Project Plan:

Abstract

List of Tables

List of Figures

1. Introduction

1.1 Overview

1.2 Report Structure

1.3 Aims and Objectives

2. Literature review

3. Design & Methodology

3.1 Overview

3.2 Functional Requirements

3.3 Non-Functional Requirements

3.4 Clustering

3.5 Path Finding

3.6 GUI

3.7 System Design

4. Implementation

4.1 GUI

4.2 Clustering

4.3 Weather Data

4.4 Flight Path Correction

4.5 Map Drawing

4.6 Database Management

4.7 Path-Finding

4.8 HTML Output

5. Testing

5.1 Acceptance Testing

5.2 Performance Testing

5.3 Path Finding Algorithms

5.4 User Testing

Abstract

Introduction

Drones are an emerging technology for delivery. Many companies are developing systems to utilise them, such as Amazon with their “Prime Air” platform, and DHL with their “Parcelcopter”. They hold many benefits over traditional road-based delivery methods, such as lower cost, faster delivery and lower environmental impact. However, there are some obstacles to overcome before they see widespread use, such as legal issues and low flight range.

As with any delivery method, it is important that drones do not waste time. Their routes need to be optimised in order to speed up delivery for customers and companies alike. Because of this, it is vital that a schedule is created and maintained that details where each drone will be going and when.

Aims and Objectives

The aim of this project is to create a system to take input of customers and their locations, and create an efficient schedule for these customers. The following objectives must be completed in order to succeed at the task:

- Implement a simple Graphic User Interface (GUI)

- Develop a system to generate routes between customers

- Present the results graphically

- Allow comparison of different algorithms

Structure of Report

Chapter 2 – Literature Review

Chapter 3 – Design and Methodology to be implemented

Chapter 4 – Implementation of said design

Chapter 5 – Testing

Chapter 6 – Evaluation

Chapter 7 – Conclusions and future work

Literature Review

bla bla bla bla bla

Design and Methodology

This chapter will discuss the requirements of the project. The requirements are a direct result of the aims and objectives, and are steps required to complete these.

The methodology taken to implement the requirements is then explored.

Requirements

The requirements have been split into categories depending on importance using the MoSCoW system. Each requirement is classed as either “Must”, “Should”, “Could”, “Won’t”. For this project, “Won’t” has not been detailed.

“Must” requirements are those that the project cannot go without.

“Should” are important requirements, but not completely essential to the project.

“Could” are additional features that would be nice to have, but are a bonus, and their omission doesn’t affect the project in any way.

Functional Requirements

The functional requirements are functions of the implemented system. These detail tasks that the completed system will complete.

Must:

- A GUI – It is crucial to make the application usable to anyone that it has a simple GUI. This GUI has several characteristics that must be implemented in order to complete this task:

- A real map of a city. It allows the user to test out and see real-world results of their use.

- Input orders from the GUI. It is no good having a GUI if the user still has to input data using some other means

- Controllable parameters. One of the main aims of the project was to allow comparison of parameters. It will help users to see how adding or taking away drones for example will affect the results.

- Division of customers into smaller groups. As discussed in the Literature Review, it is vital that customers are broken down into manageable groups. Without this feature, the software will likely take far too long to complete path-finding, will be less likely to produce a good route, and also will not create a realistic scenario of multiple drones being able to fly at once.

- Sensible routes must be created. This is perhaps the main function of the application, and without it there is little to show. Routes must be created among customer groups that are sensible and follow some kind of rules in an attempt to optimise them.

Should:

- Multiple path finding algorithms implemented. This falls under the aim of comparing input parameters. It will be interesting for users to see and compare different algorithms, however providing there are other parameters that can be controlled, and one path-finding algorithm is included, this is not vital functionality.

- Consider weather conditions. One of the big drawbacks of drones is that they are greatly affected by the weather. In order to simulate a real-world system more accurately, this should be taken into consideration. Weather data such as wind speed, direction and precipitation may all be considered.

- Interactive user input. In a real-world system, the application would be running constantly and waiting for new orders to come in. If a new order is placed, the system shouldn’t shut down while it deals with this, it should be able to change an existing route or create a new one to accommodate the new customer.

Could:

- A display of the current state of routes as they are created. This would be an interesting feature to allow the user to see how the path-finding algorithm selected works.

- When running, show the location of each drone as they move through their route. This would be a bonus feature where the system is running in real time and updates with progress of drones along their routes.

Non-Functional Requirements

Non-Functional Requirements are those that do not detail functionality of the solution. They outline things such as hardware and software environments and how the solution should be delivered.

Must:

- Use python for development. Python has been selected due to familiarity from previous projects. Python is known for being useful for implementing AI solutions, which this project aims to do. Additionally there are a vast range of libraries to aid in development.

Should:

- Completed product should ship as a .exe file. This file type will run in windows, which the majority of users around the world will be running. With a little configuration this type of file can also run on Linux and MacOS. It allows users to simple double click the file and have the program run, without worrying about installing python themselves.

- Use an appropriate API for weather data if this function is implemented. An API should be selected that allows current or at least hourly weather data to be gathered either for free or for a low cost.

- Have a fast runtime. As this software is designed to emulate a real-world system, it is vital that the time taken to create a schedule is as fast as possible. If the system takes several hours to produce a solution, it would not be viable for real world use.

Could:

- Use scikit-learn library for clustering algorithms. The library provides a range of algorithms, such as both that “kMeans” and “affinity propagation”, which are detailed in the literature review. Developing a clustering algorithm ourselves is possible, however it is not the aim of this project to do.

- Use pyeasyga for the genetic algorithm. This is another free library for python. This one implements the Genetic Algorithm. As with clustering, the task that this project is aiming to solve is not whether or not this algorithm can be implemented, thus using a library is a good time-saving method.

- Could use PyQ to produce a GUI. PyQt is a powerful tool for creating attractive GUIs. It will allow a clean and simple user interface to be developed quickly.

Software Components

GUI

The graphic user interface is a key part of this project. It serves to both allow input and show resulting output. There are many options for creating a GUI using Python, however having reviewed several I felt that PyQt5 was the best fit for the job.

PyQt5

Qt is a free, open-source toolkit for creating GUIs. It is largely cross-platform, meaning it will run on Windows, Linux and macOS with no custom code required per operating system. It sees widespread use by many large companies, such as Sky, Mercedes-Benz and LG. Many popular applications are built using Qt, such as Google Earth, Adobe Photoshop Elements and VirtualBox. They have a library available for Python that enables attractive GUIs to be built quickly, this has been used to develop the application.

The library allows simple integration between interface and program functions. There is a wealth of documentation for PyQt5, many up to date tutorials and guides for assisting in production of the application. The code itself is simple to learn and write which will help speed up development.

Some alternative libraries for creating the GUI were considered, however they were found to be either too basic or too complex for the scope of this project. PyQt5 allows us to design the GUI to the exact standard we need for this project.

Clustering

Clustering is the second main section of the project. The clustering algorithm is responsible for breakdown down the overall problem into manageable chunks.

Scikit-learn kMeans

Scikit-learn is a free machine learning library for Python. It has many different algorithms implemented that we are free to pick up and use. In particular we are interested in the kMeans class. This is an implementation of the kMeans clustering algorithm, which we have chosen to use for grouping our customers by location. Using just a few lines of code we are able to sort our customers based on their location into however many groups we desire.

Currently the clusters are designed to each contain as long a route as possible. This allows drones to use as much battery capacity is possible before returning back to the depot.

It also gives us the ability to input a new customer and discover which existing cluster they are closest to, which is a useful way of saving processing power and time when we have already calculated the clusters.

Weather Data

I wanted to include weather data in my solution to better emulate a real life scenario. For this I wanted to get the current wind speed and bearing. Darksky.net offer a free API to provide such data. A request is made to the API to that returns a set of data based on a latitude longitude pair, in this case for Aberdeen. From here we are able to only select the data we need and use this to calculate the correct flight path and speed of each drone along each leg of each route. This helps us to realistically report resulting routes to the user. It also enables our search algorithms to better optimise since they can attempt to find routes that have a tailwind more often than a headwind.

Path Finding

Path finding is the final major step in our project. Once the data has been broken into clusters, we can attempt to find a route through the customers in each cluster. Weather data is taken into account while creating routes. The effect of wind speed and bearing on flight time is used to optimise the routes and decrease the time taken.

E6B

The data available to us is fairly limited. Even using a weather API we know the wind speed and bearing, and the drone speed and desired bearing. I attempted to use this data in vector calculations in order to calculate the actual direction a drone must fly, and how fast it will be going, however I was missing one data point for each calculation I attempted.

Upon further research I discovered the E6B, a device created in the late 1930s to calculate flight path and speeds. Using the E6B, we can first calculate the angle that the drone must fly in to reach the target. Once we have this angle, we use it to calculate the actual resulting speed of the drone when the wind is considered.

From here it is a simple matter of performing a speed distance time calculation to give us the time taken to complete each leg of the journey.

Greedy Best First

Greedy best first is a fairly straightforward algorithm. It was expected that we would be able to find a library to use to save time implementing this, however none was found. Because of this, I designed and implemented this myself in Python.

The basic flow of the algorithm is to calculate the time taken to get from the current location to each other location that is unvisited and pick the one with the smallest corresponding time. The process repeats until there are no more unvisited locations, at which point it returns home. Due to this simplicity, the code was written reasonably quickly and tested to confirm it works as expected.

Genetic Algorithm

A library was thankfully available for the genetic algorithm, as this algorithm is more complex and would have taken longer to develop and test than greedy best first. The library is “pyeasyga” and as with scikit-learn, it allowed me to rapidly implement the genetic algorithm.

The code that was written simply had to outline how the algorithm should behave at each stage:

- Creation of individuals is done by taking the latitude and longitude pairs of customers in each cluster and randomly assigning them a position in a list.

- Crossover is done by randomly selecting a position and swapping data from either side of two individuals in order to create two new children.

- Mutation randomly selects two customers in a route and swaps their position in the route. For example if the list goes 1 – 2 – 3 – 4 – 5 and customer 2 and 4 are selected, the resulting route would be 1 – 4 – 3 – 2 – 5.

- Selection is done by taking a random sample of 10% of the population. The individuals in the sample are ranked according to the fitness function and the best individual is selected.

- The fitness function calculates the total length of the route and from there the total time taken for a drone to complete the route. The time is the fitness that is returned. Notably, time is used for this parameter rather than distance. This is the case since wind is taken into consideration. Drones have a amount of time they can fly, however the distance is in part dictated by external factors such as the wind speed and bearing relative to the drone.

HTML Output

There are two parts to the output, visualising the data input and output on a map, and modifying the HTML file containing the map. These tasks are completed with Folium and BeautifulSoup, two libraries freely available for Python.

Folium

The drawing of the map is done with Folium. Folium allows data to be drawn on a real map. In our case we can have latitude longitude pairs with associated item names. This data can be used to place markers on the map at those exact locations. Once routes have been calculated we are able to draw lines on the map to represent the path of each drone. The map is saved as a regular HTML file.

BeautifulSoup

BeautifulSoup is used to easily modify the HTML file. As we want to be able to see a quick summary of various parameters and the resulting output, an external file is preferable to having the user read data which has just been printed to the command line or a text file. With BeautifulSoup we are able to read the HTML file generated by Folium and modify the body of the file to include data from elsewhere in the application, such as the number of drones and the number of routes required.

Architecture

These components are brought together to create a simple workflow.

