UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

Máster Universitario en Ingeniería Aeronáutica

Trabajo de Fin de Máster

Implementación y Diseño de un Sistema Avanzado de Guiado y Control del Movimiento en Superficie para el Aeropuerto de Madrid

José Ibero Alastuey

Curso 2021-2022

9 de Septiembre de 2023

Título: Implementación y Diseño de un Sistema Avanzado de Guiado y Control del Movimiento en Superficie para el Aeropuerto de Madrid

Autor: JOSÉ IBERO ALASTUEY

Tutor: ROCÍO GUTIÉRREZ RICHAUD

Titulación: Máster Universitario en Ingeniería Aeronáutica

Curso: 2021/2022

RESUMEN

En el siguiente Trabajo Fin de Máster, se aborda la necesidad de mejorar las operaciones en tierra en el aeropuerto de Madrid. Se identifica una gran oportunidad para implementar un Sistema Avanzado de Guiado y Control del Movimiento en Superficie (A-SMGCS). El sistema actualmente en uso se considera obsoleto y presenta ciertas desventajas en comparación con sistemas más avanzados, los cuales reducen los tiempos de taxi, aumentan la seguridad y disminuyen la carga de trabajo de los controladores.

Además, los avances tecnológicos, la inteligencia artificial y el desarrollo del Machine Learning permiten la creación de herramientas y sistemas que, hasta hace unos años, resultaban complejas y difíciles de obtener. Aprovechando esta evolución, resulta interesante considerar esta opción con el propósito de convertir el Aeropuerto de Madrid en uno de los aeropuertos más eficientes, seguros y sostenibles.

En este trabajo se destacan los beneficios que esto traería al aeropuerto de Madrid, presentando las diferentes fases de implementación necesarias para que el proyecto sea ejecutado con éxito y cumpla con todos los requisitos existentes. Además, con el objetivo de demostrar el tipo de herramienta que se pretende desarrollar y los posibles beneficios que puede aportar, se ha creado un código en Python que resalta las posibles mejoras en los tiempos de taxi. A través de este análisis, se logra reducir significativamente el tiempo de taxi, lo que confirma la capacidad de mejorar las operaciones de las aeronaves en el aeropuerto.

Palabras clave: A-SMGCS, Control, Guiado, Python, Ruta, Taxi

ABSTRACT

In the following Master's Thesis, the need to enhance ground operations at Madrid Airport is addressed. A significant opportunity is identified to implement an Advanced Surface Movement Guidance and Control System (A-SMGCS). The current system in use is considered obsolete and exhibits certain disadvantages compared to more advanced systems, which reduce taxi times, increase safety, and decrease controller workload.

Furthermore, technological advancements, artificial intelligence, and the development of Machine Learning enable the creation of tools and systems that, until a few years ago, were complex and challenging to attain. Leveraging this evolution, it is interesting to consider this option with the aim of transforming Madrid Airport into one of the most efficient, secure, and sustainable airports.

This work highlights the benefits that this would bring to Madrid Airport, presenting the different implementation phases required for the project to be executed successfully and meet all existing requirements. Additionally, in order to demonstrate the type of tool intended to be developed and the potential benefits it can provide, a Python code has been created that highlights possible improvements in taxi times. Through this analysis, a significant reduction in taxi time is achieved, confirming the ability to enhance aircraft operations at the airport.

Keywords: A-SMGCS, Control, Guidance, Python, Route, Taxi

Agradecimientos

A mis padres y hermana, quienes siempre han sido mi mayor apoyo y fuente de inspiración. Gracias por creer en mí, por sacrificarse para brindarme esta oportunidad y por su amor incondicional que me ha impulsado a seguir adelante y llegar a donde me encuentro hoy.

Agradezco a mi tutora de trabajo, Rocío, por su orientación experta y apoyo constante a lo largo de este viaje académico. Sus consejos, planteamientos y conocimientos han sido invaluables en la realización de este trabajo.

A mis amigos, con quien he compartido risas, desafíos y momentos inolvidables durante estos años. Gracias por estar a mi lado y por ser una fuente de motivación constante.

Por último, agradecer a todas las personas que, de una forma u otra, han influido positivamente en mi trayectoria académica y personal.

Este trabajo fin de máster es un reflejo de todos vosotros y de lo que he aprendido a lo largo de este camino. Gracias por ser parte de esta experiencia.

Con cariño y agradecimiento sincero,

José

Índice de contenidos

Figuras

Tablas

.

Códigos

Capítulo 1. Introducción

A lo largo de la historia, el tráfico aéreo ha evolucionado para mejorar las operaciones, siendo cada vez más seguras, menos costosas y, últimamente, buscando que cada vez sean menos contaminantes. Estos objetivos siguen formando parte de los buscados en el presente, sin embargo, el gran crecimiento de la demanda de pasajeros y mercancía hace que se tengan que implementar nuevas tecnologías para alcanzar mejores resultados. Además, cabe destacar que se están llegando a ciertos números de demanda, los cuales generan momentos de estrés y saturación, tanto en el espacio aéreo como en el entorno aeroportuario. Debido a ello, es necesario buscar aquellas soluciones que permitan reducir los riesgos y efectos que puedan tener estos momentos de alta demanda, manteniendo el sistema en un nivel óptimo de funcionamiento.

No obstante, para llegar al punto donde se encuentra la aviación actualmente, la tecnología ha tenido que avanzar continuamente para poder cumplir con los requisitos impuestos. Este avance ha sido tal que, a día de hoy, los aeropuertos son mega infraestructuras conformadas por diversos sistemas de alta complejidad. De este modo, para que todos estos sistemas funcionen en concordancia, es necesario que los mismos pasen numerosas pruebas que demuestren las ventajas que pueden aportar. Aparte, estos deben funcionar con los demás sistemas del entorno, sin crear conflictos entre ellos.

Actualmente, el tráfico aéreo sigue en continuo crecimiento, y aunque se hayan pasado por diversas crisis económicas, o sanitarias, todo apunta a que la demanda va a seguir en aumento. Por este motivo, los sistemas e infraestructuras deben adecuarse a esta previsión para así poder mantener los niveles de seguridad y de operabilidad adecuados con los esperados.

Uno de los problemas que se va a encontrar la industria es la saturación de tráfico en el entorno aeroportuario, donde se va a alcanzar en ciertos momentos a la capacidad máxima del aeropuerto. Es por ello por lo que, es necesario buscar soluciones a este problema. No obstante, aunque la solución más fácil resulte ser la construcción de nuevos puestos de estacionamiento, calles de rodaje, e incluso pistas o terminales, estas obras llevan tiempo y, además, necesitan que el aeropuerto tenga el espacio suficiente para poder hacer estas mejoras. Por ello, se buscan soluciones a corto plazo que permitan optimizar los aeropuertos, de forma que no se alcance tan rápidamente a esa capacidad máxima, dando un margen mayor para poder realizar las obras o buscar alternativas ante este problema.

En el presente trabajo se expone una solución que, con la ayuda de la nueva tecnología relacionada a la inteligencia artificial, permita reducir el estrés de los controladores en aquellas situaciones en las que el tráfico aéreo es elevado y pueda causar problemas en el entorno aeroportuario. El sistema pretende realizar un control avanzado de aeronaves en tierra que, mediante la información de la posición del resto de aeronaves en tierra, pueda proponer rutas optimizadas para llegar desde el puesto de estacionamiento hasta la pista de despegue o

viceversa. Con ello, se podrá reducir los tiempos de taxi a lo largo del aeropuerto, mejorar la seguridad de las operaciones en tierra y reducir la contaminación en el entorno aeroportuario. Con este sistema, los beneficiados son varios, desde el aeropuerto y las mismas aerolíneas, hasta el usuario final, donde verá que su tiempo de viaje es reducido.

Este sistema pretende enfocarse basándose en diferentes conceptos, entre los que se encuentran: la seguridad en el entorno aeroportuario, medio ambiente, tiempos de taxi, controladores aéreos y reducción de los factores humanos.

1.1. Seguridad

El entorno aeroportuario, en especial el lado tierra, es un lugar altamente restringido, donde la seguridad prima por encima de cualquier otro aspecto. Aquí, las aeronaves y los distintos vehículos de handling están en continuo movimiento desplazándose por las distintas zonas del aeropuerto. Para ello, es necesario un control para que todo este tráfico sea ordenado y seguro, evitando todos aquellos riesgos. En consecuencia, las distintas normas de circulación ya sean en calles de rodaje o en plataforma, además de las propias instrucciones por parte de los controladores, son necesarias para que se lleve a cabo de manera correcta.

No obstante, aun intentando cumplir con todas las normas y teniendo en cuenta todas aquellas instrucciones, sigue habiendo numerosos incidentes a diario, tanto entre varias aeronaves o vehículos, como aeronaves circulando de manera errónea por culpa de los factores humanos. Ciertos datos indican que existen 600 incursiones de pistas al año, lo cual demuestra la necesidad de encontrar sistemas que mejoren y faciliten la circulación en tierra [1].

Un incidente reciente donde la seguridad se vio afectada por un incorrecto uso de las calles de rodaje en Madrid-Barajas fue el 22 de enero de 2022 donde una aeronave Embraer 190 de Air Europa Express, tomó la salida rápida K4 de la pista 32R, cuando el AIP indica que se debe tomar la K5. Debido a ello, se redujo de manera considerable la seguridad con la siguiente aeronave que pretendía aterrizar ya que estuvieron cerca de estar ambas en pista al mismo tiempo [2].

Lo anterior demuestra la necesidad de encontrar formas de controlar y vigilar la posición de las aeronaves en mayor medida, reduciendo las decisiones que pueden tener los pilotos o controladores que supongan un riesgo al resto de aeronaves. De este modo, herramientas como la que se presentará en futuros apartados será vital para traer efectos positivos en los aeropuertos con alto tráfico y calles de rodaje complejas.

1.2. Operativa en aeropuertos

Los aeropuertos con mayor número de operaciones son complejos entornos, donde las aeronaves recorren grandes distancias para poder alcanzar el punto del aeropuerto deseado. Debido a la necesidad de alojar una gran cantidad de aeronaves, los aeropuertos se dividen por terminales, las cuales pueden depender de aerolíneas o alianzas. Asimismo, según la procedencia o destino de la aeronave en cuestión, una pista puede ser utilizada con cierta preferencia. Todo ello implica que la aeronave esté obligada a recorrer distancias excesivas, que a su vez conforman un recorrido laberíntico mientras realizan el taxi.

Tomando como referencia los mayores aeropuertos mundiales, muchos de ellos alcanzan los 20 minutos de tiempo de taxi, por tanto, en aquellas operaciones de largo radio donde el avión despega y aterriza en aeropuertos de gran tamaño, esto puede suponer hasta 40 minutos más de viaje [3]. Esto resulta en una reducción de la satisfacción del pasajero al añadir más tiempo a su viaje, el gran consumo de combustible que esto supone, con sus respectivos efectos económicos y medio ambientales.

Debido a todo lo anterior, un control y un sistema bien optimizado para reducir estos tiempos implican un beneficio a tener en cuenta. Asimismo, en aquellos momentos de alta demanda de aeronaves, ayudará a poderla manejar del mejor modo posible.

1.3. Medio Ambiente

La industria aeronáutica tiene objetivos claros marcados para los próximos años en lo que respecta a las emisiones de dióxido de carbono y resto de gases de efecto invernadero. Estas políticas, provenientes de organismos públicos obliga a las distintas aerolíneas y aeropuertos a mantener el entorno libre de este tipo de gases. En caso de que esto no se cumpla y los niveles admisibles no sean alcanzados, las distintas compañías deberán compensar estas emisiones mediante multas económicas. Por ello, actualmente la industria aeronáutica está en búsqueda de todas aquellas medidas que permitan reducir de manera considerable los efectos. Es por ello que sistemas de reducción de tiempo de taxi son de gran utilidad para que, en una llegada o una salida, las emisiones sean reducidas en el entorno aeroportuario.

Como se menciona en el apartado anterior, los tiempos de taxi se pretenden reducir considerablemente. El motivo por el cual se consiguen dichos beneficios es gracias a la reducción de tiempos entre conflictos entre diversas aeronaves, y la conexión óptima y de menor tiempo hasta la posición deseada. Por ello, a la larga, aunque los beneficios por operación puedan verse como menores e insignificanticas, supone una gran mejora, especialmente, en aquellos aeropuertos de grandes dimensiones, donde los recorridos por las calles de rodaje son extremadamente altos.

Todo ello permite mejorar el entorno aeroportuario, añadiendo diversos beneficiados, entre los que se encuentran las compañías, el gestor aeroportuario o, incluso, los mismos residentes a zonas cercanas al aeropuerto.

1.4. Controladores aéreos

En aeropuertos de cierto tamaño, los aeropuertos están controlados por equipos conformados por numerosos controladores aéreos, encargados de dar todas aquellas autorizaciones e instrucciones para la marcha segura de aeronaves. Dependiendo del área, tarea o tipo de trabajo que realizan, los controladores pueden estar divididos en diversos grupos. De este modo, estos se especializan en una característica específica, facilitando así su trabajo y siendo más cómodo a la hora de trabajar.

Esta posición tiene una gran responsabilidad ya que de ellos depende que los aviones circulen con seguridad, tanto mientras la aeronave se encuentra en vuelo como rodando por el entorno aeroportuario. Por ello, el controlador debe estar con plenas capacidades para que la operación sea llevada a cabo con seguridad.

No obstante, en ocasiones, las capacidades pueden verse reducidas por situaciones inesperadas, ya sean meteorológicas o de alta demanda. Los momentos de estrés hacen que el controlador no actúe de la misma forma que manteniendo plena conciencia de todo lo que le rodea. Por ello, para realizar su trabajo, necesitan de herramientas que les faciliten su trabajo y así, poder tomar decisiones de manera más rápida y precisa.

1.5. Factor humano

Contemplando todo lo mencionado anteriormente, la complejidad que supone controlar las aeronaves en entornos aeroportuarios altamente difíciles es un gran reto. Estas complicaciones son tanto para los controladores como para los pilotos, por ello, intentar reducir los posibles riesgos al mínimo es esencial para que se mantenga la confianza en la industria.

De este modo, un aspecto muy importante para tener en cuenta es el factor humano presente en las operaciones, donde puede tener grandes efectos sobre la operación. Al fin y al cabo, la mayoría de los incidentes existentes en operaciones aeronáuticas son debidos a factores humanos u organizativos, donde, a partir de causas humanas, se toman decisiones incorrectas que acaban poniendo en riesgo las operaciones. Es por ello por lo que, evitar todo este tipo de situaciones es esencial para mantener el nivel de seguridad adecuado. En consecuencia, herramientas que sirvan de ayuda para reducir que el factor humano pueda ser un riesgo

sobre la operativa de aeronaves resulta muy beneficioso para la industria. Con ello, la confianza de los principales clientes y beneficiados es aumentada, dando una mayor reputación al aeropuerto en cuestión. Además, sirve de medida de seguridad para los controladores, teniendo un sistema más que les ayuda en sus tareas diarias, lo cual les da un mayor control en situaciones de alta demanda.

Capítulo 2. Objetivos

Partiendo de la situación actual, donde los aeropuertos cada vez cuentan con mayores números de operaciones, y son más las personas dispuestas a viajar en avión, se debe buscar una solución que facilite la operativa en aeropuertos. De este modo, en el presente trabajo se presenta una propuesta de sistema de gran utilidad para los controladores aéreos, permitiéndoles trabajar con mayor comodidad en todo lo que respecta a los movimientos en tierra de aeronaves. Esta herramienta está enfocada a los aeropuertos de gran tamaño, donde los tiempos de rodaje son elevados, habiendo numerosas rutas que permiten llegar de un punto a otro. Por tanto, poder optimizar, manteniendo los niveles de seguridad representa un gran beneficio para la industria, teniendo varios implicados los cuales disfrutarán de estas ventajas.

Por tanto, los objetivos del presente documento es plantear el correcto diseño del proyecto, el cual permita integrarlo en la operativa habitual de un aeropuerto. Para ello, se deben tener numerosos factores que pueden influir en las propiedades de la herramienta, además de que debe ser simple para los controladores, económicamente viable y fiable.

De la misma manera, para que la ejecución del proyecto se lleve de manera satisfactoria, se describe el alcance del proyecto, además de los distintos requisitos que debe cumplir la herramienta para cumplir con todas las normas de aviación civil. Por otro lado, será necesario analizar la situación actual a detalle, conociendo posibles competidores, pudiendo, así, desarrollar un plan estratégico, observando la viabilidad del proyecto.

Cabe destacar que, el presente documento también busca mostrar una primera versión de la herramienta a desarrollar, sirviendo de punto inicial, para que, a posteriori, se puedan implementar el resto de las tecnologías que ayudarían a la compleción del proyecto. Es importante señalar que el proyecto vendrá determinado por un cronograma, donde se presentarán los principales hitos del mismo, además de incluir todas aquellas tareas o trabajos a desarrollar para que el proyecto se ejecute cumpliendo el alcance y tiempo.

Por último, en este documento se pretende buscar todas aquellas causas de riesgos posibles que pueden darse durante la ejecución del proyecto, las cuales pueden tener graves efectos sobre los resultados de este. Además, se señalarán posibles alternativas, las cuales servirán en caso de que, por motivos externos, el proyecto deba cambiar su alcance o variar los objetivos inicialmente propuestos.

Todo lo anteriormente mencionado describe los objetivos de este documento, teniendo como objetivo común el mismo, siendo este que una herramienta, que optimice significativamente los movimientos en tierra, trae beneficios significativos. Por ello, un correcto y positivo empleo de nuevas tecnologías como es la inteligencia artificial y que con la propia retroalimentación de datos es posible crear un producto que traiga ventajas para numerosos interesados.

Capítulo 3. Oportunidad de negocio

En el presente apartado se exponen los distintos argumentos que demuestran la necesidad existente en la actualidad de desarrollar e incluir herramientas de gestión de tráfico aéreo en el entorno aeroportuario. Con el actual crecimiento de los aeropuertos y la saturación de estos, resulta esencial que herramientas como la que se expone a continuación se ponga en explotación lo antes posible. Por ello, en los próximos apartados se analizan los antecedentes que han ocasionado la situación actual y se presenta una posible idea de negocio para solucionarlo. Asimismo, también se muestran aquellos beneficios y beneficiados dados al emplear una herramienta como la que se pretende diseñar.

3.1 Antecedentes

La aviación a lo largo de los años ha sufrido numerosas mejoras debido al aumento del interés de las personas. Desde la creación del primer aeroplano en 1903 por los hermanos Wright, hasta el complejo sistema existente a día de hoy, donde volar en avión resulta esencial para numerosas personas, hace que la aviación haya cambiado en gran medida. Asimismo, con el aumento de la demanda de tráfico aéreo, el crecimiento de los aeropuertos y la correspondiente gestión del tráfico ha llevado a fuertes cambios que se adecuen a los niveles de seguridad y calidad acordes al servicio dado.

Es a partir de los años 40 cuando la aviación comercial verdaderamente empieza a crecer. Los avances tecnológicos debidos a la primera y la segunda guerra mundial posibilitan el nacimiento de un mercado que permite el transporte de personas alrededor del mundo en forma de negocio. Es por ello por lo que, los principales beneficiados se ven obligados a regular dicho mercado, ajustándose a medidas necesarias para que el transporte por aire fluya de manera organizada. De este modo, tras una conferencia donde se pretendía actualizar los distintos acuerdos existentes para aviación civil, diversos estados acaban firmando uno de los mayores acuerdos existentes hasta hoy en día, llamado Convenio sobre Aviación Civil Internacional o Convenio de Chicago. En él, se definen una serie de requisitos para la correcta operación de aeronaves. Entre los puntos más importantes, se destaca la primera parte, referida a la navegación aérea, donde se exponen los 42 primeros artículos, haciendo mención a todas aquellas normas en relación a las obligaciones por parte de los estados y las aerolíneas. Aquí se exponen desde los documentos que debe llevar la aeronave para un vuelo internacional hasta las acciones que puede realizar un estado a una aeronave.

Asimismo, cabe destacar que, para aquellas operaciones dentro de un mismo territorio, es el estado el encargado de decidir las normas aplicables, debido a que el Acuerdo de Chicago está orientado a aquellos vuelos internacionales. No obstante, muchos estados, hacen uso de este acuerdo internacional para generar su propia regulación, donde muchos de los puntos

coinciden, pudiendo simplificar o facilitar las operaciones. En el caso de España, la normativa vigente para los vuelos dentro del territorio español es el Reglamento del Aire, donde una gran cantidad de puntos son copiados o ajustados a partir del Convenio de Chicago.

Por otro lado, es importante señalar que el acuerdo internacional, incluye una serie de anexos referentes a distintos aspectos esenciales en el transcurso de las operaciones. Aquí, se incorporan normas, definiciones y prácticas recomendadas que están en constante actualización para que se puedan aplicar correctamente en el presente.

Con el aumento del tráfico mundial y las necesidades de cada momento, la OACI se ha visto obligada a incorporar nuevos anexos. Uno de ellos es el caso del Anexo 14 Volumen I, centrado en el diseño y operación de aeródromos, habiendo sido publicada su primera edición en 1990. Con respecto al anexo previamente mencionado, debido al aumento de las operaciones, este pretende incluir una serie de normas que obliga a los distintos aeropuertos a asemejarse, facilitando el trabajo a las distintas personas involucradas. Por ello, este anexo es esencial para que los aeropuertos puedan crecer de una manera organizada y correcta, cumpliendo con aquellos requisitos de seguridad, manteniendo un flujo organizado. Además, permite que los aeropuertos no se vuelvan en un caos operativo que pueda afectar gravemente a la continuidad y flujo de las operaciones.

No obstante, el crecimiento del tráfico ha hecho que los principales aeropuertos del mundo aumenten su tamaño, incrementen el número de pistas y tengan diversas terminales en función del trayecto y operación de la aeronave. De este modo, los aeropuertos, desde hace ya años, son infraestructuras complejas donde existe un continuo movimiento de vehículos, personas y aeronaves, pudiendo, a veces, llegar a ser estresante. Para que todo ello pueda funcionar correctamente, es necesario que haya personas dedicadas a la gestión de estos flujos, priorizando ciertos movimientos por encima de otros.

En el caso del lado aire, uno de los mayores problemas existentes es el complejo sistema de calles de rodaje, que permite llevar a aeronaves desde su puesto de estacionamiento a la pista o viceversa. Para ello, cumpliendo con los requisitos mostrados en el Anexo 14, se facilita dicho trabajo al piloto, sin embargo, conocer la ruta más corta puede ser tarea complicada. Es por ello por lo que, para gestionar estos flujos de la manera más organizada posible, el trabajo del controlador de tierra o superficie (GMC – Ground Movement Control o GND) es esencial. Este es el encargado de controlar el recorrido y autorizaciones oportunas para el correcto flujo de aviones o vehículos en el área de maniobras del aeropuerto, como pueden ser calles de rodadura o pistas inactivas. Asimismo, en aquellos aeropuertos aún más complicados existen equipos llamados Ground Movement Planners, dedicados al cálculo de trayectorias en tierra, consiguiendo así optimizar las rutas que van a realizar las aeronaves.

Es importante señalar que, el transporte aéreo sigue en pleno crecimiento, donde la demanda por viajar va en aumento, y por ello, los aeropuertos deben ajustarse a dicho incremento. Para conseguir resultados favorables, los aeropuertos se verán obligados a vivir modificaciones, obras y reconstrucciones, a parte de las relativas tareas de mantenimiento habituales. Este

tipo de trabajos son comunes en grandes aeropuertos, los cuales dificultan la planificación de las trayectorias de aeronaves por las calles de rodaje. Esto hace que los controladores encargados de este tipo de funciones sufran altos niveles de estrés, que junto a situaciones inesperadas y la saturación del propio aeropuerto, tengan que realizar un elevado esfuerzo para gestionar todo ello de la mejor manera posible.

Por otro lado, en lo que respecta al control de aeronaves en las distintas fases del vuelo, ya sea en ruta, aproximación o despegue, existen herramientas que facilitan el trabajo de los controladores. Este tipo de programas, consiguen predecir trayectorias y avisar sobre posibles conflictos entre varias aeronaves en el espacio aéreo. En cambio, en el área de maniobras del aeropuerto, todo este trabajo se realiza principalmente por los propios controladores. Esto junto a las políticas referentes a la reducción de la huella de carbono en el entorno aeroportuario, reducción de retrasos y planificación acorde a problemas en tiempo real hace que haya grandes necesidades en la mejora del control del transporte aéreo en el área de maniobras.

3.2 Situación actual

El creciente aumento del tráfico y la saturación de los aeropuertos han hecho que los distintos organismos tengan que innovar y buscar soluciones para reducir la complejidad de los propios aeropuertos. Para ello, se han desarrollado diversas herramientas y conceptos, siendo una de las más importante el A-SMGCS (Advanced Surface Movement, Guidance and Control System).

El A-SMGCS conceptualizado por EUROCONTROL, permite mejorar el tráfico de aeronaves en plataforma y calles de rodadura mientras mantiene los niveles de seguridad. No obstante, esto no es una herramienta en sí, al estar basado en diversos sistemas, y son diversas las empresas que comercializan herramientas como estas. Dentro de las distintas opciones, se destaca, InNOVA Routing and Guidance producto de INDRA [4] o SAAB con su propio A-SMGCS HMI [5].

Este tipo de sistemas tienen una aplicación similar a la que se pretende obtener con la propuesta de este trabajo. En este caso, son herramientas conformadas de manera modular, con los objetivos de aportar seguridad, orden y autorizaciones al flujo de vehículos, tanto para aeronaves como para vehículos de tierra en los aeropuertos. Estas herramientas, funcionan independientemente de la cantidad de tráfico, complejidad de las calles de rodaje, visibilidad e incluso línea de vista del controlador desde la torre de control. Esto genera una gran ventaja ya que reduce la carga de trabajo al controlador considerablemente y, además, le permite estar en pleno control de todo lo que está ocurriendo en el área de maniobras, aunque no lo pueda ver físicamente.

Estos tipos de sistemas deben cumplir con la normativa presente en el documento de OACI Doc 9830: Advanced Surface Movement Guidance and Control Systems (A-SMGCS) Manual, el cual presenta todo lo necesario para diseñar un sistema como este.

Entre las funcionas básicas que debe cumplir este tipo de sistemas son las de vigilancia, control, planificación/routing y guiado. Asimismo, se definen 4 niveles de A-SMGCS dependiendo del nivel que se quiere alcanzar [6]:

- A-SMGCS Nivel 1. En este nivel se orienta la herramienta a una mejora en la vigilancia y los procedimientos, consiguiendo así cubrir el área de maniobras completa, incluyendo los vehículos terrestres y los movimientos de las aeronaves. Asimismo, en este nivel, la información ofrecida debe mostrar la identificación de cada vehículo, así como el estado de las autorizaciones.

- A-SMGCS Nivel 2. En el nivel 2 se añade una mejoría en la seguridad, protegiendo pistas y áreas designadas, así como procedimientos asociados. De este modo, se generan señales de alerta para los controladores en caso de que exista algún conflicto entre los vehículos y aeronaves con respecto a las áreas restringidas.

- A-SMGCS Nivel 3. A diferencia del nivel 2, en el nivel 3 ya se empieza a tener control sobre los conflictos presentes entre distintas aeronaves y vehículos en el área de maniobras. Como solución, se integra un sistema mejorado de guiado y planificación para los controladores.

- A-SMGCS Nivel 4. En este último nivel, el sistema propone soluciones para cada conflicto, además de planificación y guiado automático, tanto para pilotos como para controladores.

Asimismo, en la actualidad, EUROCONTROL opta por una diferenciación funcional. Entre estas funcionalidades se encuentran el servicio de vigilancia, el servicio de soporte en materia de seguridad aeroportuaria, el servicia de ruta y el servicio de guiado. A continuación, se hace una breve explicación sobre cada uno de ellos:

- **Servicio de vigilancia:** provee al tráfico aeroportuario de conciencia situacional mediante un identificador, mostrando posición y control de las aeronaves y vehículos dentro de un área de servicio. Todo ello se representa mediante pantallas que ofrece interacción humana mediante un HMI (Human Machine Interface).

- **Servicio de soporte en seguridad aeroportuaria:** contribuye a las operaciones en el lado aire mediante una mejora en la seguridad, previniendo incidentes o posibles problemas debidos a errores humanos por parte de los controladores, pilotos o

conductores de vehículos terrestres. Este servicio depende directamente del servicio de vigilancia y se basa en tres funciones básicas que se explican a continuación:

- **Runway Monitoring and Conflict Alerting (RMCA):** hace referencia a una herramienta que alerta de conflictos en el corto plazo. Está especialmente orientada al área cercana a la pista y también detecta posibles conflictos entre aeronaves y vehículos. Para su correcto funcionamiento hace uso de los datos obtenidos por el servicio de vigilancia y reglas y parámetros predefinidos.
- **Conflicting ATC Clearances (CATC):** evita que existan ciertas autorizaciones que puedan generar problemas con previas autorizaciones. En ciertos casos puede ocurrir que debido a un error o, posibles conflictos que no se esperaban, se den autorizaciones que puedan tener efectos en la seguridad. Por ello, esta herramienta intenta alertar a los controladores sobre estos problemas.
- **Conformance Monitoring Alerts for Controllers (CMAC):** proporciona a los controladores con alertas acordes cuando el sistema A-SMGCS detecta la discrepancia entre procedimientos y autorizaciones con respecto a la pista, calles de rodaje o en plataforma.

- **Servicio de ruta:** genera una ruta para cada vehículo o aeronave teniendo en cuenta parámetros del aeropuerto o por inputs dados por el controlador. De este modo, se calculan los tiempos de taxi basados en la ruta, y compartidos con el A-CDM. Como requisito necesario, encontrado en el manual de OACI, es necesario que exista la posibilidad que el controlador pueda modificar o crear una nueva ruta. Además, debe aceptar otros tipos de maniobras, como los push-back. Este servicio es esencial para el correcto funcionamiento de otros servicios como el servicio de guiado o el de soporte en la seguridad aeroportuaria, especialmente aquellos relacionados con los servicios de alerta cuando existe una desviación o cambio no autorizado de la ruta por parte de los vehículos o aeronaves.
- **Servicio de guiado:** proporciona las siguientes funciones, con la ayuda de los inputs del controlador y los servicios de vigilancia y ruta:
  - Reajuste de las luces que alumbran la calle de rodaje (TCL- Taxiway Centerline Lights)
  - Control de las barras de parada.
  - La activación automática de los carteles de estacionamiento avanzados (A-VDGS- Advanced-Visual Docking Guidance Systems).

Entre los sistemas utilizados en los principales aeropuertos europeos, se destaca un programa de SESAR llamado Airport Safety Nets, con una duración de 3 años, y que finalizó a finales del 2019. En él se señalaba la utilización de las luces en pista para evitar incursiones indebidas, funcionando en el aeropuerto de París Charles de Gaulle. También, se destaca el empleo del

servicio de alerta a vehículos de tierra para avisar de entrar en zonas restringidas, o si existe algún conflicto con aeronaves. Esto último se empezará a introducir en otros aeropuertos europeos. Asimismo, según el programa SESAR, se están desarrollando las tareas para empezar a utilizar los servicios CATC y CMAC en los principales aeropuertos cumpliendo con los requisitos europeos. Además, estos dos servicios mencionados, también se están valorando para su empleo mediante cámaras de vigilancia, lo cual permita ser utilizados en aquellos aeropuertos de menor tráfico, pudiendo así funcionar con torres de control remotas [7].

Por otro lado, la situación actual parece señalar que pocos aeropuertos alcanzan los niveles 3 y 4 de los sistemas A-SMGCS. Como excepciones se encuentran principalmente el aeropuerto de Heathrow, que emplea el sistema InNOVA Ground de Indra para mejorar las operaciones en el área de maniobras, reduciendo así los riesgos y efectos [8]. Este sistema ayuda a dicho aeropuerto a operar a niveles altos de capacidad de manera continuada, lo cual demuestra un claro beneficio.

En cuanto a los aeropuertos de la región sudamericana, se observa mediante una nota de prensa sobre un seminario de la OACI realizado en 2021, que muchos aeropuertos contienen algún tipo de sistema A-SMGCS pero no muy avanzado. A parte de que la información existente acorde a cada aeropuerto es escasa, son pocos los aeropuertos que incluyen sistemas de alerta por incursión en pistas, lo cual refleja la alta posibilidad de mejora en estos aeropuertos [9].

En el entorno español, se destacan los aeropuertos de mayor tamaño, que son Madrid, Barcelona y Palma de Mallorca. En Barcelona y Palma, hasta el año 2021, no se había integrado un sistema A-SMGCS de nivel 2, mientras que Madrid sigue en proceso de integrar el nivel 2 entre sus herramientas:

Overall situation of Implementation Objectives

Main Objectives	Topic	Progress at the end of 2021	Status	2021	2022	2023	2024	2025	2026	>2026
AOM13.1	Harmonise Operational Air Traffic (OAT) and General Air Traffic (GAT) Handling	100%	Completed							
AOM19.4	Management of Predefined Airspace Configurations	8%	Ongoing		*					
AOM19.5	ASM and A-FUA	65%	Ongoing		*					
AOM21.2	Initial Free Route Airspace	57%	Ongoing		*					
AOM21.3	Enhanced Free Route Airspace Operations	6%	Ongoing					*		
AOP04.1(LEBL)	Advanced Surface Movement Guidance and Control System A-SMGCS Surveillance (former Level 1)	100%	Completed							
AOP04.1(LEMD)	Advanced Surface Movement Guidance and Control System A-SMGCS Surveillance (former Level 1)	100%	Completed							
AOP04.1(LEPA)	Advanced Surface Movement Guidance and Control System A-SMGCS Surveillance (former Level 1)	100%	Completed							
AOP04.2(LEBL)	Advanced Surface Movement Guidance and Control System (A-SMGCS) Runway Monitoring and Conflict Alerting (RMCA) (former Level 2)	100%	Completed					*		
AOP04.2(LEMD)	Advanced Surface Movement Guidance and Control System (A-SMGCS) Runway Monitoring and Conflict Alerting (RMCA) (former Level 2)	13%	Ongoing					*		
AOP04.2(LEPA)	Advanced Surface Movement Guidance and Control System (A-SMGCS) Runway Monitoring and Conflict Alerting (RMCA) (former Level 2)	100%	Completed					*		
AOP05(LEBL)	Airport Collaborative Decision Making (A-CDM)	100%	Completed							
AOP05(LEMD)	Airport Collaborative Decision Making (A-CDM)	100%	Completed							
AOP05(LEPA)	Airport Collaborative Decision Making (A-CDM)	100%	Completed							
AOP10(LEMD)	Time-Based Separation	0%	Not Applicable				*			
AOP11.1(LEBL)	Initial Airport Operations Plan	0%	Planned			*				
AOP11.1(LEMD)	Initial Airport Operations Plan	0%	Planned			*				
AOP11.1(LEPA)	Initial Airport Operations Plan	0%	Planned			*				
AOP11.2(LEBL)	Extended Airport Operations Plan	0%	Planned							2027

Figura 1: Situación de 2021 de los principales aeropuertos españoles [10]

Asimismo, se puede observar como la implementación de las herramientas de automatización siguen en planificación, aun estando en un estado muy preliminar:

Main Objectives	Topic	Progress at the end of 2021	Status	2021	2022	2023	2024	2025	2026	>2026
AOP11.2(LEMD)	Extended Airport Operations Plan	0%	Planned							2027
AOP11.2(LEMG)	Extended Airport Operations Plan	0%	Planned							2027
AOP11.2(LEPA)	Extended Airport Operations Plan	0%	Planned							2027
AOP12.1(LEBL)	Airport Safety Nets	15%	Ongoing					*		
AOP12.1(LEMD)	Airport Safety Nets	6%	Ongoing					*		
AOP12.1(LEPA)	Airport Safety Nets	15%	Ongoing					*		
AOP13(LEBL)	Automated Assistance to Controller for Surface Movement Planning and Routing	0%	Planned					*		
AOP13(LEMD)	Automated Assistance to Controller for Surface Movement Planning and Routing	0%	Planned					*		
AOP13(LEPA)	Automated Assistance to Controller for Surface Movement Planning and Routing	0%	Planned					*		
AOP14(LEMH)	Remote Tower Services	50%	Ongoing							2030
AOP14(LEVX)	Remote Tower Services	50%	Ongoing							2030
AOP15(LEMD)	Enhanced traffic situational awareness and airport safety nets for the vehicle drivers	50%	Ongoing							2030
AOP16	Guidance assistance through airfield ground lighting	0%	Not Applicable							2030
AOP17(GCFV)	Provision/integration of departure planning information to NMOC	100%	Completed							2030
AOP17(GCLP)	Provision/integration of departure planning information to NMOC	100%	Completed							2030
AOP17(GCRR)	Provision/integration of departure planning information to NMOC	100%	Completed							2030
AOP17(GCTS)	Provision/integration of departure planning information to NMOC	100%	Completed							2030
AOP17(GCXO)	Provision/integration of departure planning information to NMOC	100%	Completed							2030
AOP17(LEAL)	Provision/integration of departure planning information to NMOC	100%	Completed							2030
AOP17(LEBB)	Provision/integration of departure planning information to NMOC	100%	Completed							2030
AOP17(LEIB)	Provision/integration of departure planning	100%	Completed							2030

Figura 2: Situación de 2021 de los principales aeropuertos españoles (II) [10]

Todo ello hace que sea una oportunidad fundamental desarrollar una herramienta que pueda predecir las trayectorias de las aeronaves y, de este modo, proporcionar rutas al controlador dependiendo de los distintos requisitos que este quiera. Además, ajustarse a las necesidades de los controladores de cada aeropuerto resulta esencial para el éxito del proyecto.

3.3 Sistemas existentes

Como se menciona con anterioridad, este tipo de sistemas ya existen y están definidos según las funcionalidades que cada uno ofrece. Además, estos dependen del aeropuerto donde esté en funcionamiento, lo cual hará que las características que ofrezcan puedan variar un poco. No obstante, a continuación, se muestran los 4 sistemas principales existentes en el mercado y que tienen la tecnología más avanzada.

3.3.1 InNOVA AIR – INDRA

Esta herramienta es una de las soluciones que ofrece INDRA para evitar los problemas referentes a la alta concentración de tráfico en los entornos aeroportuarios. El objetivo que tienen con esta herramienta es mejorar la conciencia situacional y simplificar las decisiones que deben de tomar los pilotos y controladores. Todo ello lo implementa en una interfaz fácil de interactuar [4].

La mayor ventaja que tiene este producto con respecto a otras compañías es la posibilidad de integrarlo con otros productos ofrecidos por INDRA relacionados al control de tráfico aéreo. De este modo, incluyen en diversos paquetes, la posibilidad de incluir el control en tierra, el control en vuelo o el de torres remotas.

Lo que respecta a su herramienta A-SMGCS, se enfoca en proponer rutas de rodaje para aeronaves, vehículos en tierra o aeronaves remolcadas, partiendo de una serie de datos predefinidos, principalmente basándose en el plan de vuelo. Además, si se tiene una funcionalidad activa, se indicará si una aeronave se ha desviado de la trayectoria inicial, y en este caso, propondrá una nueva ruta. En el caso de haber un conflicto, el sistema propondrá que aeronave debe parar.

Por otro lado, este sistema hace uso del concepto “Follow the Greens” (FtG), encendiendo así las luces de centro de pista, indicándole al piloto que ruta debe tomar. Esto permite al piloto poder seguir el camino correcto independientemente de la visibilidad y las condiciones meteorológicas.

Los aeropuertos de Heathrow y de Budapest son los principales aeropuertos en utilizar este tipo de tecnología. Se destaca el aeropuerto de Budapest ya que no es un aeropuerto con un tráfico tan alto como Heathrow, no obstante, en este caso, se trata de una concesión donde se pretende operar dicho tráfico de manera remota. Para ello, han incluido esta herramienta de A-SMGCS, sin embargo, la principal utilidad es para poder controlarlo de manera remota, por ello, se entiende que gran parte de las ventajas ofrecidas por dicha herramienta no están siendo aprovechadas en este aeropuerto [11].

3.3.2 A-SMGCS HMI – SAAB

La interfaz HMI de SAAB, aparte de tener la interfaz en sí, contiene las herramientas que incluyen la vigilancia, el soporte de seguridad en el aeropuerto, diseño de rutas y guiado. De este modo, en la pantalla, se muestra aquella información referente a la posición de las aeronaves sobre un mapa del mismo aeropuerto, etiquetas de la información más importante para cada aeronave, además del estado del sistema. Asimismo, también se incluye la

información relativa a los aviones en cola para aterrizar y para despegar, teniendo toda la información a un simple click para el controlador [5].

Este sistema emplea diversos sensores, tantos propios de SAAB como externos para conocer la posición de las aeronaves, siendo estos los MLAT, SR-3 SMR y VL-1090.

Por otro lado, la función de diseño de rutas automático calcula la ruta según se encuentre la aeronave en posición de salida o llegada. Además, el sistema debe conocer una serie de parámetros del aeropuerto para que pueda calcular la ruta óptima rápidamente. Del mismo modo que INDRA; también contiene la función de "Follow the Greens" para indicar la dirección que tiene que llevar la aeronave. Cabe destacar que, este sistema incluye el control de las luces de las barras de parada, las cuales también se encenderán dependiendo de la situación de las aeronaves en cuestión.

En este caso, este sistema está presente especialmente en Estados Unidos, siendo el sistema utilizado por los principales aeropuertos del país. El aeropuerto internación General Mitchell de Milwaukee en Wisconsin fue el primero en utilizar este sistema y, por tanto, está aprobado por la FAA.

3.3.3 TowerPad A-SMGCS – FREQUENTIS

Frequentis, una empresa tecnológica austriaca, dedicada al control de tráfico aéreo especialmente, también ha desarrollado una herramienta de A-SMGCS que contiene funcionalidades realmente interesantes. Esto le ha hecho ser una de las herramientas utilizadas en algunos de los principales aeropuertos de Oriente Medio.

Tecnológicamente, esta herramienta ofrece soluciones de vigilancia, seguridad, control, planificación, diseño de rutas y guiado, mientras tiene en cuenta procedimientos del aeropuerto en cuestión y limitaciones de las infraestructuras. Además, la principal misión de este sistema es mejorar la seguridad, rendimiento, capacidad y necesidades medioambientales. Asimismo, también destaca que realizan el mantenimiento y ajustes propuestos por el cliente. Todo ello les hace alcanzar una mejoría en el rendimiento de un 10% [12].

No obstante, cabe destacar que esta herramienta está diseñada para operar en aquellos momentos de tráfico es algo más bajo, necesitando el trabajo de los controladores para aquellos momentos de mayor flujo y estrés. Por tanto, el beneficio presente en esta herramienta es algo menor ya que no consigue optimizar todas las operaciones diarias, viéndose limitado a ciertas horas.

Por otro lado, también incluye funcionalidades como "Follow the Greens", autorizaciones electrónicas o la iluminación de los paneles en las calles de rodaje. En cuanto a la información

obtenida de la posición de las aeronaves, el sistema hace uso de diversos métodos, como son: MLAT y ADS-B, radar de superficie, cámaras y el "Multisensor Data Fusion" (MSDF), donde a partir de datos procedentes de distintos sensores, es capaz de reconocer la posición de la aeronave [13].

3.3.4 AiPRON Manager – ADB SAFEGATE

Esta herramienta desarrollada por la empresa ADB Safegate pretende alcanzar el nivel 4 como sistema A-SMGCS, pudiendo resolver conflictos y realizando una planificación y guiado automatizado, independientemente de las condiciones meteorológicas. Asimismo, al igual que en los casos anteriores, este sistema incluye la utilidad de "Follow the Greens", y busca que el consumo de energía del aeropuerto se reduzca pudiendo apagar estas luces en caso de no ser necesitadas [14].

Es importante señalar que esta herramienta se desarrolló específicamente para los aeropuertos de Abu Dhabi y, para ello, la implementación se ha llevado a cabo en 5 fases. En primer lugar, se hizo una auditoría o estudio de la situación presente en dichos aeropuertos. Con ello, se conseguía justificar si la herramienta era necesaria o no. Luego, se diseñaron y mejoraron las luces en las áreas de maniobras, además de las mismas infraestructuras. Por último, se integró el sistema para el control y monitoreo de aeronaves, incluyendo también sistemas avanzados en torre, los cuales les permite tener movimiento en tierra automatizado.

Cabe destacar que, esta herramienta potencia, sobre todo, las operaciones en plataforma, diseñada en un principio para que los movimientos en plataforma sean más ágiles, controlando los tiempos de fuera de calzos de manera muy precisa. Asimismo, el sistema incluye el control e indicación automatizada de los carteles de los puestos de estacionamiento, que permite a los pilotos saber con facilidad hacia qué posición se deben dirigir.

3.3.5 Comparativa

Habiendo analizado las distintas funcionalidades de los distintos sistemas presentes en el mercado, se observa que, aunque todos tienen unos objetivos comunes, las funcionalidades pueden variar. A modo de resumen, se presenta la siguiente tabla que muestra los diversos outputs que ofrece cada sistema, sirviendo de análisis para alcanzar una versión mejorada de todos ellos.

Tabla 1: Comparativa de funcionalidades de sistemas presentes en el mercado

Funcionalidades		InNOVA AIR (INDRA)	A-SMGCS HMI (SAAB)	TowerPad A-SMGCS (FREQUENTIS)	AiPRON Manager (ADB SAFEGATE)
Nivel 1	Mejora en la vigilancia	✓	✓	✓	✓
	Mejora en los procedimientos	✓	✓	✓	✓
	Cobertura del área de maniobras completa	✓	✓	✓	✓
	Movimientos de vehículos terrestres	✗	✓	✓	✓
	Movimientos de aeronaves	✓	✓	✓	✓
	Identificación de cada vehículo	✗	✓	✓	✓
	Estado de las autorizaciones	✗	✓	✓	✓
Nivel 2	Mejora en la seguridad	✓	✓	✓	✓
	Protección de pistas	✓	✓	✓	✓
	Protección de áreas designadas	✓	✓	✓	✗
	Protección de procedimientos asociados	✓	✓	✓	✓
	Señales de alerta para los controladores con respecto a áreas restringidas	✓	✓	✓	✓
Nivel 3	Control sobre los conflictos entre aeronaves	✓	✓	✓	✓
	Control sobre los conflictos entre vehículos	✗	✓	✓	✗
	Control sobre los conflictos entre vehículos y aeronaves	✗	✓	✓	✗
	Sistema mejorado de guiado	✓	✓	✓	✓
	Sistema mejorado de planificación	✓	✗	✓	✓
Nivel 4	Propuesta de soluciones por conflicto	✓	✗	✗	✓
	Planificación automática	✗	✗	✗	✓

	Guiado automático	✓	✓	✓	✓
Servicio de vigilancia		✓	✓	✓	✓
Servicio de soporte en seguridad aeroportuaria	Runway Monitoring and Conflict Alerting	✓	✓	✓	✓
	Conflict ATC Clearances	✗	✓	✓	✗
	Conformance Monitoring Alerts for Controllers	✓	✓	✓	✓
Servicio de ruta		✗	✗	✓	✓
Servicio de guiado	Taxiway Centerline Lights	✓	✓	✓	✓
	Barras de parada	✗	✓	✗	✗
	Advanced Visual Docking Guidance Systems	✓	✗	✗	✓

Como se puede observar, los cuatro sistemas cumplen con la mayoría de las funcionalidades principales, alcanzando prácticamente la totalidad de los niveles 1 y 2, no obstante, al alcanzar el nivel 3 y 4, no llegan a alcanzar todas las funcionalidades completas. Es importante señalar que, ciertos sistemas no tienen en cuenta el tráfico de vehículos por la plataforma aparte de las propias aeronaves. También se observa, que únicamente 2 sistemas tienen en cuenta la propuesta de nuevas rutas en caso de conflictos, lo cual resulta esencial para la disminución de la carga de trabajo al controlador.

Por otro lado, el único sistema de realizar planificación automática es el de ADB Safegate, el cual se basa de información de otros aeropuertos para conocer de antemano cual será la ruta a realizar. Esto se considera esencial para mantener el control del aeropuerto en la etapa estratégica y pretáctica. Asimismo, esto resulta positivo ya que permite ajustar y corregir posibles problemas o conflictos con anterioridad, lo cual sirve al sistema para conocer dichos datos de antemano y así, poder reaccionar de manera más rápida.

En lo que respecta al servicio de soporte en seguridad aeroportuaria, se destaca que, aunque dos sistemas sepan resolver conflictos proponiendo nuevas rutas, estos no revisan las autorizaciones previas, las cuales pueden entrar en conflicto con las siguientes autorizaciones. Esto puede causar posibles conflictos futuros y disminuir la seguridad del sistema, por ello, se destaca que esta cualidad es esencial para el correcto control de aeronaves y vehículos.

Por último, el sistema de guiado se centra en la herramienta llamada “Follow the Greens”, la cual todos los sistemas incluyen, sin embargo, otras funcionalidades que resultan interesantes son la iluminación de carteles, tanto a lo largo de las calles de rodaje, como en los puestos de estacionamiento, y las barras de parada, las cuales pueden estar automatizadas con las autorizaciones y tener en cuenta el resto del tráfico.

Capítulo 4. Propuesta

Habiendo definido la situación actual, se observa que existe un problema en aquellos aeropuertos de gran tamaño, donde las calles de rodadura tienen una estructura compleja con diversas posibilidades para realizar diferentes rutas. Además, tras analizar el estado actual de la tecnología presente en los aeropuertos se propone como idea de negocio, el desarrollo y puesta en funcionamiento de una herramienta que permita optimizar el tránsito de aeronaves en el área de maniobras. De este modo, se pretende reducir considerablemente los tiempos de taxi en el aeropuerto, aportando una serie de beneficios que se indicarán más adelante.

Cabe destacar que esta herramienta pretende introducir un sistema innovador y que le da un valor añadido, desmarcándose del resto de competidores, donde mediante inteligencia artificial y la utilización de bases de datos obtenidas durante la operativa del aeropuerto, permite realimentar el sistema para que, cada vez, este esté más optimizado. Esto ayuda a los controladores aéreos de superficie, pudiendo coordinar de manera más fácil el flujo de aeronaves, conociendo también de forma más precisa los tiempos de taxi para evaluar las mejores opciones.

Además, habiendo analizado la herramienta del resto de competidores, se diseña una nueva que incluye todos aquellos puntos que no se han considerado, haciéndola ser una más completa y superior. Para ello, se incluirá todos los vehículos presentes en el área de maniobras, se integrará con el Network Manager Operations Centre (NMOC) y A-CDM, para así poder planificar las rutas que van a realizar las aeronaves con anterioridad, y que en caso de que ocurra algo inesperado, esta pueda actuar de manera más rápida. Asimismo, se incluirá un sistema de guiado de mejores características, controlando tanto la señalética como las barras de parada, siendo así más cómodo para el piloto, y disminuyendo posibles riesgos.

Por otro lado, la herramienta considera el aumento de tráfico aéreo para los próximos años, existiendo numerosos planes por parte de los aeropuertos para el aumento y mejora de sus infraestructuras. De este modo, se prevén numerosos cierres y modificaciones en el área de maniobra que obligue a la modificación de las rutas utilizadas normalmente para llevar a los aviones desde el puesto de estacionamiento hasta la pista. Esto implica que serán necesarios que se analicen cuáles de las distintas posibles rutas en cada aeropuerto pueda ser la óptima dependiendo del tipo de operación que se esté llevando a cabo y el número de aviones presentes en las pistas de rodaje. Esto significa que la herramienta que se pretende diseñar podrá incluir aquellas calles de rodaje cerradas por obras, y el tipo de operación (en Madrid, por ejemplo, distinguirá entre configuración Norte, Sur, diurna o nocturna), estimando la mejor ruta para cada operación.

Cabe destacar que esta herramienta, al tener en cuenta el tráfico real en cada momento, este también contendrá la información de aquellos imprevistos presentes que puedan cerrar ciertas calles de rodaje por diversos motivos durante momentos singulares. Esto consigue que el

recálculo de rutas sea más rápido y óptimo, facilitando el trabajo de los controladores, reduciendo también aquellos momentos de estrés.

Teniendo todo lo anterior en consideración, la herramienta se pretende integrar dentro de las prácticas habituales utilizadas en las torres y centros de control. Esto permitirá que los controladores de superficie o los encargados de la gestión de la planificación del movimiento en superficie puedan reducir su carga de trabajo, manteniendo los niveles óptimos de seguridad y calidad.

4.1 Beneficios esperados

Esta herramienta será de gran utilidad para numerosos interesados, trayendo una gran cantidad de beneficios. Entre los principales beneficios se encuentran:

- **Tiempo de taxi:** uno de los principales objetivos de la herramienta es reducir el tiempo que lleva a las aeronaves a realizar el taxi. De este modo, optimizar las operaciones en los aeropuertos.

- **Control de imprevistos:** al tener información en tiempo real, permite conocer y controlar aquellas variables que pueden tener un efecto en la operativa dentro del entorno aeroportuario.

- **Retrasos de aeronaves:** mediante la posibilidad de priorización de aeronaves según considere el controlador, este puede tener en cuenta aquellas aeronaves con retraso, favoreciéndolas y consiguiendo reducirlo.

- **Dependencia del controlador:** en la actualidad, el controlador tiene plena decisión sobre las distintas rutas en taxi. Esto, en ocasiones, puede generar situaciones donde el controlador se vea desbordado, causando momentos de estrés. Esto puede reducir la capacidad del controlador y causar errores no deseados.

- **Capacidad aeroportuaria:** se ve mejorada ya que se reduce la saturación en

el entorno de las calles de rodaje, lo que optimiza las operaciones. Asimismo, se le permite dar prioridad a aquellas aeronaves que por cierto motivo puedan estar causando problemas.

- **Emisiones:** siendo uno de los objetivos principales por parte de la industria aeronáutica, la herramienta permitirá reducir el tiempo de taxi, lo cual reducirá considerablemente las emisiones durante esta fase.

- **Conflictos:** la herramienta conocerá en todo momento la posición de las aeronaves presentes en el aeropuerto. De este modo, podrá conocer aquellos posibles conflictos y notificárselos al controlador con tiempo suficiente. Asimismo, la herramienta será capaz de proponer una nueva ruta, o dejar al controlador elegir la que considere.

Todo lo anterior hará que haya una serie de beneficiados al utilizar la herramienta. Entre ellos, se incluyen los siguientes:

- **Aeropuertos:** podrá aumentar el número de operaciones al estar optimizando las operaciones en el área de maniobras. Esto traerá beneficios económicos al aumentar el número de vuelos y, por tanto, el número de pasajeros. Por otro lado, podrán conocer con anterioridad el efecto de unas obras sobre las calles de rodaje, pudiendo adecuar dichas obras a las necesidades aeroportuarias.

- **Controladores aéreos:** al ser una herramienta que reducirá la cantidad de trabajo que deben de realizar, las tareas repetitivas se ven reducidas. Asimismo, aquellas ocasiones donde la carga de trabajo sea muy elevada, se le proponen soluciones óptimas en cada caso, que el controlador únicamente debe decidir si está de acuerdo. De este modo, el estrés generado se ve

reducido.

- **Aerolíneas:** verán una mejora operativa en aquellos aeropuertos saturados, lo cual permitirá reducir su tiempo de operación, posibilitando mayor cantidad de operaciones, o mayores márgenes en caso de retrasos. Asimismo, conseguirán una mayor satisfacción de sus clientes, lo cual puede generar una ventaja competitiva.

- **Pasajeros:** verán una reducción en la duración de su vuelo, disminuyendo el rechazo hacia ciertos aeropuertos, donde los retrasos pueden ser constantes en ciertas épocas del año.

- **A-CDM:** el poder integrar esta herramienta con el A-CDM (Airport Collaborative Decission Making) trae una gran ventaja hacia los usuarios de A-CDM. Este sistema donde participan IATA y EUROCONTROL, entre otros, optimiza la utilización de los recursos existentes en los aeropuertos, con el objetivo de mejorar el tráfico en el entorno aeroportuario. Para ello, se intenta que los operadores de aeropuertos, aerolíneas, empresas de handling y Network Manager trabajen de manera más transparente, intercambiando información importante sobre previsión de tiempos [15]. Compartiendo esta información entre aeropuertos permite organizarse también de mejor forma, consiguiendo así optimizar la operación teniendo en cuenta el estado del tráfico procedente de otros aeropuertos.

  De este modo, integrando la herramienta presentada en este trabajo con el A-CDM, se conseguiría una información aún más precisa, consiguiendo así unos mayores beneficios. Es importante señalar que, la herramienta expuesta, al trabajar en tiempo real, puede comunicar al A-CDM constantemente, teniendo todos los participantes informados rápidamente.

Capítulo 5. Definición del proyecto

En este apartado se presenta la definición del proyecto que se pretende llevar a cabo para reducir aquellos riesgos e inconvenientes existentes en la actualidad. A día de hoy, muchos son los aeropuertos que sufren retrasos importantes debidos al excesivo tráfico que opera en ellos. De este modo, organizar todo el tráfico de manera correcta puede ser complicado, lo cual herramientas como la presentada en este trabajo resulta en potentes beneficios tanto para los propios aeropuertos, como para el mismo viajero. Consecuentemente, en este apartado se muestra una descripción del proyecto que se pretende llevar a cabo, destacando el alcance que tiene y la localización del mismo. Asimismo, se analizan los interesados del proyecto expuesto. Por último, se declaran aquellos requisitos esenciales para que el proyecto se lleve a cabo de manera satisfactoria y un diseño conceptual.

5.1 Descripción del proyecto

Como se menciona con anterioridad, los aeropuertos son entornos en continua evolución, donde las operaciones son constantes y no pueden paralizarse. Por ello, con el presente crecimiento de la demanda de tráfico aéreo, la necesidad de construir aeropuertos de mayor tamaño ha sido un factor importante para alcanzar la situación actual, donde los flujos de tráfico en el entorno aeroportuario resultan complejos y estresantes.

De este modo, resulta obligatorio el desarrollo de herramientas para ayudar a los controladores de tierra a realizar sus tareas de organización y así, permitan dar autorizaciones manteniendo los niveles de seguridad acordes al sector. Es por este motivo por el que el proyecto presentado pretende desarrollar una herramienta de tipo A-SMGCS (Advanced Surface Movement, Guidance and Control System) para mejorar y ayudar al trabajo de los controladores y, además, trayendo diversos beneficios al aeropuerto en cuestión.

En este caso, se pretende desarrollar una herramienta de nivel 4, donde se propongan soluciones para cada conflicto, y al mismo tiempo, permita planificar con anterioridad y ofrezca guiado automático. Para ello, haciendo uso de los datos obtenidos del propio aeropuerto, información radar y del propio A-CDM, se consigue conocer el estado de ese momento del aeropuerto, y mediante cálculo de trayectorias en superficie e inteligencia artificial, la herramienta permite calcular cual es la trayectoria a seguir por la aeronave.

Esta herramienta pretende distinguir aquellos movimientos de despegue o aterrizaje, calculando así la trayectoria por las calles de rodaje óptima tanto en tiempo como en combustible. Del mismo modo, podrá reconocer aquellos posibles riesgos, interferencias o incumplimientos de autorizaciones, proporcionando un extra de seguridad, con alertas y avisos al controlador.

Como valor diferencial frente a otros competidores presentes en el mercado, esta herramienta pretende incluir el tipo de aeronave y la aerolínea que la está operando. Esto es debido a que cada aerolínea tiene diversas políticas con respecto a la velocidad y forma de realizar el taxi. En ocasiones, ciertas compañías pueden obligar a sus pilotos a utilizar únicamente un solo motor para realizar el taxi, lo cual puede causar un taxi más lento que si estuviese ocurriendo con los dos motores. Teniendo todo lo anterior en cuenta, la herramienta consigue calcular de manera más precisa y exacta el tiempo que le debe llevar a la aeronave hasta alcanzar su puesto de estacionamiento o el comienzo de pista. Así, además, se consiguen obtener posibles conflictos de manera más precisa debido a cruces con otras aeronaves.

Por otro lado, se pretende que este cálculo de nuevas rutas se realice en tiempo real, debido a que, si existe alguna modificación de la ruta por algún motivo externo, el sistema o herramienta pueda recalcular la mejor ruta nuevamente de manera rápida. Cabe destacar que esto presenta una ventaja considerable cuando se trata de un aeropuerto con diferentes salidas rápidas de pista, pudiendo obtener la mejor ruta dependiendo de qué salida tome la aeronave.

Cabe destacar que, conociendo el tipo de aeronave que pretende aterrizar y la aerolínea, el sistema hará un primer cálculo de ruta considerando la salida rápida más probable. Asimismo, a la hora de aeronaves en despegue, tendrá en consideración las estelas de las aeronaves para que el tiempo en caso de mucho tráfico sea el menor posible. También, el sistema conocerá aquellas aeronaves que se encuentran en retraso. Con todos estos datos se pretende que el sistema sepa interpretar los datos y declarar cual tiene una mayor prioridad frente a las otras aeronaves.

Todo ello, integrado con los demás sistemas presentes en el aeropuerto, es decir, la iluminación de las calles de rodaje, la señalética y los carteles de los puestos de estacionamiento (A-VDGS- Advanced-Visual Docking Guidance Systems), cree un sistema mucho más automatizado y fácil de operar tanto para el controlador como para el piloto.

Por último, es importante señalar que esta herramienta debe ser robusto y de funcionamiento continuo, lo cual implica que se deben realizar las pruebas adecuadas que corroboren que el sistema es adecuado y se pueda poner en funcionamiento de forma segura y organizada. Esto es esencial ya que el entorno aeroportuario es un sistema que puede ser peligroso si no está vigilado y controlado correctamente y que, además, mueve grandes cantidades de dinero, lo cual implica que no puede causar errores ya que tendría elevadas consecuencias.

5.2 Localización del proyecto

Habiendo analizado la situación presente de los principales aeropuertos españoles, se observa que estos se encuentran desactualizados en cuanto a este tipo de tecnología. Es por ello por lo que, se ha decidido iniciar este programa en el Aeropuerto Adolfo Suárez Madrid Barajas,

preparando dicha herramienta para este aeropuerto, y pudiendo integrarla en el resto de los aeropuertos grandes de España o incluso mercados internacionales.

Se ha considerado el aeropuerto de Madrid-Barajas como opción inicial ya que presenta diversas condiciones que hace que una herramienta como esta pueda generar grandes beneficios. En primer lugar, y como ya se mencionaba con anterioridad, en este aeropuerto aún no se ha implementado el nivel 2 del A-SMGCS, lo cual implica que la herramienta actual solo presenta las autorizaciones ya dadas por los controladores. De este modo, todo el trabajo se realiza de manera manual, lo cual implica un elevado esfuerzo por parte de los controladores, que, en ocasiones, puede generar elevadas cargas de trabajo. Asimismo, se destaca que no existe un sistema de alertas que señale acciones indebidas por las aeronaves o demás vehículos en las calles de rodaje. Es por ello por lo que, se considera que este aeropuerto necesita una actualización prioritaria en su sistema de gestión de tráfico en superficie.

Cabe destacar que su infraestructura, formado por 4 pistas y 4 edificios terminales más uno satélite, favorece la necesidad de buscar soluciones a mejoras en la navegación por superficie. Tener un lado aire tan grande demuestra que puede ser complejo de seguir y controlar, lo cual es otra ventaja más que demuestra que el aeropuerto de Madrid es una gran opción.

Por otro lado, se señala que el aeropuerto es el más importante de España, siendo el primero por tráfico de pasajeros, carga aérea y operaciones. Además, es considerado el quinto de Europa por número de pasajeros y el vigésimo cuarto del mundo [16].

5.2.1 Terminales del aeropuerto

Como ya se adelantaba antes, el aeropuerto de Madrid Barajas tiene 4 pistas, donde 2 están destinadas a las salidas y las otras 2 para las llegadas. Del mismo modo, el aeropuerto está dividido también en 4 terminales con diversas zonas, estando las terminales T1, T2 y T3 unidas por 6 zonas de embarque, mientras que la T4 tiene una zona satélite llamada T4S, la cual se une mediante un tren automático subterráneo. Las terminales están divididas por alianzas, operando ciertas aerolíneas en algunas terminales específicas y, a su vez, quedan divididas según el tipo de operación ya sea Schengen, UE no Schengen y terceros países [16].

En las terminales T4 y T4S, se destacan las operaciones de la alianza OneWorld, siendo el principal usuario Iberia. En cuanto a la terminal T4, los vuelos son principalmente nacionales e internacionales tipo Schengen, sin embargo, en la terminal satélite, aquí ya se operan la mayor parte de los vuelos internacionales no Schengen.

Por otro lado, las terminales T1, T2 y T3 tienen una mezcla de tipos de vuelos, sin embargo, los internacionales no Schengen se encuentran principalmente en el Dique Sur, siendo un edificio

anexo a la Terminal 1, el cual tiene sus correspondientes controles de pasaporte para asegurar el correcto control de pasajeros.

Es importante señalar que, aunque no se mencione con anterioridad, existe además dos zonas terminales añadidas con una misión de carga y mantenimiento situadas al sur del aeropuerto, una junto al dique Sur, y la otra entra las pistas 32R y 32L.

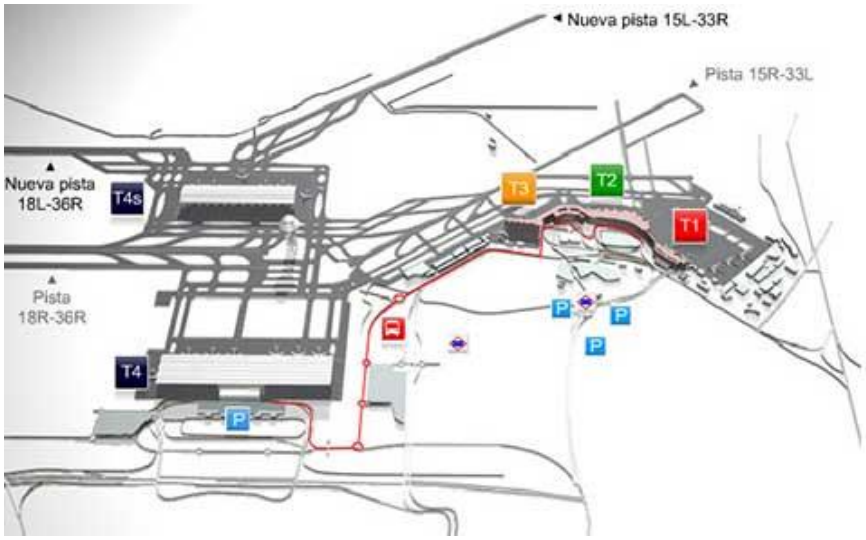


Figura 3: Disposición del aeropuerto de Madrid-Barajas

5.2.2 Configuraciones de pistas

Este aeropuerto tiene las siguientes configuraciones principales, definidas en el AIP [17]:

- Entre las 07:00 y las 23:00 LT (Local Time):
  - Preferente: Configuración Norte
    - Llegadas: 32L/32R
    - Salidas: 36L/36R
  - No preferente: Configuración Sur
    - Llegadas: 18L/18R
    - Salidas: 14L/14R
- Entre las 23:00 y las 07:00 LT:
  - Preferente: Configuración Norte
    - Llegadas: 32R
    - Salidas: 36L
  - No preferente: Configuración Sur
    - Llegadas: 18L
    - Salidas: 14L

Teniendo lo anterior en cuenta, el camino que recorrerá la aeronave variará según si es salida o llegada y la configuración del propio aeropuerto. Para ello, en el AIP también se definen cuáles son las rutas que deben seguir en cada caso.

Asimismo, es importante mencionar que la utilización de ciertas pistas vendrá definida por la procedencia o destino que tiene la aeronave en cuestión. Esto significa que, en el caso de una aeronave de procedencia del Este, esta tomará las pistas del Este para el aterrizaje, en este caso 18L en configuración Sur y 32R en configuración Norte (las configuraciones están descritas en el siguiente apartado). Este criterio geográfico queda definido en el AIP, en los datos del aeropuerto, y el encargado de controlar estos flujos son Madrid ACC y Madrid TMA. Solo en casos excepcionales de capacidad de ciertas pistas, se podrá no utilizar dicho criterio.

Por tanto, siguiendo el criterio anterior y atendiendo a las terminales que debe usar cada aerolínea, existen ciertas compañías que tienen que realizar largas distancias de taxi. Un claro ejemplo es Korean Airlines, la cual, al venir o salir hacia el Este, emplearía la pista 32R para aterrizar y 18R para despegar. Sin embargo, su puesto de estacionamiento se sitúa en el Dique Sur, en la terminal 1, por tanto, tiene que recorrer de un punto del aeropuerto a otro, pasando por numerosos Hot Spots en mitad y añadiendo un tiempo considerable a su operación.

5.2.3 Torres de control

El aeropuerto de Madrid cuenta con tres torres de control, teniendo cada una de ellas el control de ciertas pistas y funciones, siendo las tres operadas por ENAIRE. Estas son [18]:

- **Torre Norte:** Esta se sitúa junto a la terminal T4 Satélite, y se trata de la principal torre de control del aeropuerto, realizando la mayor parte de las tareas de control de tráfico.

- **Torre Oeste:** Esta se encuentra emplazada junto a la terminal T4 y es la encargada de gestionar el movimiento en tierra de aeronaves en las zonas cercanas a la T4.

- **Torre Sur**: Esta torre es la antigua torre de control del aeropuerto y se ubica en la terminal T2. En ella se gestiona el control sobre los movimientos en tierra de las aeronaves por los entornos cercanos a las terminal T1, T2 y T3, así como las áreas de carga.

Esto resulta esencial conocer ya que permite ver como se encuentra el aeropuerto dividido según zonas de control y, por ello, adecuar la herramienta a las necesidades de los controladores. Como simplicidad, una primera aproximación puede ser la realización de dicha herramienta enfocada a una zona reducida para comprobar el comportamiento positivo que trae.

5.2.4 Operaciones

Las operaciones a lo largo de los últimos años han ido en aumento constante, hasta el año donde se paralizo el tráfico aéreo debido al COVID. Sin embargo, después de este, las operaciones han vuelto a aumentar alcanzando los niveles anteriores. Además, se espera que este tráfico siga en aumento, superando los valores preCOVID.

A continuación, se muestra un gráfico con la cantidad de pasajeros por año de los últimos 22 años:

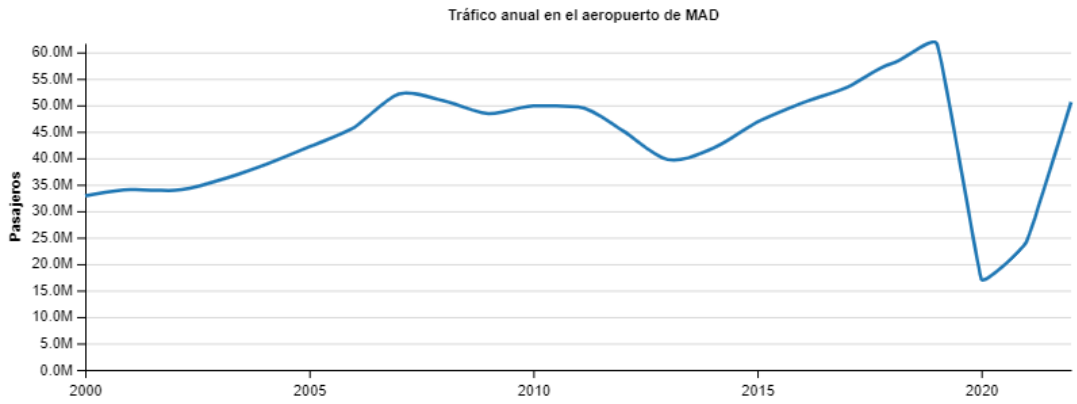


Figura 4: Trafico anual en el aeropuerto de Madrid [18]

Cabe destacar que, dentro de los 50 millones de pasajeros aproximadamente, el 72% representa pasajeros en trayectos internacionales, mientras que el 28% restante son nacionales. Dentro de las operaciones internacionales, destacan Lisboa, París-Orly, Roma, Londres-Heathrow y Bogotá como las rutas con más pasajeros:

Tabla 2: Número de pasajeros por destino (año 2022) [18]

Puesto	Ciudad	Pasajeros
1	Lisboa (Portugal)	1.538.930
2	París-Orly (Francia)	1.372.064

3	Italia Roma (Italia)	1.291.377
4	Londres-Heathrow (Reino Unido)	1.184.916
5	Bogotá (Colombia)	1.095.936
6	Ámsterdam (Países Bajos)	980.722
7	México (México)	907.328
8	Bruselas (Bélgica)	887.208
9	París-Charles de Gaulle (Francia)	871.273
10	Fráncfort del Meno (Alemania)	794.852
11	Múnich (Alemania)	782.962
12	Oporto (Portugal)	779.479
13	Buenos Aires (Argentina)	760.528
14	Nueva York-JFK (Estados Unidos)	748.423
15	Miami (Estados Unidos)	689.639
16	Lima (Perú)	689.453
17	Londres-Gatwick (Reino Unido)	636.045
18	São Paulo-Guarulhos (Brasil)	628.327
19	Zúrich (Suiza)	584.046
20	Ginebra (Suiza)	524.183
21	Santo Domingo-Las Américas (República Dominicana)	513.425
22	Milán-Malpensa (Italia)	481.447
23	Dublín (Irlanda)	474.401
24	La Habana-Jose Martí (Cuba)	452.447
25	Berlín (Alemania)	431.133
26	Estambul- New Airport (Turquía)	417.132
27	Santiago de Chile (Chile)	417.004
28	Viena (Austria)	400.522
29	Tel Aviv (Israel)	399.047
30	Venecia-Marco Polo (Italia)	397.073
31	Doha-Hamad (Catar)	387.082
32	Londres-Stansted (Reino Unido)	370.797
33	Bucarest (Rumanía)	344.255
34	Cancún (México)	338.810
35	Dubái (Emiratos Árabes)	320.928

De este modo, se observa que dentro de los primeros 35 destinos internacionales con mayor cantidad de pasajeros, se encuentran 15 destinos intercontinentales, lo cual implica aeronaves de gran tamaño. No obstante, considerando los vuelos nacionales, la mayor parte de las aeronaves que operan en dicho aeropuerto son las medianas, típicamente de las familias Airbus A320 y Boeing 737.

Estos datos pueden resultar necesarios a la hora de obtener conclusiones para los primeros análisis de la herramienta. Además, servirá como referencia para el periodo de prueba, antes de ponerse en funcionamiento.

5.2.5 Rutas predefinidas

A continuación, se describen las rutas predefinidas para las operaciones de llegadas y salidas de pista, y su posterior transcurso por las calles de rodaje. Cabe destacar que las rutas a seguir en plataforma no quedan definidas en este apartado ya que la cantidad de ellas es muy elevada, sin embargo, estas se pueden encontrar en el AIP con facilidad.

También, se encuentran descritos en detalle aquellos puntos considerados de vital importancia como son los Hot Spots del aeropuerto, siendo estos los puntos donde mayores problemas pueden ocasionarse.

5.2.5.1 Salida rápida de pista en aterrizajes

En primer lugar, para la operativa de llegadas, se declaran las preferencias en las calles de salida rápida:

CATEGORIA DE AERONAVE POR ESTELA TURBULENTA // AIRCRAFT CATEGORY DUE TO WAKE TURBULENCE	RWY 32L DIST THR-RET		RWY 32R DIST THR-RET		RWY 18L DIST THR-RET		RWY 18R DIST THR-RET	
	IZQUIERDA LEFT	DERECHA RIGHT	IZQUIERDA LEFT	DERECHA RIGHT	IZQUIERDA LEFT	DERECHA RIGHT	IZQUIERDA LEFT	DERECHA RIGHT
PESADA // HEAVY	L3 (1) 2373 m	L4 (2) 1815 m	K4 2400 m	–	–	Y4 2400 m	Z7 2352 m	Z8 2352 m
MEDIA (REACTORES) MEDIUM (JET)	L5 (1) 1852 m		K5 1800 m			Y5 1800 m		Z10 1926 m
MEDIA (PROP) + LIGERA MEDIUM (PROP) + LIGHT	L7 (3) 1518 m							

(1) Y girar a la izquierda en TWY A, esperar corto de la primera intersección con TWY G // And turn left on TWY A, hold short of first TWY G intersection.
(2) Y esperar corto de TWY A // And hold short of TWY A.
(3) Y seguir instrucciones ATC // And follow ATC instructions.

Figura 5: Calles de salida rápida recomendadas para el Aeropuerto de Madrid-Barajas [17]

Esta información presente en la tabla anterior resulta esencial para que la herramienta pueda partir de unos datos primarios. En el futuro, en base a la retroalimentación de los datos históricos, obtendrá con mayor claridad como suele ser la funcionalidad dependiendo de la operación en cuestión. Además, esto formará parte de las restricciones y requisitos del código, una vez se defina.

5.2.5.2 Calles de rodaje para llegadas y salidas

En la siguiente tabla, se encuentran descritas todas las rutas existentes para la operación de salida como de entrada, teniendo en cuenta a que pista se dirige o procede la aeronave y desde o a que terminal, diferenciando entre Terminal 123, Terminal 4 y Terminal 4-S. A continuación, se presenta un resumen de todas las posibles rutas:

Tabla 3: Rutas a recorrer por las aeronaves en el Aeropuerto de Madrid-Barajas [17]

Inicio	Final	Operación	Configuración	Ruta estándar	Rampas
RWY32L	T-123	Llegadas	Norte	L7, L5 o L3, TWY A hasta A11	Rampa 7 Rampa 6 Rampa 5 Rampa 4 Rampa 3 Rampa 2 Rampa 1 Rampa 0
RWY32L	T-4	Llegadas	Norte	L7, L5 o L3, TWY A, incorporarse a TWY M por la primera posible, continuar hasta M13, J3 (punto de transferencia J3-2)	Rampa 10 Rampa 11 Rampa 12 Rampa 13
RWY32L	T-4S	Llegadas	Norte	• L4, EA1 • L4, L42, L2, B1 • L2, B1	Rampa 20 Rampa 21 Rampa 22 Rampa 23
RWY32R	T-123	Llegadas	Norte	• K5, KA4, KA3, KB2 • K5, KA4, KC3, KC2 • K4, KC3, KC2 • K3, KB2 a TWY A hasta A11	Rampa 7 Rampa 6 Rampa 5 Rampa 4 Rampa 3 Rampa 2 Rampa 1 Rampa 0

RWY32R	T-4	Llegadas	Norte	• TWY A, H2, H3 (punto de transferencia H3-2) • K5, KA4, KC3, KC2, TWY A, H2, H3 (punto de transferencia H3-2) • K4, KC3, KC2, TWY A, H2, H3 (punto de transferencia H3-2) • K3, KB2, TWY A, H2, H3 (punto de transferencia H3-2)	Rampa 10 Rampa 11 Rampa 12 Rampa 13
RWY32R	T-4S	Llegadas	Norte	• K5, KA4, KA3, KB2, TWY A • K5, KA4, KC3, KC2, TWY A • K3, KB2, TWY A • K5, KA4, ..., KA1, o K4, KA3, …, KA1 • K3, KA2, KA1	Rampa 20 Rampa 21 Rampa 22 Rampa 23
T-123	RWY36L	Salidas	Norte	(desde TWY) M10 (punto de transferencia M10-2), …, M17, R5 o R6 o R7, R8, Z2	Rampa 7 Rampa 6 Rampa 5 Rampa 4 Rampa 3 Rampa 2 Rampa 1 Rampa 0
T-4	RWY36L	Salidas	Norte	R3 (punto de transferencia R3-2), … , R1, Z4	Rampa 10 Rampa 11 Rampa 12 Rampa 13
T-4S	RWY36L	Salidas	Norte	N/A	Rampa 20 Rampa 21 Rampa 22 Rampa 23
T-123	RWY36R	Salidas	Norte	(desde TWY) M10 (punto de transferencia M10-2), …, M17. Desde: • M18, ..., M31, NY13, Y1 • M18, ..., M32, N13, Y2 • M18, ..., M33, B13, Y3.	Rampa 7 Rampa 6 Rampa 5 Rampa 4 Rampa 3 Rampa 2 Rampa 1 Rampa 0

T-4	RWY36R	Salidas	Norte	S3 (Punto de transferencia S3-2), M15, ..., M31: • NY13, Y1 • M32, N13, Y2 • M32, M33, B13, Y3.	Rampa 10 Rampa 11 Rampa 12 Rampa 13
T-4S	RWY36R	Salidas	Norte	N/A	Rampa 20 Rampa 21 Rampa 22 Rampa 23
RWY18R	T-123	Llegadas	SUR	• Z10, ZW3, W1, W2, MZ6, …, MZ3, M15, …, M11 • Z8, W1, W2, MZ6, …, MZ3, M15, …, M11 • Z4, ZW1, V1, V2, MZ4, MZ3, M15, …, M11	Rampa 7 Rampa 6 Rampa 5 Rampa 4 Rampa 3 Rampa 2 Rampa 1 Rampa 0
RWY18R	T-4	Llegadas	SUR	Se abandonará RWY 18R por el lado derecho de la misma. • Z10, ZW3, W1, W2, W3 (punto de transferencia W3-2). • Z8, W1, W2, W3 (punto de transferencia W3-2). • Z4, ZW1, V1, AZ5, AZ6, W2, W3 (punto de transferencia W3-2).	Rampa 10 Rampa 11 Rampa 12 Rampa 13
RWY18R	T-4S	Llegadas	SUR	Se abandonará RWY 18R por el lado izquierdo de la misma.	Rampa 20 Rampa 21 Rampa 22 Rampa 23
RWY18L	T-123	Llegadas	SUR	• Y5, AY, BY13, M34, ..., M11 • Y4, BY13, M34, ..., M11 • Y3, A33, N13, M32, ..., M11	Rampa 7 Rampa 6 Rampa 5 Rampa 4 Rampa 3 Rampa 2 Rampa 1 Rampa 0

RWY18L	T-4	Llegadas	SUR	• Y5, AY, BY13, M34, ..., M14, H3 (punto de transferencia H3-2) • Y4, BY13, M34, ..., M14, H3 (punto de transferencia H3-2) • Y3, A33, N13, M32, ..., M14, H3 (punto de transferencia H3-2).	Rampa 10 Rampa 11 Rampa 12 Rampa 13
RWY18L	T-4S	Llegadas	SUR	• Y5, AY, BY13, M34, M33 • Y4, BY13, M34, M33 • Y3, A33, N13	Rampa 20 Rampa 21 Rampa 22 Rampa 23
T-123	RWY14R	Salidas	SUR	(desde TWY) A10 (punto de transferencia A10-2), …, A12, punto de espera en pista.	Rampa 7 Rampa 6 Rampa 5 Rampa 4 Rampa 3 Rampa 2 Rampa 1 Rampa 0
T-4	RWY14R	Salidas	SUR	S3 (punto de transferencia S3-2), S2, A17, puntos de espera en pista LC, LD, LE. Se utilizará como ruta alternativa R3 (punto de transferencia R3-2).	Rampa 10 Rampa 11 Rampa 12 Rampa 13
T-4S	RWY14R	Salidas	SUR	N/A	Rampa 20 Rampa 21 Rampa 22 Rampa 23
T-123	RWY14L	Salidas	SUR	(desde TWY) A10 (punto de transferencia A10-2), …, A12, …, A27, A28: • A29, K1, punto de espera en pista • KB2, K2, punto de espera en pista • KB2, K3, punto de espera en pista.	Rampa 7 Rampa 6 Rampa 5 Rampa 4 Rampa 3 Rampa 2 Rampa 1 Rampa 0

T-4	RWY14L	Salidas	SUR	S3 (punto de transferencia S3-2), S2, A17, ..., A28: • A29, K1, punto de espera en pista • KB2, K2, punto de espera en pista. • KB2, K3, punto de espera en pista.	Rampa 10 Rampa 11 Rampa 12 Rampa 13
T-4S	RWY36R	Salidas	SUR	N/A	Rampa 20 Rampa 21 Rampa 22 Rampa 23

Como se puede observar, la cantidad de opciones son altas, e incluso se complica más cuando se tiene que tener en cuenta el factor de las rampas o plataformas, las cuales no se incluyen en la tabla anterior. Esto demuestra cómo puede ser una tarea complicada para el controlador, el cual debe conocer con exactitud todas estas calles de rodaje y el tipo de operación que está realizando la aeronave. Esto sumado a situaciones de estrés donde el tráfico es elevado puede llevar a que se cometan errores, los cuales, pueden reducir la operativa del aeropuerto. Es por ello por lo que, una herramienta como la propuesta resulta en una solución ventajosa para controlar el flujo de manera segura y optimizada.

5.2.5.3 Hot Spots

Por otro lado, es importante destacar que existen zonas donde el flujo de tráfico interfiere entre ellos, lo cual puede generar aún mayores problemas. Estos puntos, llamados lugares críticos (Hot Spots), también quedan definidos en el AIP y se dividen entre los de configuración Norte y los de configuración Sur.

Con respecto a los lugares críticos en configuración norte, se encuentran [19]:

1. HS-1-NORTE: las aeronaves que pretenden abandonar la pista por la salida rápida tendrán prioridad.

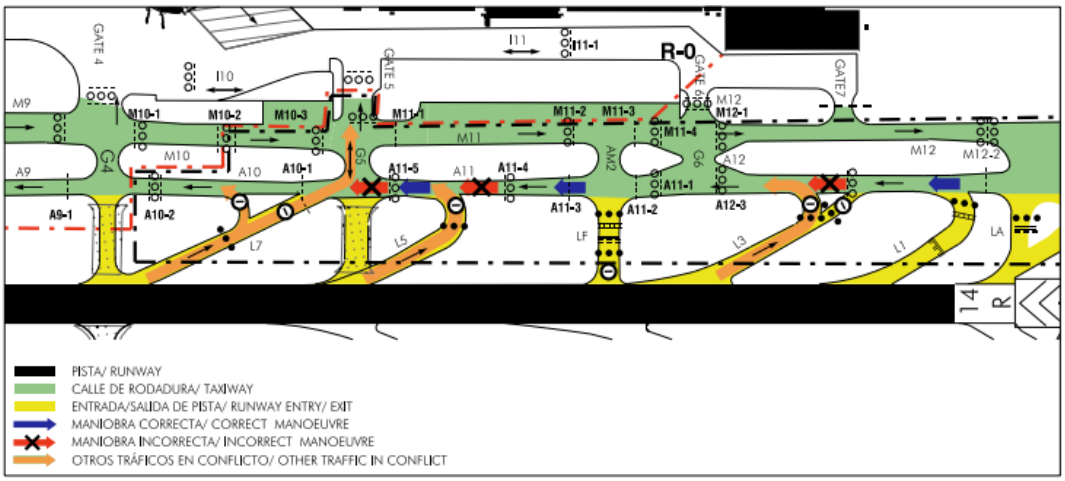


Figura 6: HS-1-NORTE

2. HS-2-NORTE: las aeronaves que circulan por M10 y A10 deben detenerse en los puntos de transferencia acordes.

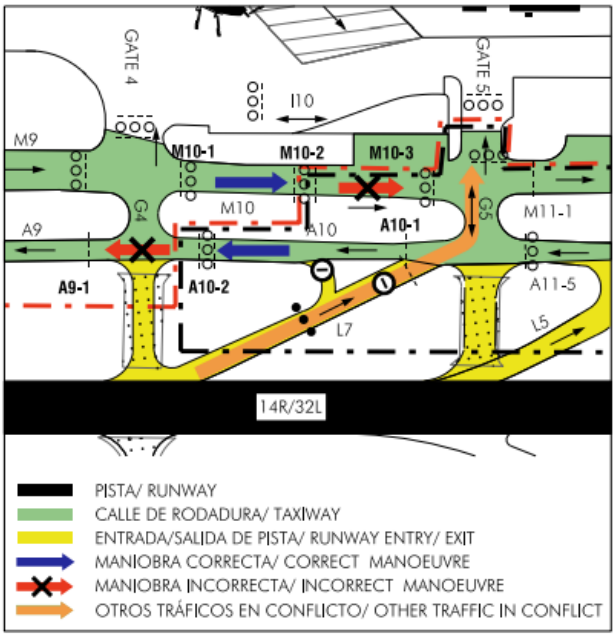


Figura 7: HS-2-NORTE

3. HS-3-NORTE: aeronaves que estén en M13 deben detenerse en el punto de espera intermedio.

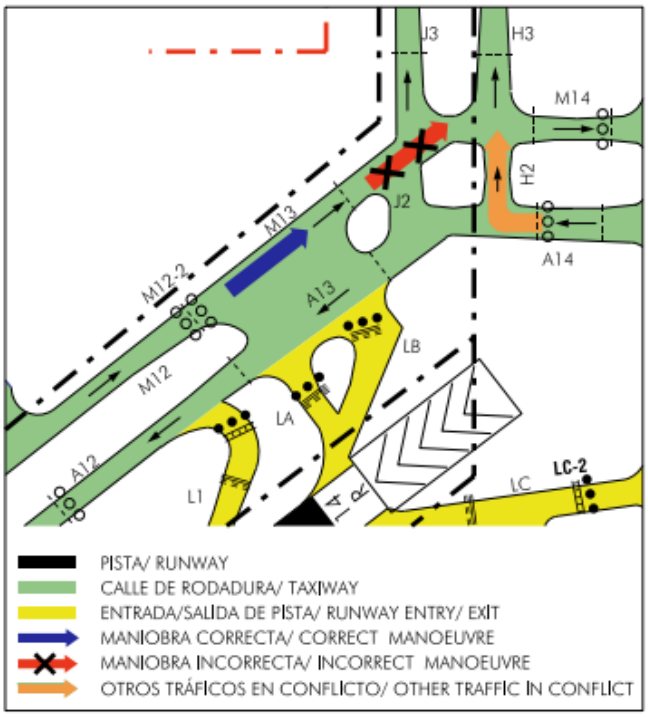


Figura 8: HS-3-NORTE

4. HS-4-NORTE: aeronaves rodando por A17 deben girar a la izquierda en A14.

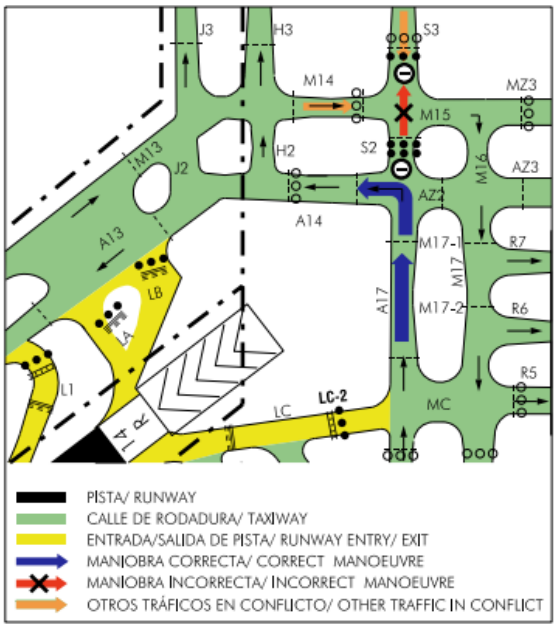


Figura 9: HS-4-NORTE

5. HS-5-NORTE: aeronaves en Y1, Y2 e Y3 no deben acceder a pista sin autorización.

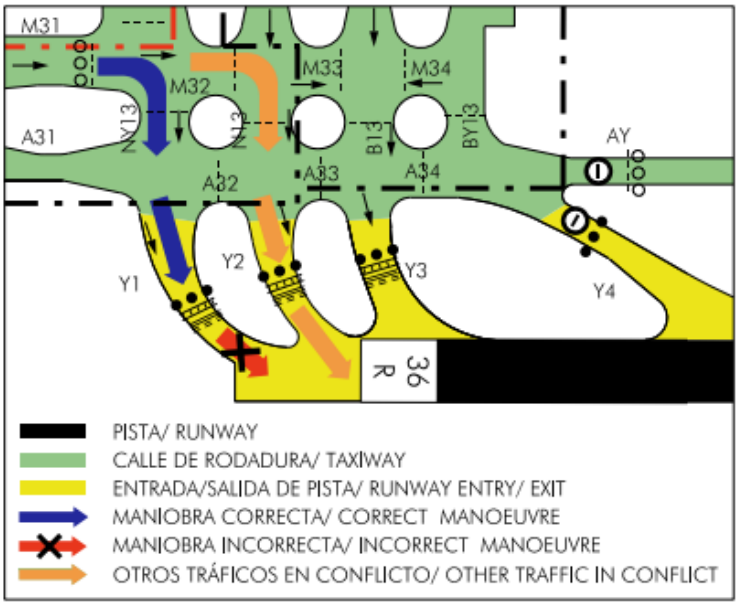


Figura 10: HS-5-NORTE

6. HS-6-NORTE: aeronaves en calle de rodaje M en dirección a RWY 36R deben continuar recto y no girar a la derecha.

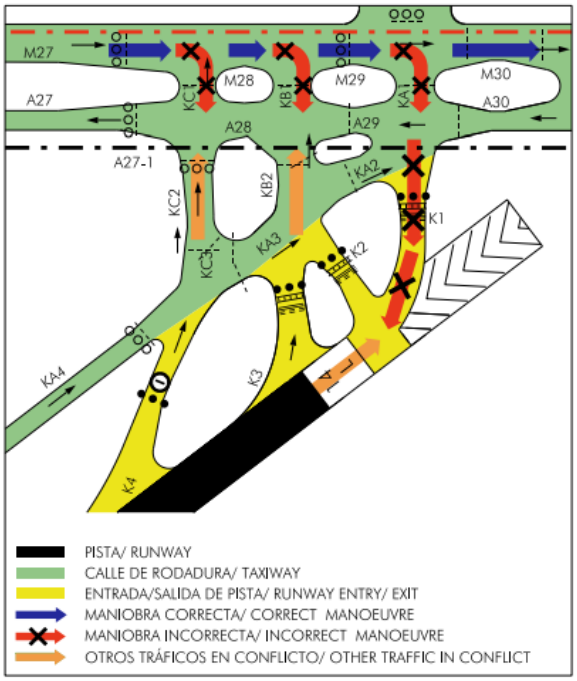


Figura 11: HS-6-NORTE

7. HS-7-NORTE: aeronaves en los puntos de espera deben pararse lo más próximo al punto de espera de la pista.

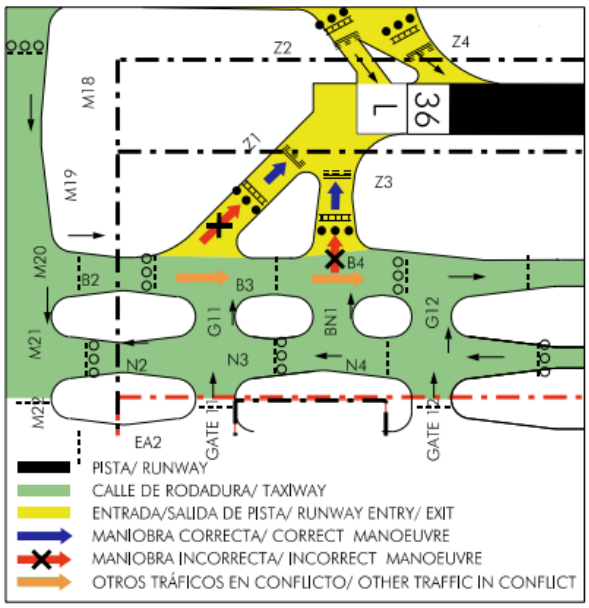


Figura 12: HS-7-NORTE

Con respecto a la operación en configuración Sur, solo existen 3 zonas críticas [20]:

1. HS-1-SUR: aeronaves por M10 y A10 deben detenerse en los puntos de transferencia.

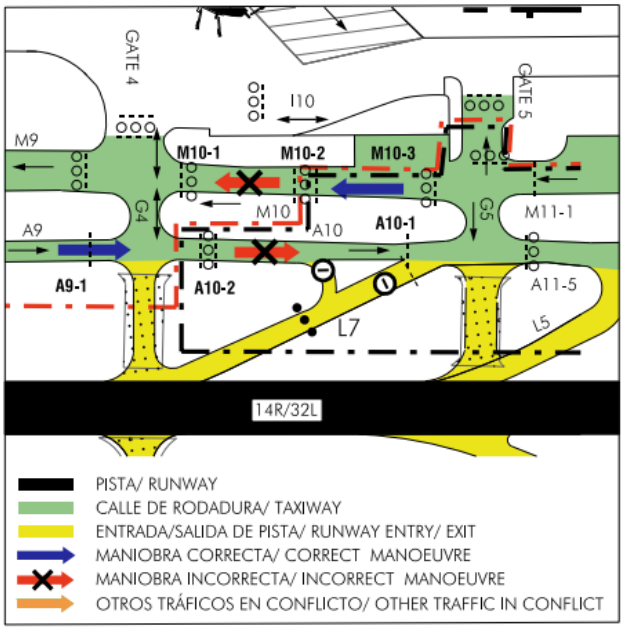


Figura 13: HS-1-SUR

2. HS-2-SUR: aeronaves provenientes de RWY 18R por MZ instruidas a esperar, deben detenerse en el punto de espera intermedio de MZ3.

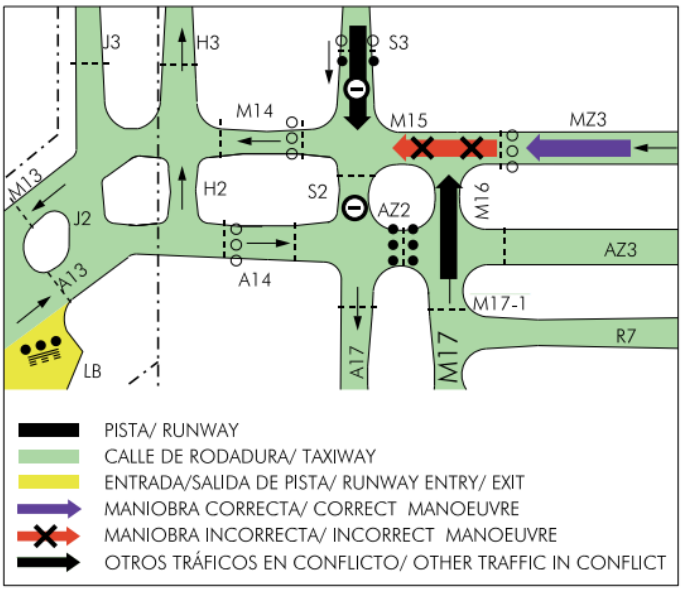


Figura 14: HS-2-SUR

3. HS-3-SUR: aeronaves esperando para entrar en pista (14R) deben posicionarse lo más próxima al punto de espera.

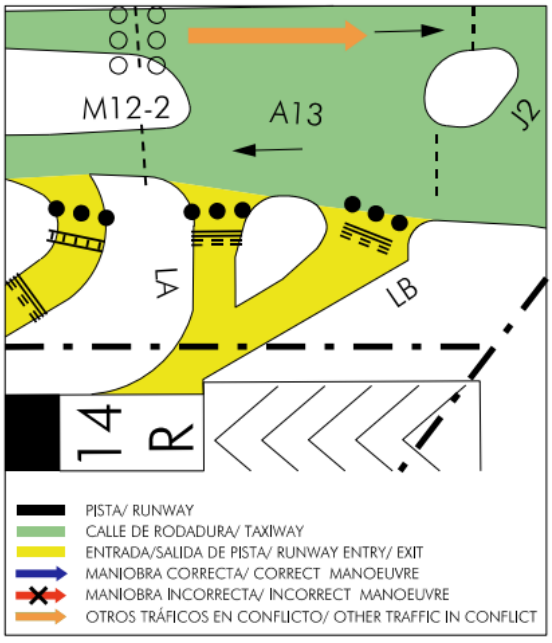


Figura 15: HS-3-SUR

5.2.5.4 Tiempos de taxi

Eurocontrol realiza análisis anuales del tiempo medio de taxi en distintos aeropuertos mundiales, incluyendo los tiempos para el aeropuerto de Madrid. Estos datos muestran los valores obtenidos para el taxi-in, es decir, el taxi cuando la aeronave aterriza hasta que alcanza su puesto de estacionamiento, el taxi-out medio, lo cual es desde su puesto de estacionamiento hasta la pista de despegue, y por último, ofrece también la media de taxi-out según las estelas turbulentas.

En el caso de Madrid, los datos son [21]:

- Taxi-in: promedio de 9,1 minutos con una desviación típica de 3,7
- Taxi-out: promedio de 17,1 minutos con una desviación típica de 5,2
- Taxi-out según estela turbulenta:
    - Heavy: promedio de 17,6 minutos con una desviación típica de 6,1
    - Medium: promedio de 17,1 minutos con una desviación típica de 5.

Cabe destacar que estos datos son del verano de 2021, donde la aviación empezó a recuperarse después del COVID. Por ello, estos valores irán en aumento a medida que el tráfico alcance las operaciones habituales.

Toda esta información es de alta relevancia ya que muestra que una mejoría considerable es posible, más existiendo una gran diferencia entre el taxi-in y el taxi-out. Además, la desviación típica también demuestra que estos tiempos son relativos y que dependerán considerablemente de la elección de pista y puesto de estacionamiento, así de posibles retrasos debidos a conflictos.

5.2.5.5 Puesta en común

Teniendo todo lo anterior en cuenta, se observa que el aeropuerto de Madrid es un gran candidato para ser uno de los aeropuertos que utilicen este tipo de sistemas y, por tanto, una mejora del nivel 2 al nivel 4 parece ser beneficioso, posicionándolo como aeropuerto pionero y optimizado, siendo más seguro y responsable con el medio ambiente. Asimismo, cabe destacar que un paso intermedio por el nivel 3 es posible, sin embargo, con el notorio avance de la tecnología actual, y el aumento esperado en las operaciones en Madrid resulta interesante ir directamente por el nivel máximo. Por ello, se podría comenzar a desarrollar una primera versión de la herramienta la cual ya pueda ofrecer servicios de gestión de tráfico en superficie. Además, al conocer los puntos donde los posibles conflictos pueden ocurrir, obliga a prestar especial atención a ellos al diseñar el sistema.

En caso de querer integrar esta herramienta en otros aeropuertos, un análisis previo como el anterior será vital para que funcione correctamente. Además, este proceso demuestra la gran

utilidad que tendrá la herramienta para la operación segura de aeronaves en los aeropuertos y, también, se observa que Madrid es un buen aeropuerto donde poder comenzar este proyecto.

5.3 Interesados del proyecto

En el proyecto de desarrollo e implementación de un Sistema Avanzado de Guiado y Control del Movimiento en Superficie para el aeropuerto de Madrid-Barajas existirán una gran cantidad de interesados que verán de primera mano las ventajas que les puede suponer. Estas ventajas pueden ser desde beneficios económicos hasta confianza por parte del viajero, lo cual hace que haya numerosos beneficiados.

En primer lugar, se destaca el gestor aeroportuario, en este caso AENA. Esto será debido a la mejora en tiempos de taxi, disminución de las emisiones en el aeropuerto, mayor capacidad, y reducción de conflictos en el área de maniobras. Asimismo, al ser una herramienta que puede ser integrada fácilmente en otros aeropuertos, le permitirá a AENA mejorar la mayor parte de sus instalaciones, obteniendo una mejor valoración como gestor aeroportuario.

Por otro lado, un segundo gran interesado es ENAIRE, al ofrecer los servicios de navegación aérea en la mayor parte de aeropuertos españoles. De este modo, servirá como herramienta la cual podrá empezar a utilizar y conseguir que sus servicios sean más seguros y eficientes. Esto, al igual que ocurre con AENA, genera una mejor visión como empresa, abriéndoles puertas a nuevos contratos con futuros clientes. De igual forma, para los controladores de movimientos en tierra resulta una herramienta esencial para reducir esos momentos de estrés y facilitar su trabajo.

Las empresas de handling también tendrán un especial interés sobre el uso de este tipo de herramientas ya que los movimientos en tierra estarán mejor organizados y, en cuanto surja un problema, obtener una solución por parte de la torre de control será más rápido. Cabe destacar que en el aeropuerto de Madrid-Barajas, las empresas de handling que están en actual funcionamiento son: GroundForce, Iberia y WFS Servicios Aeroportuarios [22].

En cuanto a las aerolíneas, estas también tendrán un beneficio con el empleo de la herramienta propuesta, ya que reducirá el tiempo de taxi que, al largo plazo, reducirá considerablemente las emisiones de combustible y los costes del mismo. De igual manera, podrán reducir aquellos vuelos que estén operando en retraso, mejorando la imagen de la compañía y la satisfacción del cliente. En el caso del aeropuerto de Madrid-Barajas, los mayores interesados serían Iberia, Iberia Express, Air Europa y Ryanair, entre otros, no obstante, hasta 74 compañías llegan a operar en el aeropuerto, siendo estas, tanto vuelos con pasajeros, como de carga. Asimismo, para los propios empleados de las aerolíneas resulta beneficioso, ya sean pilotos, lo cual facilitará su trabajo y les proporcionará un extra en materia de seguridad en las operaciones, como para los que planifican las operaciones de la

compañía, ya que permite calcular con mayor exactitud el tiempo que empleará la aeronave para realizar el taxi. Al igual, podrá conocer en tiempo real este valor en caso de tener que realizar modificaciones inoportunas ya sea durante el handling del avión o en el próximo destino.

Por último, se destaca que el pasajero final también resulta ser un claro interesado en dicho proyecto ya que reduce su tiempo de viaje. Además, la operación se llevará con mayor fluidez y de manera más segura, lo cual da confianza y satisfacción por la aerolínea en la que vuela y el mismo aeropuerto.

Capítulo 6. Diseño del proyecto

6.1 Alcance del proyecto

Definir de manera clara el alcance del proyecto es esencial para que este pueda llevarse a cabo en tiempo y forma. Resulta de vital importancia conocer hasta qué punto se declara el proyecto ya que, con el avance de la tecnología, se puede llegar al riesgo de querer mejorarlo continuamente y que, finalmente, no se pueda presentar un proyecto finalizado. No obstante, esto no implica que en un futuro no se pueda actualizar y mejorar la herramienta ajustándose a las necesidades de cada momento.

De este modo, el alcance queda definido por los siguientes puntos:

- Desarrollar una interfaz de usuario que permita a los controladores de superficie vigilar y controlar las aeronaves en el área de maniobras. En esta interfaz de usuario se mostrará la ruta óptima para cada momento, pudiendo ser aceptada por el controlador y, del mismo modo, pudiendo dar autorizaciones a las aeronaves para que puedan ejecutar los movimientos pertinentes.
- La herramienta debe conocer el estado del aeropuerto y la posición de las aeronaves. Para ello, se hará uso del radar de superficie, A-CDM y la información del Network Manager (EuroControl), entre otros. Además, permitirá introducir posibles cierres inesperados en las calles de rodaje, consiguiendo así recálculos de rutas.
- Mediante bases de datos, las operaciones quedarán registradas para la posterior realimentación de la herramienta y aumento de capacidades de la misma. Estos datos servirán para conocer cómo opera cada aerolínea en lo que respecta a velocidad de taxi y salidas rápidas tomadas habitualmente.
- La herramienta se desarrolla de manera inicial para el aeropuerto de Madrid-Barajas, con las rutas de taxi ya establecidas en el AIP. En el futuro, estas podrían ser modificadas bajo aprobación de AENA y ENAIRE, una vez la herramienta demuestre sus altas capacidades. Asimismo, la herramienta podrá integrarse en otros aeropuertos una vez establecida en el aeropuerto de Madrid-Barajas.
- La herramienta alertará al controlador en caso de problemas, ya sea por movimientos inadecuados por aeronaves o vehículos terrestres, o por fallo del sistema.
- La herramienta será integrada con las luces, señales y letreros a lo largo de las calles de rodaje, facilitando el trabajo a los pilotos. De este modo, la aeronave solo tendrá que seguir los caminos mostrados por las luces, sin tener que depender de comunicaciones continuas con el controlador de superficie.
- La herramienta trabajará en tiempo real, es decir, propondrá continuamente las mejores alternativas en caso de haberlas. Para ello, tendrá en cuenta las calles de rodaje abiertas, otras aeronaves en conflicto con la ruta de la que está operando y el punto final al que se dirige la aeronave en cuestión. También, indicará el tiempo

estimado de taxi para tener un mayor control sobre las aeronaves y los agentes handling.

- Al tratarse de información de elevada importancia, la herramienta deberá tener un servicio de operación de funciones reducidas en caso de problema, con el fin de no paralizar la operación en el aeropuerto.

Teniendo en consideración todo lo anterior, ahora es posible calcular una posible estimación de tiempo hasta la ejecución del proyecto, las mismas capacidades de la herramienta y los hitos o entregables más destacados. Como se puede observar, estos puntos también definen parcialmente los objetivos del proyecto, lo cual es imprescindible tenerlos claros para la correcta ejecución del mismo.

6.2 Definición de requisitos

A la hora de diseñar un nuevo proyecto, resulta imprescindible definir una serie de requisitos, los cuales se ajustarán a las necesidades del cliente y la normativa legal existente. En este caso, al tratarse de un proyecto aeronáutico, la industria está altamente regulada por distintos organismos, ya sean internacionales o nacionales, los cuales definen una serie de criterios a cumplir por todos los involucrados. A continuación, se presentan los siguientes requisitos regulatorios o documentos a cumplir para que la herramienta propuesta pueda ser validada y certificada para su posterior puesta en funcionamiento:

- Documento de OACI Doc 9830: Advanced Surface Movement Guidance and Control Systems (A-SMGCS) Manual. En él se encuentran los puntos a cumplir en el desarrollo de herramientas como la que se pretende crear. Entre los puntos y requisitos más importantes, se encuentran [23]:
  - Requisitos operacionales
  - Requisitos de desempeño
  - Guiado en la aplicación de requisitos operacionales y de desempeño.
  - Problemas de implementación
- La herramienta deberá cumplir con los puntos especificados en el AIP de ENAIRE para el Aeropuerto de Madrid-Barajas, destacando los siguientes documentos:
  - AD 2-LEMD
  - AD 2-LEMD GMC 1
  - AD 2-LEMD GMC 2
  - AD 2-LEMD ADC 1
  - AD 2-LEMD PDC 1
  - AD 2-LEMD PDC 2
- Documento de OACI Doc 4444: Gestión del tránsito aéreo. En este documento se definen aquellas distancias de separación obligatorias tanto en llegadas como en salidas, las cuales serán esenciales para la predicción de rutas por parte de la herramienta [24].

Por otro lado, el proyecto también contendrá otro tipo de requisitos ya sean de negocio, funcionales, técnicos o de calidad. En los siguientes puntos se presentan el resto de los requisitos:

- **Requisitos de negocio:** la herramienta deberá cumplir con las exigencias económicas esperadas, lo cual implica sacar unos ingresos económicos que hagan rentable la inversión realizada.
- **Requisitos funcionales:** están muy ligados al propio alcance del proyecto, el cual pretende tener una interfaz de usuario de fácil uso para simplificar el trabajo de los controladores. Además, la herramienta tendrá que cumplir con los requisitos regulatorios expuestos además de todas aquellas rutas de rodaje ya existentes en el aeropuerto de Madrid-Barajas.
- **Requisitos técnicos:** en cuanto a estos requisitos, al ser una herramienta que obtiene parte de sus datos a partir de otros sistemas ya integrados en el aeropuerto, esta deberá poder conectarse a ellos. Entre los sistemas a los que deberá poder intercambiar datos serán:
    - Sistema de luces, señalética y carteles presentes en el aeropuerto
    - A-CDM
    - Radar de superficie
    - Sistema ATC en vuelo para el control de aeronaves en llegadas
    - Network Manager
- **Requisitos de calidad:** será prioritario que el sistema sea robusto y funcione aún bajo problemas eléctricos en el aeropuerto, mediante una fuente de energía secundaria. Asimismo, en caso de perder cierta información por fallo de alguna conexión o del sistema, se posibilitará una opción de funciones reducidas para no paralizar el tránsito en el aeropuerto. Asimismo, se busca cumplir con la normativa de calidad ISO-9001, dando así confianza a los clientes del producto.

6.3 Diseño conceptual

En primer lugar, antes de entrar en profundidad como va a ser el funcionamiento detallado de la herramienta, se muestra un diseño conceptual de cómo debería funcionar. Para ello, se presenta el siguiente esquema de flujo del sistema, el cual muestra cómo actúa el sistema:

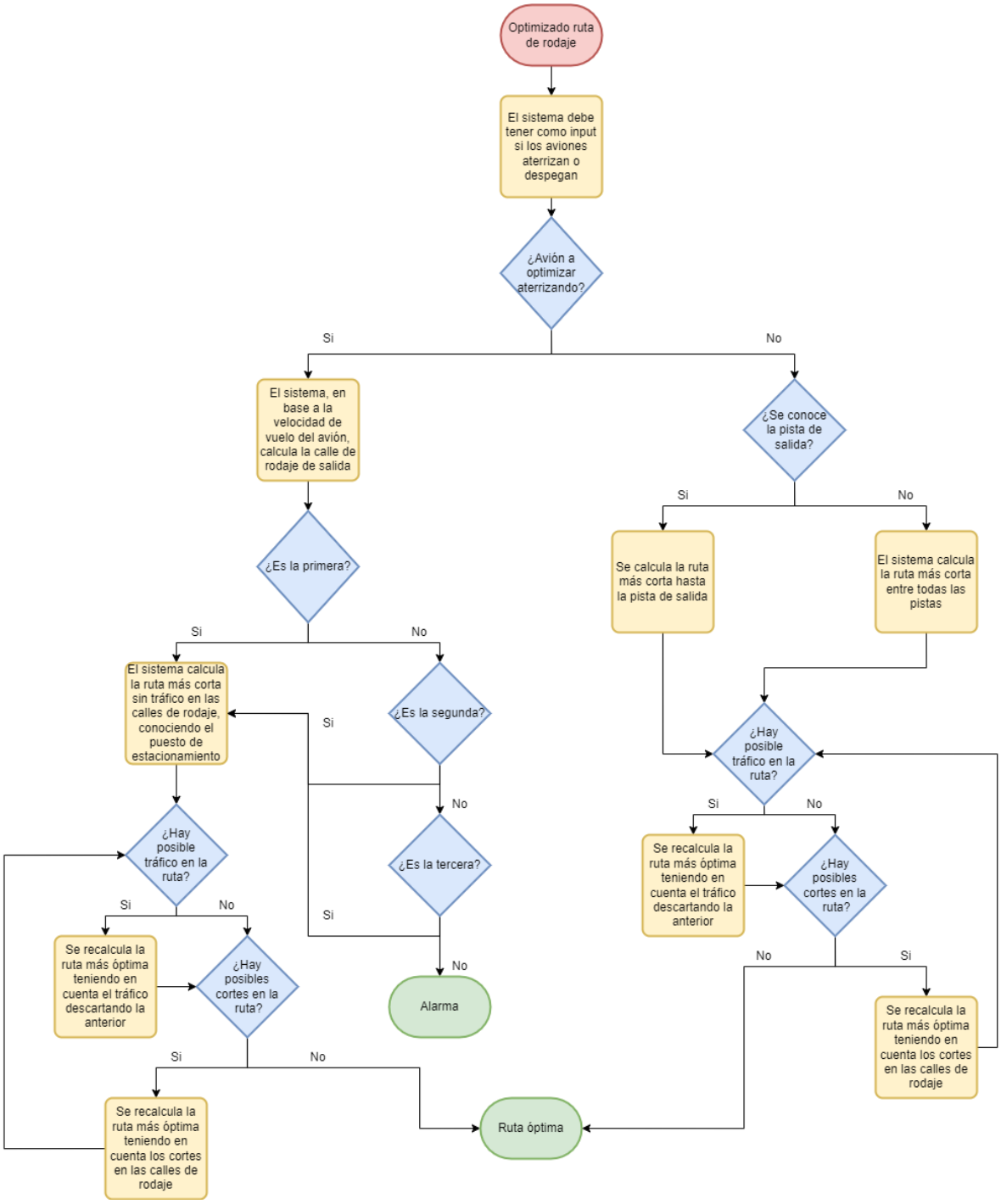


Figura 16: Diagrama de flujo del sistema

Como se puede observar, la herramienta recalcula la ruta en caso de haber un corte en la calle de rodaje elegida, primeramente. Además, analiza todas aquellas rutas sin conflictos con otros tráficos. En este diagrama no se declara el tipo de configuración del aeropuerto, el cual vendría en la parte superior, donde a partir de este input, tendrá unos datos de partida u otros,

referentes a las calles de rodaje a utilizar. Del mismo modo, tampoco se establecen en este diagrama los inputs aportados por el controlador, el cual deberá dar las autorizaciones referentes a cada punto crítico, y posibles alarmas que le puedan saltar debido a conflictos entre diversos aviones. No obstante, esto se desarrollará en mayor nivel de detalles más adelante.

De este modo, la forma de funcionar queda resumida según las entradas y salidas mostradas a continuación:

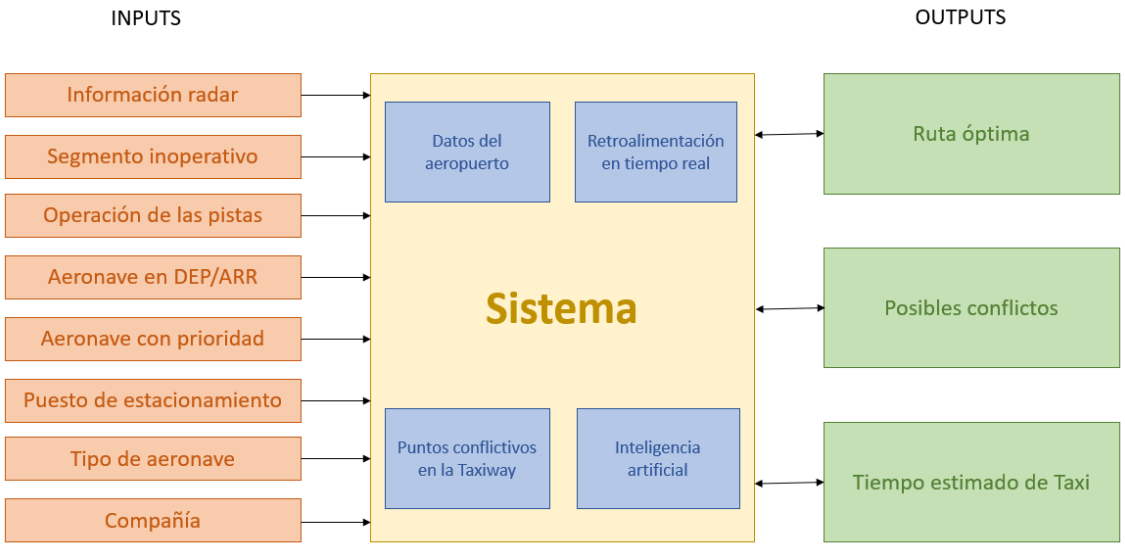


Figura 17: Entradas y salidas del sistema

En la anterior figura se puede analizar que, el sistema tendrá una serie de entradas las cuales destacan la información referente a la propia aeronave (tipo y aerolínea), la configuración y datos en tiempo real de la situación en el aeropuerto y la información radar entre otros. Con ello se obtiene los outputs esenciales para el controlador, los cuales son la ruta óptima, los posibles conflictos y el tiempo estimado de taxi.

Para que todo ello funcione de manera ordenada y correcta, es necesario que el sistema pueda obtener una realimentación de todo lo que está calculando para poder adelantarse a posibles conflictos futuros. Además, es necesario declarar en el sistema toda aquella información correspondiente a calles de rodaje y puntos críticos de las mismas calles para que pueda reconocer el recorrido por el que puede circular una aeronave. Por último, se destaca la inteligencia artificial que, mediante toda esa retroalimentación, podrá calcular la ruta óptima para cada aeronave en cada momento, y poder ajustarla en caso necesario.

Por otro lado, a la hora de representar todos los datos, será necesario el diseño de una interfaz de usuario que permita al controlador ver todo lo que está ocurriendo en tiempo real. Esta mostrará un mapa en el cual se podrán observar todas aquellas aeronaves en el área de

maniobras, y pinchando en cada una de ellas se observará información importante al respecto y la ruta óptima:

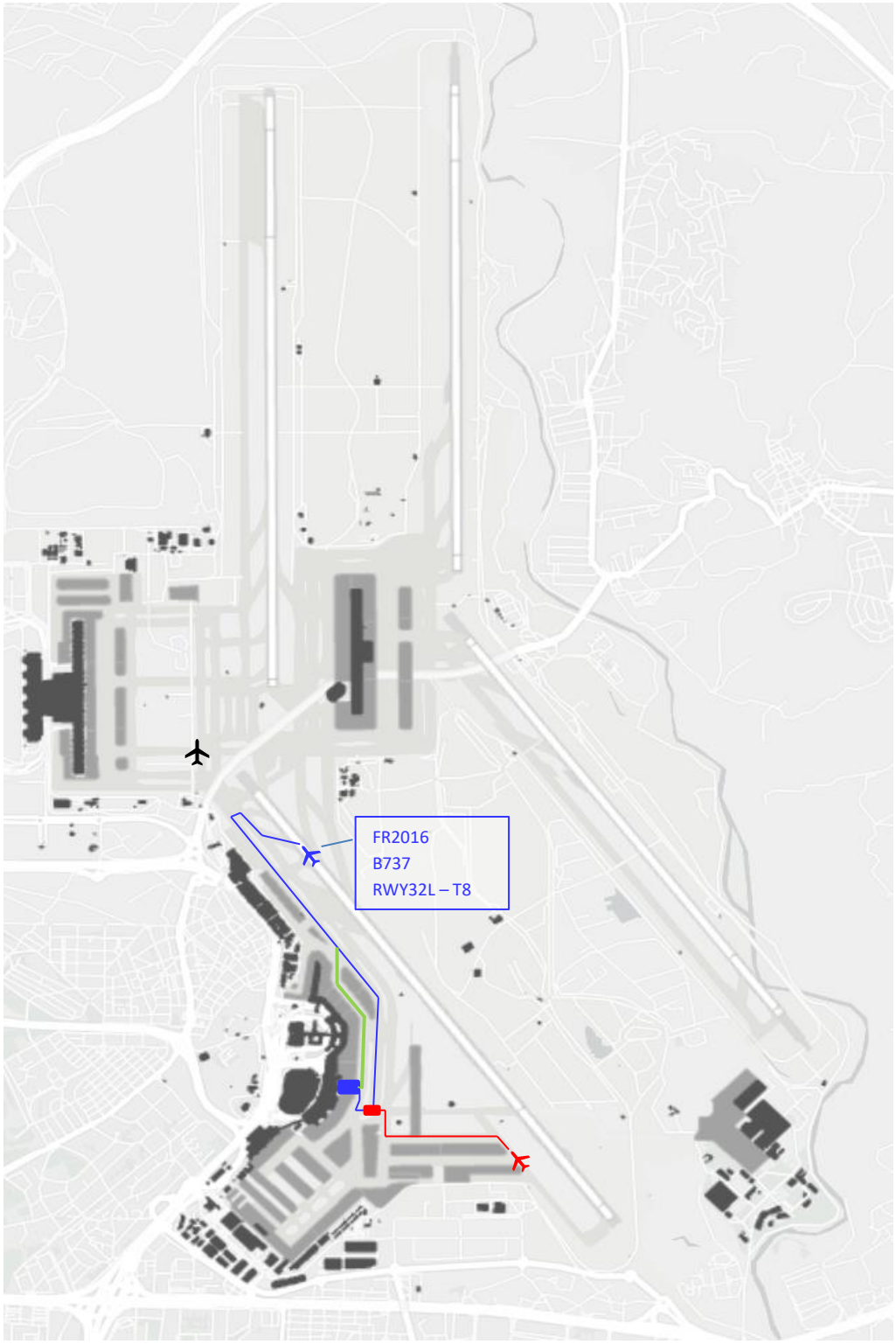


Figura 18: Interfaz de usuario preliminar

Como se observa en la imagen anterior, se puede observar tres aeronaves, siendo la azul bajo estudio (FR2016). Se ve como esta tiene un conflicto con la aeronave roja, mostrando donde va a ocurrir dicho conflicto. Debido a ello, el sistema ofrece una ruta alternativa que reduce el conflicto y permite alcanzar el puesto de estacionamiento (T8) de manera más rápida (camino verde). Asimismo, se observa la aeronave negra, la cual, al estar en otra zona del aeropuerto y no influir sobre la que está bajo estudio, no aparece ninguna información al respecto.

Teniendo todo lo anterior en consideración se puede proceder a diseñar la herramienta a más bajo nivel. No obstante, un estudio preliminar con el diseño conceptual resulta esencial para poder presentar la idea a los interesados en primera instancia y demostrar que realmente existe una necesidad y que trae ciertos beneficios el poder utilizarla.

Capítulo 7. Plan estratégico

El plan estratégico detalla cómo se va a desarrollar el proyecto en el corto, medio y largo plazo, utilizando para ello, proyecciones tanto cuantitativas como cualitativas que permiten analizar aquellos hechos para alcanzar los objetivos marcados. En este caso, se va a realizar un análisis estratégico, evaluando el entorno interno y externo, además de la propia competencia, la cual ya se describió en apartados anteriores. De este modo, se desarrolla el plan de acción que debe seguir el proyecto para que cumpla las expectativas impuestas. En este plan de acción se incluye desde los mismos objetivos del proyecto y resultados esperados hasta el cronograma del propio proyecto.

7.1 Análisis estratégico

En el siguiente análisis estratégico, se pretende demostrar las capacidades del proyecto, demostrando que este ofrece servicios los cuales la competencia no ofrece. Considerando los proyectos de la competencia, anteriormente descritos, se procede a realizar un análisis del entorno tanto interno como externo y, a su vez, una evaluación de la competencia.

7.1.1 Análisis de entorno interno y externo

Para la realización del análisis del entorno, se utiliza la matriz DAFO, donde se describen de manera simplificada las debilidades y fortalezas del proyecto, siendo esto el análisis interno, y para el análisis externo se describen las amenazas y oportunidades del mismo. Con esta herramienta se consigue detectar rápidamente aquellas debilidades y amenazas, teniendo así una visión global del proyecto, pero que se pueden tener en cuenta en el momento de desarrollo para poder reducirlas al máximo,

Cabe destacar que, la matriz DAFO realizada está orientada al proyecto ejecutándose en el aeropuerto de Madrid, el cual cuenta con un sistema A-SMGCS de nivel 2 en la actualidad. Por tanto, la implementación de un sistema como este representa una mejora notable.

A continuación, se presenta la matriz:

Tabla 4: Matriz DAFO

Fortalezas

- Sistema A-SMGCS de nivel 4 no utilizado en el aeropuerto, lo que supone una gran mejoría.
- Cantidad y efectos de los errores humanos más bajos por parte de pilotos y controladores.
- Mejora la capacidad aeroportuaria en el aeropuerto de Madrid.
- Reducción de emisiones durante Taxi.
- Previsión de conflictos entre aeronaves y vehículos en el área de maniobras.
- Priorización de aeronaves según necesidades.
- Ventaja para aerolíneas y pasajeros al reducir el tiempo de Taxi.
- Reducción del trabajo repetitivo de los controladores.
- Reducción en gastos eléctricos con el control de luminarias y señalética.
- Mayor facilidad para la toma de decisiones por parte de los controladores.
- Mejor organización en el aeropuerto, conociendo posibles riesgos con anterioridad.
- Control de autorizaciones más exhaustiva, evitando posibles conflictos entre ellas.

Debilidades

- Nuevos sistema que los controladores deben aprender a utilizar.
- Alta capacidad de cálculo necesaria.
- Pocos potenciales clientes para el futuro (aeropuertos con calles de rodaje complicadas y muy saturados).
- Difícil integración con sistemas actuales de control.
- Variables difíciles de conocer (ej. Velocidad de Taxi de distintas aerolíneas).
- Procesos complicados de integración, con necesidad de periodos de prueba exhaustivos.
- Riesgos con efectos muy altos.
- No pertenecer a una empresa ya ubicada en el mercado con un respaldo económico importante.

Oportunidades

- Cumplir con los requisitos impuestos de reducción de emisiones en el entorno aeroportuario, convirtiendo al aeropuerto de Madrid en un espacio más verde.
- Posibilidad de crecimiento del aeropuerto, aumentando su capacidad al tener una mejora en su organización y control.
- Otros aeropuertos interesados en el proyecto.
- Oportunidad de expansión dado por la confianza de un proveedor de servicios de gestión de tráfico aéreo.
- Consorcio con empresa considerada en el entorno de la gestión de tráfico aéreo.
- Convertir al aeropuerto de Madrid en un aeropuerto referencia mundial
- Obtener mejores notas de satisfacción por parte de los usuarios del aeropuerto de Madrid.
- Control del efecto de posibles obras o situaciones con anterioridad para poder abordarse con el menor efecto posible.

Amenazas

- Competidores de gran tamaño que decidan integrar sus productos similares en el aeropuerto de Madrid
- Resistencia al cambio por parte de los controladores.
- Posibles imprevistos no considerados en el software.
- Bajo interés por parte del aeropuerto de Madrid al considerar que los picos de tráfico son ocasionales.
- La existencia de un fallo que reduzca la confianza del cliente.
- La transmisión de datos incorrectos por parte de un sistema externo necesario para el correcto funcionamiento del A-SMGCS propuesto.
- Estancamiento del tráfico aéreo en Madrid, descartando la necesidad de productos como tales.
- Evolución más rápida de la tecnología con respecto a lo esperado que ofrezca productos mejores.

7.1.2 Evaluación de la competencia

Para la evaluación de la competencia se ha utilizado el modelo de las 5 fuerzas de Porter, permitiendo conocer en profundidad el sector en el que se encuentra el proyecto, anticipar y preparar posibles cambios en el espacio de trabajo, implementar mejoras al proyecto propuesto y, por último, identificar nuevas oportunidades de negocio. Para ello, se va a hacer uso de la información que se presentó con anterioridad donde se describían los distintos sistemas en funcionamiento de 4 empresas del sector.

7.1.2.1 Fuerza 1 - Poder de negociación del cliente

El aeropuerto de Madrid o el proveedor de servicios de control de tráfico aéreo, en este caso ENAIRE, tiene un alto poder de negociación al haber pocos clientes en el mercado para este tipo de tecnologías. Por tanto, cada aeropuerto implica una cantidad significativa en el volumen de negocio. Asimismo, este tipo de tecnologías son idóneas para aeropuertos grandes con calles de rodaje complicadas y con elevada saturación. Por ello, los potenciales clientes para este proyecto resultan ser reducidos.

Además, el aeropuerto de Madrid mueve mucho tráfico aéreo, y garantiza la seguridad del mismo. En caso de que algún sistema falle, las consecuencias y efectos son muy altos, especialmente económicos si es necesario el cierre de operaciones. Por ello, el aeropuerto debe asegurarse que el sistema cumpla con los requisitos marcados y que en caso de fallo no genere este tipo de efectos.

Por otro lado, la regulación existente está muy definida, lo cual hace que el sistema se deba adecuar a dicha normativa, bajando el poder de negociación del cliente.

También es importante señalar que, existen pocas alternativas en el mercado, teniendo que hacer todas las empresas dedicadas a la creación de sistemas como este un estudio previo de la operativa en el aeropuerto. Esto hace que la integración sea lenta para un aeropuerto específico, lo cual reduce el poder de negociación del cliente ya que no puede demostrar que otra compañía tendría el producto disponible en un tiempo considerablemente menor.

7.1.2.2 Fuerza 2 - Poder de negociación del proveedor

Se trata de un sistema relativamente sencillo al ser esencialmente software desarrollado internamente. Por tanto, la cantidad de proveedores es pequeño, siendo necesario únicamente unas pantallas que cumplan con los requisitos de operatividad y continuidad,

además de unos servidores acordes a los estándares. En este tipo de tecnologías existen diversos proveedores en el mercado, lo cual reduce su poder de negociación.

Por otro lado, el lenguaje de programación, al ser de libre uso y disponible para cualquier usuario, no será necesaria su negociación. El único posible problema sería el empleo de ciertas librerías que realicen funciones específicas, las cuales pueden generar problemas en el futuro.

Para que el sistema funcione, como se vio con anterioridad, la herramienta necesita de información radar para el correcto conocimiento de la posición de la aeronave en el aeropuerto. Para ello, es necesario que esta provenga de sistemas presentes en el aeropuerto, como puede ser el radar de superficie. No obstante, esta información también es necesaria para otros datos relativos a otros interesados, por ello, el aeropuerto y el proveedor de servicios de navegación aérea son los principales interesados en conocer dicha información. Por tanto, este proveedor debería ya haber negociado con anterioridad con el propio aeropuerto y, por tanto, tener un poder de negociación para este proyecto menor.

7.1.2.3 Fuerza 3 - Amenaza de entrada de nuevos competidores

Resulta relativamente sencillo para empresas con el conocimiento adecuado en trayectorias, inteligencia artificial y optimización para crear una herramienta similar. Sobre todo, aquellas empresas ya dedicadas al desarrollo y diseño de herramientas de gestión de tráfico aéreo. No obstante, al igual que el proyecto que se presenta en este documento, estas empresas deberán realizar un estudio previo del aeropuerto para poder realizar la herramienta correspondiente.

Por otro lado, una vez creado el producto e integrado en los aeropuertos, los competidores verán complicado entrar en el aeropuerto de Madrid ya que el desarrollo de este tipo de herramientas resulta costoso y lleva un tiempo largo de trabajo el cual puede ser insostenible si no es certera su puesta en operación.

Asimismo, si no se tratan de empresas relacionadas con el mundo de la gestión de tráfico aéreo, puede resultar aún más complicado ya que la confianza del cliente no es alta, y este se decantará por empresas ya constituidas en el mercado y con conocimientos y experiencia previa. Por ello, esto también resulta una debilidad, para este proyecto y, por ende, buscar un consorcio con una empresa ya situada en el mercado puede ser una clara ventaja.

7.1.2.4 Fuerza 4 - Amenazas de productos y servicios sustitos

Una posible amenaza que resulta interesante para el propio aeropuerto y gestor de navegación aérea sería un producto que integre el servicio de optimización en las calles de

rodaje con los sistemas de gestión de tráfico aéreo una vez las aeronaves estén volando. Esto facilitaría el uso de la herramienta para los controladores ya que la interfaz a aprender sería solamente una, y el intercambio de posiciones en la torre de control podría llevarse de manera más sencilla. Además, esto generará menos miedo al cambio ya que el sistema de control de la navegación aérea es bastante más avanzado y permite funcionalidades que el actual de control de movimientos por superficie no ofrece. A día de hoy, el que ofrece dichos servicios en España es INDRA con su producto llamado SACTA, encargado del control de tráfico aéreo en los aeropuertos españoles [25].

Por otro lado, no existe un amplio mercado de sistemas que controle el tráfico de aeronaves en tierra, y los productos que ya hay creados, no tienen una fácil integración sobre otro aeropuerto. Para ello, como se comenta con anterioridad, es necesario un estudio del aeropuerto, con las rutas de las calles de rodaje, hot spots y configuración del aeropuerto. Por ello, la integración de este tipo de productos también conllevaría un tiempo hasta que sea posible.

Un servicio sustituto, que debido al avance de la tecnología sea posible sería la integración de productos en las propias aeronaves o sobre las calles de rodaje, generando una menor dependencia en los controladores de tierra. Sin embargo, este tipo de productos resulta altamente complicados de realizar en el corto plazo ya que implica que todas las aeronaves deben añadir este tipo de sistemas, añadiendo peso a la misma y, a su vez, adaptar los sistemas del aeropuerto para que esta funcione. Esto significa que los cambios dentro del aeropuerto y las aeronaves son mucho mayores, con una integración más compleja.

Asimismo, habiendo analizado los productos ofrecidos por los demás competidores, se observa que existan ciertas características que no llegan a desarrollar, lo cual puede generar desinterés por parte del aeropuerto de Madrid y el correspondiente proveedor de servicios de navegación aérea ENAIRE.

7.1.2.5 Fuerza 5 - Rivalidad entre competidores existentes

Las empresas reconocidas dedicadas al desarrollo y diseño de herramientas de gestión de tráfico aéreo son pocas en el mundo. Por tanto, existe un oligopolio, habiendo más aeropuertos con gran cantidad de tráfico que empresas dedicadas al control del mismo. Además, la aplicación de este tipo de sistemas se está llevando a cabo en aeropuertos de alto tráfico con un potencial de crecimiento significativo como son los de Oriente Medio, o por aeropuertos donde se opera a niveles muy cercanos a su máxima capacidad como es Heathrow Airport en Londres. Por tanto, existe todavía mercado explorable para las empresas ya consolidadas y nuevas empresas.

Por otro lado, cabe destacar que la mayoría de estos suministradores de productos de navegación aérea ya conocen el mercado y tienen bien definidos sus clientes, por tanto, aunque existe cierta rivalidad entre ellos, cada uno tiene su espacio dentro del mercado.

7.2 Plan de acción

El plan de acción del proyecto está conformado por todo aquello que sirva como hoja de ruta para alcanzar las metas y objetivos marcados. A continuación, se presentan los objetivos y metas, los resultados esperados e indicadores para medir dichos éxitos, suposiciones, limitaciones y. por último el cronograma e hitos principales del proyecto.

7.2.1 Objetivos y metas

Entre los objetivos estratégicos marcados para este proyecto se encuentran los siguientes:

- Llevar a cabo de forma exitosa el proyecto de implementación de un sistema avanzado de control de vehículos por superficie en aeropuertos, de modo que se obtengas los beneficios esperados.
- Ejecutar dicho proyecto en tiempo y coste según el planificado.
- Crear una fidelidad con los clientes, en este caso el aeropuerto de Madrid, AENA y ENAIRE.
- Generar interés por otros posibles clientes, para poder aplicar dicho producto en sus aeropuertos.

Una vez definidos dichos objetivos, se proponen las siguientes metas y estrategias para cada uno de ellos.

Tabla 5: Objetivos, metas y estrategias

Objetivo	Meta	Estrategia
Llevar a cabo de forma exitosa el proyecto	Completar el producto e integrarlo en el aeropuerto de Madrid de manera segura.	Monitorizar periódicamente los avances del proyecto, considerando los entregables y tareas a realizar. Asimismo, corregir posibles desviaciones.

Ejecutar el proyecto en tiempo y coste	Que el producto se ejecute siguiendo el cronograma establecido y que no se desvíe en gran medida de los costes estimados.	Controlar la estructura de desglose de trabajo, analizando que las tareas y paquetes a realizar llevan el tiempo y el coste indicado.
Crear una fidelidad con los clientes	Hacer que el cliente confíe en el producto y proyecto para realizar el mantenimiento y posibles futuras actualizaciones	El proyecto debe ser certificado por los organismos regulatorios de manera satisfactoria, viendo que se trata de un producto conforme a los requisitos y seguro.
Generar interés en otros aeropuertos	Nuevos aeropuertos apuesten por el producto y sea posible su integración en los mismos.	Demostrar los beneficios obtenidos en el aeropuerto de Madrid, asegurando que es posible en otros aeropuertos.

Para poder cumplir con los objetivos descritos anteriormente, se deben diseñar acciones concretas con metas y entregables que deben cumplir. Para ello, se presentan los siguientes planes de acción relativos al proyecto:

- **Plan de acción 1:** firma del contrato con el aeropuerto de Madrid para la realización del proyecto.
    - **Acción 1.1.:** crear un acuerdo con el alcance y requisitos deseados por el cliente, además las fechas de comienzo y finalización del mismo.
    - **Meta 1.1.:** firma de un contrato entre el cliente y el proyecto.
    - **Entregable 1.1.:** contrato firmado por ambas partes interesadas.

- **Plan de acción 2:** inicio, ejecución y finalización del producto
    - **Acción 2.1.:** establecer los tiempos para realizar cada actividad necesaria para la compleción del producto. Asimismo, establecer los responsables de las distintas áreas: safety, test, ingeniería, software, arquitectura.
    - **Meta 2.1.:** definición de responsables para cada área técnica y definición de tiempos para cada tarea. Supervisión periódica, analizando que los requisitos y alcance se está cumpliendo.
    - **Entregable2.1.:** estructura de desglose de trabajo definiendo tiempos, actividades y responsables.

- **Plan de acción 3:** integración del sistema con los demás sistemas en el aeropuerto.
    - **Acción 3.1.:** obtener todas las entradas disponibles del aeropuerto para la correcta visualización de la posición de las aeronaves.
    - **Meta 3.1.:** ver de manera exacta y precisa la posición de las aeronaves sobre la interfaz de usuario, analizando que el software creado funciona acorde a las especificaciones.
    - **Entregable 3.1.:** sistema integrado en la torre de control con los demás equipos.

- **Plan de acción 4:** Cierre de contratos con los proveedores del hardware
    - **Acción 4.1.:** definir los proveedores que suministraran las pantallas, cableado y servidores para la correcta operación del proyecto, acordando contratos de suministro, mantenimiento y garantías.
    - **Meta 4.1.:** Tener los contratos con los proveedores necesarios para la correcta ejecución del proyecto firmados.
    - **Entregable 4.1.:** Contratos firmados con los proveedores.

- **Plan de acción 5:** obtención de permisos para operar.
    - **Acción 5.1.:** solicitar la certificación conforme a los estándares de la Comisión Europea y EASA.
    - **Meta 5.1.:** obtener la certificación favorable para la operativa segura del sistema.
    - **Entregable 5.1.:** Certificación de operabilidad del sistema.

7.2.2 Resultados esperados e indicadores

Para el presente caso de negocio, a continuación, se muestran los resultados esperados en la siguiente tabla. Para ello, se ha añadido la situación inicial de dicha acción y los indicadores de seguimiento a tener en cuenta para realizar el correcto control.

Tabla 6: Resultados esperados e indicadores asociados

Objetivo	Acción	Situación Inicial	Resultado Esperado	Indicadores de seguimiento
Obtención del contrato con el aeropuerto de Madrid	Negociar con el aeropuerto de Madrid sobre la idoneidad de este producto en su aeropuerto, analizando alcance y requisitos.	Presentar al aeropuerto de Madrid, AENA y ENAIRE el producto que se ofrece y las mejoras que la herramienta ofrece.	Firma de la concesión, indicando alcance y requisitos, además de fechas de inicio y finalización del mismo.	Informes semanales del avance de las negociaciones.
Inicio, ejecución y finalización del producto	Establecer los periodos para cada actividad necesaria para la correcta ejecución del proyecto. Se establecerán los responsables de cada área.	Se cuenta con la normativa de OACI y Eurocontrol acorde a sistema A-SMGCS, la cual debe contemplar el sistema a diseñar. Se tiene un conocimiento previo del aeropuerto de Madrid.	Ejecución del proyecto en el tiempo establecido, mostrando las claras ventajas que presenta.	Informes semanales y mensuales del estado de cada área del proyecto.
Integración con el resto de los sistemas aeroportuarios	Con el producto ya desarrollado, hacerlo funcionar con el resto de los sistemas aeroportuarios.	Se conoce los sistemas presentes en el aeropuerto a los que se deberá adaptar la señal para que dicha información se integre correctamente	La posición de las aeronaves y vehículos se obtiene correctamente y se muestra de manera precisa en el sistema.	Reuniones bimensuales describiendo el estado de los sistemas en el aeropuerto en caso de necesitar cambios.
Firma de contratos con proveedores	Definir los proveedores que proporcionarán las pantallas táctiles para mostrar la información al controlador y los servidores encargados de realizar todos los cálculos.	Estudio del mercado con los principales proveedores disponibles para dichos equipos.	Firma de contratos donde se indiquen los suministros, mantenimientos y garantías a ofrecer.	Informes mensuales del avance de los procesos de licitación y de firma de contratos.

7.2.3 Suposiciones

Para el presente proyecto, se destacan las siguientes suposiciones preliminares:

- Como explicado con anterioridad, las rutas de rodaje existentes son mantenidas y no se cambiará la forma de operar en Madrid. Asimismo, los controladores de tierra mantendrán sus zonas de operación, evitando que haya posibles problemas por desconocimiento de Hot Spots.
- La tecnología de las luces y señalética en el entorno aeroportuario de Madrid puede ser controlado de manera remota, lo cual permite automatizarlo mediante sistemas electrónicos.
- El proyecto se financia por inversores externos, ya sea el propio aeropuerto de Madrid; AENA o ENAIRE.
- El producto se llevará a cabo siguiendo las normas y requisitos impuestos en el manual "*DOC 9830 – Advanced Surface Movement Guidance and Control System Mnaual*" de la OACI y el documento de EUROCONTROL llamado "*Specification for Advanced-Surface Movement Guidance and Control System (A-SMGCS) Services*".
- Antes de ponerse en operación, el producto deberá llevar a cabo las consecuentes acciones de certificación por organismos supranacionales como la EASA y la Comisión Europea.

7.2.4 Limitaciones

En este proyecto existen una serie de limitaciones que dificultarán ciertas tareas, sin embargo, esto no implica ser un impedimento para afrontar el proyecto con garantías de éxito. A continuación, se presentan las limitaciones más significativas:

- Limitada experiencia en comparación a otras empresas del sector con años y proyectos relacionados al control del tráfico aéreo.
- Desconocimiento del procedimiento de taxi de cada aerolínea o pilotos concretos, teniendo las mismas aerolíneas normas para sus aeronaves al respecto. No obstante, con el tiempo y la realimentación de datos sobre el sistema, este se irá ajustando a tiempos más precisos.
- Ciertos problemas que obliguen a cerrar calles de rodaje, como pueden ser objetos que puedan dañar la integridad de las aeronaves, deberán ser introducidos manualmente al sistema. El sistema podrá reconocer que las aeronaves están evitando ciertos recorridos, pero no identificará cual es el problema exacto. Asimismo, el sistema, a partir de un cierto número de operaciones que evitan una calle de rodaje concreta, este la cerrará y programará todas las rutas sin contenerla.
- El diseño del sistema está limitado a las condiciones de continuidad, disponibilidad, integridad y fiabilidad, mostrando que es seguro de operar y que no va a causar errores.

- El diseño queda limitado al aeropuerto de Madrid, en caso de quererlo extrapolar a otros aeropuertos, un estudio previo del mismo es necesario para poder integrarlo en la herramienta.

7.2.5 Cronograma e hitos principales

Para poder definir el cronograma del proyecto, este se va a dividir entre las distintas tareas a realizar, las cuales van a venir dadas por la estructura de desglose de trabajo (EDT). En esta EDT se definen los tiempos por actividad, el responsable de cada tarea y se definirán las personas necesarias para la ejecución de dicho proyecto.

Como se comenta con anterioridad, de acuerdo a los intereses por parte del aeropuerto de Madrid para poner en operación dicho sistema, el número de personas puede variar, lo cual tiene un efecto directo sobre el tiempo que se tarda en realizar dichas actividades. No obstante, a continuación, se presenta un esquema a alto nivel que permite visualizar a grandes rasgos el enfoque del proyecto.

En primer lugar, se describen las principales áreas técnicas para la correcta ejecución del proyecto [26]. Con ellas, ya se pueden definir los principales hitos para el correcto desarrollo del proyecto. Una vez realizado lo anterior, se presenta el cronograma del proyecto, teniendo en cuenta las principales tareas del mismo, sin embargo, considerando la EDT, estos se considerarán paquetes de trabajo que contendrán a su vez subtareas de menor tamaño, consiguiendo así un control más fácil del estado del proyecto.

7.2.5.1 Áreas Técnicas

7.2.5.1.1 Equipo de Ingeniería

El equipo de ingeniería será el equipo encargado de definir y describir cómo debe ser el correcto funcionamiento del sistema. Para ello, mediante reuniones habituales con el cliente, se discutirán las posibilidades del sistema y posteriormente, ellos se encargarán de definir los requisitos, necesarios para el equipo de software, encargado de desarrollar el sistema.

Asimismo, el equipo de ingeniería será el equipo que estará en contacto con los controladores, viendo como realizan su trabajo, para ajustar el sistema a las necesidades del día a día. Además, durante el periodo de pruebas, esta área será la encargada de analizar posibles errores, identificando cual es el requisito por el que falla dicho sistema. De este modo, el proyecto queda definido por un alcance y unos requisitos bien definidos.

Para la correcta comprensión del sistema, se dividirán los requisitos a nivel de sistema, describiendo como debe funcionar el sistema, mostrando que ocurre si cierta acción se realiza en el mismo. Y, por otro lado, se describirán requisitos a nivel de subsistema, siendo requisitos de más bajo nivel, señalando como debe comportarse el software a nivel de señales.

De este modo, el equipo estará conformado por Ingenieros Aeroespaciales principalmente.

7.2.5.1.2 Equipo de Software

El equipo de software será el encargado de codificar el producto a partir de los requisitos definidos por el equipo de ingeniería. Ellos se encargarán de seguir de manera ordenada y organizada los distintos requisitos y tener un control de versiones. El equipo de software funcionará publicando diversas versiones del software quedando reflejados los cambios realizados. Cada versión será testada por el equipo de pruebas, analizando posibles fallos en el sistema. Por ello, el equipo de software debe conocer que fallos se han corregido y que modificaciones se han realizado, para que, en caso de que dicha modificación genere otros problemas, se pueda volver a analizar las versiones anteriores.

En este caso, el equipo estará conformado por ingenieros informáticos con alto conocimiento en codificación y experiencia en aviación.

7.2.5.1.3 Equipo de Pruebas

Una vez el software se encuentre lo suficientemente completo, el equipo de pruebas será el encargado de realizar los tests oportunos. Para ello, comprobarán que todos los requisitos se cumplen acordemente. En caso contrario, se comunicará tanto al equipo de software y el equipo de ingeniería. Serán estos dos equipos quienes decidirán entre ellos si se trata de un problema en los requisitos o un problema en el propio software.

El equipo de pruebas también será el encargado de mostrarle a los controladores cómo funciona dicha herramienta y realizar el entrenamiento oportuno. Asimismo, al igual que el equipo de software, este equipo deberá llevar un control sobre los fallos comunicados ya que deberán ser probados en versiones posteriores analizando que estos mismos se han corregido acorde a los requisitos.

Para que esta tarea se haga de manera satisfactoria, el equipo de pruebas recopilará todos aquellos procedimientos utilizados para probar el software conforme a los requisitos. De este modo, en caso de realizar pruebas formales con cliente, estos tests se podrán realizar de manera más rápida y automatizada.

El equipo de pruebas estará conformado por Ingenieros Aeroespaciales, pudiendo ser útil incluir controladores aéreos al equipo también.

7.2.5.1.4 Equipo de Arquitectura

Este equipo será el encargado de estudiar la arquitectura necesaria de hardware para poder integrar dicho sistema entre los demás sistemas del aeropuerto. Para ello, deben conocer de manera eficaz cuales serán todas las señales de entrada sobre el sistema para que el software funcione acorde a estas mismas.

Por otro lado, este equipo también se encargará de contactar con los proveedores de pantallas y servidores, analizando que el acople es eficaz y no existen errores dados a ruido en las señales. Para ello, se encargarán de construir en un banco de pruebas, el cual también será testado por el equipo de pruebas.

Por último, este equipo se encargará de que la integración una vez se vaya a poner en funcionamiento se lleve de manera apropiada y no haya errores entre el hardware en la torre de control.

El equipo de arquitectura estará compuesto por distintos perfiles como ingenieros electrónicos, ingenieros informáticos e ingenieros aeroespaciales.

7.2.5.1.5 Equipo de Safety y Calidad

El equipo de Safety y Calidad será el equipo encargado de comprobar que todo el proceso se está ajustando a la normativa aplicable y que, de este modo, se estén llevando los pasos correctos para que el sistema pueda ser certificado sin ningún problema. Para ello, estudiarán los manuales de OACI y Eurocontrol, así como el correspondiente a la certificación de EASA, y lo compararán con los requisitos de sistema descritos por el equipo de ingeniería. En caso de haber discrepancias, el equipo de safety se comunicará con el equipo de ingeniería para corregir posibles errores.

Asimismo, este equipo será el encargado de realizar todos los procesos análogos a la certificación, por ello, deberán asegurar un control y organización dentro del proyecto, además de que indicarán las pautas a seguir por el resto de equipos para que el proyecto sea completado de manera satisfactoria.

El equipo de safety y calidad estará formado por perfiles ingenieriles, preferiblemente aeroespaciales con conocimientos de certificación.

7.2.5.1.6 Oficina de Gestión de Proyectos

La oficina de gestión de proyectos, conformado en casi su totalidad por el Project Manager, será la encargada de asegurar que el proyecto se está ejecutando tanto en tiempo, como en coste acordado inicialmente. Además, asegurará que el proyecto está coordinado tanto internamente como externamente, observando que los distintos equipos internos se están comunicando correctamente para que el proyecto se esté ejecutando de manera correcta. También mantendrá conversaciones con los proveedores y con el cliente, conociendo posibles conflictos de interés, resolviéndolo en el menor tiempo posible.

Asimismo, el Project Manager será el máximo responsable del proyecto, siendo así la cara visible y punto de contacto en caso de necesidad. Cualquier problema o riesgo que pueda afectar al proyecto deberá serle comunicado, y será él la persona indicada para decidir el criterio a seguir.

7.2.5.2 Hitos principales

Los hitos principales del proyecto presentado siguen una estructura similar a numerosos proyectos donde es necesario el desarrollo de un producto, con una serie de pruebas declarando que el producto funciona acorde a unos requisitos, posteriormente, una vez el producto se considera finalizado, una certificación acorde a estándares definidos y, por último, la puesta en funcionamiento del producto.

De este modo, a continuación, se presentan los hitos principales del proyecto:

7.2.5.2.1 Hito 1 – Creación de una versión prototipo de la herramienta

En primer lugar, se realiza una versión prototipo y preliminar de la herramienta, la cual sea capaz de demostrar las capacidades y el potencial que tiene dicho sistema para mejorar la operativa en el aeropuerto. El objetivo de este hito es convencer de la necesidad de dicha herramienta al cliente, sirviendo como punto de inicio de conversaciones para contratar el producto. Asimismo, esto permite alcanzar el interés de posibles inversores o futuros clientes.

En este hito, seguir unos requisitos no es tan primordial como buscar la idea general del producto. Para ello, el equipo de ingeniería se reunirá con el equipo de software para solicitar las necesidades del sistema de manera global. Al mismo tiempo, el Project Manager tratará de

convencer a los posibles interesados a conocer el proyecto, demostrando las capacidades que tiene y los beneficios que trae al aeropuerto de Madrid, en este caso.

7.2.5.2.2 Hito 2 – Formalización del contrato y definición de requisitos

Tras haber convencido a los interesados del proyecto, el cliente, junto al Project Manager, firmarán el Plan para la Dirección del Proyecto, donde se definirán todos aquellos asuntos importantes relativos al alcance del mismo proyecto, el cronograma y los costes estimados del mismo. Este documento resulta esencial para demostrar que el proyecto ha cumplido las expectativas y para poder controlar el proyecto en caso de desviación.

Una vez firmado el documento anterior, el equipo de ingeniería se reunirá con los controladores y equipo técnico del cliente para revisar las necesidades del proyecto, además de adecuar los requisitos del software. Mientras tanto, el equipo de software, en base a la versión preliminar creada para el hito 1, empieza a implementar los cambios y requisitos añadidos por el equipo de ingeniería.

Por otro lado, el equipo de arquitectura empezará a estudiar y analizar como entran las señales de otros sistemas presentes en el aeropuerto al sistema planteado. El objetivo es armonizar este tipo de señales ya existentes con las que se utilizarán en la herramienta para que la integración sea fácil. Al mismo tiempo, estos irán comunicando dichas señales e información de relativa importancia al equipo de software, para que lo tengan en cuenta a la hora de diseñar el software.

7.2.5.2.3 Hito 3 – Entrega de versiones intermedias del Software

El equipo de software entregará una versión conforme a los avances que hayan podido realizar. Una vez entregada, empezarán a mejorarla añadiendo más funciones y corrigiendo posibles errores detectados.

Con la versión entregada, el equipo de pruebas realizará los tests necesarios para confirmar que el software funciona conforme a la especificación. En caso contrario, estos lo comunicarán, y el equipo de ingeniería analizará dicho problema. En caso de ser un problema de requisito, estos mismos se encargarán de corregir el requisito, sin embargo, en caso contrario, se lo harán saber al equipo de software para que mejoren dicha funcionalidad.

7.2.5.2.4 Hito 4 – Entrega de versión final del Software

Una vez se acerque la fecha de entrega y se cumplan con los requisitos marcados por el cliente, el software será entregado como versión final. El equipo de pruebas lo testará en el banco de pruebas, revisando que cumple las especificaciones correspondientes y que no existen errores fatales. Cabe destacar que, para este momento, el equipo de arquitectura ya debería haber entregado una maqueta del hardware a utilizar, siendo probado también y comprobando que el software y el hardware se integra de manera correcta.

Los posibles errores y fallos encontrados serán registrados, comentándoselos al cliente. De este modo, los equipos de software y de ingeniería trabajarán por resolverlos para futuras versiones.

Con la entrega del software, este pasará a ser revisado por el cliente viendo que está conforme a lo especificado en los requisitos marcados. Para ello, el equipo de pruebas realizará las pruebas formales con el cliente, y con la ayuda del equipo de ingeniería, se demostrará que los posibles fallos presentes no suponen un riesgo elevado, siendo los beneficios de la herramienta muy superiores.

7.2.5.2.5 Hito 5 – Conformidad del cliente y certificación del Software

Una vez el paso anterior se ha terminado, el estado del cliente es conforme con el producto desarrollado, por tanto, se procede a realizar las pruebas de certificación. Para ello, el equipo de safety y calidad han ido documentando el proceso llevado durante el proyecto de manera acorde a las necesidades de certificación. De este modo, durante la certificación se demostrará que los pasos seguidos son correctos y que cumple con la normativa vigente.

7.2.5.2.6 Hito 6 – Puesta en operación

Tras recibir un resultado positivo de las pruebas de certificación, se procede a integrar el sistema en el entorno aeroportuario. Para ello, se utilizan momentos de poco tráfico donde los efectos en caso de fallo puedan ser bajo, teniendo siempre un sistema de respaldo en caso de ser necesario. Durante estos cortos periodos, los controladores realizan múltiples pruebas, confirmando que la herramienta funciona correctamente y trae los beneficios esperados. Los posibles fallos encontrados durante este periodo serán supervisados por el equipo de ingeniería. Esta acción es importante ya que los controladores pueden exigir algo que no se encuentra dentro de los requisitos una vez en operación y, por tanto, el equipo de ingeniería debe demostrar que esa funcionalidad se encuentra fuera del alcance.

Si el resultado vuelve a ser positivo y se confirma que el producto funciona como se esperaba, se decide una fecha para integrarlo definitivamente y, con ello, se integra entre los sistemas del aeropuerto.

7.2.5.2.7 Hito 7 – Formalización del cierre del proyecto

Una vez se haya integrado dicho equipo entre los sistemas aeroportuarios, el Project Manager firmará con el cliente el cierre del proyecto, cumpliendo con el alcance marcado inicialmente. Cabe destacar que, en ocasiones, el cliente puede solicitar el mantenimiento de la herramienta, necesitando que haya equipos encargados de mejorarla en caso de cambios sustanciales en el aeropuerto, o corregir aquellos posibles errores que quedaron pendientes inicialmente. Asimismo, también, a medida que se utiliza la herramienta, los controladores pueden exigir funcionalidades o correcciones que mejoren aún más la herramienta. Por tanto, formalizar un contrato de este tipo puede resultar beneficioso para ambas partes.

7.2.5.3 Cronograma

Habiendo definido los distintos hitos principales del proyecto, se presenta el cronograma propuesto para la creación de una herramienta A-SMGCS de avanzado nivel como la descrita con anterioridad enfocada al aeropuerto de Madrid. Como ya se menciona en previos apartados, estos tiempos pueden variar según los intereses por parte del cliente y la cantidad de trabajadores presentes en el proyecto.

A continuación, se presenta el diagrama de Gantt resumido, sin embargo, en el Anexo 1, se puede encontrar el diagrama de Gantt con todas las actividades definidas en la EDT.

ID	EDT	Nombre de la tarea	Comienzo	Fin	Sep -23	Oct -23	Nov -23	Dec -23	Jan -24	Feb -24	Mar -24	Apr -24	May -24	Jun -24	Jul -24	Aug -24	Sep -24	Oct -24	Nov -24	Dec -24	Jan -25	Feb -25	Mar -25	Apr -25	May -25	Jun -25	Jul -25	Aug -25
1	1	Creación prototipo de la herramienta	01/09/2023	21/12/2023																								
10	2	Formalización del contrato/requisitos	22/12/2023	16/08/2024																								
23	3	Entrega de versiones intermedias	19/08/2024	14/03/2025																								
31	4	Entrega de versión final del SW	17/03/2025	30/05/2023																								
40	5	Conformidad y certificación del SW	05/05/2025	27/06/2025																								
46	6	Puesta en operación	30/06/2025	11/08/2025																								
54	7	Formalización del cierre del proyecto	12/08/2025	12/08/2025																								

Figura 19: Diagrama Gantt del proyecto

Capítulo 8. Primer Hito

Como se explica con anterioridad, el objetivo del primer hito es conseguir un prototipo, que permita visualizar el producto que se pretende desarrollar. Para esto, se va a hacer uso del lenguaje de programación de Python, que, mediante librerías de uso libre, permita alcanzar los objetivos marcados para este prototipo.

Cabe destacar que, este primer hito solo representa una versión inicial, la cual puede servir para demostrar los posibles beneficios que presenta utilizar herramientas como esta para mejorar el flujo de aeronaves en tierra en el aeropuerto de Madrid. Asimismo, es importante señalar que, se trata de una versión preliminar y, por tanto, las funciones son limitadas, y puede generar posibles errores, por ello, no puede ser integrada en la operativa habitual del aeropuerto al estar todavía lejos del alcance del proyecto.

8.1 Objetivos del hito 1

Los objetivos del hito 1 se basan principalmente en generar un código en Python que permita calcular la ruta más corta en base a distancia y tiempo, considerando pequeños posibles conflictos. De esta forma, se consigue analizar las posibles mejoras en tiempo de rodaje teniendo en cuenta aquellas circunstancias que puedan darse durante las operaciones.

En esta versión inicial, los conflictos se generarán manualmente, simulando posibles trayectorias entre varias aeronaves que tendrán problemas entre ellas en la rodadura. Centrándose en una de las aeronaves, la cual tiene una prioridad menor, esta se verá obligada a modificar su ruta o realizar una parada en el punto de parada, dependiendo que le hará perder un tiempo de rodaje para evitar dicho conflicto. De este modo, se estudiará otra posible ruta que le haga reducir el tiempo de rodaje comparándolo así con el caso de mantener la ruta inicial, donde se vería obligada a parar, dejar pasar a la otra aeronave y, después, proseguir con su ruta.

Por otro lado, se va a hacer un modelo específico para ciertas operaciones del aeropuerto, no incluyendo todas las calles de rodaje, ya que se encuentran fuera del alcance del mismo. De este modo, se incluirá la configuración Norte y solo destacando las operaciones de llegadas. No obstante, querer incluir las demás configuraciones resulta sencillo ya que solo se debe actualizar la lista de nodos y vecinos, consiguiendo así un programa mucho más amplio.

Asimismo, los resultados obtenidos se pretenden proyectar por pantalla, pudiendo visualizar las rutas a seguir por las aeronaves desde el punto inicial hasta su punto final, así como la ruta alternativa.

8.2 Análisis técnico

El análisis técnico pretende describir todos aquellos detalles necesarios para la elaboración de esta primera versión, la cual, como se comenta con anterioridad, será una versión preliminar con funciones muy limitadas pero que, permitirá ver los posibles beneficios de herramientas como estas. A continuación, se menciona todo lo relacionado con el prototipo a desarrollar, un esquema funcional del mismo, las herramientas utilizadas para su creación, el procedimiento seguido y finalmente los resultados obtenidos.

8.2.1 Prototipo

El prototipo, como se menciona con anterioridad, se trata de una herramienta en versión preliminar, que tiene como objetivo final demostrar los beneficios que puede traer incorporar un A-SMGCS de nivel 4 en el aeropuerto de Madrid. Para ello, en base a una serie de nodos predefinidos, correspondiendo a las intersecciones entre las distintas calles de rodaje del aeropuerto, se construye una red de nodos y relaciones entre los mismo.

Las relaciones entre nodos sirven para definir si la calle de rodaje es de dos direcciones o de una únicamente y, a su vez, indicar que nodos vecinos le corresponden a cada nodo. Asimismo, es importante señalar que, mediante esta relación entre nodos se define el tiempo transcurrido entre un nudo y el siguiente.

Cabe destacar que, en esta versión preliminar, se supone que el movimiento de la aeronave por la calle de rodaje es constante, manteniendo la misma velocidad durante todo el taxi. Además, no se consideran las distintas rampas de las plataformas ya que añade un volumen muy alto de cálculos que puede tener consecuencias en tiempo de cálculo. Sin embargo, en estas, al depender principalmente de la posición de estacionamiento, el tiempo y las posibles desviaciones para reducir el tiempo de taxi es prácticamente despreciable. No obstante, de cara a la versión final, estas rampas si se tendrán en cuenta, teniendo la ruta completa desde la pista hasta el puesto de estacionamiento y viceversa. En el caso de querer incluir estas rampas a las plataformas, solo será necesario incluir los nodos de las mismas. De este modo, el sistema podrá calcular la ruta más corta desde el punto inicial y final, ambos siendo definidos por el usuario.

8.2.2 Esquema funcional

En la siguiente figura, se presenta el flujograma seguido por el código para obtener las posibles rutas más cortas:

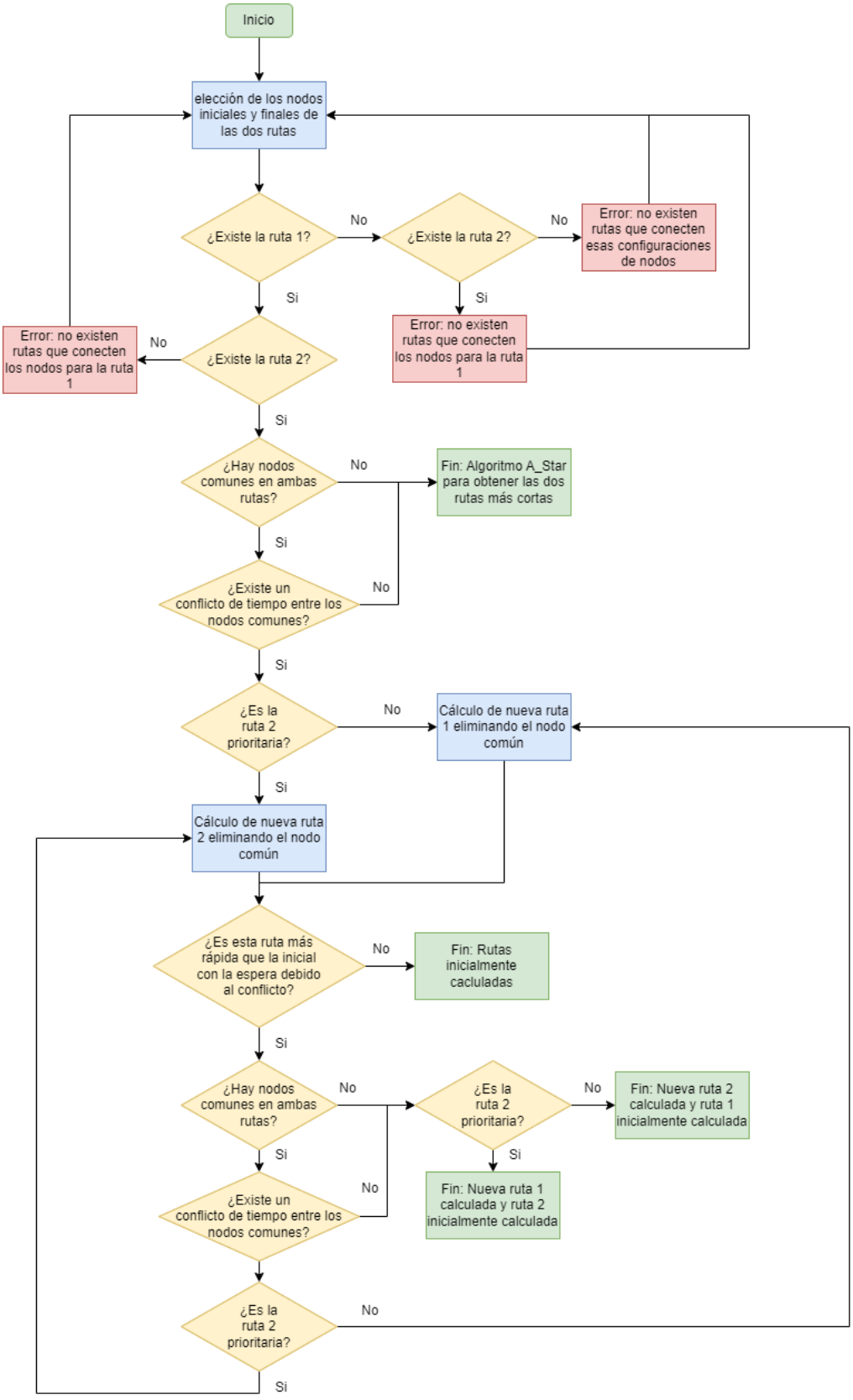


Figura 20: Flujograma del código para el Hito 1

Como se puede observar, el código se inicializa con la elección de los nodos para las dos rutas a estudiar. Cabe destacar que, en esta primera versión o hito, solo se incluyen posibles conflictos entre dos aeronaves únicamente. Por tanto, para ello, el programa debe conocer la posición inicial de la aeronave (nodo inicial) y la posición final (nodo final). Al tratarse de una operación de llegada en configuración norte, se realizará mediante una aeronave que pretende salir de la pista 32L y dirigirse hacia la T4 o T4S y otra aeronave la cual aterriza por la 32R y se dirige hacia el dique sur de la T1. De este modo, un posible conflicto es posible.

Conociendo esto, el programa determina si existe una posible ruta para alcanzar los resultados, es decir, si existen nodos que conecten el punto inicial y el punto final. Para ello, el programa hace uso de un conjunto que determina los nodos vecinos para cada nodo. Esto se realiza tanto para la ruta 1 como para la ruta 2. Si una de las dos rutas no contiene nodos suficientes que unan los dos nodos especificados, un mensaje saldrá por pantalla indicando que la ruta planteada entre esos dos nodos no existe.

Habiendo analizado que la ruta existe se procede a hacer otro análisis fundamental. El sistema debe conocer si existen nodos comunes de ambas rutas, es decir, si ambas aeronaves pasan por el mismo punto en algún momento de su ruta. En el caso de que no tengan nodos comunes, el programa ya puede calcular la ruta óptima, siendo la que deben de llevar ambas aeronaves. En el caso contrario, el programa debe conocer si existe un conflicto en dicho nodo común. Para ello, el programa determina en qué momento temporal la aeronave cruzará por dicho nodo y, en base a lo anterior, si el intervalo de tiempo para ambas rutas es similar, existirá un conflicto. En caso de no haber conflicto, el programa calcula nuevamente las rutas óptimas para cada aeronave y lo imprime como resultado final. En cambio, si el conflicto existe, el programa, en base a que ruta es prioritaria, calcula una nueva ruta para la aeronave no prioritaria. Para ello, en base a la ruta anterior, se elimina el nodo común y se desarrolla una nueva ruta considerando que ese nodo no existe.

Una vez obtenida la nueva ruta para la aeronave no prioritaria, se determina si esta es más rápida que la ruta calculada inicialmente añadiendo la posible espera debido al conflicto a la ruta inicial. Si la nueva ruta es más lenta, se utiliza la ruta calculada inicialmente. Sin embargo, si la nueva ruta es más corta se vuelve a mirar si existen nodos en común y si hay conflicto en ese nodo. En el caso de no haber nodos, o no haber conflictos en él, se escoge esta nueva ruta como ruta a seguir. En el caso de haber conflicto, se vuelve a calcular una nueva ruta sin el nuevo nodo que crea el nuevo conflicto. De este modo, se vuelve a determinar si esta es más rápida que la ruta inicial con la respectiva espera, y se vuelve a determinar si tiene nodos en común con la otra aeronave y si esto supone un conflicto. Este último proceso se realiza hasta alcanzar la ruta más rápida.

8.2.3 Herramientas utilizadas

Para la realización de esta versión preliminar, se utilizan una serie de herramientas para poder alcanzar con éxito con los objetivos descritos anteriormente. A continuación, se presentan dichas herramientas:

- En primer lugar, se emplea Python como lenguaje de programación al ser un lenguaje con numerosas librerías y que, a su vez, resulta fácil de manejar y rápido de computar. Asimismo, la versión de Python utilizada es la 3.8, que, aunque fue publicada en 2019, presenta ventajas de estabilidad y, además, permite utilizar librerías para ciertas aplicaciones.
- Para el cálculo de la ruta óptima en cada caso, se hace uso del algoritmo llamada A-Star, la cual, permite definir un conjunto de nodos, los cuales tienen dependencias entre sí. De este modo, definiendo el punto inicial, el punto final, y los tiempos de demora entre nodos, permite calcular la ruta más corta entre todas las opciones. En el caso presentado, los nodos son los distintos puntos de las calles de rodaje donde las aeronaves podrán tomar distintas decisiones con respecto al camino a seguir. Cabe destacar que, también existen nodos donde solo haya una dependencia o vecino, debido a la forma de circular por el aeropuerto de Madrid, no obstante, se ha considerado como un nodo ya que puede haber puntos de parada o de cruce con otras aeronaves.
- El mapa utilizado para representar las calles de rodaje y sobrescribir encima la ruta óptima es un mapa de libre uso, siendo muy útil para aplicaciones donde información cartográfica es importante. Este se llama OpenStreetMap y sería ideal que en un futuro la herramienta se integre plenamente en las coordenadas proporcionadas por la página web mediante una API (Application Programing Interface), o incluso, tener una representación gráfica única, que facilite la visualización de las aeronaves por el entorno aeroportuario. No obstante, como idea inicial se ha utilizado imágenes obtenidas de dicho mapa y mediante ello sacar la ruta óptima como se explicará más adelante.

8.2.4 Supuestos del hito 1

Para la ejecución del código de este primer hito se realiza una serie de supuestos que permiten simplificar el problema general. Con ello se consigue ver los beneficios para unos casos específicos, no obstante, a medida que el código se vaya mejorando con el tiempo, los resultados alcanzados serán más precisos y exactos, pudiendo observar las verdaderas ventajas operativas de la herramienta.

En el caso que se presenta, se consideran los siguientes supuestos:

- Los conflictos son únicamente entre dos aeronaves.
- Las aeronaves van a una velocidad constante de 47 km/h
- Solo aeronaves en llegadas en configuración Norte
- Las paradas o puntos de espera en caso de conflicto se realizan en el nodo anterior al del conflicto. Al añadir más nodos, las paradas se muestran con mayor claridad y más cerca al punto de espera, pero resulta insignificante en esta versión preliminar el añadir tantos nodos.
- El tiempo de espera en dicho punto es de 40 s.
- Para que se considere conflicto, los aviones pasarán por el mismo nodo entre un margen de 35 s.
- Las aeronaves prioritarias se añadirán manualmente.
- El nodo de salida de pista es seleccionado manualmente.
- Se trabaja en pixeles que luego son convertidos a km.
- No se consideran aquellos puntos de parada obligatorios ya que la herramienta pretende convertir los trayectos más seguros.

8.2.5 Procedimiento

Para la elaboración de esta versión preliminar, se ha hecho uso de una herramienta preliminar generada en Python, que permite sacar los nodos presentes en las calles de rodaje del aeropuerto de Madrid. Posteriormente, conociéndolos distintos nodos, se realiza un análisis de las dependencias de cada nodo. Para ello, hay que destacar que ciertos nodos estarán conectados con otros en una sola dirección, mientras que otros podrán ser en ambas direcciones. Con ello se consigue obtener las distintas rutas que pueden hacer los aviones dentro del área de maniobras. Cabe destacar que, toda esta información se obtiene del AIP y los sistemas utilizados para salir y entrar en pista están explicados en la Tabla 3. Seguidamente, mediante un código, priorizando ciertos parámetros de tiempo o distancia, permite decidir cuál es la ruta más corta para dicha aeronave. Por último, se incluye el mismo procedimiento para una segunda aeronave que pretende ir desde un punto del aeropuerto a otro. Habiendo dicho conflicto, el sistema recalcula el camino más rápido para una de las aeronaves concretas.

A continuación, se presenta en más detalle el procedimiento, explicando el código desarrollado para la ejecución de este hito 1.

8.2.5.1 Paso 1. Obtención de nodos

Como primer paso, se debe desarrollar un código que permita, a partir de una imagen del terreno, obtener la ubicación de los nodos en dicha imagen. Para ello, se desarrolla un código

que, tras pinchar en la posición del nodo en la imagen, el código calcule la posición del nodo. Actualmente, al utilizar una imagen para representar las calles de rodaje del aeropuerto, no es necesario tomar las proyecciones cartográficas, siendo más sencillo utilizar este método. No obstante, de cara a utilizar esta herramienta para futuros proyectos en otros aeropuertos, una mejora de la misma podría ser la utilización de coordenadas en vez de posiciones sobre una imagen.

Para la ejecución de este paso se ha hecho uso del siguiente código:

Código 1: Coordenadas de nodos

```
from PIL import Image, ImageTk
import tkinter as tk

# Cargar el mapa
image = Image.open('AeropuertoBarajas.JPG')

# Crear pestaña
window = tk.Tk()

# Crear un cuadro en la pestaña
canvas = tk.Canvas(window, width=image.width, height=image.height)
canvas.pack()

# Convertir la imagen a un objeto
photo_image = ImageTk.PhotoImage(image)

# Mostrar imagen
canvas.create_image(0, 0, anchor='nw', image=photo_image)

# Función que muestre las coordenadas al clickar
```

```
def show_coordinates(event):
    x, y = event.x, event.y
    print(f"({x}, {y})")


# Conectar los clicks con la imagen
canvas.bind('<Button-1>', show_coordinates)


# Loop
window.mainloop()
```

Como se puede observar, el código se compone de distintas partes. En primer lugar, se importan las librerías necesarias para que el código funcione. Después, se carga el mapa, obtenido de la imagen sacada de la web de OpenStreetMap. Con ello, se crea una sección del código que, al correr el código, se abra una pestaña que contenga dicha imagen. Posteriormente, se crea la función que permita obtener las coordenadas, y las escriba por terminal cuando se pincha en el nodo deseado. Por último, se une la función de pinchar con el ratón y la de mostrar las coordenadas por pantalla.

De esta forma, el resultado es el siguiente:

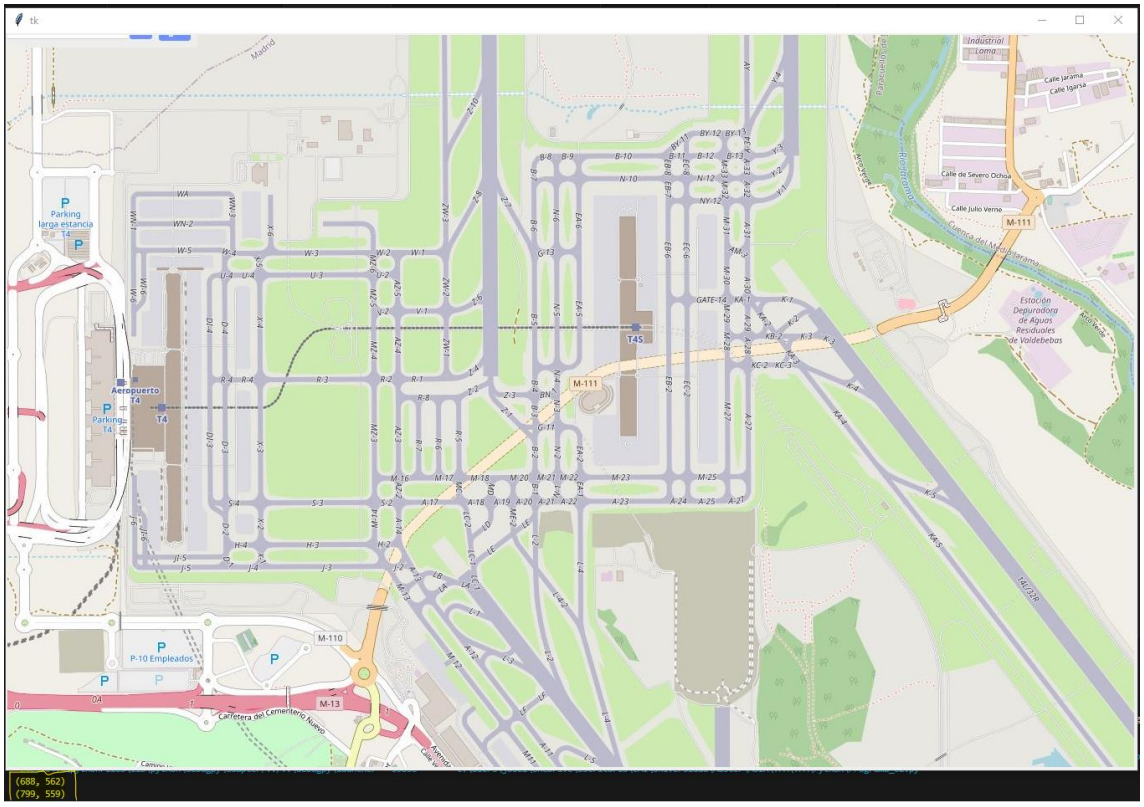


Figura 21: Mapa con coordenadas impresas por pantalla

Por tanto, con este código, se seleccionan, a partir de las rutas seguidas en la Tabla 3, los distintos nodos presentes en el sistema. Principalmente, estos vendrás dados a partir de los cruces entre calles de rodaje y puntos calientes donde la aeronave deba hacer una parada para recibir instrucciones o autorizaciones.

Los nodos son apuntados para su posterior utilización en el código que calcula la ruta más corta entre todas las posibles opciones. De este modo, es esencial saber que nodo corresponde a que, y con qué nodos tiene dependencias, sabiendo así, como es el movimiento de la aeronave en ese punto.

8.2.5.2 Paso 2. Obtención de la ruta más corta

Para la obtención de la ruta más corta, es necesario conocer todos aquellos nodos o puntos conflictivos que están presentes en el aeropuerto. Por tanto, tras haber sacado sus posiciones mediante el código anterior, se definen los nodos definidos. A continuación, se presenta la forma en la que se escribe, y en el Anexo 3 se muestra el código completo:

Código 2: Definición de nodos

```
# Definición de nodos

nodes = {

    1: (1141, 565),          #'R32K5'   Pista 32R

    2: (1050, 528),          #'K5KA4'

    3: (1038, 432),          #'32RK4'

    4: (947, 396),           #'K4KC3'

...

}
```

Cabe destacar que, para mantener un orden y un control de la ubicación de los nodos, se dejan especificadas las calles de rodaje que cortan con dicho nodo. Asimismo, ciertos movimientos curvos, como pueden ser aquellas calles de salida rápida de pista se han considerado en pequeños tramos para poder suponerlo como conexiones de líneas rectas.

Tras ello, se definen los nodos vecinos. Aquí hay que prestar especial atención a la dirección que debe seguir la aeronave ya que dependiendo de si cierta calle de rodaje es de una sola dirección, los vecinos podrán verse solo en un nodo y no en el nodo vecino. Cabe destacar que esto también va a variar dependiendo de la configuración con la que está operando el aeropuerto en dicho momento indicado. A continuación, la sección del código donde se definen las dependencias:

Código 3: Dependencias entre nodos

```
# Definición de los nodos vecinos de cada nodo

neighbors = {

    1: [2, 3, 6],

    2: [4],

    3: [4],

    4: [5, 7],


}
```

Habiendo realizado lo anterior, se procede a definir la ecuación que va a determinar la distancia entre nodos. Esta puede verse modificada en función de las necesidades de cada momento, pudiendo depender en la distancia o en el tiempo o las prioridades o necesidad de paradas obligatorias en ciertas rutas. En el código, esta ecuación quedaría definida del siguiente modo:

Código 4: Ecuación para determinar la distancia entre nodos

```
# Definición de la ecuación que obtiene el tiempo/distancia entre
nodos

def distance(node1, node2):

    x1, y1 = node1

    x2, y2 = node2

    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
```

Para el cálculo de la ruta más corta, se hace uso del algoritmo A-Star, el cual sabe distinguir, mediante los nodos vecinos y la distancia entre nodos, que ruta debe seguir para alcanzar el resultado óptimo. Para ello, el algoritmo comprueba entre el punto inicial y el nodo estudiado si la ruta escogida es la más corta. Posteriormente, pasa al siguiente nodo, y vuelve a comprobar que ese punto sea el más corto comparandolo desde el punto inicial. En la siguiente parte del código es donde se refleja el algoritmo A-Star:

Código 5: Algoritmo A-Star

```
# Definición del algoritmo A*

def a_star(start, goal):

    # Inicialización de la frontera mediante el nodo inicial

    frontier = [(0, start)]

    # Inicialización de conjunto de nodos visitados

    visited = set()

    # Inicialización del diccionario de padres

    parents = {start: None}

    # Inicialización del diccionario de coste de ruta
```

```
    path_costs = {start: 0}


    while frontier:
        # Coger el nodo fronterizo con coste menor
        _, current = heapq.heappop(frontier)
        # Si el nodo escogido es el correcto, parar
        if current == goal:
            break
        # En caso contrario, expandir el espectro de búsqueda
        for neighbor in neighbors[current]:
            # Calcular el coste de la ruta desde el punto inicial
hasta el vecino desde el nodo escogido
            new_cost = path_costs[current] +
distance(nodes[current], nodes[neighbor])
            # Si el vecino no se ha calculado todavía, o si el nuevo
coste es más bajo que el anteriormente calculado
            if neighbor not in visited or new_cost <
path_costs[neighbor]:
                # Actualizar el coste de la ruta
                path_costs[neighbor] = new_cost
                heuristic = distance(nodes[neighbor], nodes[goal])
                priority = new_cost + heuristic
                # Añadir el vecino a la frontera
                heapq.heappush(frontier, (priority, neighbor))
                # Añadir el padre del vecino
                parents[neighbor] = current
                # Marcar el vecino como visitado
                visited.add(neighbor)


    # Reconstruir el camino desde el punto inicial hasta el final
```

```
    path = []
    current = goal
    while current is not None:
        path.append(current)
        current = parents[current]
    path.reverse()

    # Return el coste total y la ruta
    return path, path_costs[goal]
```

Como se puede observar, dicho algoritmo parte de dos puntos, el punto inicial y el punto final. Para ello, mediante una línea de código, se obtiene la ruta y el coste de dicha ruta. Cuando se menciona el coste de la ruta, este se refiere al tiempo o distancia que debe recorrer la aeronave para realizar dicho trayecto. La línea de código es la siguiente:

Código 6: Definición del nodo inicial y final

```
# Definición del nodo inicial y final
path, cost = a_star(31, 61)
```

Por último, se realiza la parte del código donde se representa y se imprime por pantalla la trayectoria y el coste de esta. Para ello, se utiliza la imagen empleada para obtener las coordenadas de los nodos y, en ella, se representa la trayectoria óptima. Además, en la terminal, se imprimen los nodos por los que pasa la ruta y el coste de la misma. A continuación, se muestra el código empleado:

Código 7: Representación de la ruta y el coste final

```
# Imprimir la ruta y el coste final
print(f"Shortest path: {path}")
print(f"Total cost: {cost}")
```

```
# Cargar la imagen

bg_image = plt.imread("AeropuertoBarajas.JPG")


# Crear un gráfico con la imagen de fondo

fig, ax = plt.subplots()

ax.imshow(bg_image)


# Dibujar los nodos

for node in path:

    ax.scatter(nodes[node][0], nodes[node][1], s=10, color='blue')


# Unir puntos con líneas

for i in range(len(path)-1):

    node1 = path[i]

    node2 = path[i+1]

    ax.plot([nodes[node1][0], nodes[node2][0]], [nodes[node1][1],
nodes[node2][1]], color='blue')


# Imprimir gráfico

plt.show()
```

De este modo, la representación quedaría de la siguiente forma:

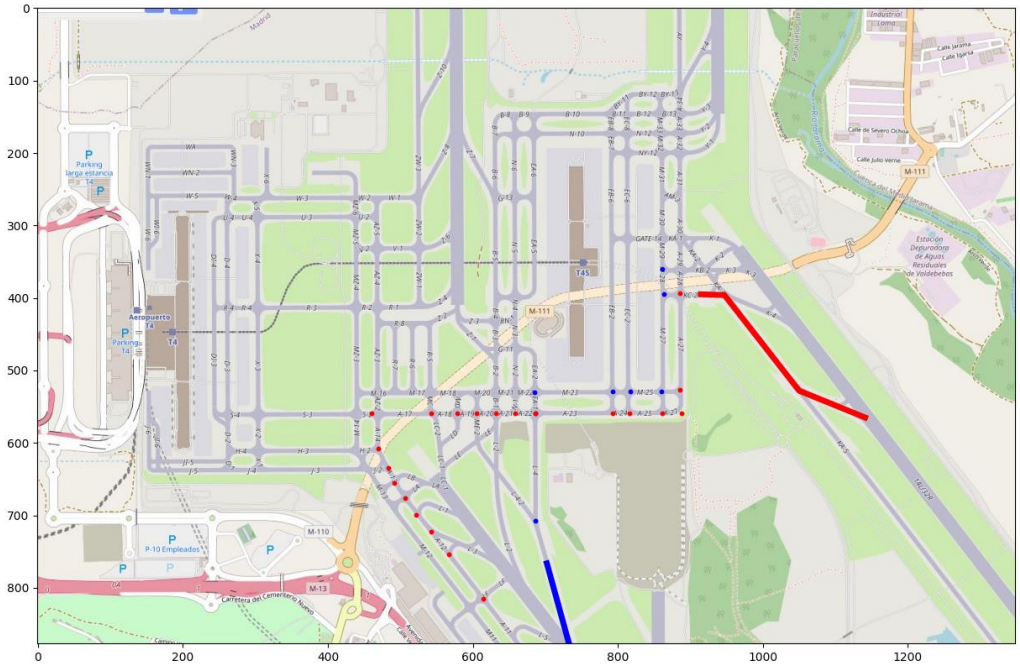


Figura 22: Representación gráfica

Como se observa en dicha figura de ejemplo, se observan dos rutas, siendo la roja la prioritaria la cual seguirá el camino de puntos rojos, que representan los nodos pertenecientes a la ruta. Por otro lado, la azul, al haber un conflicto en uno de los nodos, se parará en el nodo anterior o, escogerá una ruta más rápida para alcanzar su meta.

8.2.6 Resultados obtenidos

En este apartado se presenta un ejemplo, donde se reflejan las ventajas que podría traer un sistema como el expuesto en el aeropuerto de Madrid. Para ello, se van a utilizar dos aeronaves, una que acaba de aterrizar en la 32R y se dirige hacia el dique Sur y otra que aterriza desde la 32L y se dirige hacia la T4S, como podría ser aeronaves procedentes de América pertenecientes a la alianza OneWorld.

En primer lugar, en el caso presentado, el camino que realiza la aeronave que aterriza por la 32R podría referirse a aquellas aeronaves de índole internacional no Schengen, que proceden del Este, pudiendo venir de Arabia o Asia, por ejemplo. Para ello, en esta primera ruta, se indicará que salga por la salida rápida K5 y el nodo final será el cruce entre las calles de rodaje A12 y LF. En la siguiente imagen se puede observar la ubicación de estos dos nodos:

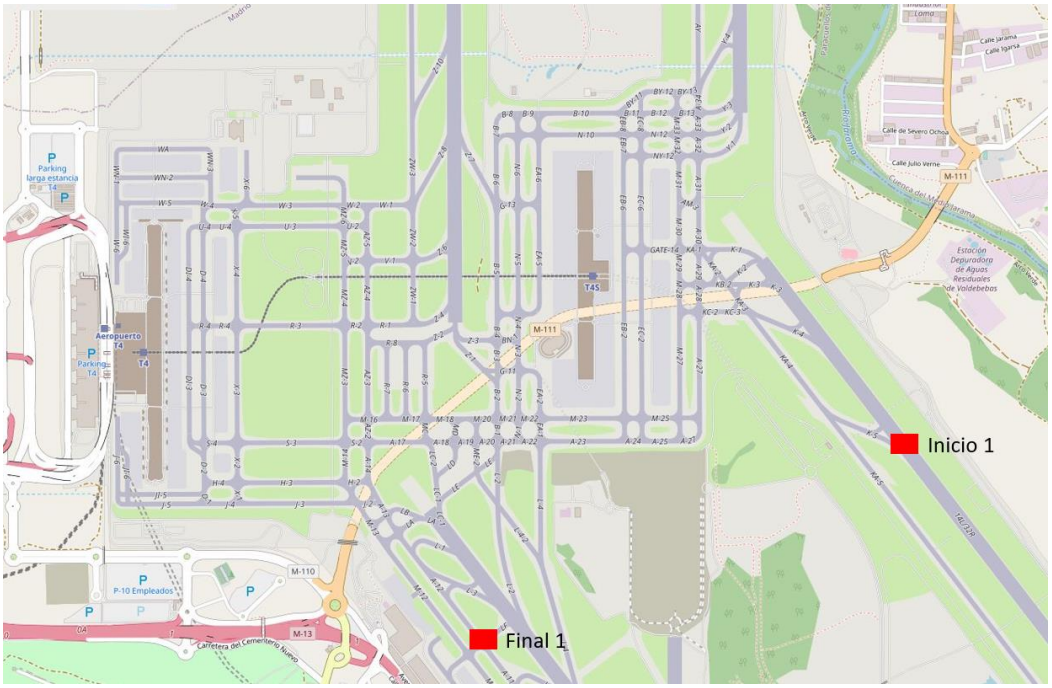


Figura 23: Punto inicial y final de la ruta 1

Por otro lado, la segunda ruta se dirigirá desde la salida rápida de la 32L con L4 y su último nodo será el cruce entre la calle M28 y KB2. Es posible que esta ruta no sea la ruta que realizan habitualmente las aeronaves que se dirigen a la T4S, sin embargo, esto queda fuera del interés del trabajo ya que se pretende centrar el problema en el conflicto presente. A continuación, se muestra la figura del punto inicial y final de esta segunda ruta:

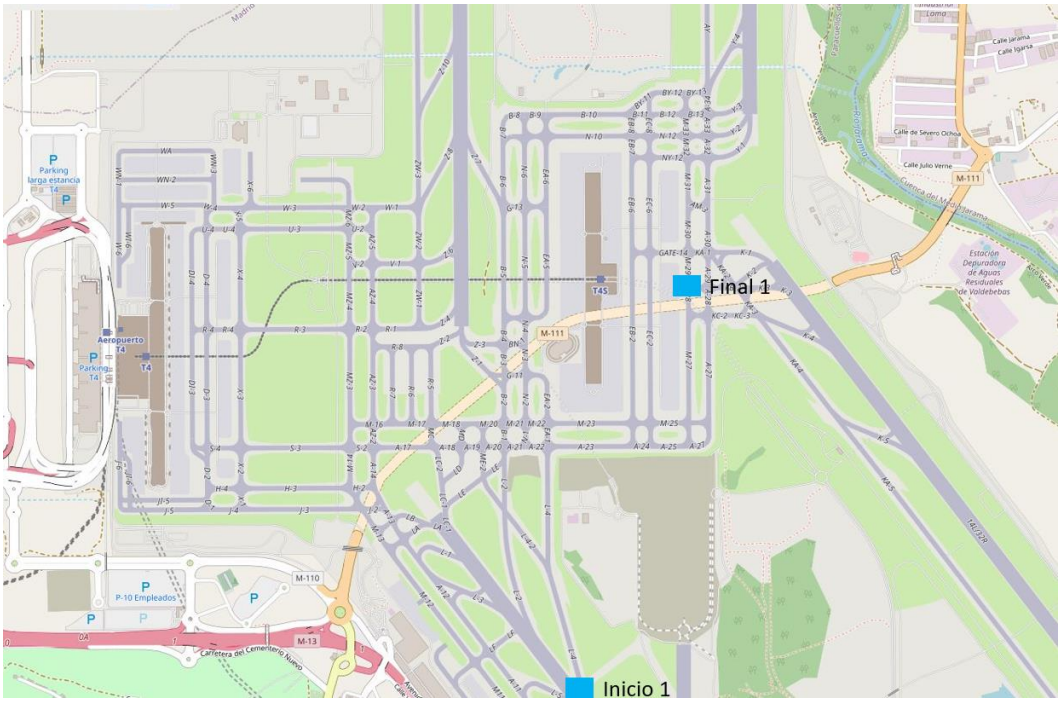


Figura 24: Punto inicial y final de la ruta 2

Por tanto, los nodos que deben seguir ambas rutas son:

- Ruta 1: [1, 2, 4, 5, 59, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 22, 23, 24, 25, 26, 27, 28, 29]
- Ruta 2: [31, 32, 13, 55, 56, 57, 58, 60, 61]

Que, siendo representadas gráficamente se observan las siguientes dos rutas:

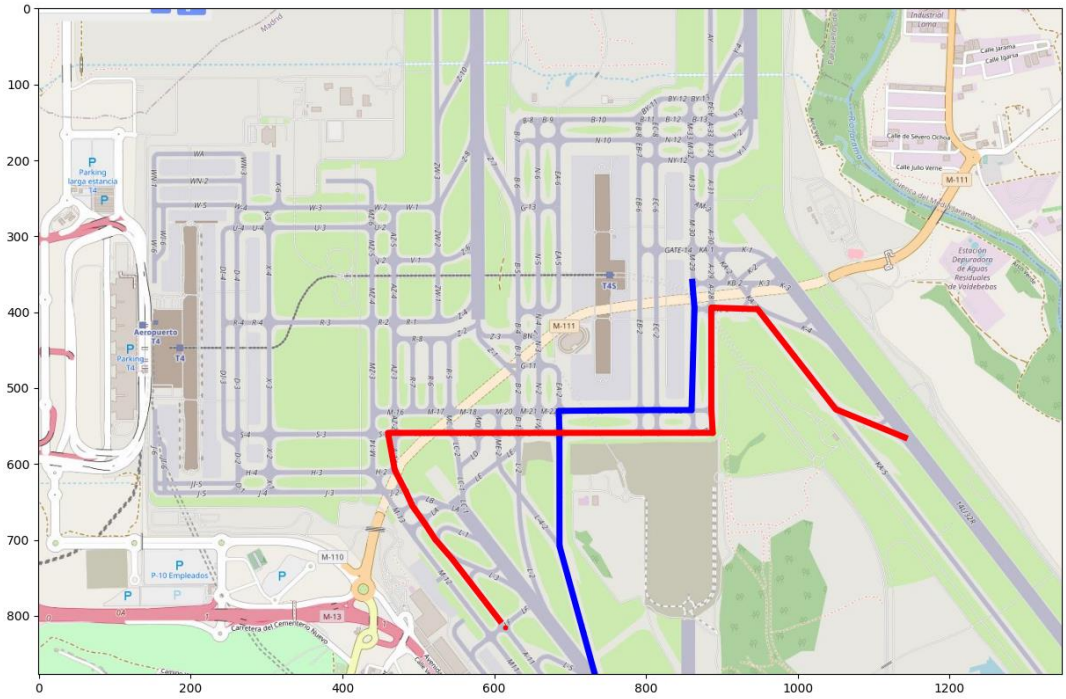


Figura 25: Ruta 1 y Ruta 2 impresas por pantalla

Como se puede observar, existe un nodo conflictivo, siendo el nodo 13, al tener ambas rutas dicho nodo dentro de sus rutas. Además, al cruzar este nodo dentro de un mismo margen, la aeronave azul (ruta 2) deberá hacer una parada y esperar hasta que la aeronave roja haya cruzado el punto y se encuentre a una distancia suficientemente segura. Cabe destacar que, en el código se declara que la aeronave roja es la prioritaria. De este modo, a continuación, se observa como la aeronave azul realiza la espera en el nodo anterior:

Figura 26: Conflicto y espera de la aeronave azul

Como se comentaba en apartados anteriores, el código está escrito para que la espera se realice en el nodo anterior al conflicto. Como la base de datos de nodos es reducida, es por ello por lo que la espera la realiza en el punto que se observa. No obstante, a medida que más nodos vayan siendo incorporados a la base de datos, la espera se realizará más cerca del nodo conflictivo. Sin embargo, esto de cara al análisis no afecta ya que el tiempo de espera es el mismo, lo cual no importa en que parte de la ruta se debería realizar.

Por tanto, con la información anterior se obtienen los siguientes datos de las rutas generadas:

- **Distancia ruta 1:** 4,307 km
- **Distancia ruta 2:** 2,449 km

Y, sabiendo que la velocidad de las aeronaves es constante en 47 km/h, se determina que el tiempo para realizar cada ruta es:

- **Tiempo ruta 1:** 5 minutos y 30 segundos
- **Tiempo ruta 2:** 3 minutos y 8 segundos

No obstante, como se ha observado, debido al conflicto, la segunda aeronave, que realiza la ruta 2 se ve obligada a hacer una parada. Esta parada, dura 40 segundos, por tanto, el tiempo total de la ruta 2 es:

- **Tiempo total ruta 2:** 3 minutos 48 segundos

Una vez analizando la trayectoria más corta en distancia, se procede a calcular la trayectoria más corta en tiempo, la cual es la que se obtiene del código realizado en la primera versión del software. El resultado obtenido es el mostrado en la siguiente figura:

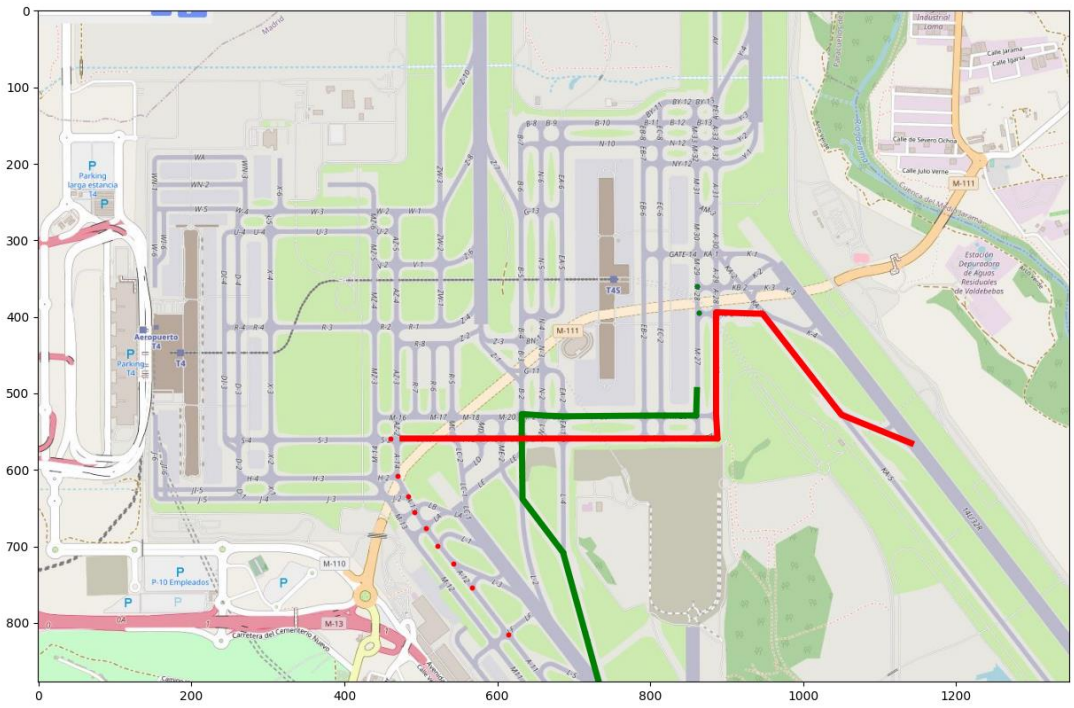


Figura 27: Ruta 2 optimizada en tiempo

Como se puede observar, la ruta seguida por la aeronave 2 ya no es la misma que la calculada inicialmente. Esto es porque al cambiar de ruta, se evita dicha espera. El conflicto entre las dos aeronaves ya no ocurre por tema de tiempo y, por tanto, la ruta tiene un coste en tiempo menor. Esta nueva ruta se denominará ruta 3, la cual queda marcada por la línea verde y tiene los siguientes costes:

- **Distancia ruta 3:** 2,709 km
- **Tiempo ruta 3:** 3 minutos 28 segundos

Por tanto, el tiempo total sería 20 segundos más que la ruta 2 sin el tiempo de parada debido al conflicto, sin embargo, sería 20 segundo más rápida que la ruta 2 añadiendo el tiempo de conflicto. En cambio, la distancia a recorrer sí sería algo más elevada, 260 metros más.

Este ahorro de tiempo supone un ahorro de tiempo de casi el 9% con respecto a la ruta original. Esto es una cifra considerable, teniendo en cuenta que aquí solo se está valorando un conflicto en una de las rutas. Por tanto, si se empezasen a añadir todas las aeronaves que

operan al mismo tiempo se podría ahorrar un tiempo destacado. Esto supondría un ahorro de tiempo al largo plazo de combustible para las aerolíneas, y una mayor satisfacción por parte de los pasajeros.

Asimismo, en este caso, permite al controlador estar algo menos saturado ya que sería un conflicto evitado, donde no sería necesario dar instrucciones o autorizaciones a las dos aeronaves, lo cual resulta en otro beneficio claro.

Cabe destacar que, los tiempos calculados anteriormente son bastante más bajos a los tiempos medio de taxi para el aeropuerto de Madrid mencionados en apartados previos. Esto se debe a que las rutas no consideran los tiempos que transcurre la aeronave por las plataformas o el rodaje en pista. A su vez, se asume que la velocidad es constante lo cual puede no ser siempre cierto, sin embrago, dicho valor no afectaría en gran medida a los resultados obtenidos.

Por otro lado, teniendo en cuenta las cifras de tráfico en el aeropuerto de Madrid, y el número de aeronaves que recorren las calles de rodaje, se puede asegurar que dicho beneficio resulta posible. Aunque es difícil de obtener un valor concreto del número de aeronaves que se verían beneficiadas por dicho sistema a lo largo de un año, sabiendo que existe una mejora, se da por sentado que el beneficio existe. Por tanto, teniendo en cuenta que dicha mejoría afectase al 5% del tráfico de Madrid, siendo principalmente en los momentos con mayor cantidad de tráfico de aeronaves, se obtiene un resultado muy favorable.

En 2022, en el aeropuerto de Madrid se registraron 351.960 operaciones [27], por tanto, asumiendo que la mitad son taxi-in y la mitad de taxi-out. De este modo, considerando los datos promedios de tiempos de taxi, 9,1 minutos para taxi-in y 17,1 minutos para taxi out, se obtiene que:

$$157.980\ operaciones * 9{,}1\ minutos = 1.437.618\ minutos\ de\ taxi\ in$$
$$= 23.960\ horas\ de\ taxi\ in$$

Por tanto, asumiendo que en el 5% de los vuelos se consigue un ahorro de un 9% del tiempo de taxi se llega a un ahorro de:

$$157.980\ operaciones * 0{,}05 * 9{,}1\ minutos * 0{,}09 = 107,8\ horas\ de\ taxi\ in\ ahorradas$$

Realizando el mismo cálculo para el taxi-out, se alcanzan los siguientes resultados:

$$157.980\ operaciones * 17{,}1\ minutos = 2.701.458\ minutos\ de\ taxi\ out$$
$$= 45.024{,}3\ horas\ de\ taxi\ out$$

$$157.980\ operaciones * 0{,}05 * 17{,}1\ minutos * 0{,}09 = 202{,}6\ horas\ de\ taxi\ out\ ahorradas$$

Asumiendo que el 5% de los vuelos pueden ser optimizados con mejores rutas, reduciendo su tiempo de taxi hasta un 9%, se obtiene que se puede ahorrar hasta 310 horas de taxi.

Teniendo en cuenta que una aeronave promedio consume una media de 12 litros por minuto mientras realiza el taxi, se obtiene que se estaría ahorrando 223.200 litros de combustible al año en el aeropuerto de Madrid. Esto implica un ahorro de 174.826,40 € en combustible ahorrados por las aerolíneas.

Aunque pueda parecer un valor bastante bajo, supone un ahorro para las aerolíneas, que cada vez, intentan buscar formas de ahorrar para poder ofrecer billetes más económicos. Asimismo, no se puede olvidar que los beneficios son muchos más aparte del económico, siendo este uno más que demuestra el valor añadido del sistema.

Capítulo 9. Próximos pasos

Como se puede observar en el apartado anterior, el presente código está en una versión preliminar y aún quedan numerosos pasos para alcanzar el producto final. Para poder completar el proyecto, será necesario recibir inputs de otros sistemas, los cuales permitirán un grado de exactitud y precisión mucho más elevado. No obstante, al tratarse de un proyecto de grandes características, es posible que estos datos, inputs o información en tiempo real, no se reciban hasta los últimos pasos del proyecto. Por tanto, para incorporar cierta información, se hará uso de datos simulados que sirvan como señales de entrada que luego puedan ser sustituidas por las reales.

Por tanto, una vez los requisitos se hayan escrito y las funcionalidades y especificaciones deseadas por el cliente se hayan determinado, las siguientes fases a tener en cuenta para los Hitos 3 y 4 en el desarrollo del código son:

- Fase 1. Añadir un mayor número de nodos que permitan hacer los puntos de espera más cerca a los nodos conflictivos e incluir todos aquellos nodos de las plataformas.
- Fase 2. Incorporar los conflictos entre una mayor cantidad de aeronaves en diferentes nodos, incorporando alertas y prioridades que pueda seleccionar el controlador.
- Fase 3. Conseguir datos de velocidad de aeronaves en tiempo real mediante radar ADS-B, almacenando la información de cada aeronave, piloto y aerolínea en una base de datos para poder hacer un análisis posterior.
- Fase 4. Incorporar las señales de entrada de aeronaves en llegadas y EOBT para aeronaves en salidas, lo cual permita conocer la ruta a seguir de manera anterior.

Seguidamente, existirán dos fases más relacionadas con la integración del código y la puesta en funcionamiento de manera progresiva y correcta:

- Fase 5. Pruebas de SW.
- Fase 6. Puesta en funcionamiento de manera progresiva.

9.1 Descripción detallada de la continuación del código

A continuación, se presentan cada fase descrita de manera más detallada, demostrando el nivel de madurez que alcanzará el código una vez toda la parte de desarrollo de Software se haya completado.

9.1.1 Fase 1

El objetivo de la fase 2 será incorporar todos aquellos nodos, los cuales quedaron sin incluir en la versión prototipo. Asimismo, para hacerlo de manera más limpia y simplificada, de cara a poder modificarlo de manera más clara en un futuro, esto se realiza en un archivo XML, pudiéndolo editar en Excel. De esta forma, teniendo varios ficheros, acordes a las distintas configuraciones del aeropuerto, se puede tener un control y una simplicidad del código mucho mayor, lo cual reduciría significativamente el tiempo de cálculo y mejorando así la fluidez del Software.

Además, en esta fase se añaden los nodos relacionados a las plataformas. Para ello se deberá tener especial cuidado a las formas de operar, siguiendo en todo momento las indicaciones pertinentes, presentes en el AIP.

Mediante el formato XML, el poder reconfigurar el aeropuerto en caso de obras resulta extremadamente rápido y fácil, lo cual permite adaptar el Software a la situación presente en el aeropuerto en prácticamente tiempo real. Además, esto supone una clara ventaja ya que el código permanecería intacto, modificando únicamente el fichero Excel, lo cual genera menos problemas de cara a la validación del Software.

Por otro lado, sería ideal implementar estas coordenadas utilizando coordenadas UTM, lo cual, permitiría adecuar la posición a cualquier tipo de mapa desechando la necesidad de adaptar la posición de los nodos a la imagen tomada, siendo el sistema utilizado en la versión preliminar. Esto permitiría utilizar mapas de mayor tamaño y que el operador o controlador enfocase la región o zona del aeropuerto que desea.

9.1.2 Fase 2

En la fase 3, se tiene propuesto la incorporación de todos los conflictos posibles. Actualmente, la versión preliminar solo valora la posibilidad de dos rutas que se dirigen de un punto inicial hasta un punto final. Sin embargo, en la operación de aeropuertos, son múltiples las aeronaves que circulan por las vías de rodaje y resto de vehículos presentes. Por tanto, se debe incluir la opción de añadir un número elevado de aeronaves y vehículos. Para alcanzar dicho objetivo, se debe añadir el cálculo de las rutas óptimas para todas las posibles rutas. Para ello, es necesario incorporar en el código los mismos bucles y condiciones impuestos para 2 aeronaves, pero en este caso para un número N de aeronaves o vehículos.

Además, en esta fase, el código debe adaptarse para poder seleccionar la prioridad de cada vehículo en cada caso. Esto quiere decir que, en vez de priorizar una aeronave por otra, priorizarla por nodos. Por tanto, en cada nodo conflictivo, indicar cual es la aeronave que tiene preferencia, siendo esa la que se dejará pasar primero. Para ello, cuando el sistema calcule la

existencia de un nodo conflictivo, lo cual implica que van a encontrarse dos aeronaves en un mismo punto durante un intervalo de tiempo, saltará una alarma que el controlador deberá contestar, indicando la aeronave prioritaria. Para ello, el controlador se basará en el resto de información percibida por el sistema, es decir, si la aeronave llega con retraso, posibilidad de despegue inminente o puesto de estacionamiento libre. De este modo, el sistema irá modificando las rutas de los aviones a medida que estos inputs por parte del controlador se vayan desarrollando.

Cabe destacar que el controlador podrá previsualizar las consecuencias que podría tener cierto input sobre el resto de las trayectorias. Con toda esa información, la decisión podrá tomarse con una mayor seguridad y certeza de que no existen posibles conflictos futuros.

9.1.3 Fase 3

A medida que el resto de las fases están realizándose, se empezará a probar dicho sistema con datos reales de aeronaves moviéndose ya en el aeropuerto de Madrid. Para ello, mediante un receptor ADS-B, el cual resulta considerablemente económico, se pueden tener las posiciones de las aeronaves en el entorno aeroportuario. Del mismo modo, como opción incluso más económica, se podría utilizar la información proporcionada por la página web *FlightRadar24*. Con estos datos, se genera una base de datos de Big Data, donde se almacena todos los datos relativos al Plan de Vuelo, incluyendo uno identificando el piloto de la aeronave.

La idea de generar esta base de datos es que el sistema pueda retroalimentarse de estos datos para conocer las rutas óptimas con mayor precisión, identificando aquellas aeronaves que realizan el Taxi de forma más lenta o que salida rápida suelen tomar. De este modo, se pueden identificar y calcular ciertos riesgos o conflictos potenciales reduciendo así la necesidad de cálculo en tiempo real. Además, esto permite comunicar al controlador aéreo una serie de datos con anterioridad, lo cual resulta más sencillo adaptar su carga de trabajo según las necesidades de cada instante.

A medida que la base de datos se vaya generando, y los datos puedan ser clasificados, se podrán observar ciertas tendencias en ciertos caminos, las cuales servirán para mejorar el sistema. Del mismo modo, cierta base de datos podría ser empleado en otros aeropuertos donde las mismas aerolíneas operen y tengan el mismo tipo de tráfico. Por tanto, se trata de una herramienta diferencial, la cual le dará un valor añadido al sistema y a futuras integraciones en otros aeropuertos.

9.1.4 Fase 4

Como se explicaba en apartados anteriores, los sistemas de navegación se alimentan de datos procedentes de otros sistemas que permiten conocer información de las aeronaves con anterioridad. Esto se trata de las fases estratégicas y pre-tácticas. Con ello, se consigue identificar posibles conflictos que puedan ocurrir el día de operación bajo estudio. Tras identificarlos, el sistema podrá asegurar que dichos problemas generen el menor problema posible, reduciendo los conflictos y los tiempos de taxi todo lo posible. Asimismo, incorporar toda esta información contrastándola con la obtenida con la del A-CDM para alcanzar un nivel de rendimiento óptimo, conociendo el estado del aeropuerto en todo momento. Esto permitirá que el Software sea más fluido en la fase táctica, generando ventajas al controlador, ya que toda la posible información previamente analizada ya se habrá tenido en cuenta para indicar las rutas y las esperas para cada aeronave dentro del área de maniobras.

9.2 Integración y puesta en funcionamiento

9.2.1 Fase 5

La quinta fase, la cual tiene relación con todo lo que respecta a las pruebas y testing del Software se realiza en los hitos 3, 4, 5 y 6. Es por ello por lo que, se trata de una de las tareas fundamentales para el éxito del proyecto. En este caso, se realizarán pruebas a medida que nuevas versiones del Software vayan publicándose y, finalmente, unas pruebas de certificación que aseguren el correcto funcionamiento del sistema, indicando que es seguro de operar.

Para demostrar que el Software cumple con todos esos requisitos que han publicado el equipo de Ingeniería, se desarrollan una serie de pruebas. Habrá un conjunto de tests, llamados de regresión, que servirán como demostración de que el Software funciona. Para ello, estos tests deben cubrir la mayor parte de los requisitos considerados fundamentales.

Una vez una nueva versión sea lanzada, se deberán probar todas aquellas funcionales que se han implementado nuevas, y a su vez, testar todo aquello que ha sido corregido. Para testar las funcionales corregidas, se podrán emplear pruebas pertenecientes al conjunto de no regresión, o se podrán realizar tests más específicos para dicho error. Cabe destacar que, en caso de que en una de estas pruebas se vea una anomalía en el Software, se deberá abrir una incidencia indicando el error, de modo que pueda ser corregido por el departamento de Software.

Por otro lado, es importante señalar que todas aquellas pruebas quedarán escritas en una base de datos, que permitan ser revisadas por el equipo de Ingeniería, los cuales se encargarán de comprobar que todos los tests cumplen con los requisitos impuestos.

Por último, se deberá realizar una fase de certificación. Para ello, una vez se tenga la versión final, la cual contiene todas las funcionalidades, se realizarán unas pruebas formales, las cuales validará EASA y ENAIRE, demostrando que el sistema es seguro y no contiene fallos significativamente graves. En esta prueba se ejecutarán todos aquellos tests realizados a lo largo del desarrollo del sistema, desde las pruebas de regresión, hasta aquellos hechos para verificar que cierta funcionalidad ha sido corregida.

Una vez hechas estas pruebas, se procedería a la última fase del desarrollo del Software, la cual sería la implementación de este en la torre de control.

9.2.2 Fase 6

Tras haber pasado las pruebas formales, se comienza la puesta en funcionamiento del sistema de manera progresiva. Esto se realiza plenamente en el sexto hito del proyecto. Para ello, se hará en reducidos pasos durante 3 meses, utilizando aquellos periodos del aeropuerto donde el tráfico es mucho menor, por ejemplo, durante la noche. De este modo, introduciéndolo de manera progresiva, se observarán posibles errores en los momentos menos críticos. Además, esto permitirá tener a los controladores aéreos más concentrados en posibles fallos del sistema al haber un menor tráfico.

Con ello, se pretende integrar todas aquellas funcionalidades como pueden ser la iluminación de las calles de rodaje que indicarán el recorrido que debe realizar la aeronave, las barras de parada en caso de conflicto y necesidad de parar y la señalética de los puestos de estacionamiento. Esto, al no poderse testar de manera remota, durante esta fase se comprobará que funciona correctamente y los sistemas anexos se alimentan de la información del A-SMGCS correctamente.

9.3 Posibles mejoras

El proyecto podrá incluir mejoras a lo largo de todo el periodo de desarrollo de Software. Para ello, el equipo de Ingeniería se reunirá con los controladores aéreos de manera habitual para discutir temas y funcionalidades requeridos. Asimismo, una vez puesto en operación el sistema, se permitirán realizar mejoras e implementarlas en futuras actualizaciones, siendo así un beneficio para ENAIRE y el aeropuerto de Madrid.

Por otra parte, de cara al futuro, sería interesante integrar dicho sistema en un sistema en la nube el cual permita conocer el estado de otros aeropuertos en tiempo real dentro de la misma herramienta. Así, en vez de utilizar nodos introducidos manualmente con el archivo XML, se podría emplear información proveniente de herramientas y librerías ya predefinidas

mediante inteligencia artificial. Un ejemplo es *Overpass Turbo* donde se devuelven las coordenadas de todas las calles de rodaje del aeropuerto de Madrid, identificando todas las calles de rodaje y plataformas de las distintas terminales.

Además, un punto a considerar que podría ser beneficioso es poder ajustar o indicar a los pilotos que velocidad se les recomienda realizar el taxi en cada momento, ajustándose al tráfico existente en cada momento. No obstante, esto trae serias complicaciones ya que dificulta determinar o eliminar posibles conflictos ya que se aumenta de manera considerable las opciones y prioridades para cada aeronave.

Capítulo 11. Análisis de riesgos

La ejecución de este proyecto representa ciertos riesgos, especialmente aquellos relacionados con la competencia de otras empresas ya posicionadas en el mercado. Aunque el número de empresas en el mundo que se dedican a realizar este tipo de sistemas no es muy elevado, la posición, experiencia y reconocimiento que tienen hacen que la entrada de una nueva empresa de reciente creación sea difícil. Por ello, el producto debe ser mejor, más barato y tener una imagen como imagen de empresa fiable.

Asimismo, estas empresas, las cuales ya tienen entre sus productos sistemas similares o, incluso, ya ofrecen un A-SMGCS para el aeropuerto de Madrid, aunque sea de un nivel inferior, tienen ya facilidad de convencer para que el aeropuerto apueste por dichos sistemas. No obstante, es posible que la adaptación de dichos sistemas al aeropuerto de Madrid resulte una tarea compleja, necesitando un largo tiempo de ejecución.

Por otro lado, otro posible riesgo que podría influir en la ejecución del proyecto sería la incorporación de los datos radar y demás parámetros relacionados a la operativa de aeronaves en los aeropuertos. Para ello, se debe conocer las empresas que proporcionan dicha información, ya sea el Network Manager por parte de Eurocontrol o la que utlice ENAIRE al ser la encargada de ofrecer los servicios de navegación aérea.

Es por ello por lo que, será necesario obtener un precontrato antes de comenzar en el desarrollo de la herramienta, conociendo claramente las necesidades del cliente o, incluso, considerar la opción de crear un consorcio con una empresa con un reconocimiento ya adquirido, desarrollando dicha herramienta en paralelo. De este modo, se pueden incluir funcionalidades ya presentes en las herramientas desarrolladas previamente por dicha empresa, ofreciendo un producto aún más completo, el cual genere una mayor satisfacción al cliente.

Además, cabe destacar que, en el sector aeronáutico, la seguridad es esencial siendo parte fundamental de todos los proyectos, desde la fabricación de aeronaves, hasta la operativa de las mismas por los entornos aeroportuarios. Por ello, la herramienta y sistema deben cuidar que la seguridad y cualquier riesgo posible dentro del área de maniobras sea considerado y reflejado para tomar la decisión correcta. Por ello, el sistema propuesto debe tener un proceso de pruebas y certificación acorde a las necesidades exigidas. Por tanto, se debe asegurar que la herramienta aprueba las fases de certificación para que no haya ningún problema de recertificación o insatisfacción por el cliente. Posteriormente, una vez puesta en operación, los errores que pueda generar la herramienta deben ser mínimos, sin generar un efecto negativo en la futura comercialización de la herramienta en otros aeropuertos.

Por último, se debe considerar que la herramienta pueda sufrir una parada en operación. Aunque el objetivo es que sea continua y disponible en todo momento, podrá existir algún periodo de excepción no valorado o imposible de mantener el servicio. Por tanto, deberá

contener un sistema de apoyo que permita operar en funciones mínimas, el cual de soporte al controlador y no pierda el total control sobre las aeronaves. De este modo, el aeropuerto no tenga que parar la operativa del mismo.

Capítulo 12. Análisis de alternativas

En caso de que el sistema no sea accesible para el aeropuerto de Madrid, el mismo proyecto puede realizarse en otros aeropuertos en crecimiento. Debido a la estructura modular del código, donde el mismo principio puede utilizarse en cualquier aeropuerto, cambiando únicamente la definición de nodos y vecinos, permitiendo una versatilidad muy rápida y fácil. Por tanto, en caso de que el aeropuerto de Madrid no promueva el desarrollo de esta herramienta, existen muchos otros aeropuertos en los cuales se puede emplear la herramienta, motivándolos a mejorar sus procedimientos de navegación por tierra, consiguiendo una gran optimización. Con ello, se alcanza a aumentar la cartera de posibles clientes significativamente.

Por otro lado, otra posibilidad es la utilización de la herramienta para el cálculo de las rutas óptimas en caso de que existan obras en ciertas calles de rodaje en los aeropuertos. Esto permitiría hacerles conocer a los controladores que posibles opciones tienen para reducir el tiempo de taxi de las aeronaves cuando están en configuraciones no habituadas a ellas. Debido a ello, conseguirían mejorar la capacidad del aeropuerto, aunque el aeropuerto no esté operando en las mejores condiciones. Simplemente, utilizando los nodos oportunos y el algoritmo A-Star, con el que el código obtiene la ruta más corta, se puede obtener aquellas posibilidades en caso de obras.

Del mismo modo, se podría adaptar el código para controlar los vehículos de tierra, ya sean jardineras, vehículos de carga de equipajes o los vehículos de emergencia entre otros. Con ello, se conseguiría reducir las emisiones en gran medida, ahorrando también en combustible, los cuales reduciría considerablemente los gastos del aeropuerto. Es importante señalar que, en ciertos aeropuertos, como puede ser el de Madrid, las rutas que realizan estos vehículos son complejas y que, a su vez, deben tener en cuenta el curso de las aeronaves. Por tanto, podría ser una posible mejora para tener en cuenta.

Todo lo anterior ofrece infinidad de posibilidades que permiten hacer que la herramienta resulte en un producto de especial atractivo para numerosos clientes mundiales. Además, debido al crecimiento constante y a la popularidad por operar en aeropuertos de cada vez, mayor tamaño, hace que herramientas como estas sean oportunidades para mejorar la operación, consiguiendo disminuir los conflictos, tiempos de taxi, emisiones de gases contaminantes y, consecuentemente, aumentar la seguridad en el entorno aeroportuario. Es por ello por lo que, aunque el producto no pueda ofrecerse en el aeropuerto de Madrid, existen más mercados como posibles aeropuertos emergentes en el Sureste Asiático, Sudamérica o la Península Arábica, entre otros. No obstante, un estudio preliminar de los beneficios y la necesidad real de herramientas como estas será necesario para ofrecer al cliente el mejor producto adaptado a los requisitos infraestructurales y de navegación.

Capítulo 13. Conclusiones

Las herramientas de optimización están cada vez más presentes en el día a día de las personas, proyectos y sistemas, ya que los beneficios a largo plazo son claros. Es por ello por lo que, aplicar tecnología que mejoren en tiempo, coste o capacidad en las áreas de maniobras de un aeropuerto resulta una idea a valorar, mas cuando el tráfico aéreo está en aumento.

Habiendo analizado el estado actual del aeropuerto de Madrid en el ámbito de sistemas de control de trayectorias en superficie, se observa que existe una gran oportunidad para implementar un sistema que sustituya al actual, el cual, está cada vez más anticuado con respecto a la tecnología actual.

Como se analiza, este sistema presente hoy en día en Barajas es de un sistema A-SMGCS que pretende ser nivel 2 en un corto plazo, por lo que las alertas son limitadas, dependiendo en gran medida del trabajo realizado por los controladores. Además, este sistema se centra especialmente en áreas restringidas como son las pistas y calles de salida rápida. Es por ello por lo que, el margen de mejora es muy amplio, pudiéndose crear un sistema que, de manera relativamente sencilla, gracias a la innovación en Inteligencia Artificial, pueda reconocer cualquier tipo de conflicto y asignar las mejores soluciones. De este modo, los beneficios y mejoras son sustancialmente altos, desde una toma de decisiones mejor por parte de los controladores hasta un menor tiempo de taxi. Por tanto, se trata de un producto que pretende generar una mejoría en muchas áreas del aeropuerto, siendo beneficiados aerolíneas, pasajeros, controladores y pilotos, entre otros.

Asimismo, como se observa en la Figura 1 y Figura 2, el aeropuerto de Madrid, aun siendo el aeropuerto de España con más operaciones y con unas calles de rodaje complejas por sus diferentes configuraciones de operaciones, no está a la altura de los aeropuertos de Palma y Barcelona. En estos dos aeropuertos, el sistema A-SMGCS implementado actualmente es de nivel 2, y ya se tiene planes para mejorarlo a nivel 4 en un futuro cercano. Es por ello por lo que, el aeropuerto de Madrid debe tomar un paso al frente y ajustar la calidad de servicio a los niveles que merece ofrecer.

Para ello, habiendo considerado esta una gran oportunidad para la ejecución de un proyecto, en el presente trabajo se presenta una solución, que de manera simplificada y a un alto nivel, podría dar solución a los futuros problemas que podría tener Barajas si no toma cartas en el asunto. De este modo, se ha diseñado la implementación de un sistema A-SMGCS de nivel 4, ofreciendo funcionalidades fuera del alcance de otros productos ya presentes en el mercado.

Haber estudiado el mercado resulta esencial para poder cubrir todas las posibles necesidades que podría tener el aeropuerto. En este caso, a partir de los aeropuertos más grandes del mundo, se ha investigado que tipo de servicios se le ofrecían para el movimiento de aeronaves en tierra. Con esta información se consigue rellenar la Tabla 1, la cual es esencial para conocer

debilidades de otros competidores y oportunidades para el producto desarrollado. De esta tabla se obtiene que la mayoría de los sistemas no son completos, es decir, algunos ofrecen un mejor sistema de planificación, control y diseño de nuevas trayectorias para aeronaves en conflictos, mientras que otros carecen de sistemas que permitan que la aeronave pueda navegar por el aeropuerto sin necesidad de estar bajo las instrucciones continuas del controlador de superficie.

Asimismo, a modo de estudio, en el presente trabajo se realiza un estudio de los beneficios en términos de tiempo que podría representar un sistema como este en el aeropuerto de Madrid. Para ello, se ha realizado una versión prototipo que sirva a modo de explicación de los resultados que se pretenden comprender y así, también, servir de punto de partida de cara a mejorar dicho sistema hasta tener una versión más completa.

En esta versión preliminar se hizo uso de una configuración del aeropuerto, en este caso llegadas y en configuración norte. Tras analizar los efectos de la modificación de una trayectoria en caso de que haya un conflicto en la ruta, se observa que puede haber un beneficio del orden del 9% de tiempo de taxi. Este valor está en el orden de magnitud de la mejoría en el tiempo de taxi de otros fabricantes de sistemas A-SMGCS. Por tanto, se estaría hablando de un valor muy considerable que podría representar una gran mejoría en las operaciones del aeropuerto de Madrid. Además. Conociendo esto, extrapolando a la saturación actual del aeropuerto y, en especial al tráfico esperado para los próximos años, se puede asegurar que el beneficio es claro en lo que a ahorro de tiempo de taxi se refiere. A este beneficio habría que añadir todos los comentados anteriormente, seguridad, capacidad, emisiones de gases contaminantes, coste de combustible, carga de trabajo de los controladores, etc.

Con todo lo anterior realizado, se exponen las distintas fases a seguir para la correcta ejecución del código. Cabe destacar que estas 6 fases propuestas van de la mano con los hitos del proyecto, los cuales, destacando el diagrama de Gantt creado, el proyecto debería finalizarse en 2 años, teniendo 1 año y 4 meses para las fases de diseño de los requisitos y la evolución del Software. Es necesario este tiempo tan extenso para que el código no contenga errores, y se pueda testar acorde a la especificación. Además, será esencial escuchar al cliente, el cual propondrá también ciertas funcionalidades y diseños acordes a las necesidades de los controladores. Del mismo modo, será necesaria que se reciba información acorde a la arquitectura de los equipos ya presentes, necesario para poder integrar las señales del radar de superficie y demás sistemas que se alimentan o proporcionan información del A-SMGCS propuesto.

Por otro lado, también es necesario tener en cuenta que es posible que sea necesario actualizar otros sistemas dentro del aeropuerto. Esto sería el caso de las luces de guiado, barras de parada y señalética, tanto de las calles de rodaje como las de los puestos de estacionamiento. Esto es debido a que serán salidas del sistema diseñado en este trabajo y

aporta un gran valor añadido para poder reducir de manera considerable las comunicaciones entre el piloto y el controlador de superficie.

En conclusión, tras haber analizado el tráfico presente en el aeropuerto de Madrid, el estado actual de la tecnología A-SMGCS y los datos esperados para los próximos años, se observa que una actualización es necesaria. Un nuevo sistema que permita mejorar la capacidad, aumentando a la vez la seguridad y reduciendo la carga de trabajo de los controladores, resulta esencial para mantener el nivel de calidad del aeropuerto. Además, permitiría mejorar ciertas características de tiempo de taxi, lo cual vendría de la mano en la reducción de vuelos con retraso y una mayor satisfacción de los pasajeros y aerolíneas. Esto, además, podría atraer a potenciales clientes, o más operaciones diarias.

No obstante, es importante señalar que estas herramientas tienen mayores efectos en condiciones muy específicas, donde el tráfico es considerablemente elevado, pero tiene opciones a reducir la ruta de taxi. Existirán periodos de operación donde sea inservible ya que las aeronaves seguirán el camino más rápido directamente al no encontrarse con más aeronaves durante su camino. Sin embargo, seguiría mejorando la seguridad al estar pendiente de cualquier conflicto presente en las calles de rodaje y reduciendo la responsabilidad del controlador.

Por último, cabe destacar que, el sistema propuesto, al tener un diseño modular, es fácilmente adaptable a otros aeropuertos o modificaciones en la rodadura del aeropuerto de Madrid. Esto también presenta una clara ventaja competitiva ya que la capacidad de crecimiento y reconocimiento es muy alta. Es por ello por lo que, parece que sistemas como estos se harán un hueco en todos los aeropuertos del mundo, donde aprovechando la Inteligencia Artificial, se mejorarán las operaciones, pudiendo, así, tener una aviación más segura, responsable con el medio ambiente y eficiente.

Bibliografía

[1] AEROCLUB AVIATOR, "SEGURIDAD, MARCAS Y SEÑALES DE PISTA Y CALLED DE RODAJE," *BOLETÍN Segur. OPERACIONAL Nº 003*, p. 20, 2021.

[2] COMISIÓN DE INVESTIGACIÓN DE ACCIDENTES E INCIDENTES DE AVIACIÓN CIVIL, "INFORME TÉCNICO IN-006/2022," 2022.

[3] L. Ahlgren, "Which Airports Have The Longest Taxi Times?," *Simple Flying*, 2021. https://simpleflying.com/which-airports-have-the-longest-taxi-times/ (accessed Apr. 03, 2023).

[4] INDRA, "InNOVA Routing and Guidance," Asker, Norway.

[5] SAAB, "A-SMGCS." https://www.saab.com/products/a-smgcs (accessed Jan. 29, 2023).

[6] SKYbrary Aviation Safety, "Advanced Surface Movement Guidance and Control System (A-SMGCS) ." https://www.skybrary.aero/articles/advanced-surface-movement-guidance-and-control-system-smgcs (accessed Jan. 29, 2023).

[7] SESAR, "SESAR Joint Undertaking | Airport safety nets - SAFE." https://www.sesarju.eu/projects/safe (accessed Jan. 29, 2023).

[8] INDRA, "El aeropuerto de Heathrow comienza a operar con la tecnología A-SMGCS de última generación de Indra | indra," Aug. 30, 2019. https://www.indracompany.com/es/noticia/aeropuerto-heathrow-comienza-operar-tecnologia-smgcs-ultima-generacion-indra (accessed Jan. 29, 2023).

[9] OACI, "RESUMEN DE DISCUSIONES RLA06/901 - SEMINARIO VIRTUAL DE CONTROL, GUÍA Y MOVIMIENTO EN SUPERFICIE DE LA OACI PARA LA REGIÓN SAM (21SAMSMGCS)," pp. 1–10, 2021.

[10] EUROCONTROL, "LSSIP 2021 - SPAIN LOCAL SINGLE SKY IMPLEMENTATION," p. 196, 2021.

[11] INDRA, "Indra will make Budapest the major international airport in Europe to manage landings and take offs remotely ," *indracompany.com*, Oct. 26, 2021. https://www.indracompany.com/en/noticia/indra-will-make-budapest-major-international-airport-europe-manage-landings-take-offs (accessed Apr. 07, 2023).

[12] World ATM Congress, "The Frequentis Aviation Arena," *worldatmcongress.org*, Jun. 15, 2022. https://www.worldatmcongress.org/frequentis-aviation-arena (accessed Apr. 07, 2023).

[13] Frequentis, "TowerPad A-SMGCS," 2022.

[14] ADB SAFEGATE blog, "Abu Dhabi Airports reveal cutting edge airfield lighting & guidance system at Abu Dhabi International Airport (AUH) ," *ADF SAFEGATE*, Jun. 20, 2022. https://blog.adbsafegate.com/watch-the-video-from-abu-dhabi-international-airports-about-the-development-and-completion-of-the-a-smgcs-level-4-follow-the-greens-solution/ (accessed Apr. 07, 2023).

[15] "Airport collaborative decision-making (A-CDM) | EUROCONTROL." https://www.eurocontrol.int/concept/airport-collaborative-decision-making (accessed Jan. 29, 2023).

[16] Aeropuerto de Madrid-Barajas, "Aeropuerto Madrid-Barajas Adolfo Suárez." https://www.aeropuertomadrid-barajas.com/ (accessed Feb. 25, 2023).

[17] ENAIRE, "AD 2-LEMD 1," *AIP*, p. 42, 2022.

[18] Wikipedia, "Aeropuerto Adolfo Suárez Madrid-Barajas," *Wikipedia.org*. https://es.wikipedia.org/wiki/Aeropuerto_Adolfo_Suárez_Madrid-Barajas (accessed Apr. 08, 2023).

[19] ENAIRE, "AD 2-LEMD GMC 1," *AIP*, p. 6, 2022.

[20] ENAIRE, "AD 2-LEMD GMC 2," *AIP*, pp. 1–4, 2023.

[21] EUROCONTROL, "Taxi times - Summer 2021 ," *eurocontrol.int*, Mar. 25, 2022. https://www.eurocontrol.int/publication/taxi-times-summer-2021 (accessed Apr. 08, 2023).

[22] AENA, "Empresas de handling y autoasistencia en los aeropuertos," 2023. https://www.aena.es/es/aerolineas/operar-en-aena/aspectos-operativos-y-comerciales/handling-y-autoasistencia.html (accessed Feb. 26, 2023).

[23] ICAO, *Doc 9830 - Advanced Surface Movement Guidance and Control Systems (A-SMGCS) Manual*. 2004.

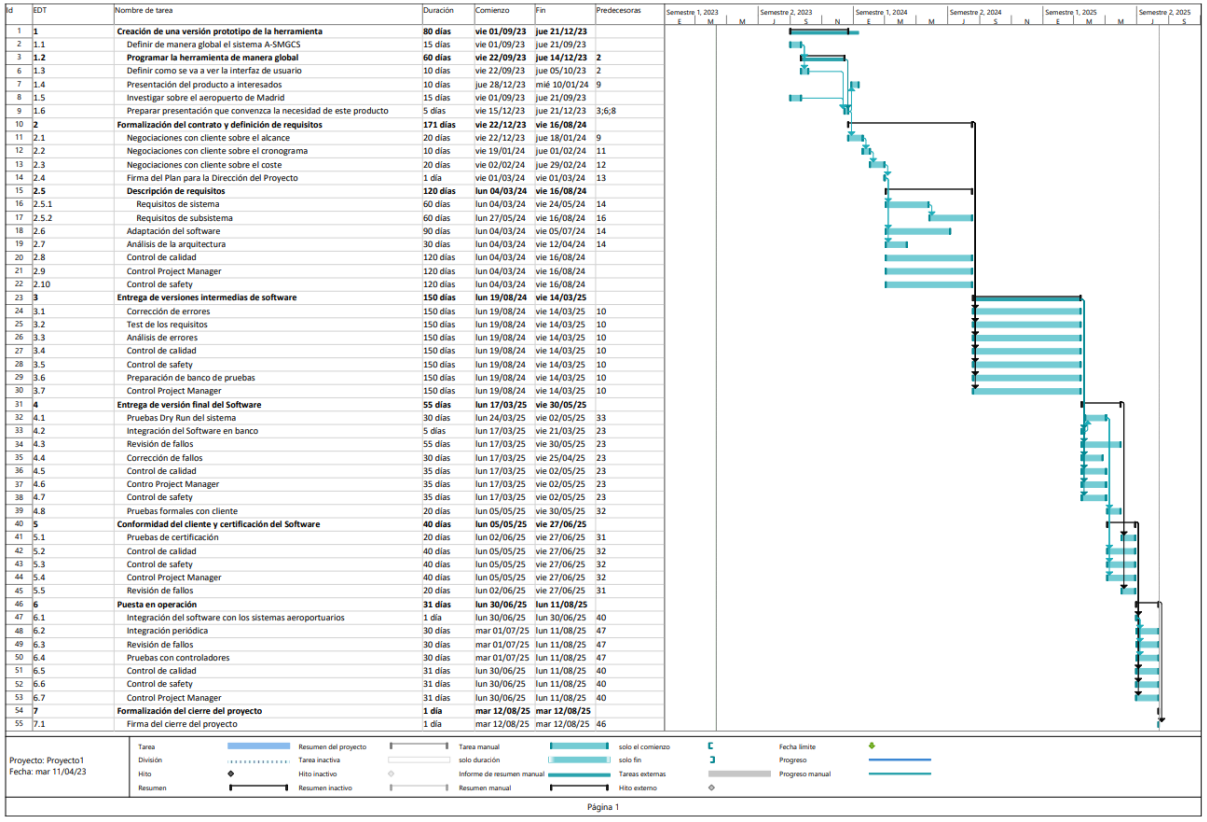
[24] Organización de Aviación Civil Internacional - OACI, *Doc 4444 ATM/501: Gestión del tránsito aéreo*. 2007.

[25] ENAIRE, "SACTA," *enaire.es*. https://www.enaire.es/servicios/atm/sistemas_de_gestion_del_transito_aereo_atm/sacta (accessed Apr. 09, 2023).

[26] EUROCONTROL, "Advanced Surface Movement Guidance and Control Systems (A-SMGCS) - Implementatio Manual," vol. 1.0, p. 76, 2011.

[27] "El Aeropuerto Adolfo Suárez Madrid-Barajas cierra 2022 con una recuperación del 82% del tráfico de 2019 y más de 50,6 millones de pasajeros," *AENA*, Jan. 10, 2023. https://www.aena.es/es/prensa/el-aeropuerto-adolfo-suarez-madrid-barajas-cierra-2022-con-una---recuperacion-del-82-del-trafico-de-2019-y-mas-de-506-millones-de---pasajeros.html (accessed Aug. 15, 2023).

Anexo 1. Diagrama de Gantt completo



Id	EDT	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**1**	**Creación de una versión prototipo de la herramienta**	**80 días**	**vie 01/09/23**	**jue 21/12/23**	
2	1.1	Definir de manera global el sistema A-SMGCS	15 días	vie 01/09/23	jue 21/09/23	
3	**1.2**	**Programar la herramienta de manera global**	**60 días**	**vie 22/09/23**	**jue 14/12/23**	**2**
6	1.3	Definir como se va a ver la interfaz de usuario	10 días	vie 22/09/23	jue 05/10/23	2
7	1.4	Presentación del producto a interesados	10 días	jue 28/12/23	mié 10/01/24	9
8	1.5	Investigar sobre el aeropuerto de Madrid	15 días	vie 01/09/23	jue 21/09/23	
9	1.6	Preparar presentación que convenza la necesidad de este producto	5 días	vie 15/12/23	jue 21/12/23	3;6;8
10	**2**	**Formalización del contrato y definición de requisitos**	**171 días**	**vie 22/12/23**	**vie 16/08/24**	
11	2.1	Negociaciones con cliente sobre el alcance	20 días	vie 22/12/23	jue 18/01/24	9
12	2.2	Negociaciones con cliente sobre el cronograma	10 días	vie 19/01/24	jue 01/02/24	11
13	2.3	Negociaciones con cliente sobre el coste	20 días	vie 02/02/24	jue 29/02/24	12
14	2.4	Firma del Plan para la Dirección del Proyecto	1 día	vie 01/03/24	vie 01/03/24	13
15	**2.5**	**Descripción de requisitos**	**120 días**	**lun 04/03/24**	**vie 16/08/24**	
16	2.5.1	Requisitos de sistema	60 días	lun 04/03/24	vie 24/05/24	14
17	2.5.2	Requisitos de subsistema	60 días	lun 27/05/24	vie 16/08/24	16
18	2.6	Adaptación del software	90 días	lun 04/03/24	vie 05/07/24	14
19	2.7	Análisis de la arquitectura	30 días	lun 04/03/24	vie 12/04/24	14
20	2.8	Control de calidad	120 días	lun 04/03/24	vie 16/08/24	
21	2.9	Control Project Manager	120 días	lun 04/03/24	vie 16/08/24	
22	2.10	Control de safety	120 días	lun 04/03/24	vie 16/08/24	
23	**3**	**Entrega de versiones intermedias de software**	**150 días**	**lun 19/08/24**	**vie 14/03/25**	
24	3.1	Corrección de errores	150 días	lun 19/08/24	vie 14/03/25	10
25	3.2	Test de los requisitos	150 días	lun 19/08/24	vie 14/03/25	10
26	3.3	Análisis de errores	150 días	lun 19/08/24	vie 14/03/25	10
27	3.4	Control de calidad	150 días	lun 19/08/24	vie 14/03/25	10
28	3.5	Control de safety	150 días	lun 19/08/24	vie 14/03/25	10
29	3.6	Preparación de banco de pruebas	150 días	lun 19/08/24	vie 14/03/25	10
30	3.7	Control Project Manager	150 días	lun 19/08/24	vie 14/03/25	10
31	**4**	**Entrega de versión final del Software**	**55 días**	**lun 17/03/25**	**vie 30/05/25**	
32	4.1	Pruebas Dry Run del sistema	30 días	lun 24/03/25	vie 02/05/25	33
33	4.2	Integración del Software en banco	5 días	lun 17/03/25	vie 21/03/25	23
34	4.3	Revisión de fallos	55 días	lun 17/03/25	vie 30/05/25	23
35	4.4	Corrección de fallos	30 días	lun 17/03/25	vie 25/04/25	23
36	4.5	Control de calidad	35 días	lun 17/03/25	vie 02/05/25	23
37	4.6	Contro Project Manager	35 días	lun 17/03/25	vie 02/05/25	23
38	4.7	Control de safety	35 días	lun 17/03/25	vie 02/05/25	23
39	4.8	Pruebas formales con cliente	20 días	lun 05/05/25	vie 30/05/25	32
40	**5**	**Conformidad del cliente y certificación del Software**	**40 días**	**lun 05/05/25**	**vie 27/06/25**	
41	5.1	Pruebas de certificación	20 días	lun 02/06/25	vie 27/06/25	31
42	5.2	Control de calidad	40 días	lun 05/05/25	vie 27/06/25	32
43	5.3	Control de safety	40 días	lun 05/05/25	vie 27/06/25	32
44	5.4	Control Project Manager	40 días	lun 05/05/25	vie 27/06/25	32
45	5.5	Revisión de fallos	20 días	lun 02/06/25	vie 27/06/25	31
46	**6**	**Puesta en operación**	**31 días**	**lun 30/06/25**	**lun 11/08/25**	
47	6.1	Integración del software con los sistemas aeroportuarios	1 día	lun 30/06/25	lun 30/06/25	40
48	6.2	Integración periódica	30 días	mar 01/07/25	lun 11/08/25	47
49	6.3	Revisión de fallos	30 días	mar 01/07/25	lun 11/08/25	47
50	6.4	Pruebas con controladores	30 días	mar 01/07/25	lun 11/08/25	47
51	6.5	Control de calidad	31 días	lun 30/06/25	lun 11/08/25	40
52	6.6	Control de safety	31 días	lun 30/06/25	lun 11/08/25	40
53	6.7	Control Project Manager	31 días	lun 30/06/25	lun 11/08/25	40
54	**7**	**Formalización del cierre del proyecto**	**1 día**	**mar 12/08/25**	**mar 12/08/25**	
55	7.1	Firma del cierre del proyecto	1 día	mar 12/08/25	mar 12/08/25	46

Figura 28: Anexo 1 - Diagrama de Gantt completo

Anexo 2. Código de selección de nodos

```
from PIL import Image, ImageTk
import tkinter as tk

# Load the image
image = Image.open(ImagenAeropuerto.JPG')

# Create a tkinter window
window = tk.Tk()

# Create a canvas to display the image
canvas = tk.Canvas(window, width=image.width, height=image.height)
canvas.pack()

# Convert the image to a PhotoImage object
photo_image = ImageTk.PhotoImage(image)

# Display the image on the canvas
canvas.create_image(0, 0, anchor='nw', image=photo_image)

# Create a function to display the coordinates when the mouse is
clicked
def show_coordinates(event):
    x, y = event.x, event.y
    print(f"({x}, {y})")

# Bind the canvas to the mouse click event
canvas.bind('<Button-1>', show_coordinates)

# Start the tkinter mainloop
window.mainloop()
```

Anexo 3. Código de optimización

Código 8: Optimización de rutas y conflictos

```
import math
import heapq
from queue import PriorityQueue
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
from matplotlib.animation import FuncAnimation
import time



# Define the 10 nodes and their coordinates
nodes = {
    1: (1141, 565),          #'R32K5'   Pista 32R
    2: (1050, 528),          #'K5KA4'
    3: (1038, 432),          #'32RK4'
    4: (947, 396),           #'K4KC3'
    5: (886, 394),           #'KC2A27'
    6: (986, 364),           #'32RK3'
    7: (922, 362),           #'K3KB2'
    8: (887, 362),           #'KB2A28'
    9: (888, 559),           #'A27A25'
    10: (861, 559),          #'A27A25 2'
    11: (816, 559),          #'A25A24'
    12: (793, 559),          #'A24A23'
    13: (686, 559),          #'A23A22'
    14: (658, 559),          #'A22A21'
    15: (632, 559),          #'A21A20'
    16: (605, 559),          #'A20A19'
    17: (578, 559),          #'A19A18'
    18: (543, 559),          #'A18A17'
    19: (460, 559),          #'A17S2'
    20: (441, 559),          #'S2S3'
    21: (304, 559),          #'S3S4'
    22: (469, 608),          #A14A13
    23: (483, 635),          #J2A13
    24: (492, 655),          #A13LB
    25: (507, 676),          #A13LA
    26: (522, 699),          #A13L1
    27: (543, 723),          #A13A12
    28: (567, 754),          #A12L3 curva
    29: (614, 815),          #A12LF
    30: (658, 872),          #A11L5
    31: (732, 875),          #32LL4    Pista 32L
    32: (686, 708),          #L4L42
    33: (633, 638),          #L42L2
```

```
    34: (664, 788),          #32LL2
    35: (636, 684),          #L2 intermedio
    36: (719, 874),          #32LL5
    37: (664, 854),          #L5 intermedio
    38: (654, 783),          #32LLF
    39: (637, 762),          #32LL3
    40: (580, 737),          #L3 intermedio
    41: (597, 828),          #M11M12
    42: (489, 686),          #M12M13
    43: (473, 664),          #LBM13
    44: (453, 637),          #M13J3
    45: (440, 611),          #M13M14
    46: (442, 529),          #M14M16
    47: (478, 532),          #M16AZ3
    48: (493, 530),          #M16R7
    49: (517, 529),          #M17R6
    50: (542, 529),          #M17R5
    51: (578, 529),          #M18MD
    52: (606, 532),          #M19M20
    53: (632, 527),          #M20M21
    54: (660, 529),          #M21M22
    55: (685, 530),          #M22M23
    56: (793, 529),          #M23M24
    57: (817, 529),          #M24M25
    58: (860, 529),          #M25M27
    59: (886, 527),          #M25A27
    60: (864, 395),          #M27KC2
    61: (861, 360),          #M28KB2


}



# Define a function to calculate the Euclidean distance between two
points
def distance(node1, node2):
    x1, y1 = node1
    x2, y2 = node2
    equation = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
    return equation

# Define a function to get the neighbors of a node
# Define the neighbors of each node
neighbors = {
    1: [2, 3, 6],
    2: [4],
    3: [4],
    4: [5, 7],
    5: [59, 60],
    6: [7],
```

```
    7: [8],
    8: [5, 61],
    9: [10],
    10: [11, 58],
    11: [12, 57],
    12: [13, 56],
    13: [14, 55],
    14: [15, 54],
    15: [16, 53],
    16: [17, 52],
    17: [18, 51],
    18: [19, 50],
    19: [20, 22, 47],
    20: [21, 46],
    21: [],
    22: [23, 45],
    23: [24, 44],
    24: [25, 43],
    25: [26, 42],
    26: [27, 25],
    27: [28, 26],
    28: [29, 27],
    29: [30, 41],
    30: [29],
    31: [32, 36, 34, 38, 39],
    32: [33, 13],
    33: [15],
    34: [35, 38],
    35: [33],
    36: [37, 34],
    37: [30],
    38: [29],
    39: [40],
    40: [28, 27],
    41: [42],
    42: [43, 25],
    43: [44],
    44: [45, 23],
    45: [22, 20],
    46: [47],
    47: [19, 48],
    48: [49],
    49: [50],
    50: [18, 51],
    51: [17, 51],
    52: [16, 53],
    53: [15, 54],
    54: [14, 55],
    55: [13, 56],
    56: [12, 57],
    57: [11, 58],
    58: [10, 59, 60],
    59: [9],
```

```
    60: [5, 61],
    61: [8],

    }


def has_possible_route(start, goal, neighbors):
    visited = set()
    stack = [start]

    while stack:
        current = stack.pop()
        visited.add(current)
        if current == goal:
            return True
        for neighbor in neighbors.get(current, []):
            if neighbor not in visited:
                stack.append(neighbor)

    return False


# Call the A* algorithm to find the shortest path from nodes
start_node = 31
end_node = 61
start_node1 = 1
end_node1 = 29


# Check if there are possible routes
if has_possible_route(start_node, end_node, neighbors) and
has_possible_route(start_node1, end_node1, neighbors):

    # Define the A* algorithm
    def a_star(start, goal, nodes, neighbors):


        # Initialize the frontier with the start node
        frontier = [(0, start)]
        # Initialize the set of visited nodes
        visited = set()
        # Initialize the dictionary of parents
        parents = {start: None}
        # Initialize the dictionary of path costs
        path_costs = {start: 0}

        while frontier:
            # Get the node in the frontier with the lowest cost
            _, current = heapq.heappop(frontier)
            # If the current node is the goal, we're done
            if current == goal:
                break
            # Otherwise, expand the current node
```

```
            for neighbor in neighbors[current]:
                # Calculate the cost of the path from the start to
the neighbor through the current node
                new_cost = path_costs[current] +
distance(nodes[current], nodes[neighbor])
                # If the neighbor has not been visited yet, or if
the new cost is lower than the previous cost
                if neighbor not in visited or new_cost <
path_costs[neighbor]:
                    # Update the path cost and heuristic estimates
for the neighbor
                    path_costs[neighbor] = new_cost
                    heuristic = distance(nodes[neighbor],
nodes[goal])
                    priority = new_cost + heuristic
                    # Add the neighbor to the frontier
                    heapq.heappush(frontier, (priority, neighbor))
                    # Update the parent of the neighbor
                    parents[neighbor] = current
                    # Mark the neighbor as visited
                    visited.add(neighbor)

        # Reconstruct the path from the start to the goal
        path = []
        current = goal
        while current is not None:
            path.append(current)
            current = parents[current]
        path.reverse()


        # Return the path and the total cost
        return path, path_costs[goal]


    # Create a new dictionary without the specified node
    # nodes1 = {node: coords for node, coords in nodes.items() if
node != node_to_remove}
    # neighbors1 = {node: [neighbor for neighbor in neighbor_list if
neighbor != node_to_remove] for node, neighbor_list in
neighbors.items()}
    path, cost = a_star(start_node, end_node, nodes, neighbors)

    print(cost)
    path1, cost1 = a_star(start_node1, end_node1, nodes, neighbors)


    if path is None:
        print("No route found between nodes", {start_node}, "and",
{end_node})

    else:
        cost_total = cost  * 0.003521
```

```
        # Print the path and the total cost
        print(f"Shortest path: {path}")
        print(f"Total cost: {cost_total:.3f} km")


    # Def to return the common_nodes between 2 paths
    def common_nodes (path, path1):

        common_nodes = []

        for com_nodes in path:
            for com_nodes1 in path1:
                if com_nodes == com_nodes1:
                    common_nodes.append(com_nodes)

        # print("Common nodes:", common_nodes)

        return common_nodes



    # Load the background image
    bg_image = plt.imread("Captura.JPG")

    # Create a figure with the background image
    fig, ax = plt.subplots()
    ax.imshow(bg_image)




    # Create an initial line with no data
    line, = ax.plot([], [], color='blue')

    # Function to initialize the animation
    def init():
        line.set_data([], [])
        return line,

    all_x_values = []
    all_y_values = []
    all_x1_values = []
    all_y1_values = []
    all_x11_values = []
    all_y11_values = []
    all_x22_values = []
    all_y22_values = []

    # Step for iterations
    step = 270

    # Delay to start the simulation of the routes (blue - red)
```

```
    delay = 270
    delay1 = 0

    def delay_print(nodes, path, delay):
        # Delay of one of the roots defined by user (path)
        node6 = nodes[path[0]]
        x3 = np.linspace(node6[0], node6[0], delay)
        y3 = np.linspace(node6[1], node6[1], delay)

        return x3, y3


    x3, y3 = delay_print(nodes, path, delay)
    x2, y2 = delay_print(nodes, path1, delay1)




    # Cost to stop waiting for the other plane to pass
    delay11 = 150

    # Aircraft with priority (1 = red is priority)
    priority_red = 1


    # Def to obtain the cost_delta between each routes to calculate
the delay at the end of the simulation
    def cost_delta(cost, delay, cost1, delay1, delay11, alarm):
    # Equation to determine a brake at the end

        if priority_red == 1 and alarm < 130:

            cost_delta = cost + delay - cost1 - delay1 + delay11

        elif priority_red != 0 and alarm < 130:

            cost_delta = cost + delay - cost1 - delay1 - delay11

        else:

            cost_delta = cost + delay - cost1 - delay1


        return cost_delta




    # Def to compare the new_cost vs previous cost
```

```
    def new_cost_delta(new_cost, cost, delay11, new_path, path,
alarm):

        common_nodes3 = common_nodes(new_path, path)
        if not common_nodes3:

            new_cost_delta_val = new_cost - cost - delay11

        elif common_nodes3 and alarm > 130:

            new_cost_delta_val = new_cost - cost - delay11

        else:

             new_cost_delta_val = new_cost - cost

        return new_cost_delta_val



    def simu_delay(cost_delta1, nodes, path, path1):
        if cost_delta1 > 0:

            # Delay of one of the roots defined by user at the end
if route is longer
            node6 = nodes[path[-1]]
            delay2 = 0
            x4 = np.linspace(node6[0], node6[0], delay2)
            y4 = np.linspace(node6[1], node6[1], delay2)

            # Delay of one of the roots defined by user at the end
if route1 is longer
            node7 = nodes[path1[-1]]
            delay3 = int(cost_delta1)
            x5 = np.linspace(node7[0], node7[0], delay3)
            y5 = np.linspace(node7[1], node7[1], delay3)

        else:

            # Delay of one of the roots defined by user at the end
if route is longer
            node6 = nodes[path[-1]]
            delay2 = abs(int(cost_delta1))
            x4 = np.linspace(node6[0], node6[0], delay2)
            y4 = np.linspace(node6[1], node6[1], delay2)

            # Delay of one of the roots defined by user at the end
if route1 is longer
            node7 = nodes[path1[-1]]
            delay3 = 0
            x5 = np.linspace(node7[0], node7[0], delay3)
            y5 = np.linspace(node7[1], node7[1], delay3)
```

```
        return x4, y4, x5, y5




    # Define alarm
    def alarm (start_node, common_nodes1, start_node1, delay,
delay1, nodes_blue, neighbors_blue, nodes_red, neighbors_red):

        if common_nodes1:
            common_path, common_cost = a_star(start_node,
common_nodes1[0], nodes_blue, neighbors_blue)
            common_path1, common_cost1 = a_star(start_node1,
common_nodes1[0], nodes_red, neighbors_red)


            # Alarm in case of accident
            alarm = abs(common_cost + delay - common_cost1 - delay1)

        else:
            alarm = 1000

        return alarm


    def priority_path(path, path1, priority_red):
        if priority_red == 1:
            path_nopriority = path

        else:
            path_nopriority = path1
        return path_nopriority



    # Plot the nodes to join as small circles
    for node in path:
        ax.scatter(nodes[node][0], nodes[node][1], s=10,
color='blue')

    for node in path1:
        ax.scatter(nodes[node][0], nodes[node][1], s=10,
color='red')

    common_nodes1 = common_nodes (path, path1)
    alarm1 = alarm (start_node, common_nodes1, start_node1, delay,
delay1, nodes, neighbors, nodes, neighbors)
    print(f"alarma {alarm1}")

    # Node to remove from nodes nd neighbors and new set of nodes
```

```
and neighbors
    def selected_node_index(common_nodes, nodes, neighbors, path,
path1, priority_red):
        # Find the index of the selected node in the path
        path_nopriority = priority_path(path, path1, priority_red)
        selected_node_index1 =
path_nopriority.index(common_nodes[0])

        # Create a new dictionary without the specified node
        nodes1 = {node: coords for node, coords in nodes.items() if
node != common_nodes[0]}
        neighbors1 = {node: [neighbor for neighbor in neighbor_list
if neighbor != common_nodes[0]] for node, neighbor_list in
neighbors.items()}

        return selected_node_index1, nodes1, neighbors1



    if alarm1 < 130:


        # Find the index of the selected node in the path
        selected_node_index1, nodes1, neighbors1 =
selected_node_index(common_nodes1, nodes, neighbors, path, path1,
priority_red)




        if priority_red == 1 and has_possible_route(start_node,
end_node, neighbors1):



            # Obtaining the new path without the conflicting node
            new_path, new_cost = a_star(start_node, end_node,
nodes1, neighbors1)

            # Obtaining common_nodes
            common_nodes2 = common_nodes (new_path, path1)

            # Obtaining if there is an alarm
            alarm2 = alarm (start_node, common_nodes2, start_node1,
delay, delay1, nodes1, neighbors1, nodes, neighbors)

            print(f"alarma 2: {alarm2}")

            # Obtaining the coste delta (which route is faster, new
or old)
```

```
            new_cost_delta_val = new_cost_delta(new_cost, cost,
delay11, new_path, path, alarm2)

            print(f"nuevo coste: {new_cost_delta_val}")



            if new_cost_delta_val < 0 :

                selected_node_index2, nodes2, neighbors2 =
selected_node_index(common_nodes2, nodes1, neighbors1, new_path,
path1, priority_red)

                if alarm2 < 130 and has_possible_route(start_node,
end_node, neighbors2):


                    # Obtaining the new path without the conflicting
node
                    print(common_nodes2)
                    new_path1, new_cost1 = a_star(start_node,
end_node, nodes2, neighbors2)

                    # Obtaining common_nodes
                    common_nodes3 = common_nodes (new_path1, path1)


                    # Obtaining if there is an alarm
                    alarm3 = alarm (start_node, common_nodes3,
start_node1, delay, delay1, nodes2, neighbors2, nodes, neighbors)


                    # Obtaining the coste delta (which route is
faster, new or old)
                    new_cost_delta_val1 = new_cost_delta(new_cost1,
cost, delay11, new_path1, path, alarm3)



                    if new_cost_delta_val1 < 0 :

                        selected_node_index3, nodes3, neighbors3 =
selected_node_index(common_nodes3, nodes2, neighbors2, new_path1,
path1, priority_red)

                        if alarm3 < 130 and
has_possible_route(start_node, end_node, neighbors3):


                            # Obtaining the new path without the
conflicting node
```

```
                            new_path2, new_cost2 =
a_star(start_node, end_node, nodes3, neighbors3)

                            # Obtaining common_nodes
                            common_nodes4 = common_nodes (new_path2,
path1)


                            # Obtaining if there is an alarm
                            alarm4 = alarm (start_node,
common_nodes4, start_node1)



                            # Obtaining the coste delta (which route
is faster, new or old)
                            new_cost_delta_val2 =
new_cost_delta(new_cost2, cost, delay11, new_path2, path, alarm4)



                        else:

                            cost_delta1 = cost_delta(new_cost1,
delay, cost1, delay1, delay11, alarm3)
                            x4, y4, x5, y5 = simu_delay(cost_delta1,
nodes, new_path1, path1)


                                # Plot the nodes to join as small
circles
                            for node in new_path1:
                                ax.scatter(nodes[node][0],
nodes[node][1], s=10, color='green')

                            # Join the nodes with lines for path
                            for i in range(len(new_path1)-1):
                                node1 = nodes[new_path1[i]]
                                node2 = nodes[new_path1[i+1]]

                                speed = distance(node1,node2)
                                num_points=int(speed)
                                # Calculate the x and y coordinates
for the growing line
                                x = np.linspace(node1[0], node2[0],
num_points)
                                y = np.linspace(node1[1], node2[1],
num_points)

                                # Extend the list with the x values
from this iteration
                                all_x_values.extend(x)
                                all_y_values.extend(y)
```

```
                                all_x = np.concatenate((x3,
all_x_values, x4))
                                all_y = np.concatenate((y3,
all_y_values, y4))


                                # Create an initial line with no
data
                                line, = ax.plot([], [],
color='green')

                                # Function to initialize the
animation
                                def init():
                                    line.set_data([], [])
                                    return line,



                                # Function to update the animation
frame
                                def animate(frame):
                                    line.set_data(all_x[:frame],
all_y[:frame])
                                    line.set_color('green')
                                    line.set_linewidth(5)
                                    return line,

                                # Create the animation
                                ani = FuncAnimation(fig, animate,
init_func=init, frames=len(all_x), blit=False, interval=step)



                            # Join the nodes with lines for path1
                            for i in range(len(path1)-1):
                                node3 = nodes[path1[i]]
                                node4 = nodes[path1[i+1]]

                                speed1 = distance(node3,node4)

                                num_points1=int(speed1)
                                # Calculate the x and y coordinates
for the growing line
                                x1 = np.linspace(node3[0], node4[0],
num_points1)
                                y1 = np.linspace(node3[1], node4[1],
num_points1)
```

```
                                # Extend the list with the x values
from this iteration
                                all_x1_values.extend(x1)
                                all_y1_values.extend(y1)

                                all_x1 = np.concatenate((x2,
all_x1_values, x5))
                                all_y1 = np.concatenate((y2,
all_y1_values, y5))


                                # Create an initial line with no
data
                                line1, = ax.plot([], [],
color='red')

                                # Function to initialize the
animation
                                def init1():
                                    line1.set_data([], [])
                                    return line1,



                                # Function to update the animation
frame
                                def animate1(frame):

                                    line1.set_data(all_x1[:frame],
all_y1[:frame])
                                    line1.set_color('red')
                                    line1.set_linewidth(5)
                                    return line1,


                                # Create the animation
                                ani2 = FuncAnimation(fig, animate1,
init_func=init1, frames=len(all_x1), blit=False, interval=step)


                            # Show the plot
                            plt.show()


                    else:

                        cost_delta1 = cost_delta(cost, delay, cost1,
delay1, delay11, alarm2)
                        x4, y4, x5, y5 = simu_delay(cost_delta1,
nodes, path, path1)

                        path_nopriority = priority_path(path, path1,
```

```
priority_red)
                        # Split the path into two sections
                        first_section =
path_nopriority[:selected_node_index2 ]
                        second_section =
path_nopriority[selected_node_index2 - 1 :]

                        print("First section:", first_section)
                        print("Second section:", second_section)


                        for i in range(len(first_section)-1):
                            node1 = nodes[first_section[i]]
                            node2 = nodes[first_section[i+1]]

                            speed = distance(node1,node2)
                            num_points=int(speed)


                            # Calculate the x and y coordinates for
the growing line
                            x = np.linspace(node1[0], node2[0],
num_points)
                            y = np.linspace(node1[1], node2[1],
num_points)


                            # Extend the list with the x values from
this iteration
                            all_x_values.extend(x)
                            all_y_values.extend(y)



                        for i in range(len(second_section)-1):
                            node11 = nodes[second_section[i]]
                            node22 = nodes[second_section[i+1]]

                            speed2 = distance(node11, node22)
                            num_points2=int(speed2)
                            x11 = np.linspace(node11[0], node22[0],
num_points2)
                            y11 = np.linspace(node11[1], node22[1],
num_points2)

                            all_x11_values.extend(x11)
                            all_y11_values.extend(y11)



                            # Delay of one of the roots defined by
user
                            node_stop = nodes[second_section[0]]
```

```
                            x111 = np.linspace(node_stop[0],
node_stop[0], delay11)
                            y111 = np.linspace(node_stop[1],
node_stop[1], delay11)

                            all_x = np.concatenate((x3,
all_x_values, x111, all_x11_values, x4))
                            all_y = np.concatenate((y3,
all_y_values, y111, all_y11_values, y4))


                            # Create an initial line with no data
                            line, = ax.plot([], [], color='blue')

                            # Function to initialize the animation
                            def init():
                                line.set_data([], [])
                                return line,



                            # Function to update the animation frame
                            def animate(frame):
                                line.set_data(all_x[:frame],
all_y[:frame])
                                line.set_color('blue')
                                line.set_linewidth(5)
                                return line,

                            # Create the animation
                            ani = FuncAnimation(fig, animate,
init_func=init, frames=len(all_x), blit=False, interval=step)



                        # Join the nodes with lines for path1

                        for i in range(len(path1)-1):
                            node3 = nodes[path1[i]]
                            node4 = nodes[path1[i+1]]

                            speed1 = distance(node3,node4)

                            num_points1=int(speed1)
                            # Calculate the x and y coordinates for
the growing line
                            x1 = np.linspace(node3[0], node4[0],
num_points1)
                            y1 = np.linspace(node3[1], node4[1],
num_points1)
```

```
                            # Extend the list with the x values from
this iteration
                            all_x1_values.extend(x1)
                            all_y1_values.extend(y1)

                            all_x1 = np.concatenate((x2,
all_x1_values, x5))
                            all_y1 = np.concatenate((y2,
all_y1_values, y5))


                            # Create an initial line with no data
                            line1, = ax.plot([], [], color='red')

                            # Function to initialize the animation
                            def init1():
                                line1.set_data([], [])
                                return line1,



                            # Function to update the animation frame
                            def animate1(frame):

                                line1.set_data(all_x1[:frame],
all_y1[:frame])
                                line1.set_color('red')
                                line1.set_linewidth(5)
                                return line1,


                            # Create the animation
                            ani2 = FuncAnimation(fig, animate1,
init_func=init1, frames=len(all_x1), blit=False, interval=step)


                        # Show the plot
                        plt.show()

                else:

                    cost_delta1 = cost_delta(new_cost, delay, cost1,
delay1, delay11, alarm2)
                    x4, y4, x5, y5 = simu_delay(cost_delta1, nodes,
new_path, path1)

                    print(f"coste ruta 3: {new_cost*0.003521}")

                    # Plot the nodes to join as small circles
                    for node in new_path:
                        ax.scatter(nodes[node][0], nodes[node][1],
s=10, color='green')
```

```
                    # Join the nodes with lines for path
                    for i in range(len(new_path)-1):
                        node1 = nodes[new_path[i]]
                        node2 = nodes[new_path[i+1]]

                        speed = distance(node1,node2)
                        num_points=int(speed)
                        # Calculate the x and y coordinates for the
growing line
                        x = np.linspace(node1[0], node2[0],
num_points)
                        y = np.linspace(node1[1], node2[1],
num_points)

                        # Extend the list with the x values from
this iteration
                        all_x_values.extend(x)
                        all_y_values.extend(y)

                        all_x = np.concatenate((x3, all_x_values,
x4))
                        all_y = np.concatenate((y3, all_y_values,
y4))


                        # Create an initial line with no data
                        line, = ax.plot([], [], color='green')

                        # Function to initialize the animation
                        def init():
                            line.set_data([], [])
                            return line,



                        # Function to update the animation frame
                        def animate(frame):
                            line.set_data(all_x[:frame],
all_y[:frame])
                            line.set_color('green')
                            line.set_linewidth(5)
                            return line,

                        # Create the animation
                        ani = FuncAnimation(fig, animate,
init_func=init, frames=len(all_x), blit=False, interval=step)



                    # Join the nodes with lines for path1
                    for i in range(len(path1)-1):
```

```
                        node3 = nodes[path1[i]]
                        node4 = nodes[path1[i+1]]

                        speed1 = distance(node3,node4)

                        num_points1=int(speed1)
                        # Calculate the x and y coordinates for the
growing line
                        x1 = np.linspace(node3[0], node4[0],
num_points1)
                        y1 = np.linspace(node3[1], node4[1],
num_points1)



                        # Extend the list with the x values from
this iteration
                        all_x1_values.extend(x1)
                        all_y1_values.extend(y1)

                        all_x1 = np.concatenate((x2, all_x1_values,
x5))
                        all_y1 = np.concatenate((y2, all_y1_values,
y5))



                        # Create an initial line with no data
                        line1, = ax.plot([], [], color='red')

                        # Function to initialize the animation
                        def init1():
                            line1.set_data([], [])
                            return line1,



                        # Function to update the animation frame
                        def animate1(frame):

                            line1.set_data(all_x1[:frame],
all_y1[:frame])
                            line1.set_color('red')
                            line1.set_linewidth(5)
                            return line1,


                        # Create the animation
                        ani2 = FuncAnimation(fig, animate1,
init_func=init1, frames=len(all_x1), blit=False, interval=step)

                    # Show the plot
```

```
                    plt.show()



            else:


                cost_delta1 = cost_delta(cost, delay, cost1, delay1,
delay11, alarm1)
                x4, y4, x5, y5 = simu_delay(cost_delta1, nodes,
path, path1)

                path_nopriority = priority_path(path, path1,
priority_red)
                # Split the path into two sections
                first_section =
path_nopriority[:selected_node_index1 ]
                second_section =
path_nopriority[selected_node_index1 - 1 :]

                print("First section:", first_section)
                print("Second section:", second_section)


                for i in range(len(first_section)-1):
                    node1 = nodes[first_section[i]]
                    node2 = nodes[first_section[i+1]]

                    speed = distance(node1,node2)
                    num_points=int(speed)


                    # Calculate the x and y coordinates for the
growing line
                    x = np.linspace(node1[0], node2[0], num_points)
                    y = np.linspace(node1[1], node2[1], num_points)


                    # Extend the list with the x values from this
iteration
                    all_x_values.extend(x)
                    all_y_values.extend(y)



                for i in range(len(second_section)-1):
                    node11 = nodes[second_section[i]]
                    node22 = nodes[second_section[i+1]]

                    speed2 = distance(node11, node22)
                    num_points2=int(speed2)
                    x11 = np.linspace(node11[0], node22[0],
```

```
num_points2)
                    y11 = np.linspace(node11[1], node22[1],
num_points2)

                    all_x11_values.extend(x11)
                    all_y11_values.extend(y11)



                    # Delay of one of the roots defined by user
                    node_stop = nodes[second_section[0]]
                    x111 = np.linspace(node_stop[0], node_stop[0],
delay11)
                    y111 = np.linspace(node_stop[1], node_stop[1],
delay11)

                    all_x = np.concatenate((x3, all_x_values, x111,
all_x11_values, x4))
                    all_y = np.concatenate((y3, all_y_values, y111,
all_y11_values, y4))


                    # Create an initial line with no data
                    line, = ax.plot([], [], color='blue')

                    # Function to initialize the animation
                    def init():
                        line.set_data([], [])
                        return line,



                    # Function to update the animation frame
                    def animate(frame):
                        line.set_data(all_x[:frame], all_y[:frame])
                        line.set_color('blue')
                        line.set_linewidth(5)
                        return line,

                    # Create the animation
                    ani = FuncAnimation(fig, animate,
init_func=init, frames=len(all_x), blit=False, interval=step)



                # Join the nodes with lines for path1

                for i in range(len(path1)-1):
                    node3 = nodes[path1[i]]
                    node4 = nodes[path1[i+1]]

                    speed1 = distance(node3,node4)
```

```
                    num_points1=int(speed1)
                    # Calculate the x and y coordinates for the
growing line
                    x1 = np.linspace(node3[0], node4[0],
num_points1)
                    y1 = np.linspace(node3[1], node4[1],
num_points1)



                    # Extend the list with the x values from this
iteration
                    all_x1_values.extend(x1)
                    all_y1_values.extend(y1)

                    all_x1 = np.concatenate((x2, all_x1_values, x5))
                    all_y1 = np.concatenate((y2, all_y1_values, y5))


                    # Create an initial line with no data
                    line1, = ax.plot([], [], color='red')

                    # Function to initialize the animation
                    def init1():
                        line1.set_data([], [])
                        return line1,



                    # Function to update the animation frame
                    def animate1(frame):

                        line1.set_data(all_x1[:frame],
all_y1[:frame])
                        line1.set_color('red')
                        line1.set_linewidth(5)
                        return line1,


                    # Create the animation
                    ani2 = FuncAnimation(fig, animate1,
init_func=init1, frames=len(all_x1), blit=False, interval=step)


                # Show the plot
                plt.show()

        elif priority_red == 1 and not
has_possible_route(start_node, end_node, neighbors1):


            cost_delta1 = cost_delta(cost, delay, cost1, delay1,
```

```
delay11, alarm1)
            x4, y4, x5, y5 = simu_delay(cost_delta1, nodes, path,
path1)


            path_nopriority = priority_path(path, path1,
priority_red)
            # Split the path into two sections
            first_section = path_nopriority[:selected_node_index1 ]
            second_section = path_nopriority[selected_node_index1 -
1 :]

            print("First section:", first_section)
            print("Second section:", second_section)


            for i in range(len(first_section)-1):
                node1 = nodes[first_section[i]]
                node2 = nodes[first_section[i+1]]


                speed = distance(node1,node2)
                num_points=int(speed)


                # Calculate the x and y coordinates for the growing
line
                x = np.linspace(node1[0], node2[0], num_points)
                y = np.linspace(node1[1], node2[1], num_points)


                # Extend the list with the x values from this
iteration
                all_x_values.extend(x)
                all_y_values.extend(y)



            for i in range(len(second_section)-1):
                node11 = nodes[second_section[i]]
                node22 = nodes[second_section[i+1]]

                speed2 = distance(node11, node22)
                num_points2=int(speed2)
                x11 = np.linspace(node11[0], node22[0], num_points2)
                y11 = np.linspace(node11[1], node22[1], num_points2)

                all_x11_values.extend(x11)
                all_y11_values.extend(y11)



                # Delay of one of the roots defined by user
```

```
                node_stop = nodes[second_section[0]]
                x111 = np.linspace(node_stop[0], node_stop[0],
delay11)
                y111 = np.linspace(node_stop[1], node_stop[1],
delay11)

                all_x = np.concatenate((x3, all_x_values, x111,
all_x11_values, x4))
                all_y = np.concatenate((y3, all_y_values, y111,
all_y11_values, y4))


                # Create an initial line with no data
                line, = ax.plot([], [], color='blue')

                # Function to initialize the animation
                def init():
                    line.set_data([], [])
                    return line,



                # Function to update the animation frame
                def animate(frame):
                    line.set_data(all_x[:frame], all_y[:frame])
                    line.set_color('blue')
                    line.set_linewidth(5)
                    return line,

                # Create the animation
                ani = FuncAnimation(fig, animate, init_func=init,
frames=len(all_x), blit=False, interval=step)



            # Join the nodes with lines for path1

            for i in range(len(path1)-1):
                node3 = nodes[path1[i]]
                node4 = nodes[path1[i+1]]

                speed1 = distance(node3,node4)

                num_points1=int(speed1)
                # Calculate the x and y coordinates for the growing
line
                x1 = np.linspace(node3[0], node4[0], num_points1)
                y1 = np.linspace(node3[1], node4[1], num_points1)



                # Extend the list with the x values from this
```

```
iteration
                all_x1_values.extend(x1)
                all_y1_values.extend(y1)

                all_x1 = np.concatenate((x2, all_x1_values, x5))
                all_y1 = np.concatenate((y2, all_y1_values, y5))


                # Create an initial line with no data
                line1, = ax.plot([], [], color='red')

                # Function to initialize the animation
                def init1():
                    line1.set_data([], [])
                    return line1,



                # Function to update the animation frame
                def animate1(frame):

                    line1.set_data(all_x1[:frame], all_y1[:frame])
                    line1.set_color('red')
                    line1.set_linewidth(5)
                    return line1,


                # Create the animation
                ani2 = FuncAnimation(fig, animate1, init_func=init1,
frames=len(all_x1), blit=False, interval=step)


            # Show the plot
            plt.show()



        elif priority_red != 1 and has_possible_route(start_node1,
end_node1, neighbors1):


            # Obtaining the new path without the conflicting node
            new_path, new_cost = a_star(start_node1, end_node1,
nodes1, neighbors1)



            # Obtaining common_nodes
            common_nodes2 = common_nodes (path, new_path)


            # Obtaining if there is an alarm
```

```
            alarm2 = alarm (start_node, common_nodes2, start_node1,
delay, delay1, nodes, neighbors, nodes1, neighbors1)

            # Obtaining the coste delta (which route is faster, new
or old)
            new_cost_delta_val = new_cost_delta(new_cost, cost1,
delay11, new_path, path1, alarm2)




            if new_cost_delta_val < 0 :

                selected_node_index2, nodes2, neighbors2 =
selected_node_index(common_nodes2, nodes1, neighbors1, path,
new_path, priority_red)

                if alarm2 < 130 and has_possible_route(start_node1,
end_node1, neighbors2):



                    # Obtaining the new path without the conflicting
node
                    print(common_nodes2)
                    new_path1, new_cost1 = a_star(start_node1,
end_node1, nodes2, neighbors2)


                    # Obtaining common_nodes
                    common_nodes3 = common_nodes (path, new_path1)


                    # Obtaining if there is an alarm
                    alarm3 = alarm (start_node, common_nodes3,
start_node1, delay, delay1, nodes, neighbors, nodes2, neighbors2)

                    # Obtaining the coste delta (which route is
faster, new or old)
                    new_cost_delta_val1 = new_cost_delta(new_cost1,
cost1, delay11, new_path1, path1, alarm3)




                    if new_cost_delta_val1 < 0 :

                        selected_node_index3, nodes3, neighbors3 =
selected_node_index(common_nodes3, nodes2, neighbors2, path,
```

```
new_path1, priority_red)

                        if alarm3 < 130 and
has_possible_route(start_node1, end_node1, neighbors3):



                            # Obtaining the new path without the
conflicting node
                            new_path2, new_cost2 =
a_star(start_node1, end_node1, nodes3, neighbors3)


                            # Obtaining common_nodes
                            common_nodes4 = common_nodes (new_path2,
path)


                            # Obtaining if there is an alarm
                            alarm4 = alarm (start_node,
common_nodes4, start_node1)

                            # Obtaining the coste delta (which route
is faster, new or old)
                            new_cost_delta_val2 =
new_cost_delta(new_cost2, cost1, delay11, new_path2, path1, alarm4)



                        else:

                            # Plot the nodes to join as small
circles
                            for node in new_path1:
                                ax.scatter(nodes[node][0],
nodes[node][1], s=10, color='green')

                            cost_delta1 = cost_delta(cost, delay,
new_cost1, delay1, delay11, alarm3)
                            x4, y4, x5, y5 = simu_delay(cost_delta1,
nodes, path, new_path1)

                            # Join the nodes with lines for path
                            for i in range(len(path)-1):
                                node1 = nodes[path[i]]
                                node2 = nodes[path[i+1]]

                                speed = distance(node1,node2)
                                num_points=int(speed)
                                # Calculate the x and y coordinates
for the growing line
                                x = np.linspace(node1[0], node2[0],
```

```
num_points)
                                y = np.linspace(node1[1], node2[1],
num_points)

                                # Extend the list with the x values
from this iteration
                                all_x_values.extend(x)
                                all_y_values.extend(y)


                                all_x = np.concatenate((x3,
all_x_values, x4))
                                all_y = np.concatenate((y3,
all_y_values, y4))


                                # Create an initial line with no
data
                                line, = ax.plot([], [],
color='blue')

                                # Function to initialize the
animation
                                def init():
                                    line.set_data([], [])
                                    return line,



                                # Function to update the animation
frame
                                def animate(frame):
                                    line.set_data(all_x[:frame],
all_y[:frame])
                                    line.set_color('blue')
                                    line.set_linewidth(5)
                                    return line,

                                # Create the animation
                                ani = FuncAnimation(fig, animate,
init_func=init, frames=len(all_x), blit=False, interval=step)



                            # Join the nodes with lines for path1
                            for i in range(len(new_path1)-1):
                                node3 = nodes[new_path1[i]]
                                node4 = nodes[new_path1[i+1]]

                                speed1 = distance(node3,node4)

                                num_points1=int(speed1)
                                # Calculate the x and y coordinates
```

```
for the growing line
                                x1 = np.linspace(node3[0], node4[0],
num_points1)
                                y1 = np.linspace(node3[1], node4[1],
num_points1)



                                # Extend the list with the x values
from this iteration
                                all_x1_values.extend(x1)
                                all_y1_values.extend(y1)

                                all_x1 = np.concatenate((x2,
all_x1_values, x5))
                                all_y1 = np.concatenate((y2,
all_y1_values, y5))


                                # Create an initial line with no
data
                                line1, = ax.plot([], [],
color='green')

                                # Function to initialize the
animation
                                def init1():
                                    line1.set_data([], [])
                                    return line1,



                                # Function to update the animation
frame
                                def animate1(frame):

                                    line1.set_data(all_x1[:frame],
all_y1[:frame])
                                    line1.set_color('green')
                                    line1.set_linewidth(5)
                                    return line1,


                                # Create the animation
                                ani2 = FuncAnimation(fig, animate1,
init_func=init1, frames=len(all_x1), blit=False, interval=step)


                            # Show the plot
                            plt.show()
```

```
                    else:


                        cost_delta1 = cost_delta(cost, delay,
new_cost, delay1, delay11, alarm2)
                        x4, y4, x5, y5 = simu_delay(cost_delta1,
nodes, path, new_path)


                        path_nopriority = priority_path(path, path1,
priority_red)
                        # Split the path into two sections
                        first_section =
path_nopriority[:selected_node_index2 ]
                        second_section =
path_nopriority[selected_node_index2 - 1 :]

                        print("First section:", first_section)
                        print("Second section:", second_section)


                        for i in range(len(first_section)-1):
                            node1 = nodes[first_section[i]]
                            node2 = nodes[first_section[i+1]]

                            speed = distance(node1,node2)
                            num_points=int(speed)


                            # Calculate the x and y coordinates for
the growing line
                            x = np.linspace(node1[0], node2[0],
num_points)
                            y = np.linspace(node1[1], node2[1],
num_points)


                            # Extend the list with the x values from
this iteration
                            all_x_values.extend(x)
                            all_y_values.extend(y)



                        for i in range(len(second_section)-1):
                            node11 = nodes[second_section[i]]
                            node22 = nodes[second_section[i+1]]

                            speed2 = distance(node11, node22)
                            num_points2=int(speed2)
                            x11 = np.linspace(node11[0], node22[0],
num_points2)
                            y11 = np.linspace(node11[1], node22[1],
```

```
num_points2)

                            all_x11_values.extend(x11)
                            all_y11_values.extend(y11)



                            # Delay of one of the roots defined by
user
                            node_stop = nodes[second_section[0]]
                            x111 = np.linspace(node_stop[0],
node_stop[0], delay11)
                            y111 = np.linspace(node_stop[1],
node_stop[1], delay11)

                            all_x = np.concatenate((x3,
all_x_values, x111, all_x11_values, x4))
                            all_y = np.concatenate((y3,
all_y_values, y111, all_y11_values, y4))


                            # Create an initial line with no data
                            line, = ax.plot([], [], color='red')

                            # Function to initialize the animation
                            def init():
                                line.set_data([], [])
                                return line,



                            # Function to update the animation frame
                            def animate(frame):
                                line.set_data(all_x[:frame],
all_y[:frame])
                                line.set_color('red')
                                line.set_linewidth(5)
                                return line,

                            # Create the animation
                            ani = FuncAnimation(fig, animate,
init_func=init, frames=len(all_x), blit=False, interval=step)



                        # Join the nodes with lines for path1

                        for i in range(len(path)-1):
                            node3 = nodes[path[i]]
                            node4 = nodes[path[i+1]]

                            speed1 = distance(node3,node4)
```

```
                            num_points1=int(speed1)
                            # Calculate the x and y coordinates for
the growing line
                            x1 = np.linspace(node3[0], node4[0],
num_points1)
                            y1 = np.linspace(node3[1], node4[1],
num_points1)



                            # Extend the list with the x values from
this iteration
                            all_x1_values.extend(x1)
                            all_y1_values.extend(y1)

                            all_x1 = np.concatenate((x2,
all_x1_values, x5))
                            all_y1 = np.concatenate((y2,
all_y1_values, y5))


                            # Create an initial line with no data
                            line1, = ax.plot([], [], color='blue')

                            # Function to initialize the animation
                            def init1():
                                line1.set_data([], [])
                                return line1,



                            # Function to update the animation frame
                            def animate1(frame):

                                line1.set_data(all_x1[:frame],
all_y1[:frame])
                                line1.set_color('blue')
                                line1.set_linewidth(5)
                                return line1,


                            # Create the animation
                            ani2 = FuncAnimation(fig, animate1,
init_func=init1, frames=len(all_x1), blit=False, interval=step)


                        # Show the plot
                        plt.show()

                else:

                        # Plot the nodes to join as small circles
```

```
                    for node in new_path:
                        ax.scatter(nodes[node][0], nodes[node][1],
s=10, color='green')

                    cost_delta1 = cost_delta(cost, delay, new_cost,
delay1, delay11, alarm2)
                    x4, y4, x5, y5 = simu_delay(cost_delta1, nodes,
path, new_path)

                    # Join the nodes with lines for path
                    for i in range(len(path)-1):
                        node1 = nodes[path[i]]
                        node2 = nodes[path[i+1]]

                        speed = distance(node1,node2)
                        num_points=int(speed)
                        # Calculate the x and y coordinates for the
growing line
                        x = np.linspace(node1[0], node2[0],
num_points)
                        y = np.linspace(node1[1], node2[1],
num_points)

                        # Extend the list with the x values from
this iteration
                        all_x_values.extend(x)
                        all_y_values.extend(y)

                        all_x = np.concatenate((x3, all_x_values,
x4))
                        all_y = np.concatenate((y3, all_y_values,
y4))


                        # Create an initial line with no data
                        line, = ax.plot([], [], color='blue')

                        # Function to initialize the animation
                        def init():
                            line.set_data([], [])
                            return line,



                        # Function to update the animation frame
                        def animate(frame):
                            line.set_data(all_x[:frame],
all_y[:frame])
                            line.set_color('blue')
                            line.set_linewidth(5)
                            return line,

                        # Create the animation
```

```
                        ani = FuncAnimation(fig, animate,
init_func=init, frames=len(all_x), blit=False, interval=step)



                    # Join the nodes with lines for path1
                    for i in range(len(new_path)-1):
                        node3 = nodes[new_path[i]]
                        node4 = nodes[new_path[i+1]]

                        speed1 = distance(node3,node4)

                        num_points1=int(speed1)
                        # Calculate the x and y coordinates for the
growing line
                        x1 = np.linspace(node3[0], node4[0],
num_points1)
                        y1 = np.linspace(node3[1], node4[1],
num_points1)



                        # Extend the list with the x values from
this iteration
                        all_x1_values.extend(x1)
                        all_y1_values.extend(y1)

                        all_x1 = np.concatenate((x2, all_x1_values,
x5))
                        all_y1 = np.concatenate((y2, all_y1_values,
y5))


                        # Create an initial line with no data
                        line1, = ax.plot([], [], color='green')

                        # Function to initialize the animation
                        def init1():
                            line1.set_data([], [])
                            return line1,



                        # Function to update the animation frame
                        def animate1(frame):

                            line1.set_data(all_x1[:frame],
all_y1[:frame])
                            line1.set_color('green')
                            line1.set_linewidth(5)
                            return line1,
```

```
                        # Create the animation
                        ani2 = FuncAnimation(fig, animate1,
init_func=init1, frames=len(all_x1), blit=False, interval=step)

                    # Show the plot
                    plt.show()



            else:

                cost_delta1 = cost_delta(cost, delay, cost1, delay1,
delay11, alarm1)
                x4, y4, x5, y5 = simu_delay(cost_delta1, nodes,
path, path1)


                path_nopriority = priority_path(path, path1,
priority_red)
                # Split the path into two sections
                first_section =
path_nopriority[:selected_node_index1 ]
                second_section =
path_nopriority[selected_node_index1 - 1 :]

                print("First section:", first_section)
                print("Second section:", second_section)


                for i in range(len(first_section)-1):
                    node1 = nodes[first_section[i]]
                    node2 = nodes[first_section[i+1]]

                    speed = distance(node1,node2)
                    num_points=int(speed)


                    # Calculate the x and y coordinates for the
growing line
                    x = np.linspace(node1[0], node2[0], num_points)
                    y = np.linspace(node1[1], node2[1], num_points)


                    # Extend the list with the x values from this
iteration
                    all_x_values.extend(x)
                    all_y_values.extend(y)



                for i in range(len(second_section)-1):
```

```
                    node11 = nodes[second_section[i]]
                    node22 = nodes[second_section[i+1]]

                    speed2 = distance(node11, node22)
                    num_points2=int(speed2)
                    x11 = np.linspace(node11[0], node22[0],
num_points2)
                    y11 = np.linspace(node11[1], node22[1],
num_points2)

                    all_x11_values.extend(x11)
                    all_y11_values.extend(y11)



                    # Delay of one of the roots defined by user
                    node_stop = nodes[second_section[0]]
                    x111 = np.linspace(node_stop[0], node_stop[0],
delay11)
                    y111 = np.linspace(node_stop[1], node_stop[1],
delay11)

                    all_x = np.concatenate((x3, all_x_values, x111,
all_x11_values, x4))
                    all_y = np.concatenate((y3, all_y_values, y111,
all_y11_values, y4))


                    # Create an initial line with no data
                    line, = ax.plot([], [], color='red')

                    # Function to initialize the animation
                    def init():
                        line.set_data([], [])
                        return line,



                    # Function to update the animation frame
                    def animate(frame):
                        line.set_data(all_x[:frame], all_y[:frame])
                        line.set_color('red')
                        line.set_linewidth(5)
                        return line,

                    # Create the animation
                    ani = FuncAnimation(fig, animate,
init_func=init, frames=len(all_x), blit=False, interval=step)



                # Join the nodes with lines for path1
```

```
                for i in range(len(path)-1):
                    node3 = nodes[path[i]]
                    node4 = nodes[path[i+1]]

                    speed1 = distance(node3,node4)

                    num_points1=int(speed1)
                    # Calculate the x and y coordinates for the
growing line
                    x1 = np.linspace(node3[0], node4[0],
num_points1)
                    y1 = np.linspace(node3[1], node4[1],
num_points1)



                    # Extend the list with the x values from this
iteration
                    all_x1_values.extend(x1)
                    all_y1_values.extend(y1)

                    all_x1 = np.concatenate((x2, all_x1_values, x5))
                    all_y1 = np.concatenate((y2, all_y1_values, y5))


                    # Create an initial line with no data
                    line1, = ax.plot([], [], color='blue')

                    # Function to initialize the animation
                    def init1():
                        line1.set_data([], [])
                        return line1,



                    # Function to update the animation frame
                    def animate1(frame):

                        line1.set_data(all_x1[:frame],
all_y1[:frame])
                        line1.set_color('blue')
                        line1.set_linewidth(5)
                        return line1,


                    # Create the animation
                    ani2 = FuncAnimation(fig, animate1,
init_func=init1, frames=len(all_x1), blit=False, interval=step)


                # Show the plot
                plt.show()
```

```
        elif priority_red != 1 and not
has_possible_route(start_node1, end_node1, neighbors1):

            cost_delta1 = cost_delta(cost, delay, cost1, delay1,
delay11, alarm1)
            x4, y4, x5, y5 = simu_delay(cost_delta1, nodes, path,
path1)


            # Join the nodes with lines for path
            for i in range(len(path)-1):
                node1 = nodes[path[i]]
                node2 = nodes[path[i+1]]

                speed = distance(node1,node2)
                num_points=int(speed)
                # Calculate the x and y coordinates for the growing
line
                x = np.linspace(node1[0], node2[0], num_points)
                y = np.linspace(node1[1], node2[1], num_points)

                # Extend the list with the x values from this
iteration
                all_x_values.extend(x)
                all_y_values.extend(y)

                all_x = np.concatenate((x3, all_x_values, x4))
                all_y = np.concatenate((y3, all_y_values, y4))


                # Create an initial line with no data
                line, = ax.plot([], [], color='blue')

                # Function to initialize the animation
                def init():
                    line.set_data([], [])
                    return line,



                # Function to update the animation frame
                def animate(frame):
                    line.set_data(all_x[:frame], all_y[:frame])
                    line.set_color('blue')
                    line.set_linewidth(5)
                    return line,

                # Create the animation
                ani = FuncAnimation(fig, animate, init_func=init,
```

```
frames=len(all_x), blit=False, interval=step)


            path_nopriority = priority_path(path, path1,
priority_red)
            # Split the path into two sections
            first_section = path_nopriority[:selected_node_index1 ]
            second_section = path_nopriority[selected_node_index1 -
1 :]

            print("First section:", first_section)
            print("Second section:", second_section)

            # Join the nodes with lines for path1
            for i in range(len(first_section)-1):
                node3 = nodes[first_section[i]]
                node4 = nodes[first_section[i+1]]

                speed1 = distance(node3,node4)

                num_points1=int(speed1)
                # Calculate the x and y coordinates for the growing
line
                x1 = np.linspace(node3[0], node4[0], num_points1)
                y1 = np.linspace(node3[1], node4[1], num_points1)



                # Extend the list with the x values from this
iteration
                all_x1_values.extend(x1)
                all_y1_values.extend(y1)


            for i in range(len(second_section)-1):
                node11 = nodes[second_section[i]]
                node22 = nodes[second_section[i+1]]

                speed2 = distance(node11, node22)
                num_points2=int(speed2)
                x11 = np.linspace(node11[0], node22[0], num_points2)
                y11 = np.linspace(node11[1], node22[1], num_points2)

                all_x11_values.extend(x11)
                all_y11_values.extend(y11)


                # Delay of one of the roots defined by user
                node_stop = nodes[second_section[0]]
                x111 = np.linspace(node_stop[0], node_stop[0],
delay11)
                y111 = np.linspace(node_stop[1], node_stop[1],
delay11)
```

```
                all_x1 = np.concatenate((x2, all_x1_values, x111,
all_x11_values, x5))
                all_y1 = np.concatenate((y2, all_y1_values, y111,
all_y11_values, y5))


                # Create an initial line with no data
                line1, = ax.plot([], [], color='red')

                # Function to initialize the animation
                def init1():
                    line1.set_data([], [])
                    return line1,



                # Function to update the animation frame
                def animate1(frame):

                    line1.set_data(all_x1[:frame], all_y1[:frame])
                    line1.set_color('red')
                    line1.set_linewidth(5)
                    return line1,


                # Create the animation
                ani2 = FuncAnimation(fig, animate1, init_func=init1,
frames=len(all_x1), blit=False, interval=step)

            # Show the plot
            plt.show()


    else:


        cost_delta1 = cost_delta(cost, delay, cost1, delay1,
delay11, alarm1)
        x4, y4, x5, y5 = simu_delay(cost_delta1, nodes, path, path1)

        # Join the nodes with lines for path
        for i in range(len(path)-1):
            node1 = nodes[path[i]]
            node2 = nodes[path[i+1]]

            speed = distance(node1,node2)
            num_points=int(speed)
            # Calculate the x and y coordinates for the growing line
            x = np.linspace(node1[0], node2[0], num_points)
            y = np.linspace(node1[1], node2[1], num_points)

            # Extend the list with the x values from this iteration
```

```
            all_x_values.extend(x)
            all_y_values.extend(y)

            all_x = np.concatenate((x3, all_x_values, x4))
            all_y = np.concatenate((y3, all_y_values, y4))


            # Create an initial line with no data
            line, = ax.plot([], [], color='blue')

            # Function to initialize the animation
            def init():
                line.set_data([], [])
                return line,



            # Function to update the animation frame
            def animate(frame):
                line.set_data(all_x[:frame], all_y[:frame])
                line.set_color('blue')
                line.set_linewidth(5)
                return line,

            # Create the animation
            ani = FuncAnimation(fig, animate, init_func=init,
frames=len(all_x), blit=False, interval=step)




        # Join the nodes with lines for path1
        for i in range(len(path1)-1):
            node3 = nodes[path1[i]]
            node4 = nodes[path1[i+1]]

            speed1 = distance(node3,node4)

            num_points1=int(speed1)
            # Calculate the x and y coordinates for the growing line
            x1 = np.linspace(node3[0], node4[0], num_points1)
            y1 = np.linspace(node3[1], node4[1], num_points1)



            # Extend the list with the x values from this iteration
            all_x1_values.extend(x1)
            all_y1_values.extend(y1)

            all_x1 = np.concatenate((x2, all_x1_values, x5))
            all_y1 = np.concatenate((y2, all_y1_values, y5))
```

```
            # Create an initial line with no data
            line1, = ax.plot([], [], color='red')

            # Function to initialize the animation
            def init1():
                line1.set_data([], [])
                return line1,



            # Function to update the animation frame
            def animate1(frame):

                line1.set_data(all_x1[:frame], all_y1[:frame])
                line1.set_color('red')
                line1.set_linewidth(5)
                return line1,


            # Create the animation
            ani2 = FuncAnimation(fig, animate1, init_func=init1,
frames=len(all_x1), blit=False, interval=step)



        # Show the plot
        plt.show()

elif not has_possible_route(start_node, end_node, neighbors) and not
has_possible_route(start_node1, end_node1, neighbors):

    print("There is no possible route between", start_node, "and",
end_node)
    print("There is no possible route between", start_node1, "and",
end_node1)


elif not has_possible_route(start_node, end_node, neighbors) and
has_possible_route(start_node1, end_node1, neighbors):

    print("There is no possible route between", start_node, "and",
end_node)
    print("There is possible route between", start_node1, "and",
end_node1)

elif has_possible_route(start_node, end_node, neighbors) and not
has_possible_route(start_node1, end_node1, neighbors):

    print("There is possible route between", start_node, "and",
end_node)
    print("There is no possible route between", start_node1, "and",
end_node1)
```