



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, 26 de abril de 2020

ÍNDICE GENERAL

Índice general	i
Índice de figuras	iii
Acrónimos	vii
1 Abstract	1
2 Introducción	3
3 Conceptos principales	5
3.1 Sistemas de navegación por satélite	5
3.1.1 GIS	5
3.1.2 Fuentes de datos	6
3.2 Estructura de los ficheros GPS eXchange Format (GPX)	7
3.3 Open Street Map	9
4 Arquitectura de la aplicación	11
4.1 <i>TrackSimulator</i>	11
4.2 Funcionalidades de <i>TrackSimulator</i>	13
4.3 Requerimientos de <i>TrackSimulator</i>	14
4.4 Implementación de TrackSimulation	15
4.4.1 Flujo de datos	15
4.4.2 Herramientas externas utilizadas	16
5 Análisis de los datos Geospatial Position System (GPS)	19
5.1 Preparación del entorno de datos	19
5.1.1 Importación de datos GPS a <i>TrackSimulator</i>	19
5.1.2 Importación de la red compleja de datos	20
5.2 Map Matching	20
5.2.1 Hidden Markov Model (HMM)	21
5.2.2 Aproximación HMM a Map-matching	21
5.2.3 Algoritmo de Viterbi	22
5.2.4 Ejemplo de aplicación del algoritmo de Viterbi	23
5.2.5 Implementación de Map-matching	25
5.3 Explotación del dato	25
5.3.1 Análisis de distancias	25
5.3.2 Demostración de análisis	27

6 Simulación de trayectorias	29
6.1 Generación de un camino	29
6.2 Generación de los segmentos	30
6.3 Generación del punto pseudo-aleatorio	31
6.4 Exportación de trayectoria a fichero .GPX	31
7 Experimentacion	33
7.1 Experimentación sin importación de datos reales	33
7.2 Experimentación con importación de datos reales	33
8 Guía de instalación y uso	35
9 Conclusiones	37
9.1 Posibles mejoras	37

ÍNDICE DE FIGURAS

3.1 Ejemplo de la aplicacion ArcGis	6
3.2 Ejemplo de la aplicacion QGIS	6
3.3 Ejemplo de territorio en formato ráster. [?]	7
3.4 Ejemplo de Ruta del Puig de l'Ofre, Mallorca, Illes Balears, España.	8
3.5 Ejemplo de código de un fichero .GPX	8
3.6 Ejemplo de muestra de OSM de las Illes Balears, España.	9
4.1 Ejemplo de detección real de la trayectoria de un individuo por la Serra de Tramuntana, Illes Balears, Mallorca, Spain	11
4.2 Diagrama de funcionamiento de TrackSimulator	12
4.3 Diagrama de funcionamiento de TrackSimulator	14
4.4 Flujo de datos de TrackSimulator	15
5.1 Estructura de un <i>waypoint</i> dentro de un fichero .GPX	19
5.2 Ejemplo de estructura LineString	22
5.3 Ejemplo de aplicación de algortimo de Viterbi para detección de trayectorias.	23
5.4 Ejemplo de aplicación de algortimo de Viterbi para detección de trayectorias.	24
5.5 Ejemplo de aplicación de algortimo de Viterbi para detección de trayectorias.	24
5.6 Muestra de análisis: Distancia punto a proyección.	27
5.7 Muestra de análisis: Distancia punto a punto.	27
6.1 Generación de camino.	29
6.2 Generación de puntos.	29
6.3 Ejemplo de generación de punto a partir de una distancia d y un ángulo α .	31
6.4 Ejemplo de detección de proyección cercana y generación de punto.	31

LISTINGS

5.1	Implementación interactor get_map_matching_impl	25
5.2	Declaración interfaz hidden_markov_model	25
5.3	Método add_next_point_distance	26
5.4	Método add_point_projection_distance	26
6.1	Método create_path	30
6.2	Método simulate_segment	30

ACRÓNIMOS

GNSS Sistema Global de Navegación por Satélite

GPS Geospatial Position System

GIS Geospatial Information System

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

CAPÍTULO

1

ABSTRACT

CAPÍTULO



INTRODUCCIÓN

En este capítulo se realizará una introducción del documento.

CAPÍTULO

3

CONCEPTOS PRINCIPALES

En esta sección se pretenden introducir los conceptos básicos referentes al tipo de información, herramientas y formatos de datos geográficos con los que se trabaja durante toda la propuesta.

3.1 Sistemas de navegación por satélite

Los Sistema Global de Navegación por Satélite (GNSS) son mecanismos formados por una red de satélites utilizados para situar elementos en la superficie terrestre. Destacan los sistemas GALILEO, GLONASS y GPS, de origen Europeo, Ruso y Americano respectivamente, siendo este último el que explicaremos con más detalle. 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". [?] Esta información tiene una precisión de 100 y 156 metros para la componente horizontal y vertical, respectivamente al 95 % de probabilidad [?].

3.1.1 GIS

Los Sistemas de Información Geográfica, Geospatial Information System (GIS), son aplicaciones que permiten almacenar, manipular y presentar 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.

3. CONCEPTOS PRINCIPALES

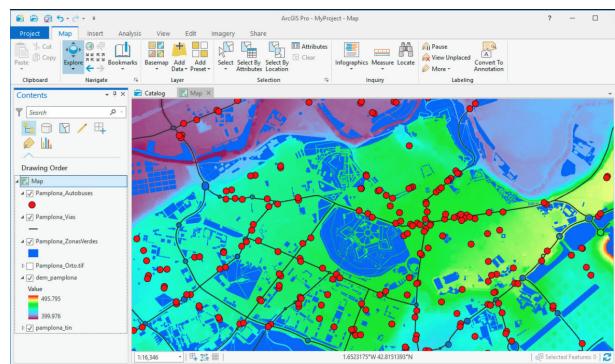


Figura 3.1: Ejemplo de la aplicacion ArcGis

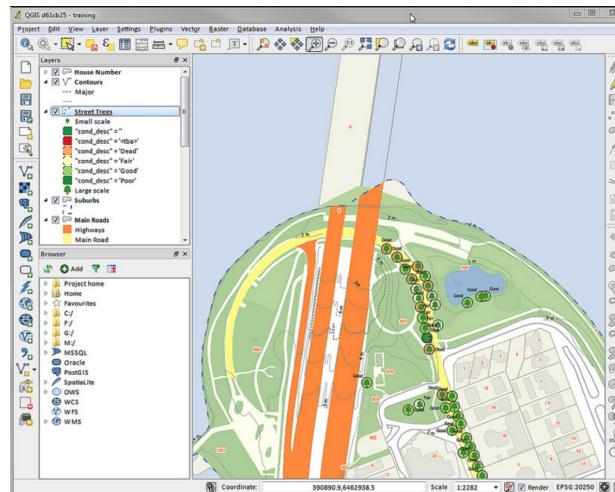


Figura 3.2: Ejemplo de la aplicacion QGIS

3.1.2 Fuentes de datos

Estas herramientas necesitan que los datos obtenidos por GPS sigan un formato compatible. Estas herramientas soportan una gran diversidad de formatos que se dividen en ráster y Vectoriales.

Formato ráster

La representación de los datos se hace a partir de píxeles regulares de valor único. La geografía de un espacio queda descrita en una matriz en la que cada cuadrícula almacena la información como altitud o superficie. Entre los formatos ráster más populares estan Esri Grid, GeoTIFF, JPEG2000 [?].

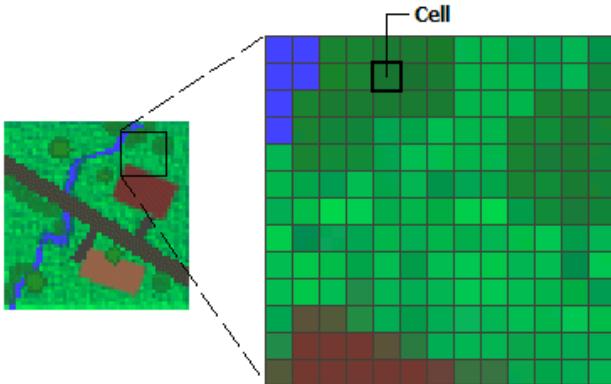


Figura 3.3: Ejemplo de territorio en formato ráster. [?]

Formato vectorial

los formatos vectoriales se basan en la representación mediante puntos, líneas y polígonos. De este segundo grupo destacamos los siguientes:

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

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 [?].

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 [?].

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.

Route (rteType) Lista ordenada de Waypoints. Representan una sucesión de puntos hacia un destino. Contiene una secuencia de elementos descriptivos de carácter opcional.

3. CONCEPTOS PRINCIPALES

Tracks (trkType) Lista ordenada de puntos que escriben un camino. Contiene el mismo tipo de secuencia de elementos descriptivos de carácter opcional.

Podemos ver en la siguiente figura un ejemplo de una ruta de una ruta de senderismo por la Serra de Tramuntana de Mallorca.

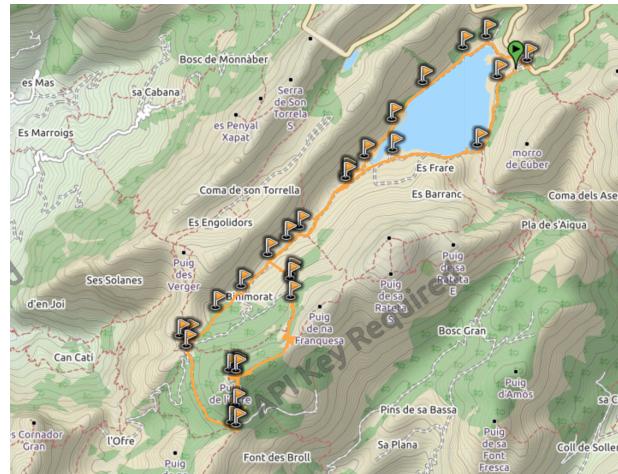


Figura 3.4: Ejemplo de Ruta del Puig de l'Ofre, Mallorca, Illes Balears, España.

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx creator="Garmin Connect" version="1.1"
  xmlns:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/11.xsd"
  xmlns:ns3="http://www.garmin.com/xmlschemas/TrackPointExtension/v1"
  xmlns="http://www.topografix.com/GPX/1/1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ns2="http://www.garmin.com/xmlschemas/GpxExtensions/v3">
<metadata>
  <link href="connect.garmin.com">
    <text>Garmin Connect</text>
  </link>
  <time>2019-04-23T17:53:09.000Z</time>
</metadata>
<trk>
  <name>Cuestas - Trote</name>
  <type>trail_running</type>
  <trkseg>
    <trkpt lat="39.56717558205127716064453125" lon="2.6237960346043109893798828125">
      <ele>30.20000762939453125</ele>
      <time>2019-04-23T17:53:09.000Z</time>
    <extensions>
      <ns3:TrackPointExtension>
        <ns3:atemp>26.0</ns3:atemp>
        <ns3:hr>86</ns3:hr>
        <ns3:cad>79</ns3:cad>
      </ns3:TrackPointExtension>
    </extensions>
    </trkpt>
    <trkpt lat="39.56718388013541698455810546875" lon="2.62377818115055561065673828125">
      <ele>30.20000762939453125</ele>
      <time>2019-04-23T17:53:10.000Z</time>
    <extensions>
      <ns3:TrackPointExtension>
        <ns3:atemp>26.0</ns3:atemp>
        <ns3:hr>86</ns3:hr>
        <ns3:cad>79</ns3:cad>
      </ns3:TrackPointExtension>
    </extensions>
    </trkpt>
    <trkpt lat="39.56719695590436458587646484375" lon="2.62376007623970508575439453125">
      <ele>29</ele>
      <time>2019-04-23T17:53:11.000Z</time>
    <extensions>
      <ns3:TrackPointExtension>
```

Figura 3.5: Ejemplo de código de un fichero .GPX

3.3. Open Street Map

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.

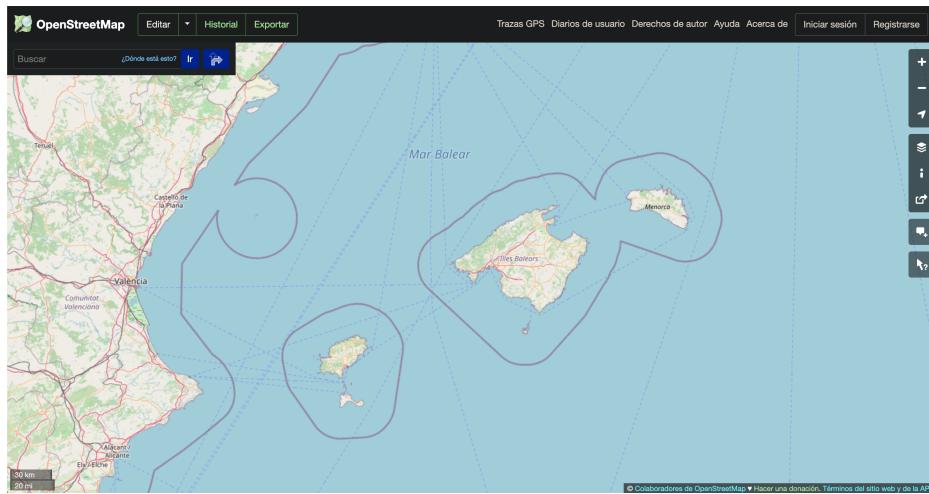


Figura 3.6: Ejemplo de muestra de OSM de las Illes Balears, España.

ARQUITECTURA DE LA APLICACIÓN

En esta sección presentamos la arquitectura de la aplicación. Realizaremos una descripción de las funcionalidades que presenta y de los módulos de los que consta.

4.1 *TrackSimulator*

TrackSimulator es un generador pseudo-aleatorio de trayectorias geoposicionales.

Entendemos 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 X se muestra un ejemplo de una detección del recorrido de un individuo.

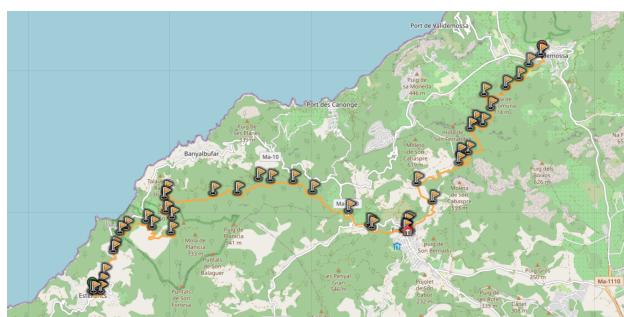


Figura 4.1: Ejemplo de detección real de la trayectoria de un individuo por la Serra de Tramuntana, Illes Balears, Mallorca, Spain

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 detección del dispositivo

4. ARQUITECTURA DE LA APLICACIÓN

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 fidelidad a 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. Este problema tiene 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.

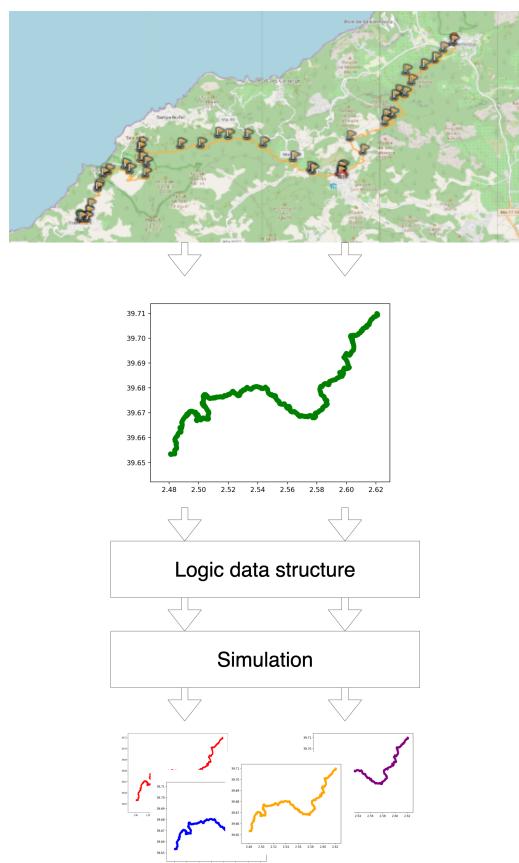


Figura 4.2: Diagrama de funcionamiento de TrackSimulator

La aplicación crea una sucesión de puntos que equivalen a un recorrido dentro de un plano geográfico. Se ha determinado que la aplicación realiza la generación de

forma pseudo-aleatoria debido a que la producción de los diferentes puntos no sigue ningún patrón o regularidad.

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.

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. La aplicación permite:

Importación y análisis de archivos GPX La aplicación soportará la importación de un archivo GPX concreto, o bien la ruta de una carpeta con diversos archivos GPX. Estos archivos GPX deben contener trayectorias realizadas en el espacio delimitado.

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 Como la distancia entre las capturas de posición GPS en el fichero.

Distancia relativa entre punto de ruta y punto de trayectoria Como la distancia del punto GPS detectado de la trayectoria a la punto proyección dentro de la ruta.

Frecuencia de paso por segmento de ruta Como la frecuencia de paso por el segmento x_a, x_b relativo a todos los segmentos desde x_a .

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 impresión por pantalla de información gráfica de los resultados del análisis.

Creación de una trayectoria a partir de parámetros Con los resultados de la información analizada se puede realizar una simulación de trayectorias dentro del espacio geográfico.

Exportación de trayectorias a fichero GPX La aplicación permite realizar una exportación de las trayectorias simuladas en formato GPX.

Visualización de la trayectoria La aplicación mostrará una representación gráfica de la trayectoria tanto analizada como simulada dentro del territorio geográfico.

4. ARQUITECTURA DE LA APLICACIÓN

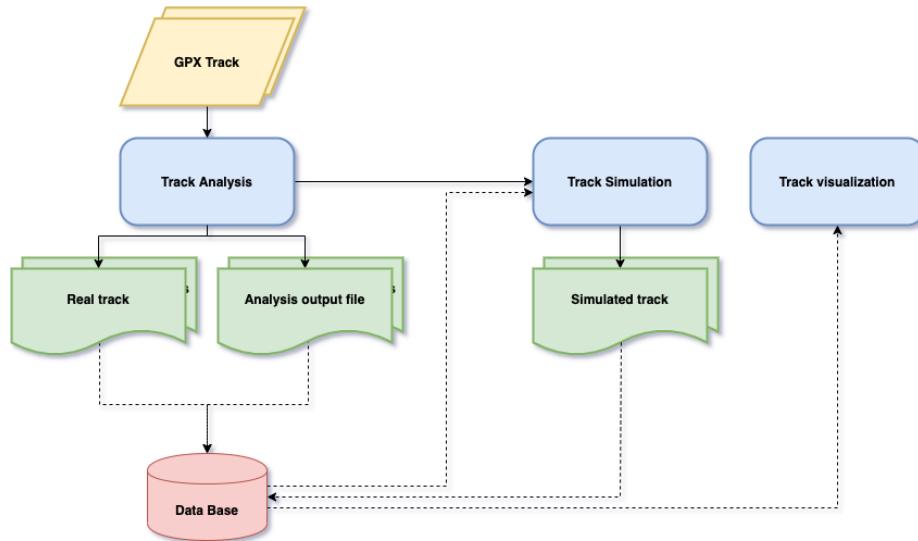


Figura 4.3: Diagrama de funcionamiento de TrackSimulator

4.3 Requerimientos de *TrackSimulator*

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

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.

Análisis de ruta La aplicación debe realizar un análisis de la ruta que proporcione por una parte indicadores y por otro valores medibles, cuantificables y utilizables para realizar una simulación lo más precisa posible.

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.

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.

Muestra de gráficas del análisis de los datos La aplicación debe poder realizar una muestra de la información resultante para su visualización por parte del usuario.

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.

Obtención del camino más probable a partir de parámetros de entrada La aplicación debe poder generar el camino más probable dada un análisis previo y unos parámetros de entrada.

Simulación de puntos GPS a partir de resultado de análisis La aplicación debe generar los puntos de cada uno de los segmentos necesarios dado un camino.

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

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

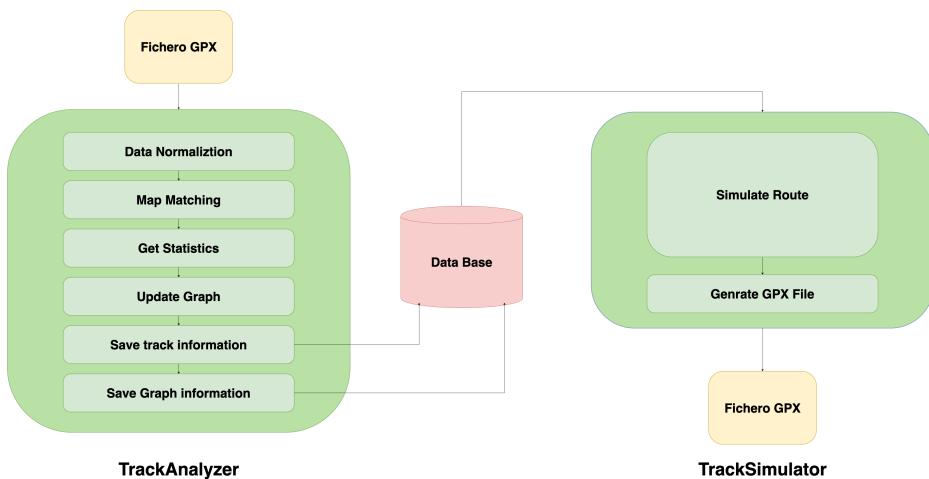


Figura 4.4: Flujo de datos de TrackSimulator

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 ma-

4. ARQUITECTURA DE LA APLICACIÓN

peadas, 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). Las ventajas de las NO-SQL-DB han hecho que con el paso del tiempo, los desarrollos se vean orientados al tipo de almacenamiento no relacional. 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 JSON clave-valor.

MongoDB elimina las limitaciones de las bases de datos relacionales [?]. 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, la aplicación respondería de forma eficiente y mucho más precisa.

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

Para la realización de las funcionalidades comentadas en la parte anterior se ha hecho uso de las siguientes librerías externas:

gpypy Librería para la manipulación de ficheros GPX. Mediante esta librería se puede realizar una importación del fichero para su posterior manipulación. De esta

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

ANÁLISIS DE LOS DATOS GPS

5.1 Preparación del entorno de datos

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*. El primer paso a realizar es el parseo del fichero. De esta forma encontramos un objeto track, que contiene segmentos que a su vez contienen los diferentes waypoints. De los que hemos hablado anteriormente. Los waypoints tienen la estructura que podemos ver en esta imagen:

```
<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 *waypoint* 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 resto de información no será usada para el desarrollo de la propuesta, queda una estructura en forma de lista del siguiente tipo de elemento.

5. ANÁLISIS DE LOS DATOS GPS

Punto preprocesado		
Campo	Tipo	Descripción
longitud	Float	Posición del nodo respecto al eje horizontal.
latitud	Float	Posición del nodo respecto al eje vertical.
Point(longitud, latitud)	Point	Estructura de punto para tratamiento interno.

Cuadro 5.1: Estructura lista de puntos.

5.1.2 Importación de la red compleja de datos

Como apunta G. Boeing en su artículo [?], para abordar correctamente la problemática del análisis de rutas dentro del espacio urbano se debe plantear con 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 sección anterior. Mediante el uso de un corto número de instrucciones se consigue importar toda la estructura de la zona del Castell de Bellver.

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 el la trayectoria a simular no tiene en cuenta el sentido de la carretera. Por otra parte, añadiremos 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) está definido 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 de la siguiente forma una vez aplicado el procesado:

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.

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. Los puntos GPS se

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

Cuadro 5.3: Estructura arista.

transmiten a partir de un dispositivo. Este dispositivo administra la serie de puntos GPS correspondientes a una trayectoria con un cierto error y desviación. Esta serie de puntos son puntos aislados y no tienen relación directa con el modelo lógico establecido. Definiremos como **Map-matching** al proceso que nos permite asignar las coordenadas GPS a un modelo lógico establecido. Existen diversos procedimientos para abordar el problema del *Map-matching*. Nuestra aproximación al problema la realizamos aplicando el modelo probabilístico de las cadenas de Markov, que explicamos en el siguiente apartado.

5.2.1 HMM

HMM sigue el modelo estadístico tradicional de Markov. [?]. 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 y se traslada de estado en estado. Esta transición se le denomina *step*. Los *steps* entre un estado S_i y S_j cumple una **probabilidad de transición**.

Por otro lado encontramos la **probabilidad de emisión**. Esta probabilidad responde a observar o en 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 HMM a Map-matching

Nuestra estructura del territorio 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 está almacenada en un *Shape* de forma que se trata de una sucesión de puntos GPS tal y como se muestra en la siguiente imagen:

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

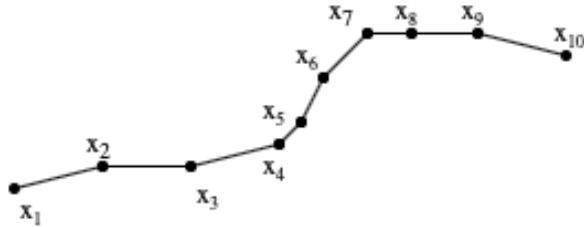


Figura 5.2: Ejemplo de estructura LineString

Por otra parte tenemos 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: Una serie de observaciones y un modelo. 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 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 [?].

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

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.

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

La implementación del algoritmo se realiza de forma iterativa para todos los puntos GPS de la trayectoria. Por cada punto se buscarán las proyecciones en un radio

determinado. De esta forma se realiza una primera criba de proyecciones que no serán probables. Por cada una de las proyecciones se calculará 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. En la propuesta se tiene en cuenta el posible desfase temporal de los diferentes dispositivos localizadores de GPS. Si el camino más corto tiene 8 o más nodos se descartará la trayectoria, al no ser valorable para el análisis.

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 5.3 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.



Figura 5.3: Ejemplo de aplicación de algoritmo de Viterbi para detección de trayectorias.

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. 5.4

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. 5.5

5. ANÁLISIS DE LOS DATOS GPS

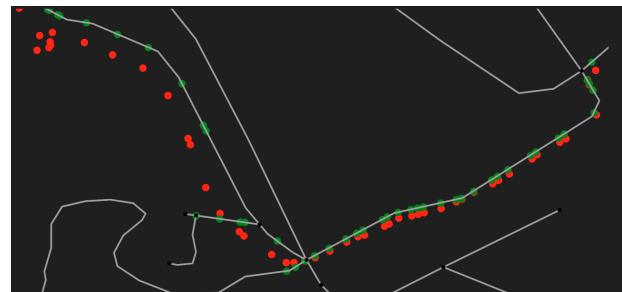


Figura 5.4: Ejemplo de aplicación de algoritmo de Viterbi para detección de trayectorias.

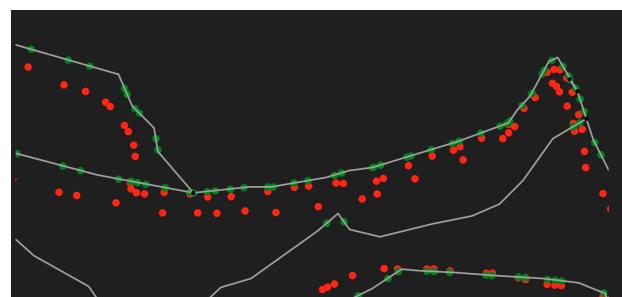


Figura 5.5: Ejemplo de aplicación de algoritmo de Viterbi para detección de trayectorias.

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*

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 5.1: Implementación interactor *get_map_matching_impl*

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

```

Algoritmo 5.2: Declaración interfaz *hidden_markov_model*

5.3 Explotación del dato

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

5.3.1 Análisis de distancias

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** [?]. Esta

5. ANÁLISIS DE LOS DATOS GPS

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 encontramos 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 nos permite saber que caminos se han atravesado para generar la trayectoria. Con esta información podemos generar la frecuencia de paso por cada uno de los caminos. 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 $c_{1,\dots,4}$ para un nodo n_x la probabilidad del punto será la siguiente:

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

siendo d el número de detecciones en en la arista.

Esta información será analizada y tratada de forma que el proceso de simulación genere una trayectoria basada en información real. La implementación de estos cálculos se realizan en el *interactor get_analysis_statistics_impl*. 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['
5         Point_lat']).haversine_distance(
6             Point(x['point_lon_shift']),
7             x['point_lat_shift']),
8             axis=1)
9     return data.drop(columns=['point_lat_shift', 'point_lon_shift'])

```

Algoritmo 5.3: 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         ']),
4             x['Point_lat']).haversine_distance(
5                 Point(x['Projection_lon']),
6                 x['Projection_lat']),
7                 axis=1)
8     return data

```

Algoritmo 5.4: Método add_point_projection_distance

5.3.2 Demostración de análisis

A continuación se mostrarán los resultados de un análisis utilizando la metodología mostrada en la parte superior del documento, se cuenta con una muestra 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: Como se ve

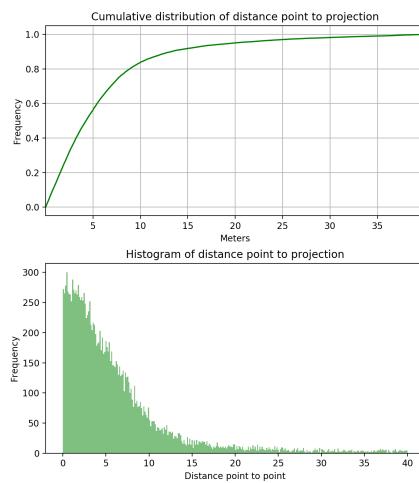


Figura 5.6: Muestra de análisis: Distancia punto a proyección.

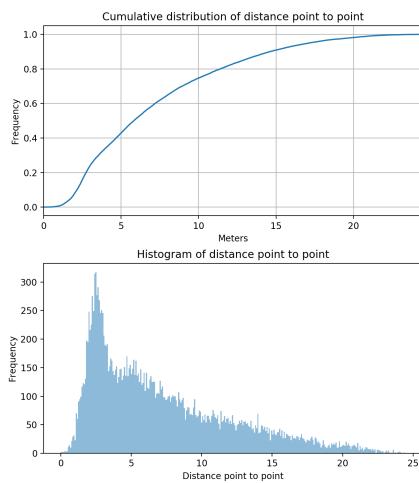


Figura 5.7: Muestra de análisis: Distancia punto a punto.

en la imagen 6.1 el 80 % de la distancia entre el punto y la proyección con el camino correcto está establecido entre 0 y 10 metros dentro.

En la figura 6.2 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 deberlo a la frecuencia del dispositivo, como a la velocidad de individuo, que puede variar en función de la pendiente del camino.

SIMULACIÓN DE TRAYECTORIAS

Definiremos 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 que cumpla los mismos criterios. El proceso de simulación (figura)de esta propuesta tiene dos procesos principales. Primero se generará el camino dada las probabilidades analizadas desde el nodo inicio seleccionado. Posteriormente se realizará la simulación de la trayectoria por el camino seleccionado.



Figura 6.1: Generación de camino.

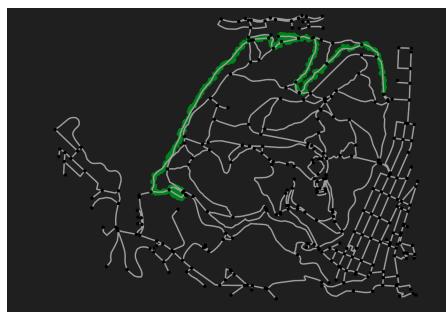


Figura 6.2: Generación de puntos.

6.1 Generación de un camino

La estructura lógica comentada en la sección 5.1.2 nos permite almacenar en las aristas la información referente a la frecuencia relativa de paso. Para generar un camino, se realizará un proceso iterativo por el cual mediante un nodo origen, asignado por parámetro de entrada, se seleccionará 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 cumpliría dicha distribución. El proceso de selección de aristas continuará hasta que la suma de

6. SIMULACIÓN DE TRAYECTORIAS

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 simulará cada uno de los segmentos individualmente. Para ello se identificara el nodo origen y final del segmento y se generarán 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_track_impl*

```
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.
7 graph.get_nodes()[origin_node][ 'y' ])
7     target_point = TrackPoint(self.graph.get_nodes()[target_node][ 'x' ], self.
8 graph.get_nodes()[target_node][ 'y' ])
8     try:
9         dest, aux = self.calculate_point(segment, origin_node, target_node,
10 origin_point, target_point)
11         next = dest
12         while aux > DISTANCE_TO_FINAL_NODE:
13             dest, aux = self.calculate_point(segment, origin_node,
14 target_node, next, target_point)
15             next = dest
16     except KeyError:
17         pass
18 return segment
```

Algoritmo 6.2: Método *simulate_segment*

6.3 Generación del punto pseudo-aleatorio

Es en este punto de la lógica 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 α como se muestra en la figura 6.3

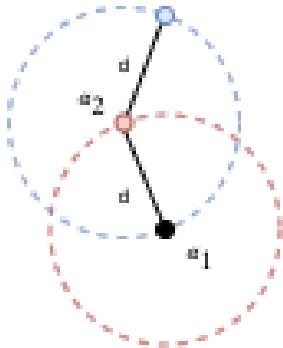


Figura 6.3: Ejemplo de generación de punto a partir de una distancia d y un ángulo α .

Cada punto será generado teniendo en cuenta el punto anterior. Se encontrará la proyección del segmento más cercano y se centrará el ángulo en el nodo inmediatamente posterior, con una desviación β parametrizada y modificable. El proceso queda ilustrado en la figura 6.4

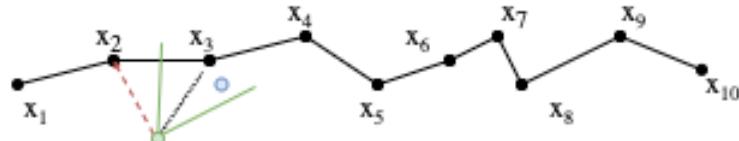


Figura 6.4: Ejemplo de detección de proyección cercana y generación de punto.

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

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 en la figura 6.2 mostrada en la sección 5.3.1.

6.4 Exportación de trayectoria a fichero .GPX

En esta sección se pretende describir la característica del aplicativo para realizar la exportación a GPX.

CAPÍTULO



EXPERIMENTACION

7.1 Experimentación sin importación de datos reales

En este apartatado se pretende realizar la muestra de resultados de la generación de 10 rutas creadas mediante la simulación sin datos almacenados. Al no tener datos almacenados dentro del sistema. La simulación corresponderá a una elección aleatoria del 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 10 metros, por lo que no aparecerá variabilidad en la muestra.

Se han realizado una simulación de 10 rutas. El resultado son 10 ficheros GPX que están situados en la carpeta X adjuntada con este documento.

[INSERTAR IMAGEN DE ANÁLISIS DE ESAS 10 RUTAS](#)

7.2 Experimentación con importación de datos reales

En este apartado se pretende realizar la muestra de resultados de la generación de 10 rutas creadas mediante la simulación a partir de los datos almacenados. Para la simulación se han analizado 25 rutas de un mismo individuo sobre el territorio delimitado del Castell de Bellver, Mallorca, España. Debido a la falta de muestra se ha tomado la decisión de realimentar al sistema con la misma muestra, de forma que el número de detecciones por segmento sea mayor. De esta forma se da lugar a una simulación con mayor probabilidad de replicar las decisiones tomadas por el individuo.

[INSERTAR IMAGEN DE MAPA DE CALOR DEL CASTILLO](#)

[INSERTAR IMAGEN DE ANÁLISIS DE ESAS 10 RUTAS](#)

CAPÍTULO

8

GUÍA DE INSTALACIÓN Y USO

En esta sección se explicará como poder ejecutar la aplicación dentro de nuestro equipo, así como los comandos básicos para la ejecución de la aplicación.

CAPÍTULO

9

CONCLUSIONES

En esta sección se realizará una explicación sobre el desarrollo de la propuesta.

9.1 Posibles mejoras

En esta sección se detallaran las posibles mejoras que se podrían implementar en la propuesta.

