UNIVERSIDAD DE PLAYA ANCHA

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE COMPUTACIÓN E INFORMÁTICA

CARRERA DE INGENIERÍA INFORMÁTICA

OPTIMIZACIÓN DE RUTA EN DRONE DJI.

Proyecto de Título II para optar al Título de Ingeniero Informático con mención en gestión de la información y al grado de Licenciado en Ciencias de la Ingeniería.

RODRIGO ANTONIO GUZMÁN CASTRO

Profesor Guía: Dr. Franklin Johnson Parejas

Valparaíso, Chile

2019

Contenido

FIGURAS

RESUMEN

El presente proyecto se realizó en los Laboratorios de Informática de la Universidad de Playa Ancha con el fin de encontrar una forma óptima de realizar misiones de vuelos mediante puntos geolocalizados en un Drone.

A través de investigaciones, se determinó que utilizar tecnología computacional es una de las formas más adecuadas de trabajar esta problemática. Con este tipo de tecnología se puede desarrollar una aplicación móvil que permite calcular una ruta más corta.

Estudiando el problema, se detectó que una de las formas para calcular mejores caminos es utilizar algoritmos de optimización en grafos, para los cuales existe una categoría que resuelve el problema llamada metaheurísticas en esta investigación nos centraremos en la metaheurística llamada “optimización de colonias de hormiga” también conocido como ACO (Ant Colony Optimization).

El objetivo de esta tesis es analizar, desarrollar e implementar una aplicación a través de las tecnologías móviles e informáticas, específicamente utilizando algoritmos metaheurísticos y lenguajes de programación como JAVA para lograr encontrar los caminos más cortos de las rutas seleccionadas por el usuario.

Finalmente, de los resultados obtenidos y expuestos, se llega a la conclusión de que la implementación de estos algoritmos de optimización sería una solución para la problemática presentada, además de reducir considerablemente el tiempo de vuelo y el gasto de batería necesitado en un Drone.

ABSTRACT

This project was carried out in the Computing Laboratories of the University of Playa Ancha in order to find an optimal way to perform flight missions with geolocated points using a Drone.

Through investigations, it was detected that using computer technology is one of the ways to work on this problem. With this type of technology you can develop a mobile application that allows you to calculate a shorter route.

Studying the problem, it was detected that one of the ways to calculate better paths is to use optimization algorithms in graphs, for which there is a category that solves the problem called metaheuristics in this research. We will focus on the metaheuristics called "optimization of ant colonies." also known as ACO (Ant Colony Optimization).

The objective of this thesis is to analyze, develop and implement an application through mobile and computer technologies, specifically using metaheuristic algorithms and programming languages such as JAVA to find the shortest routes of the all the ones selected by the user.

Finally, from the results obtained and shown, it is concluded that the implementation of this optimization would be a solution for this specific problem, as well as considerably reducing the flight time and battery expenditure needed in a Drone.

CAPÍTULO I: INTRODUCCIÓN.

En el presente proyecto se realizó un estudio de los procesos que involucra una misión de vuelo, basada en la optimización de rutas mediante puntos con un Drone, con el fin de darle una solución mediante la tecnología informática y específicamente mediante metaheurísticas.

Este proceso utiliza una aplicación existente correspondiente a GSDemo que ofrece un mapa y la capacidad de enviar una aeronave a recorrer lugares asignados en un mapa geo localizado.

Dentro de las metaheurísticas existentes se tomó la decisión de utilizar la metaheurística bio-inspirada ACO la cual provee una solución satisfactoria al problema que se plantea y logra reducir los tiempos de viaje, distancias y esto conlleva a recorrer más puntos con el mismo uso de batería.

1.1 IDENTIFICACIÓN DEL PROBLEMA.

Actualmente, en las aplicaciones que existen para el público abierto un usuario puede hacer que el Drone viaje a los puntos asignados de una misión en el orden que él pensó que sería el más adecuado.

Para este proyecto se llegó al acuerdo de utilizar un algoritmo que pudiera tomar puntos geolocalizados y calcular sus distancias, para luego optimizar el recorrido y redefinir el orden en el cual se visitan dichos puntos.

Los programas compatibles con el Drone presentan un mecanismo manual de ruteo que no tiene ningún tipo de ajuste, por eso es necesario agregar un tipo de algoritmo que optimice el rendimiento del Drone de manera significativa para reducir el consumo de batería y reducir el tiempo de vuelo en que llega a destino.

Las misiones son uno de los procesos más importantes que podemos asignarle a un Drone, dado que si necesitamos viajes rápidos entre muchos puntos. Para un humano no será posible decidir rápidamente la ruta más óptima en distancia para la misión especificada, si es que esta cuenta con muchos puntos.

1.2 JUSTIFICACIÓN.

El procedimiento que se lleva a cabo para comenzar una misión en las aplicaciones se realiza de forma manual a través de la elección de puntos. Este método es una forma muy deficiente para recorrer muchos destinos, por lo que es necesario aplicar una optimización.

Las misiones pueden ser utilizadas para distintos propósitos, tales como películas, fotos, búsqueda de personas y en algunos casos, entrega de productos en los cuales si deseamos recorrer muchos lugares, decidir de manera manual no es muy recomendado.

En el caso de que un usuario esté interesado en realizar una misión que involucre recorrer 40 puntos o más, rápidamente se dará cuenta que la ruta que escogió no será optima y la aeronave perderá tiempo entre los caminos.

Por lo tanto, la propuesta es implementar un módulo capaz de mejorar la ruta seleccionada por el usuario utilizando la tecnología metaheurística como solución al cálculo de ruta.

1.3 ALCANCE.

El proyecto está basado en la aplicación GSDemo, por lo que los procesos de análisis, diseño, desarrollo e implementación están basados en esta aplicación Demo, para luego ser expandido a los demás Drones DJI si se desea.

Por lo que se pretende abordar los siguientes puntos:

- Investigar de qué manera se puede obtener un recorrido optimizado utilizando tecnología Informática.
- Facilitar la manera de recorrer la mayor cantidad de puntos en el menor tiempo posible que se pueda calcular.
- Poder obtener la ruta más corta de una cantidad de puntos finitos y recorrer todos ellos con un Drone DJI.
- Una vez obtenida una ruta más corta realizar vuelos de prueba y reales.

1.4 OBJETIVOS.

A continuación, se presenta el objetivo general del proyecto, el cual está acompañado de los objetivos específicos.

1.4.1 OBJETIVO GENERAL

- Desarrollar una aplicación capaz de optimizar los recorridos en la misión de un drone mediante metaheurística ACO.

1.4.2 OBJETIVOS ESPECÍFICOS

- Realizar un estudio sobre las tecnologías metaheurísticas que puedan ser aplicadas a la obtención de una ruta optimizada.

- Desarrollar un módulo que sea capaz de calcular una ruta utilizando ACO mediante los puntos asignados en la aplicación GSDemo.

- Una vez obtenida una ruta optimizada, realizar vuelos de simulación y proceder a realizar vuelos de manera real.

1.5 ESTRUCTURA DEL TRABAJO.

El presente trabajo cuenta con una estructura en dos partes. La primera parte consta de todo el estudio que se realizó a como investigación de factibilidad y la segunda parte engloba todas las herramientas tanto de software, como de hardware que se utilizaron para implementar la optimización informática.

En el Capítulo II: ANÁLISIS Y SOLUCIÓN DEL PROBLEMA, se analiza más a fondo el proceso de una misión, y se explica de forma general la manera en que se le dio solución a este problema.

En el Capítulo III: CONTEXTO TECNOLÓGICO, se analizan todas las tecnologías utilizadas para el desarrollo de esta aplicación y como se integran cada una de ellas.

En el Capítulo IV: ANÁLISIS Y DISEÑO DEL SISTEMA, se analiza la metodología a implementar para el desarrollo de la aplicación y todos los diseños que se realizaron para el proyecto, además de explicar las funciones más importantes del desarrollo.

En el Capítulo V: DESCRIPCIÓN DE LA APLICACIÓN, se describe las características que tiene esta aplicación a través de imágenes representativas de esta.

CAPÍTULO II: ANÁLISIS Y SOLUCIÓN DEL PROBLEMA.

Antes de introducirse a la tecnología metaheurística, primero se tuvo que hacer un estudio al funcionamiento del Drone, con el fin de comprender el actual de las misiones que realiza.

2.1 INTRODUCCIÓN A LA APLICACIÓN.

Para el diseño de este trabajo, primero se tuvo que analizar los procesos que se realizan y el funcionamiento de ellos.

Como se ve en la Ilustración 1, esta es la forma en que el usuario elige el recorrido que toma el Drone mediante el marcado de puntos, a este recorrido le falta una intervención para ser óptimo debido a que el ser humano no posee la capacidad de optimizar las rutas de manera automática.

Ilustración 1: Mapeo de puntos, extraída de GSDemo.

- En primer lugar, el control del Drone debe estar conectado al dispositivo móvil Android.
- Luego se debe encender el Drone para que se vincule con el control.
- Con el Drone conectado se mostrara el estado de la aeronave y el nombre del producto. Si todo está en orden el botón “OPEN” se pondrá azul y se podrá acceder a la aplicación
- El usuario presiona “ADD” y selecciona en el mapa los puntos a los que desea que el Drone viaje.
- Se presiona el botón “CONFIG”. Ver Ilustración 2
- Se selecciona la altitud, velocidad, acción al terminar, donde comenzar y se presiona “FINISH” al completar los datos
- Se presiona “UPLOAD” y si la misión es válida se recibirá un mensaje en pantalla confirmando la misión.
- Se presiona “START” y comenzará la misión.

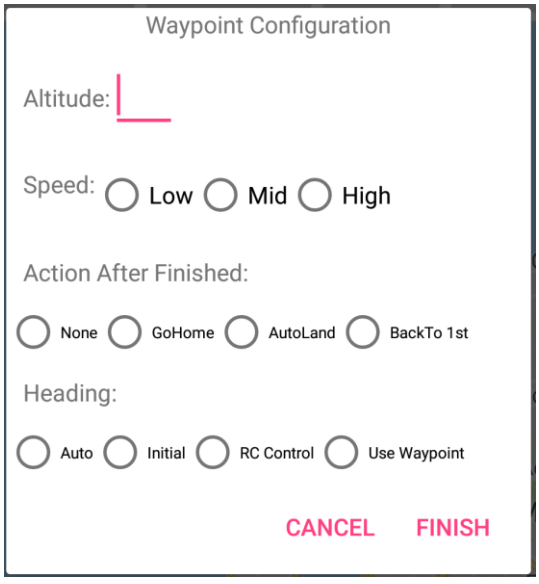

Ilustración 2: Vista "CONFIG", extraída de GSDemo

2.2 SOLUCIÓN GENERAL IMPLEMENTADA.

Este problema se solucionó mediante la tecnología metaheurística ACO. A continuación, se explica cómo funciona el módulo a grandes rasgos, explicado mediante diagrama.

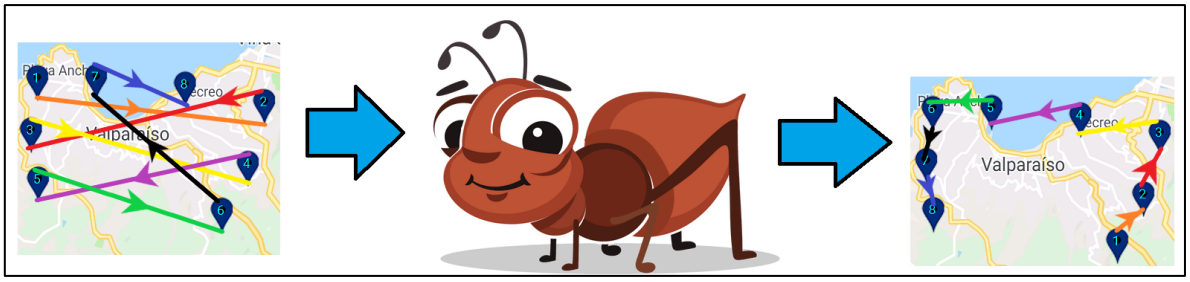

Ilustración 3: Funcionamiento del módulo, elaboración propia.

La Ilustración 3, representa el funcionamiento general del módulo, el cual consta de una optimización ACO, la cual busca una manera de recorrer los puntos seleccionados en busca de un viaje más corto.

- Como primera instancia se utilizan puntos sin optimizar.
- Una vez capturados los puntos, mediante un algoritmo informático, se procede a calcular la ruta más corta.
- Una vez terminado el proceso de cálculo, la aplicación entrega un mensaje de confirmación si se desea cambiar la ruta sin optimización.
- Al seleccionar Si, se cambia la ruta anteriormente y se utiliza el nuevo orden.

CAPÍTULO III: CONTEXTO TECNOLÓGICO.

Mediante un algoritmo informático se hizo posible la solución a esta problemática. A continuación, se describe brevemente las herramientas utilizadas para desarrollar este programa.

3.1 SISTEMAS INFORMÁTICOS Y TECNOLOGÍAS.

Tal como se especificó en los capítulos previos, la aplicación posee un proceso fundamental el cual consiste en las misiones, en la cual podemos recorrer muchos lugares en un solo viaje. Gracias a los sistemas informáticos es posible generar herramientas que gestionen diferentes usuarios y procesos de manera segura y eficaz.

Un sistema informático como todo sistema, es el conjunto de partes interrelacionadas, hardware, software y de recurso humano que permite almacenar y procesar información. El hardware incluye computadoras o cualquier tipo de dispositivo electrónico inteligente, que consisten en procesadores, memoria, sistemas de almacenamiento externo, etc. El software incluye al sistema operativo, firmware y aplicaciones, siendo especialmente importante los sistemas de gestión de bases de datos. Por último el soporte humano incluye al personal técnico que crean y mantienen el sistema (analistas, programadores, operarios, etc.) y a los usuarios que lo utilizan [1].

En los últimos años los sistemas informáticos se han hecho más presente, cada vez más las instituciones y empresas requieren la necesidad de estos sistemas, ya que estas manejan información valiosa, en el cual un sistema informático puede reservar de ella.

A continuación, analizaremos todas las tecnologías que se utilizaron para el desarrollo de este proyecto, y como primera instancia se explicara la manera de conseguir una ruta optimizada.

3.2 EL VENDEDOR VIAJERO.

Investigando el problema del vendedor viajero, podemos notar una similitud entre la problemática de este proyecto con el vendedor por lo cual explicaremos a fondo en que consiste.

¿Qué es el Problema del Vendedor Viajero?

El problema de los vendedor viajero es un problema en la teoría de grafos que requiere el ciclo de grafos más eficiente (es decir, la menor distancia total) que un vendedor puede tomar a través de cada una de las n ciudades. No se conoce ningún método general de solución, y el problema es NP-completo [2].

Dado un conjunto finito de ciudades, y costos de viaje entre todos los pares, visitar todas las ciudades exactamente una vez a costo mínimo.

Dentro de la **Teoría de la Complejidad**, que es la rama de la **Teoría de la Computación** que estudia, de manera teórica, la complejidad inherente a la resolución de un problema computable, se le cataloga como un problema **NP-completo**, lo que supone que los recursos computacionales necesarios para encontrar una solución óptima crecen de forma exponencial con la entrada del problema.

Si subimos hasta 10 el número de ciudades de la red, entonces las posibles rutas se disparan hasta más de tres millones (10!=3.628.80010!=3.628.800).

Por ello, intentar resolver el TSP en redes relativamente simples mediante el método de generación y comparación de todas las rutas es absolutamente inabordable mediante los medios computacionales disponibles actualmente, lo que hace que sea necesario utilizar otros procedimientos que, aunque no obtengan la solución óptima, sí proporcionen una respuesta aproximada lo suficientemente optimizada en un tiempo razonablemente bajo.

Específicamente en el caso de **n** ciudades se define las variables de decisión de la siguiente forma:

$$x_{ij} = \begin{cases} 1\ si\ se\ llega\ a\ la\ ciudad\ i\ a\ la\ ciudad\ j \\ 0\ en\ cualquier\ otro\ caso \end{cases}$$

Ecuación 1: variables de decisión.

Sea d_{ij} la distancia de la ciudad *i* a la ciudad *j*, donde $d_{ij} = \infty$, el modelo del agente o vendedor viajero corresponde a:

Minimizar

$$\sum_{i=0}^{n}\sum_{j=0}^{n} d_{ij}x_{ij}$$

sujeto a

$$\sum_{j=0}^{n} x_{ij} = 1\ , \qquad i = 1,2,\ldots.,n \quad (1)$$

$$\sum_{i=0}^{n} x_{ij} = 1\ , \qquad j = 1,2,\ldots.,n \quad (2)$$

La solución forma un circuito (3)

Ecuación 2: modelo TSP.

3.3 METAHEURISTICA.

El término metaheurística se obtiene de anteponer a heurística el sufijo “meta” que significa “más allá” o “a un nivel superior”

Los conceptos actuales de lo que es una metaheurística están basados en las diferentes interpretaciones de lo que es una forma inteligente de resolver un problema.

Las metaheurísticas son: estrategias generales de diseño de procedimientos heurísticos para la resolución de problemas con un alto rendimiento [3].

Con las estrategias metaheurísticas podemos obtener una solución, al problema del vendedor viajero lo cual satisface las necesidades de este proyecto.

Investigaremos que soluciones existen clasificadas como metaheurísticas.

3.3.1 METAHEURISTICAS BIO-INSPIRADAS.

Una clase particular de las metaheurísticas es la “metaheurística bio-inspirada”, esto debido a que sus reglas para elegir soluciones se basan en conceptos biológicos.

Entre las metaheurísticas bio-inspiradas se encuentran:

- Algoritmos evolutivos:

  Los algoritmos evolutivos son procedimientos de optimización que generalmente se basan en una población de ideas en conflicto para resolver un problema en particular. Las ideas compiten por la supervivencia, lo que brinda la oportunidad de generar nuevas ideas, que a su vez compiten por la supervivencia. Por lo tanto, la idea de la variación iterada y la selección que es común a los procesos evolutivos se modela en un algoritmo y se usa para mejorar iterativamente la calidad de las soluciones [4].

Un ejemplo de algoritmo evolutivo corresponde a:

El algoritmo genético es una técnica de búsqueda basada en la teoría de la evolución de Darwin, que ha cobrado tremenda popularidad en todo el mundo durante los últimos años. Se presentarán aquí los conceptos básicos que se requieren para abordarla, así como unos sencillos ejemplos que permitan a los lectores comprender cómo aplicarla al problema de su elección.

- Cúmulos de partículas:

  La optimización de enjambre de partículas (PSO) es un enfoque estocástico basado en la población para resolver problemas de optimización continua y discreta.

  En la optimización de enjambres de partículas, los agentes de software simples, llamados partículas, se mueven en el espacio de búsqueda de un problema de optimización. La posición de una partícula representa una solución candidata al problema de optimización en cuestión. Cada partícula busca mejores posiciones en el espacio de búsqueda al cambiar su velocidad de acuerdo con las reglas inspiradas originalmente en los modelos de comportamiento de aves en bandada.

  La optimización de enjambre de partículas pertenece a la clase de técnicas de inteligencia de enjambre que se utilizan para resolver problemas de optimización [5].

- Sistemas inmunes artificiales:

  Los sistemas inmunológicos artificiales están inspirados en las características del sistema inmunitario de los mamíferos y utilizan la memoria y el aprendizaje como un enfoque novedoso para resolver problemas. La idea fue propuesta por primera vez por Farmer et al. en 1986, con algunos trabajos importantes sobre redes inmunes realizados por

Bersini y Varela en 1990. Estos sistemas adaptativos son muy prometedores. Se han desarrollado muchas variantes en las últimas dos décadas, incluyendo el algoritmo de selección clonal, el algoritmo de selección negativa, las redes inmunes y otros [6].

- Colonia de hormigas:

  La optimización de colonias de hormigas estudia sistemas artificiales que utilizan el comportamiento de las colonias de hormigas existentes para solucionar problemas de optimización discretos. En el año 1999 Dorigo, Di Caro y Gambardella definieron la optimización metaheuristica de colonias de hormigas

  El primer sistema ACO fue introducido por Marco Dorigo en su Ph.D. Tesis (1992), y fue llamado Ant System (AS). AS es el resultado de una investigación sobre enfoques de inteligencia computacional para la optimización combinatoria que Dorigo realizó en el Politecnico di Milano en colaboración con Alberto Colorni y Vittorio Maniezzo. AS se aplicó inicialmente al problema del vendedor ambulante y al problema de asignación cuadrática [7].

De acuerdo al estudio realizado todas estas metaheurísticas son viables al momento de considerar una optimización dada esta situación se eligió ACO por poseer conocimientos previos de cómo funcionaba esta metaheurísticas explicada anteriormente a lo largo del aprendizaje académico. Por lo tanto explicaremos ACO a fondo con el objetivo de comprender su funcionalidad.

3.3.2 COMPORTAMIENTO COLECTIVO DE HORMIGAS.

Se debe recordar que las hormigas son prácticamente ciegas, y, sin embargo, moviéndose prácticamente al azar, acaban encontrando el camino más corto desde su nido hasta la fuente de alimentos (y regresar). Es importante hacer algunas consideraciones:

1. Por una parte, una sola hormiga no es capaz de realizar la labor anterior, sino que termina siendo un **resultado del hormiguero completo**.

2. No lo hacen sin "instrumentos", sino que una hormiga, cuando se mueve, deja una señal química en el suelo, depositando una sustancia denominada **feromona**, para que las demás puedan seguirla.

Los siguientes pasos explican, de forma intuitiva, porqué la forma de proceder de las hormigas hace aparecer caminos de distancia mínima entre los nidos y las fuentes de comida:

1. Una hormiga (**exploradora**) se mueve de manera aleatoria alrededor de la colonia.
2. Si esta encuentra una fuente de comida, retorna a la colonia de manera más o menos directa, dejando tras sí un **rastro de feromonas**.
3. Estas feromonas son atractivas, las hormigas más cercanas se verán atraídas por ellas y seguirán su pista de manera más o menos directa (lo que quiere decir que a veces pueden dejar el rastro), que les lleva a la fuente de comida encontrada por la exploradora.
4. Al regresar a la colonia con alimentos estas hormigas depositan más feromonas, por lo que fortalecen las rutas de conexión.

5. Si existen dos rutas para llegar a la misma fuente de alimentos, en una misma cantidad de tiempo, la ruta más corta será recorrida por más hormigas que la ruta más larga.
6. En consecuencia, la ruta más corta aumentará en mayor proporción la cantidad de feromonas depositadas y será más atractiva para las siguientes hormigas.
7. La ruta más larga irá desapareciendo debido a que las feromonas son volátiles (**evaporación**).
8. Finalmente, todas las hormigas habrán determinado y escogido el **camino más corto**.

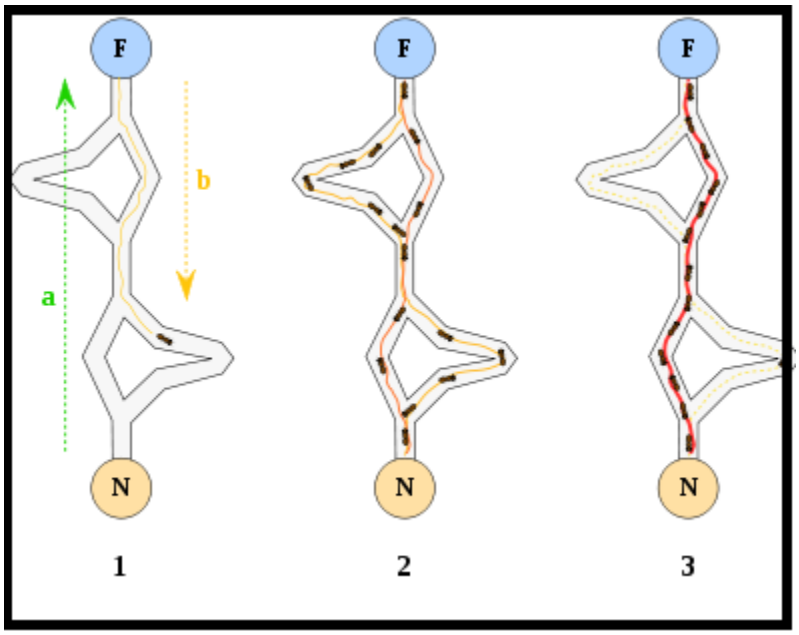

Ilustración 4: Camino con más feromona.

3.3.3 ALGORITMO DE OPTIMIZACION DE COLONIAS DE HORMIGAS.

Inspirada en el **comportamiento colectivo** de las hormigas en su búsqueda de alimentos. Veamos cómo utilizar estas características comunicativas de las colonias de hormigas para resolver un problema computacionalmente duro como es el TSP:

Denotaremos estas ciudades por $\{C0, \dots, CN-1\}$, y denotaremos por d_{ij} la distancia (el coste de la arista) entre las ciudades Ci y Cj. De forma explícita, las ciudades pueden verse como puntos de un espacio de 2 dimensiones, y la distancia entre ellas se calcula por medio de la distancia entre 2 puntos habitual (entre las ciudades $Ci = (x_i, y_i)$ y $Cj = (x_j, y_j)$) el resultado será:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Ecuación 3: distancia entre 2 puntos.

Para cada hormiga, la transición de la ciudad i a la ciudad j en una iteración del algoritmo depende de:

1. **Si la ciudad ha sido ya visitada, o no, en el ciclo que está construyendo ($\boldsymbol{Ji}$):** Cada hormiga mantiene en memoria las ciudades que ya ha visitado en el recorrido actual, y únicamente considera en cada paso las ciudades que no ha visitado todavía, que denotaremos por $\boldsymbol{J}i$. De esta forma, aseguramos que al final la hormiga ha construido un recorrido válido. (Este paso puede traer complicaciones si no todas las conexiones entre ciudades están permitidas, ya que es probable comenzar a generar un recorrido que no tiene posibilidades de ser completado; en este caso, una solución podría ser, por ejemplo, que la hormiga anula el recorrido que está construyendo y comienza un nuevo).

2. **La inversa de la distancia a dicha ciudad, $\boldsymbol{V_{ij} = 1/d_{ij}}$, que es lo que se llama *visibilidad:*** Esta medida es una información local que mide, de alguna forma, la bondad de escoger Cj estando en la Ci, y puede ser usada por las hormigas para que la distancia entre ciudades consecutivas sea una característica que intervenga en la selección del recorrido que está construyendo. Normalmente, esta información suele ser estática, ya que las distancias de las ciudades son invariables a lo largo de la ejecución del algoritmo, pero es fácil imaginar escenarios en los que los costes de paso de un

nodo a otro del grafo sean cambiantes y el algoritmo podría aplicarse para ir obteniendo buenas soluciones en este entorno dinámico.

3. **La cantidad de feromona que hay depositada en la arista que une ambos nodos, que denotaremos por** $T_{ij}(t)$: Esta cantidad se actualiza en cada paso, dependiendo de la cantidad de hormigas que han pasado por ella y de que el recorrido final de las hormigas que han usado esta conexión haya sido bueno (en relación con los demás). De alguna forma, mide la **inteligencia colectiva del hormiguero**, ya que es información que depende del conjunto de hormigas que están ejecutando el algoritmo. A diferencia de la visibilidad, esta medida proporciona una información más global, ya que la feromona que tiene una arista indica lo buena que es esa arista en conjunción con otras para dar una buena solución.

Una vez consideradas las condiciones anteriores, la **probabilidad de que la hormiga K vaya de** Ci **a** Cj **en la construcción del recorrido actual**, viene dada por una expresión del tipo siguiente:

$$p_{ij}^{k}(t) = \frac{\left[T_{ij}(t)\right]^{\alpha}\left[V_{ij}\right]^{\beta}}{\sum_{l \in J_i^k}[T_{il}(t)]^{\alpha}[V_{il}]^{\beta}}, si\ j\ \in\ J_i^k$$

$$p_{ij}^{k}(t) = 0\ , si\ j \notin J_i^k$$

Ecuación 4: Formula ACO.

Donde α y β son dos parámetros ajustables que controlan el peso relativo de cada una de las medidas en la heurística resultante. Se puede observar que los valores anteriores definen una función de probabilidad en cada nodo para cada hormiga, ya que se ha normalizado para que la suma sea 1. Además, hemos de tener en cuenta que:

- Si $\alpha = 0$, las ciudades más cercanas en cada paso son las que tienen mayor probabilidad de ser seleccionadas, lo que se correspondería con el algoritmo voraz clásico en su versión estocástica (con múltiples puntos de inicio, ya que, como veremos, las hormigas se colocan inicialmente en una ciudad al azar). En consecuencia, las hormigas no usan el conocimiento de la colonia para mejorar su comportamiento, que viene definido por la cantidad de feromona que hay en las aristas.

- Si $\beta = 0$ únicamente interviene la feromona, lo que experimentalmente se comprueba que puede llevar a recorridos no muy buenos y sin posibilidad de mejora.

Algunas variantes para mejorar la eficiencia de este algoritmo introducen, por ejemplo:

- **Sistema Elitista**: Añadir en cada paso feromona a las aristas del mejor recorrido encontrado hasta el momento, haya sido encontrado en el paso actual o no.

- **Sistema por Ranking**: Es el usado en esta entrada, cada hormiga deposita feromona proporcional a la bondad de la solución encontrada.

- **Sistema de Colonias**: Todas las hormigas dejan la misma cantidad de feromona, independientemente de la bondad de la solución, pero la ejecución no es sincronizada, sino que las hormigas que acaban antes pueden volver a comenzar una nueva generación de camino. De esta forma, los caminos más cortos, tendrán más posibilidad de ser repetidos. Es el sistema que ocurre en la naturaleza.

3.3.3 PSEUDOCODIGO DE OPTIMIZACION DE COLONIAS DE HORMIGAS.

A continuación, definiremos el pseudocódigo de ACO para comprender como se elabora la optimización.

```
ACO ( )
    {
        Definición de parámetros
        Inicialización de rastro de feromona
        While(criterio de término no satisfecho)
      {
        Generar hormigas y sus soluciones;
        Actualizar rastros de feromona;
        Aplicar acciones opcionales;
      }
     }
```

Ilustración 5: Pseudocódigo Colonias de hormiga.

3.4 FORMULA DE HARVESINE.

Para la el cálculo de distancias en un mapa geo localizado de manera rápida y eficaz se utilizará la formula harvesine que permite calcular distancias mediante latitud y longitud.

La **fórmula del semiverseno** es una importante ecuación para la navegación astronómica, en cuanto al cálculo de la distancia de círculo máximo entre dos puntos de un globo sabiendo su longitud y su latitud. Es un caso especial de una fórmula más general de trigonometría esférica, la **ley de los semiversenos**, que relaciona los lados y ángulos de los "triángulos esféricos" [8].

Estos nombres derivan del hecho que suele expresarse en términos de la función haversine, dada por

$$\boldsymbol{haversin}\,(\boldsymbol{\theta}) \;=\; \boldsymbol{sen}\,\mathbf{2}\,(\boldsymbol{\theta}/\mathbf{2})$$

Para cualquier par de puntos sobre una esfera:

$$harvesin\left(\frac{d}{R}\right) = haversin(\varphi_1 - \varphi_2) + \cos(\varphi_1)\cos(\varphi_2)\, haversin(\Delta\lambda).$$

Ecuación 5: Formula Harvesina.

Donde

haversin es la función haversine, haversin $(\theta) = sen\,2\,(\theta/2) = (1 - cos\,(\theta))/2$

d es la distancia entre dos puntos (sobre un círculo máximo de la esfera, véase distancia esférica),

R es el radio de la esfera,

φ 1 es la latitud del punto 1,

φ 2 es la latitud del punto 2, y

Δ λ es la diferencia de longitudes

Hay que tener en cuenta que el argumento a la función haversine se supone que debe darse en radianes. En grados, haversin (d / R) de la fórmula se convertiría en haversin $(180 \cdot d / \pi R)$.

3.5 DRONE UTILIZADO.

DJI PHANTOM 4 PRO es una cámara aérea inteligente semiprofesional capaz de filmar video en 4K a 60fps y hasta 100 mbps, así como captar fotografías de 20 megapixeles. Su capacidad para evitar obstáculos en 4 direcciones le permiten esquivar obstáculos inteligentemente durante el vuelo. Utilizando los modos TapFly y ActiveTrack mejorados a través de la aplicación DJI GO 4, puede volar a

cualquier parte visible en la pantalla o seguir un objetivo móvil de manera fácil y fluida con un solo toque en la pantalla. La nueva cámara utiliza un sensor CMOS de una pulgada que ofrece una claridad sin precedentes, así como un bajo nivel de ruido e imágenes de mayor calidad.

Ilustración 6: Drone Dji Phantom 4 Pro, extraído de DJI.com.

Daremos a conocer las características técnicas que el Drone utilizado posee:

AERONAVE	
Peso (batería y hélices incluidas)	1388 g
Tamaño diagonal (sin hélices)	350 mm
Velocidad de ascenso máx.	Modo-S: 6 m/s (19.7 ft/s) Modo-P: 5 m/s (16.4 ft/s)
Velocidad de descenso máx.	Modo-S: 4 m/s (13.1 ft/s) modo-P: 3 m/s (9.8 ft/s)
Velocidad máx.	72 km/h (45 mph) (modo-S) 58 km/h (36 mph) (modo-A) 50 km/h (31 mph) (modo-P)
Resistencia al viento máx.	10 m/s
Tiempo de vuelo máx.	30 minutos aprox.
Rango de temperatura de funcionamiento	De 0 a 40 °C (de 32 a 104 °F)

Sistemas de posicionamiento por satélite	GPS / GLONASS

SISTEMA DE VISIÓN

Sistema de visión	Sistema de visión frontal Sistema de visión posterior Sistema de visión inferior
Rango de velocidad	≤50 km/h (31 mph) a 2 m (6.6 pies) del suelo
Rango de Altitud	0 - 10 m (0 - 33 pies)
Rango de Operación	0 - 10 m (0 - 33 pies)
Rango de detección de obstáculos	0.7 - 30 m (2 - 98 ies)
Campo de visión	Frontal: 60º (horizontal), 27º (vertical) Posterior: 60º (horizontal), 27º (vertical) Inferior: 70º (de frente y hacia atrás), 50º (a izquierda y derecha)
Frecuencia de detección	Frontal:10Hz Posterior:10Hz Inferior: 20 Hz
Entorno operativo	Superficie con un patrón definido y una iluminación adecuada (lux > 15)

3.5.1 APLICACIÓN GSDEMO.

GSDemo-GoogleMap está diseñado para aprender a implementar el DJIWaypoint Mission y familizarse con el uso de DJIMissionManager. Además sabrás como probar la API Waypoint Mission con el Simulador DJI PC. Este demo está basado en Google Map.

3.5.2 SIMULADOR DJI.

DJI Assistant 2 For Phantom: Es un software que trae utilidades para los equipos DJI tales como:

- Firmware Update
- Data Upload
- Black Box
- Calibration
- Simulator
- WiFi Settings
- Restore Factory Defaults
- DJI Privacy Policy and Terms of Use of the DJI Assistant 2 app

Para este proyecto se utilizó el simulador para probar el algoritmo y las rutas obtenidas.

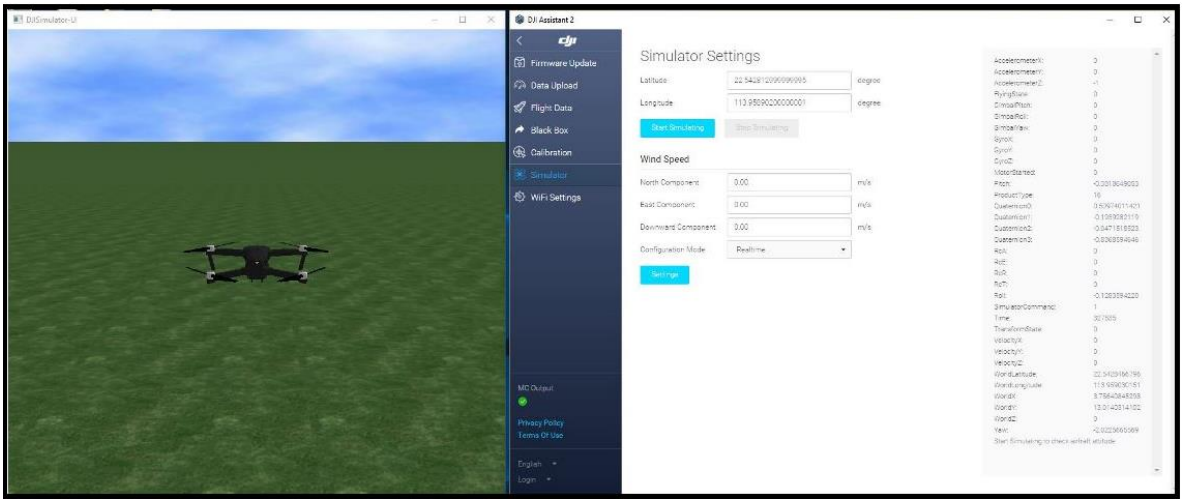

Ilustración 7: Simulador Dji Assitant 2.

3.6 DISPOSITVO MOVIL ANDROID.

Android es un sistema operativo basado en el núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos

inteligentes, tabletas y también para relojes inteligentes, televisores y automóviles. Inicialmente fue desarrollado por Android Inc [9].

Para esta aplicación es necesario un dispositivo con mínimo:

Android (4.4.4) Kit Kat como versión mínima para lanzar la aplicación.

3.7 TECNOLOGÍA DE LA PROGRAMACIÓN.

Una vez analizadas las tecnologías de ruta y hardware que se utilizaron para esta aplicación, procedemos a explicar el lenguaje con el que está basado este proyecto. Se utilizó el lenguaje Java, ya que es uno de los más flexibles a la hora de realizar aplicaciones, además que gran parte de la documentación provista por DJI y la aplicación básica para el Drone está basada en este lenguaje.

3.7.1 JAVA.

Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado y cada día se crean más. Java es rápido, seguro y fiable. Desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet, Java está en todas partes [10].

3.7.2 ANDROID STUDIO.

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y se basa en IntelliJ IDEA . Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece aún más funciones que aumentan tu productividad durante la compilación de apps para Android, como las siguientes:

- Un sistema de compilación basado en Gradle flexible
- Un emulador rápido con varias funciones
- Un entorno unificado en el que puedes realizar desarrollos para todos los dispositivos Android
- Instant Run para aplicar cambios mientras tu app se ejecuta sin la necesidad de compilar un nuevo APK
- Integración de plantillas de código y GitHub para ayudarte a compilar funciones comunes de las apps e importar ejemplos de código
- Gran cantidad de herramientas y frameworks de prueba
- Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versión, etc.
- Compatibilidad con C++ y NDK
- Soporte incorporado para Google Cloud Platform, lo que facilita la integración de Google Cloud Messaging y App Engine [11].

3.8 ARQUITECTURA TECNOLÓGICA.

Durante este capítulo se definieron distintas tecnologías que se utilizaron en el desarrollo de este proyecto, tanto software como hardware. A continuación, en la Ilustración 8 se muestra el funcionamiento de estas tecnologías.

Ilustración 8: Funcionamiento de la aplicación, elaboración propia.

Mediante la ilustración que acabamos de ver podemos observar el funcionamiento general de la aplicación, incorporando las tecnologías vistas previamente.

El control del Drone se conecta al dispositivo android y se ejecuta la aplicación móvil. Luego se enciende el Drone y se conecta al control para así recibir la información que la aplicación envía y poder realizar las acciones que se deseen.

.

CAPÍTULO IV: ANÁLISIS Y DISEÑO DEL SISTEMA.

Antes de comenzar con el análisis a fondo de este proyecto, se debe mencionar la metodología que se utilizó y el equipo a cargo para el desarrollo de esta aplicación.

4.1 METODOLOGÍA DE TRABAJO.

Para el desarrollo de este trabajo se utilizó una metodología evolutiva, ya que al ser una modificación de modulo y se requerían cambios constantes de acuerdo a las solicitudes del cliente se hacían cambios rápidos para cumplir con los nuevos requerimientos.

4.1.1. PROTOTIPO EVOLUTIVO

El modelo de ciclo de vida de software es una representación abstracta de un proceso de software. El software como todos los sistemas complejos evolucionan, por lo que no es extraño que a medida del desarrollo de un software los requerimientos del negocio y el proyecto también cambien. Es por ello que se presenta un modelo de ciclo de vida, el ciclo de Vida de Prototipado Evolutivo

El modelo de prototipos permite que todo el sistema, o algunas sus partes, se construyan rápidamente para comprender o aclarar aspectos, tiene el mismo objetivo que un prototipo de ingeniería, donde los requerimientos o el diseño requieren la investigación repetida para asegurar que el desarrollador, el usuario y el cliente tengan una comprensión unificada tanto de lo que se necesita como de lo que se propone como solución [12].

4.2 REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES.

En la ingeniería de software y en la ingeniería de sistemas es necesario realizar este proceso, el cual consiste en recopilar y analizar todas las actividades requeridas por un cliente para crear e implementar el sistema que se requiere construir.

4.2.1 REQUERIMIENTOS FUNCIONALES.

Los requerimientos funcionales describen la funcionalidad que se espera que el software provea. Estos dependen del tipo de software, del sistema a desarrollar y de los posibles usuarios. A continuación se definirán los requerimientos funcionales de nuestra aplicación.

- Número de identificación del requerimiento.
- Nombre del requerimiento.
- Descripción general del requerimiento.

Número	RF01
Nombre	Conectar el Drone
Descripción	Para comenzar la aplicación es necesario que el usuario tenga un drone DJI conectado al dispositivo Android, de otra manera el botón “OPEN” no se habilitará.
Prioridad	Alta.

Tabla 1: Drone conectado.

Número	RF02
Nombre	Marcado de puntos
Descripción	Los usuarios deben presionar el botón “ADD” para marcar los puntos en la aplicación y así marcar el recorrido de misión.
Prioridad	Alta.

Tabla 2: Marcado de puntos.

Número	RF03
Nombre	Localizar
Descripción	El usuario debe apretar el botón “LOCATE” antes de comenzar una misión, para que el Drone sea marcado en el mapa en una posición.
Prioridad	Alta.

Tabla 3: Localizar el Drone.

Número	RF04
Nombre	Optimizar
Descripción	El usuario debe apretar el botón “ACO” para obtener una ruta optimizada y aceptarla si así lo desea.
Prioridad	Media.

Tabla 4: Optimización de ruta.

4.2.2 REQUERIMIENTOS NO FUNCIONALES.

Los requerimientos no funcionales, son requisitos que son utilizados como características para valorar la aplicación, en cambio, como se dijo anteriormente, los funcionales representan a las operaciones o funcionamiento lógico de la aplicación.

Para representar a los requisitos no funcionales, se usará la misma estructura utilizada previamente.

Número	RNF01
Nombre	Mostrar distancia
Descripción	Es necesario comparar la distancia total antes de la optimización y después de ella para que tenga sentido el resultado.
Prioridad	Alta.

Tabla 5: Mostrar Distancia.

Número	RNF02
Nombre	Mostrar que la aeronave completo el viaje
Descripción	Una vez completado el viaje mostrar por pantalla que el viaje ha finalizado con éxito.
Prioridad	Alta.

Tabla 6: Subir ruta a la aeronave.

CAPÍTULO V: DESCRIPCIÓN DE LA APLICACIÓN

En este capítulo se mostraran los cambios realizados a la aplicación GSDemo para realizar el cálculo de ruta en el lenguaje java utilizando el algoritmo metaheurístico ACO.

5.1 DESARROLLO DEL MÓDULO DE CÁLCULO.

Este proyecto se desarrolló en Android Studio, utilizando el lenguaje de programación Java. Además se trabajó con el código de ACO para calcular la ruta más corta.

Demostraremos primero el pseudocódigo utilizado para su mayor comprensión antes de ver la implementación en java.

```
ACO()
{
 Inicializamos atributos
 Inicializamos la feromona de manera aleatoria
 Para cada hormiga
 {
  Bajo una posibilidad aleatoria de recorrer el camino
  Agregamos hormigas activas
  Ajustamos feromona
  Procesamos y comparamos la ruta más corta
  Asignamos la ruta más corta
  Si quedan hormigas activas
  Procesamos y comparamos la ruta más corta
  Asignamos la ruta más corta
 }
```

Ilustración 9: Pseudocódigo utilizado, elaboración propia

Primero como se observa en la ilustración 10, se debe inicializar la cantidad de hormigas que tendremos funcionando y debemos definir la probabilidad de que recorra un destino.

Inicializamos los ExecutorService que se encargarán de trabajar en hilos y de manera ininterrumpida hasta completar los cálculos.

Estableceremos el valor del objeto shorestRoute y definiremos la cadena para conocer el nombre de la posición con RutaMasCorta, DistanciaMasCorta nos entrega el valor de distancia que se obtuvo.

Y finalmente, la última variable, esta se encarga de mantener a las hormigas activas hasta que finalicen su recorrido.

```
static final int NUMBER_OF_ANTS = 1000;
static final double PROCESSING_CYCLE_PROBABLITY = 0.5;
static ExecutorService executorService =
        Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
static ExecutorCompletionService<Ant> executorCompletionService =
        new ExecutorCompletionService<Ant>(executorService);
private Route shortestRoute = null;
private static String RutaMasCorta;
private static double DistanciaMasCorta;
private int activeAnts = 0;
```

Ilustración 10: Variables iniciales, elaboración propia.

La Ilustración 11, representa una de las funciones más importantes de este software, corresponde al algoritmo ACO similar al pseudocódigo explicado anteriormente, además muestra por consola y hace un set a los datos importantes como los datos obtenidos.

```
public static void Driver() throws IOException {
    System.out.println("> "+NUMBER_OF_ANTS+" Hormigas Artificiales");
    Driver driver = new Driver();
    driver.printHeading();
    AntColonyOptimization aco = new AntColonyOptimization();
    //System.out.println("> Driver main inicio loop de hormigas");
    IntStream.range(1,NUMBER_OF_ANTS).forEach(x->{
        /*System.out.println("\n Driver executorCompletionService.submit nueva hormiga");*/
        executorCompletionService.submit(new Ant(aco,x));
        driver.activeAnts++;
        if (Math.random()>PROCESSING_CYCLE_PROBABLITY) driver.processAnts();
    });
    //System.out.println("\n Driver main fin loop de hormigas");
    driver.processAnts();
    driver.SetArrayRuta(Arrays.toString(driver.shortestRoute.getCities().toArray()));
    FinalRoute = driver.shortestRoute.getCities();
    driver.SetDistanciaMasCorta(driver.shortestRoute.getDistance());
    System.out.println("\n Ruta optima : "+Arrays.toString(driver.shortestRoute.getCities().
            toArray()));
    System.out.println("Distancia : " + driver.shortestRoute.getDistance());
}
```

Ilustración 11: Función de extracción, elaboración propia.

En la función de la Ilustración 12, se utiliza la función processAnts que se encarga de asignar una ruta a cada executorCompletionService correspondiente a una hormiga, cada hormiga recibe una ruta y se evalúa si la ruta encontrada es más corta que la anterior comparando la variable shortestRoute.

```
private void processAnts() {
    while(activeAnts > 0) {
        //System.out.println("Driver main take hormiga");
        try {
            Ant ant = executorCompletionService.take().get();
            Route currentRoute = ant.getRoute();
            if(shortestRoute == null || currentRoute.getDistance() < shortestRoute.getDistance()) {
                shortestRoute = currentRoute;
                StringBuffer distance = new StringBuffer("      "+String.format
                        ("%.2f",currentRoute.getDistance()));
                IntStream.range(0, 21-distance.length()).forEach(k-> distance.append(" "));
                System.out.println(Arrays.toString(shortestRoute.getCities().toArray())+ " |"
                        + distance + "| " + ant.getAntNumb());
            }
        }
        catch (Exception e) { e.printStackTrace();}
        activeAnts--;
    }

}
```

Ilustración 12: Función processAnts (), elaboración propia.

En la Ilustración 13, podemos observar la clase City que corresponde a cada punto en el mapa, establecemos las constantes EARTH_EQUATORIAL_RADIUS y CONVERT_DEGREES_TO_RADIANS para su posterior uso e inicializamos las variables de longitud y latitud tanto en radianes como en grados. En el Constructor asignamos el nombre y convertimos los grados en radianes para facilitar el cálculo de distancias y almacenamos los grados.

```java
package AC0;

import ...

public class City {
    private static final double EARTH_EQUATORIAL_RADIUS = 6371.1370D;
    private static final double CONVERT_DEGREES_TO_RADIANS = Math.PI/180D;
    private double longitude;
    private double longitudedegree;
    private double latitude;
    private double latitudedegree;
    private float altitude;
    private String name;


    public City(String name,double latitude, double longitude) {
        this.name = name;
        this.latitude = latitude * CONVERT_DEGREES_TO_RADIANS;
        this.longitude = longitude * CONVERT_DEGREES_TO_RADIANS;
        this.latitudedegree = latitude;
        this.longitudedegree = longitude;
    }
```

Ilustración 13: Proceso de identificar, elaboración propia.

En la ilustración 14 podemos ver cómo se calcula la distancia entre puntos utilizando la fórmula de harvesine y los getters de cada uno de los atributos.

```java
    public double measureDistance(City city) {
        double deltaLongitude = (city.getLongitude() - this.getLongitude());
        double deltaLatitude = (city.getLatitude() - this.getLatitude());
        double a = Math.pow(Math.sin(deltaLatitude/ 2D),2D)+
                Math.cos(this.getLatitude())* Math.cos(city.getLatitude())* Math.pow(Math.sin(deltaLongitude/2D),2D);
        return EARTH_EQUATORIAL_RADIUS * 2D * Math.atan2(Math.sqrt(a), Math.sqrt(1D-a));
    }

    public String getName() {return name;}
    public double getLatitude() { return this.latitude;}
    public double getLatitudeDegree() { return this.latitudedegree;}
    public double getLongitude() { return this.longitude;}
    public double getLongitudeDegree() { return this.longitudedegree;}
    public String toString() { return this.name;}

}
```

Ilustración 14: Función measureDistance, elaboración propia.

5.2 EDICIÓN EN GSDEMO.

En la ilustración 15 definimos las Listas initialRoute y FinalRoute en conjunto con el objeto ACOH quien maneja el comportamiento de las hormigas, el resto de inicializaciones pertenecen a la documentación DJI la cual permite realizar misiones y crear las listas de ruta.

```
private Button locate, add, clear,ACO;
private Button config, upload, start, stop;
int C=1,B=1;
private boolean isAdd = false;
private double droneLocationLat = 181, droneLocationLng = 181;
private final Map<Integer, Marker> mMarkers = new ConcurrentHashMap<Integer, Marker>();
private Marker droneMarker = null;
private float altitude = 100.0f;
private float mSpeed = 10.0f;
static public List<City> initialRoute = new ArrayList<City>();
static public List<City> FinalRoute = new ArrayList<City>();
private List<Waypoint> waypointList = new ArrayList<>();
public static WaypointMission.Builder waypointMissionBuilder;
private FlightController mFlightController;
private WaypointMissionOperator instance;
private WaypointMissionFinishedAction mFinishedAction = WaypointMissionFinishedAction.NO_ACTION;
private WaypointMissionHeadingMode mHeadingMode = WaypointMissionHeadingMode.AUTO;
Driver ACOH = new Driver();
```

Ilustración 15: Inicialización Initial y final Route

En la ilustración 16 se observa la función markWaypoint que fue modificada para una mejor visualización del orden que se seleccionó y el obtenido.

```
private void markWaypoint(LatLng point){
    //Create MarkerOptions object
    //Adding number to the waypoint
    Bitmap.Config conf = Bitmap.Config.ARGB_8888;
    Bitmap bmp = Bitmap.createBitmap( width: 120,  height: 120, conf);
    Canvas canvas1 = new Canvas(bmp);
    // paint defines the text color, stroke width and size
    Paint color = new Paint();
    color.setTextSize(35);
    color.setColor(Color.CYAN);
    // modify canvas
    canvas1.drawBitmap(BitmapFactory.decodeResource(getResources(),
            R.drawable.pin2),  left: 0,  top: 0, color);
    canvas1.drawText( text: ""+C,  x: 30,  y: 50, color);
    MarkerOptions markerOptions = new MarkerOptions();
    markerOptions.position(point);
    markerOptions.icon(BitmapDescriptorFactory.fromBitmap(bmp));
    //markerOptions.title(Integer.toString(C));
    Marker marker = gMap.addMarker(markerOptions);
    mMarkers.put(mMarkers.size(), marker);
}
```

Ilustración 16: Modificación visibilidad marcado de puntos

En la ilustración 17 se encuentra el código correspondiente al botón “ACO” agregado el cual desempeña las funciones de cálculo y mensaje de confirmación de cambio de ruta.

```
DecimalFormat df = new DecimalFormat( pattern: "#.###");
if(initialRoute.size()>2){
try {
      ACOH.Driver();
    } catch (IOException e) {
    e.printStackTrace();
    }
    String dfx = df.format(ACOH.GetDistanciaMasCorta());
    builder.setCancelable(true);
    builder.setTitle("Confirmación");
    builder.setMessage("¿Desea utilizar está ruta? \n "+ACOH.GetArrayRuta()+" \n"+dfx+" Kilometros ");
    builder.setPositiveButton( text: "Aceptar",
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    C=1;
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() { gMap.clear(); }
                    });
                    waypointList.clear();
                    for(int i=0;i<FinalRoute.size();i++){
                        Waypoint mWaypoint = new Waypoint(FinalRoute.get(i).getLatitudeDegree(), FinalRoute.get(i).getLongitudeDegree(), altitud
                        LatLng point = new LatLng(FinalRoute.get(i).getLatitudeDegree(),FinalRoute.get(i).getLongitudeDegree());
                        waypointList.add(mWaypoint);
                        waypointMissionBuilder.waypointList(waypointList).waypointCount(waypointList.size());
                        markWaypoint(point);
                        C++;
                    }
                }
            });
```

Ilustración 17: Botón ACO

5.3 VISTAS.

Las vistas son las capas de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y la delega a la plantilla apropiada: se puede decir que es como un puente entre los modelos y las plantillas.

La Ilustración 18 representa la necesidad de conectar el Drone a la aplicación de no cumplirse esta condición no se podrá acceder a la aplicación.

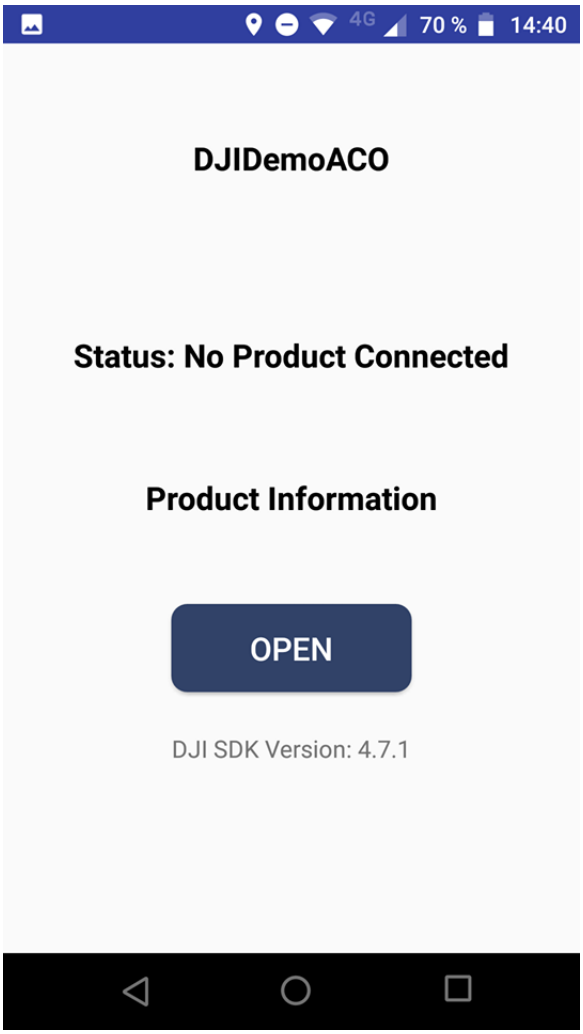

Ilustración 18: Primera vista aplicación, elaboración propia.

En la ilustración 19 podemos ver la ubicación por defecto que considera y ver el nuevo botón ACO que realiza las funciones de optimización.

Ilustración 19: Mapa para misiones y localización, Elaboración propia.

En la ilustración 20 Podemos observar un mapeo de puntos en un orden específico para completarse una misión. Esta aún no ha sido optimizada por lo que el recorrido es poco eficiente.

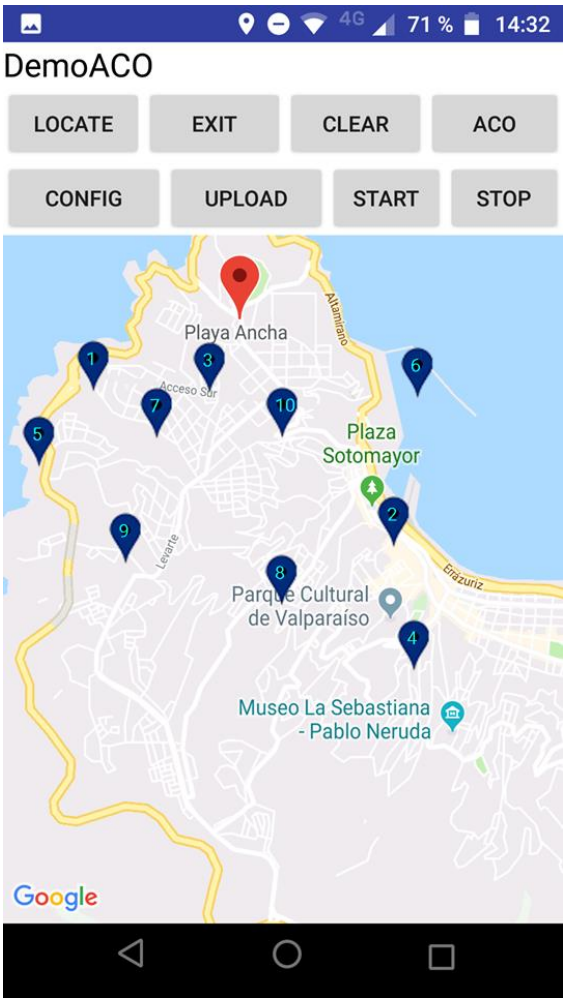

Ilustración 20: Mapeo de puntos con dígitos, elaboración propia.

En la ilustración 21 una vez presionado el botón “ACO” nos demostrará un orden a evaluar en comparación al que teníamos anteriormente, mostrando la distancia total de la ruta que existía y la nueva.

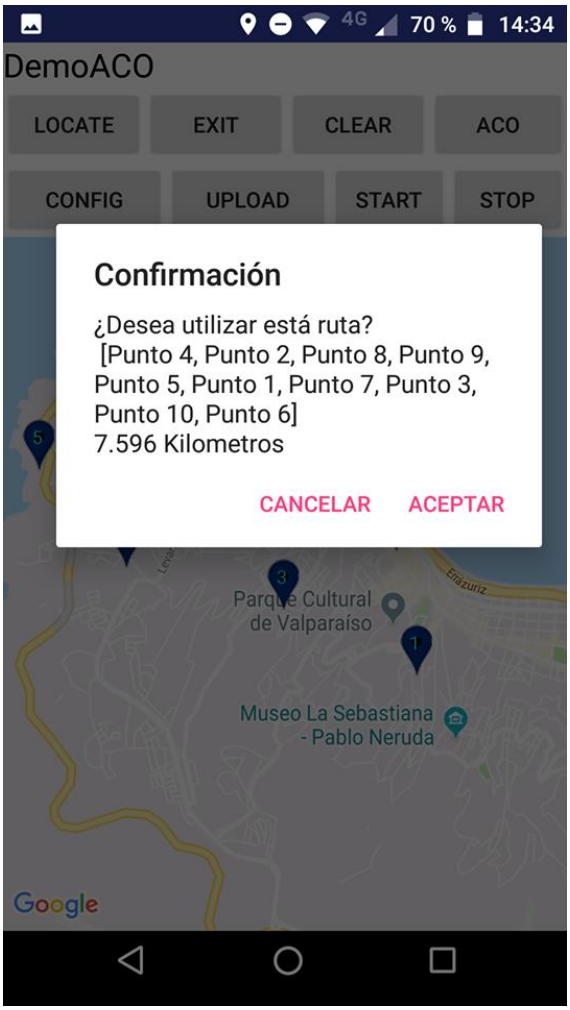

Ilustración 21: Botón “ACO”, elaboración propia.

En la ilustración 22 podemos observar que una vez confirmada la ruta el orden de los puntos es cambiado y si la misión anteriormente es válida se realizara una vez asignados los datos de “CONFIG” y subida a la aeronave con “UPLOAD”

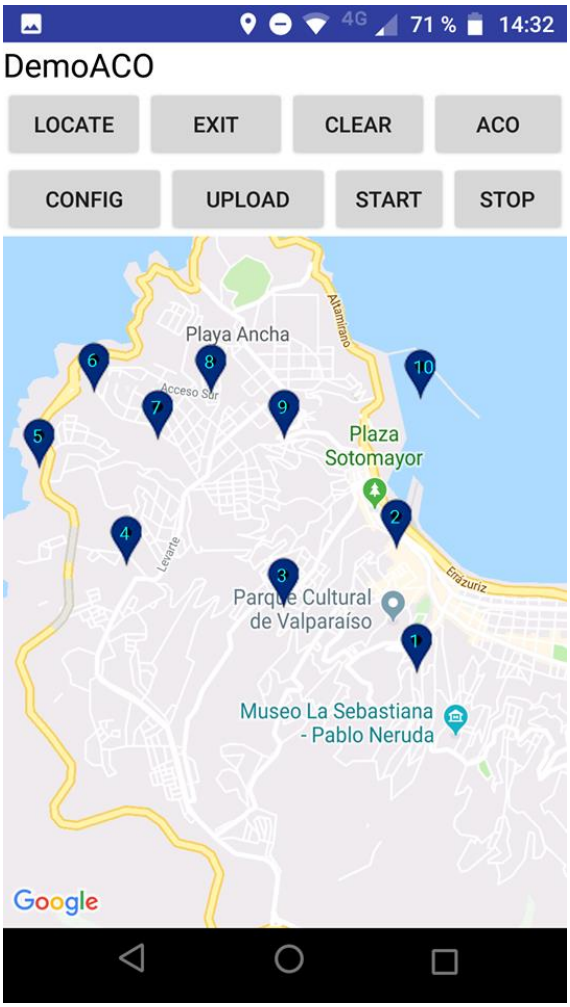

Ilustración 22: Ruta optimizada, elaboración propia.

CAPÍTULO VI: CONCLUSIONES.

Se logró desarrollar un módulo de cálculo e integrarlo a la aplicación ya existente, el cual, es bastante útil para obtener un mejor recorrido. A partir de esto, se pudo llegar a las siguientes conclusiones:

- Mediante el modulo, un usuario puede acceder la cantidad que desee de puntos y, rápidamente obtendrá un resultado asignado para su recorrido sin tener que intervenir de alguna manera.
- Mediante ACO y la formula harvesine es posible optimizar rutas de gran variedad sin recurrir a terceros que puedan alterar el cálculo de distancias con latitudes y longitudes.
- Durante el desarrollo de este presente proyecto, se dio la oportunidad de trabajar con DJI, empresa la cual ha documentado muy bien las funciones implementadas en sus aeronaves tanto como para el consumidor como el desarrollador. Por otra parte el simulador es una gran herramienta para evitar dañar las aeronaves en vuelos alterados mientras se probaba el algoritmo.

6.1 CONCLUSIONES RESPECTO AL OBJETIVO GENERAL

- El objetivo general del proyecto fue “Desarrollar una aplicación capaz de optimizar los recorridos en la misión de un Drone mediante metaheurística ACO”.

Mediante la tecnología informática y metaheurística fue posible desarrollar un módulo con la capacidad de tomar una ruta, trabajarla, y obtener una ruta más eficiente considerando la distancia entre cada uno de los puntos asignados. Esto fue logrado gracias a las capacidades de optimización de ACO y las herramientas de JAVA las cuales son muy flexibles en muchos ámbitos.

6.2 CONCLUSIONES RESPECTO A LOS OBJETIVOS ESPECÍFICOS

El primer objetivo específico fue “Realizar un estudio sobre las tecnologías metaheurísticas que puedan ser aplicadas a la obtención de una ruta optimizada”.

Mediante el estudio realizado a las tecnologías metaheurísticas, se logró elegir un algoritmo adecuado para la optimización de ruta. Pese a que existen otros algoritmos alternativos, esta era la mejor opción dado el conocimiento previo obtenido a lo largo del ciclo académico.

El Segundo objetivo específico fue “Desarrollar un módulo que sea capaz de calcular una ruta utilizando ACO mediante los puntos asignados en la aplicación GSDemo”.

Gracias a la gran documentación que existe en la web sobre la programación de ACO y los Drone DJI, el desarrollo y la carga de ruta al Drone fueron completamente posibles. Además, la página oficial de DJI ofrece soporte para los desarrolladores interesados en hacer modificaciones de software a todos los Drone registrados que poseen.

El tercer objetivo específico fue “Una vez obtenida una ruta optimizada, |realizar vuelos de simulación y proceder a realizar vuelos de manera real”.

Ya con la ruta optimizada, se conectó la aeronave al simulador para comprobar que la aplicación entregara rutas validas que se pudieran realizar. Una vez realizadas las pruebas simuladas exitosas el proyecto se encuentra listo para probar su funcionalidad en espacios de mayor amplitud.

Las limitaciones de este trabajo, las encontramos en dos sentidos. Como primera instancia el hardware la aeronave tiene un límite de batería y una señal gps que debe ser estar presente en todo momento. Hablando de software la forma de conexión solo permite manipular un Drone a la vez, por lo que está limitado a solo 1 por aplicación.

Dentro de lo estudiado, se han tocado muchos tópicos en algún punto de este proyecto abarcando una gran variedad de información tal como:

Se aprendió a utilizar un drone para el cual se descubrieron protocolos de conexión que permitían interactuar con código abierto, debido a esto se recurrió a la programación de tecnologías móviles, y buscando soluciones se consideró rápidamente como un problema de combinatoria que fue resuelto con una metaheurística bio-inspirada mejor conocida como ACO.

Posterior al desarrollo del módulo encargado de obtener mejores rutas, se puede tomar en consideración que aún existen maneras de seguir optimizando ya que, aún se puede explorar más, es posible considerar la altura, el estado del viento(velocidad y/o sentido) y otras variables que podrían afectar el tiempo en que se hace el recorrido.

REFERENCIAS

[1] S. López, P. Morales (s.f.), “Desarrollo del Sistema Informático” [online]. Disponible en: http://es.calameo.com/read/002236482dd587d1dcbe9

[2] Weisstein, Eric W. (s.f) "Traveling Salesman Problem, "Disponible en:http://mathworld.wolfram.com/TravelingSalesmanProblem.html

[3] José Moreno Pérez. Metaheurísticas: Concepto y Propiedades,

http://www.tebadm.ulpgc.es/almacen/seminarios/MH%20Las%20Palmas%202.pdf

[4] Gary B. Foge (2008) Evolutionary Algorithms, https://web.archive.org/web/20080424073828/http://www.scholarpedia.org/article/Evolutionary_algorithms

[5] Marco Dorigo et al. (2008) Particle swarm optimization. Scholarpedia, 3(11):1486., revision #91633 Disponible en:

http://www.scholarpedia.org/article/Particle_swarm_optimization

[6] Farmer J.D. (1986).The immune system, adaptation and machine learning, *Physica D*, **2**, 187-204

[7] Marco Dorigo (s.f), About ACO http://www.aco-metaheuristic.org/about.html

[8] José de Mendoza y Ríos (1795). Memoria sobre algunos métodos nuevos de calcular la longitud por las distancias lunares y explicaciones prácticas de una teoría para la solución de otros problemas de navegación. Imp. Real

[9] ¿Qué es el sistema operativo Android? (s.f.) http://www.ictea.com/cs/index.php?rp=/knowledgebase/8974/iQue-es-el-Sistema-Operativo-Android.html

[10] ¿Qué es la tecnología Java y para qué la necesito? (s.f.) https://www.java.com/es/download/faq/whatis_java.xml

[11] Conoce Android studio (s.f.) https://developer.android.com/studio/intro/

[12] Modelo de prototipos (s.f.) https://www.ecured.cu/Modelo_de_prototipos

BIBLIOGRAFÍA DE APOYO

Omar Wilton Saavedra Salazar. (2011, 02 de Mayo).Archive for 'Java SE' Category Recuperado el 9 de Agosto de 2017 a partir de https://codecix.wordpress.com/category/java/java-se/

Alex Rodríguez. (s.f.) ¿Qué es Java? Concepto de programación orientada a objetos. Recuperado el 19 de Agosto de 2017 a partir de https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=368:ique-es-java-concepto-de-programacion-orientada-a-objetos-vs-programacion-estructurada-cu00603b&catid=68&Itemid=188

Caparrini, F., & Work, W. (2018). Algoritmos de hormigas y el problema del viajante http://www.cs.us.es/~fsancho/?e=71

Pressman, R. (2010). Ingeniería del software: un enfoque práctico (7a. ed.). México, D.F., MX: McGraw-Hill Interamericana. Sitio Web: http://www.ebrary.com

Dr.Gary B. Fogel, Evolutionary algorithms https://web.archive.org/web/20080424073828/http://www.scholarpedia.org/article/Evolutionary_algorithms

Yang, Xiaohu Shi , Maurizio Marchese, Liang An (2008) ant colony optimization method for generalized TSP problem https://www.sciencedirect.com/science/article/pii/S1002007108002736

Andrés Aranda, Javier Jiménez (2013), Optimización rutas de transporte http://eprints.ucm.es/23027/1/Memoria_OptimizacionRutasTransporte.pdf

Emine Es Yurek,H. Cenk Ozmutlu (2018), A decomposition-based iterative optimization algorithm for traveling salesman problem with drone https://www.sciencedirect.com/science/article/pii/S0968090X18304662

Quang Minh Ha,Yves Deville,Quang Dung Pham,Minh Hoàng Hà (2018), On the min-cost Traveling Salesman Problem with Drone https://www.sciencedirect.com/science/article/pii/S0968090X17303327

Yong Sik Chang,Hyun Jung Lee (2018), Optimal delivery routing with wider drone-delivery areas along a shorter truck-route https://www.sciencedirect.com/science/article/pii/S0957417418301775

Tavanaab, Khalili-Damghanic, J.Santos-Arteaga, Zandif (2017) Drone shipping versus truck delivery in a cross-docking system with multiple fleets and products https://www.sciencedirect.com/science/article/pii/S095741741630687X

DJI Developer (s.f.), https://developer.dji.com/mobile-sdk/

Documentation Introduction (s.f.), https://developer.dji.com/mobile-sdk/documentation/introduction/index.html