Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Sistemas Autónomos de Cooperación Robótica en Vehículos Heterogéneos

Autor: Mario Aguilera Alcalde

Tutor: Dr. Pablo Muñoz Martínez

Co-tutor: Dra. Mª Dolores Rodríguez Moreno

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Sistemas Autónomos de Cooperación Robótica en Vehículos Heterogéneos

Autor: Mario Aguilera Alcalde

Tutor: Dr. Pablo Muñoz Martínez

Co-tutor: Dra. Mª Dolores Rodríguez Moreno

Tribunal:

Presidente: D. Manuel Prieto Mateo

1st Vocal: D. Eliseo García García

2nd Vocal: D. Pablo Muñoz Martínez

Dia Depositado: 11 Julio, 2022

Resumen

En este trabajo consideramos el escenario de exploración de una superficie planetaria con un sistema formado por un UAV (Unmanned Aerial Vehicle) y un UGV (Unmanned Surface Vehicle). El objetivo es alcanzar un conjunto de puntos objetivo minimizando la distancia de viaje.

Se partirá de un algoritmo ya existente llamado TERRA (utilización y colaboración de UAVs y UGVs para poder realizar envíos de paquetes de forma automática y sin interacción humana) desarrollado en Matlab para adaptarlo a Python. Además, se tratarán los diversos problemas de traducción y adaptación del lenguaje, así como aproximar la solución que propone el algoritmo TERRA a una realidad.

Palabras clave: Cooperación Robótica, TERRA, UAVs, UGVs.

Abstract

In this work we consider the scenario of exploration of a planetary surface with a system consisting of a UAV (Unmanned Aerial Vehicle) and an a system consisting of a UAV (Unmanned Aerial Vehicle) and a UGV (Unmanned Surface Vehicle). UGV (Unmanned Surface Vehicle). The objective is to reach a set of target points points while minimizing the travel distance.

It will be based on an existing algorithm called TERRA (use and collaboration of UAVs and UGVs). of UAVs and UGVs to be able to send packages automatically and without human interaction) developed in Matlab for interaction) developed in Matlab and adapted to Python. In addition, we will the various language translation and adaptation problems, as well as approximate the solution proposed by the approximate the solution proposed by the TERRA algorithm to a reality.

Keywords: Robotic Cooperation, TERRA, UAVs, UGVs.

Agradecimientos

Me gustaría agradecer en primer lugar a mi tutor Pablo Muñoz Martínez, así como mi Cotutora María Dolores Rodríguez Moreno por ayudarme y guiarme en la realización del trabajo.

Además, me gustaría agradecer a Fernando Ropero Pastor por ayudarme con la herramienta *TERRA*.

También querría dar las gracias a mis compañeros y amigos que me han ayudado y acompañado durante toda esta maravillosa y trepidante aventura en el Grado.

Finalmente, me gustaría dar las gracias a mi familia por apoyarme y acompañarme durante todos estos años, en especial a mi hermano Alejandro y a mis padres José María y Pilar sin los cuales todo esto no hubiera sido posible, así que una parte de esto es vuestro.

Índice general

Índice de figuras

Índice de tablas

Lista de Acrónimos

UAVs: **U**nmanned **A**erial **V**ehicles.

USVs: **U**nmanned **S**urface **V**ehicles.

TERRA: The coopera**T**ive **E**xplo**R**ation **R**outing **A**lgorithm.

SMOP: **S**mall **M**atlab **O**ctave **P**ython.

OMPC: **O**pen-Source **M**atlab-To- **P**ython **C**ompiler.

TSP: **T**ravelling **S**alesman **P**roblem.

UGV: **U**nmanned **G**round **V**ehicle.

BFS: **B**readth **F**irst **S**earch.

HSP: **H**itting **S**et **P**roblem.

CR: **C**ooperación **R**obótica.

ACO: **A**nt **C**olony **O**ptimisation.

GA: **G**enetic **A**lgorithm.

NNS: **N**earest **N**eighbor **S**earch.

Capítulo 1.

Introducción y objetivos

1.1. Introducción

Los últimos años han sido testigos de la difusión exponencial de la tecnología de vehículos aéreos no tripulados (*Unmanned Aerial Vehicle*, UAVs, por sus siglas en inglés). UAVs, también conocidos como drones, son hoy en día omnipresentes, abriéndose a avances e innovaciones en una variedad de casos de uso y escenarios de aplicación, incluido el control perimetral, la vigilancia remota, gestión de emergencias, entrega de mercancías y aplicaciones militares, por nombrar algunos ejemplos [3]. Teniendo en cuenta sólo la logística en el sector transporte, organismos especializados estiman un aumento del tamaño del mercado de más de 29 mil millones de dólares estadounidenses (USD) de 2022 a 2027 [4], y empresas populares como Amazon ya están probando plataformas para entregar mercancias a seleccionados clientes que utilizan drones.

Por desgracia, los UAVs representan una tecnología de doble uso. Tecnología que puede ser utilizada por entidades con un fin malicioso. De hecho, ya se han realizado varios ataques con drones tanto autónomos como pilotados a distancia que habían sido equipados con armas o explosivos [5]. A la par, las principales industrias de mercancías están retrasando la introducción de drones, debido a la notable preocupación sobre la seguridad de las personas, así como manipulación tanto de la mercancía del interior como del dron [6].

Esto se ve reflejado en la siguiente cita: ***'Alguien va a tener que morir o ser mutilado para que se tomen en serio estas cuestiones de seguridad'*** (Cheddi Skeete, ex gerente de proyectos de drones de Amazon).

Todas las preocupaciones anteriormente comentadas motivaron contribuciones de la comunidad científica. Por un lado, una variedad de soluciones para la detección de un dron que se aproxima se han introducido, basado en radar [7], reconocimiento visual de objetivos, Radio-Análisis de frecuencia (RF) y análisis de sonido [8].

En este TFG, se tratará en concreto el problema y las implicaciones que tiene la cooperación robótica centrándose en el problema de la última milla (Last Mille Delivery Problem, LMDP, por sus siglas en inglés) que consiste en la última parte de la entrega del pedido al cliente, así como el problema del viajante (TSP) [9] por sus siglas en inglés (Travelling Salesman Problem) donde dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?

Este es un problema NP-Hard dentro en la optimización combinatoria, muy importante en investigación operativa y en ciencias de la computación, así como la búsqueda de vecinos más cercanos y el problema de conjunto de cobertura, así como la optimización de rutas de tierra (UGV) y aire (UAVs).

1.2. Objetivos

Por todo esto, a fin de mejorar y reducir costes, así como mejorar la eficiencia de los envíos, este trabajo desarrollará una herramienta basada en el algoritmo ya existente TERRA [1] creado en Matlab y se traducirá a Python que dotarán al modelo una mayor portabilidad. Dicha herramienta permitirá seleccionar las diferentes ubicaciones de los puntos de reparto que tendrá que pasar el robot para la realización de la ruta y que mediante un modelo de sistemas autónomo de cooperación robótica calcule y muestre por pantalla la mejor ruta para realizar la entrega del servicio seleccionado. Todo ello es posible gracias a que TERRA dispone de un robot que sirve como estación de recarga paraque los UAVs entreguen los paquetes.

Para llevarlo a cabo, los objetivos particulares fijados serán los siguientes:

- Estudiar los algoritmos de selección de caminos o rutas cooperativas.
- Estudiar las herramientas actuales de traducción de código.
- Desarrollar el algoritmo TERRA en Python.
- Pruebas unitarias y de integración para la validación de los resultados.

1.3. Estructura del proyecto

El TFG está dividido de la siguiente forma:

- Capítulo 1: Introducción y objetivos.
- Capítulo 2: Se abordan las herramientas que se van a utilizar junto a conceptos fundamentales para comprender el desarrollo del trabajo.
- Capítulo 3: Exponer el desarrollo de la aplicación creada.
- Capítulo 4: Se plantearán los resultados tras una validación de la aplicación.
- Capítulo 5: Se presentarán las conclusiones y líneas futuras.
- Anexo 1: Se tratarán los aspectos éticos, económicos, sociales y ambientales relacionados con el proyecto.
- Anexo 2: Se analizará el proyecto desde el punto de vista monetario creando un presupuesto económico.

Capítulo 2.

Marco Teórico

En este capítulo se van a abordar los fundamentos teóricos y conceptos más importantes para entender el estado del arte, los principales problemas que existen y como se van a tratar en el proyecto.

Por un lado, se analizarán conceptos fundamentales para el desarrollo del trabajo como la Cooperación Robótica (CR), explicando sus ventajas y problemas. A continuación, se explicarán algunos de los algoritmos de planificación de rutas existen. Más tarde se verá una solución utiliznando el algoritmo TERRA. Después, se verán diferentes herramientas que se han utilizado para realizar la aplicación. Finalmente, se expondrá el funcionamiento de *TERRA*, aplicación que se ha usado como base del proyecto.

2.1. Cooperación Robótica

En esta sección se va a abordar el concepto de CR y sus diferencias respecto a la cooperación tradicional sin el uso de unidades no tripuladas. Por último, se verán cuáles son los principales problemas que tiene.

2.1.1. Visión actual

Actualmente, la CR está en un momento de contínua mejora y avance tecnológico, si bien su auge fue hace unos años, se está intentando ya encontrar la manera eficiente y real donde robots y personas puedan coexistir, cooperar y colaborar. Ya existen casos donde su uso está siendo utilizado como en el sector Automovilístico (en las fábricas de producción) o en entornos más industriales para sujetar placas muy pesadas o para realizar un control de calidad de la fabricación.

Por otro lado, la CR [10] El hombre y el robot interactúan entre sí en un campo de trabajo común. Por ejemplo, el robot coloca algo para el hombre o ambos realizan diferentes tareas al mismo tiempo en el mismo componente.

2.1.2. Problemática Cooperación Robótica

La CR está basada en una colaboración entre robot y humano donde no siempre va a ser lo idóneo, ya que al tener que colaborar con un humano se pueden perder o ver disminuidas las ventajas que nos aporta la utilización de un robot como el que no tiene por qué descansar (salvo el periodo de carga), el aguante a temperaturas extremas o condiciones adversas sin alimento o bebida.

Esto se puede entender bien con un ejemplo donde la guardia marítima quiera estar controlando la posible llegada ilegal de contrabando de sustancias ilegales con la utilización de un UAV que siendo controlado por el humano vaya con una cámara haciendo un control del mar.

En este caso nos daremos cuenta y es evidente que cada cierto tiempo tendrán que irse alternando diferentes personas para controlar el UAV o bien descansar dicha persona, por no hablar de la cualificación necesaria para saber controlar dicho robot.

Ante esto, nace la pregunta: ¿Cómo podemos mantener los beneficios de la CR, pero manteniendo todos los beneficios de los robots sin perder prestaciones? La respuesta es la utilización del algoritmo TERRA.

2.2. Análisis de diferentes algoritmos de planificación de rutas

En esta sección se van a estudiar diferentes algoritmos de planificación de rutas viendo su forma de actuar y sus pros y contras respecto a TERRA y por qué la elección de nuestro algoritmo.

2.2.1. Asignación de tareas de búsqueda y rescate de helicópteros y UAVs

La asignación de tareas de UAVs es importante en la búsqueda y rescate para garantizar una actividad de búsqueda y rescate efectiva y ordenada. Sin embargo, pocos estudios han investigado las influencias del entorno operativo y el rendimiento del UAV en la asignación de tareas de búsqueda y rescate. Teniendo en cuenta las influencias del viento y el terreno a baja altitud en el consumo de energía y el rendimiento de los UAVs, en este estudio se proponen modelos de asignación de tareas y selección de la posición de lanzamiento del UAV (desde el helicóptero) [11]. Dado que la zona de detección de los UAVs se ve afectada por factores topográficos, se utilizó el método de análisis de componentes principales para determinar el nivel de búsqueda y rescate en cada punto y la resistencia de vuelo estacionario del UAV, que se determinó mediante análisis de conglomerados. Teniendo en cuenta los factores de influencia del consumo de energía de la batería del UAV, como el rendimiento del UAV y el viento a baja altitud, se construyó y resolvió un modelo de selección de la posición de lanzamiento del UAV mediante el algoritmo de murciélago binario mejorado que exhibió una precisión de cálculo mejorada en comparación con otros tres algoritmos. De acuerdo con el resultado de la planificación de la posición de lanzamiento, se estableció y resolvió un modelo de optimización multiobjetivo (minimización del costo total de búsqueda y rescate, número de UAVs y equilibrio de tareas de múltiples UAV)

utilizando el algoritmo genético de clasificación no dominado-II [12]. Finalmente, los resultados experimentales indicaron que la solución a través de objetivos múltiples no tuvieron una ventaja de coste en comparación con las de un solo objetivo, pero exhibió una ventaja evidente en el tiempo de finalización de la tarea a través del análisis de sensibilidad del problema de asignación de tareas de los UAVs. Un ejemplo visual de los dispositivos que se crearon se puede ver en la Figura 2.1.

Figura 2.1: Vehículo aéreo no tripulado.

2.2.2. Planificación de la ruta de entrega de un sistema de robot heterogéneo bajo las restricciones de la red de carreteras

Chen et al. [13] se centran en el problema de planificación de rutas de un sistema de robots heterogéneo aplicado a la entrega de paquetes en entornos urbanos. El sistema consta de un vehículo terrestre no tripulado (UGV) y un vehículo aéreo no tripulado (UAV). El UGV está restringido a conducir en la red de carreteras. El UAV descarga paquetes del UGV y los entrega a los clientes. Este estudio considera los esquemas de ruta de UGV y UAV juntos sobre la base de una cooperación aire-tierra. Se propone una estrategia de dos pasos que combina la optimización de colonias de hormigas (ACO) y el algoritmo genético (GA) para desacoplar sus rutas. En el primer paso, el ACO se utiliza para buscar las rutas del UGV. En el segundo paso, el GA se emplea para resolver las rutas del UAV una vez que las rutas del UGV están predeterminadas en el primer paso. Los resultados de la simulación muestran que el método puede resolver eficazmente el problema de la entrega heterogénea y obtener una ruta óptima para el UAV.

Es una herramienta que realiza al igual que TERRA un cálculo matemático (en nuestro caso mediante la función 'Optimización Gravitacional' que será explicado más adelante) para determinar la mejor ruta o la que cree óptima. Pero por otro lado no se asemeja lo suficiente a nuestro modelo debido a que es otro problema el que intenta resolver respecto a TERRA ya que en ningún momento este modelo se plantea utilizar UAVs donde el gran problema del tráfico terrestre se eliminaría, así como una reducción en el tiempo al ser más rápidas las rutas aéreas que ir por un camino físico.

2.2.3. Aplicación de algoritmos de planificación de rutas para UAVs en agricultura de precisión

Los UAVs de múltiples rotores [14], aunque originalmente se diseñaron y desarrollaron para fines militares y de defensa, en los últimos diez años han ganado impulso, especialmente para aplicaciones civiles, como búsqueda y rescate, topografía y mapeo, cultivos agrícolas y monitoreo. Gracias a sus capacidades de vuelo estacionario y de despegue y aterrizaje vertical (VTOL) y la capacidad de llevar a cabo tareas con total autonomía, ahora son una plataforma estándar tanto para investigación como para usos industriales. Sin embargo, si bien la arquitectura de control de vuelo está bien establecida en la literatura, todavía existen muchos desafíos en el diseño de sistemas autónomos de guía y navegación para que el UAV pueda trabajar en entornos restringidos y desordenados o también en interiores. Existe una completa y exhaustiva literatura sobre los numerosos métodos y enfoques para abordar los problemas de planificación de rutas para UAV de múltiples rotores. En particular, la inclusión de una revisión de la investigación relacionada en el contexto de la Agricultura de Precisión (AP) como se muestra en la siguiente Figura 2.2 proporciona una presentación unificada y accesible para los investigadores que están iniciando sus esfuerzos en este tema.

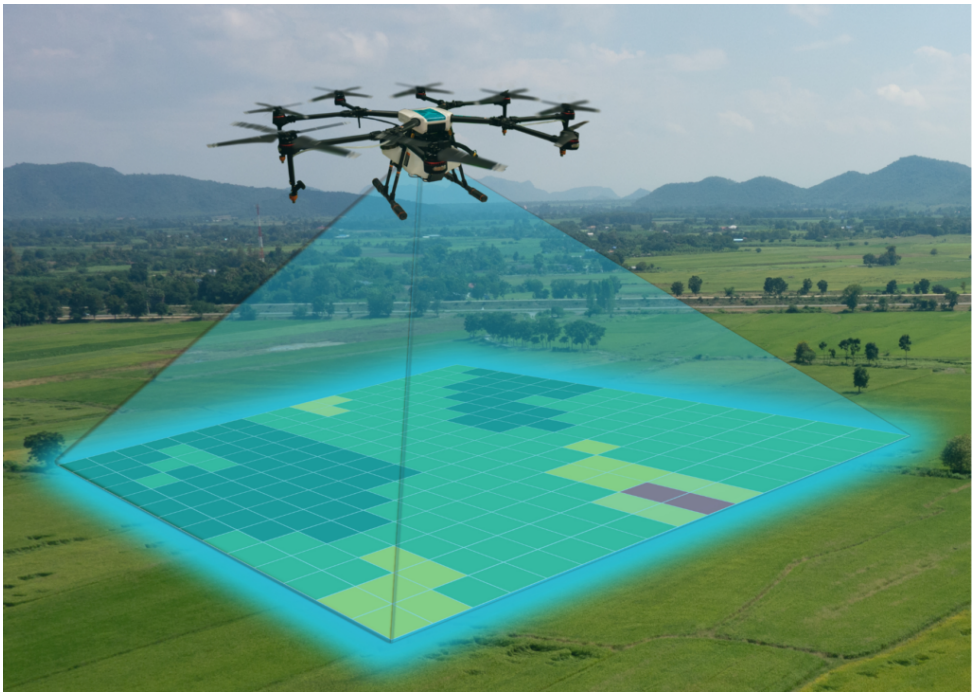


Figura 2.2: Dron en agricultura de precisión.

2.3. TERRA

Los sistemas UGV-UAV simples y heterogéneos representan un tema de investigación vibrante para la exploración cooperativa. Por lo tanto, animados por el futuro de las aplicaciones cooperativas UGV-UAV, como como la Misión Marte 2021 fue creado el algoritmo de planificación de rutas cooperativas UGV-UAV [1], que explota un paradigma de cooperación UGV-UAV para abordar un problema de exploración concreto. La literatura existente muestra una amplia diversidad de sistemas de exploración UGV-UAV cooperativos, pero ninguno de ellos está concebido para exploraciones autónomas de alto nivel, (donde tanto el UGV como el UAV son sistemas totalmente autónomos) ni están concebidos para problemas de exploración concretos.

En la siguiente sección presentamos un nuevo problema de exploración que puede ser la base de problemas de exploración complejos. A continuación, describimos el algoritmo de planificación de trayectorias cooperativas para resolver el problema de exploración. Este algoritmo implementa un paradigma de cooperación UGV-UAV ejecutando un conjunto ordenado de etapas computacionales. A continuación, presentamos la extensión a los problemas de exploración como la búsqueda de vecinos más cercanos, el problema del conjunto de cobertura, el problema del viajante y la restricción de energía UAV y de la distancia. El problema de exploración que proponemos se denomina formalmente como el Problema de Enrutamiento de UAVs y Estaciones de Carga UGVs con Restricciones Energéticas (ECU-CSURP).

2.3.1. Problemas que surgen en TERRA

Las siguientes subsecciones describen los problemas particulares y los algoritmos que tienen estado en estudio en el algoritmo TERRA.

2.3.1.1. Búsqueda de vecinos más cercanos

La búsqueda de vecino más cercano (NNS) es una forma de búsqueda de proximidad y consiste en encontrar el punto de un conjunto dado más cercano a un Punto dado.

La naturaleza intrínseca de este problema lo clasifica como un problema de proximidad. Esto lo hizo Preparata y Shamos [15] para abordar el problema desde una perspectiva geométrica perspectiva. En su trabajo, Preparata y Shamos [15] formularon la loci del problema de proximidad para resolver el NNS, que establece lo siguiente: Dado un conjunto 'S' de 'N' puntos en el plano, para cada punto 'pi' en 'S', ¿cuál es el lugar geométrico de los puntos (x, y) en el plano que están más cerca de pi que a cualquiera otro punto de 'S'? Como la solución de este problema es una partición del avión en regiones, notaron que, solo buscando en este espacio de regiones, podrían resolver el NNS. Así, Preparata y Shamos [15] fueron los primeros en proponer los diagramas de Voronoi [16] para resolver el Problema NNS.

2.3.1.2. Problema del conjunto de cobertura

El problema del *Set Covering* [17], también conocido por sus siglas SCP es un problema clásico en combinatoria, ciencias de la computación y teoría de la complejidad computacional. Es un problema que ha llevado al desarrollo de técnicas fundamentales para el campo de los algoritmos de aproximación. También es uno de los problemas de la Lista de 21 problemas NP-completos de Karp cuya NP-completitud fue demostrada en 1972.

Dado un conjunto de elementos (1,2,.., m) (llamado universo) y 'n' conjuntos cuya unión comprende el universo, el problema del conjunto de cobertura consiste en identificar el menor número de conjuntos cuya unión aun contiene todos los elementos del universo. Por ejemplo, sea U = (1,2,3,4,5) y los conjuntos S = ((1,2,3),(2,4),(3,4),(4,5)). Claramente, la unión de todos los conjuntos de 'S' contiene todos los elementos de 'U'. Sin embargo, podemos cubrir todos los elementos con el siguiente conjunto de elementos, con menor número de elementos: ((1,2,3),(4,5)).

2.3.1.3. Problema del viajante

El problema del vendedor viajero [9] (problema del vendedor ambulante, problema del agente viajero o problema del viajante, TSP) por sus siglas en inglés (*Travelling Salesman Problem*) responde a la siguiente pregunta: dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen? Este es un problema NP-Hard dentro en la optimización combinatoria, muy importante en investigación operativa y en ciencias de la computación.

El problema fue formulado por primera vez en 1930 y es uno de los problemas de optimización más estudiados. Es usado como prueba para muchos métodos de optimización. Aunque el problema es computacionalmente complejo, se conoce gran cantidad de heurísticas y métodos exactos, así que es posible resolver planteamientos concretos del problema desde cien hasta miles de ciudades.

2.3.1.4. Restricción de energía UAV y de la distancia

El problema de exploración que proponemos se denomina formalmente como el enrutamiento de UGV de la estación de carga y el UAV con restricciones de energía Problema (ECU-CSURP).

El objetivo de la ECU-CSURP es encontrar un enrutamiento cooperativo para el sistema simple UGV-UAV para permitir que el UAV visite cada punto objetivo mientras intenta minimizar la distancia total de viaje. Para ello tendremos que Modelar el espacio euclidiano R2 como un área donde la distancia recorrida por ambos sistemas robóticos es directamente proporcional al tiempo empleado y la energía consumida en un viaje.

Por lo tanto, cuanto menor sea la distancia recorrida, menor será el tiempo empleado y menor la energía consumida, y viceversa. Además, la restricción de energía del

UAV se modela como la distancia máxima que el UAV puede viajar con una batería completamente cargada.

2.3.2. Explicación y funcionamiento de las diferentes fases de TERRA

El algoritmo TERRA consiste en la utilización y colaboración directamente de UAVs y UGVs sin la necesidad estricta de tener una supervisión humana donde no se prive de ningún beneficio del uso de dicha técnica.

El funcionamiento está estructurado en 5 subfases que se explicarán de una forma más detallada en el Capítulo 3:

- **Fase I: Diagramas de Voronoi:** Los diagramas de Voronoi son un método de interpolación basado en la interpolación euclidiana que se crear al unir los puntos entre sí, trazando las mediatrices de los segmentos de unión. Las intersecciones de estas mediatrices determinan una serie de polígonos en un espacio bidimensional alrededor de un conjunto de puntos de control, de manera que el perímetro de los polígonos generados sea equidistante a los puntos vecinos y designan su área de influencia. De este modo podremos hallar la mejor posición para nuestros vehículos autónomos terrestres (UGVs) para que pueda desplegar los (UAVs) para inspeccionar las diferentes zonas.

  A continuación, se muestra una figura 2.3 para que se vea de una forma más visual.

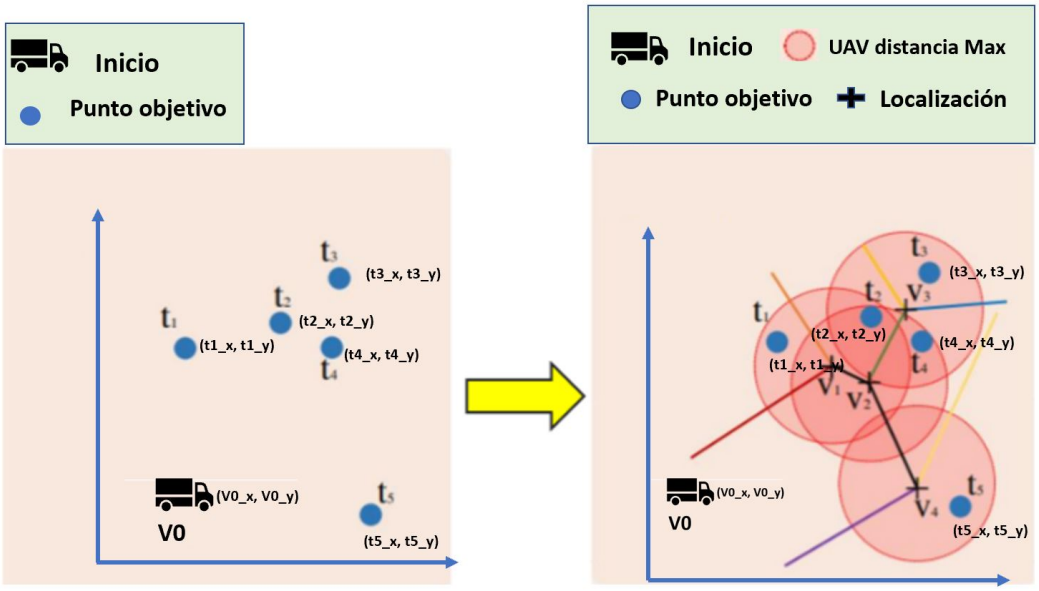


Figura 2.3: Diagrama Voronoi Fase I TERRA.

Por último, también se muestra en la Figura 2.4 un ejemplo real en Python del cálculo del diagrama de Voronoi para 4 puntos.

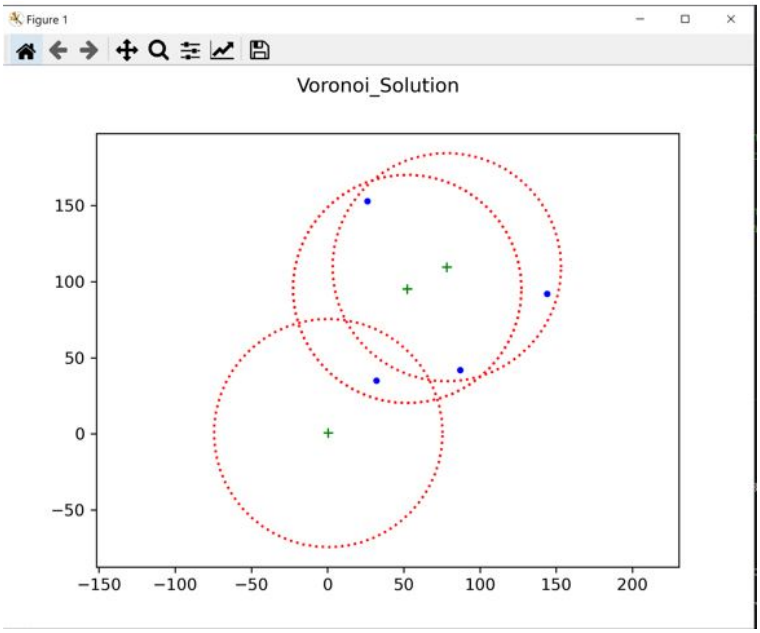


Figura 2.4: Solución para 4 puntos de Voronoi.

- **Fase II: Optimización Combinatoria:** Encontrar un subconjunto de ubicaciones que cubran todos los puntos destinos.

  A continuación, se muestra una figura 2.5.

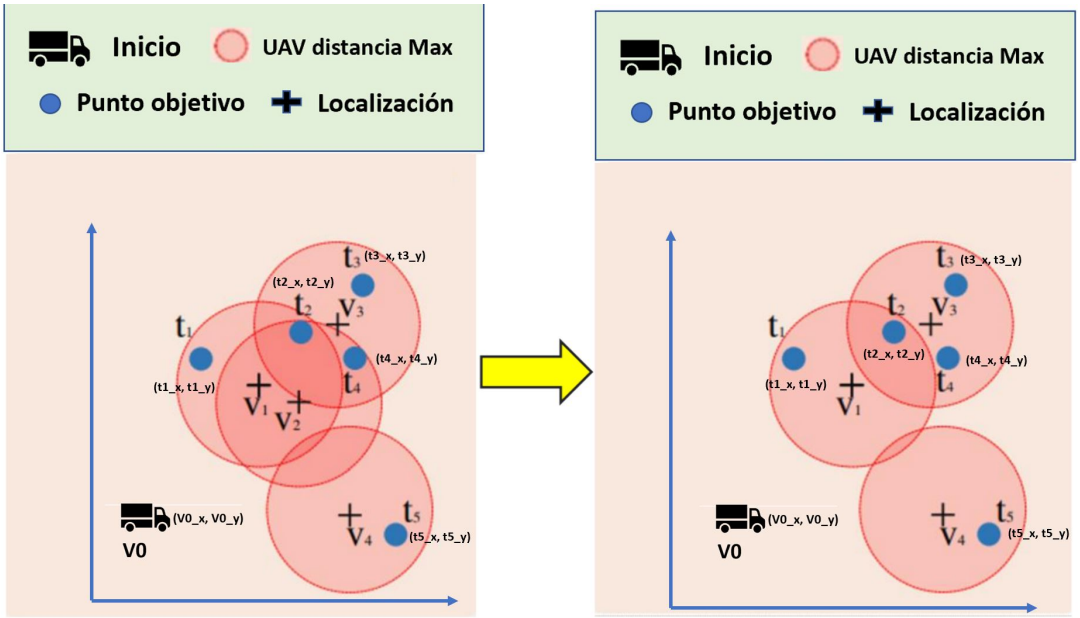


Figura 2.5: Optimización Combinatoria Fase II TERRA.

Por último, al igual que en la Fase I se muestra en la Figura 2.6 la salida del ejemplo real en Python para 4 puntos tras haber aplicado la optimización combinatoria reduciendo el número de subconjuntos que cubren todos los puntos destinos pasando de 3 subconjuntos a tan solo 2.

- **Fase III: Optimización Gravitacional:** el objetivo de esta fase es reducir la

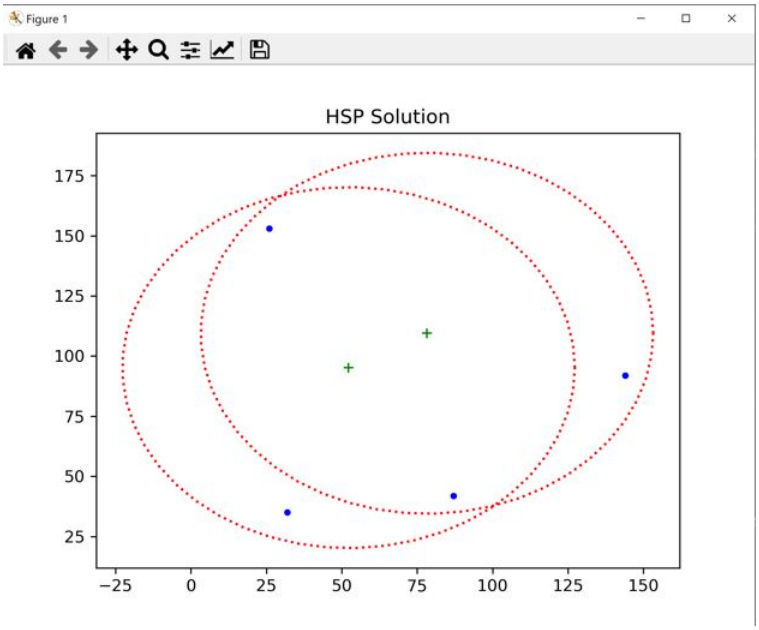


Figura 2.6: Solución para 4 puntos tras aplicar la optimización a los subconjuntos.

distancia recorrida del UGV con el fin de optimizar los recursos, en especial el tiempo y la energía. En este método se calculará un punto "p" al borde de las zonas V1, V2, etc, para reducir de ese modo la distancia.
A continuación, se muestra una figura 2.7 para que se vea de una forma más visual.

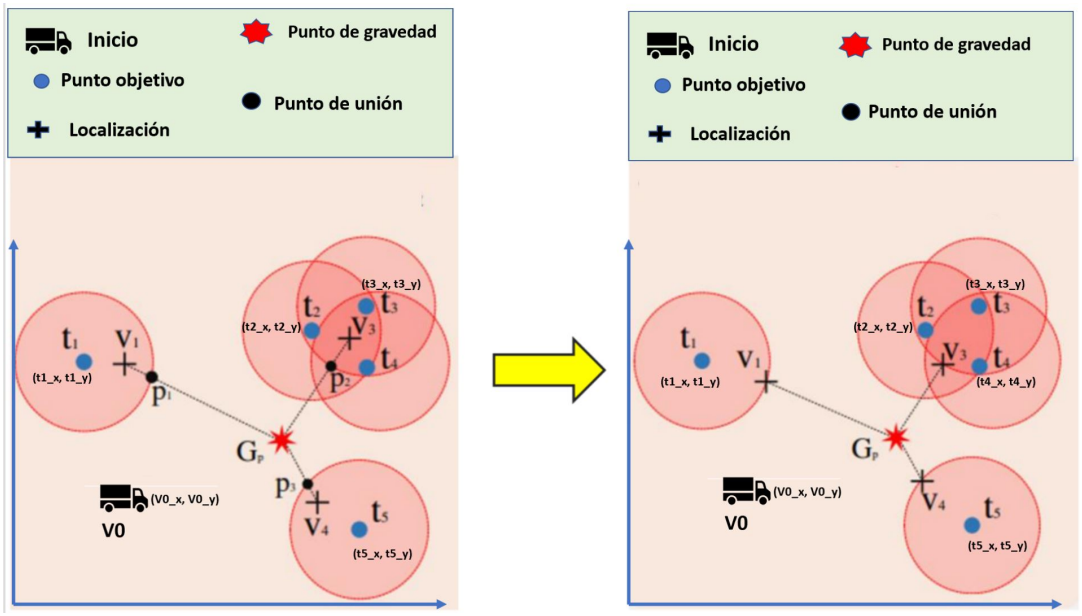


Figura 2.7: Optimización Gravitacional Fase III TERRA.

- **Fase IV: Algoritmo Genético:** en esta fase se obtendrá la ruta más corta del UGV mediante Algoritmos Genéticos donde para ello debemos convertir nuestras soluciones a nuestro problema en vectores matemáticos a través de un mecanismo de Inteligencia Artificial.
  A continuación, se muestra en la figura 2.8 para que se vea de una forma más visual.

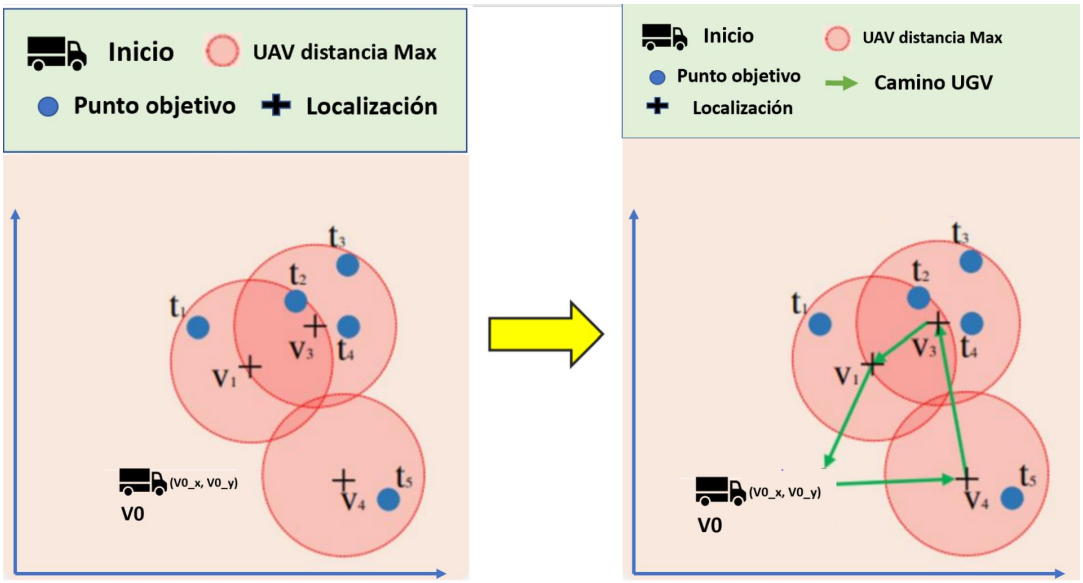


Figura 2.8: Algoritmo Genético Fase IV TERRA.

- **Fase V: Algoritmo de Búsqueda:** Este algoritmo de búsqueda calcula la ruta dirigida más corta del UAV para cada sub-recorrido.
  A continuación, se muestra en la figura 2.9 como desde cada vértice se calcula el camino de cada UAV.

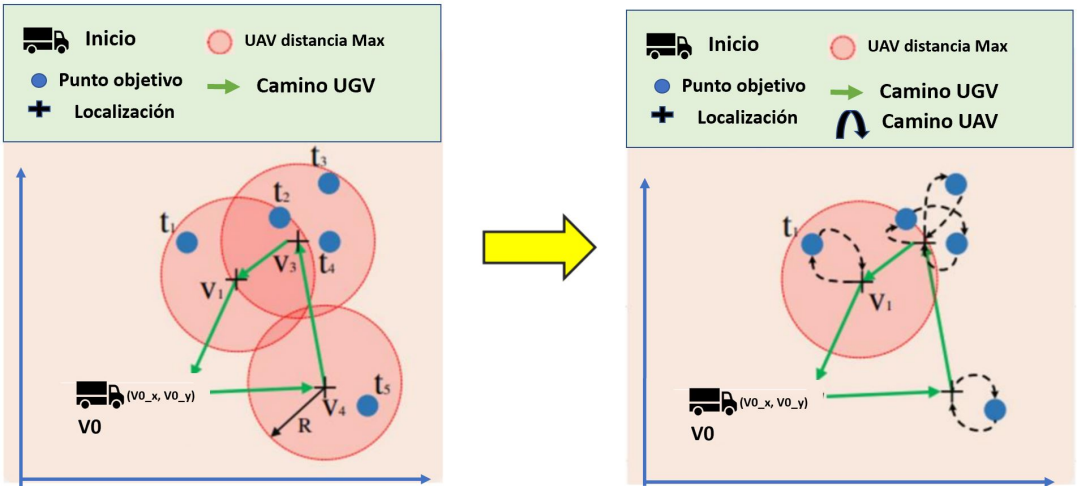


Figura 2.9: Algoritmo de Búsqueda Fase V TERRA.

Finalmente, al igual que en las dos primeras Fases, en la Figura 2.10 se muestra la solución final en Python aplicada a 4 puntos donde se muestra en el mismo color verde la ruta que seguirá el UGV.

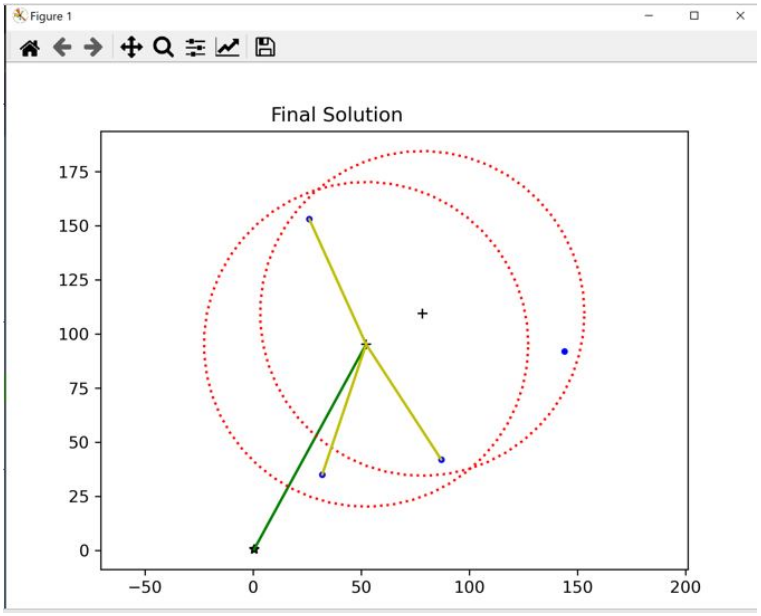


Figura 2.10: Solución final del algoritmo para 4 puntos.

2.4. Análisis de herramientas

En este sección se van a examinar las principales herramientas de traducción de codigo Matlab a Python que existen, donde primero se analizará individualmente cada una de ellas y finalmente una comparación de todas ellas descomponiendo los aspectos más relevantes.

2.4.1. SMOP (Small Matlab and Octave to Python compiler)

SMOP [18] es una API que se puede descargar utilizando GitHub en la dirección: 'https://github.com/victorlei/smop' donde tras clonar el repositorio se podrá hacer uso de dicha aplicación.

Esta app fue descartada ya que solo sirve para códigos muy simples y en este caso **no funcionó para traducir ni el script más sencillo que disponía TERRA**. Puede ser útil para traducir unas pocas líneas de código, pero no para proyectos grandes.

Un uso de esta herramienta sería la traducción de este fragmento que se muestra en la figura 2.11 donde la función *size* de Matlab (código de la izquierda) devuelve un número variable de valores de retorno, lo que corresponde a devolver una tupla en Python. Dado que las funciones de Python desconocen el número esperado de valores de retorno, su número debe pasarse explícitamente en *nargout* (código de la derecha).

```
[m,n] = size(ai);              03     m,n=size_(ai,nargout=2)
```

Figura 2.11: Traducción de Matlab to Python.

2.4.2. Matlab2python

Matlab2python [19] es un proyecto de código abierto que comenzó basándose en el anteriormente mencionado SMOP pero tiene capacidades de uso más simples que este. En SMOP, la traducción depende de 'libsmop' pero matlab2python usa numpy para traducir declaraciones relacionadas con arreglos. La instalación de forma análoga clonando el repositorio de GitHub con la siguiente dirección: 'https://github.com/ebranlard/matlab2python '.

Respecto a su uso, también fue descartado por solo servir para la traducción de códigos muy simples y dando errores y muy mala traducción con códigos complejos.

A continuación, se van a mostrar dos pruebas que se realizaron donde se intentó traducir primero el *Test Launcher* que es uno de los scripts principales del algoritmo TERRA como se observa en la Figura 2.12 el cual dio error y directamente al igual que con otros scripts que se probaron no se pudo traducir nada. El formato de uso es el que se muestra en la figura donde primero se introduce el nombre del archivo original en Matlab en este caso: *Test Launcher.m* luego un '-o' y el nombre que se le quiera dar en Python.

```
C:\Users\mario\pruebaJunio\matlab2python>python matlab2python.py Test_Launcher.m -o Test_Launcher.py
Traceback (most recent call last):
  File "C:\Users\mario\pruebaJunio\matlab2python\matlab2python.py", line 49, in <module>
    main(sys.argv[1:])
  File "C:\Users\mario\pruebaJunio\matlab2python\matlab2python.py", line 44, in main
    mparser.matlab2python(opts.filelist,opts)
  File "C:\Users\mario\pruebaJunio\matlab2python\matlabparser\parser.py", line 433, in matlab2python
    matlab2python(f,opts)
  File "C:\Users\mario\pruebaJunio\matlab2python\matlabparser\parser.py", line 442, in matlab2python
    PY=MF.toPython(backend='m2py')
  File "C:\Users\mario\pruebaJunio\matlab2python\matlabparser\parser.py", line 339, in toPython
    stmp=parse_matlab_lines(stmp,backend)
  File "C:\Users\mario\pruebaJunio\matlab2python\smop\main.py", line 22, in parse_matlab_lines
    stmt_list = parse.parse(buf if buf[-1] == '\n' else buf + '\n')
  File "C:\Users\mario\pruebaJunio\matlab2python\smop\parse.py", line 844, in parse
    p = parser.parse(
  File "D:\TFG\anaconda\lib\site-packages\ply\yacc.py", line 331, in parse
    return self.parseopt(input, lexer, debug, tracking, tokenfunc)
  File "D:\TFG\anaconda\lib\site-packages\ply\yacc.py", line 823, in parseopt
    p.callable(pslice)
  File "C:\Users\mario\pruebaJunio\matlab2python\smop\parse.py", line 694, in p_return_stmt
    p[0] = node.return_stmt(ret=ret_expr)
NameError: name 'ret_expr' is not defined

C:\Users\mario\pruebaJunio\matlab2python>
```

Figura 2.12: Traducción de Matlab a Python del script Test Launcher.

2.4.3. OMPC (Open-Source Matlab-To-Python Compiler)

OMPC [20] es un proyecto de código abierto de nivel medio donde ciertas operaciones el programa directamente no es capaz de comprender ni traducir, este algoritmo se descartó desde un primer momento al ser mucho más simple y poco útil si se quería introducir algo de abstracción matemática, algo que también ocurría con los dos anteriormente comentados.

2.4.4. Comparación de herramientas

Tras analizar las principales herramientas como se puede ver en la Figura 2.13 se descartaron todas por lo que se utilizó directamente un entorno visual *Visual Studio* en el cual se fue traduciendo primero toda la estructura de los scripts donde se tuvo que ir siguiendo una metodología que más adelante se indagará más en ella.

Los aspectos que se tuvieron en cuenta para la realización de dicha comparación de herramientas fueron por un lado la 'Traducción de código' que se divide en diferentes sub-apartados donde se fue probando la traducción de códigos muy simples hasta códigos muy complejos. Otra característica que se analizó fue el 'Contenido didáctico', es decir, la documentación disponible y el soporte de cada herramienta. Por último, se compararon las diferencias en las 'Licencias de uso' dando prioridad a que fueran gratuitas (donde en este caso todas cumplían esta característica).

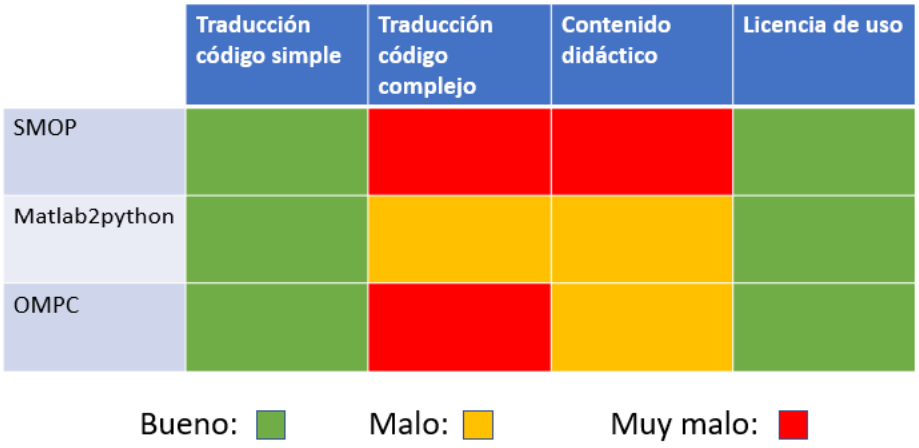


Figura 2.13: Principales herramientas de conversión.

Aquí se muestra un ejemplo para clarificar todo lo anteriormente comentado y el motivo con un ejemplo práctico y real donde en la primera figura 1 se puede ver una comparación primero con el código en Matlab original utilizando un script de TERRA, luego el que resulta al traducirlo y el resultado final que se realizó a mano sin utilizar ninguna herramienta.

En la línea 1, se muestra la función 'isempty' que se encarga de verificar si un vector está vacío. En la línea 2 se encuentra la función 'size' la cual nos devuelve el tamaño de un arreglo. En la línea 3 se encuentra la función 'meshgrid' que función es transformar el dominio especificado por un vector único o dos vectores 'x' e 'y' en matrices 'X' e 'Y' con el objeto de usarlas en la evaluación de funciones de dos variables. Las filas de 'X' son copias del vector 'x' y las columnas de 'Y' son copias del vector 'y'. Por último, la línea 4 contiene la función 'reshape' cuyo objetivo es cambiar el tamaño y la forma de un arreglo. Por ejemplo, remodele una matriz de 3 por 4 para convertirla en una matriz de 2 por 6.

```
1 if isempty(dmat)
2     nPoints = size(xy,1);
3     a = meshgrid(1:nPoints);
4     dmat = reshape(sqrt(sum((xy(a,:)-xy(a',:)).^2,2)),nPoints,nPoints);
5 end
```

Listing 1: isempty() en Matlab [21].

A continuación se muestra como quedó el código *isempty()* en la Figura 2 al traducirlo donde realiza algún cambio pero no tiene, ni sentido, ni utilidad ya que solo valen estos conversores (y eso que se está utilizando Matlab2Python que es el más valorado) para códigos simples y sin dependencias de otros scripts.

```
1 if len(dmat)==0:
2       nPoints = xy.shape[1-1]
3       a = np.meshgrid(np.arange(1,nPoints+1))
4       dmat = reshape(np.sqrt(np.sum((xy(a,:)-xy(np.transpose(a),:))**2,2-1))
5     ,nPoints,nPoints)
6
```

Listing 2: Parte del código isempty() utilizando la herramienta Matlab2Python [21].

Por último se muestra la versión definitiva que se ha utilizado y creado del *isempty()* en la Figura 3 de forma manual (como todo el código) al descartar el uso de cualquiera de las herramientas de traducción anteriormente comentadas. Esta versión definitiva en comparación con la versión de Matlab es mucho más compleja y pasa de tan solo ocupar 3 líneas a las que se muestran a continuación, siendo un total de 35 líneas.

```
if len(dmat)==0:
        #nPoints = xy.shape[1-1]
        nPoints = len(xy)
        #a = np.meshgrid(np.arange(1,nPoints+1))
        a = []
        #meshgrid
        lista_aux = list()
        for i in range(1, nPoints+1):
            lista_aux.append(i)
        for i in range(0, nPoints):
            a.append(lista_aux)
        a = np.array(a)
        indice = 0
        lista_xy_a = list() #es lo mismo que xy[a,:] en Matlab
        for i in a:
            for i2 in a:
                lista_xy_a.append(xy[i2[indice] -1])
            indice = indice +1
        lista_xy_a = np.array(lista_xy_a)
        a_transpose = np.transpose(a)
        indice = 0
        lista_xy_a_transpose = list() #es lo mismo que xy[a_transpose,:] en Matlab
        for i in a_transpose:
            for i2 in a_transpose:
                lista_xy_a_transpose.append(xy[i2[indice] -1])
            indice = indice +1
        lista_xy_a_transpose = np.array(lista_xy_a_transpose)
        aux_dmat = np.sum((lista_xy_a - lista_xy_a_transpose) ** 2, 1)
        aux_dmat_np = []
        for i in aux_dmat:
            aux_dmat_np.append([i])
        aux_dmat_np = np.array(aux_dmat_np)
        dmat_sqrt =   np.sqrt( aux_dmat_np)
        dmat_aux = dmat_sqrt.reshape(nPoints,nPoints)
        dmat = dmat_aux
    N,__ = xy.shape
    nr,nc = dmat.shape
```

Listing 3: Traducción de la Figura 2.12 sin utilización de herramienta [21].

A continuación, se va a comentar las principales diferencias que hay respecto al código que he tenido que crear en Python respecto al Matlab en este fragmento:

Lo primero que se puede apreciar es que *isEmpty()* no existe como función en Python por lo que para sacar la longitud hay que usar *len()*. Por otro lado, la variable *a* de Matlab como se puede observar en el código de Python de la línea 9 donde se crea hasta la 16 se debe calcular completamente desde cero.

Por otro lado como se comenta en la línea 5 la realización de la operación *meshgrid* que en Matlab si existe en Python hay que realizar su funcionamiento de forma manual ya que en *np.mesgrid* que crear *matlab2python* no es el funcionamiento que se desea, todo ello se desarrolla de la línea 10 hasta la 16.

A continuación, en Matlab se hace uso de la función reshape() que otra vez el conversor *Matlab2Python* pone esa función la cual, no existe en Python o no funciona de la forma correcta por lo que se ha tenido que crear *dnmat()* a mano desde la línea 22 hasta la línea 30 y de la línea 38 hasta la 48.

Justo en medio nos encontramos otro inconveniente y es que en Matlab lo que simplemente es *xy transpose*, en el código definitivo es lo siguiente que se muestra de la línea 32 a la línea 37 en la Figura de arriba.

Por todo lo anterior comentado, el uso de herramientas para traducir código de Matlab a Python puede tener cabida para alguien que quiera comprobar o verificar una línea simple en un proyecto de universidad o en su día a día, **pero nunca y en ningún caso para ningún script serio o meramente complejo** como se puede ver en el ejemplo anterior (no siendo este ejemplo de las partes del código más complejas a la hora de traducir).

Además, hay que tener en cuenta que al haber muchas dependencias como se mostrará en el próximo capítulo 3 no tendrá sentido ya que la herramienta no tiene conocimiento de ello ni aun que lo tuviese podría modificarlo.

Finalizando quiero dejar claro que traducir código del lenguaje Matlab a Python no tiene apenas similitudes. Para realizar la traducción recomendaría la utilización de compiladores donde pueda ir viendo o imprimiendo las variables y los distintos valores para entender bien lo que está sucediendo y de ese modo ir poco a poco modificando el código de una forma lógica y estructurada.

Como ninguna herramienta satisfacía las necesidades, **se tuvo que traducir de forma manual todo el código de Matlab a Python** con el consecuente gasto de tiempo y esfuerzo que eso conlleva como se puede ver con la función que se acaba de explicar isEmpty() **pasando de ocupar 3 líneas en Matlab a 47 en Python**, o por ejemplo el script 'tspGaUgv' el cuál en Matlab ocupa 361 líneas de código frente a las 532 en Python donde en todos los scripts se tuvo que hacer de forma manual ocupando un total de 3242 líneas de código y con cambios grandes y visibles como luego se podrán ver en los siguientes apartados donde se detallarán en especial algunos de los cambios más significativos que se tuvo que hacer del código de Matlab a Python.

Capítulo 3.

Análisis y desarrollo

En este capítulo se va a explicar el objetivo de la aplicación a desarrollar junto a una explicación de cada una de sus partes y los distintos elementos que se han creado para conseguir el funcionamiento final.

3.1. Objetivo

En esta sección se va a explicar cuál es el objetivo del algoritmo que se va a crear y qué problemas intenta solucionar.

Nos encontramos en un momento donde cada vez hay más dispositivos inteligentes [22], mayor número de algoritmos que desean una cohesión con el uso de robots. De forma paralela estamos en un momento donde hay un gran desconocimiento por parte de los usuarios respecto a cómo poder cooperar o diseñar modelos para poder trabajar en conjunto o delegar en dichas máquinas para trabajos que para las personas sería mucho más costoso implementar.

Para intentar solventar este problema, este trabajo migrará y mejorará la portabilidad del algoritmo TERRA escrito en Matlab a Python para ayudar a otros usuarios y empresas que quieran seguir mejorando el código al ser mucho más legible y portable en Python.

3.2. Elección de Python frente a Matlab

En esta sección se va a explicar las principales diferencias entre Matlab y Python, así como las principales ventajas e inconvenientes de cada uno y el motivo por el qué se eligió Python por encima de otros lenguajes de programación.

Una de las primeras preguntas que nos surgen cuando empezamos en el mundo de la programación es: *¿De la cantidad de lenguajes de programación que existen hoy dia, cual debo elegir?* y para ello la respuesta no es la que mucha gente puede pensar ya que es: *Depende*, de para que quieras utilizar dicho lenguaje o que herramienta estés pensando en desarrollar.

En el caso de que el uso que se le quiera dar sea meramente de recopilación de datos científicos o matemáticos puede ser interesante utilizar Matlab o por lo menos antiguamente (más adelante ahondaré en este tema).

Si de lo contario queremos desarrollar un código/modelo/aplicación que sea altamente portable, actualizable y mejorable con el paso de los años lo mejor es el uso de lenguajes muy utilizados dentro del mundo de la ingeniería y los cuales no sean muy complicados no tanto de programar en dicho lenguaje, pero sí que sea fácil su compresión escrita al leer cualquier script.

Como se muestra en la Figura 3.1 Python es el lenguaje más utilizado en el mundo con más de un 25.90 % de uso frente al décimo puesto que ocupa Matlab con tan solo un 2.0 % de uso.

Figura 3.1: Lenguajes de Programación más utilizados.

Por dicho motivo se eligió Python para el desarrollo de este algoritmo para poder hacerle más versátil que le permita plantearse en un futuro integrarlo en aplicaciones reales. Las principales ventajas que nos ofrece *Python* [23] son muy diversas pero si nos tuviésemos que quedar con algunas serían las siguientes:

- Lenguaje de alto nivel.
- Polivalente y de paradigmas.
- Gran variedad de Bibliotecas y frameworks.
- Portabilidad.
- Gratis y de código abierto.
- Baja curva de aprendizaje.
- Una comunidad muy amplia.

Por último, por ahondar un poco más respecto a *"Matlab tenía sentido anteriormente para la utilización de datos matemáticos"* es porque hoy en día existen diversos frameworks y bibliotecas como *numpy* o *panda* altamente conocidas en el sector que solventan todos estos problemas.

3.3. Problemas que surgieron para coexistir todas las versiones y herramientas

Uno de los principales problemas es que *Visual Studio*, (que es un editor de código fuente desarrollado por Microsoft, el cual incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código), no identifica manualmente la versión necesaria para cada import o biblioteca de Python que se utiliza, por lo que, puede que una herramienta solo funcione en Python 3.0 y otra en 1.2 donde está deprecada en la actualidad y no podrá ser usada.

Para solventar este problema se llegó a la solución de utilizar *AnaConda* [24] que es una distribución libre y abierta de diferentes lenguajes donde en este caso nos interesa Python que se utiliza en Aprendizaje Automático y que permite unificar diferentes versiones de Python creando un entorno donde todas ellas puedan coexistir y de este modo solventaremos el problema anterior.

Todo ello se instaló siguiendo la documentación oficial de Anaconda para Windows donde fue necesario crear un entorno visual y gráfico explícitamente para poder probar el algoritmo y los diferentes scripts.

En definitiva, se tuvo que usar Anaconda para converger todas las diferentes versiones en nuestro editor Visual Studio ya que utilizando sólo la última versión de Python hay muchas funciones usadas en varios scripts que no funcionaban o estaban en desuso (deprecated) a partir de cierta versión de Python.

Para ver de una forma más visual lo anteriormente comentado se muestra un primer diagrama en la Figura 3.2 donde al utilizar directamente funciones con diferentes versiones de Python el compilador daría error.

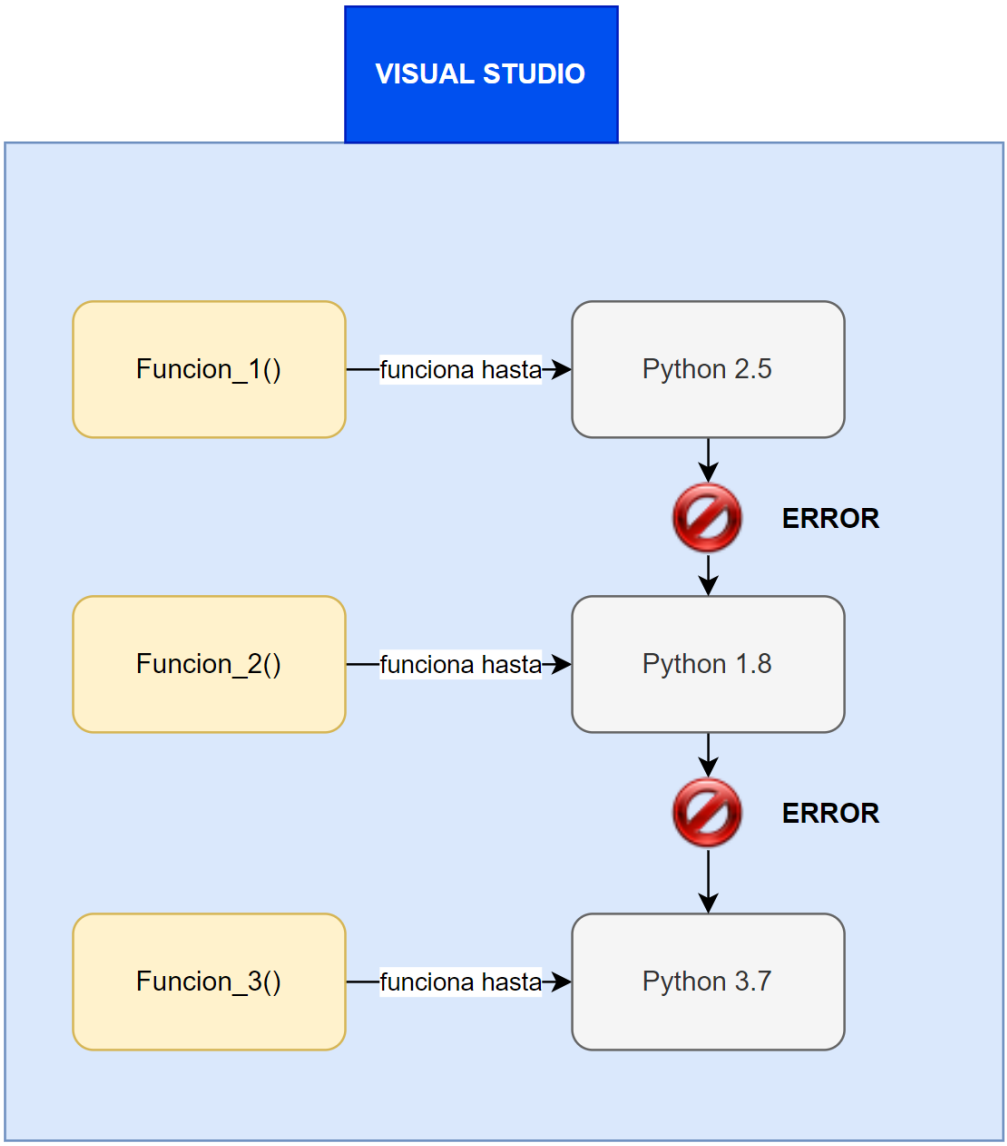


Figura 3.2: Utilización de funciones con diferentes versiones de Python.

En cambio, en la figura 3.3 se muestra cómo se cohesionan las diferentes versiones de las funciones y ya no daría error al utilizarlo en nuestro compilador Visual Studio.

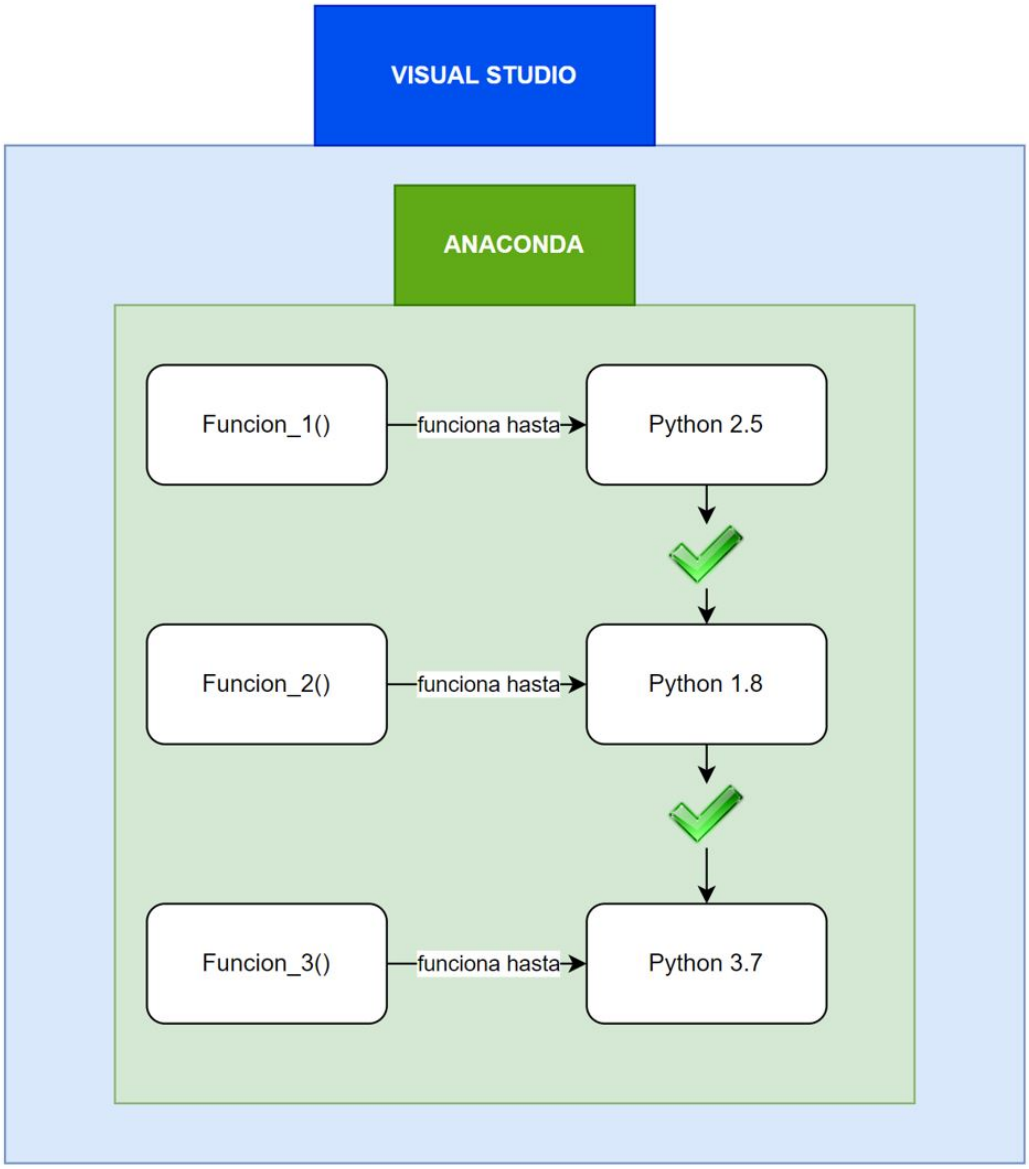


Figura 3.3: Utilización de funciones con diferentes versiones de Python con Anaconda.

3.4. Problemas en los scripts

Una vez vistos los problemas iniciales de versiones vamos a ir comentado los errores y dificultades de los scripts principales que orquestan el algoritmo así como fueron surgiendo para crear o adaptar muchas funciones o estructuras que existen en Matlab, pero en Python no tienen ninguna equivalencia.

3.5. Diferencias generales entre Matlab y Python

En esta sección se van a tratar las diferencias principales que surgieron al traducir de forma manual los scripts de Matlab a Python y como se fueron solventando los problemas que fueron apareciendo.

3.5.1. Script 'Búsqueda en amplitud y Build matrix solution'

El principal problema que tenía este script como en muchos más que vamos a ir viendo fue la función *struct2table* como se muestran en las siguientes dos figuras, la primera en Python (Figura 4) frente a la de Matlab (Figura 5) la cual era utilizada en muchas funciones y al no existir ninguna función que traduzca una estructura a una tabla se tuvo que optar por usar diccionarios y crear una función que hiciese eso manualmente.

```
1  def is visited(list = None,node = None):
2      nodeIdx = 0
3      if (not len(list)==0 ):
4          list = myStruct2table(list)
5          list = np.array([list[:,np.arange(2,4+1)],list[:,6]])
6          node = myStruct2table(node)
7          node = np.array([node[:,np.arange(2,4+1)],node[:,6]])
8          r,__ = list.shape
9          for i in np.arange(1,r+1).reshape(-1):
10             if (list[i,:] == node[1,:]):
11                 nodeIdx = i
12                 break
13
14     return nodeIdx
```

Listing 4: Parte del código is visited() Python.

```
function [nodeIdx] = is visited(list, node)
    nodeIdx = 0;
    if (~isempty(list))
        list = struct2table(list);
        list = [list(:,2:4) list(:,6)];
        node = struct2table(node);
        node = [node(:,2:4) node(:,6)];
        [r,~] = size(list);
        for i=1:r
            if (list{i,:}==node{1,:})
                nodeIdx = i;
                break;
            end
        end
    end
end
```

Listing 5: Función is visited() Matlab.

```
1  #función struct2table en Python: convierte una estructura a una lista
2  def struct2table(list):
3      #arr = np.array(estructura)
4      try:
5         arr = np.array(list)
6      except:
7         print('error en arr = np.array(estructura)')
8         arr = list
9      lst = []
10
11     for x in arr:
12         lst.append(x)
13     return lst
```

Listing 6: Función struct2table() Python.

La función *struct2table()* como se puede ver en la Figura 7 es una función que en Matlab convierte una estructura escalar en una tabla utilizando las opciones predeterminadas en este caso en un array. Esto en Python no existe en ninguna Librería (a diferencia de los arrays y métodos matemáticos que se llaman con la biblioteca *numpy* como se ve en todo el código).

Por tanto en el script 'bfs' se definió y creo una función propia *struct2table()* que convierte una estructura a una lista donde se utiliza además la función *append()* como se observa en la línea 12. Esta función se declara para poder utilizarla llamándola o importándola a otros scripts que tienen el mismo problema y así solventándolo de una forma más práctica.

También surgieron muchos problemas con la operación ":"de Matlab que recorre hasta el final de dicha fila o columna todos los elementos que queden en el array. Esta funcionalidad tampoco existe en Python y tuvo que ser modificado con el uso de bucles y otras funciones.

3.5.2. Foptimus.py

Este script utiliza muchas fórmulas matemáticas donde para ello Matlab sí que cobra mucho más sentido, pero muchas de ellas no tenían traducción directa en Python y se tuvieron que ir adaptando con librerías de diferentes versiones y modificaciones haciendo uso para que coexistan con Anaconda (anteriormente comentado).

Un ejemplo de ello fue la función *VPA* que permite a las variables trabajar con doble precisión en coma flotante, es decir, para medidas de alta precisión. Para ello se utilizó la librería *SymPy* que trabaja con este tipo de variables y funciones. A continuación se muestra un ejemplo de su uso:

```
syms x
p = sym(pi);
piVpa = vpa(p) %VPA por defecto calcula valores con 32 decimales.

%Resultado:
piVpa =
3.1415926535897932384626433832795
```

Listing 7: Ejemplo de uso de la función VPA para calcular valores matemáticos [21].

Por último, un error que había en todos los scripts al utilizar el uso de *":"* ya que es una operación propia y única de Matlab que no tiene conversión específica en Python. Tiene 3 usos los cuales son:

- Creación de vectores.
- Indexación.
- Iteración de bucles 'for'.

Un ejemplo práctico de su uso sería dada una *Matriz A* como en la Figura 3.4 :

$$A = \begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix}$$

Figura 3.4: Matriz A

La operación en Matlab A(:,1) obtiene los elementos de todas las filas de la columna *"1"*, es decir, [1, 3, 5]. Por tanto, una forma para solucionar el problema en Python es implementar un bucle *"for"* que recorra las filas y coja todos los elementos, imitando la operación *":"* de Matlab.

3.5.3. Voronoi

El diagrama de Voronoi de un conjunto de puntos en el plano es la división de dicho plano en regiones, de tal forma, que a cada punto le asigna una región del plano formada por los puntos que son más cercanos a él que a ninguno de los otros objetos. De este modo podremos hallar la mejor posición para que cada UGV pueda desplegar los UAVs que inspeccionen las diferentes zonas. Esto se puede ver en la Figura 3.5

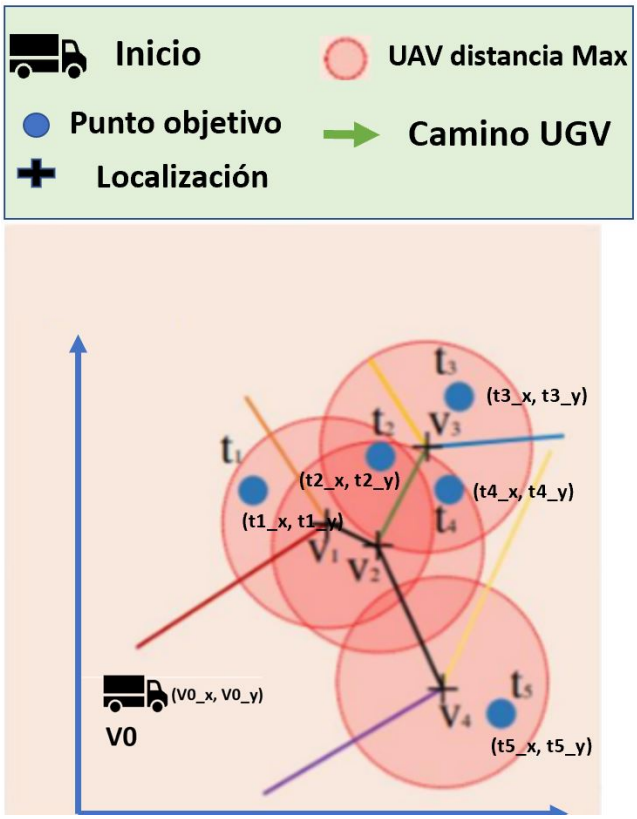


Figura 3.5: Voronoi [1].

Dicho script fue uno de los más complejos de solucionar, ya que su buena optimización y correcto funcionamiento era fundamental para poder continuar el proyecto.

El principal problema que hubo fue a la hora de aplicar la biblioteca de Voronoi. Esto se debe a que la función 'Voronoi' en Matlab funcionaba pasándole dos parámetros de entrada, las 'coordenadas X' y las 'coordenadas Y'. Dado que en Matlab la función Voronoi solo tiene un parámetro de entrada que es el número total de coordenadas, sin separar los puntos en 'X' e 'Y', se tuvo que crear una función cuyo nombre fue *lista futura* para poder resolver dicho problema. Dicha función se muestra en la Figura 8 que se recorre con un bucle 'for'.

Luego como problemas adicionales que sucede en todos los scripts de Python respecto a Matlab es la forma de llamar a las estructuras donde en vez de acceder con un "."se accede con "[]".

```
###### FUNCION CAMBIAR ARRAY #####

lista futura = list()
for i in range (0, len(VOR_Vertices[0])):
    lista futura.append([float(VOR_Vertices[0][i]), float(VOR_Vertices[1][i])])

points = np.array(lista futura)
```

Listing 8: Función cambiar array.

3.5.4. script 'ruta de cálculo UGV'

Este script fue sin duda el mayor reto a nivel de programación, siendo uno de los más extensos y con mayor complicación ya que casi el 100 % de las funciones que existen en Matlab de este código no existía su traducción a Python y se tuvo que crear todas las funciones manualmente para poder incorporar cosas como la función transpose() entre otros.

Otro error en concreto es a la hora de recorrer los bucles *for* ya que al disponer en Matlab de *":"* como se muestra en la Figura 9, en Python hay que hacerlo completamente diferente como se muestra en la Figura 10, donde **arange()** crea una instancia de arrays con valores espaciados uniformemente y **reshape()** se usa para dar una nueva forma a una matriz sin cambiar sus datos.

```
1 for j=1:r
```

Listing 9: Uso de bucles 'for' en Matlab.

```
1 for j in np.arange(0, r).reshape(-1):
```

Listing 10: Uso de bucles 'for' en Python.

3.5.5. Uav compute path y search uav path

Ambos códigos los unifico en esta sección ya que el correcto funcionamiento de uno depende directamente del otro, ya que si una de las dos no funciona no se podría utilizar el otro script.

El principal problema fue que al ir variando los valores de los OPEN (que son una lista), el código daba errores y se tuvo que crear diccionarios específicos para la mayoría de funciones y estructuras para que dependan solo de sí mismas sin depender de otras funciones que puedan ir variando sus valores. En la siguiente figura 11 se muestra como era el uso en Matlab en la línea 1 en verde y lo que se tuvo que realizar en Python para que no dependiese de OPEN, así como añadir diccionarios donde tuve que hacer un listado de diccionarios, todo ello, más explicado estará en el *GitHub*.

```
1 #node = OPEN(j)
2 node = {'parent' : OPEN[j]['parent'],'x' : OPEN[j]['x'],'y' : OPEN[j]['y'],
3 'g' : OPEN[j]['g'],'k' : OPEN[j]['k'],'h' : OPEN[j]['h'],'f' : OPEN[j]['f']}
```

Listing 11: Ejemplo de creación de diccionarios en el código search uav path.

3.5.6. TERRA

El script de TERRA es la base del modelo donde las principales diferencias fueron cambiar todos los formatos de entradas (inputs) y salidas (outputs) a la hora de llamar a las funciones ya que su forma de utilización es completamente diferente a Matlab.

Para una correcta verificación paso por paso de que todo estaba funcionando correctamente se fue generando las salidas y entradas para verificar su correcto funcionamiento antes de pasar siempre a la siguiente función.

3.5.7. Estructura de Clases

En esta sección se va a explicar cómo está estructurado el código dividido en tres grandes bloques.

Primero un primer bloque donde se hablará de los scripts que generan el algoritmo, un segundo bloque con los scripts que componen TERRA y un tercer bloque donde se incluyen los scripts más globales que se llaman desde otras funciones.

3.5.7.1. Primer Bloque

La primera parte está compuesta por los dos Launchers (o lanzadores) que son los scripts encargados de lanzar el resto de scripts y el *scene generator o Generador de escenarios*. El launcher principal es *Test Launcher* que se encarga de lanzar con la ayuda de *Test 2D* todo el programa optimizando la forma de llamar a las funciones y scripts. El *Test 2D* llama a *Generador de escenario* donde como su nombre indica es el script que se encarga de crear el entorno que vamos a utilizar donde se incluyen las coordenadas que se deberán abordar, el área, etc.

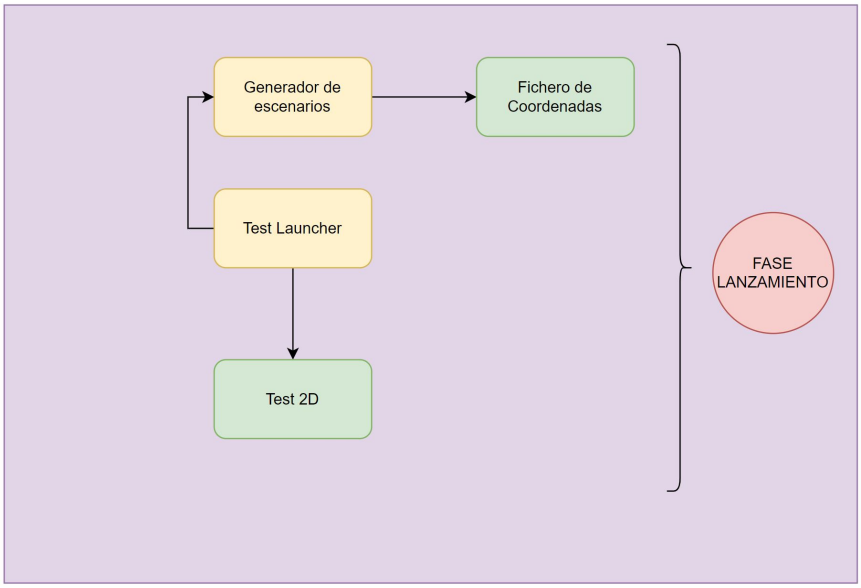


Figura 3.6: Primer Bloque.

3.5.7.2. Segundo Bloque

El siguiente gran bloque está formado principalmente por *TERRA* y todos los scripts que se llaman desde dentro. Este script se encarga de primero llamar todas las variables y figuras necesarias para luego ir llamando al resto de funciones que son:

- Fase 1: Cubrir cada punto de destino con una ubicación.
    1. Voronoi Covering Time Optimization o Optimización del tiempo de cobertura.
    2. Greedy scp.
- Fase 2: Encontrar un subconjunto de cardinalidad mínima de ubicaciones que cubren cada punto objetivo.
    1. Optimización Combinatorial.
- Fase 3: Reducir la distancia de desplazamiento del UGV.
    1. Optimización Gravitacional.
- Fase 4: Calcular la ruta dirigida más corta del UGV.
    1. Problema del viajante / TSP.
    2. Ruta de cálculo UAV.
    3. Ruta de cálculo UGV.

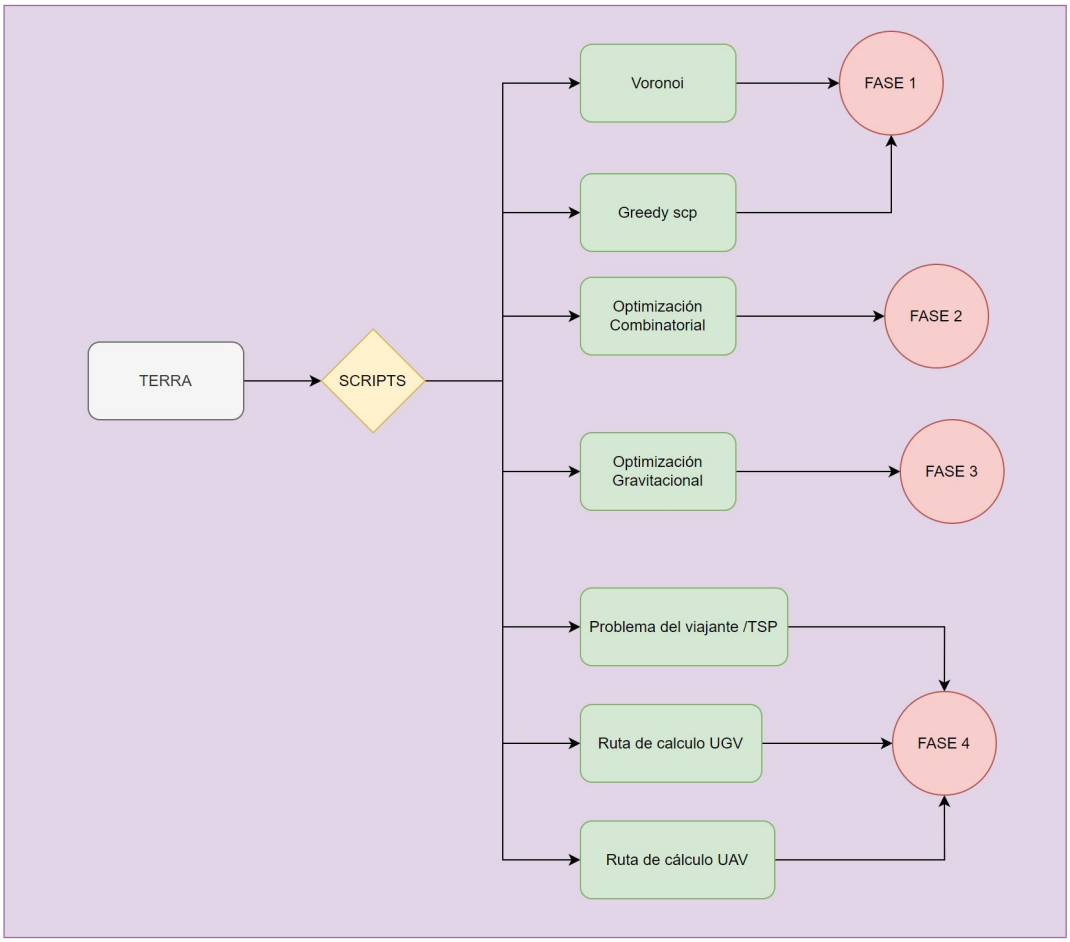


Figura 3.7: Segundo Bloque.

3.5.7.3. Voronoi Optimización del tiempo de cobertura

En este script el primer paso es calcular el diagrama de Voronoi sobre las coodenadas no cubiertas, donde lo primero que se hace es buscar e interceptar los vertices más cercanos para cada ubicación. El segundo paso es eliminar los vértices duplicados y realizar esto para cada iteración donde al final habrá que actualizar las tablas con el vértice de la mínima distancia.

3.5.7.4. Greedy scp

Este script se encarga de arreglar el problema que se crea a la hora de utilizar algoritmos basados en heurísticas donde, dado una serie de números por ejemplo (1,2,3,4,5) divididos en diferentes subconjuntos como por ejemplo (1,3) (4,5) (2,4,5,1) (3,4,1) (3,5) es capaz de encontrar el menor número de subconjuntos que incluyan todos los números, es decir: el 1,2,3, y 5 que sería los subconjuntos (1,3) y (2,4,5,1).

En definitiva, se basa en encontrar un conjunto de soluciones que nos permita cubrir un conjunto de necesidades al menor coste posible, en nuestro caso, la distancia mínima.

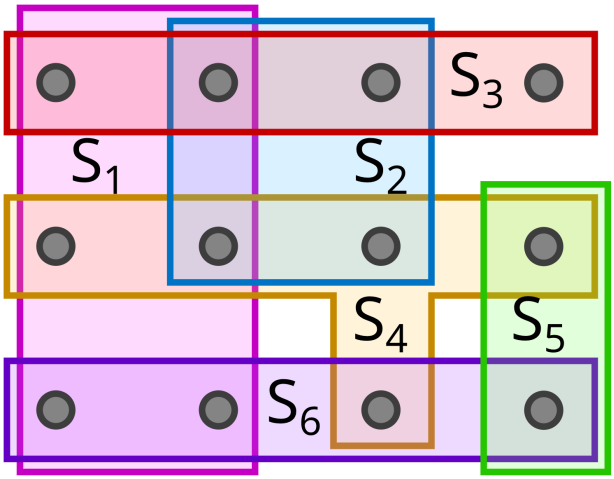


Figura 3.8: Ejemplo de selección de subconjuntos.

3.5.7.5. Ruta de cálculo UGV

Este script solventa los TSP o Problema del viajante para el UGV en el algoritmo de TERRA utilizando como entradas o inputs la lista de vértices y los UGV's utilizando los Parámetros del algoritmo genético y devolviendo como salida u output UGV path, es decir el camino del UGV.

3.5.7.6. Ruta de cálculo del UAV

Esta función se encarga de computar o generar el camino para los UAVs haciendo uso de los dos scripts que más abajo se comentarán que son *search uav operation* y *search uav path*.

3.5.7.7. Build matrix solution

Esta función como su nombre indica se llama desde el script de TERRA y consiste en crear la solución final con el camino para el UGV y el UAV y poder ser luego representados.

Gravitational Optimization

Esta función se encarga principalmente de Optimizar el algoritmo del camino del UGV.

3.5.8. Tercer Bloque

En este bloque final se incluyen el resto de scripts que son nombrados por scripts más pequeños que se encargan o bien de realizar funciones matemáticas o de realizar ciertas funciones para un mejor desarrollo de la aplicación y cuya misión es completar la fase final, es decir la fase 5 que consiste en **calcular la ruta dirigida más corta del UAV para cada sub-ruta** utilizando los scripts siguientes:

- Búsqueda en amplitud.
- Ordenar de mayor a menor.
- Buscar la operación del UAV.
- Buscar la operación del UGV.
- Buscar la ruta del UAV.
- Buscar la ruta del UGV.

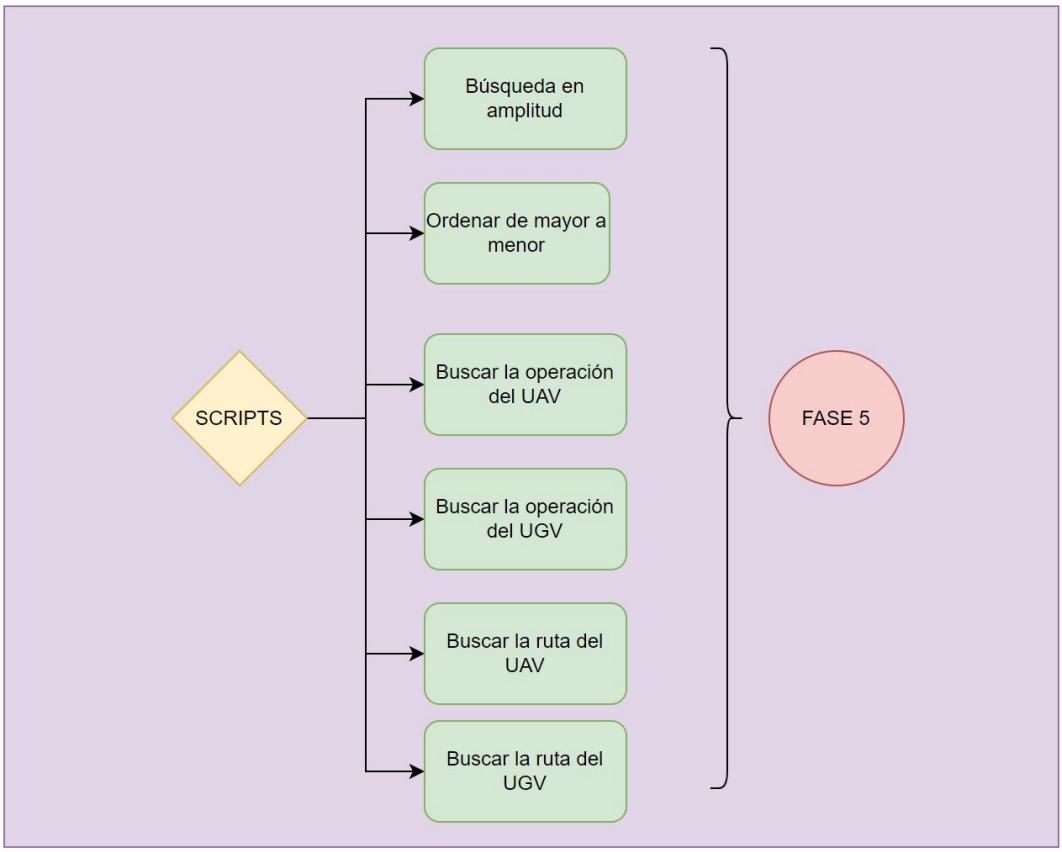


Figura 3.9: Tercer Bloque.

3.5.8.1. Búsqueda en amplitud

En este script se encuentra la función BFS o Breadth-first [25] search que en español significa 'Búsqueda en amplitud' un algoritmo de búsqueda para árboles como el que se muestra en la figura 3.10. Primero, una vez se ha seleccionado un nodo se recorre explorando todos los nodos vecinos en el nivel de profundidad actual antes de bajar al siguiente nivel, es decir primero se recorrería el nodo 'A' y al ser el único de su nivel se pasaría al nivel justo inferior que es el de los nodos 'B', 'C' y 'D' donde se acabaría la búsqueda en el último nivel explorando los nodos 'E' y 'F'.

El orden de la búsqueda quedaría como: 'A', 'B', 'C', 'D', 'E', 'F'. Donde en el caso del algoritmo de TERRA se utiliza para ordenar los 'WPs' del camino para el UAV, es decir, el orden que seguirá.

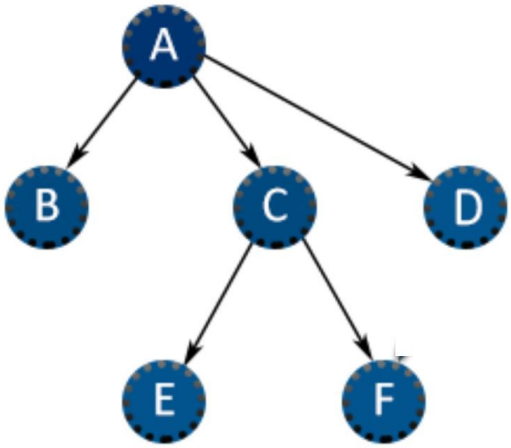


Figura 3.10: Algoritmo Bfs.

3.5.8.2. Ordenar de menor a mayor

Este script se encarga como su nombre deja entrever un algoritmo que ordena las 'listas' desde el nodo con un valor más bajo hasta el nodo con un valor más alto.

3.5.8.3. Buscar la operación del UAV y buscar la ruta del UAV

Son dos scripts bastante complejos a lo que programación se refiere tiene como objetivo calcular la trayectoria dirigida más corta del UAV entre las múltiples subrutas del UAV para minimizar la distancia total que recorren los UAV, donde el objetivo es encontrar los múltiples subcaminos más cortos donde se visite cada punto objetivo y vuelva a la carga vinculada, es decir al UGV.

Para ello se utiliza un algotimo de búsqueda similar al **Algoritmo de búsqueda basado en A*** que busca el camino con menor coste entre un nodo origen y un objetivo. Esto se puede ver más detenidamente en el GitHub pulsando *aquí*

3.6. Funcionamiento

En esta sección se va a explicar el funcionamiento a alto nivel de como sería el funcionamiento del código, y las conexiones y dependencias que existen entre todas las diferentes secciones.

Todo este funcionamiento se puede observar con el siguiente diagrama de flujo de la Figura 3.11 donde se representa qué sucede desde que un usuario inicia la aplicación hasta que obtiene un reporte con las gráficas finales.

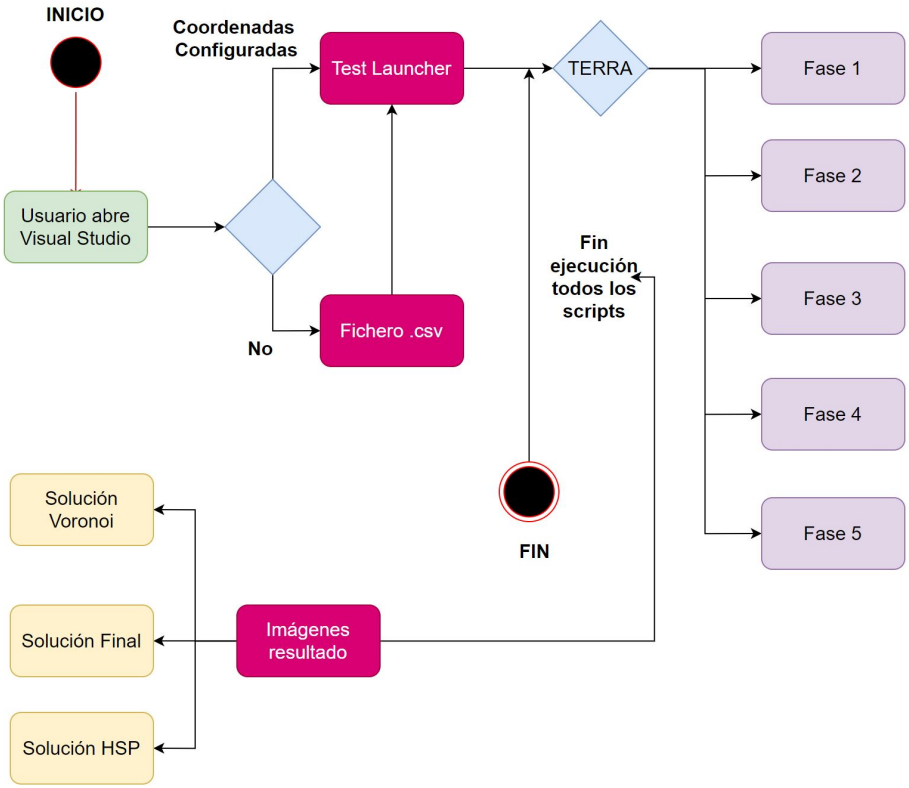


Figura 3.11: Diagrama de Flujo del funcionamiento.

Para ver el código más en profundidad se puede pinchar en el siguiente *enlace* a mi *GitHub* donde se encuentra un fichero Readme con una explicación de su uso. De forma análoga el código y explicación de la aplicación final creada se puede encontrar *aquí*.

En estos repositorios se puede ver tanto la fecha de la última modificación de los ficheros, así como la posibilidad de descargarlos.

De forma adicional, si se desea, se puede pulsar la siguiente imagen para direccionarse a un vídeo explicativo de la aplicación final que se ha grabado y está disponible en YouTube.

Capítulo 4.

Validación y Resultados

En este capítulo se van a exponer los resultados de la validación realizada a la aplicación una vez terminada.

4.1. Objetivos

Para concluir el proyecto, se deseaba realizar una prueba de la aplicación para así poder realizar una validación de la propia aplicación y modelo.

Esta prueba consistió en ir relatando y explicando lo más claramente posible los diferentes pasos que se deben seguir para conseguir hacer funcionar la aplicación. La idea de esta parte del proyecto era evaluar el algoritmo de TERRA creado en dos diferentes puntos:

- Cómo de buena es la solución respecto a los resultados originales del algoritmo en Matlab.
- Utilidad de la aplicación de cara a descubrir posibles puntos mejorables en el futuro.

4.2. Entorno de la aplicación

En este apartado se va a explicar el entorno y los pasos a seguir desde que se abre en el ordenador hasta que se ejecuta la aplicación. Para ello se va a utilizar el compilador *Visual Studio*, herramienta open source que tendremos que desplegar desde la lupa de nuestro dispositivo (en mi caso un portátil) como se adjunta en las siguientes dos capturas donde en la primera Figura 4.1, se muestra su búsqueda en Windows y en la segunda Figura 4.2, se puede observar el código una vez ya abierto Visual Studio.

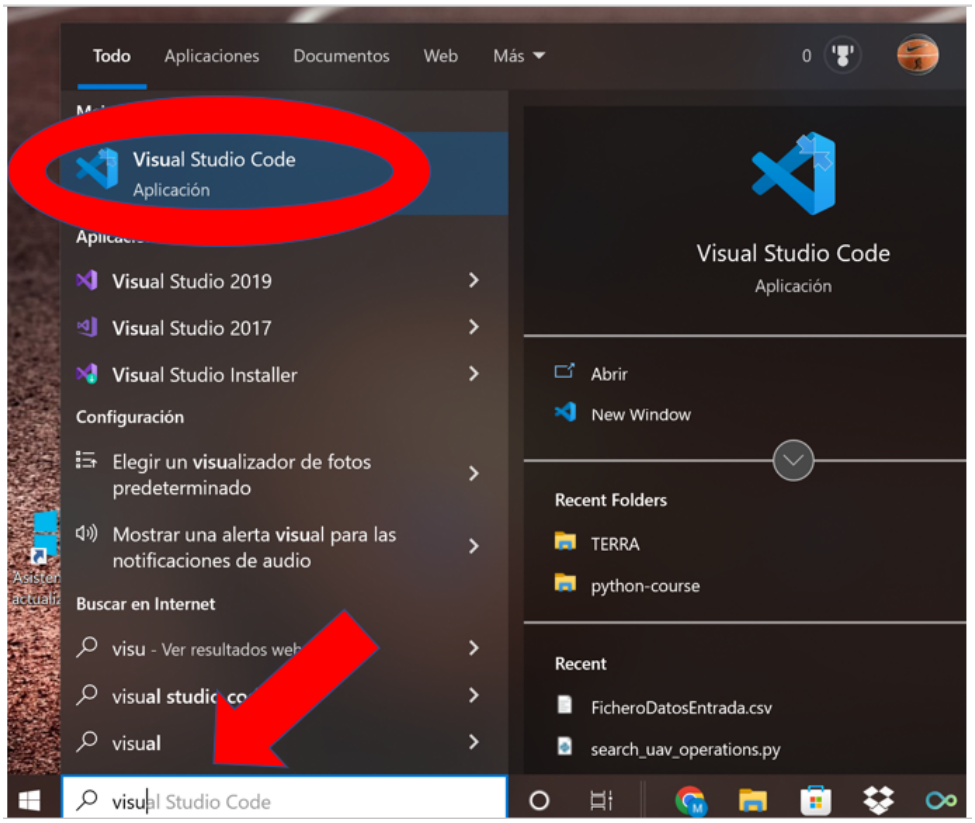


Figura 4.1: Lanzar la aplicación Visual Studio

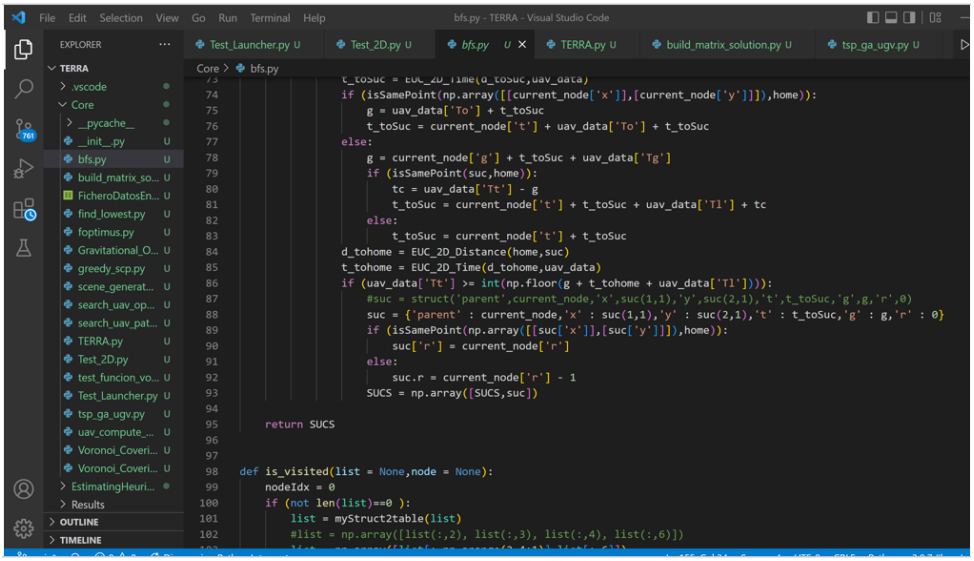


Figura 4.2: Entorno Visual Studio de TERRA

4.3. Explicación de los pasos

En esta sección se va a ir explicando cada paso que tendría que seguir y visualizar el usuario para comprender que es lo que estamos obteniendo.

Para facilitar la experiencia al usuario como ya se ha ahondado en el Capítulo 3 se ha introducido un fichero .csv que se podrá modificar con los datos que el usuario necesite y prefiera sin tener que ser aleatorios para facilitar el uso para una posible colaboración con un entorno final real.

El .csv consta de las coordenadas de los puntos elegidos (formados por las *coordenadas x e y*) así como el número de puntos totales que se desplegarán en el mapa, es decir en este caso 4 puntos donde el punto 1 tiene la coordenada (26, 153) el punto 2 (87, 42), el punto 3 (144, 92) y el punto 4 (32, 35).

	A	B	C
1	CoordX	CoordY	Num_puntos
2	26	153	4
3	87	42	
4	144	92	
5	32	35	

Figura 4.3: Fichero .csv de entrada

Una vez se ha configurado el csv como se muestra en la Figura 4.3 nos colocamos en el script Test Launcher donde tendremos que ir a la esquina superior derecha y seleccionar el botón *Run* tal y como se muestra en la Figura 4.4 donde se abrirá de manera adicional un terminal donde podremos ver como se ejecuta el código.

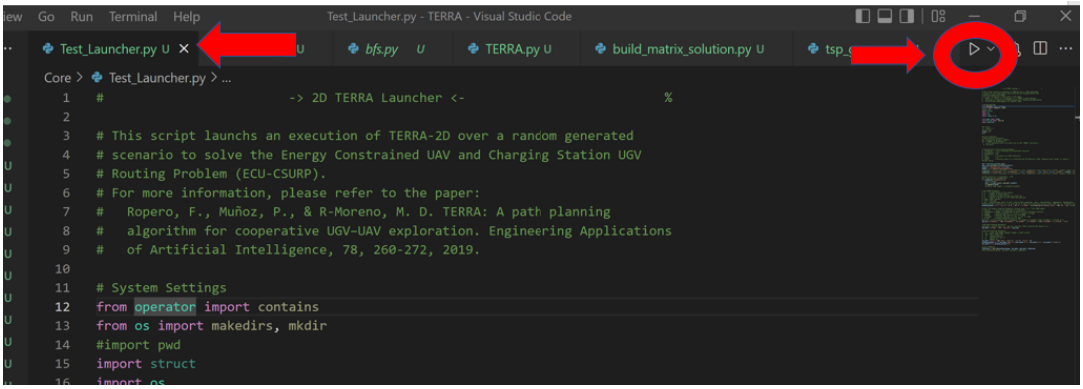


Figura 4.4: Test Launcher.

A continuación, se va a mostrar una tabla que se obtuvo tras realizar la validación, **la cual constó de 32 ejecuciones** de forma tanto unitaria, es decir por fases, como en conjunto comparando los resultados obtenidos en Matlab frente a los resultados en Python, dichas validaciones se realizaron tanto en las diferentes Fases como en la solución final donde a continuación se va a mostrar el procedimiento que se siguió para guardar y visualizar las coordenadas finales que sigue el UGV.

De entre los datos que se representan en las Figuras 4.7, 4.8 y 4.9 y que se pueden validar se encuentran:

- **Puntos destinos:** Las ubicaciones (azules) a las que nuestro algoritmo deberá llegar a través de los UAVs
- **El camino del UAV:** El recorrido de los UAVs hasta alcanzar su objetivo.
- **Subconjuntos:** Los distintos conjuntos (circunferencias rojas) que engloban las diferentes ubicaciones (azules).
- **El recorrido final:** El camino final que realiza el UGV (color verde) desde el inicio del algoritmo hasta que finaliza su misión.

Se ha elegido para la siguiente fase mostrar sólo el análisis del dato más relevante el cual es el camino que recorre el UGV (con color verde.)

La siguiente Figura 4.5 muestra un 'csv' donde se muestra la línea verde que sigue el UGV con sus diferentes coordenadas representadas con la coordenada 'X' e 'Y' respectivamente (como más adelante se muestra en el ejemplo). Este fichero se obtuvo como se muestra en la Figura 12 tras incorporar dichas líneas de código que nos permitiesen guardar la solución del camino final del UGV para poder compararlo y analizarlo frente al algoritmo en Python después de ir cambiando los puntos por los que debe pasar el algoritmo con las 32 pruebas.

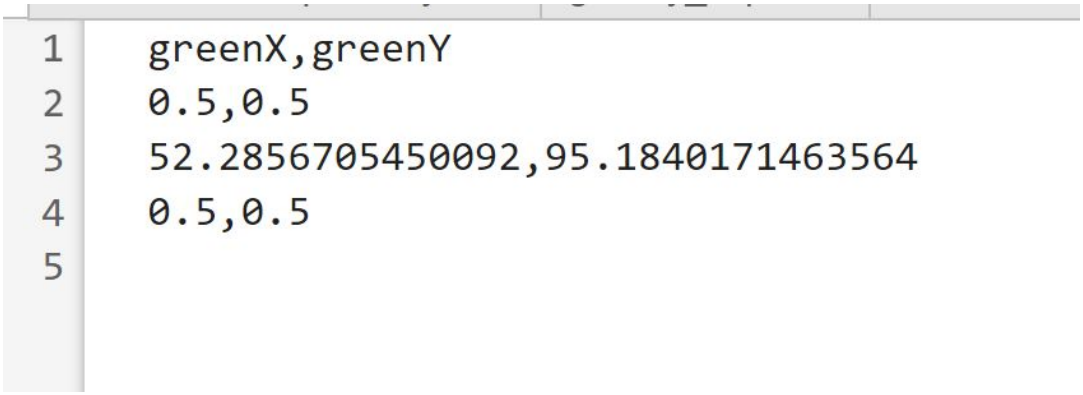


Figura 4.5: Csv de la ruta del UGV en Matlab.

```
% Create a table with the data and variable names
%T = table(Ax, Ay, Bx, By, Rx, Ry,  'VariableNames', { 'greenX', 'greenY',} );
T = table(Ax, Ay,   'VariableNames', { 'greenX', 'greenY'} );
% Write data to text file
disp('T');
disp(T);
writetable(T, '/MATLAB Drive/TERRA/Core/GreenData.txt')
```

Listing 12: Código para almacenar el camino del UGV en Matlab.

A su vez en Python se ha creado otro 'csv' que muestra las coordenadas que sigue el UGV como se puede observar en la Figura 4.6 y el cual fue creado utilizando el lenguaje Python como se muestra en la Figura 13 donde en la línea 1 dentro de 'header' se escriben los nombres que queremos que tengan dichas variables al ser almacenadas en el 'csv', en la línea 2 dentro de 'data' las variables que queremos guardar, en nuestro caso las coordenadas 'X' e 'Y' de la ruta del UGV.

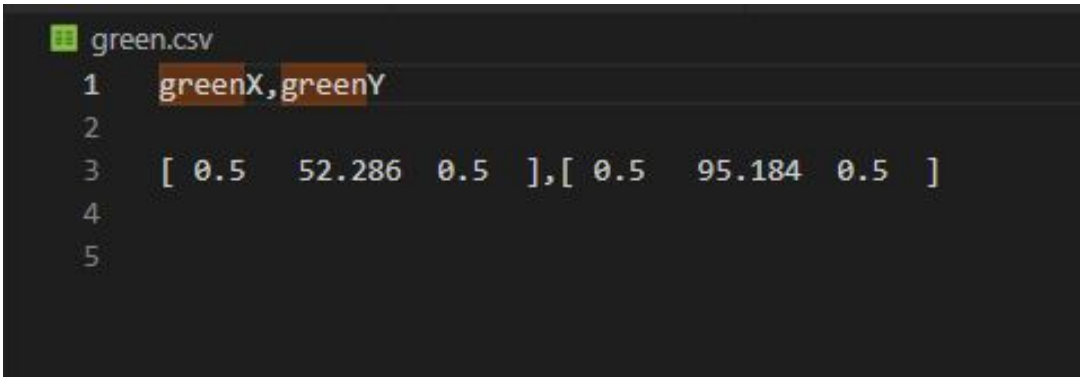


Figura 4.6: Csv de la ruta del UGV en Python.

```
header = ['greenX', 'greenY']
    data = [ugv_path[:,0], ugv_path[:,1]]

    with open('green.csv', 'w', encoding='UTF8') as f:
        writer = csv.writer(f)

        # write the header
        writer.writerow(header)

        # write the data
        writer.writerow(data)
```

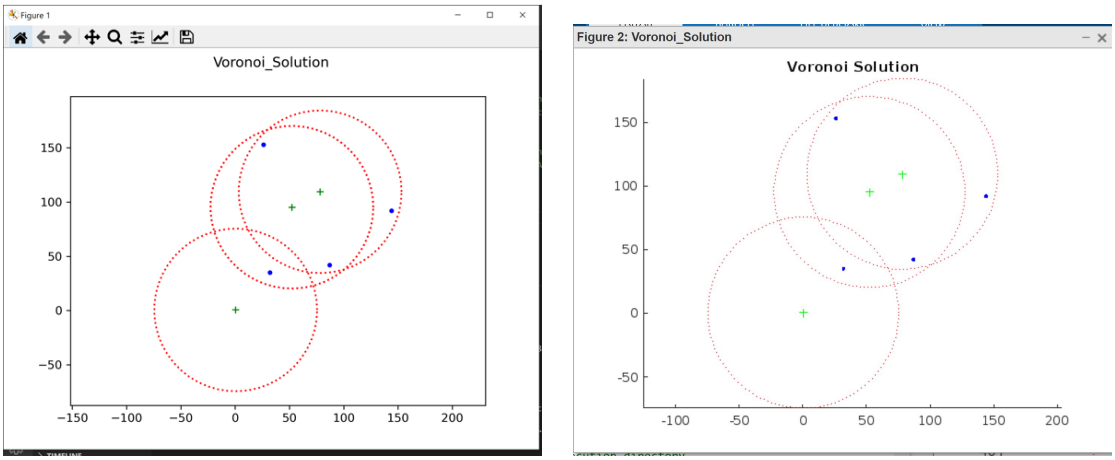
Listing 13: Código para almacenar el camino del UGV en Python.

Analizando los resultados, se puede observar que el formato de dichos 'csvs' es diferente, ya que mientras que en Python coloca todas las coordenadas de las 'X' y todas las coordenadas de las 'Y' en una misma celda y en horizontal en Matlab se muestra de forma vertical cada una de las coordenadas por separado. La única diferencia que hay entre ambas soluciones es que en Matlab tiene 13 decimales y en Python a partir del tercer decimal redondea.

Dicho ejemplo se ha probado 32 veces con diferentes datos siendo siempre la única diferencia que en Python redondea a partir del tercer decimal.

Por último se van a ir mostrando las diferentes Figuras de las soluciones a nivel gráfico para poder entender y verificar de una forma visual los resultados, en todas las Figuras se mostrará al lado izquierdo (Figura a) los resultados obtenidos tras ejecutar el algoritmo TERRA en Python frente a las Figuras al lado derecho (Figura b) serán los resultados obtenidos con la versión inicial de Matlab que nos servirán para verificar si los resultados concuerdan.

Una vez ejecutado el código lo primero que se visualiza es Voronoi Solution en la Figura 4.7 donde se visualizan los 4 puntos en color azul y los diferentes caminos que puede llegar a tomar. Comparando la solución 4.8 el resultado de Matlab concuerda con el de Python.

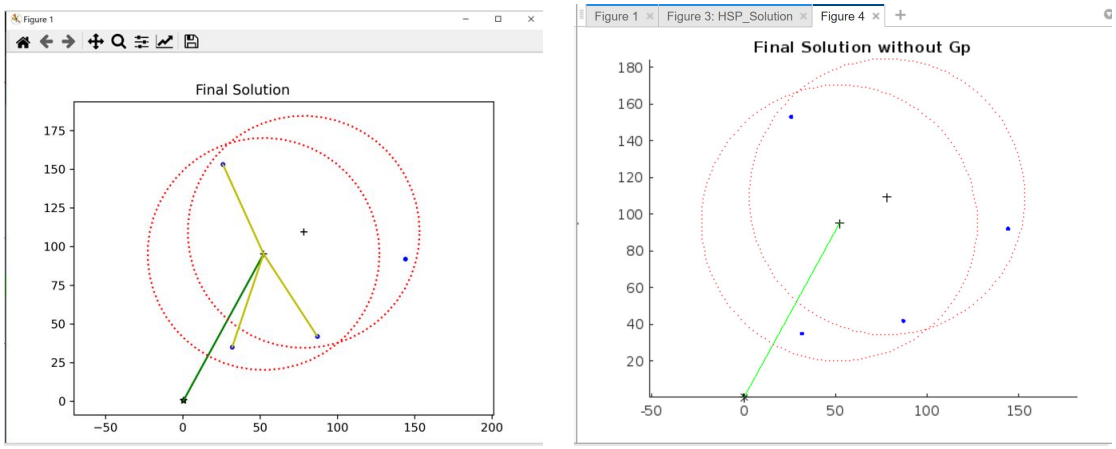


(a) *(b)*

Figura 4.7: Solución Voronoi Python *(a)* Solución Voronoi Matlab *(b)*

A continuación, volvemos a darle al botón 'continue' en nuestro compilador 'Visual Studio' y se ejecutarán las diferentes fases previamente explicadas en el Capítulo 3 donde se visualizará en la Figura 4.8 la Solución Final sin solución global donde se puede ya observar en verde el trayecto definitivo que realizará el UGV y las diferentes posiciones que tomarán los UAVs.

Tal y como se muestra en la Figura 4.9, se puede ver que la solución en Python (a) frente a la solución en Matlab (b) es la misma por lo que podemos dar como validada esta parte.



(a) *(b)*

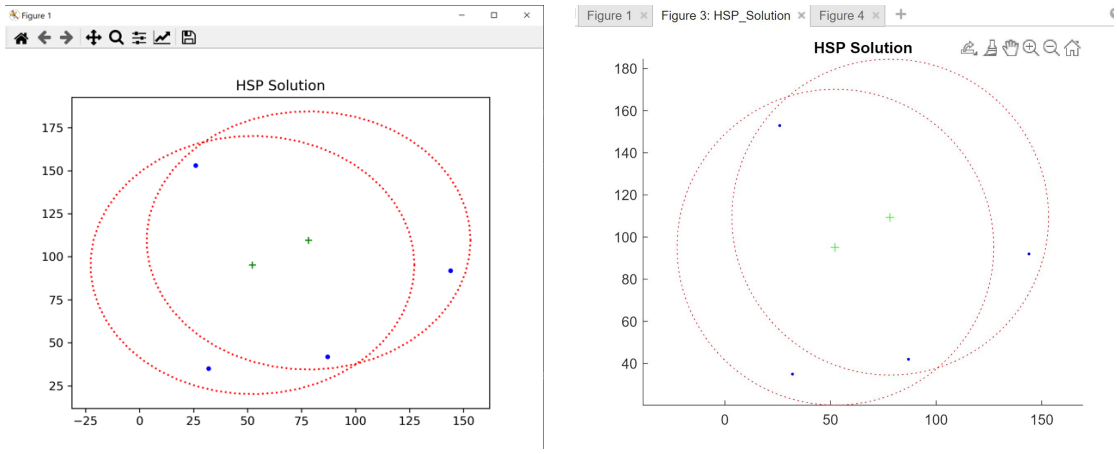
Figura 4.8: Solución Final sin la Posición Global *(a)* Python *(b) Matlab*

Finalmente se muestra la última Figura 4.9, la Solución del conjunto de bloques. Esta etapa tiene como objetivo encontrar un conjunto mínimo de vértices cuyas áreas garanticen la cobertura total del conjunto de puntos objetivo, es decir que engloben todos los puntos azules, es decir, de las 3 regiones (círculos rojos) sólo serán necesario la utilización de estos dos para englobar todos los puntos.

En informática a este proceso se le llama HSP (*Hitting Set Problem* o en castellano problema del conjunto de cobertura) que sería la globalización del problema 'Greedy SCP' que ya fue explicado con un ejemplo visual en el Capítulo 3.

Por último, comparamos la solución obtenida en Python frente a Matlab y podemos verificar que los resultados son correctos.

Justo al cerrar esta imagen el algoritmo TERRA termina de ejecutarse y cerrarse ya exitosamente, dando por concluida la parte de comprobación y verificación de una forma más visual y gráfica que todo funciona correctamente.



(a) *(b)*

Figura 4.9: Solución HSP Python *(a)* Solución HSP Matlab *(b)*

4.4. Utilidad de la aplicación en el día a día

A raíz de la prueba de la aplicación y las respuestas obtenidas en las Figuras que se han ido visualizando, se pueden arrojar varios resultados.

Lo primero, es que al utilizar un .csv en vez de datos randomizados como la versión precedente podemos modificar de una forma más sencilla a la vez que exportar e importar datos de una manera más fácil y más clara de cara a poder utilizarlo en un entorno final real.

Por otro lado, podemos observar que al representar los datos de forma visual es mucho más intuitivo de cara a poder presentárselo a una empresa así como gente ajena al proyecto y que pueda llegar a comprender su funcionamiento y quiera interesarse más en el algoritmo de TERRA que si les hubiéramos mostrado una cantidad de datos en una hoja plana donde dificultaría mucho su explicación.

Capítulo 5.

Conclusiones

5.1. Conclusiones

TERRA ha demostrado la posibilidad de crear un modelo de cooperación robótica sin la necesidad de incorporar el factor humano solo teniendo que indicar los puntos y destinos el UGV debe seguir utilizando el 'fichero .csv'. Además, se ha demostrado la posibilidad de usar un modelo mucho más optimizado como Python respecto a Matlab que puede ayudar en proyectos existentes y futuros manteniendo la calidad y eficiencia del modelo asegurando la constante evolución de imports y pluggins que existen en este lenguaje de programación.

Además, con la utilización del algoritmo TERRA en Python, se elimina uno de los principales problemas de Matlab que es el coste de las licencias [26] junto a algunas inconsistencias de su lenguaje. El coste de una licencia de MATLAB es a día de hoy de 2000€ más los toolboxes que se requieran (1000€/ud) más 5000€ para el compilador que nos permitirá comercializar nuestro trabajo, mientras que tanto Python como sus paquetes o librerías son gratuitos y está cubierto por la licencia BSD (licencia de software libre) por lo que se permite su comercialización sin ningún tipo de restricción.

Por otro lado, se ha demostrado la posibilidad de usar nuevos pluggins y sistemas más eficientes y con un mayor abanico de posibilidades como es Python frente a Matlab donde las líneas futuras podrán desarrollar una mayor versatilidad de progresión.

Por consiguiente, examinando los objetivos que se habían planteado antes de empezar el trabajo, se puede observar cómo se han estudiado las herramientas para intentar traducir el código. Más tarde se analizó el estado del arte de las aplicaciones existentes. Finalmente, se creó la aplicación usando dichas tecnologías analizadas y con el lenguaje de programación Python con el objetivo de mejorar e incorporar al proyecto una mayor portabilidad y escalabilidad y se probó el funcionamiento de dicha aplicación con un estudio y validación con diferentes gráficas e imágenes mostradas en el Capítulo 4.

5.2. Líneas futuras

La aplicación tiene una limitación al tener que instalar *Visual Studio* y disponer de un lector de .csv ya sea por ejemplo uno de los más conocidos como es *Excel*. Por ello, en vistas a poder acercar esta aplicación a la mayor cantidad de entes y empresas se debería investigar y estudiar una forma de poder anexionar y fusionar en una aplicación móvil o más visual ambas cosas para una mayor eficiencia de uso y mayor claridad para un usuario que desconozca la programación.

De forma adicional, se podría añadir una funcionalidad extra a los reportes para indicar mejoras o posibles diferentes trayectos. Para conseguir esta funcionalidad sin poner en riesgo la privacidad de los usuarios se podría hacer uso de un Aprendizaje Automático Federado donde los datos se gestionen dentro del propio dispositivo y no se almacenen en la nube pudiendo poner en peligro la información personal de los usuarios como las ubicaciones de sus hogares, nombres, etc.

A fin de conseguir dicha lista se podría usar en un futuro alguna herramienta de código abierto como *TensorFlow* herramienta de *Google* para realizar todo lo previamente comentado, para mantener la privacidad de los datos personales de los usuarios, ayudado de un servidor ya sea por ejemplo en Heroku u otro organismo de la nube.

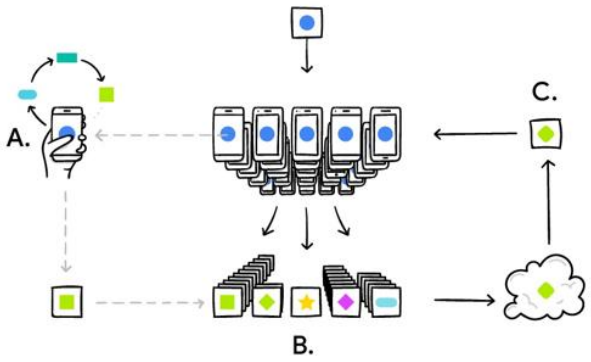


Figura 5.1: Funcionamiento Aprendizaje Automático Federado, Fuente: Google AI Blog [2].

Finalmente, vista la utilidad de TERRA, se podrían pensar nuevos proyectos donde se pueda aprovechar su potencial con proyectos nuevos o ya existentes para intentar anexionarlo logrando un proyecto mucho más potente.

Bibliografía

[1] Fernando Ropero, Pablo Muñoz, and María D. R-Moreno. Terra: A path planning algorithm for cooperative ugv–uav exploration. *Engineering Applications of Artificial Intelligence*, 78:260–272, 2019.

[2] Brendan McMahan and Daniel Ramage. *"Federated Learning: Collaborative Machine Learning without Centralized Training Data"*. *Google AI*. `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html` (accessed March 16, 2021).

[3] Hanif Ullah, Nithya Gopalakrishnan Nair, Adrian Moore, Chris Nugent, Paul Muschamp, and Maria Cuevas. 5g communication: An overview of vehicle-to-everything, drones, and healthcare use-cases. *IEEE Access*, 7:37251–37268, 2019.

[4] MarketsAndMarkets. 'Drone Logistics and Transportation Market by Solution (Warehousing, Shipping, Infrastructure, Software), Sector (Commercial, Military), Drone (Freight Drones, Passenger Drones, Ambulance Drones), and Region - Global Forecast to 2027'. *MarketsAndMarkets*. `https://www.marketsandmarkets.com/Market-Reports/drone-logistic-transportation-market-132496700.html` (accessed Jun 21, 2022).

[5] BBC. 'Ataques petroleros saudíes: drones y misiles lanzados desde Irán: EE. UU.'. *BBC*. `https://www.bbc.com/news/world-middle-east-49733558` (accessed Jun 23, 2022).

[6] Omar Kardoudi . 'Los informes que confirman el desastre de los drones de reparto de Amazon'. *ElConfidencial*. `https://www.elconfidencial.com/tecnologia/novaceno/2022-04-17/futuro-envios-drones-amazon-desastre_3408415/` (accessed Jun 23, 2022).

[7] María Mercedes Plaza Benítez . 'Sistemas de detección y neutralización de UAVs'. *DocPlayer*. `https://docplayer.es/81611982-Sistemas-de-deteccion-y-neutralizacion-de-uavs.html` (accessed Jun 22, 2022).

[8] Muhammad Zohaib Anwar, Zeeshan Kaleem, and Abbas Jamalipour. Machine learning inspired sound-based amateur drone detection for public safety applications. *IEEE Transactions on Vehicular Technology*, 68(3):2526–2534, 2019.

[9] Wikipedia . 'Problema del viajante'. *Wikipedia*. `https://es.wikipedia.org/wiki/Problema_del_viajante` (accessed Jun 24, 2022).

[10] Francois Chollet. *Deep Learning with Python*. Shelter Island, NY 11964: Manning, 2018, pp. 4-5.

[11] Ming Zhang, Wei Li, Mengmeng Wang, Songrui Li, and Boquan Li. Helicopter–uavs search and rescue task allocation considering uavs operating environment and performance. *Computers Industrial Engineering*, 167:107994, 2022.

[12] Universidad de Granada . 'Algoritmos evolutivos para problemas multiobjetivos'. *UGR*. https://sci2s.ugr.es/sites/default/files/files/Teaching/GraduatesCourses/Bioinformatica/Tema%2012%20-%20MOEAS.pdf (accessed Jun 24, 2022).

[13] Yang Chen, Mengqing Chen, Zhihuan Chen, Lei Cheng, Yanhua Yang, and Hui Li. Delivery path planning of heterogeneous robot system under road network constraints. *Computers Electrical Engineering*, 92:107197, 2021.

[14] Amin Basiri, Valerio Mariani, Giuseppe Silano, Muhammad Aatif, Luigi Iannelli, and Luigi Glielmo. A survey on the application of path-planning algorithms for multi-rotor uavs in precision agriculture. *Journal of Navigation*, 75(2):364–383, 2022.

[15] Franco P Preparata and Michael I Shamos. *Computational geometry: an introduction*. Springer Science Business Media, 2012, pp. 4-5.

[16] Georges Voronoi. *Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les parallélloèdres primitifs., publisher = reine und angewandte Mathematik 134, year = 1908, pp. 198-287,.*

[17] Wikipedia . 'Problema del conjunto de Cobertura'. *Wikipedia*. https://es.wikipedia.org/wiki/Problema_del_conjunto_de_cobertura (accessed Jun 24, 2022).

[18] *'SMOP'. Github*. https://github.com/victorlei/smop#:~:text=SMOP%20is%20Small%20Matlab%20and,to%20be%20faster%20than%20octave. (accessed May 30, 2022).

[19] *'5 Ways To Convert Matlab To Python'. Python Pool*. https://www.pythonpool.com/convert-matlab-to-python/ (accessed May 30, 2022).

[20] *'5 Ways To Convert Matlab To Python'. Python Pool*. https://www.pythonpool.com/convert-matlab-to-python/ (accessed May 30, 2022).

[21] Alejandro Aguilera Alcalde. 'FederatedLearnig Demo'. *GitHub*. https://github.com/alejandroaguileraalcalde-ing/FederatedLearnig_Demo (accessed March 16, 2021).

[22] M.P. Ïnternet de las Cosas: en 2021 habrá 28.000 millones de dispositivos conectados
. *Expansión*. https://www.expansion.com/economia-digital/innovacion/2016/06/01/574eaceee2704e12328b4668.html (accessed March 16, 2021).

[23] *'Ventajas python'. KeepCoding*. https://keepcoding.io/blog/ventajas-y-desventajas-de-python/ (accessed May 30, 2022).

[24] *'Anaconda'. anaconda*. https://www.anaconda.com/ (accessed May 30, 2022).

[25] Redacción. 'BFS frente a DFS'. *Open4Tech*. https://open4tech.com/bfs-vs-dfs/ (accessed Jun 3, 2022).

[26] CAChemE. 'Python para usuarios de Matlab'. *Cacheme*. https://www.cacheme.org/python-para-usuarios-de-matlab/ (accessed Jun 16, 2022).

Anexo A.

Aspectos éticos, económicos, sociales y ambientales

En esta sección se analizarán los principales impactos y responsabilidades relacionadas con el trabajo.

A.1. Introducción

Como bien se ha expuesto en la Introducción, la Cooperación Robótica es algo en constante crecimiento en las últimas décadas y algo que inevitablemente ha llegado para quedarse e intentar solucionar y facilitar la vida a los seres humanos. A continuación, vamos a ir describiendo los principales impactos y consecuencias del algoritmo TERRA y la implicación de traducirlo.

A.2. Descripción de impactos relevantes relacionados con el proyecto

Este proyecto tiene diferentes impactos entre los que destacan un impacto social, económico, medioambiental y una responsabilidad ética y profesional.

Existe un impacto social al ser una aplicación que afecta a las personas de todas las edades, ya que hoy en día todo el mundo utiliza el servicio de mensajería de paquetes habiendo conseguido el creador de la compañía más importante en este sector, Amazon, ser el hombre más rico del mundo. Por otro lado, también supone un impacto económico ligado al uso de UAVs y UGVs que podrán suponer una abaratación en costes medioambientales al conseguir reducir los gastos en combustibles contaminantes y no reciclables y una responsabilidad ética y social al involucrar los datos y privacidad de los usuarios en las mejoras propuestas de Aprendizaje Automático en el apartado 'Líneas Futuras'. Esto se verá en mayor detalle en el siguiente apartado.

A.3. Análisis detallado de alguno de los principales impactos

En primer lugar, supone un impacto social al ser una aplicación dirigida a las principales marcas u empresas que quieran ahorrar costes y eficiencia a la hora de trasportar paquetes a través de nuestros UAVs y UGVs que estarán en contacto directo con las personas a la hora de recibir los paquetes.

En segundo lugar, implica un impacto económico en diferentes ámbitos. Por un lado, permite mejorar la eficiencia y mejora de los modelos de entrega de paquetes. Esta tecnología permite que se consiga una revolución en el sector y ámbito tecnológico sobre todo en el de la ingeniería que tendrá la motivación de buscar crear cada vez mejores herramientas y robots más eficientes para reducir gastos ya sea reduciendo el peso de los UAV para reducir el consumo de sus baterías, ayudar a mantener un mundo más limpio a través de grandes inyecciones de dinero en laboratorios y centros industriales de mejoras y pruebas.

En tercer lugar, este TFG tiene un impacto medioambiental al proponer el uso de drones que tengan un funcionamiento que utilice meramente energías renovables que apoyen el plan de sostenibilidad marcado por la Unión Europea de 2030.

Por último, tiene responsabilidad ética y profesional. Esto es sumamente importante ya que una mala configuración del modelo TERRA o de los robots en la fase final de acercamiento a los usuarios finales que compren los paquetes puede poner en riesgo su salud por tanto será de rigor cumplimiento una buena securización y privacidad de los usuarios y del código base que no pueda ser vulnerabilitado.

A.4. Conclusiones

Como se ha podido ver en las anteriores secciones, *TERRA* es un algoritmo que demuestra la posibilidad de ser respetuoso con el medio ambiente y mantener la eficiencia de sistemas de Cooperación Robótica teniendo un impacto social, económico y medioambiental positivo en ellos y la industria tecnológica.

La conversión del algoritmo a Python también supone una mejor portabilidad que podrá ayudar a ser utilizado con otros ecosistemas que son más compatibles con lenguajes como Python frente al uso de Matlab.

Anexo B.

Presupuesto económico

En este Apéndice se va a detallar el presupuesto económico del Trabajo Fin de Grado. Para ello se hará una división del coste de personal que ha supuesto el proyecto y por otro lado los costes de recursos materiales.

- **Personal**: Para calcular el coste de personal que ha supuesto el proyecto se ha tenido en cuenta el salario medio de un ingeniero junior, el cual se sitúa en 23.000-24.000 € brutos anuales. Este estudio se ha hecho en base a conocidos recién licenciados en el propio grado así como en base a mi experiencia personal de las diferentes ofertas que he recibido este último año.

  El trabajo comenzó a finales de enero hasta mediados de junio de 2022. De forma aproximada se puede estimar un total de 6 meses x 15 días/mes x 5 h/día, que resulta en un total de 450 horas.

  Teniendo en cuenta el salario medio previamente comentado y la dedicación, se ha reflejado el coste de personal (ver Tabla B.1).

  Cabe mencionar que también se ha tenido en cuenta el salario de mi co-tutores así como reuniones con un ex alumno de la universidad creador del algoritmo TERRA, el cual se refleja adicionalmente en la Tabla B.1.

	Coste horario (€)	**Horas**	**Total (€)**
Director del trabajo	60	30	1800
Reuniones con ex-alumno	20	30	600
Estudiante de ingeniería	15	450	6750
TOTAL			**9150**

Tabla B.1: Costes de personal.

- **Costes de recursos materiales**: se ha tenido en cuenta los dispositivos que se han utilizado para realizar el proyecto, los cuales han sido un ordenador Windows HP con 4GB de memoria RAM, la aplicación de Matlab cuyo gasto corre a cuenta de la propia Universidad y un disco duro externo de 2TB. Todo ello sin tener en cuenta el gasto de un entorno de trabajo o la propia luz, que en el punto que estamos es altamente relevante.

  El coste de la licencia de Matlab Mathworks tiene un precio anual de 800, pero dicho cargo se ha descontado ya que al utilizar una licencia universitaria ha supuesto un gasto de 0.

  Todo esto se puede ver reflejado en (ver Tabla B.2).

	Tiempo de vida (años)	Uds.	Coste (€)	Amortización (€/mes)	Uso (meses)	Total (€)
Portátil	4	1	899	14,98	6	89,88
Disco duro WD 2 TB	5	1	69	1,1	6	6,6
TOTAL						**98,68**

Tabla B.2: Costes de recursos materiales.

Finalmente, la suma de los costes de personal y recursos materiales se ve reflejada en Tabla B.3, siendo el total de 11.190,90€.

	Coste
Costes de personal	9150€
Costes de material	98,68€
Subtotal	9.248,68€
IVA	1.942,22€
Total	11.190,90€

Tabla B.3: Costes totales.