



Universidad de
Oviedo



ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

ÁREA DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

DISEÑO DE SISTEMA DE ALINEAMIENTO AUDIO-PARTITURA SEMIAUTOMÁTICO

D. FERNÁNDEZ COCAÑO, Iván
TUTOR: D. MUÑOZ MONTORO, Antonio Jesús

FECHA: Julio, 2022

Agradecimientos

Me gustaría transmitir mi agradecimiento a todos aquellos que me ayudaron y apoyaron a lo largo de mi etapa universitaria, ya que sin su ánimo y apoyo no habría sido posible finalizar mis estudios.

En primer lugar, dar las gracias a mi familia, mis padres y mi hermana Alba que durante toda la carrera han estado apoyándome para seguir adelante.

Agradecer también a mi tutor, Antonio Jesús Muñoz, que siempre me dio facilidades y soluciones para todas las dudas con las que me encontraba. Sin la atención que me dio a lo largo del desarrollo del trabajo, este no podría haberse realizado.

Y por supuesto, expresar mi agradecimiento a todos los profesores del grado de Ingeniería Informática en Tecnologías de la Información de la Universidad de Oviedo, por proporcionarme todos estos años una formación tan valiosa y unas experiencias que nunca olvidaré.

Finalmente me gustaría agradecer a todos los amigos que hice estos años. Claudia, Mateo, Nico, Lucas, PT, Rodrigo, Xira, Penín, Jon Landini... Y muchos más que quedan por nombrar. Espero que nos sigamos viendo muchos años más.

Resumen

Este trabajo consiste en el desarrollo de una aplicación software que permita corregir los errores que un algoritmo de alineamiento audio-partitura automático puede cometer al enfrentarse a audios complejos. Este software representa los datos obtenidos de un algoritmo de alineamiento audio-partitura y permite al usuario interactuar con ellos. El objetivo es conseguir que la intervención humana pueda corregir los resultados obtenidos por el algoritmo, gracias a ello se puede conseguir un alineamiento más preciso y libre de errores. La aplicación se realizó en el entorno de desarrollo MATLAB con su herramienta de diseño de interfaces de usuario GUIDE, mismo entorno utilizado por el algoritmo de alineamiento del que se obtienen los datos para modificar. Se realizaron varias pruebas para testear el comportamiento de la herramienta desarrollada, obteniendo resultados satisfactorios.

Índice general

1. Introducción.....	13
2. Objetivo y alcance	15
3. Estudios y análisis previos.....	17
3.1.- Estado del arte en el alineamiento audio-partitura	17
3.2.- Algoritmo de alineamiento	20
3.3.- Estudio de viabilidad	21
3.4.- Software empleado	22
3.4.1.- Entorno de desarrollo MATLAB	22
3.4.2.- Interfaces gráficas de usuario (GUI).....	22
3.4.3.- Uso de GUIDE y comparativa con AppDesigner	23
3.4.4.- Objetos gráficos en MATLAB.....	24
3.4.5.- Github y Git	25
4. Análisis del Sistema	27
4.1.- Visión general.....	27
4.2.- Modos de ejecución	29
4.3.- Detalles de implementación	29
4.3.1.- Apertura y guardado de ficheros.....	30
4.3.2.- Movimiento del camino	31
4.3.3.- Corrección del camino	31
4.3.4.- Navegación del gráfico	32
4.3.5.- Modo evaluar matriz	34
4.3.6.- Puntos de anclaje.....	35
5. Requisitos del Sistema.....	37
5.1 Requisitos.....	37
5.2 Análisis de requisitos del sistema	38

6. Diseño del sistema	45
6.1.- Diseño de arquitectura	45
6.2.- Handles.....	46
6.3.- Métodos de la aplicación.....	48
6.4.- Modos de ejecución	51
6.5.- Diseño interfaz	52
7. Planificación temporal del desarrollo	55
8. Presupuesto del desarrollo efectuado.	59
9. Diseño y ejecución de pruebas.....	61
9.1.- Diseño de pruebas	61
9.1.1- Clases de equivalencia.....	61
9.1.2.- Pares de caminos	63
9.2.- Resultados de ejecución	65
10. Conclusiones y trabajo futuro	79
10.1.- Conclusiones	79
10.2.- Trabajo futuro	80
11. Bibliografía	81
12. Manual de usuario de mAlignment.....	83

Índice de imágenes

Figura 2.1.- Aplicación mAlignment.....	16
Figura 3.1.- Uso de GUIDE en MATLAB.	23
Figura 3.2.- Jerarquía de objetos MATLAB.....	24
Figura 3.3.- Github, software de control de versiones.....	25
Figura 4.1.- Aplicación iniciada.	27
Figura 4.2.- Aplicación en funcionamiento.	28
Figura 4.3.- Modos de ejecución.	29
Figura 4.4.- Diálogo de cargar fichero.	30
Figura 4.5.- Camino modificado por el usuario.	31
Figura 4.6.- Creación del segmento a corregir.	32
Figura 4.7.- Segmento corregido	32
Figura 4.8.- Zona del gráfico sin ampliar.	33
Figura 4.9.- Zona del gráfico ampliada.	33
Figura 4.10.- Navegación entre modos de navegación y ejecución.	34
Figura 4.11.- Valores de un punto de la matriz de costes.....	34
Figura 4.12.- Puntos de anclaje en el camino.	35
Figura 4.13.- Vector de anclaje.	36
Figura 5.1.- Fichero <i>.mat</i> con puntos de anclaje.....	39
Figura 5.2.- Diálogo de guardado.....	39
Figura 5.3.- Aviso de carga de fichero incorrecto.	40
Figura 5.4.- Tramo corregido por el usuario.	41
Figura 5.5.- Segmento a corregir seleccionado.	41
Figura 5.6.- Tramo corregido.	42
Figura 5.7.- Coordenadas de un punto.....	42
Figura 5.8.- Valor de un punto de la matriz.	43

Figura 5.9.- Tamaño de la nube de puntos en la interfaz.	43
Figura 6.1.- Patrón Singleton	46
Figura 6.2.- Interfaz antigua de la aplicación.....	53
Figura 6.3.- Interfaz actual, al iniciar la aplicación.	54
Figura 7.1.- Planificación Gannt 1.	56
Figura 7.2.- Planificación Gannt 2.	57
Figura 7.3.- Planificación Gannt 3.	57
Figura 7.4.- Planificación Gannt 4.	58
Figura 9.1.- Esquema de estados de la aplicación para aplicar pruebas de caminos.	64
Figura 9.2.- Aviso de error al cargar el fichero.....	66
Figura 9.3.- Gráfica de la aplicación mostrando puntos de anclaje y cambios realizados.....	67
Figura 9.4.- Confirmación del guardado.	68
Figura 9.5.- Aviso de punto de anclaje.	70
Figura 9.6.- Aviso tamaño de nube mínimo.....	71
Figura 9.7.- Aviso de tramo del segmento con puntos de anclaje.	72
Figura 9.8.- Aviso de que el segmento ya está formado.	73
Figura 9.9.- Estadísticas de un punto de la matriz.	74

Índice tablas

Tabla 5.1.- Tabla de los requisitos de usuario de la aplicación.	38
Tabla 6.1.- Métodos de inicialización.	48
Tabla 6.2.- Métodos de gestión de modos.	50
Tabla 6.3.- Métodos auxiliares.	51
Tabla 8.1.- Tabla del capítulo I del presupuesto: Costes de personal.	59
Tabla 8.2.- Tabla del capítulo II del presupuesto: Costes de hardware y software.	59
Tabla 8.3.- Resumen de los capítulos.	60
Tabla 8.4.- Costes parciales y total final del presupuesto.	60
Tabla 9.1.- Tabla de los caminos por cubrir.	65
Tabla 9.2.- Tabla de situaciones de prueba derivadas con sus casos de prueba asociados.....	65

1. Introducción

En este trabajo se va a enfrentar el problema del alineamiento automático audio-partitura de una forma semiautomática, es decir, se partirá ya de un algoritmo que dará una serie de resultados de alineamiento automático, pero la aplicación se encargará de añadir la posibilidad de la intervención humana en estos datos, creando así la posibilidad de corregir decisiones erróneas tomadas por el algoritmo. La aplicación creada se concentrará en permitir la visualización de estos resultados y su corrección y navegación por parte de un usuario.

El objetivo de este proyecto es el de desarrollar una aplicación software capaz de modificar los resultados automáticos de un algoritmo de alineamiento audio-partitura.

Este sistema añade una capa de intervención humana al proceso de alineamiento, lo que permitirá afinar los resultados obtenidos y corregir los problemas que puedan haber surgido durante el alineamiento automático. La aplicación, por tanto, incluirá herramientas que permiten al usuario ampliar zonas del gráfico que muestra los datos y moverse sobre ellos, además de distintas funciones para corregir estos resultados.

La aplicación se desarrolla usando MATLAB, con la plataforma de desarrollo de interfaces de usuario GUIDE para diseñar la interfaz gráfica del usuario.

El trabajo se organiza en distintos apartados, que se listarán a continuación con un breve resumen de su contenido:

- **Apartado 1 – Introducción:** Se introduce el trabajo, contextualizándolo y resumiendo su objetivos y alcance.
- **Apartado 2 – Objetivos y alcance:** Se describe en detalle el objetivo de la realización del proyecto y el alcance que este tiene.
- **Apartado 3 – Estudios y análisis previos:** Se presentan los estudios y análisis previos necesarios para el desarrollo del presente proyecto.
- **Apartado 4 – Análisis del sistema:** Se introduce la aplicación, su funcionamiento y funcionalidades.
- **Apartado 5 – Requisitos del sistema:** Se presentan los requisitos recogidos del sistema y su explicación.

- **Apartado 6 – Diseño del sistema:** Se justifican las decisiones tomadas a nivel de diseño, tanto de interfaz como de patrones usados a la hora de programar el software.
- **Apartado 7 – Planificación temporal del desarrollo:** Se introduce la planificación del desarrollo temporal del proyecto y la documentación.
- **Apartado 8 – Presupuesto del desarrollo efectuado:** Se presenta el coste de desarrollar el presente proyecto.
- **Apartado 9 – Diseño y ejecución de pruebas:** Se explican las técnicas de prueba usadas para testear la aplicación y expone los resultados del testing.
- **Apartado 10 – Conclusiones y trabajo futuro:** Se presentan las conclusiones obtenidas durante todo el desarrollo del proyecto y una serie de ideas para implementar en un futuro que podrían mejorar la aplicación.
- **Apartado 11 – Bibliografía:** Se incluyen las referencias a las publicaciones citadas en el trabajo.
- **Apartado 12 – Manual de usuario de mAlignment:** Se presenta el documento que explica la instalación y el uso de la aplicación para un usuario nuevo.

2. Objetivo y alcance

El objetivo principal de este proyecto es el de desarrollar una herramienta gráfica mediante MATLAB que permita a un usuario modificar los resultados automáticos obtenidos por un algoritmo de alineamiento audio-partitura.

Esto significa añadir una capa más al alineamiento, es decir, tras obtener los resultados que un algoritmo de alineamiento logra de un audio frente a su partitura, tener una intervención humana sobre estos resultados para afinar y corregir el resultado, permitiendo corregir los típicos errores que se pueden encontrar en el alineamiento automático.

Para el desarrollo de esta herramienta software, se debieron alcanzar con éxito una serie de hitos que permitieron adquirir los conocimientos necesarios para crear la herramienta, entre los que más importancia tienen son:

- La comprensión del concepto de alineamiento automático audio-partitura para comprender exactamente el funcionamiento del algoritmo.
- Aprender el uso del lenguaje de programación MATLAB y adaptar la forma de programación a una que sea óptima en este lenguaje (por ejemplo, el uso de vectorización al programar).
- Aprender sobre el desarrollo de interfaces de usuario en MATLAB mediante la herramienta GUIDE.
- Comprender el funcionamiento de la programación orientada a eventos dentro de las GUI de MATLAB (callbacks, *handles*, etc.)
- Realizar un desarrollo software incremental, que permita a lo largo del tiempo ir completando la aplicación, mejorarlo e ir añadiendo nuevas funcionalidades siempre evaluando prototipos funcionales.

Estos puntos fueron de vital importancia para lograr la creación de la aplicación, que se espera pueda ser de utilidad en el futuro cuando se necesite un enfoque semiautomático para corregir o modificar los resultados que proporcione un algoritmo de alineamiento y haber logrado estos hitos ha permitido la creación de la aplicación software que se presenta en este

trabajo y que permitirá una visualización de los resultados de un alineamiento de audio-partitura, su modificación de diversas maneras y la oportunidad de guardar dicha modificación cambios para más adelante hacer uso de ellos para otras aplicaciones.

La aplicación software obtenida tras este desarrollo se nombrará como mAlignment. Esta aplicación permite al usuario modificar los resultados de un alineamiento automático de diversas formas con el uso del ratón y el teclado. También se disponen de estadísticas que proporcionarán al usuario información sobre los datos evaluados, como el tempo medio de la obra o el tempo medio de un segmento de esta.

Todos los cambios que el usuario realice podrán ser cargados y guardados en ficheros *.mat*, la extensión que ofrece MATLAB para guardar información, que se puede cargar después en su *workspace*.

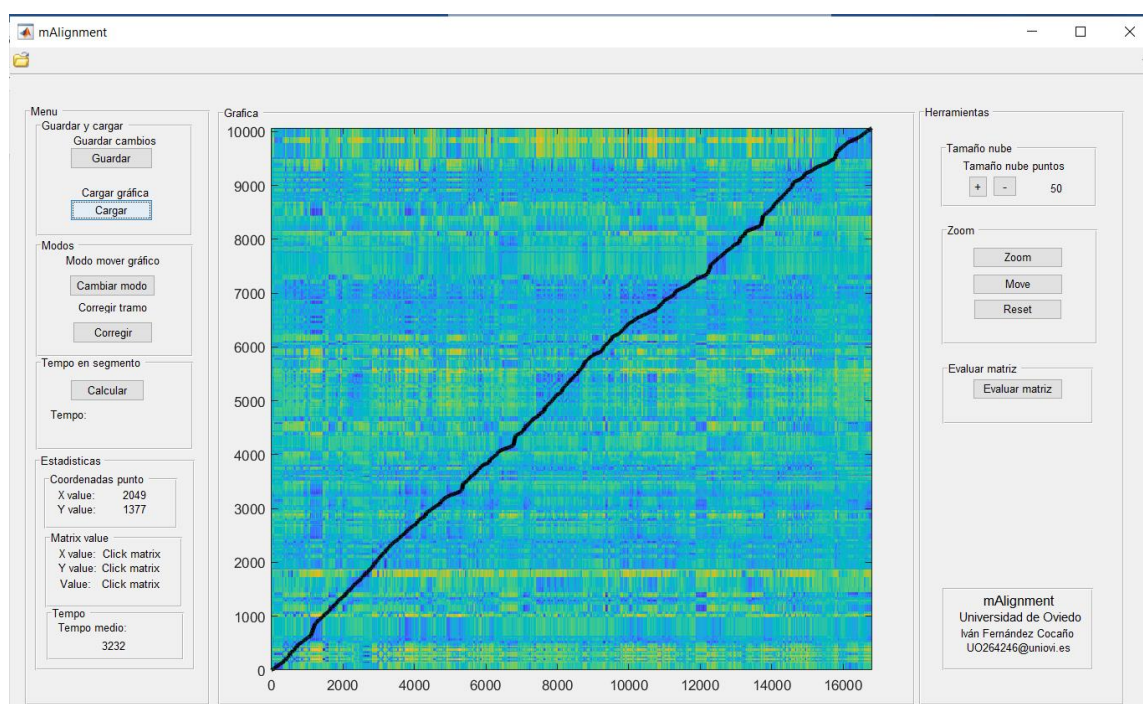


Figura 2.1.- Aplicación mAlignment.

3. Estudios y análisis previos

En este apartado se describirán los principales aspectos teóricos que se necesitaron para la realización del proyecto, además del lenguaje de programación, herramientas y demás recursos utilizados para la creación de la aplicación.

3.1.- Estado del arte en el alineamiento audio-partitura

El alineamiento de audio-partitura es un problema que consiste en la sincronización entre una grabación de música clásica y su correspondiente partitura, que lleva siendo investigado desde mediados de los 80. Este problema es más complejo de lo que a simple vista puede parecer, ya que una misma composición musical puede ser interpretada de forma muy distinta en términos de tempo, dinámica, textura, orquestación, etc. Esto dificulta la tarea de alineamiento, pues esta información no aparece en las partituras originales. Además de este problema, cabría mencionar que existen otras cuestiones que dificultan el alineamiento automático, como las grabaciones de mala calidad con ruido de fondo, errores cometidos por los músicos durante la interpretación o incluso repeticiones de secciones musicales que no están contempladas en la partitura original. Con el desarrollo de esta aplicación, se busca paliar estos problemas anteriormente citados.

Tradicionalmente el alineamiento automático se suele dividir en dos fases: la fase de *feature extraction* o extracción de características y el propio alineamiento. En la primera fase, se obtienen las características de las señales de audio, que proporcionan una información específica sobre el contenido musical de estas. Ambas señales, la de la partitura y la del audio de entrada se representan en dos secuencias que deben ser alineadas: $U = (u_1, \dots, u_n, \dots, u_N)$ y $V = (v_1, \dots, v_t, \dots, v_T)$ donde N representa el número de instantes en el tiempo de la partitura y T el del audio de entrada. Hay diversos métodos para enfrentarse a los problemas que surgen en esta primera fase, como el de análisis a través de la factorización de matrices no negativas.

En la segunda fase, la de alineamiento, se busca la mejor forma de encajar las características obtenidas del audio de entrada con los datos que se tienen de la partitura, realizando así el alineamiento entre ambos.

También, a la hora de realizar el alineamiento, se distinguen dos tipos de alineamiento según si el audio de entrada está grabado o se recibe en directo. La diferencia entre ambos tipos requiere abordar el problema de distintas formas: una es la forma *online*, que se corresponde con un alineamiento en el que la entrada del audio se introduce en directo según se va grabando.

Una de las formas usadas para enfrentarse a este alineamiento *online* es la propuesta por Dixon [1], en la que se propone usar una función de coste local para comparar pares de audio y segmentos de las partituras:

$$d(n, t) = f(u_n, v_t)$$

Para esta fórmula $f(\cdot)$ es cualquier tipo de función de comparación y $d(n, t)$ es el coste de alinear u_n con v_t . Con la anterior ecuación se calcula una matriz de coste acumulado en la que está almacenada el camino de mínimo coste de (n, t) . Dado un tiempo t específico, el resultado del método de Dixon es el punto (n, t) con el mínimo coste. La desventaja de este método es que el algoritmo tiende a perderse en la partitura y no es capaz de recuperarse debido a las condiciones que se aplican para su formación.

Desde el punto de vista del alineamiento audio–partitura *online* podrían destacarse una serie de usos [2]:

- Acompañamiento musical automático

El acompañamiento musical consiste en tomar como entrada una partitura y ser capaz de reconocer en cada instante lo que un instrumento solista está tocando en directo. Así, se puede ajustar ese audio de entrada y producir un acompañamiento sonoro sintético en directo para el músico.

- Paso de páginas automático

Se puede usar el alineamiento para mostrarle a un músico la parte de la partitura que está interpretando. El alineamiento reconoce en que punto de la partitura se encuentra en todo momento y de forma automática avanza para mostrarle lo que debe tocar, aunque el músico tenga errores o un tempo distinto al registrado exactamente en la partitura.

- Separación online de fuentes con información de partituras

Aunque son más complejos que su contraparte offline, también existen sistemas para separar las fuentes/instrumentos que componen una mezcla de audio.

La otra forma de realizar alineamientos automáticos es la *offline*, en la que ya se tiene un archivo grabado de la interpretación de la pieza musical, por lo que durante el alineamiento se puede acceder al audio en su totalidad. Esto suele proporcionar resultados más robustos. Ambos tipos de alineamiento tienen sus aplicaciones y usos.

En cuanto a la versión *offline* del alineamiento audio-partitura podrían destacarse aplicaciones como [3] [4]:

- Editores de audio inteligentes

Estos editores se usan actualmente en el ámbito de la edición de música profesional.

Permiten a los usuarios aplicar distintos efectos a grabaciones, como, por ejemplo, corregir el tono de forma automática en grabaciones de voz, es decir, un software capaz de editar audios y cambiar cualquier tono desafinado para ajustarlo a su correcta tonalidad.

- Servicios de identificación de música [3]:

Una forma de encontrar una composición musical determinada en una base de datos en la que se disponen de las partituras en un formato simbólico como MIDI es a través del uso del alineamiento audio-partitura. Se basa en un proceso de búsqueda en donde se alinea el audio de entrada con todas las partituras almacenadas en la base de datos. Cuando se encuentra la obra que mejor coincide con el audio de entrada esta se presenta al usuario. Es un método costoso, pero útil y que incluso puede usarse para indexar bases de datos de audios o MIDI.

- Separación de fuentes sonoras

El alineamiento audio-partitura puede usarse para separar las distintas fuentes sonoras que componen una grabación musical. Para lograr esto, el alineamiento se usa como una etapa previa a la separación.

Para la realización de este trabajo, se van a obtener ejemplos de un alineamiento de tipo *offline*. La información de salida del alineamiento se representará en la aplicación para que el usuario pueda modificarlos. Es importante conocer el funcionamiento del algoritmo y los datos que se obtendrán de él, por tanto, a continuación, se resume lo más importante del algoritmo.

3.2.- Algoritmo de alineamiento

El algoritmo de alineamiento automático usado para obtener los datos de alineamiento es un alineamiento de tipo offline, que usa factorización espectral y DTW para obtener sus resultados.

La factorización espectral consiste en aprender patrones espectrales para las combinaciones de notas concurrentes que se den en el MIDI proporcionado. El DTW (Dynamic time warping) es un algoritmo usado para determinar similitudes entre dos secuencias temporales que pueden ser de distinta velocidad [5].

Este algoritmo divide el proceso de alineamiento en dos fases: preprocesado y alineamiento.

En la fase de preprocesado se sintetiza el MIDI de entrada. A este audio sintético se le aplica la STFT (*short-term Fourier transform*) para aprender sus patrones espectrales. Cada uno de estos patrones se corresponde con un conjunto de notas musicales tocadas en un mismo instante en la partitura. Cada combinación de estas notas se llama *score unit*. De esta forma, cada *score unit* se denota como u_n y se corresponde con el patrón espectral del instante n . Así, la secuencia total de *score units* se representa como U . En esta fase, también se aplica la STFT sobre la señal de audio de entrada y se obtiene un vector v_t de patrones espectrales para cada *frame* temporal t .

Después del preprocesado, viene la fase de alineamiento, en la que se calcula una matriz de coste. Esta matriz de costes se obtiene de las diferencias entre los patrones espectrales que se obtienen de la partitura y los que se obtienen del audio de entrada. Con estas diferencias pueden verse las zonas que más similitud tendrán entre el audio de entrada y la partitura. Posteriormente, el DTW será el encargado de generar y encontrar el camino óptimo dentro de esta matriz de costes. En primer lugar, se calculará una segunda matriz, llamada D , de la siguiente forma [5]:

$$D(i, j) = \psi(u_i, v_j)$$

Donde la matriz D tiene $I \times J$ elementos, que representan el coste entre todos los dos puntos de toda la serie. El coste de la función ψ puede ser cualquiera que retorne cero para una coincidencia perfecta o un valor positivo en otro caso.

En el segundo paso del alineamiento, se usará un enfoque tradicional de DTW. En este sentido, se genera de forma recursiva una matriz de *warping* C :

$$C(i, j) = \min \begin{cases} C(i, j - c_j) + D(i, j) \\ C(i - c_i, j) + D(i, j) \\ C(i - c_i, j - c_j) + \sigma D(i, j) \end{cases}$$

A partir de esta matriz C se obtiene el camino óptimo $w = w_1, \dots, w_k, w_K$ con recursión desde $C(I, J)$. Para generar el camino se necesitará además que se cumplan varias condiciones, como por ejemplo empezar por el principio del audio y terminar en el fin de este, que el tempo no puede ser exageradamente largo o lento, etc. En este camino, cada w_k es un par ordenado (i_k, j_k) en los que $(i, j) \in w$, lo que se traduce en un alineamiento entre los puntos u_i y v_j .

Estos dos conceptos, los de la matriz de coste y el camino óptimo que se genera a partir de esta, son de vital importancia para el desarrollo de la aplicación, pues van a ser datos con los que el usuario podrá interactuar.

3.3.- Estudio de viabilidad

El algoritmo de alineamiento empleado en este trabajo se encuentra en fase de desarrollo e investigación, de ahí que esté implementado en MATLAB, Esto afecta al desarrollo del presente proyecto ya que, en caso de querer desarrollar la aplicación en un lenguaje de programación como Python o Java, se necesitaría convertir todo el algoritmo a ese lenguaje propios. Este sería un trabajo muy pesado y que se saldría de los objetivos del proyecto.

Como no se plantea un uso comercial de la herramienta y por las herramientas software que ofrece MATLAB, de las que se hablará en el siguiente apartado, se decide realizar la aplicación usando también el entorno de desarrollo MATLAB.

Esto permite aprovechar plenamente el algoritmo de alineamiento automático del que partimos, sin necesidad de alterarlo gravemente ni empezar de cero, por lo que el enfoque del proyecto puede centrarse totalmente en el desarrollo de la herramienta software.

3.4.- Software empleado

Durante la realización del trabajo, se usaron distintas herramientas software desarrollar la aplicación final. En este apartado se explicarán las herramientas empleadas y como se usaron durante todo el proceso de desarrollo.

3.4.1.- Entorno de desarrollo MATLAB

MATLAB (“MATrix LABoratory”) es una herramienta de software matemático de uso extendido en centros de investigación, universidades y desarrollo.

MATLAB se desarrolla en un lenguaje de programación propio interpretado, que puede ejecutarse tanto en el entorno interactivo como a través de un archivo de script de extensión **.m*. Usando MATLAB se puede realizar con facilidad cálculos matriciales, desarrollo y ejecución de algoritmos, aprendizaje profundo, análisis de datos, creación de interfaces de usuario, etc.

MATLAB percibe todos los datos con los que trabaja como matrices y permite la vectorización del código, lo que puede reducir lo que en otros lenguajes serían varias líneas de código en una sola. Aunque ya de mano dispone de muchas funcionalidades, estas pueden ser ampliadas con *toolboxes* creadas por otros usuarios de la comunidad que se pueden aprovechar para el desarrollo de las aplicaciones. Este lenguaje, usado en multitud de aplicaciones para procesado de imagen y señal permite el desarrollo de manera más eficiente y sencilla que la programación en otros lenguajes como pueden ser Java o C++.

Debido a todas estas ventajas, y aprovechando que el algoritmo de alineamiento automático que se usa para obtener los datos que se van a representar está en MATLAB, se usa este entorno de desarrollo.

3.4.2.- Interfaces gráficas de usuario (GUI)

Las interfaces gráficas de usuario son la forma en la que un usuario puede interactuar con un dispositivo sin necesidad de programar o interactuar con él a través de una consola de comandos. Es una abstracción que permite a usuarios realizar acciones sin necesidad de tener conocimientos de programación. El ejemplo típico de GUI sería el de los entornos gráficos de Windows o Android, gracias a los cuales un usuario sin conocimientos puede operar con el

ordenador de muchas maneras. Para el correcto diseño de una interfaz, es importante definir los requisitos de usuario, diseñar prototipos de GUI y evaluarlos para ir incrementalmente mejorando el resultado hasta tener una interfaz gráfica comprensible y amigable con el usuario.

3.4.3.- Uso de GUIDE y comparativa con AppDesigner

En principio, se podría de manera manual programar una GUI en MATLAB, creando una figura y de forma programática colocar todos los botones o elementos que se necesiten, pero este método es muy tedioso y lleva a programar todo a mano, perdiendo una gran cantidad de tiempo.

Por suerte, MATLAB proporciona GUIDE, un entorno gráfico para diseñar interfaces de usuario y que permita codificar las funciones por separado. GUIDE otorga mucha facilidad para añadir componentes y customizarlos en nuestra figura, lo que facilita la tarea del diseño de GUIs. Otras ventajas que tiene es que genera automáticamente código plantilla para los *callback* (funciones que se ejecutan como reacción a un evento) de los elementos colocados y permite una total libertad para editarlo.

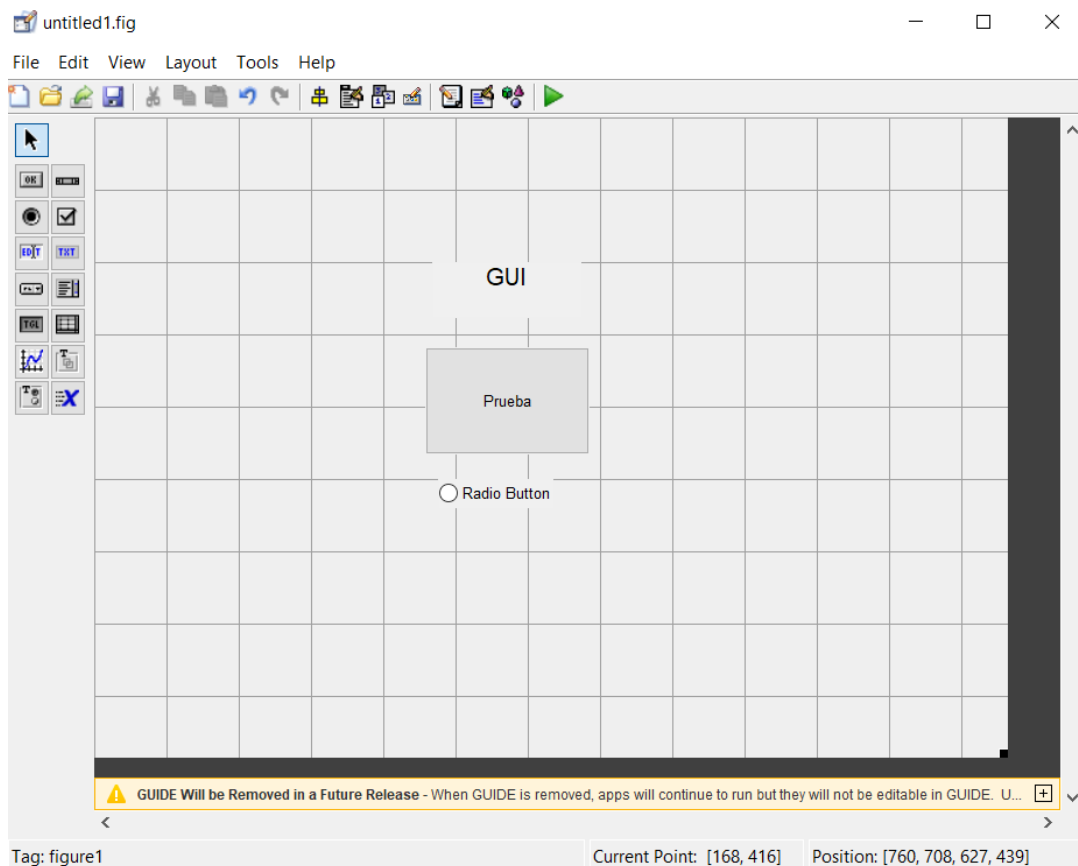


Figura 3.1.- Uso de GUIDE en MATLAB.

GUIDE genera dos ficheros, un archivo *.fig* que representa la figura con la localización de los componentes que forman la GUI y el archivo *.m*, que contiene las funciones que controlan la aplicación.

MATLAB ofrece también el AppDesigner, otro entorno de desarrollo de GUIs que aún no dispone de todas las funciones gráficas de MATLAB ni tiene funciones de menús y barra de herramientas. Además, limita el acceso al programador de todo el código. Sin embargo, ofrece otras facilidades, como una creación de GUIs que puede parecer visualmente más atractiva y *callbacks* específicos para cada objeto. Para este proyecto se decide usar GUIDE ya que proporciona más versatilidad.

3.4.4.- Objetos gráficos en MATLAB

Con estos objetos se pueden mostrar datos de forma visual en MATLAB, por ejemplo, creando gráficos con líneas de texto y ejes en la ventana de una figura. Cuando se crea una gráfica MATLAB se sigue una serie de pasos de forma automática para crearla y configurarla, este conjunto de propiedades se puede controlar para modificar el comportamiento y la apariencia de la gráfica.

Los objetos gráficos de MATLAB siguen esta jerarquía:

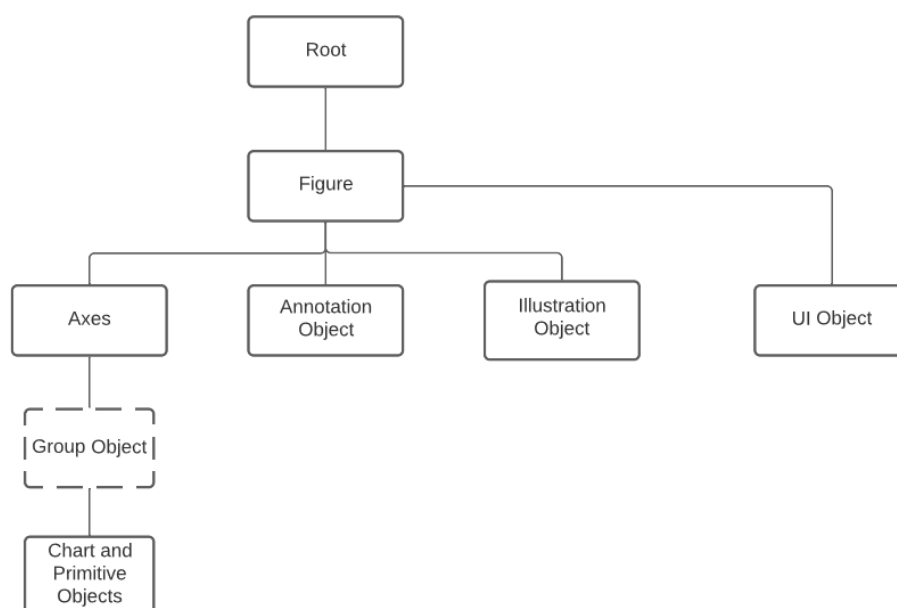


Figura 3.2.- Jerarquía de objetos MATLAB.

Los objetos pueden contener otros objetos, como ocurre en el caso de los ejes, en el que se tiene una figura que es la ventana que contiene los ejes y a su vez, los ejes contienen las líneas dibujadas y textos utilizados para representar la gráfica. Esta relación se refleja en las propiedades *Parent* y *Children*. La propiedad *Parent* de la línea que este dibujado en un eje será el propio eje.

En *mAlignment* se tienen los siguientes objetos gráficos:

- Una figura gráfica que contiene todos los elementos, es decir, la ventana de la aplicación.
- Los paneles, botones, textos y demás elementos de la GUI contenidos en la figura.
- Los ejes que contienen el objeto que representa los datos de alineamiento.
- Las líneas y puntos dibujados para representar el camino que el algoritmo escogió para el alineamiento.

3.4.5.- Github y Git

Git es un software de control de versiones que permite controlar versiones de forma distribuida con facilidad.

En este proyecto se usó Github, que implementa todas estas funcionalidades de Git además de facilitar enormemente su uso con su aplicación de escritorio, que permite con facilidad subir los cambios realizados al repositorio local y online.

Una ventaja del uso de Github es que permite controlar las versiones de la aplicación que se van realizando, y en caso de que se comenta un error al cambiar código, se puede regresar siempre a una versión anterior y empezar de nuevo desde una versión antigua de la aplicación, pero sin errores.



Figura 3.3.- Github, software de control de versiones.

4. Análisis del Sistema

La aplicación mAlignment es una aplicación de MATLAB que ofrece una interfaz que permite la interacción del usuario con datos cargados desde ficheros *.mat* con la intención de poder visualizarlos y corregirlos a su gusto. Tras las modificaciones se podrá guardar los cambios realizados para usarlos en un futuro.

4.1.- Visión general

Cuando se abre la aplicación, la interfaz nos muestra todos sus paneles y botones bloqueados salvando los de cargar datos de ficheros. Al no haber datos que representar, la gráfica aparece en blanco sin ningún tipo de representación gráfica.

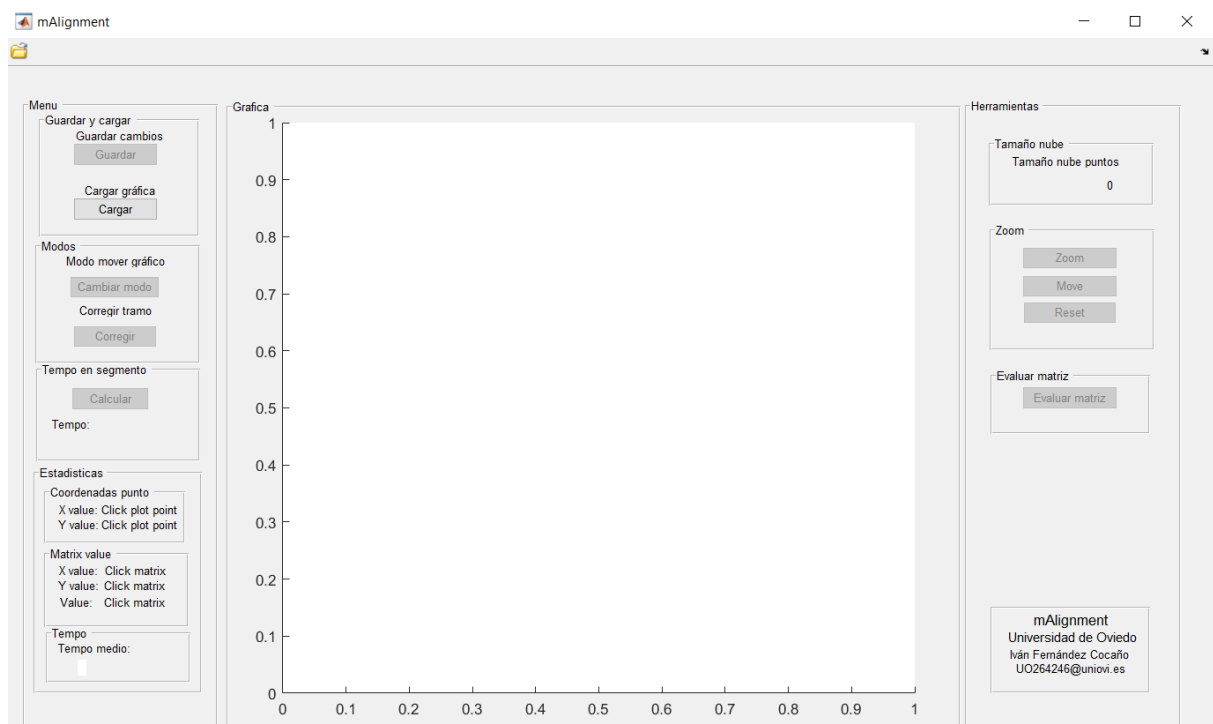


Figura 4.1.- Aplicación iniciada.

Una vez se cargan los datos del alineamiento automático, se mostrará en el panel de Gráfica la matriz de costes y el camino determinado. Estos datos se podrán cargar desde un fichero *.mat*.

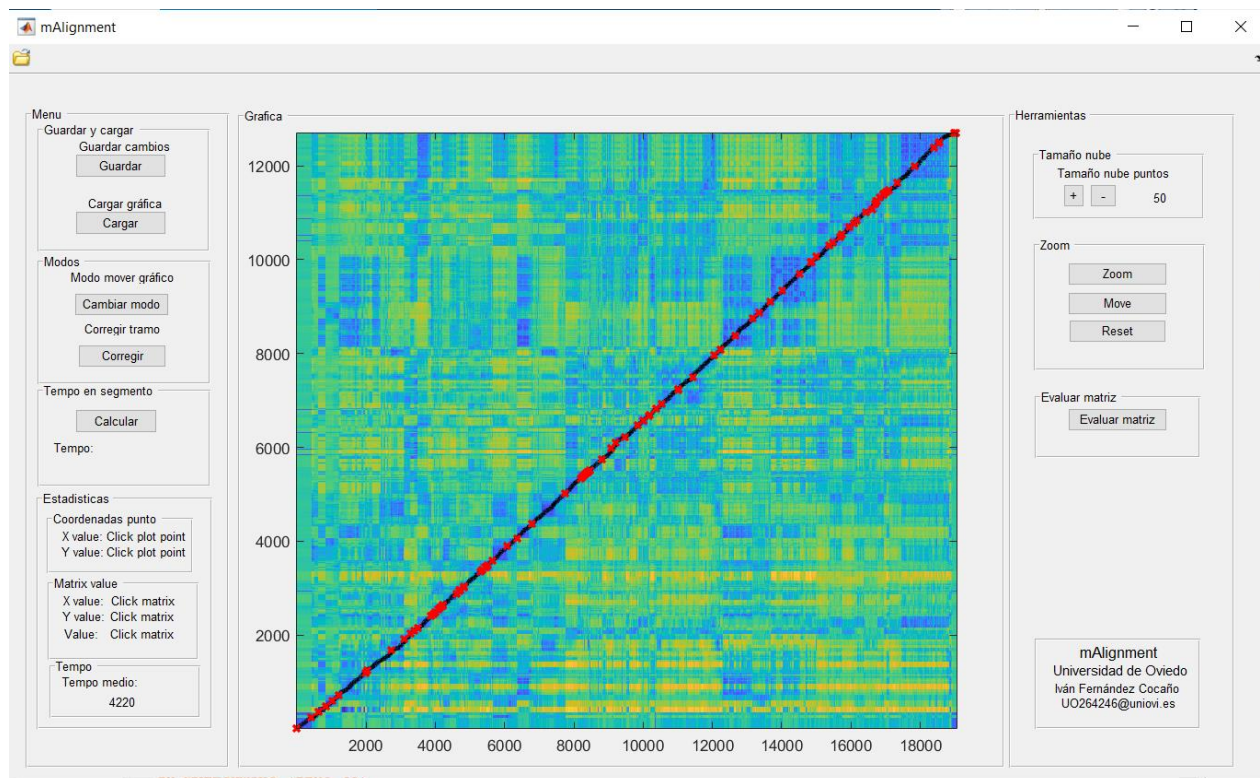


Figura 4.2.- Aplicación en funcionamiento.

Tras cargar los datos de alineamiento automático, el usuario tiene libertad para interactuar con los datos con las herramientas que ofrece la aplicación. Este software permitirá al usuario realizar una serie de funciones sobre estos datos, que en resumen son:

- Apertura y guardado de ficheros *.mat*. Esto consiste en poder cargar y guardar resultados del alineamiento automático.
- Mover el camino generado por el alineamiento automático usando el ratón. Se moverán una serie de puntos de un tamaño determinado por el usuario, lo que se conoce como nube de puntos.
- Corrección del camino generado por el alineamiento automático mediante un proceso de interpolación.
- Evaluar las coordenadas de cualquier punto del camino. El usuario podrá visualizar esta información haciendo click en cualquier punto del camino.
- Evaluar los valores de la matriz de costes haciendo click en puntos de la figura representada.
- Navegación del gráfico: zoom, movimiento del gráfico una vez ampliado, etc.
- Interacciones con puntos de anclaje: Los puntos de anclaje son puntos especiales que no se deben poder modificar por el usuario de ninguna forma. La aplicación tiene la

capacidad de trabajar tanto con alineamientos automáticos que posean puntos de anclaje como aquellos que no los tengan.

4.2.- Modos de ejecución

La aplicación permuta entre distintos estados de ejecución que se corresponden con las funcionalidades antes mencionadas. En general, se puede hablar de cuatro modos de ejecución, mover gráfico, corregir, evaluar matriz y los modos de navegación de gráfico.

Los distintos modos de ejecución y las transiciones que se pueden dar entre ellos se representan a continuación:

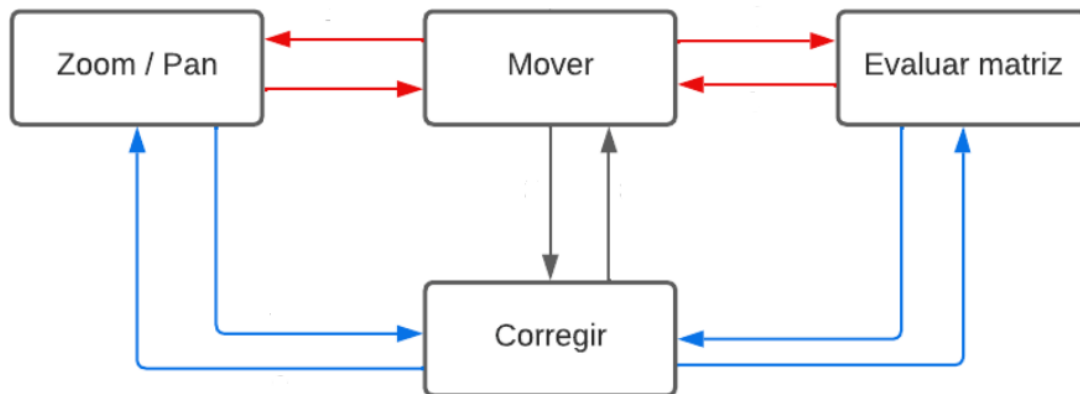


Figura 4.3.- Modos de ejecución.

A la hora de permutar entre modos, hay ciertas restricciones. Esto se representa con los colores de las flechas en el esquema, cuando se pasa a un estado por una flecha de un color rojo o azul, solo se podrá cambiar de estado a través de una transición del mismo color. Dentro de la aplicación se controla los modos en los que se está mediante variables que actúan como *boolean* almacenadas en la estructura *handles*.

4.3.- Detalles de implementación

Este apartado explica las decisiones tomadas durante el desarrollo del software que permitieron a la aplicación dar la funcionalidad que se espera de ella.

4.3.1.- Apertura y guardado de ficheros

Para la carga de ficheros se usa el diálogo de MATLAB *uigetfile* que abre una ventana con los ficheros de la carpeta actual y le da al usuario la opción de escoger una o cerrar el diálogo.

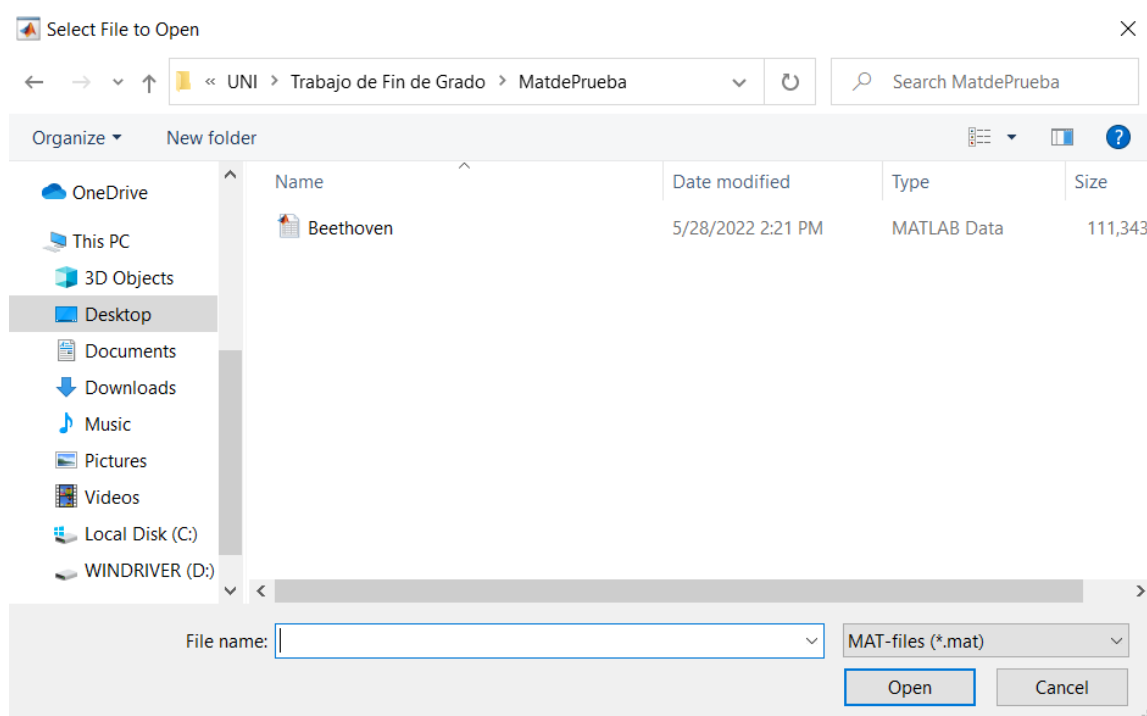


Figura 4.4.- Diálogo de cargar fichero.

Una vez elegido el fichero, se controla que como mínimo posea las variables *matrizTmxTr* y *p* que se corresponden con la matriz de costes y el camino generado por el alineamiento automático. Teniendo estas dos variables se cargan en el *workspace* de MATLAB y se inicializa la estructura *handles*, que es la que mantendrá toda esta información y permitirá el acceso a estos datos a los métodos de la aplicación.

Respecto al guardado de ficheros, se hace de una forma parecida, esta vez usando el dialogo *uisave*, que sirve para guardar las variables que interesan en un fichero *.mat*.

Si los datos cargados previamente contienen puntos de anclaje, el fichero guardado almacenará también esta información.

4.3.2.- Movimiento del camino

Para modificar el alineamiento, la aplicación debe estar en modo mover gráfico. En este modo de ejecución, el usuario puede interactuar con el camino trazado por el alineamiento automático haciendo click en él y moviéndolo al punto deseado.

De este modo se pueden corregir errores del alineamiento. El funcionamiento del código de esta funcionalidad consiste en una serie de *callbacks* anidados que permiten el cambio de las coordenadas del eje de ordenadas del camino representado. Se parte de la función *buttondown*, que se usa como *callback* cuando el usuario hace click en un punto del camino. Esta función, si está en modo mover gráfico asigna una función más que va a reaccionar a los eventos del usuario, la función *move*. Este método se va a ejecutar cuando el usuario mueva el ratón. La función *move* se encarga de que las coordenadas del eje de ordenadas del camino se ajusten en tiempo real a donde el usuario decida arrastrar el ratón. Se moverán tantos puntos como tamaño tenga la nube de puntos elegida por el usuario.

Cuando el usuario decide soltar el ratón, se llama a un *callback* más que se asignó en la función *move*: La función *buttonup*, que se encarga de que al soltar el click, el movimiento pare, así, la aplicación vuelve a estar como al principio del movimiento, pero con los valores del camino modificados.

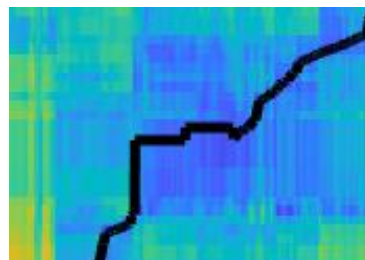


Figura 4.5.- Camino modificado por el usuario.

4.3.3.- Corrección del camino

Para corregir el camino la aplicación debe estar en modo mover corregir. La corrección del camino consiste en que el usuario elija un segmento del camino y que las coordenadas de este segmento se corrijan, creándose una interpolación con la ecuación de la recta entre ambos extremos del segmento. Por lo tanto, esta funcionalidad debe permitir al usuario dibujar su segmento y después corregirlo.

Se aprovecha la función `buttondown` anteriormente descrita, que en esta ocasión al estar en modo corregir asigna un *callback* distinto a la aplicación, que se activa cuando el usuario le da a una tecla.

Cuando el usuario aprieta en un punto y le da a la tecla ‘x’, se dibujará en la aplicación uno de los extremos del segmento que puede corregir. Una vez ambos extremos están seleccionados, se puede corregir el segmento dándole al botón de corregir, lo que hará que se cambien todas las coordenadas entre esos dos puntos por una interpolación en línea recta entre ambos extremos.

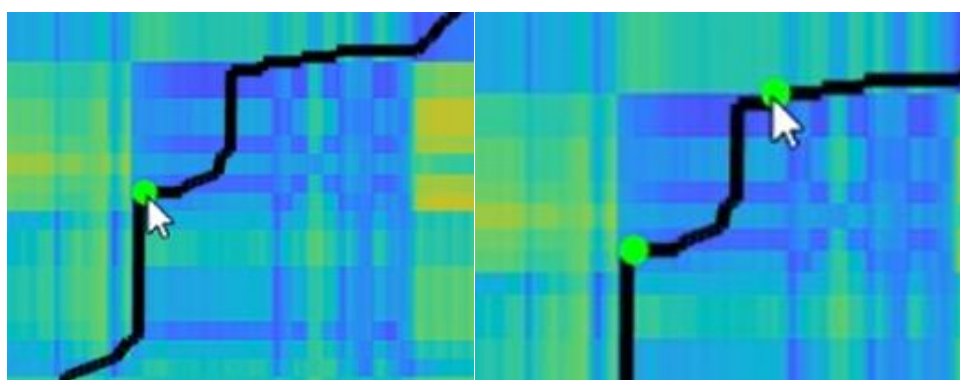


Figura 4.6- Creación del segmento a corregir.

Tras darle al botón de corregir:

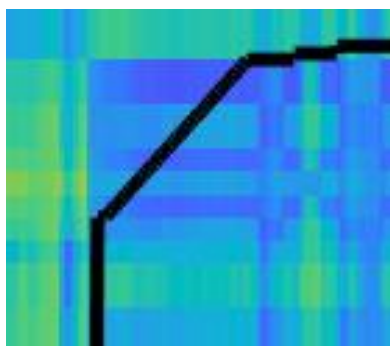


Figura 4.7.- Segmento corregido

4.3.4.- Navegación del gráfico

Hay diversas formas de interaccionar con los datos representados en la aplicación. Se dispone de un modo zoom que permite ampliar puntos del gráfico a gusto del usuario. Por ejemplo, si se encuentra una zona que puede resultar de interés cambiar o ampliar, basta con

activar el modo zoom y con el ratón seleccionar un cuadrado en esa zona. De esa forma se pasará de:

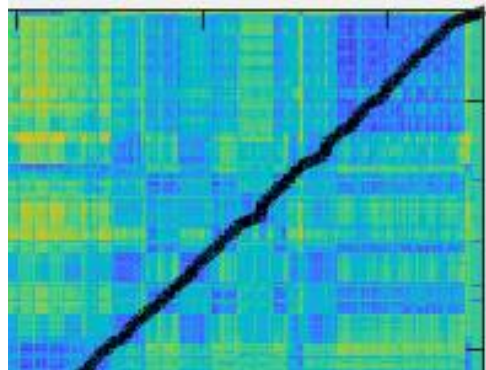


Figura 4.8.- Zona del gráfico sin ampliar.

A una versión ampliada de la misma zona:

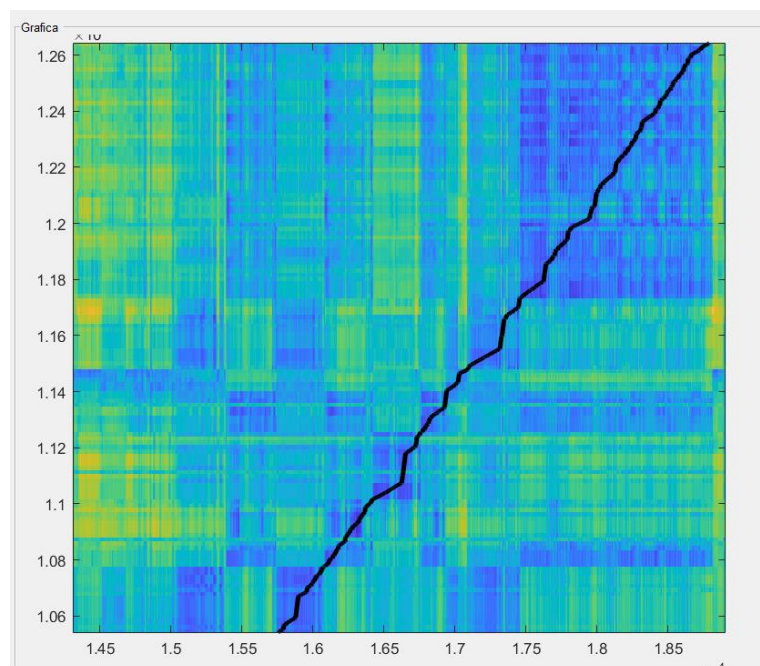


Figura 4.9.- Zona del gráfico ampliada.

Esto se realiza de forma sencilla ya que MATLAB incluye ya para todas las figuras un modo zoom que se puede activar con código mediante la sentencia `zoom on`. Para controlar en qué momento debe estar activado o no el zoom, se usa una variable boolean y así según el estado de esta variable se sabe si se debe activar el zoom o no cuando el usuario interaccione con el botón de zoom.

También existe el modo pan, que tiene un funcionamiento análogo al del zoom, con una variable que controla si está activado o no y un botón para que el usuario pueda activarlo cuando desee. Este modo permite al usuario hacer click en una zona del gráfico ampliado y arrastrar al ratón para moverse en ese mismo nivel de zoom y recorrer el gráfico.

El usuario se puede mover entre los modos de zoom y pan sin problemas, ya que no hay limitaciones en esas transiciones.

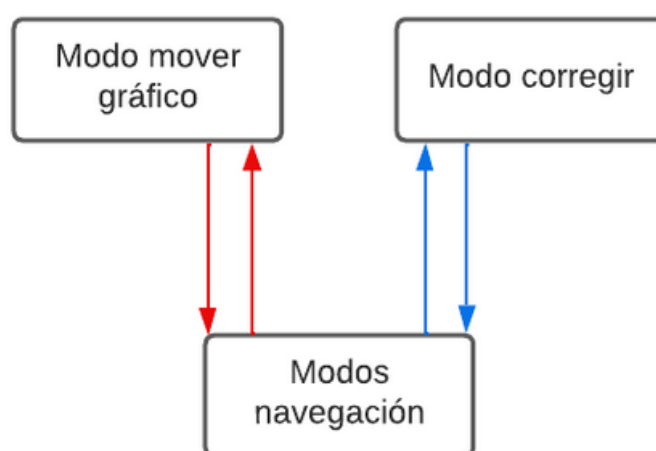


Figura 4.10.- Navegación entre modos de navegación y ejecución.

4.3.5.- Modo evaluar matriz

El modo evaluar matriz sirve para que el usuario pueda ver los valores de la matriz de costes generada por el alineamiento automático. Cuando está activado y el usuario hace click en cualquier punto de la matriz, se reflejan los valores de la matriz en el panel de estadísticas:

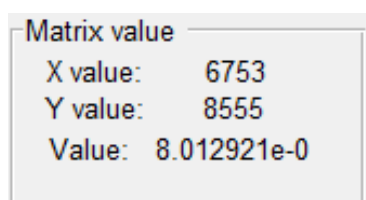


Figura 4.11.- Valores de un punto de la matriz de costes.

Se controla el estado de este modo con una variable boolean, como en el resto de los casos. Para su funcionamiento, cuando se activa se prepara el objeto *Axes* que representa todos los valores del alineamiento automático para que cuando el usuario haga click en un punto de la matriz se ejecute el *callback* evalMatrix, que recogerá el valor de las coordenadas y el valor

de coste almacenado en ese punto de la matriz de costes y lo representará en el panel que se muestra en la Figura 4.11.

4.3.6.- Puntos de anclaje

Los puntos de anclaje son puntos especiales generados durante el alineamiento automático que no se deberían poder mover, aunque el usuario así lo quisiera. De esta forma, cuando el software se ejecuta considerando estos puntos de anclaje, aparecen una serie de limitaciones extra.

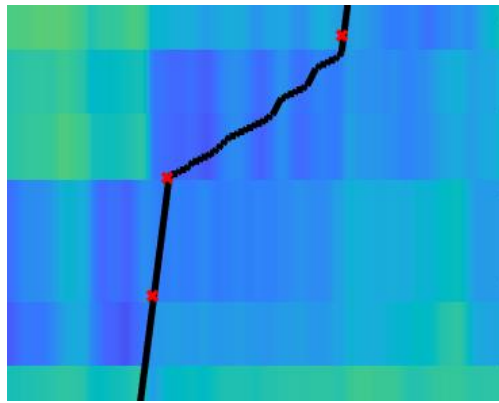


Figura 4.12.- Puntos de anclaje en el camino.

La implementación de estos puntos de anclaje consiste en el uso de un vector que tiene el mismo tamaño que el número de puntos que contiene el camino.

Cada elemento del vector será un indicador de si en ese punto del eje X hay un punto de anclaje, ya que los valores que puede tomar cada elemento es 0 en caso de que no haya punto de anclaje y 1 si lo hay. Por ejemplo, en un conjunto de datos de alineamiento automático en los que el punto de origen es un punto de anclaje, el vector de anclaje mostraría esto en los primeros 50 valores:

```
K>> handles.anclaje(1:50)

ans =

Columns 1 through 18

    1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

Columns 19 through 36

    0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

Columns 37 through 50

    0     0     0     0     0     0     0     0     0     0     0     0     0     0
```

Figura 4.13.- Vector de anclaje.

Como se puede observar en la Figura 4.13, el primer elemento del vector es 1, ya que se corresponde con un punto de anclaje. En el resto de los elementos mostrados no hay puntos de anclaje, por lo que su valor es 0. De este modo se rellena el resto de los elementos para todos los puntos que abarca el camino. Gracias a este vector se podrá comprobar la existencia de puntos de anclaje al mover y corregir puntos, permitiendo al software actuar de la forma esperada en caso de encontrarlos, es decir, impidiendo la corrección y el movimiento.

5. Requisitos del Sistema

En este apartado se tratarán los distintos requisitos que se recogieron para poder ofrecer todas las funcionalidades necesarias de la aplicación. Estos requisitos van a reflejar los requerimientos que va a tener la aplicación. Primero se listarán todos los requisitos definidos con un pequeño resumen de su descripción y a continuación, se detallarán más a fondo cada uno de ellos.

5.1 Requisitos

Los requisitos del sistema son las expectativas que se tienen respecto a nuestra aplicación, y que son evaluados para crear funcionalidades que coincidan con estas necesidades para que haga lo que se espera de ella.

En concreto, los requisitos de usuario de los que se parte para el desarrollo de mAlignment son:

ID	Descripción	Ref	Importancia
1	Gestión de los ficheros usados		
1.1	El usuario debe poder cargar ficheros .mat con datos de alineamiento.	R01	Alta
1.2	El usuario debe poder guardar los cambios que realizó en un fichero .mat con el nombre que quiera.	R02	Alta
1.3	Se debe gestionar que los archivos .mat tengan las variables necesarias para el funcionamiento del programa.	R03	Alta
2	Movimiento de los puntos que forman el camino		
2.1	El usuario debe poder clickar en uno de los puntos del camino	R04	Alta
2.2	Tras clickar en el, se podrá arrastrar arriba o abajo y se moverá hasta donde elija el usuario, cuando deje de clickar parará el movimiento.	R05	Alta
3	Corrección de segmento del camino		

3.1	Dado un segmento del camino, el usuario puede hacer que ese segmento, desde su origen hasta el final se conviertan en una línea recta.	R06	Alta
4	Muestra de estadísticas		
4.1	Se deben mostrar los valores X e Y de un punto que quiera el usuario.	R07	Alta
4.2	Se debe poder evaluar los valores X, Y y el peso almacenado en ese punto de la matriz de costes.	R08	Alta
4.3	Se debe mostrar el valor del tiempo medio de los datos cargados	R09	Media
4.4	Elegido un segmento del camino por el usuario, se debe poder calcular su tiempo medio.	R10	Media
5	Tamaño de la nube de puntos que se mueve		
5.1	El tamaño de la nube de puntos se puede hacer más grande o pequeño según quiera el usuario.	R11	Alta

Tabla 5.1.- Tabla de los requisitos de usuario de la aplicación.

5.2 Análisis de requisitos del sistema

Los requisitos del sistema permiten tener una visión global de lo que la aplicación necesita hacer y del alcance que esta tiene. Son una forma de documentar unas necesidades a nivel funciona de la aplicación.

En este apartado se hablará de los requisitos del sistema establecidos antes, describiendo exactamente lo que se busca con cada uno de ellos y hablando de la implementación que se le dio dentro de la aplicación. Los requisitos del sistema se dividen en varios grupos, empezando por:

1. Gestión ficheros usados

Estos requisitos tratan toda la problemática de cargar los ficheros *.mat* en los que se guardan los resultados del alineamiento automático para que el usuario pueda interactuar con ellos y a continuación pueda guardar los cambios realizados.

1.1 El usuario debe poder cargar ficheros *.mat* con datos de alineamiento.

La aplicación cumple este requisito funcional permitiendo a través de un icono en el menú y de un botón de carga, cargar cualquier fichero *.mat* que contenga como mínimo una variable *p* con los datos del camino que se obtuvieron del algoritmo y una variable *matrizTmxTr*, que es la matriz de coste obtenida también del alineamiento automático.

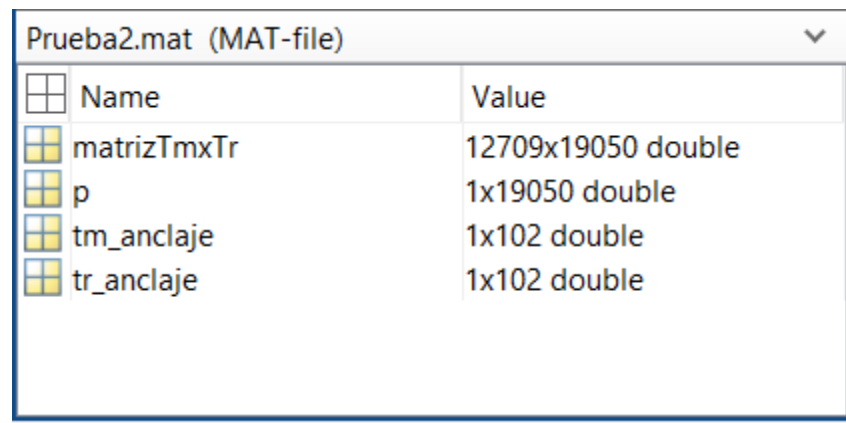


Figura 5.1.- Fichero *.mat* con puntos de anclaje.

1.2 El usuario debe poder guardar los cambios que realizó en un fichero *.mat* con el nombre que quiera.

La aplicación permite el guardado de fichero mediante el uso del diálogo *UISave* que proporciona MATLAB, que es análogo a cualquier guardado de otros programas como el de Word, que permite al usuario escoger donde guardar el fichero y darle nombre. Para guardar basta con darle al botón de guardado y entonces se abrirá este dialogo de guardado.

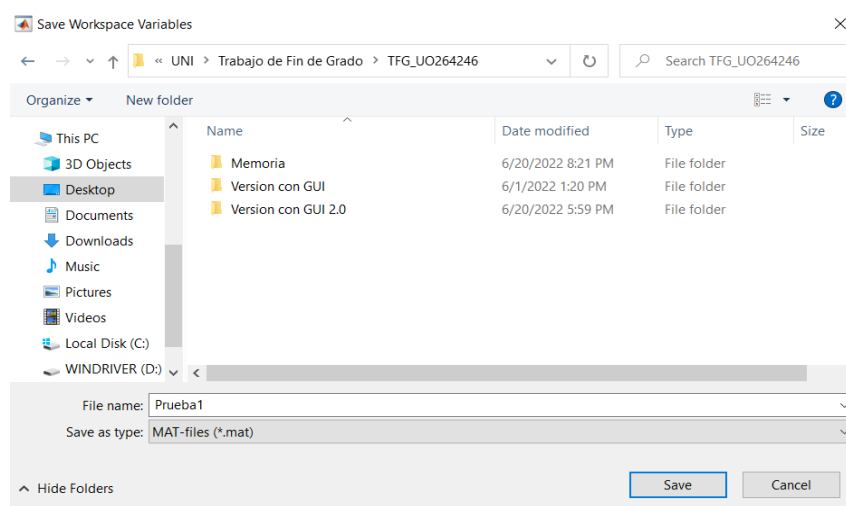


Figura 5.2.- Diálogo de guardado.

1.3 Se debe gestionar que los archivos *.mat* tengan las variables necesarias para el funcionamiento del programa.

Para el correcto funcionamiento de la aplicación, se necesita como mínimo cargar un fichero *.mat* que contenga las variables que almacenen el camino y la matriz de costes. De este modo, si tratamos de cargar un fichero que no contenga estas dos variables, el programa no funcionará y dará un aviso al usuario de que lo que trata de cargar no sirve.

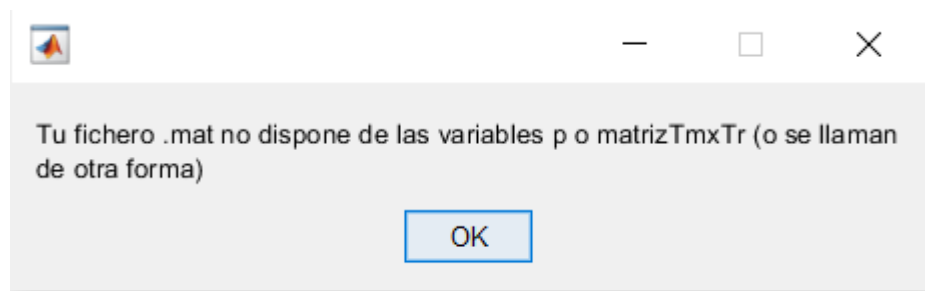


Figura 5.3.- Aviso de carga de fichero incorrecto.

2. Movimiento de los puntos que forman el camino.

Todos estos requisitos reflejan la necesidad funcional de que la aplicación permita el movimiento del camino según desee el usuario. Esta interacción se realiza a través del ratón.

2.1 El usuario debe poder hacer click en uno de los puntos del camino.

El usuario podrá interactuar con cualquier punto del camino para comprobar sus estadísticas. La gestión de estas interacciones se realiza con *callbacks*.

2.2 Tras hacer click en un punto, se podrá arrastrar arriba o abajo y mover hasta donde elija el usuario, que cuando suelte el ratón parará el movimiento.

Este requisito funcional es el que permite al usuario corregir el camino a su gusto, moviendo puntos de una posición a otra en el eje de ordenadas usando el ratón.

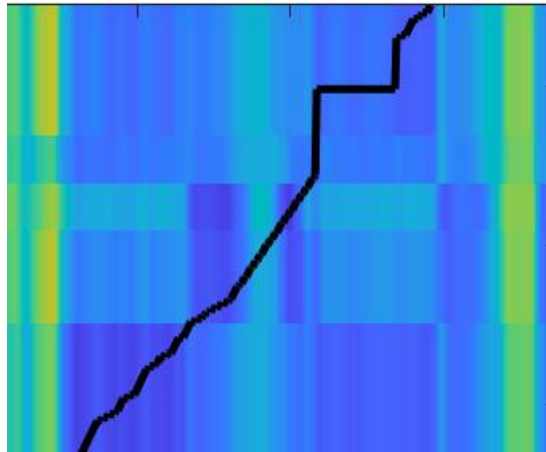


Figura 5.4.- Tramo corregido por el usuario.

3. Corrección del segmento del camino.

Estos requisitos funcionales reflejan la necesidad de poder corregir un segmento del camino seleccionado por el usuario con una interpolación. Como solución para esta interpolación se decide usar una interpolación lineal entre los puntos del segmento que designe el usuario.

3.1 Dado un segmento del camino, el usuario puede hacer que ese segmento desde su origen hasta el final se convierta en una línea recta.

El usuario podrá seleccionar ambos puntos haciendo click en un punto que desee seleccionar como punto del segmento y dándole a la tecla X, siempre que esté en modo corregir.

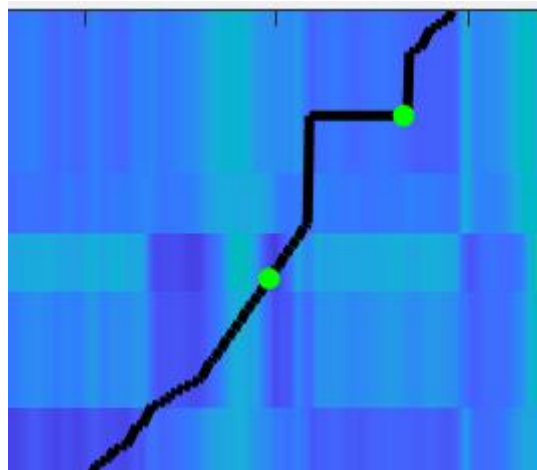


Figura 5.5.- Segmento a corregir seleccionado.

Si finalmente decide corregir el tramo:

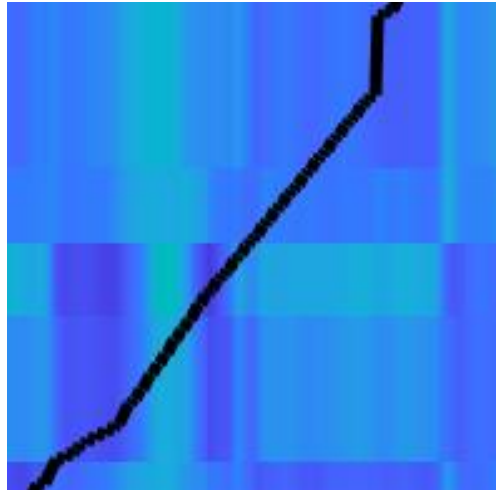


Figura 5.6.- Tramo corregido.

4. Muestra de estadísticas

Estos requisitos funcionales son esenciales para poder mostrarle la información del alineamiento automático al usuario y que este pueda tomar decisiones basándose siempre en información fácilmente accesible.

4.1 Se deben mostrar los valores X e Y de un punto que quiera el usuario.

Siempre que el usuario haga click en un punto del camino, las coordenadas de este se representarán en el panel de estadísticas.

Coordenadas punto	
X value:	9789
Y value:	6434

Figura 5.7.- Coordenadas de un punto.

4.2 Se debe poder evaluar los valores X, Y y el coste almacenado en ese punto de la matriz de costes.

Este requisito se implementa como un modo en el que el usuario puede hacer click en cualquier punto de la matriz y obtener sus coordenadas y el valor de coste almacenado en ese punto. Estos valores se mostrarán en el panel de Matrix value de la aplicación.

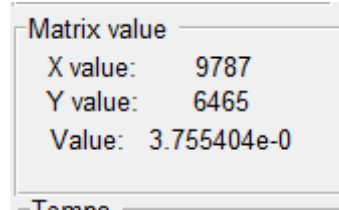


Figura 5.8.- Valor de un punto de la matriz.

4.3 Se debe mostrar el valor del tiempo medio de los datos cargados.

El tiempo medio es una estadística más que puede tener interés para el usuario, por lo que en cuanto se cargan unos datos de alineamiento, se representa el valor del tiempo medio en su respectivo panel en la interfaz.

4.4 Elegido un segmento del camino por el usuario, se debe poder calcular su tiempo medio.

De la misma forma que se pueden pintar dos puntos para corregir un tramo, se pueden pintar dos puntos y evaluar el tiempo medio en ese tramo del segmento con un botón si al usuario le interesa.

5. Tamaño de la nube de puntos que se mueve.

El movimiento de los puntos del camino se realiza con una nube de puntos. Es decir, aparte del punto en el que hace click el usuario, se mueven una serie de puntos adyacentes hacia la izquierda y la derecha. El número de puntos que se mueven depende del tamaño de la nube de puntos, que se puede modificar en la interfaz de usuario.

5.1 El tamaño de la nube de puntos se puede hacer más grande o pequeño según quiere el usuario.

Para cumplir esto el usuario dispone de unos botones en la interfaz para aumentar y disminuir la interfaz de puntos a su gusto después de cargar cualquier fichero con un alineamiento automático.

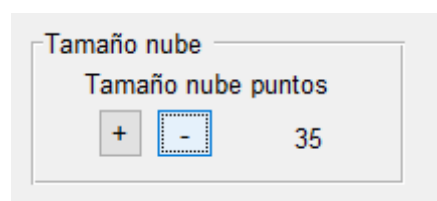


Figura 5.9.- Tamaño de la nube de puntos en la interfaz.

6. Diseño del sistema

En este apartado se describirán los detalles sobre la arquitectura de la aplicación y decisiones de diseño más relevantes que se tomaron durante el desarrollo del software.

6.1.- Diseño de arquitectura

Como todas las aplicaciones GUIDE de MATLAB, *mAlignment* posee un objeto *handles* que tiene todos los datos de la aplicación y que se debe mantener siempre actualizado con los últimos cambios.

Un problema encontrado en el desarrollo de esta aplicación fue que, debido a la complejidad de esta, en ocasiones se debían llamar a *callbacks* dentro de *callbacks* (Un *callback* en MATLAB es una función que se ejecuta como resultado de una acción.) lo que dificulta mantener la estructura *handles* que contiene todos los datos de la aplicación persistente entre las distintas llamadas.

Para solucionar esto se emplea una función central para el funcionamiento del programa, la función *checkHandles*. Esta función, la primera vez que se ejecuta, crea una variable *handles* persistente (es decir, solo existe una instancia de ella en toda la ejecución) a la que se puede acceder en cualquier punto del programa invocando la función sin parámetros, es decir, que funcionará como un típico método *Get*. También, si se lanza el mismo método con una variable *handles* como parámetro, esta función hará que a la variable persistente se le asigne el valor de la introducida por parámetro, es decir, funcionará como un método *Set*. De este modo se puede guardar cualquier cambio en la estructura y además se puede acceder a ella de forma sencilla.

Este mecanismo sería similar al patrón de diseño de *singleton*, que consiste en restringir la creación de objetos pertenecientes a una clase a un único objeto para garantizar que siempre que se use este objeto se obtenga la misma instancia.

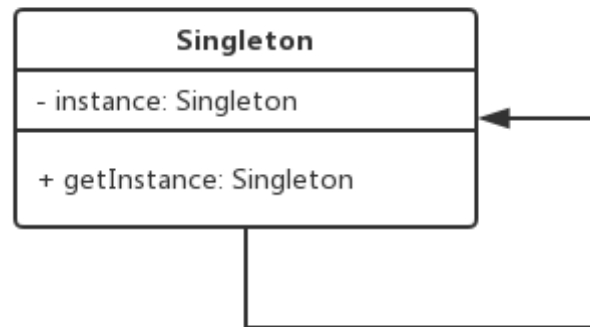


Figura 6.1.- Patrón Singleton

6.2.- Handles

El objeto *handles* en una GUI es una estructura, es decir, un tipo de dato que almacena otros datos en contenedores llamados campos que pueden contener cualquier tipo de datos y a los que se puede acceder a través del *handles*. En *mAlignment*, el *handles* contiene referencias a todos los datos que son de interés para la aplicación para que se pueda acceder a estos de forma sencilla y que siempre estén actualizados gracias al patrón *singleton* que se implementa.

Los datos que almacena *handles* en esta aplicación con su descripción son:

- **Output**

Output es una estructura que contiene todas las variables que forman la GUI de la aplicación. Desde esta variable se pueden acceder a botones, paneles, etiquetas, etc.

- **Figura**

Figura es un objeto de la clase *ejes*, y es el eje en el que se muestran todos los datos de alineamiento.

- **Matriz**

Esta variable almacena un objeto de clase *imagen*, que es la que se dibuja mediante el método *imagesc* usando la matriz de coste obtenida del alineamiento automático para usarla como fondo de los ejes.

- **Xdata, Ydata**

Estas dos variables de *handles* son *arrays* de enteros que representan los valores X e Y del camino dibujado en los ejes. Ydata se modificará en los modos de corregir y mover gráfico, por lo que tener su acceso directo es importante.

- **Mode**

Esta variable almacena un entero que vale 0 o 1 según el modo de la aplicación sea mover gráfico o corregir, sirve para mantener coherencia dentro de la aplicación, ya que podemos acceder en todo momento al estado en el que se encuentra la aplicación.

- **Cloud_size**

Cloud_size es un entero que almacena el tamaño de la nube de puntos que se mueve cuando se cambia la posición de un punto en el modo mover gráfico. Se puede ampliar y reducir desde la GUI y este valor modificado se almacena en *handles* para que el acceso a este sea siempre coherente con el estado de la aplicación.

- **Zoom, Pan, Matriceval**

Estas tres variables funcionan se usan como boolean en Matlab, es decir, valen 0 si su valor es false o 1 si es true. Su uso permite a la aplicación controlar en que momento están activados o desactivados los modos de zoom, pan y evaluar matriz respectivamente.

- **PuntosCorreccion**

Esta variable es un entero que almacena el número de puntos dibujados en la aplicación para elegir un segmento del camino. En todo momento solo puede haber dos puntos dibujados (los extremos de un segmento que elija el usuario), por lo que es importante tener almacenado este número.

- **Puntocorreccion1, Puntocorreccion2**

Estas variables almacenan en un objeto line la representación gráfica de los puntos que el usuario puede dibujar en el gráfico. De esta forma se puede acceder a ellos, sus coordenadas, borrarlos si es necesario, etc.

- **IndiceOrig, IndiceFin**

Estas variables representan dentro de las coordenadas de Xdata, el punto en el que están localizados los puntos del segmento que dibuja el usuario.

- **Linea**

La variable línea mantiene una referencia a la representación gráfica creado en los ejes usando las coordenadas almacenadas en Xdata e Ydata.

- **Puntos_anclaje**

Esta variable almacena los plot de los puntos de anclaje en un objeto Line en caso de que los haya, si no, almacena un 0.

- **Tm_anclaje, Tr_anclaje**

Estas variables se corresponden con las coordenadas de los puntos de anclaje que se cargan desde fichero y que se usarán para dibujar los puntos de anclaje dentro de la aplicación.

- **Anclaje**

La variable anclaje consiste en un vector que es del mismo tamaño que el eje de coordenadas de X, ya que cada elemento se corresponde con un índice de este eje. En caso de que no haya puntos de anclaje, todo el vector estará relleno de 0, pero si hay puntos de anclaje, el índice de X en el que haya un punto de anclaje valdrá 1 en esta variable. De este modo se podrá comprobar fácilmente en que puntos se encuentran los puntos de anclaje y se modelará el comportamiento de la aplicación con esta información.

6.3.- Métodos de la aplicación

Los métodos de la aplicación se dividirán en tres tablas, según su función dentro de la aplicación. Estarán divididos entre métodos de inicialización, métodos de gestión de modos y métodos auxiliares.

Métodos de inicialización

Se incluyen aquí los métodos que inicializan la aplicación y que sirven para cargar y almacenar los cambios realizados:

Método	Parámetros	Descripción	Retorno
inicializar	hObject, handles	Este método inicializa todos los campos de la estructura <i>handles</i> .	Nulo
openFile_ClickedCallback	hObject, eventdata, handles	Método que controla el callback del menú para abrir ficheros.	Nulo
loadButton_Callback	hObject, eventdata, handles	Método del botón de cargar ficheros, análogo al anterior.	Nulo
saveButton_Callback	hObject, eventdata, handles	Método del botón que controla el callback para guardar ficheros.	Nulo

Tabla 6.1.- Métodos de inicialización.

Métodos de gestión de modos

Estos métodos agrupan todos aquellos que son necesarios para el funcionamiento de los modos de la aplicación y las transiciones entre estados que existen. De esta forma, se hablará tanto de los métodos callback que ocurren cuando haces click en un botón como métodos que realizan funciones de estos modos, como los encargados de dibujar los puntos en el modo corrección. De este modo:

Método	Parámetros	Descripción	Retorno
changeModeButton_Callback	hObject, eventdata, handles	Método encargado de gestionar el cambio de modos entre corregir y mover gráfico.	Nulo
buttonup	-	Método que se encarga de que al levantar el ratón el usuario tras hacer click en un punto del camino se obtenga el comportamiento esperado.	Nulo
buttondown	Obj,~,handles	Método que gestiona lo que pasa al clickar en un punto del camino, con distintos resultados según el modo de ejecución.	Nulo
move	hObject, ~, Obj, ind, ydata, cloud_size, handles	Método que gestiona el movimiento del gráfico cuando el usuario clicka y arrastra el ratón en el eje de las Y.	Nulo
correctButton_Callback	hObject, eventdata, handles	Se encarga de corregir el tramo de un segmento si están ambos extremos dibujados.	Nulo
zoomButton_Callback	hObject, eventdata, handles	Activa/Desactiva el modo zoom.	Nulo
panButton_Callback	hObject, eventdata, handles	Activa/Desactiva el modo pan	Nulo
zoomOutButton_Callback	hObject, eventdata, handles	Desactiva tanto el modo zoom como el pan y restaura el zoom del gráfico al original.	Nulo

evalMatrixButton_Callback	hObject, eventdata, handles	Activa el modo evaluar matriz.	Nulo
setPuntoCambio	~, ~, hObject, ind	Método encargado de dibujar los puntos del segmento en el modo corrección cuando el usuario lo quiere. Gestiona todo el movimiento del usuario, devolviendo un vector y_data que contiene los valores de las coordenadas y cambiadas.	Nulo
gestionPuntos	ydata, ind, move, cloud_size		n_ydata

Tabla 6.2.- Métodos de gestión de modos.

Métodos auxiliares.

Son los métodos en los que se apoyan los anteriores para realizar cálculos y consultas sobre los datos (por ejemplo, para saber dónde hay puntos de anclaje) cuando lo necesitan.

Método	Parámetros	Descripción	Retorno
plusCSButton_Callback	hObject, eventdata, handles	Método encargado de gestionar el cambio de modos entre corregir y mover gráfico	Nulo
minusCSButton_Callback	hObject, eventdata, handles	Método que se encarga de que al levantar el ratón el usuario tras hacer click en un punto del camino se obtenga el comportamiento esperado.	Nulo
tempoSegmento_Callback	hObject, eventdata, handles	Método que gestiona lo que pasa al clickar en un punto del camino, con distintos resultados según el modo de ejecución.	Nulo
getTempo	inicio, fin	Este método recibe los índices de inicio y fin del segmento y calcular el tempo medio, que retorna en la variable tempo	tempo
getTempoMedio	-	Calcula el tempo medio del audio en su totalidad y lo retorna en la variable tempo.	tempo

isAnclaje	rankVector, anclajearray	Retorna un booleano bool con valor 1 si hay un punto de anclaje en los rangos introducidos por rankVector del array anclajearray.	bool
corregirTramo	handles	Funcion que corrige el segmento elegido por el usuario. Retorna n_ydata, los nuevos valores de las coordenadas y tras corregirlo.	n_ydata
gestionCloudSize	hObject, key, handles	Se encarga de manejar tanto el aumento del tamaño de la cloud_size como su disminución, retornando un handles con el nuevo cloud_size.	handles
evalMatrix	Obj, ~, handles	Método encargado de obtener las estadísticas de un punto en el que clicka el usuario.	Nulo
setButtons	bool	Este método desactiva o activa múltiples botones de la aplicación según se le introduzca un booleano bool por parámetro true o false.	Nulo
setButtons	id, mode	Este método activa o desactiva el botón que tenga el identificador id según el valor del parámetro mode, que debe ser on o off según se quiera o no activar.	Nulo

Tabla 6.3.- Métodos auxiliares.

6.4.- Modos de ejecución

Debido a las funcionalidades que necesita la aplicación para implementar los requisitos de usuario, se debe implementar varios modos de ejecución dentro de la aplicación y lograr que estos funcionen de forma coherente entre sí, pudiendo navegar entre ellos sin encontrarse errores.

Los modos de ejecución que dispone la ejecución son:

- Modo mover

Este modo de ejecución prepara la aplicación para que en caso de que el usuario haga click en un punto del gráfico pueda moverlo de forma vertical para colocarlo en el tramo que elija. Desde este modo se puede acceder a el resto de los modos a través de botones.

- Modo corregir

En este modo el usuario puede elegir un segmento del camino dibujando dos puntos que lo empiecen y acaben para medir el tempo medio de ese tramo o corregir el tramo, interpolando ambos puntos con una línea recta. Los puntos no se conservan una vez se cambia de modo.

- Modo evaluar matriz

Este modo sirve para que el usuario pueda consultar los valores de coordenadas y de coste la matriz de fondo.

- Modo navegación del gráfico

Esto incluye los modos zoom y pan, que al ser ambos modos muy similares se tratan en el mismo punto, son los modos que permiten al usuario observar el gráfico de distintas formas y hacer zoom en aspectos concretos o moverse en el gráfico una vez se hizo zoom hasta un punto concreto.

6.5.- Diseño interfaz

A lo largo del desarrollo del proyecto se tuvieron varios diseños de interfaz que mejoraron incrementalmente hasta obtener la interfaz final.

La aplicación empezó con una primera versión en la que ni siquiera existía una interfaz GUI más allá de la Figura creada por MATLAB, pero tras el aprendizaje de la herramienta GUIDE ya se creó una interfaz de usuario basada en dos botones para abrir los datos y crear en una figura externa el gráfico que se podía mover, corregir y usar las propias funcionalidades de la figura para hacer zoom y moverlo:

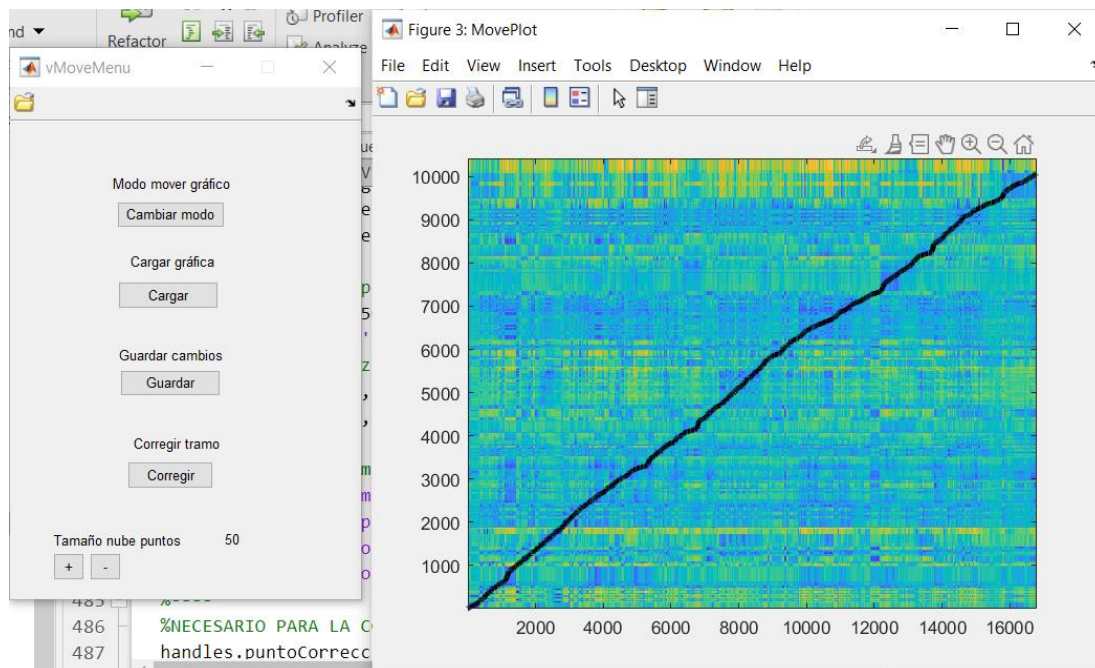


Figura 6.2.- Interfaz antigua de la aplicación.

Como se puede ver en la Figura 6.2, es una interfaz muy básica. Esta tenía el problema de que la figura y el menú estaban separados, lo que no permitía tener una aplicación compacta, aunque tenía la facilidad de poder usar las herramientas de *zoom*, *pan* y *datatips* incluidas en la propia figura de MATLAB, lo que ahorra programarlas.

Tras trabajar en cambiar la interfaz y una vez se logró mayor familiaridad con el funcionamiento de MATLAB y GUIDE, se pasó a implementar todas las funcionalidades de esta interfaz en una más compacta y que visualmente fuese más agradable para el usuario, en la que se muestran en distintos paneles las estadísticas que el usuario puede querer, las herramientas que antes estaban dentro de la figura y los modos en los que puede operar la aplicación.

Este incremento, que ya muestra una versión más definitiva de la interfaz se muestra así al iniciarlo:

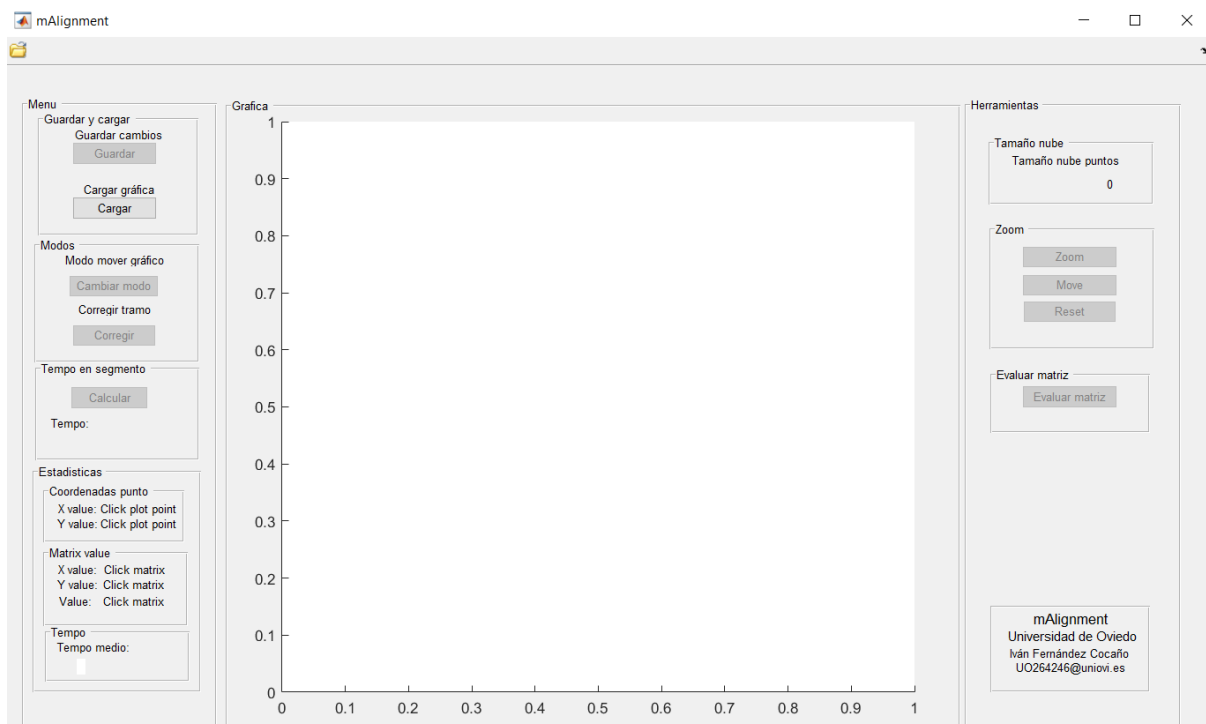


Figura 6.3.- Interfaz actual, al iniciar la aplicación.

Esta interfaz es dimensionable y mucho más clara, lo que facilita la realización de pruebas y el uso de la propia aplicación.

7. Planificación temporal del desarrollo

La planificación del proyecto se realizó mediante un diagrama de Gantt, dividiendo la planificación en dos procesos:

- **Desarrollo:** En el desarrollo se recogen todas las tareas realizadas durante el desarrollo software de la aplicación.
- **Documentación:** Las tareas de documentación recogen todo el proceso de realizar la memoria, presupuesto y documentos técnicos a presentar. Una vez el peso del desarrollo está casi terminado, se comienza a realizar la documentación.

Como se podrá ver a continuación, el desarrollo de la aplicación consta de varios hitos de entregas en las que se logra cada vez un prototipo funcional que implementa progresivamente más de las funcionalidades requeridas para la aplicación, empezando desde el modo de mover gráficos hasta los modos de zoom y pan.

De esta forma, la planificación en GANNT será:

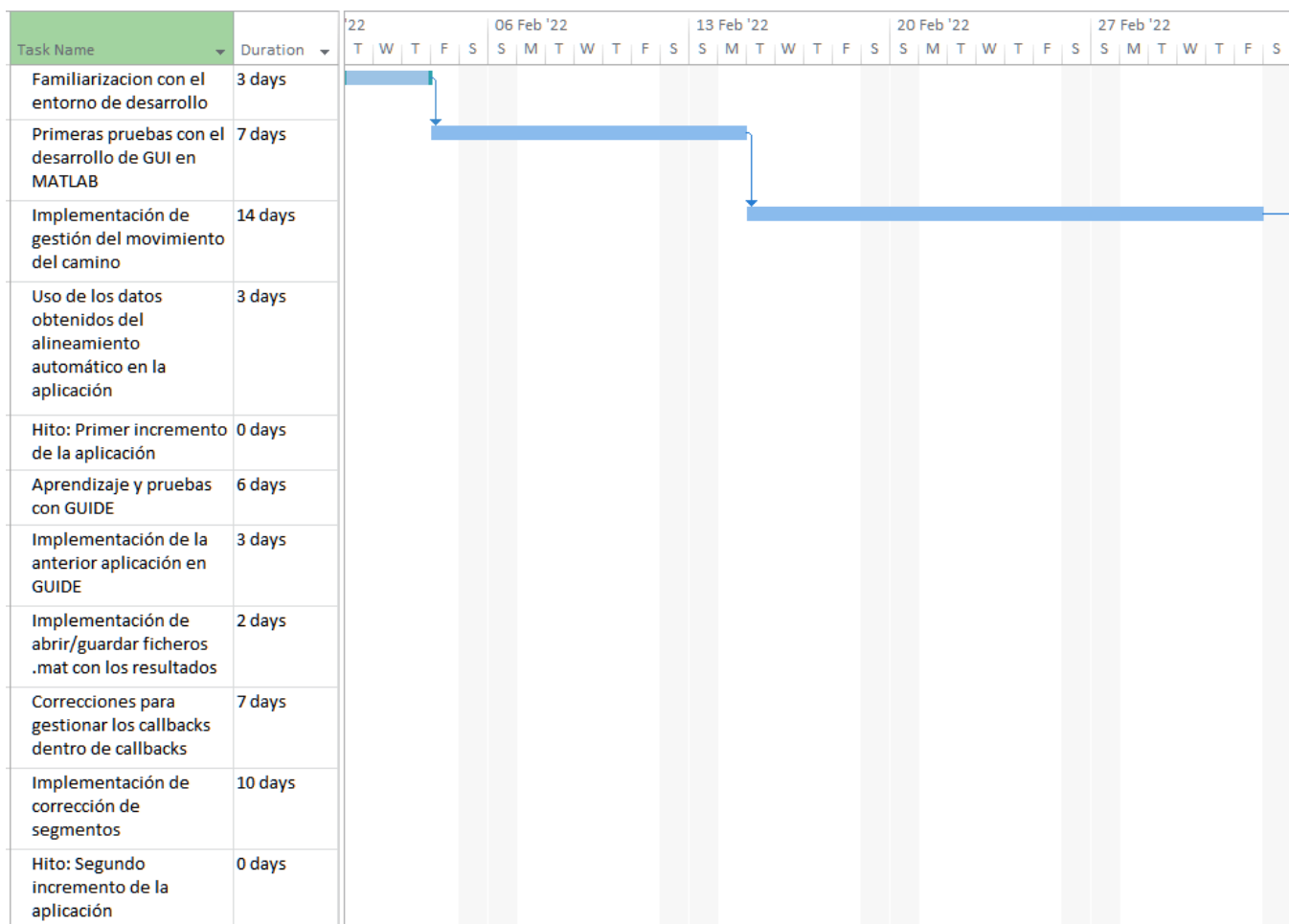


Figura 7.1.- Planificación Gannt 1.

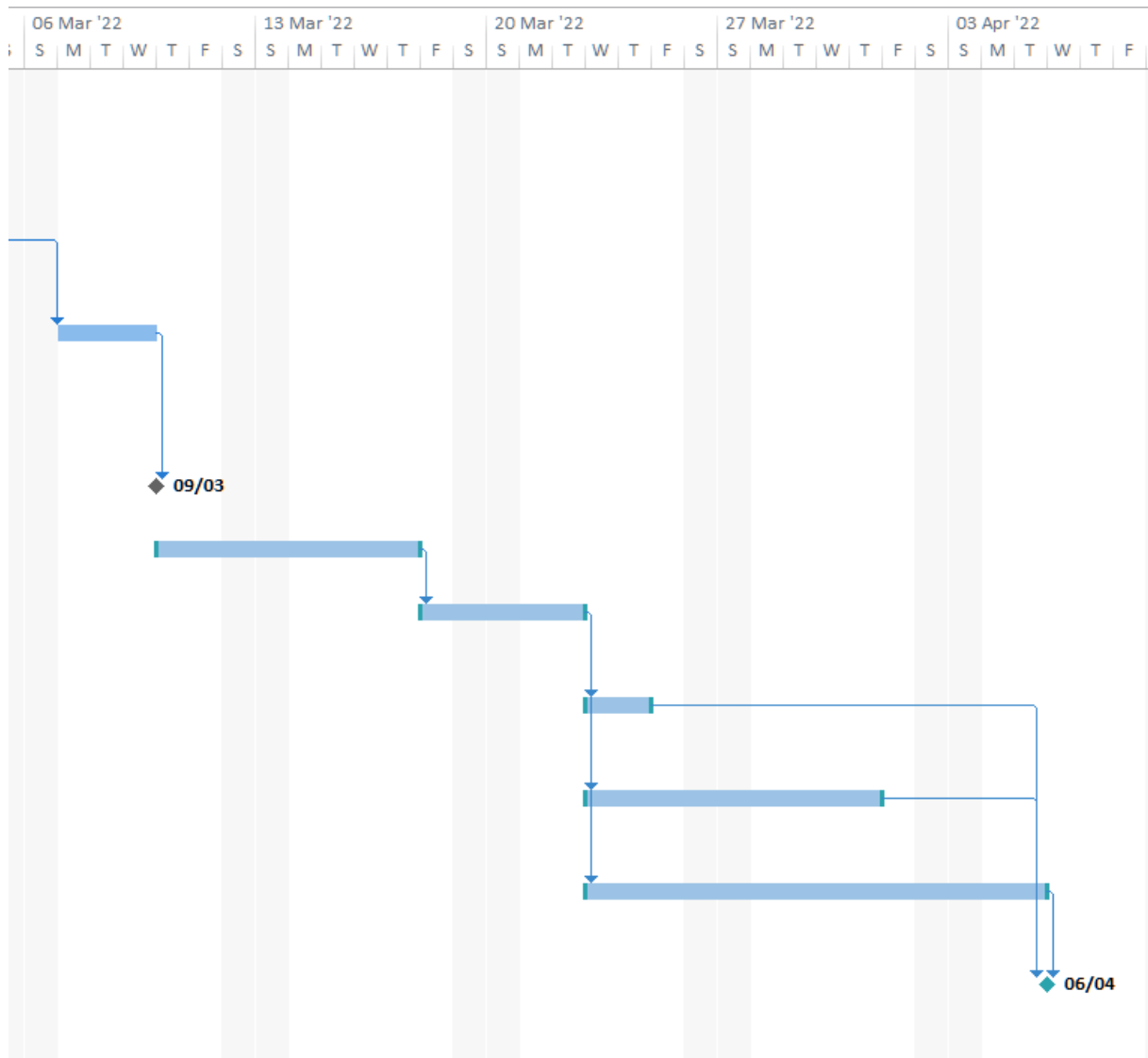


Figura 7.2.- Planificación Gannt 2.

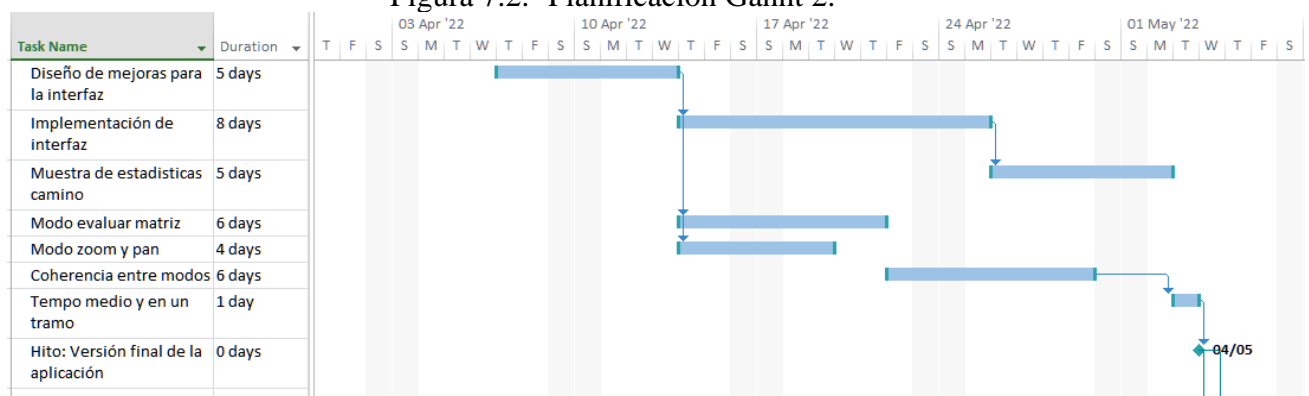


Figura 7.3.- Planificación Gannt 3.

Y la planificación de la documentación, que se comienza poco antes de finalizar el desarrollo de la aplicación:

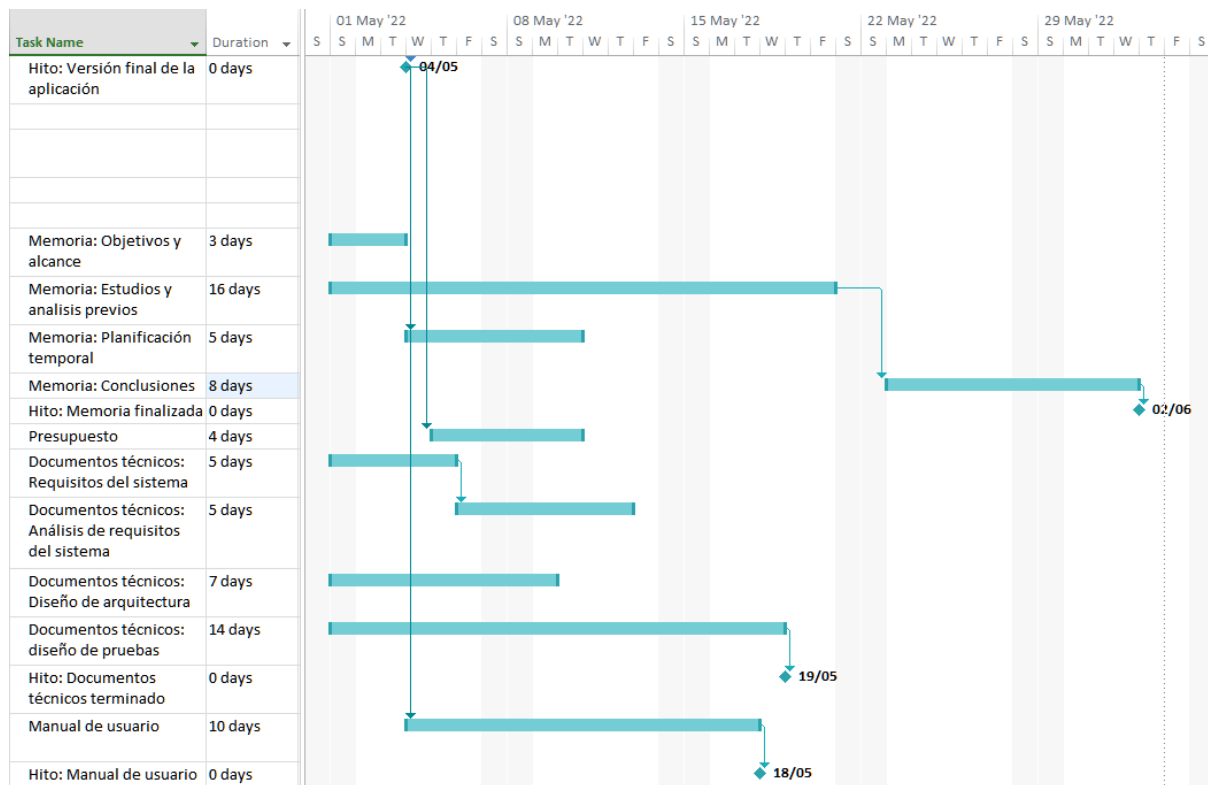


Figura 7.4.- Planificación Gannt 4.

8. Presupuesto del desarrollo efectuado.

En este capítulo se mostrarán los gastos que conlleva el desarrollo de la aplicación. Dividiremos el presupuesto en dos capítulos, el primero, de costes de personal que representa los gastos de personal de realizar el proyecto y el segundo de hardware y software que representa los costes de las herramientas necesarias para realizarlo.

En primer lugar, el capítulo I, de costes de personal:

Descripción	Ud	P. Unitario	Cantidad	Total
Técnico de Desarrollo software	h	25	550	13,750.00 €
Jefe de proyecto	h	55	55	3,025.00 €
Total				16,775.00 €

Tabla 8.1.- Tabla del capítulo I del presupuesto: Costes de personal.

Y a continuación el de hardware y software:

Descripción	Ud	P. Unitario	Cantidad	Total
PC	Ud	249	1	249.00 €
Licencia Windows 10 Pro	Ud	259	1	259.00 €
Licencia MATLAB (anual)	Ud	800	1	800.00 €
Total				1,308.00 €

Tabla 8.2.- Tabla del capítulo II del presupuesto: Costes de hardware y software.

En resumen, los costes de ambos capítulos serán:

Capítulo	Nombre	Precio
Cap. I	Coste personal	16,775.00 €
Cap. II	Hardware y software	1,308.00 €
Total capítulos (PEM)		18,083.00 €

Tabla 8.3.- Resumen de los capítulos.

Y, por tanto, se pueden calcular los beneficios industriales, costes generales e impuestos de este presupuesto de esta forma:

Costes		Total
Gastos generales	PEM (13%)	2,350.79 €
Beneficio industrial	PEM (6%)	1,084.98 €
Total parcial		21,518.77 €
Impuestos	IVA (21%)	4,518.94 €
Presupuesto total		26,037.71 €

Tabla 8.4.- Costes parciales y total final del presupuesto.

9. Diseño y ejecución de pruebas

Para el diseño de pruebas se aplicarán dos técnicas de pruebas que proporcionarán una serie de tests exhaustivos y sólidos, con el objetivo de encontrar fallos en el software que puedan ser solucionados para tener una versión final que funcione a la perfección.

9.1.- Diseño de pruebas

A continuación, se explican las dos técnicas de pruebas que se van a aplicar para generar los tests que se aplicarán al software.

La primera técnica empleada será la de clases de equivalencia, que representan un conjunto de datos para los que se supone que la aplicación tendrá un comportamiento determinado. Se identifican las clases de equivalencia y se numeran con identificaciones que se corresponderán con al menos una de las pruebas ejecutadas para garantizar la trazabilidad de nuestro diseño de pruebas. De este modo:

9.1.1- Clases de equivalencia

Se dividen las clases de equivalencia entre los “modos” que tiene la aplicación, para tener una división que haga más legible estas clases de equivalencia.

1. General

1.1 Carga de ficheros

- 1.1.1 Carga de fichero *.mat* con puntos de anclaje.
- 1.1.2 Carga de fichero *.mat* sin puntos de anclaje
- 1.1.3 Carga de fichero *.mat* al que le falta alguna variable necesaria.
- 1.1.4 Al cargar un fichero que se guarda con cambios, se observa que los cambios continúan.

1.2 Guardar ficheros

- 1.2.1 Guardar fichero *.mat* con resultados con anclaje.
- 1.2.2 Guardar fichero *.mat* sin resultados sin anclaje.

1.3 Tamaño nube de puntos

- 1.3.1 Se debe poder aumentar el tamaño de la nube de puntos.
- 1.3.2 Se debe poder reducir el tamaño de la nube de puntos

2. Modo mover gráfico

2.1 El usuario puede hacer click en un punto del camino y se debe mostrar los valores de las coordenadas de ese punto.

2.2 Si mantiene apretado y arrastra el punto puede moverlo a otro punto.

2.2.1 Si el movimiento incluye a un punto que sea de anclaje, no se podrá realizar.

3. Modo corrección

3.1 Selección de segmento del camino

3.1.1 El usuario puede hacer click en un punto y seleccionarlo como principio de un segmento a corregir.

3.1.2 Puede hacerlo una vez más para colocar otro punto y determinar así el segmento.

3.1.4 El usuario pinta dos puntos, pero primero el final y después el inicio del segmento.

3.1.5 No se pueden dibujar más puntos.

3.2 Calculo del tempo medio de un segmento.

3.2.1 Si el segmento tiene ambos puntos determinados se puede calcular el tempo medio.

3.3 Corrección del segmento

3.3.1 Si el segmento tiene ambos puntos de inicio y fin determinados se puede corregir y que pase a ser una recta.

3.3.2 Si el segmento incluye un punto de anclaje no se podrá corregir.

4. Modo evaluar matriz

4.1 Coherencia de modos

4.1.1 Se debe poder entrar al modo evaluar matriz desde el modo mover, y al volver se seguirá en el modo mover.

4.1.2 Se debe poder entrar al modo evaluar matriz desde el modo corregir, y al volver se seguirá en el modo mover.

4.2 Al hacer click en cualquier punto de la matriz muestra sus coordenadas y el valor de la matriz de costes en ese punto.

5. Modo navegación del gráfico

5.1 Coherencia de modos

5.1.1 Se debe poder entrar al modo zoom desde el modo mover, y al volver se seguirá en el modo mover.

5.1.2 Se debe poder entrar al modo zoom desde el modo corregir, y al volver se seguirá en el modo mover.

5.1.3 Se debe poder entrar al modo pan desde el modo mover, y al volver se seguirá en el modo mover.

5.1.4 Se debe poder entrar al modo pan desde el modo mover, y al volver se seguirá en el modo mover.

5.2 El modo zoom permite acercar en una parte del gráfico que elija el usuario.

5.3 El modo pan permite mover el gráfico una vez este tiene algo de zoom.

5.4 Podemos regresar al zoom original en el gráfico.

9.1.2.- Pares de caminos

En aplicaciones en las que se encuentran cambios de estado en respuestas a eventos y condiciones que causa el usuario, es de gran interés encontrar una serie de pruebas que permitan asegurar que no hay fallos en las ejecuciones de estas transiciones entre estados y, sobre todo, que en ningún momento la aplicación llega a un estado inválido.

De todas las técnicas de pruebas basadas en caminos, se aplica la técnica conocida como pares de caminos, por la cual se derivan casos de prueba para cubrir cada uno de los caminos de cada par de caminos adyacentes, logrando una gran intensidad de testeo en el funcionamiento de los estados de la aplicación y sus transiciones.

Antes de nada, se realiza un esquema en el que se pueden observar todas las transiciones de estados permitidas dentro de la aplicación. De este esquema, se sacan los casos de prueba por pares que comprobarán el funcionamiento de esas transiciones de estados:

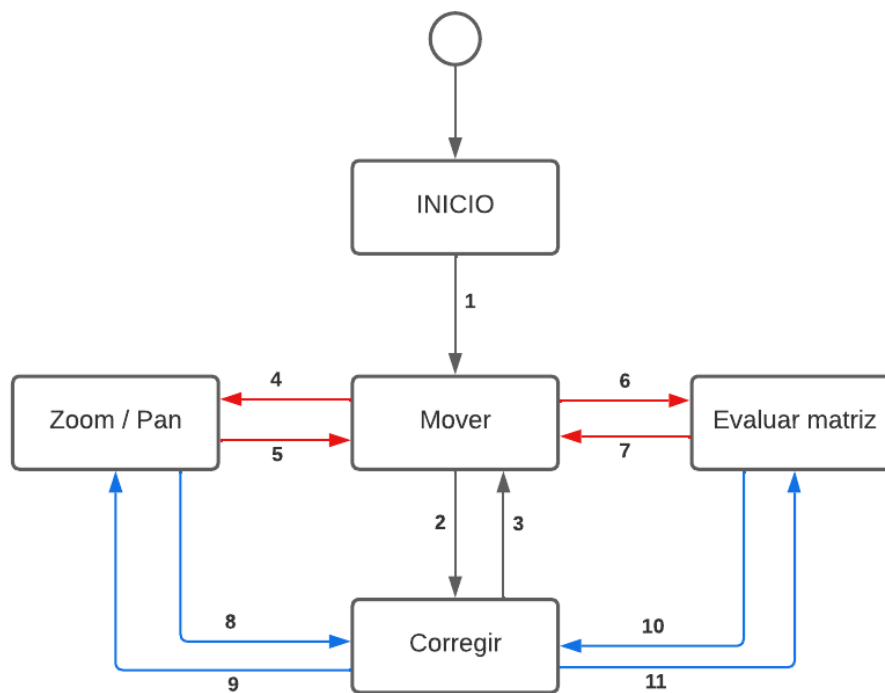


Figura 9.1.- Esquema de estados de la aplicación para aplicar pruebas de caminos.

Los colores de las flechas rojo y azul representan que una vez llegas a un estado por una flecha de ese color, al salir vuelves por una flecha de ese mismo color (Por ejemplo, si se entra desde el modo *Mover* a *Evaluar matriz* por la flecha 6, solo se podrá salir de este estado de vuelta al estado *Mover* a través de la flecha 7).

Las situaciones de prueba que se tendrán que cubrir son las que vienen de cada par de caminos adyacentes (y que cumplan la restricción antes indicada), en este caso:

Caminos a cubrir

1 -> 2

1 -> 4

1 -> 6

2 -> 3

2 -> 9

2 -> 11

3 -> 6

3 -> 4

3 -> 2

4 -> 5

6 -> 7

5 -> 2
5 -> 6
5 -> 4
7 -> 2
7 -> 4
7 -> 6
8 -> 3
8 -> 11
8 -> 9
9 -> 8
10 -> 3
10 -> 9
11 -> 10

Tabla 9.1.- Tabla de los caminos por cubrir.

Con estos caminos a cubrir se crean las situaciones de prueba, que se corresponderán con casos de prueba para probar que el software funciona como debe. Las identificaciones de estas situaciones de prueba con el caso de prueba al que corresponden son:

Situaciones	ID Caso prueba asociado
1 -> 4 -> 5 -> 2 -> 3 -> 6 -> 7	6.1
1 -> 6 -> 7 -> 2 -> 11 -> 10 -> 3	6.2
1 -> 2 -> 9 -> 8 -> 9 -> 8 -> 11	6.3
1 -> 4 -> 5 -> 4 -> 5 -> 6 -> 7 -> 4	6.4
1 -> 6 -> 7 -> 6	6.5
1 -> 2 -> 9 -> 8 -> 3	6.6
1 -> 2 -> 11 -> 10 -> 9	6.7

Tabla 9.2.- Tabla de situaciones de prueba derivadas con sus casos de prueba asociados.

De esta forma, con los casos de prueba derivados de la técnica de pares de caminos y los que se obtendrán de al menos cumplir que toda clase de equivalencia se vea probada en al menos un caso de prueba, se asegura la trazabilidad y se obtiene un diseño de pruebas potente que permitirá asegurar que la aplicación mAlignment no contenga fallos.

9.2.- Resultados de ejecución

A continuación, se muestran los resultados de las pruebas diseñadas, basadas en las clases de equivalencia, en las pruebas obtenidas por caminos y en posibles pruebas que se

consideraron de interés para encontrar posibles errores derivados de que al saber cómo fue desarrollo del software, se conoce en qué partes la implementación fue más compleja y costosa.

Se dividen las pruebas en grupos según las funcionalidades de la aplicación que cubren.

De esta forma:

1. Gestión de ficheros

En este apartado se tratan la gestión de los ficheros *.mat* que realiza la aplicación, que permite cargar y guardar ficheros.

1.1 Carga de ficheros.

Este apartado se encarga de comprobar que la carga de ficheros *.mat*, tanto si tienen anclaje como no, funciona como debe.

1.1.1 Carga de fichero incorrecto.

Esta prueba comprueba a la clase de equivalencia 1.1.3.

- Objetivo: Tratar de cargar un fichero *.mat* incorrecto.
- Entrada: Un fichero *.mat* que dispone de la variable P para el camino pero no de la matriz de costes.
- Salida esperada: Salta el aviso de que no se dispone de las variables necesarias para que funcione el software
- Resultado: Se avisa como se esperaba de falta información.

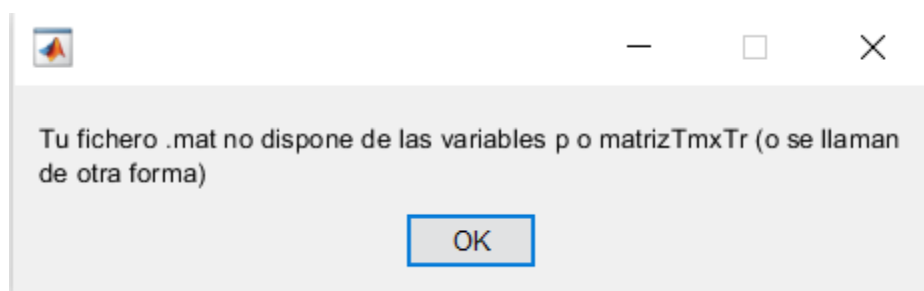


Figura 9.2.- Aviso de error al cargar el fichero.

1.1.2 Carga de fichero con anclaje.

Esta prueba comprueba las clases de equivalencia 1.1.1 y 1.1.4.

- Objetivo: Cargar un fichero *.mat* de un alineamiento que usa puntos de anclaje. Este fichero contiene un cambio que se hizo con la propia herramienta anteriormente y se refleja en la carga.
- Entrada: Fichero Prueba2.mat que contiene los datos de alineamiento y los puntos de anclaje necesarios y un cambio hecho con la herramienta que se guardó previamente.
- Salida esperada: Se carga el fichero con el anclaje y el cambio realizado se puede ver.
- Resultado: Se carga satisfactoriamente el fichero, mostrándose tanto los puntos de anclaje como los cambios que se deben ver.

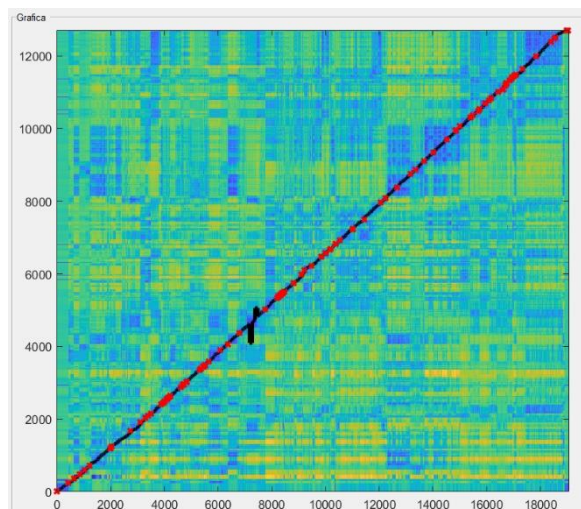


Figura 9.3.- Gráfica de la aplicación mostrando puntos de anclaje y cambios realizados.

1.1.3 Carga de fichero sin anclaje.

Esta prueba comprueba la clase de equivalencia 1.1.3.

- Objetivo: Cargar un fichero *.mat* que no contiene puntos de anclaje para comprobar el funcionamiento de la carga sin anclaje.
- Entrada: El fichero Prueba1.mat, que contiene los datos de alineamiento sin usar puntos de anclaje.

- Salida esperada: Se espera que se muestren los datos de alineamiento cargados.
- Resultado: Se muestra correctamente la matriz de costes con el camino sin puntos de anclaje, como se esperaba.

1.2 Guardar ficheros

Este apartado se encarga de comprobar que el guardado de ficheros *.mat*, tanto si tienen anclaje como no, funciona como debe.

1.2.1 Guardar fichero con anclaje

Esta prueba comprueba la clase de equivalencia 1.2.1

- Objetivo: Comprobar el funcionamiento del guardado usando un alineamiento que tenga puntos de anclaje.
- Entrada: Se guarda con el botón de guardado el alineamiento actualmente cargado que dispone de puntos de anclaje (Se puede cargar previamente Prueba2.mat como entrada para guardarlo de nuevo)
- Salida esperada: Se espera que se cree un fichero *.mat* con todas las variables, la *p*, la matriz de costes y las dos variables de anclaje necesarias.
- Resultado: Se guarda correctamente el fichero esperado.

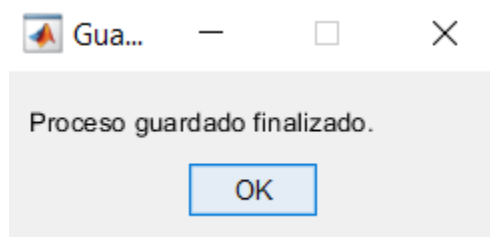


Figura 9.4.- Confirmación del guardado.

1.2.2 Guardar fichero sin anclaje

Esta prueba comprueba la clase de equivalencia 1.2.2.

- Objetivo: Comprobar el funcionamiento del guardado usando un alineamiento que no tenga puntos de anclaje.

- Entrada: Se guarda con el botón de guardado el alineamiento actualmente cargado que dispone de puntos de anclaje (Se puede cargar previamente Prueba1.mat para guardarlo de nuevo)
- Salida esperada: Se espera que se cree un fichero .mat la p y la matriz de costes.
- Resultado: Se guarda correctamente el fichero esperado.

2. Modo mover gráfico

El modo mover gráfico es uno de los modos esenciales de la aplicación, que permite al usuario mover puntos del camino generado de forma automática a su gusto. Por esto, se crean varias pruebas para evaluar su correcto funcionamiento:

2.1 Aumentar la nube de puntos, mover un punto escogido y guardar el cambio.

Esta prueba comprueba las clases de equivalencia 2.1, 2.2, 1.2.1, 1.3.1.

- Objetivo: Comprobar que la carga de un fichero y los cambios que se realicen en él se registran al guardarlo después. Se prueba también en dos cambios distintos como aumentar el tamaño de la nube de puntos hace que la parte del camino seleccionado que se mueve sea mayor.
- Entrada: El fichero Prueba2.mat, que se carga para a continuación modificarlo.
- Salida esperada: La carga, el movimiento de puntos y el aumento del tamaño de la nube de puntos funciona.
- Resultado: Se puede observar cómo funciona correctamente todo.

2.2 Tratar de mover un punto y que la nube de puntos a mover incluya un punto de anclaje.

Esta prueba comprueba las clases de equivalencia 2.2.1

- Objetivo: Comprobar que los puntos de anclaje no se pueden mover en el modo de mover gráfico.
- Entrada: Cargamos el fichero de Prueba2.mat, que contiene los datos de un alineamiento con puntos de anclaje.
- Salida esperada: Al intentar mover una serie de puntos entre los que se incluye un punto de anclaje, el programa avisa de que estos puntos son inamovibles y que por tanto no se permite el movimiento.

- Resultado: Como se espera, aparece un mensaje que avisa al usuario de que el movimiento realizado no está permitido.

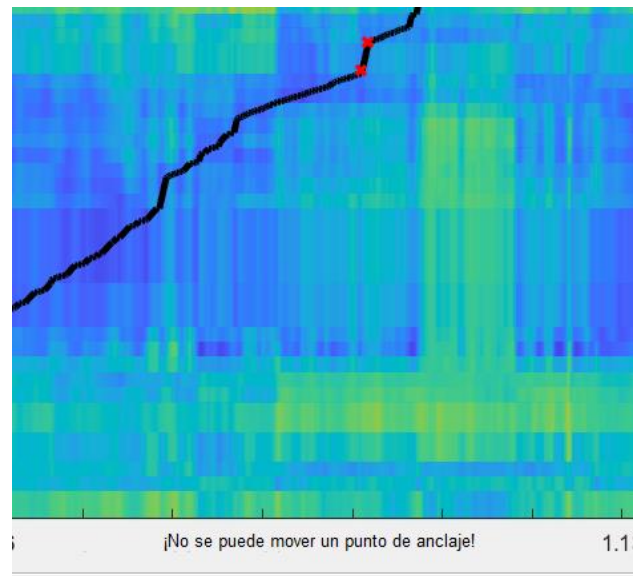


Figura 9.5.- Aviso de punto de anclaje.

2.3 Reducir el tamaño de la nube de puntos

Esta prueba comprueba las clases de equivalencia 1.3.2

- Objetivo: Comprobar que se puede reducir el tamaño de la nube de puntos del modo mover gráfico.
- Entrada: Para probarlo podemos cargar el fichero de prueba Prueba1.mat (Aunque en este caso es irrelevante que haya puntos de anclaje o no).
- Salida esperada: Se puede observar que, al reducir la nube de puntos, se mueven menos puntos al realizar cambios en el gráfico.
- Resultado: Se observan que se reduce la nube de puntos correctamente y esto afecta al modo mover gráfico.

2.4 Tratar de reducir el tamaño de la nube a 0.

Esta prueba comprueba las clases de equivalencia 1.3.3

- Objetivo: Asegurar que la nube de puntos a mover no pueda adquirir valores negativos o 0.
- Entrada: Para probarlo podemos cargar el fichero de prueba Prueba1.mat (Aunque en este caso es irrelevante que haya puntos de anclaje o no).

- Salida esperada: El tamaño de la nube de puntos no disminuirá y además se avisará al usuario de que no puede bajar de 0.
- Resultado: La nube de puntos se mantiene en tamaño 1 y se avisa al usuario como se esperaba.

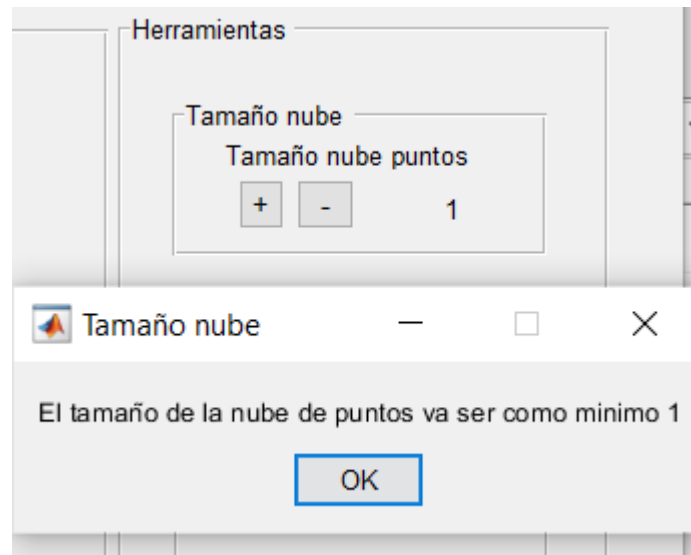


Figura 9.6.- Aviso tamaño de nube mínimo.

3. Modo corrección

El modo corrección de la aplicación permitirá al usuario seleccionar un segmento del camino y corregirlo mediante interpolación. Las pruebas para evaluar este modo son:

3.1 Corrección de los puntos

Con estas pruebas se pretende comprobar que lo relacionado con la corrección de los puntos funciona correctamente y se cumplen las restricciones necesarias de la aplicación.

3.1.1 Se elige un segmento del camino para corregirlo.

Esta prueba comprueba las clases de equivalencia 3.1.1, 3.1.2, 3.3.1

- Objetivo: Probar la funcionalidad básica del modo corregir, esto es, dibujar ambos extremos de un segmento a corregir y corregirlo.
- Entrada: Tras cargar el fichero Prueba1.mat, se corrige un segmento a elegir por el usuario.

- Salida esperada: Se dibujarán con éxito ambos puntos y luego se podrá corregir ese segmento.
- Resultado: Se dibujan ambos puntos y se corrige el segmento sin problemas.

3.1.2 Se elige un segmento del camino para corregirlo, haciendo que el punto inicial dibujado vaya delante del final.

Esta prueba comprueba las clases de equivalencia 3.1.4

- Objetivo: Demostrar que no hay restricciones a la hora de poner el segundo punto del segmento antes o después del primero que se dibuje.
- Entrada: Para probar esto se puede cargar el fichero Prueba1.mat y realizar los cambios sobre él.
- Salida esperada: Independientemente del orden de los puntos del segmento, se puede corregir.
- Resultado: Se corrige correctamente el segmento, sin importar el orden de dibujo de los puntos.

3.1.3 Se intenta corregir un tramo que incluye un punto de anclaje.

Esta prueba comprueba las clases de equivalencia 3.3.2

- Objetivo: Probar que no se puede corregir un tramo que incluya puntos de anclaje.
- Entrada: Se carga el fichero Prueba2.mat que contiene un alineamiento con puntos de anclaje.
- Salida esperada: Se espera que se muestre un mensaje al usuario de que no se puede corregir un segmento del camino que incluye un punto de anclaje.
- Resultado: Se puede visualizar el mensaje avisando de que es imposible corregir el segmento seleccionado.

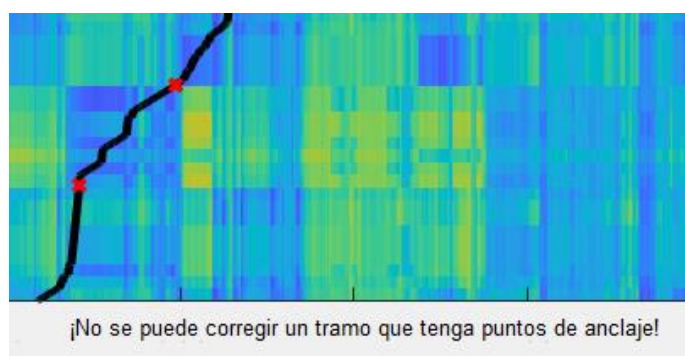


Figura 9.7.- Aviso de tramo del segmento con puntos de anclaje.

3.2 Comprobación lógica interna

Estas pruebas se enfrentan al problema de mantener una coherencia interna entre todos los cambios de modos. Esto es, que si por ejemplo dibujamos un punto, cambiamos de modo al de mover y se regresa al de corregir, el modo corregir siga funcionando correctamente y pida que dibuje dos puntos para el segmento de nuevo.

3.2.1 Después de pintar dos puntos intenta pintar uno más.

Esta prueba comprueba la clase de equivalencia 3.1.5

- Objetivo: Comprobar que la aplicación limita a 2 el número de puntos que se puede dibujar para corregir (los dos puntos que forman el segmento.)
- Entrada: Se carga el fichero de prueba Prueba1.mat.
- Salida esperada: Se avisa al usuario de que no puede dibujar más de un punto.
- Resultado: Se muestra un mensaje al usuario informándole de que no puede dibujar más puntos.

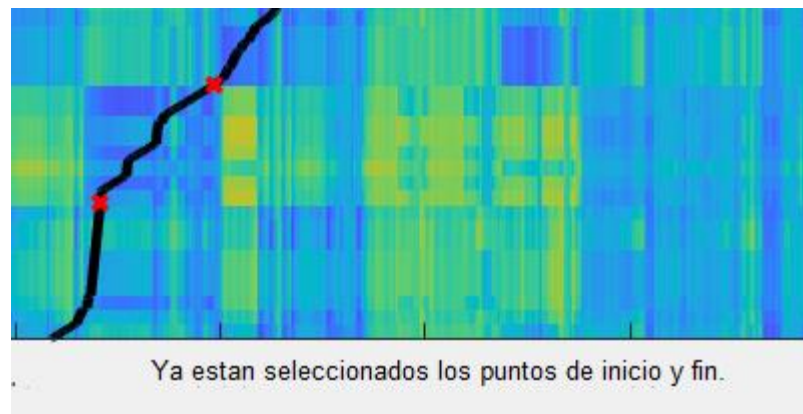


Figura 9.8.- Aviso de que el segmento ya está formado.

3.2.2 Se pinta un punto, cambia de modo a mover, vuelve a corregir, pinta dos puntos y corrige.

- Objetivo: Asegurar que hay coherencia entre los cambios de modo.
- Entrada; Cargamos el fichero de Prueba1 para probar el cambio de modos.
- Salida esperada: Toda la transición entre modos funciona correctamente.
- Resultado: La transición entre modos es coherente

3.2.3 El usuario pinta dos puntos, cambia de modo, vuelve a cambiar, pinta dos puntos y corrige.

- Objetivo: Asegurar que hay coherencia entre los cambios de modo.
- Entrada: Cargamos el fichero de Prueba1 para probar el cambio de modos.
- Salida esperada: Toda la transición entre modos funciona correctamente.
- Resultado: Funciona como se espera.

4. Modo evaluar matriz.

Con esta prueba se comprueba el funcionamiento correcto del modo de evaluar los datos de la matriz de costes.

4.1 Se evalúa un punto de la matriz de costes.

Esta prueba comprueba la clase de equivalencia 4.1

- Objetivo: Comprobar el funcionamiento del modo evaluar matriz.
- Entrada: Se carga el fichero Prueba1.mat para comprobar los valores de su matriz de costes.
- Salida esperada: Al hacer click en el modo evaluar matriz en cualquier punto de la matriz se muestran las estadísticas de ese punto.
- Resultado: Se muestran los resultados como se espera.

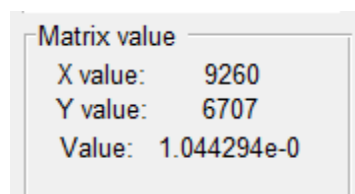


Figura 9.9.- Estadísticas de un punto de la matriz.

5. Modo zoom / pan

Estas pruebas se encargan de asegurar el funcionamiento de los modos zoom y pan de la aplicación.

5.1 Se activa el modo zoom y aumenta la zona que desea.

Esta prueba comprueba la clase de equivalencia 5.2

- Objetivo: Comprobar el modo zoom.
- Entrada: Se carga el fichero Prueba1.mat

- Salida esperada: Se hace zoom al punto que el usuario desea.
- Resultado: El modo zoom funciona correctamente.

5.2 Tras hacer zoom, se activa el modo pan para mover el gráfico a un punto determinado.

Esta prueba comprueba la clase de equivalencia 5.3

- Objetivo: Mostrar el correcto funcionamiento del modo pan.
- Entrada: Cargamos el fichero Prueba1.mat para hacer zoom en un punto determinado y después activar el modo pan.
- Salida esperada: El usuario puede moverse con el ratón arrastrando por donde desee.
- Resultado: El modo pan funciona correctamente.

5.3 Se reinicia el zoom para que se vea el gráfico en su totalidad

Esta prueba comprueba la clase de equivalencia 5.4

- Objetivo: Resetear el zoom.
- Entrada: Se carga el fichero Prueba1.mat y se hace zoom en una parte determinada.
- Salida esperada: Se espera que al resetear el zoom, la visualización de los datos vuelva a ser la misma que la que se muestra nada más cargarlos.
- Resultado: Observamos que el reinicio del zoom funciona correctamente.

6. Coherencia de estados (Pares de caminos)

Estas pruebas son las que se obtienen de la técnica de pares de caminos y que se ejecutan para asegurar que el funcionamiento entre todos los modos es correcto y que no se dan comportamientos extraños en la aplicación.

6.1 Prueba de caminos 1

Esta prueba comprueba las clases de equivalencia 4.1.1, 5.1.1.

- Objetivo: Comprobar una serie de transiciones de estado obtenidas de las pruebas de caminos por pares.
- Entrada: Se carga el fichero Prueba1.mat para realizar toda la transición de estados.

- Ejecución: El usuario va moviéndose entre los distintos estados usando los botones. Debe atravesar los estados inicio, mover gráfico, zoom, mover gráfico, corregir, mover gráfico, evaluar matriz y mover gráfico.
- Salida esperada: Que se mantenga coherente el funcionamiento de los estados de la aplicación.
- Resultado: La aplicación funciona como se espera.

6.2 Prueba de caminos 2

Esta prueba comprueba la clase de equivalencia 4.1.2.

- Objetivo: Comprobar una serie de transiciones de estado obtenidas de las pruebas de caminos por pares.
- Entrada: Se carga el fichero Prueba1.mat para realizar toda la transición de estados.
- Ejecución: El usuario va moviéndose entre los distintos estados usando los botones. Debe atravesar los estados inicio, mover gráfico, evaluar matriz, mover gráfico, corregir, evaluar matriz, corregir, mover gráfico.
- Salida esperada: Que se mantenga coherente el funcionamiento de los estados de la aplicación.
- Resultado: La aplicación funciona como se espera.

6.3 Prueba de caminos 3

- Objetivo: Comprobar una serie de transiciones de estado obtenidas de las pruebas de caminos por pares.
- Entrada: Se carga el fichero Prueba1.mat para realizar toda la transición de estados.
- Ejecución: El usuario va moviéndose entre los distintos estados usando los botones. Debe atravesar los estados inicio, mover gráfico, evaluar matriz, mover gráfico, corregir, evaluar matriz, corregir, mover gráfico.
- Salida esperada: Que se mantenga coherente el funcionamiento de los estados de la aplicación.
- Resultado: La aplicación funciona como se espera.

6.4 Prueba de caminos 4

- Objetivo: Comprobar una serie de transiciones de estado obtenidas de las pruebas de caminos por pares.
- Entrada: Se carga el fichero Prueba1.mat para realizar toda la transición de estados.

- Ejecución: El usuario va moviéndose entre los distintos estados usando los botones. Debe atravesar los estados inicio, mover gráfico, pan, mover gráfico, zoom, mover gráfico, evaluar matriz, mover gráfico, zoom.
- Salida esperada: Que se mantenga coherente el funcionamiento de los estados de la aplicación.
- Resultado: La aplicación funciona como se espera.

6.5 Prueba de caminos 5

- Objetivo: Comprobar una serie de transiciones de estado obtenidas de las pruebas de caminos por pares.
- Entrada: Se carga el fichero Prueba1.mat para realizar toda la transición de estados.
- Ejecución: El usuario va moviéndose entre los distintos estados usando los botones. Debe atravesar los estados inicio, mover gráfico, evaluar matriz, mover gráfico, evaluar matriz.
- Salida esperada: Que se mantenga coherente el funcionamiento de los estados de la aplicación.
- Resultado: La aplicación funciona como se espera.

6.6 Prueba de caminos 6

Esta prueba comprueba la clase de equivalencia 5.1.2, 5.1.4.

- Objetivo: Comprobar una serie de transiciones de estado obtenidas de las pruebas de caminos por pares.
- Entrada: Se carga el fichero Prueba1.mat para realizar toda la transición de estados.
- Ejecución: El usuario va moviéndose entre los distintos estados usando los botones. Debe atravesar los estados inicio, mover gráfico, corregir, zoom, mover.
- Esta prueba se repite también cambiando el modo zoom del final por el pan para comprobar que funciona correctamente para este modo también.
- Salida esperada: Que se mantenga coherente el funcionamiento de los estados de la aplicación.
- Resultado: La aplicación funciona como se espera.

6.7 Prueba de caminos 7

- Objetivo: Comprobar una serie de transiciones de estado obtenidas de las pruebas de caminos por pares.
- Entrada: Se carga el fichero Prueba1.mat para realizar toda la transición de estados.

- Ejecución: El usuario va moviéndose entre los distintos estados usando los botones. Debe atravesar los estados inicio, mover gráfico, corregir, evaluar matriz, corregir, zoom.
- Salida esperada: Que se mantenga coherente el funcionamiento de los estados de la aplicación.
- Resultado: La aplicación funciona como se espera.

De esta forma se realiza el diseño y la ejecución de pruebas, con una serie de pruebas que tienen la trazabilidad de todas las clases de equivalencia y las pruebas exigidas por la técnica de pares de caminos. No se encuentra ningún error en las pruebas, resultado de que a lo largo de los incrementos de la aplicación se puso sumo cuidado en que fuesen versiones totalmente funcionales y evitar así desarrollar un producto que arrastre bugs que tras varios incrementos se hiciesen demasiado grandes.

Se puso un gran esfuerzo además, en que los distintos modos de ejecución funcionen entre sí sin darse problemas unos entre otros a la hora del desarrollo de la aplicación, lo que se refleja en que, en las pruebas más complejas, como las de caminos no se encuentren fallos, ya que por ejemplo, los distintos estados en los que puede estar la aplicación y su coherencia al pasar de uno a otro fueron aspectos muy trabajados y que, durante el desarrollo se probaron y corrigieron numerosas veces hasta obtener la solución final, que no tiene fallos.

10. Conclusiones y trabajo futuro

En este apartado se contemplan las conclusiones obtenidas tras todo el desarrollo del proyecto y las posibles líneas de trabajo futuro para completar aún más la aplicación desarrollada.

10.1.- Conclusiones

Tras el final del proyecto, se tiene desarrollada una herramienta que cumple todos los objetivos iniciales propuestos y que permite a un usuario trabajar con los datos obtenidos de un alineamiento de audio-partitura automático

Cabe destacar que abordar todo el proyecto, desde su creación hasta obtener la aplicación final, permitió al autor familiarizarse con todo el entorno de desarrollo de aplicaciones GUI Matlab, no solo con la plataforma GUIDE, si no con otras aplicaciones como el AppDesigner, que en un futuro serán el estándar de desarrollo de aplicaciones GUI en MATLAB. El desarrollo en este software era nuevo para el autor, pero el enfrentarse a toda la problemática que surge de realizar un proyecto como este le permitió aprender y perfeccionar el uso de MATLAB.

En resumen, el autor considera que se lograron los hitos presentados inicialmente:

- Se comprende el concepto de alineamiento automático audio-partitura.
- Se domina el uso del lenguaje MATLAB con una programación óptima en él.
- Se aprendió a usar la herramienta GUIDE desarrollo de GUIs e incluso se introduce el uso de la herramienta AppDesigner, que también sirve para desarrollar interfaces de usuario.
- Se logra estar familiarizado con el funcionamiento de la programación orientada a eventos dentro de las GUI de MATLAB.
- Se completa un desarrollo de la aplicación incremental, una forma de desarrollar software con la que es muy útil estar familiarizado, por la relevancia que tiene en metodologías ágiles de programación como SCRUM.

10.2.- Trabajo futuro

Tras finalizar el trabajo, hay varias formas con las que se podría mejorar aún más el software desarrollado. Se pueden plantear las siguientes líneas de trabajo futuro:

- Probar distintos tipos de interpolación para corregir un tramo de alineamiento. Podría diseñarse alguna función que tuviera en cuenta la inercia y el tempo de la interpretación.
- Se podrían diseñar nuevas herramientas que permitan al usuario modificar aún más los datos que cargue, por ejemplo, que le permitiesen a él mismo asignar puntos de anclaje donde él decida.
- Habría que plantear la internacionalización de la interfaz, que ahora mismo está solo en castellano, para facilitar su uso en varios idiomas.
- Mejorar la interfaz. Hay indicaciones que podrían señalarse de forma más visual al usuario, por ejemplo, cuando el usuario trata de mover un punto de anclaje, podría señalarse el punto de anclaje que impide el movimiento cambiándolo de color momentáneamente.

11. Bibliografía

- [1] S. Dixon, "Live Tracking Of Musical Performances Using On-Line Time Warping," in *Conference on Digital Audio Effects*, Madrid, 2005.
- [2] O. Picas and J. Carabias-Orti, "An Audio To Score Alignment Method Using Velocity-Driven DTW for MIREX 2014," 2014.
- [3] A. Muñoz-Montoro, R. Cortina, S. García-Galán, E. Combarro and J. Ranilla, "A Score Identification Parallel System Based On Audio-to-Score Alignment," *The Journal Supercomputing*, pp. 8830-8844, 2020.
- [4] F. Rodriguez-Serrano, J. Carabias-Orti, P. Vera-Candeas and D. Martinez Munoz, "Tempo Driven Audio-To-Score Alignment Using Spectral Decomposition And Online Dynamic Time-Warping," *ACM Transactions On Intelligent Systems and Technology*, pp. 1-20, 2016.
- [5] J. J. Carabias-Orti, F. J. Rodriguez-Serrano, P. Vera-Candeas, N. Ruiz-Reyes and F. J. Canadas-Quesada, "An Audio to Score Alignment Framework Using Factorization and Dynamic Warping," in *ISMIR*, Malaga, 2015.

12. Manual de usuario de mAlignment

MANUAL DE USO DE
SISTEMA DE ALINEAMIENTO AUDIO-PARTITURA
SEMIAUTOMÁTICO
mAlignment

D. FERNÁNDEZ COCAÑO, Iván
TUTOR: D. MUÑOZ MONTORO, Antonio Jesús

Contenido

Capítulo 1.- Introducción.....	89
Capítulo 2.- Instalación	91
2.1.- Instalación inicial.....	91
2.2.- Sobre los datos usados.....	91
Capítulo 3.- Descripción general de la interfaz gráfica del Usuario (GUI).....	93
3.1.- Interfaz gráfica.....	93
Capítulo 4.- Modos de ejecución.....	101
4.1.- Modo mover gráfico	101
4.2.- Modo corrección.....	102
4.3.- Modo navegación gráfico	103
4.4.- Modo evaluar matriz.....	105
4.5.- Puntos de anclaje	106
Capítulo 5.- Tutorial	109
5.1.- Primeros pasos.....	109
5.2.- Ejemplo de obtención de datos	109
5.3.- Carga de ficheros	111
5.4.- Navegación del gráfico.....	113
5.5.- Mover el gráfico	114
5.6.- Corregir segmentos del camino	115
5.7.- Evaluar puntos de la matriz de costes.....	117
5.8.- Otras funcionalidades	117
5.8.1.- Guardar cambios	117
5.8.2.- Calcular tempo en un tramo	118
Capítulo 6.- Posibles modificaciones	119
6.1.- Creación de puntos de anclaje	119

6.2.- Variación de corrección de segmentos.....	120
---	-----

Índice de figuras

Figura 12.1.- Ejemplo de la aplicación en uso.	89
Figura 12.2.- Ejemplo de representaciones sin y con puntos de anclaje.	92
Figura 12.3.- La interfaz que ve el usuario.....	93
Figura 12.4.- Panel menú.....	94
Figura 12.5.- Panel guardar y cargar.	94
Figura 12.6.- Panel modos.....	95
Figura 12.7.- Panel tempo en segmento.	95
Figura 12.8.- Panel estadísticas.	96
Figura 12.9.- El panel Gráfica antes de cargar datos de alineamiento.	96
Figura 12.10 Panel Herramientas.	97
Figura 12.11 Panel tamaño nube.	97
Figura 12.12.- Panel zoom.....	98
Figura 12.13.- Gráfica aumentada en un punto concreto.	98
Figura 12.14.- Panel evaluar matriz.....	98
Figura 12.15.- Etiqueta modo evaluar.	99
Figura 12.16.- Modos de ejecución.	101
Figura 12.17.- Muestra del modo actual.....	102
Figura 12.18.- Parte del camino modificada por el usuario.....	102
Figura 12.19.-Ejemplo de segmento del camino seleccionado por el usuario.	102
Figura 12.20.- Segmento tras ser corregido.....	103
Figura 12.21.- Usuario elige una sección sobre la que hacer zoom.	104
Figura 12.22.- Gráfico ampliado.	104
Figura 12.23.- Transiciones de los modos de navegación.....	105
Figura 12.24.- Valores de un punto de la matriz de costes.....	106
Figura 12.25.- Puntos de anclaje en el camino.	106

Figura 12.26.- Ejemplo de punto de detención del código del alineamiento automático.	110
Figura 12.27.- Workspace en el punto de parada.....	110
Figura 12.28.- Guardado de fichero con datos de alineamiento.	111
Figura 12.29.- Ejemplo de fichero sin anclaje obtenido.	111
Figura 12.30.- Opciones de carga de ficheros <i>.mat</i>	112
Figura 12.31.- Opciones de carga de ficheros <i>.mat</i>	112
Figura 12.32.- Aplicación tras cargar los datos.	113
Figura 12.33.- Modos de navegación del gráfico.....	113
Figura 12.34.- Modo pan en la aplicación.	114
Figura 12.35.- Rango de movimientos en modo mover gráfico.	115
Figura 12.36.- Cambiar al modo corregir.	115
Figura 12.37.- Creación del segmento a corregir.....	116
Figura 12.38.- Segmento corregido.....	116
Figura 12.39.- Botón para entrar en el modo evaluar matriz.	117
Figura 12.40.- Ventana de guardado de cambios.....	118
Figura 12.41.- Botón de calcular tempo en un segmento.....	118

Capítulo 1.- Introducción

Con este manual se pretende orientar a nuevos usuarios en el uso de la aplicación *software* mAlignment desarrollada para permitir la corrección de los resultados obtenidos por algoritmos de alineamiento audio-partitura.

En los resultados de los alineamientos, cuando se trabaja con audios con cierta complejidad o con errores del propio músico al tocar, pueden verse decisiones que toma el algoritmo para formar el camino que están equivocados o que no tienen sentido. Con esta aplicación se pretende, visualizando la matriz de coste y el camino que el algoritmo propone, ayudar al usuario a ver dónde están los errores y que este mismo pueda corregir el camino a mano, llevándolo hasta el punto deseado.

Esta aplicación desarrollada en MATLAB ofrece una interfaz gráfica sencilla y manejable que permitirá realizar los cambios que se necesite en los resultados que dé el algoritmo, añadiendo una capa de intervención humana al alineamiento para solucionar los problemas que puedan ocurrir en este.

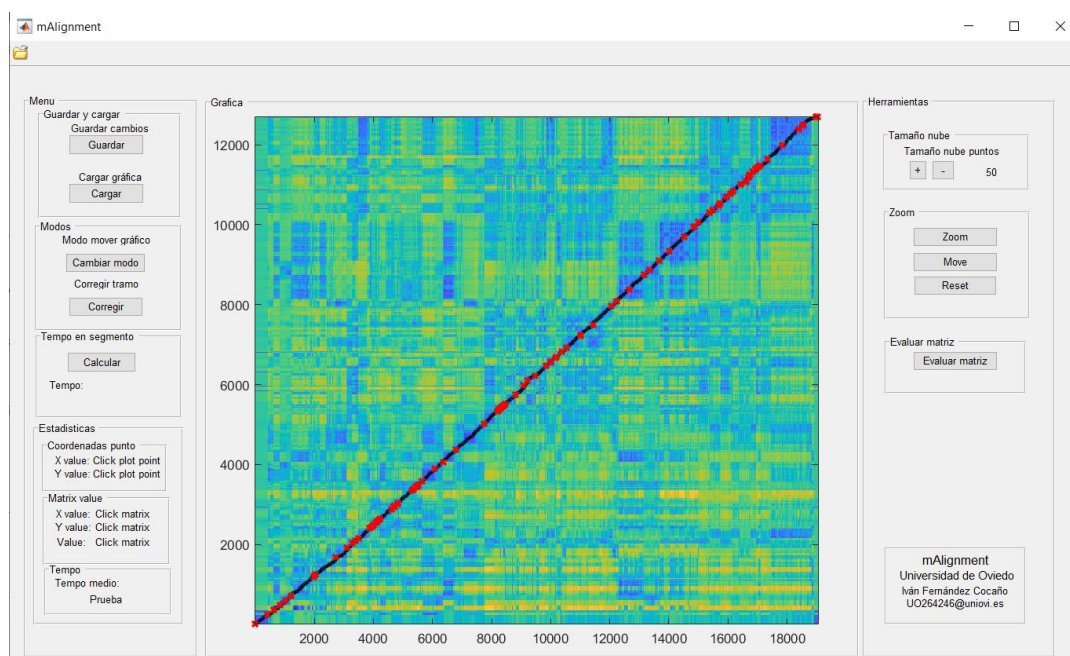


Figura 12.1.- Ejemplo de la aplicación en uso.

Capítulo 2.- Instalación

2.1.- Instalación inicial

Para el correcto funcionamiento de `mAlignment`, es necesario que el ordenador donde se quiera ejecutar disponga previamente de:

1. MATLAB R2021b instalado.
2. Descomprimir en una carpeta la aplicación `mAlignment`, que se compone de dos archivos, el fichero `mAlignment.fig` y `mAlignment.m`.

Para la ejecución de la aplicación basta con abrir el fichero `.m` y ejecutarlo como si fuese un programa cualquiera de MATLAB.

2.2.- Sobre los datos usados

En esta aplicación se pueden abrir los resultados de un alineamiento a partir de ficheros `.mat`. La única condición es que, para abrirlos, deben contener una serie de variables que contengan la información necesaria para la correcta representación de los resultados, que es:

- Una variable '`p`' que es un vector que representa el camino que el algoritmo de alineamiento escogió para los datos que se le introdujo.
- Una variable '`matrizTmxTr`' que es la matriz de costes que usó el algoritmo para la toma de decisiones.

Como `mAlignment` es capaz de funcionar también con puntos de anclaje, que son puntos que el algoritmo considera tan importantes que deben ser inamovibles incluso para el usuario de la aplicación, se puede opcionalmente también cargar un fichero `.mat` con las variables '`tm_anclaje`' y '`tr_anclaje`', que representan las coordenadas de los puntos de anclaje que se mostrarán también al usuario.

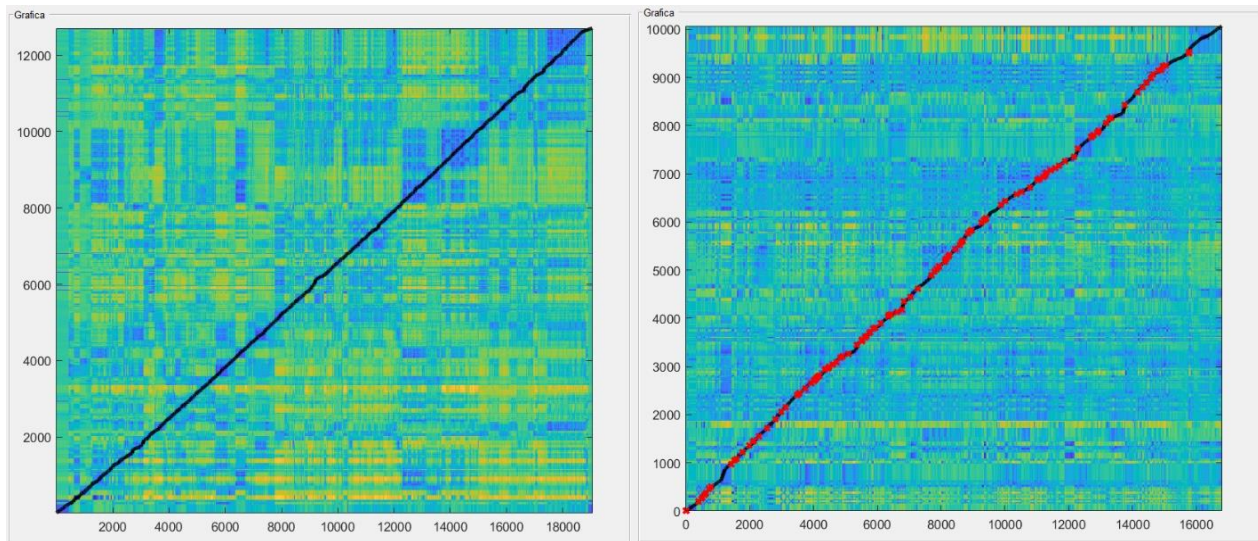


Figura 12.2.- Ejemplo de representaciones sin y con puntos de anclaje.

Como se puede observar, los puntos de anclaje se muestran al usuario de forma muy vistosa como puntos rojos. Más adelante se hablará más en profundidad de estos puntos de anclaje y de cómo interactúan con la aplicación

Capítulo 3.- Descripción general de la interfaz gráfica del Usuario (GUI)

Esta aplicación se compone por una interfaz gráfica en la que se dividen entre distintos paneles las funcionalidades a las que el usuario puede acceder, con la intención de que esté toda la información presente de forma sencilla y accesible.

A continuación, se explican todas las partes que forman la interfaz de la aplicación:

3.1.- Interfaz gráfica

En esta figura se muestra la interfaz general con la que trabajará el usuario:

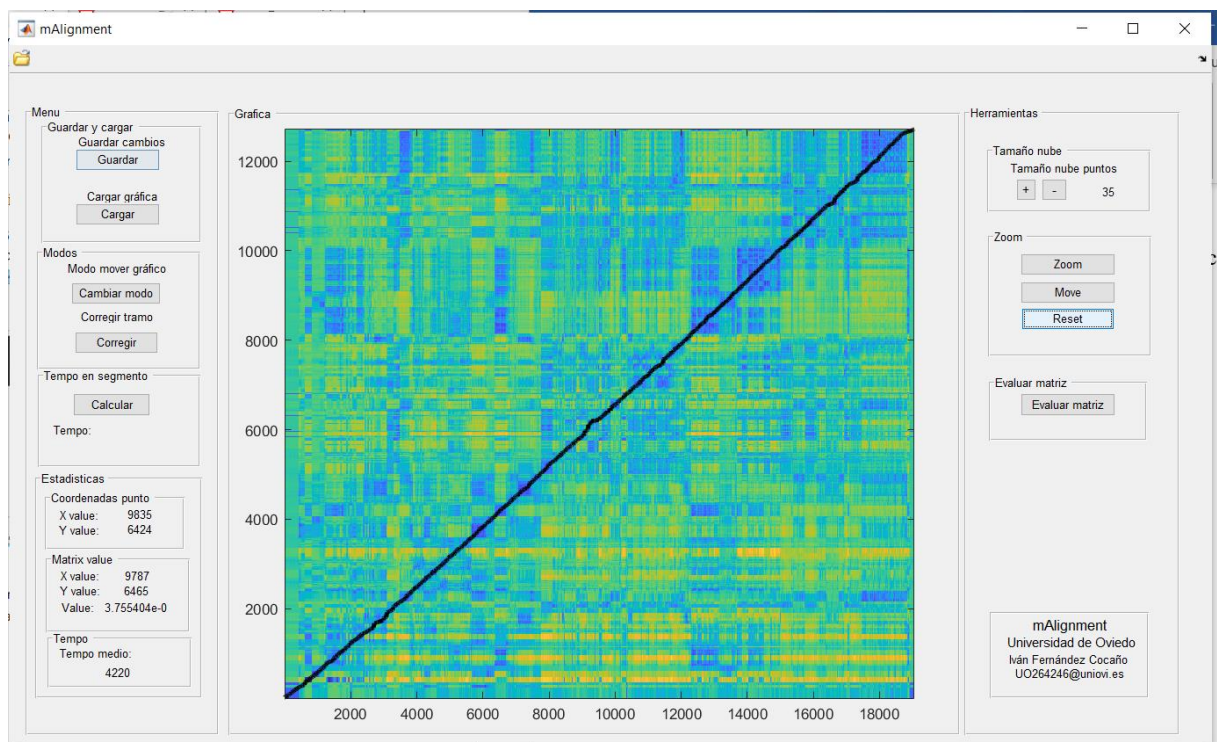


Figura 12.3.- La interfaz que ve el usuario.

Para explicar la interfaz de la aplicación por partes, se dividirán las explicaciones entre los distintos paneles, de forma que si empezamos desde el panel de la izquierda de Menú:

3.1.1.- Panel Menú

Este panel contiene más paneles que permiten al usuario interactuar y visualizar los datos de la aplicación.

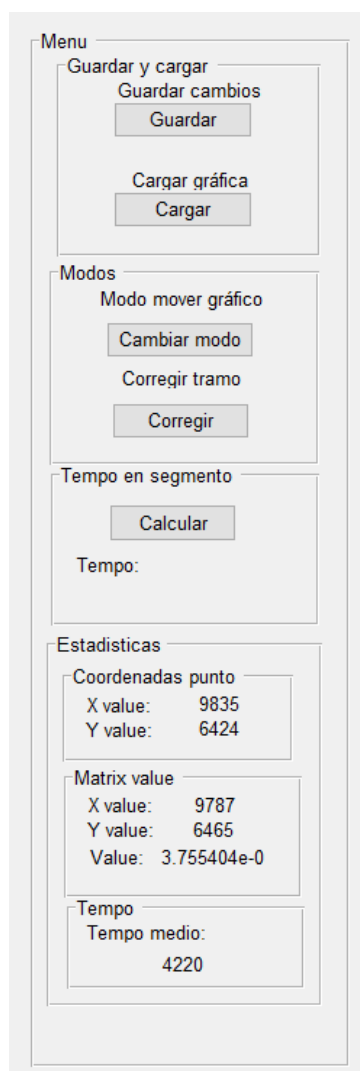


Figura 12.4.- Panel menú.

Y ahora, explicando panel a panel:

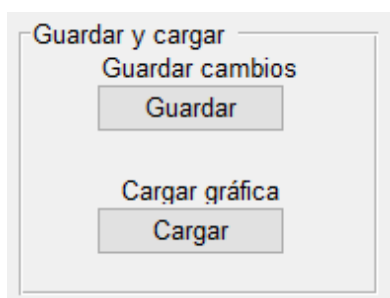


Figura 12.5.- Panel guardar y cargar.

Esta panel es el que permite al usuario acceder a los ficheros *.mat* con los datos del alineamiento automático, cargando los datos en la aplicación y haciendo que se muestre en los ejes contenidos en el panel de Gráfica (del que se hablará tras el panel Menú) los datos almacenados en ese fichero *.mat*. También se dispone del botón para guardar los cambios realizados en un nuevo fichero.

Debajo de este panel, se tiene el de modos, en el que se tiene el botón para cambiar entre los dos principales modos de la aplicación, el modo mover gráfico, que permite mover los puntos del camino obtenido del alineamiento automático y el modo corregir, que permite al usuario poner dos puntos para seleccionar un segmento del camino e interpolarlo para que pase a ser una recta.

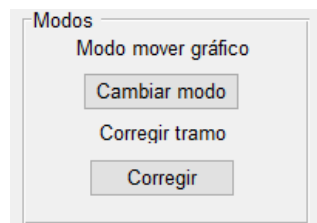


Figura 12.6.- Panel modos.

El siguiente panel, permite al usuario una vez selecciona un segmento del camino, calcular el tempo medio de ese tramo.

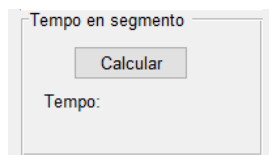


Figura 12.7.- Panel tempo en segmento.

Finalmente, el último panel contenido en Menú es el panel de estadísticas, que muestra al usuario una serie de información de los datos mostrados en el gráfico cuando interactúa con ellos.



Figura 12.8.- Panel estadísticas.

Se muestra en primer lugar las coordenadas X e Y de los puntos del camino que el usuario seleccione. El siguiente panel, muestra los valores de un punto de la matriz de coste, cuando está el modo evaluar puntos activado y el usuario hace click en un punto que le interese. El último panel, muestra simplemente el tempo medio del audio tocado, es decir, la velocidad media a la que se toca.

3.1.2.- Panel Gráfica

Este panel contiene simplemente el objeto de MATLAB *Axes* sobre el que se visualizan los datos de alineamiento automático. El usuario puede interactuar con este objeto y según el modo que esté activado en la aplicación, podrá mover puntos del camino, corregir tramos de este, hacer zoom en partes del gráfico, moverse sobre este, evaluar los valores de la matriz, etc.

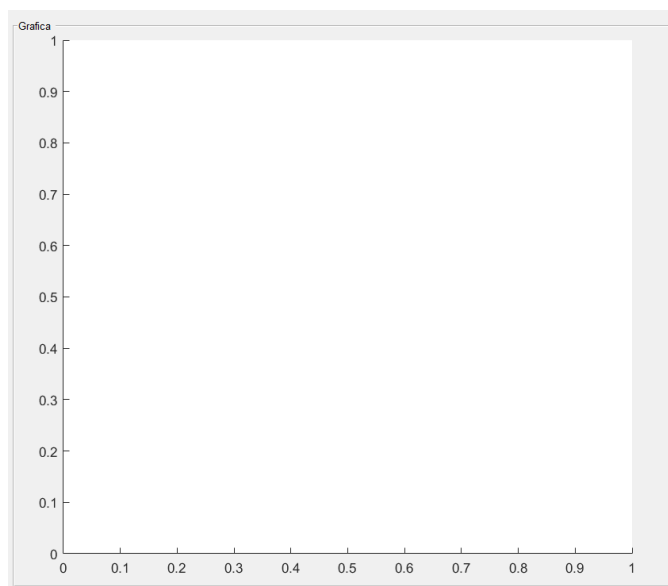


Figura 12.9.- El panel Gráfica antes de cargar datos de alineamiento.

3.1.3.- Panel Herramientas

Este panel, compuesto por tres paneles distintos, ofrece al usuario una serie de herramientas con las que interactuar con el gráfico y con parámetros de la aplicación como el tamaño de la nube de puntos que se mueve en el modo mover gráfico.

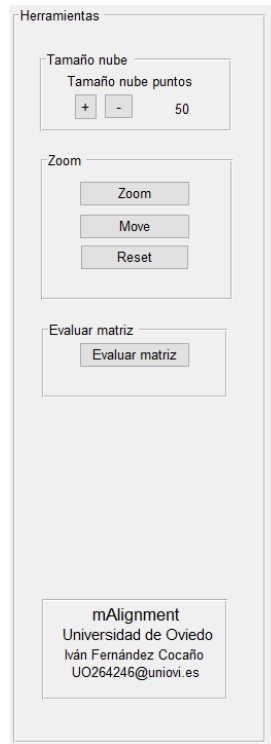


Figura 12.10 Panel Herramientas.

De nuevo se explica cada panel de la interfaz por separado, empezando por el panel Tamaño nube.

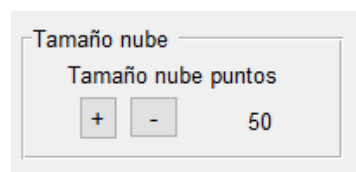


Figura 12.11 Panel tamaño nube.

Este panel permite al usuario modificar el tamaño de la nube de puntos que se mueve en el modo mover gráficos, interactuando con el botón “+” o “-“, según quiera aumentar o reducir este rango respectivamente. El valor actual del tamaño de la nube de puntos se muestra a la derecha de estos botones.

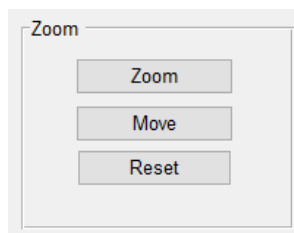


Figura 12.12.- Panel zoom.

Este panel permite al usuario entrar en el modo de navegación de la gráfica, que se compone de los modos zoom y pan. Con el botón zoom, el usuario puede acercarse a una zona que seleccione del gráfico. El botón move permite al usuario activar el modo pan, con el que puede moverse arrastrando el ratón por el gráfico, si este está ampliado a una zona concreta.

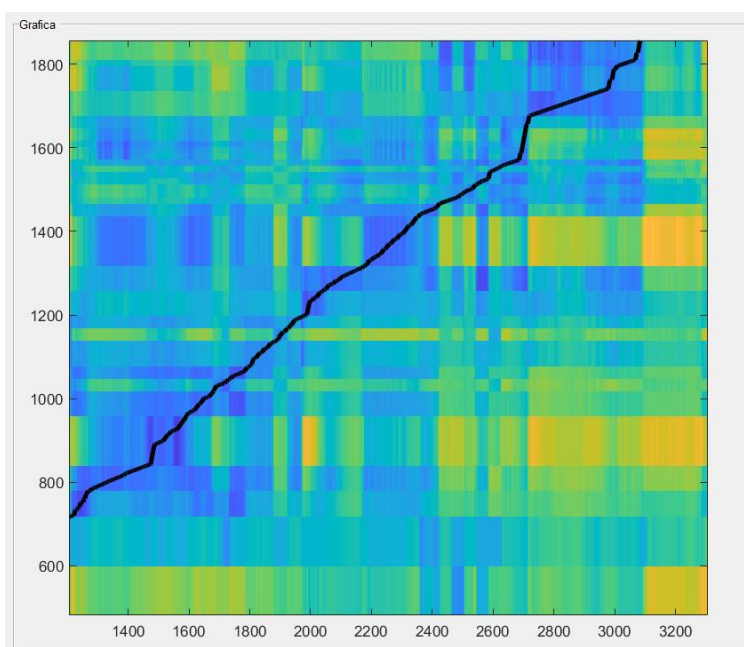


Figura 12.13.- Gráfica aumentada en un punto concreto.

Finalmente tenemos el panel de evaluar matriz:

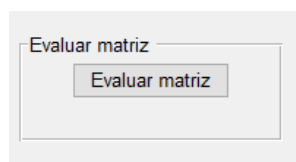


Figura 12.14.- Panel evaluar matriz.

Este panel permite al usuario activar el modo de evaluar matriz, modo en el que cuando hace click en cualquier punto de la matriz, se mostrará en el panel de estadísticas explicado anteriormente las coordenadas de ese punto y el valor de coste de la matriz almacenado en este punto.

Cuando se activa este modo, aparecerá un texto dentro del panel que notificará al usuario de que se encuentra en el modo evaluar matriz:

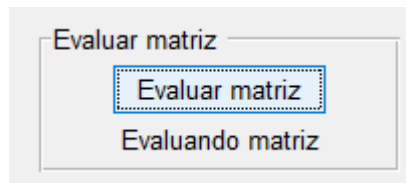


Figura 12.15.- Etiqueta modo evaluar.

De este modo podrá saber cuándo puede evaluar la matriz, lo cual es importante ya que mientras se encuentra en este modo no se pueden realizar a la vez otras funcionalidades como las de mover el gráfico o corregir.

Capítulo 4.- Modos de ejecución

La aplicación mAlignment dispone de varios modos de ejecución que se corresponden con las funcionalidades que el usuario pueda buscar para tratar con los datos de alineamiento automático. En general se pueden contar cuatro modos de ejecución, el de mover gráfico, el de corrección, el de evaluar matriz y los modos de navegación de gráfico, aunque se incluye también en este apartado la posibilidad de ejecutar la aplicación con datos de puntos de anclaje y sin ellos.

Los modos principales de ejecución y las transiciones entre ellos son:

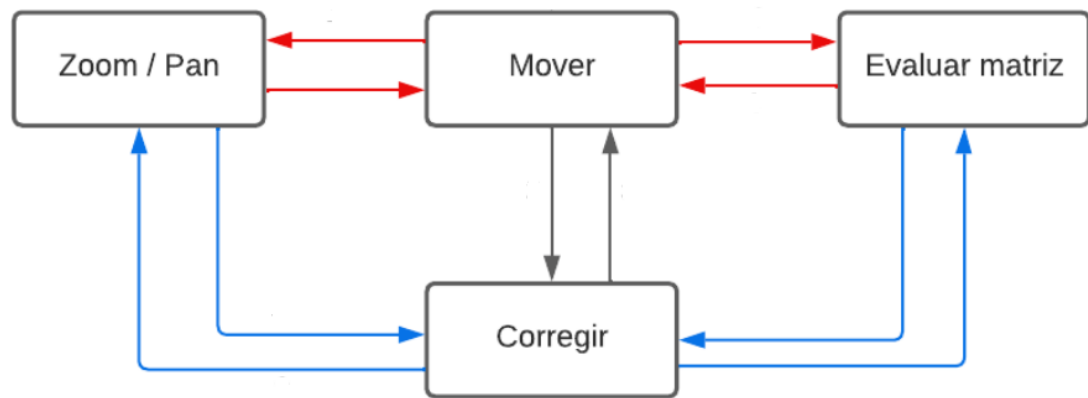


Figura 12.16.- Modos de ejecución.

Los colores representan que si vas a un modo por una flecha de un color, solo puedes tomar de vuelta otra flecha del mismo color.

4.1.- Modo mover gráfico

El modo mover gráfico es el modo de ejecución en el que la aplicación comienza a funcionar una vez se cargan los datos de alineamiento automático y se representan los datos en los ejes del panel de Gráfica. Desde este modo, el usuario puede interactuar con el camino representado en la matriz de costes, moviéndolo en el eje de ordenadas hasta puntos que considere más correctos y observando los valores de X e Y que desee. El usuario puede saber que está en este modo gracias a la interfaz, que lo muestra en una etiqueta del panel Modos.

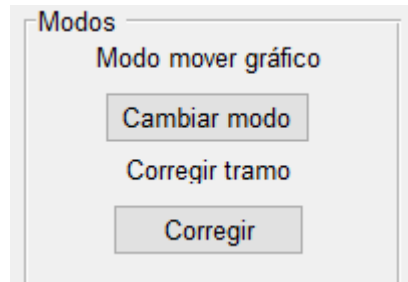


Figura 12.17.- Muestra del modo actual.

Desde este modo se puede acceder a el resto de los modos de ejecución a través de botones, es decir, puede pasarse al modo corregir, a los modos de navegación del gráfico (zoom y pan) y el de evaluar matriz.

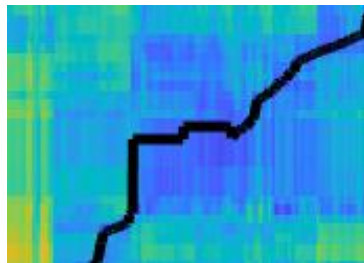


Figura 12.18.- Parte del camino modificada por el usuario.

4.2.- Modo corrección

El modo corrección se trata de un modo en el que el usuario se puede encargar de seleccionar dos puntos del camino, sin importar su distancia o el orden en el que los seleccione para crear un segmento que corregir.

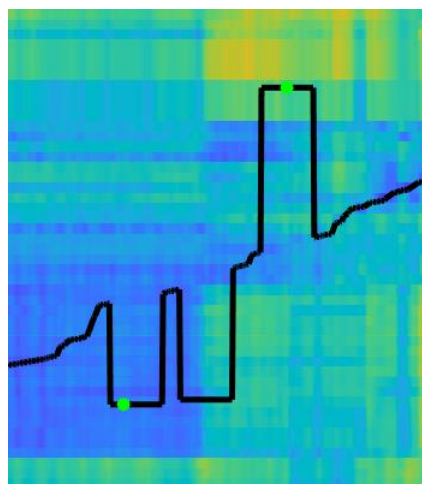


Figura 12.19.-Ejemplo de segmento del camino seleccionado por el usuario.

Una vez tiene ambos puntos elegidos, se hace una interpolación entre ambos puntos de línea recta, creando así una recta entre ambos extremos del segmento. De esta forma, tras corregir el segmento elegido anteriormente se tendría este resultado:

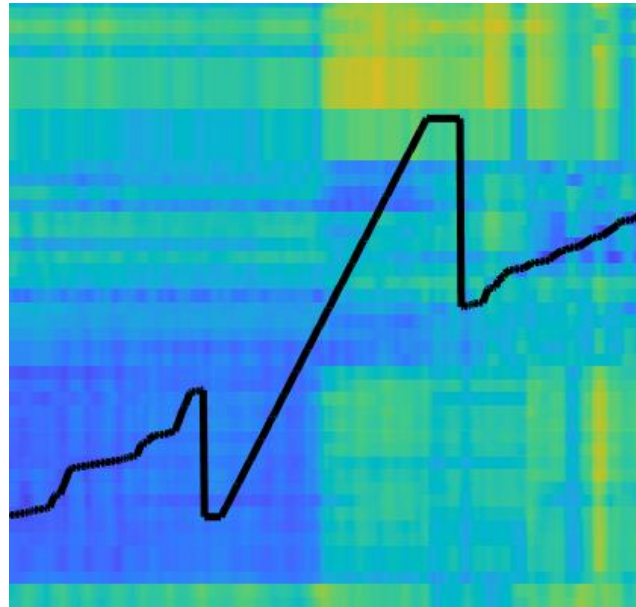


Figura 12.20.- Segmento tras ser corregido.

Los puntos seleccionados van a mantenerse en la aplicación hasta que el usuario decida salirse del modo corregir para pasar al modo mover o hasta que corrija el segmento que eligió, pero se mantendrán si este entra en el modo evaluar matriz o en los modos de navegación de gráfico.

Desde el modo corregir el usuario puede ir al resto de los modos de ejecución a través de los botones correspondientes.

4.3.- Modo navegación gráfico

Se incluyen en este apartado dos modos distintos de ejecución, el de zoom y el de pan, que tienen el mismo objetivo: Permitir al usuario navegar y moverse en los datos representados en el panel de Gráfica como le interese.

El modo zoom permite al usuario seleccionar una zona del gráfico para aumentarlo y verlo más de cerca para facilitar la visualización de los datos en un punto más concreto.

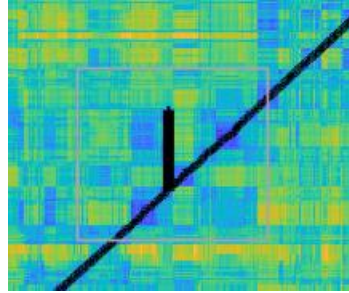


Figura 12.21.- Usuario elige una sección sobre la que hacer zoom.

Una vez se selecciona la zona y se levanta el ratón se aumenta el gráfico hasta ese cuadrado seleccionado.

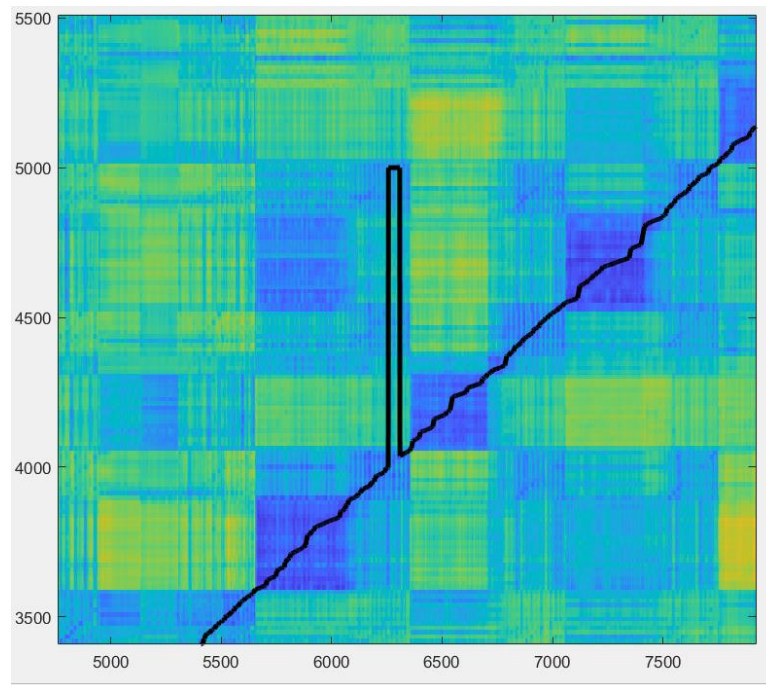


Figura 12.22.- Gráfico ampliado.

Como se puede observar, ahora el gráfico abarca toda esa zona seleccionada de forma ampliada, lo que facilita una visión mucho más clara de la información comparada con la anterior imagen.

El modo pan es el otro modo de navegación del que dispone la aplicación, y que permite al usuario hacer click en un punto del gráfico y arrastrar el ratón para desplazarse por un gráfico que este ampliado. De este modo, una vez está hecho el zoom en un punto concreto, el usuario puede desplazarse por el gráfico a su gusto.

El usuario, en cualquier momento puede decidir restaurar el nivel de zoom al que tenía nada más cargar los datos de la aplicación, apretando en el botón de reset de zoom desde

cualquiera de los modos, ya sea de pan o zoom. Esta acción hace que se salga del modo zoom/pan en el que se esté.

Para acceder a estos modos se debe partir del modo corregir o mover, y una vez se salga de los modos de navegación se regresará desde el mismo modo por el que se accedió. Explicado en un esquema:

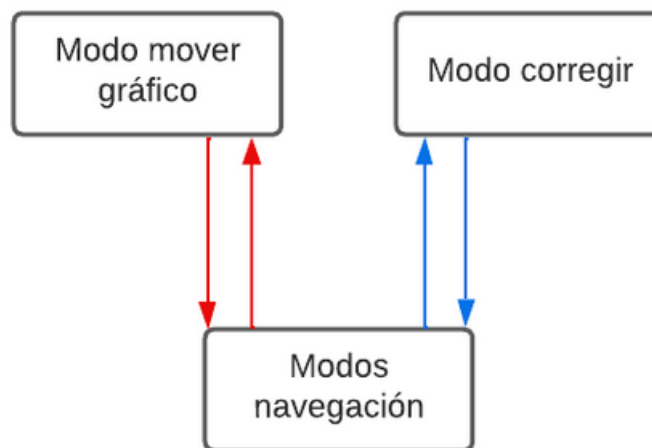


Figura 12.23.- Transiciones de los modos de navegación.

Y dentro de los modos de navegación, el usuario tendrá libertad para moverse entre los modos zoom y pan sin ninguna restricción.

4.4.- Modo evaluar matriz

El modo evaluar matriz es un modo de ejecución que permite al usuario comprobar los datos almacenados en la matriz de costes que proporcionada por el alineamiento automático. Se accede a este modo o bien desde el modo corregir o el modo mover gráfico, de una forma similar a los modos de navegación en la que una vez dejes de evaluar la matriz la aplicación regresa al estado por el que llegó.

Una vez el usuario esta en este estado, podrá evaluar las coordenadas de cualquier punto de la matriz y el valor concreto de coste de ese punto haciendo click en cualquier punto de la matriz.

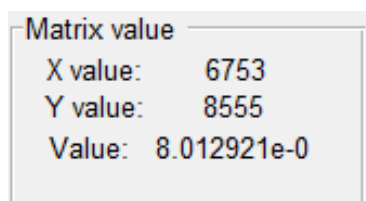


Figura 12.24.- Valores de un punto de la matriz de costes.

Desde este modo solo se puede regresar al modo por el que se entró a él, no existe la posibilidad de entrar en, por ejemplo, los modos de navegación. Además, se restringen otras funcionalidades de la aplicación como los botones de cambio de modo o de aumentar la nube de puntos mientras esta activado el modo evaluar matriz.

4.5.- Puntos de anclaje

La aplicación es capaz de representar dos tipos de alineamiento automáticos, los que disponen de puntos de anclaje y los que no. Los puntos de anclaje son puntos obtenidos durante el alineamiento que el algoritmo considera de vital importancia y que por tanto no se pueden alterar por el usuario.

De este modo, si cargamos en la aplicación datos de alineamiento que contengan puntos de anclaje, estos se representarán en el camino donde corresponden y afectarán al funcionamiento los modos de mover gráfico y corregir, ya que van a ser puntos que ni se podrán mover ni corregir de ninguna forma.

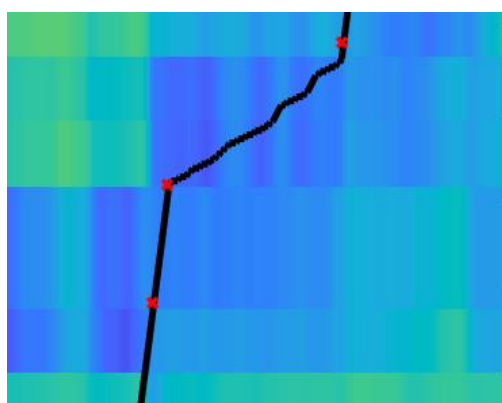


Figura 12.25.- Puntos de anclaje en el camino.

Estos puntos de anclaje afectan también a otras funcionalidades de la aplicación, como las de guardar y cargar los datos, que deben tener en cuenta la posibilidad de que exista esta información en el alineamiento automático para procesarla correctamente.

También se pueden cargar datos de alineamiento sin puntos de anclaje, de esta forma la aplicación funcionará sin las modificaciones en las funcionalidades explicadas.

Capítulo 5.- Tutorial

Este capítulo se incluye para dar al usuario una guía que pueda usar como referencia para utilizar la aplicación y tener una visión clara de cómo acceder y usar todas sus funcionalidades.

Se parte de que la aplicación ya está instalada y se puede ejecutar (véase el Capítulo 2). Los primeros pasos que se deben seguir para usar el software son:

5.1.- Primeros pasos

Antes de nada, se deben seguir una serie de pasos para comenzar la ejecución de la aplicación:

1. Encender el PC.
2. Abrir MATLAB.
3. Ejecutar el script `mAlignment.m`, que lanzará la interfaz de usuario con la que ya se podrá interactuar.
4. Pulsar el botón de cargar o en el icono de cargar para poder abrir los datos de alineamiento automático que se quieran modificar.

5.2.- Ejemplo de obtención de datos

Como ejemplo de obtención de datos de un alineamiento usaremos el algoritmo de alineamiento automático de MATLAB para poder obtener esos datos en un fichero `.mat` y después abrirlos con la aplicación.

Para esto bastará con entrar en el código del alineamiento y colocar un punto de detención durante la ejecución antes de que el código del alineamiento automático lance una imagen dibujando los datos:

```
DTWoffline.m
26 for ss_iter=1:nstates,
27     current_state = states_seq(ss_iter);
28     cstate_t_ini = states_time(1,ss_iter);
29     cstate_t_fin = states_time(2,ss_iter);
30     cstate_miditramas = cstate_t_fin - cstate_t_ini + 1;
31     matrizTmxTr(cstate_t_ini:cstate_t_fin,:)=repmat(distorsionXstate(current_state,:),cstate_miditramas,1);
32 end;
33
34 matrizTmxTr = matrizTmxTr ./ repmat(sqrt(sum(matrizTmxTr.^2,1)),m_ntramas,1);
35 matrizTmxTr(~isfinite(matrizTmxTr)) = 0;
36
37 alpha=abs(log(1-0.99)/0.4);
38 matrizTmxTr = 1-exp(-alpha*abs(matrizTmxTr));
39
40 [p,q] = dpfast_var_offline_min(matrizTmxTr(:,1:end-1),0);
41
42 if draw,
43     figure;imagesc(matrizTmxTr);axis xy;hold on;plot(q,p,'k','LineWidth',3);drawnow;
44 end;
45 return;
46
```

Figura 12.26.- Ejemplo de punto de detención del código del alineamiento automático.

De esta forma se puede ver que el workspace de MATLAB contine las dos variables que se necesitan para un alineamiento sin puntos de anclaje.

Workspace - DTWoffline	
Name ^	Value
alpha	11.5129
cstate_miditra...	261
cstate_t_fin	10419
cstate_t_ini	10159
current_state	301
distorsionXsta...	301x16794 double
draw	1
m_ntramas	10419
matrizTmxTr	10419x16794 do...
nstates	395
p	1x16793 double
q	1x16793 double
r_ntramas	16794
ss_iter	395
states_seq	1x395 double
states_time	2x395 double

Figura 12.27.- Workspace en el punto de parada.

Cuando se llegue a ese punto de la ejecución, se puede usar el comando *save* de Matlab para guardar los datos que necesita la aplicación.

```
Estimating gains from real testing data ... Done
Peforming the alignment ... 42 if draw,
fx K>> save('NuevoFichero','p','matrizTmxTr')
```

Figura 12.28.- Guardado de fichero con datos de alineamiento.

De esta forma se guardarían los datos buscados (p y la *matrizTmxTr*) en el fichero *.mat* NuevoFichero. De querer guardar datos de puntos de anclaje bastaría con buscar otro punto en el que estén en el workspace para almacenar también las variables *tr_anclaje* y *tm_anclaje*. Un ejemplo de un punto en el que se podrían acceder a estos datos durante el alineamiento automático es poniendo un punto de detención en el fichero *tempoPathOnline.m* antes de que se dibujen los puntos de anclaje.

Tras un rato de espera, que puede ser un poco largo según el tamaño de la variable *matrizTmxTr*, se guarda el fichero en la carpeta en la que se está y ya se podrá utilizar para cargar en la aplicación.



NuevoFichero.mat (MAT-file)		
	Name	Value
	matrizTmxTr	10419x16794 double
	p	1x16793 double

Figura 12.29.- Ejemplo de fichero sin anclaje obtenido.

5.3.- Carga de ficheros

Antes de nada, para poder usar la aplicación deberemos cargar unos datos de alineamiento automático. Para cargar este fichero podemos optar entre el botón de cargar de la interfaz o dándole al icono del menú, que tiene la misma funcionalidad.



Figura 12.30.- Opciones de carga de ficheros .mat.

Se abrirá una interfaz de carga en la que se seleccionará el fichero creado anteriormente para cargarlo.

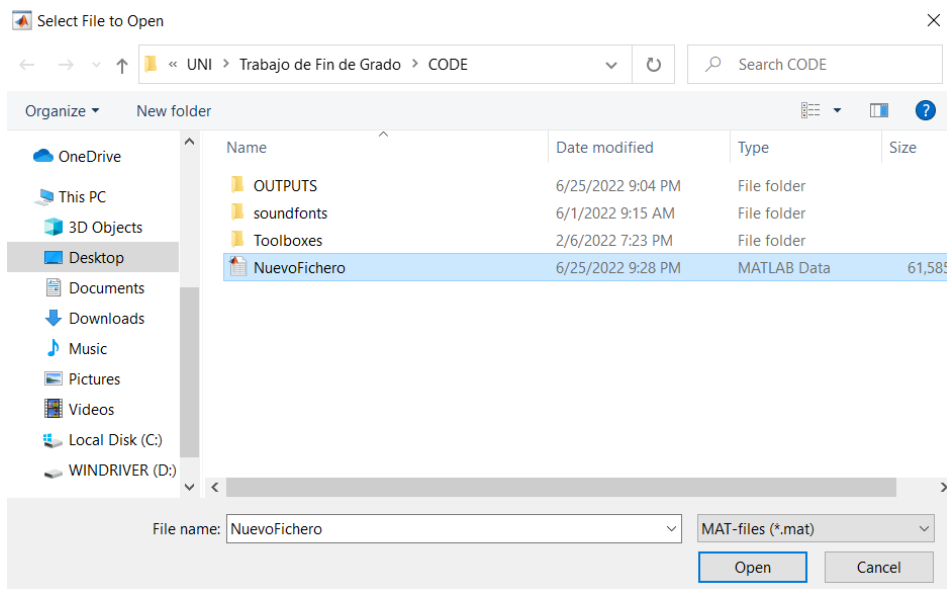


Figura 12.31.- Opciones de carga de ficheros .mat.

Tras unos momentos, la aplicación mostrará los datos en el panel de gráfica:

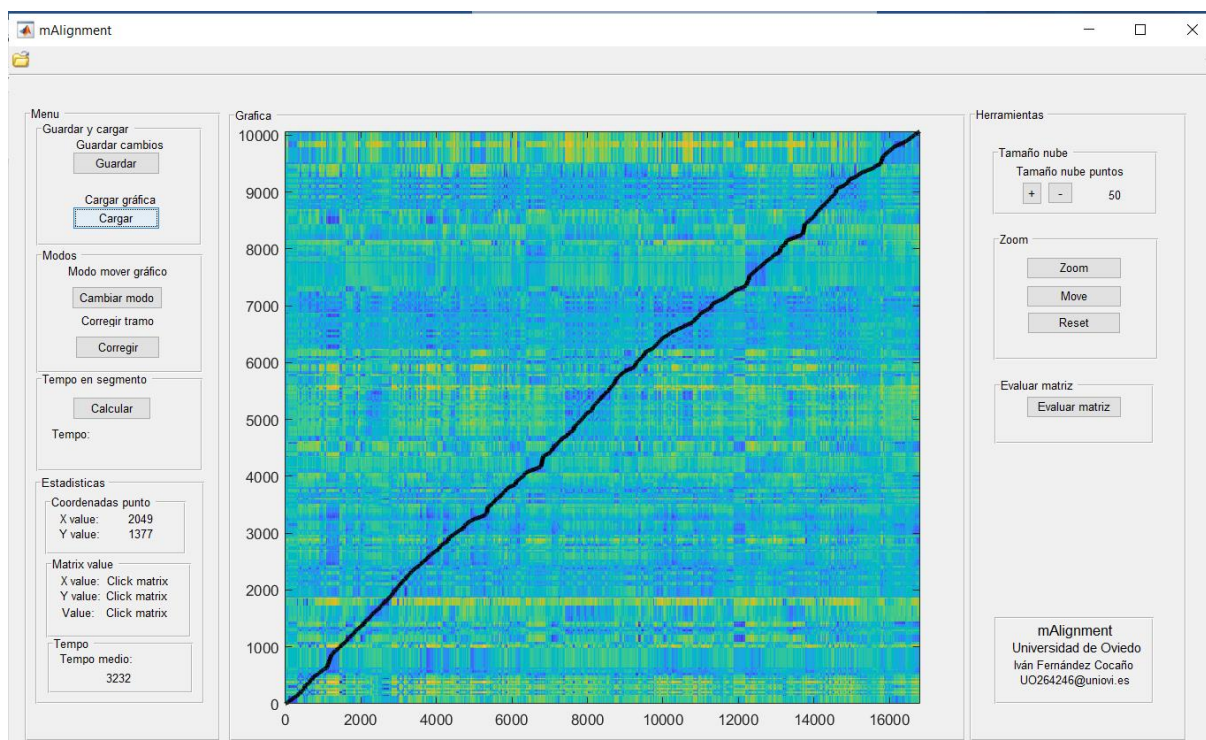


Figura 12.32.- Aplicación tras cargar los datos.

A partir de este punto, se explicarán las funcionalidades de la aplicación y como el usuario puede interactuar con los datos de alineamiento. Se empezará por los modos de navegación de los datos.

5.4.- Navegación del gráfico

Al ser un número de datos tan numeroso, en muchas ocasiones es necesario ampliar ciertas zonas de los datos que se muestran en pantalla para poder corregirlos de forma más adecuada, buscando el resultado esperado.

Para lograr esto, la aplicación dispone de dos modos de navegación para permitir visualizar los datos como el usuario quiera.

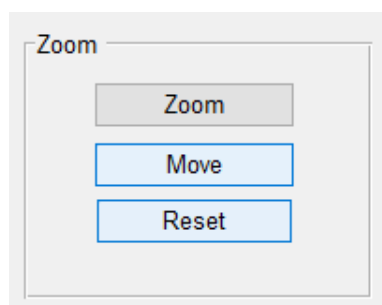


Figura 12.33.- Modos de navegación del gráfico.

El botón zoom activará el zoom de la aplicación, que permite que se seleccione una zona de la aplicación para ampliar. El cursor pasará a tener forma de cruz y haciendo click en un punto se podrá arrastrar un rectángulo o cuadrado para seleccionar una zona sobre la que se ampliará para visualizar más de cerca los datos.

La aplicación también permite una vez está ampliada en un punto, activar el modo pan, que se corresponde con el segundo botón de la imagen, *Move*. Cuando se hace click en este botón el cursor pasa a tener forma de mano y el usuario podrá hacer un click en un punto del gráfico y después arrastrar el cursor para desplazarse por el gráfico y así mover la zona ampliada.

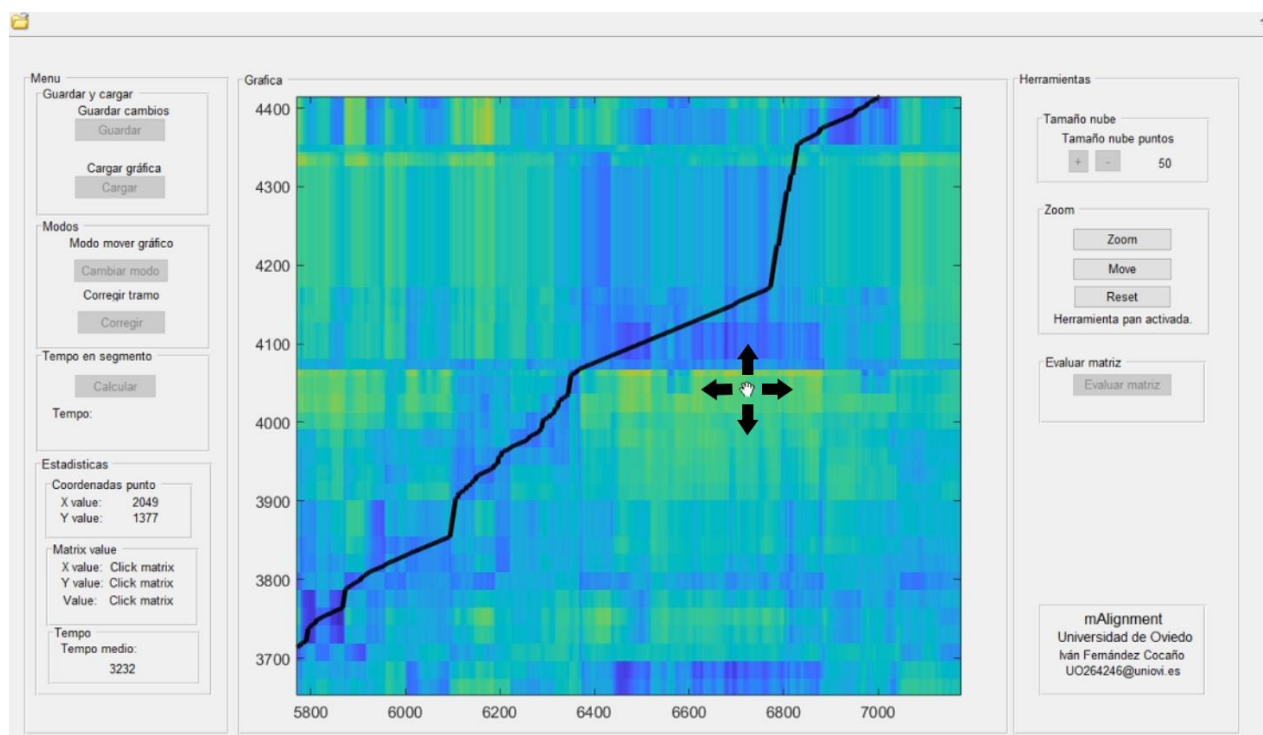


Figura 12.34.- Modo pan en la aplicación.

Una vez el usuario encuentra una zona del gráfico que quiere corregir, podrá regresar al modo corregir o mover gráfico para realizar los cambios buscados en los datos de alineamiento.

5.5.- Mover el gráfico

El modo de mover gráfico sirve para que el usuario pueda corregir zonas las partes del camino a las zonas que él vea como mejores. Para mover los puntos deseados basta con hacer click en uno de los puntos que quieres mover y arrastrar el ratón verticalmente hasta la zona deseada.

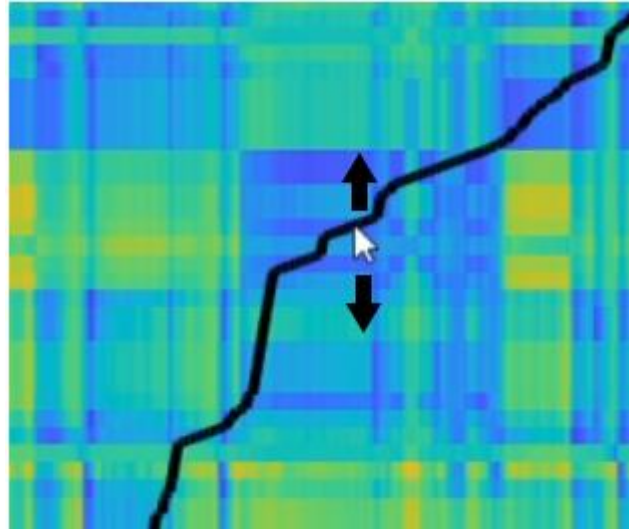


Figura 12.35.- Rango de movimientos en modo mover gráfico.

El tamaño de camino que se mueve en este modo depende del tamaño de la nube de puntos elegido por el usuario en la aplicación. Esta nube de puntos representa un rango de puntos que se mueve junto al que el usuario elige cuando hace click y arrastra, y su tamaño se puede determinar en el panel del tamaño de nube de puntos usando los botones de “+” y “-” para aumentar y disminuir el tamaño respectivamente.

Desde este modo se puede acceder a todos el resto de modos de la aplicación, es decir, los de evaluar matriz, el modo corregir y los modos de navegación de gráfico.

5.6.- Corregir segmentos del camino

El modo corregir permite al usuario seleccionar un segmento del camino y cambiarlo totalmente por una línea recta entre ambos puntos límites del segmento escogidos. Si el usuario quiere entrar a este modo, basta con usar el botón de cambiar de modo cuando se está en el modo de mover gráfico para pasar al modo de corregir.

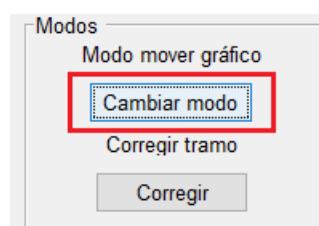


Figura 12.36.- Cambiar al modo corregir.

Para dibujar los puntos que delimitan el segmento, el usuario una vez esté en el modo corregir debe hacer click en el punto que desee y después darle a la tecla X. En ese momento, aparecerá un círculo verde en el punto seleccionado, creando el primer extremo del segmento.

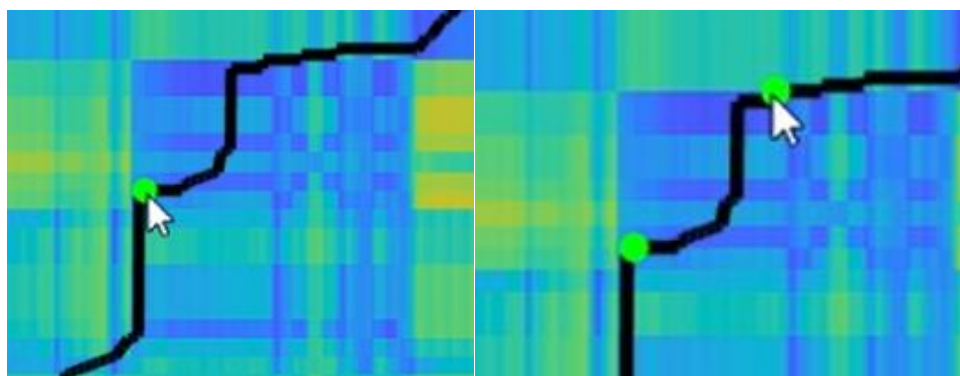


Figura 12.37- Creación del segmento a corregir.

Una vez se tiene el segmento creado, basta con hacer click en el botón de corregir para que se haga la interpolación entre ambos puntos y se obtenga así el segmento corregido:

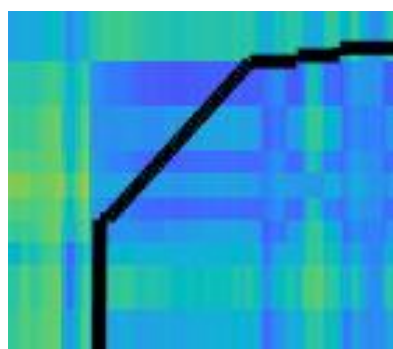


Figura 12.38.- Segmento corregido

- **Advertencia:**

Una vez se corrige el tramo no hay marcha atrás, por lo que hay que tener cuidado con esta funcionalidad. En caso de que se coloquen mal los puntos y se quiera cambiar, basta con cambiar al modo de mover gráfico y regresar al de corregir para que desaparezcan los puntos antes establecidos.

Desde este modo el usuario puede navegar a cualquiera de los otros modos, el de mover gráfico, los de navegación de gráfico o el de evaluar matriz sin ninguna restricción.

5.7.- Evaluar puntos de la matriz de costes

El modo evaluar matriz esta pensando para que el usuario pueda comprobar los valores almacenados en la matriz de coste que se representa en la aplicación. Se puede acceder a este modo desde los modos de corregir o mover gráfico, usando el botón de evaluar:

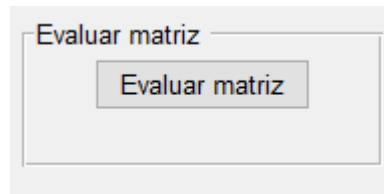


Figura 12.39.- Botón para entrar en el modo evaluar matriz.

Una vez se está en este modo la aplicación pasa a limitar ciertas funcionalidades como las de mover / corregir gráficos para que el usuario pueda hacer click en cualquier punto de la matriz de coste de fondo que le interese. Desde el modo de evaluar matriz solo se puede regresar al modo corregir o mover gráfico (aquel por el que primero se entró).

Los valores de coordenadas y coste del punto de la matriz seleccionado se mostrarán en el panel de Matrix de la aplicación.

5.8.- Otras funcionalidades

Se incluyen en este apartado otras funcionalidades de la aplicación que no encajaban en el resto de los apartados, pero que deben ser explicadas para el correcto uso de la aplicación.

5.8.1.- Guardar cambios

Después de realizar cambios en los datos de alineamiento automáticos, el usuario puede querer guardar su progreso para utilizarlo en otras aplicaciones o continuar modificarlo más adelante. Para guardar estos cambios, la aplicación permite guardar los datos de alineamiento modificados como un fichero .mat igual que los que se usan de entrada.

Para guardar los cambios, el usuario debe hacer click en el botón de guardar estando en los modos de corregir o mover gráfico. Una vez haga click se abrirá una ventana de guardar:

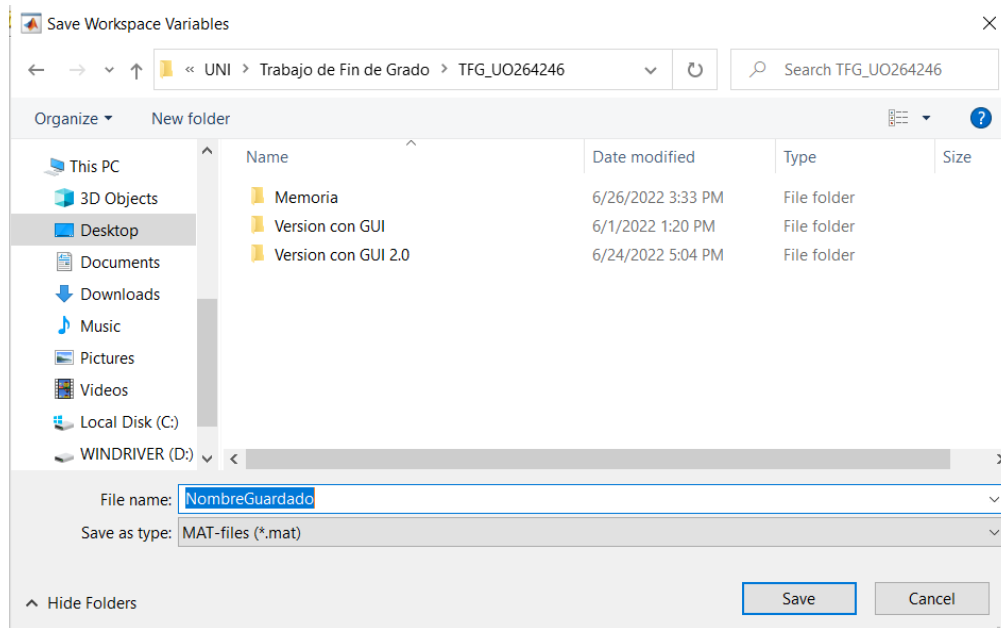


Figura 12.40.- Ventana de guardado de cambios.

5.8.2.- Calcular tiempo en un tramo

Otra funcionalidad disponible para el usuario es la de calcular el tiempo medio en un tramo del camino seleccionado. Para lograr esto, se dibuja un segmento en el modo corregir, que abarque la zona del camino cuyo tiempo medio queramos conocer.

Una vez el segmento esta seleccionado, bastará con darle al botón de calcular tiempo en segmento, que eliminará los puntos seleccionados sin corregir el tramo y mostrará en la interfaz el valor del tiempo medio del segmento.

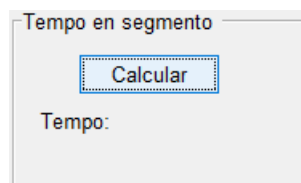


Figura 12.41.- Botón de calcular tiempo en un segmento.

Una vez se le da al botón, el resultado del tiempo medio aparecerá a la derecha de la etiqueta “Tempo”.

Capítulo 6.- Posibles modificaciones

Este apartado final se usará para indicar posibles modificaciones que podría tener la aplicación para añadir nuevas funcionalidades o modificar las que ya se tienen. Se indicará brevemente como podrían lograrse estas modificaciones con código MATLAB para que, si el usuario lo desea, pueda introducir estos cambios.

- **Advertencia:**

Hay que introducir estos cambios con cuidado para que la aplicación mantenga su integridad y correcto funcionamiento. Se recomienda guardar una copia de la aplicación previa a los cambios para poder regresar a ella en caso de que alguno de los cambios realizados genere fallos que impidan el correcto funcionamiento de la aplicación.

6.1.-Creación de puntos de anclaje

Para la creación de puntos de anclaje se podría crear un nuevo botón que permitiese al usuario al darle que se crease un punto de anclaje en el último punto del camino seleccionado.

Para poder crear un punto de anclaje habría que actualizar el valor del array que almacena todos los índices en los que hay un punto de anclaje, por lo que lo indicado sería generar un código similar a este:

```
function nuevoAnclaje()
handles=checkHandles(); %Obtenemos el objeto handles.
%Obtiene de la GUI el indice X del ultimo punto clickado.
ind=str2num(handles.output.txValue.String);
%Ponemos a 1 este valor para que se reconozca como punto de
%anclaje.
handles.anclaje(n)=1;
plot(n,handles.ydata(n),'rx','LineWidth',2); %Lo dibujamos.
checkHandles(handles); %Guardamos los cambios
end
```

Habría que llamar a esta función desde el callback del botón de añadir punto de anclaje, si se quiere programar de forma similar a la indicada.

6.2.- Variación de corrección de segmentos

Actualmente, la corrección de segmentos se realiza con una interpolación con una ecuación de la recta, de esta forma (El código del método está resumido para ver el punto que necesita ser cambiado):

```
function[n_ydata] =corregirTramo(handles)
y1=handles.ydata(handles.indiceOrig);
y2=handles.ydata(handles.indiceFin);
x1=handles.xdata(handles.indiceOrig);
x2=handles.xdata(handles.indiceFin);

if ~isAnclaje(x1:x2, handles.anclaje)
    texto=findobj('Tag', 'tAnclaje');
    set(texto, 'visible', 'off');
    %La pendiente
    m=(y2-y1)/(x2-x1);
    %Recta entre ambos puntos
handles.ydata(handles.indiceOrig:handles.indiceFin)=round(y1+m
*((x1:x2)-x1));
end
end
```

Se destaca en negrita la parte que cabría cambiar del código. Como se puede ver, se realiza una interpolación basada en la ecuación de la recta, pero cabría cambiar este código por una interpolación distinta que pueda plantear el usuario si así lo desea.