University of Oxford

Department of Engineering Science

Engineering Third Year Design Project

System and Software Design of Multi-UAV Systems for Precision Viticulture

Author:

Lewis Ince

Zipei Liu

Ipek Sahbazoglu

Fraser Rennie

Supervisor:

Min Chen

Jonathan Gammell

August 29, 2022

Contents

1 Introduction

This report details the design aspects and development processes of a framework for planning and executing a multiple unmanned aerial vehicle (UAV) precision viticulture operation. Precision Viticulture (PV) can be defined as technology-enabled management and monitoring of the spatial and temporal variability [1] of vineyards arising from factors such as pest, disease and irrigation[2]. The aim is to maximise grape yield, quality, and homogeneity while optimising the use of resources and environmental impact. Existing PV practices for field surveying (satellite or plane imagery) and agrochemical deployment (manual or ground-vehicle) are characterised by high costs, waste and operating times [3] due to low flexibility and precision. Hence, using these methods renders frequent surveying and spot-treatment economically infeasible and inefficient. This creates the need for a PV system that can provide low cost, high accuracy and frequent remote-sensing as well as resource efficient and highly precise treatment application. The use of UAVs across all industries is steadily growing due to their high flexibility, operability and agility while maintaining low cost and risk. Furthermore, UAVs are able to cover large areas much faster than people/vehicles on the ground [3]. The system we developed harnesses these competitive advantages to improve the efficiency of vineyard surveying, disease detection and agrochemical deployment methods. We designed the system to be provided as a service to the viticulturist.

The preliminary sections of the report will define the system objectives, requirements and specifications, present the service framework, which includes 2 stages of surveying to map the vineyard, identify diseased areas and classify the disease, and a spot-treatment stage, and hardware design decisions. The subsequent sections will detail the design and implementation process of three key components of our system: (i) Image Analysis, (ii) Human-robot interface (HRI) and (iii) Path Planning (PP). The image analysis section details the development of an image stitching algorithm and a machine learning classifier that use the aerial footage and vegetation indices to map the vineyard, identify diseased areas, and classify the disease. The next section presents the HRI, specifically the UI/UX design and data management of the web application developed. The app allows for the seamless execution of our framework by control and management of the system data flow between the operators and the hardware/software. The following path planning section, presents the framework design and implementation of optimal coverage path planning and task allocation for both the surveying and spot-treatment operations. Emphasis was placed on minimising the operation time, resource usage, and size of the UAV team. Finally, the report is concluded with a discussion of the results and future work.

1.1 Project Pipeline

The project pipeline consists of 3 main stages: two surveys and one actuator phase.

Ahead of the first stage, the project needs to be set up. To do this, the operator chooses a client and defines the area to survey via the HRI. On the day of a project, the HRI allocates the drones to the project through a page on the interface. The operator scans QR codes on each drone which contains their ID. The operator must be on location to deploy and manage the drones as well as monitor the system.

The first stage is the initial survey where the drones take multiple birds-eye view images of the field that are then stitched together to form a large composite image. The user defines the field boundary via the HRI application. The API makes the boundary available to the path planning algorithm which generates the optimal path for each drone. The survey drones take images at regular intervals along the path according to the requirements for image stitching. At each image position, the drones collect images from different sections of the light spectrum, such as near infrared (NIR) and visible light, via multispectral cameras. This is done using the DJI P4 Multispectral drone. The image processing algorithms extract the vegetation indices data which provide indications of the health of the field. After analysis, the data is used to choose the survey area for the next stage.

The second stage is an enhanced survey which aims to identify the diseases affecting unhealthy areas. Effective identification requires the drones to survey at a lower altitude for higher definition images. The path planning algorithm generates the paths for each drone. A machine learning model uses these images to classify the plants as diseased or not diseased. If the disease classifier classifies a disease with a certainty below the required threshold, the specimen in question is sent to the operator for confirmation.

The final stage is the action stage in which the aim is to treat the crops which are identified as diseased with the appropriate chemical. The DJI Agras T16 has chemical reservoirs and a custom spraying mechanism. A set of these drones is used to spray the crops.

Once the stages are finished, a report is generated for the client by the backend program which details the health of the field as well as information about any treatment of the field.

1.2 Requirements

1.2.1 Stage 1

Our project must generate a map with a heatmap overview on grapevine vegetation indices. Stage 1 must have automated flight control as well as time and resource efficient path planning. During stage 1, the number of UAVs must be increased to survey the entire field, without requiring any recharge. Furthermore, during this stage, we would require real-time feedback on HRI.

1.2.2 Stage 2

Our project requires individual grapevine image sampling, so that the neural network has the ability to detect different diseases, including Esca, Black Rot and Leaf Blight.

1.2.3 Stage 3

Our project requires the ability to provide preliminary treatment on plants using a spraying mechanism attached to a spraying drone, to treat the detected diseases.

1.3 Specifications

Our project specifies the need for multiple surveying drones and treatment drones, for the relevant stages. Stable communication within a typical vineyard size of around 50 acres, or 200,000 square metres is necessary. During stage 1, the survey drones must be able to cover 25 acres per hour, and fly at a constant height, discussed further during section 3.1.1. For stage 2, the survey drone must take no more than 5 seconds to decelerate to a stop to take close-up images. It also must take no more than 5 seconds to accelerate away from each grapevine, according to the path planned. This equates to a 2.4 acre/hour requirement for a 200,000 square metre vineyard, with typical row and crop spacing.

On the processing side, during Stage 1, a resolution of 2cm/pixel is required to allow for a sufficient heatmap size. An overlap of 80% or more is required such that an adequate inlier set is found for image stitching, discussed in section 3.1. For this to occur, images must be uploaded to an API, which can provide them to the image stitching algorithm in sequential order, per drone. Classifications of diseases must have at least a 90% accuracy, and a 95% sensitivity. Labelling a diseased leaf as ”healthy” can lead to spreading of fungal disease, losing viticulturists revenue, hence the importance of such a high sensitivity. After classification of disease, if the probability provided by the neural network is less than

a threshold of 70%, the image must be forwarded to an operator, as to be able to confirm or rectify the presence of the relevant disease.

The requirements for the Control systems are that during stages 1 and 2, the drone must stay within the box path planned, and for stage 3, that the spraying fork attachment must also conform to control requirements, maintaining stability throughout.

1.4 Acknowledgements

Our group would like to express our sincerest appreciation to Zipei Liu who was instrumental throughout the year in working Control systems for this project including trajectory control, collision avoidance and path alignment as well as determining the business feasibility and the financial aspect of the project.

2 HRI

2.1 Aims & Design Principles

The goal of the human-robot interface is to provide data about the system to the operator, enabling control and management of the drones. There are many design principles that affect the usability and effectiveness of the interface.[4]

Structured management of information. This is the organisation and representation of both mission critical and nominal data about the system. The architecture and flow of information should create an intuitive interface that minimises the cognitive load of the operator.

Hierarchy of information. The system manages thousands of data points per project, much of which requires data processing before being displayed to the operator. When the user us required to make a decision, there should be a clear hierarchy of data available to them. The more significant the data, the more clear and easy it is to view. Less commonly required data is accessed via an action of the operator such as clicking the drone type for more details. This leads to the idea that all data should be available to the operator through a set of actions.[5]

Minimisation of the potential for human error. The operator is assumed to be familiar with the drone system. Their familiarity helps them complete actions faster and understand what is being asked of them at each stage. To minimise errors, validation on information from the operator will mitigate errors.

Balanced workload for the operator. There is a clear necessity for a person to operate the system but the reliance on the operator should be minimal. High workload can lead to reduced vigilance and increased operator error.[6] A low workload would reduce vigilance as well as mental disengagement, both due to lack of stimulation from the interface. This can cause human error if not addressed.

2.2 Requirements

The aim of the HRI is to provide an interface for the operator to monitor progress, control drones, and view data. Since the system is mostly autonomous, the operator's workload is smaller than that of system which requires manual control of all of the drones. Most of the operator's tasks are therefore concerned with any issues that may arise as well as setting up the project constraints and goals. The operator is required to define the boundaries of the region of interest (ROI), which necessitates a visual interface in which the operator can draw the boundary. This is more effective than inputting a set of coordinates which is both inefficient and prone to error.[7]

2.3 Interface Architecture

A web-based application has been chosen for its versatility and accessibility: any device can use the software, provided that it has an internet connection and browser. The three main components of the application are the frontend, backend, and database. The backend provides a RESTful API which serves resources, manipulates data, and performs actions. The database is manipulated and maintained by the backend API. Users on any internet-enabled device can access the resources provided that they have the appropriate permissions. The frontend is the interface that users see in the browser and this focuses on visual representations of data which effectively convey the information about the system. It also provides a data flow for working with the system allowing for data input and human monitoring.[8]

2.3.1 RESTful APIs

REST is a set of architectural constraints for APIs. When a request is made to a route of the RESTful API, it will return a resource's state. The information can be delivered in various standard formats via the HTTP protocol. The most common format is JSON which is readable by both humans and machine. It is language-independent and follows a syntax of attribute-value pairs and arrays.[9] An example of some JSON representing a drone is below:

```
{
```

```
    "id": "SUR079",
    "battery": 78,
    "type": "Large Surveyor",
    "allocation": 8,
    "location": {
        "lat": 51.04175,
        "lng": 1.04124
    }
}
```


When a request is made to the API, it includes information in the “Header” as well as parameters defined in the URI. The parameters usually contain necessary information for identifying the requested resource or an option for an action. An example route with a parameter for a simple ’GET’ request to a drone resource is ’/api/drone/SUR079’. This would return the JSON data shown above since the ID parameter matches the resource.

A ’POST’ request is used to create a resource on the server. JSON data containing information about the resource is passed in the body of the request. The ID of the new resource is usually returned to the client for future reference.

’GET’ requests return a resource or set of resources defined by the parameters of the request. GET requests do not permit passing data in the body.

2.3.2 Backend Design

The backend uses a model-controller-route design. Models for each table are built by defining their fields and datatypes in a schema. The app uses Express which is a Node.js web framework that can handle HTTP requests on different URI paths (routes).

The backend makes heavy use of the Sequelize. This is a promised-based Node.js ORM tool which facilitates the processes of defining data models and associations much easier.[10] It also provides tools for querying data which enables the code to be more readable and faster to develop than using raw SQL queries.

The data models can be manipulated using CRUD operations (Create, Read, Update, Delete). Each

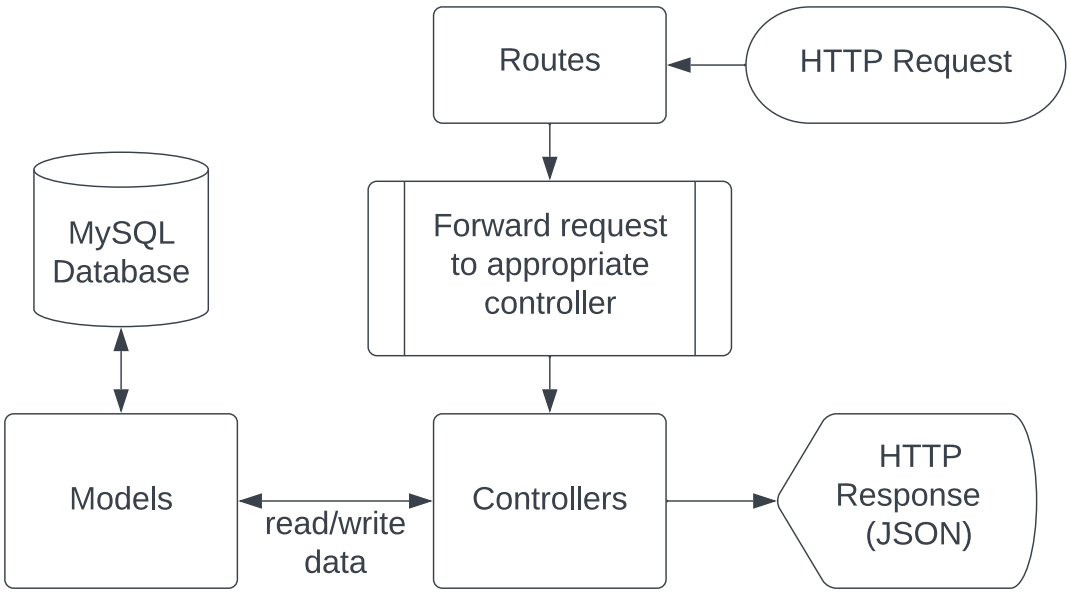


Fig. 1: HTTP Request to API

data model is a part of the overall database schema. The models define tables and create constraints on the datatypes and entries in the table. They are also used to validate and structure queries to the database. The application uses a relational database which stores and provides access to data entries that are interrelated through predefined associations. Each resource has a primary key that uniquely identifies it. For example the “drone type” model will have many “drones” associated with it, hence a many-to-one relationship.

The API has a set of routes which are the endpoints associated with actions and resources. Routing is how the API matches a URI to an action or resource. Each route has a HTTP method associated with it; ’GET’, ’POST’, ’PUT’, ’PATCH’ and ’DELETE’ which forward the request to the appropriate controller. An example of a ’GET’ request is seen below:

GET ’www.api.co.uk/api/project/:id’

The models and routes provide a framework within which the controllers can work. The controllers contain all of the logic and functions of the API. Within these, the functions use the parameters or data that has been passed in from the client to execute a predefined action. The controllers will perform the CRUD operations and can make additional API calls to external programs such as the Google Maps API. The overall pipeline of a HTTP request to the API is shown in Figure 1.

The implemented API has 3 sets of endpoints: projects, drones and clients. These 3 sets can easily be identified via their routes. Table 2 outlines the endpoints which are the paths after the domain of the

Method	Endpoint	Category	Description
GET	/api/clients	Clients	Returns a list of all clients
GET	/api/client/:client	Clients	Returns a single client
GET	/api/client/projects/:client	Clients	Returns all of the projects for a client
POST	/api/project/create	Project	Creates a project
GET	/api/project/:project	Project	Returns all of the details about a project
POST	/api/project/route	Project	Returns a path given a boundary
POST	/api/project/start/:project	Project	Sets a project to “in progress”
POST	/api/project/start/:project/:stage	Project	Sets the stage of a project
GET	/api/project/report/:project	Project	Returns PDF Report for a project
GET	/api/drones/available	Drones	Returns a list of all available drones
GET	/api/drone/location/:id	Drones	Returns the location of a drone
POST	/api/drone/location/:id	Drones	Updates a drone’s location
GET	/api/project/drones/:project	Drones	Returns a list of drones in a project
GET	/api/drone/type/:type	Drones	Returns a details of a type of drone

Fig. 2: API Route Table

API which is “http://localhost:3001” in development.

Each endpoint has a specific method coded to handle a call to that endpoint. For the function of reading a record from the database, the implementation is simple. For example, the endpoint that returns a single client has the pseudocode of:

Algorithm 1 Endpoint Controller

```
1: Parse client ID from request
2: Call Database for clients with a matching ID and limit to 1 record
3: if Record is found then
4:     Return the client and set the response status to 200 (success)
5: else
6:     Return 404 error (resource not found)
7: end if
```

2.3.3 Frontend Design

The frontend uses React which is a widely adopted JavaScript web framework that makes strong use of custom reusable components with the benefits of allowing for client side logic. React has access to a large library of open-source packages which add prebuilt functionality into the application. This allows for rapid development with a focus on design rather that implementation.

The use of functional components allows code to be reused across pages while only passing in the required data. This makes development and readability of code much more simple than a plain HTML, JavaScript and CSS website. Sass is used for design which is a superset of CSS meaning that all CSS

Fig. 3: Full Stack Web Application

is valid SCSS. It includes variables that make it easy to change the theme of the application.

Data is loaded from the API asynchronously which enables the pages to load while it waits for the data to be returned. Once the data has been returned, the JSON data is parsed into an object to be used across the page. Any changes to the data will dynamically update on the page. This means that users do not need to refresh the page for the most up-to-date information.

Figures and diagrams are generated using React Chartjs 2 which is an open-source React wrapper for the widely used chart.js library. This creates simple and effective charts that dynamically react to changing data. The charts also have some basic interactivity such as hovering for more specific data on data points.

2.3.4 Technologies Used

To link the frontend and backend, the frontend makes calls to the routes of the API which in turn, returns the appropriate data. Keeping the backend and frontend as separate programs allows for the API to be used by other user too. The API triggers the path planning program and sends data to it via the HTTP protocol. This data is then be sent back to the API which makes it available to the frontend. The database is the central storage area for data to be shared between the drones, interfaces and algorithms involved in the system. This setup is demonstrated in Figure 3.

The backend uses Express which is a Node.js web framework that handles the HTTP requests on different URI paths (routes). Middleware can be added throughout the request handling pipeline to process and format data into a suitable form for the client and drones.

Data is stored using a MySQL database. This is a relational database which suits the needs of the system well. A relational database links tables via common data such as a drone's unique ID. The primary benefit for this use case is that all agents in the system can access well formatted data through predefined API routes.

Another option for the database was to use a document-oriented database, however this was not chosen since it is not suited to the high amount of analysis and complex queries that are performed.

2.3.5 API Endpoints for Users

To make use of the centralised database, information is shared via a set of endpoints. These follow an intuitive standard that makes it easy for users to understand and use. It is important that users are aware of the data that is required to be sent in requests and what the response data will look like. This allows them to access the JSON reliably. To ensure the this, there is standard documentation of all of the endpoints.

Users can make the CRUD commands to each endpoint to retrieve, alter or store new data. To allow users to access the data, the API is hosted on a public server which enables any user to access the data. This would allow for anyone to retrieve, alter or create data on the database which is a security concern. To circumvent this, the API uses Cross-Origin Resource Sharing (CORS) which is an HTTP-header based mechanism that indicates the origin of the request. When a request to the server is made, there is a “preflight” request sent from the user that checks whether the server will permit the user to access the API. In the case of this API, all origins are allowed however any origins that are not of the frontend require an API key. In addition, only the API has access to the database through a username and password which prevents unauthorized direct database connections.

API keys are typically strings of letters and numbers that are included in the header of HTTP requests. This is a way to authenticate the user without login credentials. API keys are secret and should not be shared with other users or external parties since it allows them to access the API. When the user makes a request to the API and their origin is not the frontend, the key is checked against a list of authorized API keys. Once verified, the user is free to access the resources on the server.

2.4 UI/UX Design

UX refers to the user experience design while UI refers to the user interface design. Both element are critical to providing an effective GUI for the HRI. There are many factors that create a good interface which are discussed below.

2.4.1 Usability

The interface must allow the user to achieve their goal effectively and efficiently. At each point of the system, users should know how to interact with the interface intuitively and with ease. When the user needs to think about what is being asked for, this creates a delay and frustration. This can lead to inaccurate data entry and negligence with the system.

2.4.2 Findability

Throughout the user flow, various information will be needed at each step. The layout and design of the interface should be clear to make the data findable. A cluttered interface can cause confusion and frustration while the user searches for the information that they require. The layout choice is an important factor that affects the readability and findability of the interface. Clear divisions of sections that are consistently used for the same data allow users to quickly looks in the correct place for what they need. Keeping the sections consistent between pages and stages of the project decreases the time for searching for information.

2.4.3 Data Hierarchy

Data hierarchy relates to the importance of information at the current stage of the project. To establish the hierarchy, each stage is carefully analysed to determine what information is required to make an informed decision. The important information is shown clearly in the interface while less important information will require one or more actions for the user to access. All information is accessible to the user in some way with variety accessibility levels.

When a user initially sees a page, they scan over all of the content in the page, therefore, each section should clearly show what information it holds. There are multiple tools that can be employed to create a clear UI.[5]

The size of objects can be varied. Large elements will attract more attention. Therefore, the largest element will typically be looked at first. This element should begin the user journey and show a clear action if one is to be taken. It is, however, required that elements should create a visual balance. Elements should not be too large as this can impact and overshadow the rest of the content and information on the page.[11]

The colours on the page are another useful tool for splitting data and creating a hierarchy. It provides

another identifier of prioritisation. Stronger colours are usually used in places that is requiring some sort of interaction from the user and to place importance on data.[11]

Proximity of elements suggest that they are related to each other. Therefore, information that is important to the current action or input should be placed closely and be easily accessible. Proximity creates relationships. This limits the workload on the operator.

Alignment is a subtle but still useful tool to show information that should be noticed by the operator when completing an action. All relevant data should be aligned or slightly indented from its “parent”. This is a clear visual hierarchy, much like is commonly used with bullet points.

Negative space is the idea that the more space an element has around it, the more importance it holds. It is a simple tool that declutters the screen to prevent the amount of information on one page becoming overwhelming.

For example, all drones should be clearly shown at all stages however if all the data about every drone was on the interface at once, this would create an oversaturated screen with too much information. This makes decisions harder for the operator. Instead, more in-depth information about each drone is made available upon selection, while important details such as ID, battery and task progression is available at a high level.

2.5 Initial Designs

There were multiple iterations for designs throughout the development of the GUI. Despite using some of the UI/UX design ideas, the designs were still deemed insufficient. Figure 4 shows one of the initial designs for the dashboard. This design was weighted more heavily towards monitoring the drones rather than the data collected in each stage. It also lacks information about the current state of the project or any reminder of which client the project is for. The dashboard does define a level of data hierarchy in the drone list. Each box shows the drone ID, battery and task progress. Upon clicking the box, more data about the task at hand is shown.

This design is flawed in multiple aspects which lead to the improved design as described in the next section. The flaws in the design were identified upon analysing the initial design and speaking to team mates about the usability and effectiveness of the design. Each drone box takes up a significant area of the screen which does not scale well when there are more than 5 drones. It is also unclear which drone

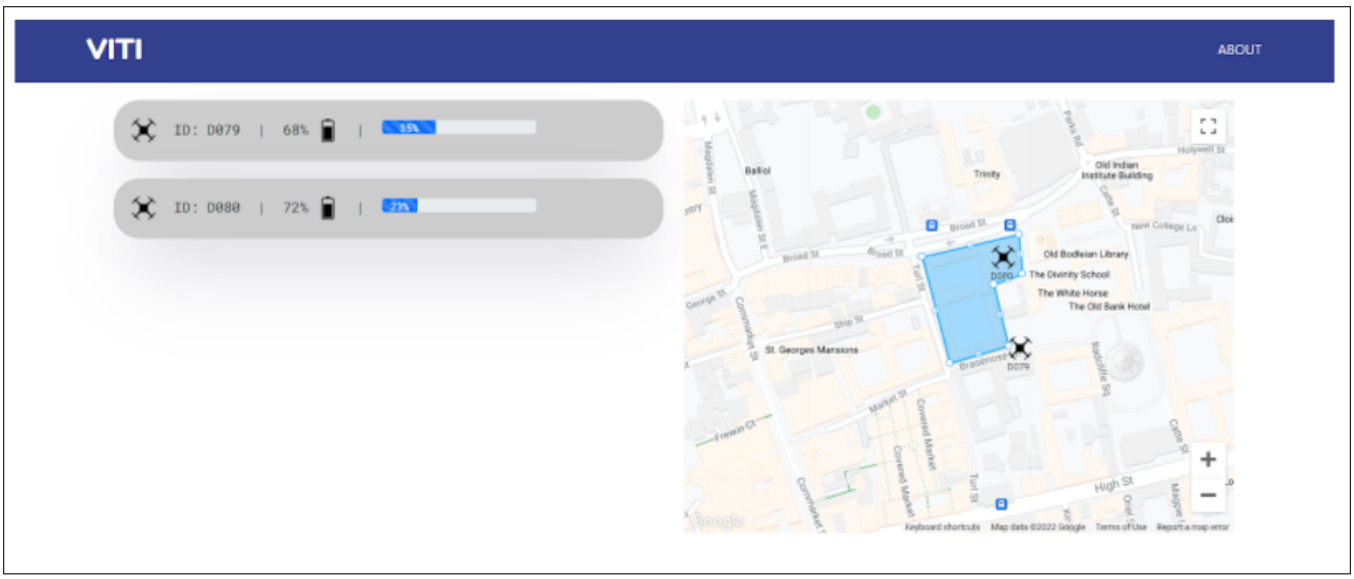


Fig. 4: Initial Dashboard Design

is associated with each box on the map since the only identifier is the ID.

The aim of the HRI is to provide an interface for the operator to monitor the various stages however this is not achieved though this interface. There is no feedback to the operator other than battery, position and task progress for each drone. The map shows a seemingly editable boundary which should not be possible at this stage. In addition, the paths of each drone are not available. This means that the operator cannot see if the drones are following the correct path.

Finally, there are no action buttons meaning that there is no control for the operator. This limits the operator to a purely monitoring role and they are not able to act upon the observations they make. These factors are addressed in the next iteration, described below.

2.6 Design Decisions

2.6.1 Setup Stage

The setup stage follows a strict user flow. It requests data from the user in steps and presents data that can be easily comprehended. When data is more understandable, the user is more likely to provide accurate data.

The user flow begins with a simple UI requesting the user to choose a client via a dropdown box. Once selected, a table shows the list of current projects associated with the client. At this point, the user can select any project that has already been set up. The information shown is the project's ID, name,

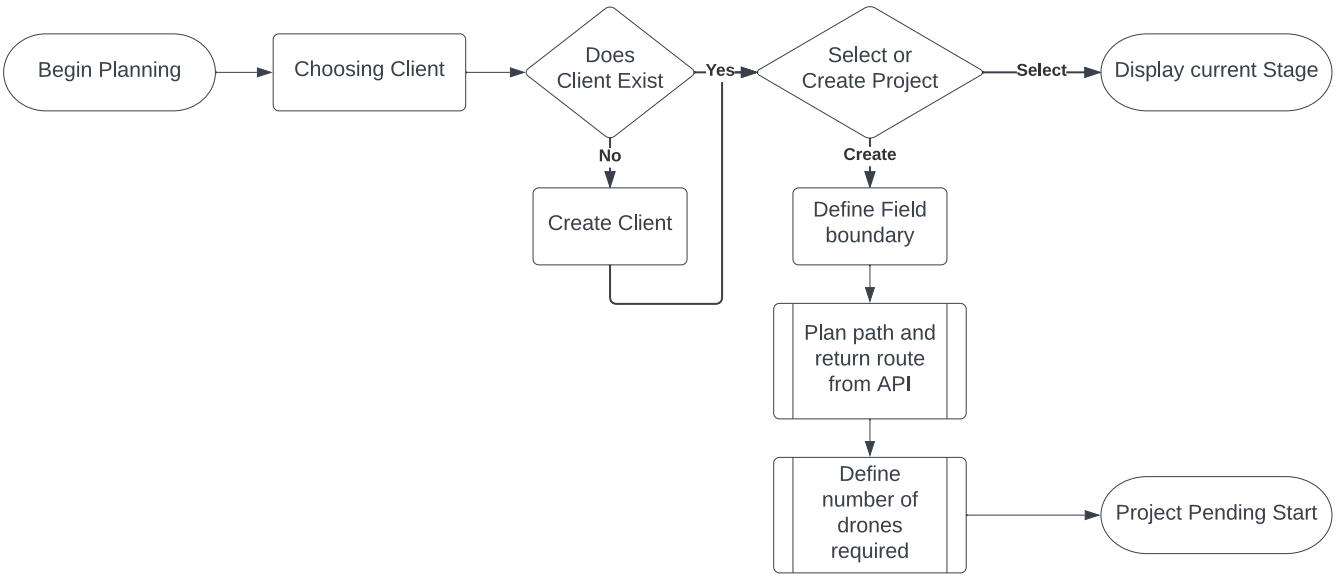


Fig. 5: Setup Stage Flowchart

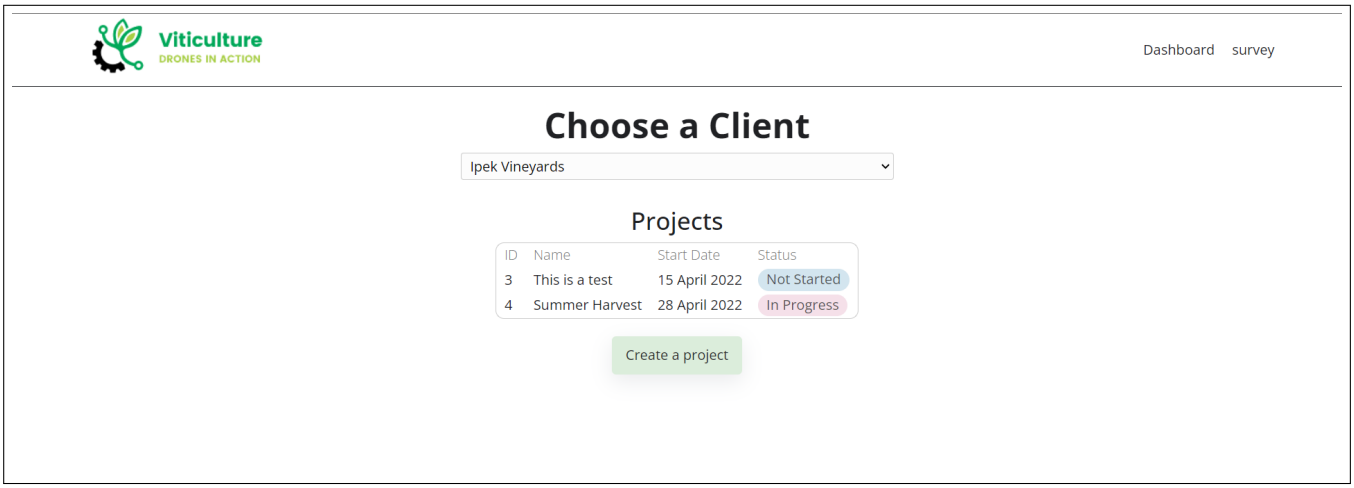


Fig. 6: Choose Client in Setup

status and start date as seen in figure 6. The projects are shown in ascending order on the start date field, meaning that the next project is shown first. If the user chooses to select a project, it simply shows the dashboard with the project information of the current stage. If the user chooses to create a project, they are taken down a different flow as shown in the flowchart in Figure 5.

There are two steps in the project planning phase. The first is to define the project name and date. Basic customer information is adjacent to the form to confirm which client that the project is for. This page is shown in Figure 7.

The form uses validation where necessary to mitigate inaccurate or bad data being entered. For example, the date must be today or in the future as a project cannot be planned for the past.

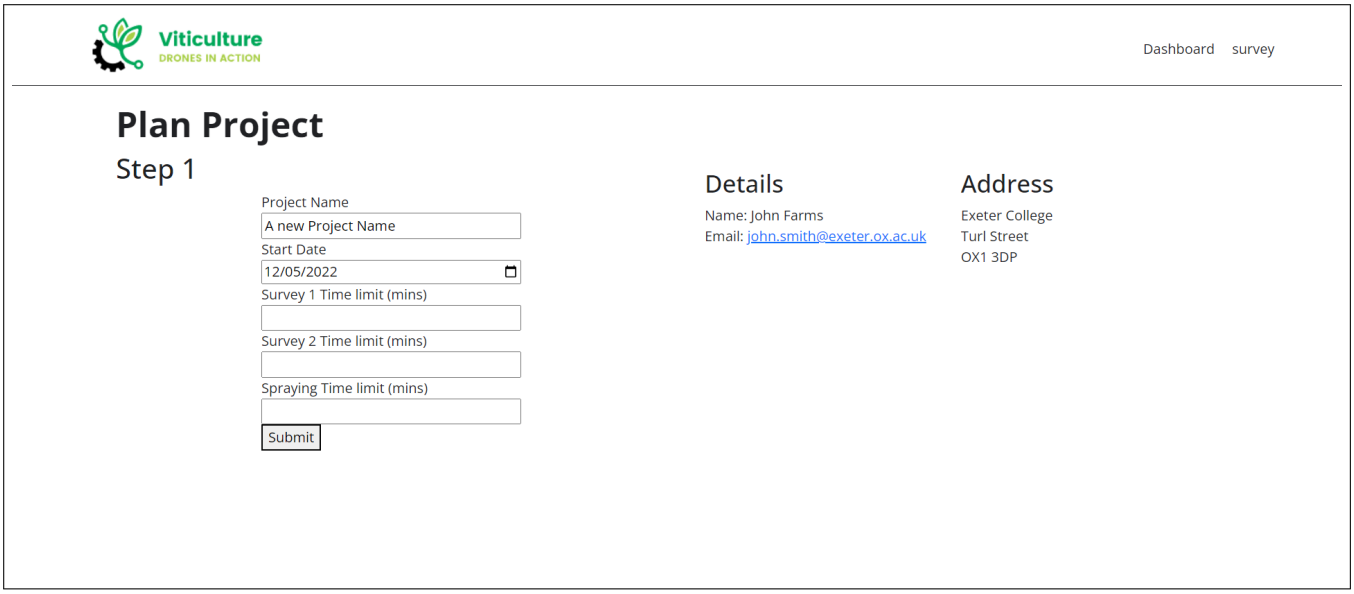


Fig. 7: Project setup

The next step in the setup is to define the boundary of the ROI. This is done manually by the user using the Google Maps API as shown in 8. A polygon is drawn onto the map which defines the boundary of the field. The polygon is initially set to a rectangle of arbitrary size. From here, the user can select the vertices of the polygon and reposition them. Additional vertices are placed by clicking the midpoint of any edge. This inserts a new vertex which can be repositioned.

Each time the user edits the boundary, the area of the ROI is recalculated to give an up-to-date value. This helps the user infer the scale of the task. The GCS coordinates are then sent to the backend which stores the boundary in the database and triggers the path planning program. The path planner returns a set of paths, one for each drone required, as a list of lists of coordinates. Once the backend has received these, they are made available on the frontend for operator review. The path planner also calculates the optimal number of drones for survey and the number of operators required.

The final page shows all of the planned data including the boundary and planned path on a map as seen on Figure 9. At this stage, the project is “not started” and can be initiated by clicking the start button which marks the project as “in progress”.

2.6.2 QR Codes

To begin the project, a set of drones must be allocated to the project. The number of drones of each type has been defined by the path planning algorithm. Drones need to be allocated to the project by

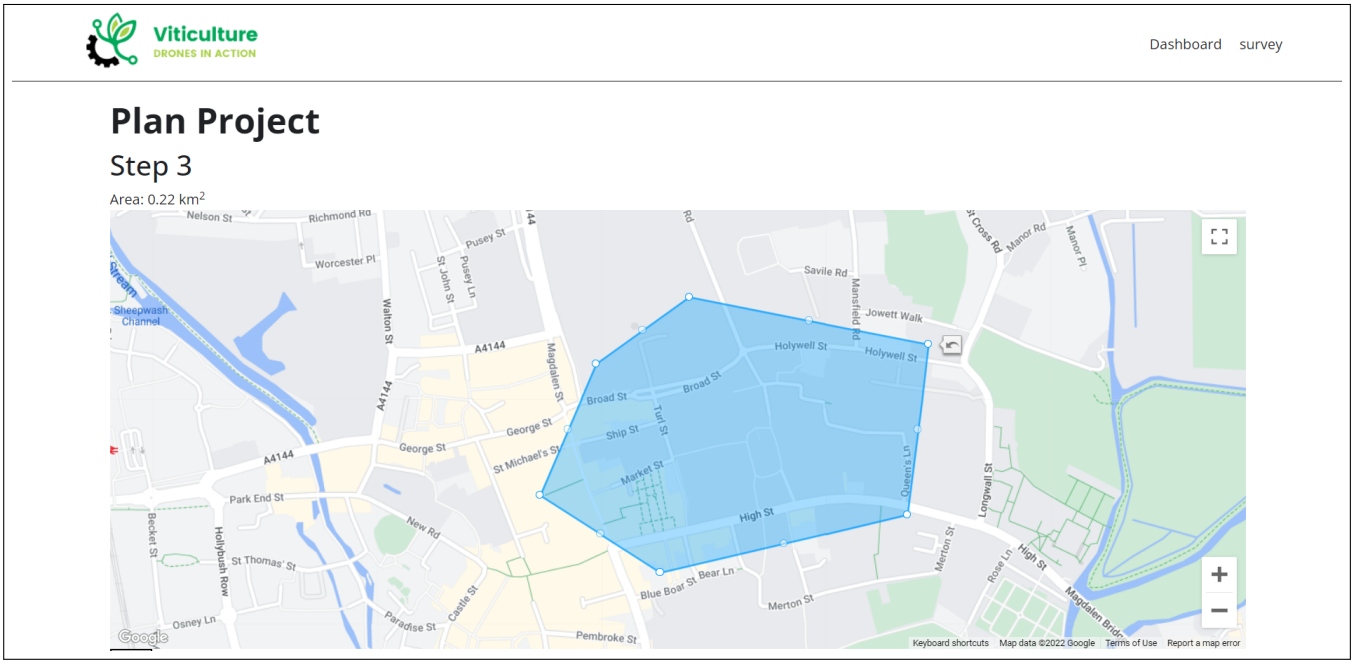


Fig. 8: Boundary Definition on Map

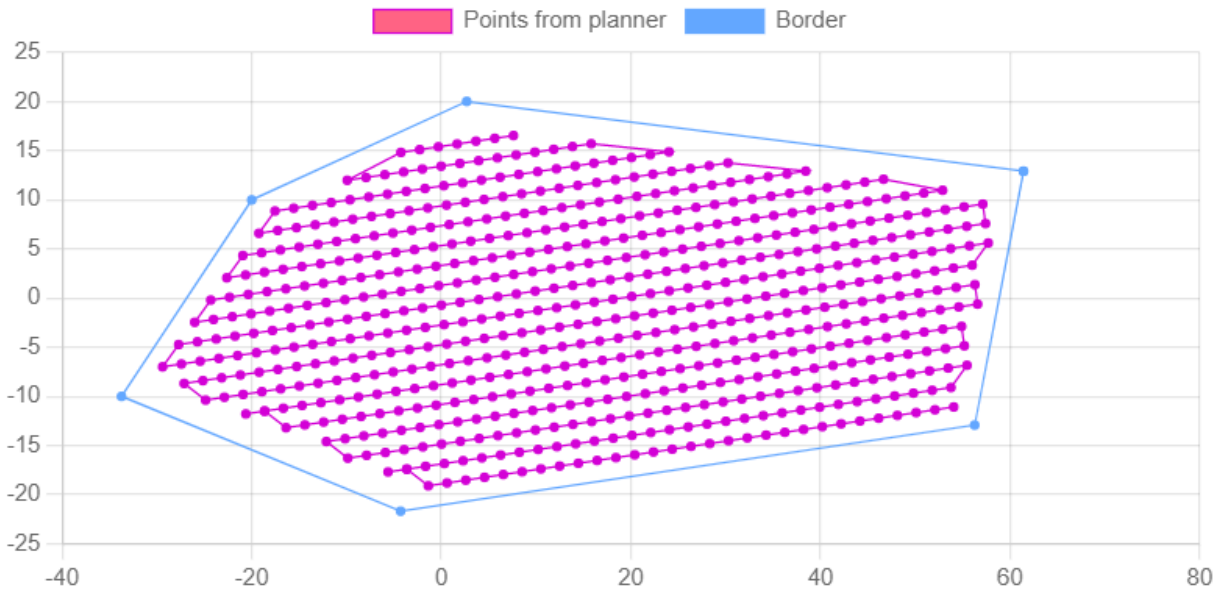


Fig. 9: Boundary and Path Results

entering IDs of the drones chosen. This can be done manually however this is slow and prone to errors. Manual entry error rates range from 18% - 40%.[12] Instead, the device's native camera, or an external webcam, is used to scan the QR code on each drone. QR codes are a strong tool as they provide fast and reliable information. QR codes can operate reliably with up to 30% of the data either missing or corrupted.[13] This makes them suitable for this use case since the drones are prone to wear and dirt which could obscure the QR codes.

This adds a new tool for the HRI to use. It expands the HRI from just a keyboard and mouse input to a visual one too. This method of entry is optimised for the repetitive data entry use case. Other options for visual entry are barcodes however these were disregarded due to their limited data payload size and lack of error correction.

The IDs are standardised by constraining each drone's QR code to the form 'Drone ID: ABC123'. Standardising mitigate errors by validating the ID as a drone's. In addition, it makes naming of drones easier and easily automatable. Survey drones are prefixed with 'SUR' followed by the number. E.g. 'SUR073'. This form is also consistent with the database. This makes the assumption that no more than 1000 drones of each type are bought.

Each drone can be scanned, validated according to the standard form and its record altered to record which project that it is allocated to. Additionally, drones cannot be scanned more than once. The design of the page shows the live webcam video feed and information about the allocation stage. The page states a clear action below the title: "Scan QR code on drone". As shown in Figure 10, the right-hand side of the page shows the number of drones remaining and a list of the scanned drones. Once all of the drones have been scanned, a success message allows the user to proceed to the next step. All of the information required for the user to scan the drones is available on the page.

When drones have been allocated, they are unable to be allocated to another project while the current one is in progress. Only once they are not needed will they be deallocated automatically.

2.6.3 Stage 1

Stage 1 is the initial surveying stage. The layout of this page has 4 main sections as seen in figure 12. At the top level, a hierarchy of data is already defined. Meta data is a critical part of the project and so is consistently displayed at the top of the dashboard.

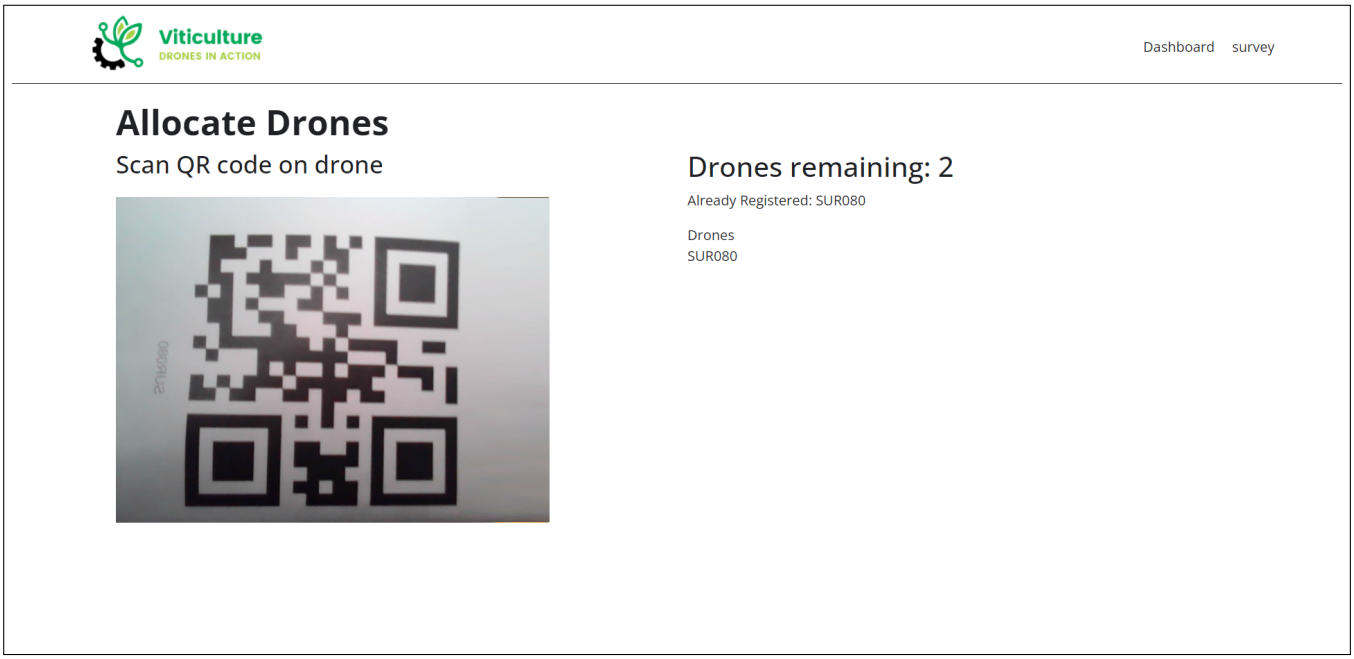


Fig. 10: Allocation Page with QR Scanner

Fig. 11: QR Code with Drone ID: ACT002

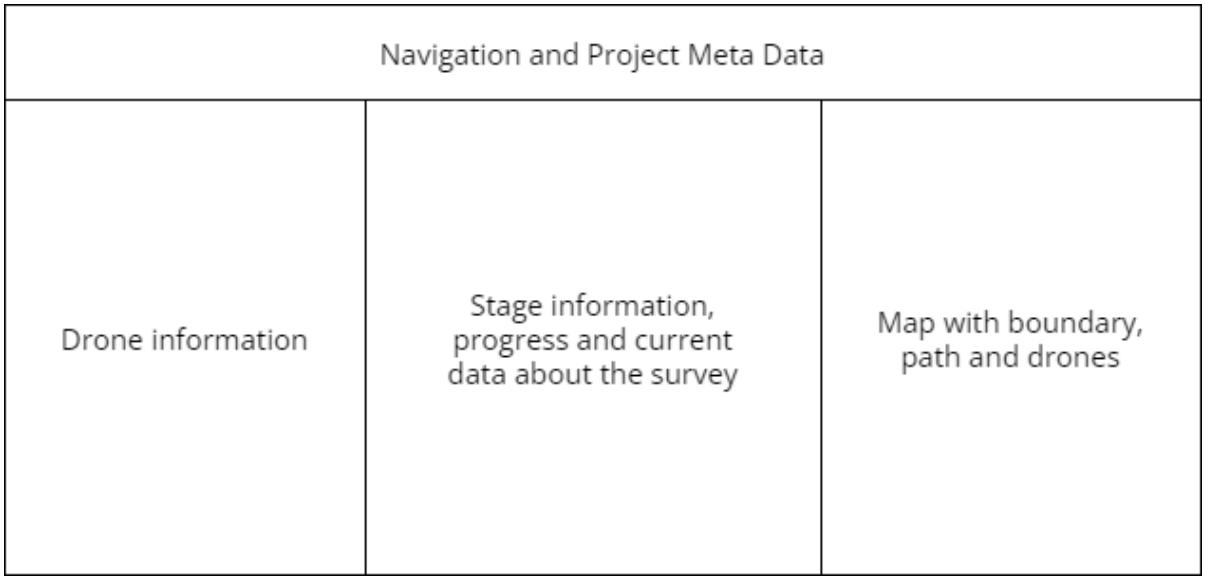


Fig. 12: Stage 1 Layout

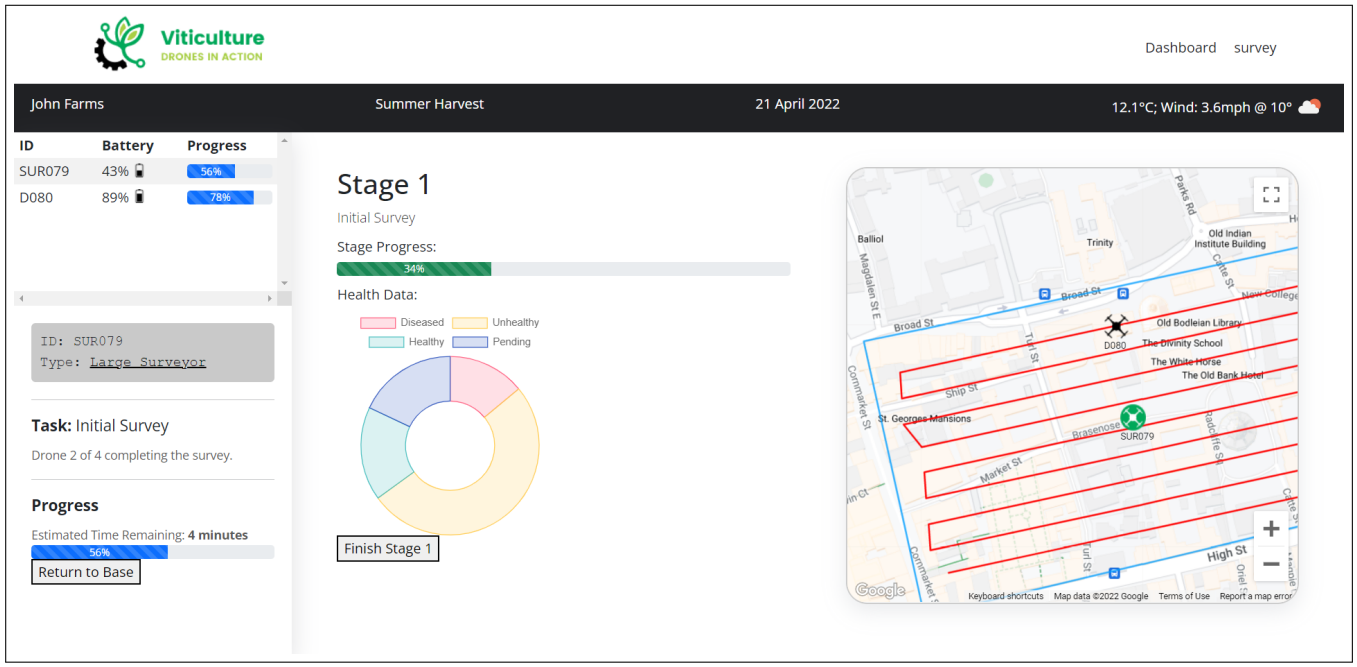


Fig. 13: Stage 1 Layout

The meta data section also show the current weather status which is live information provided via a request to the OpenWeather API using the REST paradigm. For conciseness, the information only contains the current temperature, the wind and an icon for the conditions - e.g. cloudy, raining etc. This allows the user to make an informed decision on whether to launch/continue the project or to postpone it for better weather conditions.

When the wind is above 20mph, a warning is displayed to advise the user of poor weather conditions. The P4 Multispectral drone has a maximum speed of 31 mph and has a wind limit of 22mph in its specification. During high winds, the battery levels will deplete at an accelerated rate which will cause drones to have to recharge during the survey as well as the survey taking an overall longer time.

The drone information section is an important part. It displays all of the drones that are currently assigned to the project and in action. The user is able to scroll through the list of drones that shows their top level data. Data pertinent to the operation of the drone is considered top level. This includes ID, battery and task progress. This information needs to be the most readable for the user, so a series of visual identifiers are used. This data was deemed most important because once the battery is too low or the task is 100% complete, the drone will return the the base or begin a new task. This changes the state of the system in a way that needs to be expected by the user.

Instead of listing the ID followed by two percentages, the section uses colours and commonly used icons for battery and progress. The battery icon shows 5 levels of battery percentage as seen in Figure 14. This range of battery icons provides a clear and quick visual identifier for the battery level. It is easier for the user to interpret this data than a list of percentages. Again, this is a hierarchy of data. The specific battery percentage is also listed next to the icon but is less prominent. The colour and relative size of the icon attracts more attention than the percentage which defines the hierarchy. Users initially look at the icon and then read the percentage if they want more detailed information.

A similar approach is used for the task progress. Instead of just a percentage, a progress bar containing the percentage is displayed. This bar is a simple rectangle that fills with colour to a proportion matching the percentage of task completion. For example, at 45%, the bar is 45% full as seen in Figure 15. This has a similar effect to the battery icons with the hierarchy of data.

Below the scrollable list is a subsection to show details about a particular drone. This lower level information contains more detailed data about each drone and is accessible by clicking on any drone in

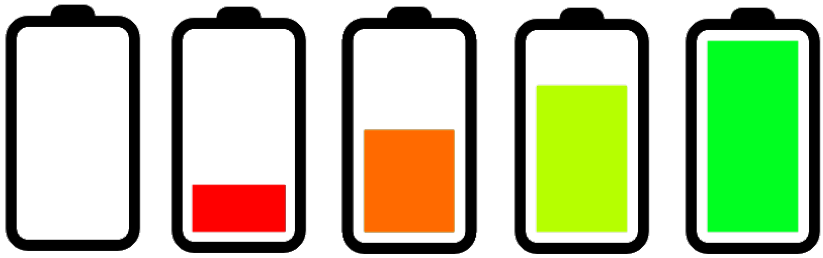


Fig. 14: Battery Icons

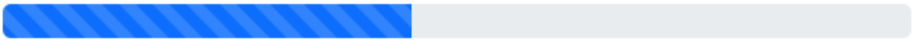


Fig. 15: Progress Bar

the list. This displays all of the information about the drone in one place. It details the ID, type, task name and description along with the progress and an estimate for the remaining time.

The type of drone is the the next level of information. The type is a link to another page which displays all of the drone information with a picture, its name and some statistics about it as shown in Figure 16. The last level of data is a link to the specification provided by the manufacturer. These are stored and served by the backend API.

Finally, an action button is included. Currently, there is only a button that prompts the drone to return to the base on command. This is spoken about in more details in the Action Buttons section.

The map section contains a map served by the Google Maps API as seen in Figure 17. On it, there are three main components: the boundary (blue), the flight path (red) and the drones. Each drone is a movable icon that is plotted according to the current location held in the database. Each drone displays its ID. When a drone has been selected in the drone section, the selected drone is highlighted on the map for clarity. Drone ”SUR079” is selected in Figure 13 and so the drone is highlighted in green on the map which is more clear in Figure 17.

2.6.4 Stage 2

Stage 2 is the secondary surveying stage during which collects higher definition images than in stage 1. The layout is very similar to that of stage 1. The differences can be seen on the map as the planned path is much more refined for each drone since this is a detailed survey at a lower altitude.

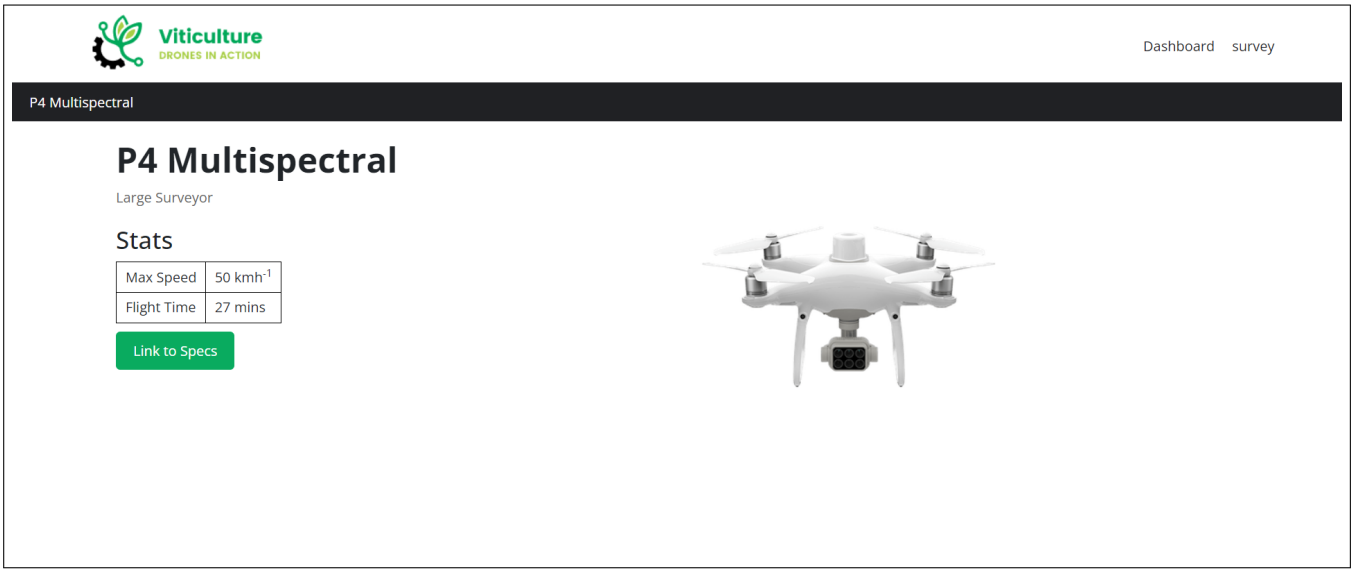


Fig. 16: Drone Type Page

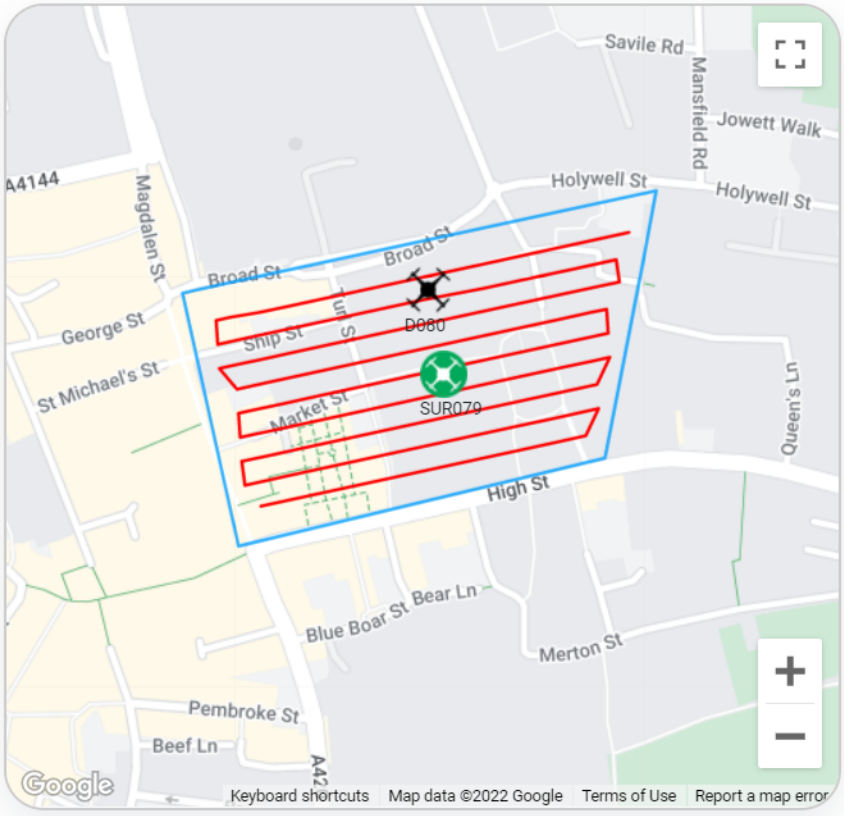


Fig. 17: Dynamic Map

The details of the previous stage are now available to be reviewed. The NDVI data has been collected and processed into a heatmap for the user to see. This image is overlayed to a static satellite image of the field. This data is used to identify what areas require a detailed scan ahead of the final spraying phase. There is a degree of uncertainty surrounding the classification of the plants. Areas that have a low confidence value need to be reviewed by the operator. This is a major role of the operator and requires them to have some knowledge of what a diseased plant looks like.

To confirm these areas, a new feature is implemented for the user input and disease verification. A modal window is used to verify the plants when the user has time to. A modal window is a graphical element subordinate to an application's main window that often has controls for the interface and requires action. This window can be viewed at the user's discretion so it is a non-blocking task. Non-blocking tasks allow the user to continue monitoring the system and complete other tasks. They do not stop other tasks from being completed. This task is, however, required to be completed to continue to stage 3.

Within the modal window in Figure 18, the specimen image is displayed clearly in the center. The set of confidence values for each disease are shown above the image to provide an indication of which disease is correct. There are images of the 3 diseases that the system actively looks for, as well as an option for no disease and unknown. Each option shows a single leaf with the disease visible. This makes choosing the disease much easier for the user as they can compare the known samples to the specimen. It also reduces the requirement for the user to have a good knowledge of diseases affecting vine leaves. The names of the diseases are also adjacent to the images. To choose the disease, the user clicks on the image. There are some cases where the user can input the disease via a text box when it is not available in the preset options. This disease will not be treated by the system but will be reported.

The progress bar shows how far along the user is in reviewing the set of images that do not pass the threshold for certainty. This information is not critical to the user but it is desirable for the user to know their progress. It increases their attentiveness as they know how many actions are remaining.

In the event that there are lots of images to be reviewed, this could become too cumbersome for the user to act on. Currently, this remains an issue that can be improved upon. A possible solution would be to group visually similar sections together and only present one candidate image to the user for confirmation. The result of the confirmation can be extrapolated to the surrounding area. Analysis into the trade-off of accuracy vs time would determine the degree to which this method is used.

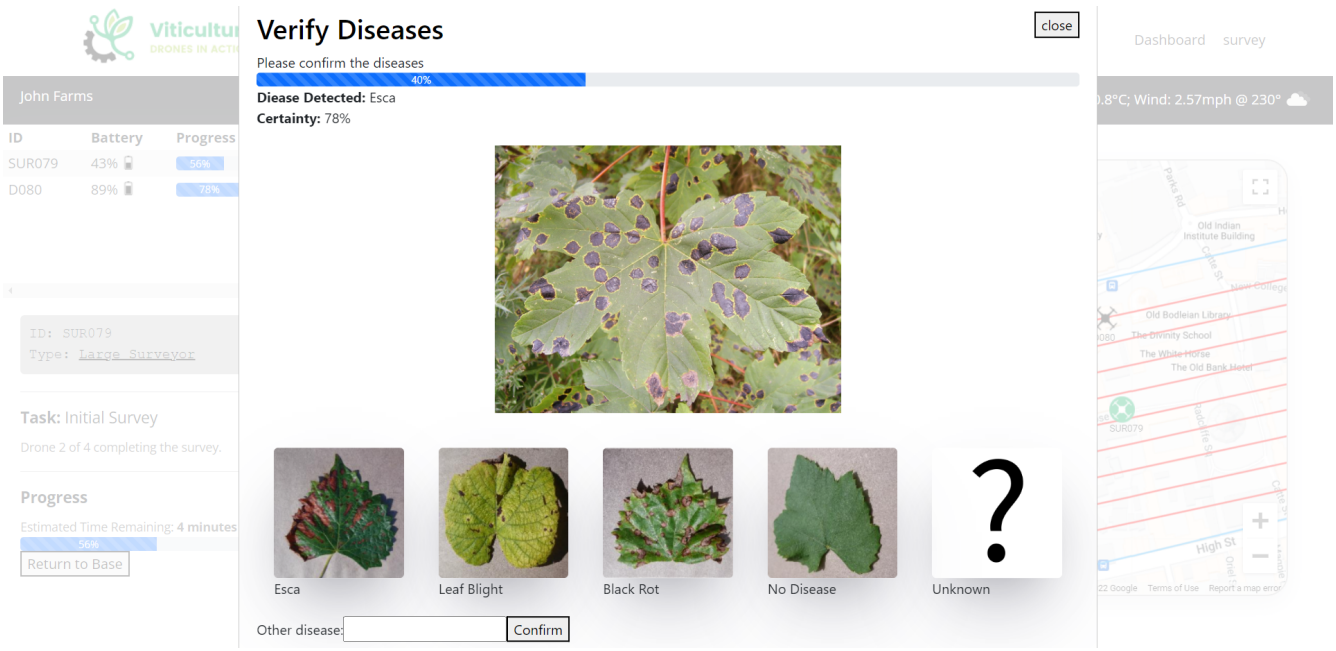


Fig. 18: Disease Confirmation Modal Window

The areas requiring a detailed scan are provided by the image processing stage which is discussed later. These areas are then sent to the path planning API which returns a set of paths for the drones.

2.6.5 Stage 3

Stage 3 is the action phase in which the spraying drone will spray the crops with the appropriate chemicals. During this stage, both the battery level and the amount of fluid in reservoir needs to be carefully monitored. To do this, any drones that are below a threshold for either battery or fluid levels are highlighted in the drone area of the dashboard to draw attention to the user. These drones are likely to return to base soon and will require either charging, more fluids or both. This is a manual operation that the user must be prepared for.

To make the data readable on the map, a set of glyphs are used. These icons contain subgraphs that tell the user information about that particular drone. The glyphs in figure 19 contains 3 subgraphs describing the battery levels, chemical levels and the orientation of the drone. Glyphs provide a lot of information in a small form factor. They can also proved a visual general overview of the system state.[14]

2.6.6 Generated Report

The generated report is sent to the client with a detailed summary of the data gathered and action taken. The report is generated when the

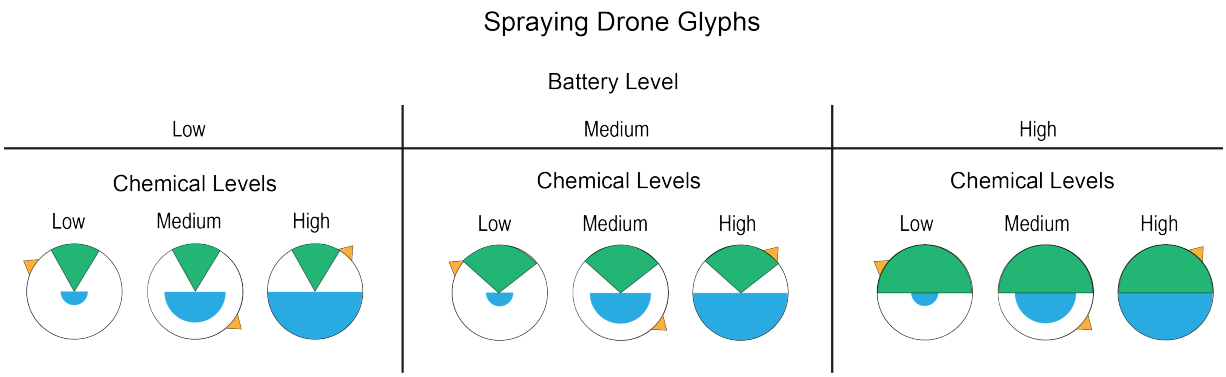


Fig. 19: Glyphs

`/api/projectReport/:project`

endpoint is visited by the client. The backend controller for this route is triggered which generates a PDF to send to the client as seen in Figure 21. The information is broken down into the 3 key stages.

Since stage 1 and 2 are both surveys, their data is merged to provide an overall summary of the state of the field. To summarise the NDVI data the report has a donut chart of the proportions of the field's health rating. The percentages of healthy, unhealthy and diseased plants make up the full summary of the field. A donut chart is an effective way to communicated this data along with colour. The whole donut makes up 100% of the field and each section is the proportion of the field that is classified as each category. Each section is a colour that is generally associated with that health as seen in Figure 20. Green is healthy, orange is unhealthy while red represents diseased crops. Visually, this representation quickly conveys the state of the field's health. Along side this is a statement describing the overall rated health of the field.

Next to the donut chart is a heatmap of the field. A satellite image of the field is retrieved from the Google Maps API onto which the heatmap is overlayed. A heatmap is a visualisation technique that represents the magnitude of an attribute on a 2D plane. This is effective in this case as it can be used to identify clusters of disease on the field. The NDVI readings are mapped onto the map using a colour range as seen in Figure 22.

The spraying phase shows boundaries of areas that have been sprayed with chemicals.

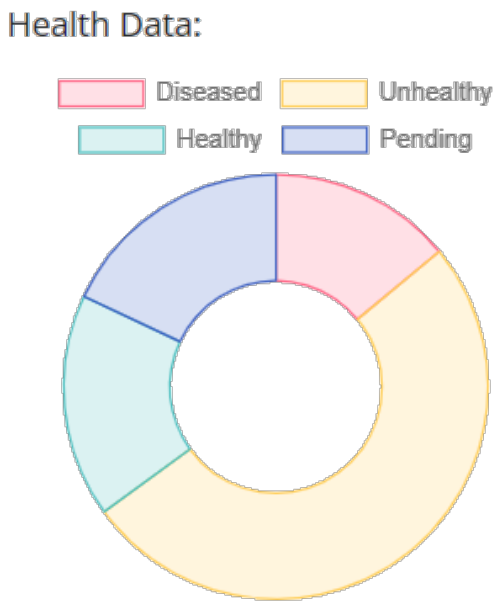


Fig. 20: Donut Chart for Field Health

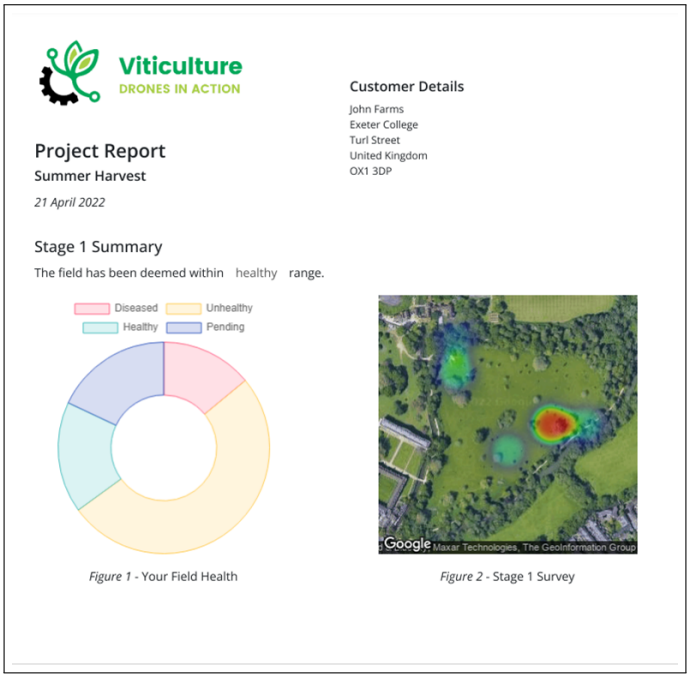


Fig. 21: Generated PDF Report

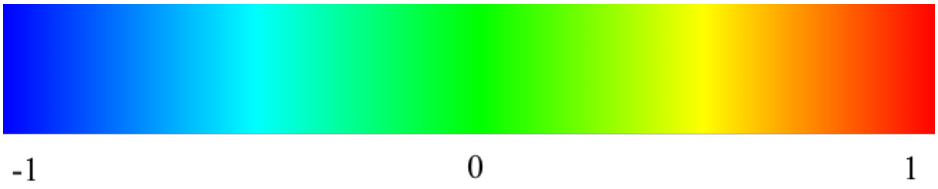


Fig. 22: Heatmap Gradient

2.6.7 Action Buttons

Each drone can have a set of actions associated with them. All drones have the option to return to base and land. Actions that only affect one drone can be found when the drone is selected in the drone menu. These alter the task that the drone is currently executing. When a drone is returning to the base, the drone will flash blue in the drone list along with an alert on the page to tell the user that one or more drones are returning. This allows the user to prepare for the incoming drones.

System wide actions are also available. If there is bad weather or the project needs to pause for any reason, all of the drones can be recalled using a system recall. This button will also include a confirmation popup since it will disrupt the project significantly. This gives the user the option to reconsider the decision and emphasises that recalling all of the drones will affect the whole system. It also mitigates errors in case the button was clicked by mistake.

2.7 Data Management

2.7.1 Path Planning

The path planning algorithm requires the boundary of the field to be send as a set of coordinates in a Cartesian coordinate system. Initially the boundary is defined as a set of GPS coordinates. These coordinates are in the geographic coordinate system (GCS) which is a spherical coordinate system.[15] Typically, the positions are communicated through two parameters: latitude (φ) and longitude (λ) as seen in Figure 23. In this system, the distance between two points cannot be calculated using Pythagoras since each degree of latitude or longitude varies in unit distance.

The coordinates must be converted from the GCS to a Cartesian coordinate system with unit lengths of 1 metre. The haversine formula [16] is used to calculate the distance between two points given their angle.

$$\mathsf{hav}(\theta) = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$d = 2R\arcsin\left(\mathsf{hav}(\theta)\right)$$

Using this, the distance of the deviance of $\Delta\varphi = \Delta\lambda = 10^{-5}$ from the center is calculated. The north/south distance (d_{lat}) is calculated by setting $\Delta\lambda = 0, \Delta\varphi = 10^{-5}$ and the east/west distance (d_{long})

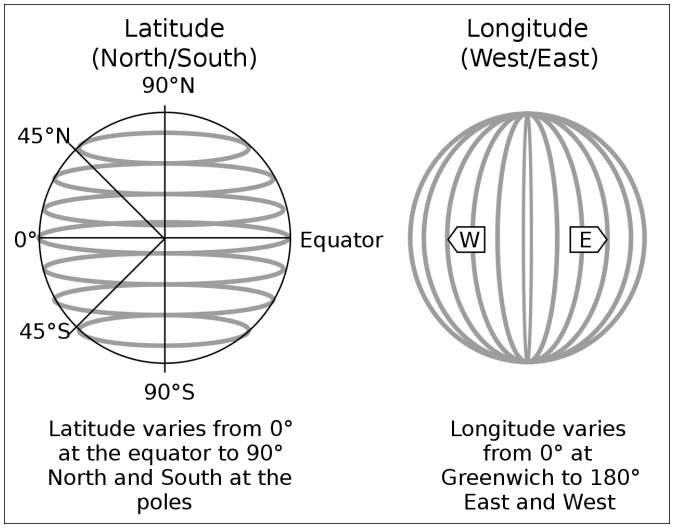


Fig. 23: Latitude and Longitude on Earth

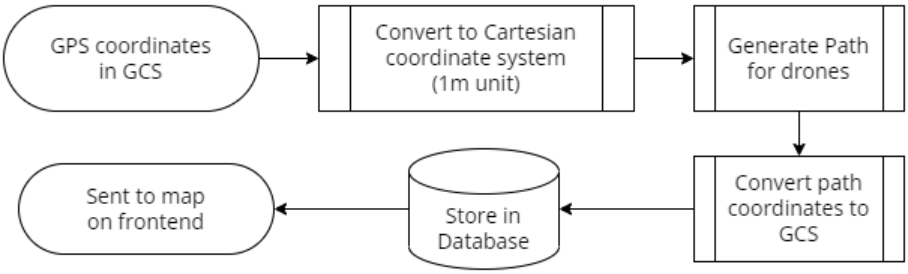


Fig. 24: Path Planning Data Flow

is calculated by setting $\Delta\varphi = 0, \Delta\lambda = 10^{-5}$. The GCS coordinates from the boundary are arbitrarily multiplied by 10,000 and their difference from the center is taken.

$$x_i = 10,000(\lambda_i - \lambda_c) \times d_{\text{long}}$$

$$y_i = 10,000(\varphi_i - \varphi_c) \times d_{\text{lat}}$$

This gives the $(\lambda_i, \varphi_i) \rightarrow (x_i, y_i)$ coordinates in the Cartesian coordinate system with a unit length of 1 metre. Here we are assuming that the unit distance in the GCS has a very small variance since the field is assumed to be relatively small and so the conversion distance for the centre is used for all points. An improvement could be that the exact conversion distance is calculated for each point.

Once the path planning algorithm has returned the path, the Cartesian coordinates are reverted back into the GCS so that they can be stored in the database. Once they are stored, they are accessible later be plotted on the map on the frontend or used by the control systems. This data flow is illustrated

in Figure 24.

The datatype that holds the boundary information is JSON which is shown below. The polygon is stored in the database using the GeoJSON standard format.[17] This is an open standard format used to represent geographical features. Here, the polygon datatype is used since it is typically used for boundaries of countries, provinces and areas of land. This requires the start and endpoint to be the same.

2.7.2 Image Data

Storing images is an expensive task, the P4 Multispectral drone used for surveys exports images with a compressed file size of approximately 2MB. A 1km^2 area has a path length of approximately 3500metres. This means that 7GB of images are stored for each km^2 of area in the initial survey. It is not effective to store these images in the database and so instead of "blob" fields, the database stores the URLs of the images in the local file system. This allows agents to access the images directly rather than creating large load on the database. Reading the URLs is a very fast process meaning that the agent can save the URLs and request the images when it requires them.

2.7.3 Database Architecture

The relational database is made up of 6 tables as seen in Figure 25. The largest table is the project table which includes all of the data about each project. A project is defined by a set of surveys and an action phase on one field of one client. Throughout the project, the statistics and information are entered by the backend as they become available. There are multiple relationships throughout the database.

The drones table has 4 relationships with it. Firstly, there is a many-to-one relationship with the projects table. This means that the projects table can have multiple drones associated with it at a time. This is because many drones can be allocated to the project via the 'drones.allocation' field. When a drone is not currently allocated, the field has a value of 'NULL'. This allows the allocation phase to search for unallocated drones via the API with a query for all drones where allocation is NULL.

The drones table also has a table of locations associated with it. Each drone can have multiple locations. The location is not held in the drones table so that a history of locations is created. When the database is queried for a drone's location, it returns the location where the droneId matches the parameter of the query. The location that was most recently created is used as this is the most up-to-date location

that the database has.

A similar reasoning is used for having a separate tasks table. Each drone has multiple tasks however only one can be active at a time which is defined in the 'tasks.isActive' Boolean field. A history of tasks is preserved for later analysis if necessary.

The final table associated with the drones table is the drone types table. This contains information about the type of drone that the individual drone is. For example, there are multiple P4 Multispectral surveying drones. It would be inefficient to save all of the specifications in the drones table as it would be repeated for each drone of the same type.

The project table has 2 associations, one of which has been discussed with the drone table. The other association is with the clients table. This table is used for business reasons. Clients that have multiple projects will be able to view all of the projects associated with their account. The name, address and email is used for billing and contact information. The address is also used for the team to travel to the correct location to carry out the project. The client coordinates (latitude and longitude) are used during the setup phase to display the correct location on the dynamic map for boundary definition.

All of the information held in the database can be queried by the backend using the Sequelize ORM. [10] Table joins all multiple tables of information to be included. For example, users can query a specific drone ID and include the type information along with it. This is returned in a simple JSON format that can be read by all users.

2.8 Results of HRI

The implemented applications effectively convey the processed data to the user. It also provides controls for the user to operate the system. Throughout the application, the pages are designed to contain the optimal amount of data to prevent the user being overwhelmed. This reduces their cognitive load and means that they are more attentive the pertinent information. The Node.js and Express backend API communicates with the database and manages the data in a way that allows users to easily and securely access and manipulate the data that they request.

Furthermore, the HRI successfully acts as a centralised component for the system as a whole by facilitating data communication, system monitoring and user control.

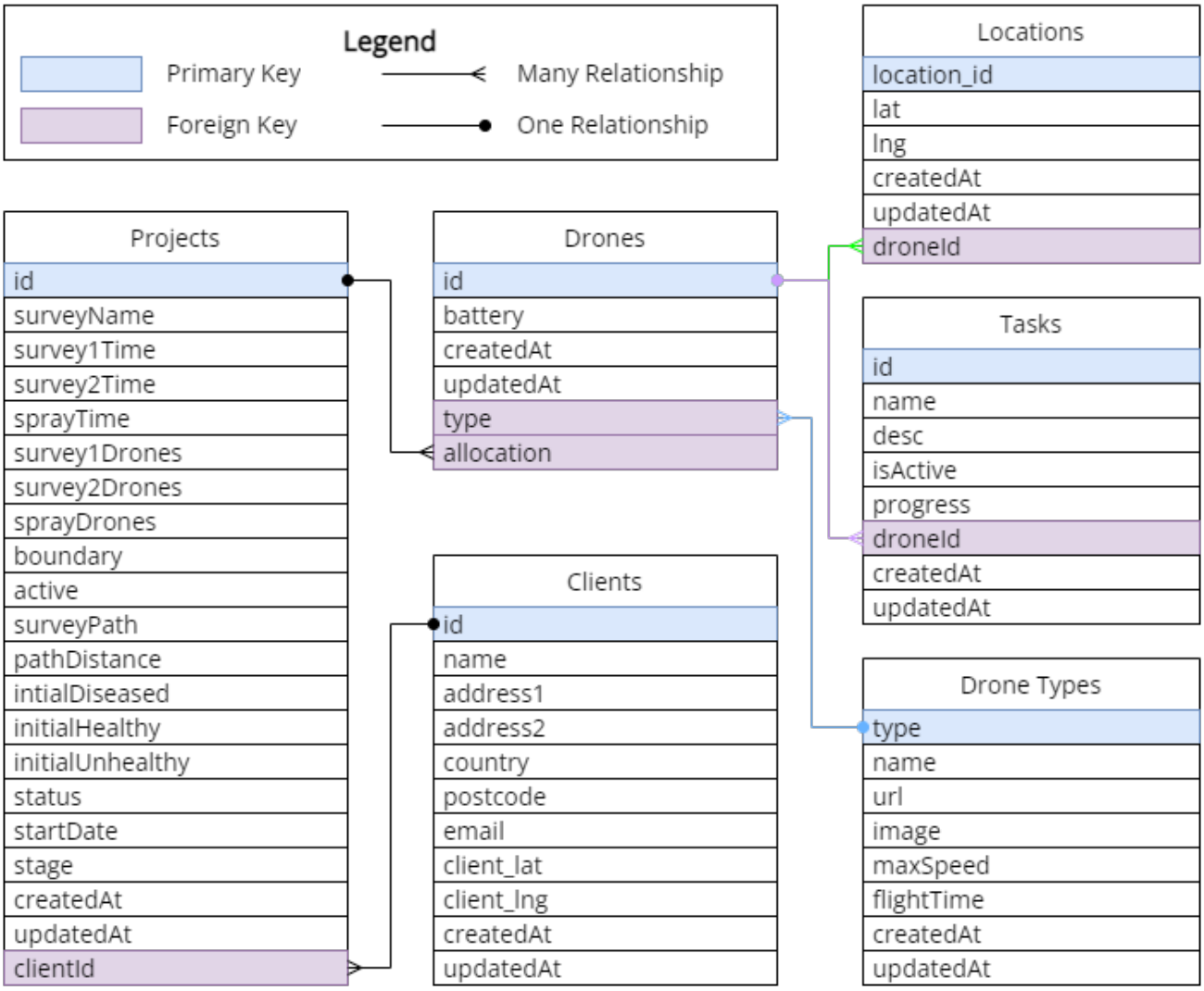


Fig. 25: Database Architecture Diagram

3 Image Analysis

Image analysis is the process of producing useful information from images, using a range of processing techniques. As described in section 1.1, image analysis is used during multiple stages of the project pipeline. This chapter looks to discuss the implementation of the three main image analysis techniques used during precision viticulture; image stitching to provide a higher resolution overview, high level vegetation index analysis to generate a heatmap of diseased vines, and image multi-class classification to determine which disease is present, using transfer machine learning.

3.1 Image Stitching

3.1.1 Rationale

To be able to determine an accurate disease heatmap using a vegetation index, a certain level of resolution is required. Publicly available satellite NDVI filtered data from the Global Agriculture Monitoring (GLAM) project has resolution in the region of 250m/pixel [18]. As the project has planned to survey and treat a 0.25km^2 vineyard, this would only produce 4 pixels for the entire vineyard, which is an unacceptable resolution for useful data to be extracted.

For publicly available satellite RGB imagery, the resolution is close to 15cm/pixel. As grapevines are placed in a regular pattern at an average of 2.5m apart in a row, images show rows of plants clearly, however this is still unacceptable for the disease heatmap, as grapevine leaf sizes range from 5 to 25 cm in width, and as leaves are where most diseases present themselves, this resolution would result in at most one pixel per leaf.

The proposed solution to this is to stitch together a panorama of images to provide a map of the vineyard at a higher resolution. The requirement to determine a sufficient resolution disease heatmap is to have at most a 2cm/pixel resolution, as to ensure even the smallest leaves have enough pixels to be analysed.

The DJI 'P4 Multispectral' camera has a field of view of 62.7° and has a maximum resolution of 1300 pixels by 1600 pixels. [19]

Using trigonometry on figure 26, the constant height requirement of stage 1 can be determined, and

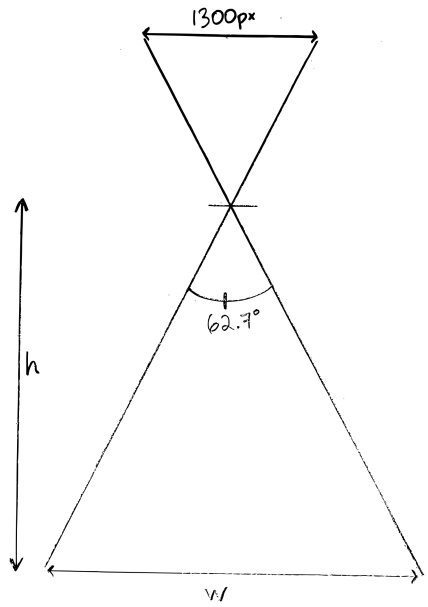


Fig. 26: Rationale to determine Stage 1 surveying height.

implemented through the controller.

$$\text{resolution} \leq 2\text{cm/pixel} \tag{1}$$

$$\frac{w}{1300\text{px}} \leq 2\text{cm/pixel} \tag{2}$$

$$w \leq 26\text{m} \tag{3}$$

$$\tan(31.35°) = \frac{h}{0.5w} \tag{4}$$

$$h \leq 7.9\text{m} \tag{5}$$

Grapevines can grow up to 3m tall, hence for stage 1, the height requirement must be between 3m and 7.9m. To allow for weather conditions or other disturbances leading to an error in flying height, it is sensible to permit a $\pm 10\%$ error when aiming to fly at a constant height of 5.5m for the initial field survey.

3.1.2 Method

Multiple methods of image stitching have been used and implemented. In this section, the decision between the two primary options, temporal stitching, or feature-based stitching, is explored. Temporal stitching would be possible for stitching images if the drone were to fly at a constant height, ensuring that the resolution of 2cm/pixel was constant, as explained in section 1.3. The drone would also need to fly at a constant velocity and take images at a constant frequency.

The combination of a constant speed, resolution and image frequency means that there would be a constant overlap between images; hence, a simple affine transformation could be applied to each subsequent image, to make a larger stitched image. The equation for the constant transformation is as follows,

$$\frac{v}{n(r \times 100)} = T \tag{6}$$

where v is the velocity in m/s, n is the frequency of images captured, in frames per second (fps), r is the resolution, in cm/pixel, and T is the temporal translation required, in pixels/frame.

Theoretically this method would be sound; however, in reality this method is far too susceptible to perturbations and would realistically need many manual edits to the overall stitched map to remove erroneous overlaps from changes in heights, or gusts of wind moving the drone away from the pre-planned path.

Rather than using temporal stitching, a more accurate method to create the high-level overview is to use an automated feature-based stitching algorithm, coded in MATLAB [20], as shown in the flowchart in figure 27. By loading images sequentially, which is possible thanks to the API seen in figure 1, finding matching features between images n and $n-1$ means that probabilistic image matching models can be ignored for this project.

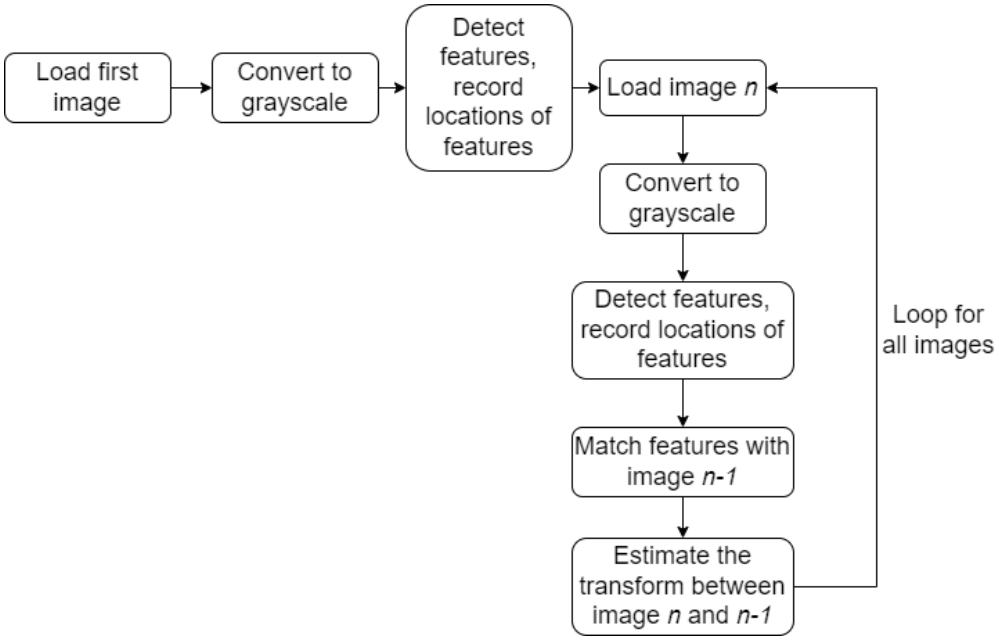


Fig. 27: A flow chart describing the image stitching algorithm.

3.1.3 Feature detection

To extract, describe and match features, SURF (Speeded Up Robust Features) is the algorithm used, published by Bay, Tuytelaars and Van Gool in 2006. [21] SURF was developed as an improvement on the SIFT (Scale Invariant Feature Transform) algorithm, published by Lowe in 2004. [22]

The SIFT algorithm uses key-points determined by maxima and minima of the difference of successively blurred images convolved with differently scaled gaussian filters.

$$D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma) \tag{7}$$

Equation 7, $D(x, y, \sigma)$ refers to the difference of gaussian image, and $L(x, y, k_n\sigma)$ is the same original image $I(x, y)$ convolved each with a different gaussian blur of scale $k_n\sigma$, that is to say,

$$L(x, y, k_i\sigma) = G(x, y, k_i\sigma) * I(x, y) \tag{8}$$

In the SURF algorithm, small approximations are used to speed up the process. SURF has also been claimed to be more robust to different image transformations. SURF uses repeated application of box filters as an acceptable approximation to convolving the image with scaled gaussian filters. This can be shown by implementing the central limit theorem. [23]

The box filter is a blurring filter, equated by taking the average value of neighbouring pixels from the input image. Box filters of differing sizes are calculated as in equation 9.

$$\text{box} = \frac{1}{n}A = \frac{1}{n}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \tag{9}$$

where $A \in \mathbb{R}^{n\times n}$.

Integral images are used by the SURF algorithm, to speed up computation of the convolution of box filters with the original image. [24] Integral images are calculated by using the following equation,

$$I(x, y) = \sum_{x'=0}^{x}\sum_{y'=0}^{y} i(x', y') \tag{10}$$

where $I(x, y)$ is the integral image value, and $i(x, y)$ is the value of the pixel at (x, y) in the original image.

Once the integral image has been equated for the whole image, the sum of intensities over any

rectangular area only requires four additions, independent of the area size. Hence for box filters, which equate an average from a square area of pixels, by using the integral image evaluated at the four corners, the process is substantially sped up.

$$\sum_{x=x_0}^{x_1}\sum_{y=y_0}^{y_1} i(x,y) = I(x_1,y_1) + I(x_0,y_0) - I(x_1,y_0) - I(x_0,y_1) \tag{11}$$

The determinant of the Hessian matrix is used to determine which points of interest are detected by the SURF algorithm, by choosing the values where the determinant of the Hessian is maximal, as well as to determine the scaling. Hessian matrices are calculated using the following formula,

$$H(x,y,\sigma) = \begin{bmatrix} L_{xx}(x,y,\sigma) & L_{xy}(x,y,\sigma) \\ L_{yx}(x,y,\sigma) & L_{yy}(x,y,\sigma) \end{bmatrix} \tag{12}$$

where $H(x,y,\sigma)$ is the Hessian matrix, and $L_{xx}(x,y,\sigma)$ and so on, is the convolution of the second-order derivative (determined by the subscript) of a Gaussian with scaling σ, with the image $I(x,y)$.

Once key-points have been extracted, the next step of SURF is to describe the key-point uniquely, such that it can be matched with the same feature in another image. This is done by calculating the distribution of Haar wavelet responses in a surrounding neighbourhood. The neighbourhood is circular to determine the orientation of each key-point as to ensure rotational invariance, and the neighbourhood is redefined by a larger square region to determine the description of each key-point.

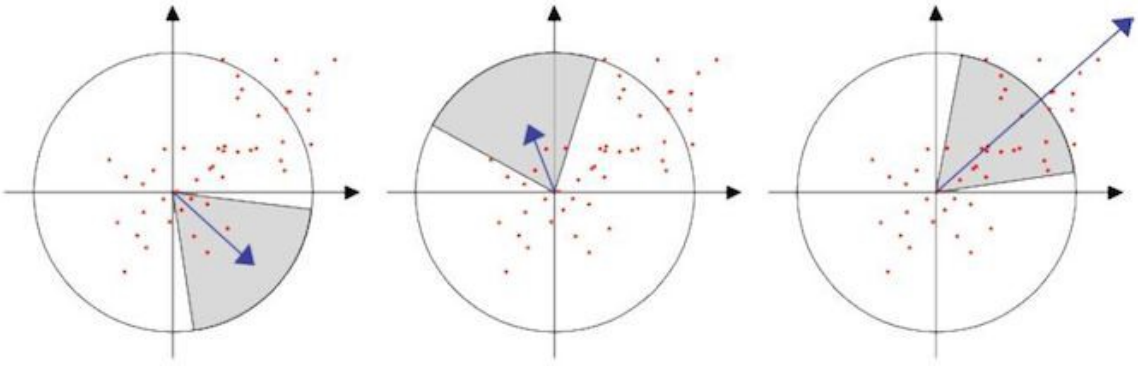


Fig. 28: Determining the orientation of key-points in SURF. [25]

Finally, by comparing the descriptors of key-points from different images, matching pairs can be found. From this stage, an estimate of the 2-D geometric transform between the images can be determined from the matching set of key-points, or ”inliers”.

As explained in ”Automatic panoramic image stitching using invariant features” (Brown and Lowe, 2007) [26], random sample consensus, or ”RANSAC” [27] is used to determine a set of randomly sampled key-points to estimate an image transform. The RANSAC algorithm is repeated for a finite number of trials, and the maximum size inlier set of overlapping key-points is chosen, as long as the homography of the corresponding key-points are within a maximum threshold distance.

This is not strictly an optimal solution; however, Brown and Lowe explain that after a large number of trials of RANSAC, e.g. $n = 500$, with an inlier probability of $p_i = 0.5$, the probability of finding the correct transformation or homography is very high, with failure probability being in the range 10^{-14}. The aforementioned paper then uses probabilistic image matching, which is unnecessary for this project due to the ability to sequentially call images from the API.

Using the MATLAB Computer Vision Toolbox, [28], an algorithm was coded to carry this out, with examples shown in figures 29 and 30.

Fig. 29: Original RGB split image of vineyard.

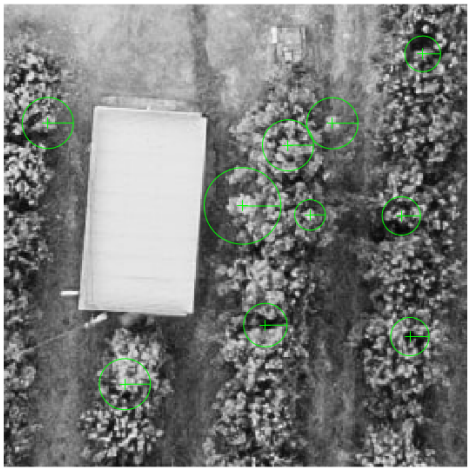


Fig. 30: Grayscale split image of vineyard, with SURF features defined with green circles, with radii proportional to importance of key-point.

3.1.4 Testing Code

To test the image stitcher in a realistic scenario, a set of aerial drone images of a vineyard would be required; however, finding this in the public domain was difficult. Instead, the image stitcher was tested in two stages. Firstly, a high resolution image of a vineyard in Faro, Portugal was attained, [29] and artificially split with a predetermined level of overlap. This is to test that the image stitcher can work with periodic, repeated rows, and not incorrectly match two similar key-points.

Fig. 31: An aerial RGB image of a vineyard in Faro, Portugal. [29]

The vineyard image was split using an image splitter, which was coded in Python, with reference to Devyanshu Shukla's image splitter on GitHub. [30]. The second test of the image stitching algorithm came by using frames of a video. This video was filmed using a drone fitted with multi-spectral filtered cameras, and RGB cameras, of a forest, to provide the image stitcher with a more realistic environment, with noise and perturbations.

Fig. 32: 3 examples of split images from the larger aerial vineyard image.

The original vineyard image had a resolution of 2242 pixels by 3992 pixels, shown in figure 31. This is a far higher resolution than what is available on the DJI P4 Multispectral.[19] This image was split into 119 images of a suitable resolution of 1280 pixels by 720 pixels, which is a valid resolution for the drone used in our project. These 119 images all had an overlap of 80% or more for edge cases, to ensure a suitable inlier set of SURF features could be found, to map a sufficiently accurate 2D transformation.

The image stitcher performed very well from the artificially split image of a vineyard. This test shows

Fig. 33: Result of stitching the split 119 images of Figure 31, using MATLAB script.

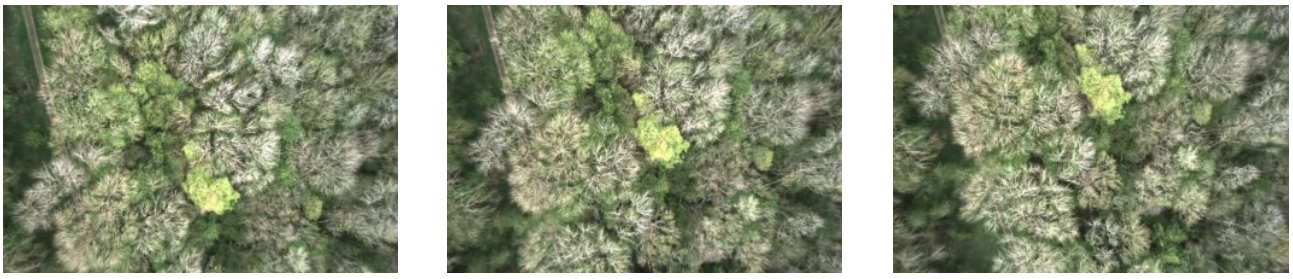


Fig. 34: 3 examples frames of multi-spectral camera drone flying above a forest

that the SURF features descriptions are suitable to be used in a vineyard, and that similar repeated rows are able to be described separately. There are small horizontal imperfections where the stitching took place, but they are minimal, and unimportant once the image is blurred in section 3.2.1.

The second test of the image stitcher is to create a map from stills taken every 0.75s from a video of a multi-spectral camera drone flying above a forest. Examples of these frames are shown in figure 34. Using frames from a video introduces noise and distortion to the edges of images, due to barrel distortion. This application will therefore be more similar to real life circumstances, even with a different type of vegetation being stitched together. [31]

The output is shown in figure 35. The image stitcher performed slightly worse in the real-world scenario, as can be seen by the imperfections along the path on the left side of the stitched image;

Fig. 35: Result of stitching the 8 frames of video using MATLAB algorithm. [31]

however, for our requirements of obtaining a heatmap of vegetation indices, this is still suitable.

One potential improvement to the stitcher is to use the inbuilt Camera Calibrator in MATLAB to attempt to remove barrel distortion. As this paper-based project does not have access to the equipment used to film the video where the stills in figure 34 were taken, the camera parameters are not available to provide a correction process to the images.

By storing the rigid 2D transformations of the RGB images as a separate variable in the MATLAB code, the transforms can be used again to stitch the images taken from the different multispectral cameras present on the DJI P4 Multispectral, discussed in section 3.2.1 such that vegetation indices can be successfully derived, as well as an RGB map for future use. As the other multispectral cameras are placed very close together on the gimble, seen in image 36, it is sensible to assume that the geometric transforms for all cameras are the same as the RGB camera transforms.

Fig. 36: Multispectral camera placement on the DJI P4 Multispectral drone.[32]

3.2 Vegetation Indices

3.2.1 Rationale

A vegetation index is a frequency transformation of two or more multispectral image bands to provide data climate trends [33], soil water content [34], and to determine the health of a plant. [35] In this section, using the prior stitched multi-spectral images generated during section 3.1, analysis is carried out to determine potentially diseased areas of the field.

A multispectral image is taken with a sensor that is sensitive to a different range of wavelengths, with the DJI P4 Multispectral drone having 5 sensors, and one RGB camera.

Sensor	Wavelength range
Red Edge (RE)	730 nm ± 16 nm
Near-Infrared (NIR)	840 nm ± 26 nm
Green (G)	560 nm ± 16 nm
Visible Light (RGB)	450 nm ± 150 nm
Red (R)	650 nm ± 16 nm
Blue (B)	450 nm ± 16 nm

Table 1: Sensors and wavelength ranges present on the DJI P4 Multispectral drone camera array. [19]

There are at least 97 different vegetation indices that have been used on vines, to determine different applications, including water stress and irrigation, yield comparison and disease detection. [36]

3.2.2 Normalised Difference Vegetation Index (NDVI)

The most commonly used vegetation index is the Normalised Difference Vegetation Index (NDVI). Measuring Vegetation (Weier and Herring, 2000)[37] explains that NDVI is a dimensionless index which describes the difference between visible and near-infrared reflectance of vegetation cover and can be used to estimate the density of greenery on an area of land. In this paper, NDVI was used to determine the vegetation cover over a 20 year period to determine which areas were under irrigation stress due to lack of water, and yearly trends. Since then, NDVI has had many other uses, including environmental change, [38] and high-level disease detection, [39] which will be explored in this section.

To calculate NDVI:

$$NDVI = \frac{(NIR - Red)}{(NIR + Red)} \tag{13}$$

The reflectances labelled “Red” and “NIR” are ratios of the reflected divided by the radiation for red visible light wavelengths and near-infrared wavelengths respectively, detected by the sensors shown in figure

36. As the reflectances are equated as ratios, these can only take values between 0 and 1, hence from formula 13, NDVI has values ranging from -1 to 1. This can be used to detect healthy and unhealthy vegetation, with the range -1 to 0 being dead plants, or non-vegetative area, 0 to 0.33 being an unhealthy plant, 0.33 to 0.67 being moderately healthy, and 0.67 to 1 being healthy vegetation.

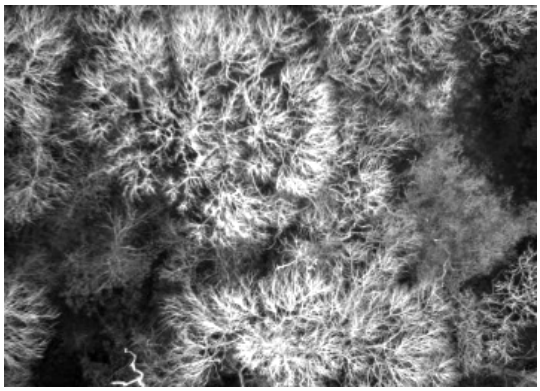


Fig. 37: Red filtered Image.

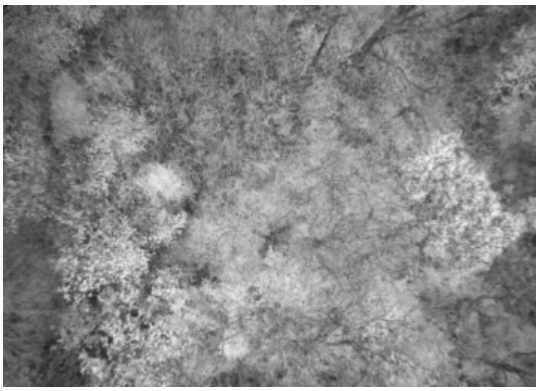


Fig. 38: Near Infrared filtered Image.

Multi-spectral stills were gathered of the same forest as discussed in section 3.1.4. A MATLAB script was coded to equate the NDVI value from the stills shown in figures 37 and 38. A colour-bar was used to display the NDVI visually, which can be sent to HRI to be shown to the customer, with green areas being healthiest and red areas showing unhealthy or inanimate areas. The result is shown in figure 39.

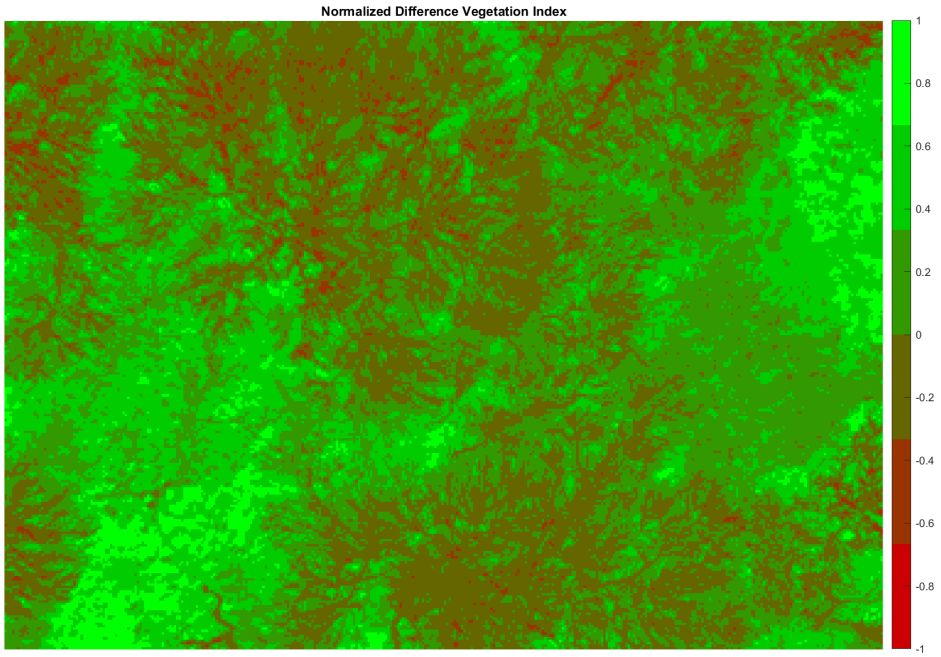


Fig. 39: NDVI equated from images 37 and 38.

However, for stage 2, a binary classification is required to determine which areas are potentially

diseased and require closer inspection and more detailed images.

To do this, a threshold is applied. If the value of $NDVI \geq 0.3$, then the pixel is determined to be "healthy", and will not be included in the area required for closer images in stage 2. If $NDVI < 0.3$, the pixel is classified as "unhealthy", and must have closer images taken to determine if the leaves are diseased, and if so, which disease is present. This threshold can be altered depending on the accuracy of previous scans. For example if the value $NDVI < 0.3$ determines too many healthy areas, then the value can be increased for future scans.

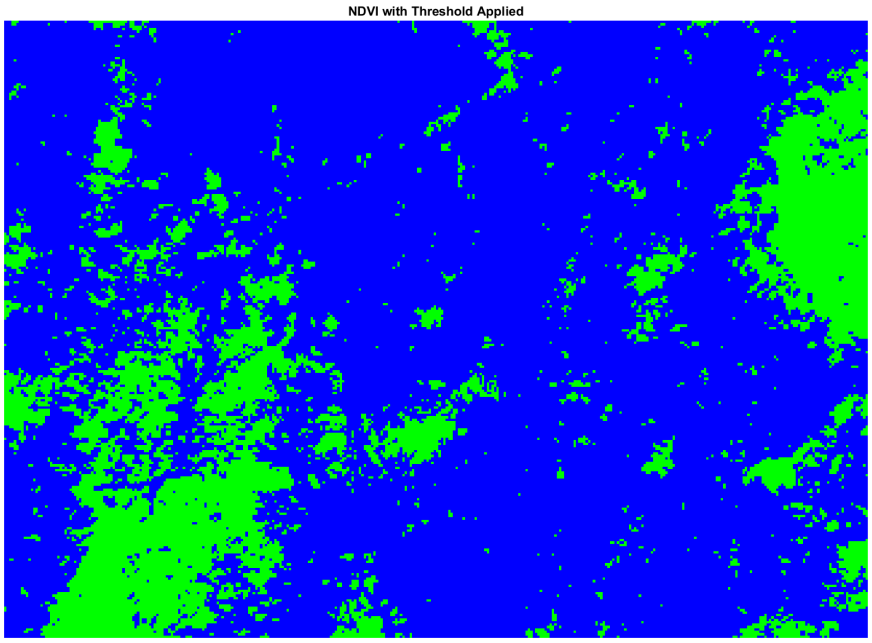


Fig. 40: NDVI from figure 39 with threshold applied.

The green areas are denoted as "healthy", and the blue areas need to be re-evaluated. Figure 40 is not a suitable heatmap to send to path planning, as discussed in section 3.1.1, each pixel is at most 2cm/pixel, hence the smallest area selected by the current threshold is $4cm^2$, too small to require more detailed images in stage 2. To appropriately deal with the pixelation, a smoothing filter can be applied. As explained in section 3.1.3, one can approximate multiple iterations of a box filter blur to be equivalent to that of a Gaussian filter. By producing a box filter of $n \times n$ window size, shown in equation 14, and convolving the original image with the box filter twice, a much smoother approximation of the NDVI image is produced.

$$\text{box} = \frac{1}{n}A = \frac{1}{n}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \tag{14}$$

where $A \in \mathbb{R}^{n \times n}$.

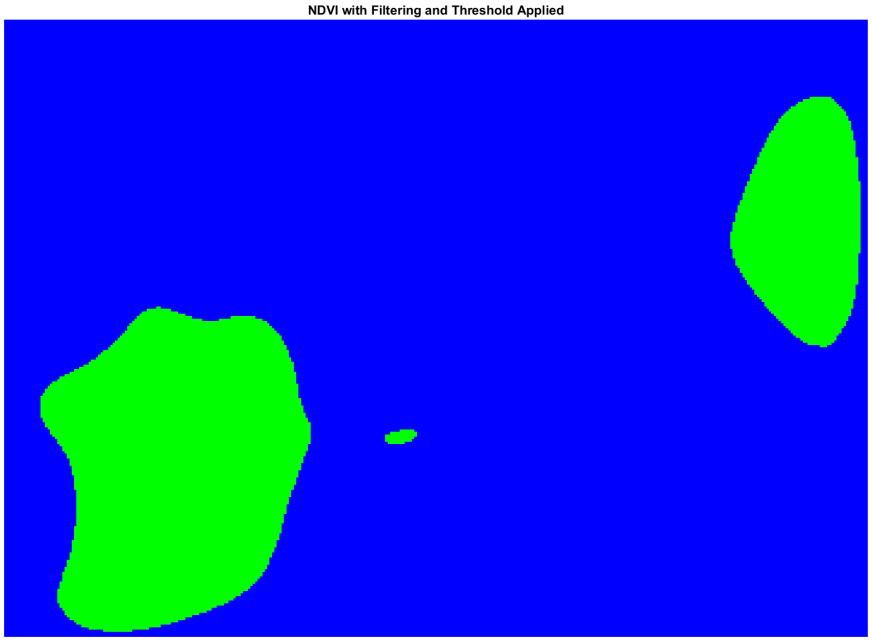


Fig. 41: NDVI threshold from figure 40 with multiple box filters applied to smooth the image.

3.2.3 ExGR

Although NDVI is the most widely used vegetation index, due to its vast range of uses, and simple implementation from just two sensors, this does not mean it will be the most suited to the task.

Kerkech et al.(2018) [35] produced a convolutional neural network to detect the presence of Esca based on an aerial view of a vineyard. The most accurate network, which finished with a 95.8% accuracy, was based around three vegetation indices; Excess Red (ExR), Excess Green (ExG), and the difference between them, ExGR. The equations for these indices are as follows,

$$ExG = \frac{(2G - R - B)}{(R + G + B)} \tag{15}$$

$$ExR = \frac{(1.4R - G)}{(R + G + B)} \tag{16}$$

$$ExGR = ExG - ExR \tag{17}$$

$$= \frac{(3G - 2.4R - B)}{(R + G + B)} \tag{18}$$

where R, G, and B correspond to the intensity of the Red, Green and Blue bands of an RGB image respectively. Equations 15, 16 and 17 were used with varying filter sizes to determine an optimal combination of vegetation indices and filter size.

Due to the nature of the reflectance variables in these vegetation indices, with equations 15, 16 and 17 only requiring red, green, and blue sensors, the bands can be sampled directly from an RGB image. Rather than using the forest image as seen in figure 39, we can instead use an aerial vineyard image to determine what a heatmap would look like for repeated rows of vines.

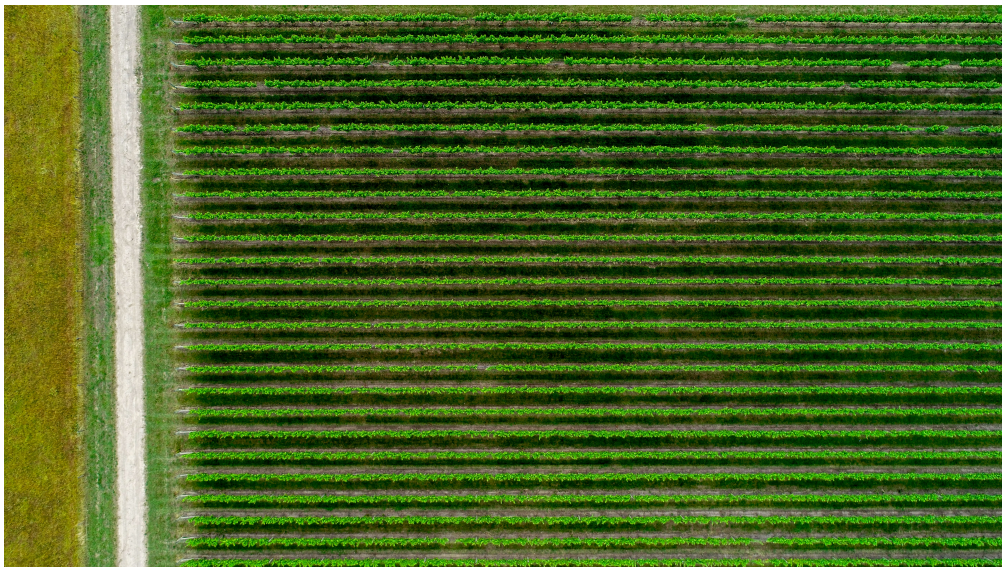


Fig. 42: Vineyard image by Alex Wigan, [40]

Due to a lack of sufficient data for a vineyard with knowledge of diseased vines, the approach of applying machine learning to determine diseased vines at this stage, as was shown in Kerkech et al. (2018) is not plausible; however, this should not turn out to be a major issue. Determining general health is all that is necessary in this section, with disease classification being determined after the second stage of our process; the close-up imaging section, where a classifier will determine accurately whether the plant is diseased, and if so which disease the plant has. Hence, using the method as discussed in subsection 3.2.2 can be used. The output for figure 42, after being convolved twice with a box filter and a threshold is shown in figure 43, where the green area is classified as healthy, and the blue area must be further inspected with closer images for disease classification.

Similarly to the NDVI analysis, the threshold can be altered depending on accuracy of previous scans. For this analysis, the threshold decision boundary value was $ExGR > 0.7$

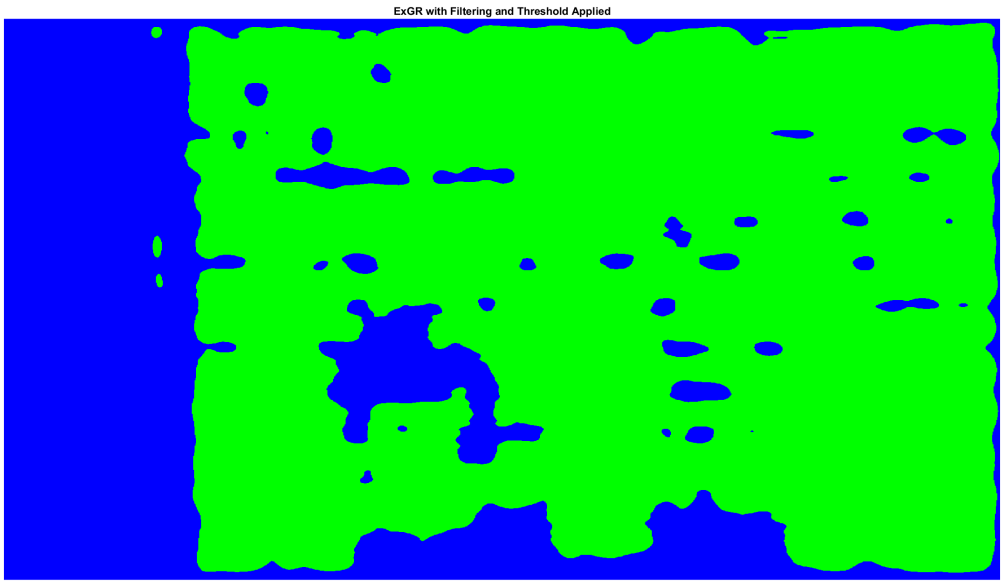


Fig. 43: Vineyard image from figure 42, with repeated box filter and threshold applied

3.3 Disease Classification

3.3.1 Rationale

For precision viticulture to treat diseases, one of the most crucial parts of the project pipeflow is to determine and classify the disease type. Different diseases have different responses for our spraying drone, and so an accurate classifier is crucial to determine which vines require treatment with which chemicals. Currently, pesticides and fungicides are sprayed over each vine, leading to more chemical usage than is required. Not only is the chemical toxic to wildlife and birds, but verification that every leaf in the vine canopy has sufficient coverage of the fungicide takes up days of person-hours. By classifying diseased and healthy plants, more efficient steps are taken to protect only the vines that are in urgent need, and organic farming can be carried out to promote further ecological balance.

Identifying which disease is present on a grape leaf is a classification problem, based on supervised learning. There are multiple ways to tackle supervised learning problems, the two main methods are training a convolutional network from scratch, and transfer learning.

Training a convolutional neural network from scratch is a time-consuming method. This method often uses gradient descent to minimise a certain loss function, and starts with no prior knowledge. Training from scratch also requires a large dataset to ensure a high accuracy and is often well suited to very specific functionality such as generating musical pieces, as there is not a lot of pre-trained models.

However, transfer learning is a faster method of producing a neural network, as it uses a pre-trained network and replaces the final layers such that the network can be trained to produce accurate results based around a different task. For the requirement, image based multi-class classification, a variety of pre-trained networks exist. This therefore, removes the requirements of training a network on a large dataset, by re-purposing neural networks that have already been trained on a large dataset. Transfer learning is the method that will be used to code our multi-class classification network.

To classify diseases, a dataset is required to train, validate, and test neural networks to determine which type of disease is present on the vines. A dataset of 4062 images was obtained from the PlantVillage dataset, [41] [42] which has images classified into 4 categories, leaves with the Esca disease, leaves with the Black Rot disease, leaves with Isariopsis Leaf Spot or leaf blight, and also healthy grape leaves.

3.3.2 Diseases

Esca, also known as “Black Measles” is a fungal disease which impacts mature grapevines. In France, an estimate of 12% of the vineyards are non-productive due to Esca, resulting in a cost of around EUR 1 billion for the country as a whole. [43]

Esca is a disease that is spread by two species of fungi releasing spores that travel by wind and rain splash, which can enter vines through pruning wounds or removal of side shoots. This disease is untreatable; however, by identifying this disease in the vineyard, the prevention of further infection of surrounding vines would be drastically improved with knowledge of local disease – by removing the diseased vine, encouraging smaller pruning cuts for vines surrounding the diseased vine and using sealants for pruning wounds to avoid further infection of local vines. [44]

Black Rot is another fungal disease that infects vines, which causes grapes, often with no warning signs on the fruit itself, to shrivel up into a hard black sphere, rendering them worthless. Spores also spread this disease from vine to vine. Depending on how early the black rot is spotted, fungicides may be successful in stopping the disease from overrunning the diseased plant. This is an option for the spray drone; however, preventative treatment of the neighbourhood of surrounding vines is more important to ensure that the disease does not spread from spores, by using a fungicide such as Ziram (Zinc dimethyldithiocarbamate). This chemical compound is dangerous, causing fatal injuries if inhaled according to the European Regulation (EC), and has been found to be highly toxic to birds. According to the US Environmental Protection Agency, Ziram can be applied up to 7 times for maximum effectiveness,

with a 7 day minimum application interval period for grapes. The maximum rate is 3 pounds of active ingredient per acres(lbs a.i./A), and hence over one crop cycle, up to 21lbs a.i./A can be used on grapes. [45]

On a vineyard of 50 acres, this equates to 476kg of active ingredient per crop cycle over the entire vineyard. By pinpointing locations of diseased vines, treatment can be localised, reducing the impact on the ecosystem. The treatment will be carried out by drones, reducing the risk to human health.

Isariopsis Leaf Spot, or ”Leaf Blight”, is another fungal disease which is less threatening than the other diseases, rarely resulting in severe yield losses. To manage the spread of this disease, fungicides like Ziram are used as a preventative measure.

Fig. 44: Example of a healthy grapevine leaf.

Fig. 45: Example of a grapevine leaf with Esca.

Fig. 46: Example of a grapevine leaf with Black Rot disease.

Fig. 47: Example of a grapevine leaf with Isariopsis Leaf Spot.

As explained in table 2, classes with more images were reduced to match the size of the smallest dataset, as to ensure that classes were not biased. Examples of these can be seen in figures 44, 45, 46

Name	Total Images	Used Images	Train	Validate	Test
Black Rot	1180	423	342	38	43
Esca	1383	423	342	38	43
Healthy	423	423	342	38	43
Blight	1076	423	342	38	43

Table 2: Splitting of the dataset [41] to use in neural networks.

and 47.

This was initially carried out by removing 380 of images from each disease class at random, using a MATLAB script. Firstly, a destination folder and an origin folder were defined. A directory was loaded containing all images in the origin folder with the JPEG type, and an integer n acted as the value of images to choose was selected, in this case, 380. In the next step, a row vector of length n was filled with randomly selected unique integers from 1 up to the total number of elements in the directory, inclusive, by using the `randperm` function. A subdirectory was created by taking the image corresponding to the random unique integer in the row vector, and finally, these images were moved to the destination folder, for the 'Training' and 'Validation' sub-datasets. This process was repeated for the 'Testing' sub-dataset, by replacing the destination and the integer n with 43, making the 'Training & Validation' dataset to 'Testing' dataset ratio roughly 90:10. The 'Training & Validation' set was subsequently split into another 90:10 ratio when training the neural networks. Separating a 'Validation' dataset is important to estimate model performance during the training process, and to avoid overfitting to the training dataset.

3.3.3 Common Convolutional Neural Network Layers

To analyse the different pretrained networks, it is worth firstly discussing the most common types of layers found in a convolutional neural network, which are convolution layers, subsampling layers, and fully connected layers.

Convolution layers apply a convolution to the input, by using a kernel, and pass the result, known as a feature map, to the next layer. By convolving an image, the size of the image will decrease based on the size of the kernel, as well as other factors, shown in the following equation

$$output = K\frac{(W-K+2P)}{S+1} \tag{19}$$

where W is the size of the input image, K is the size of the kernel, S is the stride of the kernel, P is the zero padding used on the border of the output image and n is the number of layers in the input image.

[46]

The purpose of convolution layers is to determine general features of the input image, for example edges and corners. Convolution layers can apply multiple kernels to a single image input, by introducing a third dimension. Assuming n_c kernels are applied in a single convolution layer, the output tensor, or feature map will have a depth of n_c also. [47]

Subsampling layers often follow a convolution layer, to down sample and reduce the size of the feature map. This reduces the parameters that the neural network needs to learn, reducing overall computation. There are two main types of pooling layers, max pooling, and average pooling.

Max pooling chooses the greatest value element of the feature map in the region covered by the kernel, as described in image 48.

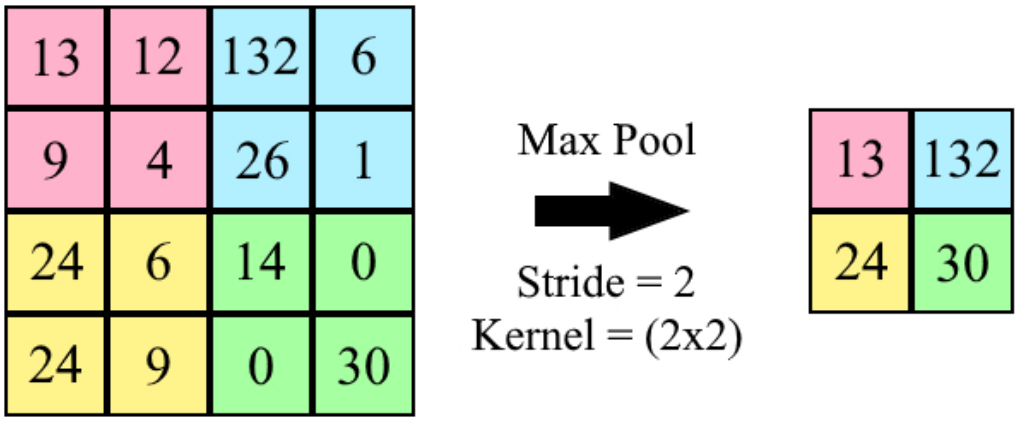


Fig. 48: An example a Max Pooling layer, created in Paint.NET

Average pooling calculates the mean of the feature map elements that are covered by the kernel, as described in image 49.

Fully connected layers multiply the inputs by a weight vector and add a bias vector, both of which can be subsequently trained and improved, an example of which is shown in Figure 50.

Once a pre-trained model has been selected, the task of the network is altered from classifying 1000 image sets, down to classifying four image sets. To do this, some of the layers must be altered. In the final fully connected layer, the output layer must be equal to the number of classes that can be classified. To implement transfer learning, this layer must be edited to have an output of 4 neurons representing the

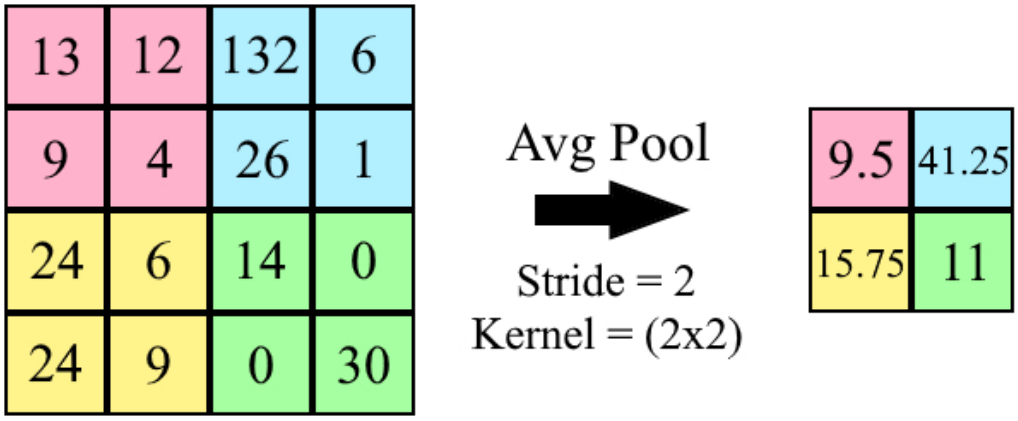


Fig. 49: An example an Average Pooling layer, created in Paint.NET

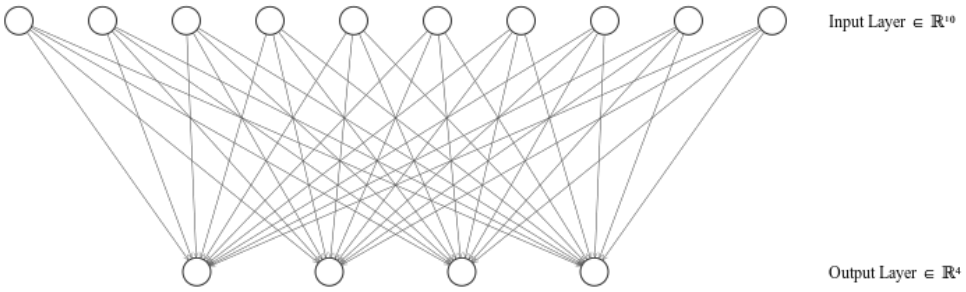


Fig. 50: An example of a fully connected layer, with an input of 10 neurons, and an output of 4 neurons.

four classes of images that must be classified by the system.

Following the final fully connected layer, an activation layer is used to normalise the output over the predicted classes based on a probability distribution. The most common of these is a softmax layer.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=0}^{n} e^{z_j}} \tag{20}$$

for $i = 1, ..., n$, where the input $z = (z_1, ..., z_n)$ is the output of the final fully connected layer.

The output of this layer will be a vector with each component being in the range (0,1), and all components adding to equal 1. Therefore, the output of the softmax layer can be seen as a vector of probabilities referring to the probabilities of the original image belonging to each class in a multi-class problem.

The final layer of each neural network is a classification layer. As our dataset was reduced to an equal number of images for each class, no external weighting must be implemented on the vector of probabilities produced by the softmax layer. This layer simply reads the highest probability from the

vector of probabilities and assigns the relevant class to that image. To implement transfer learning, the final layer of the network must be changed. This is to change the names and number of classes to assign, as we have transferred from classifying images in networks outputting 1000 classes, to networks classifying four classes.

By using MATLAB and the additional Deep Learning Toolbox we can select pre-trained models and make the required changes to apply transfer learning. [48]

3.3.4 Issues with a small dataset

Overfitting is a problem with all datasets; however, this is furthered by the fact that the dataset used is small. Overfitting is where a neural network understands the training dataset perfectly, but is unable to accurately classify unseen validation images. The small dataset also leads to the potential for outliers during training, where an image is notably different from the rest of the class within the test dataset. Sampling bias could be a problem, whereby the images of the dataset are impracticable, as they are artificially gathered in perfect conditions, in comparison to drone images of a leaves.

One way to avoid overfitting and outliers is to train the neural network with a bigger dataset, however this is not easily available for high quality diseased grapevine leaves. A valid alternative is to allow data augmentation. Data augmentation is where modifications are made to a dataset, to artificially increase the amount of data.

To apply this technique, training images are stored in an `augmentedDataStore`. An mini batch is a subset of a dataset, and an epoch is the number of subsets that are applied to the data. During training, each mini batch is augmented based on a predetermined set of adjustments, so that each epoch uses a slightly different data set. This may also help with sampling bias.

For each epoch during the training phase, the data set was augmented randomly according to Table 3 of image transformation options. Any or all of the transformations could be carried out, randomly for each image. A validation dataset is also used to test the performance of the network during the training network, to help avoid overfitting.

3.3.5 Choosing Networks

Different networks will have both positive and negative aspects to them. There are three main differing parameters to compare between the neural network choices; the first, and most important being

Transformation	Options
Image rotation	$[0°, 360°]$
X reflection	Yes/No
Y reflection	Yes/No
Scale factor	$[1, 2]$

Table 3: Transformation Options.

accuracy. The neural network should be able to classify images correctly as frequently as possible.

In a multi-class classification problem, accuracy is equivalent to the fraction of correctly predicted classes, where the correctly predicted classes are shown on the leading diagonal of the confusion matrix. From another perspective, when testing a random image, accuracy gives the probability that the model is able to correctly predict the class.[49] The equation for accuracy is shown in equation 21.

$$\text{accuracy} = \frac{True\ Positives + True\ Negatives}{True\ Positives + True\ Negatives + False\ Positives + False\ Negatives} \tag{21}$$

A slightly increased importance is placed on ensuring healthy leaves are not misclassified, as if one type of disease is misclassified as a different type of disease. This can be described through diseased leaf sensitivity.

$$\text{sensitivity} = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{22}$$

Equation 22 can be described as the true positive rate, and in the current case, 'True Positives' refer to the number of healthy leaves classified as healthy leaves. 'False Negatives' refer to the number of diseased leaves (of any class) which were falsely classed as healthy.

As described in section 1.3, close-up images of leaves must be classified with an accuracy of at least 90%, and a diseased leaf sensitivity of over 95%.

Other important factors to consider include how much memory the neural network will take up and the time taken to compute classes from new inputs. However, these factors are far less important than the accuracy. Regarding the size of the neural network, images will be uploaded and downloaded from an API, discussed in section 2.3. The neural network can be run on a dedicated system, or a laptop connected to the API, not on the drones' hardware directly. This means that size is a minimally important factor. Time taken to classify new images is important, but this can be far more lenient than the accuracy. Assuming a dataset of 1000 images of diseased leaves, the classification must be finished within 10

minutes.

In this project, transfer learning was carried out on four different pre-trained networks, to compare accuracy and time. These were SqueezeNet, GoogLeNet, ResNet-50 and EfficientNet-b0. Each of these networks were pre-trained on the ImageNet dataset, a hierarchical dataset with over a million images with a thousand classes, which produces a good basis for transfer learning. [50]

3.3.6 Results

SqueezeNet is a deep convolutional neural network developed in 2016, by researchers from DeepScale, University of California Berkeley and Stanford University. This network was designed to achieve the same high level of accuracy as AlexNet when classifying over a million images in the ImageNet dataset into 1000 classes, while using far smaller network architecture, with only 18 layers of depth. [51] GoogLeNet is a 22 layer deep convolutional neural network based on the "Inception" Network, developed by researchers at Google. [52] ResNet-50 is a 50 layer deep convolutional neural network. ResNet is a residual neural network, which stacks residual blocks on top of each other. [53] EfficientNet-b0 is a 290 layer deep convolutional neural network which scales all dimensions of depth, width and resolution based on a compound coefficient. [54]

Once the changes discussed in section 3.3.2 were carried out to the pre-trained networks, the networks were then trained on the dataset discussed in table 2. The networks were trained with a mini batch size of 36 iterations per epoch, with 10 epochs, to ensure that the entire dataset was used to train the networks. (e.g. $36 \times 10 > 342$)

The validation dataset was tested every 9 iterations, such that a smooth validation accuracy curves were output.

Name	Training Time	Final Validation Set Accuracy
SqueezeNet	2 min 22 sec	97.37%
GoogLeNet	3 min 58 sec	99.34%
ResNet-50	19 min 2 sec	95.39%
EfficientNet-b0	23 min 49 sec	93.42%

Table 4: Training Data.

Training data is shown in table 4. SqueezeNet and GoogLeNet took substantially less time to train; however, the training time was not important for our requirements, as the network will be pre-loaded onto a local system. The validation set was seen throughout the training of the network, and may be

susceptible to overfitting, and hence is tested by a completely unseen set of 172 images, as described in table 2.

Once the networks were trained on the 'Training & Validation' dataset, the unseen 'Test' dataset was classified individually on the trained networks, and confusion matrices were generated. A confusion matrix is a matrix showing the performance of a neural network, showing the predicted and true classes of the test dataset.

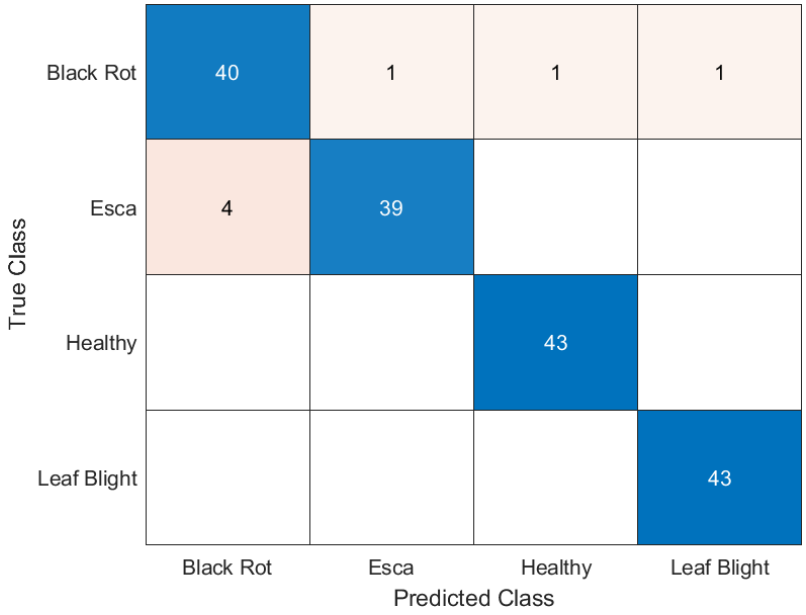


Fig. 51: Confusion Matrix for SqueezeNet.

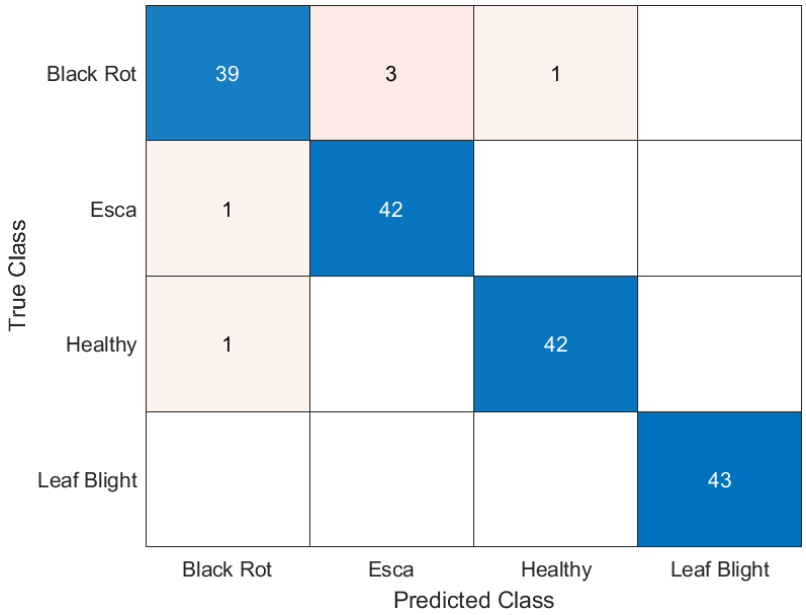


Fig. 52: Confusion Matrix for GoogLeNet.

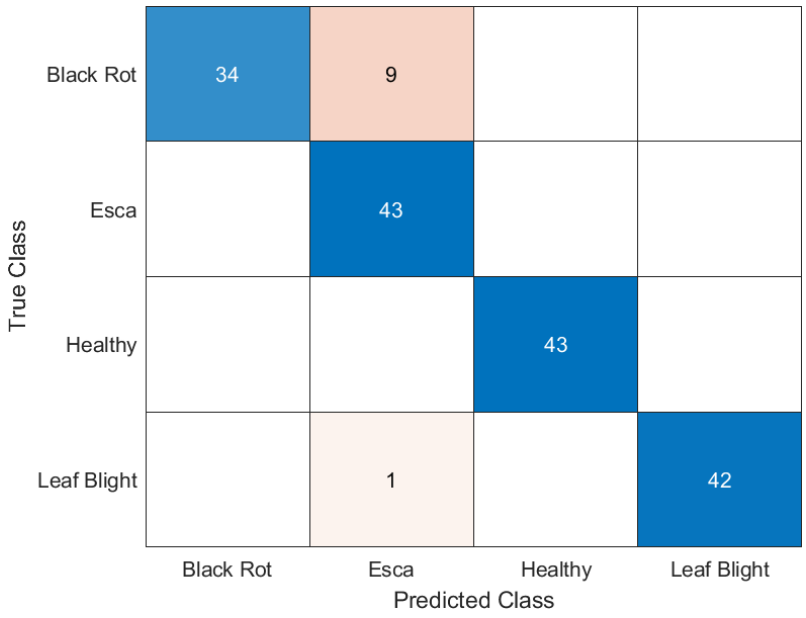


Fig. 53: Confusion Matrix for ResNet-50.

Fig. 54: Confusion Matrix for EfficientNet-b0.

The accuracy and sensitivity values are calculated from the confusion matrices in figures 51, 52, 53 and 54, by using equations 21 and 22 respectively.

Although all of the tested neural networks achieve the accuracy and sensitivity requirements, the best performing network seems to be GoogLeNet. To improve precision, the dataset would be larger such that the tests can be carried out with more images. This is one way the model can be further improved.

Name	Network Accuracy	Network Sensitivity
SqueezeNet	95.9%	99.2%
GoogLeNet	96.5%	99.2%
ResNet-50	94.2%	100%
EfficientNet-b0	93.6%	97.7%

Table 5: Testing Data.

To improve accuracy further, probabilities of classification is also determined for each image, as shown in figures 55, 56, 57 and 58. If the network provides the greatest class probability of below 70%, then the image will be sent to an operator, using the API and GUI discussed in section 2.6.4 to verify the validity of the classification. In many cases, as seen in figures 55 and 56, the network gives a very high probability to most leaves, from the 172 total images in the test dataset, 166 images have a probability of greater than 70%.

Fig. 55: A leaf with the Esca disease in the test dataset, which was correctly classified with 99.8% probability.

Fig. 56: A leaf with the Esca disease in the test dataset, which was correctly classified with 98.7% probability.

Fig. 57: A leaf with the Esca disease in the test dataset, which was correctly classified with 50.3% probability of Esca, and 49.7% probability of Black Rot. This would be forwarded to an operator for validation.

Fig. 58: A leaf with the Black Rot disease, in the test dataset, which was **incorrectly** classified with 61.6% probability of Esca, and 38.3% probability of Black Rot. This would be forwarded to an operator for validation.

This threshold of 70% can be increased, such that more images are checked, depending on the number of operators available to verify diseases.

One potential future improvement could be to use reinforcement learning, such that the neural network models would learn from previous errors, and gain more images in datasets after analysing fields, consistently improving and training the network. This would introduce a training time requirement.

3.4 Conclusion

This part of the chapter has discussed the image analysis of precision viticulture. The requirements and specifications set out in section 1.2 have been met by implementing image stitching for a higher resolution overview, vegetation index analysis to locate areas of diseased clusters of vines, and comparing neural networks used to classify diseases using machine learning.

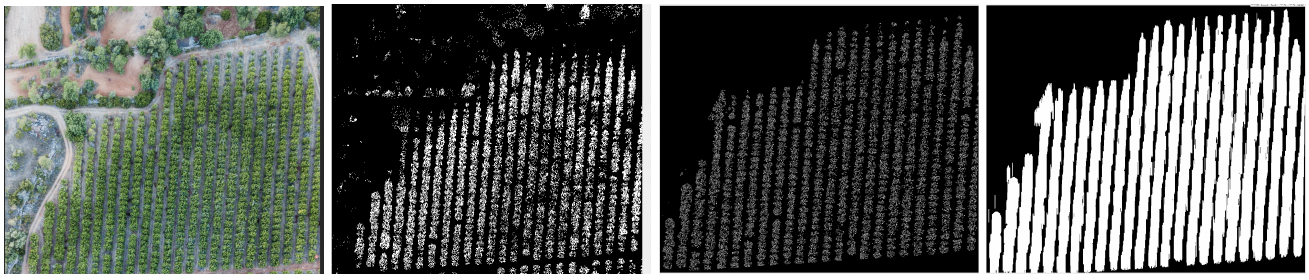
The image analysis part of this project could be improved by having a larger dataset to train the neural networks for disease classification. The image stitching subsection would be improved by having precalibrated hardware to remove barrel distortion. The disease heatmapping with vegetation indices part of the project pipeline could be improved by having prior index data about vineyards to make threshold values more accurate.

3.5 Image Processing for Path Planning

This section presents the line detection algorithm that identifies the alignment of the vine rows, to be used within the viewpoint generation algorithm of the path planning system discussed in Section 4.5.1.

3.5.1 Pre-processing and Edge Detection:

For the first step, the RGB image of the region of interest (ROI) is colour-thresholded to increase the contrast between the soil and the crop rows and this image is converted to grayscale in preparation for the edge detection(Fig.6(b)). Following this step, the Canny Edge Method[55] identifies the edges, i.e. the significant local changes in the image intensity[56]. As shown in Fig.6(c), the edge detection distinguishes the crop rows from the background, but they are uneven due to the noise in the original image. For the line detection to work accurately and reliably, The edge detected the noisy lines are evened out using the morphological dilation operation, which enlarges the boundaries of the white pixels and, as shown in Fig.6(d), creates continuous crop lines.



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

Fig. 59: (a) Original (b) HSV Threshold (c) Canny Edge Detected (d) Dilated

3.5.2 Line Detection:

Finally, a line detection operation is performed on the edge-detected and optimized image. This is done with the Hough Transform algorithm, which works as follows: Every edge point in the image can have infinite lines passing through it that can be represented by $\rho = x\,\cos(\theta) + y\sin(\theta)$ where ρ is the length of the normal line, and θ is the angle.[57]. Every edge point generates a cosine curve in the Hough Space which is a 2D plane that has a horizontal axis representing θ and the vertical axis representing ρ. If two edge points are on the same line, their Hough Space cosine curves will intersect on that specific (ρ,θ) pair. The Hough Transform algorithm then detects lines by finding the peaks, i.e. the (ρ,θ) pairs with a number of intersections larger than a certain threshold. In a conventional vineyard layout, all crop rows

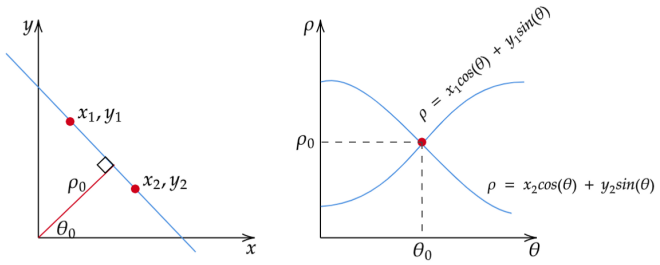


Fig. 60: Hough Space

are parallel to each other. This indicates that it is sufficient to identify the direction of one of the crop rows to determine the optimal row alignment. To this end, a Hough Transform algorithm is implemented on MATLAB, which identifies the line with the highest frequency of hough peaks and returns its angle with respect to the positive x-axis and the intersection points with the ROI polygon.

4 Complete Coverage Path Planning for Precision Viticulture

4.1 Introduction

One of the main problems pertaining autonomous flight is path planning (PP). Stages 1-3 of our Precision Viticulture (PV) system workflow involves multi-UAV flight to acquire aerial footage and deploy treatment. Using the fact that the ROI is known and static, we aim to maximise the time/resource efficiency of the UAV flight by pre-planning the paths. The multi-agent area coverage problem is a challenging optimisation problem with three sub-problems: (i) decomposition of the field to a set of viewpoints (ii) allocation of the viewpoints to the UAVs (iii) efficient and least-cost path planning for each UAV. We aim to maximise the efficiency and feasibility of our service, so practical challenges were considered– despite increasing the complexity of the PP problem. The main practicality affecting the operation cost is the pre-flight setup of the UAVs, which depends on the number of operators and UAVs. We designed and implemented two separate methodologies for solving the optimisation problem for both of the surveying stages and the spot-treatment stage. The main optimisation objectives are cost minimisation and efficiency maximisation, which are obtained through the minimisation of the operation time, resource use and number of operating drones. The path planning methodology design is based on these objectives subject to constraints and specifications aligned with the combination of the general PV system and stage-specific requirements. Following the methodology design, the algorithms were implemented on MATLAB. Upon execution, the path planning method designed takes in operation information (including

number of operators) and returns the optimal task allocation and corresponding set of flight paths that yields the fastest operation time with the least amount of resource and UAVs used. This solution will be available to all systems through the endpoints managed by the HRI. The control system will access the paths to ensure trajectory following and collision avoidance, while the HRI system will display the paths on the user interface. The validity and efficiency of the solution has been established through various simulation runs. In Section 4.6 the results of these simulations for various area/UAV scenarios are presented and discussed to provide a proof-of-concept.

4.2 Path Planning Problem Definition

The PP methodology addresses the multi-UAV path planning problems relating to Stages 1-3 of our PV System (Section 1.1.) Specifically, the stages correspond to a general survey of the whole vineyard, a detailed survey of a specific area, and spraying a specific region with an agrochemical. Stages 1-2 are grouped under the operation type Survey, as their requirements/constraints are similar. Stage 3 (operation type Spray) has significantly different requirements/constraints then the Survey type, so two types of multi-UAV CPP problems arise. To define these operation-specific problems, we will first define the general path planning problem (Note that the individual task allocated to a UAV is referred to as a mission) :

> To develop optimal area decomposition, task allocation and complete coverage path planning strategies to generate a path for each UAV part of a team of m, which has to perform either a survey or spray type operation on a pre-defined Region of Interest (ROI). The overall operation time and each UAV's mission time must be minimized and not exceed the time limit dictated by the battery duration or tank capacity.

The rest of this subsection defines the terms and assumptions of this problem and system. Most importantly, operation time, i.e. the optimization variable, is defined considering the main practical challenge of operating a multi-UAV system: pre-flight setup. It is assumed that each UAV requires this setup which takes a certain amount of time depending on the team size, available operators and the tasks such as the connection of the batteries, mounting/preparing the load (cameras and spraying tank) and GPS fixing. The time it takes for the latter are constant and known in advance, so the setup time varies only with the ratio of number of operating UAVs to the available operators (*O*). Now, the mission time to be minimized can be defined as the setup time plus the flight time for each UAV. Operation time can then be trivially

defined as the maximum mission time. It is important to note that the inclusion of the setup time means that the simple solution of using all available UAVs and minimizing their flight time will not guarantee a mission/operation time minimization. Hence, optimal task allocation, i.e. finding the optimal team size (m) given the available UAVs (M), becomes a necessity to find an optimal solution.

The ROI is defined by a set of vertices $\mathbf{V} = \{v \mid v \in \mathbb{R}\}$, and the operation base (launch/land point) is set at the origin. The ROI can be initially convex, so no modification is needed. Conversely, the ROI can be non-convex, in which case its vertices are converted to represent its convex hull. Our PV system has two types of UAVs available, P4 DJI Multispectral and DJI Agras T30, undertaking the Survey and Spray operations respectively. The technical specifications for each UAV and its relevant equipment are known, i.e. battery life and payload. Furthermore, the UAVs' flight altitude and maximum speed are pre-determined according to specifications presented in Section 1.3. All of the constant and operation-specific data are assumed to be known and will be accessed through the endpoints managed by the HRI.

4.3 Operation-Specific Requirements

The following subsection will present definitions and requirements particular to the two operation types- Survey and Spray- based on the general path planning problem and requirements, technical definitions, and assumptions presented in Section 4.2

4.3.1 Operation Type: Survey

This pertains to stages 1-2 in our PV system. For the survey type operation, a team of UAVs equipped with onboard cameras is required to obtain footage, ensuring complete coverage of the region of interest, subject to the image overlap and resolution requirements from the Image Processing Task. These requirements dictate the size of partitions for the ROI decomposition for the viewpoint generation. The survey UAVs are restricted temporally only by their battery limitations. Given this, the surveying path planning problem can be defined by two tasks:

1. Finding the optimal size for the UAV team $m \leq M$ that minimizes the operation time to completely capture the ROI.
2. Plan and assign paths to each UAV subject to the Image Processing requirements so that the operation time is minimized and is less than the battery life, also ensure each UAV in the team of

m completes its mission in minimal time.

4.3.2 Operation Type: Spray

This pertains to stage 3 in our PV system. For the spraying mission, a team of UAVs equipped with spraying tanks and nozzles is required to sufficiently coat the grapevine in the ROI with the agrochemical to treat the disease identified by the image processing system. To meet the sufficient coating requirement, the offline viewpoint generation algorithm must discretize the ROI into viewpoints aligned with the grapevine rows. Detecting crop row alignment is a problem unique to this operation type and requires an image processing solution. The UAVs are restricted temporally by battery life, payload, i.e. the spray tank capacity, and the spray rate for the spray mission. The time it takes to run out of the agrochemical and the amount of grapevine coated with it can be found using the flight speed, payload and spray rate. The maximum flight duration for the spray UAVs is then defined by the stricter one of the battery and payload restrictions. Given this, the spraying path planning problem is defined by the following objectives:

1. Finding the optimal size for the UAV team $m \leq M$ minimizes the operation time to sufficiently deploy the relevant agrochemical to the grapevine in the ROI.
2. Plan and assign paths aligned with the vine rows so that the operation time is minimized and is less than the battery life, and ensure each UAV in the team of m has enough tank capacity for its mission and completes it in minimal time.

4.4 System Design

Both of our operations are primarily concerned with covering the entire ROI in minimal time. Preliminary research revealed that the specific branch of multi-agent path planning for coverage operations is Complete Coverage Path Planning (CPP). It involves computing a feasible path encapsulating a set of viewpoints through which a robot must pass to cover the ROI completely. CPP generally involves generating viewpoints, path planning, and coverage completeness quantification [58].Various systems are designed and implemented to perform CPP, all subject to different terrain shape and conditions, obstacles and task requirements. This subsection aims to justify the preliminary design decisions pertaining to the solution of the operation-specific CPP problems of the PV system by a survey of the current CPP methods.

Viewpoint Generation: In most of the existing literature, viewpoint generation is executed with model-based or non-model based methods. Model-based approaches are typically used when the environment explored is known in advance and is static— this allows the viewpoint generation can be completed offline. Model-based approaches are suitable for inspection, modelling [58], and most agricultural applications. The main limitations of this method are due to the quality of the information available on the environment. In comparison, non-model based methods are employed when there is no prior information about the environment or when there are dynamic elements. Vineyards are assumed to be uniform, static, and obstacle-free regions, and our HRI system will provide preliminary data on the field boundaries, so there is no need for an online exploration of the environment. Therefore the designed solution will be model-based.

The next step for generating viewpoints is Region Decomposition. The majority of the works in the field can be categorized into exact decomposition or approximate decomposition[59]. Exact decomposition divides the area into non-overlapping and usually irregular cells that, upon reunion, form the exact same area[60]. Two popular methods include Trapezoidal[61] [62] and Boustrophedon Decomposition[63]. Due to the irregular and non-overlapping cells, exact methods are not convenient for our system, which requires acquiring equally-sized and overlapping image samples. Approximate methods partitions the area into a fine grid. The partitioning elements are the same size and shape, but the union of the cells only approximates the target region[64]. Typically, the grid-size is determined by the sensor characteristics [65]. The region decomposition methodology designed for our system will build on this. Our system performs two types of coverage operations (survey and spray.) Due to operation-specific and image processing requirements (both presented in Section 4.3), the UAV motion is limited to a back and forth/sweeping motion. In other words, all viewpoints/ grid cells on a particular sweeping row will be visited by the same UAV. This motion limitation allows the cells on a row to be lumped into one collective row cell, defined by the end viewpoints. Considering this, the region decomposition method for the CPP presented in this Section will follow the approximate decomposition method and subdivide the area into a sequence of row cells, building on the work of Avellar [66]. Note that the width and orientation of the row cell will depend on the operation type, so two region decomposition algorithms were designed for this PV system.

Path Planning: Path planning algorithms vary significantly given the terrain conditions, obstacles, mission objective and the decomposition method. The following algorithms surveyed are concerned with

solving CPPs areas discretized into a regular grid using the approximate cellular decomposition. These algorithms can be categorized into grid-based and graph-based. Grid-based algorithms include the Wavefront[67] [68], Spanning Tree[69][70], Deep Limitied Search[71] and various Neural Network-based[72] algorithms. It is important to note that the grid-based algorithms are computationally complex but very efficient for more complicated CPP problems with large, irregularly shaped areas covered by free-moving vehicles. Our UAVs are restrained to follow rows and operate on UK vineyards that are smaller in size and complexity. Furthermore, the algorithm will be run on a personal computer with limited RAM. So the algorithm complexity should be kept to a minimum. The grid-based solutions will not be suitable considering the practical technical limitations and complexity of the problem. So the path planning algorithm designed for this system follows and builds on graph-based algorithms. Graph-based solutions involve converting the decomposed area into a graph, i.e. a set of vertices and edges stored in an adjacency matrix. Using the graph, the CPP problem can be treated as a travelling salesman [59] or vehicle routing problem[66] [73] and solved by employing Mixed-Integer Linear Programming (MILP)[74]. To get a solution that meets the PP system objectives and requirements, certain modifications are made to the classic VRP problem, which are explained in detail in the Methodology & Implementation Section.

Complete Coverage Quantification (CCQ): If the viewpoints are generated appropriately, i.e. when all points are traveled the complete map can be reconstructed or all vine rows are sprayed, and if the mission paths collectively include all the viewpoints complete coverage is achieved. So the coverage is quantified with the number of viewpoints visited by the team of drones against the total number of viewpoints. The solution is constrained to be complete coverage by constraining its CCQ to be 1. Note that CCQ can also be used during operation to track progress.

This concludes the general design of the solution based on system specifications, requirements and existing methods. The solution pipeline is as follows: a model for the operation area is received, the area is discretized into sweeping rows using approximate decomposition, the decomposed area is converted to a graph, the CPP problem is then treated as a modified VRP optimization problem, and MILP is used to solve for optimal paths. As illustrated in Figure 61, the implementation of this Path Planning System involved a writing subprograms that complete the viewpoint generation, and path planning tasks and a main script which is the interface controlling the flow between the input, subprograms and output. The main script also includes a display function which is not included in the solution pipeline, which is implemented to provide visual proof-of-concept for the simulation runs. The methodologies, specific

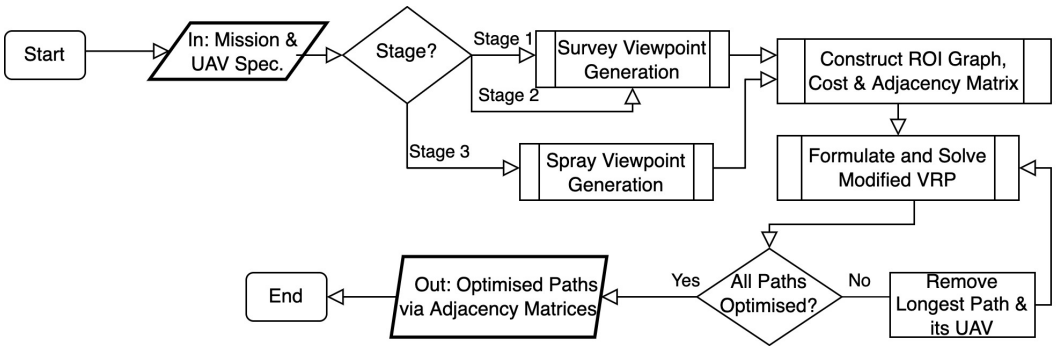


Fig. 61: Solution Flowchart

implementation details, and pseudocode is presented in the following section.

4.5 Methodology & Implementation

The MATLAB environment is used to formulate, implement and solve the Coverage Path Planning Problems defined in the previous sections. The main code is made up of 3 parts: (i) Configuration, (ii) Graph Construction, and (iii) Building and Solving the MILP Optimisation Problem. In the first part, operational variables such as the number of operators and UAVs, mission type, UAV, camera and payload specifications, and initial ROI vertices are accessed through the API developed by the HRI system. For the second part, two functions that execute the viewpoint generation for the two operation types, and a function to construct node graphs according to the viewpoints is written. The built-in Image Processing Toolbox is used for the image pre-processing and line detection parts of the viewpoint generation for the survey operation, process detailed in Section 3.5.

The third and final part is concerned with building and solving the modified VRP optimization problem (set in Section 4.5.3.) MATLAB has a built-in Optimization Toolbox that provides functions for finding parameters that minimize or maximize objectives while satisfying constraints and includes solvers for mixed-integer linear programming (MILP) [75]. T222he external YALMIP toolbox for MATLAB can also be used. This toolbox is particularly favourable as it makes the development of optimization problems extremely simple[76] and allows access to the GUROBI solver, which is the fastest commercial solver capable of solving MILP problems [77]. Both toolboxes work sufficiently, but since this subtask is mainly concerned with minimizing time, YALMIP is used in the front end, and GUROBI is used in the back end of this implementation. An optimisation function is created to build and solve the modified VRP problem given the specifications. The main script calls the optimisation function and contains the iteration loop that executes the solution strategy presented in Section 4.5.3 that ensures all mission paths are optimized.

The output of this PP system is a matrix that is a stack of the directed adjacency matrices for all UAVs, which defines the optimal paths for each UAV. Note that the adjacency matrices of all available UAVs are returned despite the optimal team size. The idle UAVs will have an all-zero adjacency matrix. The control system will then access these paths to ensure optimal trajectory tracking and collision avoidance upon execution. Additionally, the user interface developed by the HRI will display the planned paths.

4.5.1 Viewpoint Generation Algorithms

Viewpoints are placed according to the operation requirements and are paired to decompose the area into a set of rows. Allowing the the convex hull of the ROI to be represented as a graph $G = (V, E)$ where the the vertices (V) are the viewpoints and the edges (E) are the rows.

Region Definition The region of interest is defined through a list of vertices $\mathbf{V} = \{v_1, v_2..v_n\}$ assigned by the HRI system (see Figure 62). The list of vertices generates a polygon that can be convex or non-convex[78]. If the polygonal representation of the area is non-convex, as in Figure 63, the convex hull of the region, shown in Figure 64, is used.

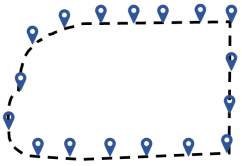


Fig. 62: Vertices defining the ROI

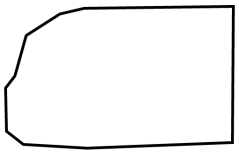


Fig. 63: Polygon of ROI

Fig. 64: Convex Hull of ROI

Region Decomposition The region is decomposed into a set of rows, which are defined by a pair of viewpoints. The decomposition process involves two tasks: determining (i)the optimal alignment of the rows and (ii) the optimal spacing of the rows. The optimization objectives are based on the specific operation and the overall system requirements. Once the rows are appropriately placed on the ROI, viewpoints are placed where the row lines intersect the ROI polygon. Presented below are the two optimal alignment/row-spacing determination methodologies for the two operation types.

Survey Operation: Row-Spacing/ Alignment: For the surveying operation, the time minimization and meeting the image processing requirements are the main factors affecting the choice for the optimal row alignment and viewpoint spacing. Each UAV will cover its assigned subregion during the surveying mission using a back and forth/sweeping motion. All UAVs are restrained to follow the same sweeping direction, i.e. the ROI must be discretized into parallel rows. The flight duration for each subregion

consists of the time taken to traverse along the rows plus the time to take a turn at the end of each row[79]. As emphasized by Huang[79], this turning motion takes a significant amount of time, and the minimization of this motion will reduce the duration of the flight significantly. The sweep direction (shown in Figure 65) is perpendicular to the row alignment and determines the number of rows the region will be decomposed. The number of rows is directly proportional to the number of turns. The sweep direction that yields the least amount of rows and therefore turns is when it is aligned with the smallest height of the ROI. Figure 65 illustrates the paths when the sweep direction is aligned with the smallest and largest height. As expected, the latter has more turns than the former. A MATLAB function is implemented

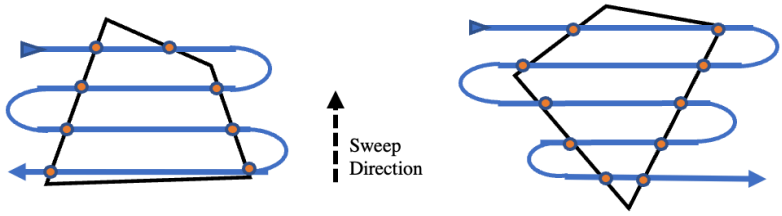


Fig. 65: Sweep Direction

to identify the orientation that yields the smallest height of the ROI polygon. The polygon is rotated by $\theta = \{0, 1.. \mid \theta \leq 2\pi\}$ using a transformation matrix, and its height is measured. This process is illustrated in Figure 66 and the psuedocode can be found in Algorithm 2. Once the alignment is set,

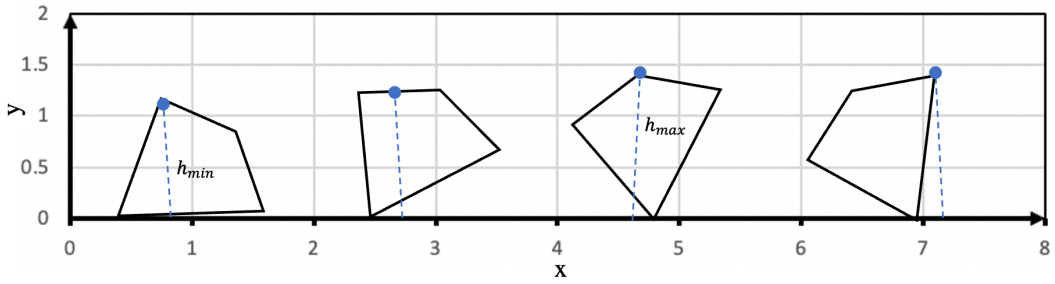


Fig. 66: Sweep Direction

the task of determining the optimal spacing remains to decompose the region into a set of rows. This width is defined entirely by the Image Processing Requirements. To produce a continuous and complete region coverage, the image stitching algorithm requires a fraction of the subsequent images to overlap. To determine the appropriate spacing (w), the size of the ground captured in a frame, i.e. the footprint of the camera, and the overlap (s) are used. The footprint is calculated using the Field of View (FoV) and Focal Length (f_l), information included in the camera's technical specifications. The following equation

23 shows the formula for calculating the optimal row spacing (w) and footprint (L).

$$\omega = L(1 - S) \text{ where } L = \tan\left(0.5 \times \mathrm{FoV}_0\right) \times 2\mathrm{f}_l \tag{23}$$

Spray Operation:Row-Spacing/Alignment: The spraying mission has additional requirements that impose strict constraints on the optimal row alignment and spacing. As with the surveying mission, time minimization and complete coverage system requirements must be met. On top of this, the UAVs are to fly along the crop rows in their assigned subregion, coating them with the relevant agrochemical. The task of finding the optimal alignment of the rows becomes an image processing operation: identifying the direction of the crop rows in the given an RGB image of the ROI. To this end, a three-step algorithm is developed: (i) Pre-processing, (ii) Edge Detection, and (iii)Line Detection. The methodologies and implementation details of this line detection algorithm are presented in Section 3.5. Once the optimal row alignment is determined with the line detection algorithm, the task of identifying the row spacing remains to conclude the decomposition process. This task can be done in two ways: determining the average distance between the subsequent lines identified by the line detection or using a constant value assuming most vineyards follow the conventional layout. The first method is inefficient and computationally challenging as the line detection algorithm identifies multiple lines on the same crop row. Accurately determining the row spacing then requires sifting the detected lines on each crop row to extract the one closest to the centerline. This is a challenging algorithm to implement. The second method relies on the assumption that the vineyard of interest is conventionally designed, i.e. the crop rows are spaced to achieve optimal sunlight efficiency. According to Kurtural[80], 3 meters is the conventional row spacing. Testing this spacing value on various images of UK vineyards returned satisfactory results. Hence the second method is used to determine the optimal row spacing. In the case of a non-conventional vineyard layout, the spacing value can be provided by the viticulturist pre-operation, and the default value can be updated using the HRI system.

Algorithm 2 Survey Viewpoint Generation

```
x ← x-coordinates of ROI
y ← y-coordinates of ROI
h ← height of ROI
w ← Camera Footprint
for angle = 1, 2, ...360 do
    Calculate Rotation Matrix for angle
    R ← Rotation Matrix
    rotated_roi ← R*[x y]'
    h_rot ← height of rotated_roi
    if h > h_rot then
        h ← h_rot
        sweep_align ← angle
    end if
end for
Rotate ROI with sweep_align
Place vertical lines in w increments
vpoints ← Line and ROI intersection points
Return vpoints
```

Algorithm 3 Spray Viewpoint Generation

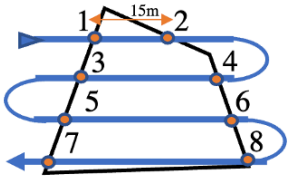
```
x ← x-coordinates of ROI
y ← y-coordinates of ROI
h ← height of ROI
w ← Conventional Row Spacing
function croprowdetect(ROI)
    Preprocess the ROI
    Apply Hough Transform
    spray_align ← angle of Hough Line
    return spray_align
end function
Rotate ROI with spray_align
Place vertical lines in w increments
vpoints ← Line and ROI intersection points
Return vpoints
```

Viewpoint Placement The placement and pairing of the viewpoints follow the same methodology for both operation types once the optimal row alignment and spacing are determined. The process is as follows: lines of infinite length aligned and spaced optimally are superposed on the ROI polygon. The viewpoints (illustrated by the red points in Figure 65) are generated by locating the intersection points of these infinite row lines and the ROI polygon. The pseudocode (Algorithm 2,3) above present the complete viewpoint generation algorithm for the two operation types.

4.5.2 Graph Representation & Cost Matrix

The viewpoint and row decomposition of the ROI allows it to be conveniently represented as a graph. A common way of representing a graph is by an adjacency matrix, which encapsulates the vertices of a graph and their relation to the other vertices [81]. For a graph of n vertices, the adjacency matrix A is a n x n matrix with non-diagonal binary elements $a_{ij} \in \{0, 1\}$ indicating connectedness. A graph can also be represented with a cost matrix, C of size n x n. The non-diagonal elements $c_{ij} \in \mathbb{R}$ hold the cost associated with traveling between vertices i and j. Figure 67 shows the directed adjacency and the cost matrix for a decomposed ROI with eight vertices, where the cost is euclidean distance. For this system, the cost matrix will contain the time it takes between each vertex, i.e. euclidean distance divided by UAV speed. This cost matrix will be used in the formulation of the optimisation problem, and the solution, i.e. the optimal paths, will be contained in a matrix that is a stack of the unique adjacency matrices for each

UAV. This matrix will be returned to the relevant systems through the endpoints.



$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \quad C = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} 0 & 15 & 7 & 19 & 14 & 22 & 20 & 25 \\ 15 & 0 & 14 & 9 & 15 & 14 & 22 & 20 \\ 7 & 14 & 0 & 19 & 7 & 19 & 14 & 16 \\ 19 & 9 & 19 & 0 & 15 & 8 & 23 & 13 \\ 14 & 15 & 7 & 15 & 0 & 20 & 8 & 22 \\ 22 & 14 & 19 & 8 & 20 & 0 & 19 & 8 \\ 20 & 22 & 14 & 23 & 8 & 19 & 0 & 21 \\ 25 & 20 & 16 & 13 & 22 & 8 & 21 & 0 \end{bmatrix} \end{array}$$

Fig. 67: Adjacency and Cost Matrix

4.5.3 Modified Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is an NP-Hard combinatorial optimization problem, first introduced by the seminal research paper by Dantzig and Ramser[82]. It is a generalization of the Traveling Salesman Problem (TSP), which is worded as:

(Def.1) A salesman is required to visit each of n cities, indexed by 1,..n. He leaves from a ”base city” indexed by 0, visits each of the n other cities exactly once, and returns to city 0. During his travels he must return to 0 exactly t times, including his final return (here t may be allowed to vary), and he must visit no more than p cities in one tour. (By a tour we mean a succession of visits to cities without stopping at city 0.) It is required to find such an itinerary which minimizes the total distance traveled by the salesman.[83]

Essentially the TSP solves the simplest problem of the VRP, which is concerned with the scenario where multiple vehicles with limited capacities are tackling the problem (Def.1) above. It follows that the CPP problems relevant to the multi-UAV PV system presented in this paper can be expressed as a Vehicle Routing Problem, which is worded as:

(Def.2) A team of M-UAVs is required to visit each of n viewpoints, indexed by 2,..n. The UAVs leave from a ”base” indexed by 1, collectively cover each of the n other viewpoints exactly once, and return to base 1. During the mission, UAVs must return to base 1 precisely one time, having completed the coverage of their assigned subregion and must not exceed their battery/payload capacity. Each UAV must be assigned a path to minimize the operation time and UAV mission times.

Note that the conventional VRP formulation will not yield an optimal solution for our CPP problem, as we are including the setup time in the cost function, and have tasks such as image acquisition or spraying

which add additional constraints to the UAV paths. So the CPP problem of this system is posed as a modified VRP problem. The updated cost function and additional constraints are further explained in Section4.5.3. In the previous subsections, we defined our ROI with a graph G = (V, E) where $\mathbf{V} = \{v_1, v_2..v_n\}$ is the vertex set, the vertex 1 is the start/terminal point set at the origin, vertices 2..n are the viewpoints and E is the edge set. A cost matrix is also defined, specifying the cost (time) associated with each edge, i.e. traveling between nodes i and j. This allows the modified VRP problem to be represented as a graph-theoretic problem [84] that has a mixed-integer linear programming solution. This subsection will present the mathematical model of the multi-agent CPP problems (defined in Section 4.2 and Def.2 above) and the solution calculation using mixed-linear integer programming.

Mathematical Model The modified VRP formulation of the CPP problems will closely follow the integer programming formulation developed by Miller-Tucker-Zemlin[83] with system-specific additions to the cost function (the optimization variable to be minimized) and constraints. The main modifications of our system are the inclusion of the setup time in the cost function and the addition of three constraints that ensure rows are followed, diagonal paths are restrained, and the team size is optimized. Without the former two constraints, our solution is not guaranteed to be complete coverage. We will start the formulation by defining the relevant parameters and definitions:

- **V =** $v_1...v_n$: vertices including the base and viewpoints.
- C_{ij} : cost (euclidean distance) of traveling between vertices i and j where C is a (nxn) matrix
- M, m $\in \mathbb{N}$: Total UAVs available, and optimal team size for a mission.
- $X \in \{0, 1\}$: Binary matrix of size (nxnxm), specifying if a UAV will fly from vertex i to j. It is a stacks of the directed adjacency matrices for all the operating UAVs. $X_{ij}^k \in \{0, 1\}$ where $1 < k \leq m$ denotes the binary value for UAV k flying between i and j.
- $t_{lim} \in \mathbb{R}$: max flight duration due to battery or payload constraints.
- $t_s^k \in \mathbb{R}$: individual set-up time a UAV (assumed to be identical for all UAVs = t_s).
- $T_s^k : \in \mathbb{R}$: cumulative setup-time for the kth UAV.
- $T_f^k : \in \mathbb{R}$: total flight time for the kth UAV.
- $S_{ij}^k \in \mathbb{R}$: speed of the kth UAV flying between the vertices i and j.

- O : number of operators available for a mission.

Cumulative Set-up Time (T_s^k): The individual set-up time (t_s) is the time it takes for the operator to complete a set of pre-flight tasks for each UAV. It is assumed that the kth UAV in a team for a given mission will be launched once the UAVs 1,..k complete their setup. If there are fewer operators available than the UAVs to be prepared, some UAVs will have to hold until the UAVs preceding them are prepared. This wait time (proportional to the number of preceding UAVs and t_s) plus the individual setup time of the UAV is defined as the cumulative setup time (T_s^k). Following this, the total mission time for the kth drone will be the time it takes to cover all its assigned vertices plus the cumulative setup time (T_s^k). This cumulative setup time is calculated as shown in Equation 24 below. Note that the summation operator returns zero if the kth UAV does not leave node 1, i.e. is not included in the operation team.

$$T_s^k = t_s \left\lceil \frac{k}{O} \right\rceil \sum_{j=1}^{N} X_{1j}^k, \text{where } k = 1, \ldots, M. \tag{24}$$

The significance of including the cumulative-setup time is showcased in the following examples. Assume a survey operation to cover a rectangular ROI decomposed into 8 rows, each taking 2.5 minutes to complete. $M = 3$ UAVs with 10 minute setup time, and O = 1 operator is available for this operation. For simplicity, we will neglect the time taken from the base to the first viewpoint, the time it takes to turn between rows and the battery limit.

$$T_s^1 = 10 \left\lceil \frac{1}{1} \right\rceil = 10 \text{ , } T_s^2 = 10 \left\lceil \frac{2}{1} \right\rceil = 20 \text{ , } T_s^3 = 10 \left\lceil \frac{3}{1} \right\rceil = 30 \text{ , } \tag{25}$$

If a single UAV was operating, it would take $2.5 \times 8 + 10 = 30$ minutes to complete the whole area. We can see from equation 26 that just the cumulative setup time for the 3rd drone (T_s^3) takes 30 minutes. This indicates that it is inefficient to use 3 drones for this given operation. To remove the 3rd UAV from the operation $\sum_{j=1}^{N} X_{1j}^3$ is set to zero. It can then be determined that the fastest operation time for this example case is when UAV 1 and UAV 2 covers 6 and 2 rows respectively. This yields an operation time of $20 \times 2.5x2 = 25$ minutes. For comparison if all three UAVs were used the best case operation time would be $30 + 2.5 \times 1 = 32.5$ minutes. The cumulative setup time decreases when multiple operators are available, so using more UAVs becomes more time-efficient. The effect of varying the operator number on the T_s^k is shown in the following example. In this case we increase the available UAVs to M = 5 and

operators to O = 2. The cumulative set-up times are as follows:

$$T_s^1 = 10\left\lceil\frac{1}{1}\right\rceil = 10\ ,\ T_s^2 = 10\left\lceil\frac{2}{2}\right\rceil = 10\ ,\ T_s^3 = 10\left\lceil\frac{3}{2}\right\rceil = 20\ ,\ T_s^4 = 10\left\lceil\frac{4}{2}\right\rceil = 20\ ,\ T_s^5 = 10\left\lceil\frac{5}{2}\right\rceil = 30\ , \tag{26}$$

UAVs 1 and 2 will be prepared simultaneously so both will have a cumulative setup time of 10. Following this the UAVs 3 and 4 will be prepared simultaneously yielding a T_s^k of 10 + 10 = 20. And lastly UAV 5 is prepared taking 10 + 10 + 10 = 30 mins. In the case where the operator number is equal to or greater than the number of available UAVs, all cumulative setup times will reduce to the individual setup time.

Flight Time (T_f^k) The flight time for UAV k is defined as the time it takes to visit every vertex it is assigned. A mathematical expression for the flight time is presented in this subsection. The UAVs will take the minimum distance path between given vertices i and j, corresponding to the euclidean distance. As defined above, C_{ij} specifies this value. Using the speed of the kth UAV flying between these nodes (given by (S_{ij}^k) and basic kinematics, we can calculate the time taken between nodes i and j. Provided the adjacency matrix for the kth UAV (X_{ij}^k) the total flight time for the kth UAV can be calculated as shown in Equation 27.

$$T_f^k = \sum_{i=1}^{N}\sum_{j=1}^{N}\frac{C_{ij}}{S_{ij}^k}X_{ij}^k \tag{27}$$

In words, if the kth UAV is assigned to fly between i and j, the adjacency matrix (X_{ij}^k) will return 1, and the euclidian distance between these vertices (C_{ij}) will be divided by the speed of the UAV (S_{ij}^k) to yield the time taken between the vertices. If the UAV is not assigned to fly between i and j then X_{ij}^k will return 0 setting the time taken between these nodes to 0. This is done for every vertex pair until the total flight time for drone k is accumulated.

Total Mission Time (T^k) As defined previously, the total mission time for the kth UAV is the sum of its flight time and cumulative setup time. Using the mathematical expressions for T_f^k and T_s^k derived above T^k can be expressed as follows:

$$T^k = T_f^k + T_s^k = \sum_{i=1}^{N}\sum_{j=1}^{N}\frac{C_{ij}}{S_{ij}^k}X_{ij}^k + t_s\left\lceil\frac{k}{O}\right\rceil\sum_{j=1}^{N}X_{1j}^k \tag{28}$$

VRP Constraints To ensure that the optimal solution meets the objectives and requirements of the PP system defined in Section 4.3, optimisation problem is constrained. The following will present the

conventional constraints for the VRP formulation. Firstly, the UAVs are constrained so that they cannot fly longer than what their battery/payload capacity allows. Note that this maximum flight duration was defined as (t_{lim}) and it is assumed that negligible battery/tank capacity loss occur during setup. This can be expressed simply as:

$$T_f^k = \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{C_{ij}}{S_{ij}^k} X_{ij}^k \leq t_{lim} \text{ for k = 1,...M.} \tag{29}$$

This constraint might render no feasible solutions when dealing with larger/longer areas. This can be due to an insufficient number of UAVs covering the whole area or the time taken to cover a single row being more than tlim. The former is fixable by introducing more UAVs to the team. The latter -for a surveying mission- can be fixed by adjusting the sweep direction so that the number of turns taken is not optimal, but the rows are feasibly long. The row alignment cannot be changed for a spraying mission as it is required for the spray UAVs to follow the crop rows. In this case, the only solution would be to either incorporate a recharge system or break down the original ROI. This would introduce additional and complicated constraints and collision-avoidance problems. Based on the average length of a grapevine row and the average size of a vineyard in the UK, it is assumed that the size of the UAV team and t_{lim} s sufficient to cover the ROIs we will be operating on.

Following the conventional VRP formulation, [83] we need to ensure every node is visited only once by only one of the UAVs in the team. The binary value X_{ij}^k stores 1 if the kth UAV travels from i to j. For a given end vertex (j) and UAV k, if we check all potential start vertices (i = 1...N) and constrain that only one X_{ij}^k returns 1, ie. the sum of X_{ij}^k is 1, it is guaranteed that this UAV will visit each node only once. To extend it to all of the UAVs available, we repeat this operation and sum X_{ij}^k for a given j, i = 1...N and k = 1...M. This is done for every possible end node j, excluding the base vertex v_1 as all UAVs must return to it. Equation 30 below presents the mathematical formulation.

Again following the conventional formulation, a constraint is implemented so that the UAV that enters one node is the one leaving it. This is once again achieved using the adjacency matrix. Let's assume the UAV k travels from vertex i to a, and must leave a to travel to j. If this is the case, the binary value for both X_{ia}^k and X_{aj}^k must be 1. This is checked for every vertex pair and drone as in Equation 31:

$$\sum_{k=1}^{M} \sum_{i=1}^{N} X_{ij}^k = 1 \text{ for j = 2,...N.} \tag{30}$$

$$\sum_{i=1}^{N} X_{ia}^{k} - \sum_{j=1}^{N} X_{aj}^{k} = 0 \text{ for a = 1,...N and k = 1...M} \tag{31}$$

The constraints mentioned above are not enough to guarantee that UAV k visits all of its assigned vertices in one tour, which is defined as a succession of vertices without stopping at the base (Def.1). As shown in Figure 68 below the solution we are looking for is the tour $\{1, 2, 3, 5, 4, 6, 7, 9, 8\}$ whereas the solution returned might be two subtours: $\{1, 2, 3, 5, 4\} \cup \{6, 7, 9, 8\}$. To avoid this and break the subtours, many

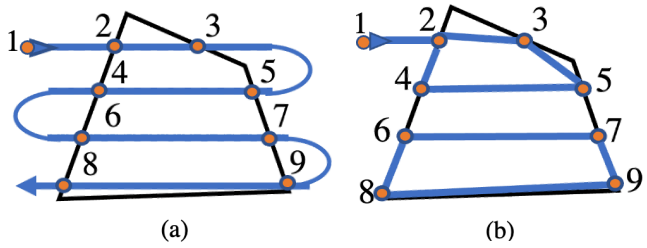


Fig. 68: (a) No Subtour Solution (b) Subtour Solution

alternative subtour elimination constraints (SEC) have been suggested. Dantzig, Fulkerson and Johnson (DFJ)[85] proposed a SEC that checks all subsets of vertices. As seen from Figure 68 if the solution includes subtours, the number of edges connecting nodes i and j, both $\in \{1, N\}$, will be equal to or more than the number of vertices. So for all the subsets of our vertices ($V_s \subset V$), excluding V itself, the number of edges must be less than the cardinality of the subset to guarantee no subtours. If V_s denotes the subset of vertices (V) this can be expressed as:

$$\sum_{i \in V_s} \sum_{j \in V_s} X_{ij}^{k} \leq |V_s| - 1 \text{ where } 2 < |V_s| < n - 1 \tag{32}$$

Note that the cardinality of V_s is bigger than 2 as at least 3 vertices are needed to create a subtour, and less than n - 1 as the original set $|V| = n$ is excluded.

Another SEC method is suggested by Miller, Tucker and Zemlin[83], which introduces a variable that denotes the order of the vertices visited, excluding the base, that is **u** $= \{u_2..u_N\}$ where **u** $\in \{1, N-1\}$. More accurately, u_i specifies the number of intermediate nodes between the base and vertex i [86].If any UAV is flying from i to j ($X_{ij} = 1$) then it must be that $u_j - u_i = 1$. If the edge i-j is not visited then ($X_{ij} = 0$) then no constraints apply. This constraint is formulated as:

$$u_i - u_j + (n-1) \sum_{k=1}^{M} X_{ij}^{k} \leq n - 2 \text{ for } i \neq j = 2..N \tag{33}$$

It can be seen that if a UAV is flying between i and j, this equation reduces to $u_i - u_j \leq -1$, which holds as $u_j - u_i = 1$ for a traveled edge. If no UAV travels between i and j then Equation 33 reduces to $u_i - u_j \leq n - 2$, which holds since the maximum value $u_i - u_j$ can take is the difference of the max. and min. of **u** which is $n - 2$. Note that this constraint also ensures that the path begins and ends at the base since it is the only vertex not restrained by this constraint.

A comparison of the accuracy for the DFJ and MTZ SEC methods show that the former gets closer to the optimum value [87]. However, practical considerations, i.e. problem complexity, memory requirement, and computation time, are more critical in the design decision for this solution. The MTZ method deals with variables and constraints of the order $O(N^2)$. The DFJ method also deals with $O(N^2)$ variables, however the constraint size is of order $O(2^N)$. The exponential growth in the constraints significantly increase the complexity of the problem, the memory requirement and the time it takes to compute a solution. Bazrafshan[87] reports that after 19 vertices a personal computer will not be able to compute an optimum value as it will run out of random access memory (RAM). In comparison to DFJ the MTZ method is easier to implement, has fewer constraints/variables and takes less time to solve. Taking into consideration this path planning algorithm will be run in-field on a laptop, the MTZ SEC formulation (Eq.33) is used for the modified VRP formulation.

Additional & System-Specific Constraints The first constraint implemented relevant to the requirements specific to this CPP is the row following. To ensure a complete map can be reconstructed for the surveying mission, and all the crop rows are coated evenly for the spraying mission, the UAVs that visit a vertex are restrained to travel to the other vertex on that row, unless they traveled from that pair vertex meaning they completed the row, or they are at the base. A typical graph structure is shown in Figure 69.We can see that the row pairs are a set of odd and even numbers. So for every graph constructed in this solution, the row pairs will be as follows: $\{(2,3),(4,5)...(N-1,N)\}$. Since we use a directed adjacency matrix, each row defined by a pair generates two edges, (i,j) and (j,i). Then to ensure each row is covered once, we need to ensure that among all the adjacency matrices of the k UAVs, either one of the edges generated by the row returns

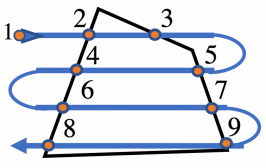


Fig. 69: Typical ROI Graph

a zero. This is formulated as shown below.

$$\sum_{k=1}^{M} X^k_{(i)(i+1)} + \sum_{k=1}^{M} X^k_{(i+1)(i)} = 1 \text{ where } i = 2, 4, 6...N \tag{34}$$

Two other constraints are implemented to restrain the solution from following the typical sweeping pattern shown in Fig.10. Once a UAV visits the secondary vertex corresponding to a row, if another row is on its designated path, it can travel to either one of the vertices defining this next row. To exemplify, after visiting the edge (2,3), the UAV can proceed to cover edges (4,5) or (5,4). In the former case, the UAV will be travelling diagonally across the field and disrupting the sweeping pattern. We need to ensure that the UAV paths avoid cutting across the field in diagonals. This can be achieved once again by utilizing the typical graph structure. The vertex the UAV must travel to after completing one row and moving on to the next depends on the parity of the vertex it is travelling from. An even-odd pair of vertices define each row, and if the UAV leaves the odd vertex to travel to another row, it must travel to the odd vertex defining that target row. Unless it is the final row, the UAV is visiting, and it is to return to the base. The same follows for leaving an even vertex. This is formulated in two constraints as shown below:

$$\sum_{k=1}^{M} X^k_{(i)(i+1)} = \sum_{k=1}^{M} \sum_{j=1,3,5..}^{N} X^k_{(i+1)(j)} \text{ where } i = 2, 4, 6...N \tag{35}$$

$$\sum_{k=1}^{M} X^k_{(i)(i-1)} = \sum_{k=1}^{M} \sum_{j=1,2,4,6..}^{N} X^k_{(i+1)(j)} \text{ where } i = 3, 5, 7...N \tag{36}$$

In the example case above the UAV traversed edge (2,3) so that $X^k_{(2)(3)} = 1$. Then according to Eq.35 $X^k_{(3)(1)} + X^k_{(3)(3)} + X^k_{(3)(5)}..X^k_{(3)(N)} = 1$. So the UAV is restrained to travel to the odd vertices, i.e. the vertices on its side of the field and avoid diagonals unless travelling to the base. Note that this will also ensure the UAVs are not taking sharp (>90°) turns between rows.

Finally, two constraints are implemented to allow the solution to consider UAV teams of varying sizes[66]. As emphasized in the cumulative setup time subsection, the size of the team given the number of operators significantly affect the total mission time. So in our solution process, any team size (m), limited by the number of available drones (M), should be considered. Using the fact that for a given solution, the team size can be determined from the number of UAVs leaving the base node, the constraints

are defined as follows:

$$\sum_{k=1}^{M}\sum_{j=1}^{N} X^k_{(1)(j)} = m \text{ and } m \leq M \tag{37}$$

This concludes the mathematical formulation of the multi-agent CPP problem as a modified VRP. The minimisation variable (total mission time), the classic VRP constraints (every node is visited once etc.), and constraints specific to this system is defined. In the next Section, these mathematical formulations will be used to implement the problem and acquire an accurate solution using Mixed-Integer Linear Programming.

Mixed-Integer Linear Programming Solution As mentioned previously, the main objective here is to minimize the time taken for the whole operation to complete and minimize the mission times for each UAV (T^k). It can be deduced that the whole operation lasts as long as the mission time of the longest operating UAV ($max\{T^k\}$). Therefore the optimization objective of the modified VRP is to minimize the maximum UAV mission time ($max\{T^k\}$). A general optimisation problem can be expressed mathematically as: minimize f(**x**) subject to $g_1\left(x_1, x_2, \ldots, x_n\right) \leq b_1) \ldots g_p\left(x_1, x_2, \ldots, x_n\right) \leq b_m$. If any of the functions $f(\mathbf{x})$ or $g_1(\mathbf{x}) \ldots g_p(\mathbf{x})$ is nonlinear the optimisation problem also is nonlinear. In our case the constraints, i.e. $g_1(\mathbf{x}) \ldots g_p(\mathbf{x})$, are linear. However the optimisation variable, i.e.maximum UAV mission time ($max\{T^k\}$), involves a maximisation function. So the objective function is: $f(\mathbf{x}) = max\{T^k\}$ which is nonlinear.

Linearisation In order to linearise this problem, a variable that represents the longest mission time ($y = max(\{T^k : k = 1..M\})$) is introduced [66]. This new variable, y, will be the new minimization variable, and a constraint will be introduced to ensure y represents the maximum of the total mission times among the drones. This constraint and the new optimization objective are expressed respectively as follows:

$$\begin{aligned} &\text{minimise } y, \\ &\text{subject to} \quad \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{C_{ij}}{V^k_{ij}} X^k_{ij} + t_s \left\lceil \frac{k}{O} \right\rceil \sum_{j=1}^{N} X^k_{1j} \leq y \text{ where } k = 1 \ldots M \end{aligned} \tag{38}$$

The decision variables, i.e. the variables that need to be identified to solve the optimization problem, are: X^k(NxNxM) matrix,**u** (N-1) column vector, m, y. Note that the team size m is restricted to be an integer. With the objective function linearised and the decision variables defined, this problem can be solved by Mixed-Integer Linear Programming.

The Optimal Solution The solution to the MILP problem mentioned above will guarantee the minimization of the operation time. However, as defined in Section4.3 this is not the only objective of the optimization. The two other objectives are to find the optimal UAV team size and the optimal path for each UAV in this optimal team. The solution for the linearised optimization problem posed above does not guarantee that these 2 objectives are fulfilled. The optimization will cease once the path for the UAV with the longest mission time is optimized. This means that fewer UAVs may cover the remaining area, or the remaining UAVs could fly shorter paths. Two strategies devised by Avellar[66] can be used to guarantee a completely optimal solution.The first strategy is a heuristic and involves incorporating either the mission time for all UAVs or the UAV team size in the objective function. Two potential modifications can be applied to the objective functions are: $f(\mathbf{x}) = y + \rho m$ or $f(\mathbf{x}) = y + \rho mean(T^k)$ where ρ is an arbitrary optimisation constant. This way, alongside finding the best y, the optimizer will also find the minimum m or minimize all paths. Note that any function of m or $mean(T^k)$ can be added to the objective function. Trial and error are required to assess the accuracy of this strategy.

Algorithm 4 Path Planning Algorithm

```
function path_optimise(available UAVs, viewpoints, setup time, operation specifications)
    Set optimisation variables
    Set objective and constraints
    Construct optimisation problem and Solve with Gurobi
    X ← stacked adjacency matrices; m ← optimal team size; v ← longest path duration
    Return X, m, v
end function
X(k) ← adjacency matrix for longest path                ▷ where k is the UAV traversing that path
while m>1 do
    Remove Viewpoints of Longest Path
    Remove UAV traversing the Longest Path and its Adjacency Matrix
    function path_optimise(reduced system and operation specifications)
        Set, construct, solve reduced optimisation problem
        Return X, m, v
    end function
    X(k) ← adjacency matrix for longest path
    m − 1
end while
```

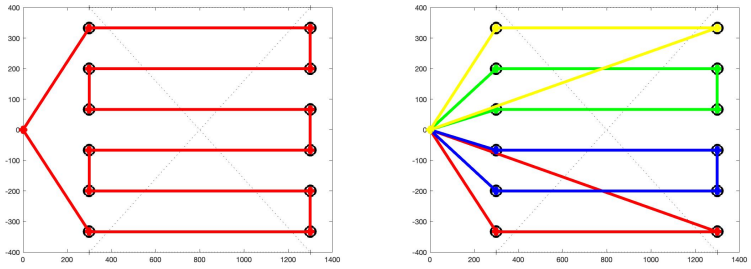
The second strategy is an iterative one that works by solving the optimization problem multiple times. The first iteration is simply running the original optimization problem on the original graph with the full set of UAVs. Then the vertices corresponding to the optimized path y (excluding the base), and the UAV flying that path are removed for the second iteration. The solution for this reduced problem will find the optimized path y for the second longest mission of the original problem. This procedure is repeated until

no UAV team size (m) reaches 1. This way, every mission path becomes the longest mission path and is optimised. This also guarantees that the optimal number of UAVs is being used.

Both strategies are implemented and have a negligible difference in computation time (discussed further in Section 4.6). The accuracy of both strategies are comparable, but the iterative strategy is more robust than the first strategy, as it doesn't involve arbitrary changes to $f(\mathbf{x})$ and ρ. Therefore the iterative strategy is used for the implementation of this CPP solution. Algorithm 4 above is the pseudocode for the overall path planning algorithm that includes the optimisation function that builds and solves the modified VRP and the iterative method. This concludes the methodologies and implementation of the algorithms concerning the PP System design. The next section aims to demonstrate the functionality of the designed system through a series of simulations.

4.6 Simulations

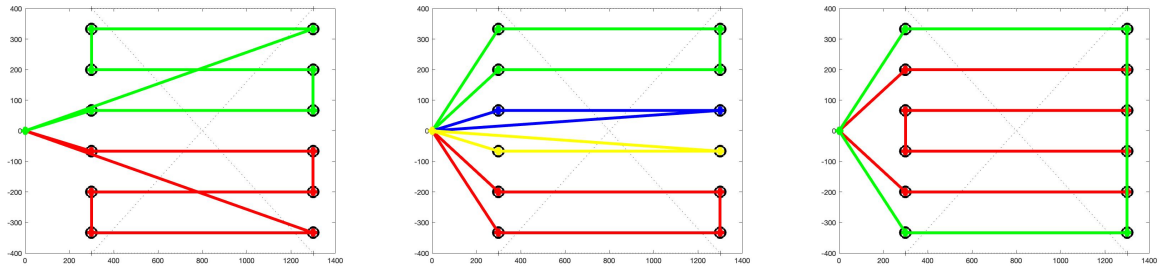
The following set of simulations of the system mainly acts as a proof-of-concept that the designed system efficiently achieves the PV system's path planning objective and requirements given the specifications and demonstrates the significance of (i) implementing practical considerations, i.e. the operator availability and setup time (t_s), (ii) system-specific additional constraints, and (iii) the two optimal solution strategies proposed in 4.5.3. The analysis will be carried out based on the effects these variables have on the optimal team size (m), generated paths, mission time (T^k) and time of computation. All trials are run with the following configuration unless stated otherwise, Mission Type: Survey; Battery Duration (t_{lim}) = 30 mins; Flight Speed: 20 m/s; Flight Altitude 180 m; ROI: Rectangular (800x1000m); Optimal Solution Strategy: Problem Iteration. The simulations are executed on MATLAB (Version: R2021a) on a MacBook Air (M1, 2020), 8 GB RAM, personal computer.



(a) (b)

Fig. 70: (a) 1-UAV (b) 4-UAV

Practical Considerations In the first set of simulations, the effect of the available operators (O), individual setup time (t_s) and the number of vertices (N) on the solution and its time of computation is demonstrated. Firstly, two cases where the cumulative setup time is not salient are demonstrated. The individual setup time (t_s) is set to 10 minutes. Figure 70(a) shows the case where there is a single UAV available (M = 1). This UAV took 20.1 minutes (including t_s) to complete the mission, and the solution was computed in 4.3 seconds. For the following figure Figure 70(b), the number of operators equals the UAV number (M = O), which is 4 in this case. The solution was computed in 128 seconds, and each UAV takes 13.7 minutes to complete the mission. As expected, the longest flight time (T_f^k) for the latter case is significantly less than the former (3.7 vs 10.1 minutes) due to task division between the UAVs.



(a) (b) (c)

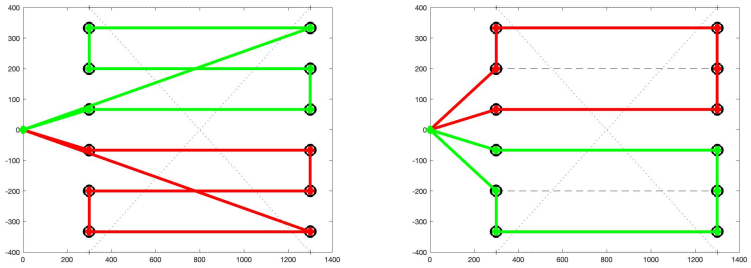
Fig. 71: (a) M = 4, O = 2 (b) Reduced t_s (c) Reduced t_s & O = 1

For the next set of example cases, the effect of varying the number of available operators (O) and t_s will be analyzed. Figure 71(a) shows the solution for the case when $M = 4$ UAVs and $O = 2$ operators are available. Individual setup time (t_s) is kept at 10 minutes. For this case only 2 UAVs are used (m = 2), and both UAVs have a mission time of 16.6 minutes and computation time is 124 seconds. Comparing this with the case in Figure 70(b), the inclusion of the cumulative setup time halved the optimal team size and consequently increased the flight time by almost twofold. When the setup time is reduced to 2 minutes, all 4 UAVs are used (m = 4), and the mission and computation time decreased to 81 seconds. The mission time for the UAVs is 7.48, 7.48, 5.92, 5.92 minutes, respectively. This solution is shown in Figure 71(b).

For the case shown in Figure 71(c), the setup time is kept at 2 minutes, but the operators are reduced to O=1. The optimal team size remains at 2 (m = 2); however, in comparison to the previous case, the mission times for the UAVs increased to 8.82 and 8.75 minutes. The computation time also increased to 109 seconds. This solution is still faster in mission and computation time when compared to the case

in Figure 71(a). Note that the paths for Figure 71(a) and Figure 71(c) are different, and the latter UAVs have a slightly longer flight time despite operating with the same m and ROI. The case for Figure 71(c) shows that more UAVs are utilized when the setup time decreases or the operator number increases. The variety in the optimal path and team size for these cases prove the importance of including the practical considerations, i.e. the available operator and cumulative setup time, in the optimization process.

Additional Constraints For the next set of simulations, the aim is to show the significance of some of the constraints. The rectangular area and initial flight speed and altitude, and the maximum flight configurations are kept the same as in the first simulations (Figure 70-71), but the UAV number is reduced to 2, and to emphasize the effect of the constraints, the setup time is set to 0. Note that omitting the setup time breaks the operator number dependency of the solution. One of the vital constraints implemented

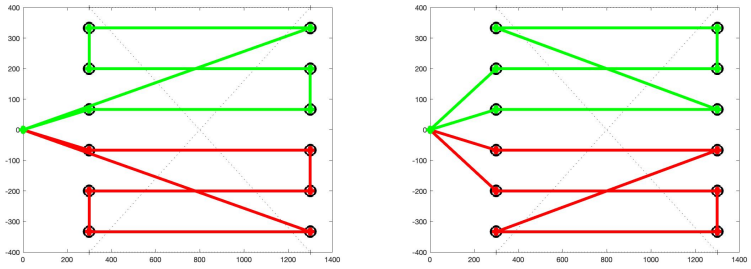


(a) (b)

Fig. 72: (a) Row Constraint Implemented b) No Row Constraint

for our optimization problem is the row pairing of the vertices, formulated in Equation 34 of Section 4.5.3. Without this additional constraint to the VRP constraints, the optimal solution is only guaranteed to visit every vertex but not completely cover the ROI. Figure 72(b) shows the case when this constraint is omitted. While all the nodes are covered as required by a VRP solution, the second and second the last rows are not traversed by any of the UAVs in operation. The mission and computation times are 4.1 minutes and 32 seconds, respectively. Complete row coverage is achieved with the row constraint implemented, as shown in Figure 72(a). As expected, the mission time increased to 6.55 minutes due to the additional rows. Although by implementing this constraint, we are compromising on the mission time minimization objective, its implementation is necessary to meet our system's main complete coverage requirement. Note that the implementation decreases the computation time to 11 seconds due to the decrease in the potential paths in the solution set.

The constraints formulated in Equations 35 and 36 of Section 4.5.3, restrains the UAVs from taking



(a) (b)

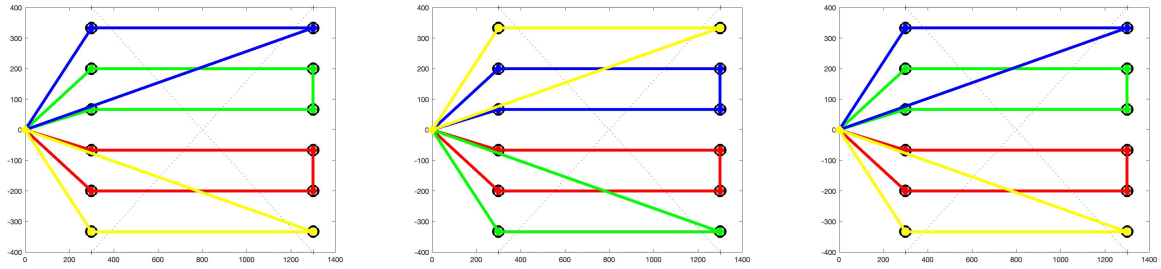
Fig. 73: (a) Diagonals Avoided b) Diagonals Allowed

diagonal paths across the ROI unless they are coming from/returning to the base. This ensures the UAVs will cover their given set of rows in sequential order. This restraint is essential for the surveying mission where the images captured need to overlap on both sides. Figure 73(b) illustrates the case when diagonals are allowed, and the constraints are not implemented. Both UAVs cover their assigned three rows by starting from the middle row, moving to one of the ends and taking a sharp turn to travel diagonally to the last row at the other end. The mission time for both UAVs is 6.44 minutes. In this case, the images acquired will not be suitable for the image stitching process.

In comparison, restraining the diagonals result in the path shown in Figure 73(a). It can be observed that both drones start the coverage from the rows at the end and follow a sequential order. The mission time for this case is 6.55 minutes, which is a minor increase. It can be concluded from these cases that while the diagonal constraint slightly increases the mission time, it is an essential constraint as it ensures the image processing requirements of the system are met. Finally, the solution set is decreased with the implementation of the constraint, so the time of computation for the constrained case is 12 seconds while the unconstrained is 22 seconds, as with the row constraint examples.

Optimal Solution Strategies The last set of simulations illustrates the significance and compares the optimal solution strategies presented in Section 4.5.3. Two strategies (modifying the objective function and iterating the optimization) are compared with each other in terms of performance and an example case where none of the strategies is implemented.

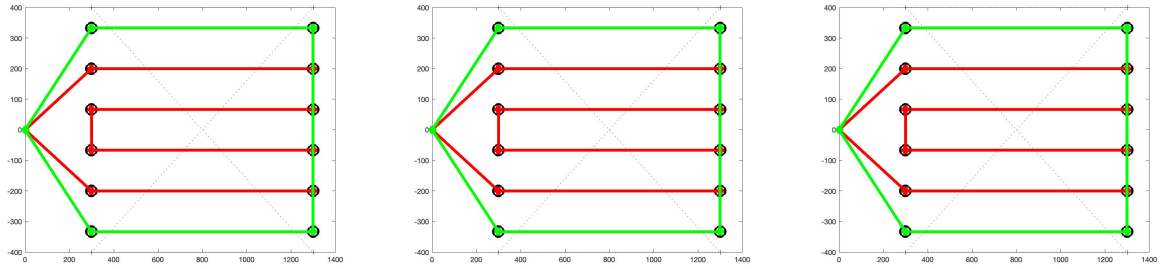
The same rectangular area with M = 4 available drones with no setup time (ts) is used for the first cases. The objective function is modified to $y+\rho mean(T^k)$ with $\rho = 10e-3$.Figure 74 shows the solutions generated when the iteration, modified objective function, and none of the strategies are implemented.



(a) (b) (c)

Fig. 74: (a) Optimisation Iteration b) Modified Objective Function (c)None Implemented

All three yield the same paths that take 3.74, 3.74, 3.72 and 3.72 minutes. All solutions were computed in similar times: 51, 49, and 49 seconds, respectively. Increasing the setup time to 2 minutes yields a similar result. Figure 75 shows the result for this case. Again, the paths generated are the same, with

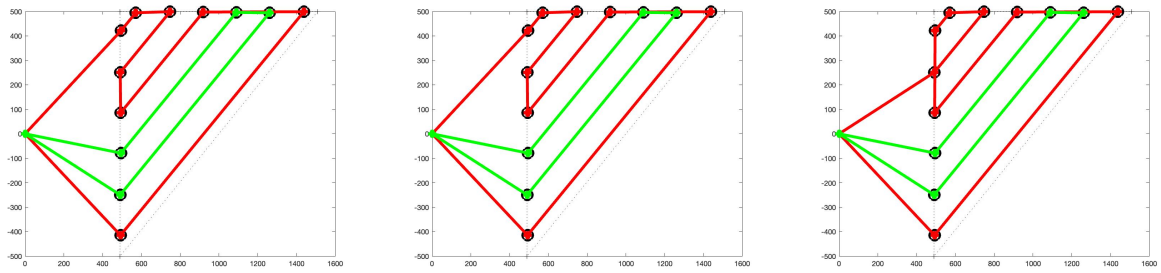


(a) (b) (c)

Fig. 75: (a) Optimisation Iteration b) Modified Objective Function (c)None Implemented

missions taking 8.75 and 8.28 minutes and the results were computed in comparable times, 110, 109 and 109 seconds, respectively. The iterative method takes the longest, although by a minimal margin, since it solves more than one optimization problem while the other two cases solve one. The time gap between the iterative and the modification strategy will increase as the number of paths and hence iterations increase. Since our system has, at most, M = 5 UAVs available per mission, we can render the difference in computation time negligible. It can be deduced from the no-strategy case generating the same solution as the others that subtours are not generally a problem for rectangular areas.

A triangular area suggested by Avellar [66] is used to illustrate the significance of this strategy. M = 2 UAVs with ts = 2 are available for this mission. Figure 76 illustrates the solutions acquired for the three cases: (a) Iterative, (b) Modified Objective, and (c) No Strategy. Following a similar pattern to the previous cases, the two strategies yield the same paths, with the mission times 8.03 and 8.16 minutes.



(a) (b) (c)

Fig. 76: (a) Optimisation Iteration b) Modified Objective Function (c)None Implemented

The time of computation for the iterative strategy is marginally higher than the modified objective strategy, 14.9 and 14.7 seconds, respectively. In comparison, the case with no strategy implemented shown in Figure 76(c) was able to optimize the longest path (shown in green) with a mission time of 8.16 minutes and therefore match (a) and (b). However, the shorter path (shown in red) was not optimized, as the optimization objective is only to minimize the longest path. So this suboptimal path contains a subtour and hence has a longer mission time of 8.12 minutes. It took 13.9 seconds to compute this incomplete solution.

These cases prove the necessity of either one of the strategies to get a completely optimal solution. While vineyards are generally rectangular, the specific surveying (Stage 2) and spraying (Stage 3) will be covering specific regions within the vineyard which are likely not be rectangular. It should be noted when $\rho = 10e - 3$ in the modified objective is decreased the strategy fails and the new solution matches the case in Figure 76(c). The ρ dependency of this strategy makes the iterative strategy the more robust and therefore it will be used for our system, despite being marginally slower.

4.7 Results & Discussions

This Path Planning section presented the design and implementation of a practical solution for performing aerial surveying and spraying using a team of UAVs for precision viticulture applications. The two-step solution design is shown to be an accurate and efficient way of finding time-efficient paths while meeting the system and mission-specific requirements of our Precision Viticulture System. It is crucial to consider the limitations of this solution. For a considerably large/long field, there might not be enough UAVs to cover the whole field, or the UAV battery/tank capacity might not be enough to last a whole row. This algorithm will render this problem infeasible and return no solution in these scenarios. A challenge of

this kind will not be encountered considering the conventional UK vineyard sizing/crop row length. If the system is to be utilised for general precision agriculture purposes, investing in more UAVs or redesigning the algorithm to allow the UAVs to recharge should be considered.

Another limitation of this algorithm is that there are no online emergency path re-planning if a UAV unexpectedly malfunctions and has to fly back to the base. In this case, the UAV is manually flown back to the base by a backup pilot. Optimally covering the remaining path of the malfunctioned UAV can be handled by deploying another UAV if available or deploying a UAV that completed its mission if the battery/payload is not limiting. This is left for future work. Note that this algorithm can be used to determine the path for the replacement UAV by defining the ROI with the remaining vertices and setting M = 1. Another area for future work is operating on 3D. This system is designed for ROIs on flat terrains, which is the convention for UK vineyards, so 2D solutions are sufficient. However, if the system was used to operate on hilltop vineyards, path planning must be redesigned to consider all three dimensions.

5 Conclusion

Surveying and managing vineyards is time-consuming, labour-intensive and is often inaccurate due to human error. The solution proposed in this report provides an effective system that tackles these common challenges. The drones are flexible, agile and unobstructed by the vegetation, making them faster at surveying compared to viticulturists. They also have the advantage of having a birds-eye-view that provides a more holistic view of the field which is otherwise inaccessible to humans. The image processing uses machine learning to accurately identify diseases of the plants while requiring minimal operator verification. This also decreases survey and disease identification time which overall makes the system more effective. The path planning algorithm optimises for team size, flight paths and resources which provides an efficient overall system with minimised time. The centralised API and database enables the optimised system to communicate and share data within itself. This is achieved through a set of predefined endpoints that are ubiquitous throughout the system. The human-robot interface allows the operator to monitor and control the system with minimal effort and cognitive load.

Both image processing and path planning work closely with the API to manipulate the data points that are used for the system to function. The image processing techniques employed create a high resolution map which successfully classifies the majority of images accurately. The path planning algorithm reaches the optimal paths for the drones given the unique set of parameters of the system. This sys-

tem's technology addresses the problems faced by viticulturists through various, optimised algorithms. This system is aimed at viticulture although due to the design of the API and system components, it can be used for a variety of other surveying and agricultural purposes. The system can be altered to analyse other crops by changing the disease detection models while path planning can remain the same for similar fields.

6 References

[1] F. J. Pierce and P. Nowak, "Aspects of precision agriculture," *Advances in agronomy*, vol. 67, pp. 1–85, 1999.

[2] A. Matese, P. Toscano, S. F. Di Gennaro, L. Genesio, F. P. Vaccari, J. Primicerio, C. Belli, A. Zaldei, R. Bianconi, and B. Gioli, "Intercomparison of uav, aircraft and satellite remote sensing platforms for precision viticulture," *Remote Sensing*, vol. 7, no. 3, pp. 2971–2990, 2015.

[3] A. Sassu, F. Gambella, L. Ghiani, L. Mercenaro, M. Caria, and A. L. Pazzona, "Advances in unmanned aerial system remote sensing for precision viticulture," *Sensors*, vol. 21, no. 3, p. 956, 2021.

[4] J. Chen, E. Haas, K. Pillalamarri, and C. Jacobson, "Human-robot interface: Issues in operator performance, interface design, and technologies," p. 100, 07 2006.

[5] L. Moreno, "Fundamentals of hierarchy in user interface design (ui)." `https://bit.ly/3yHUOc7`, 2019.

[6] J. A. Adams, "Critical considerations for human-robot interface development," 2002.

[7] M. Giuliani, C. Lenz, T. Müller, M. Rickert, and A. Knoll, "Design principles for safety in human-robot interaction," *International Journal of Social Robotics*, vol. 2, pp. 253–274, 09 2010.

[8] J. Mclurkin, J. Smith, J. Frankel, D. Sotkowitz, D. Blau, and B. Schmidt, "Speaking swarmish: Human-robot interface design for large swarms of autonomous mobile robots.," pp. 72–75, 01 2006.

[9] R. Hat, "What is a rest api?." `https://www.redhat.com/en/topics/api/what-is-a-rest-api`, 2020.

[10] S. Contributors, "Sequelize." `https://sequelize.org/`, 2022.

[11] I. D. Foundation, "The 7 factors that influence user experience." `https://www.interaction-design.org/literature/article/the-7-factors-that-influence-user-experience`, 2021.

[12] R. Panko, "Thinking is bad: Implications of human error research for spreadsheet research and practice," 02 2008.

[13] L. Taylor, "Are qr codes reliable?." `http://qrcode.meetheed.com/question18.php`, 2017.

[14] J. Abello, S. Hadlak, H. Schumann, and H.-J. Schulz, "A modular degree-of-interest specification for the visual analysis of large dynamic networks," *IEEE transactions on visualization and computer graphics*, 08 2013.

[15] Wikipedia, "Geographic coordinate system." `https://en.wikipedia.org/wiki/Geographic_coordinate_system`, 2022.

[16] Wikipedia, "Haversine formula." `https://en.wikipedia.org/wiki/Haversine_formula`, 2022.

[17] Wikipedia, "Geojson." `https://en.wikipedia.org/wiki/GeoJSON`, 2022.

[18] I. Becker-Reshef, C. Justice, M. Sullivan, E. Vermote, C. Tucker, A. Anyamba, J. Small, E. Pak, E. Masuoka, J. Schmaltz, *et al.*, "Monitoring global croplands with coarse resolution earth observations: The global agriculture monitoring (glam) project," *Remote Sensing*, vol. 2, no. 6, pp. 1589–1609, 2010.

[19] DJI, "P4 multispectral user manual v1.0." `https://dl.djicdn.com/downloads/p4-multispectral/20190927/P4_Multispectral_User_Manual_v1.0_EN.pdf`, 2019. Accessed: 21-11-2021.

[20] MATLAB, "version 9.10 (r2021a)." `https://uk.mathworks.com/`, 2021.

[21] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," in *European conference on computer vision*, pp. 404–417, Springer, 2006.

[22] G. Lowe, "Sift-the scale invariant feature transform," *Int. J*, vol. 2, no. 91-110, p. 2, 2004.

[23] P. Getreuer, "A Survey of Gaussian Convolution Algorithms," *Image Processing On Line*, vol. 3, pp. 286–310, 2013. `https://doi.org/10.5201/ipol.2013.87`.

[24] M. J. Jones, P. Viola, *et al.*, "Robust real-time object detection," in *Workshop on statistical and computational theories of vision*, vol. 266, p. 56, 2001.

[25] D. Tyagi, "Introduction to surf (speeded-up robust features)." `https://bit.ly/3KPQ4U1`. Accessed: 03-04-2022.

[26] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," *International journal of computer vision*, vol. 74, no. 1, pp. 59–73, 2007.

[27] R. C. Bolles and M. A. Fischler, "A ransac-based approach to model fitting and its application to finding cylinders in range data.," in *IJCAI*, vol. 1981, pp. 637–643, Citeseer, 1981.

[28] MATLAB, "Feature based panoramic image stitching." `https://uk.mathworks.com/help/vision/ug/feature-based-panoramic-image-stitching.html`. Accessed: 8-11-2021.

[29] R. Searle, "Aerial view of faro, portugal vineyard." https://unsplash.com/photos/fewHxUgWJ_4. Accessed: 03-04-2022.

[30] D. Shukla, "Split any image with any degree of overlap." https://github.com/Devyanshu/image-split-with-overlap. Accessed: 03-04-2022.

[31] J. Gammell, "Ndvi video of forest," 2018.

[32] DJI, "P4 multispectral." https://www.dji.com/uk/p4-multispectral. Accessed: 21-11-2021.

[33] S. M. Herrmann, A. Anyamba, and C. J. Tucker, "Recent trends in vegetation dynamics in the african sahel and their relationship to climate," *Global Environmental Change*, vol. 15, no. 4, pp. 394–404, 2005.

[34] R. Gillies, W. Kustas, and K. Humes, "A verification of the'triangle'method for obtaining surface soil water content and energy fluxes from remote measurements of the normalized difference vegetation index (ndvi) and surface e," *International journal of remote sensing*, vol. 18, no. 15, pp. 3145–3166, 1997.

[35] M. Kerkech, A. Hafiane, and R. Canals, "Deep leaning approach with colorimetric spaces and vegetation indices for vine diseases detection in uav images," *Computers and electronics in agricultur*e, vol. 155, pp. 237–243, 2018.

[36] R. Giovos, D. Tassopoulos, D. Kalivas, N. Lougkos, and A. Priovolou, "Remote sensing vegetation indices in viticulture: A critical review," *Agriculture*, vol. 11, no. 5, p. 457, 2021.

[37] J. Weier and D. Herring, "Measuring vegetation (ndvi & evi). nasa earth observatory," *Washington, DC, USA*, 2000.

[38] N. Pettorelli, J. O. Vik, A. Mysterud, J.-M. Gaillard, C. J. Tucker, and N. C. Stenseth, "Using the satellite-derived ndvi to assess ecological responses to environmental change," *Trends in ecology & evolution*, vol. 20, no. 9, pp. 503–510, 2005.

[39] S. Kumar, M. S. Röder, R. P. Singh, S. Kumar, R. Chand, A. K. Joshi, and U. Kumar, "Mapping of spot blotch disease resistance using ndvi as a substitute to visual observation in wheat (triticum aestivum l.)," *Molecular breeding*, vol. 36, no. 7, pp. 1–11, 2016.

[40] A. Wigan, "Aerial view of vineyard." https://unsplash.com/photos/_9BC1evoaps. Accessed: 11-04-2022.

[41] D. Hughes, M. Salathé, *et al.*, "An open access repository of images on plant health to enable the development of mobile disease diagnostics," *arXiv preprint arXiv:1511.08060*, 2015.

[42] G. Geetharamani and A. Pandian, "Identification of plant leaf diseases using a nine-layer deep convolutional neural network," *Computers & Electrical Engineering*, vol. 76, pp. 323–338, 2019.

[43] S. Mustacich, "A dire threat to grapevines." https://www.winespectator.com/articles/a-dire-threat-to-grapevines-51972. Accessed: 11-04-2022.

[44] M. Selim, "How to deal with 'tinder'." https://www.internationalwinechallenge.com/Canopy-Articles/how-to-deal-with-tinder.html. Accessed: 11-04-2022.

[45] U. S. E. P. AGENCY, "Efed red chapter for ziram." https://www3.epa.gov/pesticides/chem_search/cleared_reviews/csr_PC-034805_30-Oct-01_a.pdf. Accessed: 11-04-2022.

[46] R. G. Fei-Fei Li, Jiajun Wu, "Lecture notes for convolutional neural networks for visual recognition," 2022.

[47] U. Michelucci, *Advanced applied deep learning: convolutional neural networks and object detection*. Springer, 2019.

[48] T. M. I. MATLAB, "Deep learning toolbox." https://uk.mathworks.com/products/deep-learning.html. Accessed: 1-04-2022.

[49] M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: an overview," *arXiv preprint arXiv:2008.05756*, 2020.

[50] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[51] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[54] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[55] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[56] R. C. Jain, R. Kasturi, and B. G. Schunck, *Machine vision*. McGraw-Hill, 1995.

[57] S. Lee, "Lines detection with hough transform," Aug 2021.

[58] R. Almadhoun, T. Taha, L. Seneviratne, and Y. Zweiri, "A survey on multi-robot coverage path planning for model reconstruction and mapping," *SN Applied Sciences*, vol. 1, July 2019.

[59] T. Cabreira, L. Brisolara, and P. R. F. Jr., "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, vol. 3, p. 4, Jan. 2019.

[60] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, pp. 1258–1276, Dec. 2013.

[61] J.-C. Latombe, *Robot motion planning*, vol. 124. Springer Science & Business Media, 2012.

[62] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[63] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon cellular decomposition," in *Field and Service Robotics*, pp. 203–209, London: Springer London, 1998.

[64] H. Choset, "Coverage for robotics – a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1/4, pp. 113–126, 2001.

[65] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proceedings. 1985 IEEE international conference on robotics and automation*, vol. 2, pp. 116–121, IEEE, 1985.

[66] G. Avellar, G. Pereira, L. Pimenta, and P. Iscold, "Multi-UAV routing for area coverage and remote sensing with minimum time," *Sensors*, vol. 15, pp. 27783–27803, Nov. 2015.

[67] A. Zelinsky, R. A. Jarvis, J. Byrne, S. Yuta, *et al.*, "Planning paths of complete coverage of an unstructured environment by a mobile robot," in *Proceedings of international conference on advanced robotics*, vol. 13, pp. 533–538, Citeseer, 1993.

[68] V. Shivashankar, R. Jain, U. Kuter, and D. Nau, "Real-time planning for covering an initially-unknown spatial environment," in *Twenty-Fourth International FLAIRS Conference*, 2011.

[69] Y. Gabriely and E. Rimon, "Spiral-stc: An on-line coverage algorithm of grid environments by a mobile robot," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 1, pp. 954–960, IEEE, 2002.

[70] E. Gonzalez, O. Alvarez, Y. Diaz, C. Parra, and C. Bustacara, "Bsa: A complete coverage algorithm," in *proceedings of the 2005 IEEE international conference on robotics and automation*, pp. 2040–2044, IEEE, 2005.

[71] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[72] C. Luo, S. X. Yang, D. A. Stacey, and J. C. Jofriet, "A solution to vicinity problem of obstacles in complete coverage path planning," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 1, pp. 612–617, IEEE, 2002.

[73] Y. Bouzid, Y. Bestaoui, and H. Siguerdidjane, "Guidance-control system of a quadrotor for optimal coverage in cluttered environment with a limited onboard energy: Complete software," *J. Intell. Robot. Syst.*, vol. 95, pp. 707–730, Aug. 2019.

[74] E. J. Forsmo, E. I. Grøtli, T. I. Fossen, and T. A. Johansen, "Optimal search mission with unmanned aerial vehicles using mixed integer linear programming," in *2013 International conference on unmanned aircraft systems (ICUAS)*, pp. 253–259, IEEE, 2013.

[75] T. F. Coleman and Y. Zhang, "Optimization toolbox™ user's guide," 2021.

[76] J. Lofberg, "Yalmip : a toolbox for modeling and optimization in matlab," in *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, pp. 284–289, 2004.

[77] L. Gurobi Optimization, 2021.

[78] H. Azpúrua, G. M. Freitas, D. G. Macharet, and M. F. M. Campos, "Multi-robot coverage path planning using hexagonal segmentation for geophysical surveys," *Robotica*, vol. 36, pp. 1144–1166, Apr. 2018.

[79] W. Huang, "Optimal line-sweep-based decompositions for coverage algorithms," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 1, pp. 27–32 vol.1, 2001.

[80] S. K. Kurtural, "Vineyard design," tech. rep., Agriculture and Natural Resources, 2013.

[81] H. Singh and R. Sharma, "Role of adjacency matrix & adjacency list in graph theory," *International Journal of Computers & Technology*, vol. 3, pp. 179–183, 08 2012.

[82] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Manage. Sci.*, vol. 6, pp. 80–91, Oct. 1959.

[83] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *Journal of the ACM*, vol. 7, p. 326–329, oct 1960.

[84] B. Eksioglu, A. V. Vural, and A. Reisman, "The vehicle routing problem: A taxonomic review," *Computers &Industrial Engineering*, vol. 57, pp. 1472–1483, Nov. 2009.

[85] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *J. Oper. Res. Soc. Am.*, vol. 2, pp. 393–410, Nov. 1954.

[86] T. Bektaş and L. Gouveia, "Requiem for the miller–tucker–zemlin subtour elimination constraints?," *European Journal of Operational Research*, vol. 236, pp. 820–832, Aug. 2014.

[87] R. Bazrafshan, S. H. H. Zolfani, and S. M. J. M. A. e hashem, "Comparison of the sub-tour elimination methods for the asymmetric traveling salesman problem applying the SECA method," *Axioms*, vol. 10, p. 19, Feb. 2021.