Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Desarrollo de app para drones

App development for drones

Guillermo Hernández González

La Laguna, 7 de Junio de 2022

Dña. **Molina Gil, Jezabel Miriam**, con N.I.F. 78.507.682-B profesora Ayudante a Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

D. **Ricardo Aguasca Colomo**, con NIF 43.644.158-W, profesor Titular de Universidad, adscrito al Dpto. de Ingeniería Electrónica y Automática de la Universidad de Las Palmas de Gran Canaria, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

“Desarrollo de app para drones”

ha sido realizada bajo su dirección por D. **Guillermo Hernández González**,

con N.I.F. 51.154.097-G.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 21 de noviembre de 2018

Agradecimientos

A Jezabel Miriam Molina Gil y a Ricardo Aguasca Colomo por proporcionarme un TFG, asesorarme y ayudarme durante todo el proceso.

A Javier Correa y a todo el resto del equipo de ACUDROPOL por el soporte prestado y por poder facilitarnos la posibilidad de realizar una prueba de vuelo real.

A toda mi familia que ha estado apoyándome durante toda la carrera y el TFG.

Licencia

© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional.

Resumen

Cada día se vuelve más común el uso del dron en España, tanto de manera particular como de uso profesional por una empresa. Sus usos están dentro de muchos de nuestros ámbitos del día a día como vigilar las calles, fumigar o desinfectar cultivos, labores de búsqueda, uso como herramienta fotográfica o dentro de espectáculos.

El ámbito de este proyecto se propone su uso como herramienta para ayudar en situaciones de rescate o rastreo de zonas poco accesibles. El objetivo de este trabajo ha sido crear una aplicación la cual permita al usuario, dando los datos del dron y de la zona que quiere explorar, crear un fichero compatible con el controlador del dron que contenga una serie de *waypoints,* colocados dentro del área para que el dron los recorra. Estos waypoints habrán sido calculados de manera que el recorrido realizado por el dron sea el más eficiente posible.

La herramienta desarrollada se ha realizado en entorno web ya que de esta manera se puede asegurar que funciona en una gran variedad de dispositivos sin tener en cuenta sus especificaciones. La aplicación se ha diseñado con una interfaz sencilla e intuitiva para que los usuarios puedan adaptarse rápidamente a ella.

Para el desarrollo se ha usado *Javascript*, *HTML*, *CSS*, *Bootstrap* y algunas *API* de la suite de *Google* como *Maps* y *Geolocator*.

Palabras clave: Dron, Maps, Google, Javascript, API, Coordenadas...

Abstract

The use of drones is becoming more and more common in Spain, both for private and professional use by companies. Its uses are in many of our day-to-day areas such as monitoring the streets, fumigating or disinfecting crops, search work, use as a photographic tool or in performances.
The scope of this project proposes its use as a tool to help in situations of rescue or tracking of inaccessible areas. The aim of this work has been to create an application which allows the user, given the data of the drone and the area to be explored, to create a file compatible with the drone controller which contains a series of waypoints, placed within the area for the drone to travel through. These waypoints will have been calculated so that the route taken by the drone is as efficient as possible.
The tool has been developed in a web environment as this way it can be ensured that it works on a wide variety of devices regardless of their specifications. The application has been designed with a simple and intuitive interface so that users can quickly adapt to it.
Javascript, HTML, CSS, Bootstrap and some APIs from the Google suite such as Maps and Geolocator have been used for the development.

Keywords: Dron, Maps, Google, Javascript, API, Coordinates...

Índice general

Índice de figuras

Capítulo 1: Introducción

1.1. Antecedentes

En los últimos años se han encontrado una gran cantidad de nuevos usos para los drones gracias a su increíble utilidad y versatilidad. Al principio el uso de los drones se dividía en dos categorías: fotografía y rastreo de zonas. En la actualidad se pueden encontrar en:

- Fotografía profesional
- Vídeos de eventos
- Extinción de incendios
- Seguridad civil
- Búsqueda de personas desaparecidas
- Control del tráfico y de las carreteras
- Control de cosechas
- Fumigación a través de drones
- Reparto de pedidos

El ámbito de este trabajo se centra exclusivamente en el rastreo de zonas ya sea para la búsqueda de personas u objetos o para detallar la misma. El problema es que en la actualidad existen una serie de aplicaciones que pueden ayudar con esto pero tienen una serie de limitaciones que se podrían mejorar. Las limitaciones son mayoritariamente dos: exclusividad en el sistema operativo en el cual se encuentra la aplicación y ecosistemas cerrados.

La primera se debe a que la gran mayoría de aplicaciones que hay ahora mismo en el mercado no son multiplataformas y están ligadas a un sistema específico, siendo estos sistemas en su mayoría iOS y Android, aunque también hay algunas empresas que crean su propio sistema cerrado para el uso de dicha aplicación.
La segunda limitación está en parte relacionada con la primera, debido a que las empresas que suelen hacer estas aplicaciones son la misma que crean los

drones y dichas aplicaciones solo funcionan con los drones de la empresa limitando las opciones del usuario.

1.2. Objetivos

El objetivo de este proyecto es crear una aplicación que sea lo suficientemente general para que cualquier persona lo pueda usar independientemente del dispositivo, ya sea móvil u ordenador personal, que posea. También la aplicación puede ser usada independientemente del dron que posea el usuario.
La aplicación será una web para que puedan acceder a ella ~~una~~ gran cantidad diferentes de dispositivos y así no tener que depender del sistema operativo o de las características del dispositivo que quiera usar el usuario. Debido a que la aplicación tiene que funcionar en una gran variedad de dispositivos, la interfaz tiene que ser sencilla y compatible con dispositivos con pantallas táctiles pero sin sacrificar funcionalidad en dispositivos sin ellas, como puede ser un ordenador personal.
El objetivo de esta aplicación será la creación de un fichero con la ruta que el dron deberá realizar para poder pasársela al controlador del dron y que este pueda llevar a cabo el viaje.
La creación de la ruta se puede hacer desde dos puntos de vista diferentes:

- Interesa tener una ruta lo más eficiente, en otras palabras, que el dron examine toda el área con la menor distancia posible para aprovechar al máximo el rango de la batería del dron y no importe el tiempo de cálculo de la ruta de la aplicación.
- En este caso, interesa también la ruta más optimizada posible pero también nos importa lo que tarde la aplicación en crear la ruta para el viaje del dron ya que lo queremos lo más rápido posible.

Este trabajo se enfocará en el segundo caso, que se puede dar en una situación de emergencia como, por ejemplo, en el rescate de una persona.

Capítulo 2: Estado del arte

En la actualidad las aplicaciones que hay en el mercado se basan en su gran mayoría en la recopilación de datos o el análisis de imágenes. Las aplicaciones que salen de estos dos grandes grupos suelen estar hechas para acciones muy específicas como, por ejemplo, *Tesla Field Recorder* que se usa para medir interferencias magnéticas.
En la actualidad no existe realmente una aplicación conocida que se use para crear vuelo de forma automática para drones, hay aplicaciones que te permiten programar vuelos pero en estos casos tiene que ser el propio usuario el cual vaya creando punto por punto todos los *waypoints* que recorrerá el dron.

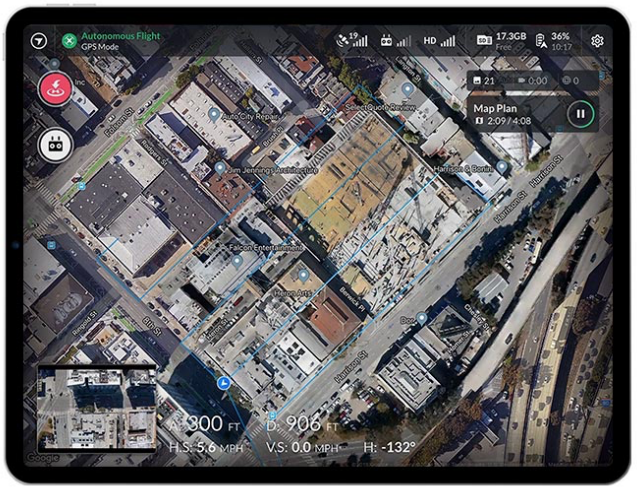
2.1. Aplicaciones existentes

DJI GO

La aplicación te permite controlar el dron, grabar y compartir los contenidos desde tu dispositivo iOS.
La aplicación también te proporciona datos meteorológicos y un mapa de google para poder planear viajes automáticos. También permite la configuración tanto del propio dron como de la cámara.

2.1.1. Imagen DJI GO.

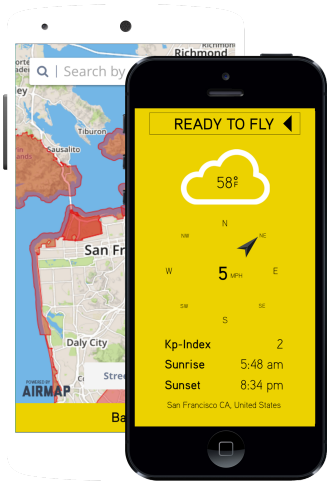
DroneDeploy



DroneDeploy es una de las aplicaciones de aviones no tripulados más populares para los pilotos que desean mapear y hacer modelos 3D usando la imagen tomada con su dron.
La aplicación te permite subir datos al servidor para que sean analizadas. También tiene la opción de proporcionar mapas de calor, mediciones detalladas para una inspección del techo y más.

2.1.2 Image DroneDeploy.

Hover



Hover es otra aplicación de drones que puedes usar para planificar tus misiones de vuelo. Utiliza los datos de Airmap y contiene un sistema de pronóstico de tiempo incorporado.
También viene con noticias para mantenerte actualizado sobre temas actuales relacionados con el gobierno, las reglamentaciones y las leyes.

2.1.3. Imagen Hover.

2.2. Tecnologías usadas

Las tecnologías usadas en este proyecto han sido las siguientes:

- Javascript (Backend, Frontend)
    - Javascript es un lenguaje de programación ligero, interpretado, orientado a objetos y es débilmente tipado. Javascript es usado en entornos web para dotar a estas de mejoras en la interfaz de usuario y poder crear páginas webs dinámicas e interactivas.
- HTML (Frontend)
    - HTML es un lenguaje de marcas que se define como el componente más básico de la Web. Define el significado y la estructura de una página web.
- CSS (Frontend)
    - CSS es un lenguaje de estilos utilizado para describir la presentación de documentos HTML o XML.
- Google APIs (Backend, Frontend)
    - Son una serie de herramientas que ofrece Google a los programadores para que puedan implementar funcionalidades de sus diferentes aplicaciones en una externa.
- Bootstrap (Frontend)
    - Bootstrap es una biblioteca multiplataforma o conjunto de herramientas para diseño de sitios web y aplicaciones.

Dentro de las APIs de Google solo se han usado 3 de las muchas que se ofrecen, en este caso han sido:

- Maps
    - Esta se ha usado para crear el mapa dentro de la página web y poder crear funciones para interactuar con él, ya sea creando marcadores o polígonos.
- Geolocator
    - Esta herramienta ha servido para poder sacar los datos necesarios de la posición del usuario, como por ejemplo, la posición actual del mismo
- Geometry
    - Esta herramienta se ha usado para poder sacar los datos necesarios de los polígonos creados por el usuario para poder calcular el viaje del dron.

Capítulo 3: Algoritmos utilizados

Antes de hablar de los algoritmos hay que tener varias cosas en cuenta. La primera es que tras tener el área dada por el usuario y el punto inicial, se va a crear una malla de puntos sobre dicha área. Estos puntos son los que luego se les pasarán a los algoritmos de búsqueda para que recorran los puntos que estén dentro del área. Un dato a tener en cuenta es que la distancia entre estos puntos es constante y esa distancia viene dada por la siguiente fórmula:

$$D = h \times tg(\frac{fov}{2})$$

Siendo D la distancia entre los puntos, h la altura del vuelo y fov es el campo de visión de la cámara.
Los algoritmos utilizados en este proyecto se van a dividir de la siguiente manera:

- Algoritmos usados en la interfaz
- Algoritmos para el cálculo de la ruta
    - Algoritmos de clusterización
    - Algoritmos de búsqueda

3.1. Algoritmos de la interfaz

Los algoritmos relacionados con la interfaz son los que permitirán al usuario una interacción más sencilla con la aplicación. Ya sea para poder ver reflejados en el mapa los datos que introducen, como la dirección del viento, o poder generar polígonos dentro del mapa de una manera rápida y sencilla.

3.1. Quickhull

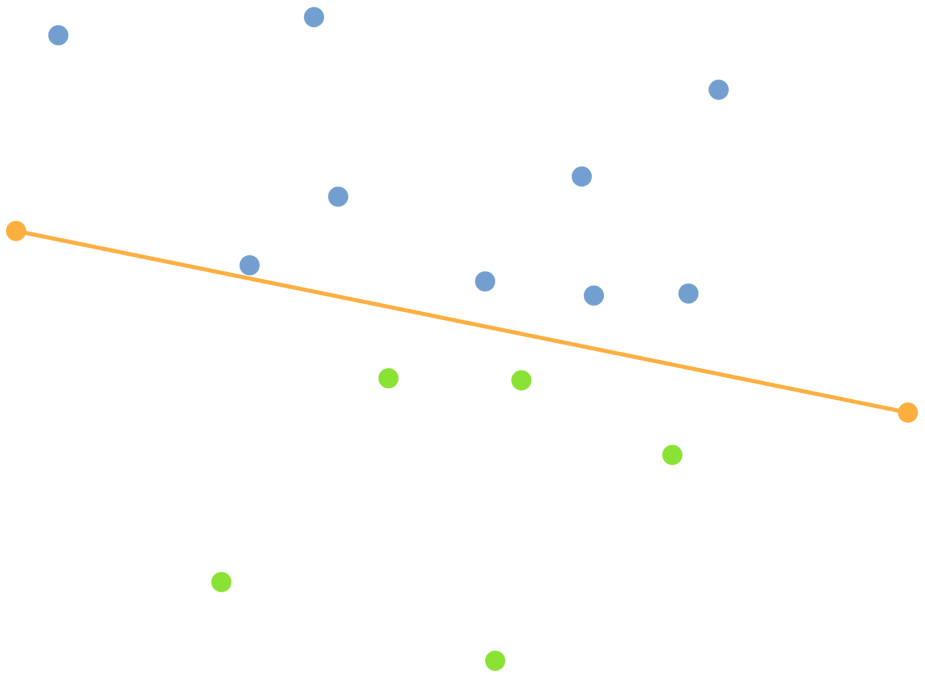
Quickhull es un método que se usa para calcular el cierre convexo de una nube de puntos en un plano 2D. El nombre del algoritmo deriva de que usa una técnica de *divide y vencerás* parecida al del algoritmo de ordenación *quicksort.*
El algoritmo funciona de una manera bastante óptima para situaciones normales pero dónde el algoritmo se puede volver bastante intensivo a la hora de usar recursos, es cuando la posición de todos los puntos de la nube estén de tal forma que tenga un gran porcentaje de simetría.

La complejidad algorítmica del algoritmo es θ (n log n) aunque en el caso de que tenga un alto porcentaje de simetría puede llegar a ser θ (n^2).

Funcionamiento

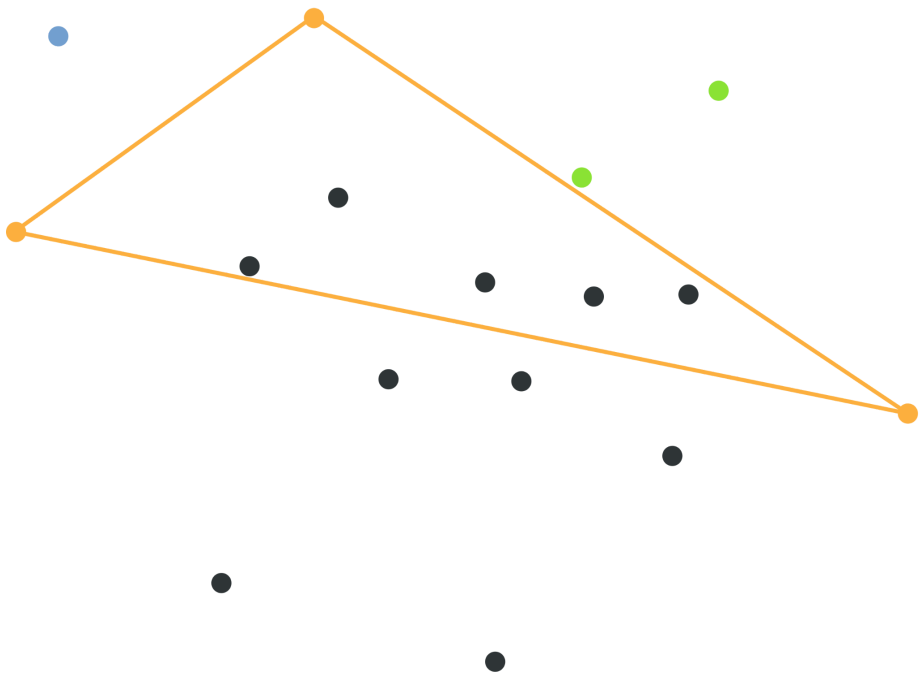
El funcionamiento de este algoritmo se puede dividir en 6 fases:

- Busca los puntos con la menor y mayor coordenada X ya que estos siempre estarán en el cierre convexo
- Trazar una línea entre esos dos puntos para dividir la nube en dos grupos



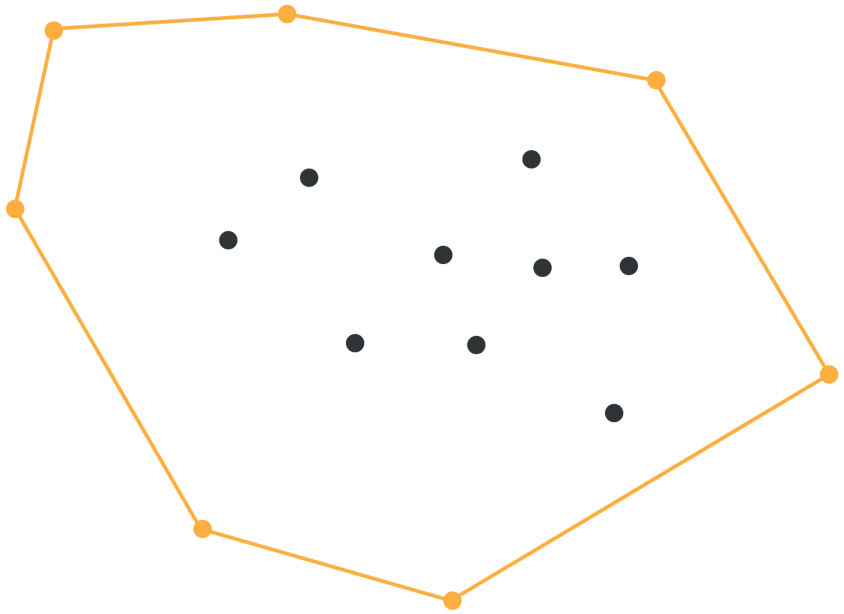
3.1.1. Imagen Quickhull 1.

- Buscar el punto más alejado a esta línea y juntarlo para crear un triángulo



3.1.2. Imagen Quickhull 2.

- Descartar los puntos que están dentro del triángulo.
- Repetir los dos pasos anteriores en todos los lados del triángulo hasta que no queden puntos que clasificar



3.1.3. Imagen Quickhull 3.

Código

```
export function chainHull_2D(P, n, H) {
 let bot = 0,
 top = (-1);
 let i;
 let minmin = 0, minmax;
 let xmin = P[0].lng();
 for (i = 1; i < n; i++) {
     if (P[i].lng() !== xmin) {
         break;
     }
 }
 minmax = i - 1;
 if (minmax === n - 1) {
     H[++top] = P[minmin];
     if (P[minmax].lat() !== P[minmin].lat())
         H[++top] = P[minmax];
     H[++top] = P[minmin];
     return top + 1;
 }
 let maxmin, maxmax = n - 1;
 let xmax = P[n - 1].lng();
 for (i = n - 2; i >= 0; i--) {
     if (P[i].lng() !== xmax) {
         break;
     }
 }
 maxmin = i + 1;
 H[++top] = P[minmin];
 i = minmax;
 while (++i <= maxmin) {
     if (isLeft(P[minmin], P[maxmin], P[i]) >= 0 && i < maxmin) {
         continue;
     }
     while (top > 0) {
         if (isLeft(H[top - 1], H[top], P[i]) > 0) {
             break;
         }
         else {
             top--;
         }
     }
     H[++top] = P[i];
}
```

```
 if (maxmax !== maxmin) {
     H[++top] = P[maxmax];
 }
 bot = top;
 i = maxmin;
 while (--i >= minmax) {
     if (isLeft(P[maxmax], P[minmax], P[i]) >= 0 && i > minmax) {
         continue;
     }
     while (top > bot) {
         if (isLeft(H[top - 1], H[top], P[i]) > 0) {
             break;
         }
         else {
             top--;
         }
     }
     H[++top] = P[i];
 }
  if (minmax !== minmin) {
     H[++top] = P[minmin];
 }
 return top + 1;
}
```

3.2. Algoritmos para el cálculo de la ruta

Los algoritmos para el cálculo están divididos en dos fases, la fase de división y la fase de búsqueda de la ruta óptima.

3.2.1. División

Hay un caso que se tiene que contemplar a la hora de intentar calcular la ruta y es que el área a recorrer no es mayor que el área estimada del dron que calculamos al principio. En el caso de que esto se dé, tendríamos que dividir el área inicial en otras más pequeñas las cuales se recorrerán individualmente y generarán ficheros únicos para cada una de ellas. Estas nuevas “áreas” a recorrer serán circunferencias cuyo diámetro será la distancia máxima que puede recorrer el dron de 1 solo viaje.
Lo único que se necesita para crear las circunferencias es la coordenada del centro y el radio. El radio es fácil de calcular, ya que es la mitad de la distancia máxima que puede recorrer el dron, pero donde hay más complejidad es a la hora de calcular los centros.
La forma que se usa para hallar los centros es a través del algoritmo *k-means.* El *k-means* es un algoritmo de clasificación no supervisada (*clusterización*) que agrupa objetos en *k* grupos basándose en alguna de sus características. En

este caso, la característica que vamos a usar es la distancia mínima entre cada objeto y la distancia del objeto al centroide de su *cluster.*

Al *k-means* se le pasará una nube de puntos generada aleatoriamente dentro del área para el cálculo de la agrupación. Para esta solución no nos interesa tanto los puntos agrupados como los *centroides* de cada grupo, ya que estos *centroides* es lo que luego se usará como punto central para crear las circunferencias.

Funcionamiento

- Calcular el número de grupos que queremos. Esto se hará dividiendo el área total de la zona a recorrer entre el área máxima que puede recorrer el dron con la carga completa de la batería. El número de grupos siempre será el resultado de esa división más uno, para tener la seguridad de que se recorre el área completa.
- Se establecen k *centroides* de forma aleatoria entre todos los puntos.

```
function getRandomCentroids(dataset, k) {
 // selects random points as centroids from the dataset
 const numSamples = dataset.length;
 const centroidsIndex = [];
 let index;
 while (centroidsIndex.length < k) {
   index = randomBetween(0, numSamples);
   if (centroidsIndex.indexOf(index) === -1) {
     centroidsIndex.push(index);
   }
 }
 const centroids = [];
 for (let i = 0; i < centroidsIndex.length; i++) {
   const centroid = [...dataset[centroidsIndex[i]]];
   centroids.push(centroid);
 }
 return centroids;
}
```

- A cada objeto se le asigna su *centroide* más cercano.

```
function getLabels(dataSet, centroids) {
 const labels = {};
 for (let c = 0; c < centroids.length; c++) {
   labels[c] = {
     points: [],
     centroid: centroids[c],
   };
 }
 for (let i = 0; i < dataSet.length; i++) {
   const a = dataSet[i];
   let closestCentroid, closestCentroidIndex, prevDistance;
   for (let j = 0; j < centroids.length; j++) {
```

```
      let centroid = centroids[j];
      if (j === 0) {
        closestCentroid = centroid;
        closestCentroidIndex = j;
        prevDistance = getDistanceSQ(a, closestCentroid);
      } else {
        // get distance:
        const distance = getDistanceSQ(a, centroid);
        if (distance < prevDistance) {
          prevDistance = distance;
          closestCentroid = centroid;
          closestCentroidIndex = j;
        }
      }
    }
    labels[closestCentroidIndex].points.push(a);
  }
  return labels;
}
```

- Se actualizan las posiciones de los *centroides* de cada grupo tomando como nuevo la posición del promedio de los objetos pertenecientes a dicho grupo.

```
function recalculateCentroids(dataSet, labels, k) {
  let newCentroid;
  const newCentroidList = [];
  for (const k in labels) {
    const centroidGroup = labels[k];
    if (centroidGroup.points.length > 0) {
      // find mean:
      newCentroid = getPointsMean(centroidGroup.points);
    } else {
      // get new random centroid
      newCentroid = getRandomCentroids(dataSet, 1)[0];
    }
    newCentroidList.push(newCentroid);
  }
  return newCentroidList;
}
```

- Se repiten los pasos 3 y 4 hasta que los *centroides* no se mueven, se mueven por debajo de una distancia umbral en cada paso o se llegue a un máximo de iteraciones.

El resto del código se puede encontrar en el siguiente enlace de github.

3.3. Búsqueda

Una vez calculado los puntos que se van a recorrer, en el caso de que no haya división serán todos los puntos que estén dentro del polígono o en el caso de que se haya dividido el polígono, todos los puntos que estén dentro de la circunferencia y del polígono original al mismo tiempo. Con estos puntos podremos pasarlos al algoritmo de búsqueda para calcular la forma más eficiente en términos de distancia de recorrer todos los puntos. Los dos algoritmos que se han probado y estudiado para este problema son:

- A* modificado
- TSP (Problema del viajante)

*3.3.1. A**

El A* es un algoritmo de búsqueda en grafos heurístico que busca el camino más corto desde un punto inicial a un punto meta. El problema que tenemos con el uso de este algoritmo es que no siempre pasa por todos los puntos a recorrer. El algoritmo se ha modificado para que se puedan volver a visitar nodos que ya han sido recorridos. Esto puede llegar a generar bucles dentro la ruta, aumentando tanto el tiempo de ejecución como la distancia recorrida. En el peor caso estos bucles pueden llegar a considerarse “infinitos” que es que el programa se quede visitando solamente 3 o más nodos sin llegar a visitar el resto.

Funcionamiento A*

- Se crea una lista con el punto inicial.
- Se añade a la lista el dron cuya distancia heurística sea la menor con respecto al último punto de la lista.
- Se comprueba si el punto que se ha añadido es el punto meta
- En el caso de que no sea así, se vuelven a repetir los pasos 2 y 3 hasta que se llegue al punto meta.

Cuando se repita el paso 2 tenemos que bloquear la posibilidad de que el dron vuelva al punto del cual viene en la iteración anterior ya que se quiere evitar la posibilidad de crear un bucle infinito entre dos puntos.

3.3.2. TSP

El problema del viajante o TSP (*Traveller Salesman Problem*) es un algoritmo que da solución a la siguiente pregunta: *Dada una lista de ciudades y las distancias entre cada una de ellas ¿Cuál es la forma más eficiente de pasar por cada una de las ciudades y volver al punto original?.* Este caso es exactamente el que se trata en este trabajo, la dificultad reside en que este problema es *NP-Hard,* esto significa que para encontrar la solución óptima nos enfrentamos a una solución que no es de orden polinómico por lo cual puede llegar a ser muy costosa de calcular si el tamaño del problema se agranda.
Con el paso del tiempo se han encontrado soluciones heurísticas que nos pueden dar una solución lo suficientemente óptima en un tiempo de cómputo razonable. Este caso es perfecto para la solución que buscamos en este

trabajo ya que nuestro tamaño del problema nunca será muy grande pues en el caso de que tenga un número muy elevado de puntos para recorrer, estos se dividirán en grupos más pequeños.
La implementación por la cual se ha optado es la que se denomina como *NN* o *algoritmo del vecino más próximo.* Esta implementación permite al algoritmo elegir la ciudad no visitada más cercana siempre como próximo movimiento.

Funcionamiento NN

- Se crea una lista de nodos visitados, se añade el nodo inicial a la lista y se pone el nodo inicial como el nodo estudiado.
- Se mira cuál es el siguiente nodo no visitado más cercano al nodo no estudiado y se mete en la lista de visitados. Este nodo se transforma en el nodo estudiado.
- Comprobamos si la lista de visitados es igual a la lista de nodos totales.
- En el caso de que no sean iguales se repite el paso 2 hasta que las dos listas sean iguales.

3.3.3. Comparación entre ambos

Para elegir qué algoritmo utilizar en el trabajo se ha hecho un estudio con casos estandarizados en una serie de dispositivos diferentes y se ha comparado los resultados, en este caso nos interesa tanto la distancia recorrida como el tiempo de ejecución.
El caso estandarizado es la misma área la cual recorreremos usando 10000, 25000, 50000 y 100000 puntos.

Dispositivos utilizados

- Macbook Pro 16” M1 Pro 16GB RAM
- Samsung Galaxy S21 Ultra
- Ipad Pro (2021)
- Un ordenador de sobremesa con las siguientes especificaciones:
    - Ryzen 7 5800x
    - 64 GB RAM (3600 MHz)
    - RTX 3080
    - Gigabyte X570S Aero G
    - Samsung SSD 970 Pro NVME

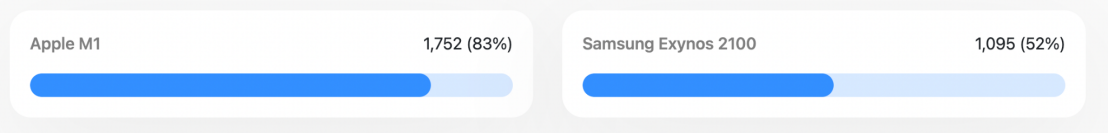
Gráficas

Las gráficas se van a dividir en dos grupos, por un lado tendremos los dispositivos móviles (Samsung S21 Ultra e iPad Pro 2021) y por otro lado tendremos los ordenadores (PC, MacBook Pro). Lo que influenciará principalmente en los resultados de estas pruebas es la velocidad de cómputo con un solo núcleo de los procesadores y la velocidad de la memoria interna de cada dispositivo. Esto se debe a que la aplicación no fue diseñada con una ejecución multihilo en mente.

Dispositivos móviles

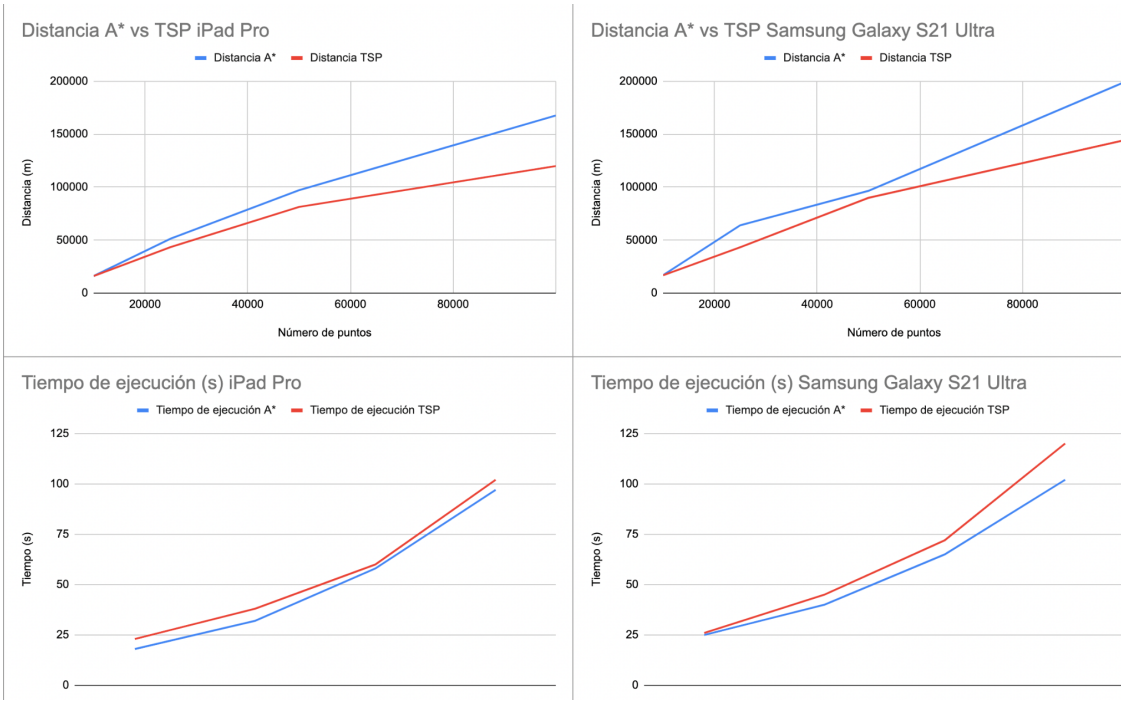
La primera comparación se hará entre ambos dispositivos móviles y servirá para comprobar, no solo el rendimiento de la aplicación, sino también la compatibilidad entre diferentes sistemas operativos, ya que se usará *Android* y *iPadOS*.

Lo primero a tener en cuenta es que, a pesar de que ambos son dispositivos móviles, no están dentro del mismo rango de potencia computacional. El Samsung S21 Ultra tiene un *Exynos 2100* mientras que el iPad Pro tiene el chip M1. Si comparamos ambos procesadores usando una serie de *benchmarks* estandarizados, como *Geekbench 5,* se puede ver rápidamente que el M1 es de media un 30% más rápido que el *Exynos 2100*.



3.3.3.1. Imagen Comparación M1 Exynos 2100.

Esto se puede ver en cierta medida en los tiempos de ejecución de los algoritmos donde el M1 tiende a tardar entre un 8 a un 20% menos tiempo en dar una solución. Esto no se puede apreciar en los resultados de las pruebas realizadas debido al tamaño del problema. En las pruebas donde el tamaño es menor no se puede apreciar diferencia ninguna entre los procesadores debido a que ambos son lo suficientemente potentes. Para poder ver alguna diferencia notable de potencia entre los dos procesadores tendremos que ir a la ejecución con más puntos o incluso realizar una prueba todavía mayor. Cuanto mayor sea el número de puntos más nos acercaremos al 30%.



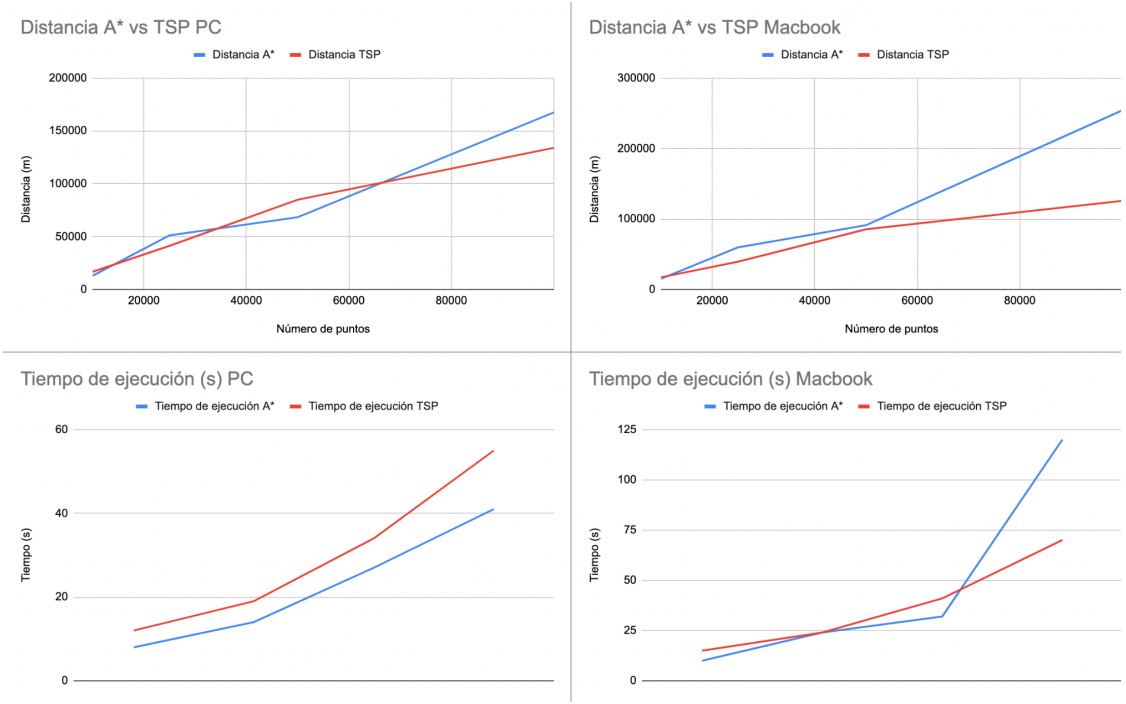
3.3.3.2. Imagen Comparación dispositivos móviles.

Ordenadores

Al contrario que con los dispositivos móviles, en esta comparación se usará hardware mucho más potente, eso significa que no se podrá ver la diferencia entre los dos dispositivos pero al eliminar esa variable, se podrá apreciar claramente la diferencia entre los dos algoritmos.

Una cosa que puede interferir en el rendimiento es que el procesador dentro del MacBook Pro (M1 Pro) no está creado usando la misma tecnología que el Ryzen 7 5800x, siendo este un procesador de la familia x86 y el M1 Pro es un procesador ARM. La gran diferencia entre estos dos tipos de tecnologías es que los procesadores ARM están desarrollados con la mentalidad de tener la mayor eficiencia energética manteniendo todo el rendimiento posible, mientras que el objetivo de los procesadores x86 (todos los que se llevan utilizando en ordenadores desde los años 90) es obtener el mayor rendimiento posible a coste de una eficiencia bastante menor. Los procesadores ARM normalmente se suelen encontrar en dispositivos móviles donde la eficiencia es realmente importante.

Este dato es importante ya que al pasar esta tecnología a un entorno completamente diferente como los ordenadores portátiles, Apple tuvo que crear una capa de traducción de instrucciones x86 a ARM llamada *Rosetta.* En el caso de que la aplicación no haya sido traducida por los desarrolladores para funcionar con ARM, esta deberá pasar por la capa de traducción. Para el proyecto esto es importante porque la librería de Google Maps de javascript todavía no tiene una versión estable para ARM, esto puede generar ejecuciones más lentas ya que tiene que pasar por Rosetta.



3.3.3.3. Imagen comparación ordenadores.

Discusión de resultados

Al igual que las pruebas, los resultados se tienen que ver por separado también. Por un lado, están los resultados de los dispositivos móviles, figura 3.3.3.2., los cuales son importantes para el estudio debido al objetivo que se ha marcado al principio del proyecto, siendo este el poder crear una ruta eficiente de la manera más rápida posible en cualquier momento. Es muy importante que la aplicación funcione correctamente en dispositivos móviles ya que será en estos dónde se ejecute la mayoría de los casos.

Por otro lado tenemos los resultados de los dos ordenadores, donde la potencia no es un problema, estos resultados servirán para poder tomar una mejor decisión a la hora de elegir los algoritmos.

Viendo los resultados de los cuatro dispositivos, figuras 3.3.3.2. y 3.3.3.3. se pueden ver una serie de patrones que se repiten en los cuatro. Primero se observa que el tiempo de ejecución es menor usando el A* en todos los casos menos en uno, donde el tiempo de ejecución en el MacBook Pro se dispara de una manera casi exponencial. Esto se debe al cambio que se le tuvo que hacer al A*, el cual permite que los nodos se puedan volver a visitar. Debido a esto se puede dar la situación en la cual se generen bucles y aumente en gran medida el tiempo de ejecución y la distancia recorrida, siendo esto justo lo que pasó en el caso de las pruebas con el MacBook Pro. Con respecto a la distancia recorrida se puede observar que, en general, el TSP siempre acaba realizando

el viaje en menos distancia que el A*. Esto se debe a que, de nuevo, para que el A* recorra todos los puntos es necesario que casi siempre haya una vuelta a un nodo ya visitado, y esto hace que sume una distancia extra que no es necesaria.

Tras haber realizado todas las comparaciones y haber estudiado los resultados, se ha seleccionado el TSP como el algoritmo que se va a utilizar en la aplicación. Aunque tarde de media más tiempo en ejecutar, la diferencia no es tan grande como para que sea importante y al mismo tiempo, el TSP no puede entrar en bucles como el A*. En caso de que el A* solo realice un bucle entre pocos nodos no debería ser un problema, pero también pueden haber casos, como en las pruebas del MacBook Pro, donde el bucle es realmente grande y esto puede llegar a dar un resultado que no sirve dada la gran distancia que recorre.

Código del TSP (NN)

El primer aspecto a tener en cuenta, es que en vez de trabajar con puntos dentro de una grid o puntos dentro de un canvas, en este problema se ha trabajado usando coordenadas geográficas. Esto significa que a la hora de calcular las distancias entre los nodos se ha necesitado una forma distinta de hacerlo. Para este proyecto se ha implementado la fórmula de *Haversine*.

```
calculateCostToCity(startPoint, finalPoint) {
   if (startPoint.x == finalPoint.x && startPoint.y == finalPoint.y) {
     return 0;
   }
   const radlat1 = (Math.PI * startPoint.x) / 180;
   const radlat2 = (Math.PI * finalPoint.x) / 180;
   const theta = startPoint.y - finalPoint.y;
   const radtheta = (Math.PI * theta) / 180;
   let dist =
     Math.sin(radlat1) * Math.sin(radlat2) +
     Math.cos(radlat1) * Math.cos(radlat2) *   Math.cos(radtheta);
    if (dist > 1) {
     dist = 1;
   }
   dist = Math.acos(dist);
   dist = (dist * 180) / Math.PI;
   dist = dist * 60 * 1.1515;
   dist = dist * 1.609344; //convert miles to km
   dist = dist * 1000; //km to m
   return dist;
 }
```

Una vez que es posible calcular la distancia entre dos coordenadas, solo queda implementar el algoritmo NN (*Nearest Neighbour*).

Primero ~~cogemos~~ el nodo inicial, en nuestro caso, viene dado por el usuario en la interfaz y ~~lo marcamos~~ como visitado. Luego se empieza un bucle para recorrer todos los nodos y el nodo que tenga la menor distancia al nodo actual. Una vez se encuentre, se marcará como visitado y se volverá a calcular la distancias a todos los nodos y se volverá a coger la menor. Seguiremos haciendo esto hasta que se hayan visitado todos los nodos. El código desarrollado se muestra a continuación.

```
 setPath() {
   let currentPoint = this.startPoint;
   this.path.push(this.grid[this.startIndex]);
   this.visited[this.startIndex] = true;
   let nextPoint = null;
   let currentIndex = this.startIndex;
   while (this.numVisited < this.grid.length) {
     let minCost = Number.POSITIVE_INFINITY;
     for (let i = 0; i < this.grid.length; i++) {
       if (!this.visited[i]) {
         let tempCost = this.calculateCostToCity(this.grid[currentIndex],
this.grid[i]);
         if (tempCost < minCost) {
           minCost = tempCost;
           nextPoint = this.grid[i];
         }
       }
     }
     if (minCost === Number.POSITIVE_INFINITY) {
       break;
     } else {
       currentIndex = this.getIndexOf(nextPoint);
       this.cost += minCost;
       this.path.push(nextPoint);
       this.visited[currentIndex] = true;
       currentPoint = nextPoint;
       this.numVisited++;
     }
   }
   if (this.calculateCostToCity(currentPoint, this.grid[this.startIndex]) ===
Number.POSITIVE_INFINITY) {
     this.path = [];
     this.cost = 0;
   } else if (this.path.length < this.grid.length) {
     this.path = [];
     this.cost = 0;
   }
   return this.path;
 }
};
```

El resto del código se puede ver en el siguiente enlace a github.

Capítulo 4: Funcionamiento de la aplicación

4.1. Datos necesarios

Antes de poder crear el archivo con la serie de *waypoints,* es necesario que el usuario proporcione una serie de datos. Los datos básicos que tiene que proporcionar el usuario son los siguientes:

- Velocidad del vuelo
- Altura del vuelo
- Autonomía del dron
- FOV de la cámara

Aparte de esos datos, también es necesario que el usuario le ponga un nombre al dron y proporcione la dirección del viento que se pasará al programa como un ángulo respecto al norte. La dirección del viento tendrá un impacto a la hora del cálculo ya que se rotará la figura para que la arista más larga del polígono que encierra el área de búsqueda esté en un ángulo de 90º con respecto a la dirección del viento. El nombre solo servirá para mostrarlo en la interfaz y permitir al usuario saber qué dron está seleccionando.
Estos datos se usarán para calcular la superficie aproximada que puede recorrer el dron usando, como máximo, un 70% de la batería. Nos interesa guardar mínimo un 30% del total de la batería en el caso de que el dron esté fuera del área a explorar y necesitemos que se mueva al área, la explore y luego vuelva a su posición inicial. Debido a esto, la superficie aproximada será un dato necesario para poder crear la ruta. Esta se podrá calcular gracias a las pruebas experimentales que realizó el grupo CryptUll, las cuales derivaron en las siguientes fórmulas:

$$Sd = 2 \times h \times v \times mt \times tg(\frac{fov}{2})$$

4.1.1. Imagen Fórmula Superficie.

Los datos de la fórmula son los siguientes:

- h: Altura del vuelo (m)
- v: Velocidad del vuelo (m/s)
- fov: Campo de visión de la cámara (°)
- mt: Tiempo de la misión (s)

Mt se calcula usando siguiente fórmula:

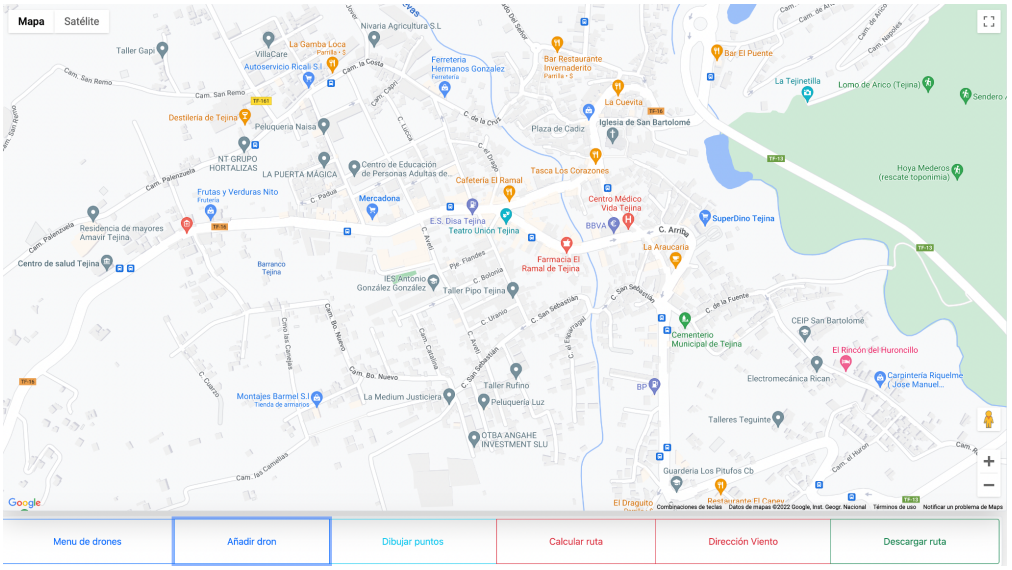
$$mt = Autonomia \times 60 \times 0.7$$

4.1.2. Imagen Fórmula tiempo de la misión.

La autonomía es el resultado de la división de la capacidad de la batería entre el consumo medio estimado de la misma. Este dato tiene que ser proporcionado por el usuario en minutos.

4.2. Interfaz

El objetivo de la interfaz es que sea sencilla y se pueda usar en todo tipo de dispositivos, tanto pantallas táctiles como ordenadores de sobremesa y portátiles.
Para conseguir esto dividiremos la interfaz en dos partes. La primera será el mapa donde el usuario dibuja el área a recorrer y, una segunda parte que es un conjunto de botones donde podrán activar y desactivar diferentes funcionalidades o introducir datos en una serie de ventanas que se les abrirán al pulsar los botones. También tendremos un menú a la derecha que podemos abrir para ver los datos de los drones que hemos introducido.

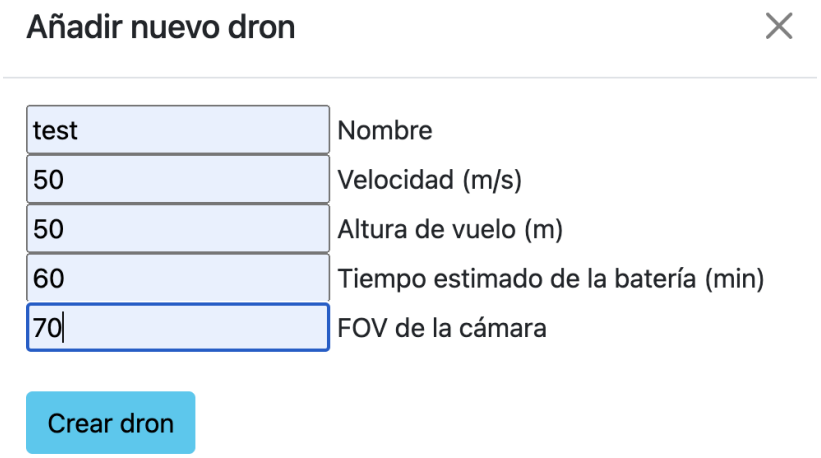


4.2.1. Imagen Interfaz General.

4.3. Ejemplo de funcionamiento

Un posible ejemplo de uso de la aplicación sería el siguiente:

- Lo primero que hacemos es crear el dron con los datos necesarios para la ejecución.



4.3.1. Imagen Interfaz menú añadir dron.

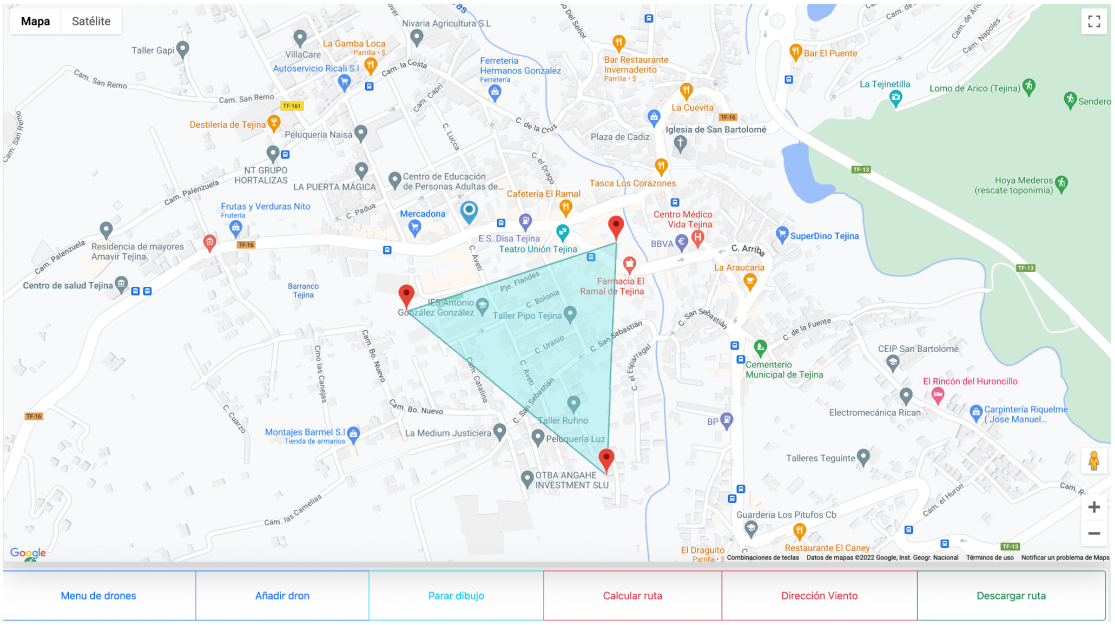
- Luego abrimos el menú de drones y activamos el dron que vamos a usar

4.3.2. Imagen Interfaz Menú de drones.

- Ponemos la dirección del viento con respecto al norte (Opcional)

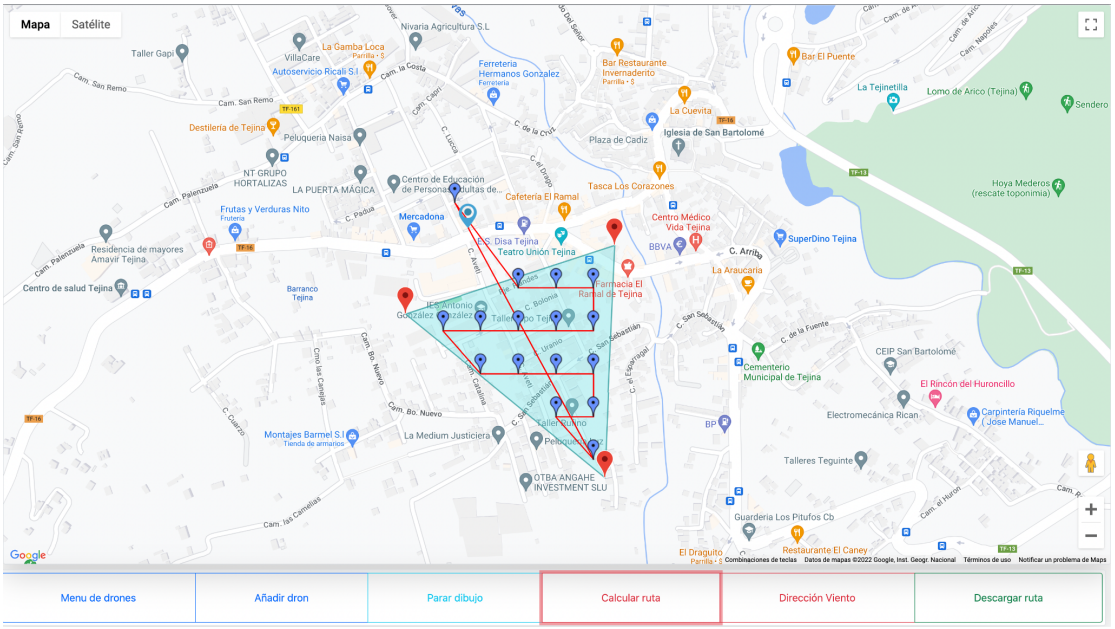
4.3.3. Imagen Interfaz Menú añadir dirección del viento.

- Creamos el punto inicial (el punto azul) y el área a explorar



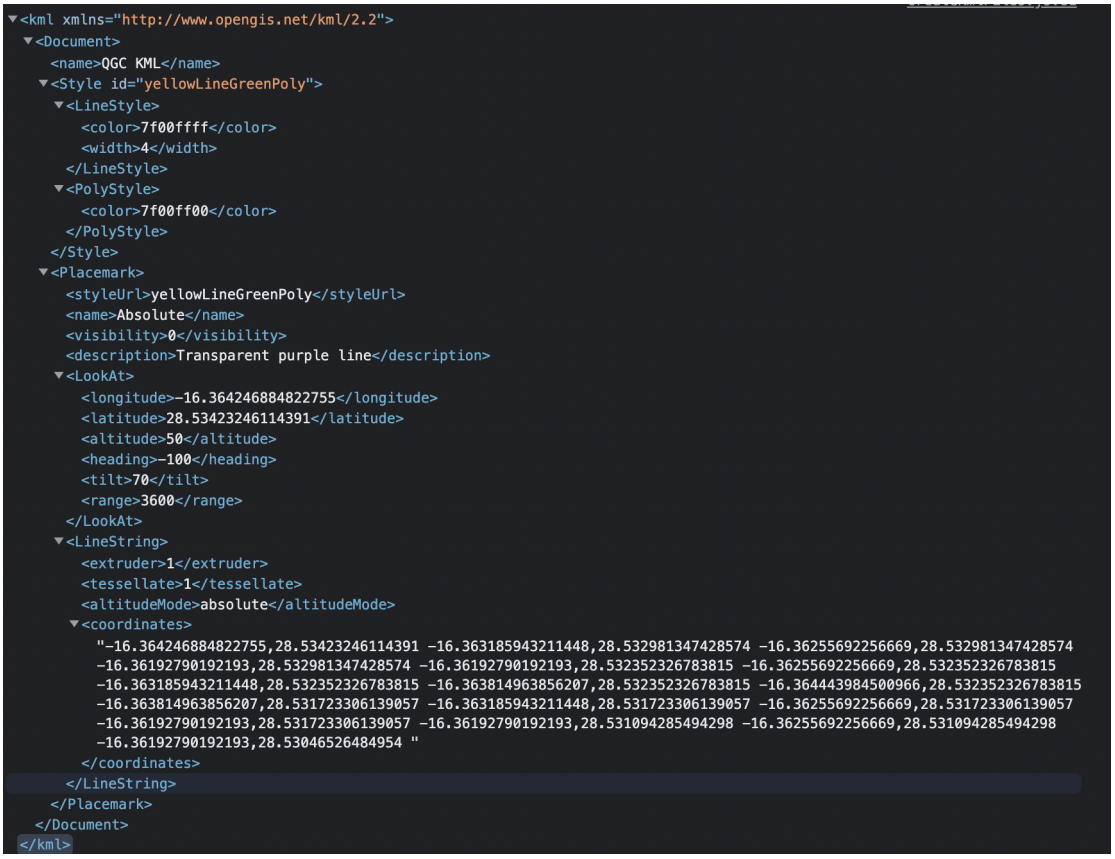
4.3.4. Imagen interfaz área dibujada.

- Pulsamos el botón para calcular la ruta y descargamos el fichero.



4.3.5. Imagen Interfaz ruta calculada.

- Una vez se haya descargado el fichero, este se verá de la siguiente manera



4.3.6. Imagen fichero de salida.

- Por último se le pasa el fichero al controlador del dron para que este realice el viaje.

Capítulo 5: Conclusiones y líneas futuras

Se han cumplido todos los objetivos que se propusieron para este proyecto. La aplicación funciona en todas las plataformas que tengan acceso a un navegador web independientemente del sistema operativo que use el dispositivo. También tras probar la aplicación en una serie de dispositivos diferentes se ha podido comprobar que el objetivo de que la aplicación sea rápida pero que al mismo tiempo nos dé una solución lo suficientemente óptima se ha cumplido. La ejecución de la aplicación con todos sus pasos, esto incluye poner los datos del dron, dibujar el área y calcular la ruta, varía entre un rango de 10 segundos a 25 segundos dependiendo si el dispositivo es un ordenador o un dispositivo móvil.
Se ha desarrollado y probado la compatibilidad de un software de optimización de trayectorias para realizar búsquedas, usando diferentes APIs de otras aplicaciones, comprobando su interoperabilidad. Para el desarrollo del proyecto se tuvo que analizar pruebas de vuelo reales, con el apoyo de ACUDROPOL, para poder conseguir los datos y fórmulas necesarias para el desarrollo del mismo. Este proyecto se ha realizado en un entorno web debido a que es la forma más sencilla, a día de hoy, de hacer una aplicación multiplataforma a la cual todo el mundo pueda acceder a ella.

Como líneas futuras de posible desarrollo de la aplicación, destacan:

- En vez del uso de circunferencias para crear las nuevas “áreas” a explorar, se pueden usar los polígonos de *Voronoi* que permite la partición de un plano euclídeo y esto se puede modificar de tal manera que nos sirva para dividir la figura en trozos de una manera automática. Estos se pueden implementar con relativa facilidad con el *algoritmo de Fortune.*
- Hacer implementaciones directas con las APIs de ciertos controladores de drones para que no sea necesario descargar un fichero sino que directamente le podamos pasar la ruta al dron.

Capítulo 6: Summary and Conclusions

All the objectives that were set for this project have been met. The application works on all platforms that have access to a web browser regardless of the operating system used by the device. Also, after testing the application on a number of different devices, we have been able to verify that the objective of making the application fast but at the same time giving us a sufficiently optimal solution has been met. The execution of the application with all its steps, including entering the drone data, drawing the area and calculating the route, varies between a range of 10 seconds to 25 seconds depending on whether the device is a computer or a mobile device.

The compatibility of a trajectory optimisation software has been developed and tested to perform searches, using different APIs from other applications, checking their interoperability. For the development of the project, real flight tests had to be analysed, with the support of ACUDROPOL, in order to obtain the data and formulas necessary for the development of the project. This project has been carried out in a web environment because it is the easiest way, at present, to create a multiplatform application that can be accessed by everyone.

As future lines of possible development of the application, the following stand out:

- Instead of using circles to create the new "areas" to explore, we can use Voronoi polygons that allow the partition of a Euclidean plane and this can be modified in such a way that it serves to divide the figure into pieces in an automatic way. These can be implemented relatively easily with Fortune's algorithm.
- Make direct implementations with the APIs of certain drone controllers so that it is not necessary to download a file but we can directly pass the route to the drone.

Capítulo 7: Presupuesto

A continuación se valoran las tareas realizadas, las horas invertidas y los materiales necesarios para este proyecto. El precio de la hora de mano de obra está calculado en base de un salario bruto de 4000 €. Se va a tener en cuenta que un mes tiene 20 días laborables y que cada día de trabajo son 8 horas. Esto nos da que el precio de cada hora es de 25 €.

Recursos Humanos

Descripción	Cantidad (h)	Precio (€)	Total (€)
Planificación del proyecto	25	25	625
Lectura de documentación	50	25	1250
Desarrollo	200	25	5000
Redacción de la memoria	20	25	500
			7375

Materiales

Descripción	Cantidad	Precio (€)	Total (€)
Ordenador sobremesa	1	3200	3200
Macbook Pro M1 Pro	1	2700	2700
iPad Pro (2021)	1	780	780
Samsung Galaxy S21 Ultra	1	999	999
			7679

Bibliografía

[1] Google Developers. 2022. *Google Maps Platform Documentation | Maps JavaScript API | Google Developers*. [online] Available at: <https://developers.google.com/maps/documentation/javascript>.

[2] Bang, B., 2020. EFFICIENT METAHEURISTIC ALGORITHMS FOR THE MULTI-STRIPE TRAVELLING SALESMAN PROBLEM. *Journal of Computer Science and Cybernetics*, 36(3), pp.233-250.

[3] Kim, D., Kim, D. and Sugihara, K., 2001. Voronoi diagram of a circle set from Voronoi diagram of a point set: I. Topology. *Computer Aided Geometric Design*, 18(6), pp.541-562.

[4] Fortune, S., 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(1-4), pp.153-174.

[5] Linda Nuryanti, 2021. A VEHICLE ROUTING PROBLEM OPTIMIZATION WITH DRONE USING TABU SEARCH ALGORITHM AND ANALYTICAL HIERARCHY PROCESS. *Majalah Ilmiah Pengkajian Industri*, 15(1).

[6] Sangwan, Shabnam. (2018). Literature Review on Travelling Salesman Problem. International Journal of Research. 5. 1152.

[7] Recchiuto, Carmine & Nattero, Cristiano & Sgorbissa, Antonio & Zaccaria, Renato. (2014). Coverage Algorithms for Search and Rescue with UAV Drones - abstract.

[8] Lundin, Gustav. (2020). Supervision for drone flight safety.

[9] Mark Otto, a., 2022. *Get started with Bootstrap*. [online] Getbootstrap.com. Available at: <https://getbootstrap.com/docs/5.2/getting-started/introduction/>.

[10] Developer.mozilla.org. 2022. *JavaScript | MDN*. [online] Available at: <https://developer.mozilla.org/es/docs/Web/JavaScript/>.

[11] Developer.mozilla.org. 2022. *HTML: Lenguaje de etiquetas de hipertexto | MDN*. [online] Available at: <https://developer.mozilla.org/es/docs/Web/HTML>.

[12] Developer.mozilla.org. 2022. *CSS | MDN*. [online] Available at: <https://developer.mozilla.org/es/docs/Web/CSS>.

[13] Moises Lodeiro-Santiago, Iván Santos-González, Pino Caballero-Gil, Cándido Caballero-Gil. Secure System based on UAV and BLE for improving SAR Missions Journal of Ambient Intelligence and Humanized Computing. Springer 2017. ISI JCR Impact Factor: 1.588