between pathogens and weeds within the context of this study, due to the primal differences in the spread characteristics. Pathogens, such as bacteria, viruses, fungi, and pests, directly jeopardize plant health. In contrast, weeds are unwanted plants competing for crop resources. While there are more distinctions within the domains of pathogens and weeds, within the scope of the project only two distinct models are implemented. These two models cover a wide range of diseases to be monitored and form a show-case for the modular structure of the architecture.

The growth of weeds is characterized by several influential factors. Most weed species generate populations that are unevenly spatially distributed over a field; weeds tend to grow in patches (Somerville et al., 2020). The weed often grows from one or more infested locations, from which it spreads non-uniformly, influenced by environmental, spatial and random factors (Doyle, 1991). The distribution of the weed can be modelled using the seed dispersal characteristics, which indicate the speed and range in which the weed spreads. The extent to which the weed spreads is also influenced by competition for resources, both with the species itself and with other plant species. As there are many factors that contribute to the spread of the weed, a lot of variability occurs which can be modelled by stochastic processes (Somerville et al., 2020).

To simulate the growth and spread of weeds in crop fields, the INVADE model offers a valuable approach. INVADE is a model of plant spread, simulating weed growth both spatially and temporally (Auld and Coote, 1990). The spatial model adopts a grid-based structure, with each cell measuring 1 square meter in size. Each grid point contains information on the state, which consists of the plant population density. The model is an extended version of their earlier model (Auld and Coote, 1980), which includes population growth rate (at a focus), proportion of the annual population increase dispersed from a focus, distance over which the dispersed fraction could occur and susceptibility of invaded areas to colonization. INVADE builds upon that by adding contributing factors such as the initial number of infested cells, the maximum population per cell (saturation value) and dispersion characteristics of the plant. The model employs stochastic processes to capture the variability observed in pathogen spread.

For pathogens, there are many factors influencing the spreading model. Most importantly, it depends on the type of pathogen, being either a bacterium, virus, fungus, or pest. There are four general influencing factors to consider for pathogens: life cycle of the pathogen, susceptibility of the host plant, reproduction rate and the dispersal. The life cycle contains a latent period in which a pathogen is infecting the plant but is not visible yet (Papaix et al., 2014), thus a so-called infection period. The susceptibility of the host plant describes the extent to which the plant is able to resist the pathogen. The reproduction rate indicates how many other plants are infected by a single infected plant. The dispersal indicates the range of the spread. For fungi, the most prominent factor influencing the spread is whether the fungus is of a shoot-infecting or root-infecting type. Root-infecting fungi spread below ground and therefore have limited dispersal range. On the contrary, shoot-infecting fungi have airborne spores which spread more easily over large areas (Van Agtmaal et al., 2017). Additionally, there are a number of less prominent influencing factors on the spread of fungi, such as competition for resources, external influences on the field, suitable host plants and the cropping history (Van Agtmaal et al., 2017).

The literature on the spread of pathogens provides valuable insights for establishing a model that predicts and simulates pathogen growth and spread. To ensure broad applicability, it has to account for many characteristics that are specific to each pathogen species. EPIMUL is a model that enables pathogen spread simulation on the scale of fields, which includes configurable variables for many of the influencing factors. EPIMUL offers a reasonable mathematical approximation, both temporally and spatially, that can be incorporated in simulation experiments. Although EPIMUL simplifies reality to some extent, it allows for the representation of diverse pathogen species by adapting the model according to their specific characteristics (Kampmeijer and Zadoks, 1977). Other models for simulation of pathogen growth are more limited, either lacking the temporal aspect, or are applied to bigger scales, such as entire landscapes (Villette et al., 2021,Papaix et al., 2014).

2.2.2 Path Planning

This section delves into the three ways that were found in which the path planning problem for pathogen and weed monitoring could be characterized: as (a variant of) the Traveling Salesman Problem (TSP), as an informative path planning problem or as a coverage problem. When considering it a TSP, the field is only partially covered by the path and the goal of the planner is to find the path with minimal cost between the selected points, or nodes. When regarding it as an informative path planning problem, the field is also only partially covered due to the limited budget, but the goal now is maximize an utility function based on the state of the field, such as entropy or probability of weeds and pathogens. The other option is to consider a coverage planner, which either covers the entire field or a predefined section of the field.

Travelling Salesman Problem The TSP refers to a problem where the shortest path is planned between the predetermined node locations, based on the challenge a salesman faces when going from town to town to sell his products, while ensuring that all towns are visited. As such, the solution to such a problem covers the selected node locations and parts of the field that are crossed by the paths between nodes. Commonly, the Concorde TSP Solver and Genetic Algorithm are applied to find a solution to the TSP. However, Xiong et al. (Xiong et al., 2017) proposed a less computationally expensive algorithm that is based on segmentation of the points with respect to their location relative to the 2D axes defining the field. This algorithm is not optimal, but still close to optimal and more suitable for real-time processing. The orienteering problem, or selective TSP, is another approach to the problem, which entails that not all nodes are visited but only the most profitable ones due to limitations of travel time (Vansteenwegen et al., 2011), where the aim is to maximize profit by selectively visiting nodes. The orienteering problem has been solved using different methods, which can be characterized into exact solutions and algorithms with heuristics (Vansteenwegen et al., 2011). Another relevant variant of the TSP is the Tourist Trip Design Problem (TTDP), which takes into account both the travel time and the time at the node. In the scope of the project, the time at the node could represent the observation at the location of the weed (Ruiz-Meza and Montoya-Torres, 2022). A variant of the TTDP is the Cruise Itinerary Problem

(CIP), which considers both the stops and the scenic route, where the speed along the route is considered based on the value of the scene (Ruiz-Meza and Montoya-Torres, 2022). In the scope of this project, this could be useful as it enables finding the most optimal route between points that needs to be monitored, where the route entails as much exploration of the field as possible. In the literature, the TTDP and CIP variants of the TSP are not considered for agricultural purposes, but mainly for the purpose of planning the most optimal route along touristic travel points. They do, however, show relevant similarities in the defined problem.

Informative Path Planning Informative path planning (IPP) entails "having a robot autonomously decide what path to take while collecting measurements, based on a probabilistic model of the quantity being studied" (Binney and Sukhatme, 2012). The objective is to maximize the observed sensor information during the chosen path, with the purpose of accurate monitoring. A method for solving this type of path planning is the branch and bound algorithm, previously designed based on feature subset selection literature, which yields the optimal solution (Binney and Sukhatme, 2012). Alternatively, a history-aware algorithm based on RRT has been applied to the problem (Witting et al., 2018), which is more advanced but also more complicated to implement. A combination of these methods led to development of the Rapidly-exploring Information Gathering (RIG) algorithm, which employs aspects of RRT combined with branch and bound to maximize information gain while remaining within a given path length budget. The RIG algorithm is sampling-based and created with a focus on scalability and motion constraints. It was designed based on the research topic of autonomously gathering data with a submarine in a lake. The algorithm computes a path offline, based on the currently known information (Hollinger and Sukhatme, 2014).

Coverage Planning "Coverage Path Planning (CPP) is the task of determining a path that passes over all points of an area or volume of interest while avoiding obstacles." (Galceran and Carreras, 2013), where approaches aim to fulfill this task in the most efficient way. Some common applications for coverage planners are cleaning or sweeping robots, lawn mowing robots and monitoring tasks. Additionally, coverage planners have also been applied in agricultural settings, such as in a multi-robot system aimed to replace big tractors with multiple smaller robots (Hameed, 2018). An important challenge for coverage planners is the complexity of the search space, which grows exponentially with an increase in area to be covered. This increase in complexity often leads to problems due to the limited computational capacity of mobile robots. The informative path planning problem can also be described as a budgeted coverage problem, where the chosen path is limited to a certain cost, mostly indicated by the path length.

Selected Approach The second approach, informative path planning, is chosen as the framework for defining the monitoring problem, since it was deemed the most applicable based on the relevant literature and the objectives and constraints of the research topic. The advantage of the IPP approach is that it is by definition focused on maximizing information gain and limited by a budget. Furthermore it does not require predefined node positions (as is the case with the Traveling Salesman Problem) or predefined feasible sections (as is the case with coverage problems). Furthermore, the RIG algorithm shows potential for the research topic, as it reveals many similarities in constraints and problem definition, such as large search space, spatial information, monitoring task and possible kinematic constraints. The RIG algorithm has not been applied to other research problems yet.

3 Architecture

The architecture for this project consists of different modules, each with a specific purpose. These modules are interconnected, but operate individually such that each module is adaptable without disrupting the other modules. This structure was chosen for the purposes of practicality and interdisciplinarity. The practical reason is that constructing and evaluating each module is more effective and simple. Interdisciplinarity allows experts to make changes easily and implement knowledge into a relevant module into the system without knowledge of every module, e.g. agronomists who can improve the spreading model without any path planning knowledge. This enables the incorporation of expert knowledge from both the agricultural and technical sectors, and the closing of the gap between these sectors.

The architecture consists of three different modules, each depending on the outputs of another module. The planner takes the entropy grid map from the spreading module into account to create an informed path through the discretized field map. This entropy map is generated based on the prior knowledge from previous measurements and the spreading models of the pathogens and weeds. Throughout all modules, the choice has been made to use a grid-based representation for the spatial distribution of pathogens and weeds and the corresponding uncertainty. This allows for easy transmissions of information and transitions between modules. In the basis, the entropy map can be interpreted as an intensity map describing the uncertainty that follows from the prediction of spread of the weeds and pathogens. The uncertainty value of a grid cell represents the possible deviation that the density value of the cell may have in the spreading map. The planner is designed with the aim to be applicable to Avular's Origin One robot, but is also generalized such that it is applicable to multiple types of robots, with different kinematic constraints. Additionally, it is applicable to multiple field types, both unstructured and structured by predefined rows. The spreading models were created based on expert knowledge from previous work in the literature, which contains the required knowledge to establish a realistic model. In these spreading models, variables such as the dispersal range, the spreading speed and other variables defining context and type will be adaptable such that it is applicable to different types of pathogens and weeds. The monitoring module takes in the planned path and executes it, hereby gathering status information, allowing an update of the world model. Based on the observed information, the spreading module can then predict the new spread. A cycle can be created simulating long-term treatment over a period of time, where the robot monitors the field periodically, updates the state of the field with the monitored information, plans the most optimal path based on the current state, and then repeats the planning step.

The modules and the interplay between them are visualized in Figure 2 and will be further explained in the following sections. A variable overview is given in Table 6 in Appendix A, which explains the most common variables used throughout the modules. Figure 2 also shows the inputs and outputs of each module, which are as follows:

1. The robot world model containing the states of the grid points and their uncertainty describing the predicted state of the crop pathogen spread
2. The generated path in terms of a series of grid points
3. The status of the observed grid points

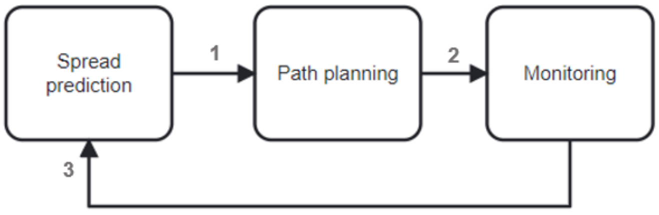


Figure 2: Flowchart of the modules

4 Spreading Module

Two models have been developed that allow simulation of the growth and spread of pathogens or weeds, respectively. These two models are different, since pathogens and weeds show differences in their dispersal, as discussed in Section 2.2.1. The models are based on expert knowledge from the agricultural field instead of using a machine learning approach to learn a model. The motive for this is the explainability of the outcomes of the model, which is lacking in machine learning approaches, though deemed necessary to be able to apply it to practical implementations (Cobbenhagen et al., 2021). As such, the spreading module bridges the gap between the agricultural and (agri)technical sectors. This chapter dives into the models and highlights the contributions made in this thesis to the original models. For details on the implementation of the models, the reader is referred to Appendix B, which provides explanation of the models through pseudo-code and further descriptions.

The pattern of spread in both of these models is similar in spirit to the mechanism as proposed by Antunes et al. (2022), though here the expected values do not follow from a Markov Chain model, but from a given rule. The expected value of weeds or pathogens in a grid cell is dependent on the values of surrounding grid points. The new grid cell value depends on its previous values and the incoming spread from other grid cells. The incoming spread from surrounding grid cells consist of a weighted sum of each current grid cell value mediated by a spreading factor and the distance between the cells:

$$E_{(x_i,y_i)} = \min(E_{max}, V_{x_i,y_i} + \sum_{x=x_i-r}^{x_i+r} \sum_{y=y_i-r}^{y_i+r} (f \cdot V_{x,y} \cdot e^{\frac{-log(2)\cdot d}{r}}), \tag{1}$$

where $V_{x,y}$ is the current density value of a grid cell, $E_{x,y}$ is the new density value of a grid cell, E_{max} is the saturation level of the density, f is the spreading factor, d is the distance between (x, y) and (x_i, y_i) and r is the spreading range.

The spreading factor differs for each type of pathogen and weed. Additionally, the range of spread could be smaller or larger per type of pathogen or weed, such that not only the direct neighbours, but also further neighbours have an influence on the expected value of a grid point. The distance has an exponentially decreasing effect on the spread of pathogens and weeds.

The spreading model for weeds is based on the INVADE model (Auld and Coote, 1990). It takes into account context variables and weed-defining variables. These context variables and type-defining variables together determine the previously mentioned spreading factor and spreading range. The context variables indicate the current state of the infection. The weed-defining variables allow to model different types of weeds into a single model using the most impacting spreading features. An overview of the context and weed-defining variables is given in Table 1. The reader is referred to Appendix B for deeper explanation on the application of these variables.

Table 1: Context and Weed-Defining Variables (INVADE)

Variable type	Variable	Description
Context	patchnr	Number of patches the infection starts with
Context	patchsize	Maximum current size of patches
Weed-defining	reproductionrate	Speed of reproduction
Weed-defining	spreadrange	Range of dispersal (r)
Weed-defining	plantattach	Boolean for crop attachment (i.e. parasitic relation with the crops or competing with the crops for resources)
Weed-defining	saturation	Maximum density of weeds (plants per grid cell) (E_{max})

The spreading model for pathogens is based on the EPIMUL model (Kampmeijer and Zadoks, 1977). Similarly to weeds, it takes into account context variables and pathogen-defining variables, though the exact variables are slightly different. An overview of the context and pathogen-defining variables is given in Table 2.

Table 2: Context and Pathogen-Defining Variables (EPIMUL)

Variable type	Variable	Description
Context	patchnr	Number of patches the infection starts with
Context	duration	Current duration of the infection
Pathogen-defining	reproductionrate	Speed of reproduction
Pathogen-defining	fraction	The fraction of pathogens that reproduce within the reproduction time
Pathogen-defining	spreadrange	Range of dispersal (r)
Pathogen-defining	saturation	Maximum density of pathogen (E_{max})

Both of these expert-based models are grid-based and allow both spatial and temporal simulation. Additionally, the models incorporate the context and type-defining variables which allow the model to be realistic for many types of pathogens and weeds, thus enabling a generalizable and accurate simulation. For detailed descriptions of the models, readers are referred to the corresponding papers on EPIMUL (Kampmeijer and Zadoks, 1977) and INVADE (Auld and Coote, 1990).

One important feature of the spreading model is the stochasticity. EPIMUL is deterministic in nature as it only outputs the expected pathogen density based on the given variables. INVADE is stochastic in nature, since the reproduction rate is given by a normal distribution with a specified standard deviation. The pathogen spreading model was adapted such that the reproduction rate was also represented by a normal distribution. This was accomplished by introducing a standard deviation, such that the reproduction rate is computed by taking a number from a normal distribution defined by the average (deterministic reproduction rate) and standard deviation. As such, both of the models are stochastic in nature, which allows to obtain a certain entropy from one moment in time to the next. This entropy value is computed by applying the standard deviation to the current spread value (see Appendix B, e.g. Listing 5 line 3).

The spreading model is a form of model-based crop growth management where a digital twin of the agricultural field is created in the form of a world model. With this approach, crops are monitored using a simulation of their growth, which predicts the state based on observations (Cobbenhagen et al., 2021). This approach allows to predict the spread of pathogens and weeds and therefore aids the decision-making process for the best control actions, as the prediction of the spread forms a source of information for the planner.

The output of the spreading model is a world model consisting of a matrix containing the status of all grid cells of the map. The status of each grid cell contains the predicted amount of weed or pathogen. Additionally, each grid point contains a level of uncertainty, representing the possible variation as predicted by the spreading model. This uncertainty characterizes the variability in possibility of spread from one observation to the next, as represented by the stochasticity in the reproduction rate. Of course, the uncertainty also increases whenever a location is unobserved for a longer time, to take potential new infections into account. As such, a world model in the form of two intensity maps (one of the actual spread and one of the uncertainty) are retrieved as outcome of the spreading module. The uncertainty intensity map serves as information to the path planner such that it can generate an informed path.

5 Path Planning Module

The planner consists of an informed path planning algorithm. It uses the information map provided by the spreading module to maximize the information gain while remaining within a specified budget, i.e. the maximum path length. This path is represented by a consecutive array of grid cells, which can then be used as input for the monitoring module (see Figure 2). The planner is based on the Rapidly Exploring Information Gathering (RIG) algorithm (Hollinger and Sukhatme, 2014), with several adaptations and additions. These adaptations and additions form the contribution of this project to the current literature, as these improve upon the current algorithms available and solve the unique challenges of this project. In this section, the RIG algorithm will be described briefly, after which the adaptations and additions made to it will be discussed in the following sections. For variable definitions, consult Appendix A.

5.1 Rapidly Exploring Information Gathering Algorithm

The RIG algorithm is a sampling-based algorithm. It combines aspects of the RRT* algorithm and the branch and bound technique. It creates a tree-structure, where each node represents the path from the start up to a location. Each node has a location, a parent and the cost and information value of the path. The pseudo-code of the algorithm as introduced (Hollinger and Sukhatme, 2014) can be found in Listing 1. It should be noted that the pseudo-code in Listing 1 shows slight differences with the original algorithm due to contributions of this thesis.

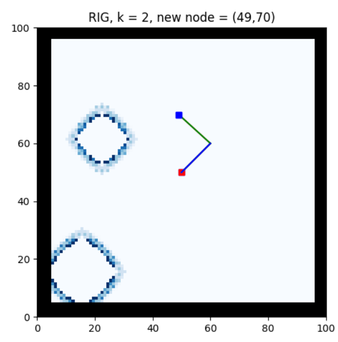
Sampling new locations happens through the method of RRT*: a new location is randomly sampled (function Sample), then the closest existing node to the sampled location is determined. When the distance between these locations is within the maximum specified step length, the sampled location will be the location of the new node. If it is not, the location will be moved closer to the closest node (function Steer). The Sample and Steer steps are visualized in Figure 3. Once this new location is determined, all existing nodes that are within the step length distance of the location will form a connection whenever feasible (i.e., a path without collision can be formed). Each connection then leads to the creation of a new node, with the determined location and the connected node as parent. There can thus be multiple nodes at a single location, with different paths leading to it. This is visible in Listing 1, where in Line 19, a new node is created at a given location with each a different near node as parent (thus with a different path leading up to it), also see Figure 4. This pattern of node creation leads to the aforementioned tree-structure. In contrast to algorithms that only extend to direct neighbours, such as A*, the random sampling method allows for quicker exploration of a large search space. RRT* is an optimized version with respect to classic RRT, as it has the ability to reroute paths. This rerouting of paths is the rewiring function, which will be described in more detail in Section 5.4.

RRT* ensures a tree-structure entailing candidate trajectories. To find the path with the best information value and the costs within the budget, branch and bound is applied. Branch and bound ensures that a node is only extended (connected to a new node) whenever the total cost of the path does not exceed the budget, and pruned otherwise. The total cost is the sum of the cumulative cost (of calculated distances) from start to current node and the cost from the node to the goal position. In addition, pruning is applied to reduce memory usage and run-time, which will be discussed in detail in Section 5.5, see also Figure 4.

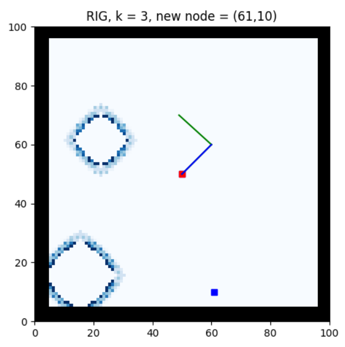
Finally, the information value is evaluated for each node, where the node with the highest information value is selected. The final path is then determined by tracing back the parents of that node.

Listing 1: RIG Pseudo-Code

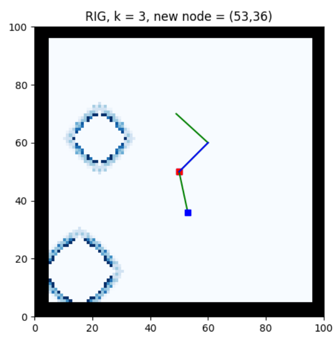
```
def Planner(config_vars, info_map):
    x_nodes=[x_start] # All nodes
    x_soln=[] # Feasible solutions
    # Iterate until the maximum amount of iterations or the stopping
        criterion is reached
    k=1
    while k<=max_iter and stop=False:
        # Randomly sample a new location:
        x_rand = Sample()
        # Connect the location to the nearest node
        x_new = Steer(x_rand)
        # Create connections to the new location from every node within
            reach
        for x_near in Near(x_new):
            # Parent of the node, defining the path
            parent=x_near
            # Cost and info of the path up to the new location
            cost=x_near.cost + Dist(x_near,node_new)
            info=x_near.info + Info(x_near,node_new)
            # Creating the new node
            node_new = Node((x_new.x,x_new.y),parent,cost,info,totalcost
                ,totalinfo)
            x_nodes.append(node_new)
            # Total cost and info includes the part to the goal location
            totalcost = node_new.cost + Dist(node_new,x_goal)
            totalinfo = node_new.info + Info(node_new,x_goal)
            # New node is a feasible solution if within budget
            if node_new.totalcost<=budget:
                x_soln.append(node_new)
        # Non-promising co-located nodes are pruned
        Pruning(x_new)
        # Computing the stopping criterion
        stop = StopCriterion(x_soln)
        k+=1
    # Rewiring the best nodes
    Rewiring(search_radius)
    # Computing the final path
    best_node = max(node.totalinfo for node in x_soln)
    best_path = ExtractPath(best_node)
    return best_path
```



(a) End of iteration 2, latest created node in blue and the best path in blue

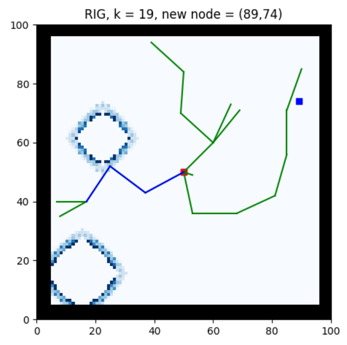


(b) Sample function executed, x_rand=(61,10) (blue)

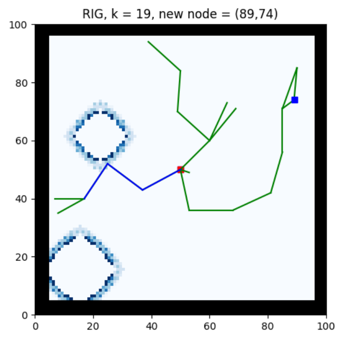


(c) Steer function executed, x_rand moved towards the nearest node (red)

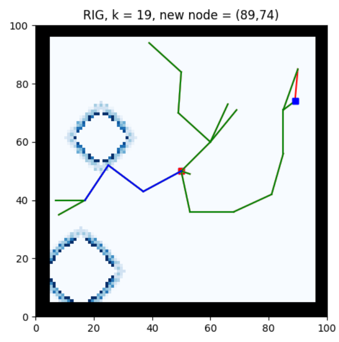
Figure 3: RIG: Sample and Steer, overlaid on entropy map



(a) New location is sampled, x_new at (89,74) (line 10)



(b) New nodes are created with near nodes (x_near) as parents (line 12)



(c) Unpromising nodes are pruned (red paths) (line 28)

Figure 4: RIG: creating new nodes and Pruning (with Listing 1 line referrals), overlaid on entropy map

5.2 Goal Location

In the RIG-graph algorithm as introduced by Hollinger and Sukhatme (2014), no goal location is specified. In this case, it simply starts from the specified starting location and then finds the path with the highest information value within a budget. In this project, it is aimed to start and end at the location of a charging station. Therefore, a required goal location is implemented.

Having an end location slightly changes the algorithm regarding the cost and information. Not only must the cumulative costs up to the last node of the path be within the budget, but the sum of these costs and the added costs from the last node to the goal location should be within the budget. Furthermore, the information that is gathered while returning to the goal location should be taken into account during optimization. For these two reasons, three extra variables where added to the node class: the final costs, the final information value and the final information path. Respectively, these represent the cost it takes to go from the node to the goal location, the information gathered on that path, and the locations that are covered by that path.

This adaptation allows to optimize over the summation of the information gathered along the entire path, including the final part to the goal location. Meanwhile, the requirement that the cost function is additive is still retained, as the cost to the goal location is independent from the cost up to the node.

5.3 Information Metric

The gathered information is calculated by summing the entropy values of all grid cells that are crossed by the path. The information is thus gathered continuously along the path, instead of only at the node locations. However, the information metric is submodular, meaning the information metric is not simply additive but dependent on the prior trajectory. More specifically, the entropy of a grid point is only added to the information metric of the trajectory if the grid point was not already earlier observed during the trajectory. As such, the best trajectory will not linger within high entropy areas and instead optimize the observed information over the whole trajectory. Crossing the same area a second time will thus not result in more information gain.

Additionally, a radius for monitoring can be specified. This radius indicates the distance around the robot that is observed in addition to the grid cell it is crossing. This radius can thus be any integer. In this project, the observation area is simply specified with a radius, thus creating a circular, 360 degrees footprint. This could, however, be adapted according to user preference to fit other observation footprints, such as a limited view.

5.4 Rewiring

Application of Rewiring One attribute of RRT* that was not implemented originally in RIG is the rewiring feature. Rewiring entails re-ordering of trajectories within a certain search radius. Rewiring allows the new node to become a parent of an existing node, thus actively rebuilding an existing branch of the tree-graph instead of extending a current branch as is done when creating new nodes. The original goal of rewiring, when applied in RRT*, is to further optimize the path by making existing trajectories even shorter. This is applied at the end of each iteration throughout the algorithm. In this project, also the information value is taken into account, which is not present in RRT*. Therefore, here the aim of rewiring is improving the amount of gathered information while remaining within the budget. Throughout the thesis, the rewiring function was redesigned multiple times, each based on the findings that resulted from earlier design steps.

In literature, the original rewiring function of RRT* is described (Noreen et al., 2016), which will be referred to in this report as path-based rewiring. Location-based rewiring is a variant to this approach, designed for this thesis. These were explored to start with, after which the final rewiring function was created, based upon the findings.

Classical applications of rewiring showed complications when applied to the problem at hand. The complete design process of earlier rewiring versions can be found in Appendix C, which describes the process of applying several forms of path-based and location-based rewiring and the conclusions derived from the process. Each of these versions are described with the aim to both present the findings and provide argumentation for following design steps leading to the final rewiring function.

The conclusions of these versions will shortly be summarized first, after which rewiring in hindsight is introduced, which is a unique solution designed for the problem at hand. It solves the relevant challenges of applying rewiring and allows for optimization through rewiring with the guarantee of improvement. The challenge when looking at rewiring is that not the minimization of costs but the maximization of gathered information is the objective of rewiring. This proves to be challenging, since the information metric is submodular, as described in Section 5.3. In the case of rewiring, this has the consequence that the entire path of a node should be considered, including the part back to the docking station. More problematic is the consequence that optimizing a path through rewiring is not guaranteed to be beneficial for children (the further extensions of the path). This contrasts with classical rewiring in RRT*, where shortening a path up to a certain node also automatically guarantees an improvement for its children. While it may be possible to account for currently existing children when rewiring, it cannot be predicted which children will be added in future iterations due to random sampling. The previously designed versions of rewiring were unable to overcome these challenges.

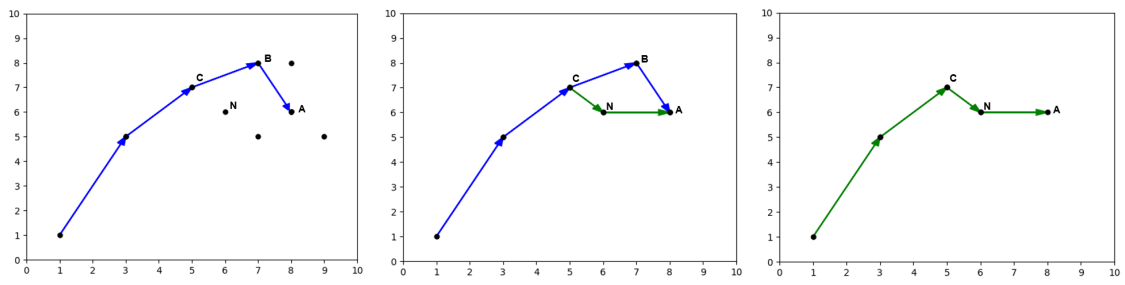
Listing 2: Hindsight Rewiring Pseudo-Code

```
def Rewiring(finalnode, nearnodes, search_radius):
    # moving back along the path until the beginning is reached
    while node.parent:
        distance = Dist(node,node.parent)
        infosteps.append((node.info-node.parent.info)/distance)
        node=node.parent
    sort(infosteps) # from high to low
    for index,infostep in infosteps:
        node = nodes[index]
        for nearnode in nearnodes: # within search_radius
            # path1: from node.parent.parent to nearnode
            # path2: from nearnode to node
            newinfo = node.parent.parent.info + info_path1 +
                                info_path2
            newcost = node.parent.parent.cost + cost_path1 +
                                cost_path2
            [newtotalcost,newtotalinfo] = Recalculate(node)
            if (newtotalcost<=budget) and (newtotalinfo>=finalnode.
                totalinfo):
                node.parent = nearnode
                node.info = newinfo
                node.cost = newcost
                nearnode.parent = node.parent.parent
```

Hindsight Rewiring With the described findings in mind, a new strategy was developed. Rewiring does not guarantee improvement for nodes that are yet to be sampled, but that is out of the question when there are no more nodes to be sampled. Hence, the idea of rewiring "in hindsight" was developed. Instead of rewiring after sampling a new node during each iteration, the rewiring is only executing after all iterations are finished, i.e. no more locations are sampled and all remaining nodes are feasible solutions. The strategy referred to as location-based rewiring is applied (see Appendix C); one node in the path is relocated. The criteria for rewiring then slightly change: rewiring occurs whenever the information value of the total path increases and the total cost of the path remains within budget. A decrease in cost is no longer required, since the function is dealing with the final paths of the algorithm, thus decreasing their length is no longer valuable. It should be noted that this is based on the chosen scenario for the thesis, where the sole goal is to maximize information within a given budget, and not to further shorten the path. The described rewiring method guarantees that no negative effects follow from the function: either no rewiring occurs and the information value remains the same, or rewiring occurs leading to an increased total information value of the path.

Also the rewiring procedure is then slightly different: instead of rewiring towards a newly sampled node, as is done in the earlier described strategies, the rewiring can now be done towards any node nearby. The rewiring of a path happens iteratively: the pieces of the path are sorted based on their information gain, after which the nodes in the path are rewired from highest to lowest gain. The information gain of a piece of path is described by the information density: the amount of gained information divided by the length of the path piece (Listing 2, line 5). This iterative method is used as the order of rewiring has an influence on the final result due to the limited budget and rewiring of the parts with the highest information gain is likely to have the biggest influence, as a small change of path in these high informative areas may already make a big difference in the total information value. The rewiring function steps for hindsight rewiring are outlined in Listing 2. Further details on the hindsight rewiring function are described in Appendix C.3.

The rewiring steps are illustrated in Figure 5. Here, the current node to be rewired is node A (indicated as "node" in Listing 2). Figure 5a shows the current path leading to node A and several near nodes within its reach. Node A is then rewired to near node N, as illustrated by 5b. The rewired path to node A is illustrated in 5c. Note that the path returning to the docking station and the information map are not illustrated, for the sake of simplicity. Node A could be the end of the path (indicated as "finalnode" in Listing 2), but it could also be somewhere along the path, where the path would then extend toward further children after node A. However, for the rewiring to be executed, always the total information of the entire path is considered, from start to end, thus taking the submodular information in account and guaranteeing an improvement in the total path.



(a) Before rewiring (b) Rewiring node A to new parent N (c) Path after rewiring

Figure 5: Rewiring steps for path-based rewiring (blue is original path, green is rewired path)

5.5 Pruning

Pruning entails the evaluation of all nodes at a single location and the deletion of the nodes which are less promising than other nodes. Pruning allows to move to a promising path quicker, as unpromising paths are taken out and therefore not extended further. This operation saves time and memory in the next iterations. RIG already contained the pruning function with the condition in Listing 3.

Listing 3: Pruning Condition 1

```
if (node2.cost<=node1.cost and node2.info>node1.info):
    prune(node1)
```

Thus, a node was only pruned if it had higher or equation costs and a strictly lower information value than other nodes at the same location. This condition ensures that there is no possibility of removing paths that could have been, or become, promising. An additional condition was added in this project, see Listing 4.

Listing 4: Pruning Condition 2

```
if (node2.cost<node1.cost and node2.info==node1.info):
    prune(node1)
```

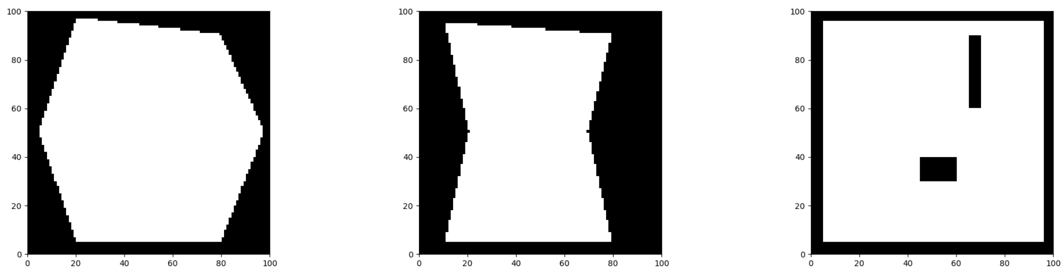
The condition is similar to the first condition, but in this case also nodes with equal information but strictly higher costs are pruned. This condition still ensures that no promising nodes are taken out, as the information value is leading for pruning. If either condition is true, the node is pruned. This condition was included to encourage pruning of non-promising nodes in the case of sparse information. In such a case, it could occur that many nodes do not encounter information in the beginning in the path, thus having equal (zero) information values. In this case, it is desirable to prune the nodes at the same location that have a higher cost, as these nodes create unnecessary long paths in areas without information, leaving less budget for the high information areas.

5.6 Field Shapes and Obstacles

An aim of the project was for the algorithm to take obstacles into account and to be able to work on both convex and non-convex fields. This was further generalized in implementation, as the field shape can be specified in the form of any kind of polygon, both convex and non-convex, as long as the polygon does not cross itself. The free space in which nodes can be sampled is given by a rectangle of fixed size, that at least encloses the polygon. Then the free space to be used by the planner is adapted based on the area that falls inside the polygon.

Non-convex fields and obstacles are dealt with within the planner in the same way it deals with field borders in general. When a new node is sampled, near nodes are determined. A path from the new node to a near node can only exist if no collision occurs. This collision includes that every point on the path falls within the polygon and does not collide with an obstacle.

A field of any polygon shape with obstacles of any form can be created in the form of a matrix, as long as it remains within the maximum size of 100x100m. Both the field borders and obstacles are defined by placing a NaN-value in the corresponding grid cell, after which the planner interprets paths through these grid cells as collisions. Several examples of created fields are given in Figure 6, where the black areas represent the NaN-values of the matrix.



(a) Convex polygon field (b) Concave polygon field (c) Rectangular polygon field with obstacles

Figure 6: Examples of fields in different polygonal shapes

5.7 Discrete Sampling

The entropy map utilized by the planner is grid-based. This does not automatically require the planner to work in a grid-based fashion. However, since the gathered information is determined based on the grid points crossed by the path, it is reasonable to have grid-based sampling as well. Grid-based sampling also reduces memory usage, as it prevents the emergence of nodes being sampled very close to each other, having almost equal cost and information values.

To allow for grid-based sampling, an extra step is incorporated in the Steer function (Listing 1). First, a location is sampled randomly, as is usual in the RIG algorithm. Next, this location is discretized such that it matches the midpoint of a grid point. When the discretized location has already been sampled before, a new location is sampled.

5.8 Stopping Criteria

The usual stopping criterion for RRT* and RIG is a maximum number of iterations (where each iteration represents a new sampled location), a maximum processing time or the availability of any solution. However, there are a number of factors influencing the rate in which the trajectory improves across iterations, including the step length, the complexity of the map and the distribution of entropy on the map. As it is difficult to estimate what the optimal number of iterations or processing time is because of these factors, one might simply choose to overestimate the maximum number of iterations required to reach a satisfactory solution. The ultimate example of overestimation is to set the maximum amount of iterations equal to the total number of grid locations that can be sampled. However, to retain a balance between runtime and quality of results, it would be desirable to stop iterating whenever the current solution is close enough to the optimal solution. For the sake of creating this balance without requiring an intuitive idea of the best number of iterations or running the risk of underestimating the number of required iterations, an extra criterion was implemented.

That criterion is based on the rate of improvement of the average information value of the top nodes. The top nodes indicate a predefined amount of nodes with the best information values. It is chosen to consider this set of nodes since this gives a more trustworthy insight in the convergence compared to only the single best node of the moment. This trustworthiness is due to the reason that averaging over a number of nodes reduces the effect of random sampling on the convergence. In other words, the stopping criterion is then less affected by the individual effect of sampled nodes. The information value of the best node often does not increase for many iterations, not only due to the fact that a new node might not be profitable, but for instance because the node has nearly reached the budget or cannot connect to the new location. Hence, the information value of the best node gives an incomplete image of the optimization status.

The number of nodes to consider can be set by the user. For the rest of the report, the number of nodes will be set to 20. The rate of improvement is obtained by taking the discrete derivative of the average information value of the top 20 of nodes over the course of a number of iterations. The derivative is then computed through division of the increase in information value by the number of iterations:

$$r = \frac{avg[t-1] - avg[t-(1+N)]}{N}, \tag{2}$$

where $avg[i] = \frac{\sum_{n=1}^{20} I_n}{20}$ for each iteration $t-i$ with t being the current iteration, I_n is the information value of top node nr n and N is the number of iterations over which the derivative is taken.

Also the number of iterations to consider can be set by the user to balance the strictness of the criterion. The derivative converges to 0 over the iterations when the average information value of the top nodes converges to a stable value. This indicates the convergence towards the (local) optimum, as optimization principles state that the zero-point of a derivative indicates a stable point. The average information value of the top 20 nodes is a non-decreasing function. It can thus be either increasing or remain stable.

The stopping criterion is reached whenever the derivative of the average information value of the top 20 nodes reaches zero. Additionally, to make sure this does not happen too early, i.e. when stability is reached early on in iterations before actual convergence, a minimum number of iterations can be set or the stopping criterion can be set stricter by increasing the number of iterations over which the derivative is computed. Two variations of the stopping criterion are tested in the simulations, which vary in the number of iterations over which the derivative is computed. The results and implications will be further discussed in Chapters 8.2.5 and 9.

5.9 Kinematic Constraints

One aim of the planner is to be versatile and easily applicable to different robot types. To this end, it has to be able to incorporate kinematic constraints of mobile robots. In agriculture, different types of robots are applied which come with their own constraints regarding kinematics. To ensure that the planned path is feasible and the calculated information gain is accurate, it is a necessity that the kinematic constraints are taken into account during exploration by the planner. If a path were to be adapted to the kinematic constraints only afterwards, i.e. through trajectory smoothening, the planner may yield kinematically infeasible paths or inaccurate information gain calculations. The latter occurs since monitoring takes place continuously along the path, hence smoothening the path changes the monitored area. With regard to kinematics, mobile robots can be divided into two categories: holonomic and non-holonomic. This division is used as a basis for the incorporation of kinematic constraints in the algorithm.

Robots with only holonomic constraints and robots that can rotate along their vertical axis, such as the Origin, can move in any direction. In this case, there are no limitations to the allowed range of angles between line segments. However, it might take time and energy to turn. Therefore, there is the option to have costs are added to the path based on the angle and the angular cost. This angular cost is a user-specified cost metric for turning 1 rad.

Robots with non-holonomic constraints cannot freely move in any direction. Their degrees of freedom are limited. In this case, there are limitations to the allowed range of angles between line segments. Two methods were identified to implement this limitation:

1. A connection between nodes simply cannot be created if the difference in direction between the previous segment and the new segment is too large
2. The limit in angle range is incorporated in the calculated path by creating a smooth trajectory

The first method is easy to implement, but severely limits the exploration. It can be implemented by applying infinite costs to paths with too sharp angles. An angle is too sharp whenever the robot is unable to make the angle within the grid cell, e.g. when it can only turn 45 degrees during a segment of 0.5 meter. However, applying this method means that many nodes may not be able to connect directly, even when this would be profitable for information gain. An example is given in Figure 7, where node B can form a connection with the newly sampled node C, while node A cannot create such a connection with node C due to the sharper angle. In this example, it can be imagined that with a smoothened trajectory, node A could connect to node C.

The second method is more complex, as the change of path also impacts the information gain and path length. Since smoothening the trajectory slightly changes the path and area covered, a path from one location to the next no longer has a singular solution with fixed cost and information value, but is dependent on the angle configuration at the beginning and end of the curve. An example is given in Figure 8, where the straight trajectory (black) from location A to B clearly has different costs and covered area from a curved trajectory (green) and another curved trajectory (blue) again has other cost and information values due to the angle configurations. The dependence of the path on the angle configurations leads to an infinite amount of possible paths from one location to another. Therefore, this solution was simplified through discretizing the angle, such that the angle configuration at the start and end of the curve is limited in the number of options. The number of options, characterizing the interval between the possible angles, is configurable. For the remainder of the report, eight possible configurations are considered: $-\pi, -3\pi/4, -\pi/2, -\pi/4, 0, \pi/4, \pi/2, 3\pi/4$. The configuration is chosen by rounding the angle to the closest option. Next, different options were implemented for the creation of a curve trajectory. These options differ in the restrictions and preferences of the driving direction. When the driving direction is forward-only, Dubins curves are used. When the driving direction can be reversed, Reeds-Shepp curves are used. If there is no preference in driving direction, meaning the robot may complete large parts of the path in reverse, the Reeds-Shepp curves are combined with the possibility to change direction for longer segments. These three options are also visualized in Figure 9, where the asterix shapes are located at the node positions and visualize the possible discrete angle configurations.

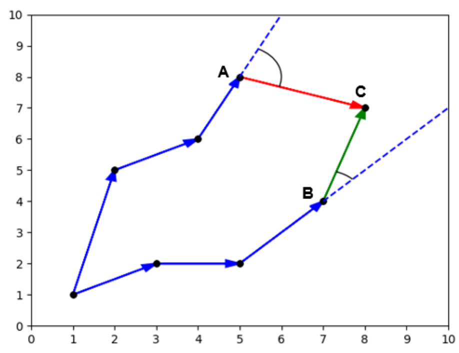


Figure 7: Node A cannot form a connection with the newly sampled node C due to the limit-exceeding angle between the line segments. Node B forms a connection with node C as it has a smaller angle between the line segments.

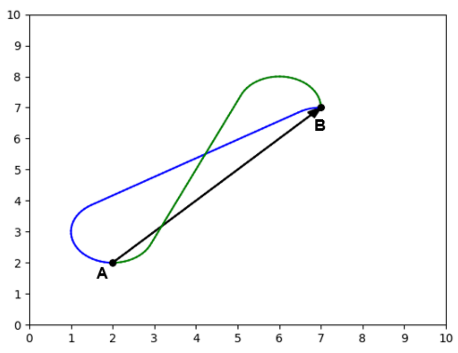
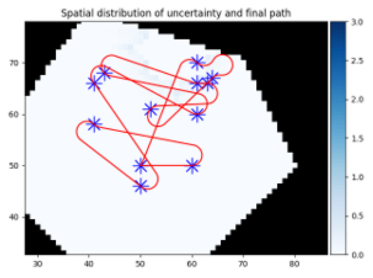


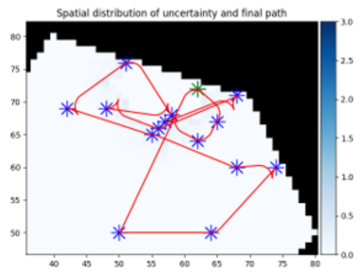
Figure 8: Straight trajectory from A to B (black) and two examples of a curved trajectory (blue and green) with different angle configurations.

Dubins curves guarantee the shortest smooth trajectory within the given constraints for forward-only vehicles. It requires a start and end position, start and end angle, and a maximum curvature to be defined. The start and end positions are simply the node positions. The end angle is fixed to be the direction of the vector between the two nodes, which is then rounded off to one of the discretized angles, as explained before. The starting angle is defined similarly, but then taking the vector direction from the start's node parent node to the start node itself. The maximum curvature is equal to the inverse of the turning radius. With these features given, the trajectory formed by consecutive Dubins curves is kinematically feasible, for the given maximum curvature, and smooth, as the end angle of one curve is equal to the starting angle of the next. Figure 10 visualizes a Dubins curve and a Reeds-Shepp curve for equal start and end locations and angles, with a maximum curvature of $1/r$.

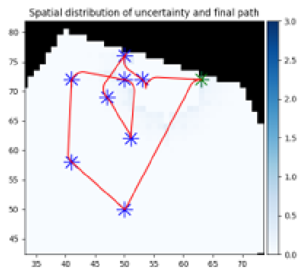
Reeds-Shepp curves guaranteed give the shortest smooth trajectory within the given constraints for reversible vehicles. Reeds-Shepp curves work very similarly to Dubins curves: it generates a kinematically feasible curve given start and end positions, start and end angles, and a maximum curvature. The method for determining the positions and angles remains equal to the Dubins-method. The benefit of Reeds-Shepp curves is that it can avoid big turns by reversing direction in order to make a turn. This yields more possibilities to create a kinematically feasible curve. Often, it is preferred still to drive forward most of the time, even when these reversals are possible, e.g. due to sensor placement. However, in case this has no influence, it could be beneficial to drive in reverse for an entire curve from one location to another. In this case, the start angle is reversed (turned 180 degrees), after which a Reeds-Shepp curve



(a) Dubins curves



(b) Reeds-Shepp curves



(c) Reeds-Shepp curves with reverse driving

Figure 9: Application of different curves between node positions

can be formed with this renewed start angle.

One disadvantage for all smooth trajectory options, even with discretization of the angles, is the increase in possible paths from one location to another. There are eight times as many paths possible with the choice of eight dicretization options, since a path is defined by the positions and the start angle. Note that the end angle does not lead to an increase in possible paths, as it is fixed to the line direction. This leads to increased memory usage when saving the paths and increased run-time when calculating the paths. To reduce this disadvantage, the paths are saved in memory based on their relative positions: the saved paths are identified by the difference in positions between the nodes and the start angle. In other words, the start of the curve is translated to the origin. As such, curves can be reused more often. This reduction of memory usage applies due to the choice to sample node locations within a grid, thus creating a discretized set of locations. Whenever one would have opted for random sampling without discretization, this method would have little effect, since the relative paths would seldom be equal to each other. When the curves are being used again, the translation is reversed to the actual positions of the nodes.

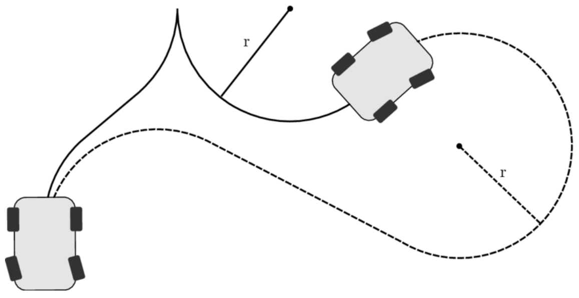


Figure 10: Dubins (dashed) and Reeds-Shepp curves (solid) (Kurzer, 2016)

For the implementation of kinematic constraints in the fields with rows, not only the cost and information values should be considered, but also the feasibility of the path and the options for the next segment of a path. Whenever non-holonomic robots are considered, with no option of driving in reverse, the options for the path are more limited, as the robot can only choose to drive through a complete row, or skip it completely, since turning back halfway is not a possibility. This restriction in movement is incorporated as well. The Dubins and Reeds-Shepp curves are then used for determining the path when changing between different rows, while the kinematic configuration determines whether turning back within rows is feasible.

In conclusion, there are six options for the kinematic configuration of the algorithm: no constraints, holonomic constraints with added costs, non-holonomic constraints with restricted angle or non-holonomic constraints with smooth trajectories, where there is a choice between Dubins curves (forward-only), Reeds-Shepp curves (reversible), or Reeds-Shepp curves with no preferred driving direction (reversible for longer time). The options are available both in the scenario with and without rows in the field. The configuration can easily be switched according to preference. The kinematic constraints are taken into account while planning the most informative path, such that the planned path matches the final path driven by the robot, and thus also the calculated information gain is based on the kinematically feasible path. Notoriously, the state space complexity is only marginally increased by the incorporation of the kinematic constraints.

5.10 Structured Fields

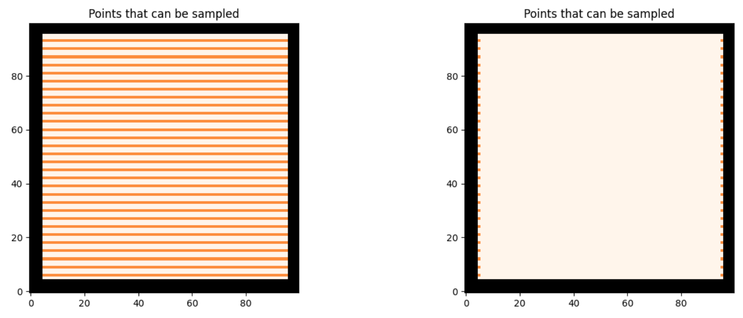
Applying RIG to structured fields, which contain rows of crops, requires a couple of adjustments. Two key processes that require adjustments are the calculation of distances and the sampling of new nodes.

Distance calculation In the case of unstructured fields the distance between locations can simply be determined using the formula for Euclidean distance. In the case of structured fields, the rows and their edges have to be considered to find the distance between nodes. For this purpose, the A* algorithm was used, which is a graph search algorithm using a heuristic based on Manhattan distance. This algorithm guaranteed finds the optimal path from one location to another and is applicable to the grid-based field map that is employed in this system. The A* approach was preferred over a task-specific function which compared the difference between the distance of taking the left or right edge of rows. This preference was due to the fact that such a task-specific approach is more dependent on assumptions about the field and therefore less robust and less generalizable to different contexts.

Sampling in rows In fields with rows, new locations should only be sampled within one of these rows. This is because it is intended for the scenario where the robot only moves in straight lines through the rows, as the Origin One robot does. Additionally, depending on the kinematic constraints, some robots may not be able to reverse within a row, thus having to cross the entire row. Therefore, the Sample function, as indicated in Listing 1, was adapted. One intuitive approach is close to the original method for sampling: sampling a random (discrete) location in the field, then determining a location in a row that lies in the direction of that random location from the nearest node. However, this means that the sampled location very likely has to be relocated when there is distance between rows, even if it is within the step length of the nearest node, thus increasing the computational load. Additionally, in the case of non-holonomic robots, it may be very inefficient to sample many points inside rows instead of at the edges, since these robots only have the option to cross entire rows anyway. An alternative was applied where the sampling in the free space happens in a more restrictive way: first, based on the kinematic constraints, all grid points that are feasible to reach are determined at the start of the algorithm, after which only these grid points can be sampled. Since these points only have to be calculated a single time, computational load is greatly reduced during sampling. Two examples of placement of all feasible grid points for sampling within a field are displayed in Figure 11, where the orange points indicate the feasible points.

The approach to find a location on this path from the nearest node to the sampled location within the step length distance was also altered to be able to apply it to structured fields, and has had several iterations before reaching the approach described below. Earlier approaches are described in Appendix D. This step is referred to as the Steer function in Listing 1. An approach was developed that coincided with the introduction of using an A* algorithm to compute the distance between points. Since the A* algorithm yields the steps (in terms of grid cells) to move from one location to another, these locations and corresponding distances could be used to compute the furthest location on a path within a step length.

Figure 12 shows the workings of the A* algorithm and taken steps in the sampling procedure. The green square indicates the nearest node, thus the node from which the sampled node should be within a maximum distance. The red square indicates the random sampled location, which is too distant from the nearest node. The blue squares indicate the steps that are obtained from the A* algorithm, which indicate the points where the algorithm turns into a different direction, where the arrows indicate the entire route from the nearest node to the random sampled node. The distances are calculated at each of these blue points, in a backwards fashion. In this case, it was found the farthest (the upper) blue step was within reach, so then the line from this point to the end (red) square is followed until the maximum distance is reached. This yields the new sampled node, the yellow circle.



(a) Possible sampling points without kinematic constraints

(b) Possible sampling points with constraints (without reversing)

Figure 11: Possible sampling points (= orange) for structured fields

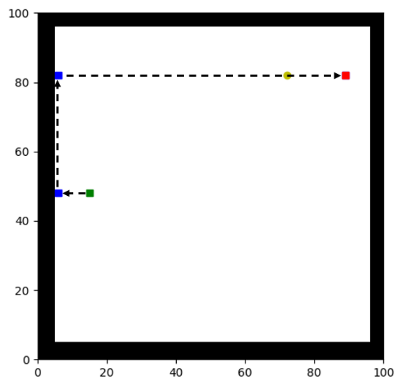


Figure 12: Steer function using the A* algorithm path

5.11 Horizon Planning

The final goal of the planner is to be applied to a field over the course of multiple days, most likely to be applied permanently on a daily basis. Up to this point, the strategy was chosen to compute a path which gains as much information as possible purely based on the current entropy distribution. This greedy strategy is, however, somewhat naive and might not be most effective in the longer term for monitoring a field. To understand the reason for this, consider Figure 13 which portrays an orienteering problem where one aims to maximize the number of visited orange points from the blue starting point within an arbitrary amount of time. Based on the initial field and point locations (Figure 13a), it might seem like a good choice to execute the route as displayed in Figure 13b, which leaves the points displayed in Figure 13c. The remaining points are now very distributed across the field, which would leave a long path if these would be gathered in a next day. However, would it have been known beforehand that the remaining points after the first route would be gathered during a second route, one might opt for a route such as the one in Figure 13d; slightly less effective by itself, but more effective when considering a two-day horizon.

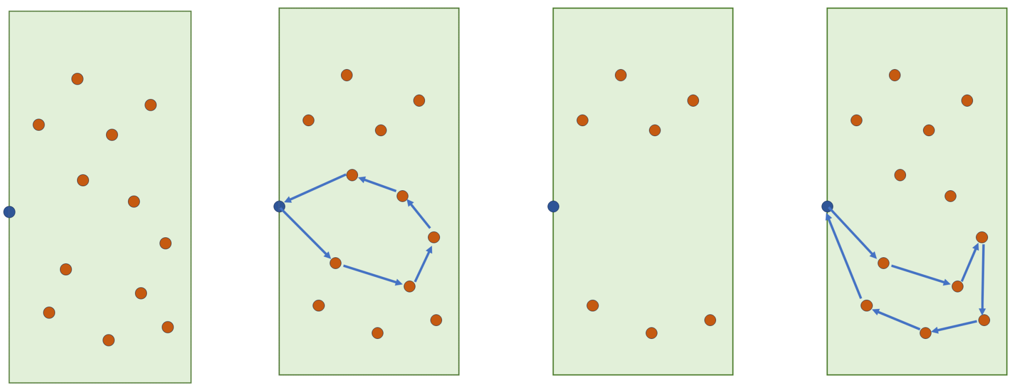

(a) Field at day 1 (b) First route (c) Field at day 2 (d) Smarter first route

Figure 13: Two-day orienteering problem

An adaption based on this principle is applied to the planner. This adaption allows the planner to plan over a horizon. The length of this horizon is configurable, but for the rest of the report a two-day horizon is considered. To apply this principle, the planner basically plans a two-day path: it plans a route consisting of two paths that each individually fall within the budget. Eventually, the best routes are split into the two paths, after which the best individual path of these is executed. In the basis, a route is thus formed that mimics the scenario where the robot would execute two paths within a single day, where it recharges in between these paths. Note that this means that both parts of the double route are based on the same entropy map.

The implementation of this strategy to plan over a horizon requires a few adaptations to the planning algorithm. A node characteristic is added to each node, which describes what part of the route it belongs to, being either the first or second path. Every node that is created that belongs to the first path automatically leads to the creation of a new node that is located at the starting point (i.e. the charging station) and resembles the start of the second path. The creation of a node in the first path thus automatically leads to the ability to continue from there on the second path. From a node within the first path, it can either continue exploring, extending the first path until the budget is reached, or it can return to the starting point to start the second path. The second path explores the field equally to how planning without a horizon happens: it maximizes the gained information within the budget. The information value of the entire route consists of the information gained by the two paths together. Areas that are crossed by both the first and second path thus only add information a single time. Using this principle, paths are expected to show more resemblance to Figure 13d, as the two paths of a route complement each other.

A possible disadvantage may occur when the entropy of the map is low and could be largely covered within a single path. In this case, it might be more effective not to look at the complementary information values of two paths, but solely prioritize a single path, since this may lead to a more informative path.

To understand this, it should be considered that with the horizon strategy, optimization is focused on routes consisting of two paths where a single path is executed in the end. However, for the majority of cases, the entropy is expected to be higher and the distribution of information is such that it is unlikely to be covered within a single path. Beneficial effects of planning over a horizon are then expected.

In theory, the horizon could be extended from a double path to any number. However, in practice this would be unlikely to be very useful, both due to the effect described above, which becomes even stronger for longer horizons, and due to the additional disadvantage of rapid increasing computational load due to the extended paths and thus high amount of extra nodes created.

To analyze the effect and efficiency of planning over a horizon, this strategy will be tested and compared to planning a single path. These tests are based on multi-day simulations, which allow to show the long-term effect of this strategy. The outcomes and conclusions are described in Chapters 8.2.6 and 9.

5.12 Specified Variables

There is a range of variables within the algorithm of which the value can be altered, which have an impact on the performance. The most important of these variables are discussed and are tested for impact during simulations (see Chapter 8). These variables are as follows and will be further explained in this section:

- Step length
- Search radius
- Number of iterations

A connection between nodes will only be made when the distance is within a certain limit. This limit is the step length, prescribing the maximum allowed distance between existing and new nodes. The step length influences the sampling, as a node will be sampled closer whenever no node is within the step length of the original new sampled node. Furthermore, the step length influences the rate of exploration, especially in the early iterations, as it limits the extension of the reach of existing nodes. Thus, a smaller step length may limit optimisation of the path. The step length also influences the amount of nodes that accumulate during each iteration, as more nodes will be created when the step length is larger, since more existing nodes will then be within the range of the new node (remember that each node represents a path from start to the current location, not just a location). A quicker growth of nodes means more memory usage and a longer run-time (due to functions having to iterate over more nodes). The impact of the step length on these described factors may be a cause for a decrease in performance and may also vary across different kinds of contexts (e.g. with/without rows and different types and sizes of fields).

In the rewiring step, the nodes within a certain range of the new node are considered for rewiring. This range is represented by the search radius. The search radius can simply be equal to the step length, or can be chosen to be larger or smaller. The choice for the search radius has an impact on the memory usage, run-time and performance of the algorithm.

A minimum and maximum amount of iterations can be prescribed. Of course, a limit on the number of iterations impacts the run-time of the algorithm. The limit also impacts performance. Preferably, the number of iterations would be such that the algorithm runs until the solution has converged to a certain path and little improvement occurs. To make this preference more likely to be reached, the stopping criterion has been introduced. Therefore, it is advisable to make the minimum and maximum amount of iterations broad. The maximum number of iterations can be used to deal with limits in run-time or memory availability.

6 Monitoring Module

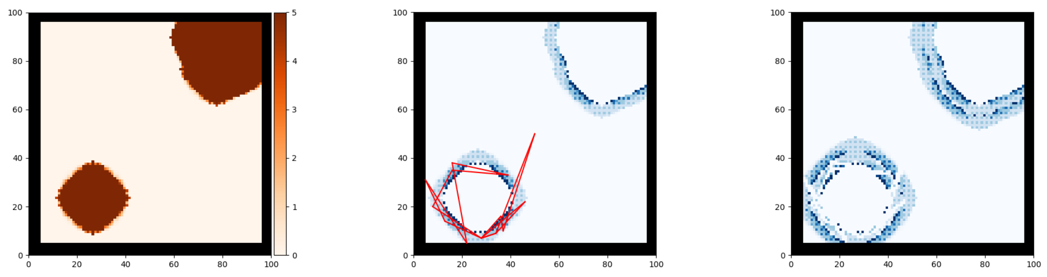
The path generated by the planner is executed, which leads the robot to monitor the field. This leads to new observed information about the growth of weeds and pathogens at each grid point, which reduces the entropy of the world model. The monitored data is imperfect, as there is always some uncertainty in measurements due to imperfect plant and weed recognition and distinction, imperfect pathogen detection, and limited camera resolution. This uncertainty is taken into account by specifying the sensor uncertainty, enabling modelling of imperfect information in simulations. The uncertainty in monitoring influences the updated world model and the entropy, but not the functioning of the planner. This is because the information remains submodular, even with uncertainty. The monitoring uncertainty originates from errors in object detection. Errors in object detection within the field of agriculture are common. Due to factors such as overlapping leaves, similar features of weeds and crops, unpredictable and non-uniform natural light conditions, and variations in databases, detecting algorithms are prone to errors leading to misclassifications of weeds, healthy crops and diseased crops, miscounting weeds and crops or falsely detecting weeds and crops (Jeon et al., 2011, Mavridou et al., 2019). These errors have an impact on the entropy of the world model. However, there is no guarantee that monitoring one area multiple times within one path will lead to a decrease in monitoring uncertainty, hence the information remains submodular, i.e. the entropy value is only added to the path information metric when encountered the first time. In the case when errors were likely to average out to the correct value, it would be worth changing the metric such that each new encounter with the same area would lead to (a partial) addition to the information metric.

The path following is outside of the scope of this project, since the software to accomplish such a task is already developed by Avular for the Origin One robot. Additionally, the motion trajectory is specific to each type of driving platform. The generated path is therefore designed to be kinematically feasible for the Origin One robot and other general driving platforms and it is then assumed that this path is perfectly followed.

7 Updating the World Model

This task is not a module in itself, but it closes the cycle, as indicated by arrow 3 in Figure 2. In this step, the observed information is given back as input to the model. Supplying this updated information, together with the spreading model and the current world model, allows to generate an updated world model. The spreading model characterizes the (simulated) ground truth of spread, whereas the world model characterizes the predicted spread from the knowledge of the robot, based on monitored data. This world model can then again be used to generate the entropy map which is then supplied to the planner to compute the path for the next cycle. The spreading model, world model and entropy map all consist of grid maps. Three of the described steps are visualized in Figure 14, which depicts the world model, the entropy map supplied to the planner and the computed path of the planner, and the updated entropy map based on the monitored data and the new spreading prediction. The new entropy is based on the new world model (i.e. uncertainty of the spread of the pathogens and weeds), but additionally a daily uncertainty can be configured for the update of the entropy map. This daily uncertainty represents the uncertainty about the state of an cell which grows whenever it has not been visited for a longer time. The entropy value of each non-visited cell is increased with the daily uncertainty value. Especially when new infections are expected, this is useful to be included as it prevents the robot from blindly depending on the current spread model and missing new pathogen or weed infestations.

The step of updating the world model follows naturally from the combination of the modules and it allows to close the cycle. The updated world model enables to establish the expected effects of the planned path on the state of the field. Additionally, with this full cycle, a long-term simulation can be run, modelling the results from applying the informed path planner consequently over a longer period of time. A long-term simulation more closely represents the needs of weed control in an agricultural field, as the end goal is to create as much crop yield as possible over the course of a growing season. This is a long-term goal, where periodic control interventions contribute to the final result. Furthermore, the step of closing the cycle will prove useful when applying this architecture of models in a real life scenario, where the monitored information can then be integrated into the world model after a robot has explored the field. As such, it can autonomously keep monitoring a field on a daily basis without the need of a user to update the supplied information.



(a) Spreading map of a pathogen (b) Planned path plotted over the entropy map (c) Updated entropy map

Figure 14: Steps of the architecture

8 Results

This chapter begins by describing the test cases of the simulation tests, tested variables, and measured outcomes. Subsequently, the results are discussed in Section 8.2.

8.1 Simulation Experiments

The experiments to test and evaluate the architecture are performed within a simulation environment. The different modules as well as the simulation environment are written in Python 3. The full code is available in GitHub (van Esch, 2023). An informed RRT* software implementation (Zhou, 2020) was modified to create a RIG implementation for this thesis due to the unavailability of the original RIG code. This section describes the tests for the different algorithm aspects that are evaluated in detail.

All aspects were tested in identical scenarios where a predetermined seed was used, serving as a pseudorandom number generator for the sampling locations, ensuring that the algorithm's typically random exploration is eliminated and experiments are repeatable. The seed allows for a deterministic scenario, including the same world model map, entropy map, pathogen characteristics, deterministic spread factor, field shape, number of iterations, order of sampled locations, and other characteristics that are configurable. There is one scenario with rows and one without rows. The results are generalizable to other types of maps and scenarios. To eliminate the impact of context variables and configurable characteristics, the decision was made to test exclusively within a specific scenario, thus focusing completely on the tested variable. This scenario was used to evaluate each of the tested aspects of the algorithm. The aspects were tested to assess their influence on the planning performance, which is defined as the ratio of the gained information over the total available information. Each tested aspect is briefly described.

Informedness refers to whether the planner is given an informative map of the entropy as opposed to having no information of entropy. To simulate uninformed planning, the planner is given a uniform entropy map. As such, it can be evaluated whether an informed planner benefits precision monitoring and whether having a spreading module providing entropy mapping is valuable.

There are four cases aimed at testing the effect of budget on performance. Performance is expected to improve with increased budget. The extent and linearity of the increase in performance is tested.

Rewiring is tested by comparing the results of the algorithm without rewiring and with rewiring. These results enable forming conclusions on the extent of the effect of rewiring.

Variations in step length and search radius are tested jointly, considering the expectation that both the absolute values and the values relative to each other influence performance. The step length for structured fields is chosen to be larger than for unstructured fields, as the row structure requires larger distances to be travelled.

The number of iterations and stopping criterion are closely intertwined, as a stricter stopping criterion likely leads to a lower number of completed iterations. A minimum number of iterations is expected to be required to achieve reasonable performance due to the sampling randomness and information sparsity in a field. Increasing in the number of iterations is expected to enhance performance, though it is expected that the improvement of the best path flattens off with increasing iterations. Therefore, a balance between the improvement of the path and the run-time should be determined. This balance can be explored through application of variations in the strictness of the stopping criterion. The mild criterion averages over 25 iterations, while the strict criterion averages over 50 iterations.

Horizon path planning is tested through a multi-day simulation, as the effect of this option is mainly detectable over the course of multiple cycles of the architecture (see Figure 2). A simulation entailing 12 cycles (days) is executed to analyze the effectiveness of horizon planning based on world model accuracy and the amount of entropy in the map. The performance is compared to the setting where no horizon planning is applied, thus each time only planning a single path.

When testing each of the aspects described above, the other aspects are kept static, to prevent cross-effects. Tables 3 and 4 show an overview of all the performed experiments and corresponding results of simulations without and with rows, respectively. There are 16 test cases in total, each applied both to the algorithm without and with rows. The tables show the configuration per test case, consisting of the underlined setting that indicates the tested aspect for each test case, and the base settings for the other aspects. For the results on the informedness and budget, additional simulation tests were conducted to provide more insights, which are described in the corresponding sections.

The performance is evaluated in terms of the ratio of information of the final path over the total amount of available information in the entropy map. In addition, the run-time is analyzed, as the run-time gives an indication of the applicability and scalability of the architecture. The run-time is the measured CPU time (for HP ZBook with Intel Core(TM) i7 Quad-Core processor) and is expressed in seconds. Due to the limited number of performed simulations, run-time is heavily influenced by the specific scenario and external factors (e.g. PC processor, internal temperature, etc.). This effect is evident in differences in run-times of test cases with identical variable settings (i.e. some of the tested aspects have the same variable settings for the base test), such as cases 4 and 7. For this reason, the run-time will only be used as a measure for relative comparisons between variable settings. In multi-cycle simulations, the world model inaccuracy is computed as follows:

$$\frac{sum(abs(M_w - M_s))}{sum(M_s)} \tag{3}$$

where M_w is the world model matrix of the robot, M_s is the real spreading model matrix after the last cycle, abs is an element-wise operation and sum indicates the grand sum of all entries. This serves as an indirect quality measurement of monitoring, since more effective modelling leads to a more accurate world model over time.

Table 3: Testing Cases with Aspects, Settings, Performance Results, and Run-Time (without rows)

#	Informed	Budget	Rewiring	Step Length	Search Radius	Stopping Criterion	Horizon	Max Iter	Performance	Run-time
1	Yes	350	Yes	40	40	No	No	500	0.32	2577
2	No	350	Yes	40	40	No	No	500	0.038	4853
3	Yes	200	Yes	40	40	No	No	500	0.23	1815
4	Yes	350	Yes	40	40	No	No	500	0.32	2346
5	Yes	500	Yes	40	40	No	No	500	0.40	3299
6	Yes	650	Yes	40	40	No	No	500	0.44	3304
7	Yes	350	Yes	40	40	No	No	500	0.32	2575
8	Yes	350	No	40	40	No	No	500	0.23	2178
9	Yes	350	Yes	20	20	No	No	500	0.37	2155
10	Yes	350	Yes	20	40	No	No	500	0.38	2367
11	Yes	350	Yes	40	20	No	No	500	0.33	2604
12	Yes	350	Yes	40	40	No	No	500	0.32	2831
13	Yes	350	Yes	40	40	Mild	No	500	0.32	1121
14	Yes	350	Yes	40	40	Strict	No	500	0.31	2202
15	Yes	350	Yes	40	40	No	Horizon	300	0.14	345194
16	Yes	350	Yes	40	40	No	Single Path	300	0.14	16700

Table 4: Testing Cases with Aspects, Settings, Performance Results, and Run-Time (with rows)

#	Informed	Budget	Rewiring	Step Length	Search Radius	Stopping Criterion	Horizon	Max Iter	Performance	Run-time
1	Yes	500	Yes	200	200	No	No	500	0.16	26351
2	No	500	Yes	200	200	No	No	500	0.058	50092
3	Yes	300	Yes	200	200	No	No	500	0.11	25381
4	Yes	500	Yes	200	200	No	No	500	0.16	26071
5	Yes	700	Yes	200	200	No	No	500	0.21	24338
6	Yes	900	Yes	200	200	No	No	500	0.25	24480
7	Yes	500	Yes	200	200	No	No	500	0.16	26503
8	Yes	500	No	200	200	No	No	500	0.08	23445
9	Yes	500	Yes	100	100	No	No	500	0.16	18997
10	Yes	500	Yes	100	200	No	No	500	0.16	19182
11	Yes	500	Yes	200	100	No	No	500	0.16	25810
12	Yes	500	Yes	200	200	No	No	500	0.16	26947
13	Yes	500	Yes	200	200	Mild	No	500	0.12	976
14	Yes	500	Yes	200	200	Strict	No	500	0.19	4718
15	Yes	500	Yes	200	200	No	Horizon	300	0.055	43687
16	Yes	500	Yes	200	200	No	Single Path	300	0.054	47310

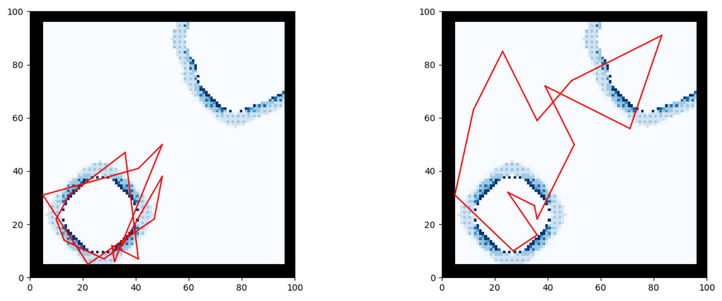
8.2 Findings

This section of the chapter discusses the performance outcomes of the test cases for the tested aspects. The results are summarized in Tables 3 and 4. Furthermore, further analysis is provided in the following paragraphs.

8.2.1 Informed Path Planner

The informed planner demonstrates performance levels that are approximately three times higher (rows) or even an order of magnitude higher (no rows) than the uninformed planner and run-times that are nearly halved. The longer run-times for uninformed planner are caused by the uniform entropy map, which leads to more nodes to iterate over, as less of them are pruned due to paths often being equally informative.

Figure 15 shows the final paths for case 1 and 2, respectively. Here, it shows how the path of the informed planner is centered more around one of the areas with high entropy, while the uninformed planner travels more randomly through the field. Figure 20 (Appendix E.1) visualizes the paths overlaid on the world model of a multi-cycle simulation, showing that the informed planner more quickly and accurately builds up the world model and focuses on the areas with relevant information. In this simulation, both the informed and uninformed planner started off without any prior information on the spreading map, thus having a uniform world model, which comes closer to the real spreading model with each cycle. After the last cycle, the inaccuracy of the world model for the informed planner is only 0.0014 compared to 0.012 for the uninformed planner. This additional simulation test demonstrates the use of the informed planner in a realistic scenario where the robot enters a field with a yet unknown state of pathogens and weeds.



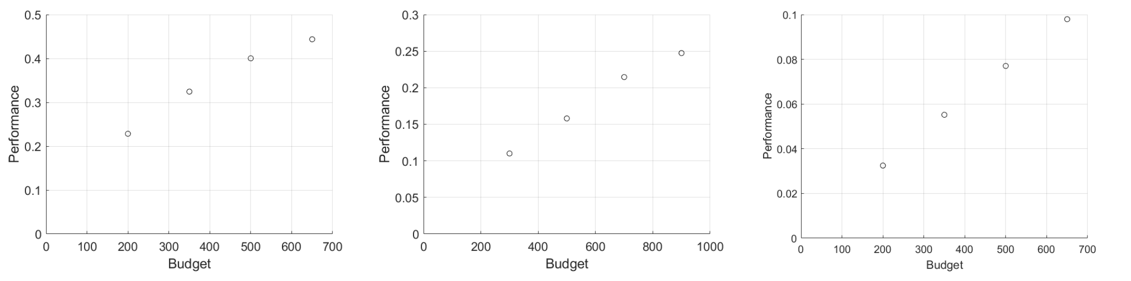
(a) Informed setting (b) Uninformed setting

Figure 15: Final paths overlaid on the entropy map

8.2.2 Budget

As can be deduced from the outcomes in Tables 3 and 4, both for the scenarios with and without rows, the performance increases with increasing budget. Figure 16 shows the performance plotted against the different tested budgets.

In the plot of the results without rows (Figure 16a), it is clear that the results form a sublinear function; the performance grows less than linearly with increasing budget. In the plot of the results with rows (Figure 16b), the sublinear relation applies as well, only with slightly smaller differences between the linear function and the data points. The non-linear relation between budget and performance is due to the nature of pathogens and weeds, leading to non-uniform spread. The non-uniform spread leads to a non-uniform entropy map, meaning a shorter budget focuses on the highest entropy areas, where increasing budget leads to increasing exploration of less informative areas, thus having a non-linear information gain. This is verified the results of tests on a uniform entropy map (see Figure 16c), which show a linear relation between varying budgets and performance.



(a) Results for simulations without rows (b) Results for simulations with rows (c) Results for simulations without rows and uniform entropy map

Figure 16: Performance over budget

8.2.3 Rewiring Function

Application of rewiring function leads to great increase in performance, varying from 39% (no rows) to 100% (with rows) increase. Since rewiring leads to application of an extra function in the algorithm, run-times have increased. However, the increases in run-time are not proportional to the increase in performance, with percentage increases of 18% (no rows) (Table 3) and 13% (with rows) (Table 4).

8.2.4 Step Length and Search Radius

Analyzing the tests performed on step length and search radius reveals that the differences in performance are minor. For the simulations without rows (Table 3), it can be seen that particularly the step length has an influence, where smaller step length gives better results. For the simulations with rows (Table 4), no differences in results can be seen based on performance (rounded to two significant numbers).

One thing that is noticeable is that both for the simulations with and without rows, performance results of test scenarios 9 and 10 are higher or equal to those of scenarios 11 and 12, while the run-times are significantly lower. The lower run-times can be brought back to the meaning of the step length: a higher step length requires looping over a higher number of nodes, thus leading to higher run-times. The number of selected nodes to iterate over during rewiring is dictated by the search radius, but since this is only executed a single time, there is no observable influence of the search radius on the run-time.

The hypothesis was that a larger step length would lead to more rapid exploration of the field and the corresponding information and thus converge more rapidly to higher performance values of paths. A possible explanation for the opposite observation regarding performance is that the number of nodes created depends on the selection of near nodes (see Line 12 in Listing 1). The selected number of near nodes depends on the step length, but also has a maximum to limit computation complexity, memory load and run-time. Reducing the step length in the first place then leads to a more thorough exploration of the field, where fewer nodes are created (due to a lower number of near nodes), and thus also the limitation on near nodes is applied less frequently, such that new nodes are created more distributed across the field. The same explanation applies to the results of the search radius, where also near nodes need to be selected for rewiring and thus a dense, distributed network of nodes allows for the most efficient rewiring.

8.2.5 Stopping Criteria and Number of Iterations

When examining the results for simulations with rows, the differences in performance between the stopping criteria are as expected: higher performance for a stricter criterion. Notably, even for the strict criterion the run-times are significantly reduced compared to simulations without a stopping criterion, such as test case 4. However, when looking at the results for simulations with rows, an irregularity seems present: the performance is slightly lower for the stricter criterion. While this may seem odd at first, delving deeper into the results gives clearer insights. The default setting for rewiring during the simulations is "on" (the rewiring function is applied). However, when analyzing the results before rewiring, it can be seen that the (rounded) average performance of the top 20 nodes (used for the stopping criterion), is slightly higher for the stricter criterion: 0.233 compared to 0.229 for the mild criterion.

8.2.6 Horizon Planning

As described before, horizon planning is tested by running a simulation of 12 days with a horizon of two cycles. The performance results and run-times in Tables 3 and 4 for cases 15 and 16 are the averages of these 12 days. The performances are close together for both cases. This could be due to the fact that no matter the horizon, the best path simply focuses on reaching as much high entropy areas as it can within the budget. When there is horizon planning applied, the second cycle of the route is therefore less effective, as it covers lower entropy areas which have not been covered yet. The run-times are shorter for horizon planning when applied to a field with rows, while for fields without rows, the run-times are shorter for single path planning. The table gives the average run-time of all days. The run-times increase each day, which is probably due to the increasing spread of the pathogen. Notably, the run-times are more than an order of magnitude higher for horizon planning than single path planning in fields without rows.

Table 5: Complementary Results for Evaluation of Horizon Planning Efficiency

Field	Setting	Final entropy	World model inaccuracy
No rows	Horizon path	1106.54	0.0
No rows	Single path	1264.11	0.0
Rows	Horizon path	2590.18	3.4e-4
Rows	Single path	2538.45	3.7e-4

Furthermore, more results can be analyzed that are meaningful for longer simulations. The first result that can be compared is the final entropy after the last day. This gives an indication of the monitoring quality over the span of multiple paths (i.e. monitoring sessions), though the entropy is also influenced by the spreading model. The values in Table 5 indicate that these final entropy values are close for the cases, where it is slightly higher for horizon planning for both field types. The second result drawn after the 12-day simulation is the world model inaccuracy. Table 5 indicates that the inaccuracy is the same for horizon planning in fields without rows and lower for horizon planning in fields with rows, each compared with the inaccuracy of single path planning, which can be explained as results from the performance values. Another reason for the implementation of horizon planning was the logic of the path: planning a double path was expected to lead to less distributed paths; paths that complement each other in coverage of high entropy areas throughout a span of multiple days. For this reason, the plotted paths are analyzed and visualized in Appendix E. For fields with rows, no significant difference in the distribution of the path across the field can be detected between the settings. For fields without rows, the single path setting already shows paths that are not very distributed, but concentrated on certain areas. Therefore, little difference occurs with the horizon paths.

8.2.7 Structured and unstructured fields

Initially, RIG was designed for unstructured search spaces (Hollinger and Sukhatme, 2014). It is not explicitly a goal of the thesis to compare the planner for structured and unstructured fields, as it has no practical meaning; farmers will not create an unstructured field to improve monitoring performance. However, a comparison allows for an evaluation of the performance of the planner in structured fields. Comparing the results raises a few noteworthy observations: in structured fields (Table 4), performance values are around half as high as in unstructured fields (Table 3) for nearly every testing case. Additionally, run-times are higher, ranging from a factor 2 to even a factor 10 higher, with a number of exceptions where run-times are lower. These observations regarding performance and run-time can be explained by the nature of the algorithm: it creates a tree-based structure, adding nodes within reach until a budget is reached. It relies on the assumptions that distance calculation is a non-complex and fast operation and that many new nodes are created in each operation. However, it is known from the design process (Section 5.10) that distance calculation with rows in not straightforward, thus leading to higher run-times, even with the applied strategies for run-time reduction. Additionally, when comparing paths with an equal amount of iterations of unstructured and structured fields (e.g. Figures 24 and 22) it seems the case that the structured paths are under-developed in comparison, seeing that often more informative paths are possible with the same budget. This may be due to the fact that the tree-based structure is less developed in structured fields, thus requiring more iterations to converge to higher performance.

An additional observation follows from Table 5, which shows that the world model inaccuracy for unstructured fields converges to zero, while the inaccuracy for structured fields remains present, although being minor.

9 Conclusion and Discussion

In this concluding chapter, the focus is on discussing the key findings and conclusions derived from the results. Subsequently, a comprehensive examination of the research is presented, encompassing evaluations and recommendations for future studies.

Several aspects were tested in the simulations. The informedness of the planner is the most important factor, as it relates to the main research goal as introduced in Chapter 1: "Designing a planning agent that takes into account information from spreading models such that it can monitor the spread of pathogens and weeds in crop fields while minimizing entropy". The performance values of and the world model accuracy values show major improvements of the informed planner over the uninformed planner, thus confirming the hypothesis for the efficiency of an informed planner for monitoring, as the informed planner has a higher ability to reduce entropy, therefore enabling a more accurate spreading model.

The results of the budget tests have significant practical implications, as the results form an indication of the required robot for a field size. An interesting conclusion that can be drawn based on Figure 16a, is that increasing budget leads to less than linear increase in performance due to a non-uniform information distribution. This result should be taken into account when deciding on the required path length (or battery) capacity of a robot to reach a performance level. In contrast, this effect is a lot weaker in the case of fields with rows, in other words: linearly increasing the budget leads to nearly linear increase in performance. Thus, for fields with rows, an investment in a robot with higher battery capacity can be expected to be worthy in relation to the expected performance. It should be noted that in both types of fields, the focus is on budgets where the robot can still not cover (even nearly) the entire fields, as it would be more efficient in that case to apply a coverage planner (see Section 2.2.2).

The original rewiring function from RRT* (Noreen et al., 2016) is not suitable to informative planning problems, as these aim to maximize a submodular information metric instead of minimizing cost, and therefore the function was modified as described in Chapter 5. The improved rewiring resulted in a significant gain in the gathered information of the planner at a small cost of a slight run-time increase (see Section 8.2).

The results on different settings for the step length and search radius are not as expected in Section 8.1, where both relative and absolute effects of these variables were anticipated. For fields with rows, no effects were observed on performance, while in fields without rows, only a smaller step length improves performance, independent of the search radius value. Explanation for the observed effects is given in Section 8.2.4. Further exploration into optimal values for these variables is suggested. As indicated in this explanation, there is likely a performance improvement with an increase in these values until reaching the optimum point, beyond which the performance stabilizes or even diminishes.

The stopping criterion's performance effects intertwine with rewiring and iterations, increasing the complexity of results for evaluation. An increasing number of iterations, caused by a stricter stopping criterion, generally leads to increased performance, but the effect of rewiring may overrule this effect. The relation of performance over iterations may also show a "knee in the curve" which forms a good point for a stopping criterion. The suggestion strengthens when one compares case 4 (no stopping criterion, so most iterations), case 14 (strict criterion, so slightly fewer iterations) and case 13 (mild criterion, so fewest iterations): case 14 shows the best results in the simulation with rows (hinting at an optimum), while this case shows lesser results in the simulation without rows (possibly indicating the optimum was already passed). However, since only two versions of the stopping criterion were tested, it is impossible to verify this suggestion. The stopping criterion might thus be a good solution to balance performance and run-time, but to draw applicable conclusions, more research is required. Future research should analyze the interaction between iterations and rewiring effects and evaluate different stopping criteria for both field types.

The efficiency of the horizon planning strategy was analyzed based on extensive comparisons between the single path and horizon path settings. The results are not unanimous on the effect of horizon planning, but still demonstrate the lack of benefits of the strategy. For fields without rows, performance is equal, run-times are majorly higher, final entropy is lower and inaccuracy is equal. For fields with rows, performance is slightly higher, run-times are lower, inaccuracy is lower, but entropy is higher. Most of the differences in these measures between the settings are minor. For both field types, clear benefits of horizon planning are lacking. It can be concluded that while non-greedy planning strategies might lead to noteworthy performance increase, the currently applied strategy does not lead to the desired results.

There are a number of discussion points that relate to different parts of the research than a specific tested aspect. The discussion points that are touched upon in the rest of the chapter are design of a modular architecture, application to fields with rows, simulation testing and the spreading module.

While the first goal of the thesis related to the informedness of the planner, the supplementary second goal was as follows: "Creating a modular software architecture for autonomously monitoring the spread of pathogens and weeds in crop fields". The basis of this architecture is outlined in Chapter 3, after which the different modules are described in detail. It can be concluded that the design goal for the modular architecture has been met. Furthermore, it is advised to maintain the modular nature of the architecture in future research for two reasons. Firstly, the modular design has proved to simplify independent testing, design and improvement of the modules throughout the project. Secondly, the modular design allows simultaneous (multi-disciplinary) work on the modules by independent research teams, which would be highly beneficial in future research such as discussed in the following discussion points.

A novelty of this research was the application of the RIG algorithm to a structured field, divided into rows of crops. This application required major adjustment throughout different functions as discussed thoroughly in Chapter 5. Drawing conclusions on the performance results is challenging and potentially misleading due to the lack of a fair comparison with other informative planners in structured fields, since there is little research available on such topics. As discussed in Section 8.2.7, performance is lower and run-times are higher in structured fields compared to unstructured fields. The run-times are not prioritized in this research - foremost because the research is more of exploratory nature and additionally because no real-time processing by the robot due to offline path computation - however, run-times relate to potential costs, applicability and scalability of the algorithm. Based on the described outcomes, the current implemented planner seems less suited for informative monitoring of structured fields. Therefore, more research is recommended on alternative informative planners for structured fields, preferably with a focus on a planner that benefits from the given structure of the field to reduce computational complexity. Such a follow-up research enables a valid comparison with the current planner and can indicate whether performance and run-times for structured fields could be improved.

Extensive real-life testing is crucial, especially for an architecture designed for changing and unpredictable environments. While more variety in the current simulation could provide additional insights, it is preferred to test in real-life settings, such as on potato fields with the Origin One robot. While the spreading module contains a stochastic element simulating part of the entropy in the real agricultural fields, the designed architecture is only to be evaluated as a whole after testing with realistic scenarios.

Closely related to the previous discussion point, is the discussion point on the spreading module, which was based on expert models in literature from the agricultural sector. These models were chosen specifically with a few features in mind: wide applicability and generalizability, spatial and temporal model aspects and the (simplistic) mathematical nature of the models. However, with the main focus of the thesis being on the planning aspect of monitoring pathogens and weeds, the spreading module has not reached its full potential. Recommended topics for future research are combinations of different pathogen and weed species within a map (and the effects of competition) and more sophisticated spreading models that allow for detailed modelling of influencing factors. This thesis begins to bridge the gap between agricultural and technical domains. Future research can build on this effort by further developing the architecture in a multi-disciplinary team of researchers. The Synergia project forms a valuable starting point for this collaboration.

Lastly, it is recommended to extend the algorithm such that it can be applied to disease control as well. It is an option to then replace the aim of monitoring with control by supplying the planner with the spreading map instead of the uncertainty map. However, it is more valuable to combine the tasks into a single planner that must balance exploration (monitoring) and exploitation (control).

To conclude, this thesis forms a foundation for informative path planning for disease monitoring. Disease monitoring is a prerequisite for control in agriculture and aligns with the challenges of precision farming. A proof of concept is created for budgeted planning for autonomous disease monitoring by mobile robots, driven by a spreading model based on expert knowledge, using observational input of previous path executions.

References

Auld, B., & Coote, B. (1980). A model of a spreading plant population. *Oikos*, 287–292.

Auld, B., & Coote, B. (1990). Invade: Towards the simulation of plant spread. *Agriculture, ecosystems & environment*, *30*(1-2), 121–128.

Avular. (2023). *The ranger - autonomous driving platform*. https://avular.com/ranger (accessed: 04.01.2023)

Binney, J., & Sukhatme, G. S. (2012). Branch and bound for informative path planning, 2147–2154.

Cobbenhagen, A., Antunes, D. J., van de Molengraft, M., & Heemels, W. (2021). Opportunities for control engineering in arable precision agriculture. *Annual Reviews in Control*, *51*, 47–55.

Doyle, C. (1991). Mathematical models in weed management. *Crop Protection*, *10*(6), 432–444.

Galceran, E., & Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, *61*(12), 1258–1276.

Hameed, I. A. (2018). A coverage planner for multi-robot systems in agriculture, 698–704.

Hollinger, G. A., & Sukhatme, G. S. (2014). Sampling-based robotic information gathering algorithms. *The International Journal of Robotics Research*, *33*(9), 1271–1287.

Jeon, H. Y., Tian, L. F., & Zhu, H. (2011). Robust crop and weed segmentation under uncontrolled outdoor illumination. *Sensors*, *11*(6), 6270–6283.

Kampmeijer, P., & Zadoks, J. (1977). Epimul, a simulator of foci and epidemics in mixtures, multilines, and mosaics of resistant and suscptible plants.

Kurzer, K. (2016). *Path planning in unstructured environments: A real-time hybrid a* implementation for fast and deterministic path generation for the kth research concept vehicle* (Doctoral dissertation). https://doi.org/10.13140/RG.2.2.10091.49444

Mavridou, E., Vrochidou, E., Papakostas, G. A., Pachidis, T., & Kaburlasos, V. G. (2019). Machine vision systems in precision agriculture for crop farming. *Journal of Imaging*, *5*(12), 89.

Noreen, I., Khan, A., & Habib, Z. (2016). Optimal path planning using rrt* based approaches: A survey and future directions. *International Journal of Advanced Computer Science and Applications*, *7*(11).

NWO. (2019a). *Miljoenensubsidie voor onderzoek naar duurzame landbouwproductie*. https://www.wur.nl/nl/nieuws/miljoenensubsidie-voor-onderzoek-naar-duurzame-landbouwproductie.htm (accessed: 07.12.2022)

NWO. (2019b). *Vijf grote interdisciplinaire consortia versterken kennis en innovatie in nederland*. https://www.nwo.nl/nieuws/vijf-grote-interdisciplinaire-consortia-versterken-kennis-en-innovatie-nederland (accessed: 07.12.2022)

Papaix, J., Adamczyk-Chauvat, K., Bouvier, A., Kiêu, K., Touzeau, S., Lannou, C., & Monod, H. (2014). Pathogen population dynamics in agricultural landscapes: The ddal modelling framework. *Infection, Genetics and Evolution*, *27*, 509–520.

Ruiz-Meza, J., & Montoya-Torres, J. R. (2022). A systematic literature review for the tourist trip design problem: Extensions, solution techniques and future research lines. *Operations Research Perspectives*, 100228.

Somerville, G. J., Sønderskov, M., Mathiassen, S. K., & Metcalfe, H. (2020). Spatial modelling of within-field weed populations; a review. *Agronomy*, *10*(7), 1044.

Van Agtmaal, M., Straathof, A., Termorshuizen, A., Teurlincx, S., Hundscheid, M., Ruyters, S., Busschaert, P., Lievens, B., & de Boer, W. (2017). Exploring the reservoir of potential fungal plant pathogens in agricultural soil. *Applied Soil Ecology*, *121*, 152–160.

van Esch, H. (2023). *Github repository for thesis on informative path planning for the monitoring of pathogens and weeds for mobile robots*. https://github.com/hildeesch/thesis/tree/master van Mourik, S. (2019). *Synergia - ecologisch gebaseerde systeemverandering met hulp van high tech in de landbouw*. https://agritechcampus.nl/sites/default/files/2022-03/4.b%20Synergia%20-%20Greenport%20NHN%2010-03-2022.pdf (accessed: 07.12.2022)

Vansteenwegen, P., Souffriau, W., & Van Oudheusden, D. (2011). The orienteering problem: A survey. *European Journal of Operational Research*, *209*(1), 1–10.

Villette, S., Maillot, T., Guillemin, J. P., & Douzals, J.-P. (2021). Simulation-aided study of herbicide patch spraying: Influence of spraying features and weed spatial distributions. *Computers and Electronics in Agriculture*, *182*, 105981.

Wageningen University & Research. (2019). *Synergia*. https://www.wur.nl/en/project/synergia.htm (accessed: 21.12.2022)

Witting, C., Fehr, M., Bähnemann, R., Oleynikova, H., & Siegwart, R. (2018). History-aware autonomous exploration in confined environments using mavs, 1–9.

Xiong, Y., Ge, Y., Liang, Y., & Blackmore, S. (2017). Development of a prototype robot and fast path-planning algorithm for static laser weeding. *Computers and Electronics in Agriculture*, 494–503. https://doi.org/https://doi.org/10.1016/j.compag.2017.11.023

Zhou, H. (2020). *Github repository for informed rrt* algorithm*. https://github.com/zhm-real/PathPlanning/blob/master/Sampling_based_Planning/rrt_2D/informed_rrt_star.py

A Appendix A: Variable Overview

Table 6 contains the most relevant variables used throughout the different modules and functions. Each variable is accompanied by a brief description and the function or module it is used in, whenever applicable. The variables come back in the methodology where they are explained in detail.

Table 6: Overview of Variables.

Variable	Description	Function
Spreading model	Grid map where each grid cell contains a value representing the predicted density of pathogens or weeds	Spreading and monitoring module
World model	Grid map containing the knowledge of the robot based on monitored data representing the spread of pathogens or weeds	Spreading and monitoring module
Entropy map	Grid map where each grid cell contains a value representing the uncertainty of the prediction of the spreading map, characterized by the possible variation as predicted by the spreading model	Spreading, planning and monitoring module
Spreading factor	Combination of context and type-defining variables determining spread	Spreading module
Information value	Uncertainty/ entropy at a grid cell	Spreading and planning module
Node cost value	Path length up to a node	Planning module
Total cost value	Path length including the part to the end position	Planning module
Node information value	Sum of information of grid cells of a path up to the node	Planning module
Total information value	Summed path information value including the part to the end position	Planning module
Budget	Maximum allowed path length	Planning module
Step length	Maximum distance between nodes to form connection	Sampling
Search radius	Radius around new node evaluated for rerouting	Rewiring
Iteration	Each time that a new location is sampled	Planning module

B Appendix B: Spreading Models

This appendix dives into the workings of the spreading models EPIMUL and INVADE. Through pseudo-code and further details, it is described how the models operate and yield spatial maps with the spread and entropy. The differences between the models is highlighted. It should be emphasized that the pseudo-code listings provide simplified versions of the functions as to enable understanding and readability for the reader, which may occasionally require reduced realism of coding.

Both the spreading model for pathogens as for weeds consist of three functions: spread, uncertainty and update. The spread function initializes the spatial spreading map and creates the initial infections. Note that this function exists purely for the sake of simulation, as in a real environment, one would not "create" any infections, but monitor the existing infections in the field. The uncertainty function creates a spatial entropy map based on the current spreading map, which is passed on to the planning module for monitoring purposes. This uncertainty function characterizes the entropy that follows from the prediction of the spread: based on the disease characteristics (of the given pathogen or weed), there is knowledge about the expected spread in the next time step. However, this expected spread comes with a certain standard deviation, which characterizes the variation in spread, caused by internal and external influences such as weather circumstances, climate, humidity and natural variation in disease spread. The uncertainty function takes the standard deviation (STD) and spatial mapping of the disease and from there generates an entropy map where the intensity (grid cell values) indicate the level of uncertainty, which in turn indicate the locations where monitoring is most wanted. The update function takes the spreading map of the previous time step and uses the disease characteristics to predict the spreading map of the next time step.

For simulation purposes, a difference is made between the actual spreading map and the world model of the robot. In reality, the actual spreading map would not be available, as this characterizes the real status of the spread of disease in the field; the ground truth. Due to the stochastic nature of the spreading models, there is a difference between the real and the predicted spreading map. The disease matrix (pathogenmatrix or weedmatrix) contains the ground truth spreading map, while the estimated matrix (pathogenmatrix_est or weedmatrix_est) contains the world model of the robot. The difference between these maps is created to mimic real circumstances where the world model is incomplete, such that monitoring remains necessary to have accurate information on the status of spread. As such, the effectiveness of the planner can be tested realistically.

For further details on the expected value of a grid cell ($E_{(x,y)}$), see Equation 1. Note that this equation already takes disease characteristics into account, such as the spreading range, the reproduction rate, the reproduction fraction (for pathogen) and the density saturation. The stochasticity is introduced by replacing this expected value by the actual value, which is created by dividing by the mean reproduction rate and multiplying by the actual reproduction rate (randomly computed from the normal distribution).

B.1 EPIMUL: Pathogen Spreading Model

The pathogen spreading model is an implementation of the EPIMUL model (Kampmeijer and Zadoks, 1977). It can be observed in the pseudo-code that the spread and update function are very similar, aside from the fact that the spread function also creates the initial epicentres for the infections.

Listing 5: EPIMUL: spread function pseudo-code

```
def pathogenspread(weed, plantmatrix):
    # simulating the stochastic reproduction rate based on the normal
        distribution
    reproductionrate = np.random.normal(pathogen.reproductionrate,
        pathogen.STD)
    # create the indicated amount of patches
    for patches in range(pathogen.patchnr):
        epicentre = (randint,randint) # randomly chosen centre of
            infection
        curspread = randint(pathogen.duration) # the current patch is
            spreading at most since the given infection duration
        # the pathogen is spreading for the current spreading duration
            from the epicentre
        for day in range(curspread):
            for (x,y) in range(-spreadrange,spreadrange):
                [x,y] = [x,y] + epicentre # location is the distance
                    from the epicentre
                pathogenmatrix[y,x] = E(x,y)*reproductionrate/pathogen.
                    reproductionrate
                pathogenmatrix_est[y,x] = E(x,y)
    uncertaintymatrix = uncertaintypathogen(plantmatrix,
        pathogenmatrix_est_new, pathogen)
    return pathogenmatrix, pathogenmatrix_est, uncertaintymatrix
```

Listing 6: EPIMUL: uncertainty function pseudo-code

```
def uncertaintypathogen(plantmatrix,pathogenmatrix,pathogen):
    # looping over every position in the field:
    for (x,y) in range(plantmatrix):
        uncertaintymatrix[y,x] = E(x,y) * pathogen.STD
    return uncertaintymatrix
```

Listing 7: EPIMUL: update function pseudo-code

```
def pathogenupdate(plantmatrix,pathogenmatrix, pathogenmatrix_est,
    pathogen):
    # simulating the stochastic reproduction rate based on the normal
        distribution
    reproductionrate = np.random.normal(pathogen.reproductionrate,
        pathogen.STD)
    # looping over every position in the field:
    for (x,y) in range(plantmatrix):
        pathogenmatrix_new[y,x] = E(x,y)*reproductionrate/pathogen.
            reproductionrate
        pathogenmatrix_est\_new[y,x] = E(x,y)
    uncertaintymatrix_new = uncertaintyweeds(plantmatrix,
        pathogenmatrix_est_new, pathogen)
    return pathogenmatrix_new, pathogenmatrix_est_new,
        uncertaintymatrix_new
```

B.2 INVADE: Weed Spreading Model

A big difference between the implementations of INVADE and EPIMUL is the application of stochasticity. As is the case with the pathogen spreading model, the weed spreading model consists of three functions: creating the spreading map, creating the entropy map and updating at the next time step. However, where the creation and updating step were very similar in the pathogen spreading model, these show major differences for weeds. This is due to the fact that originally, INVADE did not contain any stochasticity, but was completely deterministic in nature. In the weedsspread function (see Listing 8), this feature was retained. This is due to a great difference in the nature of pathogens and weeds: while pathogens spread in a more random fashion, weeds tend to form patches, from where they spread. Diverging from the original INVADE model, stochasticity in the spread is introduced in the updating step (weedsupdate function, see Listing 10) as a contribution to this thesis in a similar manner as it occurs in EPIMUL.

Listing 8: INVADE: spread function pseudo-code

```
def weedsspread(weed, plantmatrix):
    # create the indicated amount of patches
    for patches in range(weed.patchnr):
        epicentre = (randint,randint) # randomly chosen centre of
            infection
        curspread = randint(weed.patchsize) # size of patch is at most
            the given weed patchsize
        # the weed is placed everywhere within spreading distance from
            the epicentre
        for (x,y) in range(-curspread,curspread):
            [x,y] = [x,y] + epicentre # location is the distance from
                the epicentre
            # if the weed is parasitic (attaching to crops) and there is
                 a plant, density = high
            if weed.plantattach and plantmatrix[y,x]>0:
                weeddensity = 1.0
            # if the weed is competing (not attaching to crops) and
                there is no plant, density = high
            elif not weed.plantattach and plantmatrix[y,x]>0:
                weeddensity = 1.0
            # else, density is low
            else:
                weeddensity = 0.5
            weedmatrix[y,x] = weeddensity
    uncertaintymatrix = uncertaintyweeds(plantmatrix,weedmatrix,weed)
    return weedmatrix, uncertaintymatrix
```

Listing 9: INVADE: uncertainty function pseudo-code

```
def uncertaintyweeds(plantmatrix,weedmatrix,weed):
    # looping over every position in the field:
    for (x,y) in range(plantmatrix):
        # if the weed is parasitic (attaching to crops) and there is a
            plant, factor = high
        if weed.plantattach and plantmatrix[y,x]>0:
            factor = 1.0
        # if the weed is competing (not attaching to crops) and there is
             no plant, factor = high
        elif not weed.plantattach and plantmatrix[y,x]==0:
            factor = 1.0
        # else, factor is low
        else:
            factor = 0.5
        uncertaintymatrix[y,x] = E(x,y) * weed.STD * factor
    return uncertaintymatrix
```

Listing 10: INVADE: update function pseudo-code

```
def weedsupdate(plantmatrix,weedmatrix, weedmatrix_est, weed):
    # simulating the stochastic reproduction rate based on the normal
        distribution
    reproductionrate = np.random.normal(weed.reproductionrate, weed.STD)
    # looping over every position in the field:
    for (x,y) in range(plantmatrix):
        # if the weed is parasitic (attaching to crops) and there is a
            plant, factor = high
        if weed.plantattach and plantmatrix[y,x]>0:
            factor = 1.0
        # if the weed is competing (not attaching to crops) and there is
             no plant, factor = high
        elif not weed.plantattach and plantmatrix[y,x]==0:
            factor = 1.0
        # else, factor is low
        else:
            factor = 0.5
        weedmatrix_new[y,x] = E(x,y)*reproductionrate/weed.
            reproductionrate
        weedmatrix_est\_new[y,x] = E(x,y)
    uncertaintymatrix_new = uncertaintyweeds(plantmatrix,
        weedmatrix_est_new, weed)
    return weedmatrix_new, weedmatrix_est_new, uncertaintymatrix_new
```

C Appendix C: Rewiring Design Process

C.1 Path-Based Rewiring

The path-based rewiring strategy was applied first. This strategy consists of changing the parent of the node to the new node, including the whole path leading up to this new node, hence the referral "path-based". In other words: the rewiring includes the new node as a complete node, including its cost, information value and the parent.

RIG deals with an information metric alongside the path cost, in contrast to RRT*. Therefore, the criteria for rewiring are a lower cost while retaining or improving the gathered information. The steps followed by the rewiring function can be found in pseudo-code in Listing 11.

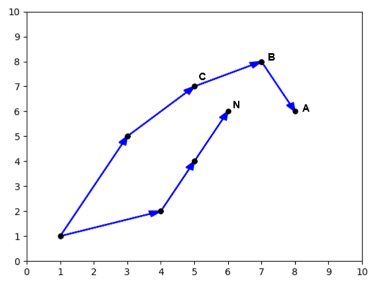
Listing 11: Path-Based Rewiring Pseudo-Code

```
def Rewiring(newnode,nearnodes, search_radius):
    for node in nearnodes: # within search_radius
        # path: from newnode to node
        newinfo = newnode.info + info_path
        newcost = newnode.cost + cost_path
        if (newcost<node.cost) and (newinfo>=node.info):
            node.parent = newnode
            node.info = newinfo
            node.cost = newcost
```

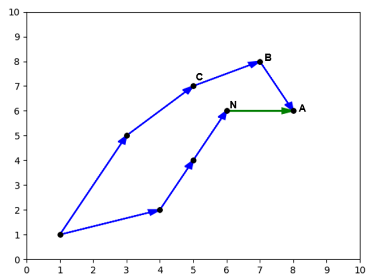
The steps are also visualized in Figure 17. A node (A, Figure 17a) is rewired, getting the new node (N) with its path as parent (Figure 17b,c), when this leads to a decrease in cost and no change or an increase in information value up to the current node (A).

A disadvantage of this method is that many paths "disappear": the path that originally led to the node is not further expanded anymore, as the branch of the tree-graph has been relocated. Also, when a path is initially just a little bit more promising than others, many nodes will then be linked to this path. Both of these features are very useful in RRT*: in this classical case, the only metric that matters is the length of the path. A shorter path simply leads to a better result. However, in this project, there has to be dealt with submodular information. Additionally, it is not the aim to have a path that is as short as possible, but it is meant to have a path as informative as possible, as long as it remains within the budget.

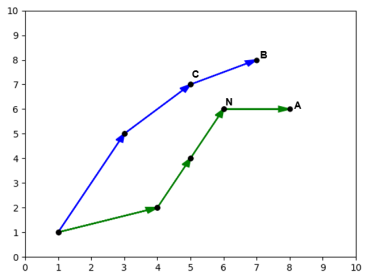
The fact that many paths are not expanded further resulted in negative effects of rewiring: applying no rewiring at all often lead to more informative paths. Rewiring leads to a decrease in the amount of different paths created and hereby limits the growth of the tree-based structure of paths and thus the broad exploration of the map. These results have lead to the conclusion that this form of rewiring does not fit the problem at hand.



(a) Before rewiring

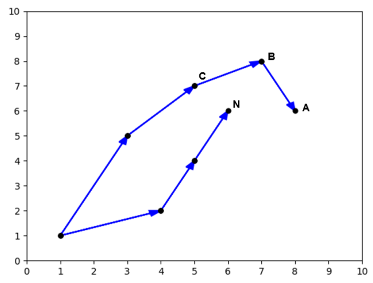


(b) Rewiring node A to new parent N

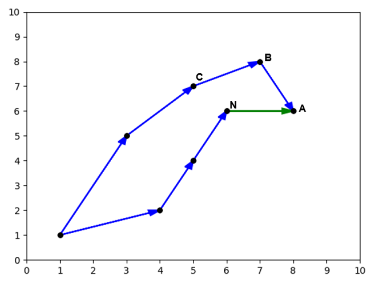


(c) Path after rewiring

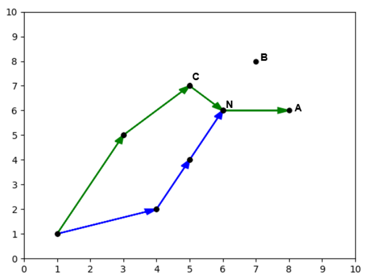
Figure 17: Rewiring steps for path-based rewiring (blue is original path, green is rewired path)



(a) Before rewiring

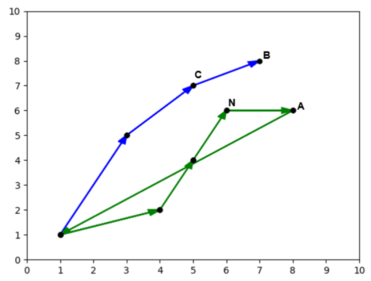


(b) Rewiring node A to new parent N

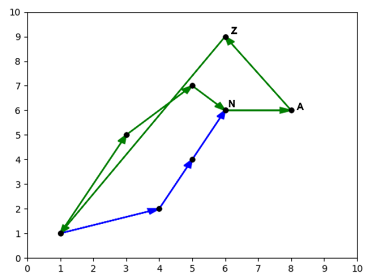


(c) Path after rewiring

Figure 18: Rewiring steps for location-based rewiring (blue is original path, green is rewired path)



(a) Total path after rewiring



(b) Total path of child after rewiring

Figure 19: Rewired paths including path back to docking station (blue is original path, green is rewired path)

C.2 Location-Based Rewiring

Location-based rewiring was created as a variant to the most commonly used strategy in RRT*. In this case, the node (A) is rewired using only the location of the new node (N) instead of the entire path up to the new node, as was the case for path-based rewiring, see Figure 18a,b,c. The criteria are still the same. The function steps are outlined in Listing 12.

Listing 12: Location-Based Rewiring Pseudo-Code

```
def Rewiring(newnode, nearnodes, search_radius):
    for node in nearnodes: # within search_radius
        # path1: from node.parent.parent to newnode
        # path2: from newnode to node
        newinfo = node.parent.parent.info + info_path1 +
            info_path2
        newcost = node.parent.parent.cost + cost_path1 +
            cost_path2
        if (newcost<node.cost) and (newinfo>=node.info):
            node.parent = newnode
            node.info = newinfo
            node.cost = newcost
            newnode.parent = node.parent.parent
```

A node (A) is rewired, meaning that the parent of the parent of the node (C, two steps back along the path) is linked to the new node and the new node (N) is then linked to the node (A). In other words, the location of the parent of the node (B) is replaced by the location of the new node (N).

This method was found to be less limiting to exploration than the path-based method. This is because during rewiring, not an entire branch of the tree-structure of paths is cut off, but it is merely restructured. As such, less paths are removed and it will not occur that when one path is particularly informative, many nodes will be rewired to the same path. Instead, it may use parts of this path by changing parent locations.

However, there is still the problem of submodular information: when the information up to the node is increasing, it may occur that the total information is not increasing (due to crossing of the path). Figure 19a shows an example where the path up to the node may be informative, but the path back to the docking station largely overlaps that path, thus yielding no new information. When applying rewiring, the total information may not increase, since the path overlaps itself for a significant part of the path from the node to the the end location, thereby not gaining new information. Additionally, it may not be beneficial for the information value of the children of the node, again with the reason of submodular information and the possibility of overlapping pieces of path. This is illustrated in Figure 19b, which shows the rewired path (to node A) and an extension to a child node (Z). Here, the total information of the node (A), including the part back to the end location (not drawn in the figure), is increasing, while the total information of the path of the child node (Z) might not be increasing due to the large overlap in the part back to the end location.

For this reason, the total information was taken into account. A recalculating method was introduced. In the rewiring function and recalculating function, both the information value up the node and the total information are considered, after which it can be decided which are rewired. Multiple methods were attempted:

- When the total information at the node is not increasing, no rewiring was applied at all

- When the total information at the node is not increasing, a copy is made of the non-rewired node before rewiring, after which the children are connected to the copy or the rewired node, based on what gives them a higher total information value
- When the total information at the node is not increasing, a copy is made of the non-rewired node before rewiring. The children are connected to both the copy and the rewired node when their total information value does not increase with the rewired node

These methods all gave different results in comparison regarding performance, but aside from a few exceptions, not applying any rewiring at all seemed to give the best results in the end. To understand this result, the difference between the aim of RRT* and RIG as mentioned before should be kept in mind: while RRT* simply minimizes point-to-point path length, RIG aims to maximize information gain within a budget. Additionally, this information metric is submodular. Even when the current children of all nodes are considered, the rewiring function can not consider the effects of rewiring to future path segments, since these are still to be created. Therefore, while rewiring may prove beneficial to a node and all its current children, it may actually lead to worse paths in following iterations where more nodes are sampled. While certain versions of the rewiring attempts described above seemed to have beneficial effects, these results were not reproducable and not guaranteed. In other words, rewiring has no guarantee of improvement since its effect on the information value of future nodes is unpredictable.

The only option to apply this type of rewiring while having the guarantee of improvement is to always save a copy of every node and every child when rewiring. However, this is infeasible due to memory constraints. As such, the memory constraints combined with the testing results lead to the conclusion that also location-based rewiring is deemed unfit for the problem at hand.

C.3 Hindsight rewiring

This section described more design features and argumentation of the hindsight rewiring function. It includes the comparison to location-based rewiring and an explanation of the paths on which rewiring is applied. For the general explanation of hindsight rewiring, see Section 5.4.

From Listing 2, it can be concluded that the rewiring function in itself is very similar to the procedure for location-based rewiring. Only the selection of the node to be rewired ("node") and the node to which this node is rewired ("nearnode") has slightly changed. Instead of considering only a single node to be rewired, rewiring in hindsight considers every node up to the final node at the end of the path to be rewired. The Recalculate function recomputes the cost and information values of all nodes further along the path, which allows to verify improvement along the full path.

This function is applied to a number of paths, based on the total information value. The reason for including multiple paths instead of only the single best is that rewiring might lead to such improvements that paths might overtake each other in information value. The reason for not including all paths is a question of balance: rewiring takes up run-time and paths with a much lower information value than the best path are unlikely to become the best path after rewiring. The number of paths that are rewired was determined through tests on the amount of improvement caused by rewiring. In independent executions with diverse maps, there is a lot of difference in the effect of rewiring; there were observations of increase in information value varying between 10-90%. However, within one execution of the algorithm, it is observed that the range of increase through rewiring is quite stable in different nodes. Mostly the range of increase is within 20% difference, for instance when all nodes increase around 40-60% through rewiring. Therefore, it was chosen to select all the nodes that have a total information value that lies within 20% of the best node (before rewiring). Using this measure, there is the highest probability of including all nodes that could become the best node after rewiring, without including an unnecessarily large amount of nodes. When the information value is below the threshold, the chances are slim that it will improve so drastically that it will be higher than the information value of the best node after rewiring.

D Appendix D: Structured Steering Design Process

The approach to find a location on this path from the nearest node to the sampled location within the step length distance was also altered throughout the project. This step is referred to as the Steer function in Listing 1. First, it was found by starting at the sampled location and then in steps of a certain length going towards the nearest node until the distance was within the step length, thus yielding a location somewhere along the path from the node to the nearest node. However, this approach has a high computational load, as it needs to calculate the new distance at every step, and additionally it may not find the furthest point within the step length from the nearest node in case the step size is bigger than 1. For this reason, another approach was implemented which is based on principles of linear optimization. Sections are taken where the distance is evaluated. The sections consist of parts of the path, moving from the new node towards the nearest node. When the end section turns out to be too distant still, the distance to the next section end is calculated. If the end of the section is within the step length, each grid point is evaluated from that section end towards the new node, until the step length is reached. As such, large parts can be skipped when finding a location within the step length and the furthest possible grid point on the path is selected. This approach reduces the computational load of sampling, but still requires a number of computations.

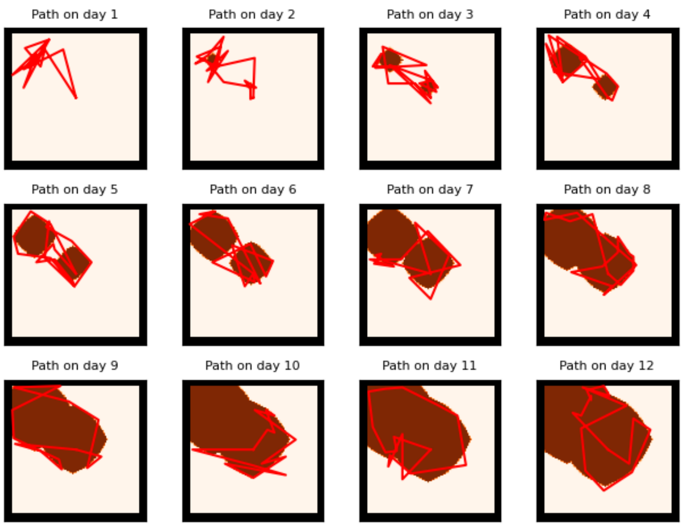
Later, a new approach was developed that coincided with the introduction of using an A* algorithm to compute the distance between points (see Section 5.10). In essence, this approach still largely uses the same principles: moving backward in sections, then moving forward in single steps.

E Appendix E: Computed Paths of Multi-Cycle Simulations

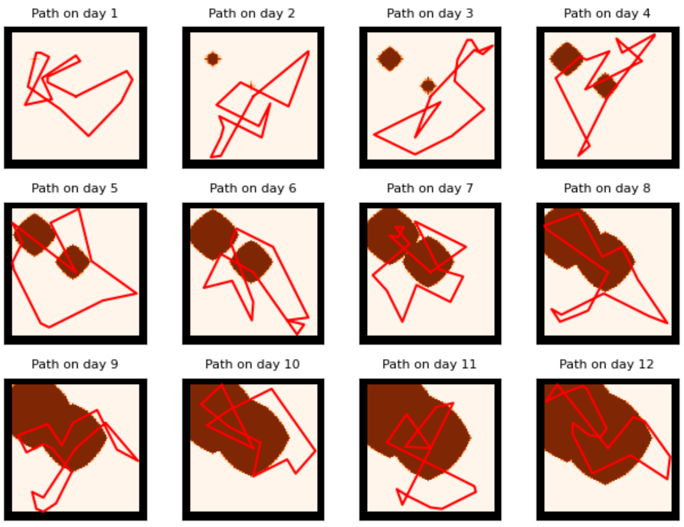
The following figures show the final paths of each day throughout a 12-day (cycle) simulation.

E.1 Informed planning

As explained in Section 8.2.1, this additional simulation visualizes even more clearly how the informed planner leads to a more accurate world model of the robot. These conclusions can be made by comparing each of the world models with the ground truth spreading model (Figure 21) and by paying attention to the paths themselves, which are more focused on the areas of spread for informed planner.



(a) Informed setting



(b) Uninformed setting

Figure 20: Final paths for 12-day simulation overlaid on the world model map

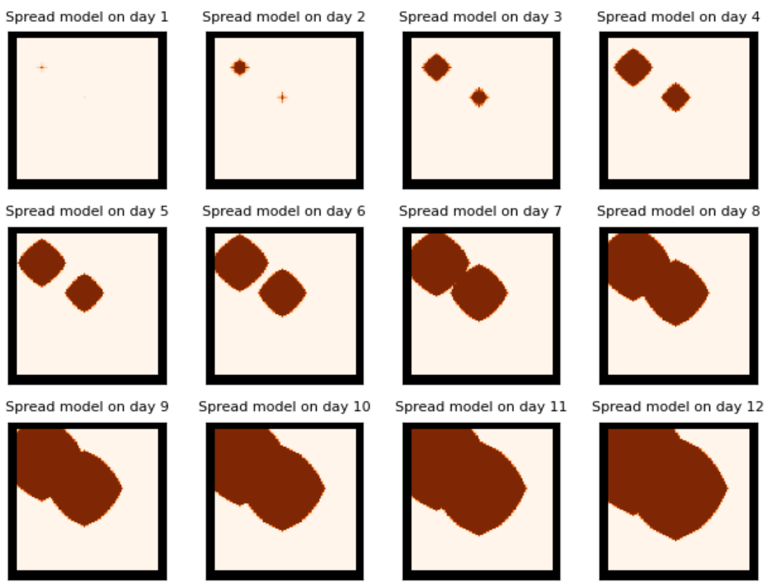


Figure 21: Ground truth spreading map for 12-day simulation

E.2 Horizon planning

The simulation is used for evaluation of the horizon planning aspect. As such, for both structured and unstructured fields, the simulation is executed once with the base setting (single path planning) and once with the horizon planning setting.

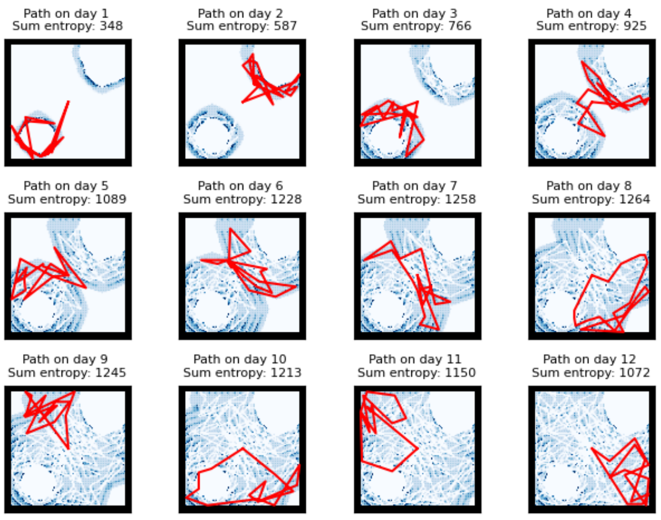


Figure 22: Single path in field without rows

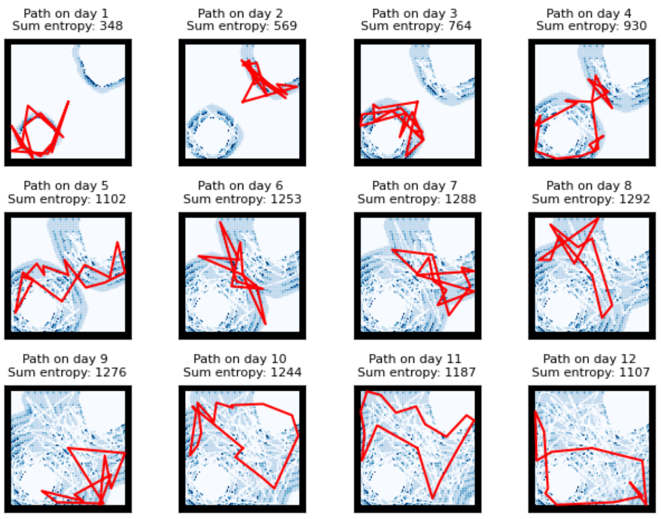


Figure 23: Horizon path in field without rows

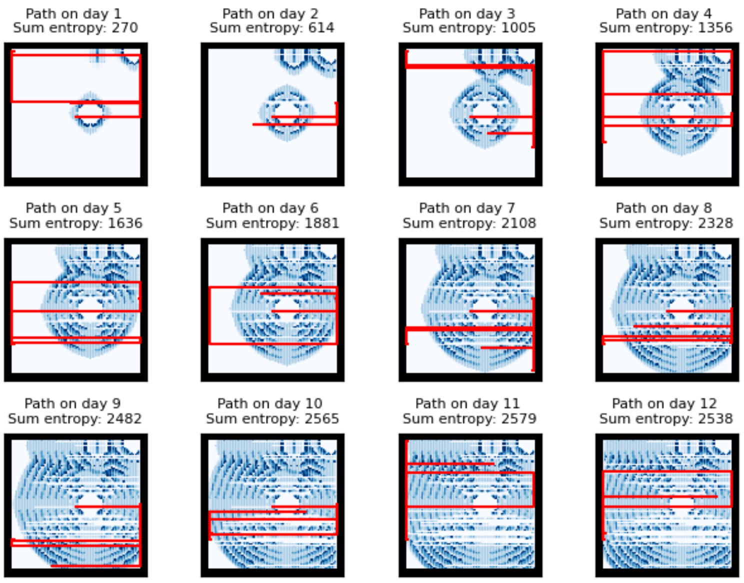


Figure 24: Single path in field with rows

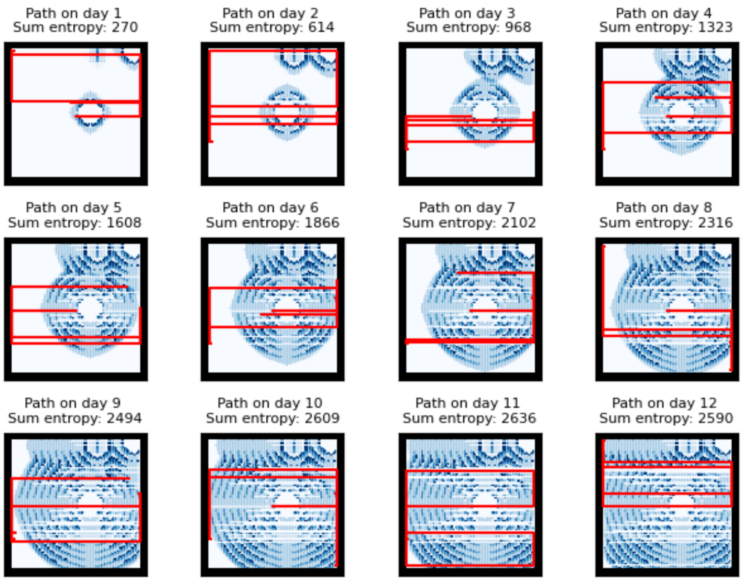


Figure 25: Horizon path in field with rows