



# Trabajo Fin de Grado

GRADO EN INGENIERÍA INFORMÁTICA

Generador pseudo-aleatorio de trayectorias

AHMED ANTONIO BOUTAOUR SANCHEZ

**Tutor**

Isaac Lera Castro

Escola Politécnica Superior  
Universitat de les Illes Balears  
Palma, 17 de mayo de 2020



# ÍNDICE GENERAL

<b>Índice general</b>	<b>i</b>
<b>Índice de figuras</b>	<b>iii</b>
<b>Acrónimos</b>	<b>vii</b>
<b>1 Abstract</b>	<b>1</b>
<b>2 Introducción</b>	<b>3</b>
<b>3 Conceptos principales</b>	<b>7</b>
3.1 Sistemas de navegación por satélite . . . . .	7
3.1.1 GIS . . . . .	8
3.1.2 Fuentes de datos . . . . .	8
3.2 Estructura de los ficheros GPS eXchange Format (GPX) . . . . .	10
3.3 Open Street Map . . . . .	11
<b>4 Arquitectura de la aplicación</b>	<b>13</b>
4.1 <i>TrackSimulator</i> . . . . .	13
4.2 Funcionalidades de <i>TrackSimulator</i> . . . . .	15
4.3 Requerimientos de <i>TrackSimulator</i> . . . . .	17
4.4 Implementación de TrackSimulation . . . . .	18
4.4.1 Flujo de datos . . . . .	18
4.4.2 Herramientas externas utilizadas . . . . .	19
4.4.3 Docker . . . . .	21
<b>5 Análisis de los datos Sistemas de Posicionamiento Global (GPS)</b>	<b>23</b>
5.1 Preparación del entorno de datos . . . . .	23
5.1.1 Importación de datos GPS a <i>TrackSimulator</i> . . . . .	23
5.1.2 Entidad TrackPoint . . . . .	25
5.1.3 Importación de caminos y carreteras al modelo de datos . . . . .	25
5.2 Map-Matching . . . . .	26
5.2.1 Hidden Markov Model (Hidden Markov Model (HMM)) . . . . .	26
5.2.2 Aproximación de Hidden Markov Model a Map-matching . . . . .	27
5.2.3 Algoritmo de Viterbi . . . . .	27
5.2.4 Ejemplo de aplicación del algoritmo de Viterbi . . . . .	29
5.2.5 Implementación de Map-matching . . . . .	31
5.3 Explotación del dato . . . . .	32

5.3.1	Análisis de distancias . . . . .	32
5.3.2	Demostración de análisis . . . . .	33
<b>6</b>	<b>Simulación de trayectorias</b>	<b>35</b>
6.1	Generación de un camino . . . . .	36
6.2	Generación de los segmentos . . . . .	37
6.3	Generación del punto pseudo-aleatorio . . . . .	38
6.4	Exportación de trayectoria a fichero .GPX . . . . .	38
<b>7</b>	<b>Experimentacion</b>	<b>41</b>
7.1	Experimentación sin importación de datos reales . . . . .	42
7.2	Experimentación con importación de datos reales . . . . .	44
7.2.1	Comparativa de resultados . . . . .	47
7.2.2	Anexo: Muestra de simulaciones con datos introducidos . . . . .	49
7.2.3	Anexo: Muestra de simulaciones sin datos . . . . .	50
<b>8</b>	<b>Guía de instalación y uso</b>	<b>53</b>
8.0.1	Instalación . . . . .	53
8.0.2	Guía de uso . . . . .	55
<b>9</b>	<b>Conclusiones</b>	<b>59</b>
9.1	Futuro trabajo . . . . .	61
9.2	Opinión personal . . . . .	61
	<b>Bibliografía</b>	<b>63</b>

## ÍNDICE DE FIGURAS

3.1 Ejemplo de la aplicacion ArcGis . . . . .	8
3.2 Ejemplo de la aplicacion QGIS . . . . .	8
3.3 Ejemplo de territorio en formato ráster. Fuente: [1]. . . . .	9
3.4 Ejemplo de Ruta alrededor del Puig de l'Ofre, Mallorca. . . . .	10
4.1 Ejemplo de detección real de la trayectoria de un individuo por la Serra de Tramuntana, Illes Balears, Mallorca, Spain . . . . .	13
4.2 Diagrama de funcionamiento de TrackSimulator . . . . .	15
4.3 Diagrama de funcionamiento de TrackSimulator . . . . .	16
4.4 Flujo de datos de TrackSimulator . . . . .	18
4.5 Comparación entre arquitectura Docker y mediante máquinas virtuales. [2]	21
4.6 Infraestructura de la propuesta de TrackSimulator . . . . .	22
5.1 Estructura de un <i>track</i> dentro de un fichero .GPX . . . . .	24
5.2 Ejemplo de estructura LineString . . . . .	27
5.3 Ejemplo teórico de detección map-matching mediante algoritmo de Viterbi	28
5.4 Análisis de la función de $P_{emission}$ para $\sigma$ . . . . .	29
5.5 Análisis de la $P_{transition}$ de transición para $\beta$ . . . . .	29
5.6 Ejemplo de aplicación de algortimo de Viterbi para detección de trayectorias.	30
5.7 Ejemplo de aplicación de algortimo de Viterbi para detección de trayectorias.	30
5.8 Ejemplo de aplicación de algortimo de Viterbi para detección de trayectorias.	31
5.9 Análisis de la distancia punto a proyección. . . . .	34
5.10 Análisis de la distancia punto a punto. . . . .	34
6.1 Generación de camino. . . . .	35
6.2 Generación de puntos. . . . .	35
6.3 Ejemplo de probabilidades dentro de un nodo intermedio entre rutas. . . .	36
6.4 Ejemplo de generación de punto a partir de una distancia $d$ y un ángulo $\alpha$ .	38
6.5 Ejemplo de detección de proyección cercana y generación de punto. . . . .	38
7.1 Mapa de calor del conjunto de rutas reales. . . . .	42
7.2 Muestra de simulación 1 (Experimentación sin datos). . . . .	43
7.3 Muestra de simulación 2 (Experimentación sin datos). . . . .	43
7.4 Muestra de simulación 3 (Experimentación sin datos). . . . .	43
7.5 Mapa de calor del conjunto de rutas simuladas en experimentación sin datos.	44
7.6 Muestra de análisis de simulación: Distancia punto a proyección en experimentación sin datos. . . . .	45

7.7	Muestra de análisis de simulación: Distancia punto a punto en experimentación sin datos. . . . .	45
7.8	Muestra de simulación 1 Experimentación con datos). . . . .	46
7.9	Muestra de simulación 2 Experimentación con datos). . . . .	46
7.10	Muestra de simulación 3 (Experimentación con datos). . . . .	46
7.11	Mapa de calor del conjunto de rutas simuladas en experimentación con datos.	46
7.12	Muestra de análisis de simulación: Distancia punto a proyección en experimentación con datos. . . . .	47
7.13	Muestra de análisis de simulación: Distancia punto a punto en experimentación con datos. . . . .	47
7.14	Comparativa análisis de la distancia punto a proyección en simulación con datos. . . . .	48
7.15	Comparativa análisis de la distancia punto a punto en simulación con datos.	48
7.16	Simulación con carga de datos inicial. Ejemplo 1. . . . .	49
7.17	Simulación con carga de datos inicial. Ejemplo 2. . . . .	49
7.18	Simulación con carga de datos inicial. Ejemplo 3. . . . .	50
7.19	Simulación con carga de datos inicial. Ejemplo 4. . . . .	50
7.20	Simulación sin datos iniciales. Ejemplo 1. . . . .	51
7.21	Simulación sin datos iniciales. Ejemplo 2. . . . .	51
7.22	Simulación sin datos iniciales. Ejemplo 3. . . . .	52
7.23	Simulación sin datos iniciales . Ejemplo 4. . . . .	52
8.1	Jerarquía en el directorio <i>/track-simulator/data</i> . . . . .	54
8.2	Directorio <i>statistics</i> tras el análisis de un conjunto de trayectorias. . . . .	56
8.3	Directorio <i>simulation</i> tras la simulación de un conjunto de trayectorias. . .	56
9.1	Muestra de resultado del proceso de map-matching. . . . .	60

## ÍNDICE DE ALGORITMOS

2.1 Ejecución simulación . . . . .	4
3.1 Ejemplo fichero GPX. Fuente: [3]. . . . .	10
5.1 Importación de una zona mediante la librería osmnx. . . . .	25
5.2 Implementación interactor get_map_matching_impl . . . . .	31
5.3 Declaración interfaz hidden_markov_model . . . . .	31
5.4 Método add_next_point_distance . . . . .	32
5.5 Método add_point_projection_distance . . . . .	33
6.1 Método create_path . . . . .	36
6.2 Método simulate_segment . . . . .	37
6.3 Método create_gpx_track . . . . .	39
8.1 Ejecución análisis . . . . .	55
8.2 Ejecución análisis . . . . .	55
8.3 Ejecución simulación . . . . .	56



## ACRÓNIMOS

**GNSS** Sistema Global de Navegación por Satélite

**GPS** Sistemas de Posicionamiento Global

**GIS** Sistemas de Información Geospatial

**XML** Extensive Markup Language

**GML** Geography Markup Language

**KML** Keyhole Markup Language

**GPX** GPS eXchange Format

**OSM** OpenStreetMap

**HMM** Hidden Markov Model

**BSP** Binary Space Partition

**SQL-DB** Structured Data Base

**NO-SQL-DB** Non-structured Data Base

**CLI** Command-line-interface

**UTC** Coordinated Universal Time

**UUID** Universally unique identifier

**PNG** Portable Network Graphics

**API** Application Programming Interface



CAPÍTULO



## ABSTRACT

El auge de la explotación de los datos para obtener ventajas estratégicas de negocio abarca gran parte de los ámbitos de la sociedad. El aumento de dispositivos capaces de captar la ubicación de un individuo mediante Sistemas de Información Geospatial (GIS) sumado al aumento de dispositivos inteligentes que los usuarios portan durante el día hace que la información de posicionamiento geográfico sea valiosa en un gran número de sectores.

**TrackSimulator** es el Command-line-interface (CLI) desarrollado en Python propuesto en el documento que permite la generación de trayectorias de forma pseudo-aleatoria. La aplicación consta de un módulo de análisis de ficheros en formato GPX, de los que se obtiene información medible y cuantitativa que es usada para generar trayectorias. Esta herramienta almacena en una base de datos el análisis del conjunto de trayectorias reales que se desee. Del proceso de análisis se obtiene una representación gráfica del comportamiento de las trayectorias. De la simulación se obtiene un fichero en GPX y una representación gráfica por cada una de las trayectorias generadas.

Este documento explica en detalle la solución propuesta. Se muestra la arquitectura de la herramienta, el flujo de datos, así como las heurísticas de análisis y los algoritmos empleados para el análisis y la simulación de trayectorias, demostrando mediante experimentación los resultados que pueden obtenerse.



CAPÍTULO

# 2

## INTRODUCCIÓN

En plena era del dato [4], la necesidad por obtener conocimiento a partir de gran cantidad de datos generados ha hecho que la cantidad de avances en este ámbito incremente notablemente. En la actualidad tenemos toda una serie de dispositivos: smartphones, smartwatches, pulseras deportivas, chalecos deportivos que están generando información diversa del individuo que lo usa. La posición geográfica es una de las principales, seguida de información temporal, frecuencia cardíaca y un conjunto de información que puede ser tratada para obtener conocimiento sobre la población.

La información del posicionamiento geográfico de los individuos que los GIS proporcionan constantemente hace que sea posible obtener, tratar, analizar y explotar dicha información. La explotación de estos datos pueden llevarse a diferentes aplicaciones. Puede ser usada para determinar la posición de un individuo, para sistemas de navegación en los que un individuo se desplaza desde un punto origen a un punto destino , la monitorización del movimiento de individuos, entre otras [5].

Una de las grandes utilidades de la explotación de estos datos consiste en el análisis de redes complejas de caminos y carreteras de forma que se puede obtener indicadores sobre el volumen de paso de individuos por caminos o carreteras, así como para la planificación de trayectorias. Un ejemplo claro de esta utilidad es la explotación del dato para el conocimiento en el transporte. Tanto en la forma en la que los individuos se trasladan al análisis de las carreras de un vehículo de transporte, tanto público como privado, como Uber o Cabify [6].

Es en este ámbito en el que aparece la necesidad de una herramienta que sea capaz de analizar información que obtienen los diferentes GIS, almacenarla y transformar este conocimiento para poder generar trayectorias similares a la realidad dentro de un territorio geográfico.

TrackSimulator es un CLI para la generación de trayectorias pseudo-aleatorias propuesta en este documento. TrackSimulator cumple una serie de requerimientos que están descritos detalladamente en el capítulo 4. Estos requerimientos son:

1. Importación de datos en formato GPX.

## 2. INTRODUCCIÓN

---

2. Análisis de trayectorias, obteniendo información medible y cuantificable.
3. Almacenamiento de las rutas reales y de los resultados del análisis en base de datos
4. Generación de fichero de gráficas resultantes del análisis
5. Lectura de la base de datos para el acceso a la información
6. Simulación de puntos GPS
7. Generación del fichero GPX correspondiente a la simulación de la trayectoria
8. Generación de fichero visualización de trayectoria

Esta propuesta está desarrollada en el lenguaje Python y utiliza diferentes librerías externas, entre las que se destaca **osmnx** para el el análisis a partir del uso de redes de caminos y carreteras [7], **pandas** para el tratamiento de datos [8] y **gpxpy** para la manipulación de ficheros GPX [9]. El resto de librerías externas utilizadas se describen en detalle en el capítulo 4

La infraestructura de la aplicación está formada a partir de contenedores de Docker, de esta forma, la aplicación puede ser ejecutada en cualquier máquina que tenga Docker instalado, eliminándose cualquier tipo de problema que pueda surgir entre dependencias del aplicativo y la máquina en la que se ejecuta. Encontramos entonces dos contenedores: el primero contiene una conexión con una base de datos MongoDB en la que almacena tanto las trayectorias analizadas como los resultados de los diferentes análisis. El segundo contenedor contiene el aplicativo *TrackSimulator*. Este contenedor contiene todo el código, dependencias y configuración necesario para que el usuario pueda analizar un conjunto de trayectorias almacenadas en ficheros GPX. Este análisis lo hace a partir de una técnica llamada *map-matching* consistente en unir los datos con la red lógica de caminos y carreteras.

Despues de la etapa de análisis, el usuario puede realizar una generación de trayectorias. Estas son generadas de forma pseudo-aleatoria a partir de la información analizada en la anterior etapa. Cada trayectoria generada es única en si misma y replica el comportamiento de los individuos analizados.

Finalmente como productos resultantes de esta herramienta encontramos ficheros con la información resultante del análisis del conjunto de trayectorias, así como ficheros GPX y representaciones gráficas de las trayectorias simuladas.

El uso de esta herramienta se hace a partir de la linea de comandos. Se usa una nomenclatura de facil entendimiento que sigue el formato que se ve en el comando 2.1

```
1 ts -c li COMAND --PARAMETERS
```

### Algoritmo 2.1: Ejecución simulación

Tanto la instalación como los comandos soportados por el aplicativo están descritos en el capítulo 8

El objetivo de este documento es explicar al lector la propuesta tanto a nivel teórico como a nivel de implementación de código, así como guiar al usuario a poder instalar el aplicativo y probarlo. Este documento se compone de 9 capítulos donde aparecen 3 diferentes bloques: el primer bloque (capítulo 3 y 4 sería introductorio al problema y la

---

arquitectura de la herramienta. Del capítulo 5 al 7 se explican los diferentes modulos que componen el aplicativo y el flujo de los datos, explicando detalladamente el análisis de los datos y el algoritmo de simulación de las trayectorias, cerrando el bloque con una muestra de los resultados la experimentación del aplicativo. Finalmente el capítulo 8 explica al lector como instalar y usar la herramienta. El capítulo 9 cierra el documento con las opiniones final.

Los capítulos son los siguientes:

- C. 1. **Conceptos principales.** Capítulo introductorio al lector. Aparecen los conceptos básicos y las herramientas conocidas dentro del marco del tratamiento de datos GPS.
- C. 2. **Arquitectura de la aplicación.** Descripción de las funcionalidades que componen la propuesta. Flujo de datos de la herramienta y requerimientos básicos que cumple. Se muestra al lector la estructura de TrackSimulator y el problema que resuelve.
- C. 3. **Análisis de los datos GPS.** Capítulo donde se explica detalladamente como se analiza el dato. Desde la preparación del entorno hasta la explotación del dato pasando detalladamente por el proceso de definición de heurísticas y parámetros.
- C. 4. **Simulación de trayectorias.** Explicación del proceso de generación de trayectorias pseudo-aleatorias. De define la forma en la que se generan individualmente los puntos GPS. Se muestra la implementación técnica destacable.
- C. 5. **Experimentación.** Muestra de resultados del uso de la herramienta. Se realizan dos experimentaciones: con introducción de datos analizados y sin introducción de datos. Se muestran gráficas comparativas y se comentan en detalle los resultados.
- C. 6. **Guía de instalación y uso.** Conjunto de pasos a seguir para instalar el aplicativo en una máquina. En ese capítulo se explican instrucciones disponibles y los parámetros necesarios para su ejecución, completando con ejemplos.
- C. 7. **Conclusión.** Capítulo de cierre del documento y de la propuesta. Se describe el futuro del aplicativo, así como la opinión personal del desarrollo del proyecto. Se concluye la propuesta.



CAPÍTULO

# 3

## CONCEPTOS PRINCIPALES

En este capítulo se realiza una introducción al lector de los conceptos básicos y las tecnologías necesarias dentro del marco de la propuesta. Se realiza a los conceptos de sistemas de navegación por satélite, qué tipos principales existen, qué tipos de formatos de datos que encontramos y las herramientas principales a tener en cuenta para el tratamiento de datos geoposicionales.

### 3.1 Sistemas de navegación por satélite

Los Sistema Global de Navegación por Satélite (GNSS) son mecanismos de detección de coordenadas geográficas formados por una constelación de satélites en órbita utilizados para situar elementos en la superficie terrestre. Estos sistemas permiten detectar las coordenadas de posición en un tiempo determinado con relativa exactitud en cualquier punto del planeta, las 24 horas del día y en cualquier situación meteorológica [10].

Destacan entre muchos sistemas operativos en la actualidad los sistemas GALILEO, GLONASS y GPS, de origen Europeo, Ruso y Americano respectivamente, siendo este último el que explicaremos con más detalle debido a su mayor uso.

El Sistema de Posicionamiento Global, también llamado GPS por sus siglas en inglés lo forma un mínimo de 24 satélites en órbita que proporcionan información de posicionamiento y tiempo accesible para cualquier usuario. El funcionamiento de estos sistemas se basan el cálculo de la distancia de un punto de la tierra a tres satélites aplicando conocimientos de "Position resection" [11]. Esta información tiene una precisión de 100 y 156 metros para la componente horizontal y vertical, respectivamente al 95 % de probabilidad [12].

Es el uso de este sistema el que permite detectar, analizar y medir la situación geográfica de un individuo, y esta unidad de información es el elemento mínimo de una trayectoria. En los capítulos posteriores se describen tanto las herramientas de manipulación de estos datos como los formatos de almacenamiento.

### 3. CONCEPTOS PRINCIPALES

#### 3.1.1 GIS

Los Sistemas de Información Geográfica, GIS, son aplicaciones que permiten almacenar, manipular y presentar la información geográfica [13]. Estos sistemas son usados en diferentes campos de estudio para el análisis y toma de decisiones. Estas herramientas permiten tratar datos relacionados con la posición de un individuo, datos de la vegetación y entorno urbanístico, permitiendo centralizar, integrar y analizar la información geográfica. Entre el gran número de aplicaciones GIS destacan ArcGIS y QGIS, siendo la primera software no libre publicado por la empresa Esri y la segunda software libre por la Open Source Geospatial Foundation.

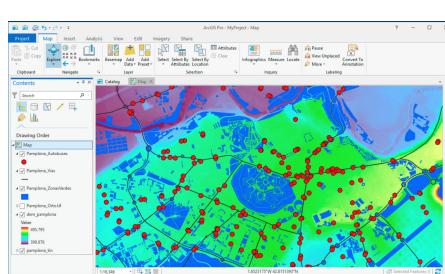


Figura 3.1: Ejemplo de la aplicación Arc-Gis

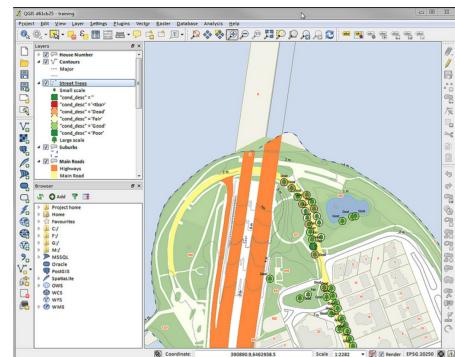


Figura 3.2: Ejemplo de la aplicación QGIS

#### 3.1.2 Fuentes de datos

Las herramientas GIS descritas en el apartado 3.1.1 necesitan que los datos obtenidos por los sistemas GPS sigan un formato compatible. La información geoposicional es creada y manipulada y almacenada en múltiples formatos. Estos se dividen en dos bloques: los formatos ráster y los vectoriales.

##### Formato ráster

En el formato ráster la representación de los datos se hace a partir de una malla cuadrículada. La geografía de un espacio queda descrita en una matriz en la que cada cuadrícula almacena la información como altitud o superficie. La gran ventaja que presenta respecto al formato vectorial es que la estructura de datos en forma de matriz es simple, no obstante consume más recursos de memoria y la resolución de la imagen en los límites entre los píxeles es poco definida si el número de píxeles es insuficiente. Entre los formatos ráster más populares están Esri Grid, GeoTIFF, JPEG2000 [14].

##### Formato vectorial

Los formatos vectoriales se basan en la representación mediante puntos, líneas y polígonos. Presenta una arquitectura de datos más compleja que el formato ráster, no obstante es más preciso. De este segundo grupo destacamos los siguientes:

### 3.1. Sistemas de navegación por satélite

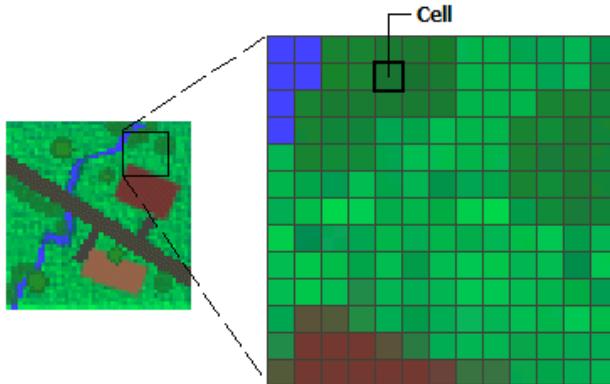


Figura 3.3: Ejemplo de territorio en formato ráster. Fuente: [1].

**Geography Markup Language (GML)** Lenguaje procedente del esquema Extensive Markup Language (XML) que almacena información geográfica [15].

**Keyhole Markup Language (KML)** Lenguaje basado en el esquema XML para la representar información geográfica en un navegador cartográfico como Google Earth o Google Maps [16].

**GPX** Formato basado en el esquema XML para el intercambio de datos GPS. Es especial para la descripción de points, tracks y routes. La principal ventaja de este formato es la capacidad de intercambio de datos entre diferentes programas en diferentes entornos (Windows, MacOS, Linux, Palm, PocketPC). Es este el formato seleccionado como entrada de datos GPS del desarrollo de este proyecto [17].

La flexibilidad del formato GPX y la representación independiente al tamaño del espacio geográfico delimitado las ventajas que determinan su uso en la continuación de la propuesta por delante del formato ráster.

Podemos ver en la siguiente figura un ejemplo de una ruta de una ruta de senderismo por la Serra de Tramuntana (Mallorca, Illes Balears, España) en formato vectorial.

### 3. CONCEPTOS PRINCIPALES

---

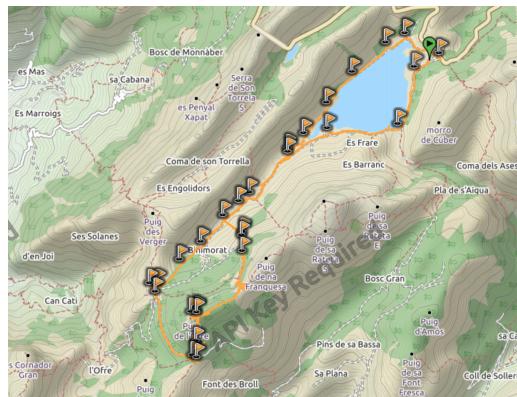


Figura 3.4: Ejemplo de Ruta alrededor del Puig de l'Ofre, Mallorca.

## 3.2 Estructura de los ficheros GPX

El formato GPX al ser basado en XML establece sus propias etiquetas y tipos con la información del fichero y del espacio geográfico que describe. La etiqueta principal gpx (gpxType) es la raíz del fichero XML. Presenta de forma obligatoria la versión y el creador del fichero, así como una cabecera de metadatos, dónde se describe la información del fichero, autor, así como restricciones de copyright de ser necesario. Los elementos esenciales para la descripción de la información geográfica son los siguientes:

**Waypoints (wptType)** Representación de un punto de interés. Consta de una secuencia de elementos opcionales para añadir posición, descripción y precisión, así como los atributos obligatorios de latitud/ longitud. Es la unidad mínima de detección de posición geográfica.

**Route (rteType)** Lista ordenada de Waypoints. Representan una sucesión de puntos hacia un destino. Contiene una secuencia de elementos descriptivos de carácter opcional. No es una representación de un camino hecho, sino que es la unión de segmentos para llegar de un punto inicial a un punto final.

**Tracks (trkType)** Lista ordenada de puntos que escriben un camino realizado. Contiene el mismo tipo de secuencia de elementos descriptivos de carácter opcional. La diferencia con una *Route* es la información que representa. En este caso si que se trata de una detección de un recorrido real.

El formato de una trayectoria en formato GPX parte de un *track* inicial, identificado con el tag <trk>. El interior está formado por una sucesión de al menos un *segment*, tag <trkseg>, que contienen a su vez la unidad mínima, un *point* (<trkpt>). La información que el formato GPX permite analizar en cada uno de los elementos es diversa. La latitud y la longitud de los elementos *point* es información requerida obligatoria para la generación de un fichero GPX válido. En el algoritmo 3.1 se observa un ejemplo de un fichero GPX:

```
1 <gpx creator="GPS Visualizer https://www.gpsvisualizer.com/" version="1.0">
```

### 3.3. Open Street Map

```
2  <wpt lat="45.44283" lon="-121.72904"><ele>1374</ele><name>Vista Ridge Trailhead
3   </name><sym>Trail
4  Head</sym></wpt>
5  <wpt lat="45.41000" lon="-121.71349"><ele>1777</ele><name>Wy' East Basin</name><
6   /wpt>
7  <wpt lat="45.41124" lon="-121.70404"><ele>1823</ele><name>Dollar Lake</name><
8   wpt>
9  <wpt lat="45.39260" lon="-121.69937"><ele>2394</ele><name>Barrett Spur</name><
10  sym>Summit</
11 </wpt>
12 <trk>
13   <name>Barrett Spur 1</name>
14   <extensions>
15     <line xmlns="http://www.topografix.com/GPX/gpx_style/0/2">
16       <color>9900ff</color>
17     </line>
18   </extensions>
19   <trkseg>
20     <trkpt lat="45.4431641" lon="-121.7295456"></trkpt>
21     <trkpt lat="45.4428615" lon="-121.7290800"></trkpt>
22     <trkpt lat="45.4425697" lon="-121.7279085"></trkpt>
23     <trkpt lat="45.4102075" lon="-121.7140608"></trkpt>
24     <trkpt lat="45.4099806" lon="-121.7134527"></trkpt>
25   </trkseg>
26   <trkseg>
27     <trkpt lat="45.4099792" lon="-121.7134610"></trkpt>
28     <trkpt lat="45.4091489" lon="-121.7134937"></trkpt>
29     <trkpt lat="45.4086133" lon="-121.7132504"></trkpt>
30     <trkpt lat="45.4080616" lon="-121.7127670"></trkpt>
31     <trkpt lat="45.3928117" lon="-121.6995661"></trkpt>
32   </trkseg>
33 .
34 .
35 .
36 </trk>
```

Algoritmo 3.1: Ejemplo fichero GPX. Fuente: [3].

## 3.3 Open Street Map

OpenStreetMap (OSM) es un mapa interactivo open-source que permite visualizar información geográfica. La gran ventaja de OSM es la cantidad de herramientas aportadas por la comunidad que permiten importar, analizar, visualizar y editar la información. El formato de fichero GPX es el más utilizado dentro de esta herramienta. La librería Osmx, que se describe posteriormente hace uso de la información de OSM para importar el modelo de caminos y carreteras necesario para el análisis de trayectorias. Es por lo tanto, la plataforma que proporciona el modelo geoespacial lógico en el que se basa la propuesta de este documento.



## ARQUITECTURA DE LA APLICACIÓN

En esta sección se presenta la arquitectura de la aplicación. Se realiza una descripción de las funcionalidades que lo componen, se definen los requerimientos necesarios para su construcción y por último una explicación de la implementación, mostrando las herramientas usadas. El objetivo de este capítulo es mostrar al lector como está estructurada la aplicación tanto a nivel de código desgranando las librerías externas de las que se ha hecho uso como de la solución de la infraestructura, apostando por una solución que facilite al individuo su uso dentro del sistema.

### 4.1 *TrackSimulator*

*TrackSimulator* es un generador pseudo-aleatorio de trayectorias geoposicionales. Se entiende como simulación al proceso de recrear el comportamiento, en este caso el camino de usuarios dentro de un espacio geográfico. Como muestra esencial y discreta del comportamiento de la población se usan registros que han sido obtenidos mediante localización GPS. En la figura 4.1 se muestra un ejemplo de una detección del recorrido de un individuo.

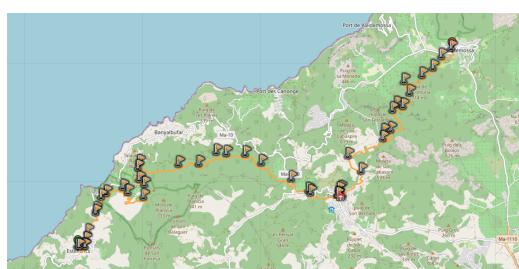


Figura 4.1: Ejemplo de detección real de la trayectoria de un individuo por la Serra de Tramuntana, Illes Balears, Mallorca, Spain

#### **4. ARQUITECTURA DE LA APLICACIÓN**

---

Para realizar un algoritmo que genere trayectorias y tengan un grado de similitud con la realidad se debe realizar un ejercicio previo de análisis y tratamiento de información. Distancias entre las detecciones, desviación entre la posición de una detección del dispositivo GPS y un camino definido son ejemplos de datos que se tienen que tener en cuenta para la obtención de resultados óptimos en cuanto a representación de la realidad.

Tanto la información muestral como la generada, así como todo el conjunto de información geoespacial debe ser almacenada en un tipo de estructura lógica que permita el acceso y manipulación de estos de la forma más eficiente posible.

Teniendo una estructura lógica que permite almacenar la información, el problema determinante será la forma en la que los datos geoposicionales, en este caso puntos GPS pasan a formar parte de este modelo. Recordemos que un punto GPS contiene únicamente información de la posición dentro de la superficie terrestre, no obstante no aporta información de la asignación de este punto a un segmento de un camino, calle, o paraje concreto. La asociación de un punto GPS a un tipo vía es un problema conocido con el nombre de *Map matching* y la propuesta de este documento a su resolución se realizará en detalle en el apartado 5.2.

Con la estructura lógica y la información integrada en ella, el análisis de la información permitirá tomar el conjunto de decisiones que permitan maximizar el grado de exactitud de la simulación para, posteriormente, realizar el algoritmo que permita realizar dicha recreación. La transformación del dato desde una trayectoria real hasta la generación final de trayectorias simuladas queda ilustrada en la figura 4.2.

## 4.2. Funcionalidades de *TrackSimulator*

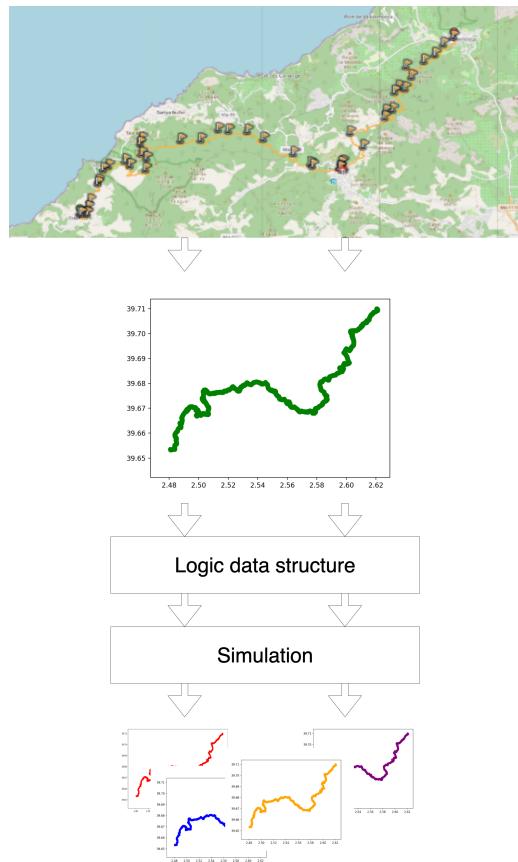


Figura 4.2: Diagrama de funcionamiento de *TrackSimulator*

La aplicación crea una sucesión de puntos que equivalen a un recorrido dentro de un plano geográfico. Se ha determinado que la generación de puntos dentro del aplicativo se realiza de forma pseudo-aleatoria, quiere decir que la producción de los diferentes puntos no sigue ningún patrón o regularidad en sí misma sino que parte del análisis realizado, al que se le añaden componentes aleatorios. La sucesión de puntos estará relacionada directamente con un camino, sendero o recorrido asignado al modelo lógico. La simulación tendrá en cuenta la frecuencia de paso relativa por el segmento como parte del proceso, así como la distancia entre los puntos uno a uno. Con este procedimiento se pretende obtener de una forma rigurosa un análisis cuantitativo de los datos aportados por las trayectorias reales para poder crear una simulación lo más real posible.

## 4.2 Funcionalidades de *TrackSimulator*

*TrackSimulator* es una aplicación. El término *aplicación* se entiende como la suma de conjunto de implementaciones de código ejecutable y del que se espera un resultado. El flujo del aplicativo se puede ver en la figura red 4.3. Consta de un único tipo de entrada, en este caso trayectorias sobre un terreno en ficheros GPX. Los dos grandes módulos

#### 4. ARQUITECTURA DE LA APLICACIÓN

representan el análisis y la simulación de los datos, donde en el primero realizará una alimentación de una base de datos con la información de las trayectorias y el resultado del análisis, por otra parte almacenará en la máquina una imagen del resultado del análisis. El módulo de simulación realizará el proceso de generación de trayectorias a partir de la lectura de los datos almacenados en la base de datos, del que también se obtienen imágenes de las trayectorias.

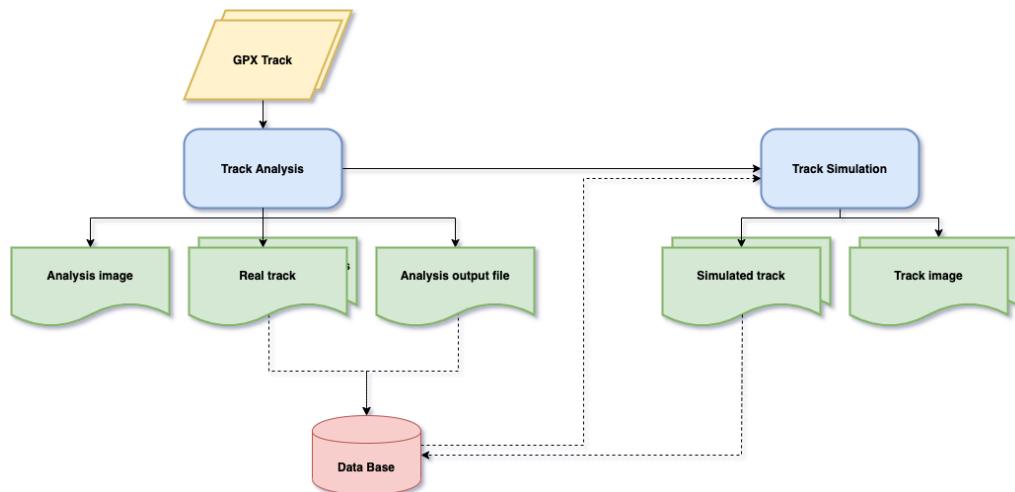


Figura 4.3: Diagrama de funcionamiento de TrackSimulator

La aplicación recopila la siguiente serie de funcionalidades:

**Importación y análisis de archivos GPX** La aplicación soportará la importación de un archivo GPX concreto, o bien de un conjunto de archivos GPX. Estos archivos GPX deben contener trayectorias realizadas en el espacio delimitado. La funcionalidad del aplicativo se basa en un territorio acotado, por lo que el conjunto de rutas a analizar deben corresponder con el territorio específico para que el proceso de análisis se complete.

La aplicación, con los datos importados por los archivos, permitirá un análisis de trayectorias aplicando técnicas de *map matching*. Del análisis se obtendrá información:

**Distancia entre puntos** Distancia entre las capturas de posición GPS en el fichero punto a punto.

**Distancia relativa entre punto de ruta y punto de trayectoria** Distancia del punto GPS detectado de la trayectoria la punto proyección dentro de la ruta, camino o vía seleccionada como más probable en el proceso de *map matching*.

**Frecuencia de paso por segmento de ruta** Frecuencia de paso por el segmento  $x_a, x_b$  relativo a todos los segmentos desde  $x_a$ .

El detalle de esta información se describe en detalle en el capítulo 5.3. Toda esta información quedará almacenada en una base de datos de forma que la información sea accesible para realizar la simulación de la ruta.

**Muestra de resultados del análisis** La aplicación permite la representación gráfica de información gráfica de los resultados del análisis. Las imágenes serán almacenadas en formato Portable Network Graphics (PNG).

**Creación de una trayectoria a partir de parámetros y exportación GPX** Con los resultados de la información analizada se puede realizar una simulación de trayectorias dentro del espacio geográfico delimitado. Los parámetros a introducir quedan detallados en el capítulo 8. Estas trayectorias quedarán representadas en formato GPX.

**Visualización de la trayectoria** La aplicación mostrará una representación gráfica de la trayectoria tanto dentro del territorio geográfico a partir del almacenamiento de imágenes en formato PNG.

### 4.3 Requerimientos de *TrackSimulator*

Explicadas las funcionalidades de la aplicación de Track Analyzer los requerimientos identificados son los siguientes:

- R.1. **Importación de datos** La aplicación debe realizar una importación de los datos desde un fichero .GPX al modelo lógico elegido para la el posterior tratamiento.
- R.2. **Análisis de trayectoria** La aplicación debe realizar un análisis de la trayectoria que proporcione por una parte indicadores y por otro valores medibles, cuantificables y utilizables para realizar una simulación lo más precisa posible.
- R.3. **Almacenamiento de las rutas reales en base de datos** La aplicación debe realizar un almacenamiento de los datos analizados en una base de datos, con el objetivo de poder ser repetible sin necesidad de importar nuevamente los datos.
- R.4. **Almacenamiento del análisis en base de datos** La aplicación debe realizar un almacenamiento de los resultados del análisis en una base de datos, con el objetivo de poder ser accesibles para la etapa de simulación.
- R.5. **Generación de fichero de gráficas resultantes del análisis** La aplicación debe poder realizar una muestra de la información resultante para su visualización por parte del usuario.
- R.6. **Lectura de la base de datos para el acceso a la información** La aplicación debe poder realizar una lectura de la base de datos para acceder a la información necesaria para la muestra de resultados o simulación de trayectorias.
- R.7. **Simulación de puntos GPS a partir de resultado de análisis** La aplicación debe poder generar el camino más probable dada un análisis previo y unos parámetros de entrada. La aplicación debe generar los puntos de cada uno de los segmentos necesarios dado un camino.

## 4. ARQUITECTURA DE LA APLICACIÓN

R.8. **Generación del fichero GPX correspondiente a la simulación de la trayectoria**  
La aplicación realizar una exportación de los datos a formato .GPX.

R.9. **Generación de fichero visualización de trayectoria** La aplicación debe poder mostrar al usuario una visualización de la trayectoria dado un fichero .GPX.

### 4.4 Implementación de TrackSimulation

#### 4.4.1 Flujo de datos

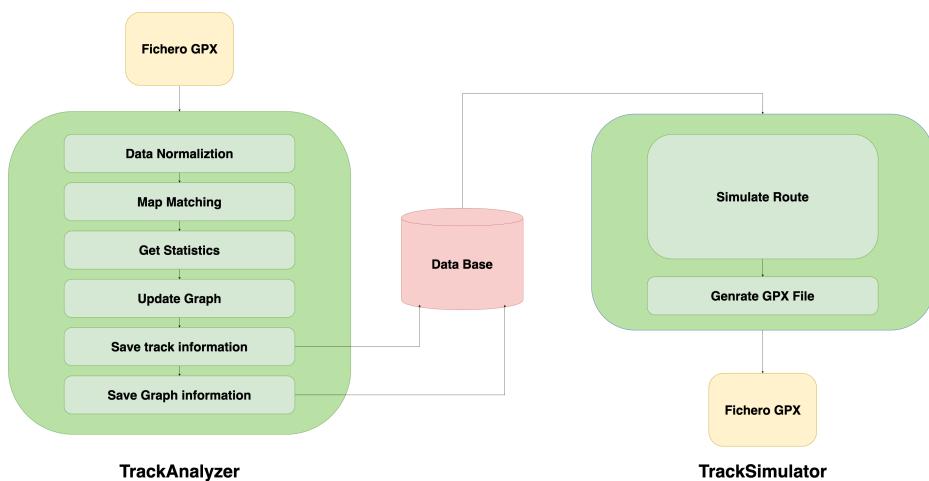


Figura 4.4: Flujo de datos de TrackSimulator

Podemos ver en la figura 4.4 de la parte superior la implementación de la aplicación dos grandes módulos. Por una parte encontramos *TrackAnalyzer*. Este módulo es el encargado de realizar todas los procesos de tratamiento de datos para su posterior análisis. Inicialmente en el alcance de este proyecto se realizará análisis de ficheros GPX por lo que es el único modelo de datos que entrará en nuestro sistema. Para realizar un análisis se realiza un proceso de tratamiento previo de datos, que se detalla en el apartado 5.1.1.

Posterior al tratamiento de datos se realizará el proceso de *Map Matching* con el que se une los datos introducidos al modelo lógico del territorio geográfico. El proceso de *Map Matching* queda descrito en detalle en el apartado 5.2.

Una vez se ha realizado la asignación por cada punto a un camino determinado se puede realizar un análisis del fichero, del cual se obtienen las métricas requeridas por el apartado 4.3. La descripción de la forma en la que se explotan los datos está detallada en la sección 5.3.

Con la explotación del dato hecha, únicamente queda guardar la información generada en una base de datos. El almacenamiento de las trayectorias analizadas y mapeadas, el resultado de los análisis y la estructura lógica con la información modificada se realizarán de forma independiente en secciones diferentes de la base de datos.

El módulo de *TrackSimulator* tiene la responsabilidad de generar la simulación de una trayectoria a partir de unos parámetros de entrada. Una vez generada la simulación, un fichero GPX con las coordenadas será el producto final del proceso.

##### 4.4.2 Herramientas externas utilizadas

Como herramientas externas utilizadas para la realizar la propuesta de este documento se destacan el tipo de almacenamiento de datos que se utilizarán, así como las librerías externas que se han usado para la implementación de la aplicación. A continuación se detallan ambas.

###### Almacenamiento del dato

Tanto de trayectorias reales como de los datos generados por el análisis deben ser almacenados de forma que sean accesibles de forma rápida. Para el almacenamiento de datos, existen dos grandes posibilidades: el uso de Bases de datos relacionales, a partir de ahora descritas como Structured Data Base (SQL-DB), o el opuesto, las bases de datos no relacionales, Non-structured Data Base (NO-SQL-DB). Para la realización de la propuesta descrita en este documento se ha decidido realizar el almacenamiento de toda la información en *MongoDB*, una de las NO-SQL-DB más usadas y que detallamos a continuación.

*MongoDB* se trata de una NO-SQL-DB de código abierto y su funcionamiento es documental. Se almacenan colecciones de documentos, que son series de elementos clave-valor en formato JSON.

*MongoDB* elimina las limitaciones de las bases de datos relacionales [18]. Permite almacenar de forma eficiente grandes cantidades de información, siendo a su vez flexible a modificaciones debido a que los documentos que se almacenan en las colecciones no tienen una taxonomía definida. Por lo que el desarrollo incremental de la aplicación y la aparición de nuevos campos dentro del modelo de datos no es un problema. Otro gran motivo para la elección de *MongoDB* como base de datos de este proyecto es la escalabilidad que ofrece, de forma que a grandes cantidades de trayectorias por almacenar, podría distribuirse utilizando servicios en la nube.

Actualmente para el desarrollo de la propuesta no se ha contado con una gran cantidad de datos. No obstante el problema ha sido planteado con el objetivo de poder analizar grandes cantidades de datos y que puedan ser almacenados y accesible mediante el uso de esta base de datos.

###### Librerías utilizadas

El lenguaje seleccionado para el desarrollo del aplicativo ha sido Python debido al ser recomendable para el tratamiento de datos. Su flexibilidad y la gran utilidad aportada por librerías externas hacen que sea idóneo para el desarrollo de la propuesta. Entendemos como librería un conjunto de código agrupado que aporta funcionalidad diversa. Las librerías externas empleadas en este proyecto son las siguientes :

- L.1. **gpxpy** Librería para la manipulación de ficheros GPX. Mediante esta librería se puede realizar una importación del fichero para su posterior manipulación. De

#### 4. ARQUITECTURA DE LA APLICACIÓN

---

esta forma se obtiene la secuencia de puntos GPS que describe una trayectoria determinada.

- L.2. **pandas** Librería para el análisis de los datos. Toda la información correspondiente a las diferentes trayectorias queda reflejada en un Dataframe para su posterior tratamiento y acceso.
- L.3. **osmnx** Librería para la extracción, visualización y análisis de redes de calles. Mediante esta librería se puede obtener toda la información referente a nuestro espacio determinado del Castillo de Bellver e importar toda la información de sus caminos y carreteras transitables. De esta forma tenemos toda una estructura de datos con información geográfica precisa del entorno. Por otra parte mediante esta librería se puede visualizar las trayectorias escogidas, así como la capacidad de almacenar imágenes de las trayectorias analizadas o simuladas.
- L.4. **matplotlib** Librería por excelencia para la representación de información de forma visual. Mediante el uso de esta librería podemos mostrar diversas gráficas de la información obtenida y analizada por los diferentes algoritmos.
- L.5. **numpy** Librería para el cálculo científico. Esta librería nos aporta diferentes estructuras de datos para el acceso y cálculo de forma eficiente a la información. Una de las principales ventajas de esta librería es la implementación de matrices de forma que el acceso a los datos incrementa su eficiencia de forma considerable.
- L.6. **geopy** Librería para el tratamiento de coordenadas. Mediante esta librería se obtienen las herramientas necesarias para tratar al par de datos (*Lat,Long*) como un punto de coordenada GPS. Por otra parte se obtiene implementación del cálculo de la distancia entre dos coordenadas.
- L.7. **itertools** Esta librería implementa diversos bloques de iteradores. El uso dentro de nuestro proyecto queda exclusivamente para la agrupación de elementos dentro de un set de datos.
- L.8. **sklearn** Librería para el análisis de datos. Esta librería nos aporta todas las herramientas necesarias para el análisis de los datos GPS. Con ella realizamos los cálculos de puntos próximos a partir de una búsqueda basada en Binary Space Partition (BSP) como veremos en la siguiente sección.
- L.9. **shapely** Librería para el tratamiento de figuras geométricas. De esta forma podemos tratar elementos como puntos GPS de forma abstracta. Por otra parte nos permite tratar las rutas del espacio a analizar como una linea de sucesivos puntos GPS.
- L.10. **pickle** Librería para la codificación-decodificación de ficheros. Mediante el uso de esta librería se realiza la exportación de la información estadística y de la estructura de grafo generada y almacenada en los ficheros .txt/edgelist sucesivamente.

#### 4.4.3 Docker

Para poder entender completamente este apartado tenemos que hacer una mención a la forma clásica de arquitectura de proyectos software a nivel de infraestructura. Esta forma tenía como particularidad el uso de máquinas virtuales para la ejecución de las diferentes aplicaciones, así como un supervisor de las máquinas virtuales. Cada máquina virtual despliega su propio sistema operativo con sus dependencias. Por otro lado, encontramos la forma mediante el uso de contenedores.

Una de las implementación más usada de contenedores es Docker. Docker es una plataforma para el desarrollo, despliegue y ejecución de aplicaciones basado en contenedores [19]. Una imagen es una plantilla de lectura donde está almacenado todo el código (normalmente compilado) junto con todas las dependencias necesarias para que pueda ser ejecutado completamente. Un contenedor es una instancia de una imagen, esta puede ser creada, desplazable y parada a partir de la Application Programming Interface (API) de Docker.

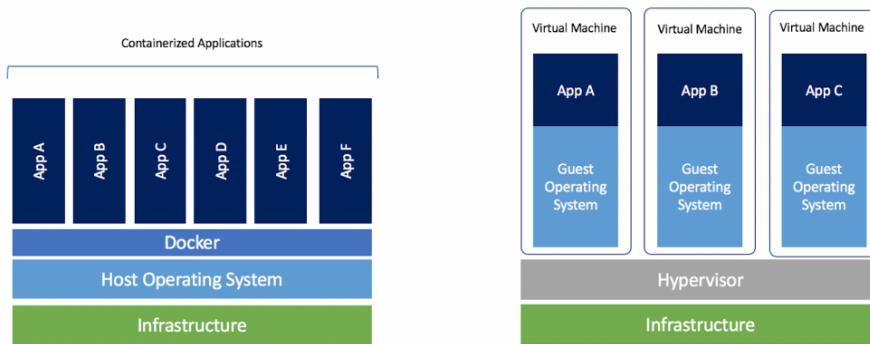


Figura 4.5: Comparación entre arquitectura Docker y mediante máquinas virtuales. [2]

Con esta nueva forma, todas las aplicaciones comparten el mismo sistema operativo, no es necesario ningún supervisor y la responsabilidad de las dependencias queda localizada en los contenedores, que contienen únicamente lo necesario para poder ejecutar la aplicación.

La ejecución del aplicativo tiene como requerimientos una serie de dependencias, que la máquina física puede tener o no en su sistema. Para añadir una capa de abstracción a la arquitectura y hacerlo replicable y desplegable en cualquier máquina se ha utilizado Docker. La ventaja de usar Docker en esta propuesta reside en la capacidad de la plataforma de usar los recursos del sistema de forma eficiente, así como la portabilidad que ofrece.

La infraestructura la propuesta es la siguiente:

#### 4. ARQUITECTURA DE LA APLICACIÓN

---

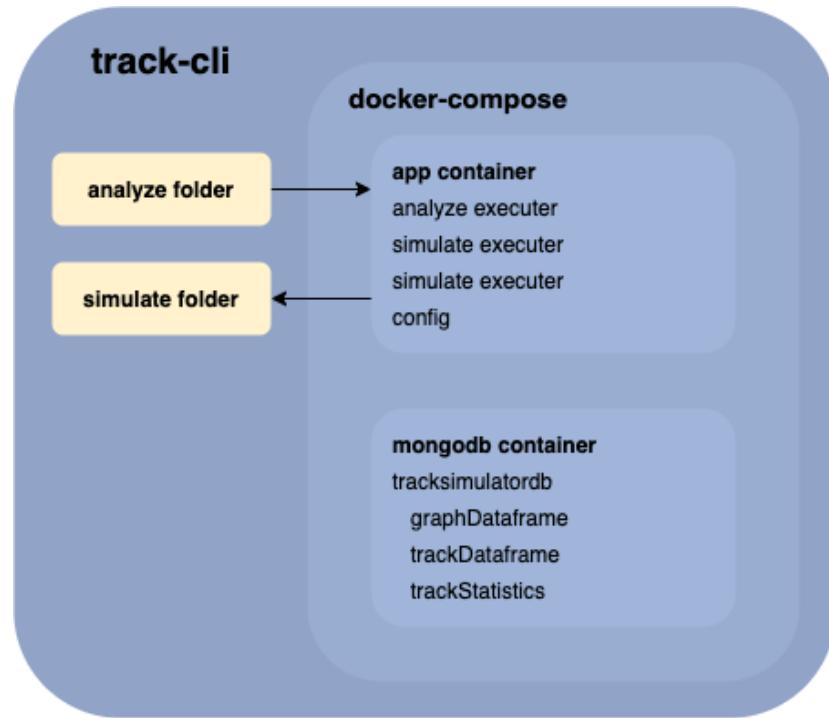


Figura 4.6: Infraestructura de la propuesta de TrackSimulator

Como vemos la aplicación constará de un CLI que se encargará de centralizar todo el aplicativo. Mediante comandos se podrá realizar cada una de las acciones de la aplicación. Es este CLI el encargado de desplegar los contenedores tanto de la base de datos como de la aplicación. Añade una serie de directorios que serán el puente de comunicación entre la aplicación y la máquina que lo ejecuta. En una carpeta se añadirán los ficheros GPX a analizar, y como salida de la simulación tendremos ficheros GPX e imágenes de la ruta en formato PNG.

CAPÍTULO

# 5

## ANÁLISIS DE LOS DATOS GPS

Con el fin de llevar a cabo una simulación de trayectorias, se tiene que realizar previamente un análisis del territorio a simular. El análisis determinará parámetros de simulación que harán posible una generación de trayectorias semejantes a las reales. En este capítulo se explica como está estructurado este análisis. Se observará la importación de los datos del territorio geográfico y del modelo lógico del terreno. Se explica el proceso de map-matching y su implementación con el algoritmo de Viterbi para la detección de los puntos. Finalmente, con el modelo de detección de puntos definido, se explica qué datos se van a explotar y de qué forma.

### 5.1 Preparación del entorno de datos

El entorno de datos con el que se trabaja dentro de la propuesta de este documento consta de dos partes fundamentales: los ficheros de trayectorias reales en formato GPX y el modelo lógico de carreteras y caminos por el que estos ficheros se sitúan. Durante esta sección se describen ambos.

#### 5.1.1 Importación de datos GPS a *TrackSimulator*

Para realizar un tratamiento de los datos a partir de un fichero GPX se realiza un uso de la librería *gpxpy*, cuya funcionalidad ha sido explicada anteriormente en el subapartado 4.4.2. El primer paso a realizar es el parseo del fichero. La palabra parseo es un vocablo adaptado del inglés (parsing) con el que se describe el proceso de análisis de código, normalmente XML para obtener y tratar la información que contienen. De esta forma encontramos un objeto *track*, que contiene *segments* que a su vez contienen los diferentes *waypoints*. De esta forma se obtiene la estructura que podemos ver en la figura 5.1 :

## 5. ANÁLISIS DE LOS DATOS GPS

---

```
<trk>
  <name>Trote </name>
  <type>trail_running</type>
  <trkseg>
    <trkpt lat="39.56738345324993133544921875" lon="2.6236434839665889739990234375">
      <ele>30.200000762939453125</ele>
      <time>2019-01-06T11:12:47.000Z</time>
      <extensions>
        <ns3:TrackPointExtension>
          <ns3:atemp>26.0</ns3:atemp>
          <ns3:hr>85</ns3:hr>
          <ns3:cad>48</ns3:cad>
        </ns3:TrackPointExtension>
      </extensions>
    </trkpt>
```

Figura 5.1: Estructura de un *track* dentro de un fichero .GPX

La información necesaria para la implementación de la solución propuesta en este documento es la posición en latitud y longitud de cada uno de los puntos. El tiempo y la elevación han quedado fuera del alcance de esta propuesta. El resto de información tanto de puntos (elevación) como de segmentos (metadatos) no será usada para el desarrollo de la propuesta, queda una estructura en forma de lista del siguiente tipo de elemento.

Punto preprocessado		
Campo	Tipo	Descripción
longitud	Float	Posición del nodo respecto al eje horizontal.
latitud	Float	Posición del nodo respecto al eje vertical.

Cuadro 5.1: Estructura lista de puntos.

Se ha decidido realizar de esta forma para facilitar la manejabilidad de los datos mediante el objeto *TrackPoint*, que es un objeto nexo a toda la funcionalidad del aplicativo.

### 5.1.2 Entidad TrackPoint

Para añadir una capa de abstracción adicional al aplicativo toda interacción dentro del sistema se realiza a partir de la entidad *TrackPoint*. Esta entidad es una clase que extiende de la clase *Point* de la librería *shapely*, de esta forma, nos aseguramos que si en un futuro la librería externa deja de funcionar podremos extender otra clase usable, siempre implementando los métodos necesarios asimismo, tenemos una entidad lógica que puede ser ampliada en futuras iteraciones del aplicativo.

### 5.1.3 Importación de caminos y carreteras al modelo de datos

En esta propuesta se ha optado por plantear la solución de importación usando una red compleja, siguiendo el modelo establecido por G.Boeing [14]. Para abordar correctamente la problemática del análisis de rutas dentro del espacio urbano se debe plantear una solución con una red compleja. Esta red está modelada en forma de grafo. Con esta estructura de datos se puede contemplar un proceso de importación, análisis, y exportación de datos geográficos. Para realizar la importación de los datos se recurre a la librería OSMnx comentada en la subsección 4.4.2. Mediante el uso de un corto número de instrucciones se consigue importar toda la estructura de la zona del Castell de Bellver.

```
1 self.graph = osmnx.graph_from_bbox(north, south, east, west)
```

Algoritmo 5.1: Importación de una zona mediante la librería osmnx.

Con la importación de los datos se ha conseguido obtener la información de OpenStreetMap, no obstante se debe realizar un tratamiento de filtrado de los datos importados debido a que aparece información que en este proyecto no tienen relevancia. Esta información se trata de características concretas de calles, peatonales o no, como la dirección. Nuestro planteamiento del problema define que la trayectoria a simular no tiene en cuenta el sentido de la carretera. Eso quiere decir que una carretera para vehículos de una dirección, se convierte en un camino de posible transito.

Por otra parte, se añade información tanto a los nodos como a las aristas del grafo para almacenar información necesaria para la solución del problema. Por lo tanto, al final del tratamiento de la red obtendremos un grafo dirigido, donde las intersecciones entre la infraestructura urbana ( calle, camino, carretera) están definidas como un nodo y la característica de esta como la arista. Por simplicidad de entendimiento para la implementación de soluciones se ha decidido aplicar dos aristas por cada nodo en común, con la información geoespacial correspondiente. La red compleja queda establecida como muestran las tablas 5.2 y 5.3 una vez aplicado el procesado:

## 5. ANÁLISIS DE LOS DATOS GPS

---

Nodo		
Campo	Tipo	Descripción
Identificador	Integer	Clave del nodo.
x	Float	Posición del nodo respecto al eje horizontal.
y	Float	Posición del nodo respecto al eje vertical.
osmid	Integer	Identificador de OSM

Cuadro 5.2: Estructura nodo.

Arista		
Campo	Tipo	Descripción
Identificador	(Integer, Integer, Integer)	Clave de la arista
osmid	Integer	Identificador de OSM
oneway	Boolean	Carretera de un único sentido.
name	String	Nombre de la carretera.
highway	String	Tipo de carretera.
length	Float	Longitud del camino.
geometry	LineString	Geometría de la carretera.
num of detections	Integer	Núm. de detecciones de puntos.
frequency	Float	Frecuencia de puntos detectados.

Cuadro 5.3: Estructura arista.

## 5.2 Map-Matching

Uno de los grandes retos del análisis de trayectorias es la identificación y asignación de cada uno de los puntos GPS en su correspondiente trayectoria. Este proceso es llamado Map-Matching. Los puntos GPS se obtienen a partir de un dispositivo que administra la serie de puntos GPS correspondientes a una trayectoria con un cierto error o desviación. Esta serie de puntos son puntos aislados y no tienen relación directa con el modelo lógico establecido, es decir, con la información geográfica asociada.

Existen diversos procedimientos para abordar el problema del *Map-matching*. La aproximación al problema en esta propuesta se realiza aplicando el modelo probabilístico de las cadenas de Markov, que explicamos en el siguiente apartado.

### 5.2.1 Hidden Markov Model (HMM)

HMM sigue el modelo estadístico tradicional de Markov [20]. En 1907, A. A. Markov comenzó el estudio de un proceso donde un experimento podía afectar a la salida del siguiente experimento. Este tipo de proceso se denominó Cadena de Markov.

Definiremos un conjunto de estados  $S = s_1, s_2, \dots, s_n$ . El proceso empieza en un estado inicial y se traslada de estado en estado. Esta transición se le denomina *step*. Los *steps* entre un estado  $s_i$  y  $s_j$  cumplen una **probabilidad de transición**.

Por otro lado encontramos la **probabilidad de emisión**. Esta probabilidad responde a observar un estado  $S$  en un instante  $t$  determinado.

Markov define como suposición de primer orden que la probabilidad de ocurrencia de un evento en un tiempo  $t$  solo dependerá de  $t - 1$ . De esta forma las observaciones  $O - 2, \dots, O - n$  no afectaran directamente a  $t$ .

### 5.2.2 Aproximación de Hidden Markov Model a Map-matching

La estructura lógica del territorio en esta propuesta está definida como un grafo formado por diferentes *nodos* y *aristas*. Los nodos corresponden a la intersección entre caminos, mientras que en las aristas se almacena la información referente a la constitución geográfica de estos.

Esta información queda almacenada en un *Shape* de forma que se trata de una sucesión de puntos GPS tal y como se muestra en la siguiente imagen:

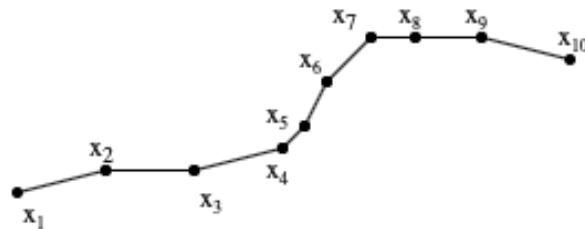


Figura 5.2: Ejemplo de estructura LineString

Cada uno de las proyecciones sobre los subsegmentos identificados entre  $x_1, \dots, x_{10}$  son proyecciones validas a una observación determinada.

Por otra parte aparece la información GPX. El fichero que almacena la información de trayectorias que se han realizado en las delimitaciones de nuestro territorio. Esta serie de puntos se identifican como **observaciones**. Tenemos entonces los dos elementos principales que se han descrito anteriormente: una serie de observaciones y un modelo. A partir de aquí, se plantea el **problema general de decodificación**: dada una secuencia de observaciones, ¿qué secuencia de estados es la más probable para generarlos? Este problema se resuelve mediante la implementación del *Algoritmo de Viterbi*, que explicamos a continuación.

### 5.2.3 Algoritmo de Viterbi

El algoritmo de Viterbi se presenta como una técnica computacional con la que se obtiene la sucesión de nodos más probable para una cadena de Markov. Para cada una de las proyecciones se obtiene una probabilidad, siendo la más alta la elegida para ser el nuevo elemento del camino de Viterbi. La figura 5.3 muestra el objetivo de este algoritmo, poder detectar que la secuencia de puntos corresponde a la trayectoria formada por los puntos  $x_n$ .

## 5. ANÁLISIS DE LOS DATOS GPS

---

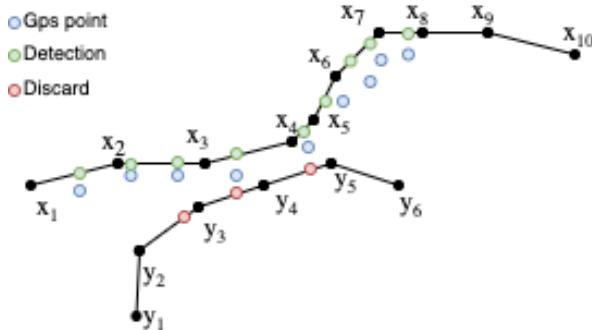


Figura 5.3: Ejemplo teórico de detección map-matching mediante algoritmo de Viterbi

La probabilidad tiene en cuenta dos factores:

**Probabilidad de emisión** Esta probabilidad es calculada por análisis espacial. Por convenio establecemos que cuanto más próxima sea la observación a la proyección más probable será que sea la elección correcta, partiendo de una distribución normal con media cero tal y como se muestra en la referencia [21].

$$P_{emission} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(d_i^j - \mu)^2}{2\sigma^2}} \quad (5.1)$$

**Probabilidad de transición** La probabilidad de llegar al estado  $S_i$  de una proyección viene determinado por el estado anterior  $S_{i-1}$  en una relación de dos aspectos. Por una parte la distancia espacial entre el  $S_i$  y  $S_{i-1}$  identificado como  $d(S_i, S_{i-1})$ . Por otra parte encontramos el camino más corto entre estos dos estados. El camino más corto definido como  $SP(S_i, S_{i-1})$  se entiende como la cantidad de nodos que se tienen que atravesar para llegar de  $S_i$  a  $S_{i-1}$ . Esta última se mantiene como una distribución exponencial de forma que a mayor número de nodos por atravesar en el camino más corto esta proyección es penalizada [21]..

$$P_{transition} = \frac{distance(p_{i-1}, p_i)}{e^{SP(p_{i-1}, p_i)}} \quad (5.2)$$

Los valores de  $\sigma$  y  $\beta$  se han obtenido mediante pruebas. Se observa en la figura 5.4 la representación de la función de probabilidad de emisión respecto al valor de  $\alpha$ . Se ha escogido  $\sigma = 4$  debido a que es un valor que da unos valores de probabilidad válidos para el rango de 5 a 10 metros.

Por otro lado, la figura 5.5 es la representación de la probabilidad de transición respecto a  $\beta$ . Se ha determinado que  $\beta = 0,1$  porque la probabilidad de transición es adecuada debido a que es tolerante con valores entre 1 y 3 nodos de distancia, y para los datos tratados ofrece valores válidos. Cabe destacar que estos valores son modificables según la exactitud con la que cuenten los datos.

La implementación del algoritmo se realiza de forma iterativa para todos los puntos GPS de la trayectoria. Por cada punto se buscan las proyecciones en un radio determinado. De esta forma se realiza una primera criba de proyecciones que no serán probables.

## 5.2. Map-Matching

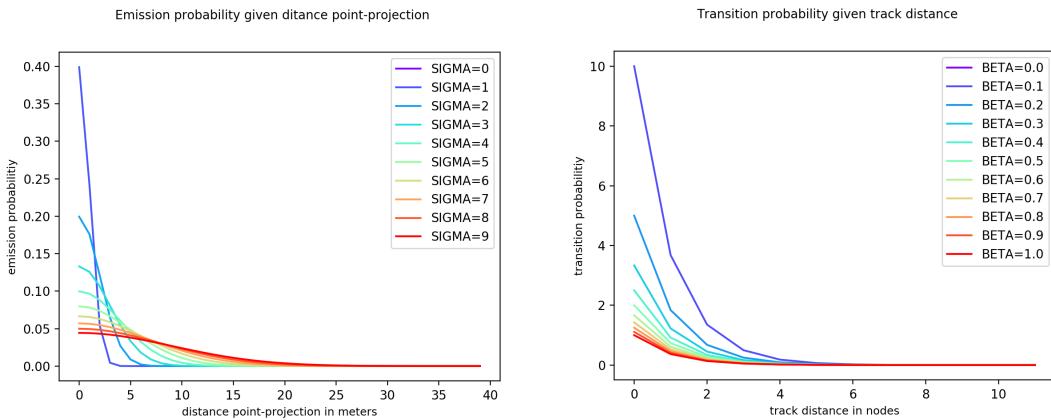


Figura 5.4: Análisis de la función de  $P_{emission}$  para  $\sigma$ .

Figura 5.5: Análisis de la  $P_{transition}$  de transición para  $\beta$ .

Por cada una de las proyecciones se calcula las probabilidades descritas anteriormente. La probabilidad de transición viene determinada por el estado anterior, tal y como se puede observar en la fórmula. Una vez tratadas todas las proyecciones. Se almacena aquella proyección con la mayor probabilidad total. Si el camino más corto entre el nodo correspondiente a la proyección  $p_i$  y  $p_{i-1}$  es mayor que uno se debe completar el camino más corto.

Con la aplicación de esta información se obtiene una secuencia de puntos GPS relacionados con la estructura de datos y queda solucionado el problema del *map-matching*.

### 5.2.4 Ejemplo de aplicación del algoritmo de Viterbi

Con el siguiente ejemplo se pueden observar los beneficios y los inconvenientes de la aplicación del algoritmo implementado en esta propuesta.

En la figura 5.6 se observa en puntos rojos la trayectoria real de un individuo dentro del territorio limitado del Castell de Bellver. Los puntos verdes corresponden a la detección de las proyecciones más probables por parte del algoritmo de Map-matching.

El algoritmo realiza una detección punto a punto de su proyección dentro del camino. Pueden aparecer situaciones en las que por desviación de precisión en la detección GPS un punto parezca situarse cerca de un camino no correcto. No obstante, la probabilidad de transición descrita en el apartado anterior se encarga de ofrecer como opción más probable, aquella donde el *shortest path* del grafo sea menor. 5.7

## 5. ANÁLISIS DE LOS DATOS GPS

---



Figura 5.6: Ejemplo de aplicación de algoritmo de Viterbi para detección de trayectorias.

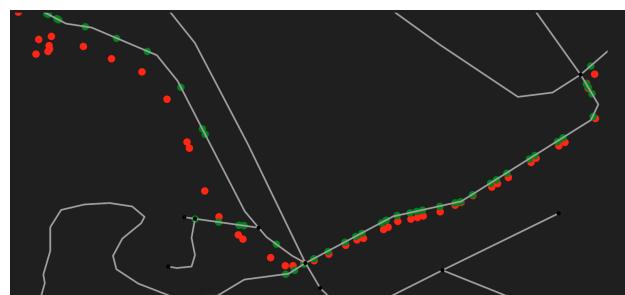


Figura 5.7: Ejemplo de aplicación de algoritmo de Viterbi para detección de trayectorias.

No obstante pueden aparecer casos aislados donde la detección GPS es tan cercana a una proyección de un camino incorrecto que hace imposible su elección respecto a la proyección correcta. 5.8

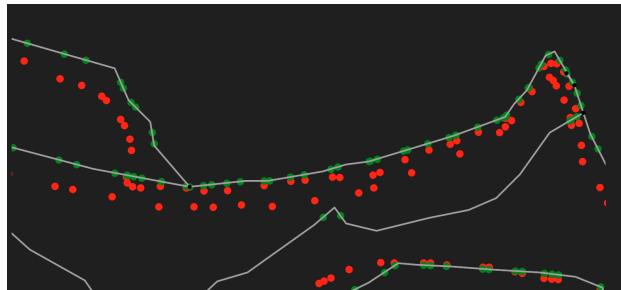


Figura 5.8: Ejemplo de aplicación de algoritmo de Viterbi para detección de trayectorias.

Cabe destacar que el funcionamiento correcto de la detección de proyecciones depende de que el individuo realice el recorrido basado en los caminos posibles por el modelo de datos obtenido de OSM. La detección de puntos por senderos no determinados previamente no pertenece al alcance de este problema.

### 5.2.5 Implementación de Map-matching

La implementación del algoritmo de map-matching se basa en dos grandes partes: el *Interactor get\_map\_matching* y la *Entity hidden\_markov\_model*

El primero de ellos, *get\_map\_matching*, se trata de una interfaz cuya única responsabilidad consiste en realizar el proceso de detección de proyecciones descrito en apartados anteriores. La interfaz se mantiene abstracta a la utilización de cualquier otro método que no sea el camino de Viterbi. Es en la implementación de esta interfaz (*get\_map\_matching\_impl*) la que almacena la lógica. En la implementación propuesta despliega el método *viterbi\_algorithm* de la entidad *hidden\_markov\_model*.

```

1 class GetMapMatchingImpl(GetMapMatching):
2     def __init__(self, detection_model: HiddenMarkovModel):
3         self.detection_model = detection_model
4
5     def match(self, points):
6         mapped_track, _ = self.detection_model.viterbi_algorithm(points)
7         return mapped_track

```

Algoritmo 5.2: Implementación interactor *get\_map\_matching\_impl*

Por otra parte, encontramos *hidden\_markov\_model*. Se trata de una entidad abstracta que almacena los métodos necesarios para implementar la detección de proyecciones. Los métodos necesarios para implementar una cadena de Markov son los cálculos de probabilidad de emisión y de probabilidad de transición. Éstos junto con la implementación del algoritmo de Viterbi describen toda la responsabilidad de esta entidad.

```

1 class HiddenMarkovModel(abc.ABC):
2     @abc.abstractmethod
3     def get_emission_prob(self, projection, point):
4         pass
5
6     @abc.abstractmethod

```

## 5. ANÁLISIS DE LOS DATOS GPS

---

```

7     def get_transition_prob(self, point, projection, next_point, next_projection):
8         :
9         pass
10    @abc.abstractmethod
11    def viterbi_algorithm(self, points):
12        pass

```

Algoritmo 5.3: Declaración interfaz hidden\_markov\_model

### 5.3 Explotación del dato

Con la problemática descrita en los apartados anteriores resuelta. Se tiene la capacidad de poder inferir toda la información planteada para la propuesta de esta solución.

#### 5.3.1 Análisis de distancias

Con la relación  $P_{real} - P_{proyectado}$  y  $P_n - P_{n+1}$  podemos generar la distancia entre estos puntos para toda la trayectoria. La distancia que calculamos es la distancia más corta sobre la superficie terrestre, la formula elegida es la formula **Haversine** [22] [23]. Esta fórmula tiene en cuenta el radio de la esfera terrestre para realizar el cálculo de la siguiente forma:

$$a = \pi/2 - lat_1 \quad (5.3)$$

$$b = \cos(lat_1) * \cos(lat_2) * \cos(lon_2 - lon_1) \quad (5.4)$$

$$c = \arccos(a + b) \quad (5.5)$$

$$d = R * c \quad (5.6)$$

Con el cálculo de la distancia se permite encontrar tanto la desviación entre la trayectoria real y la ruta real como la distancia entre las muestras de puntos GPS de la trayectoria.

La detección de la proyección permite averiguar que caminos se han atravesado para generar la trayectoria. Con esta información la frecuencia de paso por cada uno de los caminos puede ser generada. La frecuencia de paso será relativa a la frecuencia de paso de todos los caminos con el mismo nodo origen. De esta forma si hay 4 caminos  $c_{1,...,4}$  para un nodo  $n_x$ , siendo  $d$  el número de detecciones en la arista la probabilidad del punto será la siguiente:

$$p_{c_i} = \frac{d_{c_i}}{\sum d_{n_x}} \quad (5.7)$$

Esta información es analizada y tratada de forma que el proceso de simulación genere una trayectoria basada en información real. La implementación de estos cálculos se realiza en el **interactor** `get_analysis_statistics_impl` y el método `_add_next_point_distance` y describen la comparación de la distancia *Haversine* con el anterior punto, o bien con la proyección.

```

1     def __add_next_point_distance(self, data: DataFrame) -> DataFrame:
2         data['point_lon_shift'] = data.Point_lon.shift()
3         data['point_lat_shift'] = data.Point_lat.shift()
4         data['DistanceToNext'] = data.apply(lambda x: Point(x['Point_lon']), x['Point_lat']).haversine_distance(

```

```

5     Point(x[ 'point_lon_shift' ],
6             x[ 'point_lat_shift' ])) ,
7             axis=1)
8 return data.drop(columns=[ 'point_lat_shift' , 'point_lon_shift' ])

```

Algoritmo 5.4: Método add\_next\_point\_distance

```

1 def __add_point_projection_distance(self , data: DataFrame) -> DataFrame:
2     data[ 'DistancePointProjection' ] = data.apply(lambda x: Point(x[ 'Point_lon'
3             ] ,                                                 x[ 'Point_lat'
4             ]) .haversine_distance(
5                 Point(x[ 'Projection_lon' ] ,
6                         x[ 'Projection_lat' ])) ,
7                 axis=1)
8 return data

```

Algoritmo 5.5: Método add\_point\_projection\_distance

### 5.3.2 Demostración de análisis

A continuación se muestran los resultados de un análisis utilizando la metodología detallada en la parte superior del documento, se cuenta con una muestra total de 25 rutas reales realizadas recorriendo diversos caminos en el territorio acotado del Castell de Bellver. Destacamos que parte de la muestra contiene casuísticas donde el individuo ha realizado recorrido por territorio no identificado como ruta. Para sopesar este impacto dentro de los resultados se han *outliers* aquellos puntos que superen una distancia superior a 40 metros. El análisis realizado nos deja el siguiente resultado:

## 5. ANÁLISIS DE LOS DATOS GPS

---

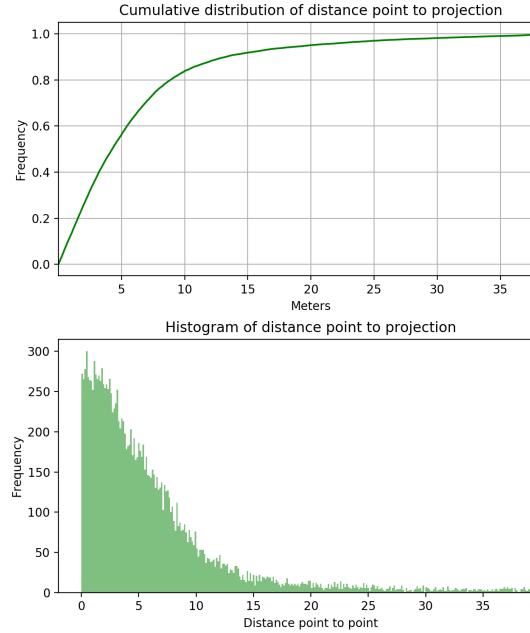


Figura 5.9: Análisis de la distancia punto a proyección.

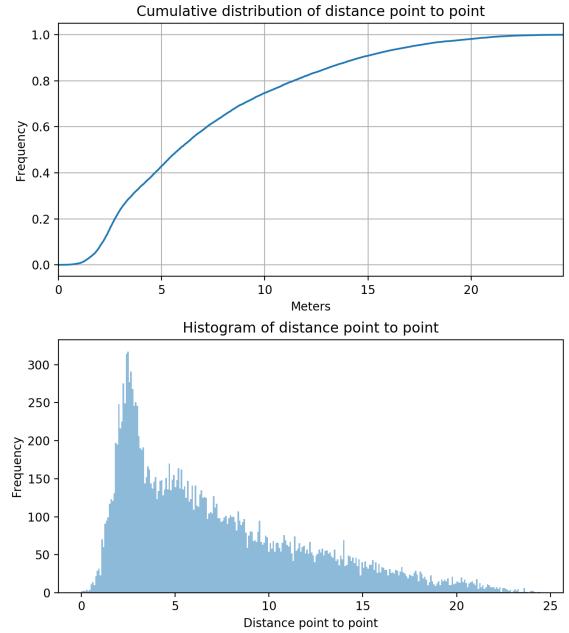


Figura 5.10: Análisis de la distancia punto a punto.

Como se ve en la imagen 5.9 el 80 % de la distancia entre el punto y la proyección con el camino correcto está establecido entre 0 y 10 metros de distancia.

En la figura 5.10 vemos la información correspondiente a la distancia entre un punto  $x_n$  y el punto  $x_{n+1}$ . El espectro de distancia en este caso es mayor, esto puede ser debido tanto a la frecuencia del dispositivo, como a la velocidad de individuo, que puede variar en función de la pendiente del camino.

## SIMULACIÓN DE TRAYECTORIAS

Se define como simulación de una trayectoria al proceso por el cual, mediante los valores medibles y cuantificables extraídos del proceso de análisis, se recrea una trayectoria aleatoria que cumpla los mismos criterios. El proceso de simulación de esta propuesta tiene dos procesos principales. Primero (figura 6.1) generar el camino dada las probabilidades analizadas desde el nodo inicio seleccionado. Posteriormente se realiza la simulación de la trayectoria por el camino seleccionado (figura 6.2).



Figura 6.1: Generación de camino.

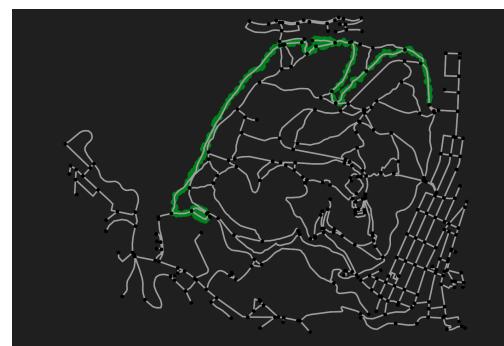


Figura 6.2: Generación de puntos.

## 6. SIMULACIÓN DE TRAYECTORIAS

---

En las siguientes secciones se describe estos dos procedimientos en detalle.

### 6.1 Generación de un camino

La estructura lógica comentada en la sección 5.1.3 permite almacenar en las aristas la información referente a la frecuencia relativa de paso. Para generar un camino, se realiza un proceso iterativo por el cual mediante un nodo origen, asignado por parámetro de entrada, se selecciona uno de los caminos, manteniendo la distribución probabilística del conjunto de aristas, de forma que si una de las aristas tiene una frecuencia del 25% de las detecciones, con un número suficiente de muestras se cumplirá dicha distribución.

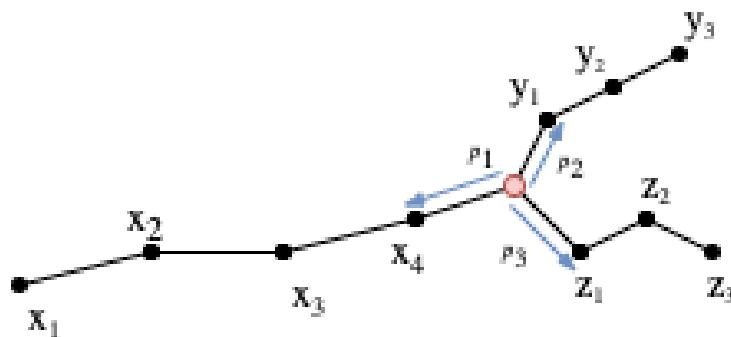


Figura 6.3: Ejemplo de probabilidades dentro de un nodo intermedio entre rutas.

El proceso de selección de aristas continúa hasta que la suma de las distancias acumule el total de distancia por recorrer, que será otro de los parámetros de entrada. La implementación de este proceso se encuentra en el método *create\_path* de la implementación del *interactor simulate\_track\_impl*

```

1  def create_path(self, origin, dist):
2      path = []
3      distance_created = 0
4      prev_node = origin
5      path.append(origin)
6      while distance_created < dist:
7          next_node = self.get_most_frequent_node(prev_node, path)
8          distance_aux = distance_created + self.graph.get_edge_by_nodes(
9              prev_node, next_node)[ 'length' ]
10         if distance_aux < dist:
11             distance_created = distance_aux
12             path.append(next_node)
13             prev_node = next_node
14         else:
15             return path, distance_created
16     return path, distance_created

```

Algoritmo 6.1: Método *create\_path*

## 6.2 Generación de los segmentos

Una vez se tiene seleccionado el camino que se realizará se simula cada uno de los segmentos individualmente. Para ello se identifica el nodo origen y final del segmento y se generan todos los puntos necesarios hasta que la distancia a ese punto sea menor a una  $d$ , donde  $d$  es una distancia parametrizada y modificable. La implementación de este proceso se encuentra en el método *simulate\_segment* de la implementación del *interactor simulate\_trackImpl*

```

1  def simulate_segment(self, segment):
2      aux = 0
3      origin_node = segment[0]
4      target_node = segment[1]
5      segment = []
6      origin_point = TrackPoint(self.graph.get_nodes()[origin_node]['x'], self.
graph.get_nodes()[origin_node]['y'])
7      target_point = TrackPoint(self.graph.get_nodes()[target_node]['x'], self.
graph.get_nodes()[target_node]['y'])
8      try:
9          dest, aux = self.calculate_point(segment, origin_node, target_node,
origin_point, target_point)
10         next = dest
11         while aux > DISTANCE_TO_FINAL_NODE:
12             dest, aux = self.calculate_point(segment, origin_node,
target_node, next, target_point)
13             next = dest
14     except KeyError:
15         pass
16     return segment

```

Algoritmo 6.2: Método *simulate\_segment*

### 6.3 Generación del punto pseudo-aleatorio

Es en este punto de la propuesta donde se realiza la generación de puntos de forma pseudo-aleatoria, utilizando información procedente de los análisis realizados. Para la creación de un punto a partir de otro se necesita una distancia  $d$  y un ángulo  $\alpha$  como se muestra en la figura 9.1.

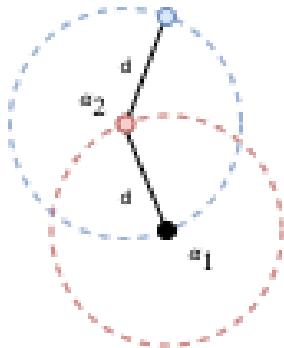


Figura 6.4: Ejemplo de generación de punto a partir de una distancia  $d$  y un ángulo  $\alpha$ .

Cada punto será generado teniendo en cuenta el punto anterior. Se encuentra la proyección del segmento más cercano y se centra el ángulo de direccionamiento en el nodo inmediatamente posterior, con una desviación  $\beta$  parametrizada y modificable. El proceso queda ilustrado en la figura ??

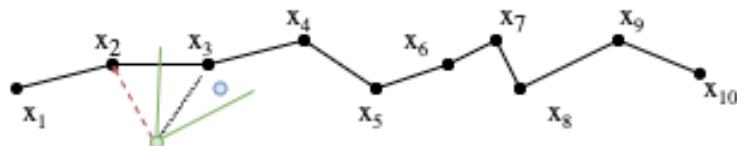


Figura 6.5: Ejemplo de detección de proyección cercana y generación de punto.

La distancia para la generación del punto se obtiene de una elección aleatoria siguiendo un proceso llamado *Inverse transform sampling*, en el que se genera un número pseudo-aleatorio a partir de una función acumulativa dada una distribución probabilística [24].

La distribución probabilística de la distancia punto a punto se obtiene a partir de la información que se genera en el proceso de análisis. Se puede ver un ejemplo de esta información en la figura 5.10 mostrada en la sección 5.3.1.

### 6.4 Exportación de trayectoria a fichero .GPX

En esta sección describe la funcionalidad del aplicativo para realizar la exportación a GPX. Esta funcionalidad es la que establece una salida al proceso de la aplicación y que permite que la trayectoria pueda ser almacenada, manipulada, visualizada y usada en otras aplicaciones.

## 6.4. Exportación de trayectoria a fichero .GPX

---

Como salida del proceso de simulación de una trayectoria se obtiene una lista de puntos. Esta lista es transformada en el formato válido a partir del recurso *gpx*. Es la implementación de esta interfaz la que utiliza la librería *gpypy*.

El método *write* de la interfaz se encarga la creación de un directorio con el formato fecha y hora YYYYmmdd\_HHMM00 (Por ejemplo 20200503\_150300). La hora está configurada en formato Coordinated Universal Time (UTC). Posteriormente se genera una sola *track* y segmento, y cada punto es convertido en un *trackpoint* de este formato. Finalmente, se genera esta trayectoria y se crea un fichero con identificador único Universally unique identifier (UUID) (simulated\_track\_d9cac860775d4454b8c4-4b97f8530bfe.gpx) y se vuelca la representación XML. Este proceso se implementa en método *create\_gpx\_track* que vemos en el algoritmo 9.

```
1 def create_gpx_track(data):
2     gpx = gpypy.gpx.GPX()
3     gpx_track = gpypy.gpx.GPXTrack()
4     gpx.tracks.append(gpx_track)
5     gpx_segment = gpypy.gpx.GPXTrackSegment()
6     gpx_track.segments.append(gpx_segment)
7     [gpx_segment.points.append(gpypy.gpx.GPXTrackPoint(point[1], point[0])) for
8      point in data]
9     return gpx.to_xml()
```

Algoritmo 6.3: Método *create\_gpx\_track*



CAPÍTULO



## EXPERIMENTACION

En este capítulo se documentan las pruebas realizadas al aplicativo propuesto con el objetivo de analizar los resultados obtenidos del análisis y la simulación. Esta experimentación parte de unas condiciones generales. Contamos con un total de 25 rutas reales realizadas en el territorio delimitado del Castell de Bellver (Mallorca). Para la realización de la experimentación contamos con un número reducido de muestras. Para que el resultado del análisis pueda representar un comportamiento destacable se han replicado las mismas rutas un total de 7 veces, contando entonces con un total de más de 200 trayectorias, de una muestra inicial de 30. De esta forma alimentamos el modelo con una cantidad de información suficiente para apreciar resultados. Estas condiciones de entrada parten de la necesidad de alimentar el modelo con una información que pueda llegar a ser representativa.

En la figura 7.1 se muestra el mapa de calor resultado del proceso de análisis:

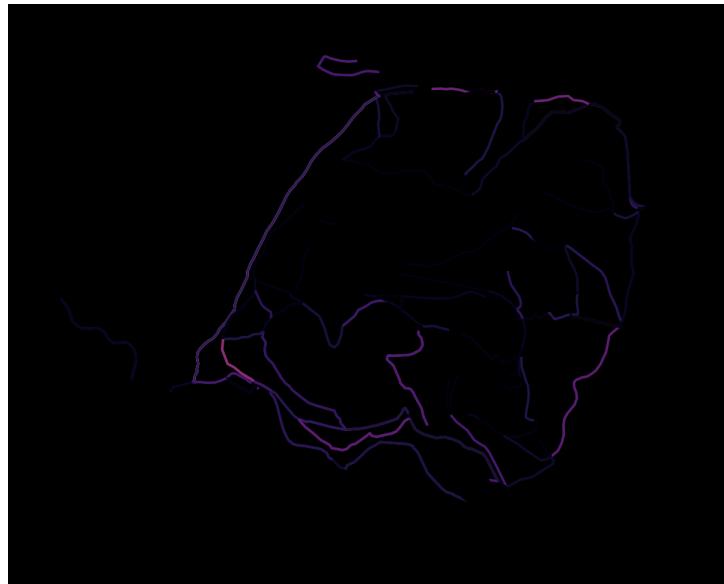


Figura 7.1: Mapa de calor del conjunto de rutas reales.

Este mapa representa la frecuencia de paso del individuo por cada uno de los caminos establecidos en el territorio. Colores oscuros representarían frecuencias bajas y el color cobra una tonalidad rojiza a medida que la frecuencia sería mayor.

Se observa que el individuo ha realizado uso mayor del perímetro del bosque del territorio. Esto concuerda con la visualización de las rutas individuales. Con un conjunto mayor de datos, el mapa de calor llegaría a mostrar colores rojizos.

Las pruebas que realizaremos se basan en dos vertientes. Por una parte se realiza un análisis de una generación de rutas donde no se han importado datos reales. Por otro parte se realiza un análisis de rutas simuladas con datos reales introducidos en el modelo.

## 7.1 Experimentación sin importación de datos reales

En este apartado realiza la muestra de resultados de la generación de 10 rutas creadas mediante la simulación sin datos de análisis almacenados. Al no tener datos almacenados dentro del sistema la simulación corresponderá a una elección aleatoria de los diferentes caminos ya que, por defecto, todos los caminos tienen la misma probabilidad de tránsito a falta de introducir datos en el modelo. Por otra parte la distancia punto a punto será una constante definida en 20 metros, por lo que no aparecerá variabilidad en la muestra. Esta constante está parametrizada y puede ser modificable por el usuario. Se han realizado una simulación de 10 rutas. Una muestra de estas simulaciones las podemos encontrar en el apartado 7.2.3. Las figuras 7.8, 7.9 y 7.10 son ejemplos de estas simulaciones.

## 7.1. Experimentación sin importación de datos reales

---



Figura 7.2: Muestra de simulación 1 (Experimentación sin datos).

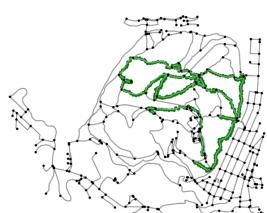


Figura 7.3: Muestra de simulación 2 (Experimentación sin datos).



Figura 7.4: Muestra de simulación 3 (Experimentación sin datos).

Por una parte, en el mapa de calor resultante (figura 7.5) no se observan patrones observables, esto es debido a la elección pseudo-aleatoria de cada uno de los segmentos seleccionados.

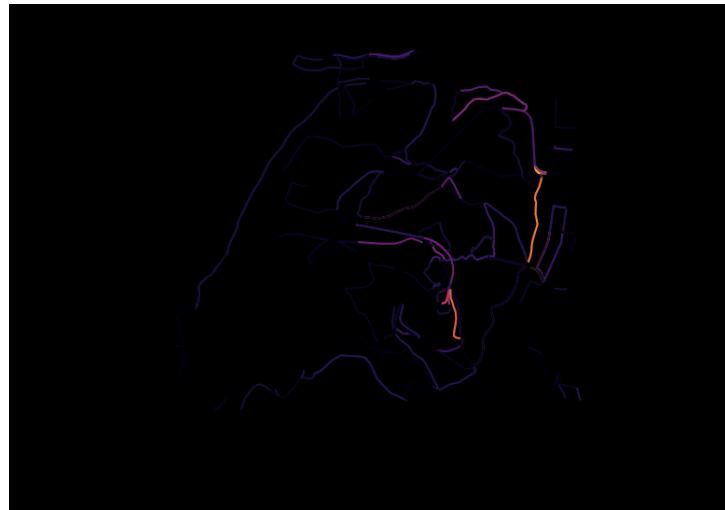


Figura 7.5: Mapa de calor del conjunto de rutas simuladas en experimentación sin datos.

Las distribuciones se observan constantes al valor parametrizado al que están asignados en caso de que no se encuentren datos para realizar la simulación (figura 7.7) por otra parte se observa la aleatoriedad en forma de distribución normal de la distancia punto a proyección, que no corresponde a ningún comportamiento parametrizado (figura 7.6).

## 7.2 Experimentación con importación de datos reales

Para la simulación, se ha alimentado al modelo con el análisis de las rutas reales de 10 km de distancia sobre el territorio del Castell de Bellver. Todas parten del origen de la entrada nordeste del recinto. Se han generado un total de 253 rutas. Posteriormente estas rutas han sido pasadas por el analizador con el objetivo de valorar si mantienen la misma distribución en las variables respecto el análisis de las rutas reales. Las figuras 7.8.

## 7.2. Experimentación con importación de datos reales

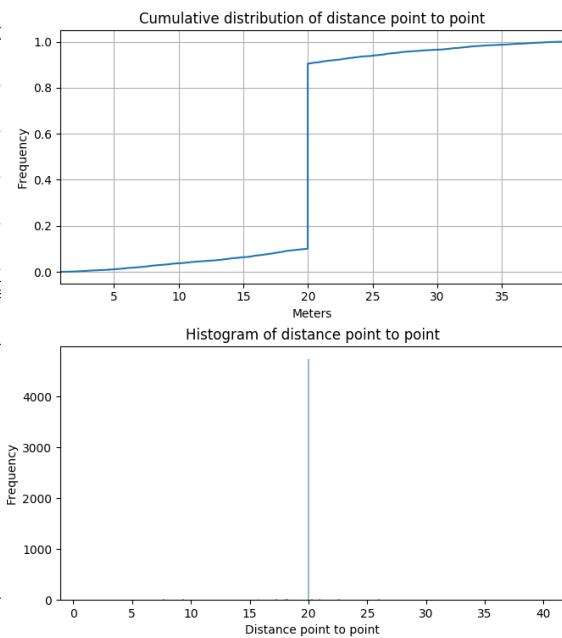
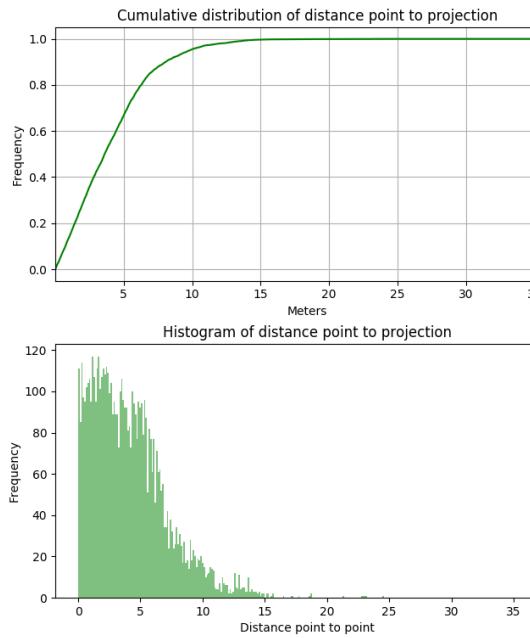


Figura 7.6: Muestra de análisis de simulación: Distancia punto a proyección en experimentación sin datos.

Figura 7.7: Muestra de análisis de simulación: Distancia punto a punto en experimentación sin datos.

La figura 7.11 corresponde al mapa de calor del análisis de las generaciones. Se puede observar una trayectoria más definida, que se corresponde con el mapa de calor mostrado anteriormente en la figura 7.1. Existen diversos aspectos que explican las diferencias entre los mapas de calor de las figuras 7.1 y 7.11. Las rutas simuladas tienen una distancia de 10 kilómetros, por lo que hacen inaccesibles ciertos caminos. Se aprecia que la parte norte y este del perímetro del bosque del Castell de Bellver es la zona más frecuente, que corresponde con las muestras reales. No obstante existen rutas que acceden a la parte sur del camino, con menos frecuencia, por lo que no son distinguibles. Las distribuciones corresponden. A nivel de distancia punto a punto se respeta completamente, en la distancia punto proyección encontramos una distribución normal mucho más acentuada.

## 7. EXPERIMENTACION

---

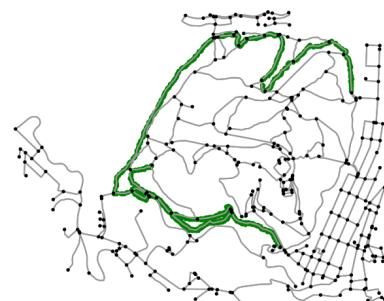


Figura 7.8: Muestra de simulación 1 Experimentación con datos).

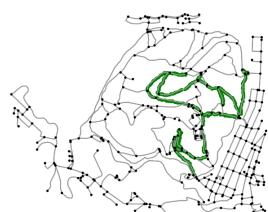


Figura 7.9: Muestra de simulación 2 Experimentación con datos).

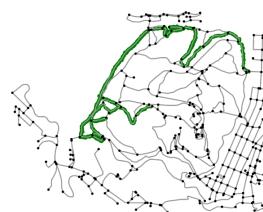


Figura 7.10: Muestra de simulación 3 (Experimentación con datos).

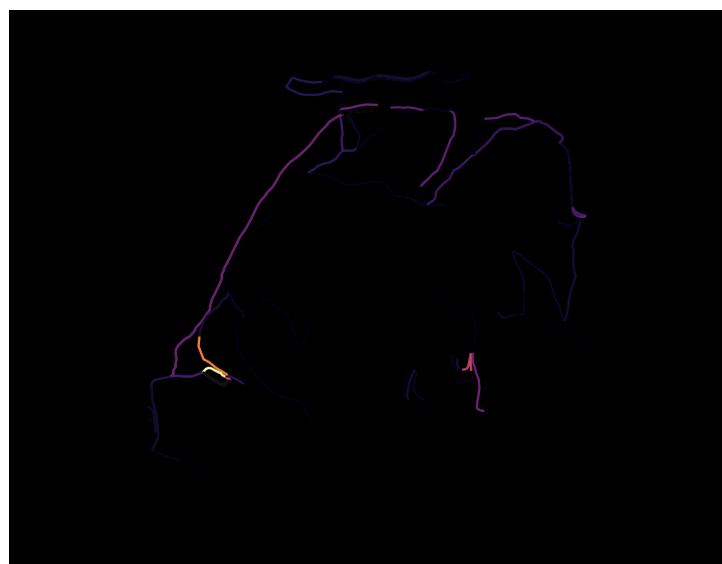


Figura 7.11: Mapa de calor del conjunto de rutas simuladas en experimentación con datos.

## 7.2. Experimentación con importación de datos reales

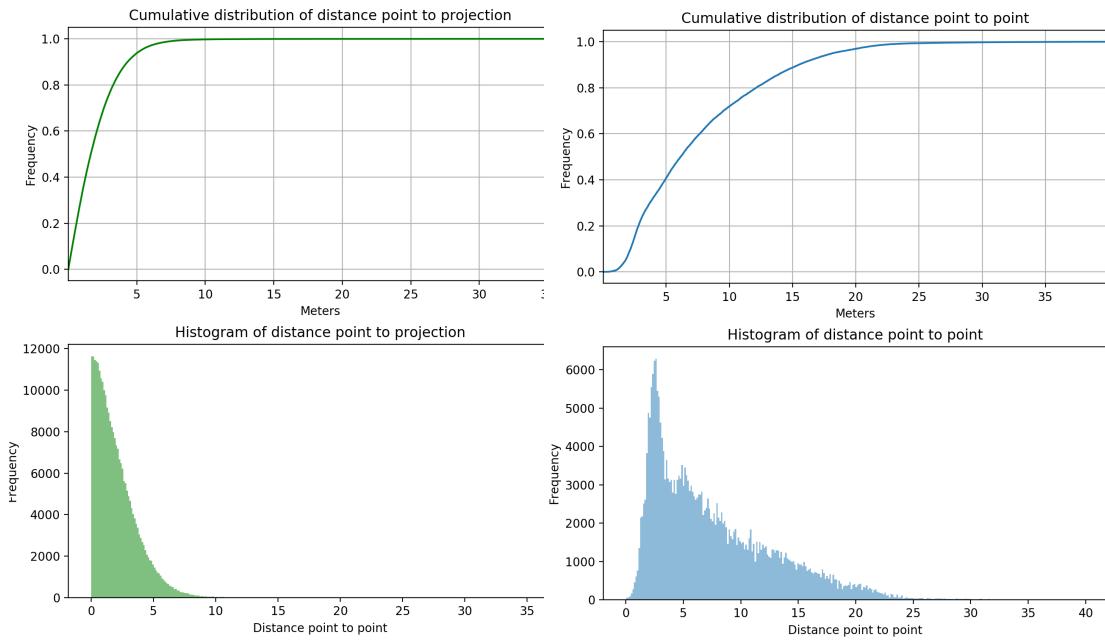


Figura 7.12: Muestra de análisis de simulación: Distancia punto a proyección en experimentación con datos.

Figura 7.13: Muestra de análisis de simulación: Distancia punto a punto en experimentación con datos.

### 7.2.1 Comparativa de resultados

En las figuras 7.14 y 7.15 se observan los resultados de la distancia punto a punto entre y la comparativa con la distribución real. Se observa que con el uso de los datos procedentes del análisis se obtiene una distribución probabilística similar a la distribución real, mientras que sin estos datos, aparece una distribución única al valor parametrizado, en este caso 20 metros.

## 7. EXPERIMENTACION

---

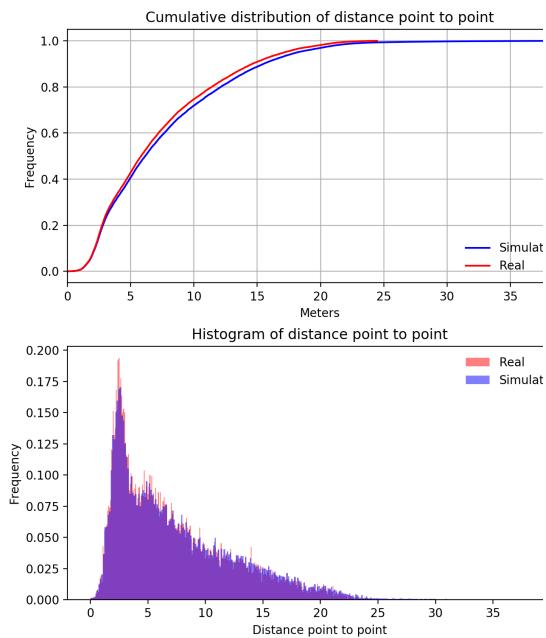


Figura 7.14: Comparativa análisis de la distancia punto a proyección en simulación con datos.

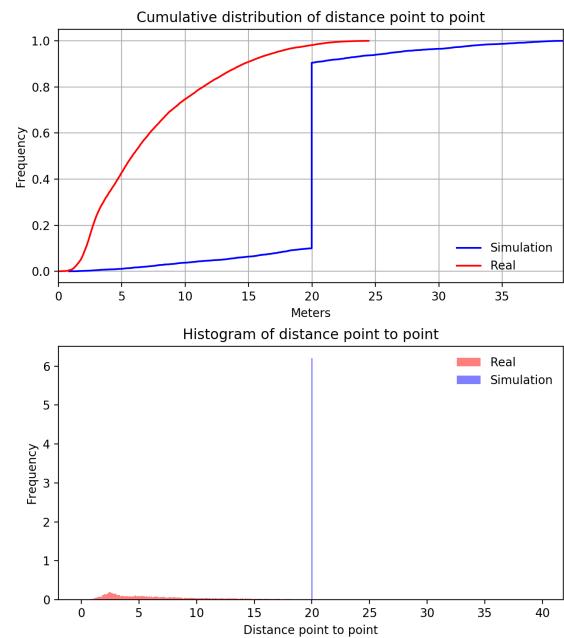


Figura 7.15: Comparativa análisis de la distancia punto a punto en simulación con datos.

Con esta información se aprecia que las simulaciones para este conjunto bajo de datos funciona con una alta similitud a las rutas reales proporcionadas.

### 7.2.2 Anexo: Muestra de simulaciones con datos introducidos



Figura 7.16: Simulación con carga de datos inicial. Ejemplo 1.

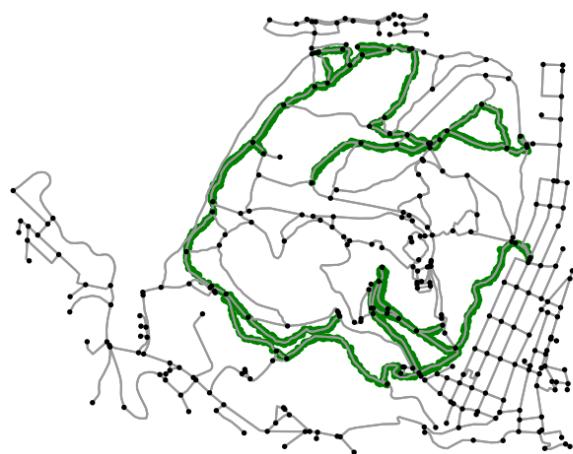


Figura 7.17: Simulación con carga de datos inicial. Ejemplo 2.

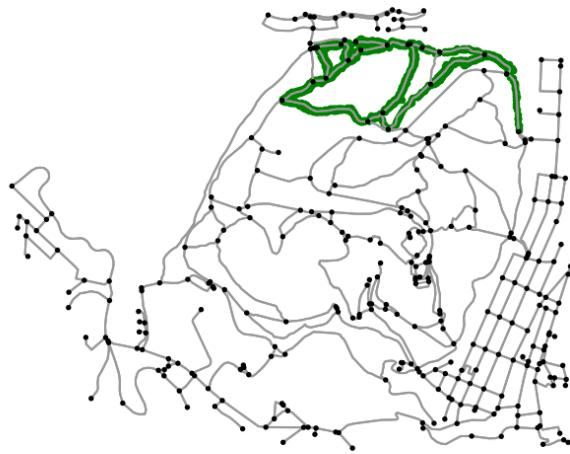


Figura 7.18: Simulación con carga de datos inicial. Ejemplo 3.

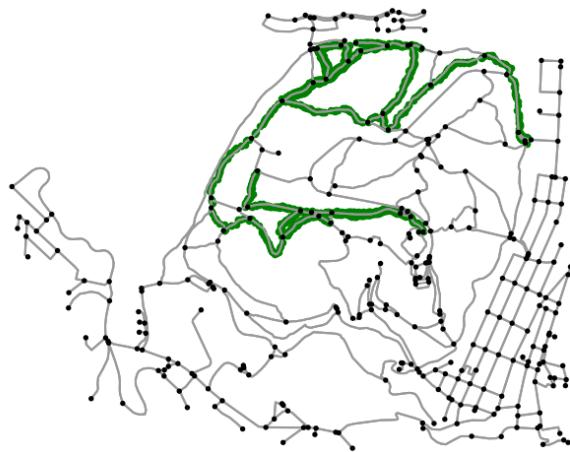


Figura 7.19: Simulación con carga de datos inicial. Ejemplo 4.

### 7.2.3 Anexo: Muestra de simulaciones sin datos

## 7.2. Experimentación con importación de datos reales

---

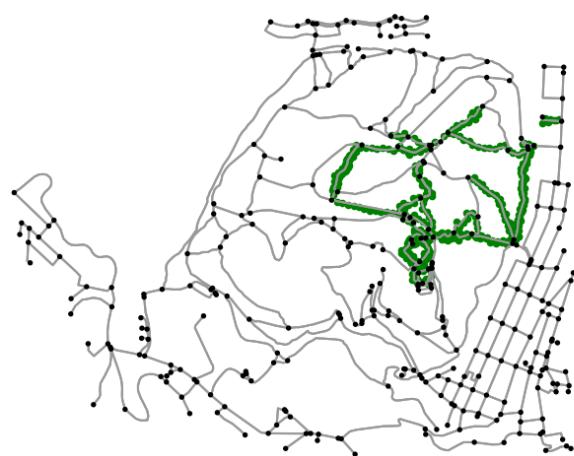


Figura 7.20: Simulación sin datos iniciales. Ejemplo 1.

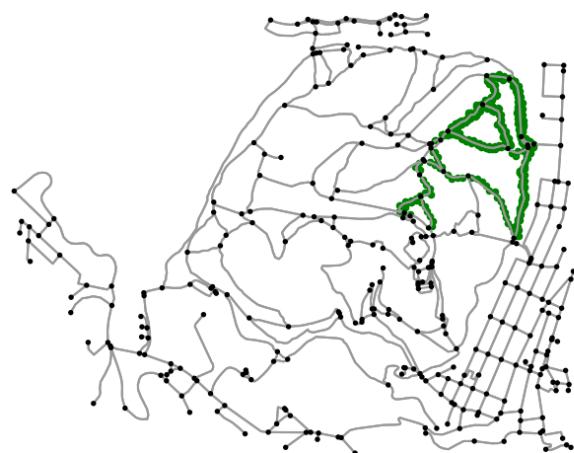


Figura 7.21: Simulación sin datos iniciales. Ejemplo 2.

## 7. EXPERIMENTACION

---

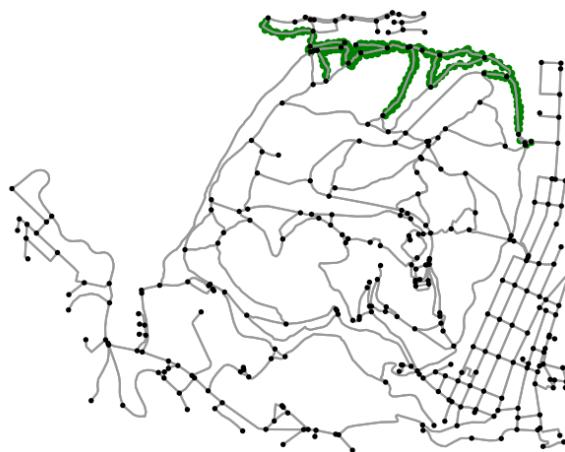


Figura 7.22: Simulación sin datos iniciales. Ejemplo 3.

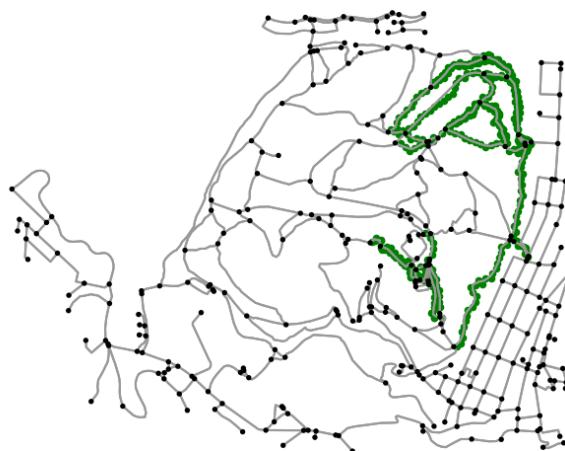


Figura 7.23: Simulación sin datos iniciales . Ejemplo 4.

CAPÍTULO

# 8

## GUÍA DE INSTALACIÓN Y USO

La propuesta de este documento tiene como aplicación resultante un CLI con un conjunto de acciones que pueden ser realizadas. Este aplicativo despliega un conjunto de contenedores de Docker para su funcionamiento independiente en cualquier máquina.

El despliegue se realiza en capas interiores de la aplicación por lo que el usuario no tiene que tener conocimientos previos de Docker para poder utilizar la herramienta. Se ha intentado facilitar el uso y comprensión de la aplicación mediante descripciones detalladas de cada uno de los parámetros en la propia herramienta, no obstante, en las siguientes secciones se detallarán cada una, como punto de inicio para que el lector pueda utilizar el aplicativo.

### 8.0.1 Instalación

La instalación se realiza a partir de un ejecutable. Este *script* realiza una serie de configuraciones del sistema, añadiendo un directorio al /Home de la máquina donde se instala. Se añade un directorio llamada *track-simulator* que será el lugar de entrada y la salida del aplicativo.

El directorio *track-simulator* contiene los siguientes subdirectorios:

- D.1. **config.** Directorio donde se sitúan los ficheros necesarios para la ejecución del aplicativo.
- D.2. **data.** Directorio para la salida de los procesos de análisis.
- D.3. **db.** Este directorio contiene la configuración de MongoDB dentro de nuestro sistema.
- D.4. **analysis.** Directorio de entrada para los ficheros a analizar, estos ficheros pueden estar organizados en subdirectorios. La ruta a estos subdirectorios será definida cuando realicemos una simulación mediante el comando *ts-cli analyze -file\_directory [SUBDIRECTORY]*

## 8. GUÍA DE INSTALACIÓN Y USO

---

D.5. **data.** Directorio para la salida de los procesos de análisis. Este directorio tiene, por lo tanto una estructura de subdirectorios como se muestra en la figura 8.0.1.

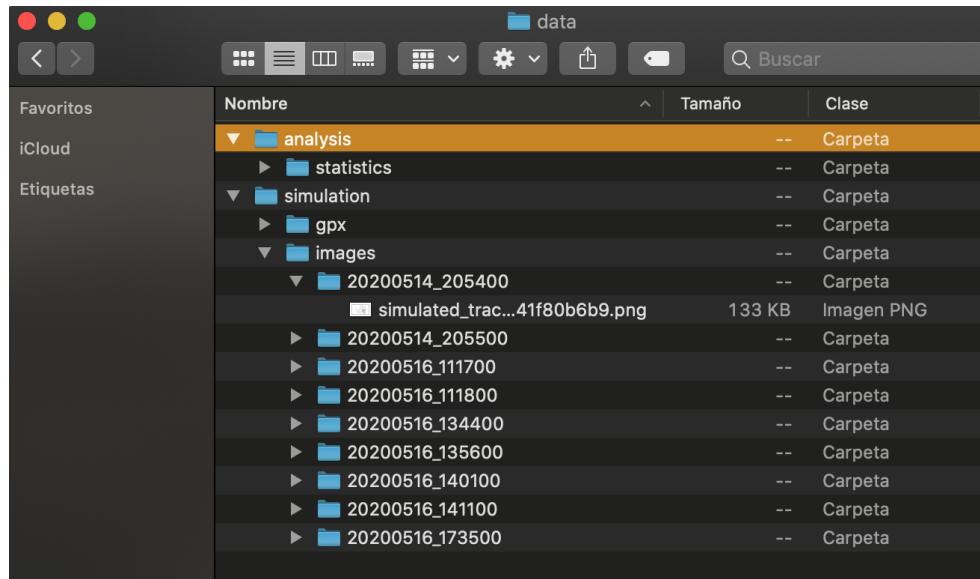


Figura 8.1: Jerarquía en el directorio */track-simulator/data*.

---

## 8.0.2 Guía de uso

La interfaz se despliega mediante el uso del comando *ts-cli*. Si no se le introduce ningún parámetro se desplegará la ayuda de forma que aparecen las acciones disponibles con una pequeña explicación. Las acciones disponibles en la primera versión de la herramienta son las siguientes:

- A.1. **simulate.** La acción simulate realiza una simulación de una trayectoria. Tiene una serie de parámetros de entrada que permiten la personalización de su comportamiento.

A.1.P1. **-distance** Distancia que se quiere simular (en metros).

A.1.P2. **-origin\_node** Nodo origen desde el que se desea simular. Los nodos a destacar para el inicio de la trayectoria pueden ser los nodos de entrada al recinto del castillo de Bellver, en este caso los siguientes nodos:

- i. 293027796: Entrada noreste.
- ii. 560055784: Entrada sureste.
- iii. 317813584: Entrada norte.

A.1.P3. **-data.** Conjunto de datos origen que se usarán para la simulación. (Por defecto se usará los datos con fecha más reciente.)

- A.2. **analyze.** La acción simulate realiza una simulación de una trayectoria. Tiene una serie de parámetros de entrada que permiten la personalización de su comportamiento.

A.2.P1. **file\_directory** Directorio de donde se realizará la lectura de los ficheros GPX para su análisis. Parte de una ruta inicial /data/analysis.

A.2.P2. **data.** Conjunto de datos origen que se usarán para la simulación. (Por defecto se usará los datos con fecha más reciente). Los datos de MongoDB se guardan con formato Graph\_Analysis\_mm-dd-YYYY.

Se puede utilizar el comando 8.1 para observar la lista de para observar la lista de parámetros de una acción en concreto.

```
1 ts-cli analyze [COMMAND] --help
```

Algoritmo 8.1: Ejecución ayuda de comando

En el comando 8.2 se tiene un ejemplo de utilización de la herramienta para analizar una conjuntos de trayectorias situadas en la carpeta *01-bellver*.

```
1 ts-cli analyze --file_directory 01-bellver
```

Algoritmo 8.2: Ejecución análisis

Cabe destacar que la carpeta *01-bellver* tiene que estar situada en el directorio *\$HOME/track-simulator/analysis*. Como resultado de esta ejecución se tendría un conjunto de datos almacenados dentro de colecciones de la base datos. Junto a esto, se podría observar resultados en la carpeta */data/analysis/statistics* como muestra la figura 8.2.

## 8. GUÍA DE INSTALACIÓN Y USO

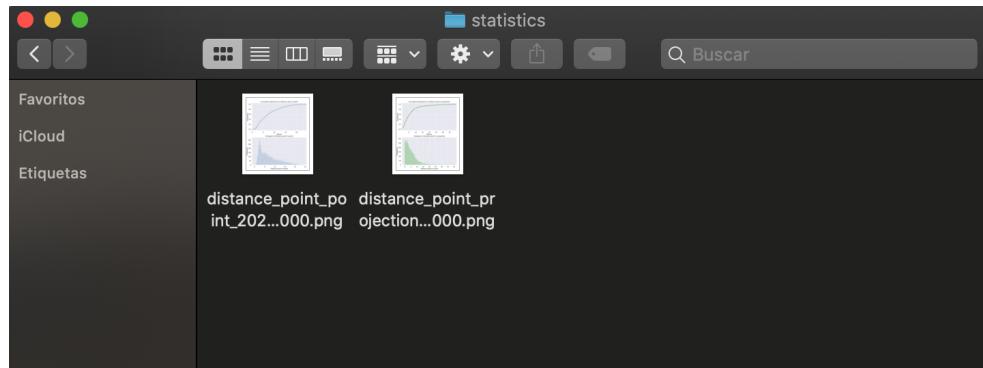


Figura 8.2: Directorio *statistics* tras el análisis de un conjunto de trayectorias.

Por otro lado, el comando 8.3 representa un ejemplo de utilización de la herramienta para simular una trayectoria de 10 Km a partir del análisis realizado día 16 de Mayo de 2020.

```
1 ts-cli simulate --distance 10000 --data Graph_Analysis_05-16-2020
```

Algoritmo 8.3: Ejecución simulación

Se tiene como resultados de la ejecución un conjunto de trayectorias, tanto ficheros GPX como una imágenes representativa en formato PNG serán almacenadas en la ruta *data/simulation* como muestra la figura 8.3.

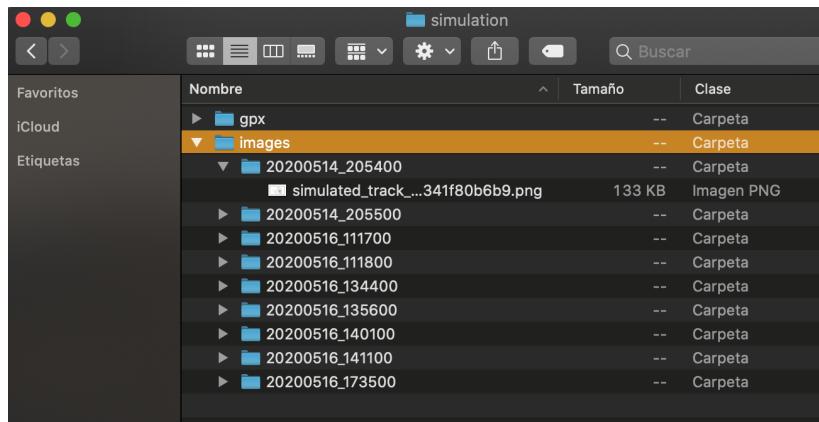


Figura 8.3: Directorio *simulation* tras la simulación de un conjunto de trayectorias.

---

Se entiende que esta aplicación puede ser utilizada por un público que no tiene experiencia en gestión de contenedores Docker, por este motivo, para eliminar los recursos utilizados para el procedimiento, por cada uso de la herramienta se despliegan y apagan los componentes. Esto supone un tiempo mayor de ejecución, no obstante se cree necesario para evitar que los recursos no sean cerrados una vez las necesidades del usuario de la aplicación se vean completas.



CAPÍTULO

# 9

## CONCLUSIONES

Este documento se ha desarrollado una propuesta de una CLI desarrollado en el lenguaje Python para el análisis y generación de trayectorias pseudo-aleatorias.

El objetivo principal de la propuesta es dar una posible solución al problema de la generación de trayectorias dentro de un terreno geoespacial. Para ello se ha realiza un trabajo de tratamiento y explotación de trayectorias reales con el objectivo de poder tener un algoritmo que pueda replicar el comportamiento de la muestra dentro del mismo terreno. Todo el flujo de datos en esta propuesta se realiza en base a ficheros en formato GPX.

Se tiene como entrada del proceso un conjunto de muestras de trayectorias reales realizadas sobre un territorio y se tiene como salida del proceso un análisis de esta muestra y un modelo con la información cargada para ser explotada en forma de generación de nuevas trayectorias.

El desarrollo del documento ha querido cubrir todas las etapas del proceso de desarrollo de la propuesta. El lector ha podido realizar una lectura en la que es introducido en los términos técnicos y herramientas utilizadas en la tipología del problema capítulo 3. Posteriormente se ha explicado de qué elementos consta la propuesta, tanto a nivel de flujo de datos como de funcionalidades e implementaciones (capítulo 4). Una de las piezas clave de esta propuesta ha sido la explicación de la aproximación al análisis de trayectorias (figura 3.1).

## 9. CONCLUSIONES

---



Figura 9.1: Muestra de resultado del proceso de map-matching .

Las heurísticas y metodologías utilizadas han sido explicadas de forma detallada en el capítulo 5. Esta explicación de tratamiento y explotación del dato se ha completado con el segundo modulo importante de esta propuesta, la simulación. Cómo se generan las trayectorias y cada uno de los puntos de forma pseudo-aleatoria, tanto a nivel conceptual como de implementación ha sido detallado en el capítulo 6. Posterior a toda la expliación teórica de la solución propuesta, se han demostrado los resultados que se pueden obtener con la herramienta. En el capítulo 7 se tratan las experimentaciones, tanto con carga como sin carga de datos, para su posterior comparaición y análisis. Finalmente, como cierre conceptual de la propuesta, el capítulo 3.1 define al lector los pasos necesarios para la instalación y uso del aplicativo.

La propuesta ha tenido que ser trabajada sobre una muestra pequeña de datos. Este hecho ha limitado en gran parte el alcance de la propuesta. Esto ha sido ventajoso en algunos puntos del desarrollo debido a la reducción de los tiempos de procesamiento. No obstante, trabajar con un mayor volumen de datos puede hacer que el modelo sea mucho más preciso. Por este motivo, uno de los conceptos transversales que han sido tratados en esta propuesta ha sido el desarrollo del aplicativo con una estructura que modular, que pueda ser ampliable y escalable.

A nivel técnico, la forma en la que ha sido desarrollado el proyecto ha sufrido cambios constantes con el objetivo de realizar una infraestructura de proyecto modular, que permita ser testeable unitariamente y abstracta. La propuesta explicada en este documento pretende ser un inicio de un aplicativo con un alcance mucho mayor, ajustable y con una estructura que apoye la mejora continua del proyecto por parte de la comunidad.

## 9.1 Futuro trabajo

La cantidad de dato GPS que se genera aumenta con el paso del tiempo. Las aplicaciones que hacen uso de estos datos y permiten que sean explotados aportan un gran avance a la comunidad. El análisis del dato que se ha realizado en esta propuesta tiene en cuenta el posicionamiento en latitud y longitud de cada uno de los puntos, no obstante, los ficheros con formato GPX pueden ofrecer una gran cantidad de información que puede ser añadida al modelo para generar trayectorias mucho más complejas y precisas. Marcas de tiempo, elevación, o la pulsación del usuario son factores que se pueden tener en cuenta para determinar como una ruta es generada y llegar a replicar trayectorias de una forma mucho más completa.

El hecho que esta propuesta se base en el análisis de puntos GPS en base a una red de caminos y carreteras hace que su utilidad aumente en territorios urbanos. El análisis de una cantidad elevada de datos (orden del millar) puede aportar una visión de comportamiento de los usuarios dentro de un territorio.

Se entiende por lo tanto, que el núcleo teórico fundamental de la simulación de trayectorias está en la explotación del dato GPS. Para ello, existen dos grandes vertientes para el trabajo futuro en esta herramienta, por un lado la expansión en la explotación del dato, añadiendo a la heurística más variables y, por otro lado, la expansión en cantidad del dato, aumentando el volumen de los datos y almacenando este en plataformas cloud que permitan llevar la evolución de esta herramienta a un siguiente nivel.

## 9.2 Opinión personal

TrackSimulator es el proyecto personal más ambicioso al que me he enfrentado hasta la fecha. El planteamiento de esta problemática me pareció interesante desde el inicio. Siempre he sentido atracción por el dato y por el valor que puede aportar explotarlo.

Con este proyecto he podido centrar mi esfuerzo en aprender cosas que durante la carrera no había tenido oportunidad. Ha sido el punto de partida para adquirir una base de conocimiento para el tratamiento de datos y desarrollo de software. Con este proyecto he podido entrar en contacto con todo un conjunto de herramientas de análisis de datos que me han hecho estar constantemente buscando alternativas y mejoras. He podido disfrutar del proceso de aprendizaje a lo largo de sus diferentes etapas, lo que me ha hecho tomar la decisión de orientar mi carrera a los datos.

Algunas partes del desarrollo de este proyecto han sido un gran reto. He tratado este proyecto como algo más que un trabajo universitario de final de carrera, mi planteamiento ha sido dejar una base técnica modular, que pueda ser perfeccionada y que permita ser mejorada tanto por mi parte como por cualquier usuario de la comunidad.

Considero que este proyecto tiene un gran potencial, añadir variabilidad al análisis, junto con la utilización de recursos que permitan el procesamiento de grandes volúmenes de datos puede hacer de esta herramienta un proyecto valioso para la comunidad que trata con la información GPS.

Uno de los grandes objetivos de este proyecto es que esta herramienta sea ampliable con facilidad. Esto ha hecho que me haya interesado en conceptos transversales que han sido aplicados en este desarrollo. He aprendido técnicas de *clean coding* y *clean architecture* que, junto con conceptos de desarrollo con contenedores, me han

## **9. CONCLUSIONES**

---

ayudado a apreciar el valor de crear código de alta mantenibilidad. Estos conceptos los encuentro de gran ayuda para mis capacidades técnicas dentro del desarrollo de software productivo y estoy agradecido de haberlos podido aprender y aplicar en este proyecto.

Estoy francamente satisfecho del resultado final del proyecto. Encuentro que existen dos productos resultantes: por un lado el gran impulso a mi inquietud y conocimiento, que me ha permitido encontrar un camino que me apasiona y decidir el camino inicial de mi desarrollo profesional. Por otro lado aporto una herramienta que puede servir de base para el desarrollo de nuevas utilidades y que seguiré mejorando con los conocimientos y experiencia que adquiera.

## BIBLIOGRAFÍA

- [1] ArcGis, “What is raster data?” [Online]. Available: <https://desktop.arcgis.com/es/arcmap/10.3/manage-data/raster-and-images/what-is-raster-data.html> (document), 3.3
- [2] J. Fong, “Are containers replacing virtual machines?” 2018. (document), 4.5
- [3] Gpsvisualizer, “Generating a map from a gpx file, dynamically.” [Online]. Available: [https://www.gpsvisualizer.com/examples/google\\_gpx.html](https://www.gpsvisualizer.com/examples/google_gpx.html) (document), 34
- [4] N. P. Borkovich, D., “Big data in the information age: Exploring the intellectual foundation of communication theory,” 2014. [Online]. Available: <https://files.eric.ed.gov/fulltext/EJ1140800.pdf> 2
- [5] GPS.gov, “Gps applications,” 2014. [Online]. Available: <https://www.gps.gov/applications> 2
- [6] I. Viskic, “Enhancing the quality of uber’s maps with metrics computation,” 2018. [Online]. Available: <https://eng.uber.com/maps-metrics-computation> 2
- [7] G. Boeing, “Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks,” *Computers Environment and Urban Systems*, pp. 126 –139, 2017. 2
- [8] P. development team, “Pandas documentation,” 2008-2014. [Online]. Available: <https://pandas.pydata.org/docs> 2
- [9] T. Krajina, “Gpxpy. project description,” 2012. [Online]. Available: <https://pypi.org/project/gpxpy/> 2
- [10] N. Garrido-Villén, “Sistemas gnss. introducción a los sistemas de posicionamiento global (4),” 2014. [Online]. Available: <https://nagarvil.webs.upv.es/sistemas-gnss-introduccion> 3.1
- [11] R. B. Langley, “The mathematics of gps,” vol. 2, pp. 45 – 50, July/August 1991. 3.1
- [12] A. El-Rabbany, *Introduction to GPS: The Global Positioning System.* Artech House Publishers, 2002. 3.1
- [13] U. S. E. P. Agency, *Geographic information Systems Guidelines document*, 1988. 3.1.1

## BIBLIOGRAFÍA

---

- [14] A. Morales, “Los formatos gis ráster más populares,” 2010. [Online]. Available: <http://mappinggis.com/2015/12/los-formatos-gis-raster-mas-populares> 3.1.2
- [15] O. G. Consortium, “Ogc® geography markup language (gml) — extended schemas and encoding rules,” 2012. [Online]. Available: <http://www.opengis.net/spec/GML/3.3> 3.1.2
- [16] ———, “Kml 2.1 reference – an ogc best practice,” *Carl Reed on behalf of Google Earth staff*, 2007. 3.1.2
- [17] Topografix, “Gpx: the gps exchange format,” 2010. [Online]. Available: <https://www.topografix.com/gpx.asp> 3.1.2
- [18] “Advantatges of nosql.” [Online]. Available: <https://www.mongodb.com/scale/advantages-of-nosql> 4.4.2
- [19] D. Inc., “Docker docs,” 2013-2020. [Online]. Available: <https://docs.docker.com> 4.4.3
- [20] H. S. Malvar, “Effective communication: Tips on technical writing,” *IEEE Signal Processing Magazine*, vol. 25, no. 3, pp. 129–132, 2008. 5.2.1
- [21] W.-J. H. Yu-Ling H., Ho-Chian C., “A hidden markov model-based map-matching approach for low-sampling-rate gps trajectories,” *IEEE*, pp. 120 –142, November 2017. 5.2.3, 5.2.3
- [22] M. T. Ltd, “Gis faq q5.1: Great circle distance between 2 points,” 2012. [Online]. Available: <https://www.movable-type.co.uk/scripts/gis-faq-5.1.html> 5.3.1
- [23] R. Sinnott, “Virtues of the haversine,” *Sky and Telescope*, p. p. 159, November 1984. 5.3.1
- [24] K. Sigman, “Inverse transform method,” 2010. [Online]. Available: <http://www.columbia.edu/~ks20/4404-Sigman/4404-Notes-ITM.pdf> 6.3